IBM System/3
Disk System
RPG II and System Additional Topics
Programmer's Guide

IBM System/3
Disk System
RPG II and System Additional Topics
Programmer's Guide

Preface

This manual assumes that you have had programming experience on the IBM System/3 Disk System. You should now be familiar with basic RPG II concepts and disk concepts presented in the following manuals:

- *IBM System/3 Disk System Introduction,* GC21-7510.
- *IBM System/3 Card and Disk System RPG II Fundamentals Programmer's Guide,* GC21-7502.
- *IBM System/3 Disk System Concepts and Programming Programmer's Guide,* GC21-7503.

You should also be familiar with the term *disk system management.* A group of system programs called disk system management loads and runs programs on the disk system.

This manual presents additional RPG II and disk concepts that can help you in programming applications. Each chapter of this manual is a separate unit of instruction. A list is provided at the beginning of each chapter which details the contents of the chapter and the concepts you should be familiar with before reading that chapter. A series of review questions is provided at the end of each chapter to help you evaluate what you have learned.

This manual discusses direct file organization and the following processing methods:

- Consecutive processing of direct files.
- Random processing of sequential and direct files.
- Processing of disk files by record address files.

It also discusses concepts and coding for the following disk system features:

- ADDROUT sort.
- Automatic file allocation.
- Multi-volume files.
- Inquiry.
- Dual programming feature (DPF).
- Storing programs and procedures on disk.

First Edition (September 1970)

Changes are continually made to the specifications herein; any such change will be reported in subsequent revisions or Technical Newsletters.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for reader's comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Programming Publications, Department 425, Rochester, Minnesota 55901.

Additional RPG II concepts are presented on the following topics:

- Controlling the performance of operations.
- Altering the order of file processing.
- Describing input fields that control processing.
- Using the printer.
- Using arrays.
- Changing data structure.

Four manuals are available for further reference:

- *IBM System/3 Disk System Operation Control Language and Disk Utilities Reference Manual,* GC21-7512.
- *IBM System/3 Disk System Disk Sort Reference Manual,* SC21-7522.
- *IBM System/3 Disk System RPG II Reference Manual,* SC21-7504.
- *IBM System/3 Disk System Operator's Guide,* GC21-7508.

Contents

CHAPTER

DIRECT FILE ORGANIZATION

1

CHAPTER 1 DESCRIBES:

Direct file organization.

How records are retrieved from direct files.

How to handle synonym records.

Two ways to process direct files.

Adding and deleting records from a direct file.

Applications for direct file organization.

Creating a direct file with RPG II.

BEFORE READING THIS CHAPTER YOU SHOULD BE ABLE TO:

Describe System/3 disk storage concepts.

Describe sequential and indexed files.

Define consecutive and random processing.

Define addition and deletion of records.

Code RPG II specification sheets to process sequential and indexed files.

Note: These topics are described in *IBM System/3 Disk System Concepts and Programming Programmer's Guide,* GC21-7503.

AFTER READING THIS CHAPTER YOU SHOULD BE ABLE TO:

Describe direct file organization.

Define relative record number and synonym records.

List the three ways direct files can be processed.

Describe how records are added to or deleted from a direct file.

List several applications where direct file organization is preferable to other file organizations.

Code the RPG II specification sheets to create (load) a direct file.

INTRODUCTION

A direct file is a file on disk in which records are assigned specific record positions. Direct file organization enables you to directly access any record in the file without examining other records or searching an index. Thus, in some processing situations, direct file organization has advantages over sequential and indexed organizations (see *When to Use Direct File Organization*).

Figure 1-1 represents direct file organization. Records are assigned specific locations, regardless of the order they are put into the file. Record locations exist for all records which can be put into the file, although not all locations contain records. The location in the file where a record will be placed is determined from a control field in the record. Records can be scattered throughout the file, depending on the distribution of the control fields. The unused record locations contain blanks. How the locations became blank is discussed in *How to Create a Direct File.*

Direct files may span multiple disk volumes. When a direct file is processed, however, all volumes containing portions of the file must be mounted on the disk drive, since every record in the file must be accessible. Therefore, multi-volume direct files are limited to two volumes with a single disk drive (one fixed volume and one removable volume) and four volumes with dual disk drives (two fixed volumes and two removable volumes).

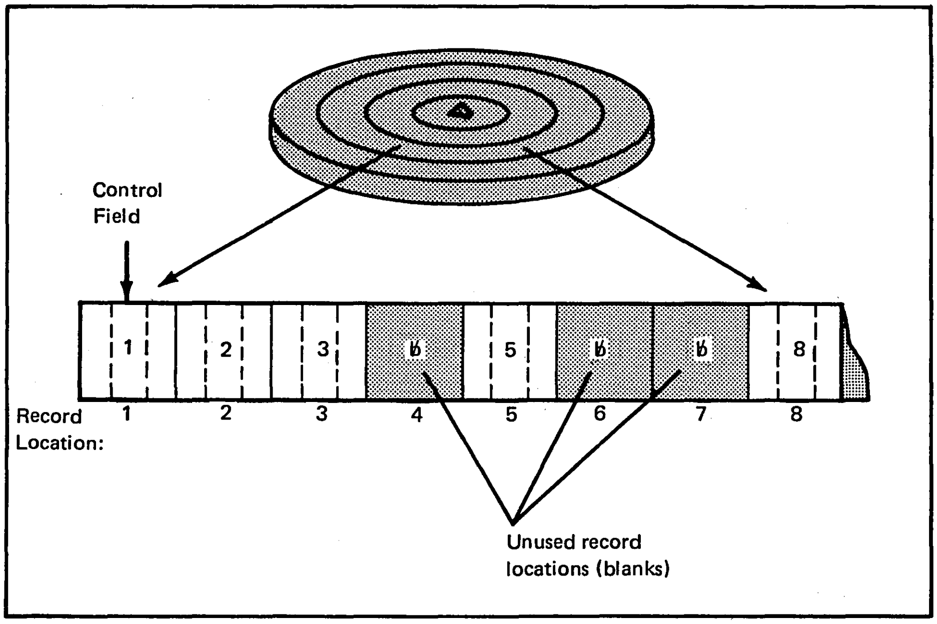


Figure 1-1. Direct File Organization

RELATIVE RECORD NUMBER

In a direct file, a record is written and retrieved *directly* by specifying the location of the record in relation to the beginning of the file. This relative position is called the relative record number. The relative record number is not a disk address, but is a positive, whole number that is converted by disk system management to the disk address of the record to be accessed.

Deriving the Relative Record Number

A relative record number is similar to the key of an indexed file or the control information in a sequential file: it is dependent upon a specific field (control field) in the record. The control field can either be used directly (without change) as a relative record number or it can be mathematically converted to provide an acceptable relative record number.

Direct Method

An easy way to derive relative record numbers is to have them correspond directly to the control fields in the records. Because the control information need not be converted into a relative record number, manipulation and programming are kept to a minimum. For example, in Figure 1-1 the record with a 1 in the control field becomes relative record number one; the record with a 5 becomes relative record number five, and so forth. This method is practical where control numbers can be assigned on a sequential basis, such as employee numbers for payroll records, student numbers in a school, and customer numbers for customer files.

Suppose a small college has an enrollment of 5,000 students. A master student file is maintained including currently enrolled students and graduates for the last two years. The master file contains approximately 7,000 words. Each student is assigned a 6-digit file number as follows:

74	9397
Expected year of graduation	A unique identification number from 1 - 9999

The identifying numbers are assigned on a sequential basis and numbers retired from the master file are available for reassignment.

A direct file with 10,000 record locations is used for the student master file, satisfying a need for fast access to each student's record. Since the identifying numbers range between 1 and 9999 and there are no duplicates, the relative record number is taken directly from the student file number. Figure 1-2 shows relative record numbers taken from the student file number being used to update student addresses.

Conversion Method

Conversion refers to any technique for obtaining a desirable range of relative record numbers from the control fields of the records. The conversion method must be used when the values in the control fields cannot be used directly as relative record numbers. For example, employee numbers in a factory range from 0001 to 1500, but only 450 numbers are in use since numbers belonging to employees who have retired or terminated have not been reused. A file large enough for 1500 records is not needed; therefore, a technique for converting the employee numbers to approximately a 1 through 500 range must be found. This provides 50 locations for file expansion.

When the conversion method is used, every possible control field in the file must convert to a relative record number in the allotted range (in this case, 1 through 500), and the resulting relative record numbers should be distributed evenly across the allotted range so that there are few *synonym* records. Synonym records are two or more records whose control fields yield the same relative record number (see *Synonym Records*). Your program must allow for synonyms if they are generated. (As a general rule, 15 percent of a file should be reserved for expansion and synonym records.)

One way to convert the range of employee numbers from 1500 to 500 is to divide the employee number by 3 and drop the remainder (thus 3 becomes 1; 6 becomes 2; 1500 becomes 500). However, unless the file is perfectly distributed, there will be synonym records. For example, if the numbers 6, 7, and 8 are present, all three become relative record number 2.

An alternate technique that produces fewer synonyms is to divide the employee number by 2 and drop the remainder. This compresses 1500 numbers to 750. There are 300 unused locations in this case, but fewer synonyms.

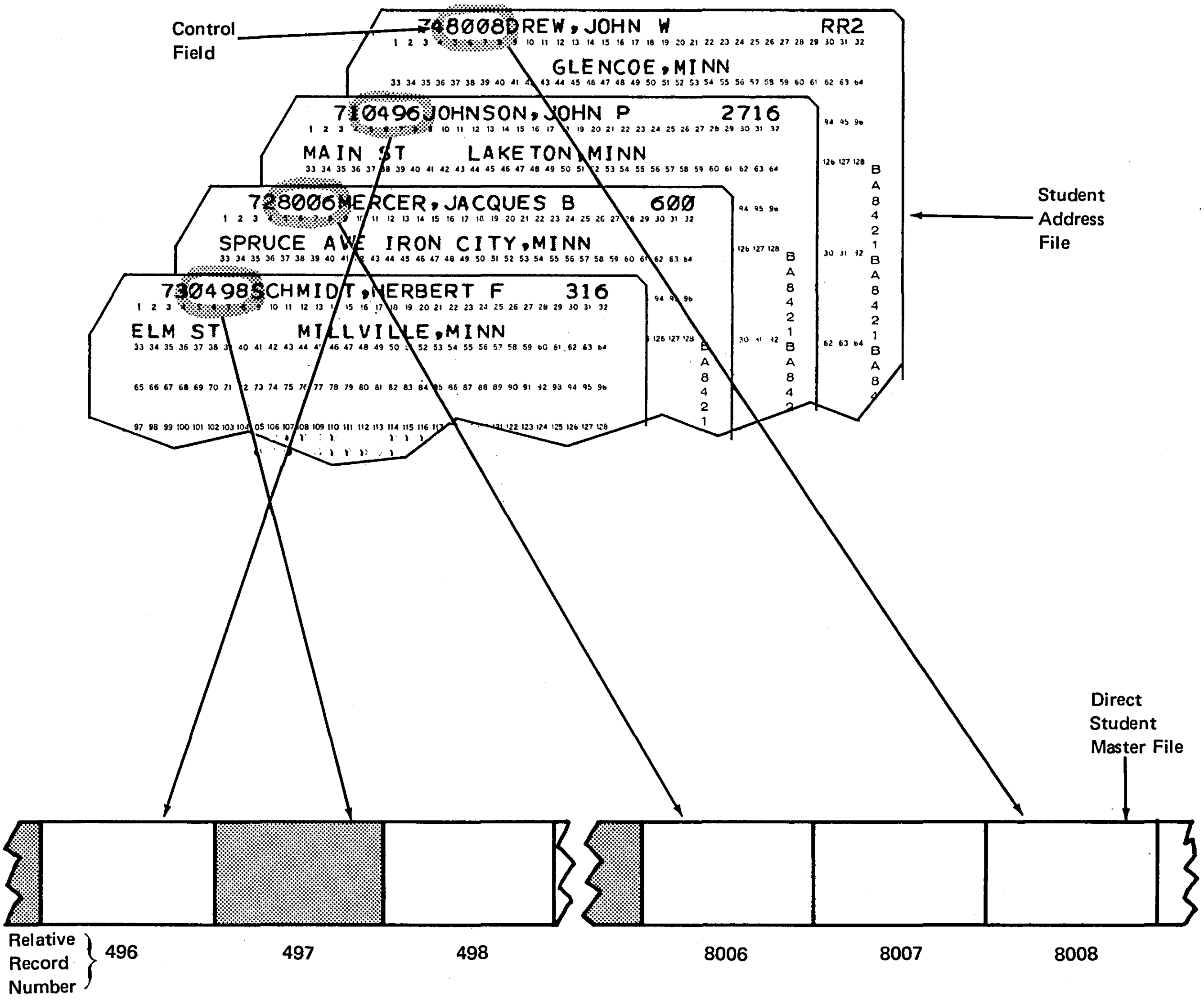


Figure 1-2. Relative Record Numbers Corresponding Directly to a Control Field

If there is no sequence to numbers in a control field (such as part numbers), a conversion technique that produces random numbers can be used. The resulting numbers should be distributed evenly within the selected range (depending upon the number of record locations needed) and should be suitable as relative record numbers (positive, whole numbers). One such technique is squaring the number in the control field and selecting certain digits from the resulting number as the relative record number. The calculation must be performed every time the program must seek a record. For example, suppose you have part numbers that consist of six digits, with certain digits having a special meaning. No two part numbers are alike. The part number is squared and, of twelve resulting digits, the center four are used as the relative record number for the parts inventory file.

Part number = 468152

468152 x 468152 = 219166629104

Relative record number = 6629

Since four digits are selected, random numbers from 1 to 9999 could be developed. Therefore, a file containing 10,000 record locations should be provided for the parts inventory.

Even the technique used in the example above is likely to produce synonym records, since the center four digits of the square of two different part numbers can be identical. If a conversion technique produces too many synonyms, it may be necessary to find a different technique or even a different file organization. The complexity of processing and programming for synonyms may outweigh the advantages of direct file organization.

Synonym Records

Two or more records whose control fields yield the same relative record number are called synonym records. Synonyms have the same relative record numbers, but contain different data. Only one of a group of synonyms can be stored in the record location which agrees with its relative record number. Therefore, you must find a way to store and retrieve the other synonyms.

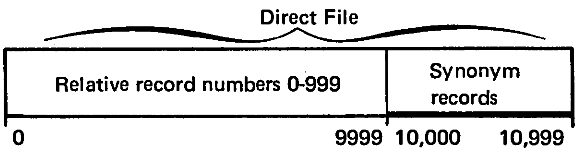
One way to handle synonyms is to link them together so that all can be found by locating the first, as in Figure 1-3. The first record is stored in the record location indicated by its relative record number. That location is called the *home location;* the record placed there is called the *home record.* The first synonym is stored in the first unoccupied record location (a location for which no relative record number was developed). The relative record number of the second location is then stored in the home record; that is, the first synonym is *linked* to the home record. The second synonym, if present, would be stored in the next unoccupied record location and would be linked to the first, and so forth.

In Figure 1-3, all records that are synonyms are loaded into the file after records that can be stored in their home location have been loaded. (See *How to Create a Direct File, Creating a File With Synonyms.*)

If a new record is added to the file, but its home location is occupied by a synonym for a different record location, that record must be treated as a synonym for its home location. Figure 1-4 shows the file that resulted from the addition of synonyms in Figure 1-3. The home location for record *C* is occupied by a synonym for record *B,* so record *C* is placed in the first unoccupied location. Since record B_1 is already linked to record B_2, record *C* must be linked through B_2 to its home location.

When you process a direct file containing synonyms, you must verify every record retrieved. For example, when you retrieve relative record 3 from the file in Figure 1-4, you get record B_1, which is a synonym for relative record 2. This is unacceptable. However, if you check the record retrieved, you find that it is a synonym. You can now chain to the relative record location, if any, indicated by the first record and retrieve the second record. You can continue this process until you find the record you want or until the chain of synonyms ends. In this case, you probably have an error because the requested record is not in the file.

A similar method for handling synonyms is to set aside a portion of the file for synonym records. Suppose, for example, a file for 8500 records is set up to provide relative record numbers between 0 and 9999. By actually setting aside enough area for 11,000 records, any synonyms developed can be stored in record locations from 10,000 to 10,999.



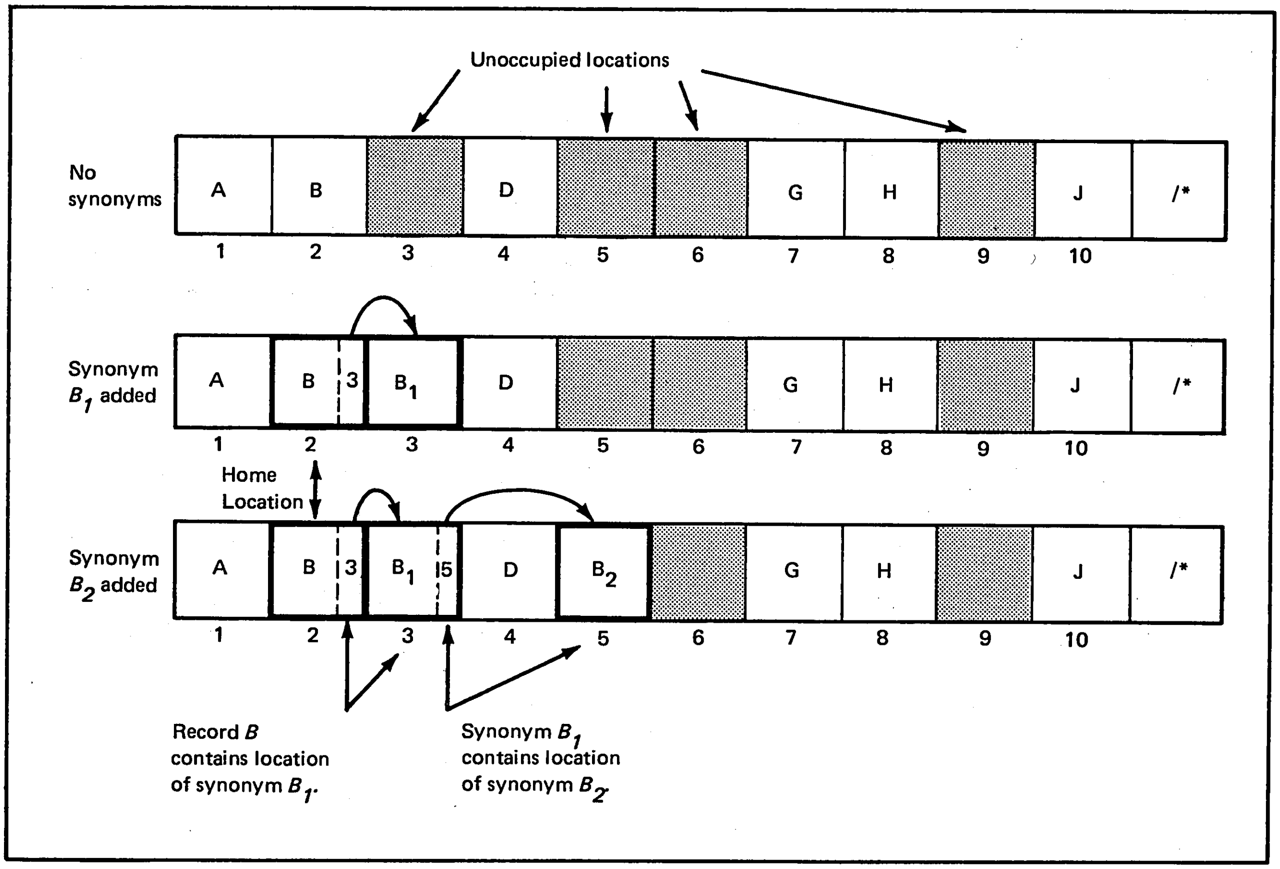


Figure 1-3. Storing Synonym Records in a Direct File

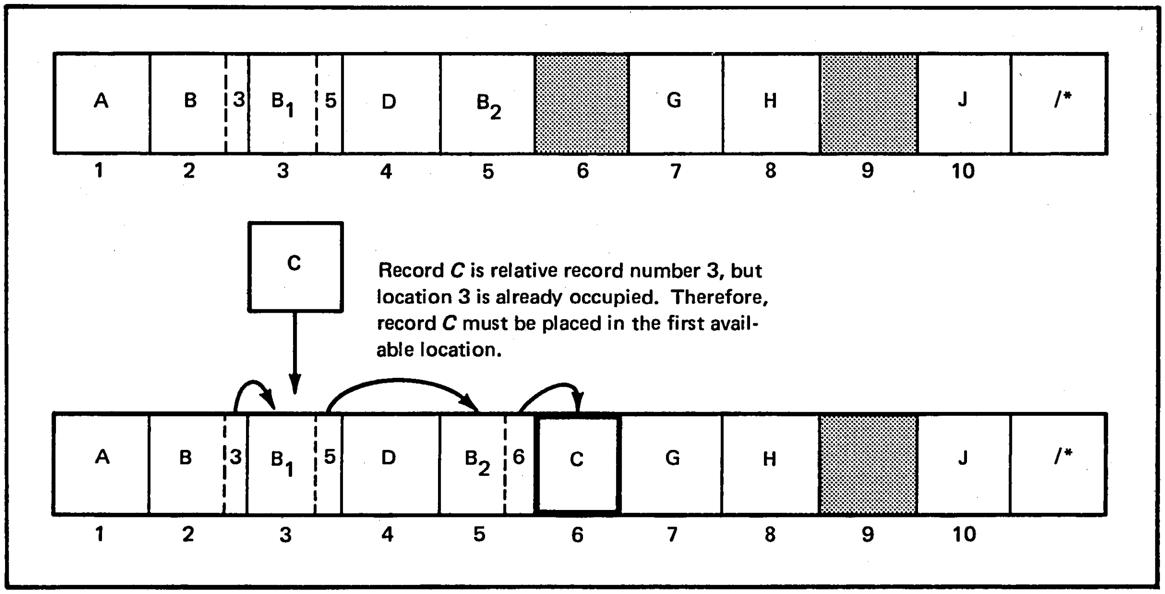
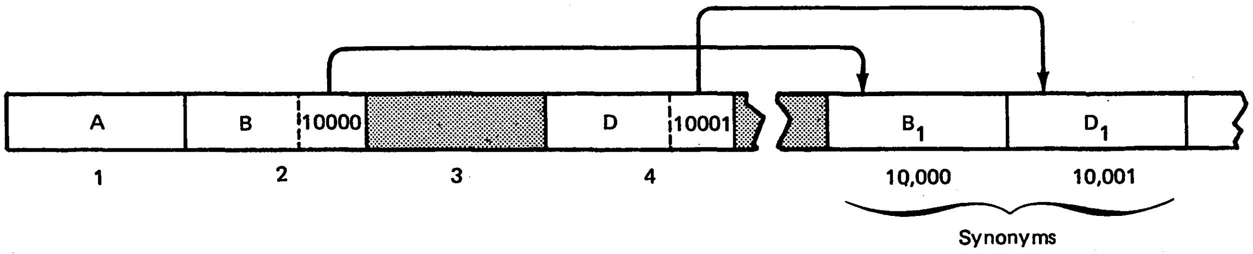

A
B
3
B_1
5
D
B_2
G
H
J
/*
1
2
3
4
5
6
7
8
9
10
C
Record *C* is relative record number 3, but location 3 is already occupied. Therefore, record *C* must be placed in the first available location.
A
B
3
B_1
5
D
B_2
6
C
G
H
J
/*
1
2
3
4
5
6
7
8
9
10

Figure 1-4. Storing a Record When Its Home Location Is Occupied

The relative record number of a synonym is stored in the home location, and a chain of synonyms is built as in the previous method.



If records are added to the file, this method can be better than the previous method, since a home location is kept free for each different relative record number. Only one seek is required for records without synonyms. However, this method wastes more space because 11,000 locations are used for 8500 records.

Other methods for handling synonyms can be devised. Whatever the method used, extra accesses are required for synonym records, and coding for verifying records is necessary.

PROCESSING DIRECT FILES

Direct files can be processed in three ways:

- Randomly by relative record number.
- Consecutively.
- Randomly by ADDROUT file (see Chapter 5).

Random Processing by Relative Record Number

Processing direct files by relative record number is similar to random processing of indexed files by key. In both cases, the file is processed randomly by the CHAIN operation code during calculation time in the RPG II object program cycle. In either type of file, only the records you specify are processed.

For direct files, the relative record number is used to locate the record you want. An index of record locations on disk is not required. The disk address of the record is calculated for you from the relative record number. Since no index search is required, random access of a direct file by relative record number can be faster than random access of an indexed file by key. (It may not be faster if a large number of synonym records exist, since the average number of seeks per record could become greater than the two required by an indexed file.)

Figure 1-5 shows the steps that occur in updating a direct master file with changes read from the MFCU. The master file is updated randomly as changes are read.

Random processing by relative record number can be used for retrieving or updating records from a direct file. (See Chapter 3, *Random Processing of Direct and Sequential Files,* for more detailed information.)

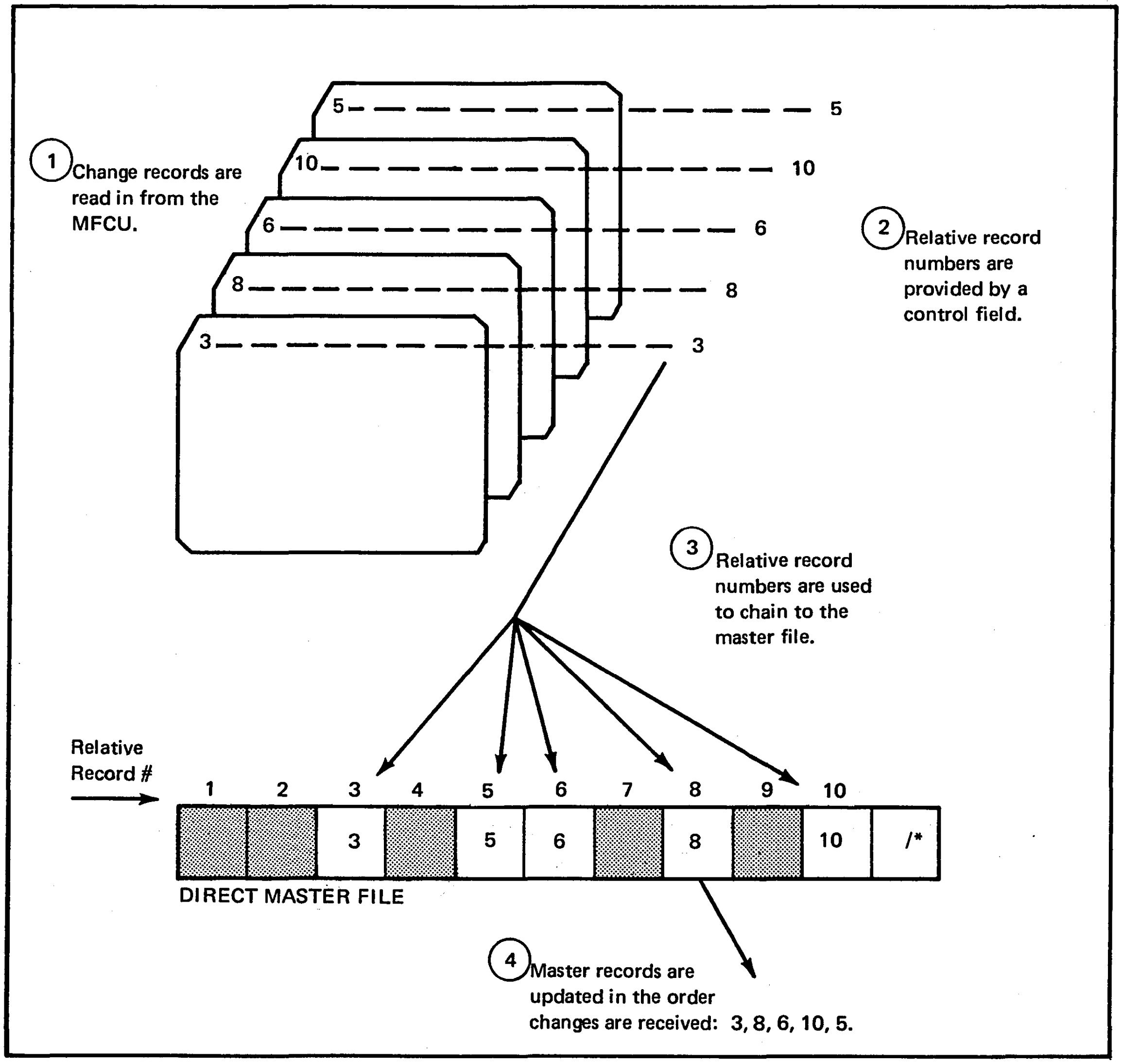
Consecutive Processing of Direct Files

If you process only a low volume of specific records from a direct file, random processing by relative record numbers is usually faster. If you process the entire file, you can process it *consecutively,* that is, one record after another from beginning to end.

In consecutive processing of both sequential and direct files, the contents of every record location is processed until the end of the file is reached or until the end of job conditions are met.

Since record locations containing blanks may be encountered in direct organization, you must allow for the blank records in your program.

Consecutive processing can be used to retrieve or update records from a direct file which is specified as a primary or secondary file. Detailed information on consecutive processing of direct files is presented in Chapter 2, *Consecutive Processing of Direct Files.*



1 Change records are read in from the MFCU.
5
10
6
8
3
5
10
6
8
3
2 Relative record numbers are provided by a control field.
3 Relative record numbers are used to chain to the master file.
Relative Record #
1
2
3
4
5
6
7
8
9
10
3
5
6
8
10
/*
DIRECT MASTER FILE
4 Master records are updated in the order changes are received: 3, 8, 6, 10, 5.

Figure 1-5. Random Processing of a Direct File

DIRECT FILES: ADDING AND DELETING RECORDS

After a file is created, file maintenance is usually necessary to keep the file current. Adding and deleting records are file maintenance functions common to all disk files.

Adding Records to Direct Files

Unlike sequential and indexed files, direct files can have space available between existing records for records to be added. (With either sequential or indexed files, new records are physically added at the end of records already in the file.) Records are added to a direct file by means of a normal update operation as follows:

1. The relative record number for the record to be added is developed.
2. The location is read into main storage.
3. If the location is blank, the new record is stored.
4. If the location is occupied, the new record is stored as a synonym (see *Synonym Records*).

In any file organization the situation can arise when records must be added, but the allotted file space is full. To add records, you must increase the total space available for the file by using the Disk Copy/Dump program to copy the file into a larger area (see *IBM System/3 Disk System Operation Control Language and Disk Utilities Reference Manual,* GC21-7512).

Deleting Records from Direct Files

As with sequential and indexed files, records in direct files can be identified for deletion by a *delete code.* This code is usually a single character at a particular location in the record. When the file is processed, your program must check for the delete code; if the code is present, the record can be bypassed.

Since the record has been deleted, the record location is available for a new record. Either a synonym for a different location can be stored or the location can be reused by assigning the relative record number to a new record. If the file contains synonyms, be careful not to delete synonyms chaining information when you delete a record and reuse the location.

Note: Records cannot be deleted from a direct file using the Disk Copy/Dump program. Because the DELETE parameter of the COPYFILE control statement causes physical deletion of identified records, the function would destroy the relative record positions on which direct file organization depends.

WHEN TO USE DIRECT FILE ORGANIZATION

When choosing a file organization, you must consider the use, size, activity, and volatility of the file.

Direct file organization can best be applied to files with the following characteristics:

- Low activity.
- Random processing (on an inquiry basis or by unordered transactions).
- Stable file size, not expanding beyond predictable limits.
- Control fields that can be used as or converted to a relative record number.

Most file uses which indicate direct file organization also indicate indexed organization. However, direct organization can have certain advantages over indexed:

- Direct file organization can require less main storage for processing because no index handling routines, index input/output areas, and master core index are needed.
- Random access of direct files can be faster, since a record can be retrieved by only a single access (seek and read or write). Similar access of an indexed file requires two accesses, one for the index and one for the data record.

Like indexed files, direct files allow immediate inquiry and response from any record in the file. This is important in applications such as:

- Demand deposit accounting when you must find the current balance of a specific account.
- Inventory control when you must retrieve information on inventory items.
- Accounts receivable when you must retrieve current customer information.

Also, like indexed files, direct master files allow processing of both ordered and unordered transactions. therefore, transactions need not be presorted. Thus, direct files can be used in a variety of jobs with several other files, sequenced or unsequenced, so long as the relative record number is furnished.

Considerations

A significant consideration in using direct file organization is developing the relative record number. If you use a simple method which produces few synonyms, direct file organization can be advantageous for you. Remember, however, that you must provide the relative record number, handle synonyms, and validate records retrieved. If the programming to perform these functions becomes too complex or requires too much main storage, you may want to consider indexed file organization as an alternative. You may waste file space by allowing for synonym records or by not reassigning relative record numbers when records are deleted. If too many synonyms are produced, the average number of seeks per record for a direct file can increase to a level where it is slower to process than an indexed file. Perhaps future additions and deletions to the file will upset the balance of your conversion technique. You must consider all these factors in choosing the file organization best suited to your needs.

A restriction inherent in direct organization or any other file organization which supports random processing is that the entire file must be online. That is, all volumes of the file must be mounted while the file is being processed. This means that a direct file is limited to two volumes in a single drive environment or four volumes in a dual drive environment. Since the number of volumes is limited, you will want to be sure that, at its maximum probable size, your direct file can be contained on the available disk space.

Summary

In summary, direct file organization can be used when:

- Direct inquiry capability is desired.
- Unordered (random) transactions are processed.
- Access speed is important to you.
- The size of the file is stable.
- The control field lends itself to developing a relative record number.

Considerations when using direct organization are:

- It can require more complicated programming and extra main storage, since the programmer must:
  1. Provide the relative record number.
  2. Handle synonym records.
  3. Validate records retrieved.
- Unused file space can result from blank record locations.
- All volumes must be online.

HOW TO CREATE A DIRECT FILE

To create a direct file, you must define a disk file as a chained output file in file description specifications (Figure 1-6). In this way, the file is uniquely identified as a direct file to disk system management. Disk system management then allocates disk space for the file and clears that space to blanks. From that point, the method you use to write data records on the file depends on whether or not you must check for synonyms among those records.

Whether or not you must check for synonyms, relative record numbers are used with the CHAIN operation code in your program to make the corresponding record locations available for loading. The data used as a relative record number in the chain operation can be a field in an input record, or it can be created in your program.

Creating a Direct File Without Synonyms

If you will not have synonyms, you can load records into a direct file in a single pass. You do this by specifying a chained output file and writing records in the file by means of the CHAIN operation. Record locations cannot be inspected before they are filled with data. If a synonym is encountered, it is written over the previous record and the previous record is lost.

Example of Creating a Direct File in this section describes the creation of a file without synonyms. This method of creating a direct file can be used when the relative record number either corresponds to a field containing sequential values or is derived in such a way that no synonyms are produced.

Creating a Direct File With Synonyms

If you have synonyms, you can create a direct file by using more than one pass to load records into the file. The exact method you use depends on your scheme for handling synonym records (see *Relative Record Number, Synonym Records*). Your first job must define the disk file as a direct file and clear the file to blanks. Once the file has been cleared, one or more subsequent jobs can be run using the update function to read record locations and check for synonyms while loading the file.

Figure 1-7 shows a method of defining a direct file and clearing it to blanks. In this method, the input card file from which the direct file is created is placed in the MFCU. The disk file is specified as a chained output file and is cleared to blanks by disk system management after the job begins. The CHANUM field from the card file is used to chain to the corresponding location in the direct file and the first record is placed in the file. The last record (LR) indicator is then turned on by a SETON operation, forcing the end of job condition. The direct file now contains a single record. This job can be immediately followed by one or more jobs which read the remaining cards from the MFCU and write out the disk records using the update function.

You learned in *Relative Record Number, Synonym Records* in this chapter that there are several ways to handle synonyms. Two methods described were:

1. Storing all synonyms in an area of the file set aside for them.

2. Storing synonyms in unused record locations between the records in the file.

After your direct file is defined and cleared to blanks, different steps are required to put records into the file for the two methods listed.

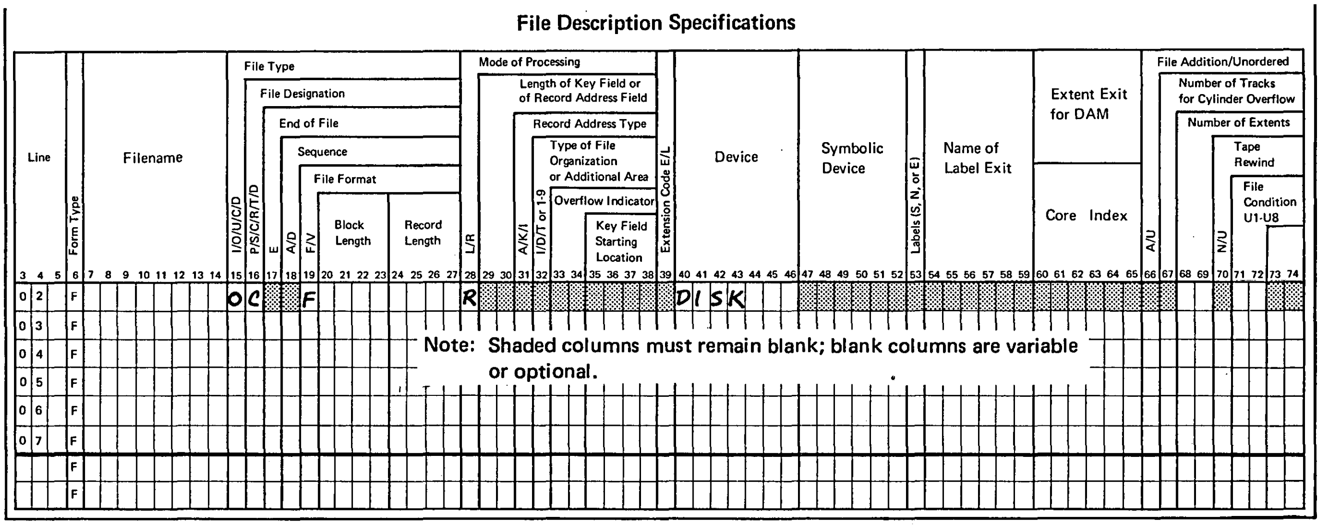
If the first method is used, all records can be placed in the direct file in a single job. That job would retrieve and check each record location before it is filled. If the location already contains a record (that is, the record to be written is a synonym), the synonym is stored in the next available location in the portion of the file set aside for synonyms. Thus, all home records and synonyms are placed in the file in a single job.

If the second method is used, two jobs are required to place home records and synonyms in the direct file. The first job loads all home records; any synonyms encountered are bypassed. The second job loads synonyms in the record locations available between home records. Both jobs are done using the update function to check each record location.

Whatever method you use to handle synonym records, you will have to devise a sequence of jobs similar to those described above. Remember:

1. A disk file is defined as a direct file by being specified as a chained output file.

2. In order to check for synonyms, you must employ the update function. Random update with direct files is described in Chapter 3, *Random Processing of Direct and Sequential Files.*



File Description Specifications

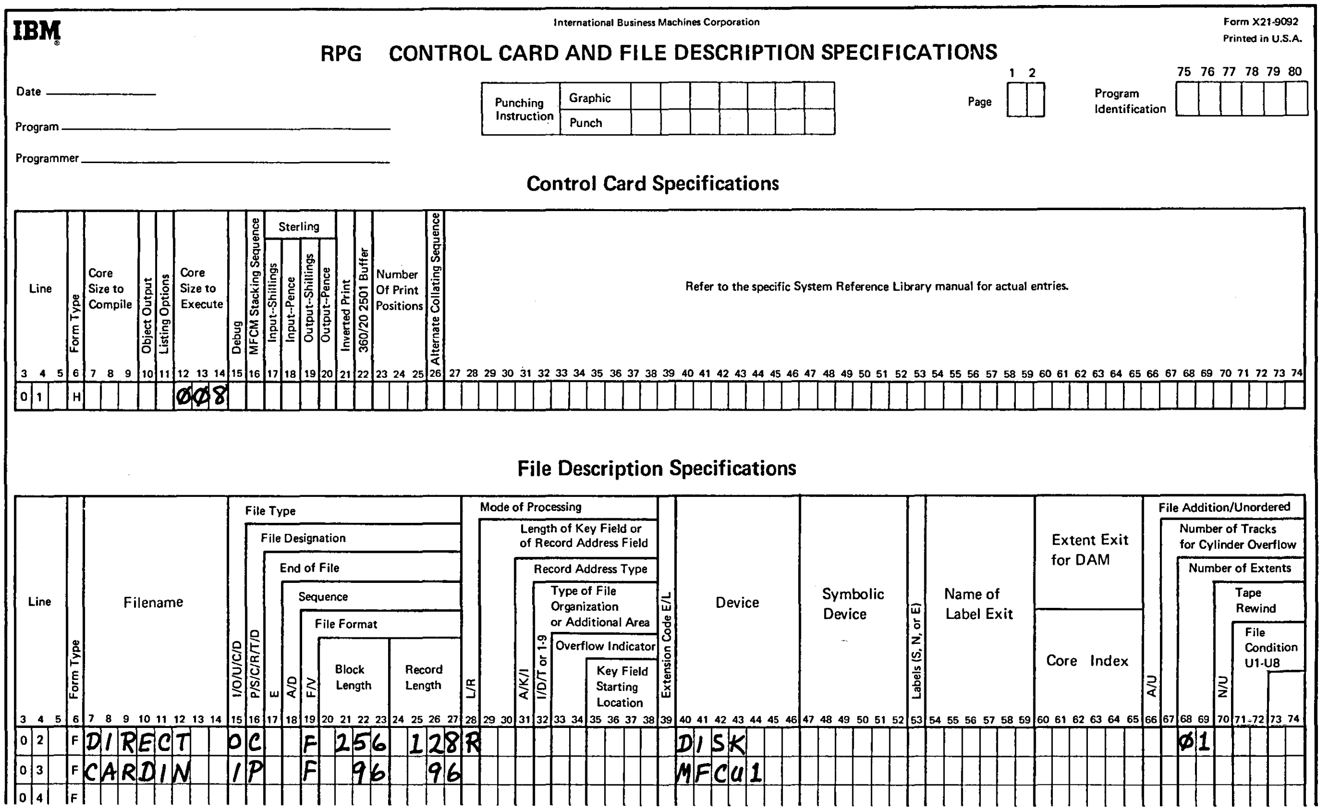
Line | Form Type | Filename | File Type: I/O/U/C/D | File Designation: P/S/C/R/T/D | End of File: E | Sequence: A/D | File Format: F/V | Block Length | Record Length | Mode of Processing: L/R | Length of Key Field or of Record Address Field | Record Address Type: A/K/I | Type of File Organization or Additional Area: I/D/T or 1-9 | Overflow Indicator | Key Field Starting Location | Extension Code E/L | Device | Symbolic Device | Labels (S, N, or E) | Name of Label Exit | Extent Exit for DAM | Core Index | File Addition/Unordered: A/U | Number of Tracks for Cylinder Overflow | Number of Extents | Tape Rewind: N/U | File Condition U1-U8

3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74

0 2	F		O	C	F	R	DISK
0 3	F						
0 4	F						
0 5	F						
0 6	F						
0 7	F						
	F						
	F						

Note: Shaded columns must remain blank; blank columns are variable or optional.

Figure 1-6. File Description Entries to Define a Direct File



IBM®

International Business Machines Corporation

Form X21-9092
Printed in U.S.A.

RPG CONTROL CARD AND FILE DESCRIPTION SPECIFICATIONS

Date ____________

Program ____________

Programmer ____________

Punching Instruction: Graphic / Punch

Page 1 2

Program Identification 75 76 77 78 79 80

Control Card Specifications

Line | Form Type | Core Size to Compile | Object Output | Listing Options | Core Size to Execute | Debug | MFCM Stacking Sequence | Sterling: Input-Shillings, Input-Pence, Output-Shillings, Output-Pence | Inverted Print | 360/20 2501 Buffer | Number Of Print Positions | Alternate Collating Sequence

Refer to the specific System Reference Library manual for actual entries.

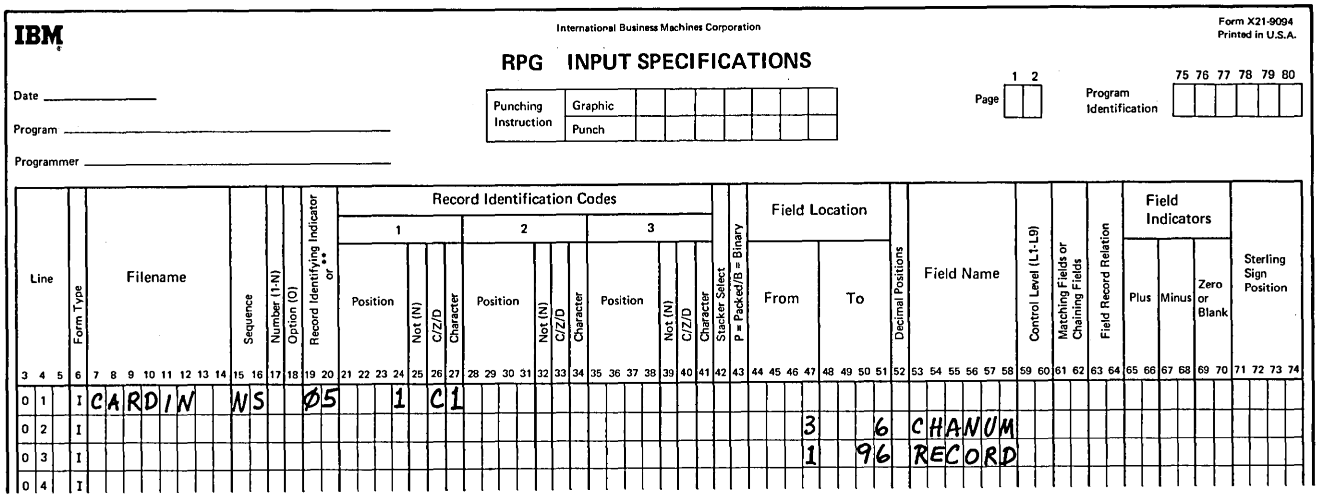
0 1	H	ØØ8

File Description Specifications

Line | Form Type | Filename | File Type: I/O/U/C/D | File Designation: P/S/C/R/T/D | End of File: E | Sequence: A/D | File Format: F/V | Block Length | Record Length | Mode of Processing: L/R | Length of Key Field or of Record Address Field | Record Address Type: A/K/I | Type of File Organization or Additional Area: I/D/T or 1-9 | Overflow Indicator | Key Field Starting Location | Extension Code E/L | Device | Symbolic Device | Labels (S, N, or E) | Name of Label Exit | Extent Exit for DAM | Core Index | File Addition/Unordered: A/U | Number of Tracks for Cylinder Overflow | Number of Extents | Tape Rewind: N/U | File Condition U1-U8

0 2	F	DIRECT	O	C	F	256	128	R	DISK	Ø1
0 3	F	CARDIN	I	P	F	96	96		MFCU1	
0 4	F									

Figure 1-7. Defining a Direct File and Clearing It to Blanks (part 1 of 4)



IBM

International Business Machines Corporation

Form X21-9094
Printed in U.S.A.

RPG INPUT SPECIFICATIONS

Date ______

Program ______

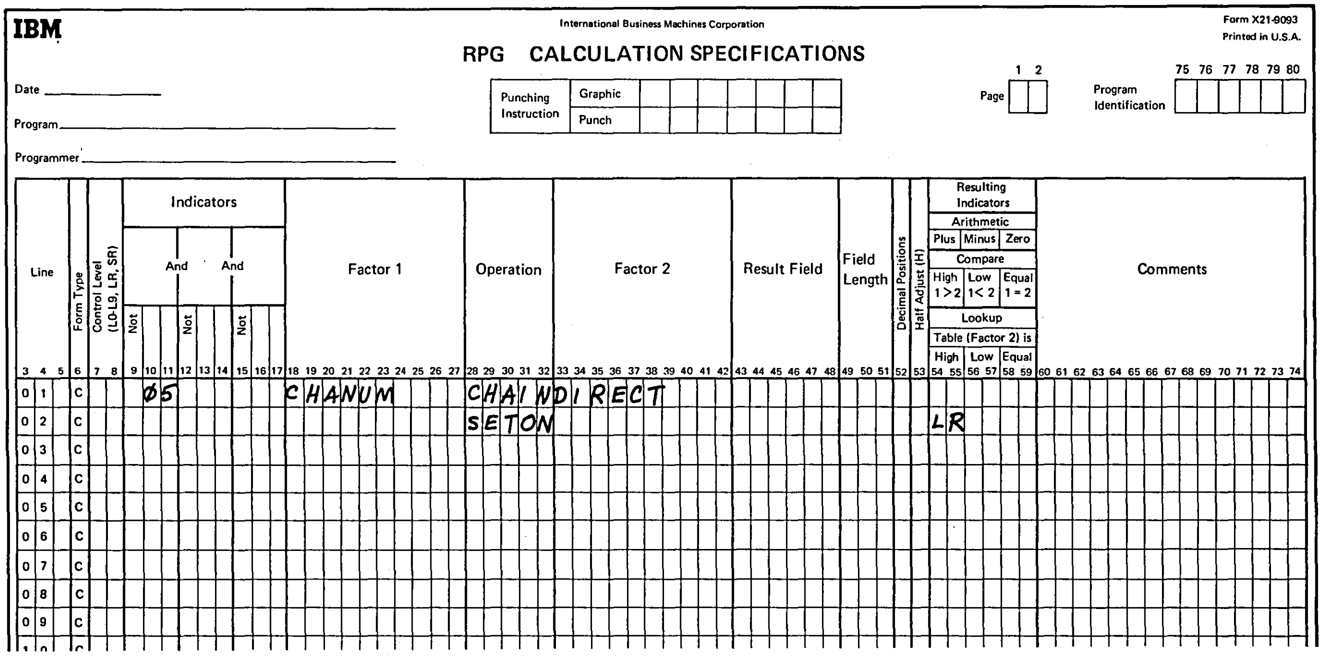
Programmer ______

Punching Instruction	Graphic									
	Punch									

Page (1 2) □□ Program Identification (75 76 77 78 79 80) □□□□□□

Line (3-5)	Form Type (6)	Filename (7-14)	Sequence (15-16)	Number (1-N) (17)	Option (O) (18)	Record Identifying Indicator or ** (19-20)	Record Identification Codes 1: Position (21-24)	Not (N) (25)	C/Z/D (26)	Character (27)	2: Position (28-31)	Not (N) (32)	C/Z/D (33)	Character (34)	3: Position (35-38)	Not (N) (39)	C/Z/D (40)	Character (41)	Stacker Select (42)	P = Packed/B = Binary (43)	Field Location From (44-47)	To (48-51)	Decimal Positions (52)	Field Name (53-58)	Control Level (L1-L9) (59-60)	Matching Fields or Chaining Fields (61-62)	Field Record Relation (63-64)	Field Indicators Plus (65-66)	Minus (67-68)	Zero or Blank (69-70)	Sterling Sign Position (71-74)
01	I	CARDIN	NS			Ø5	1		C	1																					
02	I																				3	6		CHANUM							
03	I																				1	96		RECORD							
04	I																														

Figure 1-7. Defining a Direct File and Clearing It to Blanks (part 2 of 4)



IBM

International Business Machines Corporation

Form X21-9093
Printed in U.S.A.

RPG CALCULATION SPECIFICATIONS

Date ______

Program ______

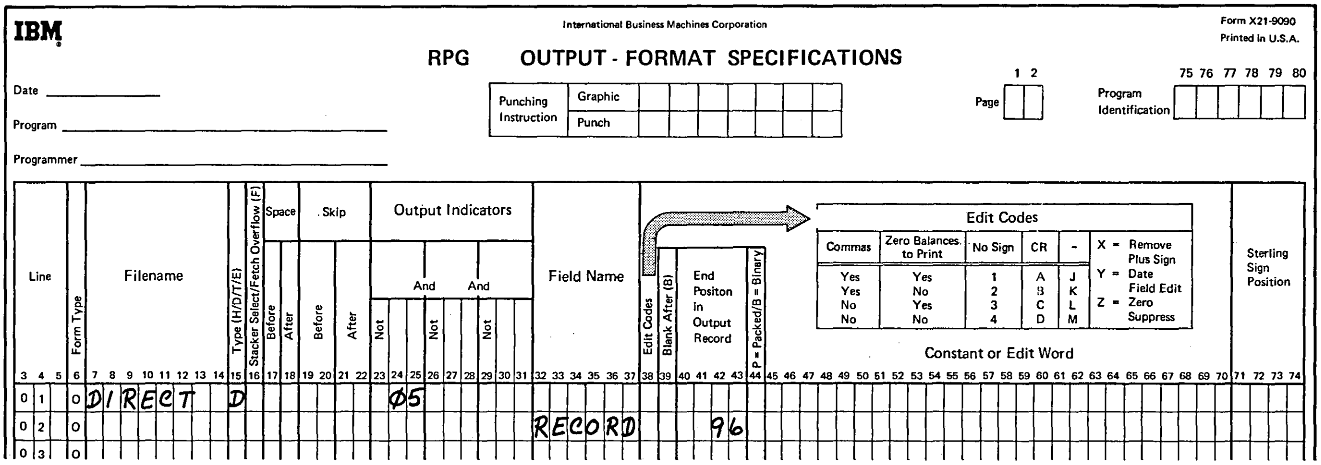
Programmer ______

Punching Instruction	Graphic									
	Punch									

Page (1 2) □□ Program Identification (75 76 77 78 79 80) □□□□□□

Line (3-5)	Form Type (6)	Control Level (L0-L9, LR, SR) (7-8)	Indicators: Not (9)	(10-11)	And Not (12)	(13-14)	And Not (15)	(16-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field (43-48)	Field Length (49-51)	Decimal Positions (52)	Half Adjust (H) (53)	Resulting Indicators Arithmetic Plus / Compare High 1>2 / Lookup Table (Factor 2) is High (54-55)	Minus / Low 1<2 / Low (56-57)	Zero / Equal 1=2 / Equal (58-59)	Comments (60-74)
01	C			Ø5					CHANUM	CHAIN	DIRECT								
02	C									SETON						LR			
03	C																		
04	C																		
05	C																		
06	C																		
07	C																		
08	C																		
09	C																		
10	C																		

Figure 1-7. Defining a Direct File and Clearing It to Blanks (part 3 of 4)



IBM International Business Machines Corporation Form X21-9090 Printed in U.S.A.

RPG OUTPUT - FORMAT SPECIFICATIONS

Date ____________

Program ____________

Programmer ____________

Punching Instruction: Graphic / Punch

Page 1 2

Program Identification 75 76 77 78 79 80

Line (3-5)	Form Type (6)	Filename (7-14)	Type (H/D/T/E) (15)	Stacker Select/Fetch Overflow (F) (16)	Space Before (17)	Space After (18)	Skip Before (19-20)	Skip After (21-22)	Output Indicators (23-31)	Field Name (32-37)	Edit Codes (38)	Blank After (B) (39)	End Position in Output Record (40-43)	P = Packed/B = Binary (44)	Constant or Edit Word (45-70)	Sterling Sign Position (71-74)
0 1	O	DIRECT	D						Ø5							
0 2	O									RECORD			96			
0 3	O															

Edit Codes

Commas	Zero Balances to Print	No Sign	CR	-	
Yes	Yes	1	A	J	X = Remove Plus Sign
Yes	No	2	B	K	Y = Date Field Edit
No	Yes	3	C	L	Z = Zero Suppress
No	No	4	D	M	

Figure 1-7. Defining a Direct File and Clearing It to Blanks (part 4 of 4)

Example of Creating a Direct File

A distributor wishes to create a customer file on disk. He lists the following as significant characteristics of the file:

- Customer numbers are assigned on a sequential basis; new customers are assigned the next higher number.
- There are few deletions from the file.
- The file will be used to process invoices, orders, and cash payments in an unordered manner.
- The file must allow direct inquiry to any customer's record.
- The file has low activity; for example, out of 5000 customer records, only 100 invoices are processed per day.

The distributor needs both direct and consecutive processing capability. These are offered by indexed and direct file organizations. Because the customer numbers are assigned consecutively, synonym records are not a consideration. For this reason, and because there will be few deletions from the file creating wasted space, direct file organization provides maximum flexibility and access speed.

His first step, then, is to create the direct file. He decides that the record format shown in Figure 1-8 satisfies his information needs. Additional fields in the record will contain information to be used in specific jobs, such as customer payments, invoicing, and sales analysis. (Various applications using the customer file are described in Chapter 2 and Chapter 3.)

The file is created from data on input cards (Figure 1-8). The customer number (CUSTNO) is used directly as the relative record number to chain to the direct file. The customer data from the input cards is then written on disk. As a check on the creation of the file, each record written on disk is also printed in the report shown in Figure 1-9.

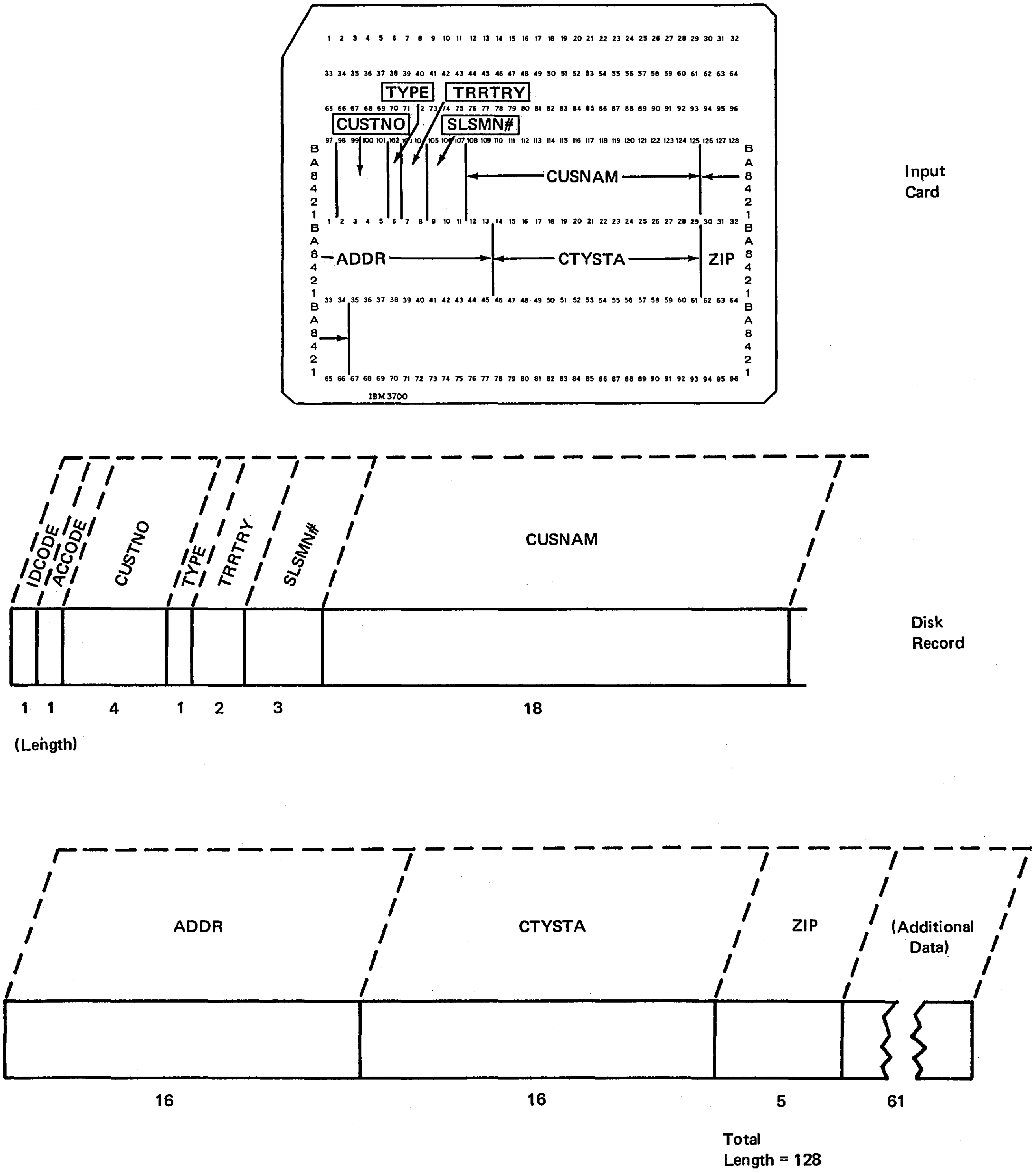
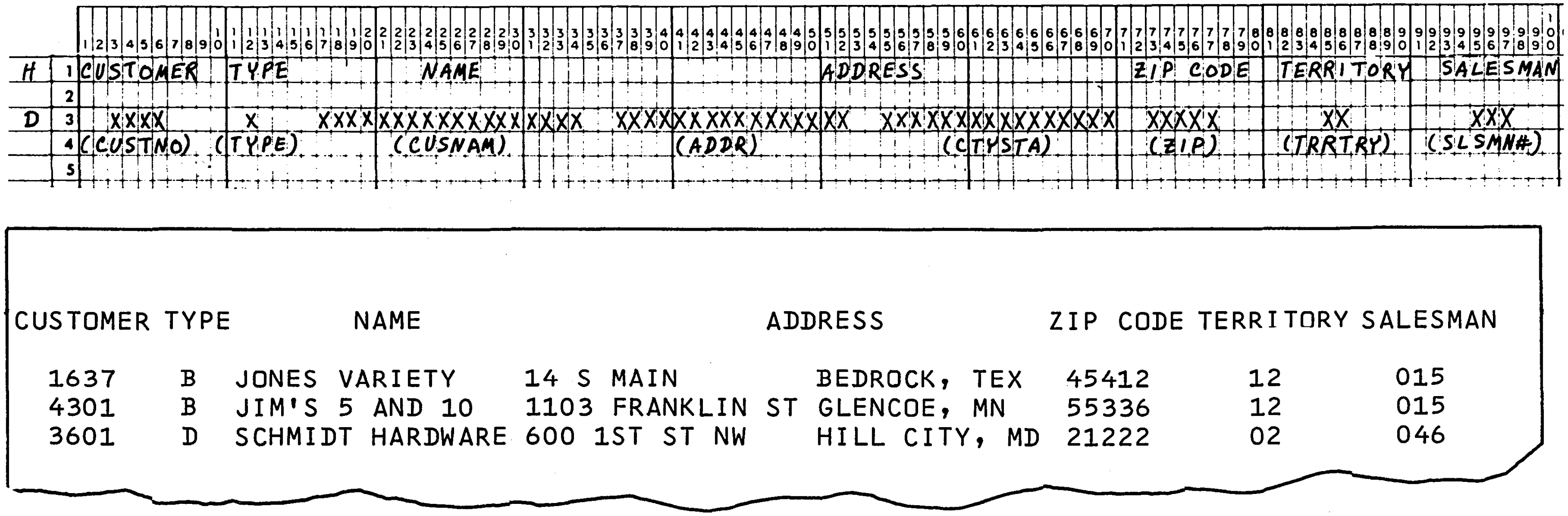


Figure 1-8. Record Formats for Creating a Direct Customer File

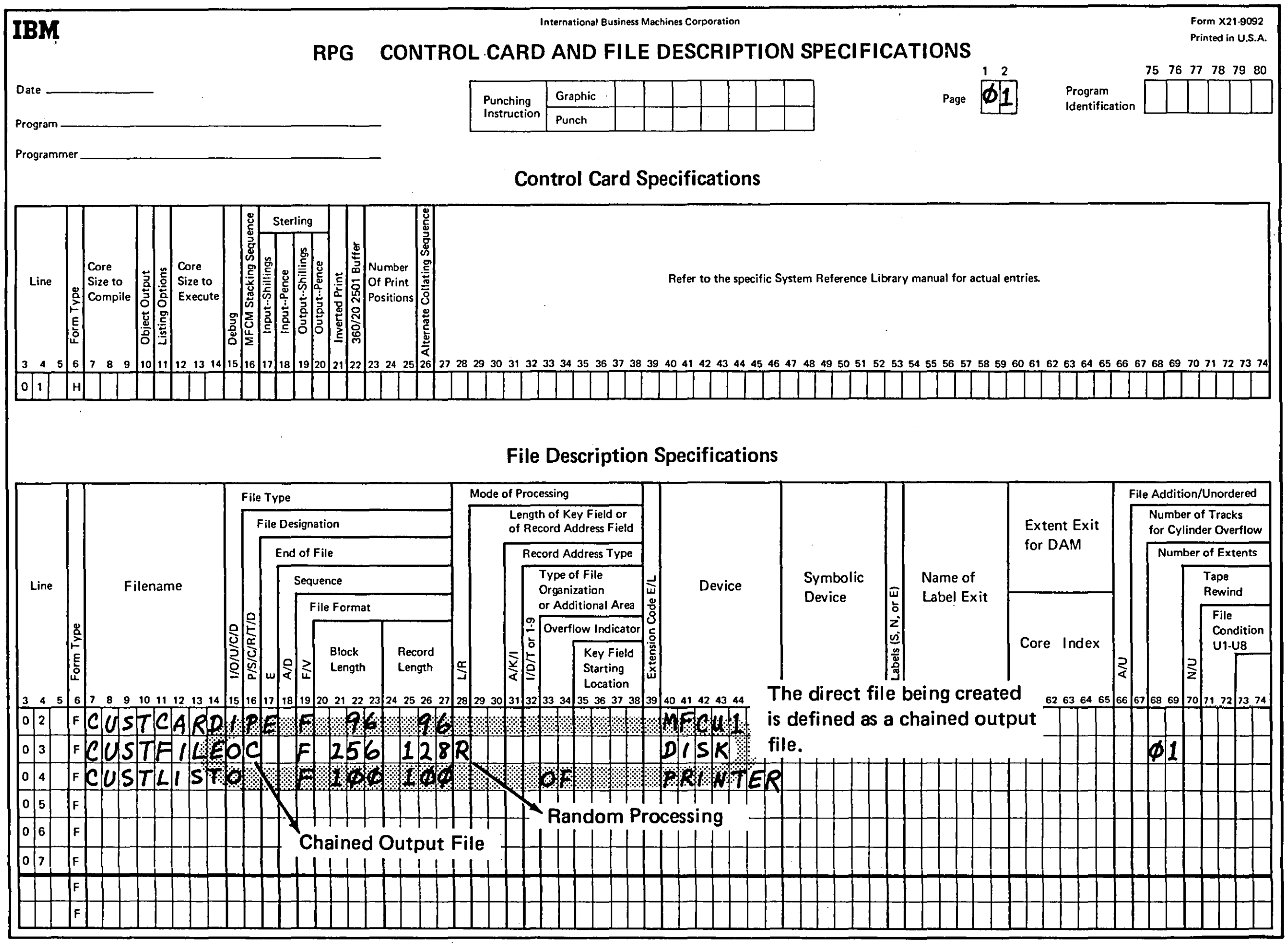


```
H  1 CUSTOMER TYPE           NAME                          ADDRESS                 ZIP CODE  TERRITORY  SALESMAN
   2
D  3   XXXX     X     XXXXXXXXXXXXXXXXXXX    XXXXXXXXXXXXXXXXX    XXXXXXXXXXXXXXXXXX   XXXXX       XX         XXX
   4 (CUSTNO) (TYPE)    (CUSNAM)            (ADDR)                  (CTYSTA)          (ZIP)     (TRRTRY)   (SLSMN#)
   5
```

```
CUSTOMER TYPE        NAME                           ADDRESS          ZIP CODE TERRITORY SALESMAN

  1637     B  JONES VARIETY    14 S MAIN         BEDROCK, TEX    45412       12        015
  4301     B  JIM'S 5 AND 10   1103 FRANKLIN ST  GLENCOE, MN     55336       12        015
  3601     D  SCHMIDT HARDWARE 600 1ST ST NW     HILL CITY, MD   21222       02        046
```

Figure 1-9. Printed Listing of Customer Records

Figure 1-10 shows the RPG II coding necessary to create the direct customer file.



IBM International Business Machines Corporation — Form X21-9092, Printed in U.S.A.

RPG CONTROL CARD AND FILE DESCRIPTION SPECIFICATIONS

Date ____ Program ____ Programmer ____ Punching Instruction: Graphic / Punch — Page Ø1 — Program Identification

Control Card Specifications

Line	Form Type	
01	H	Refer to the specific System Reference Library manual for actual entries.

File Description Specifications

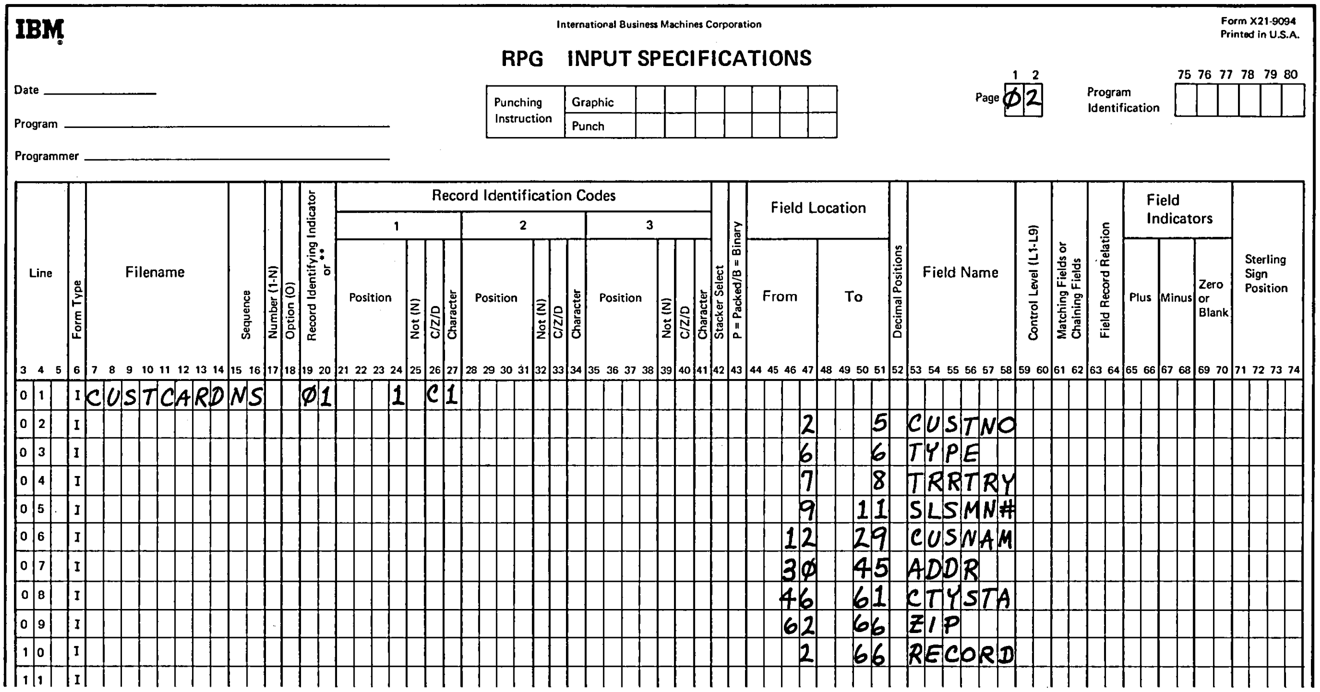
Line	Form Type	Filename	I/O/U/C/D	P/S/C/R/T/D	E	F/V	Block Length	Record Length	L/R	Overflow Indicator	Device	Number of Extents
02	F	CUSTCARD	I	P	E	F	96	96			MFCU1	
03	F	CUSTFILE	O	C		F	256	128	R		DISK	Ø1
04	F	CUSTLIST	O			F	1ØØ	1ØØ		OF	PRINTER	
05	F											
06	F											
07	F											
	F											
	F											

The direct file being created is defined as a chained output file.

Chained Output File

Random Processing

Figure 1-10. Creating a Direct File (part 1 of 4)



IBM

International Business Machines Corporation

RPG INPUT SPECIFICATIONS

Form X21-9094
Printed in U.S.A.

Date ____________

Program ____________

Programmer ____________

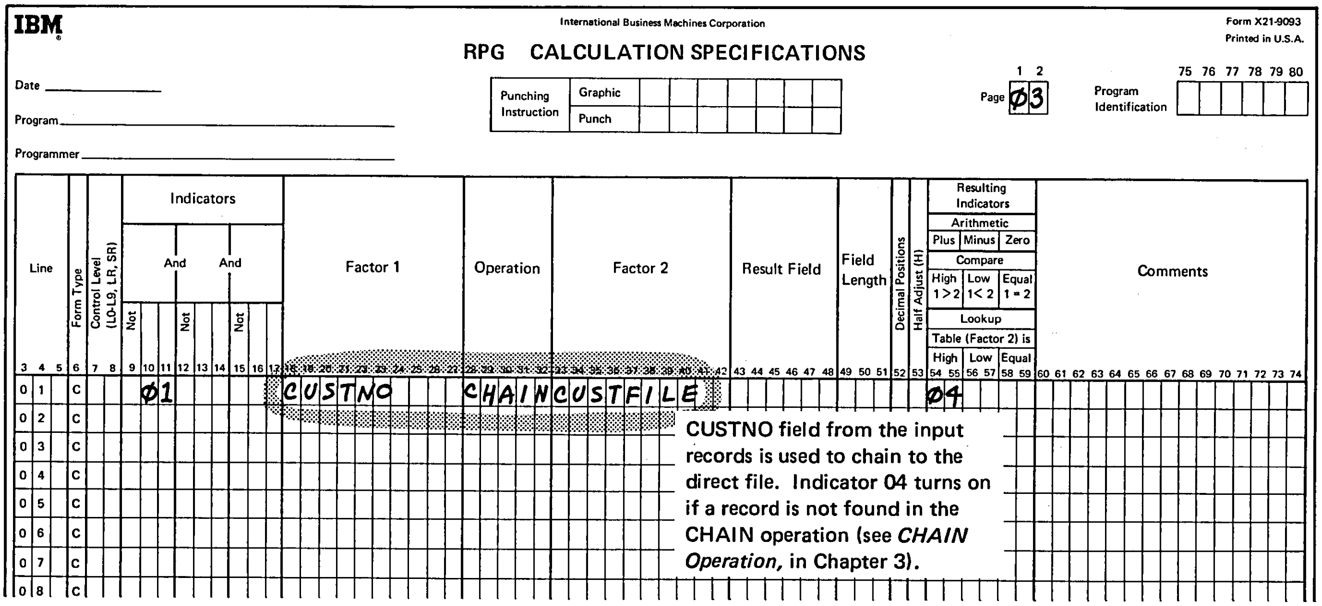
Punching Instruction: Graphic / Punch

Page ØZ

Program Identification

Line	Form Type	Filename	Sequence	Number (1-N)	Option (O)	Record Identifying Indicator or **	Record Identification Codes 1 Position	Not (N)	C/Z/D	Character	Field Location From	To	Decimal Positions	Field Name
01	I	CUSTCARD	NS			Ø1	1		C	1				
02	I										2	5		CUSTNO
03	I										6	6		TYPE
04	I										7	8		TRRTRY
05	I										9	11		SLSMN#
06	I										12	29		CUSNAM
07	I										3Ø	45		ADDR
08	I										46	61		CTYSTA
09	I										62	66		ZIP
10	I										1	66		RECORD
11	I													

Figure 1-10. Creating a Direct File (part 2 of 4)



IBM

International Business Machines Corporation

RPG CALCULATION SPECIFICATIONS

Form X21-9093
Printed in U.S.A.

Date ____________

Program ____________

Programmer ____________

Punching Instruction: Graphic / Punch

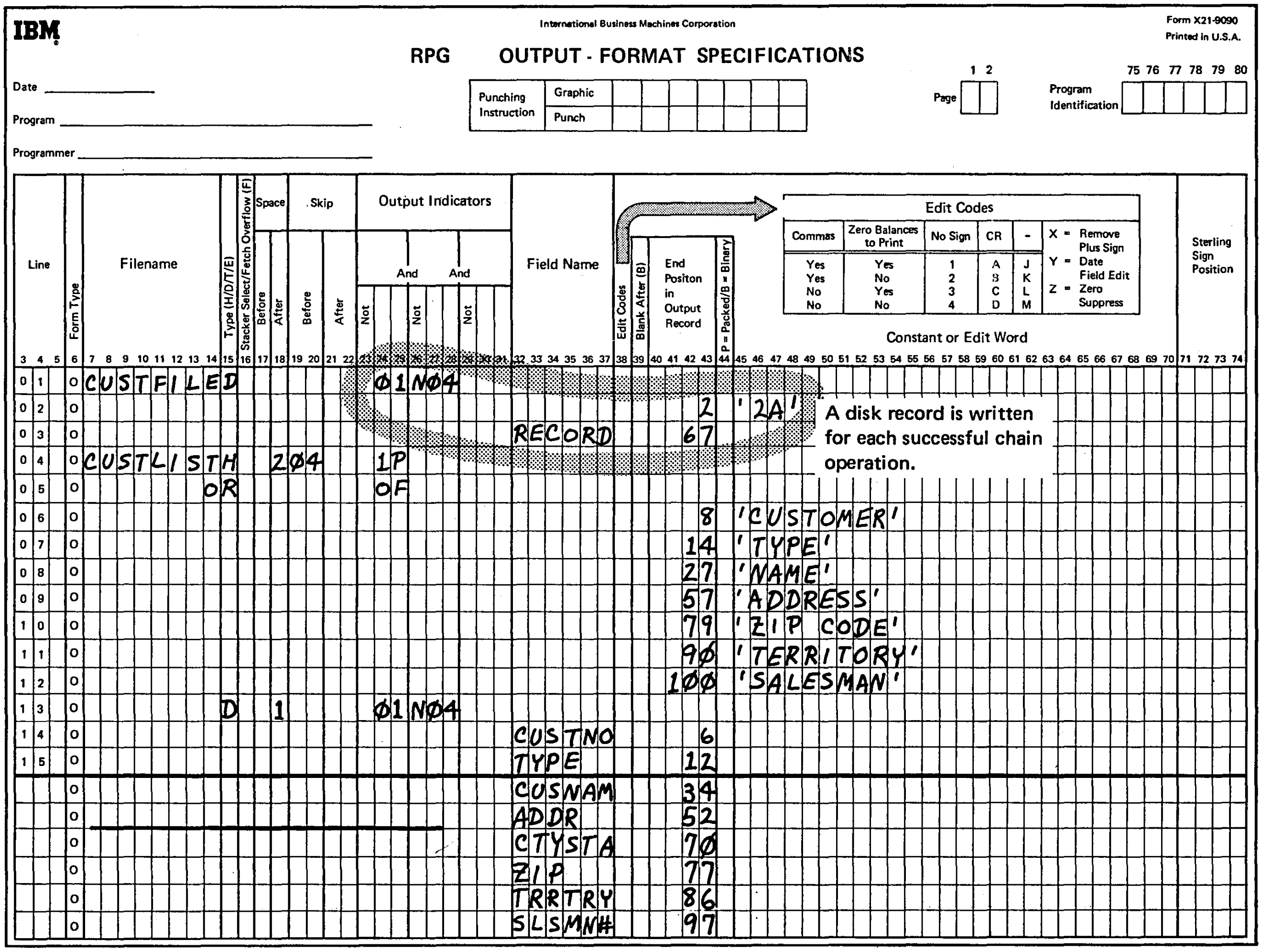
Page Ø3

Program Identification

Line	Form Type	Control Level (L0-L9, LR, SR)	Indicators	Factor 1	Operation	Factor 2	Result Field	Field Length	Decimal Positions	Half Adjust (H)	Resulting Indicators High/Plus	Low/Minus	Equal/Zero	Comments
01	C		Ø1	CUSTNO	CHAIN	CUSTFILE					Ø4			
02	C													
03	C													
04	C													
05	C													
06	C													
07	C													
08	C													

CUSTNO field from the input records is used to chain to the direct file. Indicator 04 turns on if a record is not found in the CHAIN operation (see *CHAIN Operation,* in Chapter 3).

Figure 1-10. Creating a Direct File (part 3 of 4)



IBM

International Business Machines Corporation

Form X21-9090
Printed in U.S.A.

RPG OUTPUT - FORMAT SPECIFICATIONS

Date

Program

Programmer

Punching Instruction: Graphic, Punch

Page 1 2

Program Identification 75 76 77 78 79 80

Edit Codes

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Line	Form Type	Filename	Type (H/D/T/E)	Stacker Select/Fetch Overflow (F)	Space Before	Space After	Skip Before	Skip After	Output Indicators	Field Name	Edit Codes	Blank After (B)	End Position in Output Record	P = Packed/B = Binary	Constant or Edit Word	Sterling Sign Position
01	O	CUSTFILE	D						Ø1NØ4							
02	O												2		'2A'	
03	O									RECORD			67			
04	O	CUSTLIST	H			2	Ø4		1P							
05	O	OR							OF							
06	O												8		'CUSTOMER'	
07	O												14		'TYPE'	
08	O												27		'NAME'	
09	O												57		'ADDRESS'	
10	O												79		'ZIP CODE'	
11	O												9Ø		'TERRITORY'	
12	O												1ØØ		'SALESMAN'	
13	O		D			1			Ø1NØ4							
14	O									CUSTNO			6			
15	O									TYPE			12			
	O									CUSNAM			34			
	O									ADDR			52			
	O									CTYSTA			7Ø			
	O									ZIP			77			
	O									TRRTRY			86			
	O									SLSMN#			97			

A disk record is written for each successful chain operation.

Figure 1-10. Creating a Direct File (part 4 of 4)

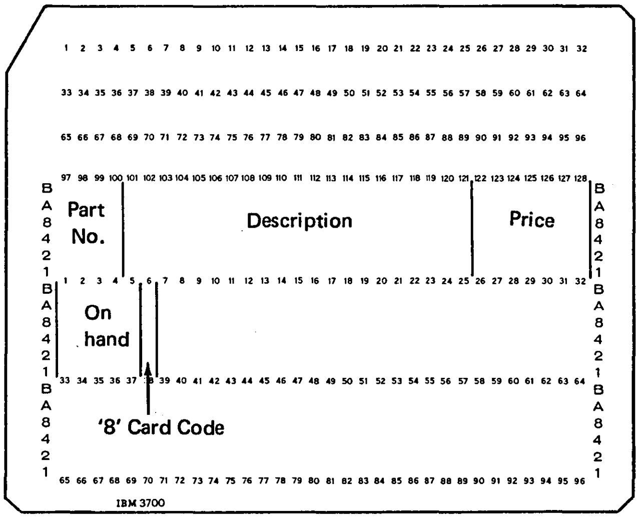
1. What distinguishes direct file organization from indexed or consecutive organization?

2. What is a relative record number? How can you determine a relative record number?

3. What must be done when a synonym record is encountered?

4. How do you add a record to a direct file?

5. Code RPG II specification sheets to create a direct file. The file is an inventory master file. The relative record numbers are the part numbers. The file is created from input cards in the following format:



The disk records should have the same format except the delete code should be in column 38. The delete code should be initialized to A. When the record is deleted, the code will be changed to D.

Answers to Review 1

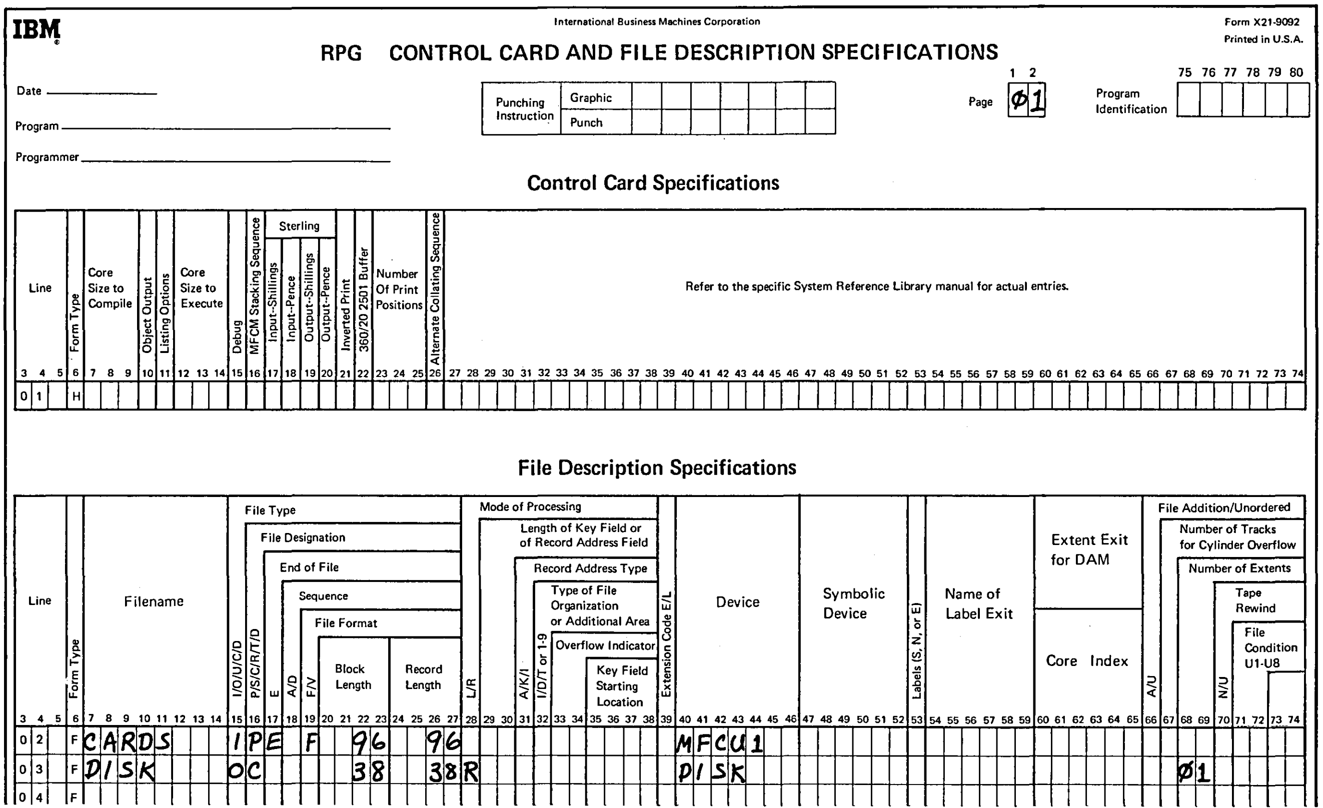
1. Records are loaded and retrieved from a direct file by specifying the relative position of the record in the file. Records can be scattered throughout the file. The sequence in which they are loaded depends on the sequence of relative positions supplied.

2. The relative record number is used to reference records in a direct file. It is the position of the record in relation to the beginning of the file. Relative record numbers can be determined in different ways. The direct method is a technique of using the control field directly as the relative record number. There are several methods of conversion by manipulating a control field mathemtically to determine an acceptable relative record number

3. When a synonym record is encountered, two control fields have been converted to the same relative record number. The programmer must provide an alternative record location for the synonym in these cases.

4. Records are added to blank or inactive locations within the file. Records can be deleted by activating the delete code and ignoring any data recorded in that record position or by blanking out the record.

5. See coding sheets (Figure 1-11).



IBM

International Business Machines Corporation

Form X21-9092
Printed in U.S.A.

RPG CONTROL CARD AND FILE DESCRIPTION SPECIFICATIONS

Date

Program

Programmer

Punching Instruction	Graphic							
	Punch							

Page (1 2) Ø1

Program Identification (75 76 77 78 79 80)

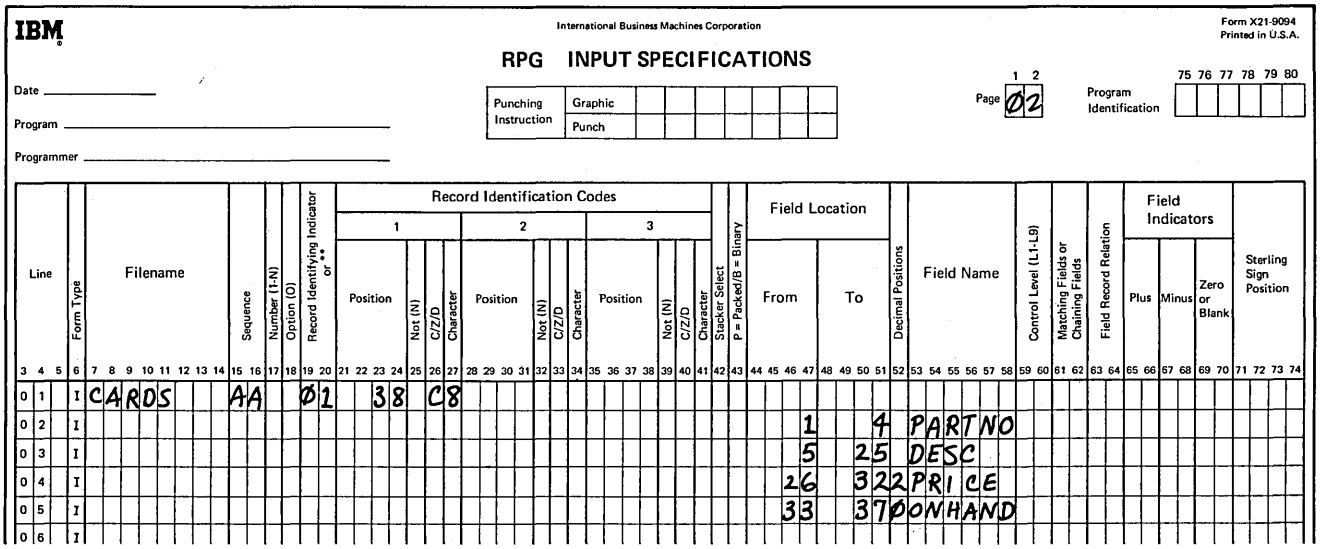
Control Card Specifications

Line (3-5)	Form Type (6)	Core Size to Compile (7-9)	Object Output (10)	Listing Options (11)	Core Size to Execute (12-14)	Debug (15)	MFCM Stacking Sequence (16)	Sterling: Input-Shillings (17)	Input-Pence (18)	Output-Shillings (19)	Output-Pence (20)	Inverted Print (21)	360/20 2501 Buffer (22)	Number Of Print Positions (23-25)	Alternate Collating Sequence (26)	Refer to the specific System Reference Library manual for actual entries. (27-74)
01	H															

File Description Specifications

Line (3-5)	Form Type (6)	Filename (7-14)	File Type: I/O/U/C/D (15)	File Designation: P/S/C/R/T/D (16)	End of File: E (17)	Sequence: A/D (18)	File Format: F/V (19)	Block Length (20-23)	Record Length (24-27)	Mode of Processing: L/R (28)	Length of Key Field or of Record Address Field (29-30)	Record Address Type: A/K/I (31)	Type of File Organization or Additional Area: I/D/T or 1-9 (32)	Overflow Indicator (33-34)	Key Field Starting Location (35-38)	Extension Code E/L (39)	Device (40-46)	Symbolic Device (47-52)	Labels (S, N, or E) (53)	Name of Label Exit (54-59)	Extent Exit for DAM / Core Index (60-65)	A/U (66)	File Addition/Unordered: Number of Tracks for Cylinder Overflow (67)	Number of Extents (68-69)	N/U (70)	Tape Rewind (71)	File Condition U1-U8 (72-74)
02	F	CARDS	I	P	E		F	96	96								MFCU1										
03	F	DISK	O	C				38	38	R							DISK							Ø1			
04	F																										

Figure 1-11. Creating a Direct Inventory File (part 1 of 4)



IBM

International Business Machines Corporation

Form X21-9094
Printed in U.S.A.

RPG INPUT SPECIFICATIONS

Date

Program

Programmer

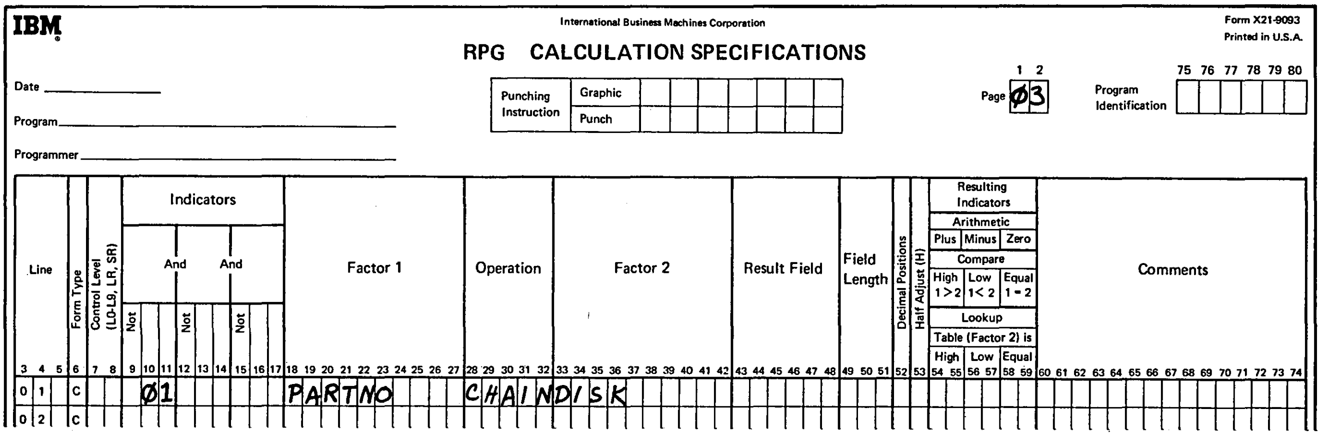
Punching Instruction	Graphic							
	Punch							

Page (1 2) Ø2

Program Identification (75 76 77 78 79 80)

Line (3-5)	Form Type (6)	Filename (7-14)	Sequence (15-16)	Number (1-N) (17)	Option (O) (18)	Record Identifying Indicator or ** (19-20)	Record Identification Codes 1: Position (21-24)	Not (N) (25)	C/Z/D (26)	Character (27)	2: Position (28-31)	Not (N) (32)	C/Z/D (33)	Character (34)	3: Position (35-38)	Not (N) (39)	C/Z/D (40)	Character (41)	Stacker Select (42)	P = Packed/B = Binary (43)	Field Location: From (44-47)	To (48-51)	Decimal Positions (52)	Field Name (53-58)	Control Level (L1-L9) (59-60)	Matching Fields or Chaining Fields (61-62)	Field Record Relation (63-64)	Field Indicators: Plus (65-66)	Minus (67-68)	Zero or Blank (69-70)	Sterling Sign Position (71-74)
01	I	CARDS	AA			Ø1	38		C	8																					
02	I																				1	4		PARTNO							
03	I																				5	25		DESC							
04	I																				26	32	2	PRICE							
05	I																				33	37	Ø	ONHAND							
06	I																														

Figure 1-11. Creating a Direct Inventory File (part 2 of 4)



IBM

International Business Machines Corporation

Form X21-9093
Printed in U.S.A.

RPG CALCULATION SPECIFICATIONS

Date

Program

Programmer

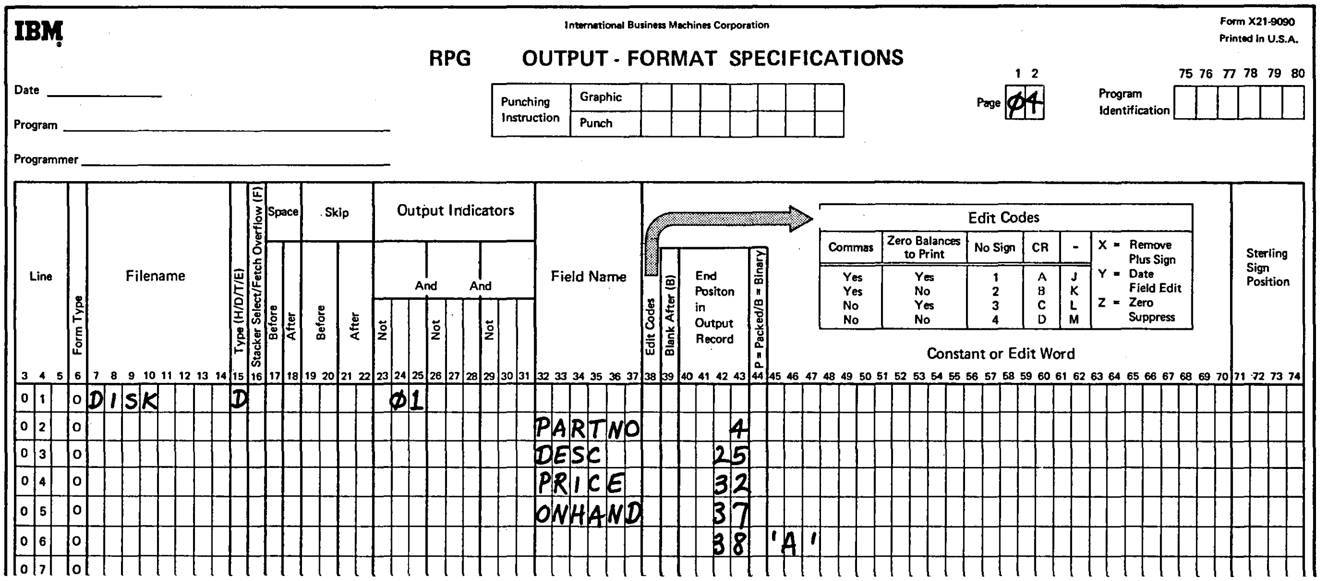
Punching Instruction: Graphic / Punch

Page Ø3

Program Identification

Line (3-5)	Form Type (6)	Control Level (L0-L9, LR, SR) (7-8)	Indicators (9-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field (43-48)	Field Length (49-51)	Decimal Positions (52)	Half Adjust (H) (53)	Resulting Indicators (54-59)	Comments (60-74)
01	C		Ø1	PARTNO	CHAIN	DISK						
02	C											

Figure 1-11. Creating a Direct Inventory File (part 3 of 4)



IBM

International Business Machines Corporation

Form X21-9090
Printed in U.S.A.

RPG OUTPUT - FORMAT SPECIFICATIONS

Date

Program

Programmer

Punching Instruction: Graphic / Punch

Page Ø4

Program Identification

Edit Codes

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Line (3-5)	Form Type (6)	Filename (7-14)	Type (H/D/T/E) (15)	Stacker Select/Fetch Overflow (F) (16)	Space Before (17)	Space After (18)	Skip Before (19-20)	Skip After (21-22)	Output Indicators (23-31)	Field Name (32-37)	Edit Codes (38)	Blank After (B) (39)	End Position in Output Record (40-43)	P = Packed/B = Binary (44)	Constant or Edit Word (45-70)	Sterling Sign Position (71-74)
01	O	DISK	D						Ø1							
02	O									PARTNO			4			
03	O									DESC			25			
04	O									PRICE			32			
05	O									ONHAND			37			
06	O												38		'A'	
07	O															

Figure 1-11. Creating a Direct Inventory File (part 4 of 4)

CHAPTER 2

CONSECUTIVE PROCESSING OF DIRECT FILES

CHAPTER 2 DESCRIBES:

Consecutive processing of direct files.

When to process direct files consecutively.

How to code the RPG II specification sheets to process a direct file consecutively.

BEFORE READING THIS CHAPTER YOU SHOULD BE ABLE TO:

Code basic RPG II programs using sequential and indexed files.

Describe consecutive processing.

Define activity and volatility of a file.

Describe, in concept, direct file organization.

Define synonym record.

AFTER READING THIS CHAPTER YOU SHOULD BE ABLE TO:

Describe, in concept, consecutive processing of direct files.

Code RPG II specification sheets to consecutively retrieve records from a direct file.

Code RPG II specification sheets to consecutively update records in a direct file.

State the conditions for selecting consecutive processing of direct files.

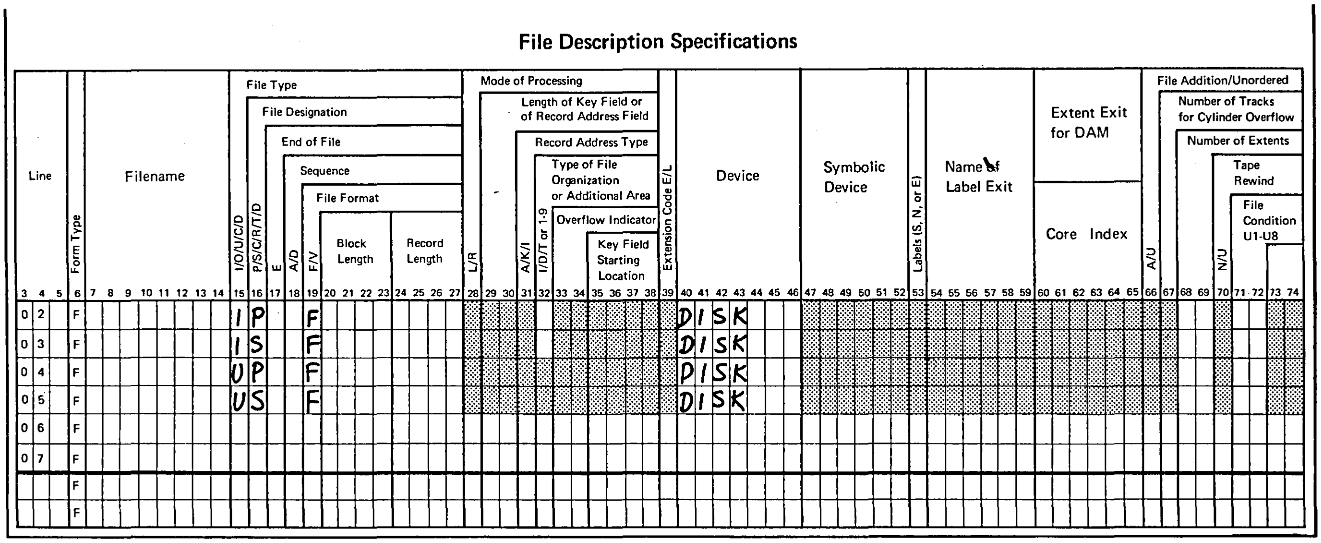
INTRODUCTION

Consecutive processing of direct files is similar to consecutive processing of sequential files. Record locations are processed one after another until end of job requirements are met. Blank record locations are processed along with those containing data. Remember that a direct file is cleared to blanks when it is created, and record locations which are not filled remain blank. See Chapter 1 for a description of direct file organization.

The File Description Sheet entries required for consecutive processing (Figure 2-1) are identical for direct and sequential file organizations. As shown in Figure 2-1, dual input/output areas can be requested (column 32 of the File Description Sheet) for primary and secondary direct input files. (The use of dual input/output areas is described in Chapter 11.) Consecutive processing can be used to retrieve and update primary and secondary direct files. It cannot be used to create a direct file.

When retrieving and updating a direct file consecutively, you may want to check each record for synonyms and handle the synonyms differently from other records. However, since consecutive processing is not dependent upon relative record numbers, a direct file can be processed consecutively without regard for synonyms.

If a consecutively processed direct file is in a logical sequence by a control field, it can be used in multifile processing. The same rules apply to direct files used in multifile processing which apply to other primary and secondary files. A complete description of multifile processing is in the *IBM System/3 Disk System RPG II Reference Manual*, SC21-7504.



File Description Specifications

Line (3-5)	Form Type (6)	Filename (7-14)	File Type I/O/U/C/D (15)	File Designation P/S/C/R/T/D (16)	End of File E (17)	Sequence A/D (18)	File Format F/V (19)	Device (40-46)
02	F		I	P			F	DISK
03	F		I	S			F	DISK
04	F		U	P			F	DISK
05	F		U	S			F	DISK
06	F							
07	F							
	F							
	F							

Note: Shaded columns are not used; blank columns are variable or optional.

Figure 2-1. File Description Specifications for Consecutive Processing

WHEN TO PROCESS DIRECT FILES CONSECUTIVELY

Consecutive processing of direct files is desirable in the following situations:

1. You want to process all or most of the records in the file (activity is high).
2. The physical sequence of the file is appropriate to your job. Either you do not care about the sequence (updating sales data, changing activity codes), or the physical sequence of the records (by account number, stock number, and so forth) is satisfactory for your purpose.

Direct files are often employed where the activity of a file is low and direct inquiry from the file is necessary. There are times, however, when the activity on a direct file is high for certain jobs, such as a writing a report where the entire file is listed. It can be desirable, in such cases, to process the file consecutively.

In Chapter 1, for example, a direct customer file is created for a distributor (see *How to Create a Direct File* in Chapter 1). The distributor selects direct file organization because the activity of the file is expected to be low, unordered transactions are to be processed, and he desires immediate inquiry capability. Each day, invoices are prepared, records are updated with sales information and customer payments, and inquiries concerning customer accounts are processed on a demand basis using random processing. At the end of each sales period, sales analysis reports are prepared and periodic adjustments are made to sales figures. These periodic jobs use consecutive processing, because all records are to be processed. Because the customer numbers used as relative record numbers are sequential, the periodic reports are in customer number sequence.

HOW TO CODE FOR CONSECUTIVE PROCESSING OF A DIRECT FILE

Consecutive Retrieval From a Direct File

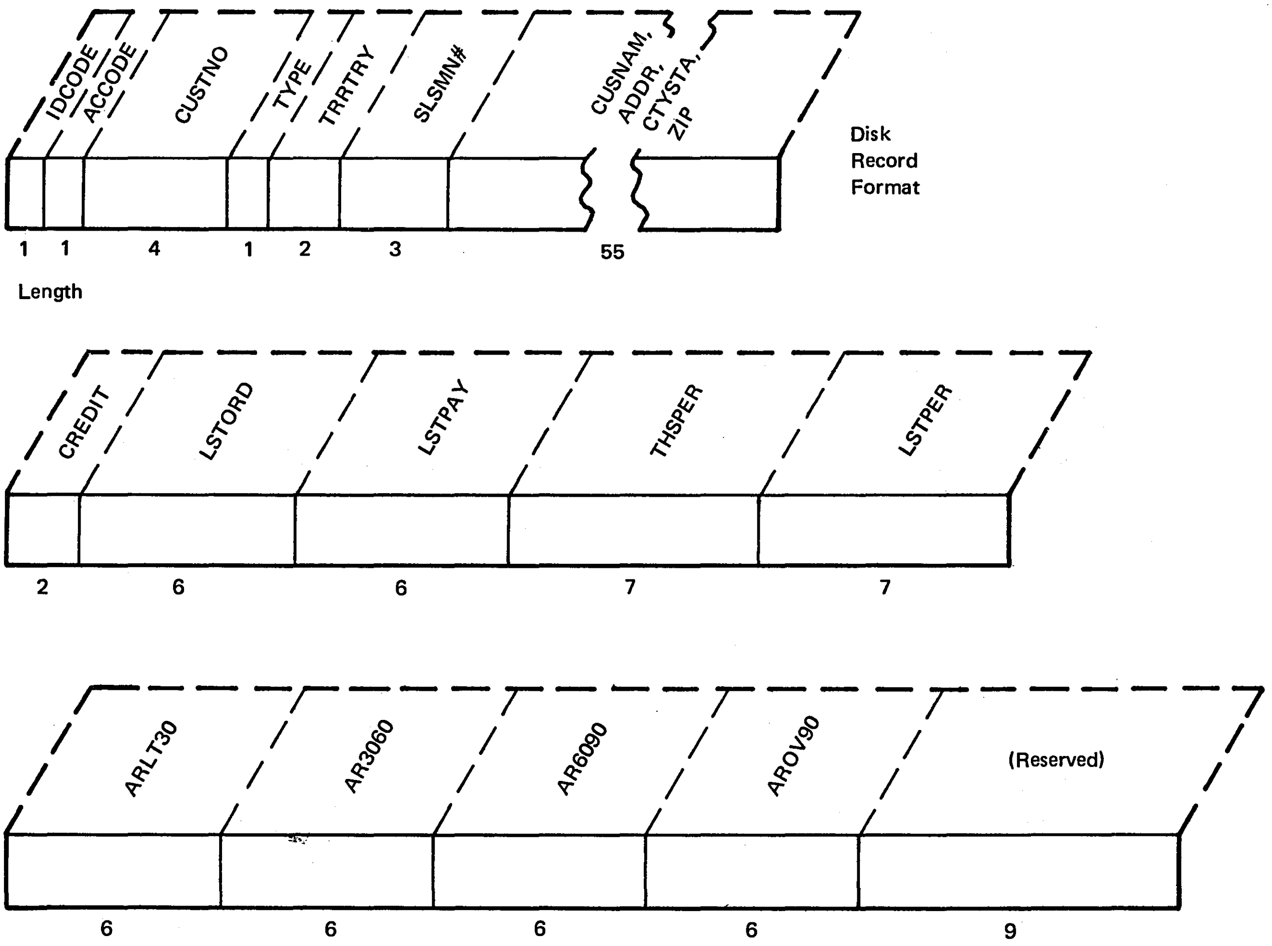
Consecutive retrieval of records from a direct file requires the same File Description Sheet entries as consecutive retrieval from a sequential file. The required entries for retrieval are shown in Figure 2-1 (I in column 15 indicates that the file is an input file). The file named in columns 7-14 must be a previously created direct file. Because the file is an input file, it must also be defined on the Input Specifications Sheet.

Example 1

Suppose the direct customer file, CUSTFILE, created in Chapter 1, is processed to produce a monthly report. This report shows all customers that have had no sales activity during the period. It is analyzed by sales personnel, who then make follow-up calls. The records of all customers are examined and the file is in sequence by customer number; therefore, the report is produced by consecutive processing of the direct file.

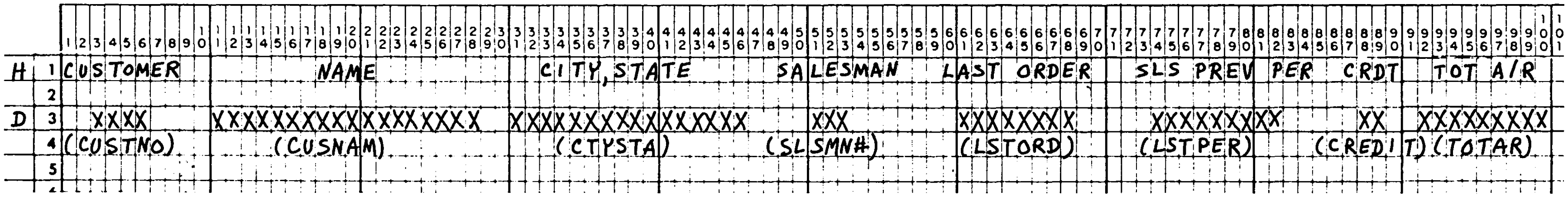
The format of the disk records in CUSTFILE is shown in Figure 2-2.

Figure 2-3 shows a part of the report produced by the consecutive processing job. The report consists of fields selected from CUSTFILE and an accumulated total for accounts receivable (TOTAR).

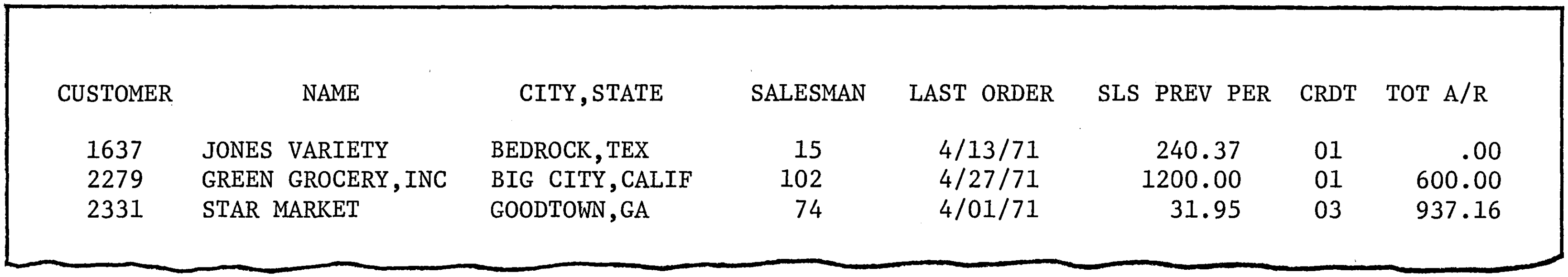


Total length of record = 128 positions

Figure 2-2. Disk Record Format for Direct Customer File, CUSTFILE



```
H 1 CUSTOMER         NAME     CITY,STATE      SALESMAN  LAST ORDER  SLS PREV PER   CRDT  TOT A/R
  2
D 3   XXXX    XXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXX    XXX       XXXXXXXX      XXXXXXXXX    XX   XXXXXXXXX
  4 (CUSTNO)         (CUSNAM)    (CTYSTA)      (SLSMN#)    (LSTORD)    (LSTPER)    (CREDIT)(TOTAR)
  5
```

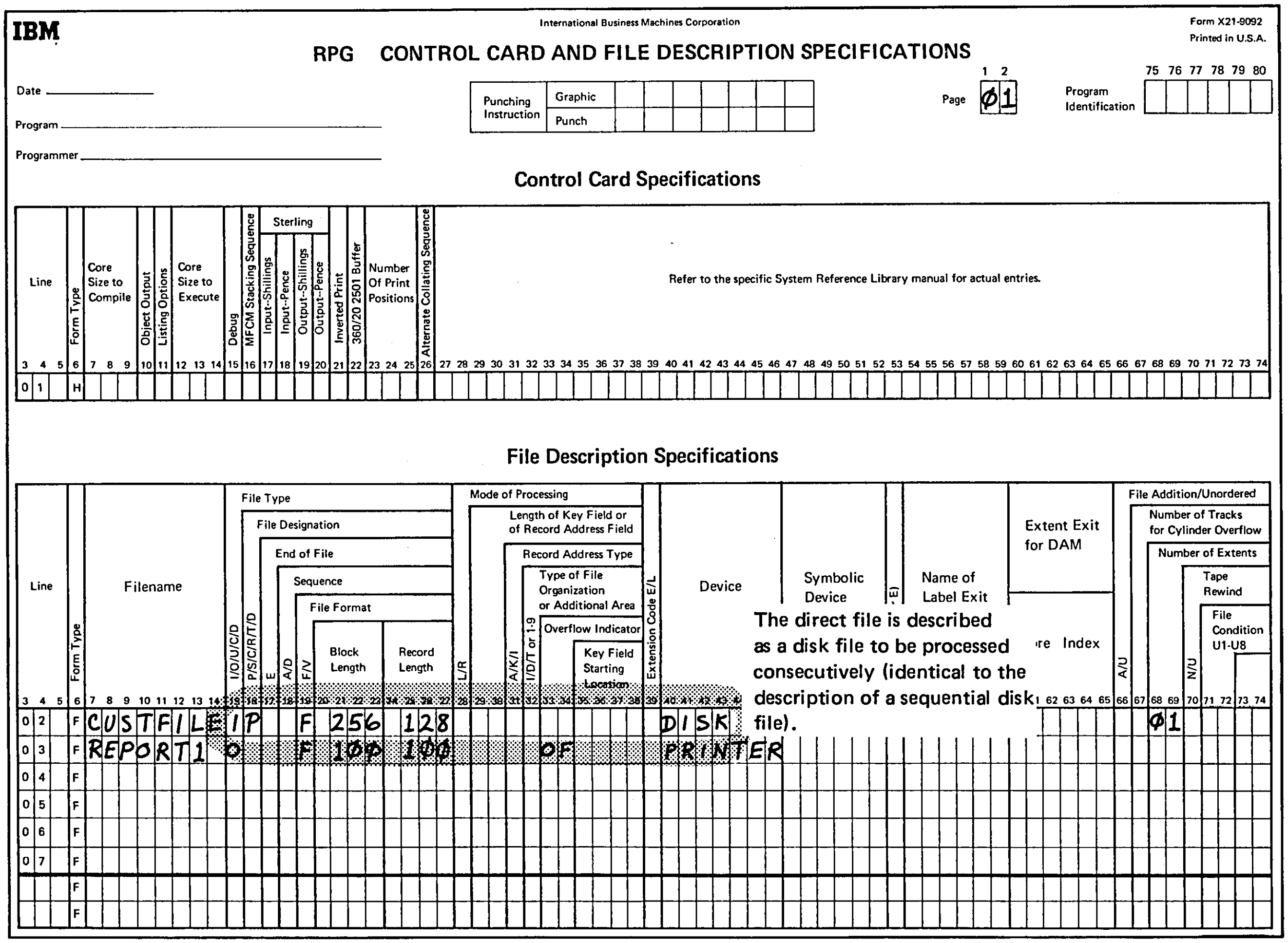


```
CUSTOMER         NAME              CITY,STATE         SALESMAN   LAST ORDER   SLS PREV PER   CRDT   TOT A/R

  1637    JONES VARIETY        BEDROCK,TEX              15        4/13/71         240.37      01         .00
  2279    GREEN GROCERY,INC    BIG CITY,CALIF          102        4/27/71        1200.00      01      600.00
  2331    STAR MARKET          GOODTOWN,GA              74        4/01/71          31.95      03      937.16
```

Figure 2-3. Report of Recently Inactive Customers from Consecutive Processing of a Direct File

Figure 2-4 shows the specification sheets necessary to consecutively retrieve records from CUSTFILE to produce REPORT1, a list of recently inactive customers.

Since the direct file probably contains blank record locations and inactive records, a technique is employed on the Input Sheet to bypass such records (Figure 2-4). If a method is not used to bypass unidentified records, the program will halt when they are encountered.



IBM International Business Machines Corporation Form X21-9092 Printed in U.S.A.

RPG CONTROL CARD AND FILE DESCRIPTION SPECIFICATIONS

Date

Program

Programmer

Punching Instruction: Graphic / Punch

Page Ø1

Program Identification

Control Card Specifications

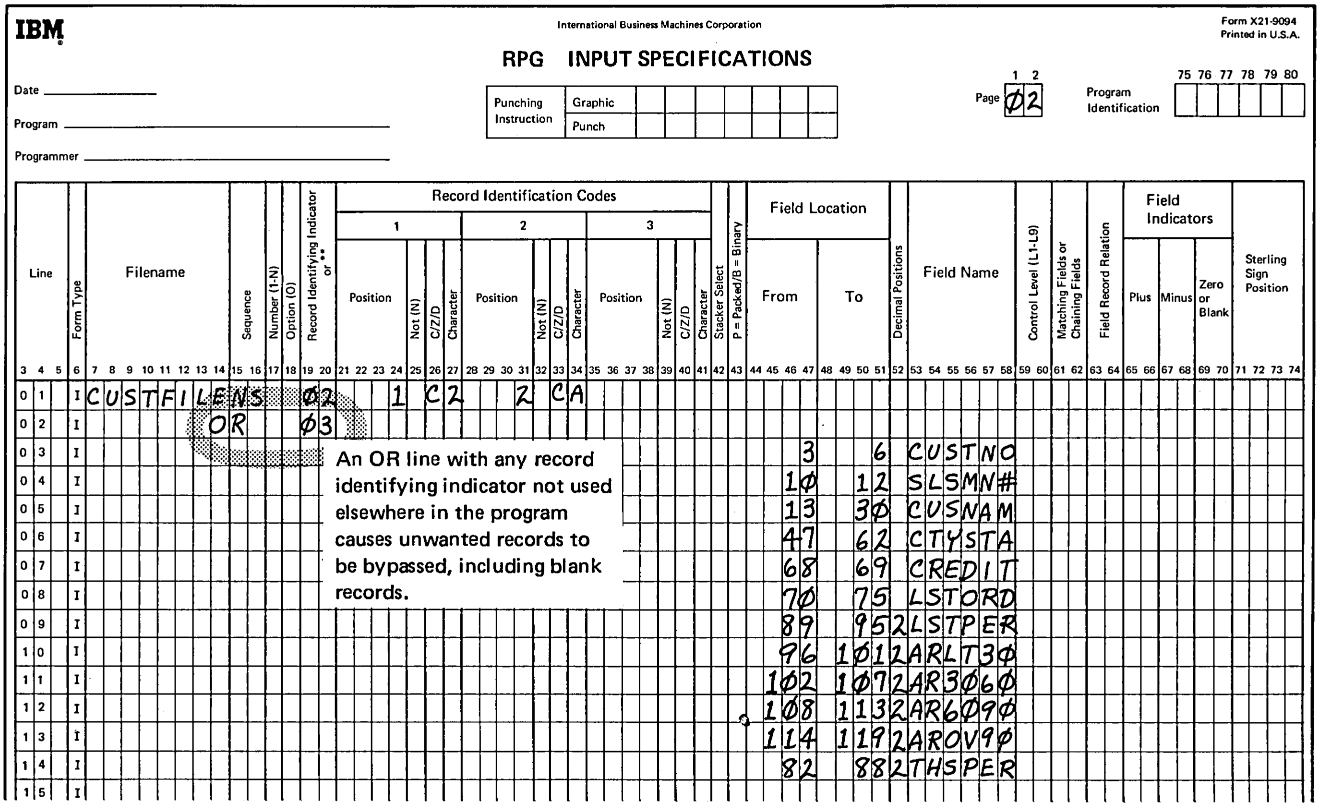
Line	Form Type	Core Size to Compile	Object Output	Listing Options	Core Size to Execute	Debug	MFCM Stacking Sequence	Input-Shillings	Input-Pence	Output-Shillings	Output-Pence	Inverted Print	360/20 2501 Buffer	Number Of Print Positions	Alternate Collating Sequence	Refer to the specific System Reference Library manual for actual entries.
01	H															

File Description Specifications

Line	Form Type	Filename	I/O/U/C/D	P/S/C/R/T/D	E	A/D	F/V	Block Length	Record Length	L/R	A/K/I	I/D/T or 1-9	Overflow Indicator	Key Field Starting Location	Extension Code E/L	Device	Symbolic Device	Name of Label Exit	Extent Exit for DAM	A/U	Number of Tracks for Cylinder Overflow	Number of Extents	Tape Rewind	File Condition U1-U8
02	F	CUSTFILE	I	P			F	256	128							DISK						Ø1		
03	F	REPORT1	O				F	1ØØ	1ØØ				OF			PRINTER								
04	F																							
05	F																							
06	F																							
07	F																							
	F																							
	F																							

The direct file is described as a disk file to be processed consecutively (identical to the description of a sequential disk file).

Figure 2-4. Consecutive Retrieval From a Direct File (part 1 of 4)



IBM

International Business Machines Corporation

Form X21-9094
Printed in U.S.A.

RPG INPUT SPECIFICATIONS

Date

Program

Programmer

Punching Instruction	Graphic								
	Punch								

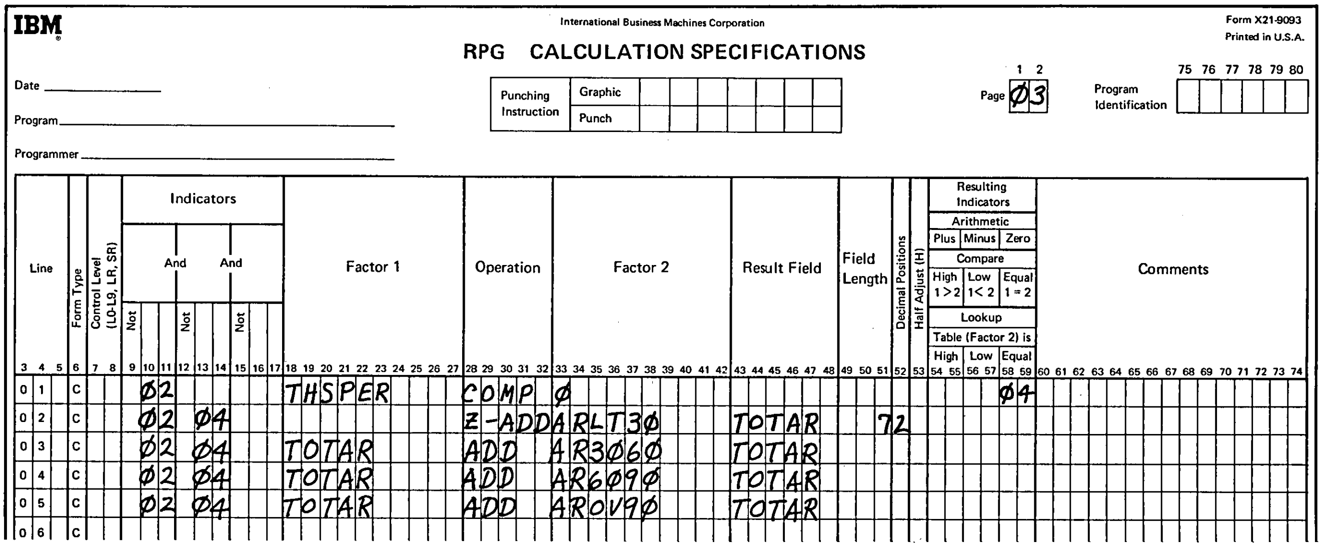
Page 1 2: Ø2

Program Identification 75 76 77 78 79 80

Line	Form Type	Filename	Sequence	Number (1-N)	Option (O)	Record Identifying Indicator or **	1 Position	Not (N)	C/Z/D	Character	2 Position	Not (N)	C/Z/D	Character	3 Position	Not (N)	C/Z/D	Character	Stacker Select	P = Packed/B = Binary	Field Location From	Field Location To	Decimal Positions	Field Name	Control Level (L1-L9)	Matching Fields or Chaining Fields	Field Record Relation	Field Indicators Plus	Field Indicators Minus	Field Indicators Zero or Blank	Sterling Sign Position
3 4 5	6	7 8 9 10 11 12 13 14	15 16	17	18	19 20	21 22 23 24	25	26	27	28 29 30 31	32	33	34	35 36 37 38	39	40	41	42	43	44 45 46 47	48 49 50 51	52	53 54 55 56 57 58	59 60	61 62	63 64	65 66	67 68	69 70	71 72 73 74
0 1	I	CUSTFILE	NS			Ø2	1		C	2	2		C	A																	
0 2	I	OR				Ø3																									
0 3	I																				3	6		CUSTNO							
0 4	I																				1Ø	12		SLSMN#							
0 5	I																				13	3Ø		CUSNAM							
0 6	I																				47	62		CTYSTA							
0 7	I																				68	69		CREDIT							
0 8	I																				7Ø	75		LSTORD							
0 9	I																				89	95	2	LSTPER							
1 0	I																				96	1Ø1	2	ARLT3Ø							
1 1	I																				1Ø2	1Ø7	2	AR3Ø6Ø							
1 2	I																				1Ø8	113	2	AR6Ø9Ø							
1 3	I																				114	119	2	AROV9Ø							
1 4	I																				82	88	2	THSPER							
1 5	I																														

An OR line with any record identifying indicator not used elsewhere in the program causes unwanted records to be bypassed, including blank records.

Figure 2-4. Consecutive Retrieval From a Direct File (part 2 of 4)



IBM

International Business Machines Corporation

Form X21-9093
Printed in U.S.A.

RPG CALCULATION SPECIFICATIONS

Date

Program

Programmer

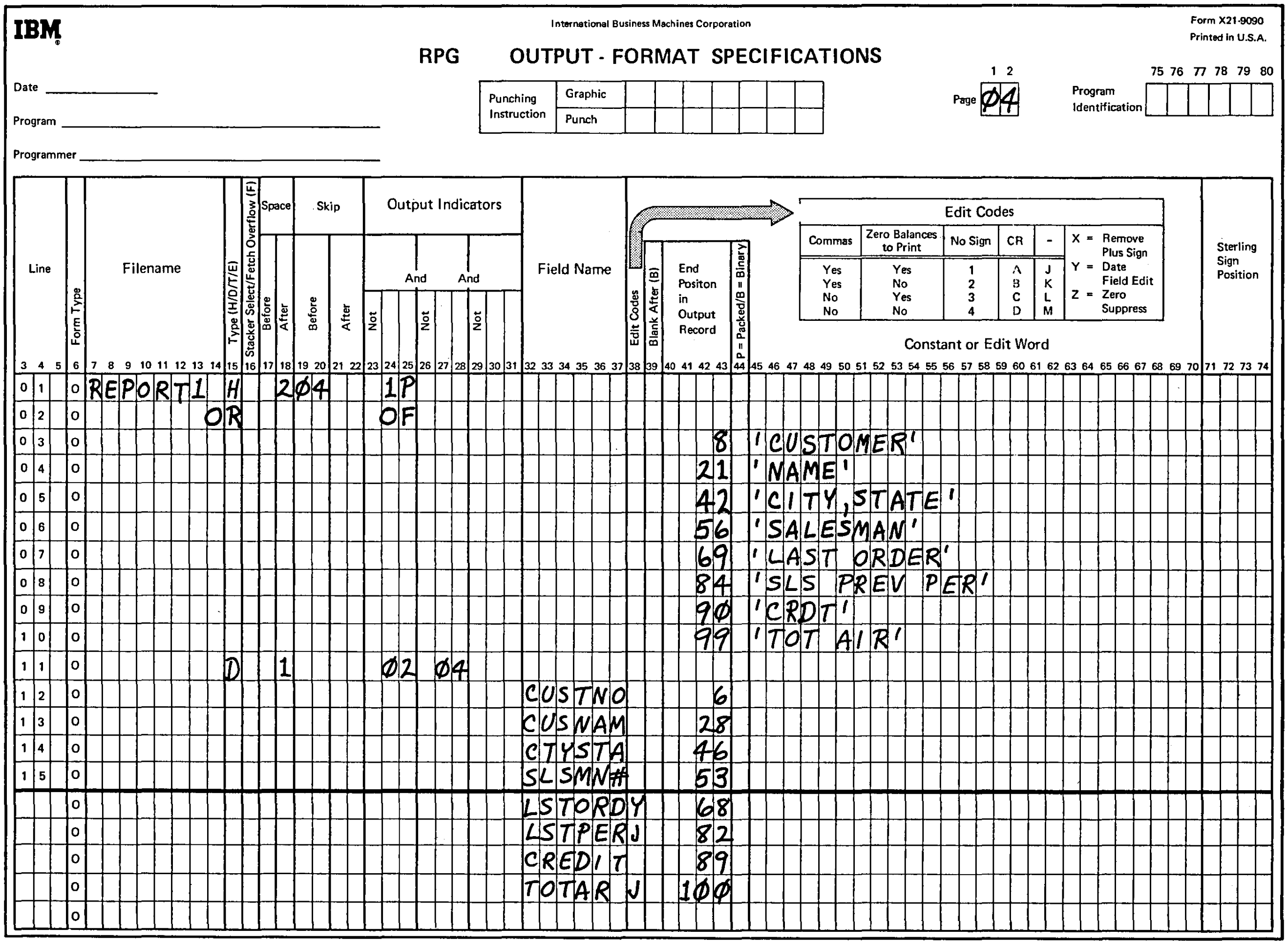
Punching Instruction	Graphic								
	Punch								

Page 1 2: Ø3

Program Identification 75 76 77 78 79 80

Line	Form Type	Control Level (L0-L9, LR, SR)	Indicators Not	Indicators	And Not	And	And Not	And	Factor 1	Operation	Factor 2	Result Field	Field Length	Decimal Positions	Half Adjust (H)	Resulting Indicators Arithmetic Plus / Compare High 1>2 / Lookup Table (Factor 2) is High	Minus / Low 1<2 / Low	Zero / Equal 1=2 / Equal	Comments
3 4 5	6	7 8	9	10 11	12	13 14	15	16 17	18 19 20 21 22 23 24 25 26 27	28 29 30 31 32	33 34 35 36 37 38 39 40 41 42	43 44 45 46 47 48	49 50 51	52	53	54 55	56 57	58 59	60 61 62 63 64 65 66 67 68 69 70 71 72 73 74
0 1	C			Ø2					THSPER	COMP	Ø							Ø4	
0 2	C			Ø2		Ø4				Z-ADD	ARLT3Ø	TOTAR	7	2					
0 3	C			Ø2		Ø4			TOTAR	ADD	AR3Ø6Ø	TOTAR							
0 4	C			Ø2		Ø4			TOTAR	ADD	AR6Ø9Ø	TOTAR							
0 5	C			Ø2		Ø4			TOTAR	ADD	AROV9Ø	TOTAR							
0 6	C																		

Figure 2-4. Consecutive Retrieval From a Direct File (part 3 of 4)



IBM

International Business Machines Corporation

RPG OUTPUT - FORMAT SPECIFICATIONS

Form X21-9090
Printed in U.S.A.

Page Ø4

Date
Program
Programmer

Punching Instruction: Graphic / Punch

Program Identification

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Line	Form Type	Filename	Type (H/D/T/E)	Space After	Skip Before	Output Indicators	Field Name	Edit Codes	End Position in Output Record	Constant or Edit Word
01	O	REPORT1	H	2	Ø4	1P				
02	O	OR				OF				
03	O								8	'CUSTOMER'
04	O								21	'NAME'
05	O								42	'CITY,STATE'
06	O								56	'SALESMAN'
07	O								69	'LAST ORDER'
08	O								84	'SLS PREV PER'
09	O								9Ø	'CRDT'
10	O								99	'TOT A/R'
11	O		D	1		Ø2 Ø4				
12	O						CUSTNO		6	
13	O						CUSNAM		28	
14	O						CTYSTA		46	
15	O						SLSMN#		53	
	O						LSTORD	Y	68	
	O						LSTPER	J	82	
	O						CREDIT		89	
	O						TOTAR	J	1ØØ	
	O									

Figure 2-4. Consecutive Retrieval From a Direct File (part 4 of 4)

Consecutive Updating of a Direct File

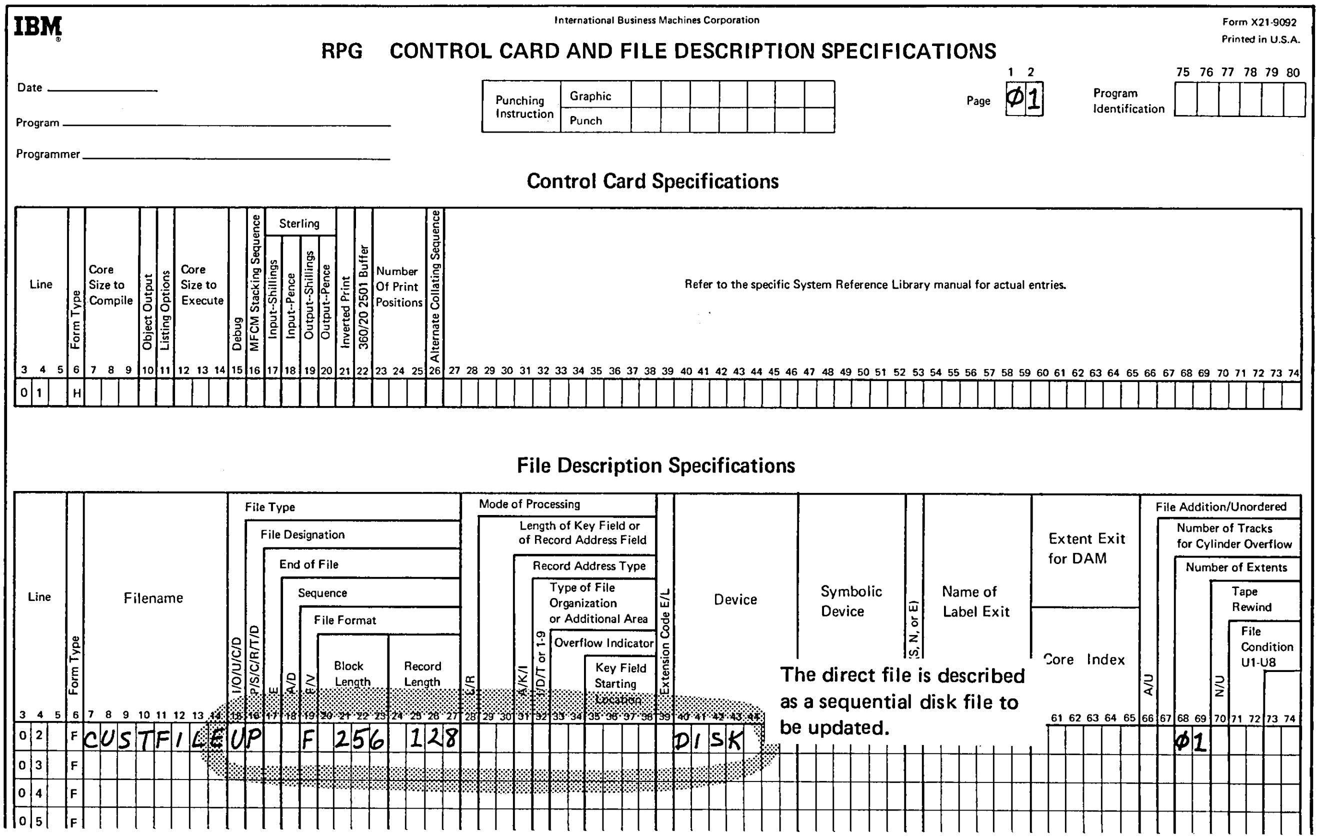
In the preceding example of consecutive retrieval, none of the fields in CUSTFILE were modified; data was taken from the disk records and used to produce a report. If, on the other hand, you want to modify certain data in the disk records, you must use the update function. If all or most of the records in the direct file are to be processed, you may want to update the file consecutively.

Consecutive updating of records in a direct file requires the same File Description Sheet entries as consecutive updating of a sequential file. The required entries for updating are shown in Figure 2-1 (U in column 15 means update). The file named in columns 7-14 must be a previously created direct file. Because the file is an update file, it must also be defined with input and output-format specifications.

Example 2

Suppose the direct customer file created in Chapter 1 and retrieved consecutively in *Example 1* is to be updated. At the end of each sales period, when all reports are completed, the sales figures for that period must be adjusted. Sales amounts for the last period (LSTPER, Figure 2-2) are replaced by the sales amounts from the current period (THSPER). The field containing the current sales amount is reset to zero for the accumulation of the next selling period. The fields containing accounts receivable overdue amounts will be updated when the monthly accounts receivable statements are written.

Figure 2-5 shows the coding necessary to consecutively update CUSTFILE. As an update file, CUSTFILE must be defined by file description specifications and the fields to be updated must be described by input and output specifications. Customer records are read, updated, and written out in the order they physically appear in the direct file.



IBM International Business Machines Corporation Form X21-9092 Printed in U.S.A.

RPG CONTROL CARD AND FILE DESCRIPTION SPECIFICATIONS

Date ______ Program ______ Programmer ______

Punching Instruction: Graphic / Punch

Page (1 2): Ø1 Program Identification (75–80)

Control Card Specifications

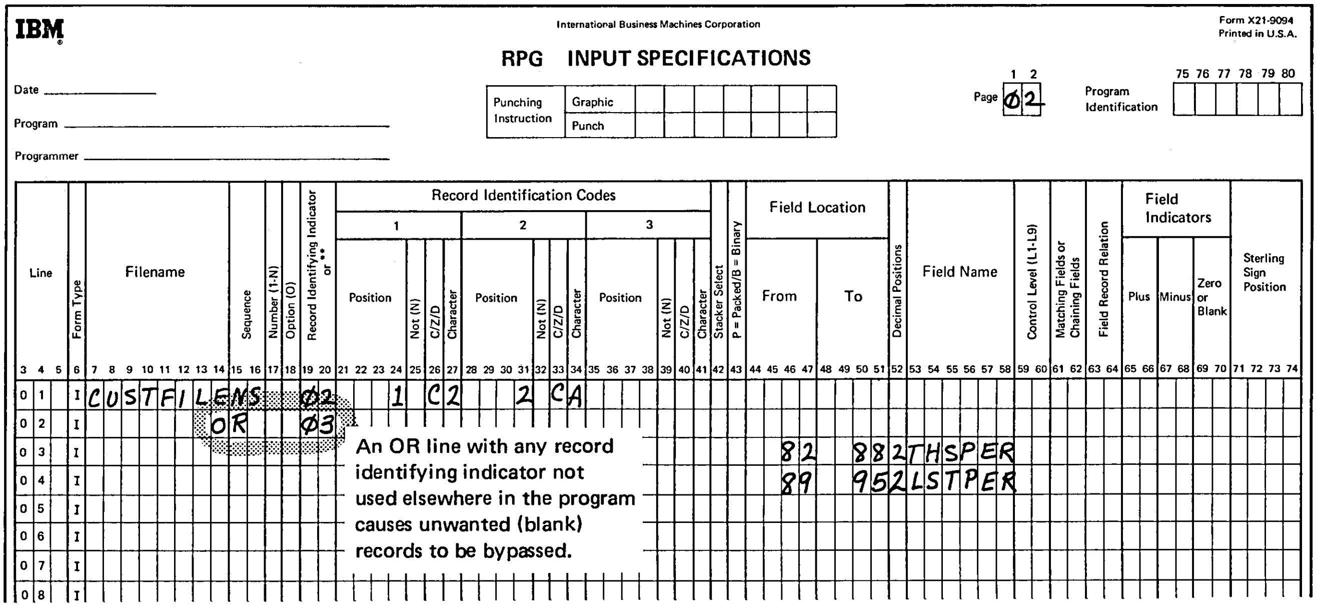
Line (3–5)	Form Type (6)	Refer to the specific System Reference Library manual for actual entries.
01	H	

File Description Specifications

Line (3–5)	Form Type (6)	Filename (7–14)	File Type (15)	File Designation (16)	End of File (17)	Sequence (18)	File Format (19)	Block Length (20–23)	Record Length (24–27)	Device (40–46)	Number of Extents (68–69)
02	F	CUSTFILE	U	P			F	256	128	DISK	Ø1
03	F										
04	F										
05	F										

The direct file is described as a sequential disk file to be updated.

Figure 2-5. Consecutive Update of a Direct File (part 1 of 4)



IBM International Business Machines Corporation Form X21-9094 Printed in U.S.A.

RPG INPUT SPECIFICATIONS

Date ______ Program ______ Programmer ______

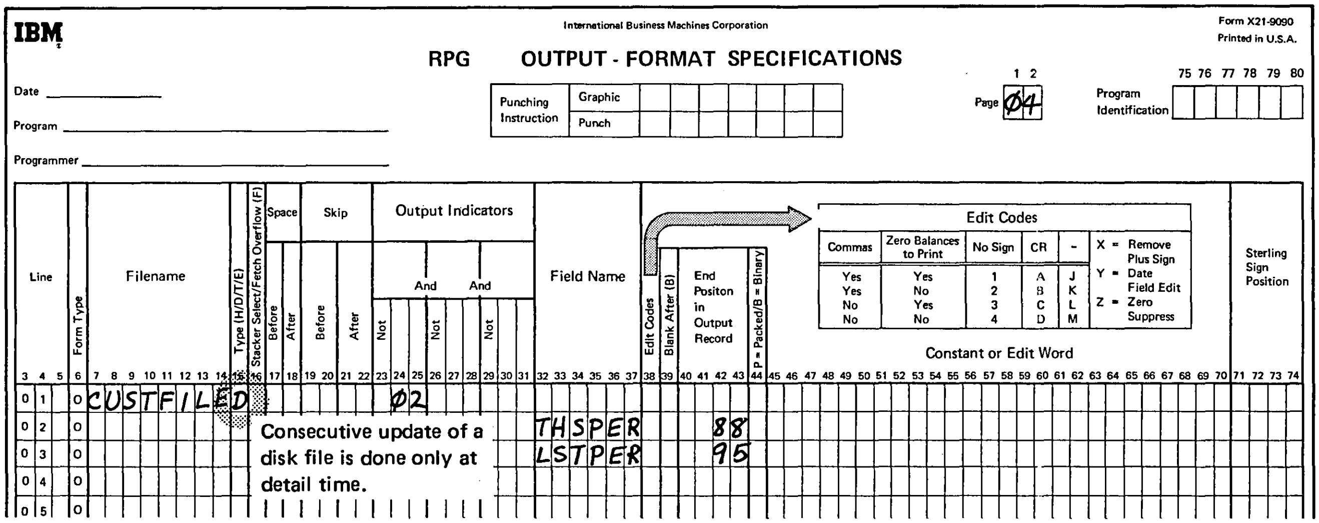
Punching Instruction: Graphic / Punch

Page (1 2): Ø2 Program Identification (75–80)

Line (3–5)	Form Type (6)	Filename (7–14)	Sequence (15–16)	Record Identifying Indicator (19–20)	Position 1 (21–24)	C/Z/D (26)	Character (27)	Position 2 (28–31)	C/Z/D (33)	Character (34)	From (44–47)	To (48–51)	Decimal Positions (52)	Field Name (53–58)
01	I	CUSTFILE	NS	Ø2	1	C	2	2	C	A				
02	I		OR	Ø3										
03	I										82	88	2	THSPER
04	I										89	95	2	LSTPER
05	I													
06	I													
07	I													
08	I													

An OR line with any record identifying indicator not used elsewhere in the program causes unwanted (blank) records to be bypassed.

Figure 2-5. Consecutive Update of a Direct File (part 2 of 4)



IBM — International Business Machines Corporation

RPG OUTPUT - FORMAT SPECIFICATIONS

Form X21-9090
Printed in U.S.A.

Date ______
Program ______
Programmer ______

Punching Instruction: Graphic / Punch

Page Ø4

Program Identification

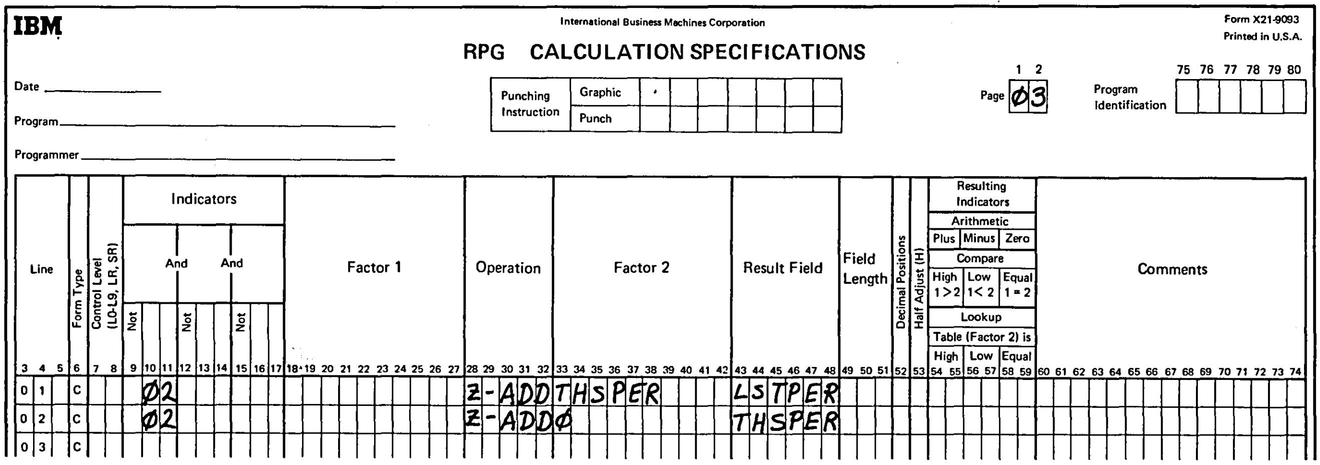
Line	Form Type	Filename	Type (H/D/T/E)	Output Indicators	Field Name	End Position in Output Record
01	O	CUSTFILE	D	Ø2		
02	O				THSPER	88
03	O				LSTPER	95
04	O					
05	O					

Consecutive update of a disk file is done only at detail time.

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Figure 2-5. Consecutive Update of a Direct File (part 3 of 4)



IBM — International Business Machines Corporation

RPG CALCULATION SPECIFICATIONS

Form X21-9093
Printed in U.S.A.

Date ______
Program ______
Programmer ______

Punching Instruction: Graphic / Punch

Page Ø3

Program Identification

Line	Form Type	Indicators	Factor 1	Operation	Factor 2	Result Field
01	C	Ø2		Z-ADD	THSPER	LSTPER
02	C	Ø2		Z-ADD	Ø	THSPER
03	C					

Figure 2-5. Consecutive Update of a Direct File (part 4 of 4)

Review 2

1. What points must you consider when processing a direct file consecutively?

2. What is the difference between coding for retrieval of a direct file consecutively and a sequential file consecutively?

Answers to Review 2

1. a. There will probably be blank and inactive records to be bypassed.

   b. The physical sequence of the file may or may not be meaningful as a logical sequence.

   c. You may encounter synonym records and must take steps to allow for them.

   d. The activity of the file for this run will probably determine whether to process randomly or consecutively.

2. Those coding routines required to take care of blanks and synonyms must be added. The File Description and Input Sheets will be identical.

CHAPTER 3

RANDOM PROCESSING OF DIRECT AND SEQUENTIAL FILES

CHAPTER 3 DESCRIBES:

When random processing is desirable.

Random processing by relative record number.

How to code the RPG II specification sheets to process direct and sequential files randomly.

BEFORE READING THIS CHAPTER YOU SHOULD BE ABLE TO:

Code basic RPG II programs using sequential and indexed files.

Describe random processing.

Describe, in concept, direct file organization.

Define synonym record.

AFTER READING THIS CHAPTER YOU SHOULD BE ABLE TO:

State the conditions for selecting random processing by relative record number.

Describe, in concept, random processing by relative record number.

Code RPG II specification sheets to randomly retrieve records from direct and sequential files.

Code RPG II specification sheets to randomly update records in direct and sequential files.

INTRODUCTION

Note: In a data processing context, the words *random* and *direct* are often used interchangeably. The word random, in this context, does not have the same meaning as it has in the term *random number,* which is a number obtained by chance. Here, *random processing* means that data records are accessed directly in a file, without regard to the physical or logical sequence of the records and without accessing other records first.

Sequential files and direct files are designed for different methods of processing. Records in a sequential file are normally processed consecutively; whereas records in a direct file are normally processed randomly. However, basic similarities exist between the two file organizations. Both consist of a data area containing only record locations, and neither uses an index. Both sequential and direct files can be processed consecutively. In fact, as shown in Chapter 2, the File Description Sheet entries for consecutive processing are identical for the two files. Because of the similarity in organization, sequential files, like direct files, can sometimes be processed randomly.

WHEN TO PROCESS DIRECT AND SEQUENTIAL FILES RANDOMLY

Direct files are organized for low activity processing. In low activity processing, a relatively low percentage of records from a file are processed at one time. For example, perhaps only 100 records per day out of 5000 are processed. Efficiency is reduced when you must read all the records in the file just to process a few, so the file is processed randomly.

Random processing enables you to process only the records you want, disregarding all other records. This immediate access is important when a file is processed on a demand basis, requiring immediate inquiry and response. Random processing also enables you to process transactions on either an ordered or an unordered basis, increasing processing versatility and eliminating the need for presorting inquiries and transactions.

Considerations

Remember that most advantages of random processing are available either with direct files or indexed files. A significant consideration when using direct file organization is that you must develop relative record numbers. If a simple method is used that produces few synonyms, direct organization can have the advantages of speed and space-saving over indexed organization. In any case, you must provide the relative record number, handle synonyms, and verify records yourself. If these functions become complex or require too much main storage, you might want to consider consecutive processing or indexed organization.

The opportunities for random processing of sequential disk files are limited. If the sequential file is in order by control fields and there are no missing or duplicate records, the contents of the control fields can be used as relative record numbers. This is a secondary use, however, since sequential files are designed for high-activity consecutive processing.

RELATIVE RECORD NUMBERS

Random processing of direct and sequential files is done by using relative record numbers. A relative record number is the numeric position of a record in relation to the beginning of the file. A relative record number must be a positive, whole number.

RANDOM PROCESSING BY RELATIVE RECORD NUMBER

Random processing of indexed files is accomplished by using the control field value (record key) to search an index. If a match is found, the record at the disk location contained in the index entry can be accessed. The control field value, therefore, is not related to the actual location of the record on disk. When processing randomly by relative record number, however, the relative record number is used by disk system management to calculate the disk location of the record. No index area and index search are required, since the control field value is directly related to the record location. Therefore, random processing by relative record number can be faster than random processing by key of an indexed file. If a large number of synonyms exist in the file, the advantage of fewer accessed required to retrieve a record may be negated by more complicated programming to handle synonyms and an increase in the average number of seeks per record due to synonyms (see *Synonym Records* in Chapter 1 for ways to handle synonym records).

Random processing by relative record number can be used to retrieve and update direct and sequential disk files. With either organization, the file is specified as a chained file to be processed by the CHAIN operation code. Records can be processed either in an ordered or an unordered manner. Processing of records in sequence is usually faster than unordered processing, since less movement of the disk access mechanism is required. Figure 3-1 shows the steps involved in random processing of a disk file by relative record number. In the figure, relative record numbers are obtained for control fields in the input records; however, they could also be generated by your program. Random retrieval includes steps one, two, and three in the figure; random update includes all five steps.

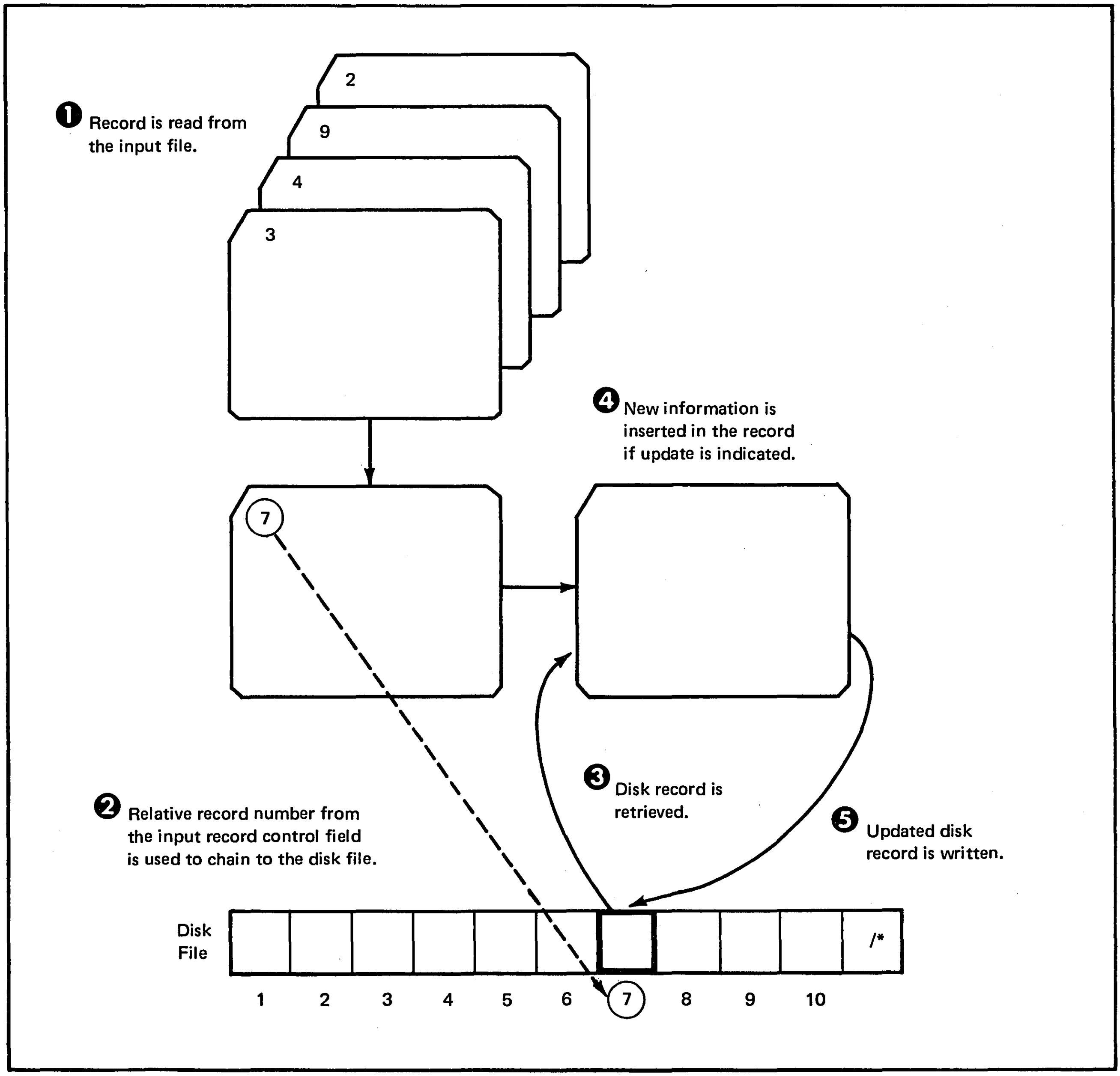

❶ Record is read from the input file.
2
9
4
3
7
❹ New information is inserted in the record if update is indicated.
❷ Relative record number from the input record control field is used to chain to the disk file.
❸ Disk record is retrieved.
❺ Updated disk record is written.
Disk File
/*
1
2
3
4
5
6
7
8
9
10

Figure 3-1. Random Processing by Relative Record Number (Direct or Sequential Disk File)

CODING FOR RANDOM PROCESSING OF DIRECT AND SEQUENTIAL FILES

Figure 3-2 shows the basic file description specifications for random processing by relative record number. The entries shown apply to both direct and sequential files.

Columns 7-14: These columns must contain the disk filename.

Column 15: This column contains an *I* entry for random retrieval or a *U* entry for random update. An *O* entry in this column with a *C* in column 16 defines a chained output file (see *How to Create a Direct File* in Chapter 1).

Column 16: The *C* in this column indicates a chained file processed randomly by the CHAIN operation code. A maximum of 15 chained files are allowed per program.

Column 19: Must contain an *F*.

Columns 20-23: A number which is equal to or a multiple of the disk record length must be entered in these columns. This entry affects the size of the input/output area allocated by RPG II. The maximum block length for disk files in 4096. If you assign a block length which is equal to the record length, an efficient block length is calculated for you by RPG II (Figure 3-3). Blocking disk records can increase the input/output efficiency of your program by reducing the number of accesses. You must be sure, however, that you have enough main storage available for your input/output area.

Columns 24-27: These columns contain the length of the disk record (1-4096). Remember that random update cannot change the record length for a file; record length is fixed when the file is created.

Column 28: This column must contain an *R* for random processing.

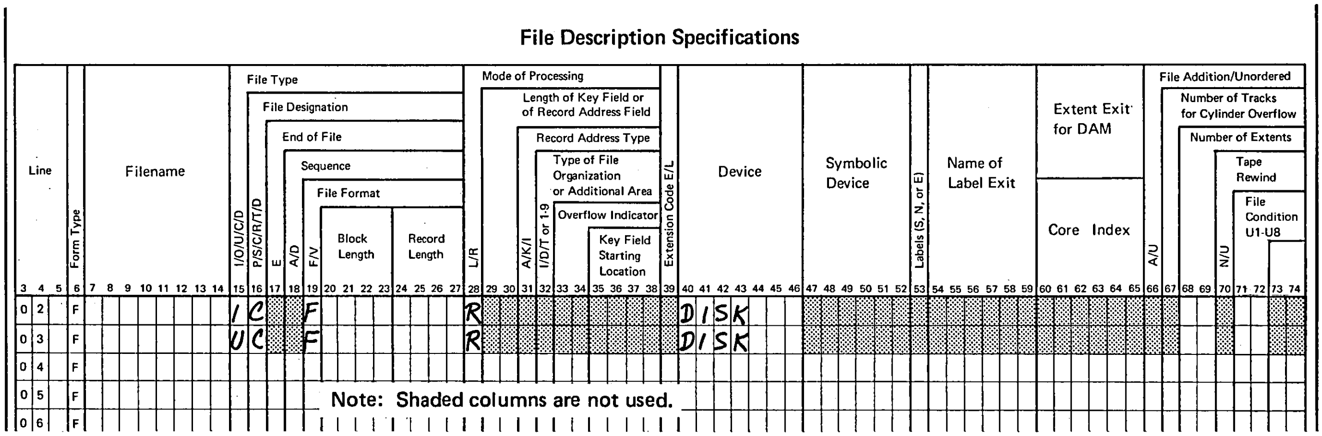
Columns 40-46: These columns contain the device name, DISK.

Columns 68-69: These columns give the number of volumes containing the file. For random processing, two volumes are allowed on a single drive and four volumes are allowed on two drives. All volumes must be online.

Columns 71-72: Direct and sequential files can be conditioned by a U1-U8 external indicator.

Record length	Block length computed by RPG II	Input/output area allocated by RPG II	Number of records per block
32	256	256	8
60	240	512	4
64	256	256	4
80	240	512	3
96	192	512	2
128	256	256	2
256	256	256	1
512	512	512	1

Figure 3-3. Block Length and Size of Input/Output Area Computed by RPG II for Random Processing By Relative Record Number



File Description Specifications

Line (3-5)	Form Type (6)	Filename (7-14)	File Type (15)	File Designation (16)	End of File (17)	Sequence (18)	File Format (19)	Block Length (20-23)	Record Length (24-27)	Mode of Processing (28)	Device (40-46)
02	F		I	C			F			R	DISK
03	F		U	C			F			R	DISK
04	F										
05	F										
06	F										

Note: Shaded columns are not used.

Figure 3-2. Basic File Description Specifications for Random Processing of Direct and Sequential Files

CHAIN Operation

In direct and sequential files, records to be accessed randomly are identified by relative record numbers in CHAIN statements. One record is read for each CHAIN statement executed. Records identified in CHAIN statements are read during calculations in the program cycle. Fields from the records can be used during detail or total calculations. For example, a record read during detail calculations can be altered during detail calculations and written out during detail output. The same applies to total calculations and total output.

Figure 3-4 shows the entries to be made in a CHAIN statement.

Columns 7-17: Indicators can be used.

Columns 18-27: Factor 1 entry is either the name of the field containing the relative record number or the relative record number itself.

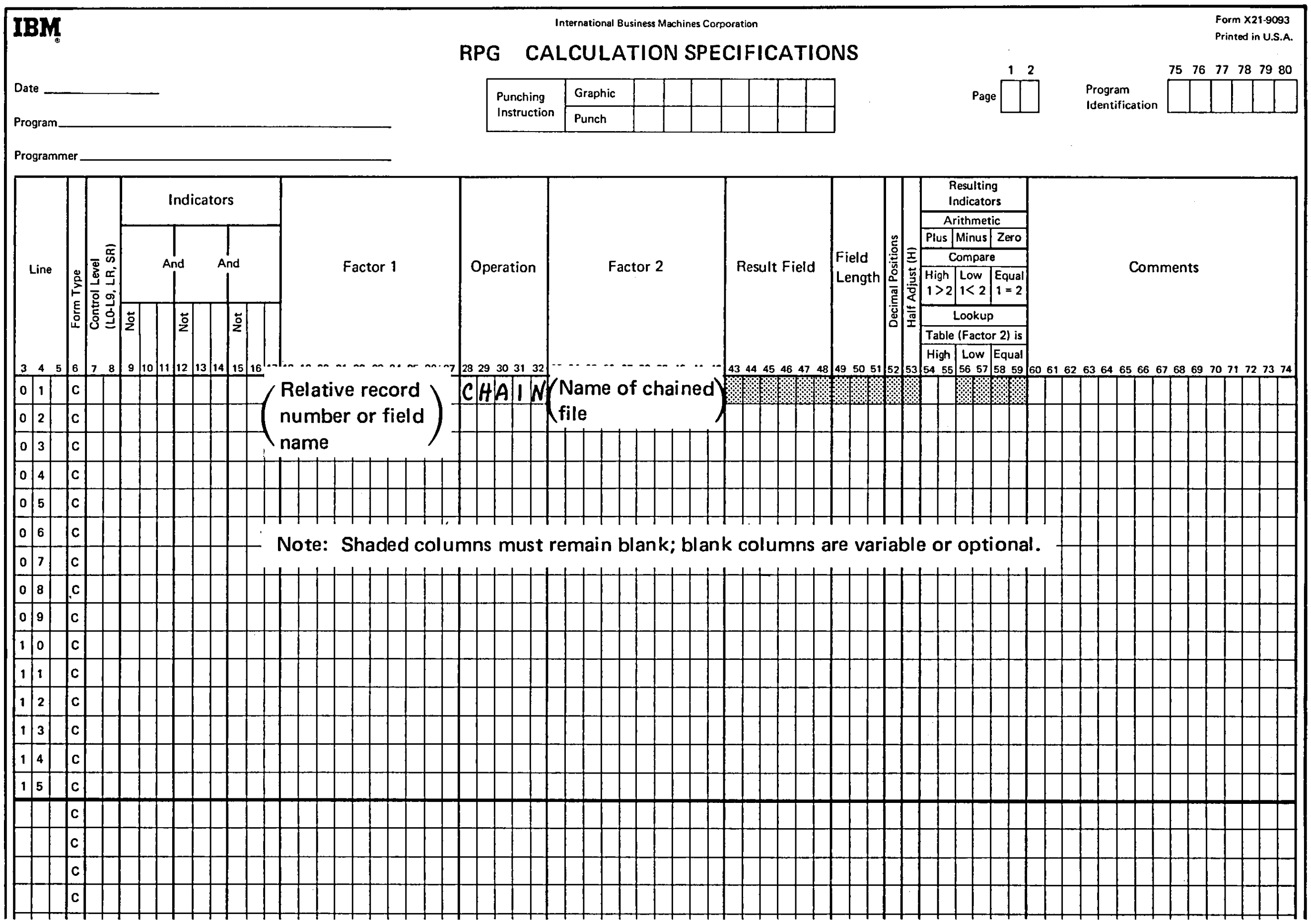
Columns 33-42: Factor 2 entry is the name of the file from which a record is read.

Columns 54-55: An indicator entered in these columns will be turned on if the record is not found. This condition occurs when:

1. The relative record number is zero or negative.

2. The relative record number is greater than the number of record locations in the file.

If the indicator in columns 54-55 is on, the chained file cannot be updated. If an indicator is not specified, the program will halt when a record is not found, displaying a 1U halt code. The program can be restarted by pressing HALT/RESET on the processing unit. Use of an indicator in columns 54-55 is recommended, because, if the bypass option is selected after a 1U halt, the next record may not be read from the same file. Therefore, the results of the bypass option may not be predictable.



IBM

International Business Machines Corporation

RPG CALCULATION SPECIFICATIONS

Form X21-9093
Printed in U.S.A.

Date

Program

Programmer

Punching Instruction: Graphic / Punch

Page 1 2

Program Identification 75 76 77 78 79 80

Line (3-5)	Form Type (6)	Control Level (L0-L9, LR, SR) (7-8)	Indicators (9-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field (43-48)	Field Length (49-51)	Decimal Positions (52)	Half Adjust (H) (53)	Resulting Indicators (54-59)	Comments (60-74)
01	C			Relative record number or field name	CHAIN	Name of chained file	▨	▨	▨	▨	54-55 / ▨ / ▨	
02	C											
03	C											
04	C											
05	C											
06	C											
07	C											
08	C											
09	C											
10	C											
11	C											
12	C											
13	C											
14	C											
15	C											
	C											
	C											
	C											
	C											

Resulting Indicators: Arithmetic — Plus, Minus, Zero; Compare — High 1>2, Low 1<2, Equal 1=2; Lookup Table (Factor 2) is — High, Low, Equal

Note: Shaded columns must remain blank; blank columns are variable or optional.

Figure 3-4. CHAIN Statement for Random Processing by Relative Record Number

Random Retrieval From a Direct File

The entries on the File Description Sheet that are characteristic of random retrieval are shown in the first coding line on Figure 3-2. Random retrieval is distinguished from random update by an *I* in column 15.

The records in the direct file to be retrieved must be further described on input specifications. On the Input Sheet, a direct file being retrieved must have an alphabetic sequence entry (columns 15-16), since sequence checking cannot be done during random processing.

If the direct file being retrieved contains synonym records, calculations must be included in the program to test for synonyms and retrieve the desired record.

Example 1

Suppose the direct customer file, CUSTFILE, created in Chapter 1 (*How to Create a Direct File*) and processed consecutively in Chapter 2 is to be retrieved randomly. The distributor wants to make demand inquiries as necessary during each day concerning customer sales and account information. Inquiries are received on cards (Figure 3-5) containing an *I* in column one followed by the customer number of the record to be retrieved. These cards are read from the primary MFCU hopper. Each time an inquiry card is read, the customer number (CSTMER) is used as the relative record number to chain to CUSTFILE.

The format of the disk records in CUSTFILE is shown in Figure 3-6.

If a record is found in CUSTFILE which corresponds to the number on the inquiry card, a response is printed in the format shown in Figure 3-7. This response lists pertinent sales information and the total accounts receivable amount.

The RPG II coding to accomplish the random inquiry application is shown in Figure 3-8. The RPG II specifications would be identical if CUSTFILE were a sequential disk file to be retrieved randomly.

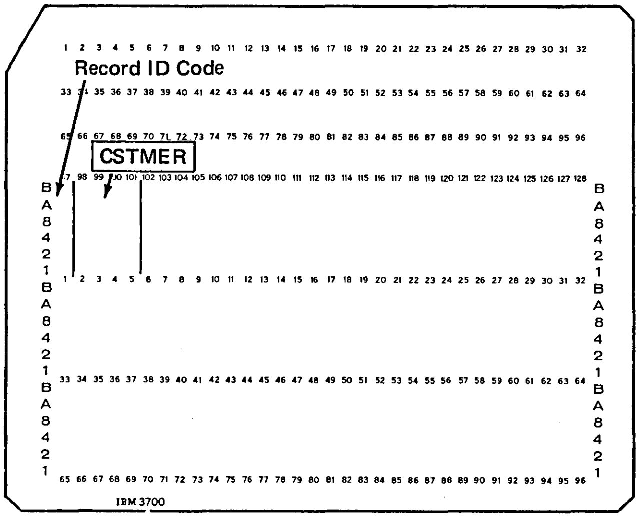
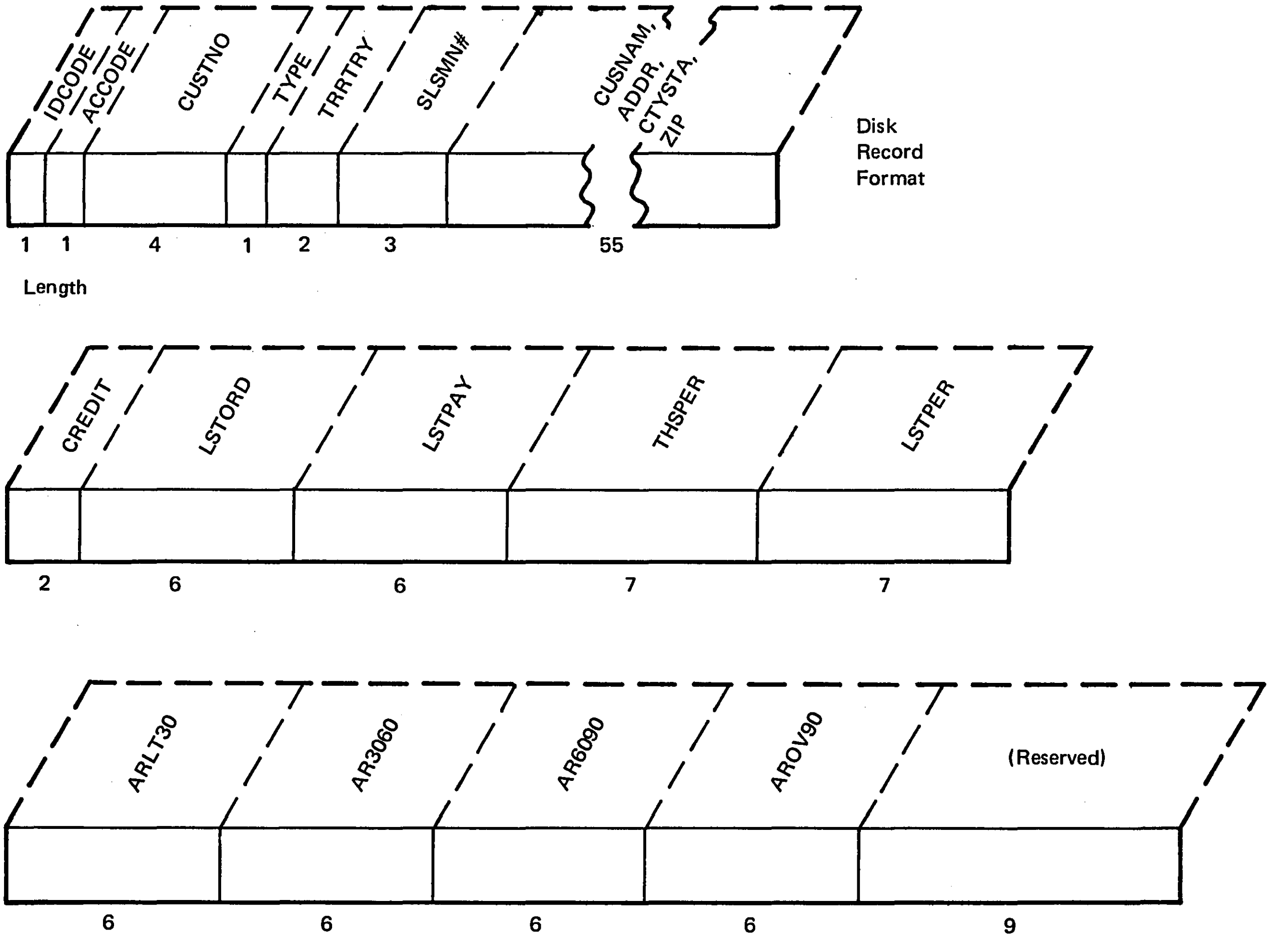
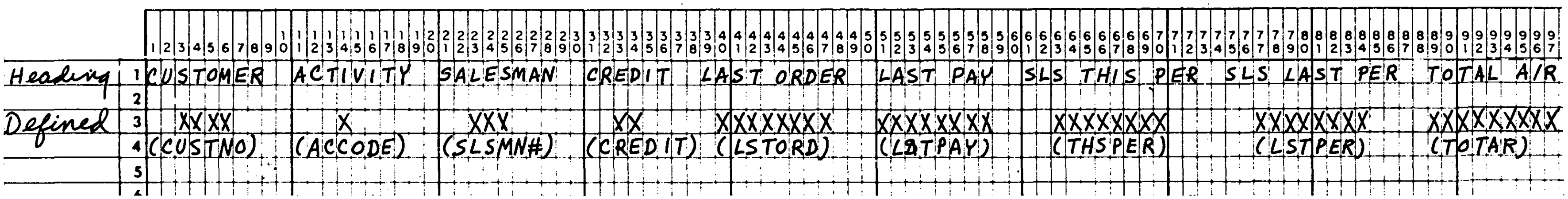


Figure 3-5. Inquiry Card Format

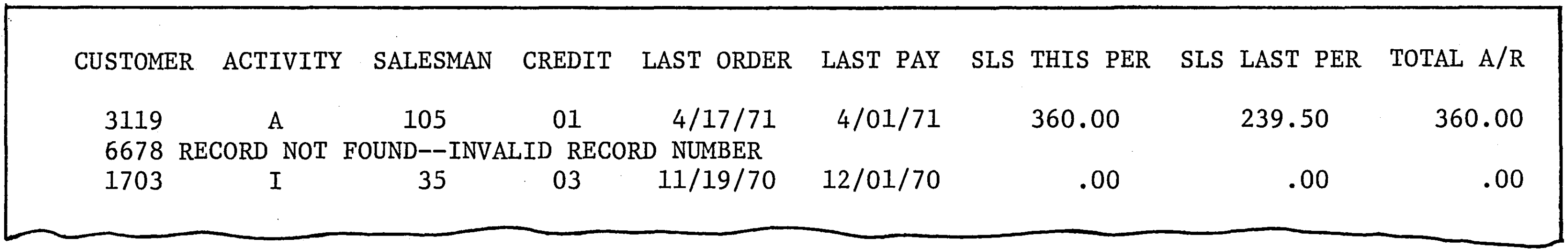


Total length of record = 128 positions

Figure 3-6. Disk Record Format for Direct Customer File, CUSTFILE

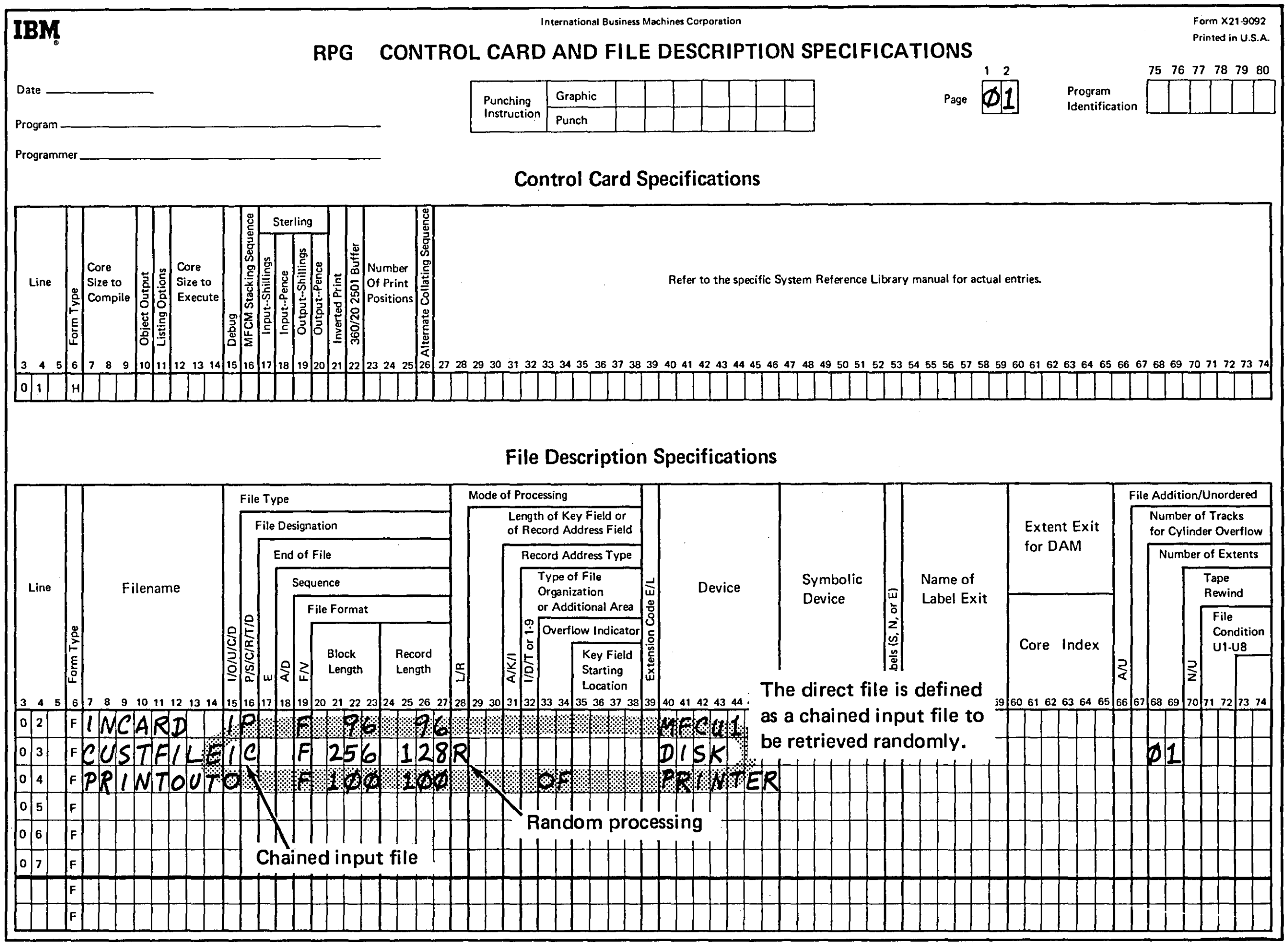


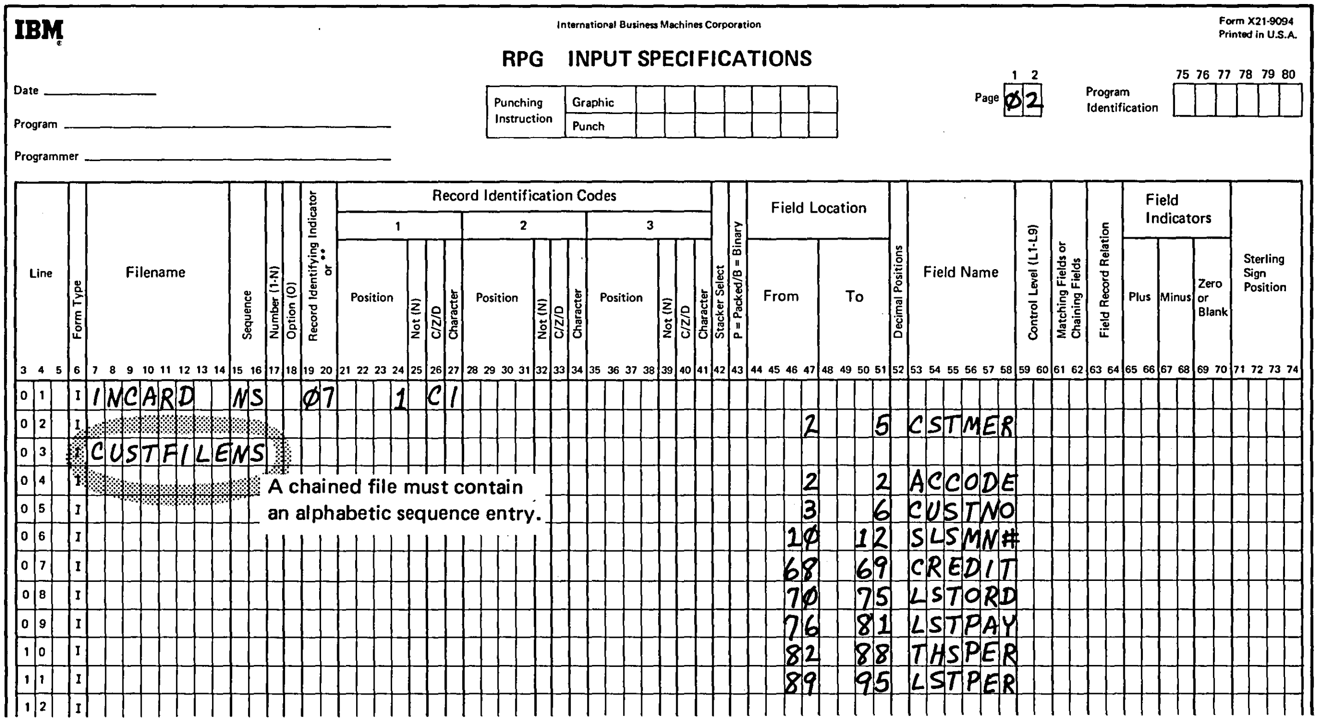
```
                    1111111111222222222233333333334444444444555555555566666666667777777777888888888899999999
           1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567
Heading  1 CUSTOMER  ACTIVITY  SALESMAN  CREDIT  LAST ORDER  LAST PAY  SLS THIS PER  SLS LAST PER  TOTAL A/R
         2
Defined  3   XXXX       X        XXX       XX     XXXXXXXX   XXXXXXXX    XXXXXXXX      XXXXXXXX    XXXXXXXXX
         4 (CUSTNO)  (ACCODE)  (SLSMN#)  (CREDIT) (LSTORD)   (LSTPAY)    (THSPER)      (LSTPER)    (TOTAR)
         5
```



```
CUSTOMER  ACTIVITY  SALESMAN  CREDIT  LAST ORDER  LAST PAY  SLS THIS PER  SLS LAST PER  TOTAL A/R

  3119       A        105       01      4/17/71    4/01/71       360.00        239.50      360.00
  6678 RECORD NOT FOUND--INVALID RECORD NUMBER
  1703       I         35       03     11/19/70   12/01/70          .00           .00         .00
```

Figure 3-7. Printer Output from Random Inquiry Requests





IBM

International Business Machines Corporation

Form X21-9094
Printed in U.S.A.

RPG INPUT SPECIFICATIONS

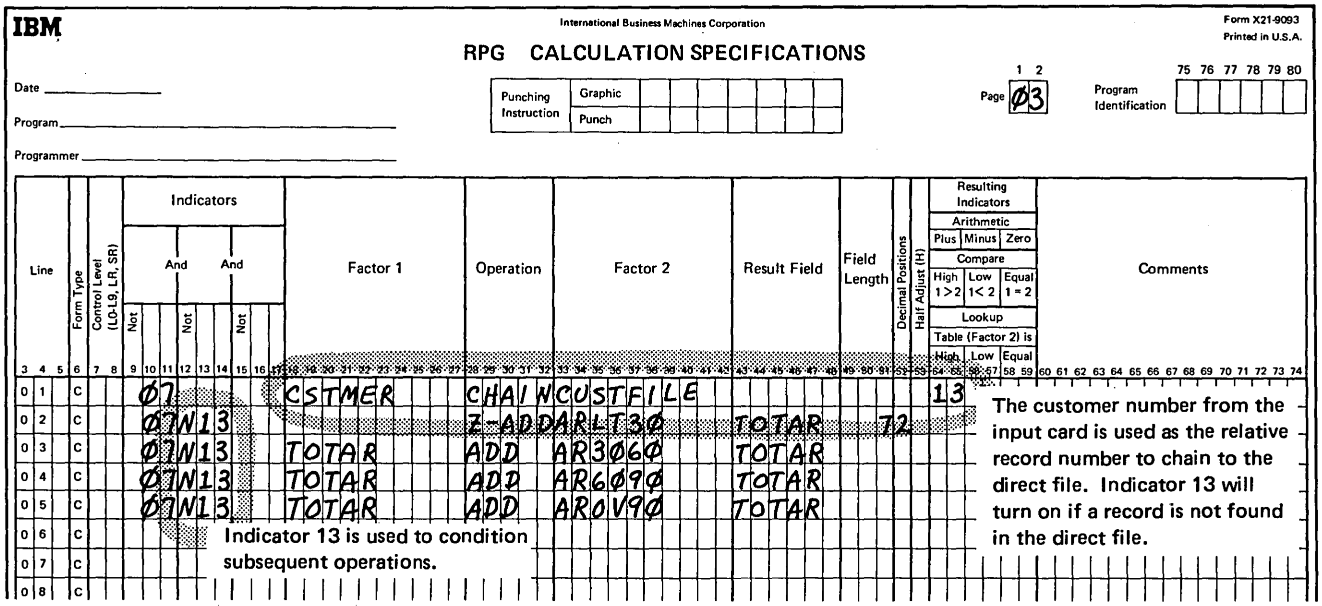
Date
Program
Programmer

Page Ø2

Line	Form Type	Filename	Sequence	Record Identifying Indicator	Position	C/Z/D	Character	From	To	Field Name
01	I	INCARD	NS	Ø7	1	C	I			
02	I							2	5	CSTMER
03	I	CUSTFILE	NS							
04	I							2	2	ACCODE
05	I							3	6	CUSTNO
06	I							1Ø	12	SLSMN#
07	I							68	69	CREDIT
08	I							7Ø	75	LSTORD
09	I							76	81	LSTPAY
10	I							82	88	THSPER
11	I							89	95	LSTPER
12	I									

A chained file must contain an alphabetic sequence entry.

Figure 3-8. Random Retrieval from a Direct File (part 2 of 4)



IBM

International Business Machines Corporation

Form X21-9093
Printed in U.S.A.

RPG CALCULATION SPECIFICATIONS

Date
Program
Programmer

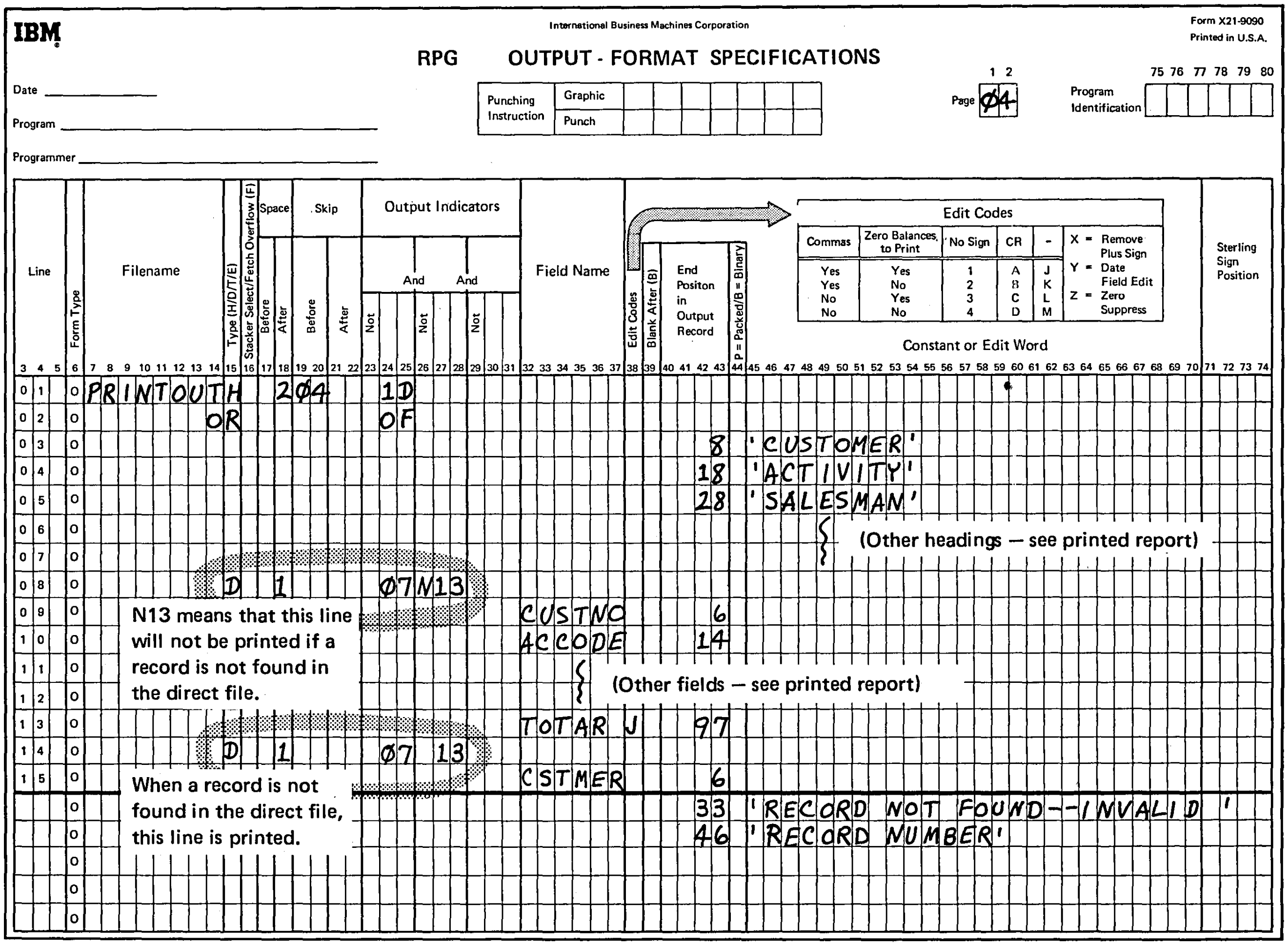
Page Ø3

Line	Form Type	Indicators	Factor 1	Operation	Factor 2	Result Field	Field Length	Decimal Positions	Resulting Indicators
01	C	Ø7	CSTMER	CHAIN	CUSTFILE				13
02	C	Ø7N13		Z-ADD	ARLT3Ø	TOTAR	7	2	
03	C	Ø7N13	TOTAR	ADD	AR3Ø6Ø	TOTAR			
04	C	Ø7N13	TOTAR	ADD	AR6Ø9Ø	TOTAR			
05	C	Ø7N13	TOTAR	ADD	AROV9Ø	TOTAR			
06	C								
07	C								
08	C								

Indicator 13 is used to condition subsequent operations.

The customer number from the input card is used as the relative record number to chain to the direct file. Indicator 13 will turn on if a record is not found in the direct file.

Figure 3-8. Random Retrieval from a Direct File (part 3 of 4)



IBM International Business Machines Corporation — Form X21-9090, Printed in U.S.A.

RPG OUTPUT - FORMAT SPECIFICATIONS

Date ______ Program ______ Programmer ______

Punching Instruction: Graphic / Punch

Page (1 2): Ø4 Program Identification (75–80):

Edit Codes:

Commas	Zero Balances to Print	No Sign	CR	-	X = Remove Plus Sign Y = Date Field Edit Z = Zero Suppress
Yes	Yes	1	A	J	
Yes	No	2	B	K	
No	Yes	3	C	L	
No	No	4	D	M	

Line (3–5)	Form Type (6)	Filename (7–14)	Type (H/D/T/E) (15)	Stacker Select/Fetch Overflow (F) (16)	Space Before (17)	Space After (18)	Skip Before (19–20)	Skip After (21–22)	Output Indicators (23–25)	And (26–28)	And (29–31)	Field Name (32–37)	Edit Codes (38)	Blank After (B) (39)	End Position in Output Record (40–43)	P = Packed/B = Binary (44)	Constant or Edit Word (45–70)	Sterling Sign Position (71–74)
01	O	PRINTOUT	H			2	Ø4		1P									
02	O	O	R						OF									
03	O														8		'CUSTOMER'	
04	O														18		'ACTIVITY'	
05	O														28		'SALESMAN'	
06	O																(Other headings — see printed report)	
07	O																	
08	O		D			1			Ø7	N13								
09	O											CUSTNO			6			
10	O											4CCODE			14			
11	O											(Other fields — see printed report)						
12	O																	
13	O											TOTAR	J		97			
14	O		D			1			Ø7	13								
15	O											CSTMER			6			
	O														33		'RECORD NOT FOUND--INVALID '	
	O														46		'RECORD NUMBER'	
	O																	
	O																	
	O																	

N13 means that this line will not be printed if a record is not found in the direct file.

When a record is not found in the direct file, this line is printed.

Figure 3-8. Random Retrieval from a Direct File (part 4 of 4)

Random Updating of a Direct File

The coding entries on the File Description Sheet that are characteristic of random update are shown in the second coding line on Figure 3-2. Random update is distinguished from random retrieval by a *U* in column 15.

The fields to be updated must be further described on both Input and Output-Format Sheets. If the direct file being updated contains synonym records, calculations must be included in the program to test for synonyms and locate the desired record.

Example 2

Each day, the distributor described in Example 1 prepares invoices for customer orders. Information from the invoices is used to update the customer file, CUSTFILE. Since this information is read from cards (Figure 3-9) in an unordered manner, a random update job is required. The input cards contain the date and total amount of the transactions for each customer. New addresses are also contained on this card when required. As each card is read, the customer number (CUSTMR) is used to chain to the direct file. The amount of the transaction is added to total sales for the period (THSPER) and to the accounts receivable amount (ARLT30). The date of the transaction is placed in the date of last order field (LSTORD) in the customer record. If an address change is indicated by an *X* in column 18 of the input card, the new customer address replaces the old. If a blank record location is encountered in processing, the input card is listed on the printer along with the statement "No master record for the above record." Similarly, if a record is not found in CUSTFILE because of an invalid relative record number (see *CHAIN Operation* in the preceding text), the input card is printed, followed by the statement "Above record not found – invalid record number."

CUSTFILE, described as a chained update file, must be described on both Input and Output-Format Specifications Sheets, since data is both read from and written on the file. The specifications are shown in Figure 3-10.

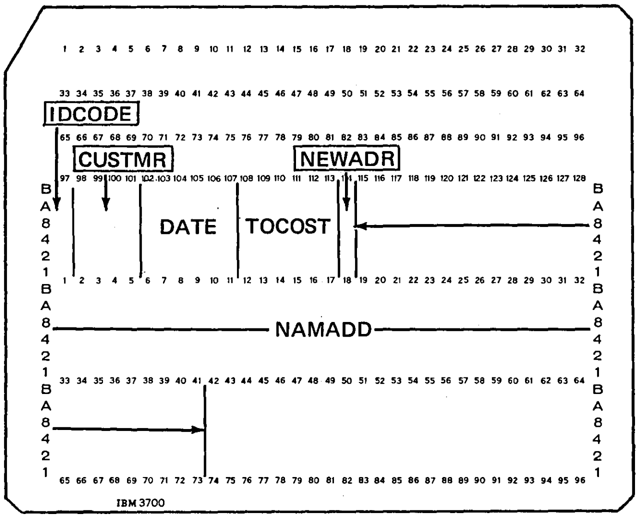
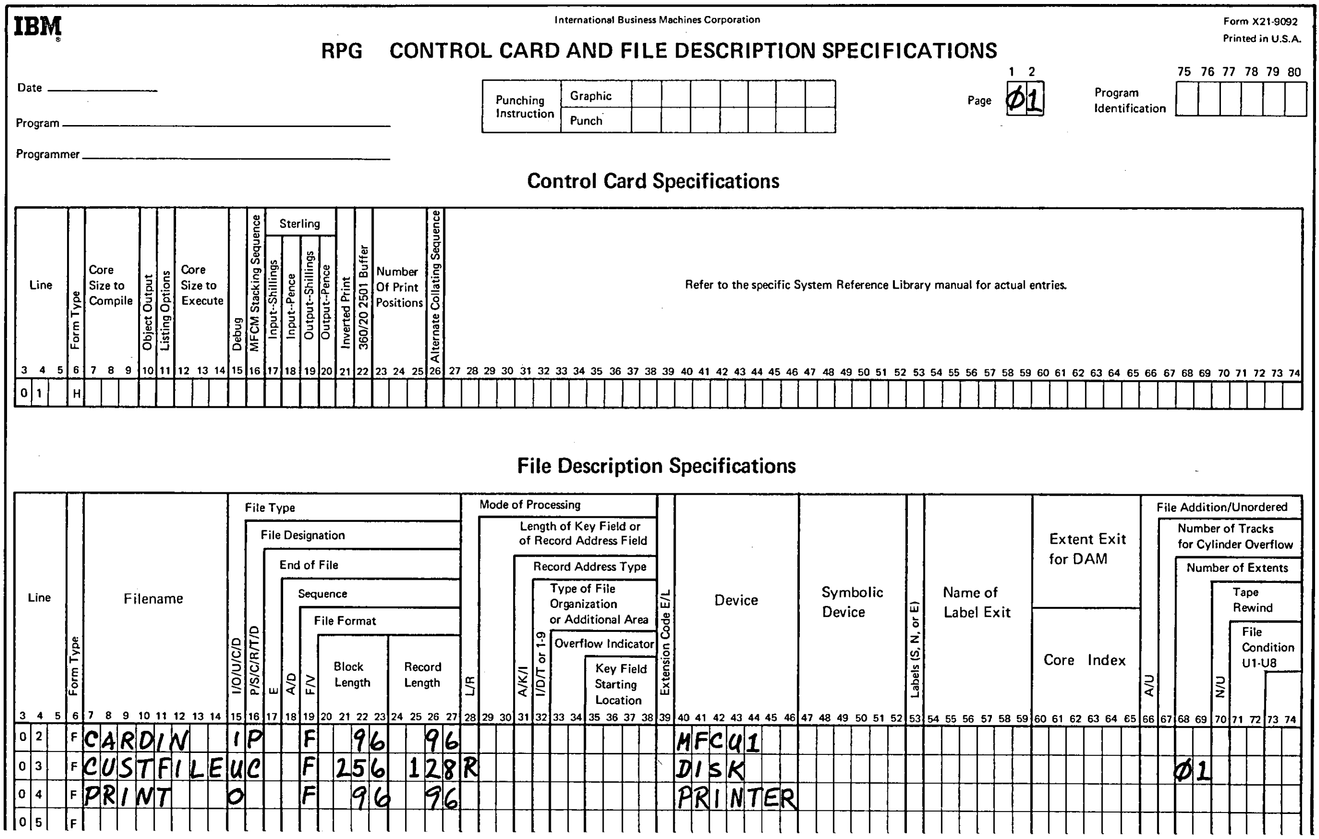


Figure 3-9. Daily Invoicing Card for Updating CUSTFILE



IBM

International Business Machines Corporation

Form X21-9092
Printed in U.S.A.

RPG CONTROL CARD AND FILE DESCRIPTION SPECIFICATIONS

Date ______

Program ______

Programmer ______

Punching Instruction: Graphic / Punch

Page Ø1

Program Identification

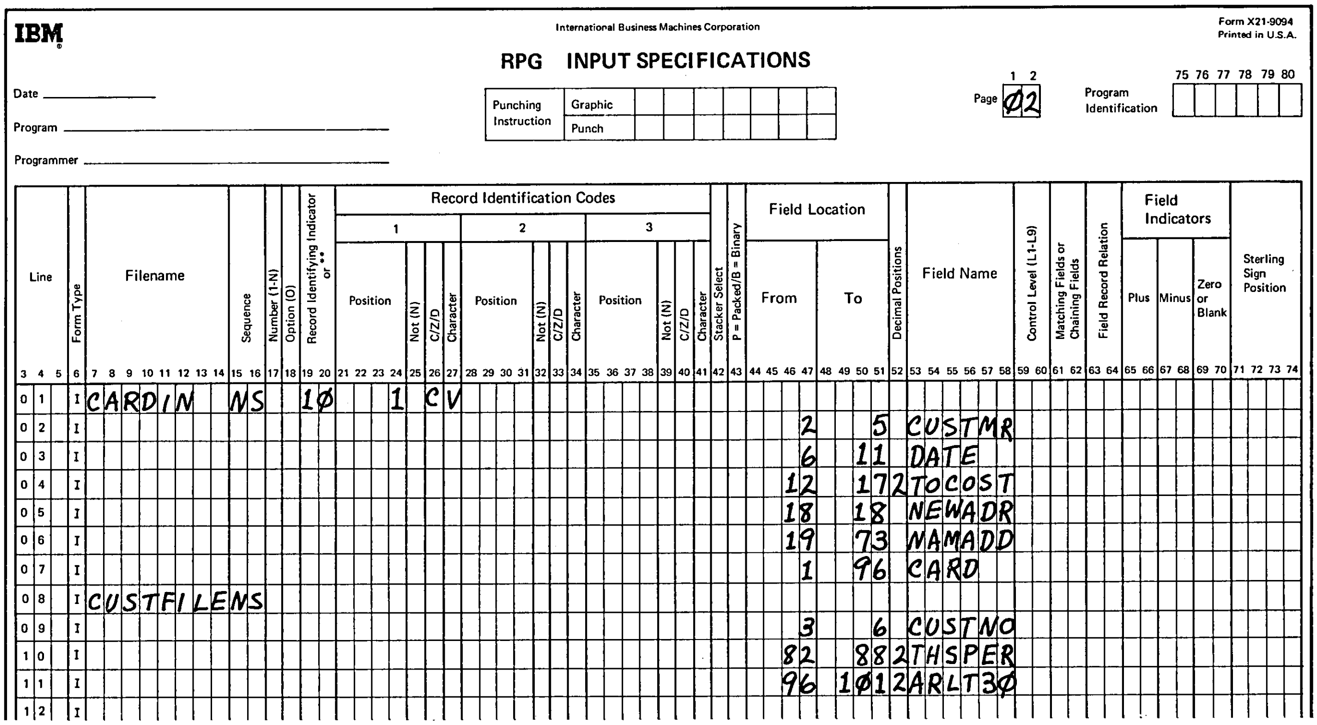
Control Card Specifications

Line	Form Type	Core Size to Compile	Object Output	Listing Options	Core Size to Execute	Debug	MFCM Stacking Sequence	Input-Shillings	Input-Pence	Output-Shillings	Output-Pence	Inverted Print	360/20 2501 Buffer	Number Of Print Positions	Alternate Collating Sequence	Refer to the specific System Reference Library manual for actual entries.
01	H															

File Description Specifications

Line	Form Type	Filename	File Type (I/O/U/C/D)	File Designation (P/S/C/R/T/D)	End of File (E)	Sequence (A/D)	File Format (F/V)	Block Length	Record Length	Mode of Processing (L/R)	Length of Key Field or of Record Address Field	Record Address Type (A/K/I)	Type of File Organization or Additional Area (I/D/T or 1-9)	Overflow Indicator	Key Field Starting Location	Extension Code E/L	Device	Symbolic Device	Labels (S, N, or E)	Name of Label Exit	Extent Exit for DAM / Core Index	File Addition/Unordered (A/U)	Number of Tracks for Cylinder Overflow	Number of Extents	Tape Rewind (R/U/N)	File Condition U1-U8
02	F	CARDIN	I	P			F	96	96								MFCU1									
03	F	CUSTFILE	U	C			F	256	128	R							DISK							Ø1		
04	F	PRINT	O				F	96	96								PRINTER									
05	F																									

Figure 3-10. Updating a Direct File Randomly (part 1 of 4)



IBM

International Business Machines Corporation

Form X21-9094
Printed in U.S.A.

RPG INPUT SPECIFICATIONS

Date ______

Program ______

Programmer ______

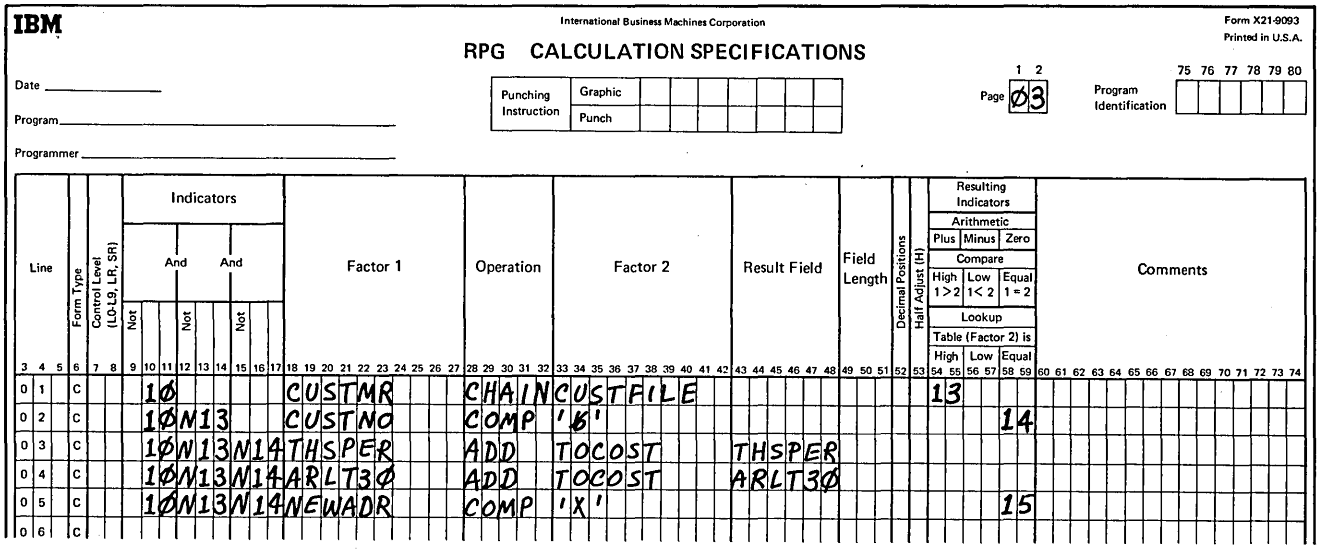
Punching Instruction: Graphic / Punch

Page Ø2

Program Identification

Line	Form Type	Filename	Sequence	Number (1-N)	Option (O)	Record Identifying Indicator or **	1 Position	1 Not (N)	1 C/Z/D	1 Character	2 Position	2 Not (N)	2 C/Z/D	2 Character	3 Position	3 Not (N)	3 C/Z/D	3 Character	Stacker Select	P = Packed/B = Binary	Field Location From	Field Location To	Decimal Positions	Field Name	Control Level (L1-L9)	Matching Fields or Chaining Fields	Field Record Relation	Field Indicators Plus	Field Indicators Minus	Field Indicators Zero or Blank	Sterling Sign Position
01	I	CARDIN	NS			1Ø	1		C	V																					
02	I																				2	5		CUSTMR							
03	I																				6	11		DATE							
04	I																				12	17	2	TOCOST							
05	I																				18	18		NEWADR							
06	I																				19	73		NAMADD							
07	I																				1	96		CARD							
08	I	CUSTFILE	NS																												
09	I																				3	6		CUSTNO							
10	I																				82	88	2	THSPER							
11	I																				96	1Ø1	2	ARLT3Ø							
12	I																														

Figure 3-10. Updating a Direct File Randomly (part 2 of 4)



IBM

International Business Machines Corporation

RPG CALCULATION SPECIFICATIONS

Form X21-9093
Printed in U.S.A.

Date

Program

Programmer

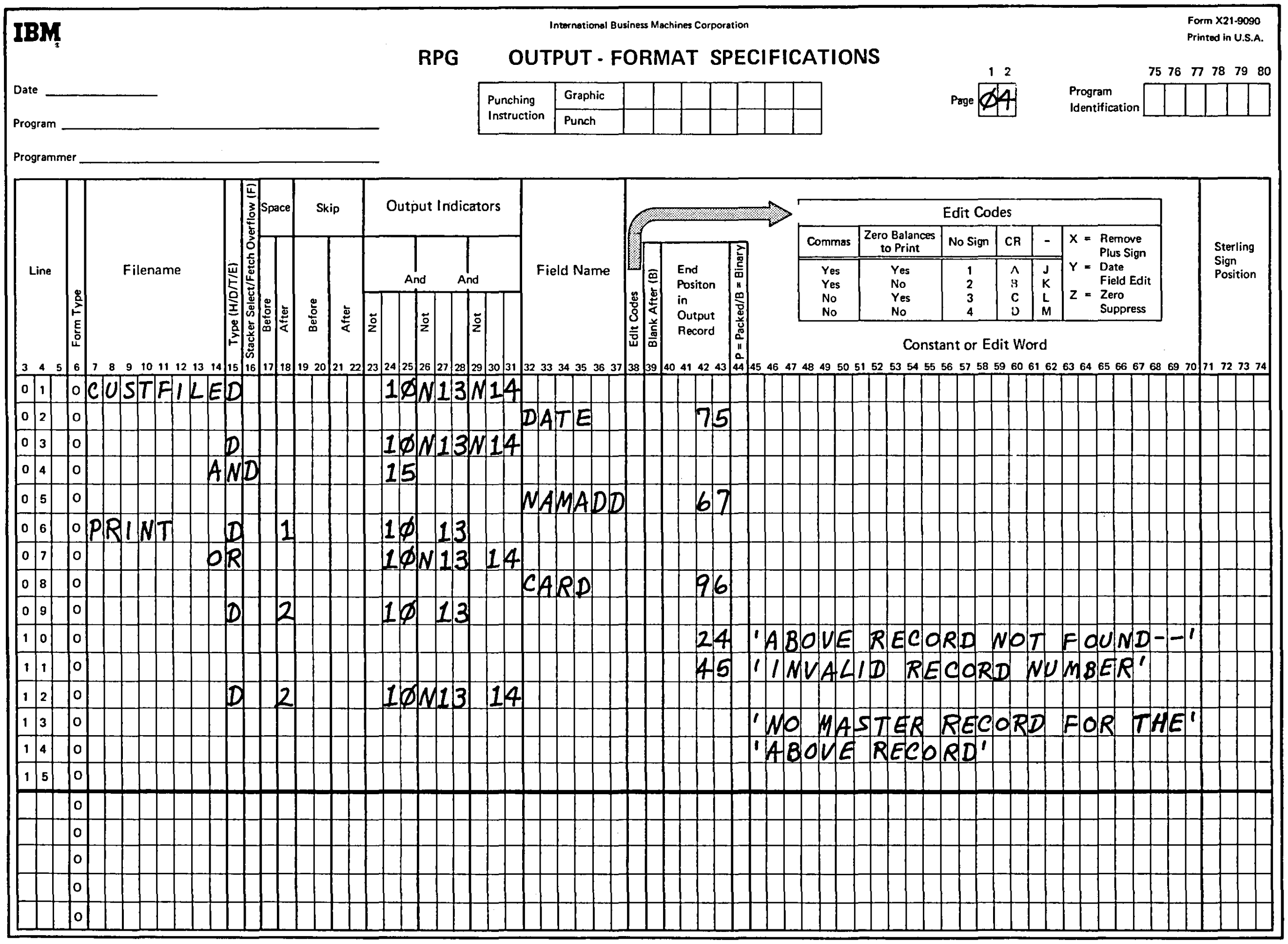
Punching Instruction	Graphic
	Punch

Page Ø3

Program Identification

Line	Form Type	Control Level (L0-L9, LR, SR)	Indicators	And	And	Factor 1	Operation	Factor 2	Result Field	Field Length	Decimal Positions	Half Adjust (H)	Plus / High 1>2 / High (54-55)	Minus / Low 1<2 / Low (56-57)	Zero / Equal 1=2 / Equal (58-59)	Comments
01	C		1Ø			CUSTMR	CHAIN	CUSTFILE					13			
02	C		1Ø	N13		CUSTNO	COMP	'b̸'							14	
03	C		1Ø	N13	N14	THSPER	ADD	TOCOST	THSPER							
04	C		1Ø	N13	N14	ARLT3Ø	ADD	TOCOST	ARLT3Ø							
05	C		1Ø	N13	N14	NEWADR	COMP	'X'							15	
06	C															

Figure 3-10. Updating a Direct File Randomly (part 3 of 4)



IBM

International Business Machines Corporation

RPG OUTPUT - FORMAT SPECIFICATIONS

Form X21-9090
Printed in U.S.A.

Date

Program

Programmer

Punching Instruction	Graphic
	Punch

Page Ø4

Program Identification

Edit Codes

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Line	Form Type	Filename	Type (H/D/T/E)	Stacker Select/Fetch Overflow (F)	Space Before	Space After	Skip Before	Skip After	Output Indicators	And	And	Field Name	Edit Codes	Blank After (B)	End Position in Output Record	P = Packed/B = Binary	Constant or Edit Word	Sterling Sign Position
01	O	CUSTFILE	D						1Ø	N13	N14							
02	O											DATE			75			
03	O		D						1Ø	N13	N14							
04	O		AND						15									
05	O											NAMADD			67			
06	O	PRINT	D			1			1Ø	13								
07	O		OR						1Ø	N13	14							
08	O											CARD			96			
09	O		D			2			1Ø	13								
10	O														24		'ABOVE RECORD NOT FOUND--'	
11	O														45		'INVALID RECORD NUMBER'	
12	O		D			2			1Ø	N13	14							
13	O																'NO MASTER RECORD FOR THE'	
14	O																'ABOVE RECORD'	
15	O																	
	O																	
	O																	
	O																	
	O																	
	O																	

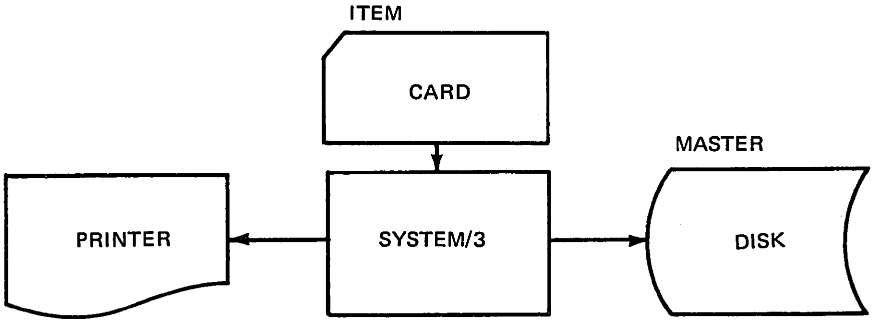
Figure 3-10. Updating a Direct File Randomly (part 4 of 4)

1. What is a relative record number?

2. How is the relative record number used to access a record randomly?

3. Code a program to update a direct file randomly:

Problem Description



Disk Record

Positions	*Contents*
1-2	Code = 20
3-5	Product number
6-25	Description
26-32	Price (2 decimal positions)
32-38	Quantity on hand
39	Activity code = A

Card Record

Columns	*Contents*
1-2	Code = 10
3-5	Part number
6-11	Customer number
12-16	Quantity ordered

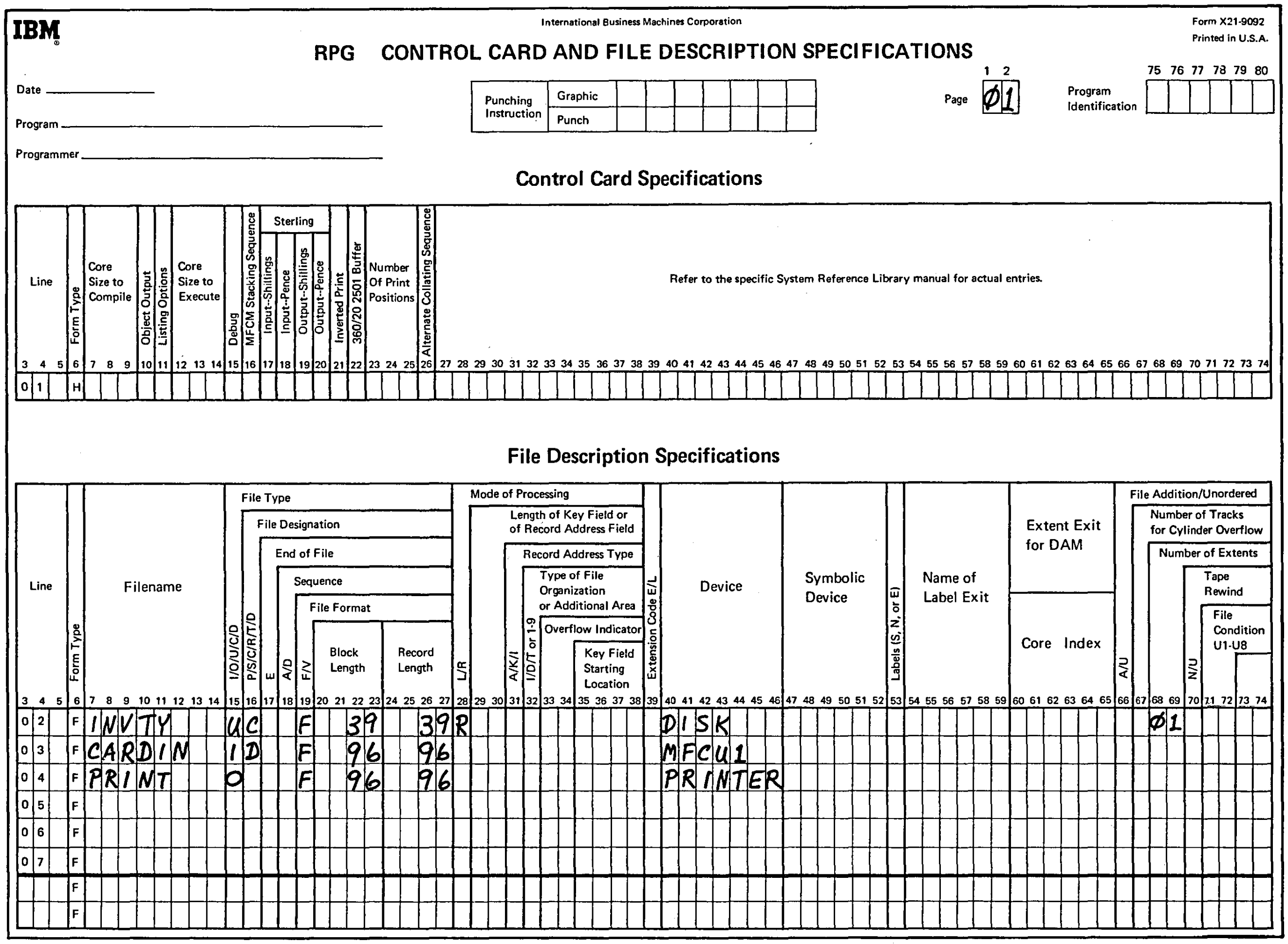
Update the disk file quantity on hand by subtracting the quantity ordered on the item card. If the quantity on hand is zero or negative, print the item number, description, order quantity, quantity on hand, and the message, "out of stock".

Answers to Review 3

1. A relative record number is a value representing the numeric position of a record in a file relative to the beginning of the file. A relative record number must be a positive, whole number.

2. The correct relative record number for the desired record must be supplied by the programmer, either as an input variable, a derived variable or a constant. This value is converted automatically to the disk address of the desired record. The disk address is used to access the required record.

3. See coding sheets (Figure 3-11).



IBM

International Business Machines Corporation

Form X21-9092
Printed in U.S.A.

RPG CONTROL CARD AND FILE DESCRIPTION SPECIFICATIONS

Date
Program
Programmer

Punching Instruction: Graphic / Punch

Page (1 2) Ø1

Program Identification (75 76 77 78 79 80)

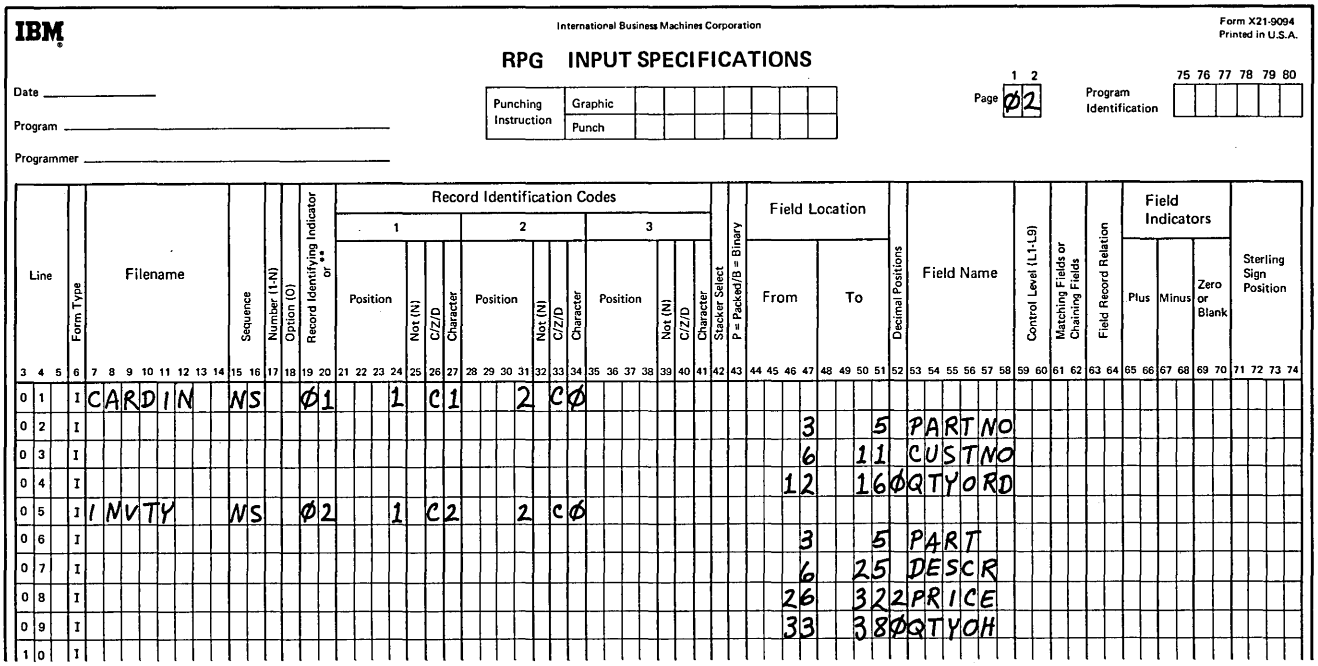
Control Card Specifications

Line (3-5)	Form Type (6)	Core Size to Compile (7-9)	Object Output (10)	Listing Options (11)	Core Size to Execute (12-14)	Debug (15)	MFCM Stacking Sequence (16)	Sterling: Input-Shillings (17)	Input-Pence (18)	Output-Shillings (19)	Output-Pence (20)	Inverted Print (21)	360/20 2501 Buffer (22)	Number Of Print Positions (23-25)	Alternate Collating Sequence (26)	27-74 Refer to the specific System Reference Library manual for actual entries.
01	H															

File Description Specifications

Line (3-5)	Form Type (6)	Filename (7-14)	File Type: I/O/U/C/D (15)	File Designation: P/S/C/R/T/D (16)	End of File: E (17)	Sequence: A/D (18)	File Format: F/V (19)	Block Length (20-23)	Record Length (24-27)	L/R (28)	Mode of Processing (29-30)	Length of Key Field or of Record Address Field (29-30)	Record Address Type: A/K/I (31)	Type of File Organization or Additional Area: I/D/T or 1-9 (32)	Overflow Indicator (33-34)	Key Field Starting Location (35-38)	Extension Code E/L (39)	Device (40-46)	Symbolic Device (47-52)	Labels (S, N, or E) (53)	Name of Label Exit (54-59)	Extent Exit for DAM / Core Index (60-65)	A/U (66)	File Addition/Unordered (67)	Number of Tracks for Cylinder Overflow / Number of Extents (68-69)	N/U (70)	Tape Rewind (70)	File Condition U1-U8 (71-72)	73-74
02	F	INVTY	U	C			F	39	39	R								DISK							Ø1				
03	F	CARDIN	I	D			F	96	96									MFCU1											
04	F	PRINT	O				F	96	96									PRINTER											
05	F																												
06	F																												
07	F																												
	F																												
	F																												

Figure 3-11. Updating a Direct Inventory File (part 1 of 4)



IBM

International Business Machines Corporation

Form X21-9094
Printed in U.S.A.

RPG INPUT SPECIFICATIONS

Date

Program

Programmer

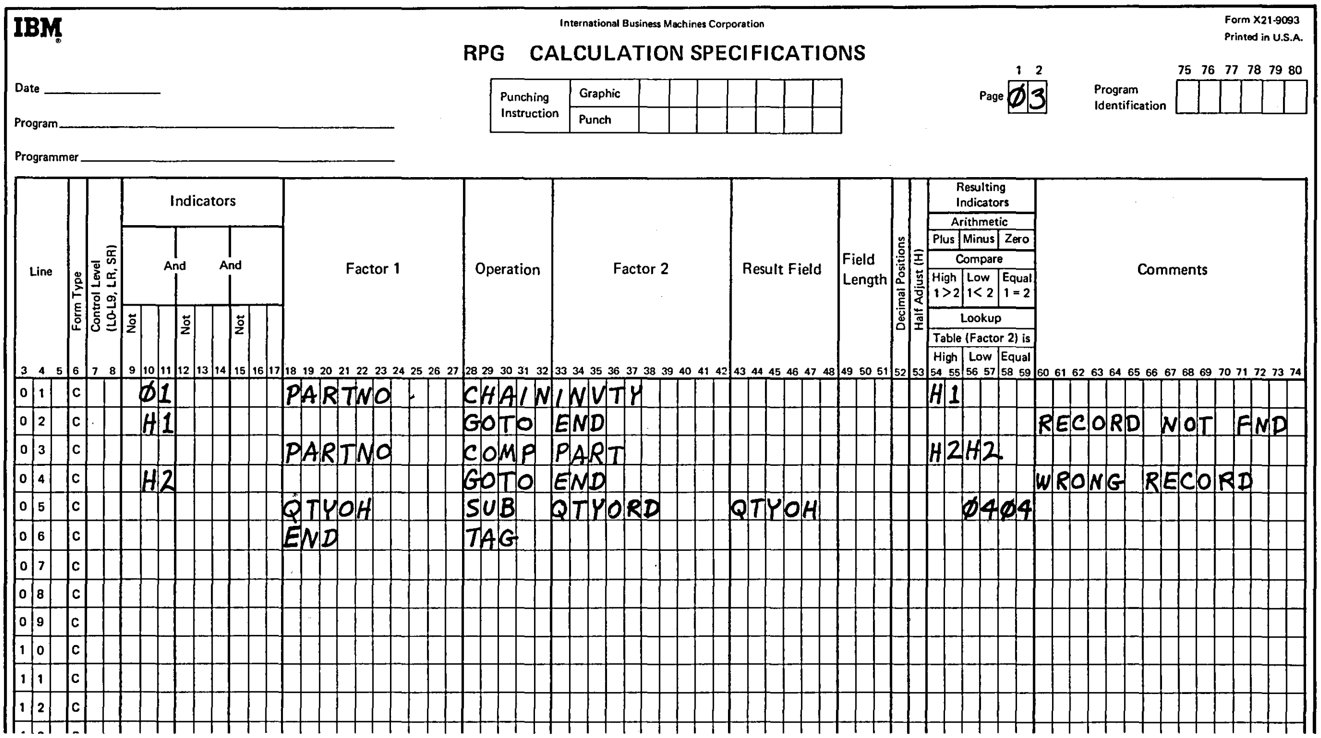
Punching Instruction: Graphic / Punch

Page Ø2

Program Identification

Line	Form Type	Filename	Sequence	Number (1-N)	Option (O)	Record Identifying Indicator or **	1 Position	1 Not (N)	1 C/Z/D	1 Character	2 Position	2 Not (N)	2 C/Z/D	2 Character	3 Position	3 Not (N)	3 C/Z/D	3 Character	Stacker Select	P = Packed/B = Binary	Field Location From	Field Location To	Decimal Positions	Field Name	Control Level (L1-L9)	Matching Fields or Chaining Fields	Field Record Relation	Field Indicators Plus	Field Indicators Minus	Field Indicators Zero or Blank	Sterling Sign Position
01	I	CARDIN	NS			Ø1	1		C	1	2		C	Ø																	
02	I																				3	5		PARTNO							
03	I																				6	11		CUSTNO							
04	I																				12	16	Ø	QTYORD							
05	I	INVTY	NS			Ø2	1		C	2	2		C	Ø																	
06	I																				3	5		PART							
07	I																				6	25		DESCR							
08	I																				26	32	2	PRICE							
09	I																				33	38	Ø	QTYOH							
10	I																														

Figure 3-11. Updating a Direct Inventory File (part 2 of 4)



IBM

International Business Machines Corporation

Form X21-9093
Printed in U.S.A.

RPG CALCULATION SPECIFICATIONS

Date

Program

Programmer

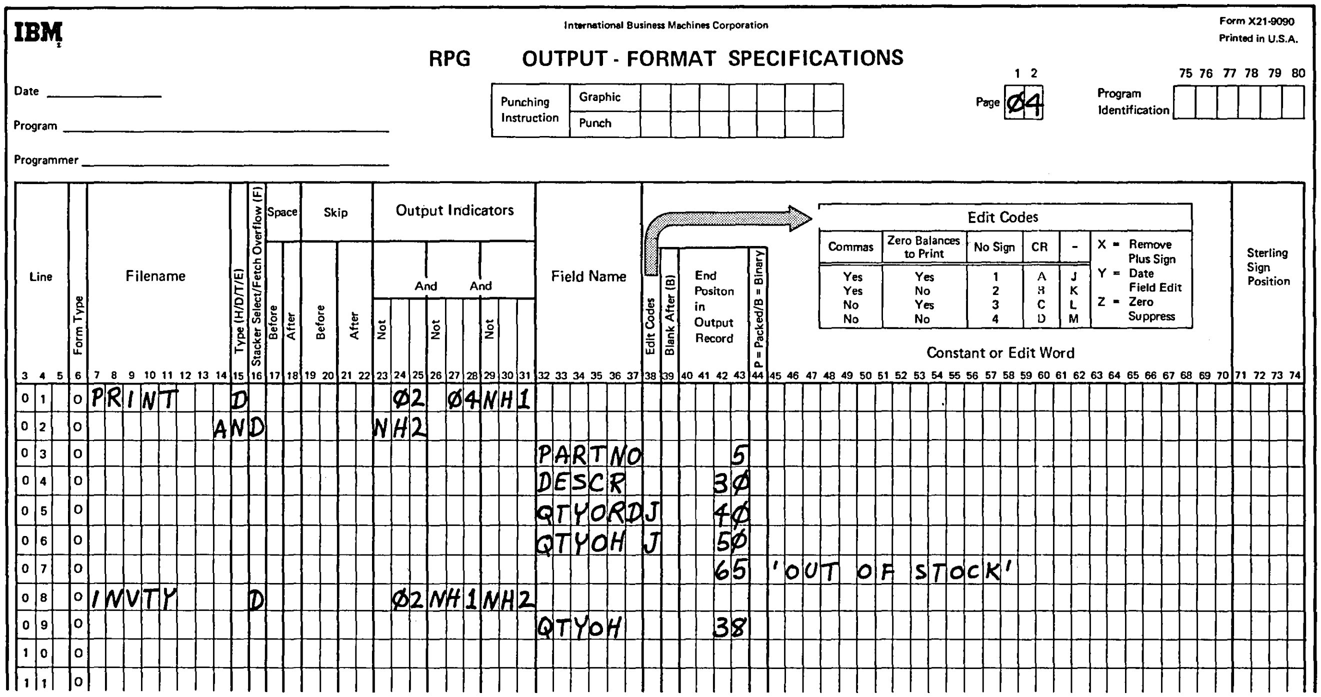
Punching Instruction: Graphic / Punch

Page Ø3

Program Identification

Line	Form Type	Control Level (L0-L9, LR, SR)	Not	Indicators	And Not	And	And Not	And	Factor 1	Operation	Factor 2	Result Field	Field Length	Decimal Positions	Half Adjust (H)	Plus / High 1>2 / High	Minus / Low 1<2 / Low	Zero / Equal 1=2 / Equal	Comments
01	C			Ø1					PARTNO	CHAIN	INVTY					H1			
02	C			H1						GOTO	END								RECORD NOT FND
03	C								PARTNO	COMP	PART					H2	H2		
04	C			H2						GOTO	END								WRONG RECORD
05	C								QTYOH	SUB	QTYORD	QTYOH					Ø4	Ø4	
06	C								END	TAG									
07	C																		
08	C																		
09	C																		
10	C																		
11	C																		
12	C																		

Figure 3-11. Updating a Direct Inventory File (part 3 of 4)



IBM

International Business Machines Corporation

Form X21-9090
Printed in U.S.A.

RPG OUTPUT - FORMAT SPECIFICATIONS

Date ____________

Program ____________

Programmer ____________

Punching Instruction	Graphic	
	Punch	

Page Ø4 (1 2)

Program Identification (75 76 77 78 79 80)

Edit Codes

Commas	Zero Balances to Print	No Sign	CR	-	
Yes	Yes	1	A	J	X = Remove Plus Sign
Yes	No	2	B	K	Y = Date Field Edit
No	Yes	3	C	L	Z = Zero Suppress
No	No	4	D	M	

Line (3-5)	Form Type (6)	Filename (7-14)	Type (H/D/T/E) (15)	Stacker Select/Fetch Overflow (F) (16)	Space Before (17)	Space After (18)	Skip Before (19-20)	Skip After (21-22)	Output Indicators Not (23)	(24-25)	And Not (26)	(27-28)	And Not (29)	(30-31)	Field Name (32-37)	Edit Codes (38)	Blank After (B) (39)	End Position in Output Record (40-43)	P = Packed/B = Binary (44)	Constant or Edit Word (45-70)	Sterling Sign Position (71-74)
01	O	PRINT	D							Ø2		Ø4	N	H1							
02	O	AND							N	H2											
03	O														PARTNO			5			
04	O														DESCR			3Ø			
05	O														QTYORD	J		4Ø			
06	O														QTYOH	J		5Ø			
07	O																	65		'OUT OF STOCK'	
08	O	INVTY		D						Ø2	N	H1	N	H2							
09	O														QTYOH			38			
10	O																				
11	O																				

Figure 3-11. Updating a Direct Inventory File (part 4 of 4)

CHAPTER

SORTING A FILE TO PRODUCE AN ADDROUT FILE

4

CHAPTER 4 DESCRIBES:

ADDROUT sort and its output.

Sequence specifications and OCL statements required for the ADDROUT sort.

How to determine storage and file sizes for the ADDROUT sort.

BEFORE READING THIS CHAPTER YOU SHOULD BE ABLE TO:

Describe tag-along sort.

Code the Sequence Specifications to execute a tag-along sort.

Describe the functions of the keyword parameters on the FILE statement.

AFTER READING THIS CHAPTER YOU SHOULD BE ABLE TO:

Describe output of an ADDROUT sort and compare it to the output of a tag-along sort.

State the advantages of using the ADDROUT sort over the tag-along sort.

List the sequence specifications required by the Disk Sort program.

List the three files used by the Disk Sort program to put out an ADDROUT file.

Calculate storage and file sizes so your requirements do not exceed the amount of main storage and disk storage you have.

Code sequence specifications and OCL statements to execute the Disk Sort program.

INTRODUCTION

The Disk Sort program performs either of two jobs:

- A tag-along sort that creates a sorted sequential file.
- An ADDROUT sort that creates a sorted record-address file.

The two types of sort differ in output. A tag-along sort produces output records that retain the control fields and the data, only the data, or only the control field:

RECORD 1		RECORD 2	
control field	data	control field	data

RECORD 1	RECORD 2
data	data

RECORD 1	RECORD 2
control field	control field

An ADDROUT sort produces output records that contain the relative record numbers of the sorted records. A relative record number indicates to disk system management the relative position (first, second, twenty-second) of a record in a file. The relative record number is a binary number contained in a 3-byte field. The disk file containing relative record numbers is known as an ADDROUT file.

The ADDROUT file can then be used as input (in the form of a record-address file) to process the source file in an RPG II program.

By sorting a file in several sequences based on different control fields in each record of the file, the file can be used to create several ADDROUT files. For example, you have a transaction file in order by stock number. By performing two ADDROUT sorts on the transaction file, you could have one ADDROUT file sequenced by customer number and another by invoice number. Consequently, the transaction file can be processed in several sequences: stock number, customer number, or invoice number.

INPUT AND OUTPUT FOR ADDROUT SORT

Note: The circled numbers in the following text relate to the circled numbers in the Figure 4-1.

Instructions for sorting a file are coded on Sequence Specifications sheets ①. Sequence specifications are comprised of three types of information: header, record type, and control field.

- Header specifications identify the type of sort you want the program to perform.
- Record type specifications identify input records you want the program to use.
- Control field specifications identify the input record fields (control fields) you want the program to use in sorting output records.

The sort program then processes input, work, and output files according to the specifications.

The input file ② contains the records you want sorted. The program sorts the control fields and the relative record numbers of the input records to be sorted according to the control fields that you specify are to determine the sequence of the output. There are no size limitations on an input file: it can be a multi-volume file.

The work file ③ contains the control fields and relative record numbers of the records that you select to be sorted from the input file. The Disk Sort program uses the work file space for sorting the relative record numbers in the order you specify before writing them out on the output file. A work file can be on the same disk as the input file if there is enough room. If the space on the disk you are using is limited, the work file can be on any other disk. You must allow enough space for the work file. Work records are used internally by the program and are not necessarily in their final form. For an ADDROUT sort, work records take the following form:

control field	relative record number

The control field in a work record is that field you specified as the control field on the Sequence sheet for a corresponding input record.

The output file ④ contains only a 3-byte relative record number for each of the sorted input records.

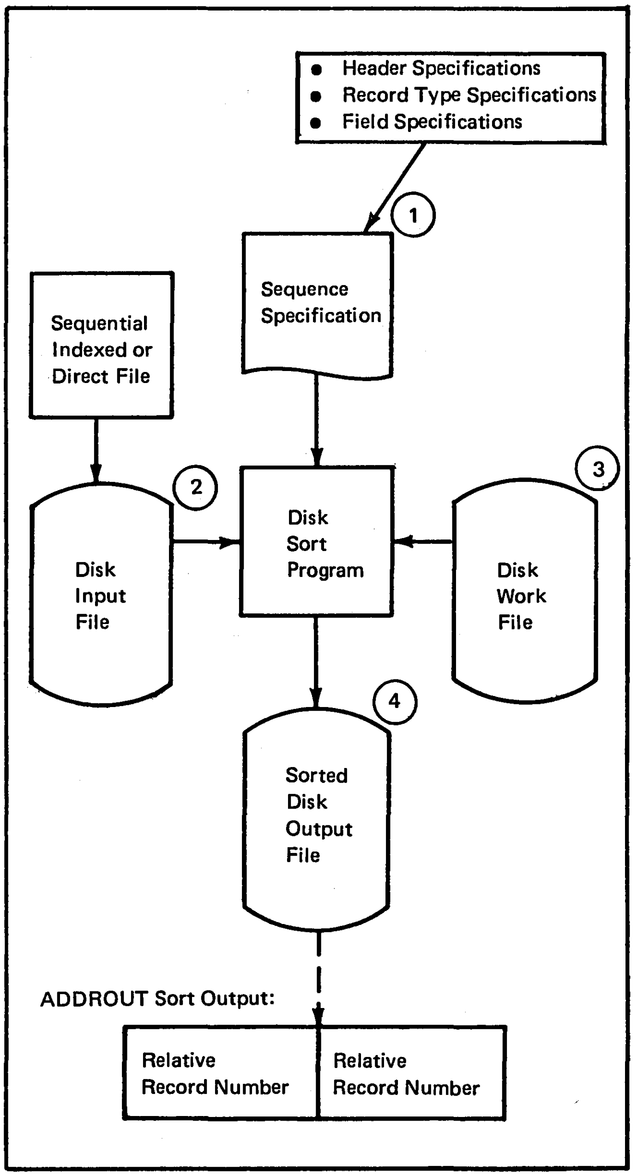


Figure 4-1. Summary of Input and Output for ADDROUT Sort

File Placement

- Records being sorted from the disk input files must be copied to the disk work file. Therefore, disk access mechanism movement between the input file and the work file should be minimized.

- Records are sorted in the work file. When you must make the work file multi-volume on the same drive, the starting track location on each volume should be approximately the same. By doing this, minimum disk access mechanism movement on the work file is maintained.

- The work file is copied to the disk output file. Disk access mechanism movement between the work and output files should be minimized.

- The placement of the input, work, and output files are inter-related.

Figure 4-2 shows the most effective placement of files for various disk drive configurations.

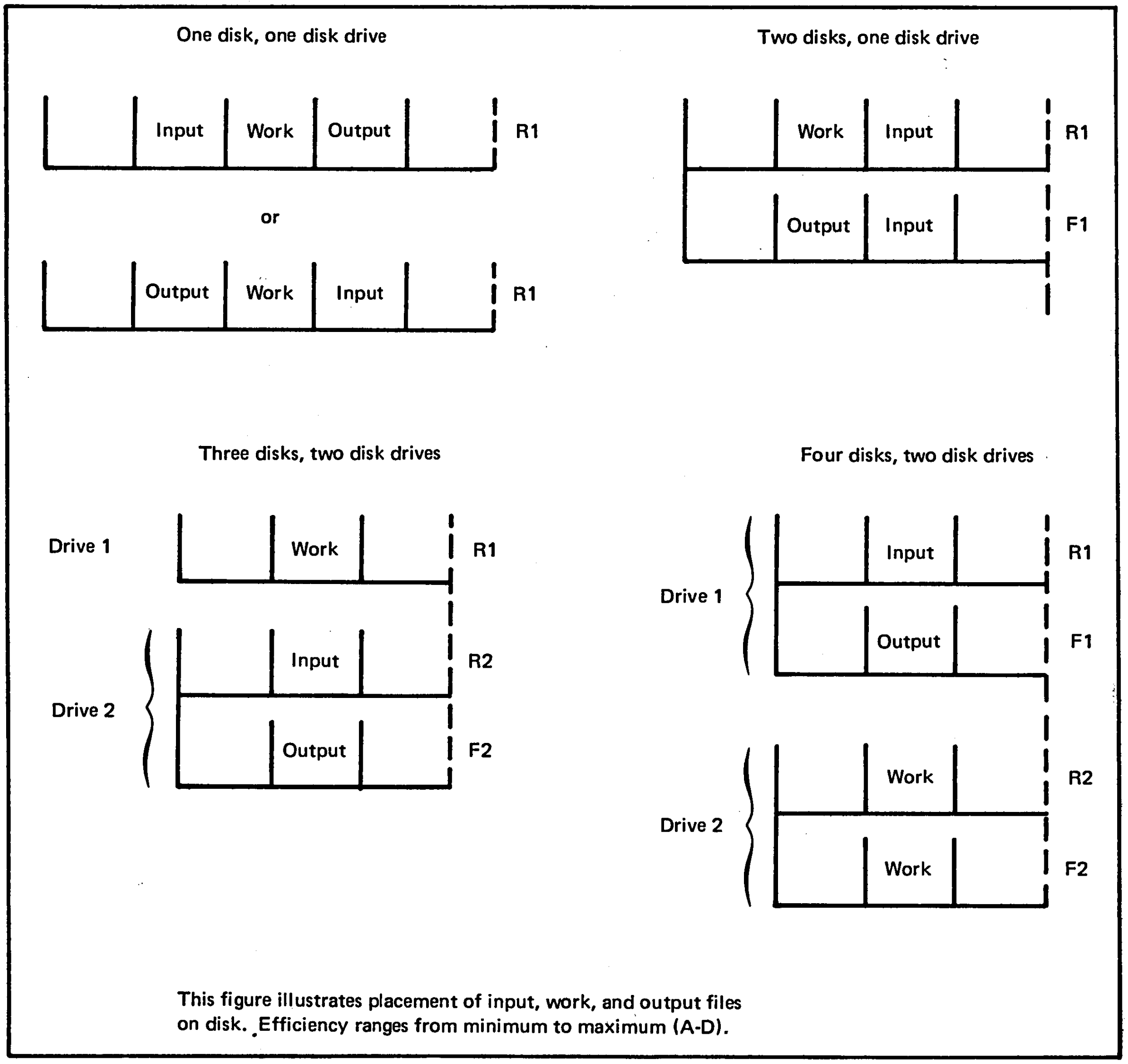

One disk, one disk drive
Input
Work
Output
R1
or
Output
Work
Input
R1
Two disks, one disk drive
Work
Input
R1
Output
Input
F1
Three disks, two disk drives
Drive 1
Work
R1
Drive 2
Input
R2
Output
F2
Four disks, two disk drives
Drive 1
Input
R1
Output
F1
Drive 2
Work
R2
Work
F2
This figure illustrates placement of input, work, and output files on disk. Efficiency ranges from minimum to maximum (A-D).

Figure 4-2. Placement of Files on Disk Drives

DETERMINING STORAGE AND FILE SIZES

Before you run a Disk Sort program, you should determine the amount of main storage available. For optimum performance, the Disk Sort program requires at least 9K bytes of main storage (K=1024). It will run in as little as 5K main storage, but takes approximately twice as long. To determine how much main storage is available to the Disk Sort program, subtract the supervisor size from your total amount of main storage.

The remainder is the amount in which you can execute the program. The remainder is also used to determine the work file size factor that is used to calculate the size of the work file.

You must then calculate file sizes so that your files will not be too large for the amount of disk storage available. You can use the following formulas to determine file sizes in number of tracks:

- Input File – Multiply the number of input records by the length of the input records. Divide by 6144.
- Work File – Multiply the number of work records by the length of the work records. Divide by 6144. Multiply the answer by the work file size factor (Figure 4-3).

  Note: The number of work records is equal to the number of records selected for sorting. If you have an 8,000 record file and all the records are to be sorted, the number of work records is 8,000 also. The length of the work record will be the sum of the control fields plus three.
- Output File – Multiply the number of work records by three. Divide by 6144.

  Note: In an ADDROUT sort, the length of the output record is always three bytes, since the relative record number is three bytes long.

Storage Size Available for ADDROUT Sort	Maximum Work File Size Factor
5K	1.52
8K	1.17
9K	1.15
12K	1.09
20K	1.07
28K	1.07

Note: This is the maximum factor. When the Disk Sort program is run, an actual factor will be printed. You can then re-calculate your file size using this factor.

Figure 4-3. Work File Size Factor for ADDROUT Sort

If your calculations result in an uneven number, always round the figure to the next higher number of tracks. Figure 4-4 illustrates the calculation of file sizes for both an ADDROUT and tag-along sort. This illustration assumes that tag-along sorts the entire record. The input files for both sorts are 66 tracks in length. Notice that the work and output files for the tag-along sort occupy 73 tracks, while these files occupy only four tracks for the ADDROUT sort. Since the size of the files for the ADDROUT sort are smaller, ADDROUT sort will take less time to sort the file.

Assume these values for this example:

Main storage for Disk Sort = 8192 bytes
Number of input records = 2000
Number of work records = 1000 (half are included)
Input record length = 200 bytes
Work size file factor = 1.17

	Tag-along sort:	ADDROUT Sort:
	Work record length = 210 bytes (control field length = 10)	Work record length = 13 bytes (control field length = 13)
	Output record length = 200 bytes (drop control field)	Output record length = 3 bytes
Input File	$\left(\frac{2000 \times 200}{6144}\right) = 65.1$ 66 tracks	$\left(\frac{2000 \times 200}{6144}\right) = 65.1$ 66 tracks
Work File	$\left(\frac{1000 \times 210}{6144}\right) \times 1.17 = 39.99$ 40 tracks	$\left(\frac{1000 \times 13}{6144}\right) \times 1.17 = 2.5$ 3 tracks
Output File	$\left(\frac{1000 \times 200}{6144}\right) = 32.6$ 33 tracks	$\left(\frac{1000 \times 3}{6144}\right) = 0.49$ 1 track
	Total tracks used = 139 tracks	Total tracks used = 70 tracks

Figure 4-4. Example of Calculating File Sizes for SORTR and SORTA

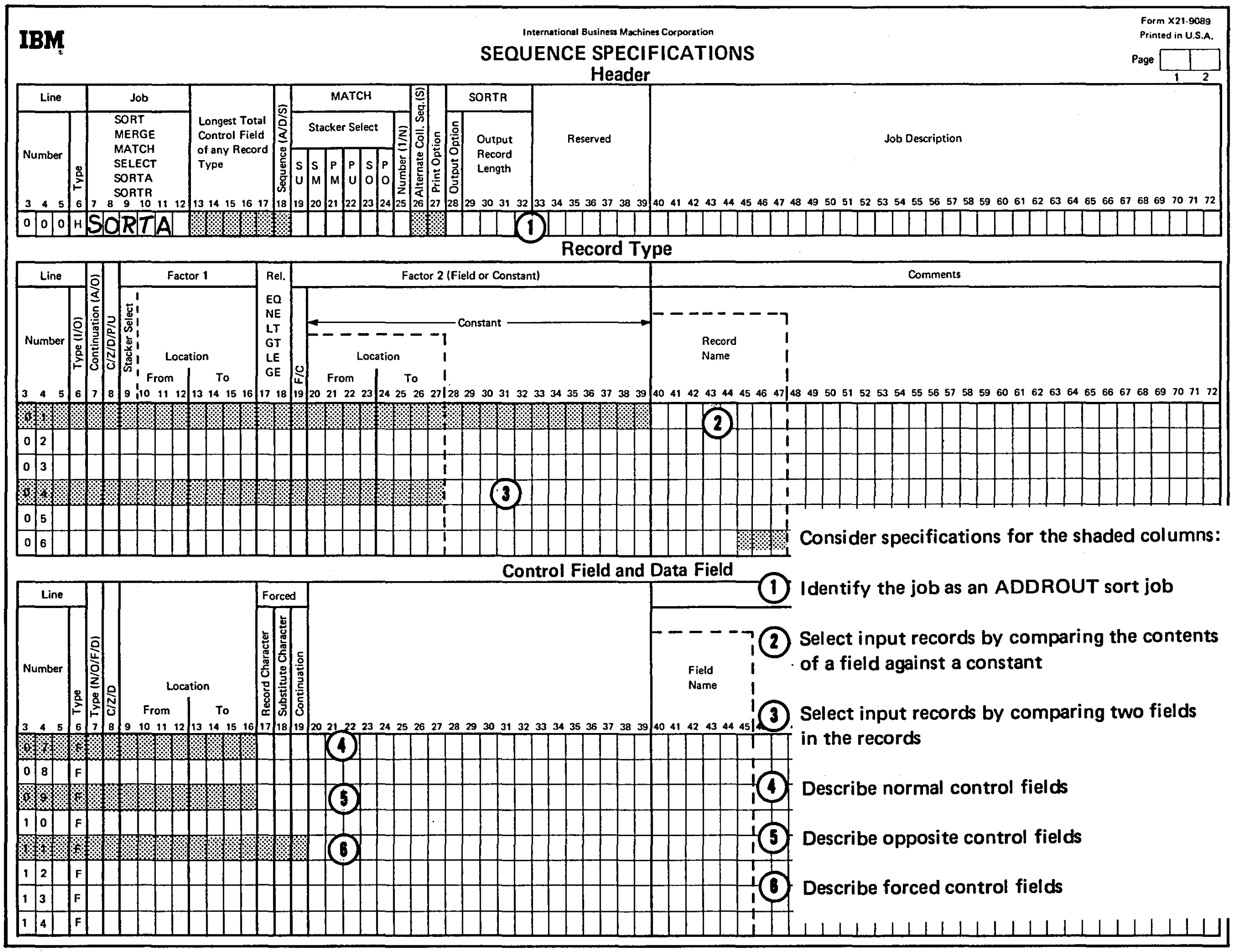
CODING SEQUENCE SPECIFICATIONS

Sequence specifications for an ADDROUT sort are similar to specifications for a tag-along sort.

There are two exceptions for an ADDROUT sort:

- Code SORTA in columns 7-11 on the header specification to identify an ADDROUT sort is to be performed.
- Do not include output record length in columns 29-32 on the header specification, because it is always three bytes.

Figure 4-5 is a column summary of the sequence specifications. If you need further explanation of column-by-column entries, refer to the *IBM System/3 Disk System Disk Sort Reference Manual,* SC21-7522.



IBM

International Business Machines Corporation

Form X21-9089
Printed in U.S.A.

SEQUENCE SPECIFICATIONS

Page 1 2

Header

Line: Number (3 4 5), Type (6) | Job: SORT, MERGE, MATCH, SELECT, SORTA, SORTR (7 8 9 10 11 12) | Longest Total Control Field of any Record Type (13 14 15 16 17) | Sequence (A/D/S) (18) | MATCH: Stacker Select S U, S M, P M, P U, S O, P O (19–24); Number (1/N) (25); Alternate Coll. Seq. (S) (26) | Print Option (27) | SORTR: Output Option (28); Output Record Length (29 30 31 32) | Reserved (33–39) | Job Description (40–72)

0 0 0 H SORTA ①

Record Type

Line: Number (3 4 5) | Type (I/O) (6) | Continuation (A/O) (7) | C/Z/D/P/U (8) | Stacker Select (9) | Factor 1: Location From (10 11 12), To (13 14 15 16) | Rel. EQ NE LT GT LE GE (17 18) | F/C (19) | Factor 2 (Field or Constant): Location From (20 21 22 23), To (24 25 26 27); Constant (28–39) | Comments: Record Name (40–47)

Lines 01, 02, 03, 04, 05, 06 ② ③

Control Field and Data Field

Line: Number (3 4 5) | Type (6) | Type (N/O/F/D) (7) | C/Z/D (8) | Location From (9 10 11 12), To (13 14 15 16) | Forced: Record Character (17), Substitute Character (18) | Continuation (19) | (20–39) | Field Name (40–45)

Lines 07 F ④, 08 F, 09 F ⑤, 10 F, 11 F ⑥, 12 F, 13 F, 14 F

Consider specifications for the shaded columns:

① Identify the job as an ADDROUT sort job

② Select input records by comparing the contents of a field against a constant

③ Select input records by comparing two fields in the records

④ Describe normal control fields

⑤ Describe opposite control fields

⑥ Describe forced control fields

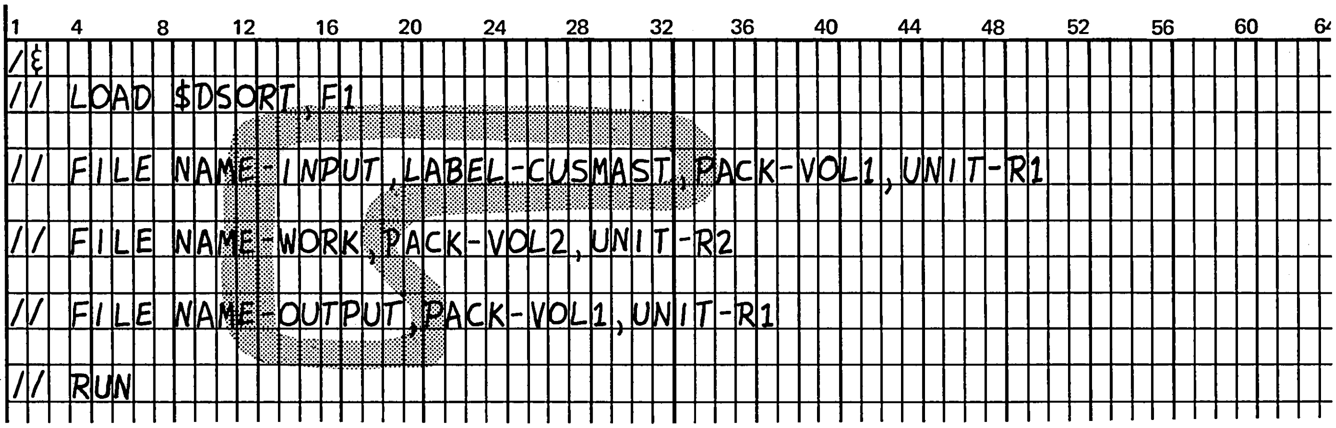
Figure 4-5. Column Summary (ADDROUT Sort)

OCL STATEMENTS

In order for the Disk Sort program to do the job you specify, you must use certain OCL statements. Figure 4-6 explains these statements.

Parameter summaries for all OCL control statements are explained in the *IBM System/3 Disk System Operation Control Language and Disk Utilities Reference Manual*, GC21-7512.

OCL statements and sequence specifications can be stored in the source library and loaded into main storage. Chapter 8, *Storing Programs and Procedures on Disk*, describes how to store programs and procedures in a source library.


1
4
8
12
16
20
24
28
32
36
40
44
48
52
56
60
64

```
/&
// LOAD $DSORT,F1

// FILE NAME-INPUT,LABEL-CUSMAST,PACK-VOL1,UNIT-R1

// FILE NAME-WORK,PACK-VOL2,UNIT-R2

// FILE NAME-OUTPUT,PACK-VOL1,UNIT-R1

// RUN
```

// LOAD $DSORT -- This statement tells disk system management to load the disk sort program from the fixed disk. $DSORT is the IBM name for the Disk Sort program. F1 is a fixed disk.

// FILE -- As you know, every file used in a program must be defined by a // FILE statement. For a disk sort job, you must always define three files: an input file, a work file, and an output file. The input file is the file you want sorted. The work file is space on the disk that the program uses to do the sort job. The output file is the new file created from the input file as a result of the sort job.

The keyword parameters for NAME are INPUT, WORK, and OUTPUT. These three words are predetermined by IBM. For the program to use a file in one of these three ways, you must correctly use these three words.

// RUN -- A RUN statement is always the last OCL statement for a job.

Figure 4-6. OCL Statements for a Disk Sort Job

EXAMPLE: ADDROUT SORT

You have a customer master file containing:

- Customer number and addresses.
- Salesman number.
- Accounts receivable amount.

The file is an indexed file processed by customer number throughout the month for invoicing and at the end of every month for customer statements. Along with the monthly statements, the sales department wants the accounts receivable for customers owing more than $2500.00 to be printed by customer number within salesman number. This informs each salesman which of his customers have large accounts receivable outstanding.

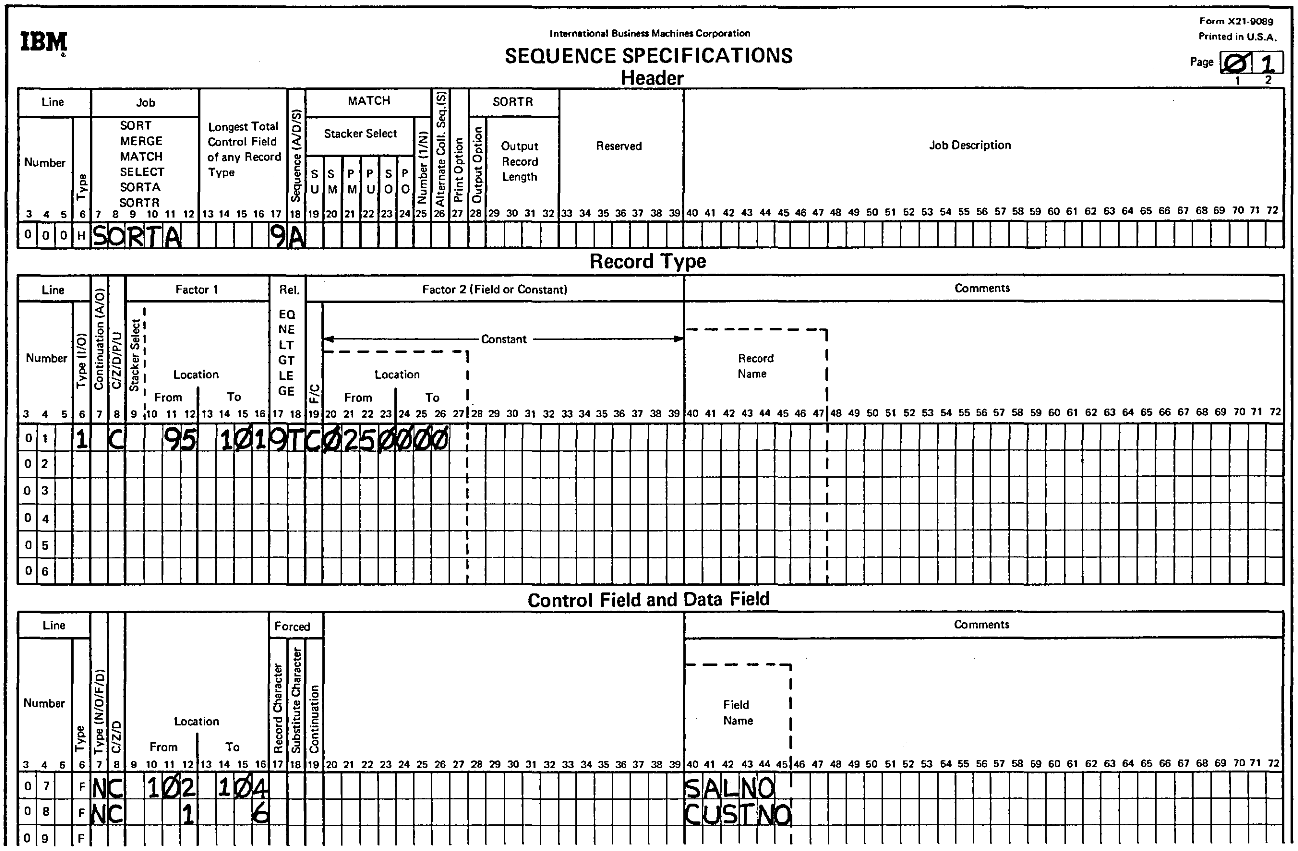
In order to do this, records of those customers who owe more than $2500.00 must be sorted according to salesman number. Because there is not enough disk space for a tag-along sort, an ADDROUT sort is used.

Figure 4-7 is a Sequence Specification Sheet for this sort. SORTA in columns 7-11 on the header specification indicates that the Disk Sort program is to perform an ADDROUT sort. The sum of lengths of the control fields is nine. (Salesman number is three positions long and the customer number is six positions long.)

Because only specific customer records are to be included in the sort, record type specifications must be coded. *I* in column 6 indicates that only those customer records with a balance of more than $2500 are to be included in the sort. (Positions 95 through 101 of the input record contain the amount owed to the company.)

The field specifications indicate that two input record fields are to be used as control fields: salesman number (SALNO) in positions 102 through 104 of the input record, and customer number (CUSTNO) in positions 1 through 6.

The *N* in column 7 of the field specification and the *A* in column 18 of the header specification indicate that the control fields are to be sorted in normal ascending sequence.



IBM

International Business Machines Corporation

SEQUENCE SPECIFICATIONS

Form X21-9089
Printed in U.S.A.

Page Ø1

Header

Line Number (3-5)	Type (6)	Job: SORT MERGE MATCH SELECT SORTA SORTR (7-12)	Longest Total Control Field of any Record Type (13-17)	Sequence (A/D/S) (18)	MATCH Stacker Select SU SM PM PU SO PO (19-24)	Number (1/N) (25)	Alternate Coll. Seq.(S) (26)	Print Option (27)	SORTR Output Option (28)	SORTR Output Record Length (29-32)	Reserved (33-39)	Job Description (40-72)
000	H	SORTA	9	A								

Record Type

Line Number (3-5)	Type (I/O) (6)	Continuation (A/O) (7)	C/Z/D/P/U (8)	Stacker Select (9)	Factor 1 Location From (10-12)	Factor 1 Location To (13-16)	Rel. EQ NE LT GT LE GE (17-18)	F/C (19)	Factor 2 (Field or Constant) Constant / Location From (20-23) To (24-27) (20-39)	Comments Record Name (40-72)
01	1		C		95	1Ø1	GT	C	Ø25ØØØØ	
02										
03										
04										
05										
06										

Control Field and Data Field

Line Number (3-5)	Type (6)	Type (N/O/F/D) (7)	C/Z/D (8)	Location From (9-12)	Location To (13-16)	Forced Record Character (17)	Forced Substitute Character (18)	Continuation (19)	(20-39)	Comments Field Name (40-72)
07	F	N	C	1Ø2	1Ø4					SALNO
08	F	N	C	1	6					CUSTNO
09	F									

Figure 4-7. Sequence Specifications to Sort Customer Records Owing More than $2,500.00

1. What output is created by the tag-along sort? How is this output used in an RPG II program?

2. What output is created by the ADDROUT sort?

3. What disk areas are required by the sort program?

4. Calculate the total disk space requirements for the following sort using ADDROUT sort:

   - 2000 records.
   - 100 bytes per record.
   - 10 byte control field.
   - 9K bytes of storage available for the Disk Sort program.

5. What are the primary reasons for choosing the ADDROUT sort over the tag-along sort?

6. Fill out the sort specification sheets for the following job, using the ADDROUT sort:

   Sort the receipt records in an inventory transaction file into purchase order number sequence to be used for purging a file of outstanding purchase orders. Receipts all have a *1* in position 96. Other transactions in the file have codes other than *1* in position 96. The purchase order number is in positions 13-18 of the records.

Answers to Review 4

1. The output of the tag-along sort is a consecutive data file. It is processed by an RPG II program as any other consecutive file would be processed.

2. The output of the ADDROUT sort is a file of the relative record numbers of records in the input file. The relative record numbers are in the sequence in which you wish to process the input file.

3. The disk areas required are: the input file, a work area, and an output area.

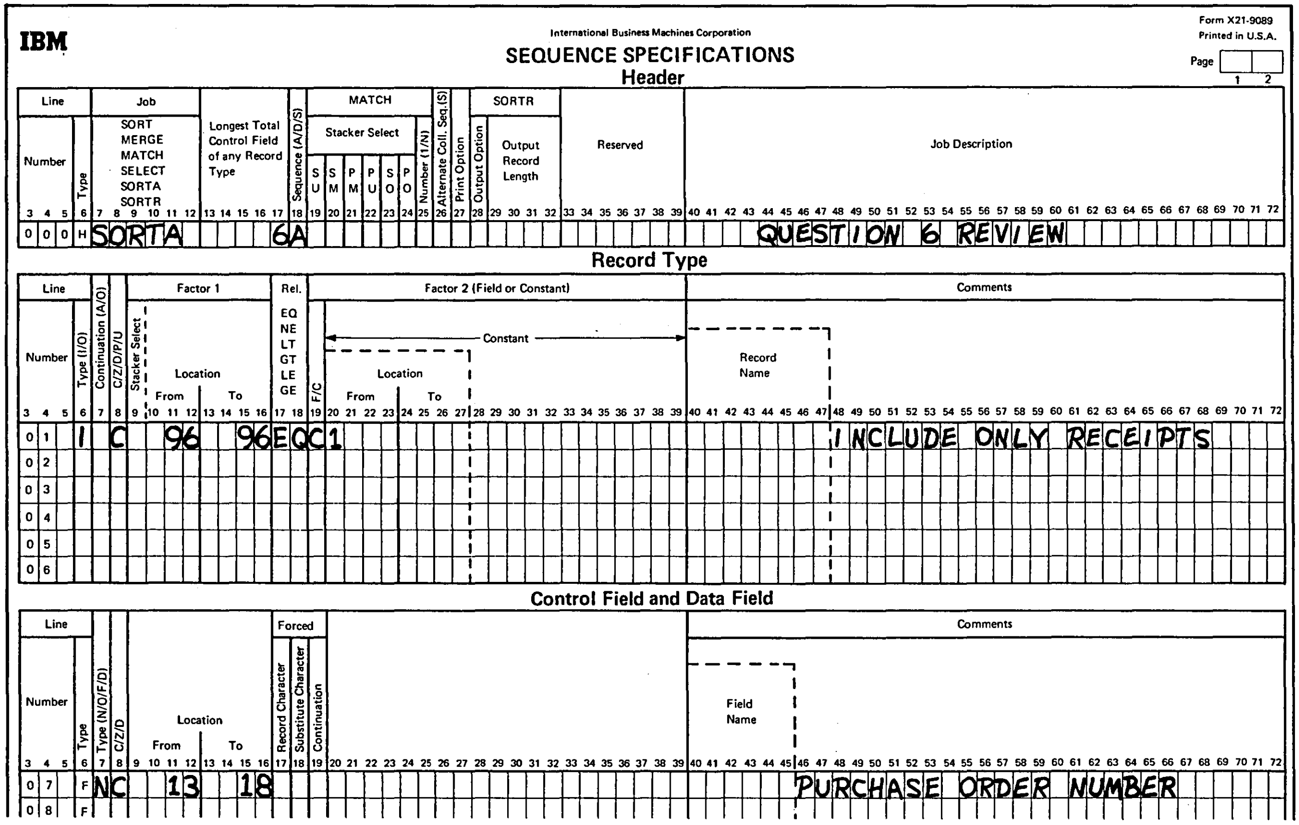
4. 2000 records X 100 bytes per record = 200,000 bytes.
200,000 bytes ÷ 6144 = 32.55 or 33 tracks for the input file.

   2,000 records X 13 bytes (3 byte relative record numbers, 10 byte control field) = 26,000 bytes work records.
26,000 bytes X 1.15 file size factor = 29,900 bytes
29,900 bytes ÷ 6144 = 4.87 or 5 tracks for the work file.

   2,000 records X 3 byte relative record numbers = 6,000 bytes.
6,000 bytes ÷ 6144 = .98 or 1 track for the output file.

5. The primary reasons for choosing ADDROUT sort over tag-along sort should be disk storage capacity and less time to sort the file.

6. See specification sheet.



IBM

International Business Machines Corporation

SEQUENCE SPECIFICATIONS

Form X21-9089
Printed in U.S.A.

Page □□

Header

Line Number (3-5)	Type (6)	Job: SORT MERGE MATCH SELECT SORTA SORTR (7-12)	Longest Total Control Field of any Record Type (13-17)	Sequence (A/D/S) (18)	MATCH Stacker Select S U (19)	S M (20)	P M (21)	P U (22)	S O (23)	P O (24)	Number (1/N) (25)	Alternate Coll. Seq.(S) (26)	Print Option (27)	Output Option (28)	SORTR Output Record Length (29-32)	Reserved (33-39)	Job Description (40-72)
000	H	SORTA	6	A													QUESTION 6 REVIEW

Record Type

Line Number (3-5)	Type (I/O) (6)	Continuation (A/O) (7)	C/Z/D/P/U (8)	Stacker Select (9)	Factor 1 Location From (10-12)	Factor 1 Location To (13-16)	Rel. EQ NE LT GT LE GE (17-18)	F/C (19)	Factor 2 (Field or Constant) Location From / Constant (20-39)	Factor 2 Location To	Record Name (40-47)	Comments (48-72)
01	I		C		96	96	EQ	C	1			INCLUDE ONLY RECEIPTS
02												
03												
04												
05												
06												

Control Field and Data Field

Line Number (3-5)	Type (6)	Type (N/O/F/D) (7)	C/Z/D (8)	Location From (9-12)	Location To (13-16)	Forced Record Character (17)	Substitute Character (18)	Continuation (19)	(20-39)	Field Name (40-45)	Comments (46-72)
07	F	N	C	13	18						PURCHASE ORDER NUMBER
08	F										

CHAPTER

PROCESSING DISK FILES BY RECORD ADDRESS FILES

5

CHAPTER 5 DESCRIBES:

Random processing by ADDROUT files.

Considerations when processing by ADDROUT files.

RPG II specifications to process by ADDROUT files.

Processing indexed files sequentially within limits.

How to create a record address file containing record key limits.

RPG II specifications to process sequentially within limits.

BEFORE READING THIS CHAPTER YOU SHOULD BE ABLE TO DESCRIBE:

How a file is sorted to produce an ADDROUT file.

Indexed file organization (record keys).

Primary files, secondary files, and the end-of-file condition.

Alphameric and numeric character sets.

The purpose of using RPG II File Description and Extension Sheets.

AFTER READING THIS CHAPTER YOU SHOULD BE ABLE TO:

Describe ADDROUT files.

Describe how to process a file by an ADDROUT file.

List the considerations when processing by ADDROUT files.

Code RPG II specifications to process a file using an ADDROUT file.

Describe record address files containing record key limits.

Describe how to process an indexed file sequentially within limits.

Code RPG II specifications to process an indexed file sequentially within limits.

List the rules for creating record address files containing record key limits.

INTRODUCTION

Record address files are input files that indicate which records are to be read from disk files and the order in which the records are to be read. There are two types of record address files:

- Files containing relative record numbers.
- Files containing record-key limits.

Files Containing Relative Record Numbers (ADDROUT Files)

A record address file that contains relative record numbers is called an ADDROUT file. (ADDROUT files are produced by the Disk Sort program.) ADDROUT files are comprised of binary 3-byte relative record numbers that indicate the relative position (first, twentieth, ninety-ninth) of records in the file to be processed.

An ADDROUT file can only be a disk file. Any file organization can be processed using an ADDROUT file.

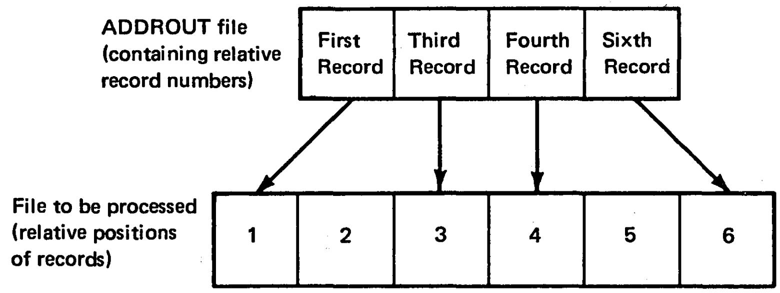
Files Containing Record Key Limits

A record address file with record key limits contains the lowest and the highest key fields for a specified section of an indexed file. Record address files containing record key limits can be entered from disk, card, or a printer-keyboard. They are used to process only indexed files. When a section of an indexed file is processed using record key limits, the processing method is known as *sequential within limits.*

Example: You have an indexed file, but want to process only the records with keys 2,000 through 3,000. The record key limits in this record address file would be 2,000 (lowest) and 3,000 (highest key field). Through RPG II specifications the appropriate section (records with keys 2,000 through 3,000) of the indexed file would be processed.

RANDOM PROCESSING BY ADDROUT FILES

All types of file organizations (sequential, indexed, or direct) used as primary or secondary files can be processed by ADDROUT file. When an RPG II program processes a file using an ADDROUT file, it reads a relative record number from the ADDROUT file, then locates and reads the record situated at that relative position in the file being processed. Only those records whose relative record numbers are located in the ADDROUT file are processed. Records are read in this manner until the end of the ADDROUT file is reached. Figure 5-1 shows an ADDROUT file used to process a disk file.


ADDROUT file (containing relative record numbers)
First Record
Third Record
Fourth Record
Sixth Record
File to be processed (relative positions of records)
1
2
3
4
5
6

Note: The RPG II program will read the ADDROUT file and find that the first record to be read is in relative position one of the file being processed. The second record to be read is in relative position three. Since all records are not read, processing by ADDROUT file is random processing.

Figure 5-1. Processing a File by ADDROUT File

Considerations for Using ADDROUT Files

The following three points should be considered for using ADDROUT files:

1. One file can be sorted in several sequences based on different control fields in each record of the file. Several ADDROUT files can be created from the same input file to be used as input to RPG II programs. For example, you have a transaction file in order by stock number. By performing two ADDROUT sorts on the transaction file, you could have one ADDROUT file sequenced by customer number and another by invoice number. Consequently, you can access the transaction file in an RPG II program by several sequences: stock number, customer number, or invoice number.

2. Less disk space is required to process a file by an ADDROUT file than by the output file of a tag-along sort becuase the output records of the ADDROUT file are only three bytes long.

3. If an ADDROUT file is used to process a multi-volume file, all volumes of that file must be mounted during processing becuase the next record required may be on any volume.

RPG II Specifications (Processing by ADDROUT File)

To process a file by an ADDROUT file in an RPG II program, additional entries must be made on the File Description and File Extension Sheets. (Input specifications need not be written for the ADDROUT file.)

File Description Specifications

FILES BEING PROCESSED MUST HAVE:

Column	Entry	Meaning
28 (Mode of Processing)	R	File is to be processed by ADDROUT file.
31 (Record Address Type)	I	File is to be processed by relative record numbers from ADDROUT file.

ADDROUT FILES MUST HAVE:

Column	Entry	Meaning
15 (File Type)	I	File is an input file.
16 (File Designation)	R	File is a record address.
17 (End-of-File)	E	Records from the file must be processed before the program can end.
19 (File Format)	F	File consists of fixed-length records.
20-23 (Block) Length)	3	Block length is three.
24-27 (Record Length)	3	The ADDROUT file consists of 3-byte relative record numbers.
31 (Record Address Type)	I	File is an ADDROUT file
32 (Type of File Organization)	T	File is an ADDROUT file.
39 (Extension Code)	E	File must be further defined on the Extension sheet.
40-46 (Device)	DISK	File is a disk file.
68-69 (Number of Extents)	number of volumes containing the file	01 is assumed if you do not code this entry.

Figure 5-2 is a sample File Description Sheet describing two input files: a master file and the ADDROUT file used to process it. The master file to be processed is coded the same as any other input file with two exceptions: Column 28 contains an *R* and column 31 contains an *I*. These two columns indicate that the file is processed by an ADDROUT file.

The ADDROUT file contains an *I* in column 15 indicating that it is an input file. Columns 16, 31, and 32 contain an *R, I,* and *T* respectively. These three columns indicate that the file is a record address file consisting of relative record numbers. Columns 20-27 and 29-30 contain 3 and 03 respectively. These columns indicate the block and record length of the file and the length of the record address field. Column 39 contains an *E* indicating that the ADDROUT file is further defined on the File Extension Sheet.

File Extension Specifications

If you are processing by ADDROUT file, entries in columns 11-18 and 19-26 on the File Extension Sheet must be coded:

- Columns 11-18 (From Filename) must contain the name of the record address file. This must be the same name given to the record address file on the File Description Sheet.

- Columns 19-26 (To Filename) must contain the name of the file to be processed. This must be the same filename that was defined on the File Description Sheet. This entry indicates that the file is to be processed by the ADDROUT file coded in columns 11-18.

Figure 5-3 is a sample File Extension Sheet corresponding to the File Description Sheet in Figure 5-2. The entries tell the compiler that MASTER is to be processed by the ADDROUT file labeled ADDROUT.

Example: Processing by ADDROUT File

You have a customer master file containing:

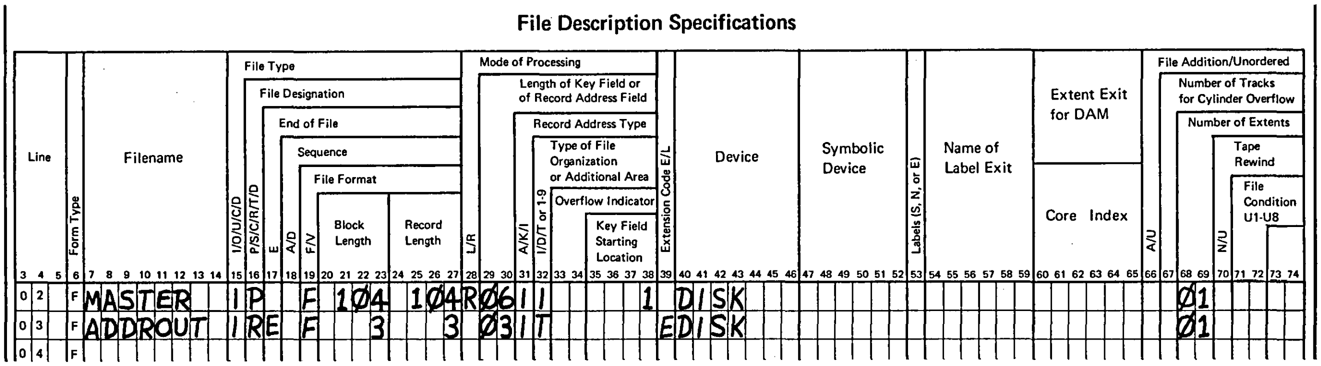
- Customer numbers and addresses.

- Salesman numbers.

- Accounts receivable amounts.

The file is an indexed file processed by customer number throughout the month for invoicing and at the end of every month for customer statements. Along with the monthly statements, the sales department wants the accounts receivable for all customers owing more than $2500.00 to be printed by customer number within salesman number. This informs each salesman which of his customers have large accounts receivable outstanding.

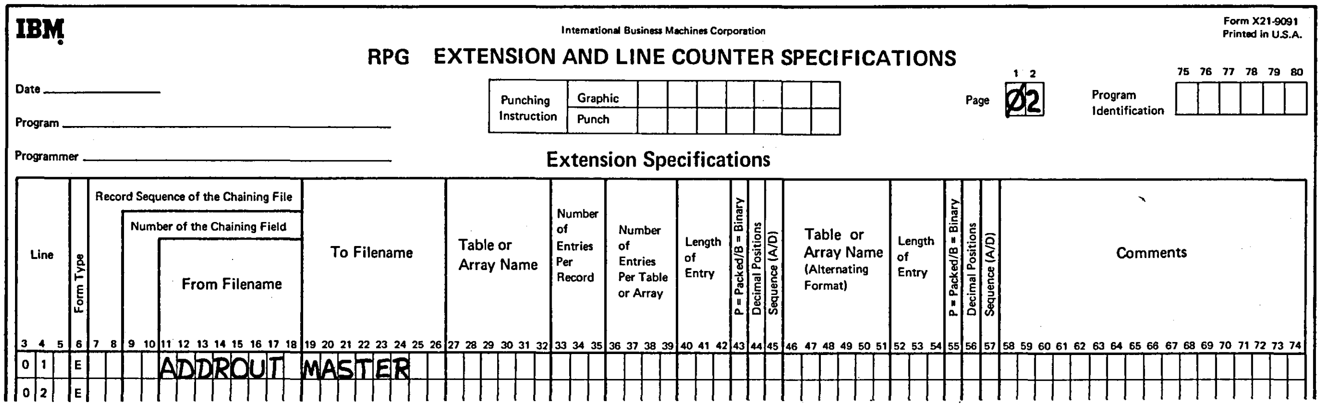
In order to do this, the records of customers who owe more than $2500.00 must be sorted according to salesman number. You sort the file using ADDROUT sort because there is not enough disk storage to use a tag-along sort. After the file is sorted, you have an ADDROUT file consisting of those records to be printed. The output of the sort becomes input to an RPG II program. Figures 5-2 and 5-3 show the RPG II entries required to use the ADDROUT file as input to an RPG II program.



File Description Specifications

Line	Form Type	Filename	File Type (I/O/U/C/D)	File Designation (P/S/C/R/T/D)	End of File (E)	Sequence (A/D)	File Format (F/V)	Block Length	Record Length	Mode of Processing (L/R)	Length of Key Field or of Record Address Field	Record Address Type (A/K/I)	Type of File Organization or Additional Area (I/D/T or 1-9)	Overflow Indicator	Key Field Starting Location	Extension Code E/L	Device	Symbolic Device	Labels (S, N, or E)	Name of Label Exit	Extent Exit for DAM / Core Index	A/U	Number of Extents	Tape Rewind	File Condition U1-U8
02	F	MASTER	I	P			F	1Ø4	1Ø4	R	Ø6	I	I		1		DISK						Ø1		
03	F	ADDROUT	I	R	E		F	3	3		Ø3	I	T			E	DISK						Ø1		
04	F																								

Figure 5-2. File Description Sheet for Processing by ADDROUT File



IBM

International Business Machines Corporation

Form X21-9091
Printed in U.S.A.

RPG EXTENSION AND LINE COUNTER SPECIFICATIONS

Date

Program

Programmer

Punching Instruction: Graphic / Punch

Page Ø2

Program Identification

Extension Specifications

Line	Form Type	Record Sequence of the Chaining File	Number of the Chaining Field	From Filename	To Filename	Table or Array Name	Number of Entries Per Record	Number of Entries Per Table or Array	Length of Entry	P = Packed/B = Binary	Decimal Positions	Sequence (A/D)	Table or Array Name (Alternating Format)	Length of Entry	P = Packed/B = Binary	Decimal Positions	Sequence (A/D)	Comments
01	E			ADDROUT	MASTER													
02	E																	

Figure 5-3. File Extension Sheet for Processing by ADDROUT File

SEQUENTIALLY PROCESSING AN INDEXED FILE WITHIN LIMITS

Processing a section of an indexed file sequentially is sometimes necessary. For example, you have a customer file with account numbers ranging from 1000 to 4999. Each week statements are sent to 1,000 of the customers. By using a record address file containing record key limits, you can tell the RPG II program what records are to be processed. This type of processing is known as sequential within limits.

Creating a Record Address File Within Limits

A record address file containing record key limits can be entered from a disk, card, or a printer-keyboard. The following rules must be observed when you are creating a record address file:

- You can use only one record address file for each RPG II program, but the record address file can contain several sets of limits.

- Only one set of limits is allowed on each record in a record address file. The length of each record is at least twice as long as the length of the record key, since each record is comprised of two keys.

- The low record key must begin in position one of the record. Each record is twice as long as the record key since each record is comprised of two keys.

- The high record key must immediately follow the low record key. No spaces are allowed between the two keys.

  If the key field were four bytes long and the low record key were 2000 and the high record key were 2999, the record would look like this:

  ```
  20002999
  ```

  Each record key can be from 1-29 characters in length.

- The length of the keys must be equal. Therefore, it may be necessary to place leading zeros in a numeric record key to make the length of the keys equal. For example, if the low record key were three positions (200) and the high record key were four positions (2999), a zero must be placed before the 200 to make it a four-position number. The record would look like this:

  ```
  02002999
  ```

  Each key length must equal the key field length you specify in columns 29-30 of the File Description Sheet. Each key length in the record address file must be equal to the key length in the indexed file.

- An alphameric record key can contain blanks.

- The same set of limits can appear on more than one record in a record address file. Therefore, records within a set of limits can be processed as many times as you wish.

- The two record keys in a set of limits can be identical. For example, both the low and high record key can be 2999. In this case, only one record (2999) will be processed.

Processing Sequentially Within Limits

Processing a section of an indexed file by record keys is known as *sequential within limits*. The RPG II program uses one set of limits (one record in a record address file) at a time. Records are read according to the arrangement of the record keys in the section of the indexed file specified by the limits. When the records identified in one section are read, the program reads another set of limits from the record address file. The program continues reading records in this manner until the end of the record address file is reached or an end-of-file condition on the indexed file is reached.

An end-of-file condition can occur if a file being processed ends before the high record key in a set of limits is reached. For example, if you specify the high record key as 2999 and the last record in that section of the file is 2800, the program ends when record 2800 is processed if there are no other sets of limits to be processed.

RPG II Specifications (Sequential Processing Within Limits)

To process a file by a record address file using RPG II, you must make additional entries on the File Description and File Extension Sheets. (Input specifications need not be written for the record address file.)

File Description Specifications

INDEXED FILE TO BE PROCESSED MUST HAVE:

Column	Entry	Meaning
28 (Mode of Processing)	L	Records are to be read from this file sequentially within limits.
31 (Record Address Type)	A	Record keys are used in processing and loading indexed files.

RECORD ADDRESS FILE CONTAINING RECORD KEYS MUST HAVE:

Column	Entry	Meaning
15 (File Type)	I	File is an input file.
16 (File Designation)	R	File is a record address file.
17 (End-of-File)	E	File must be processed before the program can end.
19 (File Format)	F	File contains fixed-length records.
20-23 (Block Length)	number	Block length for the file.
24-27 (Record Length)	number	Record length for the file.
29-30 (Length of Record Address Field)	length of the record key	Maximum length is 29 positions.
39 (Extension Code)	E	File is further defined on the Extension Sheet.
40-46 (Device)	input device	Input device for the file.

Figure 5-4 is a sample File Description Sheet describing two input files: an indexed file (MCUSTFLE) to be processed and a record address file (RAFILE) to process it. MCUSTFLE is coded as any other indexed file with two exceptions: column 28 contains an *L* and column 31 contains an *A*. Together these two columns indicate that MCUSTFLE is to be processed sequentially within limits.

RAFILE contains *R* in column 16 indicating that the file is a record address file. Columns 29-30 contain the length of the record key. In this case the record key is seven positions long. Column 39 contains an *E* indicating the file is further defined on the File Extension Sheet.

File Extension Specifications

If you are processing a file using a record address file, entries in columns 11-18 and 19-26 of the File Extension Sheet must be coded:

- Columns 11-18 (From Filename) must contain the name of the record address file. This must be the same name as the record address file on the File Description Sheet.
- Columns 19-26 (To Filename) must contain the name of the indexed file to be processed. This must be the same filename that was defined on the File Description Sheet.

These two entries indicate that the indexed file is to be processed by the record address file named in columns 11-18.

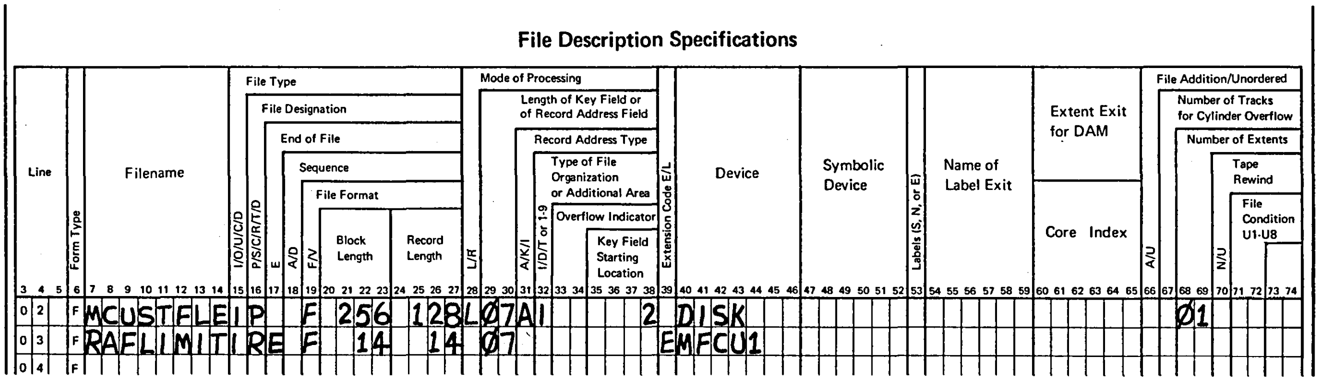
Figure 5-5 is a sample Extension Sheet corresponding to the File Description Sheet in Figure 5-4. The entries indicate that RAFILE is used to process MCUSTFLE sequentially within limits.

Example: Sequentially Processing Part of an Indexed File

You have a master customer file on disk consisting of 128-character records. The file is organized by customer number within customer class. Customers are separated into such classes as wholesalers or retailers. Together the customer number and class form a composite customer account number (key) in the form: ccnnnnn.

cc is the customer class and nnnnn is the customer number. Customer classes begin at 01 and are in ascending order. Within each customer class, customer numbers range from 00000-99999.

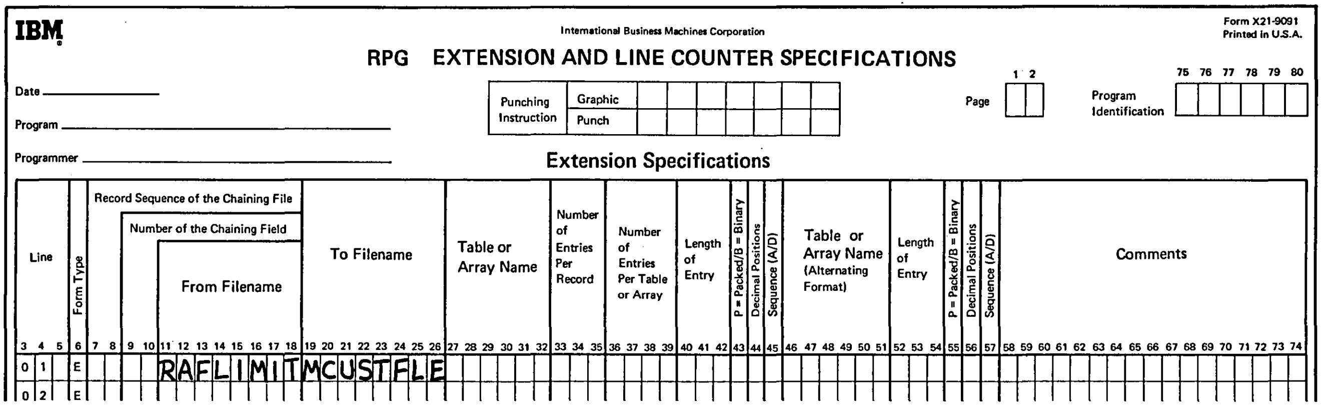
You must prepare separate reports by the customer class categories for sales analysis purposes. A record address file can be used to supply the particular class categories and customer number ranges as shown in Figure 5-6. The key in each disk record begins in column 2 and the record address file is loaded in MFCU1. Figures 5-4 and 5-5 show the necessary File Description and Extension entries for this job.



File Description Specifications

Line (3-5)	Form Type (6)	Filename (7-14)	I/O/U/C/D (15)	P/S/C/R/T/D (16)	E (17)	A/D (18)	F/V (19)	Block Length (20-23)	Record Length (24-27)	L/R (28)	Length of Key Field or of Record Address Field (29-30)	A/K/I (31)	I/D/T or 1-9 (32)	Overflow Indicator (33-34)	Key Field Starting Location (35-38)	Extension Code E/L (39)	Device (40-46)	Symbolic Device (47-52)	Labels (S, N, or E) (53)	Name of Label Exit (54-59)	Extent Exit for DAM / Core Index (60-65)	A/U (66)	Number of Tracks for Cylinder Overflow (67)	Number of Extents (68-69)	N/U (70)	Tape Rewind / File Condition U1-U8 (71-74)
02	F	MCUSTFLE	I	P			F	256	128	L	Ø7	A	I		2		DISK							Ø1		
03	F	RAFLIMIT	I	R	E		F	14	14		Ø7					E	MFCU1									
04	F																									

Figure 5-4. File Description Sheet for Processing by Record Address File Within Limits



IBM

International Business Machines Corporation

Form X21-9091
Printed in U.S.A.

RPG EXTENSION AND LINE COUNTER SPECIFICATIONS

Date

Program

Programmer

Punching Instruction | Graphic | Punch

Page 1 2

Program Identification 75 76 77 78 79 80

Extension Specifications

Line (3-5)	Form Type (6)	Record Sequence of the Chaining File (7-10)	Number of the Chaining Field	From Filename (11-18)	To Filename (19-26)	Table or Array Name (27-32)	Number of Entries Per Record (33-35)	Number of Entries Per Table or Array (36-39)	Length of Entry (40-42)	P = Packed/B = Binary (43)	Decimal Positions (44)	Sequence (A/D) (45)	Table or Array Name (Alternating Format) (46-51)	Length of Entry (52-54)	P = Packed/B = Binary (55)	Decimal Positions (56)	Sequence (A/D) (57)	Comments (58-74)
01	E			RAFLIMIT	MCUSTFLE													
02	E																	

Figure 5-5. File Extension Sheet for Processing by Record Address File Within Limits.

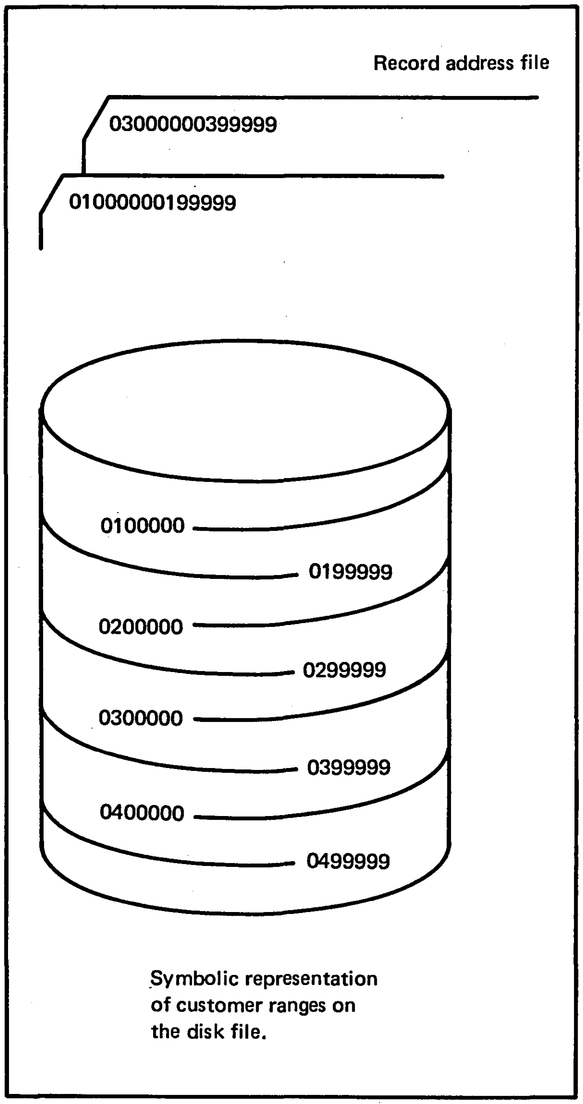


Figure 5-6. Files for the Example of Processing an Indexed File Sequentially Within Limits

1. How is the output of the ADDROUT sort used by an RPG II program for processing data?

2. Fill out the RPG II File Description and File Extension Specification Sheets for a program to access a data file using output from the ADDROUT sort.

   The data file is an indexed sequential file that contains 5000 records. Each record is 96 positions. The account number is used as the key field and is contained in the first six positions of the record.

3. In processing between limits how are the limits supplied to the program?

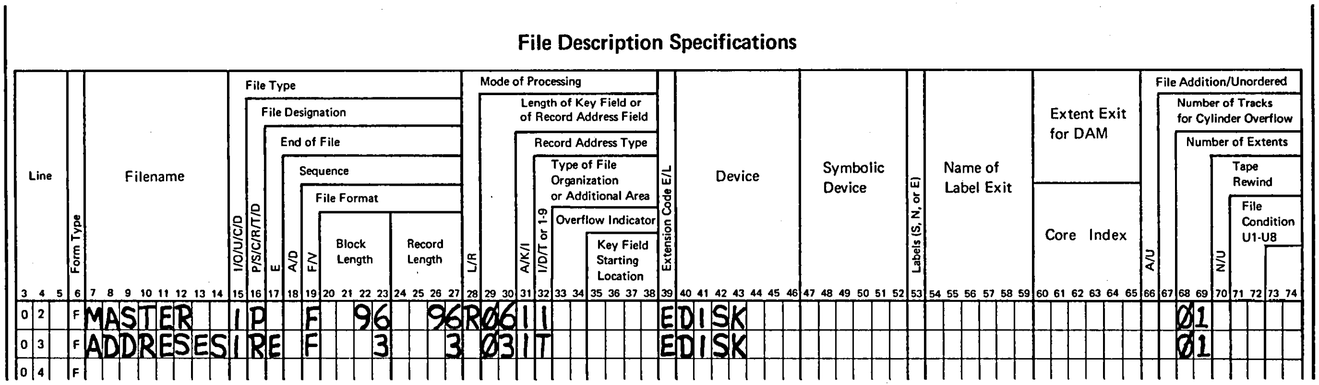
4. Describe briefly how an indexed file is processed between limits.

5. Code the File Description and File Extension Specification Sheets to define an indexed file and the record address file used to process it between limits. The key for the indexed file is customer number and is stored in positions one through six. The record address file will be read in the primary hopper of the MFCU.

Answers to Review 5

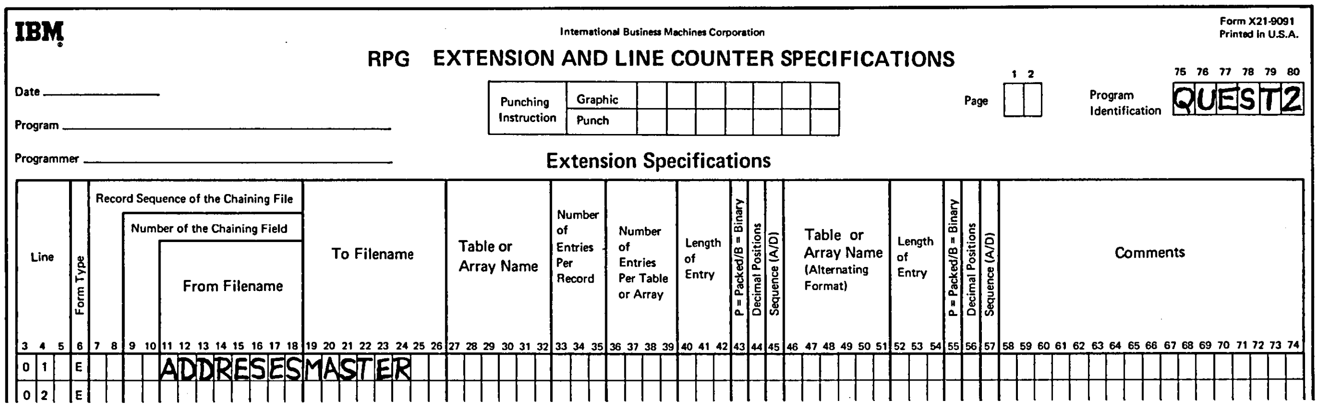
1. The output file created by an ADDROUT sort is specified as a record address file to the RPG II program. The record address file is used to access a data file for processing. The record address file contains the relative record numbers of the records in the data file to be processed. Records in the data file are accessed in the sequence in which their relative record numbers appear in the record address file.

2. See specification sheets.



File Description Specifications

Line (3-5)	Form Type (6)	Filename (7-14)	File Type I/O/U/C/D (15)	File Designation P/S/C/R/T/D (16)	End of File E (17)	Sequence A/D (18)	File Format F/V (19)	Block Length (20-23)	Record Length (24-27)	Mode of Processing L/R (28)	Length of Key Field or of Record Address Field (29-30)	Record Address Type A/K/I (31)	Type of File Organization or Additional Area I/D/T or 1-9 (32)	Overflow Indicator (33-34)	Key Field Starting Location (35-38)	Extension Code E/L (39)	Device (40-46)	Symbolic Device (47-52)	Labels (S, N, or E) (53)	Name of Label Exit (54-59)	Extent Exit for DAM / Core Index (60-65)	A/U (66)	File Addition/Unordered (67)	Number of Extents / Number of Tracks for Cylinder Overflow (68-69)	N/U (70)	Tape Rewind (71-72)	File Condition U1-U8 (73-74)
02	F	MASTER	I	P			F	96	96	R	Ø6	I	I			E	DISK							Ø1			
03	F	ADDRESES	I	R	E		F	3	3		Ø3	I	T			E	DISK							Ø1			
04	F																										



IBM

International Business Machines Corporation

Form X21-9091
Printed in U.S.A.

RPG EXTENSION AND LINE COUNTER SPECIFICATIONS

Date ______

Program ______

Programmer ______

Punching Instruction	Graphic								
	Punch								

Page 1 2 □□

Program Identification (75-80) QUEST2

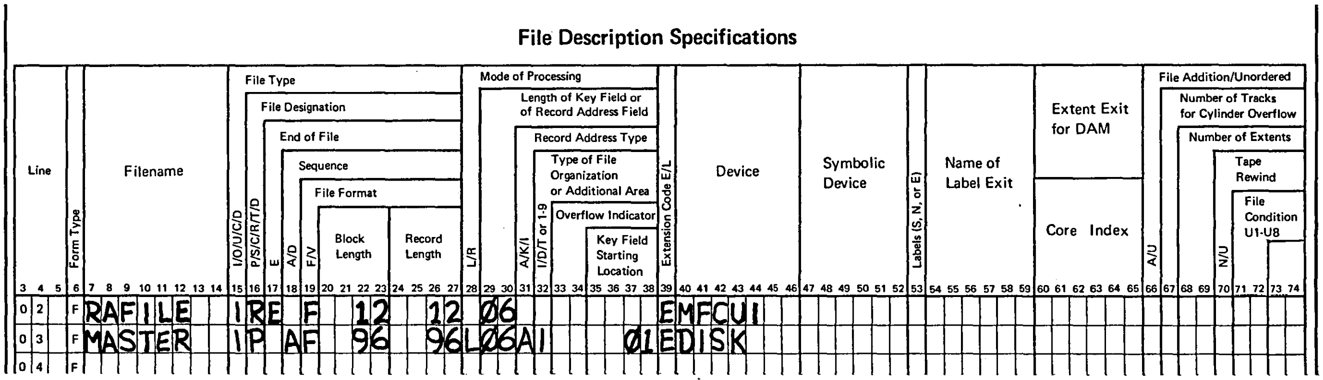
Extension Specifications

Line (3-5)	Form Type (6)	Record Sequence of the Chaining File (7-8)	Number of the Chaining Field (9-10)	From Filename (11-18)	To Filename (19-26)	Table or Array Name (27-32)	Number of Entries Per Record (33-35)	Number of Entries Per Table or Array (36-39)	Length of Entry (40-42)	P = Packed/B = Binary (43)	Decimal Positions (44)	Sequence (A/D) (45)	Table or Array Name (Alternating Format) (46-51)	Length of Entry (52-54)	P = Packed/B = Binary (55)	Decimal Positions (56)	Sequence (A/D) (57)	Comments (58-74)
01	E			ADDRESES	MASTER													
02	E																	

3. Limits are specified to the program via a record address file on cards, the printer-keyboard, or from disk.

4. The record address file contains the high and low keys to be processed. These are read from consecutive positions of the record address file records, beginning with the first position. The program accesses the record in the indexed file that has the low limit key and processes the file sequentially until the high limit key has been processed. Multiple sets of limits can be used in one program and the upper and lower limits for one set can be the same to process a single record.

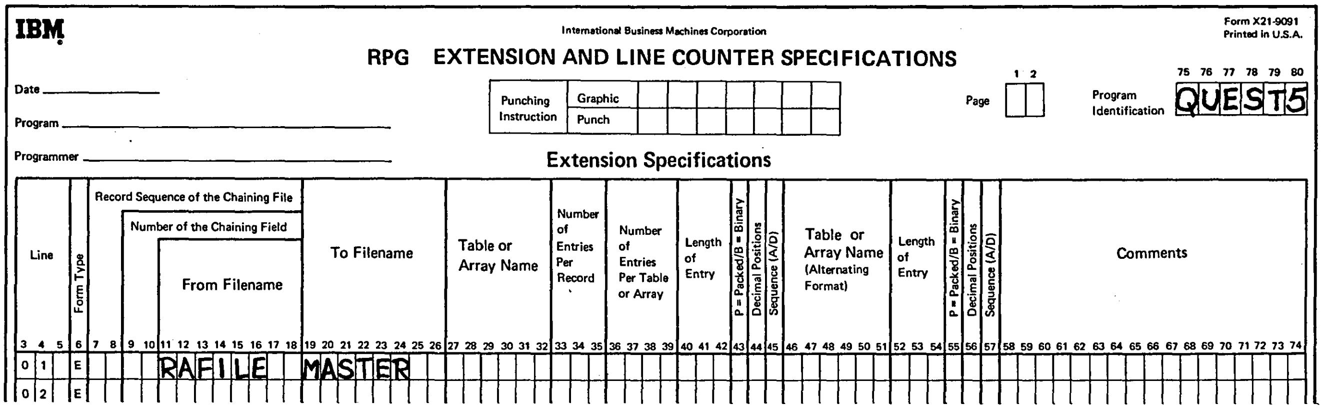
5. See specification sheets.



File Description Specifications

Line | Form Type | Filename | File Type: I/O/U/C/D | File Designation: P/S/C/R/T/D | End of File: E | Sequence: A/D | File Format: F/V | Block Length | Record Length | Mode of Processing: L/R | Length of Key Field or of Record Address Field | Record Address Type: A/K/I | Type of File Organization or Additional Area: I/D/T or 1-9 | Overflow Indicator | Key Field Starting Location | Extension Code E/L | Device | Symbolic Device | Labels (S, N, or E) | Name of Label Exit | Extent Exit for DAM | Core Index | A/U | File Addition/Unordered | Number of Tracks for Cylinder Overflow | Number of Extents | Tape Rewind: N/U | File Condition U1-U8

```
  3 4 5 6 7 8 9 10 11 12 13 14 15 ...
02 FRAFILE  IRE F  12  12 Ø6        EMFCU1
03 FMASTER  IP AF  96  96LØ6AI    Ø1EDISK
04 F
```



IBM

International Business Machines Corporation

Form X21-9091
Printed in U.S.A.

RPG EXTENSION AND LINE COUNTER SPECIFICATIONS

Date ______

Program ______

Programmer ______

Punching Instruction	Graphic								
	Punch								

Page □□ (1 2)

Program Identification (75–80) QUEST5

Extension Specifications

Line | Form Type | Record Sequence of the Chaining File | Number of the Chaining Field | From Filename | To Filename | Table or Array Name | Number of Entries Per Record | Number of Entries Per Table or Array | Length of Entry | P = Packed/B = Binary | Decimal Positions | Sequence (A/D) | Table or Array Name (Alternating Format) | Length of Entry | P = Packed/B = Binary | Decimal Positions | Sequence (A/D) | Comments

```
01 E    RAFILE  MASTER
02 E
```

CHAPTER

MULTI-VOLUME FILES

6

CHAPTER 6 DESCRIBES:

Multi-volume files.

Creating and processing multi-volume files.

Coding the OCL FILE statement and the RPG II File Description Sheet to create and process multi-volume files.

BEFORE READING THIS CHAPTER YOU SHOULD BE ABLE TO:

Describe the function of the RPG II File Description Sheet.

Describe the function and format of the FILE statement.

Describe sequential and direct file organization.

List the types of processing these file organizations permit.

Distinguish between a fixed and a removable disk.

Distinguish between cylinders and tracks.

AFTER READING THIS CHAPTER YOU SHOULD BE ABLE TO:

Define multi-volume files.

List the two ways multi-volume files are created and processed.

Code the FILE statement and RPG II File Description Sheet to create and process multi-volume files.

INTRODUCTION

A multi-volume file is a file that is contained on more than one disk (volume).

CREATING MULTI-VOLUME FILES

Coding parameters on the RPG II File Description Sheet and on the OCL FILE statement will cause disk system management to create a multi-volume file. Disk system management creates sequential or direct multi-volume files.

The way in which a multi-volume file is created depends on the number of disks the file will occupy and on the file organization (sequential or direct). Sequential files are created consecutively; direct files are created randomly.

Creating a Sequential File

When you create a multi-volume sequential file, records are placed in consecutive order on as many volumes as needed. When the first volume is filled, records are placed on the second volume.

You can mount all the disks to contain the file at the same time or you can add or replace disks as needed. When all the disks are mounted at the same time, the file is created as an *online,* multi-volume file. If disks are added or replaced during creation, the file is created as an *offline*, multi-volume file.

To create an offline, multi-volume, sequential file, you must use removable disks only. If you have a one-drive system, the first volume is created. Another disk is then placed on the disk drive, and the next volume is created. This is continued until the entire multi-volume file is completed. If you have a two-drive system, you can alternate the process from one drive to another. You can mount two removable disks, then after the first volume is created, you can replace it by another disk while the second volume is created. An offline, multi-volume, sequential file can have a maximum of 50 volumes.

When you create an online multi-volume file, you can use both fixed and removable disks. The file, however, cannot exceed the number of disks that can be on the system at one time. Therefore, a single-drive system is limited to a two-volume file, and a two-drive system can have a maximum file size of four volumes.

Creating a Direct File

Direct files are created randomly (Figure 6-1). Recall that records in sequential files are stored consecutively on the first volume until it is filled, then records are stored on the second volume. Unlike sequential file loading, direct files can have records placed alternately between the first and second volumes.

For example, assume you have an input file labeled INPUT that you want loaded on two volumes, the first of which can store 1000 records. The relative record numbers of the first five records are: 1, 1000, 3000, 500, and 4000 (Figure 6-2).

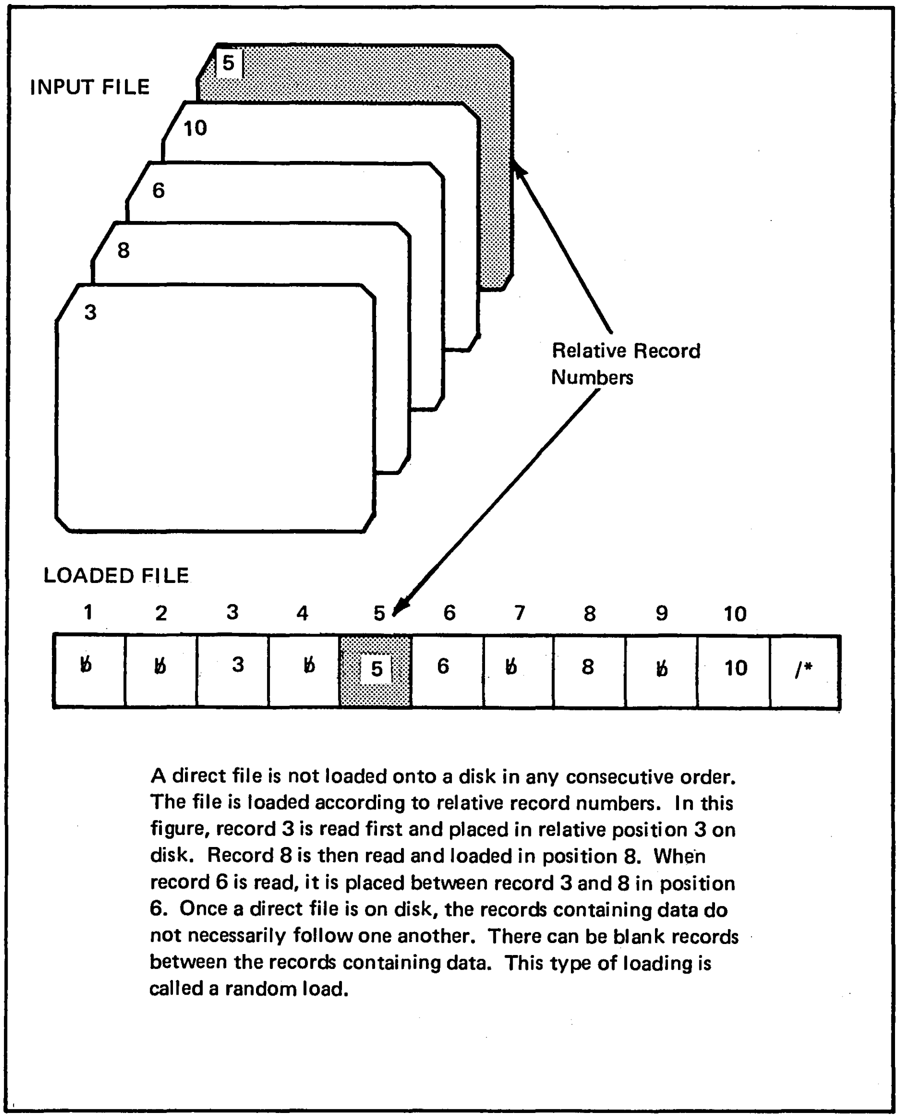
When records 1 and 1000 are read, they are placed in relative positions 1 and 1000 on the first volume. The next record is placed in relative position 3000 of the file, which is on the second volume. Records 500 and 4000 are then read and placed in their relative positions on volumes one and two respectively. Blanks are in all positions that do not contain records. Disk system management initializes the disk to blanks before creation of the file begins.

When creating a multi-volume direct file, the file must be created as an online, multi-volume file. The files can be contained on both fixed and removable disks. On a one-drive system, the file must be contained on two volumes. On a two-drive system, the maximum direct file size is four volumes.

Considerations for Creating Multi-Volume Files

No matter what type of file organization you use, you must consider these points when planning to use multi-volume files:

- Multi-volume files must be created on consecutive tracks of each volume. For example, a disk file cannot occupy tracks 20-30 and tracks 41-50 of the same volume. The file can occupy tracks 20-40 of one volume, or the data from tracks 41-50 can be placed on the second volume.
- No volume except the first volume of the file can contain scratch files during creation or addition to a file.
- Those volumes containing the multi-volume file cannot contain any other file, if you do not specify the file location for your multi-volume file.
- If you do specify a file location, ensure that no other files are on that volume in that location.
- Volume names must be unique.



1	2	3	4	5	6	7	8	9	10	
b̸	b̸	3	b̸	5	6	b̸	8	b̸	10	/*

A direct file is not loaded onto a disk in any consecutive order. The file is loaded according to relative record numbers. In this figure, record 3 is read first and placed in relative position 3 on disk. Record 8 is then read and loaded in position 8. When record 6 is read, it is placed between record 3 and 8 in position 6. Once a direct file is on disk, the records containing data do not necessarily follow one another. There can be blank records between the records containing data. This type of loading is called a random load.

Figure 6-1. Loading a Direct File

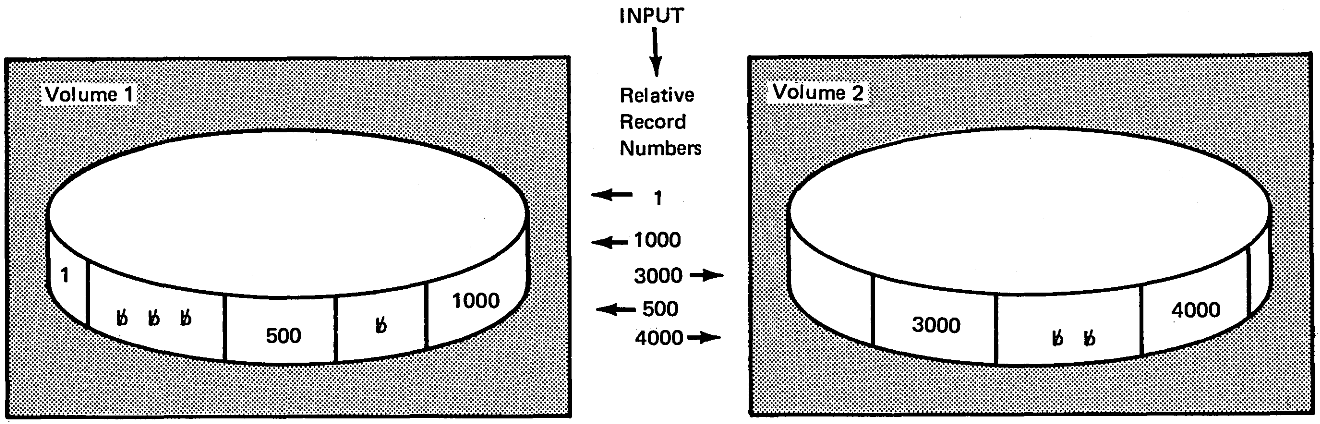


Figure 6-2. Loading a Direct File on Multi-Volumes

PROCESSING MULTI-VOLUME FILES

Multi-volume files can be processed consecutively or randomly. Disk system management reads all the records in a file when processing consecutively, but processes only specified records. When processing randomly, disk system management both reads and processes only specified records in a file.

Processing Files Consecutively

When you are consecutively processing multi-volume files with all volumes offline, all the volumes must be removable. If you have a one-drive system, you can mount a disk, wait until all of the records have been read, then mount the next disk. If you have two drives, you can mount two disks, wait until all of the records have been read from the first disk, then replace it with the next volume while your program reads from the second disk.

If you are consecutively processing multi-volume files with all volumes online, any combination of fixed and removable disks is permitted, but all must be mounted initially and remain mounted throughout the entire job.

When processing consecutively, you must consider the following:

- If you are creating or adding to a file, the job must not be cancelled between volumes, or the file must be completely reloaded. No more records can be added to the portion of the file that was completed because the program cannot recognize which disk was the last volume. (Records can be added only at the end of the last volume.) Records can be retrieved from it, however.

- As long as all file names and record lengths are identical, two files not created as a multi-volume file can still be processed as a multi-volume file. For example, two files could be created at separate times on different volumes. They could both be labeled FILEA and contain 128-position records. The two files can then be processed together as one multi-volume file. Records from this type of file can be consecutively retrieved or updated.

Processing Files Randomly

Because disk system management directly accesses specified records during random processing and the records can be on different volumes, all volumes of the multi-volume file must be online. The file can reside on both fixed and removable disks. If you have one drive, the multi-volume file is on two volumes. If you have two drives, the maximum multi-volume file size is four volumes. If a fixed disk is used, you can copy the file to the fixed disk prior to running the job and back to the removable disk for storage after the volume has been processed. This leaves the fixed disk free to perform other functions when the multi-volume file is not being processed.

Figure 6-3 is a summary of the maximum number of volumes permitted for multi-volume files.

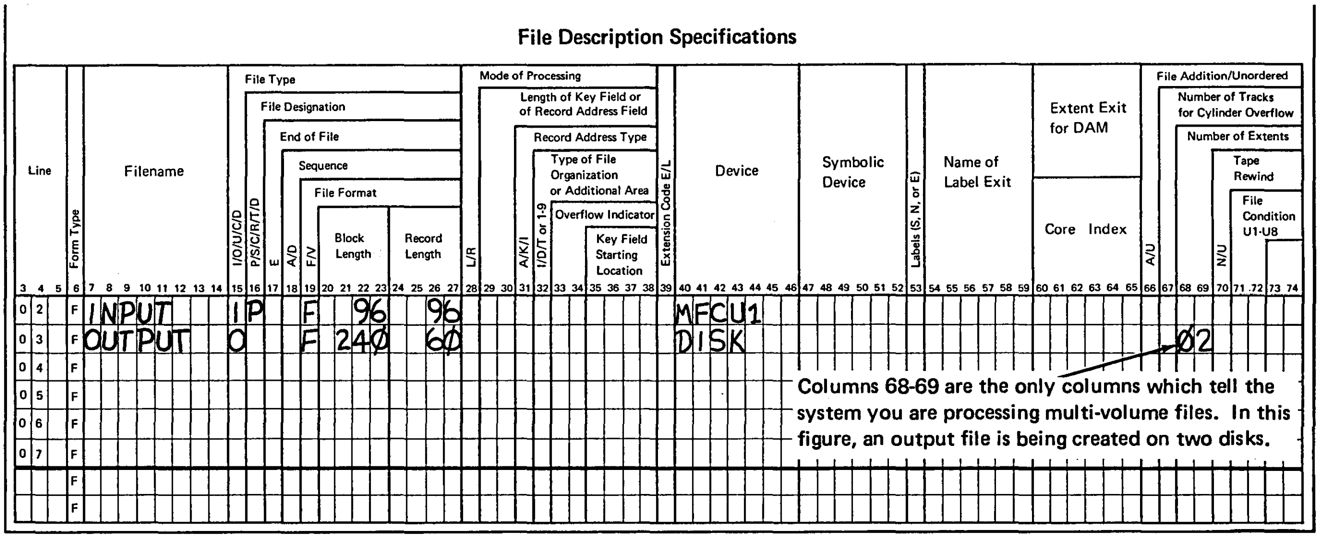
CODING THE RPG II FILE DESCRIPTION SHEET TO PROCESS MULTI-VOLUME FILES

When processing single volume files, you must enter 01 in columns 68-69 (Number of Extents) on the File Description Sheet. (An extent is definable area on disk where data is stored.) This entry tells the disk system management the number of volumes in your file. When processing multi-volume files, enter in these columns the total number of volumes that contain your file.

Figure 6-4 is an example of coding the Number of Extents specification.

	One Drive		Two Drives	
	Maximum Number of Volumes Allowed	Maximum Number of Volumes Online	Maximum Number of Volumes Allowed	Maximum Number of Volumes Online
Removable Disks Only	50	1	50	2
Removable and Fixed Disk	2	2	4	4

Figure 6-3. Maximum Number of Volumes for Multi-Volume Files



File Description Specifications

Line (3-5)	Form Type (6)	Filename (7-14)	File Type I/O/U/C/D (15)	File Designation P/S/C/R/T/D (16)	End of File E (17)	Sequence A/D (18)	File Format F/V (19)	Block Length (20-23)	Record Length (24-27)	Device (40-46)	Number of Extents (68-69)
02	F	INPUT	I	P			F	96	96	MFCU1	
03	F	OUTPUT	O				F	24Ø	6Ø	DISK	Ø2
04	F										
05	F										
06	F										
07	F										
	F										
	F										

Columns 68-69 are the only columns which tell the system you are processing multi-volume files. In this figure, an output file is being created on two disks.

Figure 6-4. File Description Specifications for Multi-Volume Files

CODING PARAMETERS ON THE FILE STATEMENT TO PROCESS MULTI-VOLUME FILES

The only difference between coding the FILE statement to process a single volume file and a multi-volume file is that you must define and code additional parameters for these keywords: PACK, UNIT, TRACKS, RECORDS, and LOCATION.

These additional parameters are necessary for two reasons:

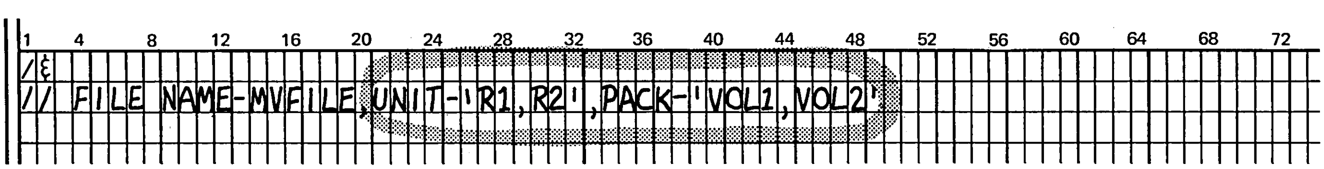
1. When files contained on more than a single volume are processed, the system must be supplied with additional information about each additional volume in order to perform all the protection and checking functions it performs.

2. Additional information is needed to determine and check the sequence in which the volumes are processed and the way they are to be mounted on the disk drives.

You should already be familiar with the format of keywords and parameters on the FILE statement for single volume files. For multi-volume files, you must code the keywords that require additional parameters as follows:

KEYWORD-'data list'

A data list is a list of parameters that must be enclosed by single quotes. Each item in the list must be separated from the next by commas, for example '50, 100, 500'. Figure 6-5 shows an example of data lists in parameters.



1
4
8
12
16
20
24
28
32
36
40
44
48
52
56
60
64
68
72

```
/&
// FILE NAME-MVFILE,UNIT-'R1,R2',PACK-'VOL1,VOL2'
```

Figure 6-5. Data Lists on the FILE Statement

Parameters for the PACK Keyword

The names of the disks that contain, or will contain, the multi-volume file must follow the keyword PACK. The PACK names must be unique. Figure 6-6 shows an example of the PACK parameter for a 3-volume multi-volume file. The volumes are named VOL1, VOL2, and VOL3.

When a multi-volume file is created, disk system management writes sequence numbers on the volumes to indicate the order in which the volumes are created. They are numbered in the order that you list their names in the PACK parameter. They are numbered in consecutive, ascending order (01, 02, and so on).

When a multi-volume file is processed consecutively, disk system management provides two checks to ensure that the disks are processed in the proper order:

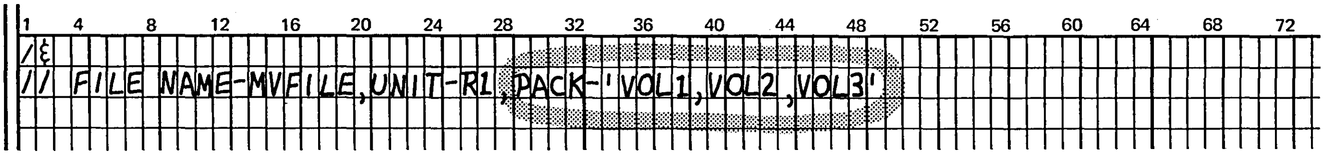
1. It checks to ensure that the disks are used in the order that their names are listed in the PACK parameter.
2. It checks the sequence numbers of the disks used to ensure that they are in ascending order (01, 03, 07 and so on).

If you are reloading a multi-volume file, the PACKS must be in consecutive, ascending order.

If the file was not created as a multi-volume file, the sequence number is ignored, since no sequence number was written at creation of the file.

Disk system management stops when it detects a disk that is out of sequence. The operator can do one of the following three things if the system stops:

- Mount the proper disk and restart the system.
- Restart the system and process the disk that is mounted (if the PACK sequence number is greater than the last one processed).
- End the program.


1
4
8
12
16
20
24
28
32
36
40
44
48
52
56
60
64
68
72

```
/ξ
// FILE NAME-MVFILE,UNIT-R1,PACK-'VOL1,VOL2,VOL3'
```

Figure 6-6. Sample PACK Parameter for Multi-Volume Files

Parameters for the UNIT Keyword

The keyword UNIT must be followed by codes indicating where on the disk drive the disks containing the file are located.

The codes are as follows:

Code	*Meaning*
R1	Removable disk on drive one
F1	Fixed disk on drive one
R2	Removable disk on drive two
F2	Fixed disk on drive two

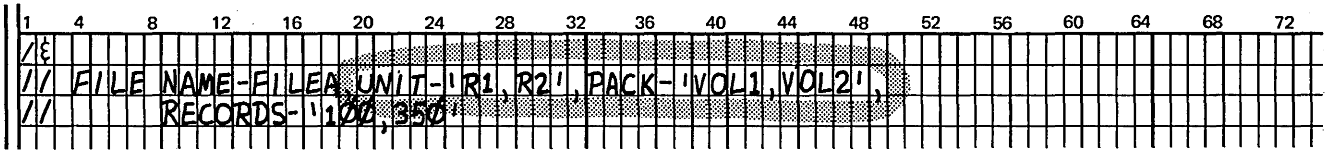
The order of codes in the UNIT parameter must correspond to the order of names in the PACK parameter. For example, assume that a direct file is being created on two disks named VOL1 and VOL2. Further assume that VOL1 is a removable disk to be used on drive one, and VOL2 is a removable disk to be used on drive two. Figure 6-7 shows the PACK and UNIT parameters for this file.

When you are creating an offline sequential file or processing an offline sequential file consecutively, you can use the same drive for more than one of the disks. The disks must be removable, however. If you do use the same drive, do not repeat the code for the drive in the UNIT parameter. When the number of codes in the UNIT parameter is less than the number of names in the PACK parameter, disk system management uses the codes alternately.

For example, assume that your program processes a file consecutively. The disks containing the file are named VOL1, VOL2, and VOL3, respectively. You intend to mount VOL1 and VOL3 on drive one, and VOL2 on drive two.

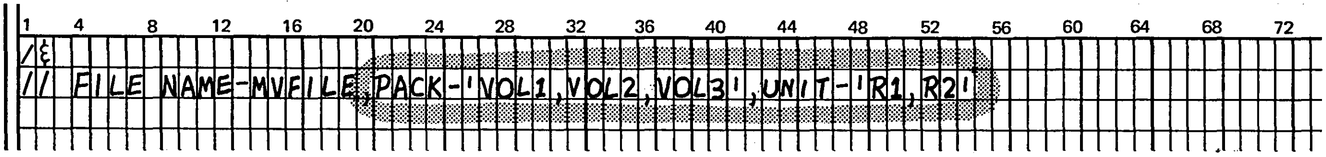
Figure 6-8 shows the PACK and UNIT parameters for the file. Disk system management uses R1, then R2, as specified for VOL1 and VOL2, and then goes back to R1 for VOL3. If, in the preceding example, all three disks were used on drive one, the UNIT parameter in Figure 6-9 would have been used. Consecutive files that are created separately as single volume files can be processed as a multi-volume file, but they must all have the same name.

If any fixed unit (F1 or F2) is specified, the number of PACK parameters must be equal to the number of UNIT parameters. The file must be an online, multi-volume file.



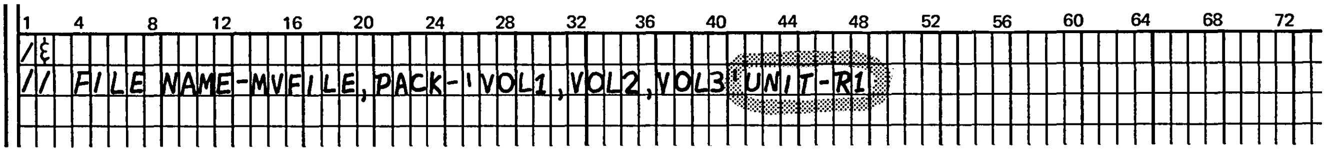
```
/&
// FILE NAME-FILEA,UNIT-'R1,R2',PACK-'VOL1,VOL2',
//      RECORDS-'1ØØ,35Ø'
```

Figure 6-7. Sample UNIT Parameter: Different Unit for Each Disk



```
/&
// FILE NAME-MVFILE,PACK-'VOL1,VOL2,VOL3',UNIT-'R1,R2'
```

Figure 6-8. Sample UNIT Parameter: Same Unit for Two Disks



```
/&
// FILE NAME-MVFILE,PACK-'VOL1,VOL2,VOL3'UNIT-R1
```

Figure 6-9. Sample UNIT Parameter: Same Unit for All Disks

Parameters for the TRACKS or RECORDS Keyword

The keyword TRACKS or RECORDS must be followed by numbers that indicate the amount of space needed on each of the disks containing the multi-volume file. The order of these numbers must correspond to the order of the names in the PACK parameter.

For example, assume that your program is creating a sequential file on three disks: VOL1, VOL2, and VOL3. The first 50 records are to be placed on VOL1, the next 500 on VOL2, and the last 200 on VOL3.

The PACK and RECORDS parameters for the file are shown in Figure 6-10.

Parameters for the LOCATION Keyword

The keyword LOCATION must be followed by the track numbers indicating where the file begins on each disk you use for the file. The order of the numbers must correspond to the order of the names in the PACK parameter.

For example, assume your program is creating a direct file on three disks: VOL1, VOL2 and VOL3. The track locations of your file on each disk are: track 198 in VOL1, track 10 in VOL2, and track 8 in VOL3.

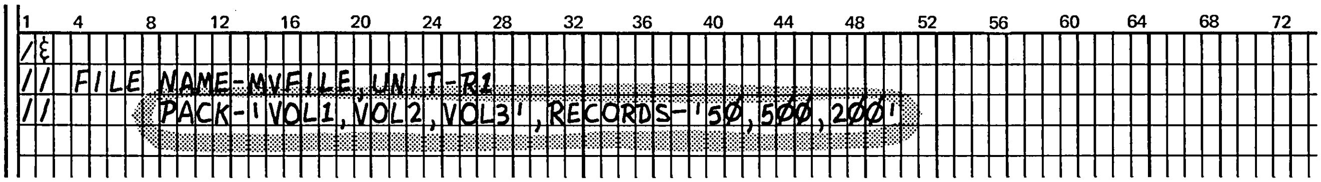
The PACK and LOCATION parameters for the file are shown in Figure 6-11. If you omit the LOCATION parameter, disk system management chooses the beginning track on each of the disks. You must either specify the starting location on all disks of a multi-volume file or on none of the disks.

If you do not give a location parameter, none of the file volumes can contain any type of file. If you do give a location parameter, make sure there are no files on that volume in that location.

Parameters for the RETAIN Keyword

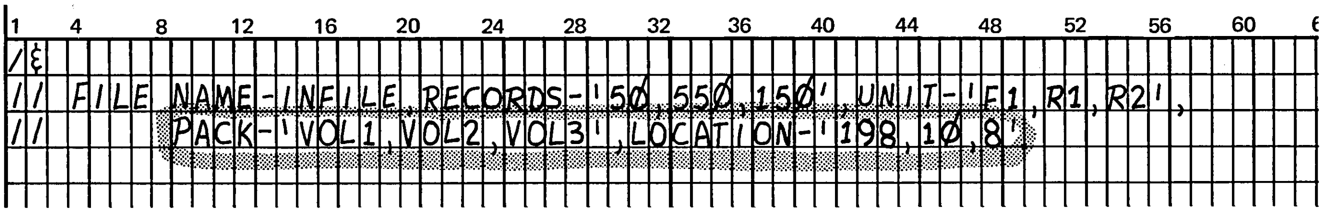
You can specify a multi-volume file as a scratch file (RETAIN-S) only if it is created on line. If RETAIN-S is used to create a multi-volume file on line, you can change it to a temporary file (RETAIN-T) only if this is also done on line.

An offline, multi-volume file defined as a scratch file cannot be processed. If, however, you change it to a temporary file, you can then process it as an offline file.



```
1  4   8   12  16  20  24  28  32  36  40  44  48  52  56  60  64  68  72
/&
// FILE NAME-MYFILE,UNIT-R1
//      PACK-'VOL1,VOL2,VOL3',RECORDS-'5Ø,5ØØ,2ØØ'
```

Figure 6-10. Sample RECORDS Parameter for Multi-Volume Files



```
1  4   8   12  16  20  24  28  32  36  40  44  48  52  56  60
/&
// FILE NAME-INFILE,RECORDS-'5Ø,55Ø,15Ø',UNIT-'F1,R1,R2',
//       PACK-'VOL1,VOL2,VOL3',LOCATION-'198,1Ø,8'
```

Figure 6-11. Sample LOCATION Parameter for Multi-Volume Files

EXAMPLE: COMPARATIVE SALES ANALYSIS

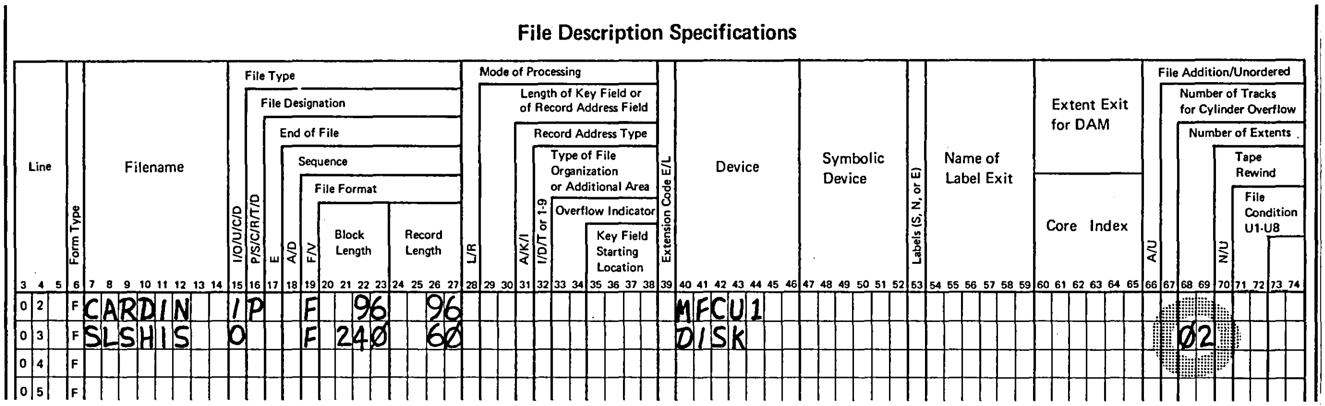
Assume that you are preparing a comparative sales analysis report for your company to analyze the sales made to each customer. You want to compare the amount of sales by product made to a customer each quarter of one year to sales made to the same customer each quarter of the previous year. This analysis will provide the sales department with information about problem areas for future sales efforts. This type of comparative sales analysis involves a great deal of historical data because data about sales for two years must be processed. Your job is to write an RPG II program to create and process a multi-volume file containing the historical sales data.

First, you must determine the number of volumes that will be required to store the data and the type of processing you desire. For this example, assume that the historical data file (SLSHIS) can be loaded onto two volumes from card files (CARDIN). Since all the data will be processed, the file organization is sequential. Remember, the only additional entry that is required to tell the system that you are creating a multi-volume file is an entry in columns 68-69 on the File Description Sheet. Figure 6-12 shows an example of this coding. Assume you have a one-drive system. Since you might expand the historical data file in the future, you decide to use removable disks. Remember, with a one drive system, it is possible to mount two volumes for loading at the same time by using both the fixed and removable disk. However, you can process only two volumes for the file. (In this case, for expansion purposes, it is better to use only removable disks.) Therefore, you would mount one volume, wait until it is filled, and then mount the next volume.

Before the volumes can be loaded, however, your job stream must indicate to disk system management what you want to do. The only difference in OCL statements between processing single-volume and multi-volume files is that keywords on the FILE statement require data list parameters. Assume that 2500 records will be placed on each disk beginning on track 55 of the first volume and track 10 of the second volume. Figure 6-13 shows an example of the FILE statement for this program.

After your file is created, you must process it. In this case, the disk output file which was created now becomes the input file. Once again the only entry in your RPG II program which tells the system you are processing a multi-volume file is the entry in columns 68-69 on the File Description Sheet (Figure 6-14).

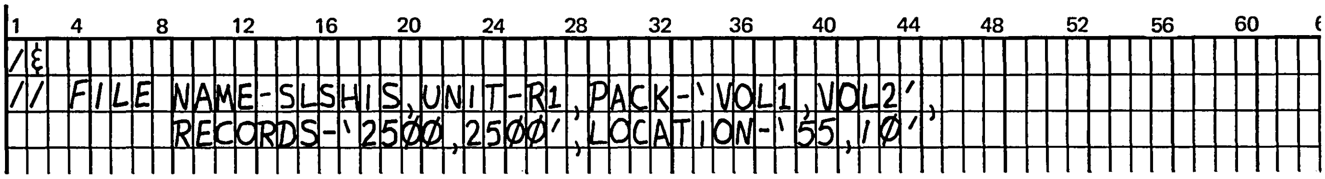
The volumes will be mounted for processing in the same manner they were mounted for creation. The FILE statement is very similar except that you do not code the LOCATION, TRACKS, or RECORDS parameters, because they are required only for loading a file. Figure 6-15 shows an example of the coding for defining SLSHIS on the File Description Sheet.



File Description Specifications

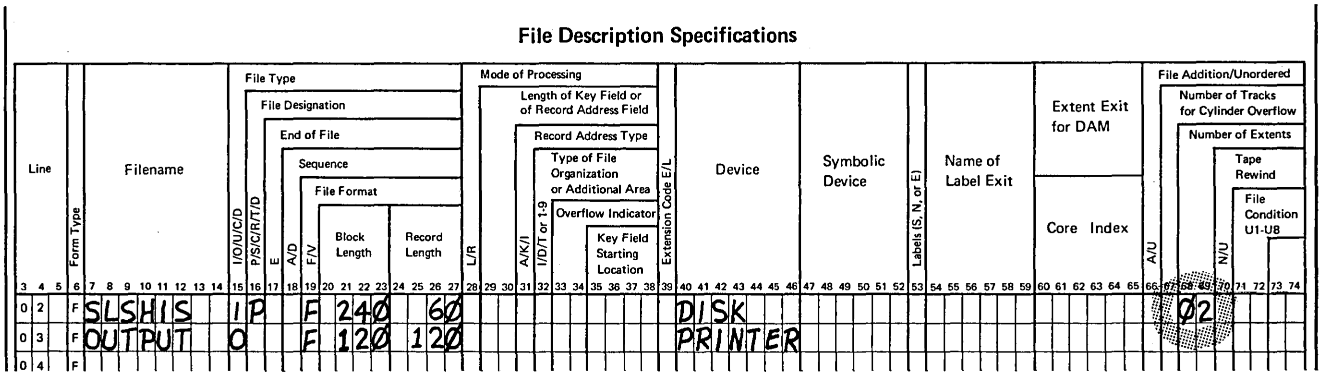
Line	Form Type	Filename	File Type	File Designation	End of File	Sequence	File Format	Block Length	Record Length	L/R	Length of Key Field or of Record Address Field	Record Address Type	Type of File Organization or Additional Area	Overflow Indicator	Key Field Starting Location	Extension Code E/L	Device	Symbolic Device	Labels (S, N, or E)	Name of Label Exit	Extent Exit for DAM / Core Index	A/U	Number of Tracks for Cylinder Overflow	Number of Extents	Tape Rewind	File Condition U1-U8
02	F	CARDIN	I	P			F	96	96								MFCU1									
03	F	SLSHIS	O				F	24Ø	6Ø								DISK							Ø2		
04	F																									
05	F																									

Figure 6-12. File Description Sheet for Loading SLSHIS


1
4
8
12
16
20
24
28
32
36
40
44
48
52
56
60

```
/*
// FILE NAME-SLSHIS,UNIT-R1,PACK-'VOL1,VOL2',
            RECORDS-'25ØØ,25ØØ',LOCATION-'55,1Ø'
```

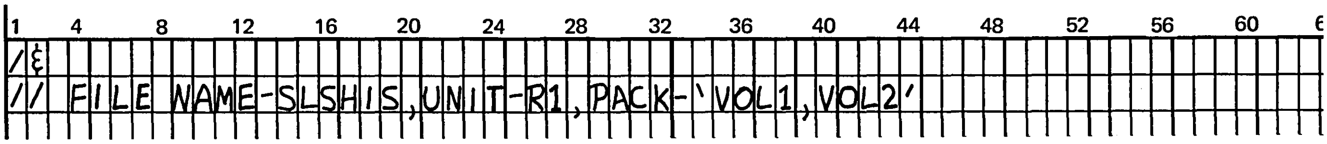
Figure 6-13. FILE Statement for Loading SLSHIS



File Description Specifications

Line	Form Type	Filename	File Type	File Designation	End of File	Sequence	File Format	Block Length	Record Length	L/R	Length of Key Field or of Record Address Field	Record Address Type	Type of File Organization or Additional Area	Overflow Indicator	Key Field Starting Location	Extension Code E/L	Device	Symbolic Device	Labels (S, N, or E)	Name of Label Exit	Extent Exit for DAM / Core Index	A/U	Number of Tracks for Cylinder Overflow	Number of Extents	Tape Rewind	File Condition U1-U8
02	F	SLSHIS	I	P			F	24Ø	6Ø								DISK							Ø2		
03	F	OUTPUT	O				F	12Ø	12Ø								PRINTER									
04	F																									

Figure 6-14. File Description Sheet for Processing SLSHIS


1
4
8
12
16
20
24
28
32
36
40
44
48
52
56
60

```
/*
// FILE NAME-SLSHIS,UNIT-R1,PACK-'VOL1,VOL2'
```

Figure 6-15. FILE Statement for Processing SLSHIS

1. What distinguishes a multi-volume file from other files?

2. What types of file organizations can be specified for multi-volume files?

3. You are going to create a file named SALES on three offline multi-volumes. Assume the following specifications:

   - The volumes are named SALES1, SALES2, and SALES3.
   - The first volume should begin on track 30 and each of the others on track 5.
   - The first volume will have 5000 records and each of the others 7000 records.
   - This is a two-drive system and the volumes should be mounted on alternate drives as the file is loaded.

   a. Write a FILE statement for creating this file.

   b. Complete the File Description Sheet for describing the file in an RPG II program.

File Description Specifications

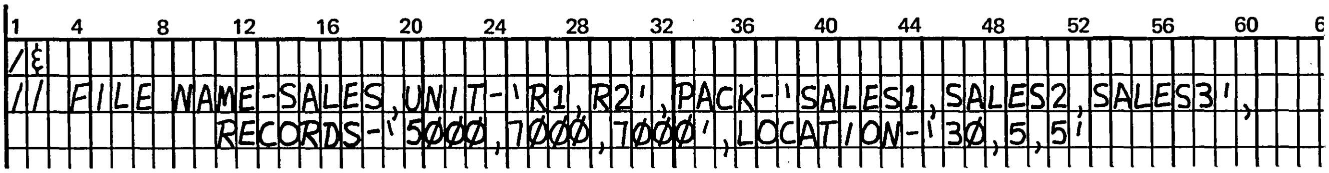
Line 3 4 5	Form Type 6	Filename 7–14	File Type I/O/U/C/D 15	File Designation P/S/C/R/T/D 16	End of File E 17	Sequence A/D 18	File Format F/V 19	Block Length 20–23	Record Length 24–27	Mode of Processing L/R 28	Length of Key Field or of Record Address Field 29–30	Record Address Type A/K/I 31	Type of File Organization or Additional Area I/D/T or 1-9 32	Overflow Indicator 33–34	Key Field Starting Location 35–38	Extension Code E/L 39	Device 40–46	Symbolic Device 47–52	Labels (S, N, or E) 53	Name of Label Exit 54–59	Extent Exit for DAM Core Index 60–65	File Addition/Unordered A/U 66	Number of Tracks for Cylinder Overflow 67	Number of Extents 68 69	N/U 70	Tape Rewind 71 72	File Condition U1-U8 73 74
0 2	F	SALES	O				F	15Ø	15Ø																		
0 3	F																										

Answers to Review 6

1. A multi-volume file is contained on more than one disk.

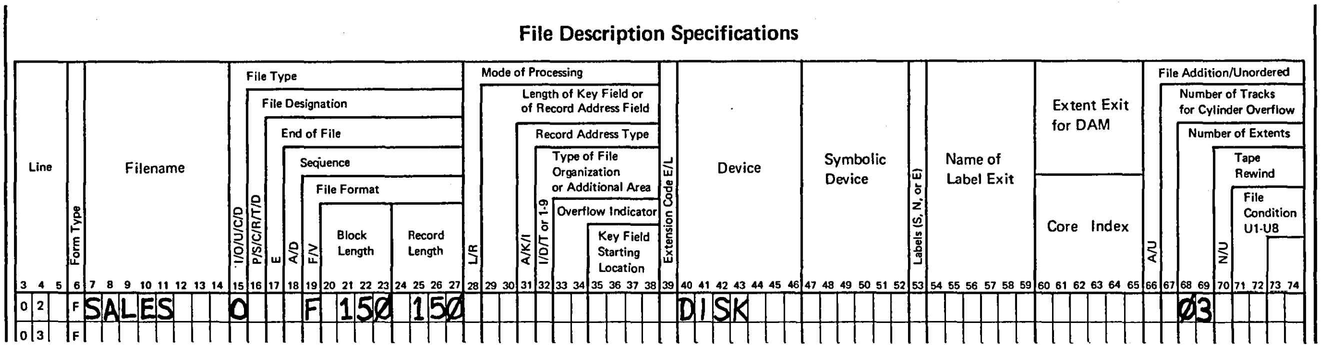
2. Sequential and Direct.

3. a.



```
1  4   8   12  16  20  24  28  32  36  40  44  48  52  56  60  6
/&
// FILE NAME-SALES,UNIT-'R1,R2',PACK-'SALES1,SALES2,SALES3',
               RECORDS-'5ØØØ,7ØØØ,7ØØØ',LOCATION-'3Ø,5,5'
```

b.



File Description Specifications

Line (3-5)	Form Type (6)	Filename (7-14)	File Type (15)	File Designation (16)	End of File (17)	Sequence (18)	File Format (19)	Block Length (20-23)	Record Length (24-27)	L/R (28)	Mode of Processing (29-30)	Record Address Type (31)	Type of File Organization or Additional Area (32)	Overflow Indicator (33-34)	Key Field Starting Location (35-38)	Extension Code E/L (39)	Device (40-46)	Symbolic Device (47-52)	Labels (S, N, or E) (53)	Name of Label Exit (54-59)	Extent Exit for DAM / Core Index (60-65)	A/U (66)	Number of Tracks for Cylinder Overflow (67)	Number of Extents (68-69)	Tape Rewind (70)	File Condition U1-U8 (71-72)	(73-74)
02	F	SALES	O				F	15Ø	15Ø								DISK							Ø3			
03	F																										

CHAPTER

AUTOMATIC FILE ALLOCATION

7

CHAPTER 7 DESCRIBES:

Automatic file allocation.

BEFORE READING THIS CHAPTER YOU SHOULD BE ABLE TO:

Describe the use of the FILE statement.

Determine the size of a file.

Define permanent, temporary, and scratch files.

Define removable and fixed disks.

AFTER READING THIS CHAPTER YOU SHOULD BE ABLE TO:

Define automatic file allocation.

Describe how disk system management performs automatic file allocation.

State the advantages and restrictions of using automatic file allocation.

Code the FILE statement to use automatic file allocation.

INTRODUCTION

You can allocate disk space for a file by determining the size of the file and the location of an available number of tracks that can contain that file. (If you have planned the location of your files, you know where files are located and the tracks that are available for further allocation. The Disk File Layout Chart, GX21-9108, is available to document your file locations.) After you have determined where to place your file, you can code the LOCATION parameter of the FILE statement to tell disk system management on which track the file is to begin. Figure 7-1, part A, is a sample FILE statement containing a LOCATION parameter to tell disk system management that FILEA is to be located on disk pack VOL1 beginning on track 10.

If, as in Figure 7-1, part B, no LOCATION parameter is coded, FILEA is located on the disk pack automatically for you. The process used by disk system management to allocate file space for you is known as *automatic file allocation.*

ALLOCATING FILE SPACE AUTOMATICALLY

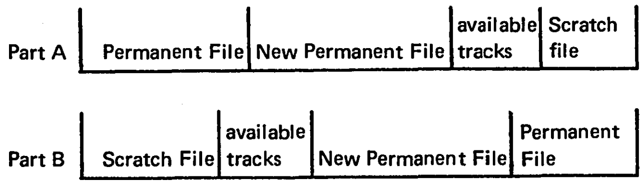
When allocating file space, disk system management calculates the length of the file and checks the volume label to determine which tracks are available for allocation. (The volume label contains the status of each track and indicates which tracks are available for allocation.) Disk system management then:

1. Finds a continuous string of available tracks.
2. Allocates space for permanent files, then temporary files, and finally scratch files, if multiple files are being allocated.

Disk system management places your file on a continuous string of available tracks that is as near to the length of your file as possible. For example, it can determine that your file is 10 tracks long and find one string of 12 available tracks and another of 15 tracks. It places your file in the string of 12 tracks because the 12-track string is closer to the length of the file.

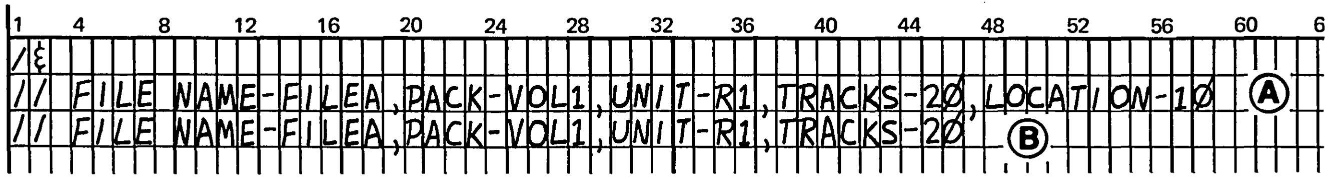
If an area is found containing the number of available tracks and two files are already on either side of the area, the new file will be placed adjacent to a file with similar attributes, if possible. For example, permanent files are placed adjacent to permanent files. Figure 7-2, parts A and B, shows a permanent file being placed adjacent to another permanent file. Files are placed adjacent to files with similar attributes, so there will be as few unused tracks between files as possible. It is more important, however, to locate a new file on a string of tracks as close to the length of your file as possible. Therefore, a permanent file could be allocated space next to a temporary or scratch file, if the number of tracks at that location can contain the permanent file.

If your file is the first file placed on a disk, the system allocates space for the file beginning at the highest numbered track. After a disk contains files and two areas are available for a new file, the file is placed beginning at the highest numbered available location. This is done to allow you as many available tracks as possible next to the object library which is located at the lowest numbered tracks, so the object library can expand, if necessary.



Disk system management determines the type of file to the left of the available tracks. If the file to the left is similar, the new file is left-adjusted (Part A). If the file to the left is not similar, it is right-adjusted (Part B).

Figure 7-2. File Placement of Automatic File Allocation



```
// FILE NAME-FILEA,PACK-VOL1,UNIT-R1,TRACKS-2Ø,LOCATION-1Ø
// FILE NAME-FILEA,PACK-VOL1,UNIT-R1,TRACKS-2Ø
```

1
4
8
12
16
20
24
28
32
36
40
44
48
52
56
60
6
A
B

Figure 7-1. FILE Statement and Use of the LOCATION Parameter

CONSIDERATIONS FOR USING AUTOMATIC FILE ALLOCATION

If you let disk system management allocate file space, you do not have to determine where to locate files. It is easier, but there are some considerations in determining whether to use automatic file allocation. After you have gained experience, you should be able to locate a file more efficiently than disk system management. Disk system management may leave a string of available tracks between files which is unusable, because it is not long enough to locate another file.

If you plan your own files, you can determine where files are located by checking the Disk File Layout Chart, if you keep your layout chart up-to-date. If you automatically allocate some files and then want to locate a file yourself, you must check the volume label to determine what tracks are available. This is done by using the File and Volume Label Display utility program. (See the *IBM System/3 Disk System Operation Control Language and Disk Utilities Reference Manual,* GC21-7512 for more information on this utility program.)

Automatic file allocation can increase the time needed to copy programs using the Copy Dump utility program. (See the *IBM System/3 Disk System Operation Control Language and Disk Utilities Reference Manual,* GC21-7512.) For example, you have used automatic file allocation and now wish to copy a file onto tracks 30 through 50 of the disk on F1. However, disk system management placed the file to be copied on tracks 50 through 70 of the disk on R1. Copying time increases when a file is copied from one location on a disk to another location on another disk, because the access mechanism must move. It would be more advantageous to allocate the file space on tracks 30 through 50 of R1 yourself so that the file can be copied onto the same tracks (tracks 30 through 50) of F1.

Automatic file allocation considers effective use of file space, but not the usage of the files. It does not consider file planning for multiple input files in a program or job-to-job transitions. If you plan your own file locations, you can place files that are used in conjunction close together on disk. When files used together are located near one another, processing time may be improved.

1. What does automatic file allocation mean?

2. What advantages are there to using automatic file allocation? Disadvantages?

3. How is automatic file allocation indicated on the FILE statement?

4. Consider the following diagram of a disk and its allocated files.

0 100

Object Library	Blank 15 tracks	Temporary File 5 tracks	Scratch File 10 tracks	Temporary File 12 tracks	Scratch File 15 tracks	Permanent File 10 tracks	Scratch File 15 tracks	Permanent File →

Where would the disk system allocate the following files?

a. A temporary file requiring 12 tracks.

b. A temporary file requiring 7 tracks.

c. A permanent file requiring 10 tracks.

d. A permanent file requiring 5 tracks.

e. A permanent file requiring 12 tracks.

Answers to Review 7

1. With automatic file allocation, the programmer is not concerned with specifying the placement of data files. The disk system automatically finds a space for each file as it is loaded.

2. Automatic file allocation is easier to use, although it can cause some wasted space on the disk. Performance time may be slower with automatic allocation when data files are not aligned between the fixed and removable disk.

3. The location parameter is omitted.

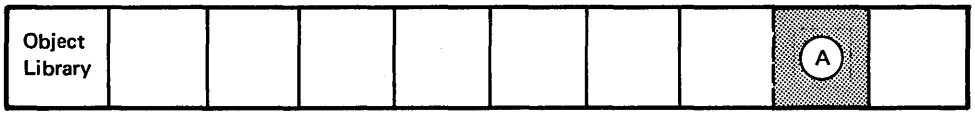
4. You should have used the following logic in determining where to place the data files:

   a. The available area closest to the required number of tracks is determined.

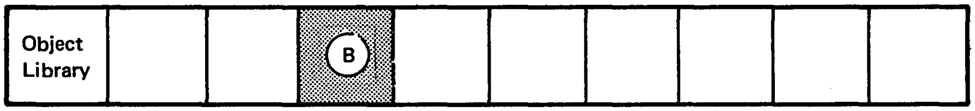
   b. If the file on the left of the available area has the same attribute as the file to be allocated, the file is left-adjusted.

   c. If not, the file is right-adjusted.

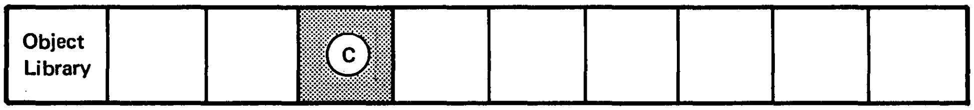
A



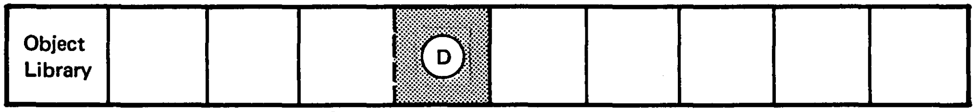
B



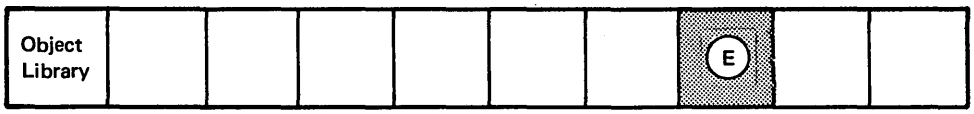
C



D



E



CHAPTER

STORING PROGRAMS AND PROCEDURES ON DISK

8

CHAPTER 8 DESCRIBES:

Source programs, object programs, and Operation Control Language (OCL), utility program control statements.

Advantages and considerations for storing programs and procedures on disk.

Object libraries and source libraries.

The Library Maintenance programs.

RPG II object output.

The CALL statement.

The COMPILE statement.

BEFORE READING THIS CHAPTER YOU SHOULD BE ABLE TO:

Describe basic disk concepts such as sector and disk system management.

Differentiate between fixed and removable disks.

Define source programs, object programs, and OCL.

Define permanent and temporary programs.

Describe how programs and OCL statements are loaded into storage from cards.

AFTER READING THIS CHAPTER YOU SHOULD BE ABLE TO:

Define procedures.

List the advantages and considerations for storing programs and procedures on disk.

Describe object libraries and source libraries.

List the functions of the Library Maintenance program.

Code the OCL and program control statements to build object and source libraries and store programs and procedures.

Describe the CALL statement.

Code the RPG II Control Card Specification Sheet for object output.

Code the COMPILE Statement.

INTRODUCTION

In the System/3 Disk System, programs and OCL statements can be stored on disk and transferred as needed into main storage.

The area in which programs are stored on disk is called a library. Two types of libraries can be located on a disk: *object libraries* and *source libraries.* Object libraries contain object programs; source libraries contain source programs, OCL statements, and utility program control statements.

When OCL statements and utility program control statements are stored in a source library, they are call *procedures.*

The System/3 Library Maintenance program can be used to:

- Allocate space for libraries.
- Enter programs and procedures into libraries.
- Maintain libraries.

More information about this program and its functions is given later in this chapter under *The Library Maintenance Program.*

ADVANTAGES OF STORING PROGRAMS AND PROCEDURES ON DISK

By storing frequently used programs and procedures on disk, you can increase the efficiency of your system operation and reduce the amount of time required to process jobs. When you need a particular program or procedure, it can be loaded from disk, reducing card handling. When programs and procedures are located on disk, firm operating procedures can be established.

Increasing System Efficiency

All programs and procedures can be placed on a master pack and copied to the fixed disk for execution. For example, you can load an entire series of application programs and procedures on a fixed disk with a minimum number of control statements. Assume you run payroll every Friday morning. On Friday, you can use a pretested procedure to transfer all the required programs and their procedures from the master pack to a fixed disk, then run payroll.

There are two library functions that make this method particularly efficient: naming conventions and object library expansion.

Naming Conventions: You can transfer all the correct programs and procedures from the master pack to the fixed disk using one Library Maintenance control statement, if you establish and use a naming convention. The names of all programs and procedures used in an application series should begin with the same letters. For example, name all payroll programs and their corresponding procedures beginning with the letters PAY. Then, with one COPY control statement, all payroll programs and procedures in both libraries will be copied onto the fixed disk. (The COPY control statement is described under *The Library Maintenance Program.*)

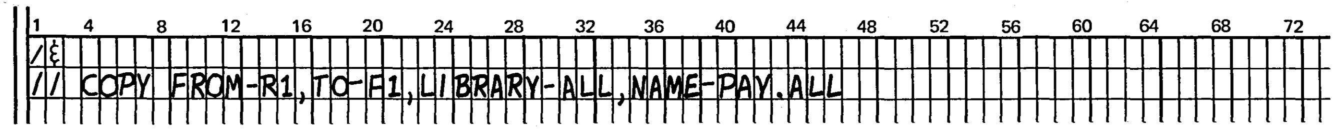
Object Library Expansion: Object libraries are capable of expanding their size for temporary entries. When you copy an object program onto the fixed disk, you can designate it as a temporary entry. Then if you add a permanent entry, reallocate the library, or delete all temporary entries, the object library will return to its normal size. Consequently, by using this function, you use a minimum amount of storage on the fixed disk, leaving it free to perform other functions when you are not using the object library.

Decreasing Processing Time

Disk system management takes more time to read programs from caras into main storage than to read programs from disk into main storage. Once your programs and procedures are located on disk, programs can be transferred quickly into main storage, thereby decreasing the amount of time to run your jobs. Operating time is also saved because the operator does not handle card decks.

Storing Programs and Their Data Files on Removable Disks

If space on the fixed disk is limited, or if you prefer to do so, programs and the data files they process can be stored on a removable disk. By placing programs and data files on the same disk, you can reduce the number of times disk packs must be changed. This is especially true if a program uses only one data file. This also allows more available space on the fixed disk, a more flexible arrangement of space for output files on the fixed disk, and placement of files to minimize access time.


1 4 8 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72

```
/&
// COPY FROM-R1,TO-F1,LIBRARY-ALL,NAME-PAY.ALL
```

There are certain things you must consider when placing both programs and data files on a removable disk, however. First, more space is required on the removable disk.

Maintaining programs on a removable disk is more difficult, because they are scattered across several disks instead of all located on a master pack. For example, if the format of an inventory record changed, you must search several packs to update all the programs using that record, rather than searching just one master pack. You should have a master pack so that you have copies of your programs if something happens to one of the other disks.

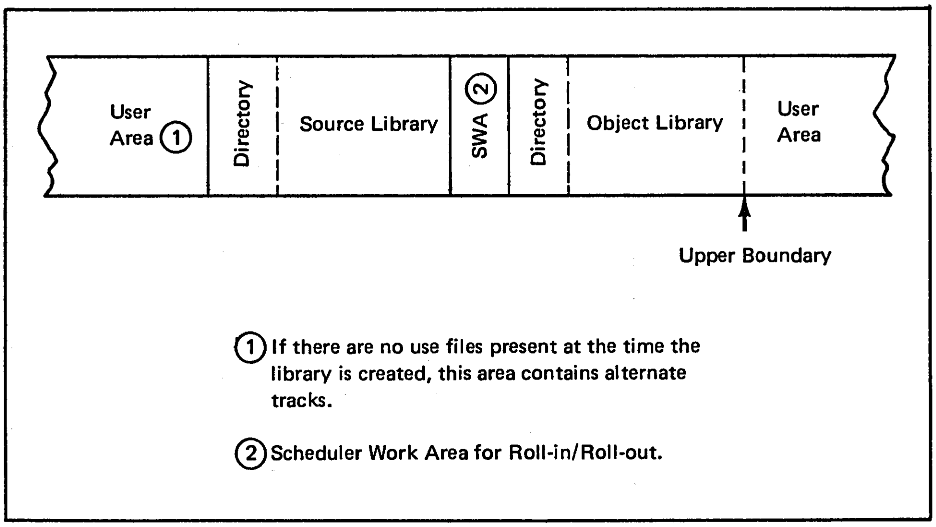
You should not place data and programs on the same packs if you are processing multi-volume files. The pack containing the program cannot be removed during the job.

LOCATION OF LIBRARIES ON DISK

You can place a source library, an object library, or both on a disk. If space is allocated for only a source or object library, the Library Maintenance program places the library in the first available disk area large enough to contain the library.

If you are allocating space for a source library on a disk containing an object library, a disk area large enough for the source library must immediately precede or follow the object library (Figure 8-1). If the disk area follows the object library, the Library Maintenance program moves the object library to allow space for the source library preceding it.

If an object library is being allocated on a disk with a source library, space for the object library must immediately follow the source library.



(1) If there are no use files present at the time the library is created, this area contains alternate tracks.

(2) Scheduler Work Area for Roll-in/Roll-out.

Figure 8-1. Relative Positions of Libraries on Disk

OBJECT LIBRARIES

The object library is an area on disk used to store object programs and routines. Object programs, or executable programs, are programs and subroutines that can be loaded for execution. Routines, or nonexecutable programs, are programs and subroutines that need further translation before being loaded for execution. Nonexecutable programs are used by the compiler and must be on the same disk pack as the compiler. Figure 8-2 is a sample object library.

The object library is one physical area containing two logically different types of entries: object programs and routines. When these entries are copied into the object library, they are given different object library designations. Object programs are given an *O* library designation; routines are given an *R* library designation. Figure 8-3 shows the logical library entries within the object library.

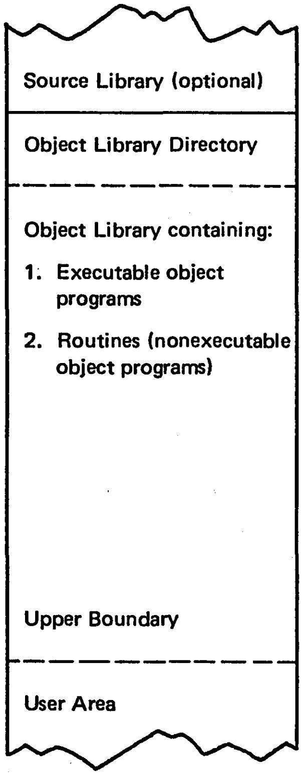
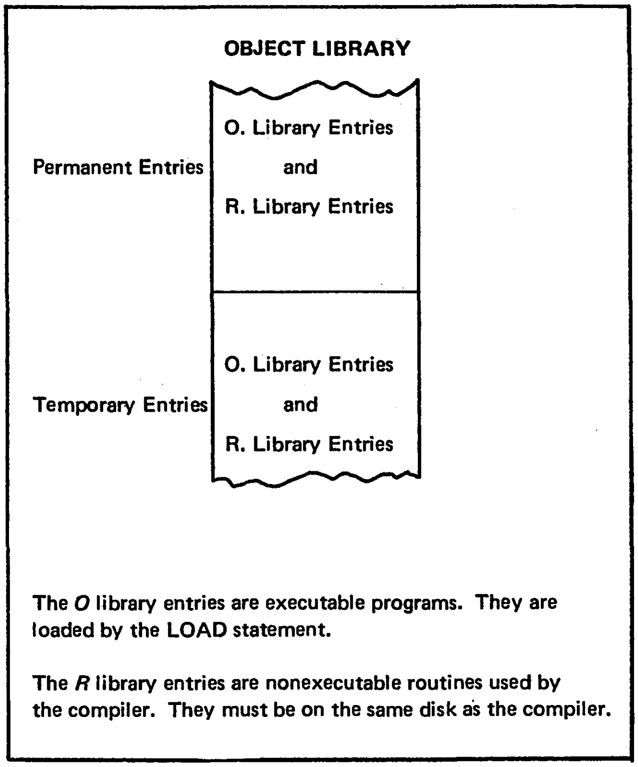

Source Library (optional)
Object Library Directory
Object Library containing:
1. Executable object programs
2. Routines (nonexecutable object programs)
Upper Boundary
User Area

Figure 8-2. Format of the Object Library

Physical Characteristics of the Object Library

Size: The minimum size of a library not on a system pack is three tracks. The minimum size of an object library on a system pack is 30 tracks. (You can build an object library on any disk pack, but you must have one library online containing the systems programs.)

For the object library consisting of system programs, the disk area forming the library must also be large enough to contain a work area for disk system management. The number of tracks for the work area space need not be included in the number of tracks you specify for the library; the Library Maintenance program calculates and assigns the additional space for you. The amount of additional space needed depends on the capacity of your system and whether your programming system has inquiry capability or the dual programming feature. Figure 8-4 is a table showing the work area size required for various system capacities.



The *O* library entries are executable programs. They are loaded by the LOAD statement.

The *R* library entries are nonexecutable routines used by the compiler. They must be on the same disk as the compiler.

Figure 8-3. Logical Parts of an Object Library

Directory: The Library Maintenance program creates a directory for every object library (Figure 8-2). The directory acts as a table of contents for the programs contained within the object library. It contains such information as the name and location of the entries. If the object library is on a system pack, three of the requested tracks are reserved for the directory. If not, only the first track is reserved for the directory.

Upper Boundary: The upper boundary of the object library (Figure 8-2) will automatically expand only if more space is needed for temporary entries and if area next to the library is available. When permanent entries are placed in the library, all the temporary entries are deleted and the object library returns to its normal size.

To make efficient use of this feature, the area next to the upper boundary of the object library should be kept free of data files. When disk system management automatically allocates file space for you, the area next to the object library is probably free because your files are placed as close to the end of the disk pack as possible. When allocating your own file space, you should allocate your files toward the end of the pack, also. This leaves room for object library expansion.

Organization of Entries: Entries are stored in the object library serially; that is, a twenty-sector program occupies 20 consecutive sectors. Temporary entries follow all permanent entries in the object library. The permanent entry is loaded into the first available space large enough to hold it, usually the space following the last permanent entry.

Gaps can occur in the object library when a permanent entry is deleted and replaced with one using fewer sectors. The Library Maintenance program scans the library to locate available sectors, then places the entry into the smallest gap large enough to hold it.

You should use the Library Maintenance program to reorganize the library when you delete permanent entries, when a great number of additions and deletions take place, or when there is no apparent room.

In reorganizing entries, the Library Maintenance program shifts entries so that gaps do not appear between them. This makes more sectors available for use.

Frequent adding, replacing, and deleting of entries causes unusable sectors. You can determine how many sectors are unusable by printing the library directory using the Library Maintenance program.

Scheduler Work Area Size

Capacity	No Inquiry Without DPF[1]	No Inquiry With DPF	Inquiry Without DPF	Inquiry With DPF	ROLL-IN/ ROLL-OUT[2]
12K bytes	2 tracks	4 tracks	6 tracks	8 tracks	4 tracks
16K bytes	2 tracks	4 tracks	7 tracks	9 tracks	5 tracks
24K bytes	2 tracks	4 tracks	8 tracks	10 tracks	7 tracks
32K bytes	2 tracks	4 tracks	9 tracks	11 tracks	9 tracks

[1] Dual Programming Feature.

[2] Tracks needed by the scheduler to retain information concerning an interrupted program.

Figure 8-4. Work Area Size

SOURCE LIBRARIES

Source libraries can contain source program statements and procedures. Examples of source statements are RPG II source programs and sequence specifications for the Disk Sort program.

Procedures are sets of OCL statements. The procedures for utility programs can include program control statements.

Entries in the source library can be comprised of any valid System/3 characters. Figure 8-5 shows the format of the source library.

The source library is one physical area containing two logically different types of entries. When these entries are copied into source libraries, they are given different source library designations. Source programs are given an *S* library designation; procedures are given a *P* library designation. Figure 8-6 shows the logical entries within the source library.

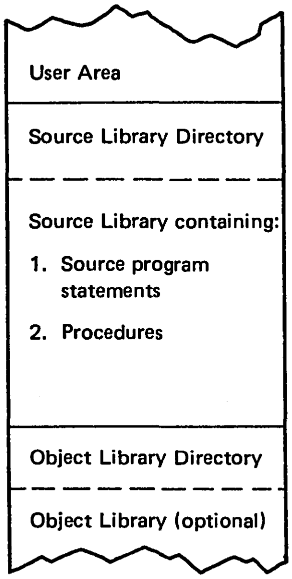

User Area
Source Library Directory
Source Library containing:
1. Source program statements
2. Procedures
Object Library Directory
Object Library (optional)

Figure 8-5. Format of the Source Library

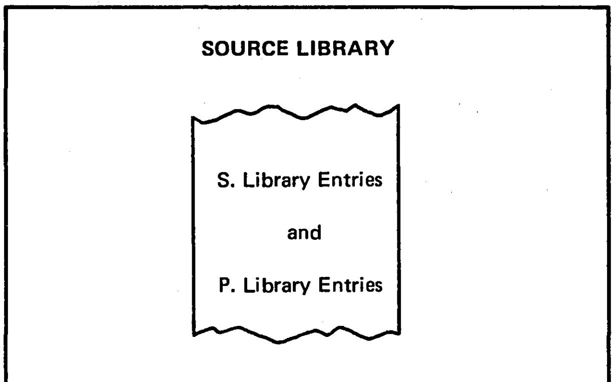
Physical Characteristics of the Source Library

Size: The minimum size of a source library is one track.

Directory: Note the area labeled source library directory in Figure 8-5. The directory acts as a table of contents for each source library entry containing such information as the name and location of each entry. The first two sectors of the first track are always assigned to the directory with additional sectors used as needed.

Organization of Entries: Entries within the source library need not be stored in consecutive sectors. An entry can be stored in widely separated sectors with each sector containing a pointer to the next sector that contains the next part of the entry.

The boundary of the source library cannot be expanded; therefore, an entry must fit within the available library space. The system provides maximum space within the prescribed limits of the source library by compressing entries. That is, all duplicate characters are removed from entries. Later, if the entries are printed or punched, the duplicate characters are reinserted.



The *S* library entries are source programs. Procedures *cannot* be executed from this library.

The *P* library entries are procedures which can be executed.

Figure 8-6. Logical Entries Within the Source Library

STORING PROGRAMS AND PROCEDURES INTO LIBRARIES

There are three methods you can use to store programs into libraries: the Library Maintenance program, a specification on the RPG II Control Card Sheet, or the COMPILE OCL statement.

The Library Maintenance Program

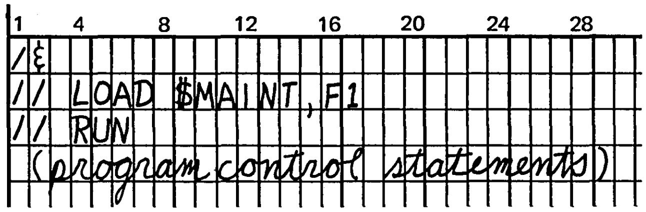
Depending upon your specifications, the Library Maintenance program can:

- Allocate space for a library. It can create, reorganize, change the size of, or delete a library.
- Delete entries from a library.
- Copy entries from one location to another within a library, from one library to another, from the input device to a library, from the library to a printer, or from a library to a punch, and give new names if requested.
- Rename library entries.

In this discussion, only creation of libraries and storing of programs and procedures into libraries from an input device are described. Maintenance functions of the program are mentioned only in general terms. More information about maintenance is in the *IBM System/3 Disk System Reference Manual,* GC21-7512.

OCL Statements and Program Control Statements

The Library Maintenance program ($MAINT) requires the same OCL statements as other utility programs. A sample job stream to load the program from F1 into storage is:

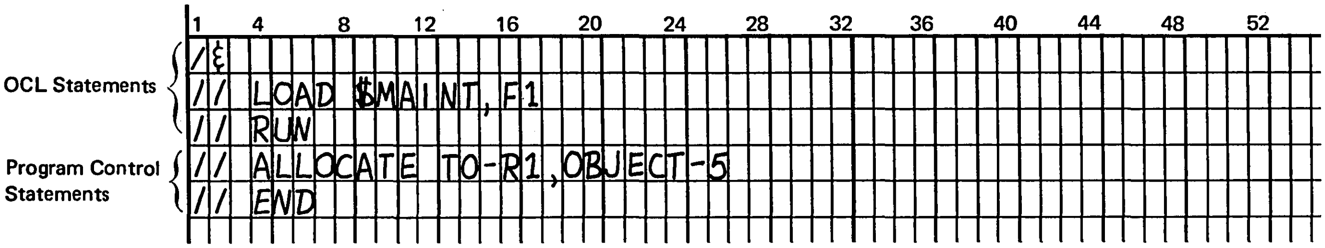


```
1  4   8   12  16  20  24  28
/&
// LOAD $MAINT,F1
// RUN
 (program control statements)
```

Program control statements follow OCL statements in the job stream and provide the program with information concerning its functions. The program control statements and their associated functions are:

- ALLOCATE: assigns or cancels disk space for libraries. Using this statement you can also reorganize or change the size of libraries.
- DELETE: removes entries from a library.
- COPY: copies entries from one location to another within a library (renaming the entries), from one library to another, from the input device to a library, from the library to a printer, or from a library to a punch.
- CEND: follows card decks to be copied from the reader into a library and indicates the end of the input to be copied.
- RENAME: changes the name of a library entry.
- END: follows the program control statement and indicates to disk system management that the job stream for the Library Maintenance program has ended.

A sample job stream loading the program and creating an object library consisting of five tracks on R1 is:



OCL Statements

```
/&
// LOAD $MAINT,F1
// RUN
```

Program Control Statements

```
// ALLOCATE TO-R1,OBJECT-5
// END
```

1
4
8
12
16
20
24
28
32
36
40
44
48
52

Remember that the library program control statements must be terminated by an END statement.

Storing Programs In an Object Library

To store object programs in an object library you must first use the Library Maintenance program to create an object library. You can then copy the program from an input device or another library into the library.

The Library Maintenance program creates object libraries according to the specifications you code on the ALLOCATE statement. Figure 8-7 shows the format of the ALLOCATE statement to create an object library. The keyword parameters for the ALLOCATE statement include TO, OBJECT, SYSTEM, and WORK.

The TO keyword parameter indicates the location of the disk drive on which the library is to be created.

The OBJECT keyword parameter indicates the number of tracks to be used for the library. If an *O* is coded, the library is deleted; if an *R* is coded, the library is reorganized.

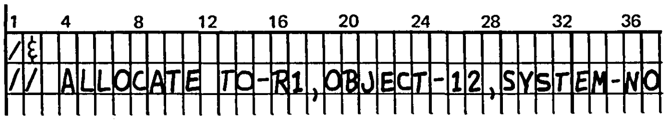
The SYSTEM keyword parameter assigns the number of tracks for the object library directory. If NO is coded, one track is assigned to the directory, and the directory will not be large enough to contain system program entries. If YES is coded, three tracks are assigned to the directory, and the directory will be large enough to contain entries for the system programs. The parameter YES must be assigned if a disk is being created to contain a minimum system.

The WORK keyword parameter indicates the drive on which a second disk containing a disk system management work area is located. A work area is required if you are:

- Reallocating space for an existing library.
- Allocating space to create a source library on a disk that contains only an object library.
- Removing a source library from a disk that also contains an object library.

Library entries are temporarily stored in the work area while the program moves and reorganizes libraries.

Creating an Object Library: Assume you are creating an object library on a disk located on R1 that consists of 12 tracks. You are not storing a minimum system in the library, so only one track is needed for the directory. The ALLOCATE statement looks like this:



```
/&
// ALLOCATE TO-R1,OBJECT-12,SYSTEM-NO
```

1
4
8
12
16
20
24
28
32
36

```
                   ⎧R1⎫                                                 ⎧R1⎫
// ALLOCATE TO-    ⎨F1⎬,OBJECT- ⎧NUMBER⎫ ,SYSTEM- ⎧NO ⎫ ,WORK-  ⎨F1⎬
                   ⎨R2⎬         ⎩  R   ⎭          ⎩YES⎭         ⎨R2⎬
                   ⎩F2⎭                                                 ⎩F2⎭
```

Figure 8-7. Format of the ALLOCATE Statement to Create an Object Library

Since you are not reallocating space for the library, note that the WORK keyword parameter is not required.

Once you have created the library, you can store object programs into it. The Library Maintenance program copies entries into a library according to the specifications you code on the COPY statement (Figure 8-8). The keyword parameters for the COPY statement are FROM, LIBRARY, NAME, TO, RETAIN, and NEWNAME.

The FROM keyword parameter indicates the location of the input file containing the entries to be copied. The input file may be on cards in the reader or on disk in a library.

The LIBRARY keyword parameter indicates the type of entry being stored into a library:

- S – Source statements to be stored in a source library.
- P – OCL procedures to be stored into a source library.
- O – Object programs to be stored into an object library.
- R – Routines to be stored into an object library.
- ALL – All types of entries are to be copied to the corresponding libraries.

The NAME keyword parameter further identifies the entries to be copied into the library. (The NAME, LIBRARY and RETAIN keyword parameters are used together to identify the entries to be copied.) The possible data that can follow NAME are:

- name – Name of the library entry to be copied.
- *characters.*ALL – Only those entries beginning with the indicated characters are to be copied. Up to five characters can be used.
- ALL – All entries (of the type indicated to the LIBRARY parameter) are to be copied.
- DIR – Directory entries for all library entries of the type indicated in the LIBRARY keyword parameter are to be copied. If the LIBRARY keyword parameter is ALL, system directory entries are also printed.
- SYSTEM – Only system programs comprising a minimum system are to be copied.
- $cc.ALL – The IBM program with the name beginning with the indicated characters ($cc) is to be copied. For example, $MA.ALL means the Library Maintenance program ($MAINT) is to be copied.

$$\text{// COPY FROM–}\left\{\begin{array}{l}\text{R1}\\\text{F1}\\\text{R2}\\\text{F2}\\\text{READER}\end{array}\right\}\text{,LIBRARY–}\left\{\begin{array}{l}\text{S}\\\text{P}\\\text{O}\\\text{R}\\\text{ALL}\end{array}\right\}\text{,NAME–}\left\{\begin{array}{l}\text{name}\\\text{characters.ALL}\\\text{ALL}\\\text{DIR}\\\text{SYSTEM}\\\text{\$cc.ALL}\end{array}\right\}\text{,TO–}\left\{\begin{array}{l}\text{R1}\\\text{F1}\\\text{R2}\\\text{F2}\\\text{PRINT}\\\text{PUNCH}\\\text{PRTPCH}\end{array}\right\}$$

$$\text{RETAIN–}\left\{\begin{array}{l}\text{T}\\\text{P}\\\text{R}\end{array}\right\}\text{,NEWNAME–}\left\{\begin{array}{l}\text{name}\\\text{characters.ALL}\end{array}\right\}$$

Figure 8-8. COPY Statement Format

The TO keyword parameter indicates on what device the output file is located. The possible devices are:

- Disk drive – R1, F1, R2, F2.
- PRINT – Entries are to be printed on the system printed.
- PUNCH – Entries are to be punched on cards.
- PRTPCH – Entries are to be both punched and printed.

The RETAIN keyword parameter identifies the status of an entry and can change the status of an existing entry. The possible parameters are:

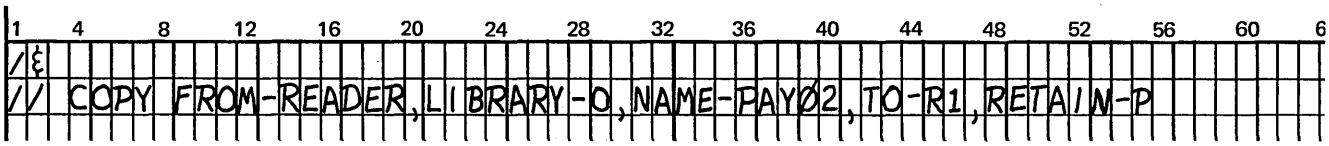
- T – Temporary entry.
- P – Permanent entry.
- R – Replaces an entry. This parameter is used if you are copying an entry into a library on a disk that already has an entry with that name. The new entry is placed in the library and the old entry is deleted. A temporary entry cannot replace a permanent entry.

The NEWNAME keyword parameter indicates the name you want used on the entries being copied on disk. Without this keyword parameter, the program uses the NAME keyword parameter. The NEWNAME-*characters*.ALL parameter indicates you want to use these characters to identify all the entries you are placing on disk instead of the characters specified in the NAME-*characters*.ALL statement.

Storing an Object Program: Assume you want to store an object program in the library created on R1. The object program is labeled PAY02 and is stored on cards. It will be a permanent entry. Figure 8-9 is the COPY statement to load PAY02. The job stream for this program is shown in Figure 8-10.

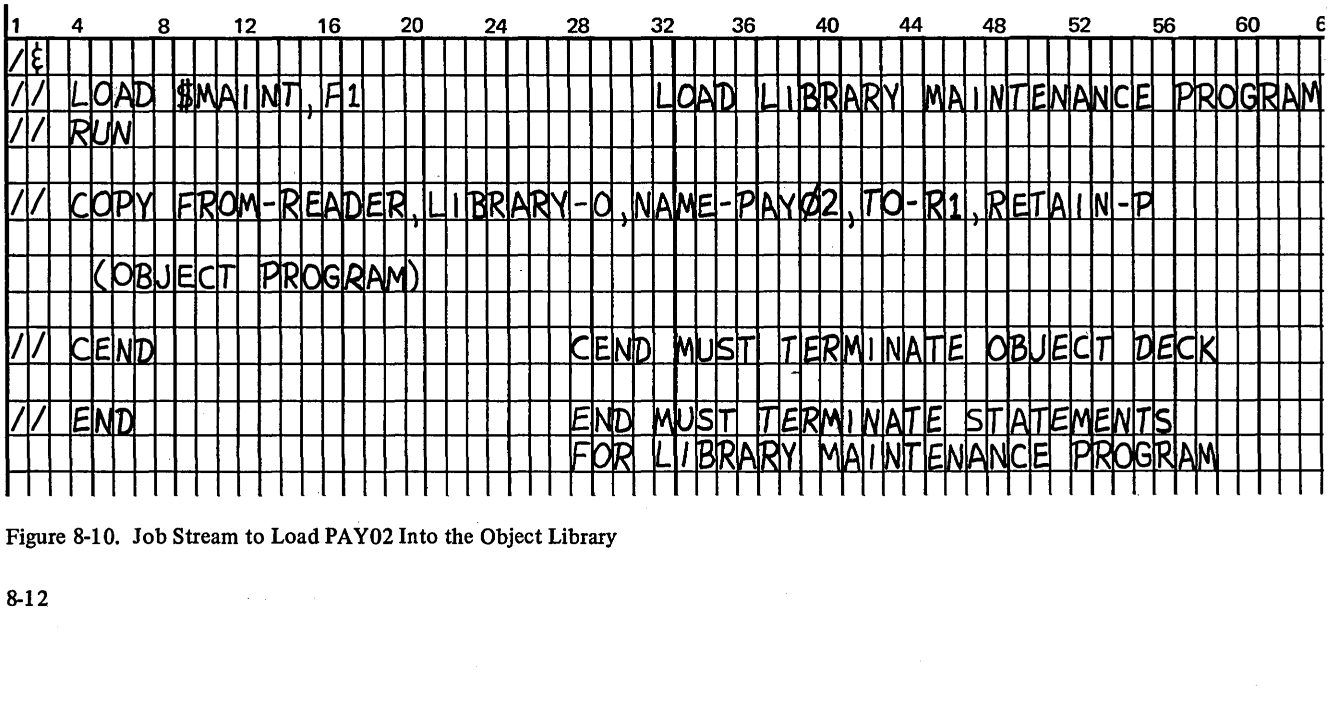
Storing Programs and Procedures in a Source Library

To store programs and procedures in source libraries, you must first use the Library Maintenance program to create a source library, then copy entries into the source library.


1
4
8
12
16
20
24
28
32
36
40
44
48
52
56
60
6

```
/&
// COPY FROM-READER,LIBRARY-O,NAME-PAYØ2,TO-R1,RETAIN-P
```

Figure 8-9. A COPY Statement for Loading PAY02 Into the Object Library


1
4
8
12
16
20
24
28
32
36
40
44
48
52
56
60
6

```
/&
// LOAD $MAINT,F1              LOAD LIBRARY MAINTENANCE PROGRAM
// RUN

// COPY FROM-READER,LIBRARY-O,NAME-PAYØ2,TO-R1,RETAIN-P

    (OBJECT PROGRAM)

// CEND                    CEND MUST TERMINATE OBJECT DECK

// END                     END MUST TERMINATE STATEMENTS
                           FOR LIBRARY MAINTENANCE PROGRAM
```

Figure 8-10. Job Stream to Load PAY02 Into the Object Library

The Library Maintenance program creates a source library according to your specifications on the ALLOCATE statement. The ALLOCATE statement to build a source library looks like this:

$$\text{// ALLOCATE TO-}\left\{\begin{matrix}\text{R1}\\\text{F1}\\\text{R2}\\\text{F2}\end{matrix}\right\}\text{,SOURCE-}\left\{\begin{matrix}\text{NUMBER}\\\text{R}\end{matrix}\right\}\text{,WORK-}\left\{\begin{matrix}\text{R1}\\\text{F1}\\\text{R2}\\\text{F2}\end{matrix}\right\}$$

The TO keyword parameter indicates on which drive the disk containing the library is located.

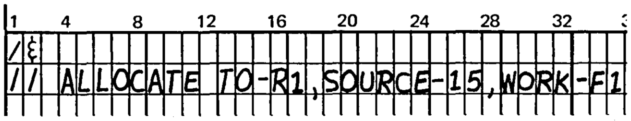
The SOURCE keyword parameter indicates the number of tracks comprising the library. If *O* is coded, the library is deleted; if *R* is coded, the library is reorganized.

The WORK keyword parameter indicates on which drive a second disk containing a disk system management work area is located. A work area is required if you are:

- Reallocating space for an existing library.
- Allocating space to build a source library on a disk containing only an object library.
- Removing a source library from a disk also containing an object library.

Library entries are temporarily stored in the work area while the program moves and reorganizes libraries.

Creating a Source Library: Assume you want to create a source library on a disk already containing an object library. The library will contain 15 tracks and be located on R1. The ALLOCATE statement looks like this:


1
4
8
12
16
20
24
28
32

```
// ALLOCATE TO-R1,SOURCE-15,WORK-F1
```

If you are allocating space for a source library on a disk that contains an object library, you must designate a work area. Your choices in this case are F1, R2, or F2, depending on which disk has the available work space.

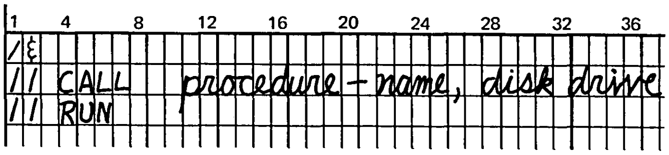
After a source library is created, you can load procedures or source programs into it. The Library Maintenance program copies entries into the library according to your specifications on the COPY statement. The format and possible keyword parameters for the COPY statement are described in the section *Creating an Object Library.*

Note: For the purpose of instruction, creation of source and object libraries have been described separately. It is most advantageous, however, to create both libraries at the same time.

Storing a Procedure in the Source Library: As stated previously under *Storing an Object Program,* the program PAY02 was loaded into an object library. Now that a source library has also been created, the procedures needed to execute PAY02 can also be stored on disk. Figure 8-11 shows the COPY statement required to enter the procedure (named PAYPRO) from cards into the library. Notice that LIBRARY-P is coded. *P* designates that a procedure is to be copied into the source library. (LIBRARY-S would indicate that a source program is being copied into the source library.)

The job stream needed to load the Library Maintenance program and copy PAYPRO into the source library is shown in Figure 8-12.

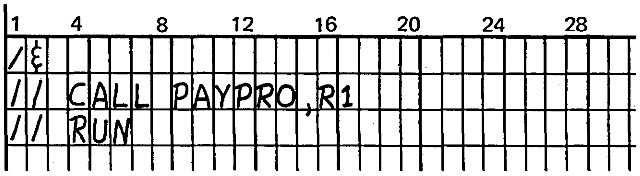
Calling Procedures: Procedures in the source library will not be executed until they are placed into a job stream by disk system management from either cards or the printer-keyboard. The job stream required to merge procedures and execute the appropriate program looks like this:



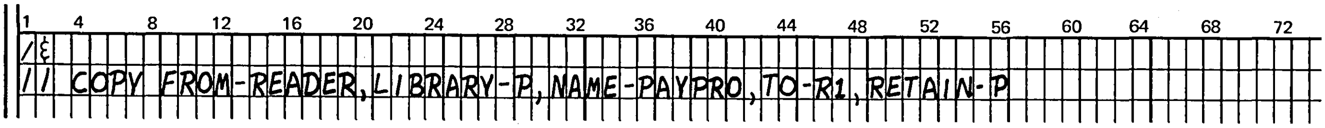
```
/&
// CALL procedure-name, disk drive
// RUN
```

CALL statements tell disk system management to merge procedures into the job stream. The CALL statements are, in effect, replaced by the procedures they identify and cannot be placed in the source library.

The statement required to merge the procedure PAYPRO into the job stream are:

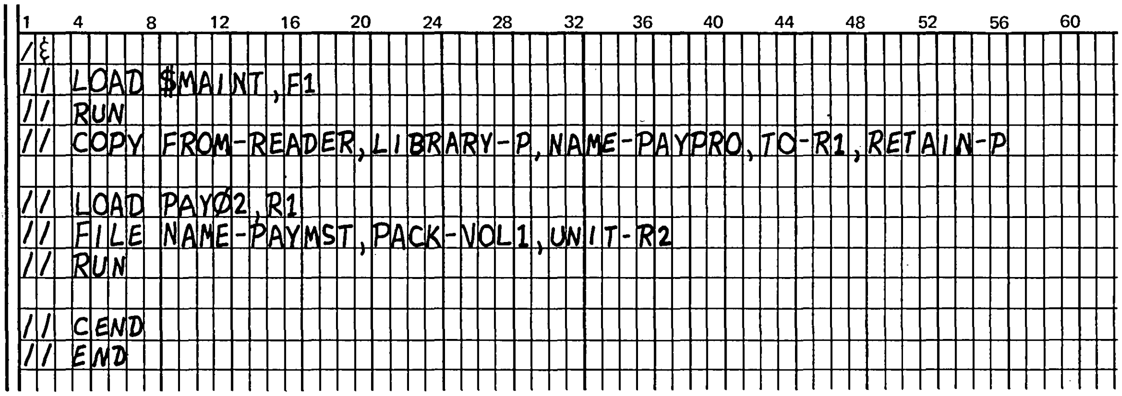


```
/&
// CALL PAYPRO,R1
// RUN
```



```
/&
// COPY FROM-READER,LIBRARY-P,NAME-PAYPRO,TO-R1,RETAIN-P
```

Figure 8-11. COPY Statement to Load PAYPRO Into the Source Library



```
/&
// LOAD $MAINT,F1
// RUN
// COPY FROM-READER,LIBRARY-P,NAME-PAYPRO,TO-R1,RETAIN-P

// LOAD PAYØ2,R1
// FILE NAME-PAYMST,PACK-VOL1,UNIT-R2
// RUN

// CEND
// END
```

This procedure is stored in the source library under the name PAYPRO.

Figure 8-12. Job Stream to Load PAYPRO Into a Source Library

Using RPG II to Store an Object Program on Disk

You can use RPG II to indicate the type of object output you want after compiling a source program. The compiled program can be stored in an object library or punched into cards. You usually want the object program written in the object library until you have corrected the severe errors in your program. When a program is written temporarily in the object library, it is overlaid by the next program written in that object library. The object program is written in the same object library containing the compiler, unless a COMPILE statement indicates otherwise. (See *Compiling and Storing a Source Program Into the Object Library* for further information.)

Column 10 of the RPG II Control Card Sheet is used to specify the object output. The following entries can be made:

Entry	*Explanation*
blank	The object program is written temporarily in the object library.
C	The object program is written permanently in the object library.
P	The object program is punched into cards.

Columns 75-80 of the control card are used to name your object program. This name is used in the library directory which also contains the location of your program on disk. The name may be comprised of any System/3 characters, but the first character must be alphabetic. If columns 75-80 are left blank, the compiler assumes the name is RPGOBJ.

Compiling and Storing a Source Program Into an Object Library

The COMPILE OCL statement tells disk system management to:

1. Compile a source program from a source library and store the object program in an object library, or

2. Compile a source program from cards and store the object in an object library.

The format of the COMPILE statement looks like this:

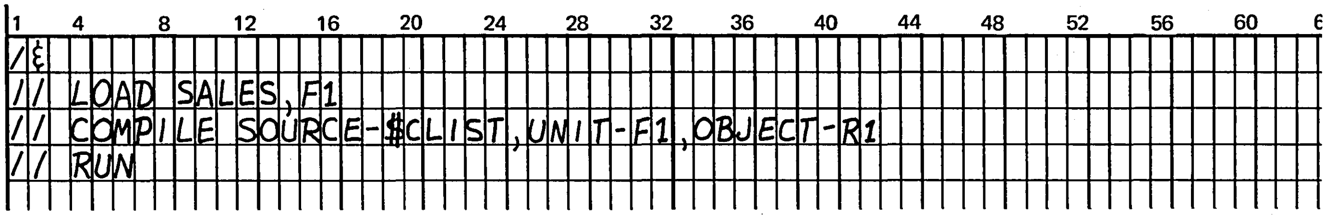
$$\text{// COMPILE SOURCE–name,UNIT–}\left\{\begin{matrix}\text{R1}\\\text{F1}\\\text{R2}\\\text{F2}\end{matrix}\right\}\text{,OBJECT–}\left\{\begin{matrix}\text{R1}\\\text{F1}\\\text{R2}\\\text{F2}\end{matrix}\right\}$$

The SOURCE keyword parameter is used if the source program is located in a source library. You must supply the same name given to the source program when it was stored in the library by the Library Maintenance program. The UNIT parameter must be used with the SOURCE parameter to identify the disk location of the source program to be compiled.

If the SOURCE keyword parameter is not used, the source program is assumed to be on cards following the RUN statement in the job stream.

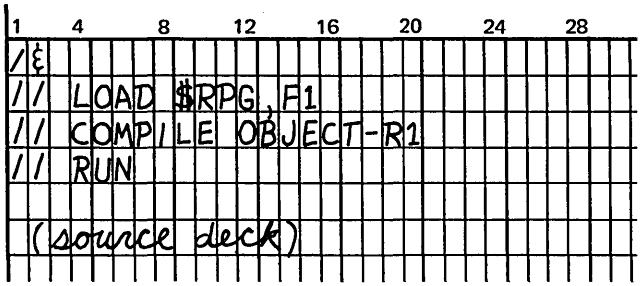
The OBJECT keyword parameter tells the system where the disk which will contain the object program is located. If the source program is on cards, the OBJECT keyword parameter is the only parameter which can be specified. If the OBJECT keyword parameter is omitted in either case, the object program is placed on the same disk pack as the compiler. The name assigned to object program in the object library is the name you assigned in the Program Identification (columns *75-80) on the RPG II Control Card Sheet. If you did not assign a name in these columns, RPGOBJ is assumed.

Sample Statements



```
/&
// LOAD SALES,F1
// COMPILE SOURCE-$CLIST,UNIT-F1,OBJECT-R1
// RUN
```

This sample job stream tells the system that the source program named SALES is located on a fixed disk on drive one (F1). The OBJECT-R1 keyword parameter tells the system to place the object program on a removable disk on drive one (R1).



```
/&
// LOAD $RPG,F1
// COMPILE OBJECT-R1
// RUN

 (source deck)
```

This sample job stream compiles a source program on cards and stores it in an object library on R1. If the OBJECT parameter was not coded, the program would be compiled and placed into the same object library as the compiler (F1).

1. What types of programs are stored in the source library? The object library?

2. What is a procedure?

3. What are two advantages of storing programs and procedures on disk?

4. What are the three ways entries can be copied into the source or object library?

5. Write the control statements to set aside ten tracks for the object library on a new disk.

6. Write the control statements to execute an object program named AR001 which is stored in the object library on F1. The program uses no disk files.

7. Write the control statements to copy the procedure in question 6 into the source library on F1.

8. Write the control statements to copy an RPG II object program stored in cards into the object library on F1 with the name AR001.

9. Write the control statements to transfer the program named AR001 in the object library on F1 to the object library on R1 and delete the current AR001 on R1.

10. Write the control statements to print out the object library directory from F1.

Answers to Review 8

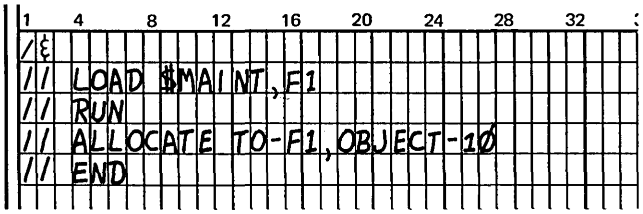
1. RPG II source programs and OCL procedures are stored in the source libraries.
Executable object programs and nonexecutable subroutines are stored in the object libraries.

2. A procedure is a set of OCL statements for a given job.

3. When the source and procedure libraries are used, time is saved loading programs and operation is made simpler.

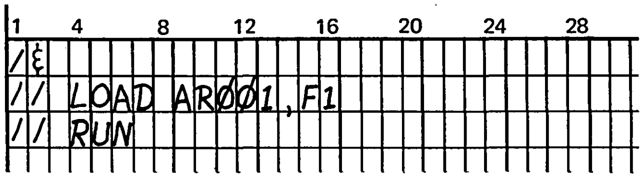
4. Entries can be cataloged into the source and object library via the library maintenance program. RPG II object programs can be cataloged into the object library at compilation time by specifying *C* or blank in column 10 of the RPG II control (HCC6) card. A blank entry specifies that the program be cataloged with a temporary attribute. *C* is used to catalog the programs permanently.

5.



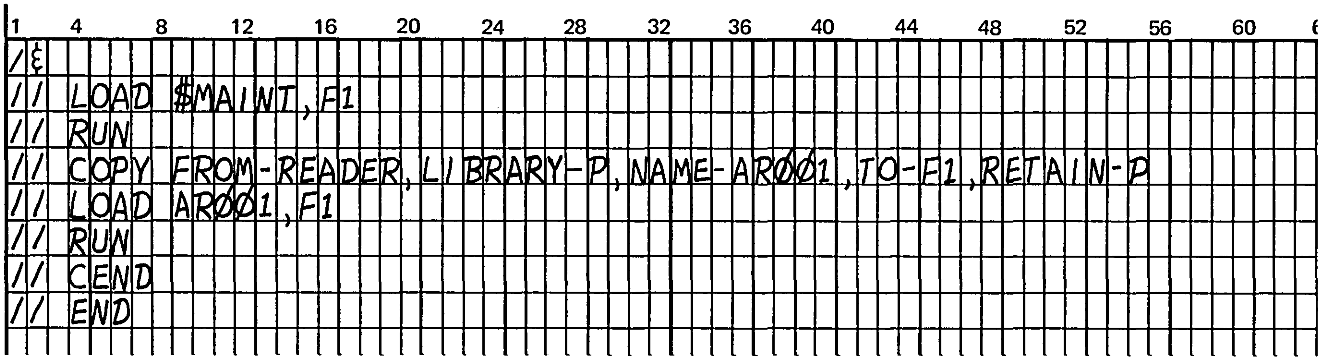
```
1  4   8   12  16  20  24  28  32
/&
// LOAD $MAINT,F1
// RUN
// ALLOCATE TO-F1,OBJECT-1Ø
// END
```

6.



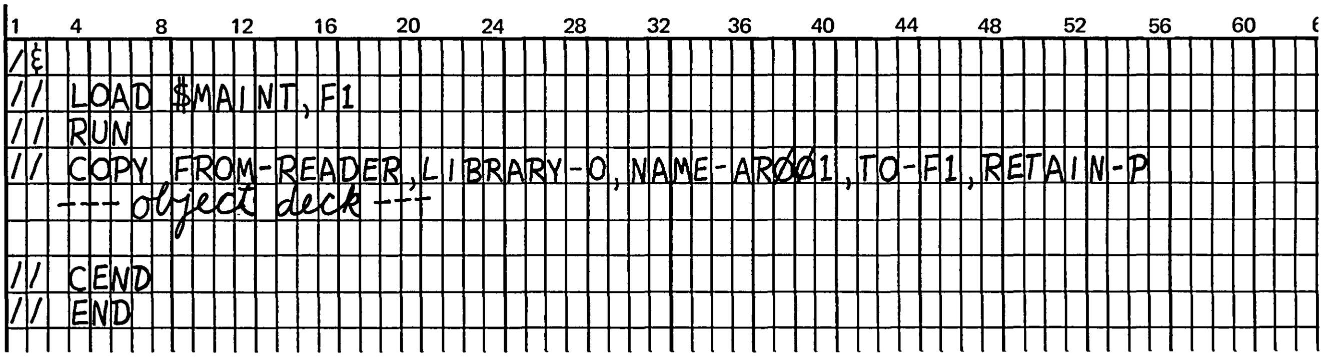
```
1  4   8   12  16  20  24  28
/&
// LOAD ARØØ1,F1
// RUN
```

7.



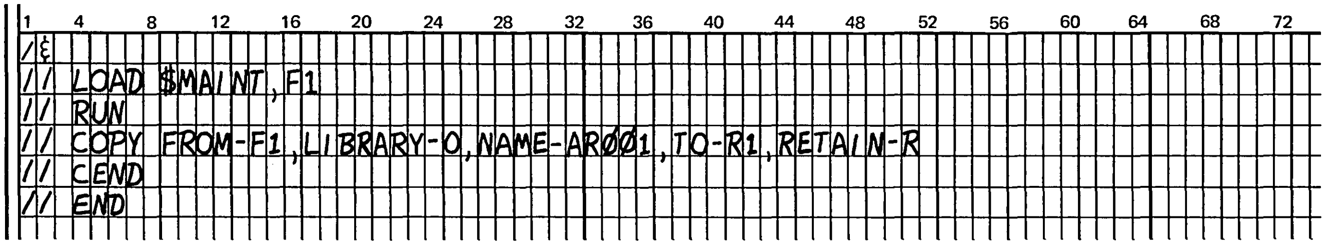
```
/&
// LOAD $MAINT,F1
// RUN
// COPY FROM-READER,LIBRARY-P,NAME-ARØØ1,TO-F1,RETAIN-P
// LOAD ARØØ1,F1
// RUN
// CEND
// END
```

8.



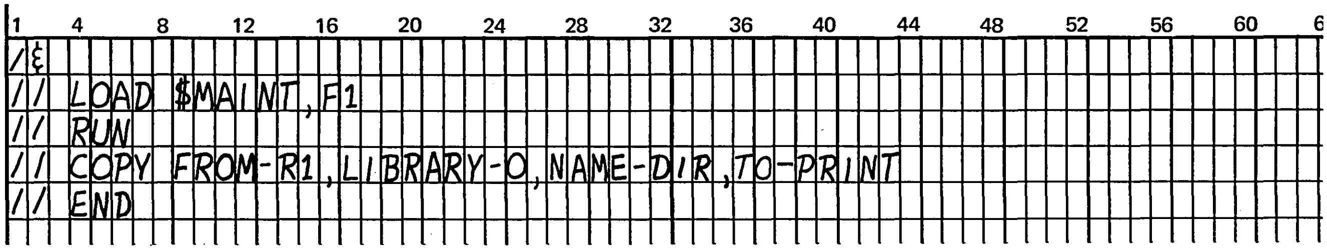
```
/&
// LOAD $MAINT,F1
// RUN
// COPY FROM-READER,LIBRARY-O,NAME-ARØØ1,TO-F1,RETAIN-P
---- object deck ---
// CEND
// END
```

9.



```
/&
// LOAD $MAINT,F1
// RUN
// COPY FROM-F1,LIBRARY-O,NAME-ARØØ1,TO-R1,RETAIN-R
// CEND
// END
```

10.



```
/&
// LOAD $MAINT,F1
// RUN
// COPY FROM-R1,LIBRARY-O,NAME-DIR,TO-PRINT
// END
```

CHAPTER 9 INQUIRY

CHAPTER 9 DESCRIBES:

Inquiry programs.

The use of the 5471 Printer-Keyboard for inquiry.

Coding RPG II control card specifications to classify inquiry programs.

How inquiry operates in a dedicated environment.

BEFORE READING THIS CHAPTER YOU SHOULD BE ABLE TO:

Define basic disk system concepts such as disk, main storage, and disk system management.

Define online.

Define object library.

AFTER READING THIS CHAPTER YOU SHOULD BE ABLE TO:

Define inquiry.

List and define the three classifications of programs for inquiry.

Describe roll-out and roll-in.

Describe the use of the 5471 Printer-Keyboard.

INTRODUCTION

In some data processing applications, inquiries that require immediate answers occur. One customer may want the status of his account; another may want to know if an item is in stock for immediate delivery. To answer these inquiries, you must be able to access certain disk records. The object program you use to retrieve this information is called an *inquiry program.*

Inquiry programs can be executed as part of a normal job stream, or they can interrupt other programs that are executing (interrupt environment). After a request for inquiry is made in an interrupt environment, the following things occur:

1. A program being executed is interrupted.

2. The current status of the program is stored on disk.

3. The inquiry program is loaded to retrieve and display the requested information.

4. The original program is reloaded.

REQUESTING INQUIRY IN AN INTERRUPT ENVIRONMENT

To interrupt a job prior to loading an inquiry program, you must make an *inquiry request.* To request inquiry, you must have a printer-keyboard such as an IBM 5471 Printer-Keyboard (Figure 9-1). On the 5471, the key labeled REQ is the Request Key. When pressed, it causes an interrupt and indicates to disk system management that an inquiry program is about to be loaded and the program that is executing must be stored on disk. The OCL statements for the inquiry program are then initiated from the printer-keyboard. (At least the READER statement indicating what input device contains the OCL statements must be entered from the keyboard.)

The inquiry program must be loaded from the object library (see *Chapter 8. Storing Programs and Procedures on Disk*). If you interrupt a program that is processing input from cards, refer to the *IBM System/3 Disk System Operator's Guide,* GC21-7508 for information on how to clear the MFCU.

Functions of the Inquiry Request Key

The Inquiry Request Key can be pressed to:

1. Interrupt an executing program and thereby enter the interrupt environment.

2. Initiate an inquiry program that is already in main storage waiting for an inquiry request to begin execution.

3. Initiate the reading of input data from the printer-keyboard for a program described in the second item of this list.

CLASSIFYING PROGRAMS FOR INQUIRY

Not all programs can be interrupted by an inquiry program. By coding specifications in column 37 (Figure 9-2) on the RPG II Control Card Sheet, you determine whether the program can be interrupted. The entries which classify the program are:

- ƀ (blank) – A *ƀ*-type program is a processing program that does not recognize an inquiry request. It cannot be interrupted.

- B – A *B*-type program is a processing program that recognizes an inquiry request, and, therefore, can be interrupted or stored on disk.

- I – While *I*-type programs can be loaded as inquiry programs in an interrupt environment (see note), a program is usually classified as an *I*-type when it is used as an inquiry program that is to remain in main storage for the servicing of inquiries. An *I*-type program can be executed *only* by an inquiry request (pressing the Request Key). An *I*-type program cannot be interrupted and stored on disk. If an input file is to be entered from the printer-keyboard for an *I*-type program, you must again press the Request Key to initiate reading of the input file.

Note: An inquiry program that interrupts a *B*-type program can be classified as *B, ƀ,* or *I*-type. An inquiry program loaded to perform a complete job is usually classified as a *B*-type program. An inquiry program loaded to answer one request or few requests is usually loaded as an *I*-type program (see *Planning Inquiry Programs* for further information). If a *B*-type program is rolled out by an inquiry program also classified as *B*-type, the inquiry program must complete execution before another inquiry request is made.

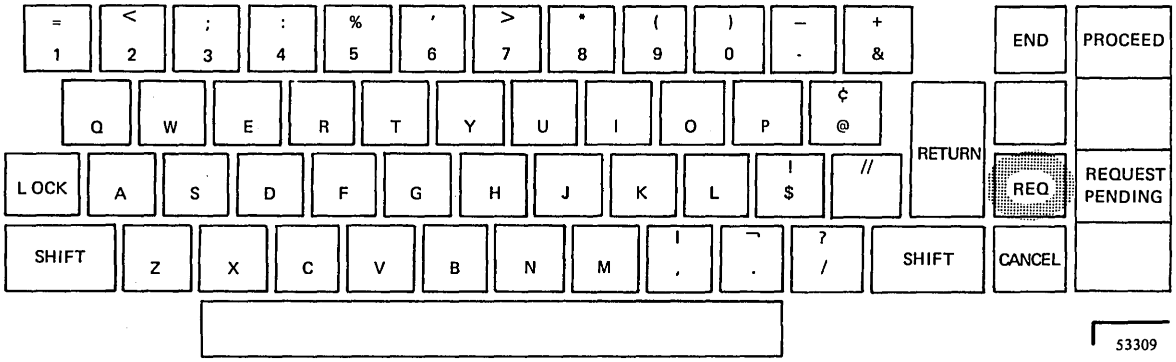
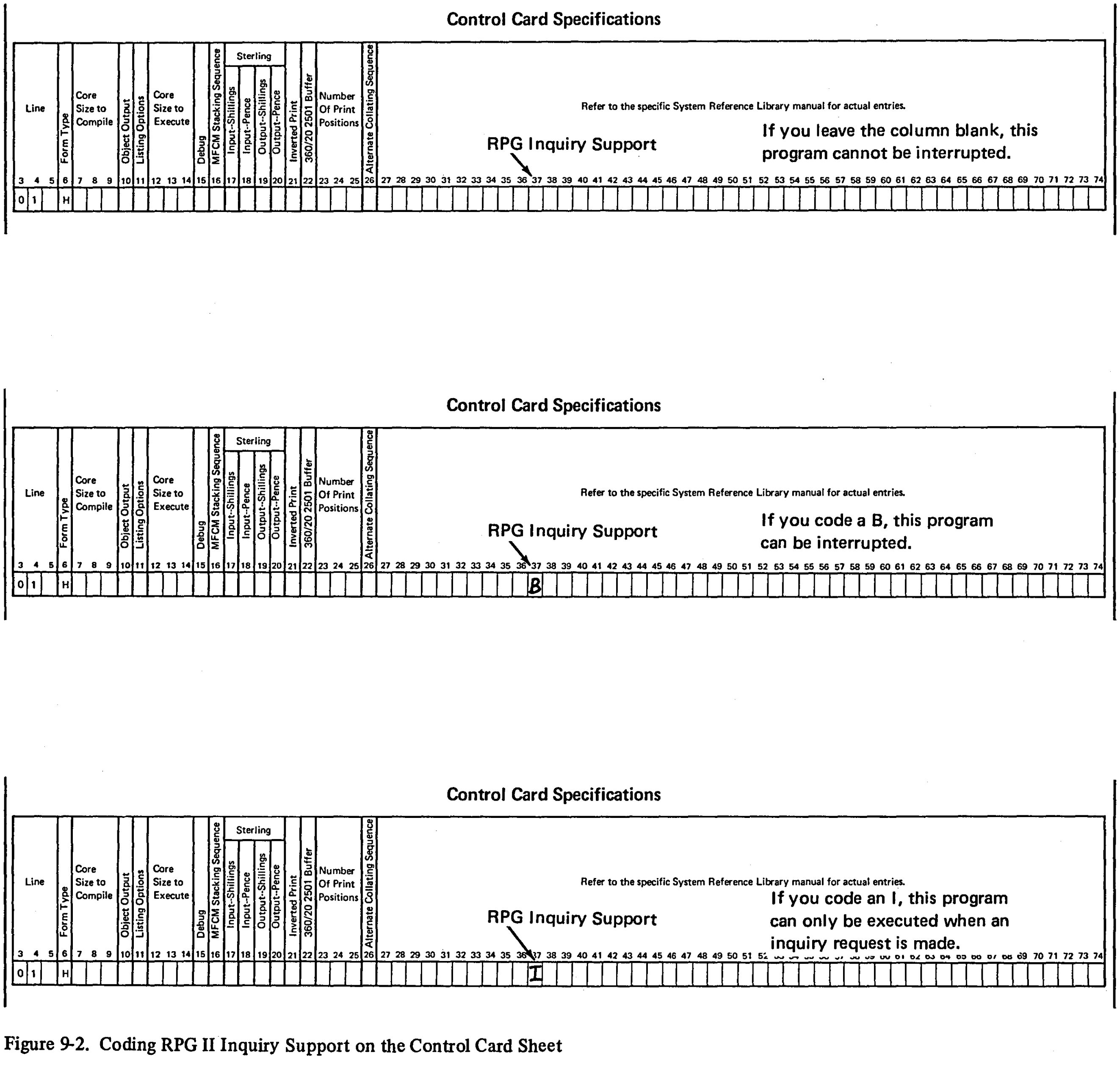


Figure 9-1. Keyboard Format of the 5471 Printer-Keyboard


Control Card Specifications
Line
Form Type
Core Size to Compile
Object Output
Listing Options
Core Size to Execute
Debug
MFCM Stacking Sequence
Sterling
Input-Shillings
Input-Pence
Output-Shillings
Output-Pence
Inverted Print
360/20 2501 Buffer
Number Of Print Positions
Alternate Collating Sequence
Refer to the specific System Reference Library manual for actual entries.
0 1 H

RPG Inquiry Support

If you leave the column blank, this program cannot be interrupted.

RPG Inquiry Support

If you code a B, this program can be interrupted.

B

RPG Inquiry Support

If you code an I, this program can only be executed when an inquiry request is made.

I

Figure 9-2. Coding RPG II Inquiry Support on the Control Card Sheet

INQUIRY IN AN INTERRUPT ENVIRONMENT

An inquiry program can be loaded into storage as any other program, or it can be loaded when an inquiry request is made to interrupt a program that is executing. When your system is controlled by one program at any one time, you have a *dedicated system.* Therefore, in an interrupt environment you must interrupt the executing program to allow the inquiry program to control the system. You request an interrupt by pressing the Request Key on the printer-keyboard. You can only interrupt *B*-type programs. As soon as the Request Key is pressed, the system sets an indicator and the executing program completes the execution cycle it is in. A system routine called roll-out then transfers the *B*-type program from main storage onto disk, retaining the current status of the program (Figure 9-3, insert A). Space is allocated for the rolled out program at system generation time. (See the *IBM System/3 Disk System Operator's Guide,* GC21-7508 for system generation procedures.) *Chapter 8. Storing Programs and Procedures on Disk* contains the scheduler work area size including space requirements for roll-out/roll-in. The inquiry program is then loaded into main storage for execution. (The inquiry program may be a *b̸, I,* or another *B*-type program.) Figure 9-3, insert B shows the inquiry program being loaded from disk into main storage. After the inquiry program reaches the end of its processing, the *B*-type program that was interrupted is transferred back into main storage by the roll-in routine (Figure 9-3, insert C). The interrupted program begins execution at the point of the interruption.

Notice that the inquiry program that was loaded does not get rolled out onto disk. Therefore, you cannot accumulate any information such as totals to be saved from one inquiry request to the next.

FILE PLANNING

When an inquiry program is loaded, the files for that program must be online. If an inquiry is received and the proper file is not online, then the inquiry cannot be processed. Your correct files must be mounted before processing can occur. This involves some file planning and job scheduling on your part. For example, if most inquiries about stock status come early in the morning, then the inventory file should be online at that time, and programs using that file such as invoicing or inventory transactions can also be run at the same time.

PLANNING INQUIRY PROGRAMS

Since *B*-type programs can be interrupted, you must determine what types of programs should be classified as *B*-type. Usually long reports that do not have to be finished immediately are classified as *B*-type. Such a report might be an end-of-month stock status report.

Inquiry programs that can interrupt *B*-type programs can be classified as *b̸, B,* or *I*-type. For example, suppose you are running an end-of-month stock status report, and now find you must run a payroll job. The payroll job can roll-out the stock status job. It is a short job that must be finished immediately. Another example of an inquiry program that might need to be loaded immediately would be a request to determine where a certain inventory item is located so that it can be shipped. Since the inventory file is online for the stock status report, the location of the item could be determined quickly by an inquiry program.

Those programs you do not want rolled out should be *b̸*-type. For example, you may be running a payroll job and checks are positioned on the printer. You may not want the payroll program rolled out, since the operator may have to remove the checks and not reposition them correctly. If you are running a teleprocessing program, you may not want it rolled out because you may lose telephone connections.

Programs classified as *I*-type can serve two purposes. In dual programming (see *Chapter 10. Dual Programming Feature*), an inquiry program can be loaded into one level and remain there to service inquiries. Such a program must be classified as *I*-type. In a dedicated system, an *I*-type program could be loaded for a length of time to answer requests. For example, an *I*-type program could be loaded during the second shift of a day to answer inquiries into the amount or location of items in a warehouse. An *I*-type program remaining in main storage can only be executed by pressing of the Request Key.

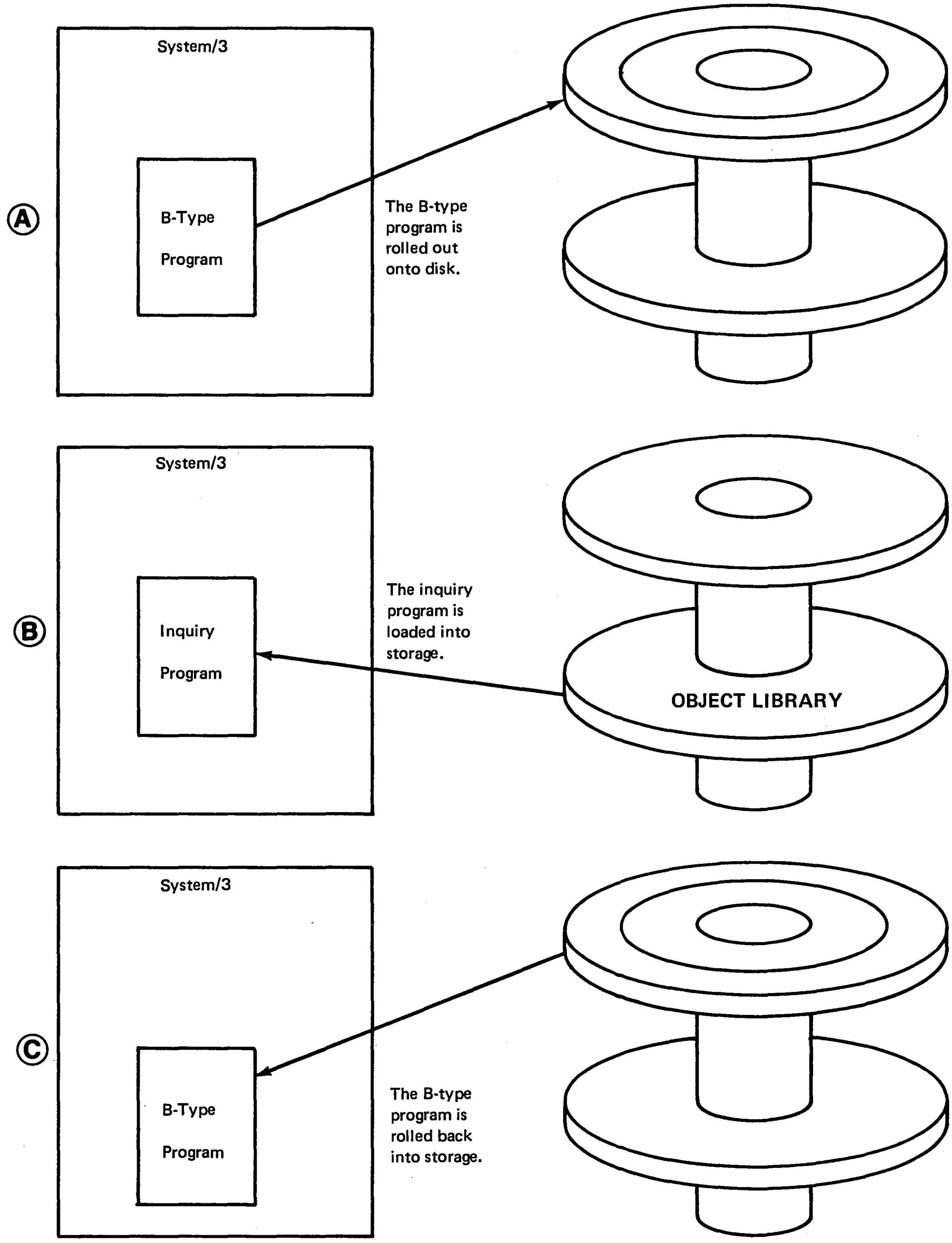

A
System/3
B-Type
Program
The B-type program is rolled out onto disk.
B
System/3
Inquiry
Program
The inquiry program is loaded into storage.
OBJECT LIBRARY
C
System/3
B-Type
Program
The B-type program is rolled back into storage.

Figure 9-3. Roll-out and Roll-in

Review 9

1. What is meant by inquiry?
2. What are the three classifications of programs related to inquiry?
3. What is meant by roll-out and roll-in?
4. What is the significance of the 5471 Printer Keyboard in inquiry?

Answers to Review 9

1. Inquiry is a request for the contents of a specific disk record. This can be either in a batch environment or an interrupt environment where an executing program is interrupted to perform the inquiry.

2. An *I*-type program is an inquiry program which can only execute upon an inquiry request. A *B*-type program can be interrupted. Any type can interrupt a *B*-type program. A *ƀ*-type (blank) program cannot be interrupted.

3. When an inquiry request is made, the executing program is halted and written out onto an area of disk, preserving the current status of the program. The inquiry program is read in from disk. When inquiry is completed, the original program is read back into storage and execution continued.

4. The 5471 Printer-Keyboard is required to perform inquiry and roll-in/roll-out. The inquiry request is initiated by pressing the Request Key on the keyboard.

CHAPTER 10 DUAL PROGRAMMING FEATURE

CHAPTER 10 DESCRIBES:

Operation of the dual programming feature (DPF).

Advantages of running programs under DPF.

Considerations for operating under DPF.

Considerations for running System/3 programs under DPF.

How to execute an RPG II program in DPF.

PARTITION statement and considerations for loading programs in a DPF environment.

BEFORE READING THIS CHAPTER YOU SHOULD BE ABLE TO DESCRIBE:

Basic disk concepts such as I/O, main storage, supervisor, processing time, and dedicated environment.

File processing, removable and fixed disks, and Initial Program Load (IPL).

The function of inquiry (ƀ, B, and I-type programs).

The function of teleprocessing.

Compilation of RPG II programs.

Function of DATE, LOG, NOHALT, HALT, IMAGE, and FORMS statement.

Overlays.

AFTER READING THIS CHAPTER YOU SHOULD BE ABLE TO:

Describe how DPF operates.

List the advantages of running under DPF.

List the considerations for operating under DPF.

Demonstrate understanding of some of the considerations for running System/3 programs under DPF.

Describe how to execute RPG II object programs in DPF.

Identify, using reference material, the OCL statements that require special considerations when loading a program in DPF.

Describe the function and coding for a PARTITION statement.

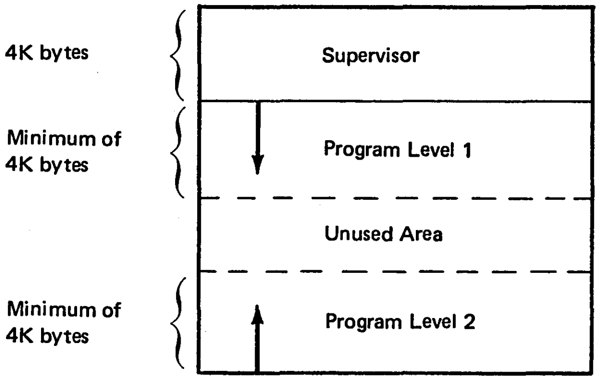
INTRODUCTION

With the dual programming feature (DPF), you can have two programs in main storage at the same time. Only one, however, can be executing instructions at any one time.

When DPF is operating, main storage contains the supervisor and two programs. Control is transferred from one program to the other whenever the program that is executing must await completion of an input or output operation. For example, one program requests a print operation, but the printer is still busy with a previous request. Control is then transferred to the other program. Similarly, one program requests that a card be read for processing. Since the program must wait until reading is completed before it can process the data, control is transferred to the other program. The second program must await completion of an input or output operation, before control is returned to the first program. Similarly, control is transferred when a halt occurs in one program level.

Most programs have a significant amount of time when they are waiting for I/O completion. If both programs are waiting, the program whose I/O completes first receives control.

Figure 10-1 shows how main storage is organized in a DPF environment. The supervisor occupies 4K (4,096) bytes of storage in DPF. The storage areas occupied by the two programs are called program level 1 and program level 2. Each level must have a minimum of 4K bytes, if the level is active.



The arrows indicate the direction in which storage is allocated to each level. If the two programs do not occupy the entire amount of storage you have, an unassigned area exists between program levels. This area can then be used by disk system management to increase the efficiency of your system operation.

Figure 10-1. Main Storage in a DPF Environment

ADVANTAGES OF RUNNING PROGRAMS IN A DPF ENVIRONMENT

Main Storage

DPF enables you to make more efficient use of your system storage. For example, if you were to run a 4K program on a 12K system in a dedicated environment, you would only be using 7K of your storage:

Program = 4K
Supervisor = 3K
Used Storage = 7K

Consequently, 5K storage is unused.

In a DPF environment, you could run two 4K programs on a 12K system and use the entire storage capacity:

Two programs = 8K
Supervisor = 4K
Used storage = 12K

Input/Output Devices

With proper planning, DPF also enables you to use your system input/output devices more effectively. In a dedicated environment, you may run a program to copy one disk to another. The MFCU and printer are not used. In DPF, you could run two programs: one to copy a disk, the other to read cards from the MFCU and print the data on the printer.

Processing Time

DPF permits more efficient use of the computer's processing time. When a program is executing, the central processing unit is executing the program's instructions. When instructions are not being executed, the processing capabilities of the computer are not used. For example, if an instruction cannot be executed because data is not available to be processed (waiting for a card to be read) or because a device is not ready to execute the requested instruction (printer is busy with a previous print instruction), execution of the program is suspended until the required conditions are satisfied. When the execution is suspended, the computer's processing time is lost because no instructions can be executed. DPF allows control to transfer to another program. That program can then begin executing instructions, thereby using the processing time.

The inquiry function of System/3 and teleprocessing (BSC) can be operated more efficiently under DPF than a dedicated environment. In a dedicated environment, an inquiry program must reside in storage or be loaded every time a request is made, consequently rolling out a program that is executing. In DPF, the inquiry program can be loaded into one program level, and a program can still execute in the other level.

If you are using teleprocessing (BSC), one program level could be dedicated to teleprocessing; the other level would be available for running other programs. For example, if messages are being relayed from one terminal to another, program level 2 can be assigned to teleprocessing. Although messages are not relayed constantly through the day, the teleprocessing program may have to be in storage at all times. Therefore, if the teleprocessing program is loaded into program level 2, it is available when needed. When the teleprocessing program is inactive, normal processing programs can use system resources.

CONSIDERATIONS FOR OPERATING UNDER DPF

You must consider the following points when planning to use DPF:

1. You must determine that you have enough storage. Because the supervisor requires 4K bytes, you could not, for example, run one 4K program and one 5K program on a 12K system. One of the two programs could not be loaded.

2. Two programs in storage must use the proper combination of I/O devices. Both program levels cannot use the MFCU or the printer.

   For example, if you were running two jobs both of which require the printer, such as an invoicing and a sales analysis job, one program could not execute because the printer would not be available. The disk and the printer-keyboard can be shared by two programs. The disk data file can be shared depending upon the type of disk file processing. Figure 10-2 shows the normal considerations for efficient file processing. Figure 10-3 shows the restrictions when a data file is shared by two program levels.

3. Care should be taken when the printer-keyboard is used as:

   - The system input device for both program levels.

   - The system input device in one level and as an input device for an RPG II program in the other level.

   If these situations arise, the operator must first determine which level is requesting information. Use of the RPG II DSPLY operation code may help determine which level is requesting information. The performance of DPF may also be less efficient, because the operator may hold up your system when keying in information.

4. Each one of these IBM programs requires dedicated use of the system: RPG II Compiler, Library Maintenance, Basic Assembler, and IBM 1255 Utility Program.

   Note: Object programs denoted by a LOAD* OCL card cannot be loaded into level 2. In order for an object program on cards to be loaded into level 2, it must first be copied from the reader to an object library and then loaded from the object library.

5. File planning is necessary to avoid problems that arise when two programs use the same disk drives. For example, if two programs were using two separate files on the same disk (Figure 10-4), the access arm may have to move every time each program requests I/O. Movement of the access arm will increase access time, slowing the performance of the program. To avoid this problem, it is most advantageous to have files for each active program on separate disk drives. You could, however, have separate files on two removable disks or one file immediately above the other file on fixed and removable disks as shown in Figure 10-5.

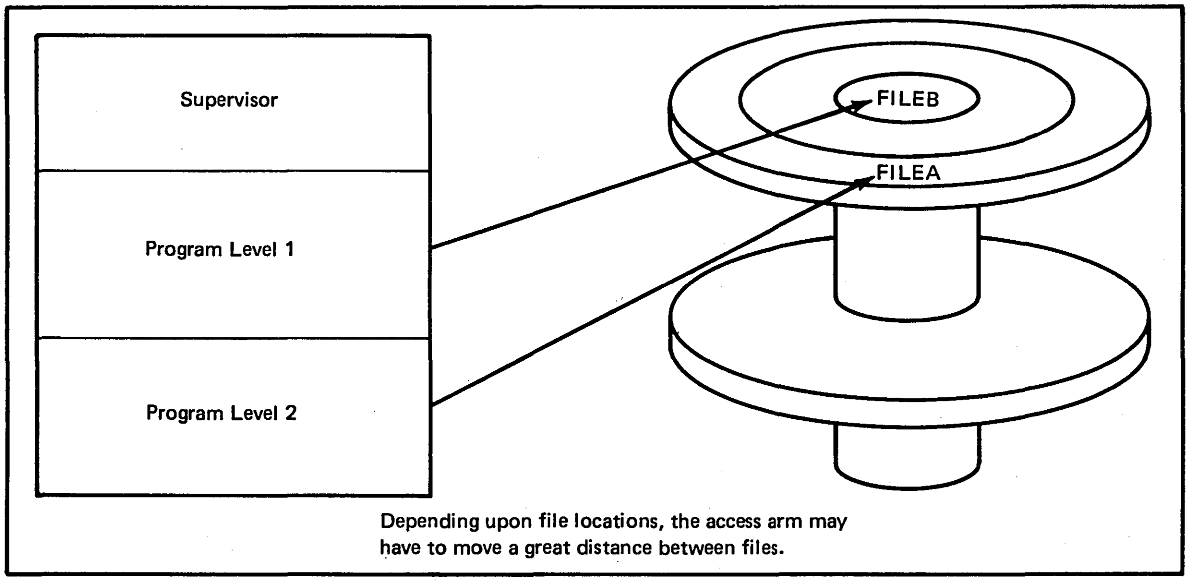
		Processing Method		
		Consecutive	Indexed	Random by Relative Record Number
File Organization	Consecutive	Yes	No	Yes
	Indexed	Create or retrieve--Yes --------- Add or update--No	Yes	Yes
	Direct	Yes	No	Yes

Note: You cannot reload a permanent file. If you reload an indexed file as a consecutive or direct file, that indexed file becomes a consecutive or direct file, respectively.

Figure 10-2. Considerations for Efficient File Planning

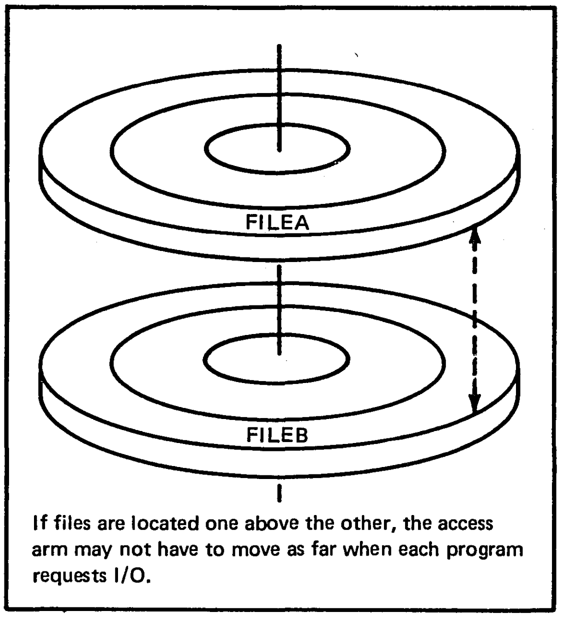
		Program Level 1		
		Read a File	Create or Add to a File	Update Records in a File
Program Level 2	Read a File	Yes	No	Yes
	Create or Add to a File	No	No	No
	Update Records in a File	Yes	No	No

Figure 10-3. Disk File Processing of a Data File Stored by Two Program Levels



Depending upon file locations, the access arm may have to move a great distance between files.

Figure 10-4. File Locations Causing Arm Movement



If files are located one above the other, the access arm may not have to move as far when each program requests I/O.

Figure 10-5. File Locations Causing Less Arm Movement

Points to Remember When Planning Files

- If two programs reference the same disk unit (R1, F1, R2, F2), they *must* be processing the same disk pack, because you cannot change a pack on that unit for each program's I/O request.

- If you load programs or procedures from a disk, or use a disk for IPL, the disk cannot be removed. In this case, it may be best to have programs and procedures on the fixed disk, leaving the removable disks free for changing.

- If one of the programs uses offline, multi-volume files, the other program must not have files on the same volumes. When a disk is replaced for one program, it may contain files still needed by the other.

- If two programs are initiated, one of which uses data files on the system drive, the job that does *not* use data files on that disk should be initiated first. Program initiation involves numerous accesses to the system programs that could greatly increase your access time. If the program using data files on the system drive were initiated first, initiation of the other program would cause the access arm to move frequently from the data files to the system programs. If the program not using data files on the system drive were initiated first, it would read the system programs and be finished with the drive, leaving it free for initiation and execution of the program using the data files on the system drive.

6. Individual programs will not necessarily run in any less time under DPF than they would in a dedicated environment. In fact, an individual program may take longer to run in DPF. A set of programs, however, may finish sooner than they would if they were run in a dedicated environment. For example, if you had two jobs to run, neither of the individual jobs may run in any less time. However, the set may be finished sooner in a DPF environment, because one program would be using any processing time that the other could not use. If the programs were run consecutively, processing time may be wasted during each program's run.

7. DPF requires efficient job scheduling because of the preceding considerations. Suppose you had four jobs to be run requiring the I/O shown in Figure 10-6. Jobs 1 and 2 and Jobs 3 and 4 can be run together, because they do not require the same I/O devices. If Job 2 finishes before Job 1, you could run Job 4 because Jobs 1 and 4 do not require the same devices. If, on the other hand, Job 1 finishes first, Job 3 could not be run with Job 2, because both jobs require the printer for output.

Program Level 1	JOB1 An inquiry program that: ● Reads printer-keyboard. ● Reads disk. ● Writes printer-keyboard.	JOB3 A stock status report that: ● Reads disk. ● Prints.
Program Level 2	JOB2 An inventory updating program that: ● Reads cards. ● Reads disk. ● Updates disk. ● Prints.	JOB4 A detail punching job that: ● Reads cards. ● Punches cards.

Figure 10-6. Job Scheduling for DPF

CONSIDERATIONS WHEN RUNNING SYSTEM/3 PROGRAMS IN A DPF ENVIRONMENT

The inquiry function, the Disk Sort Program, the Alternate Track Assignment Program, and the Disk Initialization Program require special considerations when operating in a DPF environment.

Inquiry

An inquiry program can either reside in one of the two program levels in main storage or not reside in main storage. If it is not in storage, an executing program must be rolled out when an inquiry request is made. Remember the three classifications of programs for inquiry:

- *I*-type is an inquiry program that cannot be rolled out.
- *B̸*-type cannot be rolled out.
- *B*-type can be rolled out.

If the inquiry program is in main storage, it must be an *I*-type program, and the other level must contain a *B̸*-type program. The *I*-type program is then executed when the Inquiry Request Key is pressed.

If the inquiry program does not reside in main storage, it can be any of the three program types. However, if both partitions are active, you must have a *B*-type program in level 1 and a *B̸*-type program in level 2 to operate inquiry when the inquiry program is not resident in storage. This is because the system does not allow level 2 to be rolled out upon an inquiry request. Consequently, no *B*-type program can reside in level 2.

When a *B*-type program is rolled out in level 1, the OCL statements for the inquiry program must be initiated from the printer-keyboard (at least a READER statement indicating what device contains the OCL statements must be entered). The same storage and I/O devices are available to the inquiry program as were available to the *B*-type program when it was rolled out. However, if the inquiry program is to share the same disk file as the *B*-type program, the file processing restrictions in Figure 10-2 and Figure 10-3 apply.

Disk Sort, Alternate Track Assignment, and Disk Initialization

The Disk Sort, Alternate Track Assignment, and Disk Initialization programs require a minimum of 5K bytes each to execute.

If they are loaded into program level 2, they are assigned 5K bytes unless you use an OCL PARTITION statement. (You can use an OCL PARTITION statement to indicate the size of the program you wish to run in level 2.)

The programs cannot be run in a 12K DPF system, if level 2 is active or a previous job used the PARTITION statement for level 2. (4K for the supervisor plus 4K for program level 2 leaves only 4K for the program in level 1.) If the PARTITION statement was used for the previous program, you must perform another IPL to run the programs. You can never use more than 4K for program level 2 on a 12K DPF system.

Note: If you load the Disk Sort Program into level 1, all storage except 4K bytes for the supervisor is used unless level 2 is already active or you preassigned storage to level 2 using a PARTITION statement.

EXECUTING RPG II OBJECT PROGRAMS IN A DPF ENVIRONMENT

The amount of storage available for object program execution may differ from the amount of storage available fór object program generation. When the storage sizes differ, you should indicate on the RPG II Control Card Sheet the amount of main storage the object program can use. If this amount results in overlays, some of the DPF performance advantage may be lost. Columns 12-14 (Core Size to Execute) indicate the amount of storage in which the program will execute. The entries for these columns are:

Entry	*Explanation*
Blank	Storage available for object program execution is the same as that for program compilation.
001-029	Storage available for program execution (if different from storage for program compilation).

The entry must end in column 14. The entry is some multiple of 1K bytes of storage (K = 1,024). To determine the entry, subtract the amount of storage occupied by the second program level and the supervisor from the total storage capacity of the system. It is rarely desirable to specify less than 4K since that is the minimum partition size. Figure 10-7 is a sample Control Card Sheet indicating the object program will execute in 4K bytes.

LOADING PROGRAMS IN A DPF ENVIRONMENT

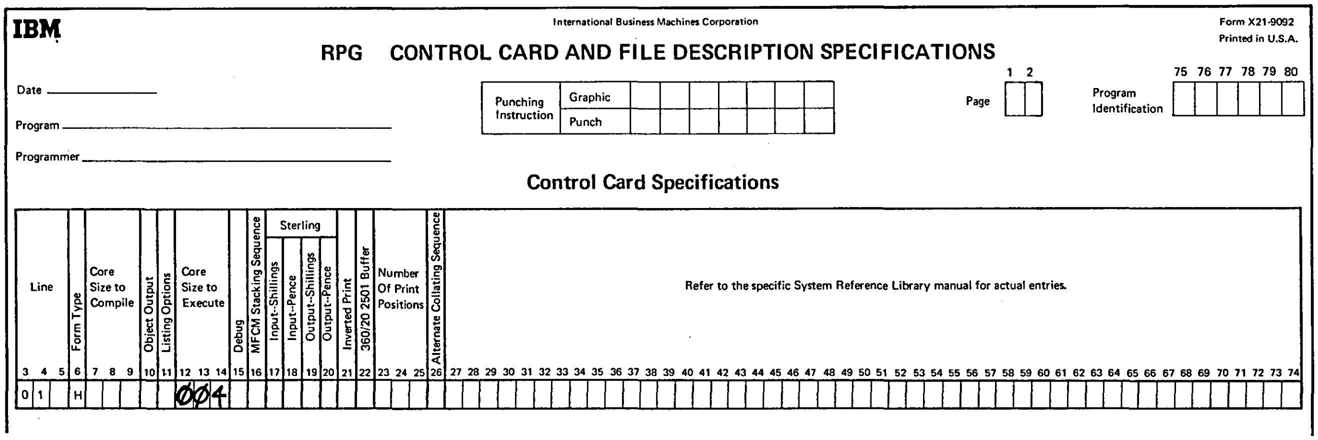
A program can be loaded into either program level first. You tell the supervisor which system input device contains the job streams for the programs by selecting the device on the Dual Program Control Switch. (Refer to the *IBM System/3 Disk System Operator's Guide,* GC21-7508 for further operating procedures.) When preparing your job streams, you should be aware of the following OCL considerations:

1. DATE statement. The DATE statement you use as an IPL statement to set the system date must be supplied with the first program loaded. Do not provide a DATE statement for the other program level.

   A DATE statement that temporarily changes the system date can be used within the set of OCL statements for programs in either program level. This DATE statement applies only to the program for which it is used.

2. LOG statement. LOG statements can be placed anywhere among the statements in either job stream. There are, however, certain restrictions on their use.

   - Only LOG statements for program level 1 can tell the system to use a different logging device. The device used for level 1 is also used for level 2.



IBM

International Business Machines Corporation

Form X21-9092
Printed in U.S.A.

RPG CONTROL CARD AND FILE DESCRIPTION SPECIFICATIONS

Date

Program

Programmer

Punching Instruction: Graphic / Punch

Page 1 2

Program Identification 75 76 77 78 79 80

Control Card Specifications

Line	Form Type	Core Size to Compile	Object Output	Listing Options	Core Size to Execute	Debug	MFCM Stacking Sequence	Sterling: Input-Shillings	Sterling: Input-Pence	Sterling: Output-Shillings	Sterling: Output-Pence	Inverted Print	360/20 2501 Buffer	Number Of Print Positions	Alternate Collating Sequence	Refer to the specific System Reference Library manual for actual entries.
3 4 5	6	7 8 9	10	11	12 13 14	15	16	17	18	19	20	21	22	23 24 25	26	27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74
0 1	H				ØØ4											

Figure 10-7. Core Size to Execute

- LOG must be on for both program levels before logging can occur. If a LOG statement for either program level stops the logging function, logging is stopped for both levels.

- When the printer is the logging device, OCL statements and message codes are not printed if the program in either level uses the printer as an output device.

Figure 10-8 shows sample LOG statements in a job stream.

3. NOHALT statement. The NOHALT statement is invalid for program level 2. The program in this level always stops after each job.

4. HALT statement. The HALT statement is ignored by program level 2.

5. IMAGE and FORMS statements. These statements are invalid if the other level has the printer allocated to it and the job cannot be run.

6. PARTITION statement. The PARTITION statement is used only in DPF.

   The PARTITION statement is used to indicate the size of the program you wish to run in program level 2. If you do not use a PARTITION statement, when loading a program into level 2, the supervisor automatically assigns 4K bytes of storage to level 2, *if* the storage is available. To ensure that storage is available for program level 2 you should use a PARTITION statement (Figure 10-9). You should only assign as much storage as needed for level 2, however, because some IBM programs can use unassigned storage to organize their performance. Only another PARTITION statement or another IPL can then change the size of program level 2.

   The PARTITION statement must be supplied in the job stream for program level 1. If can only be assigned when a program in level 2 is at end of job.

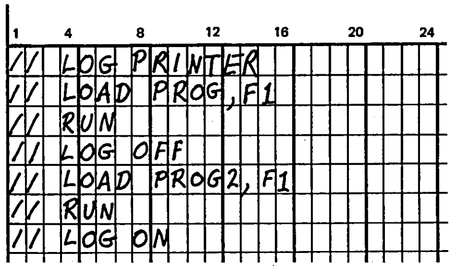
   The format of the PARTITION statement is:

```
// PARTITION    size
```

   You must state the number of bytes of storage you want to save for program level 2. The number must be equal to or greater than 4096. The amount of storage you specify is rounded to the next highest 256 bytes by the supervisor, if it is not a multiple of 256 bytes.

Sample Job Streams

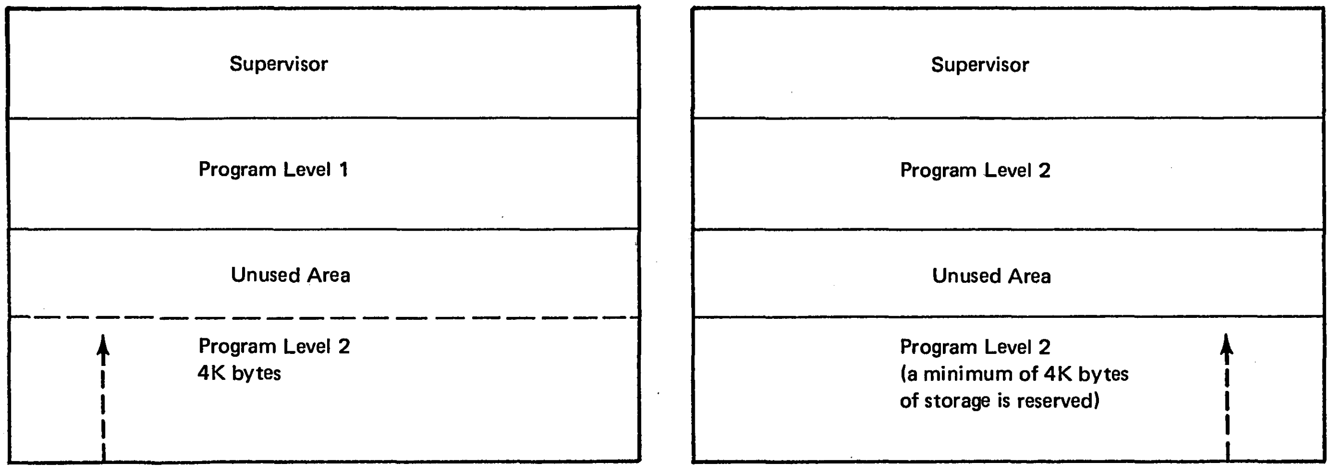
Figure 10-10 shows the job streams required to load the four jobs shown in Figure 10-6. Assume the system has the minimum system configuration plus the 5471 Printer-Keyboard and dual drives. The Dual Program Switch indicates from what device OCL statements are read. MFCU is always hopper 1, and at system generation time P-KY was assigned to the 5471 Printer-Keyboard.


1
4
8
12
16
20
24

```
// LOG PRINTER
// LOAD PROG,F1
// RUN
// LOG OFF
// LOAD PROG2,F1
// RUN
// LOG ON
```

The first LOG statement indicates that the printer is used as the logging device while program PROG1 is being run. OCL statements and error messages are not printed for program PROG2 because of the second LOG statement. The third LOG statement causes the logging device to be used again.

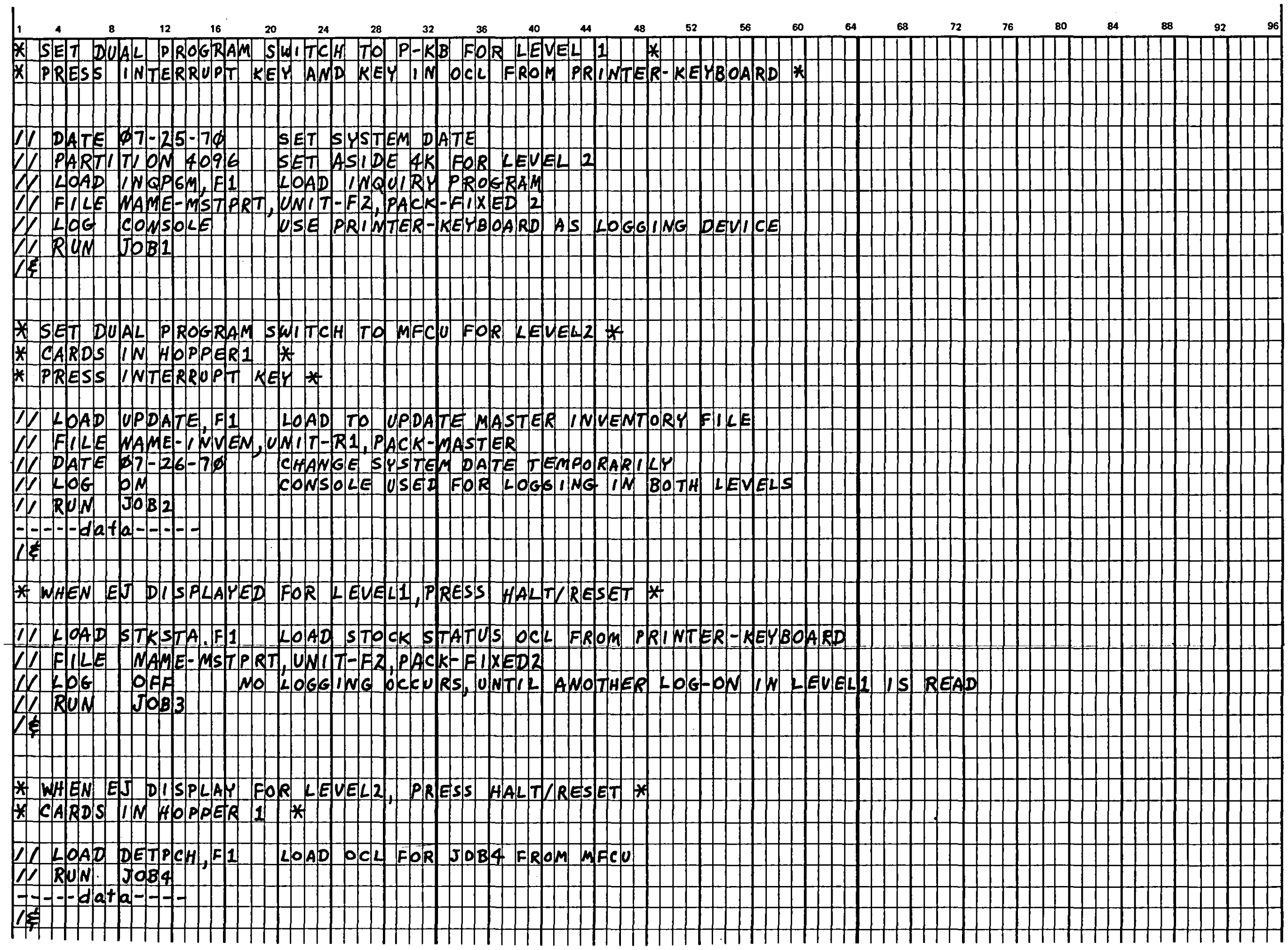
Figure 10-8. LOG Statement Example



Without a PARTITION statement
If level 1 is not using the storage and a program is loaded into level 2, it is assigned at least 4K bytes. When the program in level 2 comes to end of job, the storage for level 2 is no longer reserved and level 1 can use it.

With a PARTITION statement
If a PARTITION statement is used, the assigned storage can only be used by the program in level 2. It is reserved. Even when the program in level 2 comes to end of job that storage is reserved for future programs in level 2.

Figure 10-9. Assigning Storage to Program Level 2



```
* SET DUAL PROGRAM SWITCH TO P-KB FOR LEVEL 1   *
* PRESS INTERRUPT KEY AND KEY IN OCL FROM PRINTER-KEYBOARD *


// DATE Ø7-25-7Ø       SET SYSTEM DATE
// PARTITION 4096      SET ASIDE 4K FOR LEVEL 2
// LOAD INQPGM,F1      LOAD INQUIRY PROGRAM
// FILE NAME-MSTPRT,UNIT-F2,PACK-FIXED 2
// LOG  CONSOLE        USE PRINTER-KEYBOARD AS LOGGING DEVICE
// RUN  JOB1
/&


* SET DUAL PROGRAM SWITCH TO MFCU FOR LEVEL2 *
* CARDS IN HOPPER1  *
* PRESS INTERRUPT KEY *

// LOAD UPDATE,F1      LOAD TO UPDATE MASTER INVENTORY FILE
// FILE NAME-INVEN,UNIT-R1,PACK-MASTER
// DATE Ø7-26-7Ø       CHANGE SYSTEM DATE TEMPORARILY
// LOG  ON             CONSOLE USED FOR LOGGING IN BOTH LEVELS
// RUN  JOB2
-----data-----
/&

* WHEN EJ DISPLAYED FOR LEVEL1,PRESS HALT/RESET *

// LOAD STKSTA.F1      LOAD STOCK STATUS OCL FROM PRINTER-KEYBOARD
// FILE  NAME-MSTPRT,UNIT-F2,PACK-FIXED2
// LOG   OFF      NO LOGGING OCCURS, UNTIL ANOTHER LOG-ON IN LEVEL1 IS READ
// RUN   JOB3
/&


* WHEN EJ DISPLAY FOR LEVEL2, PRESS HALT/RESET *
* CARDS IN HOPPER 1  *

// LOAD DETPCH,F1      LOAD OCL FOR JOB4 FROM MFCU
// RUN  JOB4
-----data----
/&
```

Figure 10-10. Sample Job Stream

1. What advantages does DPF offer?

2. Indicate with *S* (shared) or *N* (not shared) which devices can be shared between the two levels of programs under DPF:

   a. MFCU

   b. Printer

   c. Printer/Keyboard

   d. Disk drive

   e. Disk file

3. What limitations apply to shared disk files?

4. Name two programs which cannot be run in a DPF environment.

5. How do you allocate storage to the two programs to be run under DPF?

Answers to Review 10

1. DPF enables you to make more efficient use of system storage, I/O devices, and processing time.

2. a. *N*

   b. *N*

   c. *S*

   d. *S*

   e. *S* (Refer to Figures 10-2 and 10-3 for instances when a disk file can be shared.)

3. Two programs in DPF cannot write to the same file.

4. Basic Assembler, RPG II compiler, Library Maintenance, 1255 Utility Program.

5. By specifying in the *H* control cards how much storage should be used to execute the programs and specifying how much storage should be allocated to the second partition with the PARTITION OCL statement.

CHAPTER 11

CONTROLLING THE PERFORMANCE OF OPERATIONS IN AN RPG II PROGRAM

CHAPTER 11 DESCRIBES:

Dual input/output areas.

Subroutines.

Exception output (EXCPT operation code).

Halt, L0, and external indicators.

Look ahead feature.

Binary field operations (BITON, BITOF, TESTB).

BEFORE READING THIS CHAPTER YOU SHOULD BE ABLE TO DESCRIBE:

RPG II object cycle.

Object library.

Function of RPG II indicators, specifically L1-L9.

Looping (GOTO-TAG).

Multi-file processing.

Use of the SETON and SETOFF operation codes.

Use of *PLACE.

Overflow and fetch overflow.

Binary data.

AFTER READING THIS CHAPTER YOU SHOULD BE ABLE TO DESCRIBE:

Function and coding for dual input/output areas.

Function and coding for subroutines.

Function and coding for exception output (EXCPT operation code).

Halt, L0, and external indicators.

Look ahead feature.

Effects of exception output and look ahead on the RPG II object cycle.

Binary field operations (BITON, BITOF, TESTB).

INTRODUCTION

There are several ways that you as a programmer can control the performance of operations in an RPG II program. This chapter discusses six programming techniques which control operations:

1. Dual input/output areas may increase the speed of operations in an RPG II program.
2. Subroutines repeat operations in a program and eliminate duplicate coding.
3. Exception output (EXCPT operation code) repeats output operations during calculation time and eliminates duplicate coding.
4. Indicators can prevent certain operations from being performed and perform total time operations without a control break.
5. Binary field operations set and test bits in storage allowing you to control operations based on certain conditions that you specify.
6. The look ahead feature allows you to alter the order of operations based on the next record in a file.

INCREASING THE SPEED OF OPERATIONS (DUAL I/O AREAS)

During a normal RPG II cycle, a record is read, calculations are performed, and output (printed or punched) is produced. The cycle is repeated for each record.

The speed at which the cycle is done depends upon the speed at which records are read and output produced. Calculations take less time than reading, printing, or punching. Reading, printing, and punching can be speeded up by using dual input/output areas.

Dual Input Areas

When dual input areas are used, the program cycle is changed. First a record is read. At the same time, calculations are being performed on this record, another record is being read. Thus, the contents of two records are in the computer at the same time. Figure 11-1 shows how the records are processed when two input areas are used.

Dual input areas can be specified for sequential or direct input files. No stacker selection can be specified, nor can the input files be specified as combined or update files.

Dual input areas require more computer storage space than one input area, because two records are in storage during each cycle. If you have a large program, you might not have enough storage space to accomodate two input areas. If your program plus two input areas require more space than is available, certain RPG II object cycle routines remain on disk during execution and are called into storage as needed. If too many routines remain on disk, the performance of your program may be decreased.

The effect of dual input areas can be determined only if you have knowledge of a program's processing requirements and experience in RPG II programming. In some cases, you can only make a final determination by actual experiementation.

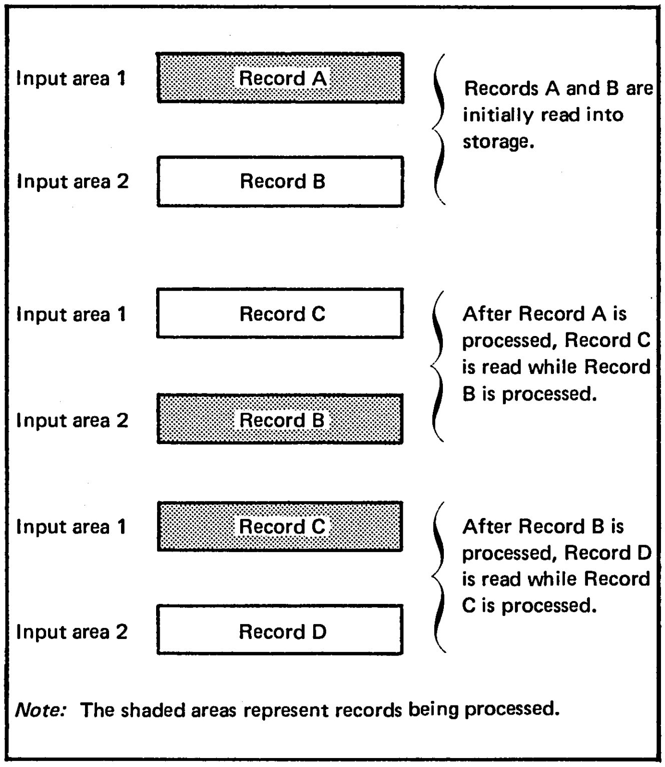

Input area 1
Record A
Input area 2
Record B
Records A and B are initially read into storage.
Input area 1
Record C
Input area 2
Record B
After Record A is processed, Record C is read while Record B is processed.
Input area 1
Record C
Input area 2
Record D
After Record B is processed, Record D is read while Record C is processed.
Note: The shaded areas represent records being processed.

Figure 11-1. Dual Input Areas

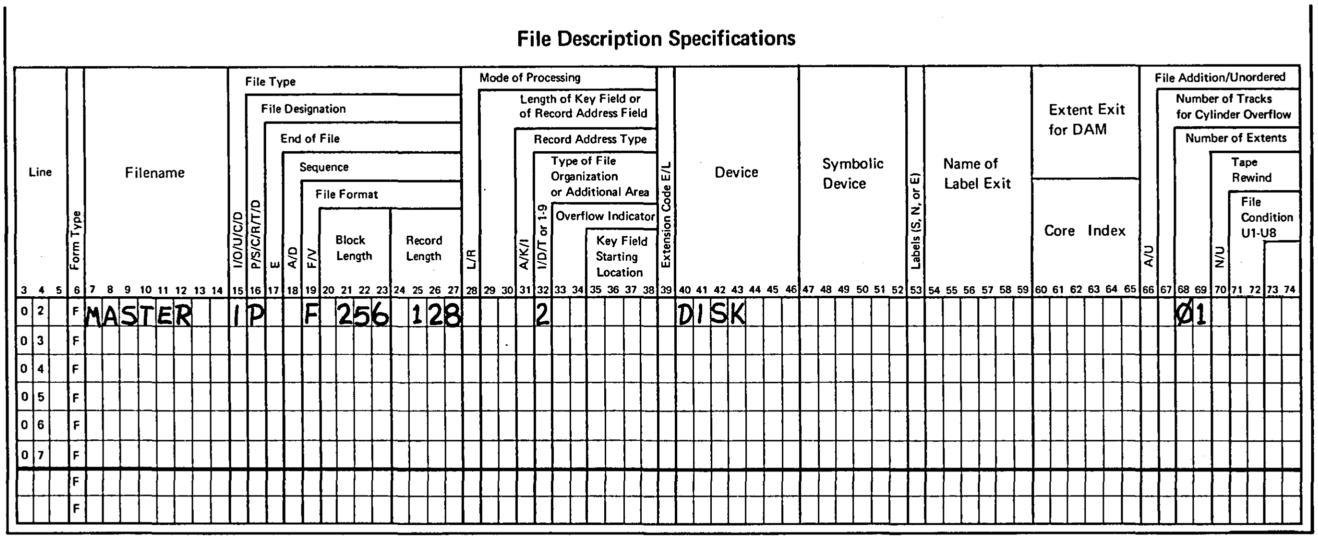
Specifications: One entry on the File Description Sheet is required to specify dual input areas; any digit 1-9 in column 32 assigns dual input areas for the specified file. Figure 11-2 shows the file MASTER has been assigned dual input areas.

Dual Output Areas

When dual output areas are used, the program cycle is changed. A record is either printed or punched at the same time calculation and output operations are being done to produce the next record. (Calculation operations are not done at the same time as punching or printing when only one output area is used.) Figure 11-3 shows how output records are produced using dual output areas.

Dual output areas, like dual input areas, require more computer storage. Consequently, the same space considerations that apply to dual input areas also apply to dual output areas. Dual output areas can only be used for sequential and direct files that do not have stacker selection entries, nor are specified as combined or update files.

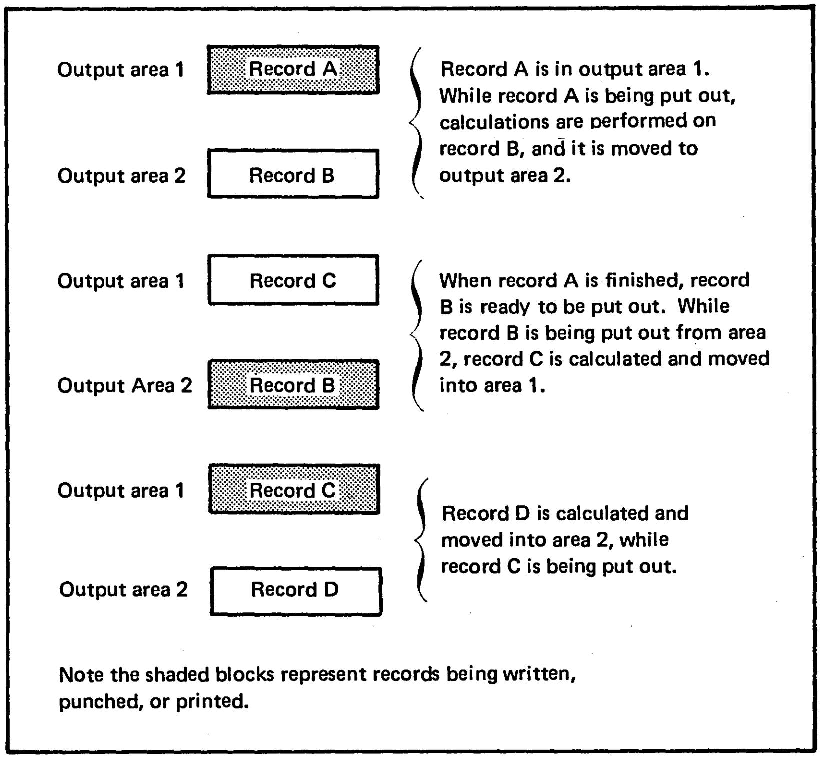
Specifications: One entry is required on the File Description Sheet to specify dual output areas, any digit 1-9 can be entered in column 32 for an output file. Figure 11-4 shows the file PRINT has been assigned dual output areas.



File Description Specifications

Line	Form Type	Filename	File Type I/O/U/C/D	File Designation P/S/C/R/T/D	End of File E	Sequence A/D	File Format F/V	Block Length	Record Length	Mode of Processing L/R	Length of Key Field or of Record Address Field	Record Address Type A/K/I	Type of File Organization or Additional Area I/D/T or 1-9	Overflow Indicator	Key Field Starting Location	Extension Code E/L	Device	Symbolic Device	Labels (S, N, or E)	Name of Label Exit	Extent Exit for DAM / Core Index	A/U	Number of Tracks for Cylinder Overflow	Number of Extents	Tape Rewind N/U	File Condition U1-U8
0 2	F	MASTER	I	P			F	256	128				2				DISK							Ø1		
0 3	F																									
0 4	F																									
0 5	F																									
0 6	F																									
0 7	F																									
	F																									
	F																									

Figure 11-2. Specifying a Dual Input Area



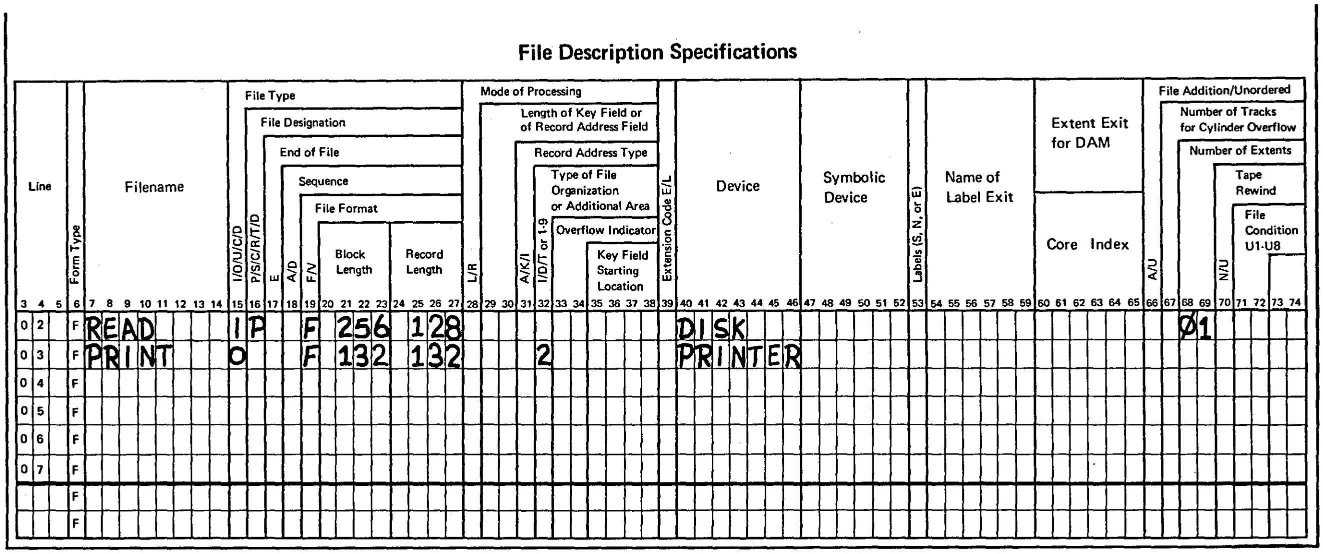
Record A is in output area 1. While record A is being put out, calculations are performed on record B, and it is moved to output area 2.

When record A is finished, record B is ready to be put out. While record B is being put out from area 2, record C is calculated and moved into area 1.

Record D is calculated and moved into area 2, while record C is being put out.

Note the shaded blocks represent records being written, punched, or printed.

Figure 11-3. Dual Output Areas



File Description Specifications

Line (3-5)	Form Type (6)	Filename (7-14)	File Type I/O/U/C/D (15)	File Designation P/S/C/R/T/D (16)	End of File E (17)	Sequence A/D (18)	File Format F/V (19)	Block Length (20-23)	Record Length (24-27)	L/R (28)	Mode of Processing: Length of Key Field or of Record Address Field (29-30)	Record Address Type A/K/I (31)	Type of File Organization or Additional Area I/D/T or 1-9 (32)	Overflow Indicator (33-34)	Key Field Starting Location (35-38)	Extension Code E/L (39)	Device (40-46)	Symbolic Device (47-52)	Labels (S, N, or E) (53)	Name of Label Exit (54-59)	Extent Exit for DAM / Core Index (60-65)	File Addition/Unordered A/U (66)	Number of Tracks for Cylinder Overflow (67)	Number of Extents (68-69)	Tape Rewind N/U (70)	File Condition U1-U8 (71-72)	(73-74)
02	F	READ	I	P			F	256	128								DISK							Ø1			
03	F	PRINT	O				F	132	132				2				PRINTER										
04	F																										
05	F																										
06	F																										
07	F																										
	F																										
	F																										

Figure 11-4. Specifying Dual Output Areas

USING SUBROUTINES TO CONTROL THE PROCESSING OF CALCULATIONS

You may want to specify parts of an RPG II program as subroutines. Subroutines can be used to:

- Reduce the storage requirements for RPG II programs. When a program exceeds available storage, certain RPG II object cycle routines remain on disk to be called in as needed. This is known as *overlay* and it could decrease performance if many routines remain on disk. Subroutines can be stored on disk in place of certain RPG II routines. By coding infrequently used routines as subroutines, you can control the way RPG II performs overlay. You must determine which subroutines should remain on disk. The compiler cannot determine which subroutines are infrequently used.

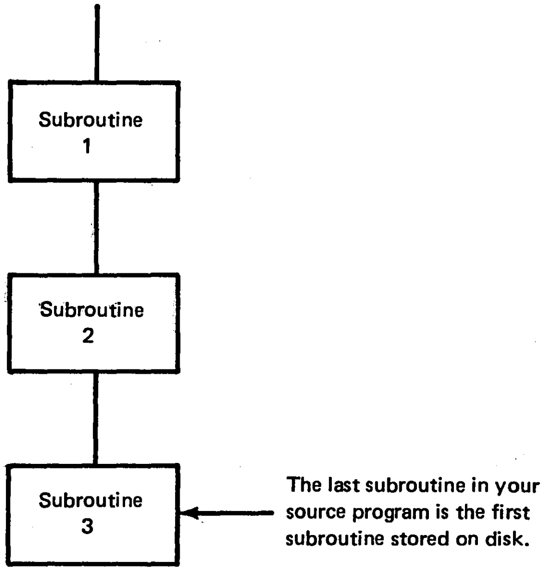
- Perform the same calculations several times during one cycle. This eliminates duplicate coding. Similarly, a subroutine can perform the same calculations in several different programs. For example, you can have a tax routine used by several invoicing jobs. By coding the routine as a subroutine, it needs to be coded and tested only once.

Controlling Overlay By Using Subroutines

By using subroutines, you can control the routines that the compiler stores on disk if overlay is necessary. You can have certain exception routines in a program, such as credit check, invalid part number, or invalid customer number, that are used less frequently than the object cycle routines the compiler stores on disk. By coding these exception routines as subroutines, the compiler can place them, instead of object cycle routines, on disk in the object library after compilation. Your main program is never entirely loaded into main storage at one time. Only as many object cycle routines or subroutines will be executed from disk as necessary.

The compiler gives priority to object cycle routines based on the normal expected frequency of use. Those routines that are seldom used are stored on disk before overflow routines.

You must give priority to subroutines to determine which subroutines, rather than object cycle routines, should be stored on disk. Those subroutines used infrequently should be the first routines stored in the object library. Priority is established through the order in which the subroutines appear at compilation time. The last subroutine in your source program will be the first subroutine stored in the object library. Consequently, you should place an infrequently used subroutine as the last subroutine in your source program:


Subroutine 1
Subroutine 2
Subroutine 3
The last subroutine in your source program is the first subroutine stored on disk.

Using Subroutines to Repeat the Same Calculations Several Times in One Cycle

In many programs, the same operation may be required several times in one cycle. When coding the job, you can specify the operations as many times as needed. This often involves large amounts of coding, however. If the same operations are done several times in succession, you can use loops (GOTO-TAG) to reduce the amount of coding.

If the same operations are not done several times in succession, but are performed at many different points in your program, creating a loop could not work. As an example, consider the job which creates a weekly Sales Commission Report. The report desired (Figure 11-5) shows two things:

1. Total commission earned by each salesman.

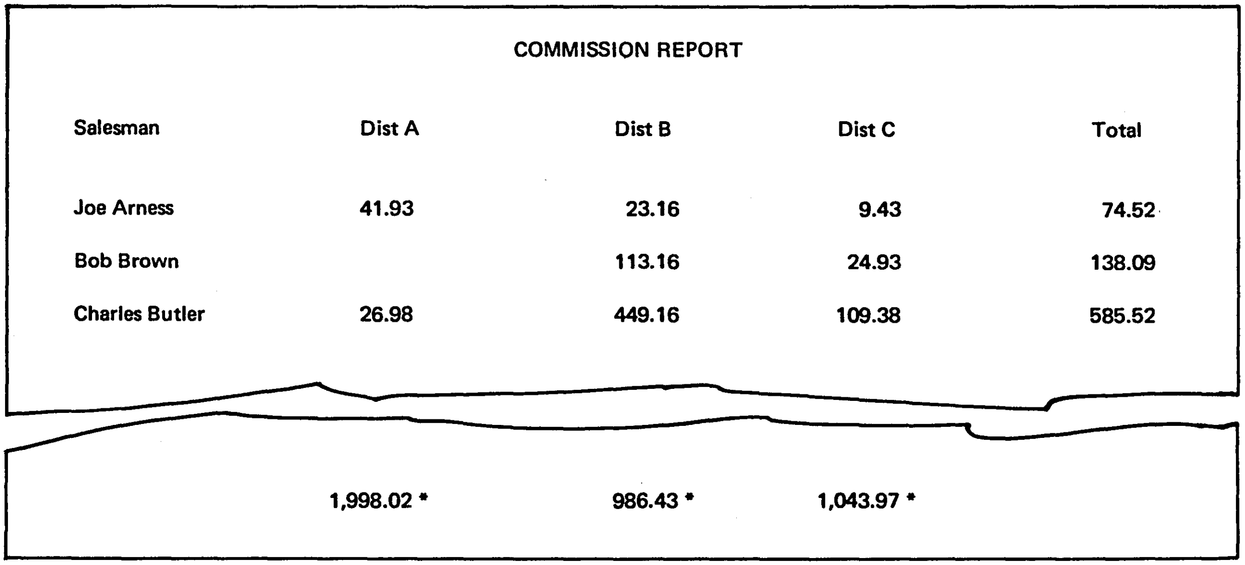
2. Total commission paid in each district.

The area in which all salesmen work is divided into three districts: A, B, and C. Some salesmen work in only one district while others can work in parts of two or more districts.

For each salesman, the input file contains a record formatted as shown in Figure 11-6. The amounts in the district fields show total weekly sales madė by that salesman in each district. If the salesman did not work a district or made no sales in that district, the field contains a zero.

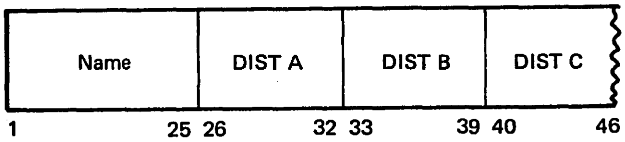
The report must contain the commission earned in each district by each salesman. In addition, total commission must be accumulated for each salesman and each district. The percentage of commission is:

- Three percent of the gross sales up to 1000.00 dollars
- Plus two percent of the gross sales between 1000.01 and 5000.00 dollars
- Plus one percent of the gross sales over 5000.00 dollars.



```
                              COMMISSION REPORT

Salesman              Dist A            Dist B          Dist C          Total

Joe Arness             41.93             23.16            9.43          74.52
Bob Brown                               113.16           24.93         138.09
Charles Butler         26.98            449.16          109.38         585.52


                    1,998.02 *          986.43 *      1,043.97 *
```

Figure 11-5. Sales Commission Report



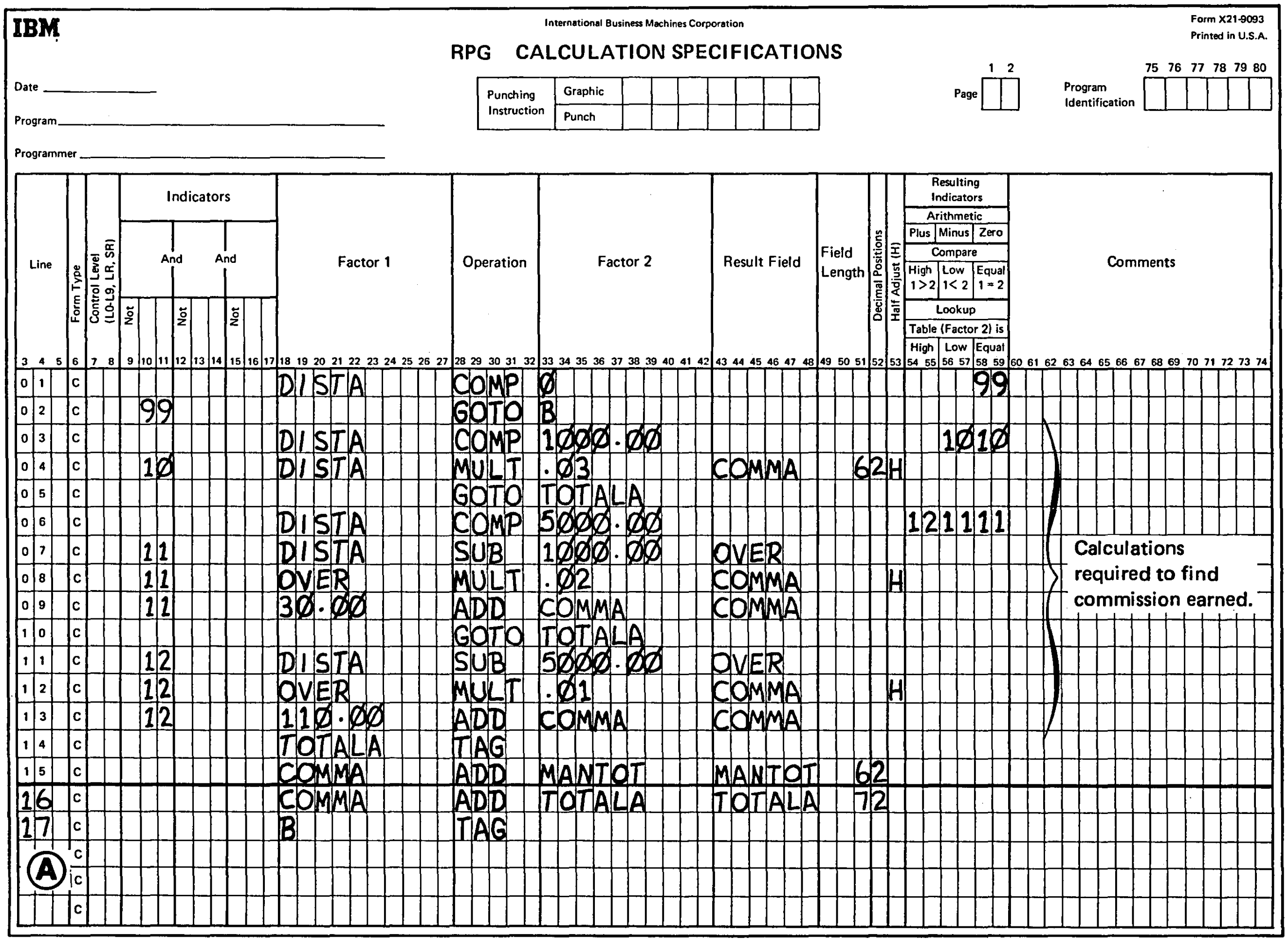
Name	DIST A	DIST B	DIST C
1 25	26 32	33 39	40 46

Figure 11-6. Input for the Sales Commission Report

Figure 11-7 shows the calculations needed to find the information required for the report. You first compare the contents of each district field to zero to find out if the salesman sold anything in that district. If it is not zero, you calculate the commission (COMM) earned. You then add commission earned to total commission for the salesman (MANTOT) and to total commission paid in each district (TOTALA, TOTALB, or TOTALC).

The calculations needed to find commission earned are the same for each district (Figure 11-7, inserts A, B, C, lines 3-16). Rather than coding these calculations three times, you can code them once and branch to them each time they are needed (Figure 11-8).

Using GOTO and TAG, you could easily branch to the calculations needed to find commission. But since you could branch to them from *three* different places, it would be difficult to determine where you should return. You could return to the point where totals are accumulated for district A, the point they are accumulated for district B, or the point they are accumulated for district C. The RPG II object program can return to the correct point in the calculations after a subroutine is used by establishing the necessary instructions to branch back to the main program.



IBM International Business Machines Corporation

RPG CALCULATION SPECIFICATIONS

Form X21-9093
Printed in U.S.A.

Date
Program
Programmer

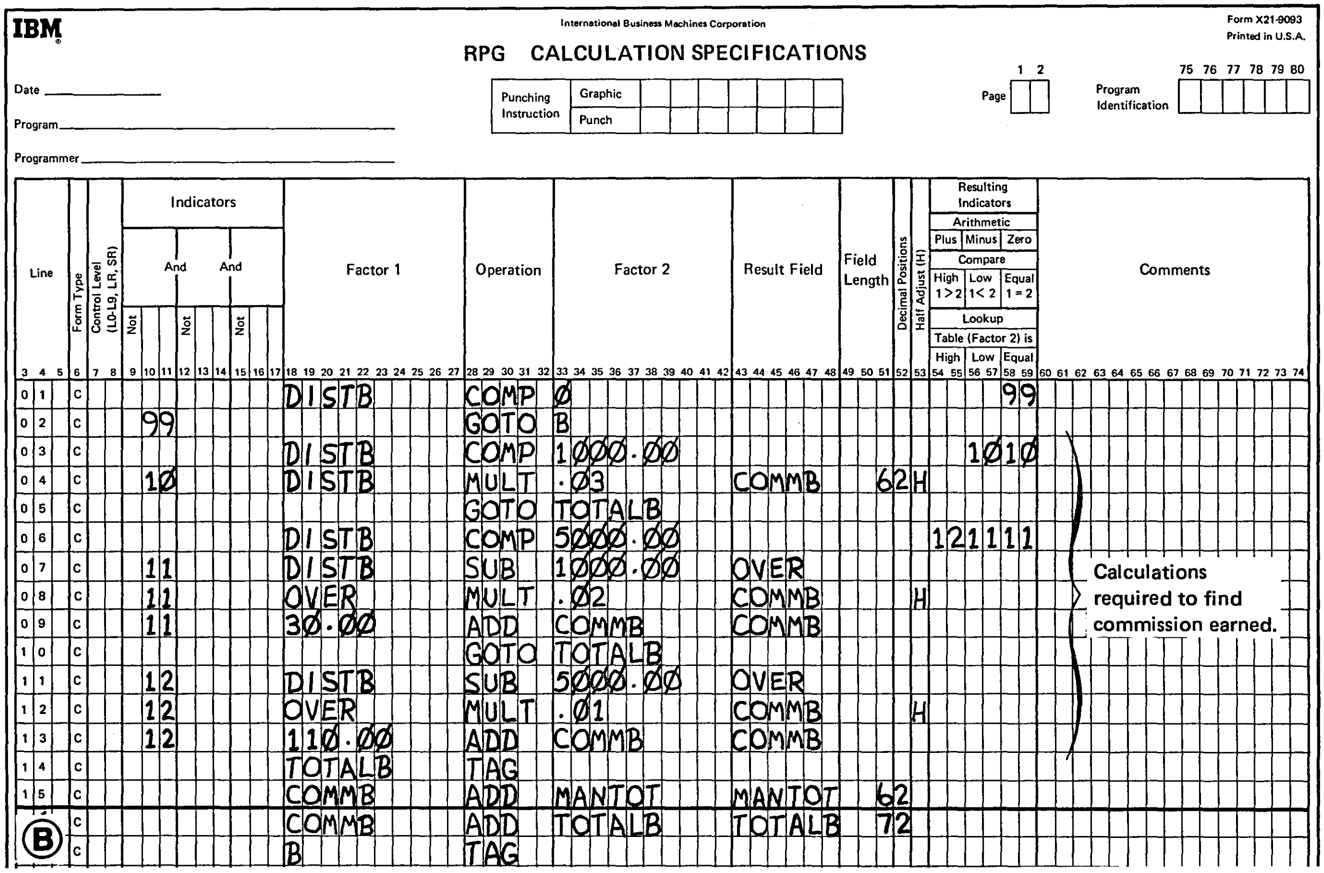
Punching Instruction: Graphic / Punch

Page (1 2)

Program Identification (75 76 77 78 79 80)

Line (3-5)	Form Type (6)	Control Level (7-8)	Indicators (9-11)	And (12-14)	And (15-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field (43-48)	Field Length (49-51)	Decimal Positions (52)	Half Adjust (H) (53)	Resulting Indicators High 1>2 (54-55)	Low 1<2 (56-57)	Equal 1=2 (58-59)	Comments (60-74)
01	C					DISTA	COMP	Ø							99	
02	C		99				GOTO	B								
03	C					DISTA	COMP	1ØØØ.ØØ						1Ø	1Ø	
04	C		1Ø			DISTA	MULT	.Ø3	COMMA	6	2	H				
05	C						GOTO	TOTALA								
06	C					DISTA	COMP	5ØØØ.ØØ					12	11	11	
07	C		11			DISTA	SUB	1ØØØ.ØØ	OVER							Calculations
08	C		11			OVER	MULT	.Ø2	COMMA			H				required to find
09	C		11			3Ø.ØØ	ADD	COMMA	COMMA							commission earned.
10	C						GOTO	TOTALA								
11	C		12			DISTA	SUB	5ØØØ.ØØ	OVER							
12	C		12			OVER	MULT	.Ø1	COMMA			H				
13	C		12			11Ø.ØØ	ADD	COMMA	COMMA							
14	C					TOTALA	TAG									
15	C					COMMA	ADD	MANTOT	MANTOT	6	2					
16	C					COMMA	ADD	TOTALA	TOTALA	7	2					
17	C					B	TAG									
	C															
Ⓐ	C															
	C															

Figure 11-7. Calculations for Sales Commission Job (part 1 of 2)



IBM

International Business Machines Corporation

RPG CALCULATION SPECIFICATIONS

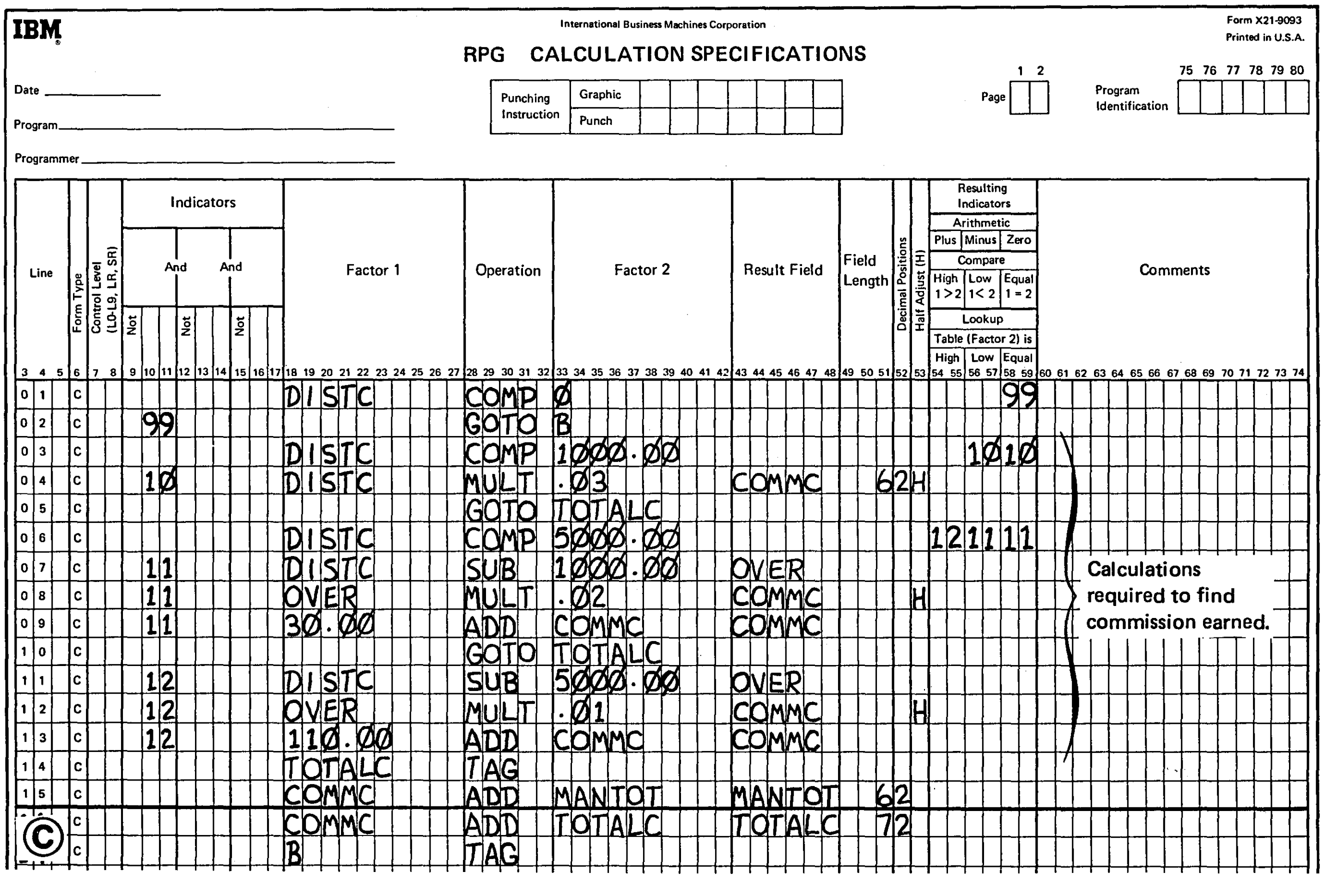
Form X21-9093
Printed in U.S.A.

Date ______ Program ______ Programmer ______

Punching Instruction: Graphic | Punch

Page 1 2 — Program Identification 75 76 77 78 79 80

Line	Form Type	Control Level (L0-L9, LR, SR)	Not	Ind.	Not	Ind.	Not	Ind.	Factor 1	Operation	Factor 2	Result Field	Field Length	Decimal Positions	Half Adjust (H)	Plus / High 1>2	Minus / Low 1<2	Zero / Equal 1=2	Comments
01	C								DISTB	COMP	Ø							99	
02	C			99						GOTO	B								
03	C								DISTB	COMP	1ØØØ.ØØ						1Ø	1Ø	
04	C			1Ø					DISTB	MULT	.Ø3	COMMB	6	2	H				
05	C									GOTO	TOTALB								
06	C								DISTB	COMP	5ØØØ.ØØ					12	11	11	
07	C			11					DISTB	SUB	1ØØØ.ØØ	OVER							Calculations
08	C			11					OVER	MULT	.Ø2	COMMB			H				required to find
09	C			11					3Ø.ØØ	ADD	COMMB	COMMB							commission earned.
10	C									GOTO	TOTALB								
11	C			12					DISTB	SUB	5ØØØ.ØØ	OVER							
12	C			12					OVER	MULT	.Ø1	COMMB			H				
13	C			12					11Ø.ØØ	ADD	COMMB	COMMB							
14	C								TOTALB	TAG									
15	C								COMMB	ADD	MANTOT	MANTOT	6	2					
B	C								COMMB	ADD	TOTALB	TOTALB	7	2					
	C								B	TAG									



IBM

International Business Machines Corporation

RPG CALCULATION SPECIFICATIONS

Form X21-9093
Printed in U.S.A.

Date ______ Program ______ Programmer ______

Punching Instruction: Graphic | Punch

Page 1 2 — Program Identification 75 76 77 78 79 80

Line	Form Type	Control Level (L0-L9, LR, SR)	Not	Ind.	Not	Ind.	Not	Ind.	Factor 1	Operation	Factor 2	Result Field	Field Length	Decimal Positions	Half Adjust (H)	Plus / High 1>2	Minus / Low 1<2	Zero / Equal 1=2	Comments
01	C								DISTC	COMP	Ø							99	
02	C			99						GOTO	B								
03	C								DISTC	COMP	1ØØØ.ØØ						1Ø	1Ø	
04	C			1Ø					DISTC	MULT	.Ø3	COMMC	6	2	H				
05	C									GOTO	TOTALC								
06	C								DISTC	COMP	5ØØØ.ØØ					12	11	11	
07	C			11					DISTC	SUB	1ØØØ.ØØ	OVER							Calculations
08	C			11					OVER	MULT	.Ø2	COMMC			H				required to find
09	C			11					3Ø.ØØ	ADD	COMMC	COMMC							commission earned.
10	C									GOTO	TOTALC								
11	C			12					DISTC	SUB	5ØØØ.ØØ	OVER							
12	C			12					OVER	MULT	.Ø1	COMMC			H				
13	C			12					11Ø.ØØ	ADD	COMMC	COMMC							
14	C								TOTALC	TAG									
15	C								COMMC	ADD	MANTOT	MANTOT	6	2					
C	C								COMMC	ADD	TOTALC	TOTALC	7	2					
	C								B	TAG									

Figure 11-7. Calculations for Sales Commission Job (part 2 of 2)

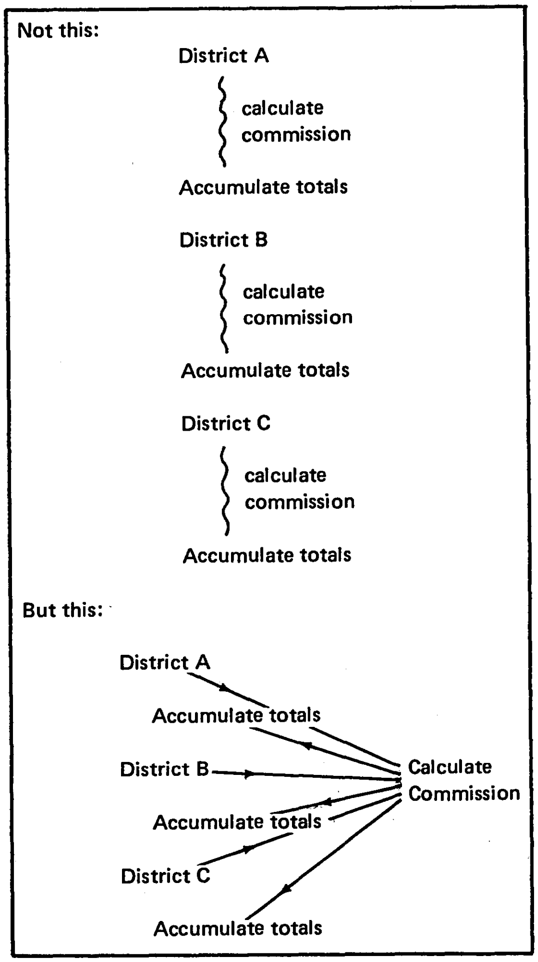
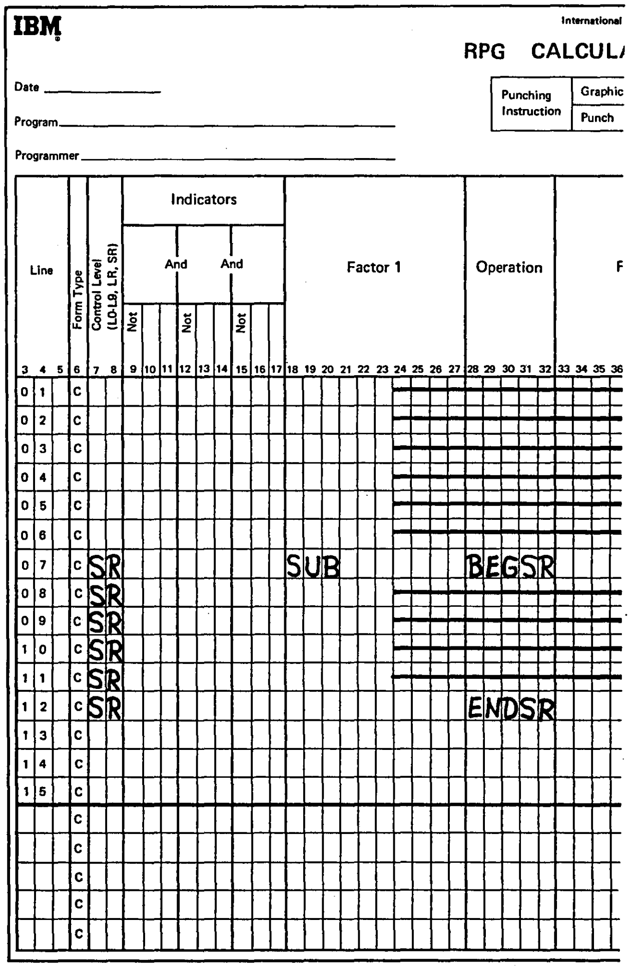


Figure 11-8. Branching to Similar Calculations


IBM International

RPG CALCUL

Date

Program

Programmer

Punching Instruction	Graphic
	Punch

Line	Form Type	Control Level (L0-L9, LR, SR)	Indicators: Not		And: Not		And: Not		Factor 1	Operation	F
3 4 5	6	7 8	9	10 11	12	13 14	15	16 17	18 19 20 21 22 23 24 25 26 27	28 29 30 31 32	33 34 35 36
0 1	C										
0 2	C										
0 3	C										
0 4	C										
0 5	C										
0 6	C										
0 7	C	SR							SUB	BEGSR	
0 8	C	SR									
0 9	C	SR									
1 0	C	SR									
1 1	C	SR									
1 2	C	SR								ENDSR	
1 3	C										
1 4	C										
1 5	C										
	C										
	C										
	C										
	C										
	C										

Figure 11-9. Structure of a Subroutine

Specifications for Coding A Subroutine

You specify subroutines on the Calculation Sheet after all detail and total operations. Every statement in the subroutine must be identified as part of the subroutine by the letters SR in columns 7-8 (Figure 11-9). In addition, the operation codes BEGSR and ENDSR must be coded to establish the beginning and end of the subroutine.

The name of each subroutine must appear in factor 1 on the same line as the BEGSR operation code (Figure 11-9). Every subroutine used in the program must have a unique name. The rules for establishing a subroutine name are the same as those for forming a field name.

Calling the Subroutine

When using GOTO and TAG, you use a GOTO operation code to branch to the next operation to be performed. When you do the operations in a subroutine, you do not branch to the subroutine; you *call* it.

When you call a subroutine, you use the execute subroutine (EXSR) operation code. This operation code can be placed anywhere in the calculation operations. Whenever the EXSR operation code is encountered, all operations in the subroutine will be performed. After the subroutine has been executed, RPG II branches back to the main program and continues execution with the next statement after the EXSR statement (Figure 11-10).

Fields Used in a Subroutine

The same fields can be used by both the subroutine and the main routine. You may define the field in either routine. However, the name and characteristics of the field must be the same in both routines.

The fields you define in a subroutine should be general so that they apply to all situations for which a subroutine is used. For example, if DISTA is used as the field name in a subroutine to calculate district sales, you *always* take information from the DISTA field when calculating commission. However, you want the routine also to handle information from the fields DISTB and DISTC. Using specific fields limits the correct use of a subroutine to one situation.

Instead, if you use a general field name such as SALES, this one subroutine can be used to calculate commission in all three districts (Figure 11-11, insert C). However, because there is no input field called SALES, you must use the Z-ADD operation code to place information in this field (Figure 11-11, insert B). The information in the appropriate district field (DISTA, DISTB, or DISTC) is moved into the field called SALES before the subroutine is executed. When finding commission earned in district A, DISTA is moved into SALES; when finding commission earned for district B, DISTB is moved into SALES, etc. In this way, you ensure that the subroutine uses the correct information each time it is called.

Using Subroutines in the Sales Commission Report Example

Now that you have learned how subroutines are used, defined, and executed, see how they are used in the Sales Commission Report job. All specifications are shown in Figure 11-11.

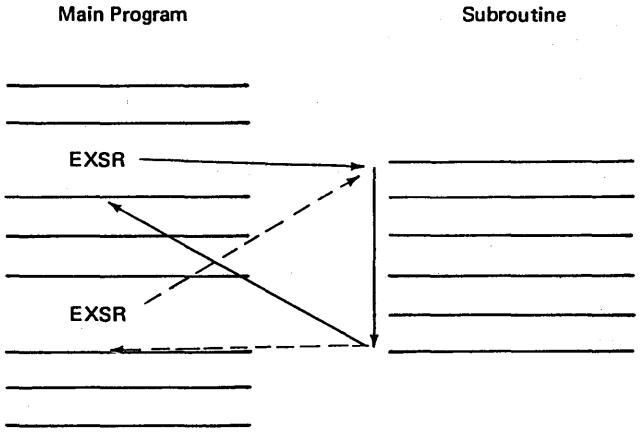
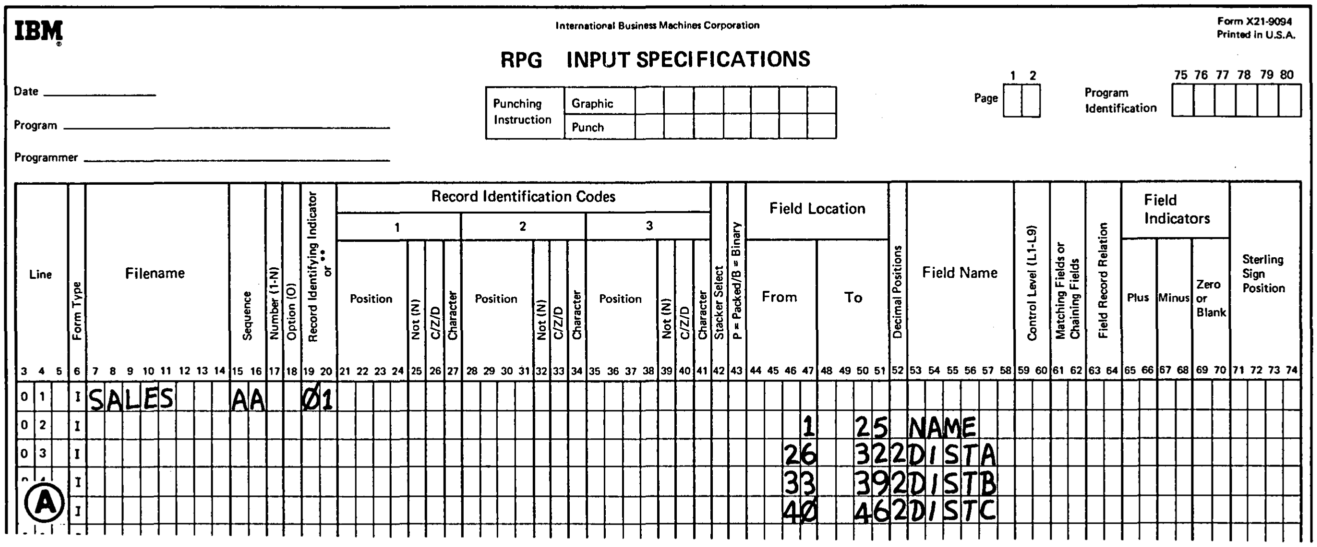


Figure 11-10. EXSR (Order in Which Calculations are Performed)

First a record is read. Now commission earned in each district must be calculated.

1. DISTA is compared to zero to see if the salesman sold anything in that district. If the field is greater than zero, commission must be calculated. If the field is zero, a branch is taken to B, where another comparison is made.
2. Before the subroutine can be called, it must be supplied with the correct amount of sales. Thus, the contents of DISTA are moved into SALES.
3. The subroutine is called by the EXSR operation code.
4. The commission is calculated by operations specified in the subroutine.
5. The subroutine is finished when ENDSR statement is executed. The instruction following EXSR is executed. The commission found by the subroutine is added to the total commission earned by the salesman (MANTOT) and to the total commission paid in the district (TOTALA).
6. Now DISTB is compared to zero to see if commission earned should be calculated. If so, information from the field DISTB is moved to SALES, and the subroutine is called. The next steps are basically the same as those already described. Follow the rest of the job.



IBM

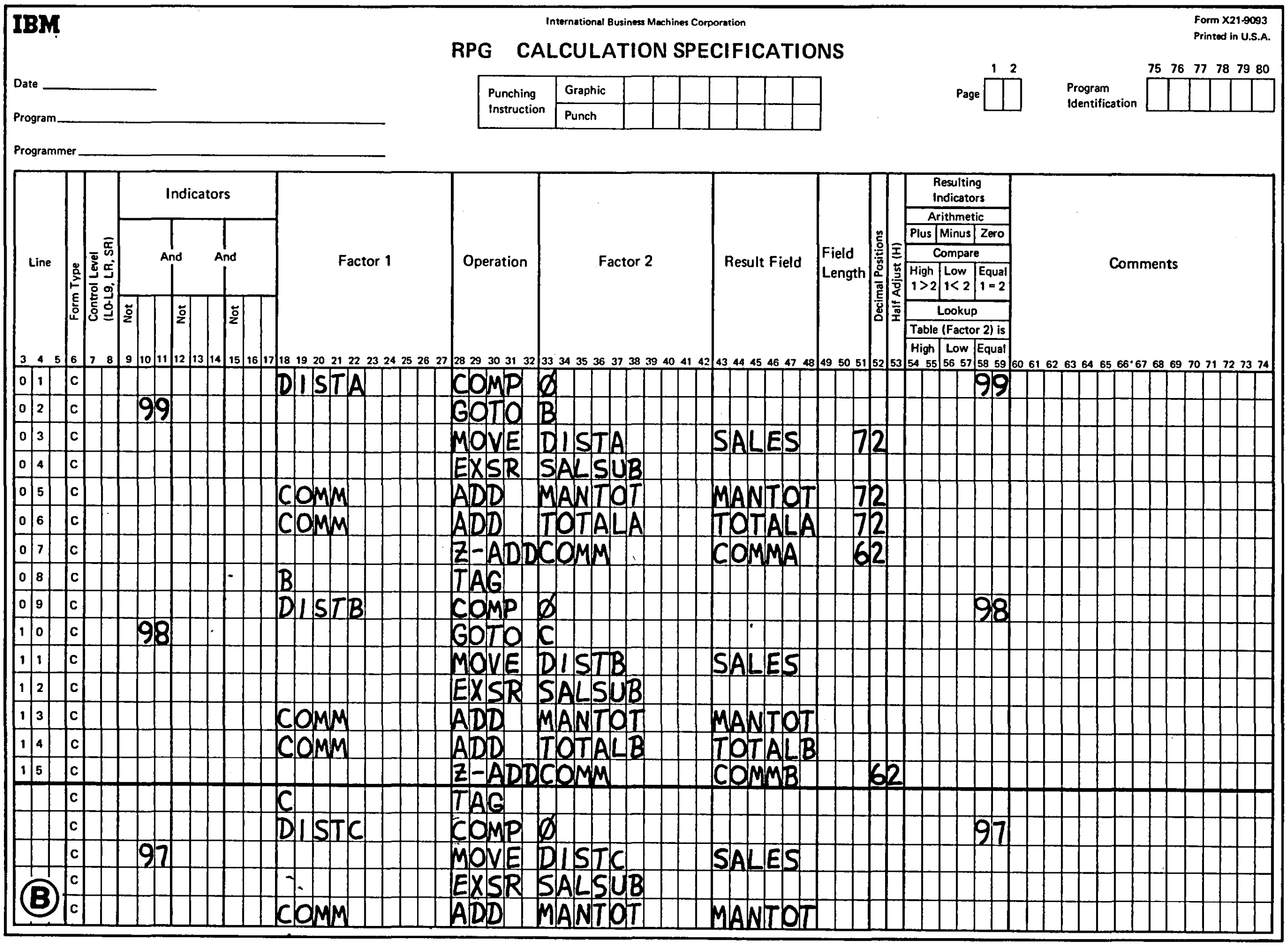
International Business Machines Corporation

RPG INPUT SPECIFICATIONS

Form X21-9094
Printed in U.S.A.

Date
Program
Programmer

Line	Form Type	Filename	Sequence	Record Identifying Indicator	Field Location From	Field Location To	Decimal Positions	Field Name
01	I	SALES	AA	Ø1				
02	I				1	25		NAME
03	I				26	32	2	DISTA
	I				33	39	2	DISTB
A	I				40	46	2	DISTC



IBM

International Business Machines Corporation

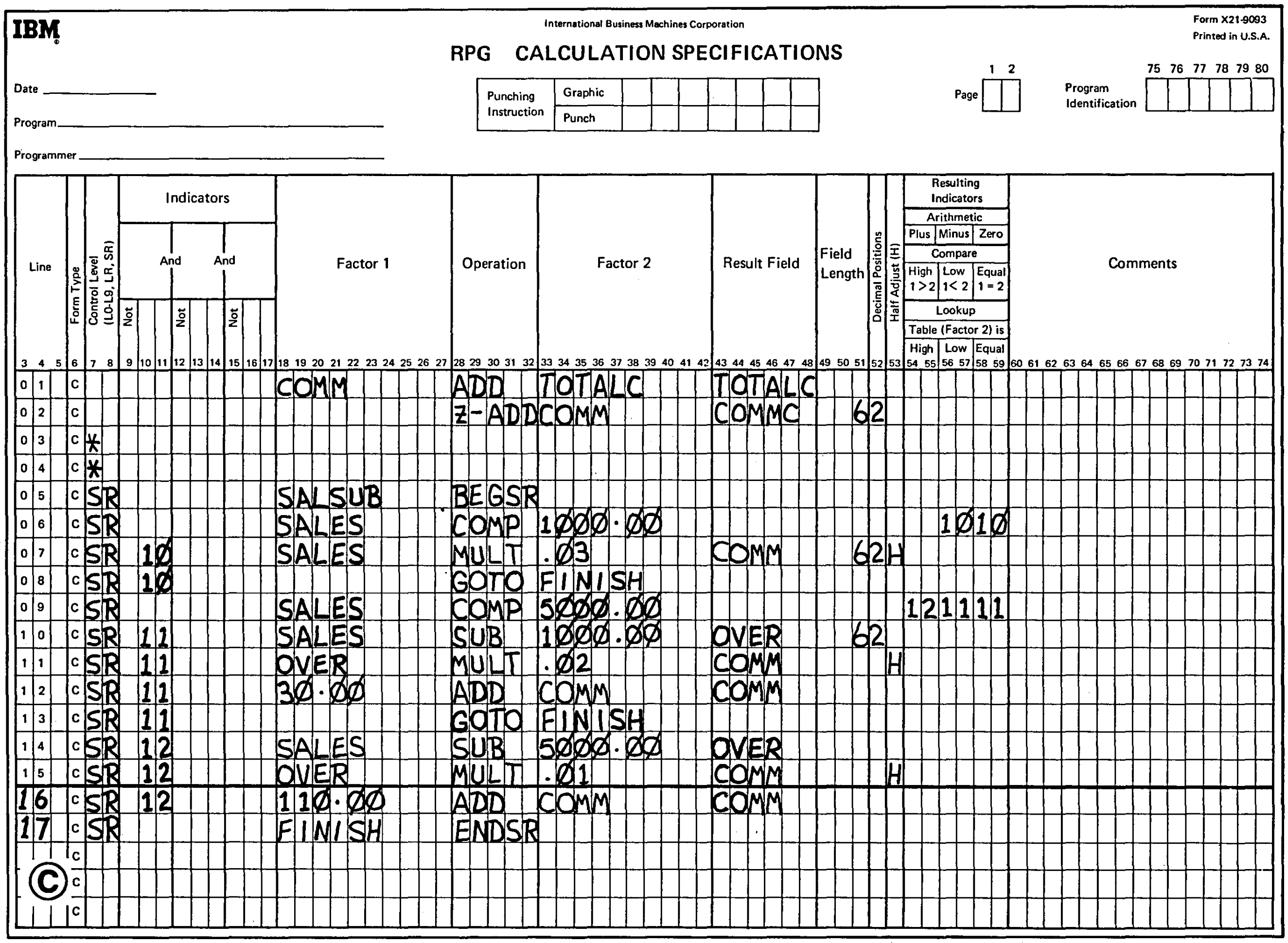
RPG CALCULATION SPECIFICATIONS

Form X21-9093
Printed in U.S.A.

Date
Program
Programmer

Line	Form Type	Indicators	Factor 1	Operation	Factor 2	Result Field	Field Length	Decimal Positions	Resulting Indicators Equal 1=2
01	C		DISTA	COMP	Ø				99
02	C	99		GOTO	B				
03	C			MOVE	DISTA	SALES	7	2	
04	C			EXSR	SALSUB				
05	C		COMM	ADD	MANTOT	MANTOT	7	2	
06	C		COMM	ADD	TOTALA	TOTALA	7	2	
07	C			Z-ADD	COMM	COMMA	6	2	
08	C		B	TAG					
09	C		DISTB	COMP	Ø				98
10	C	98		GOTO	C				
11	C			MOVE	DISTB	SALES			
12	C			EXSR	SALSUB				
13	C		COMM	ADD	MANTOT	MANTOT			
14	C		COMM	ADD	TOTALB	TOTALB			
15	C			Z-ADD	COMM	COMMB	6	2	
	C		C	TAG					
	C		DISTC	COMP	Ø				97
	C	97		MOVE	DISTC	SALES			
	C			EXSR	SALSUB				
B	C		COMM	ADD	MANTOT	MANTOT			

Figure 11-11. Sales Commission Job Using a Subroutine (part 1 of 3)



IBM International Business Machines Corporation

Form X21-9093
Printed in U.S.A.

RPG CALCULATION SPECIFICATIONS

Date ____________

Program ____________

Programmer ____________

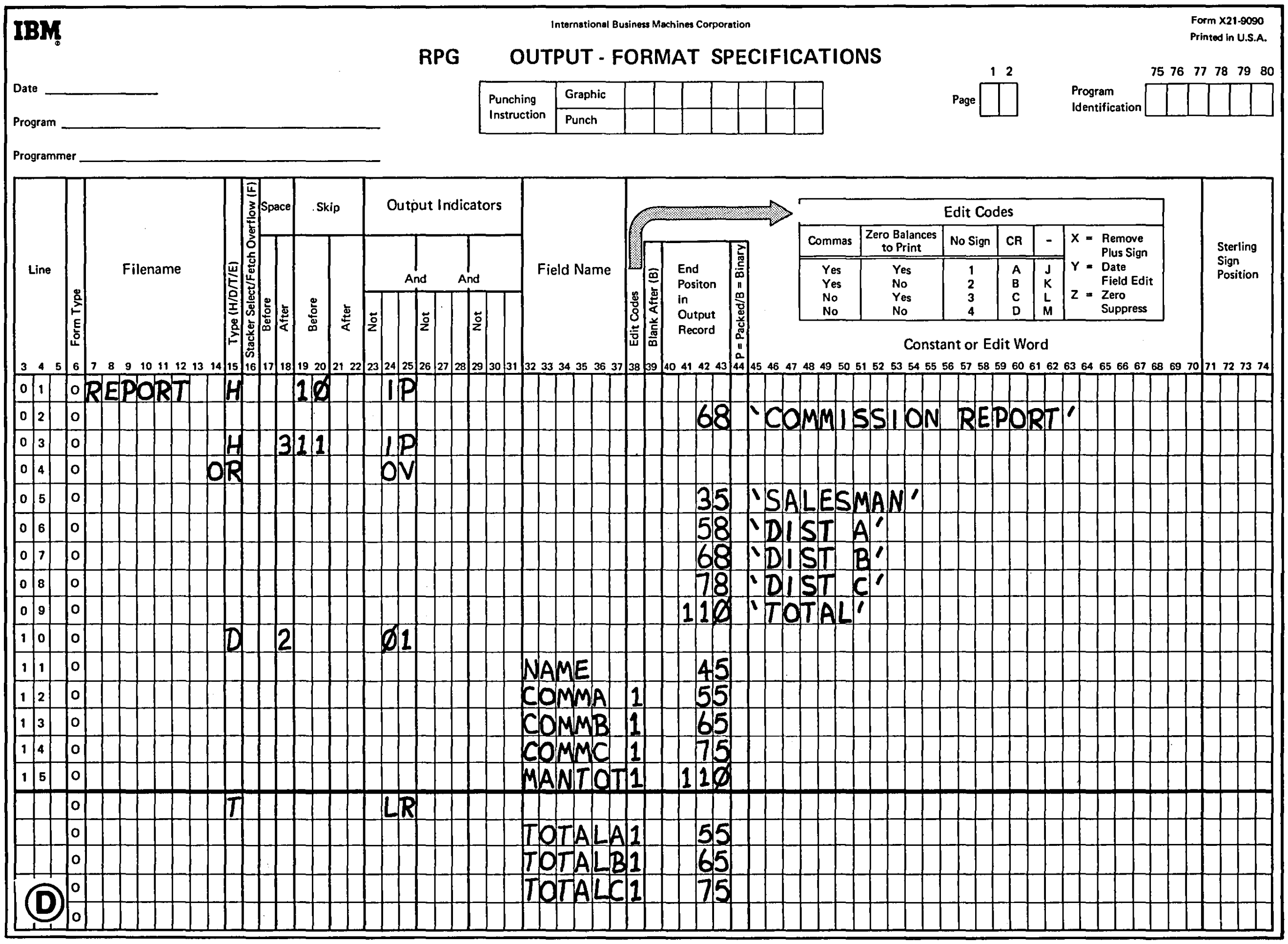
Punching Instruction: Graphic / Punch

Page 1 2

Program Identification 75 76 77 78 79 80

Line (3-5)	Form Type (6)	Control Level (L0-L9, LR, SR) (7-8)	Indicators (9-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field (43-48)	Field Length (49-51)	Decimal Positions (52)	Half Adjust (H) (53)	Resulting Indicators: Plus / High 1>2 (54-55)	Minus / Low 1<2 (56-57)	Zero / Equal 1=2 (58-59)	Comments (60-74)
01	C			COMM	ADD	TOTALC	TOTALC							
02	C				Z-ADD	COMM	COMMC	6	2					
03	C	*												
04	C	*												
05	C	SR		SALSUB	BEGSR									
06	C	SR		SALES	COMP	1ØØØ.ØØ						1Ø	1Ø	
07	C	SR	1Ø	SALES	MULT	.Ø3	COMM	6	2	H				
08	C	SR	1Ø		GOTO	FINISH								
09	C	SR		SALES	COMP	5ØØØ.ØØ					12	11	11	
10	C	SR	11	SALES	SUB	1ØØØ.ØØ	OVER	6	2					
11	C	SR	11	OVER	MULT	.Ø2	COMM			H				
12	C	SR	11	3Ø.ØØ	ADD	COMM	COMM							
13	C	SR	11		GOTO	FINISH								
14	C	SR	12	SALES	SUB	5ØØØ.ØØ	OVER							
15	C	SR	12	OVER	MULT	.Ø1	COMM			H				
16	C	SR	12	11Ø.ØØ	ADD	COMM	COMM							
17	C	SR		FINISH	ENDSR									
	C													
	C													
	C													

Figure 11-11. Sales Commission Job Using a Subroutine (part 2 of 3)



IBM

International Business Machines Corporation

Form X21-9090
Printed in U.S.A.

RPG OUTPUT - FORMAT SPECIFICATIONS

Date

Program

Programmer

Punching Instruction: Graphic / Punch

Page (1 2)

Program Identification (75 76 77 78 79 80)

Edit Codes	Commas	Zero Balances to Print	No Sign	CR	-
	Yes	Yes	1	A	J
	Yes	No	2	B	K
	No	Yes	3	C	L
	No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Line (3-5)	Form Type (6)	Filename (7-14)	Type (H/D/T/E) (15)	Space After (18)	Skip Before (19-20)	Output Indicators (23-31)	Field Name (32-37)	Edit Codes (38)	End Position in Output Record (40-43)	Constant or Edit Word (45-70)
01	O	REPORT	H		1Ø	IP				
02	O								68	'COMMISSION REPORT'
03	O		H	3	11	IP				
04	O	OR				OV				
05	O								35	'SALESMAN'
06	O								58	'DIST A'
07	O								68	'DIST B'
08	O								78	'DIST C'
09	O								11Ø	'TOTAL'
10	O		D	2		Ø1				
11	O						NAME		45	
12	O						COMMA	1	55	
13	O						COMMB	1	65	
14	O						COMMC	1	75	
15	O						MANTOT	1	11Ø	
	O		T			LR				
	O						TOTALA	1	55	
	O						TOTALB	1	65	
D	O						TOTALC	1	75	
	O									

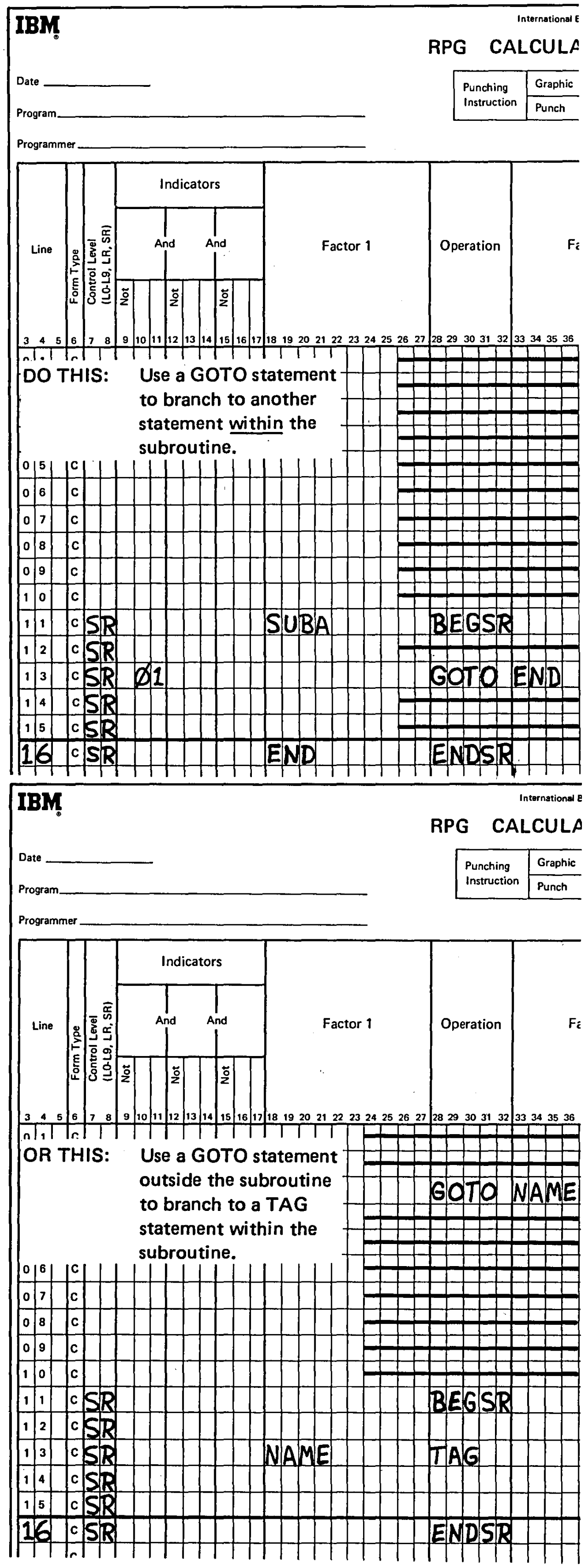
Figure 11-11. Sales Commission Job Using a Subroutine (part 3 of 3)

Using Valid Subroutine Operations

Any operation code that can be used in calculations can be used in a subroutine except BEGSR and ENDSR. This means that you can use all arithmetic, compare and testing, move look-up, EXSR, and branching operations.

There are limitations on some of the operations:

1. You may only branch to another statement *in the subroutine* when using the GOTO statement (Figure 11-12).
2. You may branch to the ENDSR statement if you put a name in Factor 1 of the ENDSR statement.
3. You may not branch to a statement outside of the subroutine.
4. You can not branch to a TAG within the subroutine from a GOTO outside of the subroutine (Figure 11-12).
5. You can not have a subroutine coded within another subroutine. However, one subroutine can call another subroutine. This means that within one subroutine you may have an EXSR statement (Figure 11-13). A subroutine, however, cannot call its caller.



IBM International B

RPG CALCULA

Date

Program

Programmer

Punching Instruction | Graphic | Punch

DO THIS: Use a GOTO statement to branch to another statement within the subroutine.

Line	Form Type	Control Level (L0-L9, LR, SR)	Indicators	Factor 1	Operation	Fa
05	C					
06	C					
07	C					
08	C					
09	C					
10	C					
11	C	SR		SUBA	BEGSR	
12	C	SR				
13	C	SR	Ø1		GOTO	END
14	C	SR				
15	C	SR				
16	C	SR		END	ENDSR	

IBM International B

RPG CALCULA

Date

Program

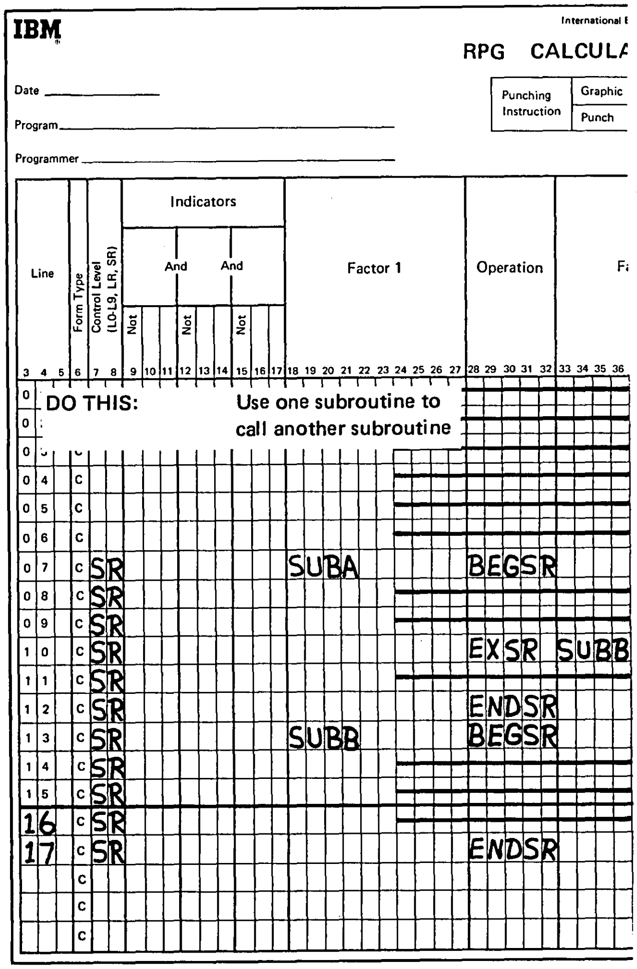
Programmer

Punching Instruction | Graphic | Punch

OR THIS: Use a GOTO statement outside the subroutine to branch to a TAG statement within the subroutine.

Line	Form Type	Control Level (L0-L9, LR, SR)	Indicators	Factor 1	Operation	Fa
					GOTO	NAME
06	C					
07	C					
08	C					
09	C					
10	C					
11	C	SR			BEGSR	
12	C	SR				
13	C	SR		NAME	TAG	
14	C	SR				
15	C	SR				
16	C	SR			ENDSR	

Figure 11-12. Branching Within a Subroutine



IBM International B

RPG CALCULA

Date

Program

Programmer

Punching Instruction | Graphic | Punch

DO THIS: Use one subroutine to call another subroutine

Line	Form Type	Control Level (L0-L9, LR, SR)	Indicators	Factor 1	Operation	Fa
04	C					
05	C					
06	C					
07	C	SR		SUBA	BEGSR	
08	C	SR				
09	C	SR				
10	C	SR			EXSR	SUBB
11	C	SR				
12	C	SR			ENDSR	
13	C	SR		SUBB	BEGSR	
14	C	SR				
15	C	SR				
16	C	SR				
17	C	SR			ENDSR	
	C					
	C					
	C					

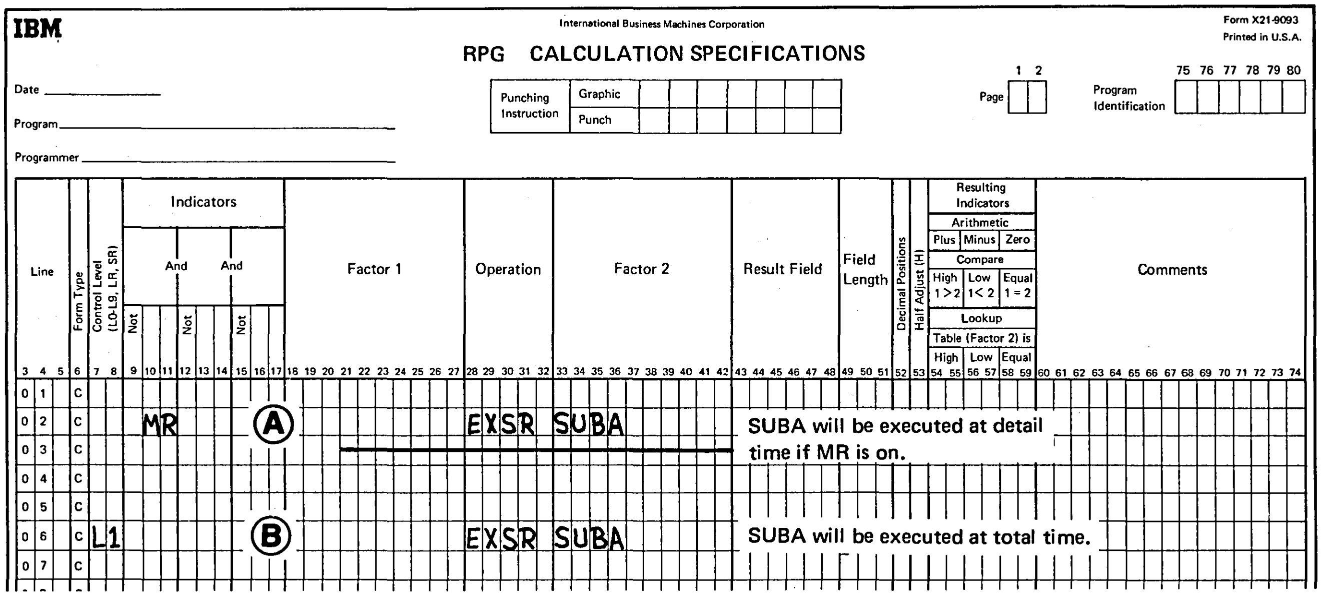
Figure 11-13. Using EXSR Within a Subroutine

Conditioning Subroutine Statements

Any indicator which is valid in columns 9-17 can be used to condition an operation within the subroutine. That operation will then be performed only when the conditions established by the indicators are satisfied. The BEGSR and ENDSR operation code, however, cannot be conditioned by any indicators.

The EXSR statement can also be conditioned by indicators. In this case, the *entire* subroutine will be performed only when conditions for the EXSR statement are met. For example, in Figure 11-14, insert A, the subroutine will be performed only if MR is on.

Control level indicators cannot be used to condition statements within a subroutine since SR must appear in columns 7-8. The indicators used on the EXSR statement determine whether the *entire* subroutine is performed at detail time or at total time (Figure 11-14, insert B).



IBM

International Business Machines Corporation

RPG CALCULATION SPECIFICATIONS

Form X21-9093
Printed in U.S.A.

Date
Program
Programmer

Punching Instruction: Graphic / Punch

Page 1 2

Program Identification 75 76 77 78 79 80

Line (3-5)	Form Type (6)	Control Level (L0-L9, LR, SR) (7-8)	Indicators (9-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field (43-48)	Field Length (49-51)	Decimal Positions (52)	Half Adjust (H) (53)	Resulting Indicators (54-59)	Comments (60-74)
01	C											
02	C		MR Ⓐ		EXSR	SUBA	SUBA will be executed at detail					
03	C						time if MR is on.					
04	C											
05	C											
06	C	L1	Ⓑ		EXSR	SUBA	SUBA will be executed at total time.					
07	C											

Figure 11-14. Conditioning Calculations Within a Subroutine

Review 11. Subroutines

1. When should a subroutine be used?

2. What are the operations used to define and execute a subroutine? What entry must be made for each calculation operation of a subroutine that is different from all other calculations?

3. What limitations in the use of GOTO and TAG apply to subroutines?

4. Where must subroutines be coded?

Answers to Review 11. Subroutines

1. A subroutine can be used whenever the same calculations must be executed at several different places in a program or when it is desired to control the number of overlays within your program.

2. The first line of a subroutine must have the BEGSR operation code in columns 28-32 with the subroutine name in factor 2. The last line in a subroutine must have ENDSR operation code in columns 28-32. This line can have a name in factor 1, and this name can then be referenced by a GOTO statement. Every subroutine operation code must have SR in columns 7-8.

3. No branches can be made from a GOTO statement within a subroutine within a subroutine to a TAG statement outside that subroutine. No branches can be made from outside the subroutine to a TAG statement within the subroutine.

4. All subroutines must appear on the Calculation Sheet after all detail and total calculations.

REPETITIVE OUTPUT (EXCPT OPERATION)

RPG II has a special operation code called EXCPT which allows you to write or punch as many records as are required during one program cycle.

Normally a record is written or punched at either detail or total *output* time. Using EXCPT, records can be put out during detail or total *calculation* time. Each time you use the operation code EXCPT, specified records are written immediately. For example, if you use eight EXCPT operation codes in succession, you can get an exception output cycle eight times. The records are identical if the data fields in the exception records are not altered between the EXCPT operation codes on the Calculation Sheet.

When you use the EXCPT operation code, you also must specify which records are to be put out during calculation time. These records are identified by an *E* in column 15 of the Output-Format Sheet. Only those output lines identified by an *E* will be put out during an exception output cycle.

Using EXCPT and *PLACE

The reserved word *PLACE duplicates fields and places them on the same line. In the discussion of *PLACE in Chapter 13, an example is used in which three mailing labels|were printed for each customer using *PLACE. If you wanted to print 15 labels for each customer, however, you could not use only the reserved word *PLACE. The only way would be to print the same three mailing labels five times in succession.

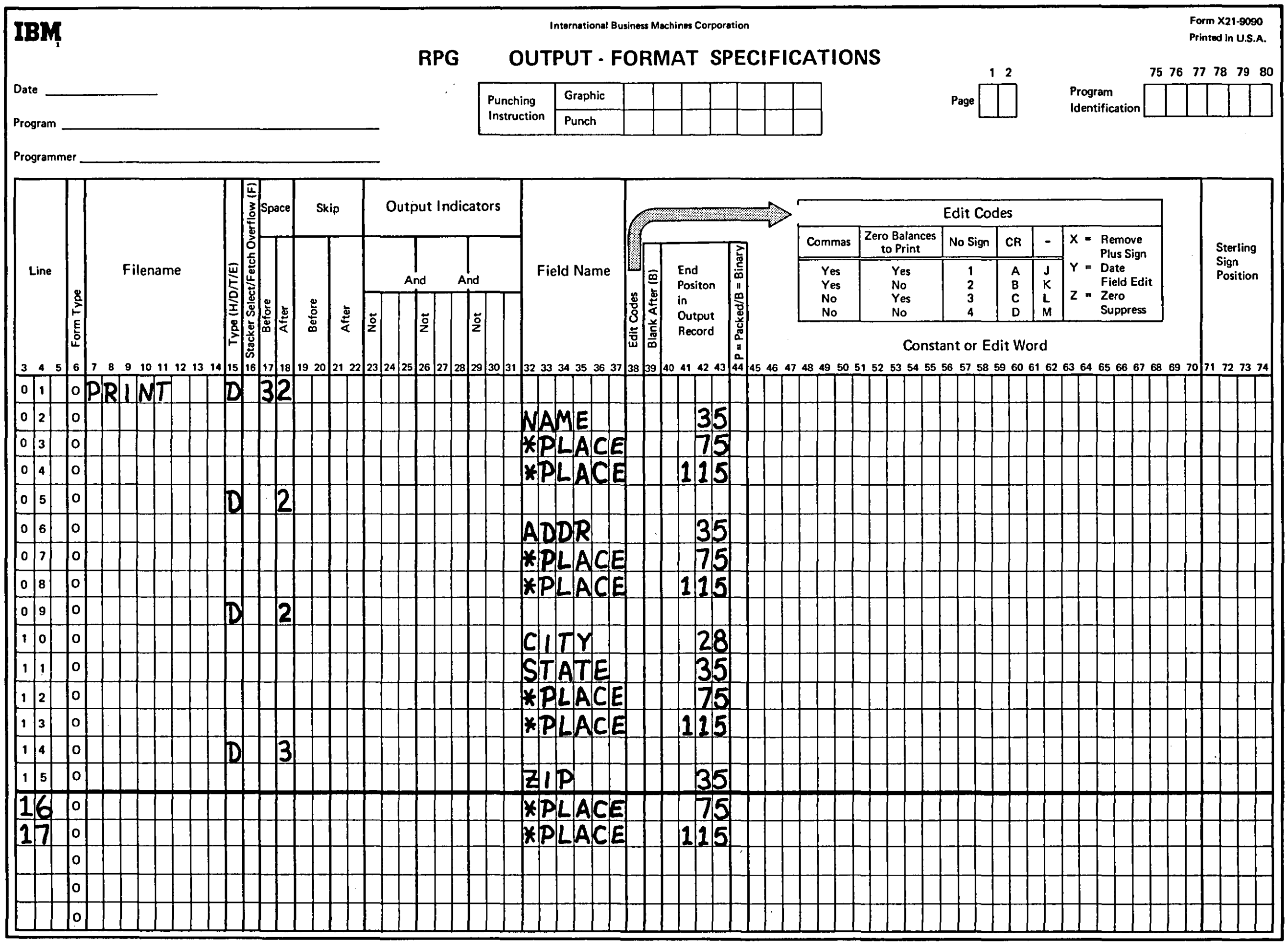
In the RPG II program cycle, each record specified is written or punched only *once* per cycle. For each record read for the job shown in Figure 11-15, the detail line specified in lines 01-04 is written only once. Remember that the *PLACE entry causes the field to be duplicated. Using *PLACE one line is printed with three identical names. The same is true for each of the other records specified. If you want to print 15 identical mailing labels, you need all records printed five times each.

Figure 11-16 shows the specifications necessary to print 15 mailing labels per customer. The *PLACE specifications on the Output-Format Sheet will cause three mailing labels to be printed side by side on the paper. Each EXCPT code used on the Calculation Sheet causes all records identified by an *E* in column 15 of the Output-Format Sheet to be printed one time in the order shown on the sheet. Because all four lines are to be printed on the mailing label, all are identified by an *E.* The five EXCPT codes will cause five rows of three mailing labels each to be printed.

Another set will not be printed at detail output time, because all records having an *E* in column 15 can be printed only at calculation time when the EXCPT operation code is encountered.

EXCPT can be used with punched cards or disk as well as printed output. It operates in the same way in all cases. Each time the EXCPT code is encountered, output lines identified by an *E* in column 15 are executed.

Only output files may have EXCPT records specified; EXCPT cannot be used for combined files. All EXCPT records must be specified after all heading, detail, and total lines on the Output-Format Sheet.



IBM

International Business Machines Corporation

Form X21-9090
Printed in U.S.A.

RPG OUTPUT - FORMAT SPECIFICATIONS

Date

Program

Programmer

Punching Instruction	Graphic								
	Punch								

Page 1 2

Program Identification 75 76 77 78 79 80

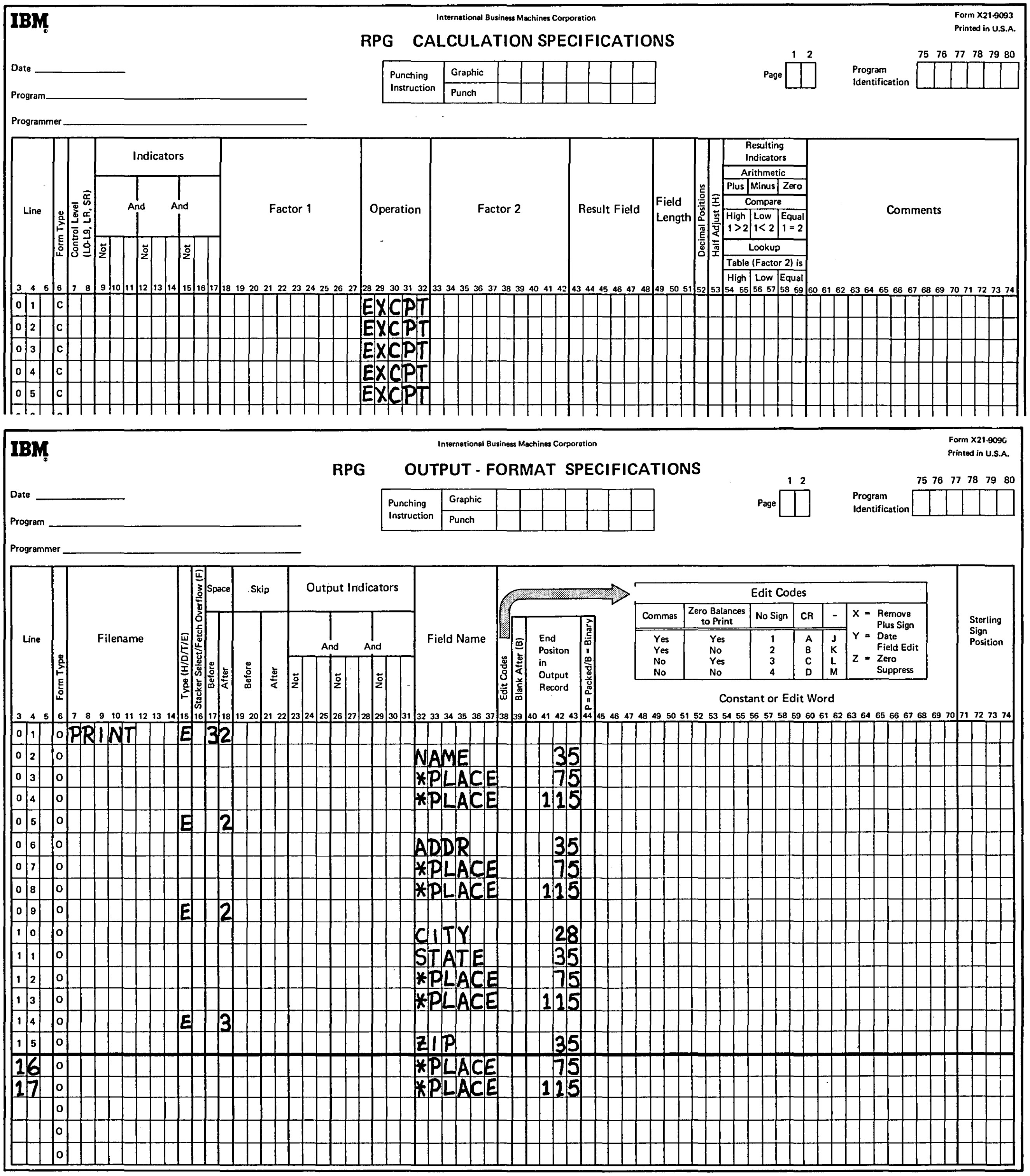
Edit Codes

Commas	Zero Balances to Print	No Sign	CR	-	
Yes	Yes	1	A	J	X = Remove Plus Sign
Yes	No	2	B	K	Y = Date Field Edit
No	Yes	3	C	L	Z = Zero Suppress
No	No	4	D	M	

Constant or Edit Word

Line (3-5)	Form Type (6)	Filename (7-14)	Type (H/D/T/E) (15)	Stacker Select/Fetch Overflow (F) (16)	Space Before (17)	Space After (18)	Skip Before (19-20)	Skip After (21-22)	Output Indicators (23-31)	Field Name (32-37)	Edit Codes (38)	Blank After (B) (39)	End Position in Output Record (40-43)	P = Packed/B = Binary (44)	Constant or Edit Word (45-70)	Sterling Sign Position (71-74)
01	O	PRINT	D		3	2										
02	O									NAME			35			
03	O									*PLACE			75			
04	O									*PLACE			115			
05	O		D			2										
06	O									ADDR			35			
07	O									*PLACE			75			
08	O									*PLACE			115			
09	O		D			2										
10	O									CITY			28			
11	O									STATE			35			
12	O									*PLACE			75			
13	O									*PLACE			115			
14	O		D			3										
15	O									ZIP			35			
16	O									*PLACE			75			
17	O									*PLACE			115			
	O															
	O															
	O															

Figure 11-15. Detail Output Operations



IBM

International Business Machines Corporation

Form X21-9093
Printed in U.S.A.

RPG CALCULATION SPECIFICATIONS

Date ____
Program ____
Programmer ____

Punching Instruction: Graphic / Punch

Page 1 2

Program Identification 75 76 77 78 79 80

Line (3-5)	Form Type (6)	Control Level (L0-L9, LR, SR) (7-8)	Indicators (9-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field (43-48)	Field Length (49-51)	Decimal Positions (52)	Half Adjust (H) (53)	Resulting Indicators (54-59)	Comments (60-74)
01	C				EXCPT							
02	C				EXCPT							
03	C				EXCPT							
04	C				EXCPT							
05	C				EXCPT							

Resulting Indicators: Arithmetic — Plus, Minus, Zero; Compare — High 1>2, Low 1<2, Equal 1=2; Lookup Table (Factor 2) is — High, Low, Equal

IBM

International Business Machines Corporation

Form X21-9090
Printed in U.S.A.

RPG OUTPUT - FORMAT SPECIFICATIONS

Date ____
Program ____
Programmer ____

Punching Instruction: Graphic / Punch

Page 1 2

Program Identification 75 76 77 78 79 80

Line (3-5)	Form Type (6)	Filename (7-14)	Type (H/D/T/E) (15)	Stacker Select/Fetch Overflow (F) (16)	Space Before (17)	Space After (18)	Skip Before (19-20)	Skip After (21-22)	Output Indicators (23-31)	Field Name (32-37)	Edit Codes (38)	Blank After (B) (39)	End Position in Output Record (40-43)	P = Packed/B = Binary (44)	Constant or Edit Word (45-70)	Sterling Sign Position (71-74)
01	O	PRINT	E		3	2										
02	O									NAME			35			
03	O									*PLACE			75			
04	O									*PLACE			115			
05	O		E			2										
06	O									ADDR			35			
07	O									*PLACE			75			
08	O									*PLACE			115			
09	O		E			2										
10	O									CITY			28			
11	O									STATE			35			
12	O									*PLACE			75			
13	O									*PLACE			115			
14	O		E			3										
15	O									ZIP			35			
16	O									*PLACE			75			
17	O									*PLACE			115			
	O															
	O															
	O															

Edit Codes

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Figure 11-16. EXCPT Operation Code Used with Exception Records

Conditioning the Use of EXCPT Operation

There are two ways you can condition an EXCPT operation: (1) on the Calculation Sheet; and (2) on the Output-Format Sheet.

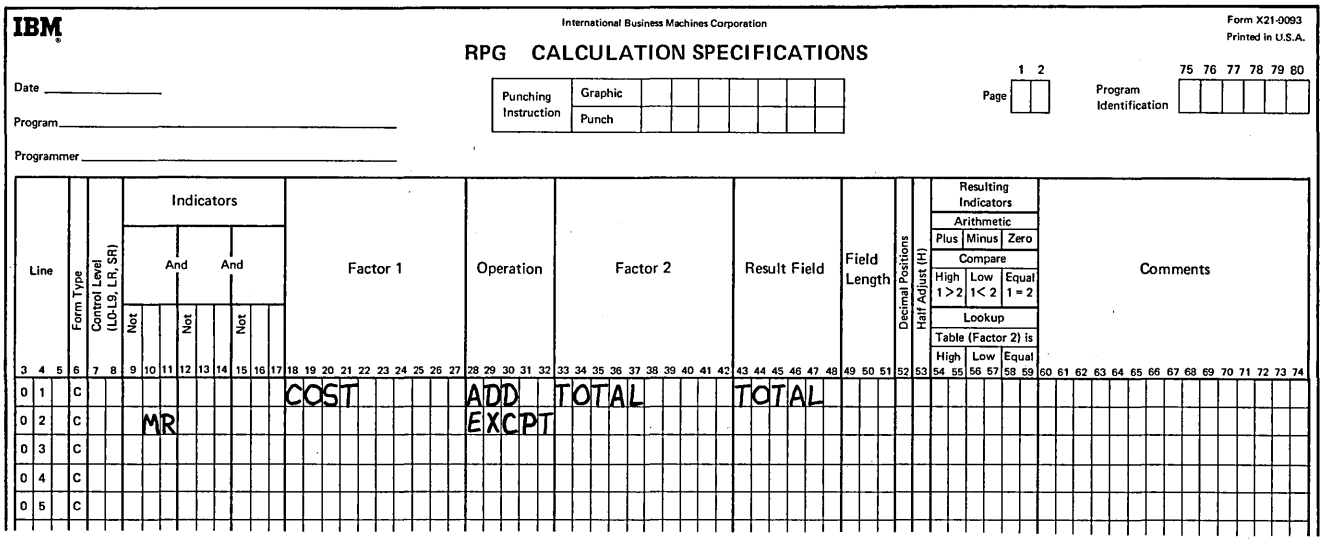
The EXCPT operation can be conditioned on the Calculation Sheet in the same way as any other operation. As shown in Figure 11-17, the EXCPT records are put out only when MR is on.

An indicator used on the Calculation Sheet controls the printing or punching of all EXCPT records. Individual EXCPT records are controlled by indicators specified in columns 23-31 of the Output-Format Sheet. These indicators are used in the same way for EXCPT records as they are for all other records.

Restriction: Overflow indicators cannot be used to condition an EXCPT line. This means that an EXCPT record cannot be a record that is printed only when the end of the page has been reached.

Remember, these lines are exceptions. They print only at calculation time, not at output time. Therefore, they could not possibly be printed when other overflow lines are.

An EXCPT line may be, however, printed on the overflow line. If it is, the overflow indicator will be turned on as usual. EXCPT lines can even *fetch* overflow. You may place an *F* in column 16 of any exception line. If the overflow indicator is on when the EXCPT line having an *F* in column 16 is reached, all lines conditioned by the overflow indicator will be printed before the exception line is printed.



IBM

International Business Machines Corporation

Form X21-0093
Printed in U.S.A.

RPG CALCULATION SPECIFICATIONS

Date
Program
Programmer

Punching Instruction: Graphic / Punch

Page (1 2)

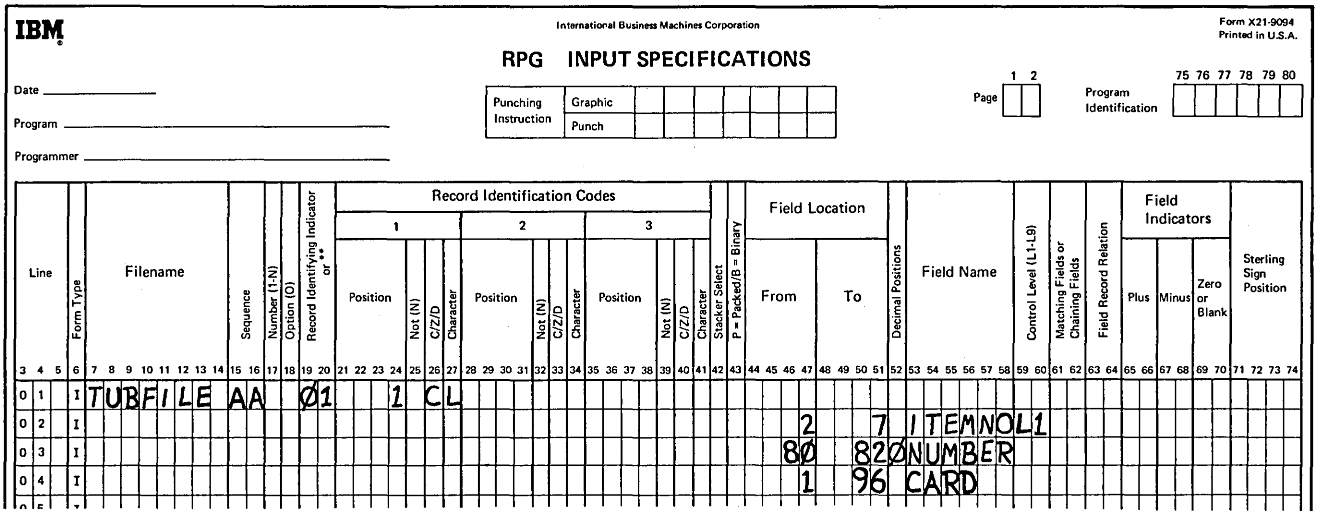
Program Identification (75 76 77 78 79 80)

Line (3-5)	Form Type (6)	Control Level (L0-L9, LR, SR) (7-8)	Indicators (9-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field (43-48)	Field Length (49-51)	Decimal Positions (52)	Half Adjust (H) (53)	Resulting Indicators (54-59)	Comments (60-74)
01	C			COST	ADD	TOTAL	TOTAL					
02	C		MR		EXCPT							
03	C											
04	C											
05	C											

Figure 11-17. Conditioning the EXCPT Operation Code

1. What occurs when the EXCPT operation code is executed?

2. In a program used to create a tub file, you need to punch a specified number of cards for each item. This number will be punched in each input card. Refer to the coded input sheet for record layouts and code the Calculation and Output-Format Sheets for the job.



IBM

International Business Machines Corporation

Form X21-9094
Printed in U.S.A.

RPG INPUT SPECIFICATIONS

Date

Program

Programmer

Punching Instruction: Graphic □□□□□□□□ / Punch □□□□□□□□

Page (1 2) □□

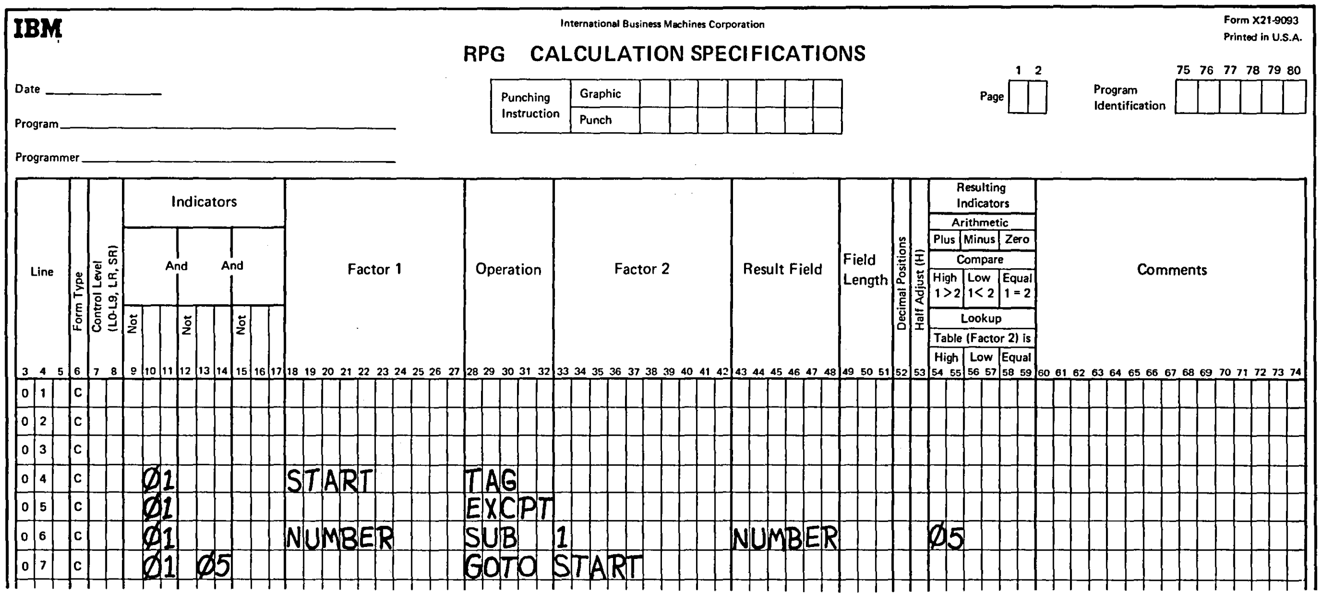
Program Identification (75 76 77 78 79 80) □□□□□□

Line (3 4 5)	Form Type (6)	Filename (7–14)	Sequence (15 16)	Number (1-N) (17)	Option (O) (18)	Record Identifying Indicator or ** (19 20)	Record Identification Codes 1: Position (21–24)	Not (N) (25)	C/Z/D (26)	Character (27)	2: Position (28–31)	Not (N) (32)	C/Z/D (33)	Character (34)	3: Position (35–38)	Not (N) (39)	C/Z/D (40)	Character (41)	Stacker Select (42)	P = Packed/B = Binary (43)	Field Location From (44–47)	Field Location To (48–51)	Decimal Positions (52)	Field Name (53–58)	Control Level (L1-L9) (59 60)	Matching Fields or Chaining Fields (61 62)	Field Record Relation (63 64)	Field Indicators Plus (65 66)	Minus (67 68)	Zero or Blank (69 70)	Sterling Sign Position (71–74)
0 1	I	TUBFILE	AA			Ø1	1		C	L																					
0 2	I																				2	7		ITEMNO	L1						
0 3	I																				8Ø	82	Ø	NUMBER							
0 4	I																				1	96		CARD							
0 5	I																														

Answers to Review 11. Exception Output

1. Immediate output for specified records occurs. These records are coded as exception records by an *E* in column 15 of the Output-Format Sheet.

2. See specification sheets.



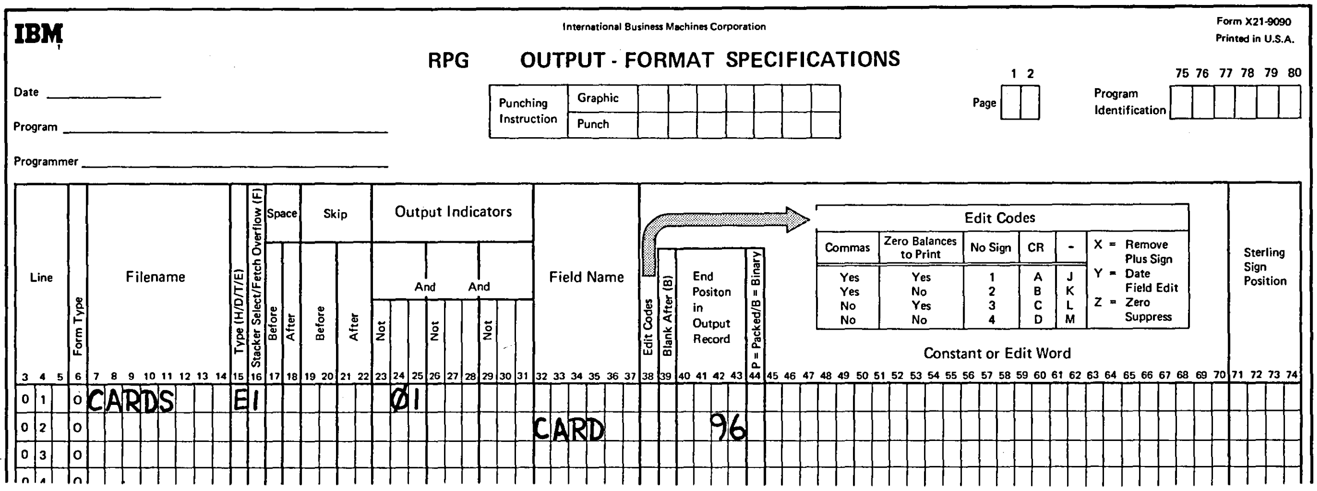
IBM International Business Machines Corporation Form X21-9093 Printed in U.S.A.

RPG CALCULATION SPECIFICATIONS

Date ______ Program ______ Programmer ______

Punching Instruction: Graphic / Punch — Page (1 2) — Program Identification (75 76 77 78 79 80)

Line (3-5)	Form Type (6)	Control Level (7-8)	Indicators (9-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field (43-48)	Field Length (49-51)	Decimal Positions (52)	Half Adjust (53)	Resulting Indicators (54-59)	Comments (60-74)
01	C											
02	C											
03	C											
04	C		Ø1	START	TAG							
05	C		Ø1		EXCPT							
06	C		Ø1	NUMBER	SUB	1	NUMBER				Ø5	
07	C		Ø1 Ø5		GOTO	START						



IBM International Business Machines Corporation Form X21-9090 Printed in U.S.A.

RPG OUTPUT - FORMAT SPECIFICATIONS

Date ______ Program ______ Programmer ______

Punching Instruction: Graphic / Punch — Page (1 2) — Program Identification (75 76 77 78 79 80)

Line (3-5)	Form Type (6)	Filename (7-14)	Type (H/D/T/E) (15)	Stacker Select/Fetch Overflow (F) (16)	Space Before (17)	Space After (18)	Skip Before (19-20)	Skip After (21-22)	Output Indicators (23-31)	Field Name (32-37)	Edit Codes (38)	Blank After (B) (39)	End Position in Output Record (40-43)	P = Packed/B = Binary (44)	Constant or Edit Word (45-70)	Sterling Sign Position (71-74)
01	O	CARDS	E	1					Ø1							
02	O									CARD			96			
03	O															
04	O															

Edit Codes

Commas	Zero Balances to Print	No Sign	CR	-	
Yes	Yes	1	A	J	X = Remove Plus Sign
Yes	No	2	B	K	Y = Date Field Edit
No	Yes	3	C	L	Z = Zero Suppress
No	No	4	D	M	

PERFORMING TOTAL OPERATIONS WITHOUT A CONTROL BREAK

In this section, you will learn to work with a special internal control level indicator L0. You will also learn another use of the L1-L9 indicators.

Internal Control Level Indicator L0

L0 is a unique control level indicator which is always on. You cannot assign it to a field, as you do L1-L9, by entering it in columns 59-60 of the Input sheet. But, you can use it to condition a calculation operation. The operation so conditioned will be done at total time for every program cycle, since L0 is always on.

The main purpose of the L0 indicator is to allow you to specify total operations when indicator L1-L9 are not available or when they cannot accomplish the job.

Consider the use of L0 in a summary punching job. Basically the job consists of finding payment due by subtracting discount received from total purchases and punching this amount along with other information into a summary card (Figure 11-18). The input file shown in Figure 11-19 consists of three record types:

- Name/address cards.
- Item cards which describe an individual item purchased by the customer.
- Blank cards which are to be summary punched.

The amount of discount each customer receives is shown by the last digit of the account number. The discount code is as follows:

0 – no discount

1 – two percent discount

2 – four percent discount

For each item card, quantity (QTY) must be multiplied by price (PRICE). The result is then accumulated in a field called TOTAL. After all cards of a group have been read, you can find the discount and net payment. These two operations are total operations that should be conditioned by control level indicators.

Is there a field which can be used as a control field? CUSTNO could be used, but if CUSTNO is defined as the control field, the blank card has no CUSTNO field. Therefore, control field contents will not be checked when this card is read, and a control break will not occur at the correct time.

However, when the name/address card is read, a control break will occur since the contents of CUSTNO on the name/address card are different from the previous name/address and item cards. Total operations will be done. Summary punching the blank card is a total operation. Thus the punching will occur, but the wrong card will be punched. The only card available for punching at total time is the one that caused the control break – the name/address card. This card instead of the blank card will be punched.

RPG II control field logic will not work for this job. The blank card, not the name/address card, should cause a control break, but it never will since it has no control field on it.

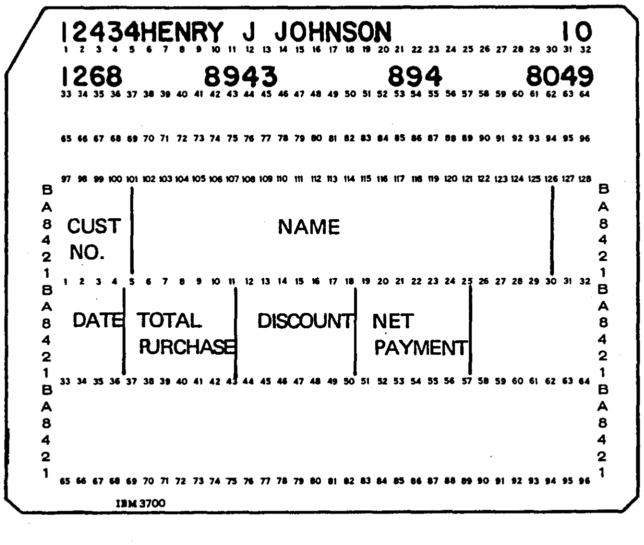
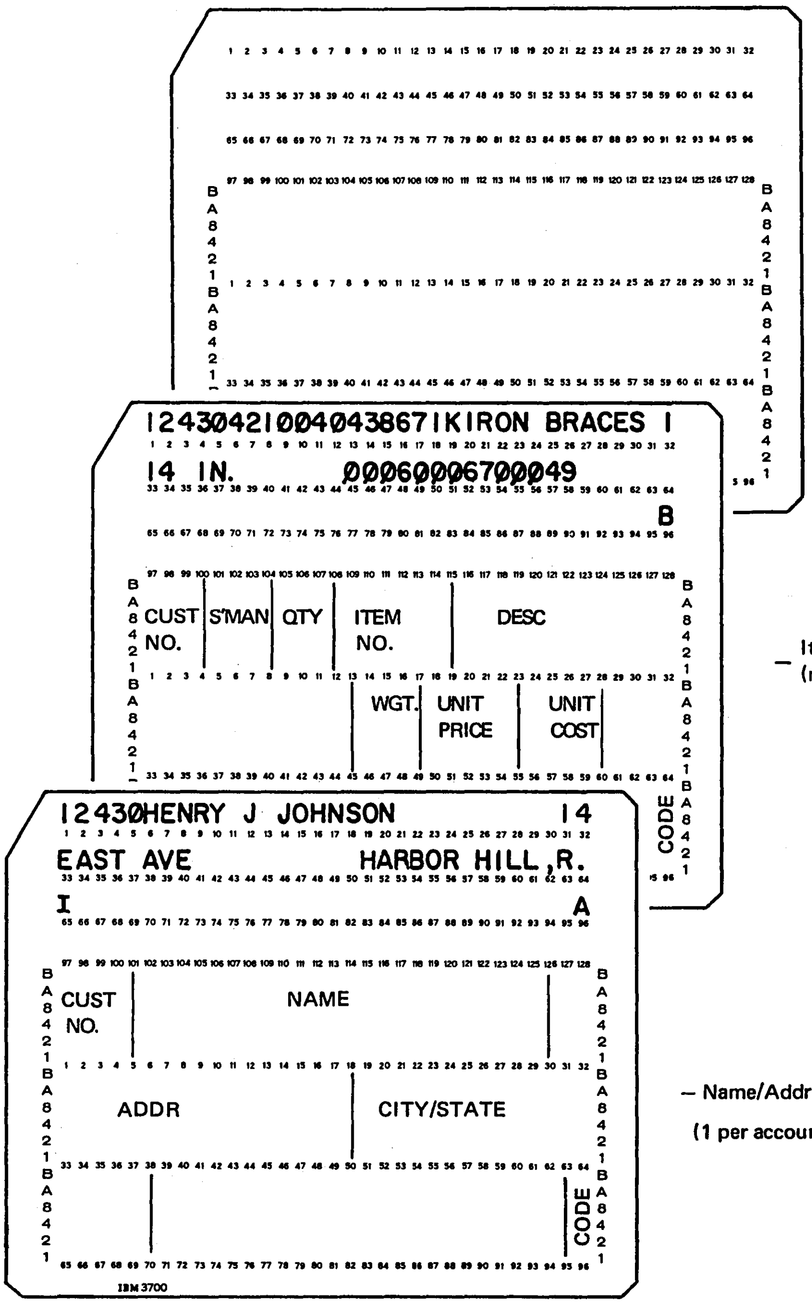


Figure 11-18. Summary Card



12430421ØØ4Ø43867IKIRON BRACES I

14 IN. ØØØ6ØØØ67ØØØ49

B

1243ØHENRY J JOHNSON 14

EAST AVE HARBOR HILL,R.

I A

— Blank Summary Cards

— Item Cards (may be several per account)

— Name/Address Cards (1 per account)

Figure 11-19. Record Types in an Input File

Causing Control Breaks

When it is necessary to do total operations but no control fields are available to cause a control break, you may use L0 to cause an *artificial* control break.

Remember that total operations are those conditioned by L1-L9 or LR. L1-L9 will not turn on unless there is a control break or unless they are set on. For the job just discussed, L1 must be artificially set on since total operations are required but a control break does not occur.

If total operations are ever to be done, they must be conditioned by L0, L1-L9, LR. L1-L9 are not available, but L0 is since L0 is always on. Thus you can use L0 to condition the operation which will set L1 on. When L1 is turned on you can do total operations necessary for the job.

In this case, you wish to do total operations when the blank card is read. 03 is the indicator assigned to the blank card. Thus, you must set L1 on when the record identifying indicator 03 is on. The SETON operation is, therefore, conditioned by both L0 and 03 (Figure 11-20).

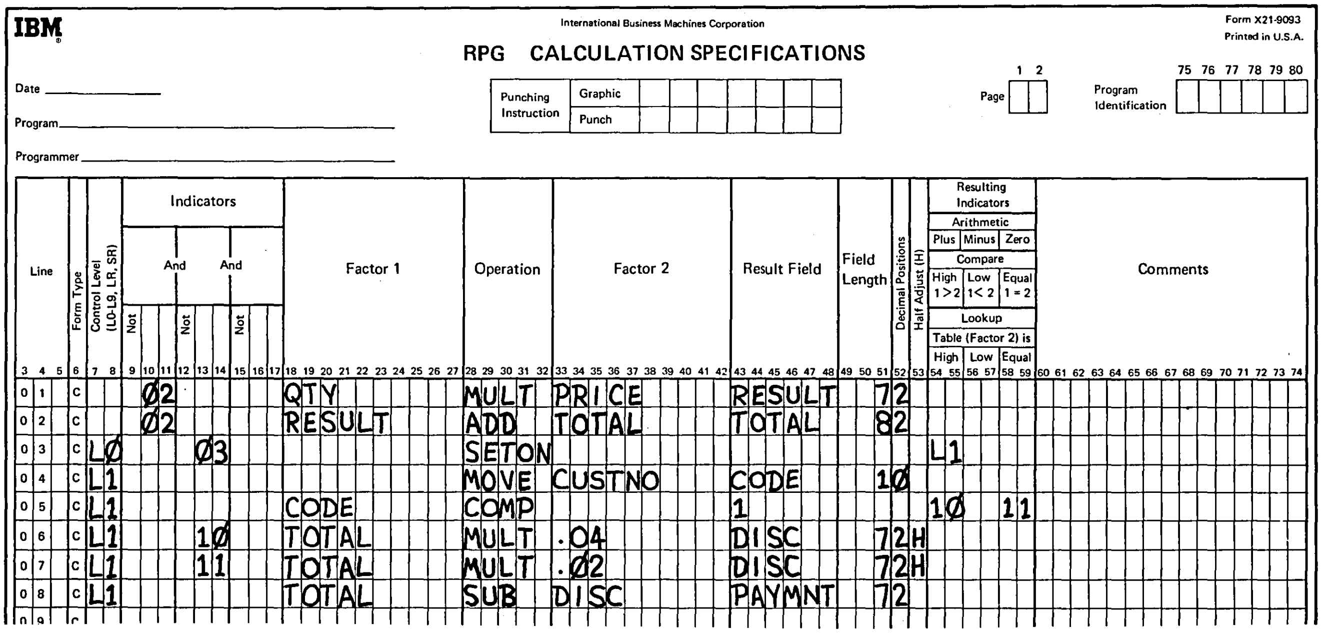
Control level indicators should be set on at total time. If they are set on at detail time, they are turned off before any operations which they condition are encountered.

Coding Control Level Indicators As Calculation Conditioning Indicators

Control level indicators are normally entered in columns 7-8 of the Calculation Sheet where they specify which calculations are to be done at total time. They may, also, be used in columns 9-17 where they indicate which detail operations are to be done on the first card of a control group.

Control level indicators are turned on near the beginning of the program cycle if the contents of the control field on the card just read are different from the contents of the previous control field. Since the indicator is not turned off until the end of the cycle, it is on during total and detail time. Thus, it is available to use as a conditioning indicator during detail time as well as total time.

When an operation is not conditioned by control level indicators specified in columns 7-8 of the Calculation Sheet, the operation is done at detail time. If the operation is conditioned by control level indicator specified in columns 9-17, and not in 7-8, the operation is still done at detail time when L1 is on. L1 is on only during the processing of the first card in a control group for only the first card in a new group causes a control break.



IBM

International Business Machines Corporation

RPG CALCULATION SPECIFICATIONS

Form X21-9093
Printed in U.S.A.

Date ____________

Program ____________

Programmer ____________

Punching Instruction: Graphic / Punch

Page 1 2 □□

Program Identification 75 76 77 78 79 80

Line (3-5)	Form Type (6)	Control Level (L0-L9, LR, SR) (7-8)	Indicators (9-11)	And (12-14)	And (15-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field (43-48)	Field Length (49-51)	Decimal Positions (52)	Half Adjust (H) (53)	Resulting Indicators: Plus / High 1>2 / High (54-55)	Minus / Low 1<2 / Low (56-57)	Zero / Equal 1=2 / Equal (58-59)	Comments (60-74)
01	C		Ø2			QTY	MULT	PRICE	RESULT	7	2					
02	C		Ø2			RESULT	ADD	TOTAL	TOTAL	8	2					
03	C	LØ		Ø3			SETON						L1			
04	C	L1					MOVE	CUSTNO	CODE	1Ø						
05	C	L1				CODE	COMP		1				1Ø		11	
06	C	L1		1Ø		TOTAL	MULT	.04	DISC	7	2	H				
07	C	L1		11		TOTAL	MULT	.Ø2	DISC	7	2	H				
08	C	L1				TOTAL	SUB	DISC	PAYMNT	7	2					
09	C															

Figure 11-20. Conditioning the SETON Operation

CONTROLLING WHEN OPERATIONS ARE PERFORMED

When you are processing a job, there can be certain conditions determing when operations should or should not be performed. This section will discuss two indicators which can control when operations are performed: halt indicators and external indicators.

Halt Indicators (H1-H9)

Halt indicators may be used to:

- Stop a program after finding an error.
- Condition calculation or output operations after finding an error. This is necessary because the system does not halt until after all calculation and detail output operations are performed for the record that caused the error. Halt indicators can be used in the same way as indicators 01-99 to condition operations.

Stopping a Program After an Error Occurs

Halt indicators are used to stop an RPG program when a specified condition is satisfied. Halt indicators can be used as record identifying, field, or resulting indicators. When halt indicators are used as record identifying indicators, a halt will be caused by a specific type of record; when used as field indicators, a halt will be caused by a specific type of input data; when used as resulting indicators, a halt will be caused by a specific type of results from calculations.

A halt indicator can be turned on at one of four different times (Figure 11-21). Its use, of course, will determine when it is turned on. The program does not halt immediately when a halt indicator is turned on. All total and detail operations remaining in the cycle are performed first; then the program halts. This means that processing will still be completed on information from the card that caused the specified condition.

After a halt, you can continue processing by pressing START on the processing unit. Halt indicators are always turned off before another program cycle begins.

Preventing Calculations From Being Performed When An Error Occurs

Halt indicators are usually used to test for an error condition in your data. Specifications shown in Figure 11-22 illustrate the use of H1 to test for an error condition. A test is made to determine if the INSTOK field in the amount card is minus, which indicates an error condition. When this occurs, H1 is turned on. Since calculations should be done when this error *does not* occur, they must be conditioned by NH1. This means that the calculations will be done only when H1 is *not* on.

Halt indicators can also be specified on the Calculation Sheet to test for an error. For example, in Figure 11-23, H1 is set on if the result of operation in line 01 is negative. If quantity in stock (INSTOK) is negative after quantity shipped (QTYSH) has been subtracted, an error has occurred. H1 turns on and the system will halt after the current cycle.

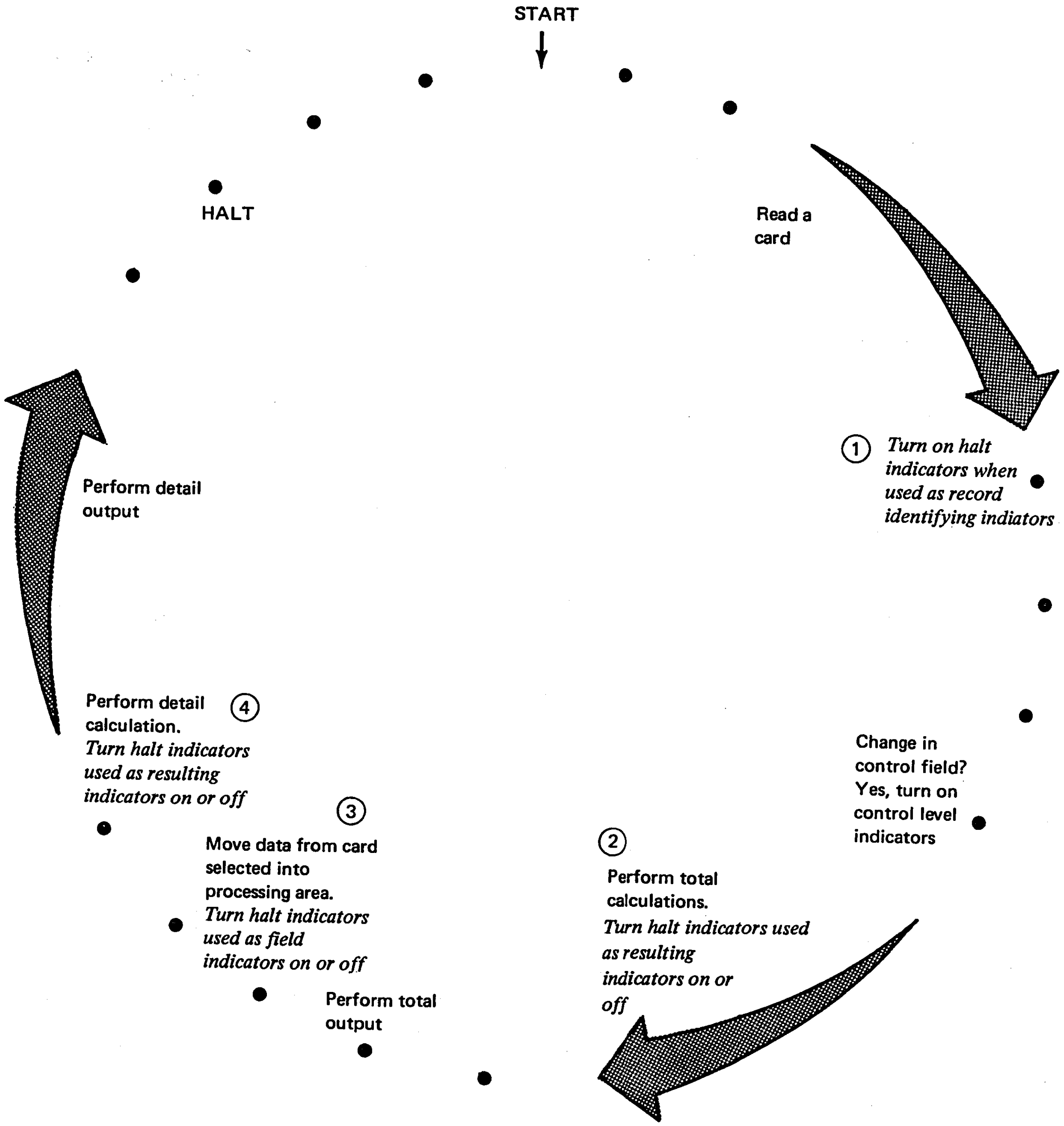
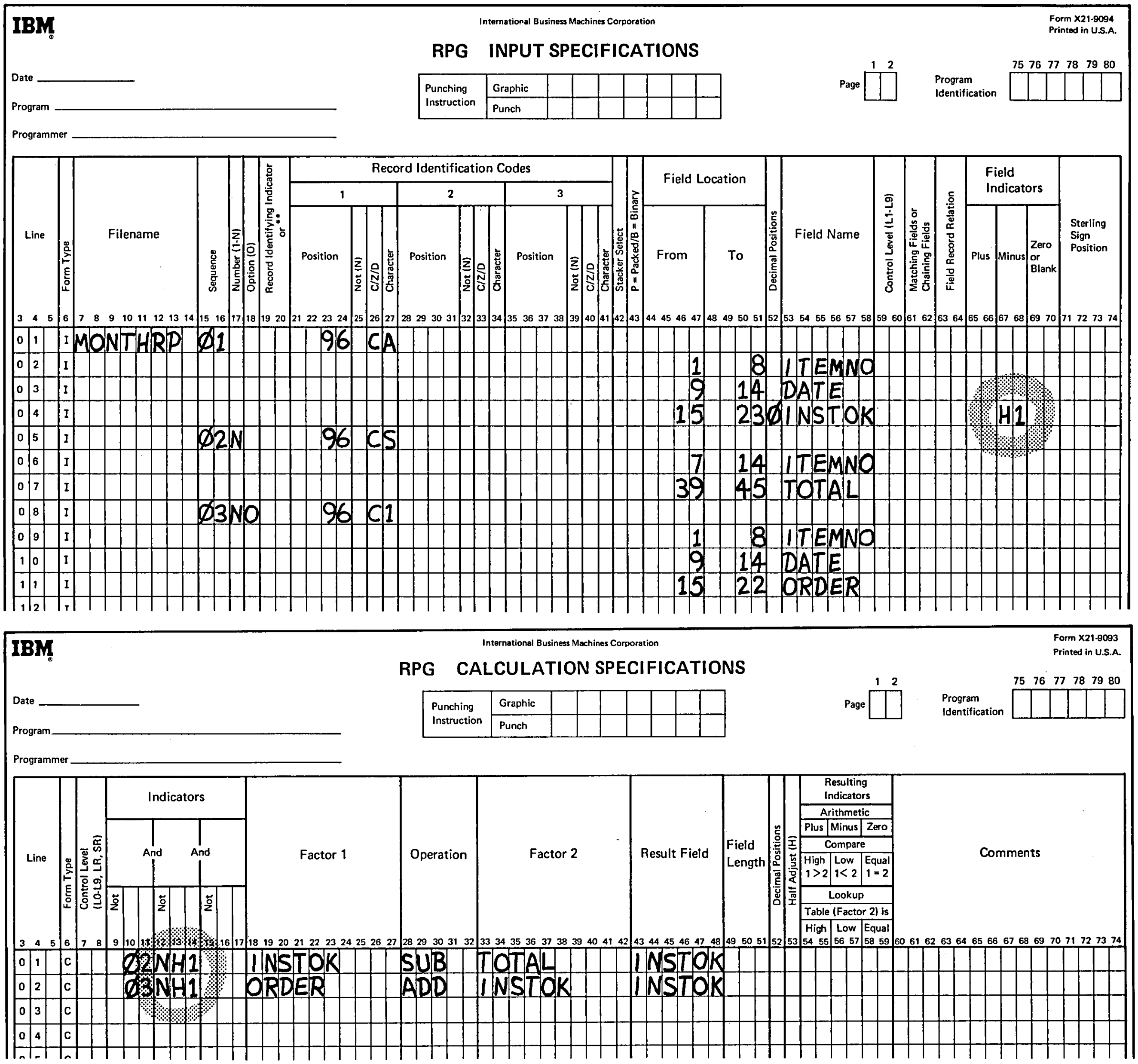

START
HALT
Read a card
1 Turn on halt indicators when used as record identifying indiators
Change in control field? Yes, turn on control level indicators
2 Perform total calculations. Turn halt indicators used as resulting indicators on or off
Perform total output
3 Move data from card selected into processing area. Turn halt indicators used as field indicators on or off
Perform detail calculation. 4 Turn halt indicators used as resulting indicators on or off
Perform detail output

Figure 11-21. Logic for Halt Indicators



IBM International Business Machines Corporation Form X21-9094 Printed in U.S.A.

RPG INPUT SPECIFICATIONS

Date ________ Program ________ Programmer ________

Punching Instruction: Graphic / Punch   Page (1 2)   Program Identification (75 76 77 78 79 80)

Line (3-5)	Form Type (6)	Filename (7-14)	Sequence (15-16)	Number (1-N) (17)	Option (O) (18)	Record Identifying Indicator (19-20)	Record Identification Codes 1: Position (21-24)	Not (N) (25)	C/Z/D (26)	Character (27)	Field Location From (44-47)	To (48-51)	Decimal Positions (52)	Field Name (53-58)	Field Indicators Plus (65-66)	Minus (67-68)	Zero or Blank (69-70)
01	I	MONTHRP	Ø1				96		C	A							
02	I										1	8		ITEMNO			
03	I										9	14		DATE			
04	I										15	23	Ø	INSTOK		H1	
05	I		Ø2	N			96		C	S							
06	I										7	14		ITEMNO			
07	I										39	45		TOTAL			
08	I		Ø3	N	O		96		C	1							
09	I										1	8		ITEMNO			
10	I										9	14		DATE			
11	I										15	22		ORDER			
12	I																

IBM International Business Machines Corporation Form X21-9093 Printed in U.S.A.

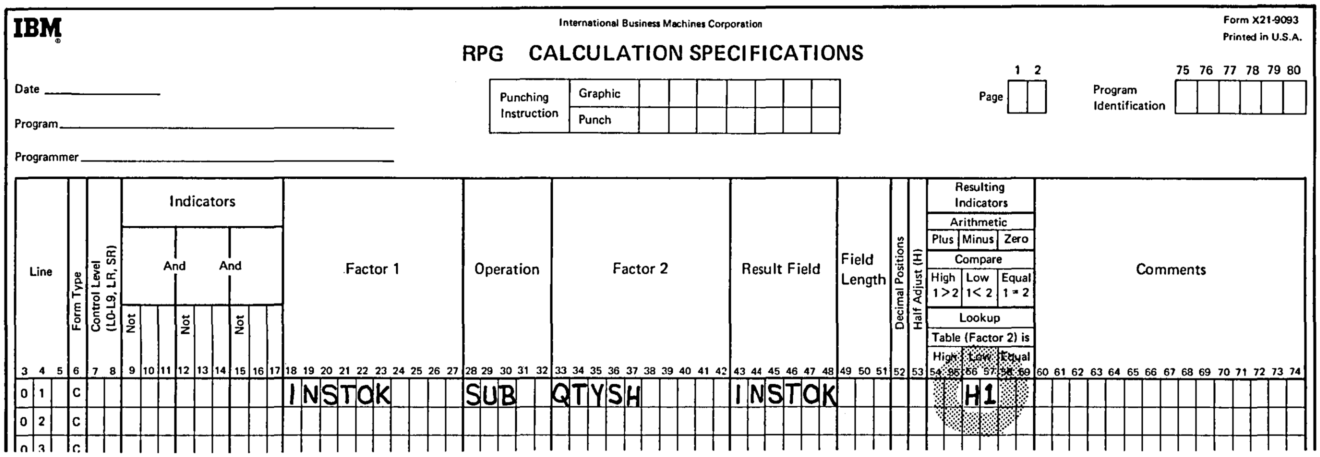
RPG CALCULATION SPECIFICATIONS

Date ________ Program ________ Programmer ________

Punching Instruction: Graphic / Punch   Page (1 2)   Program Identification (75 76 77 78 79 80)

Line (3-5)	Form Type (6)	Control Level (L0-L9, LR, SR) (7-8)	Indicators (9-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field (43-48)	Field Length (49-51)	Decimal Positions (52)	Half Adjust (H) (53)	Resulting Indicators (54-59)	Comments (60-74)
01	C		Ø2 NH1	INSTOK	SUB	TOTAL	INSTOK					
02	C		Ø3 NH1	ORDER	ADD	INSTOK	INSTOK					
03	C											
04	C											

Figure 11-22. Conditioning Calculations by a Halt Indicator



IBM

International Business Machines Corporation

Form X21-9093
Printed in U.S.A.

RPG CALCULATION SPECIFICATIONS

Date

Program

Programmer

Punching Instruction	
Graphic	
Punch	

Page 1 2

Program Identification 75 76 77 78 79 80

Line (3-5)	Form Type (6)	Control Level (L0-L9, LR, SR) (7-8)	Indicators: Not (9-11)	Indicators: And Not (12-14)	Indicators: And Not (15-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field (43-48)	Field Length (49-51)	Decimal Positions (52)	Half Adjust (H) (53)	Resulting Indicators, Arithmetic: Plus; Compare: High 1>2; Lookup, Table (Factor 2) is: High (54-55)	Minus; Low 1< 2; Low (56-57)	Zero; Equal 1 = 2; Equal (58-59)	Comments (60-74)
0 1	C					INSTOK	SUB	QTYSH	INSTOK					H1		
0 2	C															
0 3	C															

Figure 11-23. Testing Result Field for Error Conditions

External Indicators

External indicators have two major functions:

- They can condition the use of files.
- They can control which calculations should be done for a specific job run.

Using One Program To Do Two Jobs

One program can be used to do similar jobs if you use external indicators.

Consider, for example, the following jobs. Two jobs of reports are required each week. One is a sales analysis report showing what items sold during the week; the second is an inventory report showing balance on hand for each item in stock. Notice the similarity in the format of the reports (Figure 11-24).

Two files are available: The MASTER file which contains balance forward records for all items in the store and a transaction file (TRANS) which contains all the weekly sales for each item (Figure 11-25). Both files are in ascending order by item number.

The sales analysis job requires only a listing of records found in the transaction file.

The inventory report is a matching records job. When records from both files match, the number sold is subtracted from the balance on hand. The new balance is then printed on the report following the list of transactions.

The inventory job requires two files; the other only one. You can write one program to do jobs which have such different file requirements by using external indicators to tell the program when to expect two files.

Setting External Indicators

Although most indicators are set by the program, you set external indicators prior to the execution of the program. This is done by including a SWITCH statement in your Operation Control Language. The format of the SWITCH statement is:

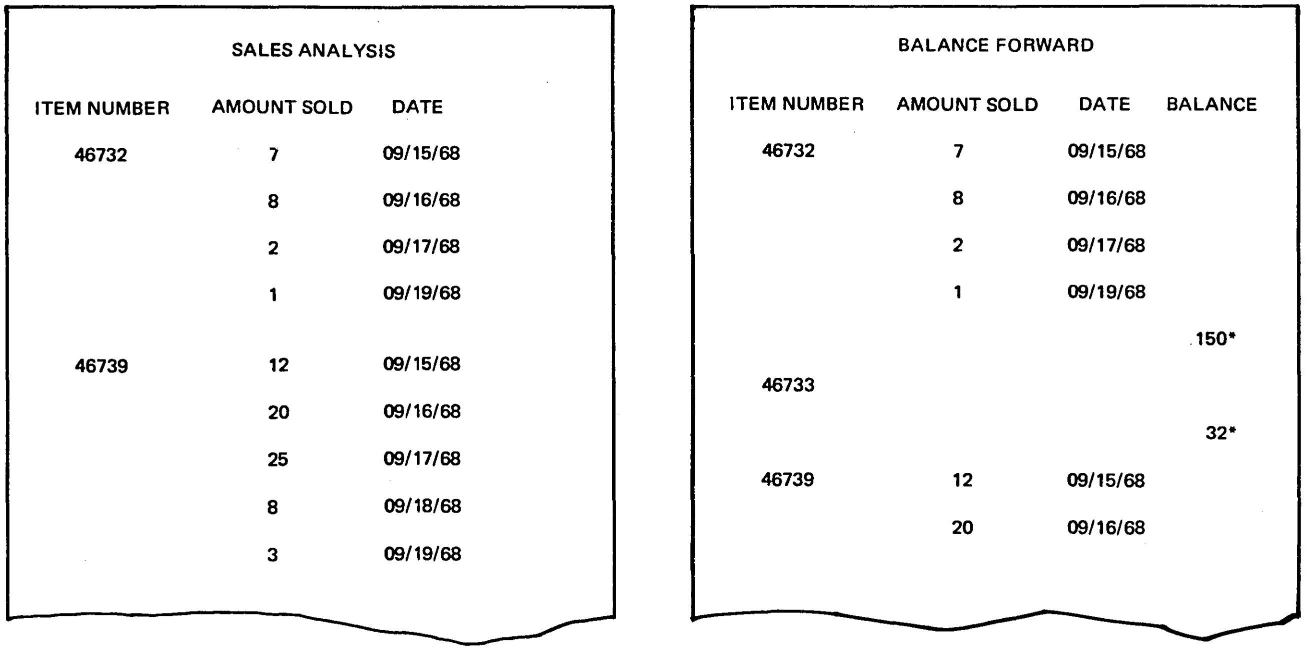
```
// SWITCH indicator   settings
```

The indicator settings are:

1 = indicator is on.

0 = indicator is off.

X = indicator is unaffected.



```
                SALES ANALYSIS

ITEM NUMBER     AMOUNT SOLD     DATE
   46732             7        09/15/68
                     8        09/16/68
                     2        09/17/68
                     1        09/19/68

   46739            12        09/15/68
                    20        09/16/68
                    25        09/17/68
                     8        09/18/68
                     3        09/19/68
```

```
                    BALANCE FORWARD

ITEM NUMBER     AMOUNT SOLD     DATE      BALANCE
   46732             7        09/15/68
                     8        09/16/68
                     2        09/17/68
                     1        09/19/68
                                              150*
   46733
                                               32*
   46739            12        09/15/68
                    20        09/16/68
```

Figure 11-24. Two Similar Reports from Two Different Jobs

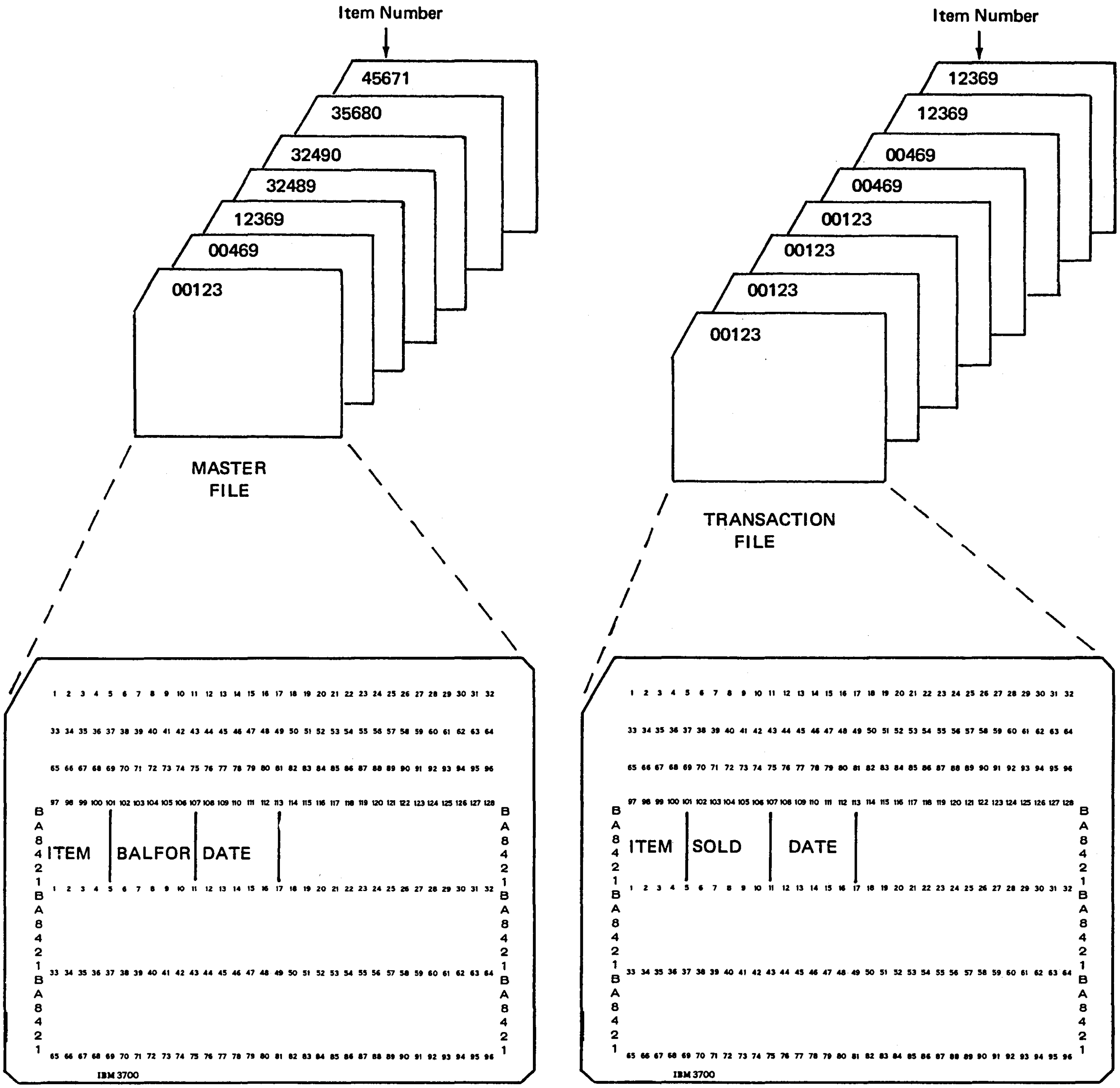


Figure 11-25. Format of Records Used in Sales Analysis and Balance Forward Jobs

Figure 11-26 shows a SWITCH statement which sets external indicators one and eight on and indicators two through six off. Indicator seven is unaffected.

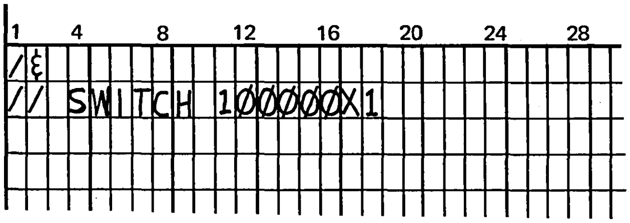
Once an indicator is set, it is not changed until you provide another SWITCH statement or perform IPL. You cannot use the SETON or SETOF operation codes with external indicators.

Using an External Indicator to Condition a File

You can assign an external indicator to a file. When the indicator is on, the file is used; when it is off, the file is not used. This, then, is how you can tell a program when to expect one file and when to expect two. Consider again the two jobs discussed previously: sales analysis and inventory. The TRANS file is needed for both jobs, the MASTER file is only needed for the inventory job. Thus, the MASTER file is assigned the U1 indicator. You set the indicator on for the inventory job (MASTER is used here) and off for the sales analysis job (MASTER is not used).

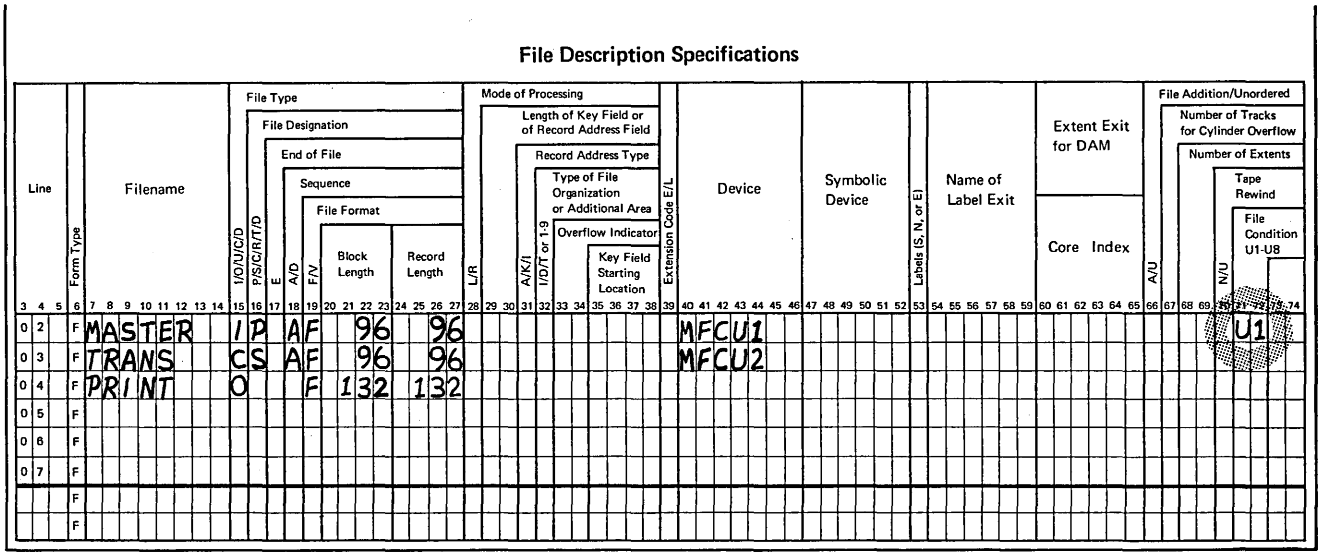
The U1 indicator is assigned to a file on the File Description Sheet in columns 71-72. Any of the eight external indicators (U1-U8) could be used. U1 was arbitrarily chosen for this job (Figure 11-27).

When writing a program to do two jobs, be certain that the jobs are similar. When the jobs require many different calculations and output operations, writing two different programs would be easier than using external indicators.



```
1  4   8   12  16  20  24  28
/&
// SWITCH 1ØØØØØX1
```

Figure 11-26. SWITCH Control Statement



File Description Specifications

Line	Form Type	Filename	File Type	File Designation	End of File	Sequence	File Format	Block Length	Record Length	Device	File Condition U1-U8
02	F	MASTER	I	P		A	F	96	96	MFCU1	U1
03	F	TRANS	C	S		A	F	96	96	MFCU2	
04	F	PRINT	O				F	132	132		
05	F										
06	F										
07	F										
	F										
	F										

Mode of Processing
Length of Key Field or of Record Address Field
Record Address Type
Type of File Organization or Additional Area
Overflow Indicator
Key Field Starting Location
Extension Code E/L
Symbolic Device
Labels (S, N, or E)
Name of Label Exit
Extent Exit for DAM
Core Index
File Addition/Unordered
Number of Tracks for Cylinder Overflow
Number of Extents
Tape Rewind

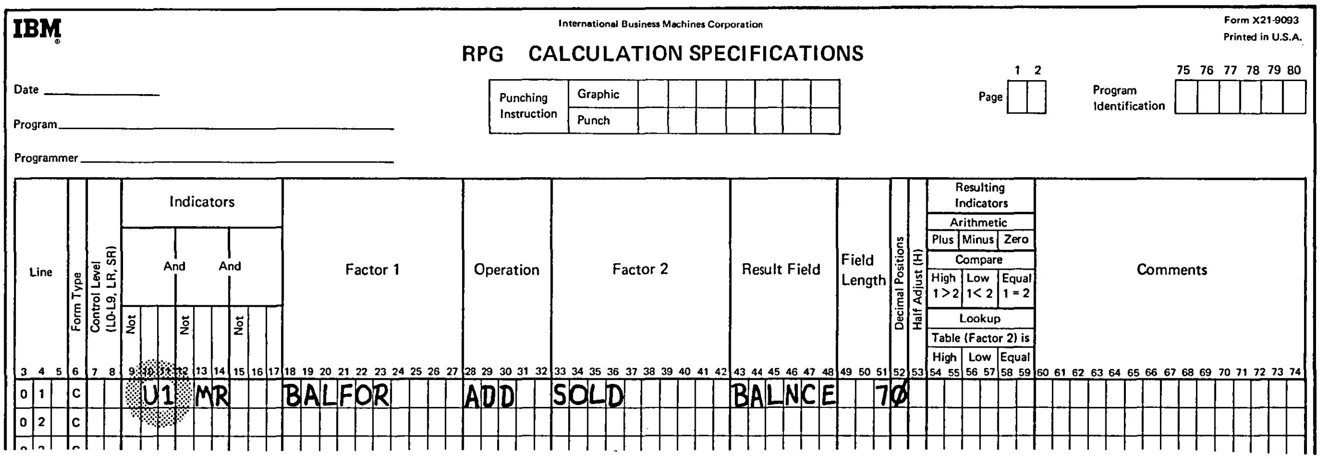
Figure 11-27. Assignment of an External Indicator

Controlling Calculations For A Specific Job When Using One Program for Several Jobs

Naturally, the calculations performed and the type of report written depends upon the job being done. Different calculation and output-format;specifications are needed for each job. In order to determine which specifications to use for a particular job, calculation and output-format specifications must also be conditioned by the external indicator.

Consider for example, calculations done for a sales analysis job. For each item in stock, monthly total sold is calculated and then added to the previous month's year-to-date total to find the current year-to-date total. In the first month of a new year, monthly totals should not be added to prior year-to-date totals because totals are not carried over from year to year. This last statement, the year-to-date addition statement, therefore, is not done for all program runs. By conditioning the statement with an external indicator, you can control when the operation is done. In Figure 11-28, the monthly total is added to prior year-to-date only when U1 is on.

When one program is written to do two similar jobs, some calculations may be used for both jobs and some for only one. Again, you use external indicators to control which calculation specifications are used for each job.



IBM

International Business Machines Corporation

Form X21-9093
Printed in U.S.A.

RPG CALCULATION SPECIFICATIONS

Date ____________

Program ____________

Programmer ____________

Punching Instruction	Graphic								
	Punch								

Page 1 2

Program Identification 75 76 77 78 79 80

Line (3-5)	Form Type (6)	Control Level (L0-L9, LR, SR) (7-8)	Not (9)	Indicator (10-11)	Not (12)	And (13-14)	Not (15)	And (16-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field (43-48)	Field Length (49-51)	Decimal Positions (52)	Half Adjust (H) (53)	Resulting Indicators: Arithmetic Plus / Compare High 1>2 / Lookup Table (Factor 2) is High (54-55)	Minus / Low 1<2 / Low (56-57)	Zero / Equal 1=2 / Equal (58-59)	Comments (60-74)
01	C			U1		MR			BALFOR	ADD	SOLD	BALNCE	7	Ø					
02	C																		
03	C																		

Figure 11-28. Conditioning a Calculation by an External Indicatcr

1. When do the following indicators turn on?

   H1-H9, L0, U1-U8

2. When are they turned off?

3. How is the RPG II logic altered by the halt indicators?

4. What are the two major uses for the external indicators, U1-U8?

Answers to Review 11. Indicators

1. H1-H9 are turned on immediately when the condition being tested is true. L0 is always on. U1-U8 is turned on by the OCL SWITCH statement.

2. H1-H9 are turned off at the beginning of the next RPG II cycle. L0 is never off. U1-U8 is never affected by RPG II. Only the OCL SWITCH statement affects these indicators.

3. When halt indicators are turned on, the program continues through all total and detail calculation and output; then halts.

4. U1-U8 indicators are used to condition certain calculations or to condition the activity of a file for a certain run of a program. These conditions can be changed without recompiling the program.

BINARY FIELD OPERATIONS (CONTROLLING SWITCHES)

RPG II provides certain operation codes which set and test individual bits in storage. These individual bits can be set and tested to allow you further control over processing. The bits are called switches and their functions are similar to that of RPG II indicators. The operation codes which set and test the bits are known as *binary field operations.* A binary field is a one-byte field containing 8 bits labeled 0-7. The bits can be set on, set off, and tested. Since each bit can be utilized, there are eight indicators in every byte.

When using binary field operations, remember how data fields are initialized by the system:

- Alphameric fields are initialized to Hexadecimal '40'.
- Numeric fields are initialized to Hexadecimal'F0'.

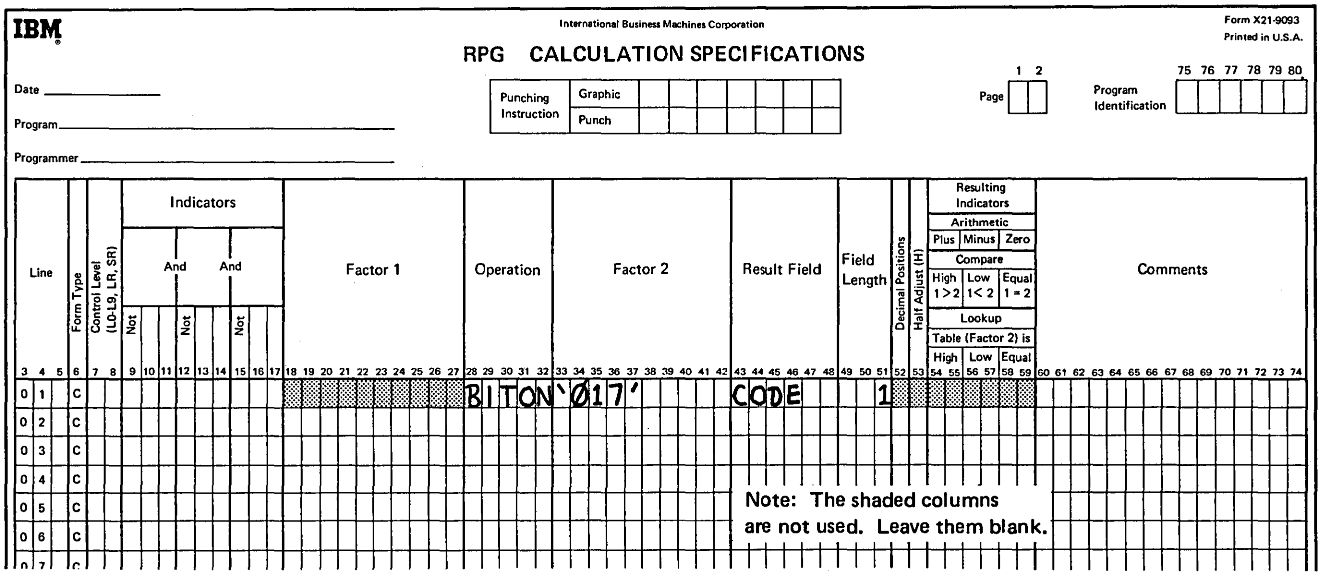
You should initialize the binary field containing the bits to be set and tested to binary zero (Hexadecimal '00') at the beginning of the program.

BITON Operation Code

Figure 11-29 shows a Calculation Sheet containing the BITON operation code. This operation code causes specified bits in Factor 2 to turn on (set to 1) in the field named as the Result Field. The field named in the Result Field must be one-position alphameric field. Since it is one position in length, a 1 must be entered in column 51 of Field Length. One or more of the eight bits can be turned on. To turn on the first bit in a field, enter 0 in Factor 2. These bit numbers must be enclosed by apostrophes.

You can use conditioning indicators in columns 7-17. You may also turn on a bit in an array element, but that array element must be one position in length.

In Figure 11-29, bits 0, 1, and 7 are set to 1 in the binary field labeled CODE.



IBM

International Business Machines Corporation

Form X21-9093
Printed in U.S.A.

RPG CALCULATION SPECIFICATIONS

Date

Program

Programmer

Punching Instruction: Graphic / Punch

Page 1 2

Program Identification 75 76 77 78 79 80

Line (3-5)	Form Type (6)	Control Level (L0-L9, LR, SR) (7-8)	Indicators: Not (9-11)	And: Not (12-14)	And: Not (15-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field (43-48)	Field Length (49-51)	Decimal Positions (52)	Half Adjust (H) (53)	Resulting Indicators: Arithmetic Plus / Compare High 1>2 / Lookup Table (Factor 2) is High (54-55)	Minus / Low 1<2 / Low (56-57)	Zero / Equal 1=2 / Equal (58-59)	Comments (60-74)
01	C						BITON	'Ø17'	CODE	1						
02	C															
03	C															
04	C															
05	C															
06	C															
07	C															

Note: The shaded columns are not used. Leave them blank.

Figure 11-29. The BITON Operation Code

BITOF Operation Code

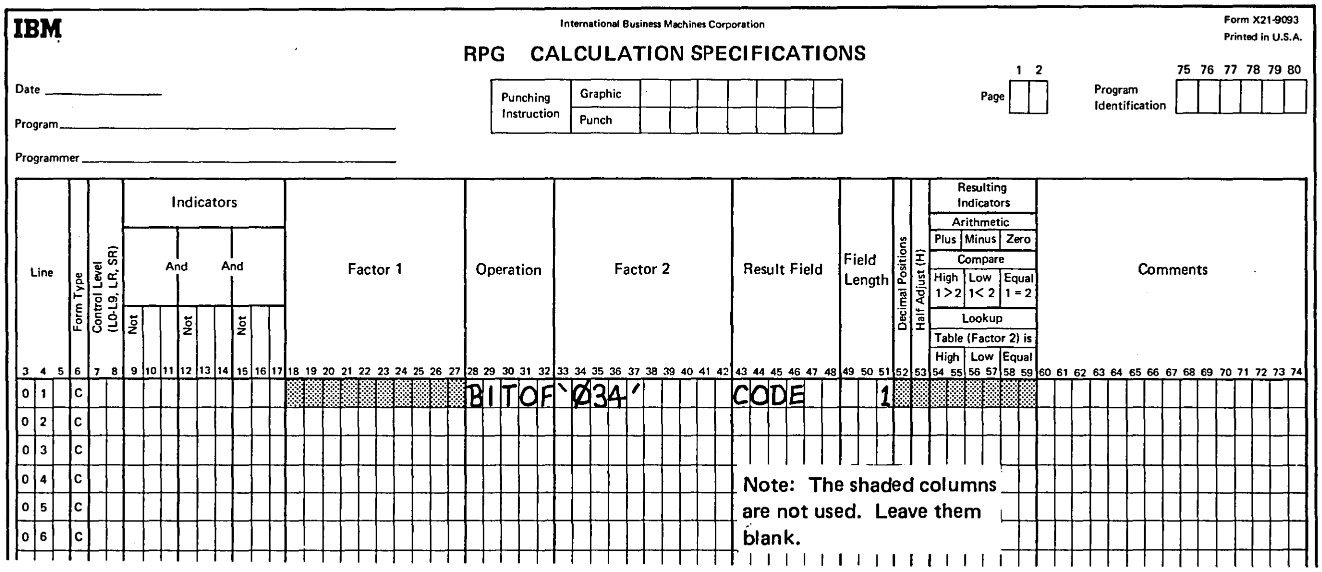
Figure 11-30 is a sample Calculation Sheet containing the BITOF operation code. This operation code causes specified bits identified in Factor 2 to turn off (set to 0) in a field named as the Result Field. In Figure 11-30, bits 0, 3, and 4 are turned off (set to 0) in the binary field labeled CODE.

All other specifications are the same as those specified under *BITON Operation Code.*

TESTB Operation Code

Figure 11-31 is a sample Calculation Sheet with the TESTB operation code. This operation code causes specified bits identified in Factor 2 to be tested for an off or on condition. Resulting indicators in columns 54-59 are set depending upon the conditions. At least one resulting indicator must be used with the TESTB operation, and as many as three can be named for one operation. Two indicators may be the same for one TESTB operation, but not three. Resulting indicators in these columns have the following meanings:

- Columns 54-55: An indicator in these columns is turned on if each bit in Factor 2 is off (set to 0).
- Columns 56-57: An indicator in these columns is turned on if two or more bits were tested and found to be of mixed status, some bits on and other bits off.
- Columns 58-59: An indicator in these columns is turned on if each bit in Factor 2 is on.



IBM International Business Machines Corporation

RPG CALCULATION SPECIFICATIONS

Form X21-9093
Printed in U.S.A.

Date ____ Program ____ Programmer ____

Punching Instruction: Graphic / Punch

Page 1 2 — Program Identification 75 76 77 78 79 80

Line (3-5)	Form Type (6)	Control Level (7-8)	Indicators (9-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field (43-48)	Field Length (49-51)	Decimal Positions (52)	Half Adjust (H) (53)	Resulting Indicators (54-59)	Comments (60-74)
01	C				BITOF	'Ø34'	CODE	1				
02	C											
03	C											
04	C											
05	C											
06	C											

Note: The shaded columns are not used. Leave them blank.

Figure 11-30. The BITOF Operation Code

In Figure 11-31, bits 4, 5, and 6 in the binary field named CODE are tested. Resulting indicator 66 is turned on if bits 4, 5, and 6 are off. If some are on and others off, resulting indicator 77 is turned on. If they are all on, resulting indicator 88 is turned on.

All other specifications are the same as those specified under *BITON Operation Code*. However, you need not define the Result Field as one position in length, since this is done when the field is used in a BITON or BITOF operation code.

Example

Fields are sometimes present in customer master files to indicate particular types of customers. When such a master file is created, each of the conditions indicating a particular customer type is represented in a card by one column. Since each card column occupies one byte of storage, four columns indicating customer types will be stored in four bytes of storage. You can use binary field operations to convert each one-byte card column to one bit of information on disk. Therefore, four bytes of information can be reduced to four bits of information on disk.

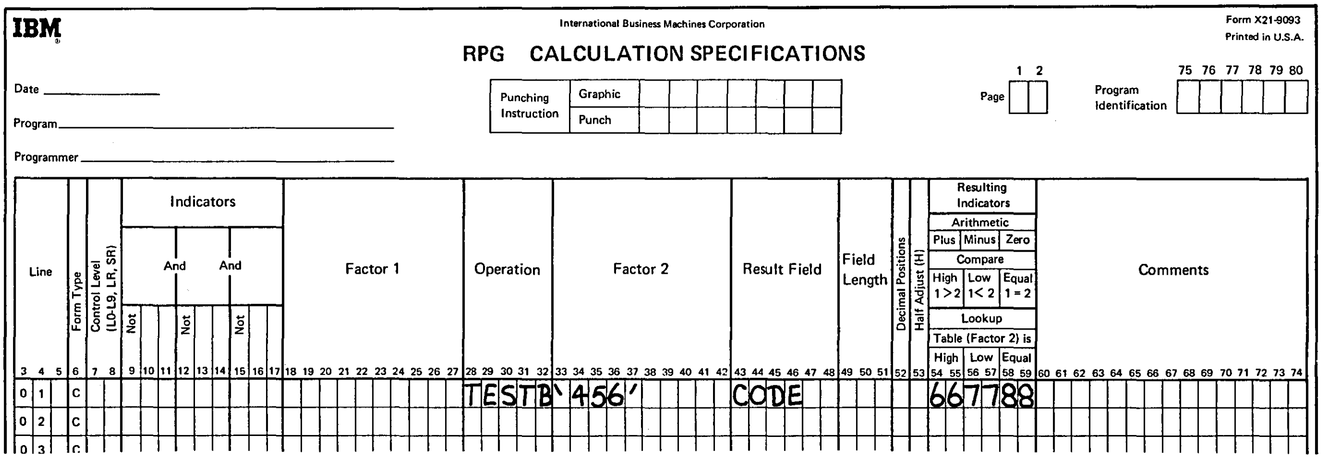
For example, assume you have a customer master file on cards. You have four columns containing the following information:

- Whether the customer is a wholesaler or retailer.
- If the customer is entitled to a discount.
- If orders should be checked by the credit department.
- If due to a bad payment history, the shipment should be sent cash on delivery.

Now you want to place the card file on disk, and the information from the four columns in four bits in a binary field labeled CHECK. The four columns will be labeled WHLSE, DSCT, CREDIT, and COD respectively. The following operations should be performed:

1. If WHLSE is equal to 1, turn on bit 0 in CHECK.
2. If DSCT is equal to 1, turn on bit 1 in CHECK.
3. If CREDIT is equal to 1, turn on bit 2 in CHECK.
4. If COD is equal to 1, turn on bit 3 in CHECK.

Figure 11-32 shows correct coding for this problem. Remember that before setting up data in a binary field, the binary field should be set to binary zero. This can be done by the BITOF instruction (Line 1, Figure 11-32).



IBM International Business Machines Corporation Form X21-9093 Printed in U.S.A.

RPG CALCULATION SPECIFICATIONS

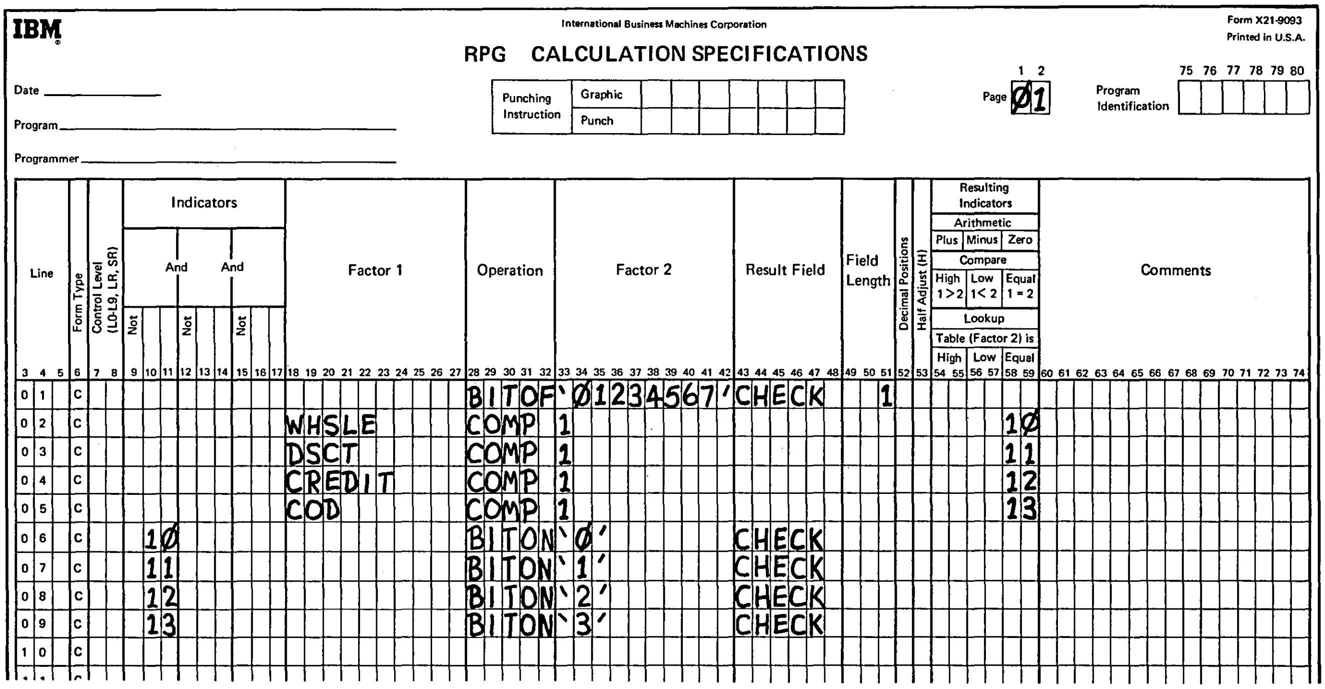
Date ____ Program ____ Programmer ____

Punching Instruction: Graphic / Punch

Page 1 2 Program Identification 75 76 77 78 79 80

Line (3-5)	Form Type (6)	Control Level (L0-L9, LR, SR) (7-8)	Indicators (9-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field (43-48)	Field Length (49-51)	Decimal Positions (52)	Half Adjust (H) (53)	Resulting Indicators High / Plus (54-55)	Low / Minus (56-57)	Equal / Zero (58-59)	Comments (60-74)
01	C				TESTB	'456'	CODE				66	77	88	
02	C													
03	C													

Figure 11-31. The TESTB Operation Code



IBM

International Business Machines Corporation

RPG CALCULATION SPECIFICATIONS

Form X21-9093
Printed in U.S.A.

Date

Program

Programmer

Punching Instruction	Graphic	
	Punch	

Page Ø1

Program Identification

Line	Form Type	Control Level (L0-L9, LR, SR)	Indicators (Not, And, Not, And, Not)	Factor 1	Operation	Factor 2	Result Field	Field Length	Decimal Positions	Half Adjust (H)	Resulting Indicators: Plus / High 1>2	Minus / Low 1<2	Zero / Equal 1=2	Comments
01	C				BITOF	'Ø1234567'	CHECK	1						
02	C			WHSLE	COMP	1							1Ø	
03	C			DSCT	COMP	1							11	
04	C			CREDIT	COMP	1							12	
05	C			COD	COMP	1							13	
06	C		1Ø		BITON	'Ø'	CHECK							
07	C		11		BITON	'1'	CHECK							
08	C		12		BITON	'2'	CHECK							
09	C		13		BITON	'3'	CHECK							
10	C													

Figure 11-32. Example of Binary Field Operations

Review 11. Binary Field Operations

1. What are bit switches used for?

2. Code the calculation specification to:

   a. Set on bits 4 and 7 in a field called TESTER.

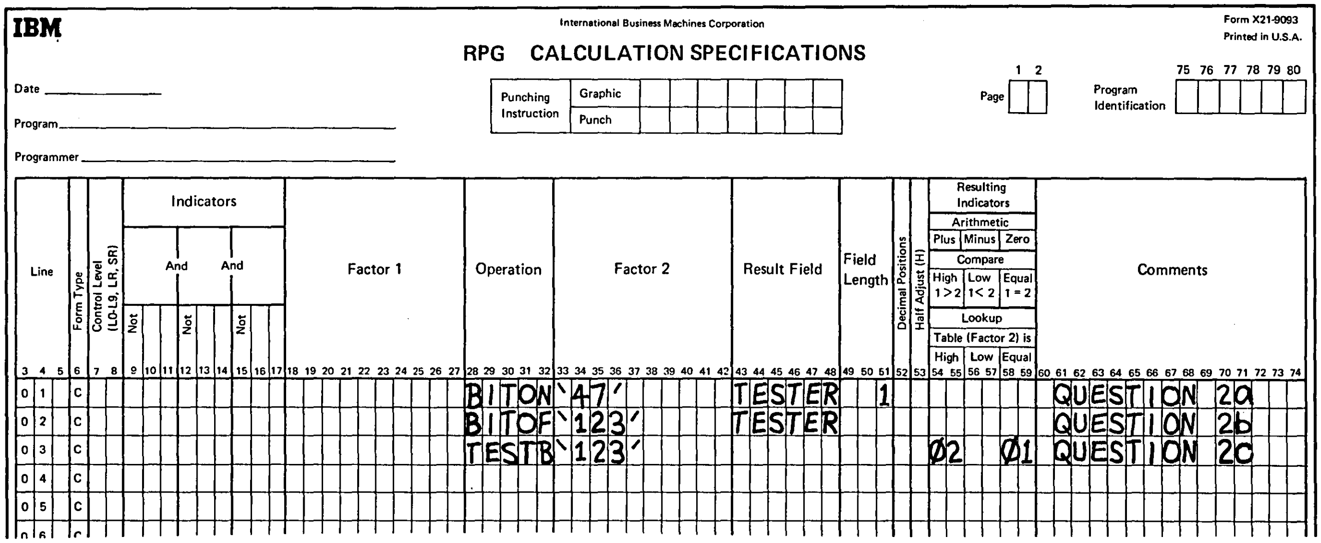
   b. Set off bits 1, 2, and 3 in TESTER.

   c. Test to see whether bits 1, 2, and 3 in TESTER are all on or all off. Set on indicator 01 if they are all on and set on indicator 02 if they are all off.

Answers to Review 11. Binary Field Operations

1. Bit switches are used to code and test for specified situations. With System/3 Disk System, they are stored in one-byte alphameric fields in storage and on disk. One example is credit information in an accounts receivable file. The first bit might mean a COD only; the second, payment due in 30 days; the third, credit limit $1000; etc. When these conditions are coded as bit switches they take up less disk space than single character codes that might be used in the same way.

2. See coding sheets.



IBM International Business Machines Corporation Form X21-9093 Printed in U.S.A.

RPG CALCULATION SPECIFICATIONS

Date ______ Program ______ Programmer ______

Punching Instruction: Graphic | Punch

Page (1 2) Program Identification (75 76 77 78 79 80)

Line (3-5)	Form Type (6)	Control Level (L0-L9, LR, SR) (7-8)	Indicators (9-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field (43-48)	Field Length (49-51)	Decimal Positions (52)	Half Adjust (H) (53)	Resulting Indicators: Plus / High 1>2 (54-55)	Minus / Low 1<2 (56-57)	Zero / Equal 1=2 (58-59)	Comments (60-74)
01	C				BITON	'47'	TESTER	1						QUESTION 2a
02	C				BITOF	'123'	TESTER							QUESTION 2b
03	C				TESTB	'123'					Ø2		Ø1	QUESTION 2C
04	C													
05	C													
06	C													

ALTERING THE ORDER OF OPERATIONS ON THE BASIS OF THE NEXT RECORD IN A FILE

Calculations to be performed may depend upon information in the record or the type of record to be processed next. For example, if, while processing a record, you know the next record is identical to the one being currently processed, you can bypass calculations for the current record.

The RPG II language has a special feature called *look ahead,* which extends the basic RPG II logic. It will allow the computer to look at information in the next record to be processed while it is processing the current record. This means that information in record B can be used while record A is being processed. By using this feature, you can write a program that uses information from the next record available for processing.

Processing Card or Disk Files

As the card (Card A) is read, data recorded on it is transferred to the input area. The card then moves on to the wait station (Figure 11-33). According to the RPG II program cycle, information is transferred from the input area to the processing area right before detail time. At detail time, then, calculations can be done on data from the card which is shown sitting in the wait station.

However, when look ahead is specified, another card (Card B) is read before detail-time operations are performed in the current cycle. Card A is stacked and information from Card A is moved to the processing area. Then information on Card B just read is transferred to the input area and is available to use for determining what detail calculations should be done on data from Card A, now in the stacker.

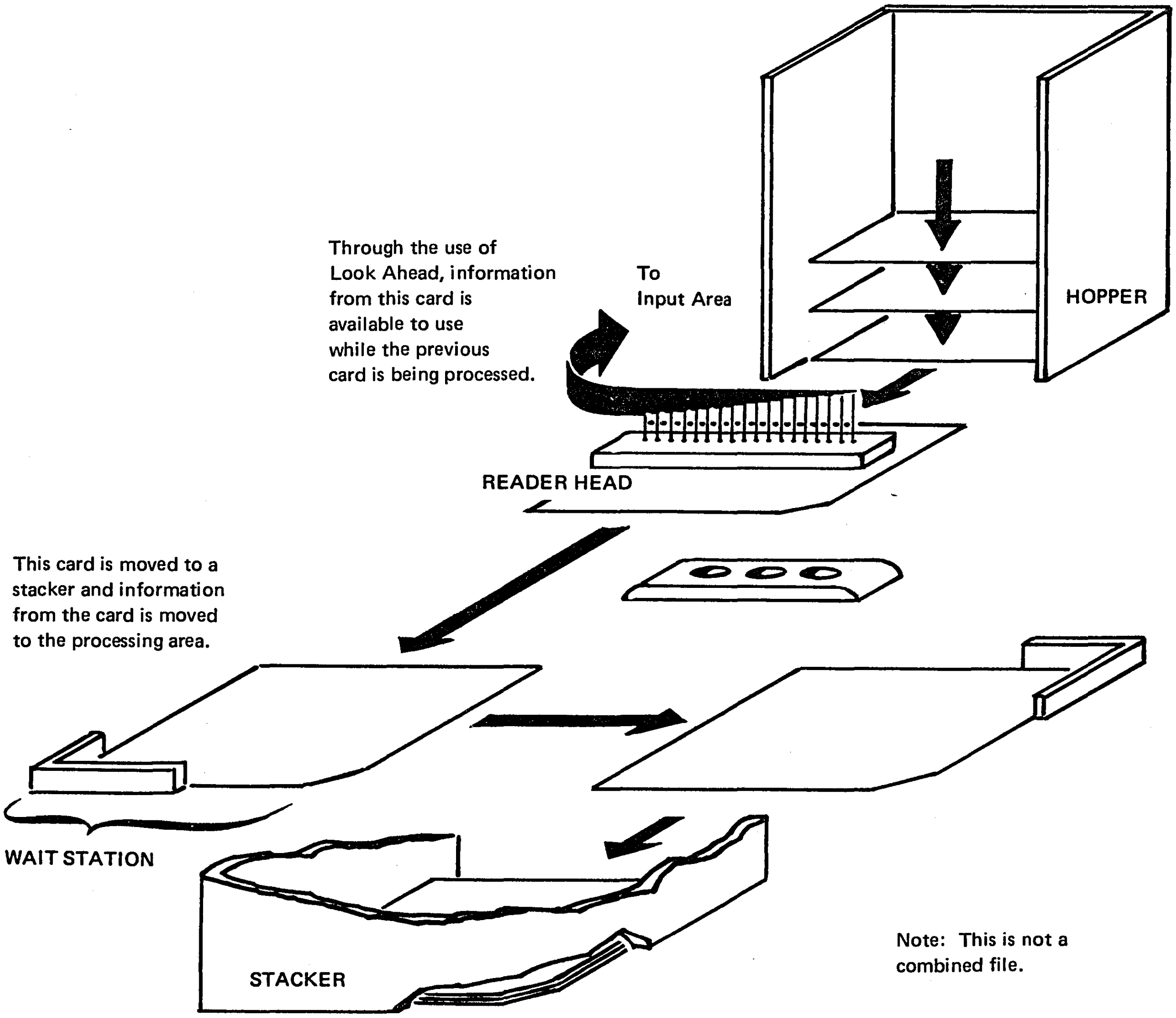

Through the use of Look Ahead, information from this card is available to use while the previous card is being processed.
To Input Area
HOPPER
READER HEAD
This card is moved to a stacker and information from the card is moved to the processing area.
WAIT STATION
STACKER
Note: This is not a combined file.

Figure 11-33. The Look Ahead Function

Figure 11-34 shows processing of three records from two disk input files, one primary and one secondary. The first record from each file is read (Figure 11-34, insert A). Figure 11-34 shows records being selected for processing. The records available for look ahead during the processing of these records are:

Record Processed	*Records Available*
P1	P2 and S1
P2	P3 and S1
S1	P3 and S2

In general, when the record being processed is from an input file, the next record in the input file is available as are the records which were read but not selected from the other files.

Checking for Duplicates

Duplicate records or records with duplicate fields are sometimes considered erroneous. Only one of the duplicates should be used for the job.

Consider, for example, the case of a company which has a large turnover in inventory items. Quite frequently new items are added and others deleted from the inventory. A number for a deleted part is to be assigned to a new part. Some mistakes have occurred, however, and one part number has been assigned to two different items. As a result of this error, inventory balances for these items have not been updated correctly, and errors have been made on customer invoices. If this situation is possible, a regular check should be made for duplicate part numbers.

Each month, a report is created showing the complete inventory. All part numbers are listed on the report. You could look through the report to check for duplicate part numbers, but it would be easier and more accurate if you could add a few specifications that would check for duplicates and indicate on the report which item numbers are duplicate.

By using the look ahead feature you have access to information that is coming up. You can then use this information to determine what operations to do. If you are processing a record with part number 64322, and you know that the next record also has part number 64322, you can print a message indicating duplicate part numbers, then halt. But, if you are processing the record with part number 64322 and you do not know that the next record also has part number 64322, you can do nothing special because you are not aware that you are processing a record which contains a duplicate entry.

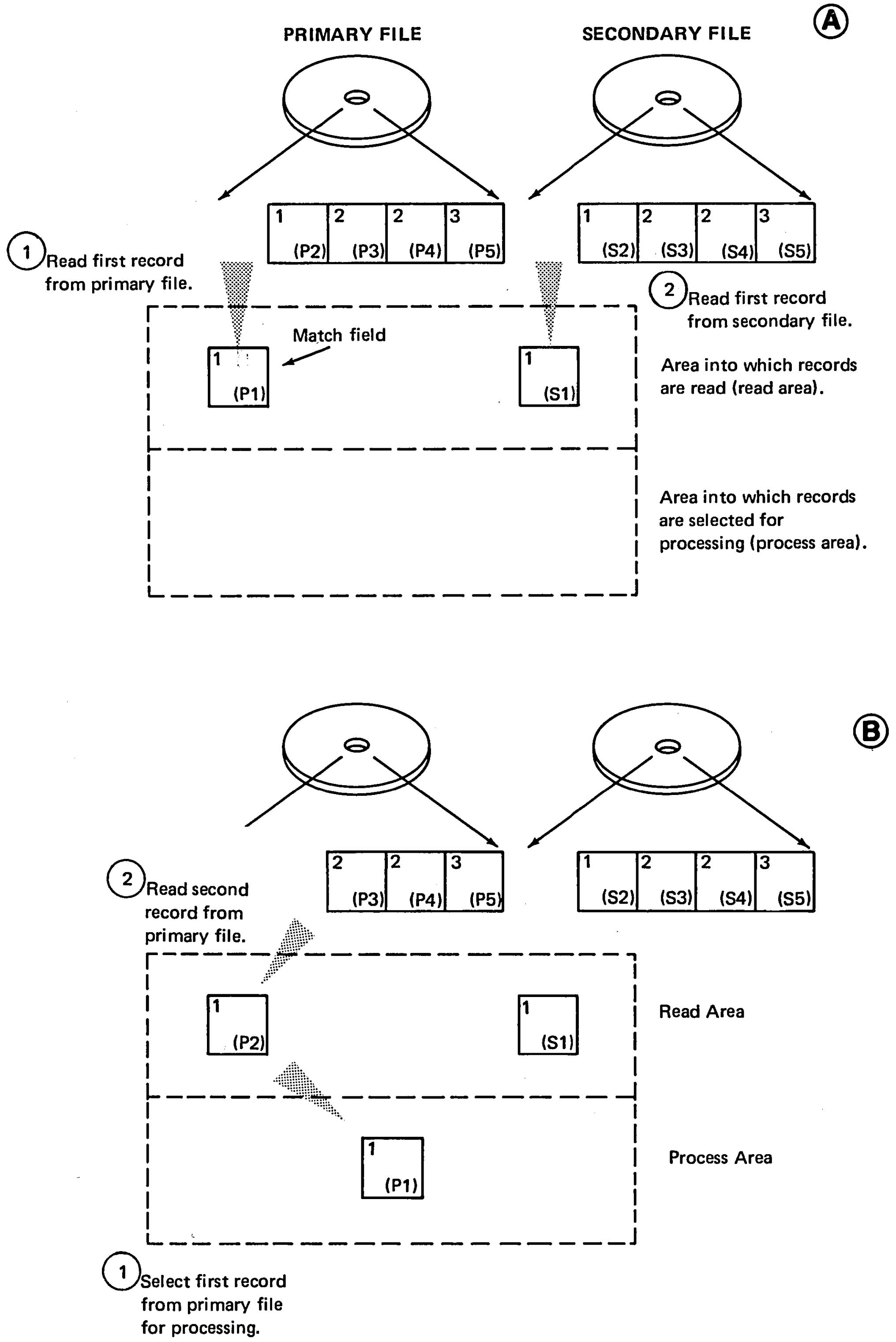

PRIMARY FILE
SECONDARY FILE
A
1 (P2)
2 (P3)
2 (P4)
3 (P5)
1 (S2)
2 (S3)
2 (S4)
3 (S5)
1 Read first record from primary file.
2 Read first record from secondary file.
Match field
1 (P1)
1 (S1)
Area into which records are read (read area).
Area into which records are selected for processing (process area).
B
2 Read second record from primary file.
2 (P3)
2 (P4)
3 (P5)
1 (S2)
2 (S3)
2 (S4)
3 (S5)
1 (P2)
1 (S1)
Read Area
1 (P1)
Process Area
1 Select first record from primary file for processing.

Figure 11-34. Available Records: Two Input Files (part 1 of 2)

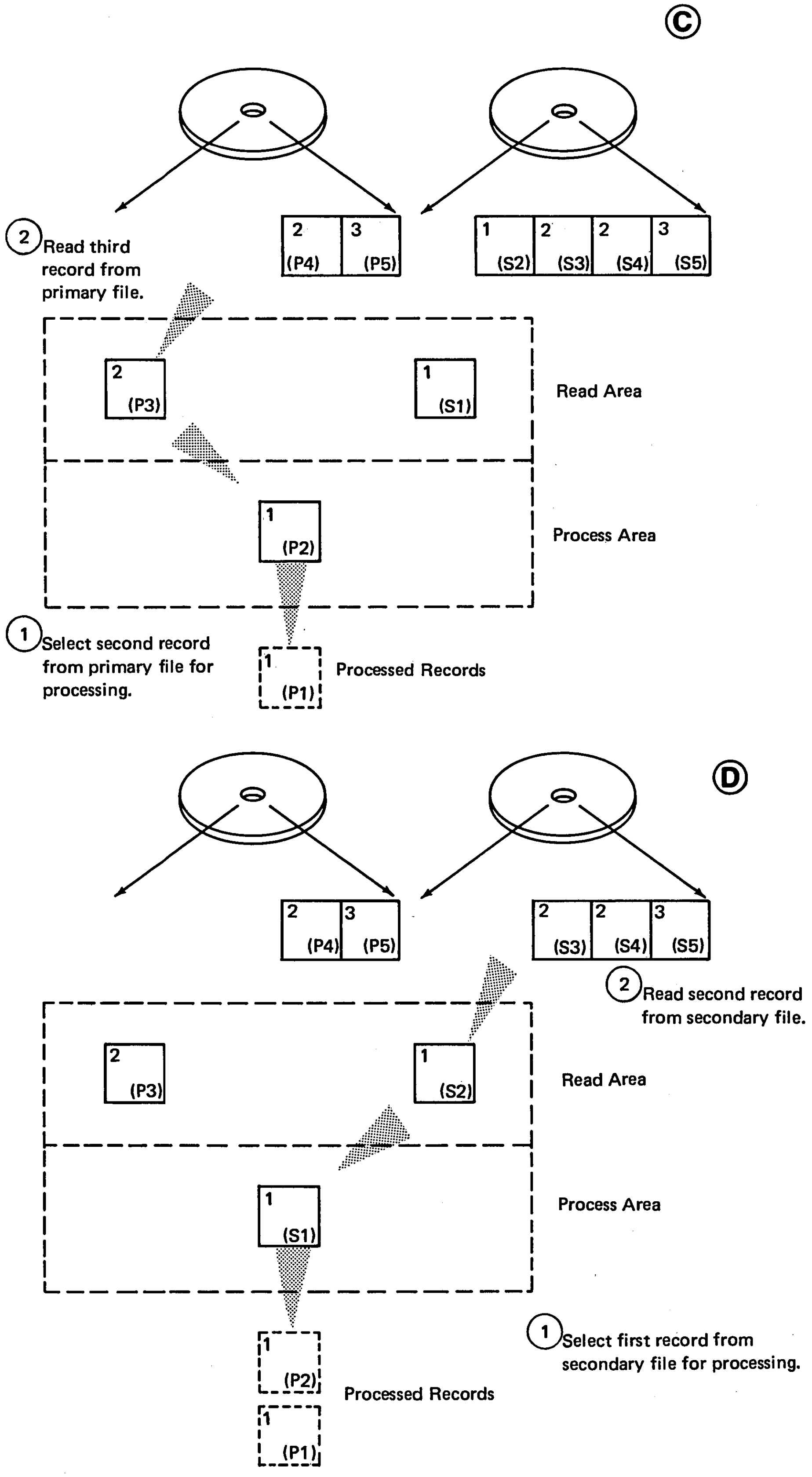


Figure 11-34. Available Records: Two Input Files (part 2 of 2)

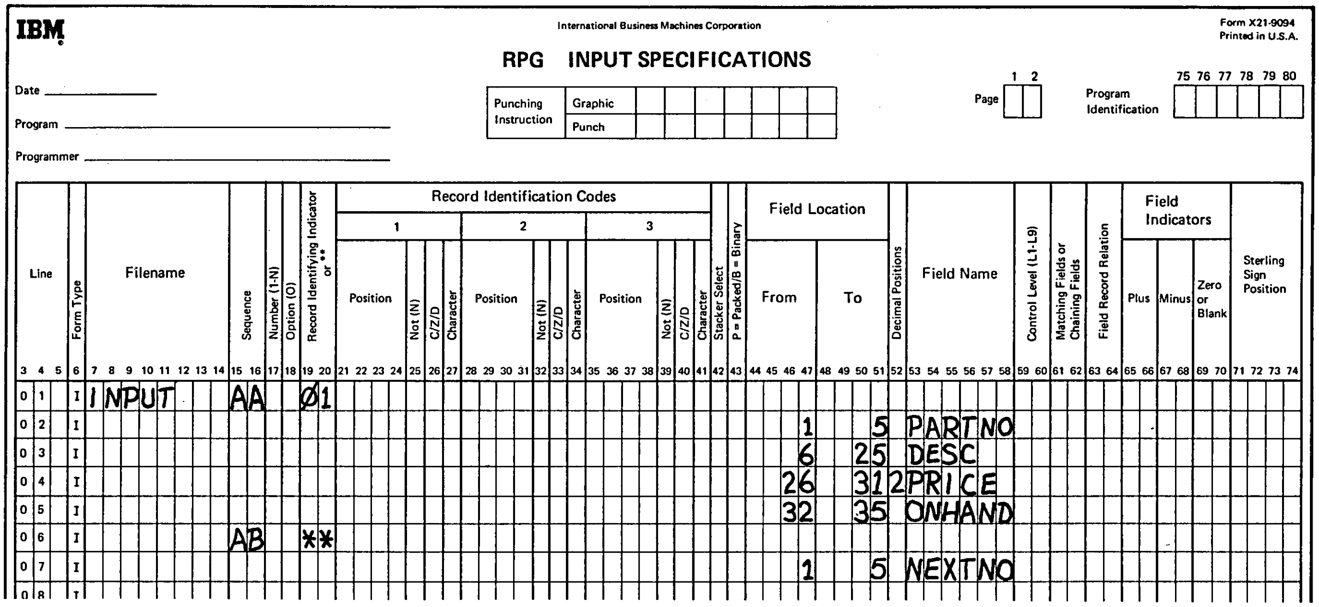
Writing Specifications for Look Ahead

Any field which you want to look at in the next record to be processed must be defined as a look ahead field. If that field is also used in normal processing (other than as a look ahead field), it must be defined in the normal way. Thus, most look ahead fields will be specified twice.

Figure 11-35, lines 01-05, shows specifications needed to describe the input file used in the inventory listing job. When checking for duplicates, PARTNO is the field you want to use when looking ahead at the next field; therefore, PARTNO must be defined as a look ahead field. The specifications in Figure 11-30, lines 06-07, do this.

All look ahead fields must be defined as being in a record type different from the others defined. This is done by using a unique alphabetic sequence entry in columns 15-16. No record identifying indicator (01-99) can be used. A double asterisk (**) is placed in columns 19-20 to specify that the fields described in the following lines are look ahead fields. Field location is also specified for look ahead fields.

Every look ahead field must be named, but the name given must be different than when it was described as a normal input field. The same field is given two names so that you can distinguish between the field on the record being processed and that same field (the look ahead field) on the record that is to be processed next (Figure 11-36).



IBM

International Business Machines Corporation

Form X21-9094
Printed in U.S.A.

RPG INPUT SPECIFICATIONS

Date

Program

Programmer

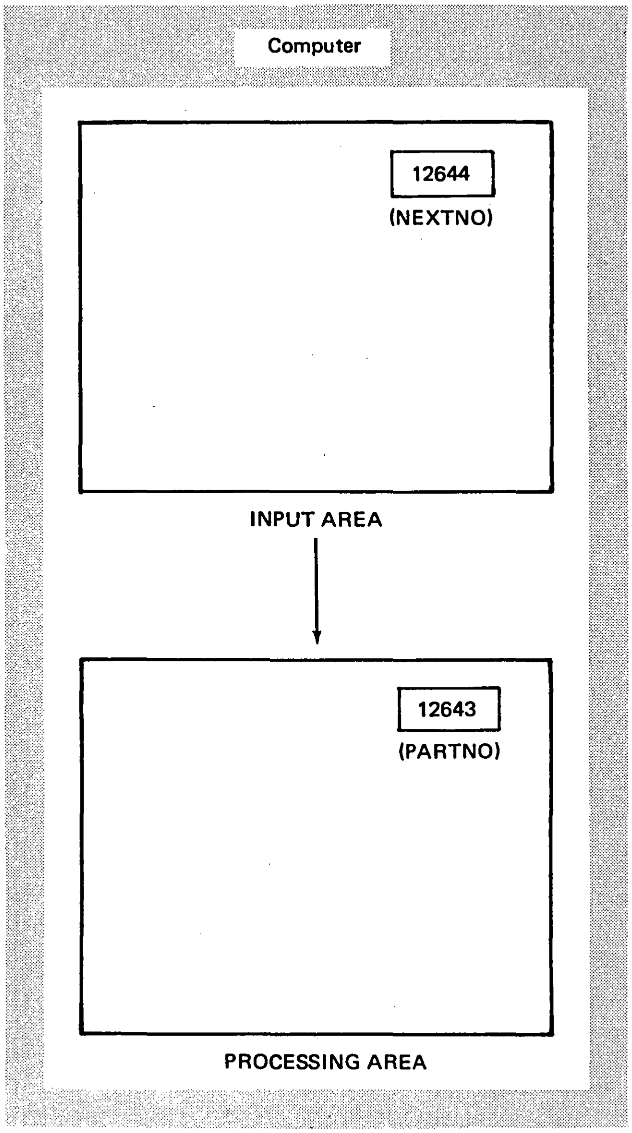
Punching Instruction: Graphic / Punch

Page 1 2

Program Identification 75 76 77 78 79 80

Line (3-5)	Form Type (6)	Filename (7-14)	Sequence (15-16)	Number (1-N) (17)	Option (O) (18)	Record Identifying Indicator or ** (19-20)	Record Identification Codes 1: Position (21-24)	Not (N) (25)	C/Z/D (26)	Character (27)	2: Position (28-31)	Not (N) (32)	C/Z/D (33)	Character (34)	3: Position (35-38)	Not (N) (39)	C/Z/D (40)	Character (41)	Stacker Select (42)	P = Packed/B = Binary (43)	Field Location From (44-47)	To (48-51)	Decimal Positions (52)	Field Name (53-58)	Control Level (L1-L9) (59-60)	Matching Fields or Chaining Fields (61-62)	Field Record Relation (63-64)	Field Indicators Plus (65-66)	Minus (67-68)	Zero or Blank (69-70)	Sterling Sign Position (71-74)
01	I	INPUT	AA			Ø1																									
02	I																				1	5		PARTNO							
03	I																				6	25		DESC							
04	I																				26	31	2	PRICE							
05	I																				32	35		ONHAND							
06	I		AB			**																									
07	I																				1	5		NEXTNO							
08	I																														

Figure 11-35. Look Ahead Specifications



NEXTNO refers to columns 1-5 in the card to be processed next.

PARTNO refers to columns 1-5 in the card being currently processed.

Figure 11-36. Look Ahead Field: A Field With Two Names

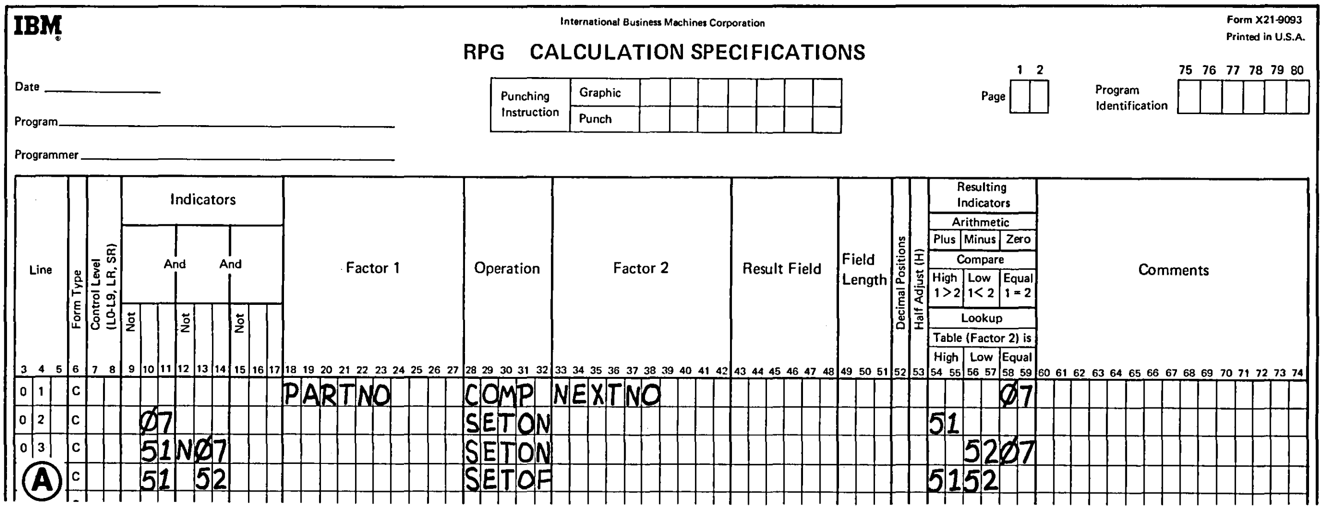
Using Look Ahead Information

Now that you have specified the look ahead field, you can use it as you would any other field. The only exceptions are that you cannot use it as a result field in calculations, nor can it be blanked after for output.

For the listing job, you have to make a comparison between part numbers from two records. If PARTNO on the record being processed is the same as NEXTNO on the next record to be processed, you wish to print a message indicating duplicate entries. If the PARTNO and NEXTNO fields do not match, there are no duplicates for that part number, and the item is merely listed.

Figure 11-37 shows specifications for the job. The operation in line 01 of the Calculation Sheet compares the part number on the record being processed (PARTNO) to the part number on the record coming next (NEXTNO). If they are equal, indicator 07 is turned on. Notice on the Output-Format Sheet that when 07 is on, the word *duplicate* is printed.

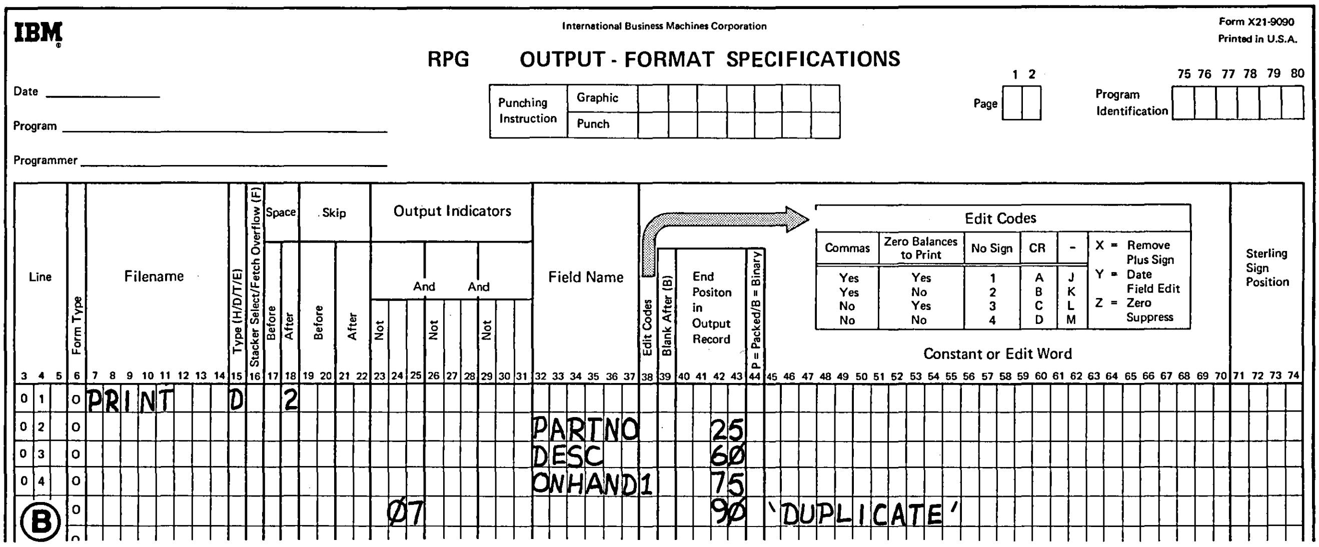
The SETON and SETOF operations in lines 02-04 of the Calculation Sheet are used so that the computer will *duplicate* the record when the second record having the duplicate part number is processed.



IBM International Business Machines Corporation Form X21-9093 Printed in U.S.A.

RPG CALCULATION SPECIFICATIONS

Line	Form Type	Control Level	Indicators	Factor 1	Operation	Factor 2	Result Field	Field Length	Decimal Positions	Half Adjust	High (54-55)	Low (56-57)	Equal (58-59)	Comments
01	C			PARTNO	COMP	NEXTNO							Ø7	
02	C		Ø7		SETON						51			
03	C		51NØ7		SETON							52	Ø7	
Ⓐ	C		51 52		SETOF						51	52		



IBM International Business Machines Corporation Form X21-9090 Printed in U.S.A.

RPG OUTPUT - FORMAT SPECIFICATIONS

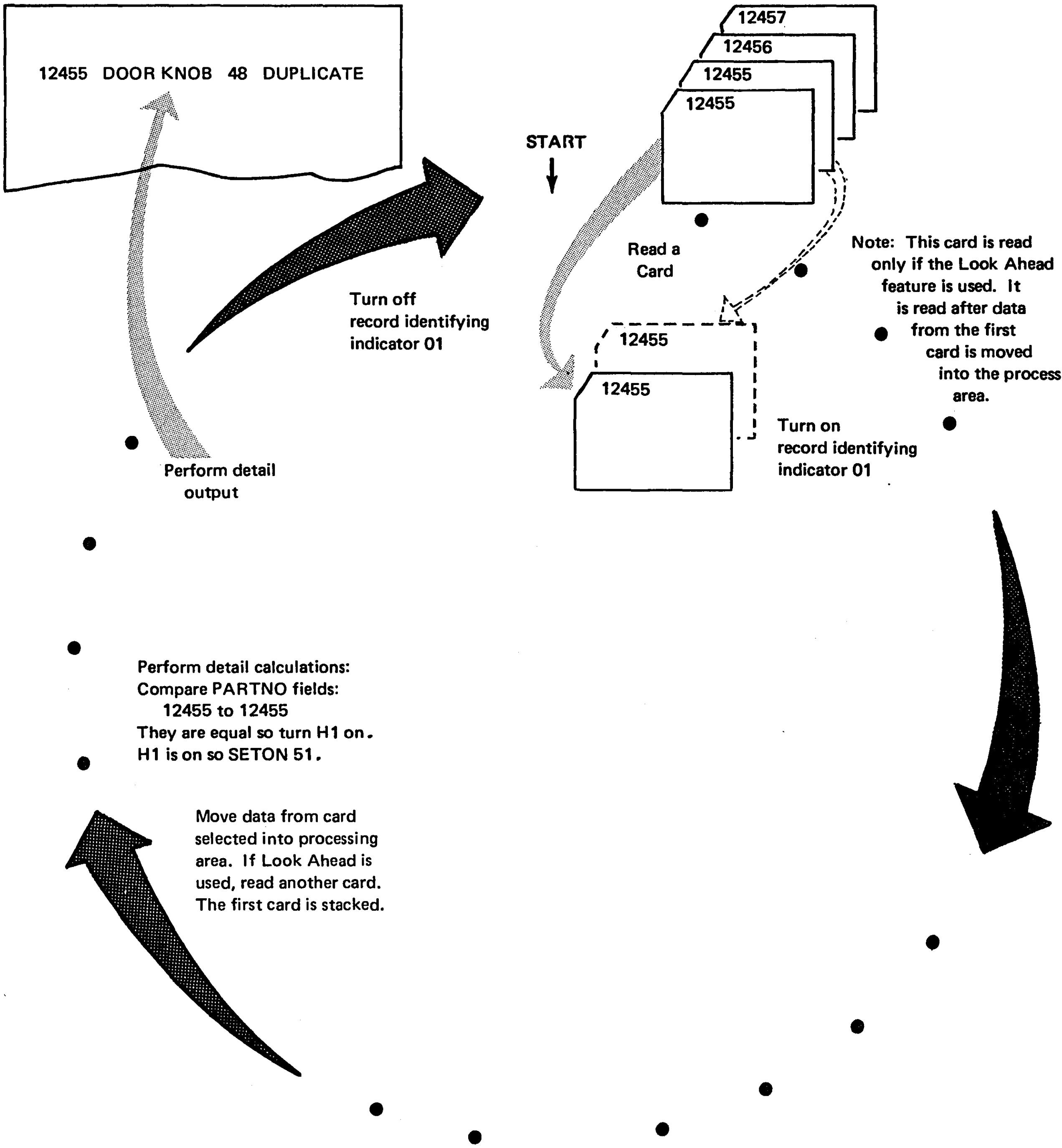
Line	Form Type	Filename	Type (H/D/T/E)	Space Before	Space After	Skip	Output Indicators	Field Name	End Position in Output Record	Constant or Edit Word
01	O	PRINT	D		2					
02	O							PARTNO	25	
03	O							DESC	6Ø	
04	O							ONHAND1	75	
Ⓑ	O						Ø7		9Ø	'DUPLICATE'

Figure 11-37. Using Information from the Look Ahead Field to Check for Duplicates

Consider, for example, records A1, A2, and B. The first two records are duplicates; the third is not. When A1 is processed, the program looks ahead to A2 and, by comparing, knows that A2 is the same as A1. When A2 is processed, the program looks ahead to B. The compare will say that A2 is not a duplicate since it does not match B1. But A2 really is a duplicate because it is the same as A1. Thus, when processing A1, you have to set an indicator which will be on when A2 is processed and which will indicate that A2 is a duplicate since it matches the previous record.

When PARTNO equals NEXTNO, 07 turns on. This, in turn, causes indicator 51, which is used to indicate that a duplicate record is processed, to turn on. During the next program cycle, the compare does not indicate duplicates; therefore 07 is not on. But 51 *is* on, meaning that the record being processed is a duplicate since the part number on it matched the part number on the previous record. Therefore, 07 is set on. Remember 07 conditions those output operations which are to be done for duplicates.

In line 04, indicator 51 is set off so that it will not indicate duplicates in the following cycle. Indicator 51 is necessary so that 51 will be set off for the *last* duplicate record, not the first. Figure 11-38 shows the program logic for this job.



Read a Card

Note: This card is read only if the Look Ahead feature is used. It is read after data from the first card is moved into the processing area.

Turn on record identifying indicator 01

Move data from card selected into processing area. If Look Ahead is used, read another card. The first card is stacked.

Perform detail calculations:
Compare PARTNO fields:
12455 to 12455
They are equal so turn H1 on.
H1 is on so SETON 51.

Perform detail output

Turn off record identifying indicator 01

Figure 11-38. Logic for Look Ahead (part 1 of 3)

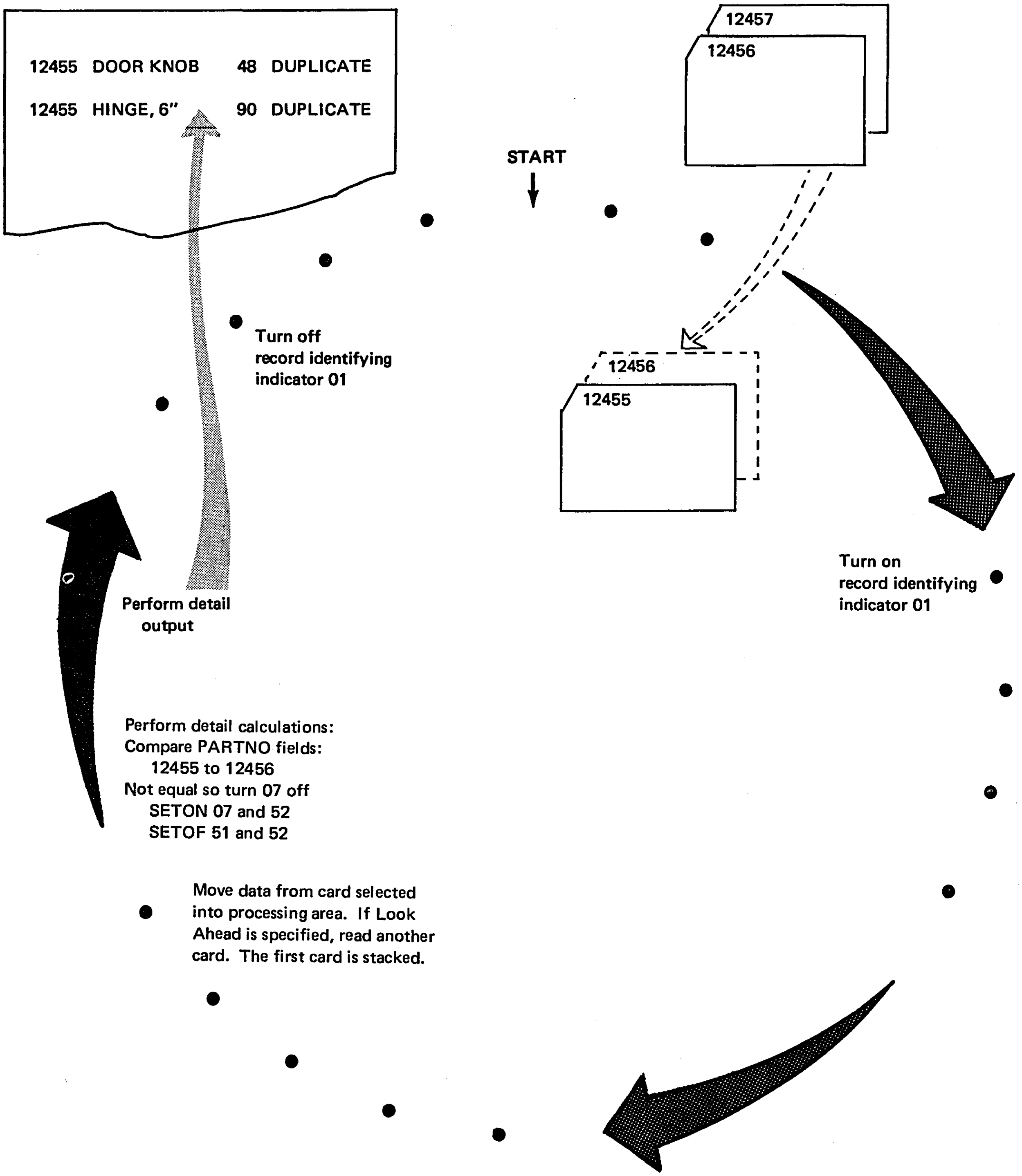
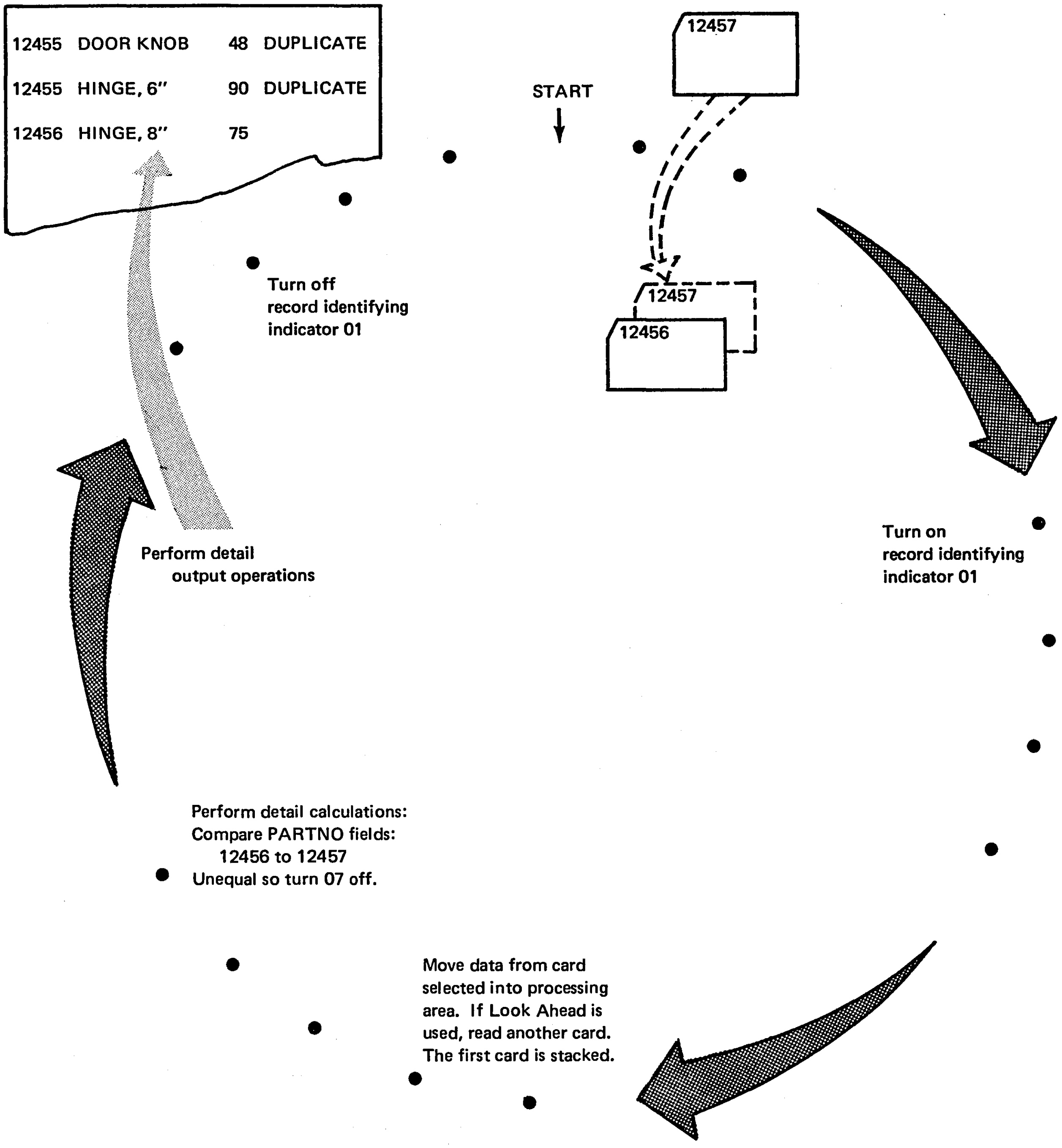


Figure 11-38. Logic for Look Ahead (part 2 of 3)



```
12455  DOOR KNOB    48  DUPLICATE
12455  HINGE, 6"    90  DUPLICATE
12456  HINGE, 8"    75
```

Turn on record identifying indicator 01

Move data from card selected into processing area. If Look Ahead is used, read another card. The first card is stacked.

Perform detail calculations:
Compare PARTNO fields:
12456 to 12457
Unequal so turn 07 off.

Perform detail output operations

Turn off record identifying indicator 01

Figure 11-38. Logic for Look Ahead (part 3 of 3)

Doing Special Operations for Only One Record in a Group

It is often important to know if and when you are processing the only record in a group. The job described in the following paragraphs is such a case.

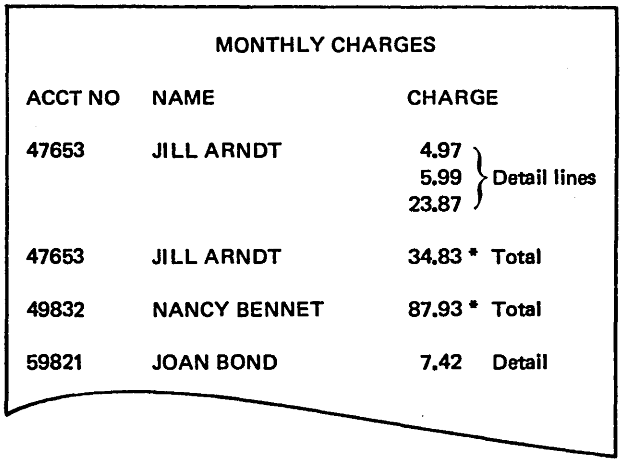
A report is prepared showing charges made by customers during the week (Figure 11-39). The input file is organized in ascending order by customer number. During the month some customers will have made one charge; others several.

When only one charge is made per customer, the total line is nearly a duplicate of the only detail line. In this case, you do not need to print both the detail and total lines because the total line will do.

But how will you know during any one program cycle whether the current record is the only one in a group? You can find out by looking at information on the next record.

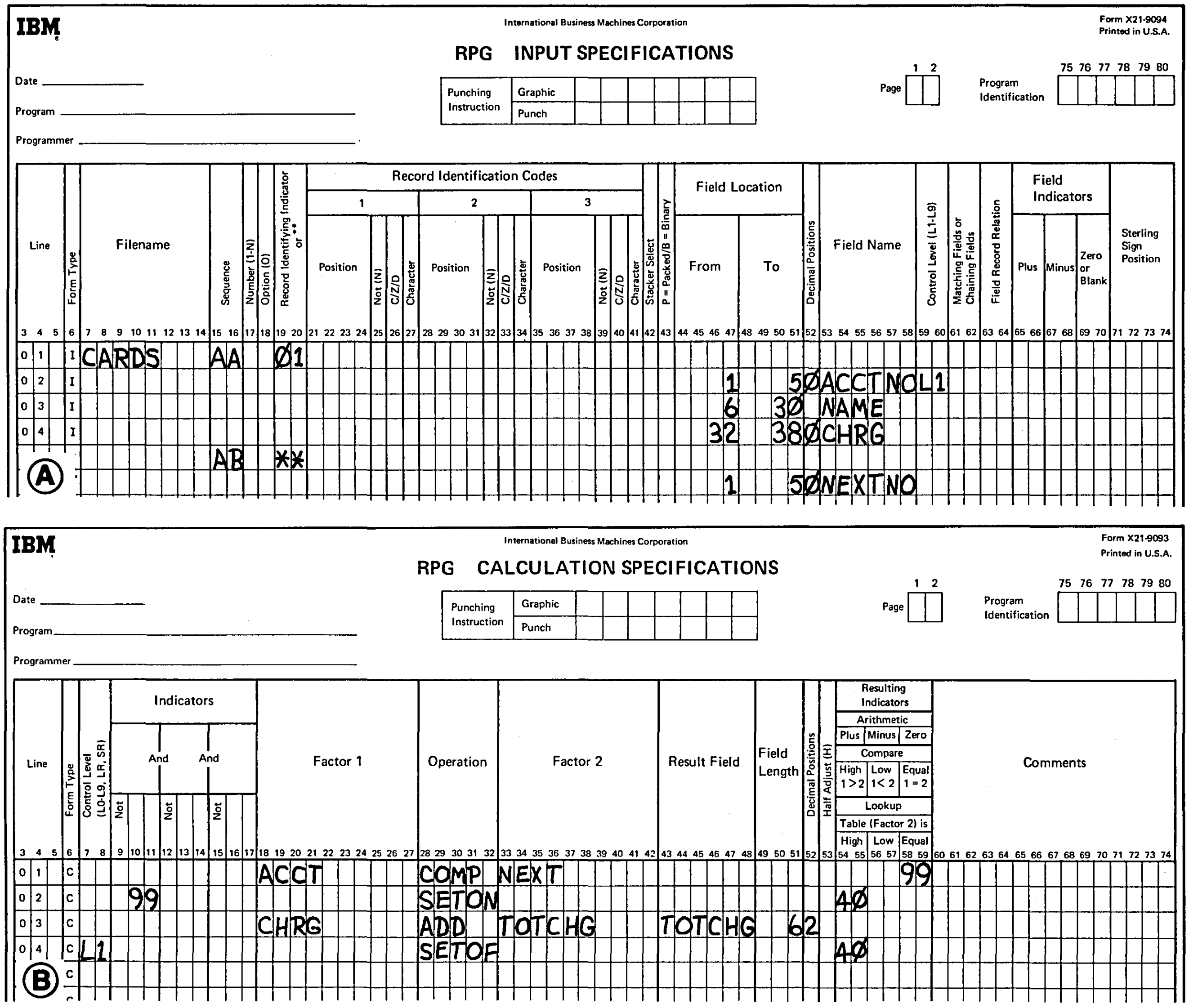
Remember that any time it is necessary to use information from the next record available for processing in order to determine what to do while processing the current record, you must use the look ahead feature. Account number is established as a look ahead field in this job. Any look ahead field specified applies to all record types. Thus each record read contains information that will be looked at before the record itself is processed. By looking ahead into this field you will know whether or not the next record to be processed is part of a new group.

Whenever a card is read, the current ACCT field is compared to the one coming up. If the fields are equal, you know you are processing a record that is not the only one in a group. Therefore, a detail line should print. If the ACCT fields are not equal, however, the current card is the only one in the group, and the detail line should not print. Figure 11-40 shows the specifications for the job.



```
                     MONTHLY CHARGES

ACCT NO    NAME                     CHARGE

47653      JILL ARNDT                 4.97 )
                                      5.99  > Detail lines
                                     23.87 )

47653      JILL ARNDT                34.83 *  Total

49832      NANCY BENNET              87.93 *  Total

59821      JOAN BOND                  7.42    Detail
```

Figure 11-39. Format of Monthly Charges Report



IBM — International Business Machines Corporation — Form X21-9094 Printed in U.S.A.

RPG INPUT SPECIFICATIONS

Date ____ Program ____ Programmer ____

Punching Instruction: Graphic / Punch — Page 1 2 — Program Identification 75 76 77 78 79 80

Line	Form Type	Filename	Sequence	Record Identifying Indicator or **	Field Location From	Field Location To	Decimal Positions	Field Name	Control Level (L1-L9)
0 1	I	CARDS	AA	Ø1					
0 2	I				1	5	Ø	ACCTNO	L1
0 3	I				6	30		NAME	
0 4	I				32	38	Ø	CHRG	
Ⓐ			AB	**					
					1	5	Ø	NEXTNO	

IBM — International Business Machines Corporation — Form X21-9093 Printed in U.S.A.

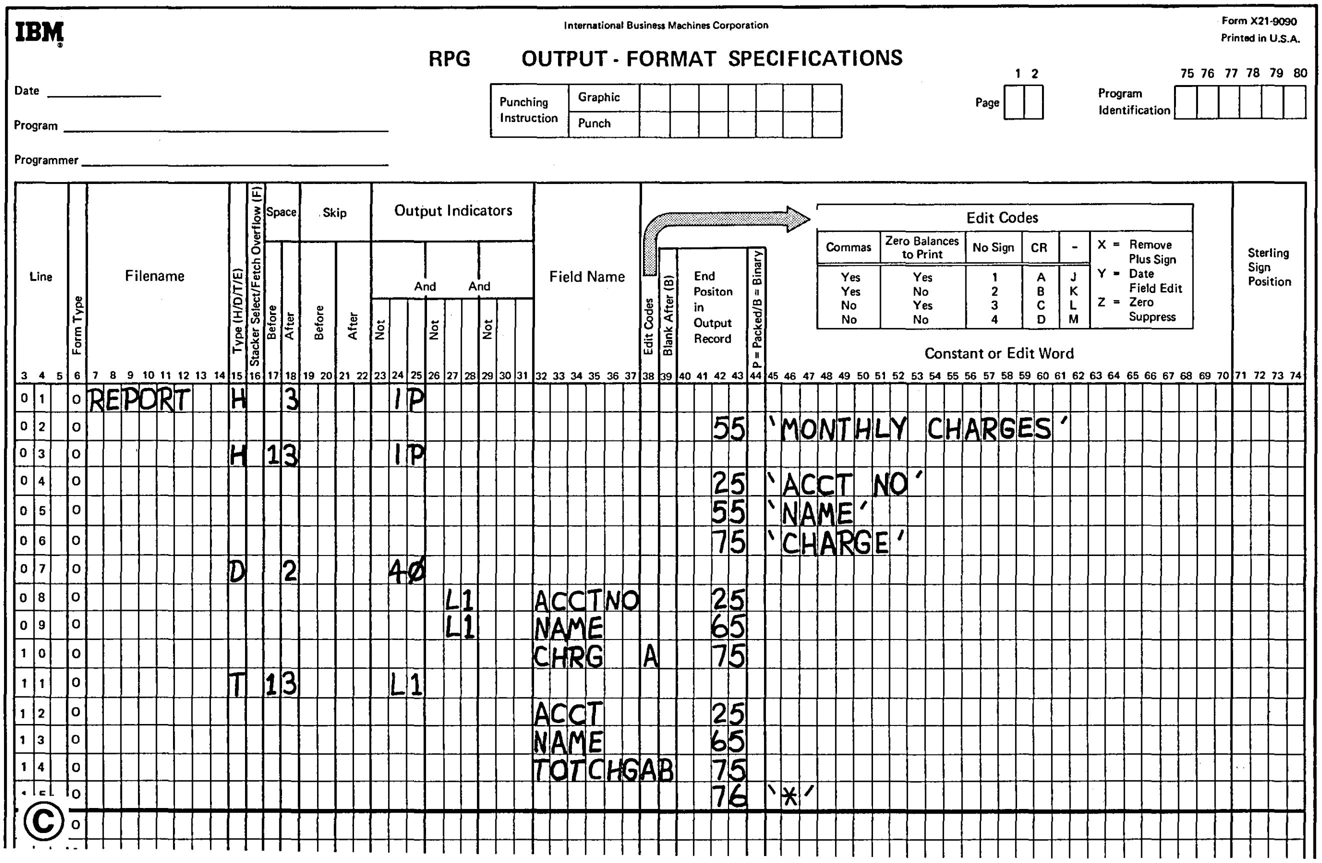
RPG CALCULATION SPECIFICATIONS

Date ____ Program ____ Programmer ____

Punching Instruction: Graphic / Punch — Page 1 2 — Program Identification 75 76 77 78 79 80

Line	Form Type	Control Level	Indicators	Factor 1	Operation	Factor 2	Result Field	Field Length	Decimal Positions	Resulting Indicators Plus / High	Minus / Low	Zero / Equal
0 1	C			ACCT	COMP	NEXT						99
0 2	C		99		SETON					4Ø		
0 3	C			CHRG	ADD	TOTCHG	TOTCHG	6	2			
0 4	C	L1			SETOF					4Ø		
Ⓑ	C											

Figure 11-40. Using Look Ahead to Find the First and Only Card in a Group (part 1 of 2)



IBM

International Business Machines Corporation

Form X21-9090
Printed in U.S.A.

RPG OUTPUT - FORMAT SPECIFICATIONS

Date ______________

Program ______________

Programmer ______________

Punching Instruction	Graphic							
	Punch							

Page (1 2) □□

Program Identification (75 76 77 78 79 80) □□□□□□

Edit Codes						
Commas	Zero Balances to Print	No Sign	CR	-		X = Remove Plus Sign
Yes	Yes	1	A	J		Y = Date Field Edit
Yes	No	2	B	K		Z = Zero Suppress
No	Yes	3	C	L		
No	No	4	D	M		

Line (3-5)	Form Type (6)	Filename (7-14)	Type (H/D/T/E) (15)	Stacker Select/Fetch Overflow (F) (16)	Space Before (17)	Space After (18)	Skip Before (19-20)	Skip After (21-22)	Output Indicators: Not (23)	(24-25)	And: Not (26)	(27-28)	And: Not (29)	(30-31)	Field Name (32-37)	Edit Codes (38)	Blank After (B) (39)	End Position in Output Record (40-43)	P = Packed/B = Binary (44)	Constant or Edit Word (45-70)	Sterling Sign Position (71-74)
01	O	REPORT	H			3				1P											
02	O																	55		'MONTHLY CHARGES'	
03	O		H		1	3				1P											
04	O																	25		'ACCT NO'	
05	O																	55		'NAME'	
06	O																	75		'CHARGE'	
07	O		D			2				4Ø											
08	O											L1			ACCTNO			25			
09	O											L1			NAME			65			
10	O														CHRG	A		75			
11	O		T		1	3				L1											
12	O														ACCT			25			
13	O														NAME			65			
14	O														TOTCHG	A	B	75			
15	O																	76		'*'	
	O																				

Figure 11-40. Using Look Ahead to Find the First and Only Card in a Group (part 2 of 2)

Doing Special Operations for the Last Record in a Group

In some jobs, it may be necessary to do special operations on the last record of a control group. This is because, unless the last record in the control group is of a different type (have different record identification), it is impossible to know when you are processing the last record in the group. When all records are of the same type, you have to know what is on the next record before you know whether or not you are processing the last record in the group. To look at information in the next record, you must use the look ahead feature.

Figure 11-41 shows four cards which are to be processed. The first three belong to one control group; the fourth is the beginning of the next group. The last card of the group (the third card in this case) requires special processing. In order to know when the last card in the group is to be processed, you must look at the account number in the next card. When it is different, you know that the last card in the group is being processed.

Additional Points to Consider About Look Ahead

You must consider the following things when you are planning to use look ahead:

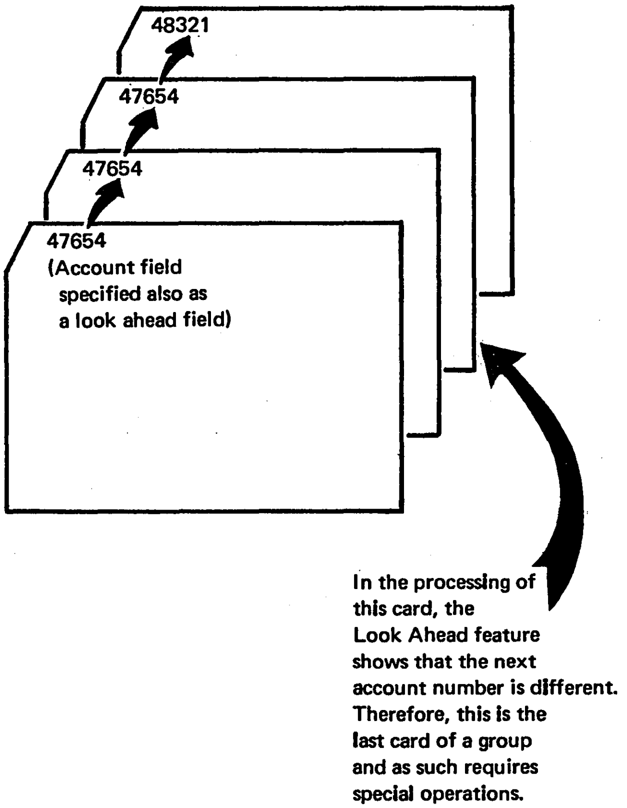
- Look ahead may be used with update or combined files, but the results are different than look ahead with input files. When look ahead is used with a combined or update file, and that file is the only input file in the program, the field looked at is not on the next record, but on the record currently being processed. Therefore, there is little use for look ahead with update or combined files in a single file program.

- Look ahead is never used with chained, demand, or output files.

- Only one look ahead record type specification may be used for a file. There may be several fields listed under that one record type specification however.

- Any look ahead fields specified apply to all types of records in the file. Therefore, all records read from the file will be treated as if they have look ahead fields.

- Look ahead is used more in jobs requiring two files than in jobs requiring one file.



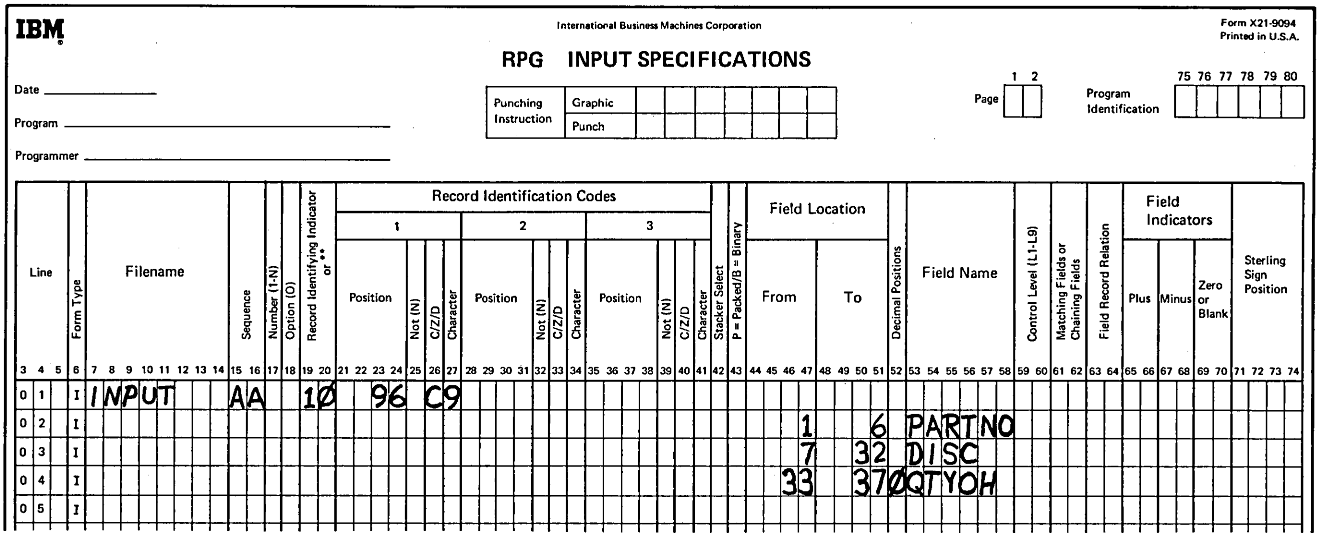
In the processing of this card, the Look Ahead feature shows that the next account number is different. Therefore, this is the last card of a group and as such requires special operations.

Figure 11-41. Using Look Ahead to Find Last Card in a Group

Review 11. Look Ahead

1. Basically, what does the look ahead facility allow you to do? What limitations apply to its use?

2. To the input specifications given add those which will allow you to look ahead in order to read the next part number (PARTNO) and next code in column 96.



IBM

International Business Machines Corporation

Form X21-9094
Printed in U.S.A.

RPG INPUT SPECIFICATIONS

Date

Program

Programmer

Punching Instruction	Graphic							
	Punch							

Page 1 2

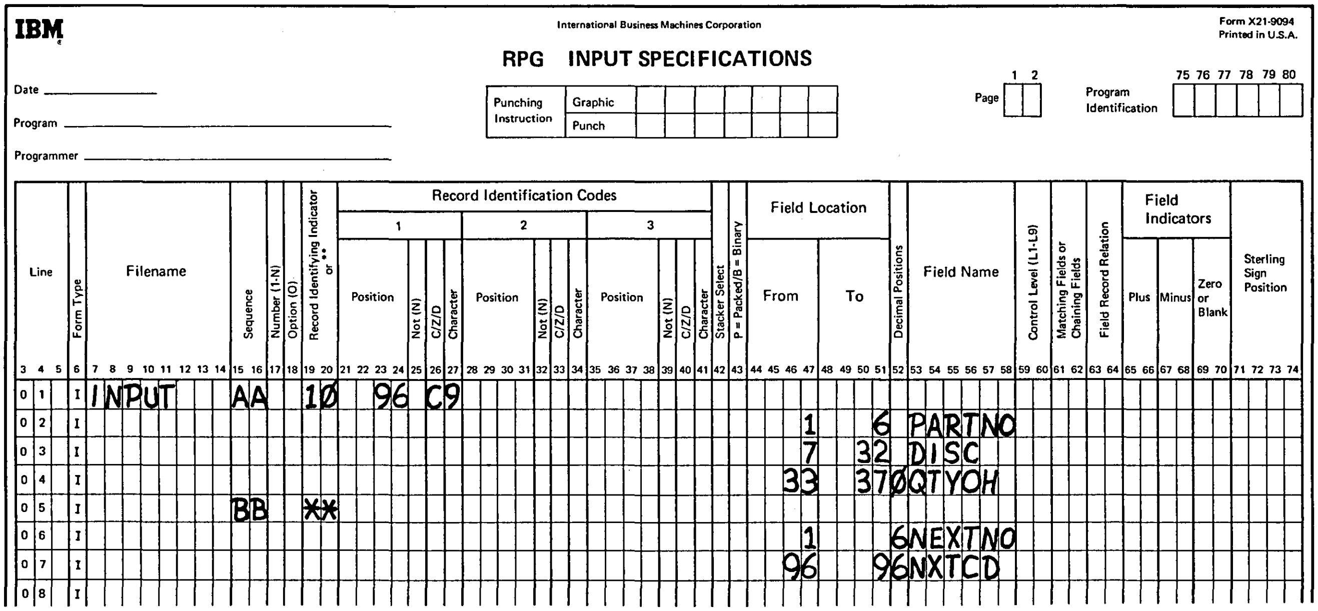
Program Identification 75 76 77 78 79 80

							Record Identification Codes 1				Record Identification Codes 2				Record Identification Codes 3						Field Location							Field Indicators			
Line	Form Type	Filename	Sequence	Number (1-N)	Option (O)	Record Identifying Indicator or **	Position	Not (N)	C/Z/D	Character	Position	Not (N)	C/Z/D	Character	Position	Not (N)	C/Z/D	Character	Stacker Select	P = Packed/B = Binary	From	To	Decimal Positions	Field Name	Control Level (L1-L9)	Matching Fields or Chaining Fields	Field Record Relation	Plus	Minus	Zero or Blank	Sterling Sign Position
3 4 5	6	7 8 9 10 11 12 13 14	15 16	17	18	19 20	21 22 23 24	25	26	27	28 29 30 31	32	33	34	35 36 37 38	39	40	41	42	43	44 45 46 47	48 49 50 51	52	53 54 55 56 57 58	59 60	61 62	63 64	65 66	67 68	69 70	71 72 73 74
0 1	I	INPUT	AA			1Ø	96		C	9																					
0 2	I																				1	6		PARTNO							
0 3	I																				7	32		DISC							
0 4	I																				33	37	Ø	QTYOH							
0 5	I																														

Answers to Review 11. Look Ahead

1. Look ahead allows you to use data on the next record to be processed. Normally only the data on the record currently being processed is available to the RPG II program. Look ahead should only be used with input files. If it is used with combined or update files, information in the look ahead field will be from the record currently being processed, not the combines or update file.

2. ** must be specified in columns 19-20 to indicate that the fields listed are to be looked at in the next card available for processing. Look ahead fields must be given different field names than those used when describing the file. A unique sequence entry must also be used.



IBM

International Business Machines Corporation

Form X21-9094
Printed in U.S.A.

RPG INPUT SPECIFICATIONS

Date

Program

Programmer

Punching Instruction: Graphic / Punch

Page (1 2)

Program Identification (75 76 77 78 79 80)

Line (3-5)	Form Type (6)	Filename (7-14)	Sequence (15-16)	Number (1-N) (17)	Option (O) (18)	Record Identifying Indicator or ** (19-20)	1 Position (21-24)	1 Not (N) (25)	1 C/Z/D (26)	1 Character (27)	2 Position (28-31)	2 Not (N) (32)	2 C/Z/D (33)	2 Character (34)	3 Position (35-38)	3 Not (N) (39)	3 C/Z/D (40)	3 Character (41)	Stacker Select (42)	P = Packed/B = Binary (43)	Field Location From (44-47)	Field Location To (48-51)	Decimal Positions (52)	Field Name (53-58)	Control Level (L1-L9) (59-60)	Matching Fields or Chaining Fields (61-62)	Field Record Relation (63-64)	Field Indicators Plus (65-66)	Field Indicators Minus (67-68)	Field Indicators Zero or Blank (69-70)	Sterling Sign Position (71-74)
01	I	INPUT	AA			1Ø	96		C	9																					
02	I																				1	6		PARTNO							
03	I																				7	32		DISC							
04	I																				33	37	Ø	QTYOH							
05	I		BB			**																									
06	I																				1	6		NEXTNO							
07	I																				96	96		NXTCD							
08	I																														

CHAPTER

DESCRIBING INPUT FIELDS THAT CONTROL PROCESSING

12

CHAPTER 12 DESCRIBES:

Control fields and split control fields.

Field record relation indicators with the OR relationship, split control fields, and match fields.

BEFORE READING THIS CHAPTER YOU SHOULD BE ABLE TO DESCRIBE:

Function and coding of input fields on the RPG II Input Sheet.

Function of RPG II indicators.

RPG II object cycle.

Match fields and matching records logic.

AFTER READING THIS CHAPTER YOU SHOULD BE ABLE TO DESCRIBE:

Function and RPG II coding for control fields and split control fields.

Function and RPG II coding for field record relation indicators.

INTRODUCTION

For every job, you must describe the type of information the RPG II Compiler will be working with. This means that you must describe the input file on the File Description Sheet and the types of records in the file on the Input Sheet.

To describe a record you must describe the fields in the record plus any information that identifies each type of record. It also means you must indicate that certain fields on the record are to be used as control fields or match fields.

CONTROL FIELDS

A basic job in any data processing installation is the preparation of detail reports that consist of one line of printing for each record read, such as a transaction listing. Figure 12-1 shows what a detail report would look like.

Because product classes are repeated for each line, the report is cluttered and hard to read. The same report (Figure 12-2) grouped by class is much easier to read. Here, all items from one class are listed together with headings used on each page to identify the information. Since all items on one page apply to the same class, the class is printed only once. Such a report is sometimes referred to as a *group-indicated* report. Group-indication is the printing of control information on one line per group. The date is printed at the bottom.

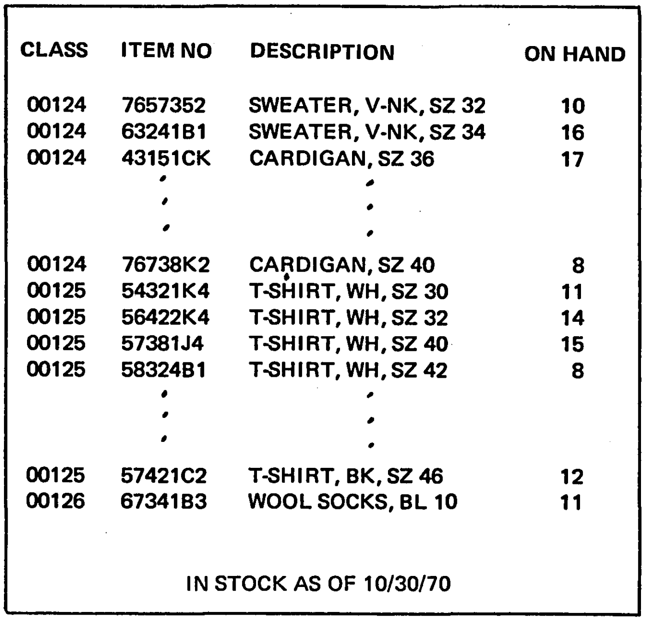
A *control field* is any field used to indicate when a certain type of processing should be done. Since the CLASS field (Figure 12-3) controls processing, it must be specified as the control field. Each time a record is read, this control field is checked for a change in contents (control break). When a control break occurs, a different type of processing or additional processing is to occur. In this case, a change in the CLASS field indicates:

1. Skip to the bottom of the page.

2. Print the date.

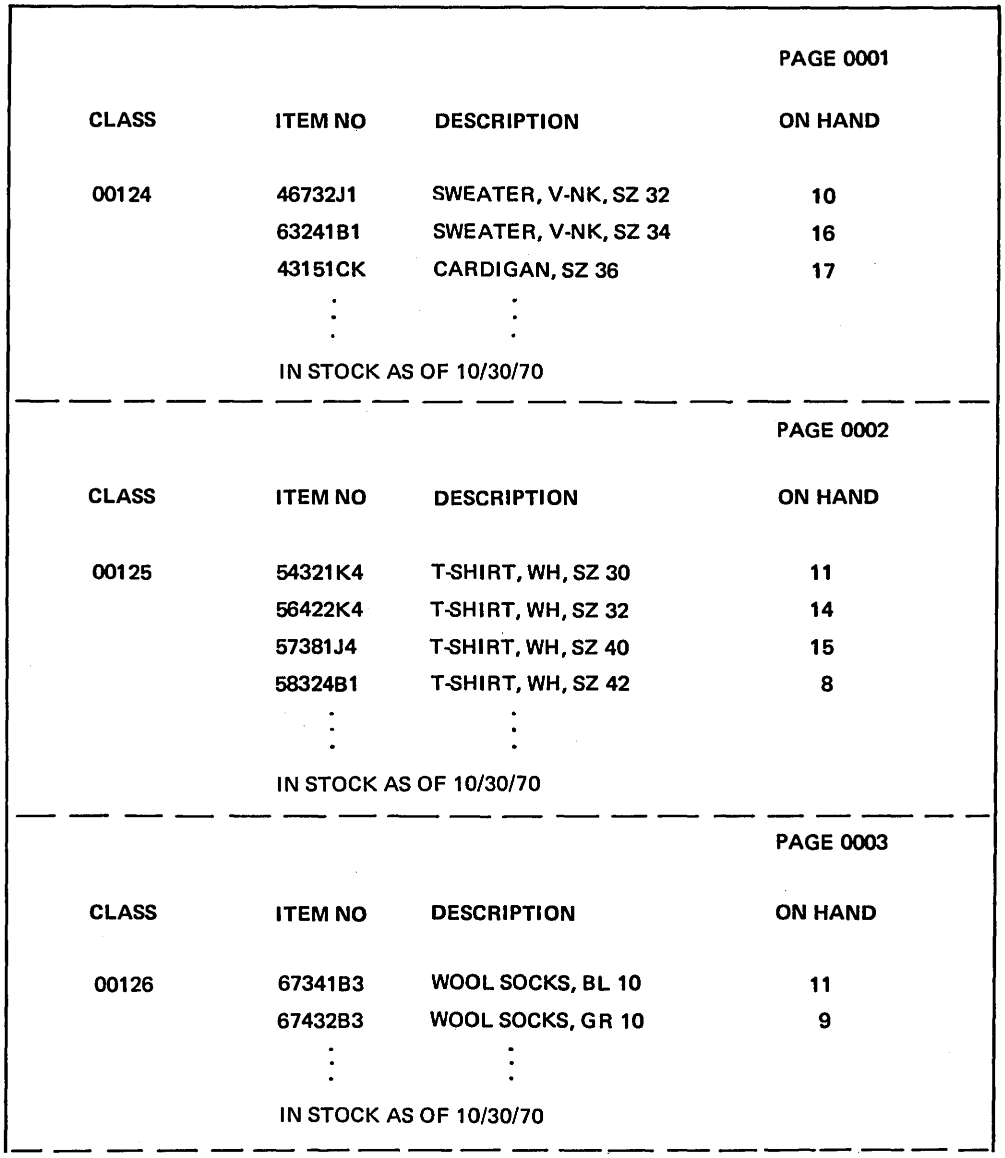
3. Skip to a new page.

4. Print heading.



```
CLASS   ITEM NO   DESCRIPTION                 ON HAND

00124   7657352   SWEATER, V-NK, SZ 32           10
00124   63241B1   SWEATER, V-NK, SZ 34           16
00124   43151CK   CARDIGAN, SZ 36                17
           .                .
           .                .
           .                .
00124   76738K2   CARDIGAN, SZ 40                 8
00125   54321K4   T-SHIRT, WH, SZ 30             11
00125   56422K4   T-SHIRT, WH, SZ 32             14
00125   57381J4   T-SHIRT, WH, SZ 40             15
00125   58324B1   T-SHIRT, WH, SZ 42              8
           .                .
           .                .
           .                .
00125   57421C2   T-SHIRT, BK, SZ 46             12
00126   67341B3   WOOL SOCKS, BL 10              11

              IN STOCK AS OF 10/30/70
```

Figure 12-1. Printed Report of all Items in Stock



```
                                                        PAGE 0001

    CLASS        ITEM NO     DESCRIPTION                ON HAND

    00124        46732J1     SWEATER, V-NK, SZ 32         10
                 63241B1     SWEATER, V-NK, SZ 34         16
                 43151CK     CARDIGAN, SZ 36              17
                    .           .
                    .           .
                    .           .
                 IN STOCK AS OF 10/30/70
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                                                        PAGE 0002

    CLASS        ITEM NO     DESCRIPTION                ON HAND

    00125        54321K4     T-SHIRT, WH, SZ 30           11
                 56422K4     T-SHIRT, WH, SZ 32           14
                 57381J4     T-SHIRT, WH, SZ 40           15
                 58324B1     T-SHIRT, WH, SZ 42            8
                    .           .
                    .           .
                    .           .
                 IN STOCK AS OF 10/30/70
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                                                        PAGE 0003

    CLASS        ITEM NO     DESCRIPTION                ON HAND

    00126        67341B3     WOOL SOCKS, BL 10            11
                 67432B3     WOOL SOCKS, GR 10             9
                    .           .
                    .           .
                 IN STOCK AS OF 10/30/70
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Figure 12-2. Report Group – Indicated by Department Number

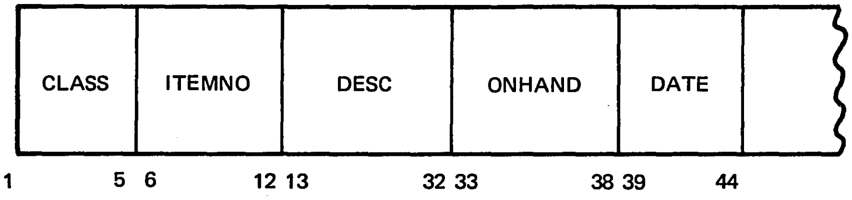
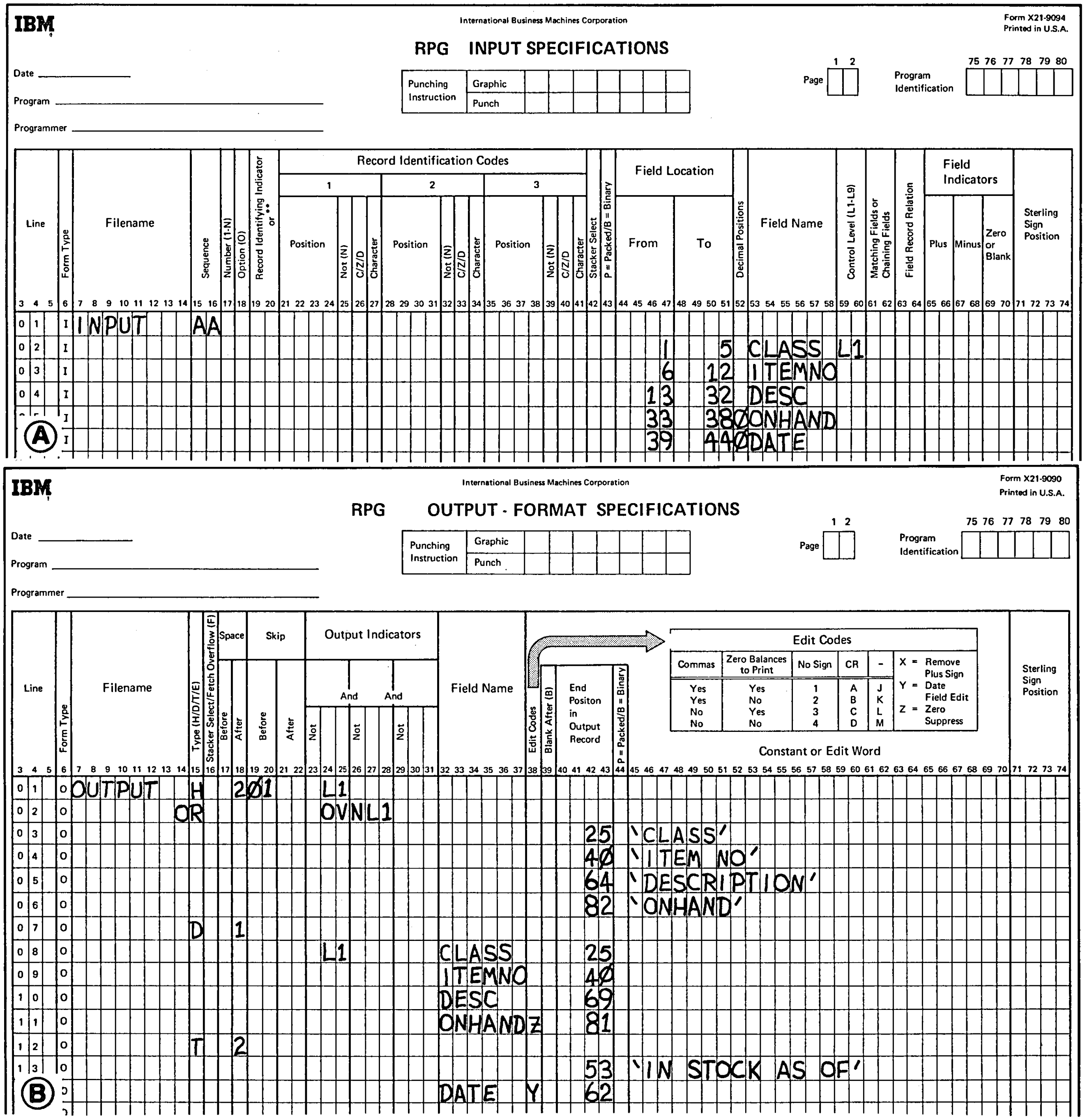


Figure 12-3. Item Record

Coding Control Fields

The RPG II specifications for the job are shown in Figure 12-4. The entry L1 on line 02 of the Input Sheet (Figure 12-4, insert A) establishes the CLASS field as a control field. When the information in the control field changes (a control break occurs) L1 is turned on. The L1 indicator is used on the Output-Format Sheet (Figure 12-4, insert B) to condition those operations which should be performed only when a control break occurs.



IBM International Business Machines Corporation Form X21-9094 Printed in U.S.A.

RPG INPUT SPECIFICATIONS

Date ______ Program ______ Programmer ______

Punching Instruction: Graphic / Punch — Page (1 2) — Program Identification (75 76 77 78 79 80)

Line (3-5)	Form Type (6)	Filename (7-14)	Sequence (15-16)	Field Location From (44-47)	Field Location To (48-51)	Decimal Positions (52)	Field Name (53-58)	Control Level (L1-L9) (59-60)
01	I	INPUT	AA					
02	I			1	5		CLASS	L1
03	I			6	12		ITEMNO	
04	I			13	32		DESC	
	I			33	38	Ø	ONHAND	
	I			39	44	Ø	DATE	

Ⓐ

IBM International Business Machines Corporation Form X21-9090 Printed in U.S.A.

RPG OUTPUT - FORMAT SPECIFICATIONS

Date ______ Program ______ Programmer ______

Punching Instruction: Graphic / Punch — Page (1 2) — Program Identification (75 76 77 78 79 80)

Edit Codes

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Line (3-5)	Form Type (6)	Filename (7-14)	Type (H/D/T/E) (15)	Space After (18)	Skip Before (19-20)	Output Indicators (23-31)	Field Name (32-37)	Edit Codes (38)	End Position in Output Record (40-43)	Constant or Edit Word (45-70)
01	O	OUTPUT	H	2	Ø1	L1				
02	O	OR				OVNL1				
03	O								25	'CLASS'
04	O								4Ø	'ITEM NO'
05	O								64	'DESCRIPTION'
06	O								82	'ONHAND'
07	O		D	1						
08	O					L1	CLASS		25	
09	O						ITEMNO		4Ø	
10	O						DESC		69	
11	O						ONHAND	Z	81	
12	O		T	2						
13	O								53	'IN STOCK AS OF'
	O						DATE	Y	62	

Ⓑ

Figure 12-4. Defining and Using a Control Field

Split Control Fields

Two separate parts of a field or two separate fields can be used as *one* control field known as a split control field. This is done by assigning the same control level indicator to both parts of the field. The compiler will consider the data in the split control fields as *one* continuous field.

Suppose you have a 3-character customer number field in the record and now need a 6-character field. The problem is how to put a larger customer number (such as 100010, 100020) in a 3-character field. You cannot change records easily because there is no room for expansion on either side of the customer number field (Figure 12-5), and to expand the field, the entire record format would have to be changed. All programs using these records would also have to be changed to accomodate the changed record format. This would be considerable work and inconvenience. RPG II provides the split control field feature to meet changing data processing needs with minimum effort.

The solution to the problem is to add a 3-character portion to the customer number field using three columns which are not adjacent to the original customer number field (Figure 12-6). The original three numerals of the customer number remain in the original field. The three additional numbers are put in the new customer number field.

At the end of each month, a report is produced consisting of:

1. Customer number.

2. A description of each purchase.

3. The cost of each purchase.

4. The total cost of all purchases.

The report is group-indicated as shown in Figure 12-7.

The customer number determines when totals would be printed and thus must be used as a control field. However, on each record the customer number is split into two parts (two fields). Both must be used in order to get the correct customer number (Figure 12-8).

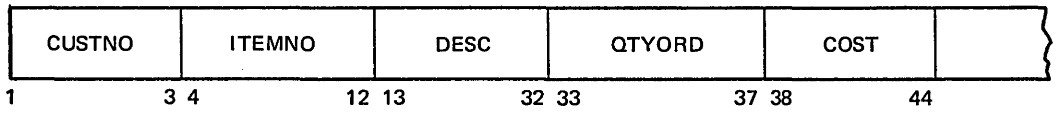


Figure 12-5. Three Digit Customer Field

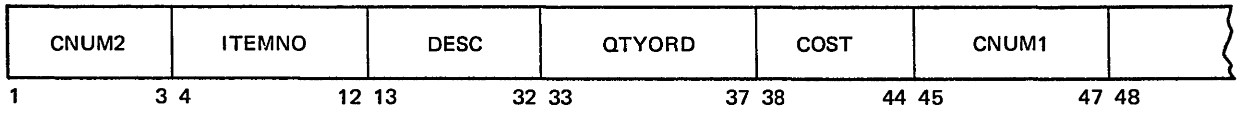
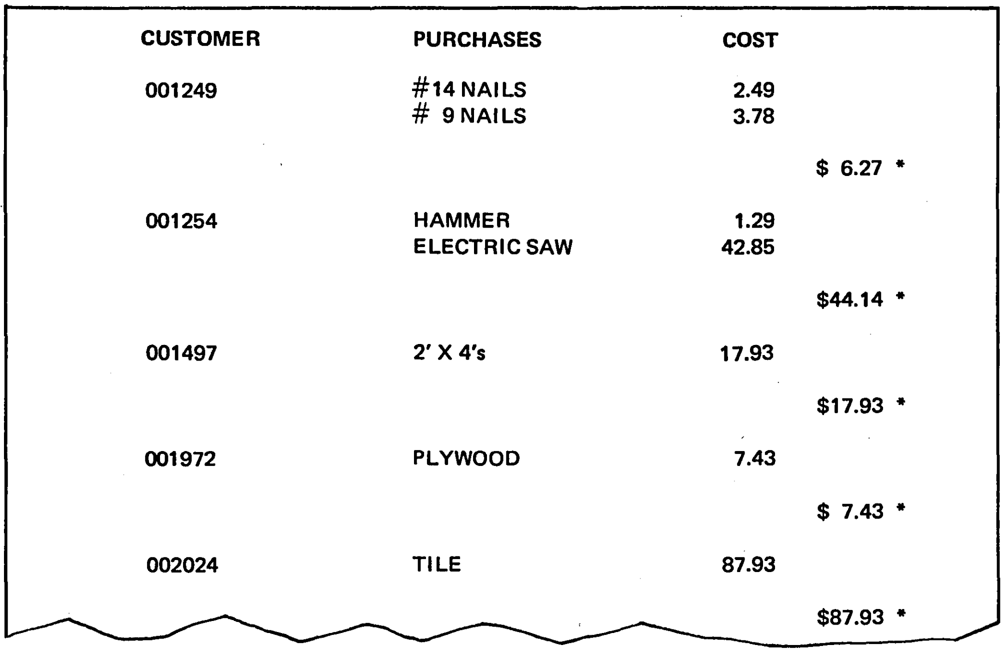


Figure 12-6. One Customer Number Split into Two Parts



```
      CUSTOMER              PURCHASES             COST

      001249                #14 NAILS             2.49
                            #  9 NAILS            3.78

                                                            $ 6.27 *

      001254                HAMMER                1.29
                            ELECTRIC SAW         42.85

                                                            $44.14 *

      001497                2' X 4's             17.93

                                                            $17.93 *

      001972                PLYWOOD               7.43

                                                            $ 7.43 *

      002024                TILE                 87.93

                                                            $87.93 *
```

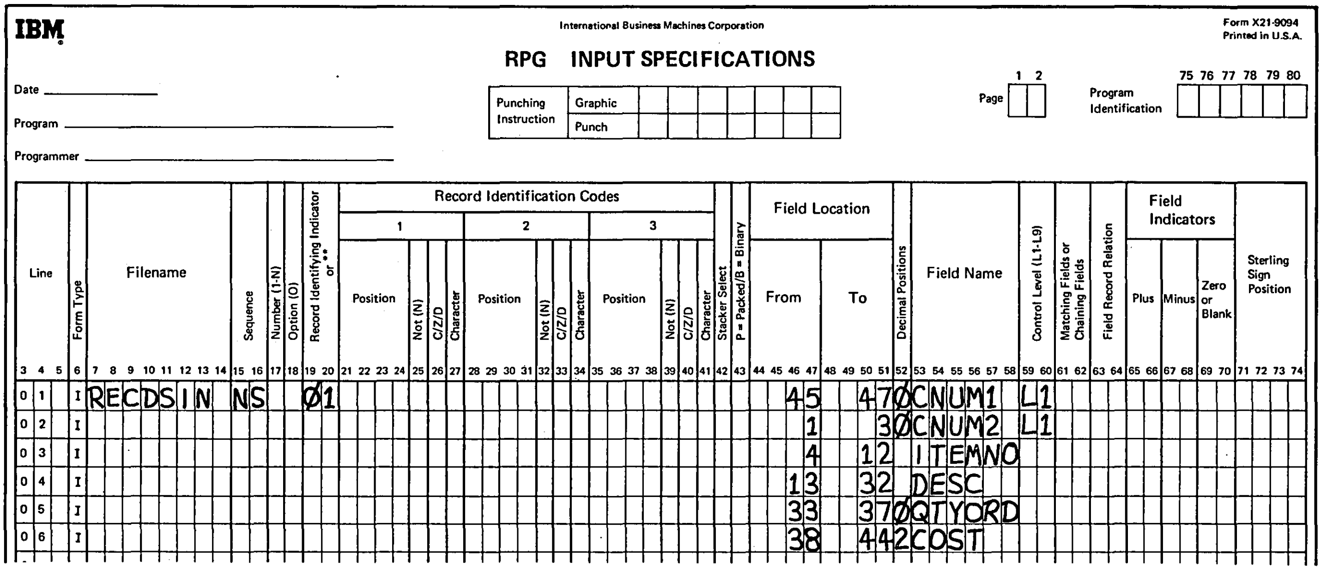
Figure 12-7. Report Group Printed by Customer Number

Coding Split Control Fields

Split control fields must be described in specification lines which follow one another (Figure 12-8).

CNUM1, the field in the high order position of the record (columns 45-47), must be specified on the Input Sheet before CNUM2, the field in positions 1-3. This is required because the three digits in CNUM2 are the first three digits of the customer number.

Parts of a split control field may be either alphameric or numeric. In this example, they were both defined as numeric (indicated by the entry in column 52). If one of them, however, had been defined as numeric and one as alphameric, they both are considered numeric by the compiler.



IBM International Business Machines Corporation Form X21-9094 Printed in U.S.A.

RPG INPUT SPECIFICATIONS

Date ______

Program ______

Programmer ______

Punching Instruction: Graphic □□□□□□□□ Punch □□□□□□□□

Page (1 2) □□ Program Identification (75 76 77 78 79 80) □□□□□□

Line (3-5)	Form Type (6)	Filename (7-14)	Sequence (15-16)	Number (1-N) (17)	Option (O) (18)	Record Identifying Indicator or ** (19-20)	Record Identification Codes 1: Position (21-24)	Not (N) (25)	C/Z/D (26)	Character (27)	2: Position (28-31)	Not (N) (32)	C/Z/D (33)	Character (34)	3: Position (35-38)	Not (N) (39)	C/Z/D (40)	Character (41)	Stacker Select (42)	P = Packed/B = Binary (43)	Field Location From (44-47)	To (48-51)	Decimal Positions (52)	Field Name (53-58)	Control Level (L1-L9) (59-60)	Matching Fields or Chaining Fields (61-62)	Field Record Relation (63-64)	Field Indicators Plus (65-66)	Minus (67-68)	Zero or Blank (69-70)	Sterling Sign Position (71-74)
01	I	RECDSIN	NS			Ø1															45	47	Ø	CNUM1	L1						
02	I																				1	3	Ø	CNUM2	L1						
03	I																				4	12		ITEMNO							
04	I																				13	32		DESC							
05	I																				33	37	Ø	QTYORD							
06	I																				38	44	2	COST							

Figure 12-8. Specifying a Split Control Group

FIELD RECORD RELATION INDICATORS

You may have some programs which process several different record types. Two or more record types might contain identical fields. To eliminate coding these identical fields for every record type you may use the OR relationship which indicates that certain fields are found on all record types. Not all fields are identical in different record types, however. You must have some way of specifying those fields found on only specific record types in the OR relationship. Field record relation indicators indicate those fields found on only specific record types.

Field record relation indicators will relate:

- A field to a specific record type in the OR relationship.
- Control fields and split control fields to a specific record type in an OR relationship.
- Match fields for more than one record type.

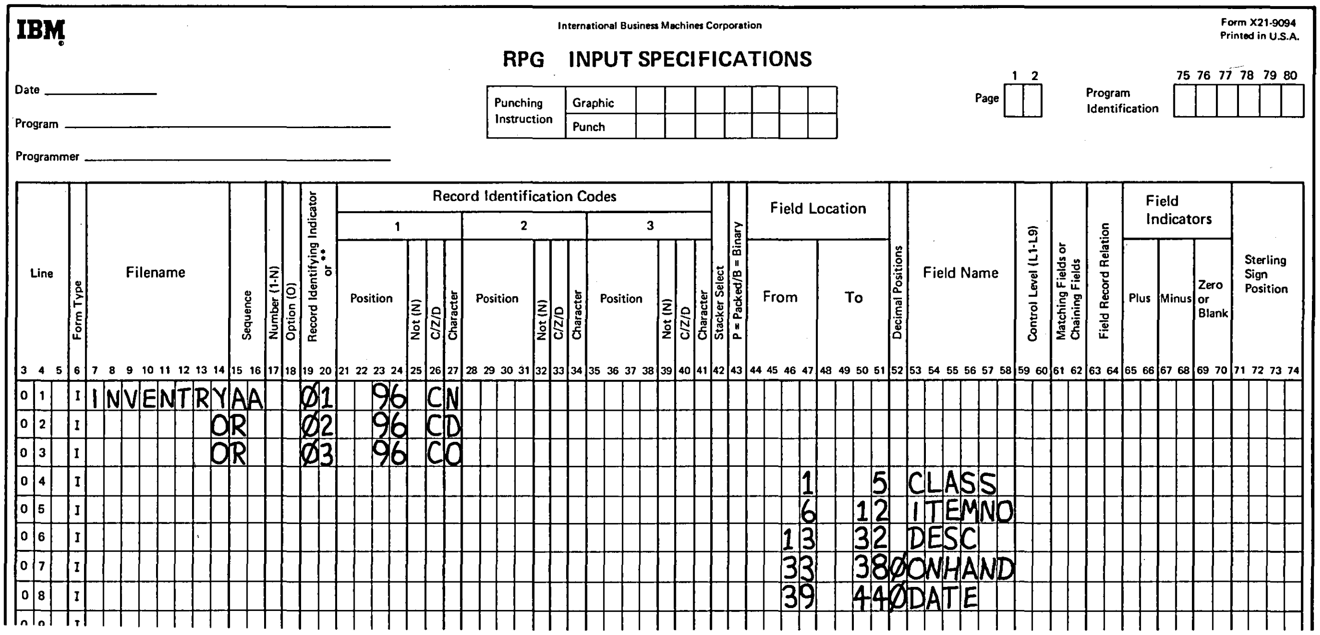
OR Relationship

You can eliminate duplicate coding by using an OR relationship to describe identical record types. This method also reduces the size of the program.

When using the OR relationship, you need to write the names of identical fields from more than one type of record only once on the Input Sheet. OR relationship specifications indicate that the fields named may be found on all of the record types. The following input specifications are necessary to set up the OR relationship:

1. Record identifying indicators (01-99) for each record type.
2. The letters OR in columns 14-15 for all record types other than the first.
3. Entries describing the record identification code of each record type (columns 21-31).

The record identifying codes must be described for *all* types of records in the file before any fields are described (Figure 12-9). The letters OR are placed before the description of each record type except the first. OR indicates that the fields listed may be found on all record types. In this example, the fields listed may be found on records identified by an *N, D,* or *O* in column 96. Identical fields are described after the entries which establish the OR relationship.



IBM

International Business Machines Corporation

Form X21-9094
Printed in U.S.A.

RPG INPUT SPECIFICATIONS

Date

Program

Programmer

Punching Instruction	Graphic	Punch

Page 1 2

Program Identification 75 76 77 78 79 80

Line	Form Type	Filename	Sequence	Number (1-N)	Option (O)	Record Identifying Indicator or **	Record Identification Codes 1 Position	Not (N)	C/Z/D	Character	2 Position	Not (N)	C/Z/D	Character	3 Position	Not (N)	C/Z/D	Character	Stacker Select	P = Packed/B = Binary	Field Location From	To	Decimal Positions	Field Name	Control Level (L1-L9)	Matching Fields or Chaining Fields	Field Record Relation	Field Indicators Plus	Minus	Zero or Blank	Sterling Sign Position
3 4 5	6	7 8 9 10 11 12 13 14	15 16	17	18	19 20	21 22 23 24	25	26	27	28 29 30 31	32	33	34	35 36 37 38	39	40	41	42	43	44 45 46 47	48 49 50 51	52	53 54 55 56 57 58	59 60	61 62	63 64	65 66	67 68	69 70	71 72 73 74
0 1	I	INVENTRY	AA			Ø1	96		C	N																					
0 2	I	OR				Ø2	96		C	D																					
0 3	I	OR				Ø3	96		C	O																					
0 4	I																				1	5		CLASS							
0 5	I																				6	12		ITEMNO							
0 6	I																				13	32		DESC							
0 7	I																				33	38	Ø	ONHAND							
0 8	I																				39	44	Ø	DATE							
0 9	I																														

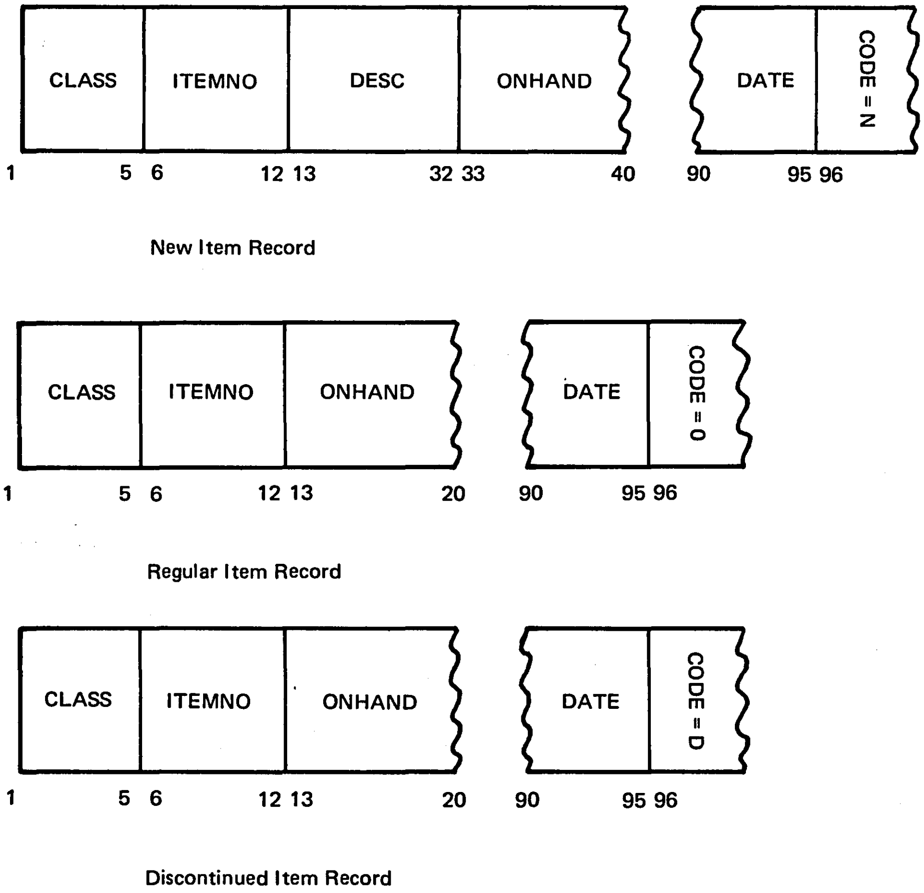
Figure 12-9. Using the OR Relationship to Describe Identical Record Types

OR Relationship With Field Record Relation Entries

In the example of printing a report by product class, all record types had identical fields (Figure 12-3). Suppose that the information on each record type is organized differently; the records have some fields which are identical and some which are not (Figure 12-10). Now you want to print only a description of new items. The record identified by an *N* is the only one with the DESC field. All card types still have CLASS, ITEMNO, DATE, and ONHAND fields.

Remember that OR relationship can be used when all fields are not identical. In this case, additional entries must be made in the field record relation columns (63-64) on the Input Sheet. The entry consists of any of the record identifying indicators (01-99) assigned to a record type specified in the OR relationship. The record identifying indicator entered in columns 63-64 relates a field to a particular record by identifying the record type in which the field is found.

When columns 63-64 are blank, the fields listed are assumed to be found in the positions specified on *all* records in the OR relationship. When an entry is specified in columns 63-64, the field is found only on the record type having that record identifying indicator.



New Item Record

Regular Item Record

Discontinued Item Record

Figure 12-10. Record Types with Some Identical Fields

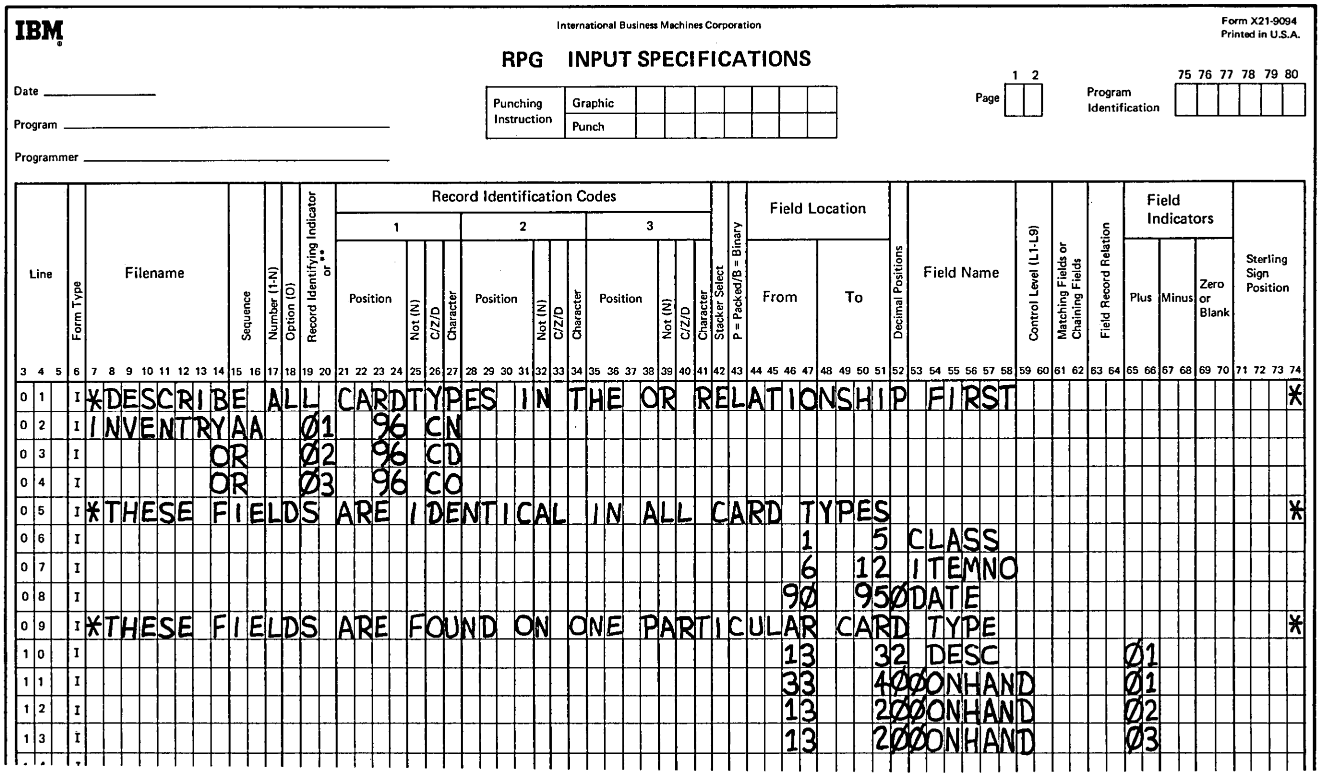
To use the OR relationship with field record relation entries you must:

1. Code the specifications describing record types in the OR relationship (Figure 12-11, lines 02, 03, and 04).

2. Describe all fields which are identical on all record types (Figure 12-11, lines 06, 07, and 08). In this example, the identical fields are CLASS, ITEMNO, and DATE.

3. Specify all fields that are found only on the first record type in the OR relationship, then the second record type, then the third, and so on (Figure 12-11, lines 10, 11, 12, and 13).

In this example, the only fields for the first record type which have not been described are DESC and ONHAND. For each field, the entry 01 must be made in columns 63-64. This entry means that DESC and ONHAND are found on only the record type 01 identified by an *N* in column 96.



IBM International Business Machines Corporation Form X21-9094 Printed in U.S.A.

RPG INPUT SPECIFICATIONS

Date ____ Program ____ Programmer ____

Punching Instruction: Graphic □□□□□□□□ Punch □□□□□□□□

Page □□ (1 2) Program Identification □□□□□□ (75 76 77 78 79 80)

```
01 I*DESCRIBE ALL CARDTYPES IN THE OR RELATIONSHIP FIRST              *
02 IINVENTRYAA  Ø1  96 CN
03 I       OR   Ø2  96 CD
04 I       OR   Ø3  96 CO
05 I*THESE FIELDS ARE IDENTICAL IN ALL CARD TYPES                      *
06 I                                        1   5 CLASS
07 I                                        6  12 ITEMNO
08 I                                       9Ø  95ØDATE
09 I*THESE FIELDS ARE FOUND ON ONE PARTICULAR CARD TYPE                *
10 I                                       13  32 DESC        Ø1
11 I                                       33  4ØØONHAND      Ø1
12 I                                       13  2ØØONHAND      Ø2
13 I                                       13  2ØØONHAND      Ø3
```

Figure 12-11. Field Record Relation

The DESC field is related to the record identified by an *N* because this is the only record type having a DESC field. ONHAND, however, is found on all record types. ONHAND must be related to the record having an *N* in column 96 because it is in a different location on this record type. The field location of ONHAND must be specified and related to the corresponding record type by the record identifying indicators (Figure 12-12, line 11).

Remember that when fields are not identical on all record types, the field must be described and related to all record types on which it is found.

All fields relating to only one record type must be entered as a group and must be given the same record identifying indicators in columns 63-64.

If most fields are common, describing the record type with field record relation usually reduces the number of specifications you must write and the amount of storage necessary to hold the instructions.

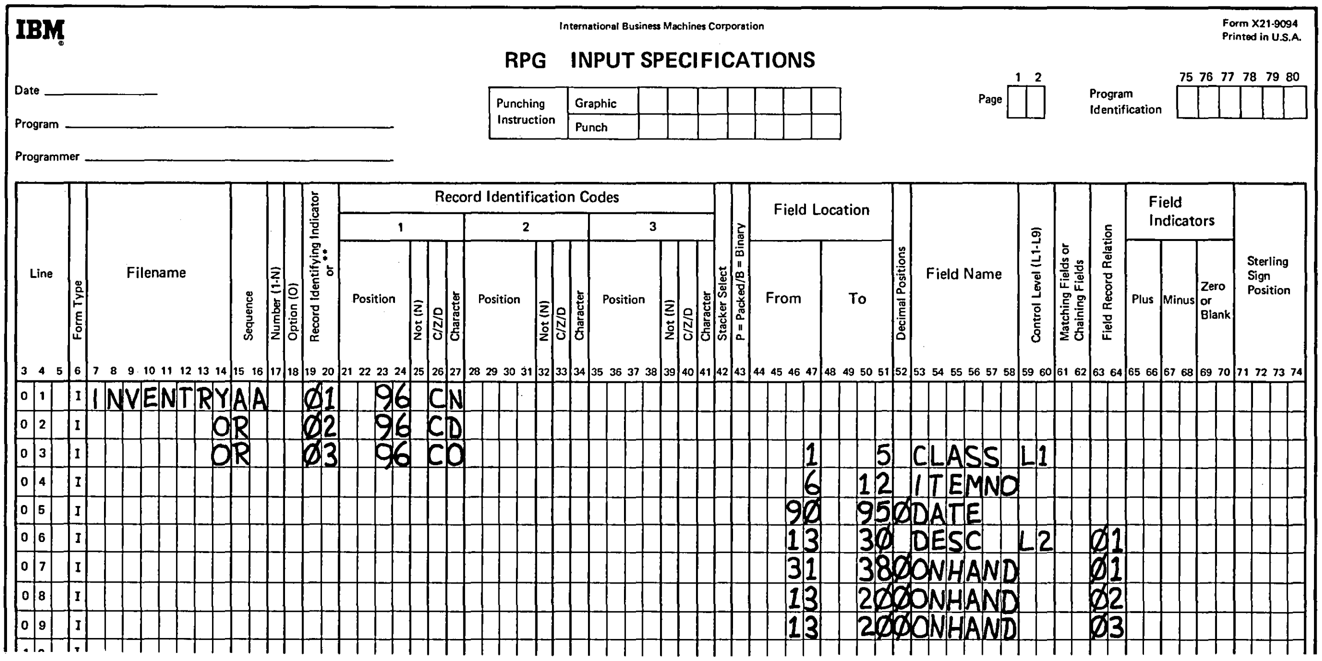
Field Record Relation with Control Fields

Control fields can also be related to a specific record type in an OR relationship by field record relation entries. In Figure 12-12 the CLASS field is a control field (L1 in columns 59-60). It is also found on all record types; blanks in the columns 63-64 indicate this. However, if a control field is found on only one record type, the control field must be related to the record type in which it is found by an entry in columns 63-64 (Figure 12-12, line 07).

The number of control fields need not be the same for every record in the OR relationship. Regardless of the number of control fields per record type, all control fields and all other fields related to the same record type are entered as a group (Figure 12-12, lines 07 and 08).

Field Record Relation with Split Control Fields

The rules applying to field record relation with control fields also apply to field record relation with split control fields. In addition, when split control fields are found on record types described in an OR relationship used with field record relation entries, all portions of the split control field must be assigned to same control level indicator and the same field record relation entry. This is necessary because all parts of a split control field are on the same record rather than on two different records.



IBM

International Business Machines Corporation

Form X21-9094
Printed in U.S.A.

RPG INPUT SPECIFICATIONS

Date ____

Program ____

Programmer ____

Punching Instruction: Graphic □□□□□□□□ Punch □□□□□□□□

Page 1 2 □□

Program Identification 75 76 77 78 79 80 □□□□□□

Line (3-5)	Form Type (6)	Filename (7-14)	Sequence (15-16)	Number (1-N) (17)	Option (O) (18)	Record Identifying Indicator or ** (19-20)	Record Identification Codes 1: Position (21-24)	Not (N) (25)	C/Z/D (26)	Character (27)	2: Position (28-31)	Not (N) (32)	C/Z/D (33)	Character (34)	3: Position (35-38)	Not (N) (39)	C/Z/D (40)	Character (41)	Stacker Select (42)	P = Packed/B = Binary (43)	Field Location From (44-47)	To (48-51)	Decimal Positions (52)	Field Name (53-58)	Control Level (L1-L9) (59-60)	Matching Fields or Chaining Fields (61-62)	Field Record Relation (63-64)	Field Indicators Plus (65-66)	Minus (67-68)	Zero or Blank (69-70)	Sterling Sign Position (71-74)
01	I	INVENTRY	AA			Ø1	96		C	N																					
02	I	OR				Ø2	96		C	D																					
03	I	OR				Ø3	96		C	O											1	5		CLASS	L1						
04	I																				6	12		ITEMNO							
05	I																				9Ø	95	Ø	DATE							
06	I																				13	3Ø		DESC	L2		Ø1				
07	I																				31	38	Ø	ONHAND			Ø1				
08	I																				13	2Ø	Ø	ONHAND			Ø2				
09	I																				13	2Ø	Ø	ONHAND			Ø3				

Figure 12-12. Field Record Relation with Control Fields

Using Match Fields With Field Record Relation For More Than One Record Type

Many fields have the same name, contain the same type of data, and are found in certain positions of any record type in the file. For example, salesmen records and other employee records could be organized as shown in Figure 12-13. For both record types, all fields are the same except the COMM and SALARY fields and the record identifying code.

When only a few fields differ, record types can be described on the Input Sheet in an OR relationship. Instead of using separate sets of input specifications, common fields need be described only once. As shown in Figure 12-14, entries in the field-record relation columns can then identify the fields which are unique to a particular record type. Notice that fields which are the same for all record types are described first. All fields related to a particular record type are then described before specifying the fields related to one of the other record types.

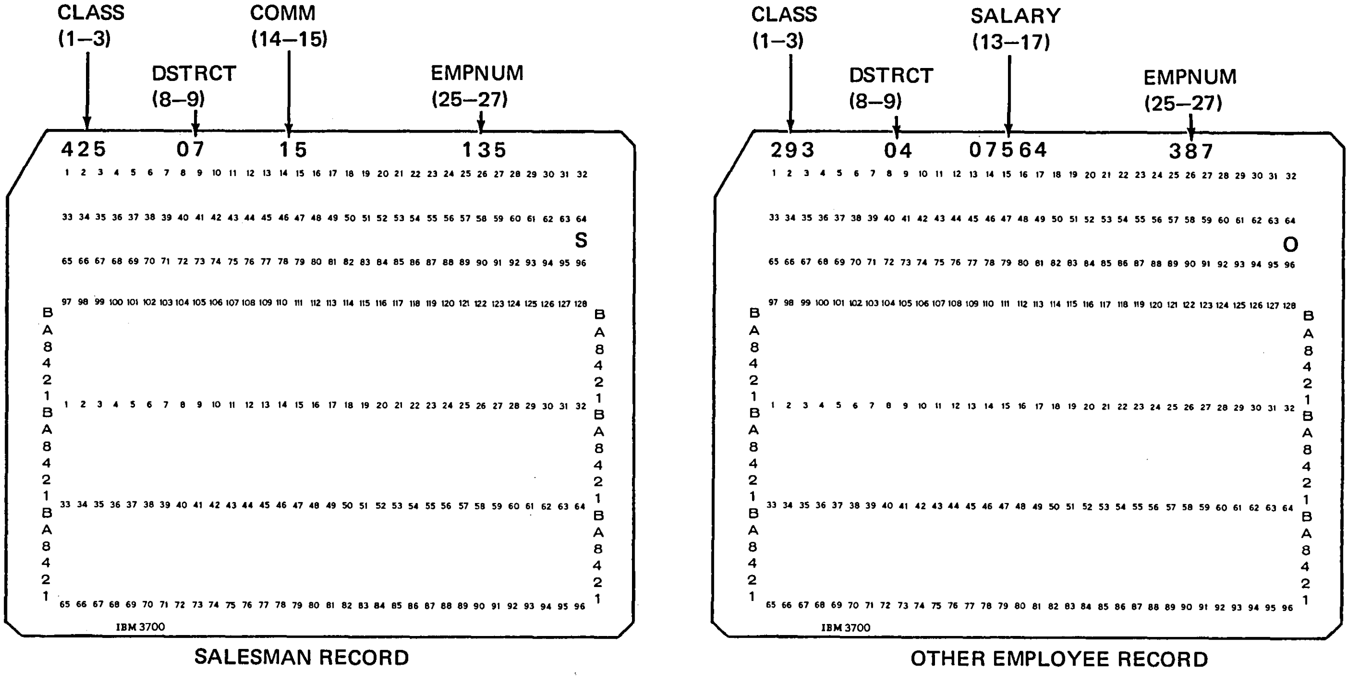
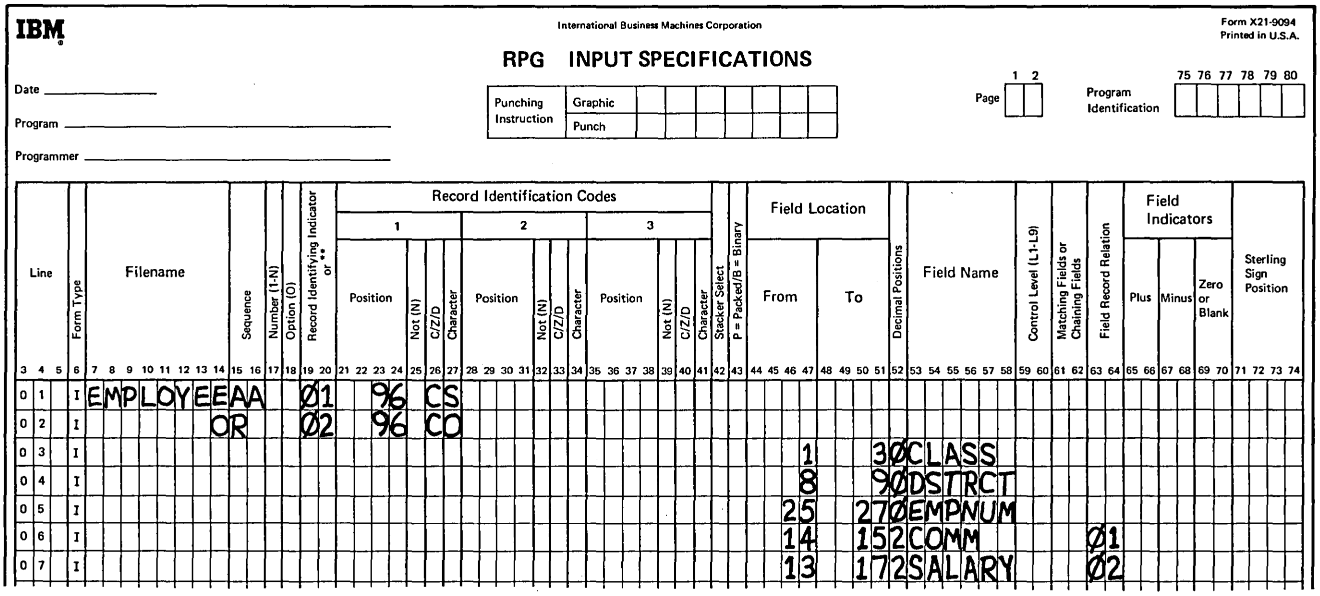


Figure 12-13. Same Match Fields for Both Record Types



IBM

International Business Machines Corporation

Form X21-9094
Printed in U.S.A.

RPG INPUT SPECIFICATIONS

Date ____________

Program ____________

Programmer ____________

Punching Instruction	Graphic							
	Punch							

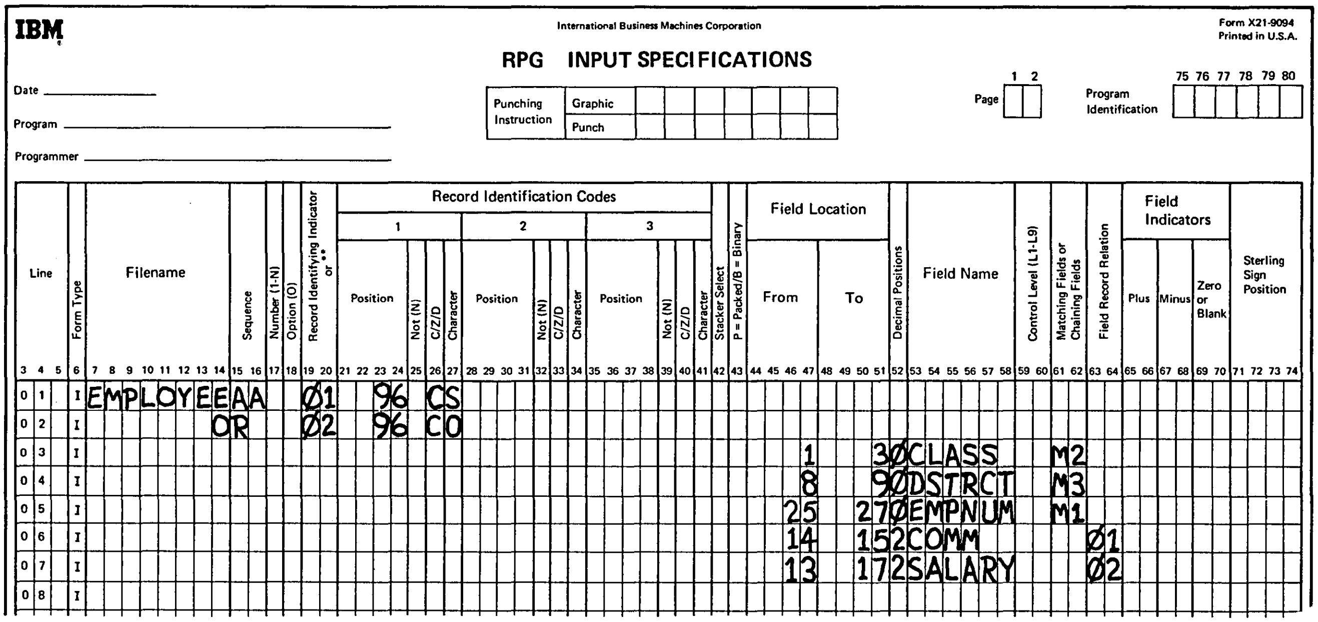
Page 1 2

Program Identification 75 76 77 78 79 80

Line (3-5)	Form Type (6)	Filename (7-14)	Sequence (15-16)	Number (1-N) (17)	Option (O) (18)	Record Identifying Indicator or ** (19-20)	1 Position (21-24)	1 Not (N) (25)	1 C/Z/D (26)	1 Character (27)	2 Position (28-31)	2 Not (N) (32)	2 C/Z/D (33)	2 Character (34)	3 Position (35-38)	3 Not (N) (39)	3 C/Z/D (40)	3 Character (41)	Stacker Select (42)	P = Packed/B = Binary (43)	Field Location From (44-47)	Field Location To (48-51)	Decimal Positions (52)	Field Name (53-58)	Control Level (L1-L9) (59-60)	Matching Fields or Chaining Fields (61-62)	Field Record Relation (63-64)	Field Indicators Plus (65-66)	Field Indicators Minus (67-68)	Field Indicators Zero or Blank (69-70)	Sterling Sign Position (71-74)
01	I	EMPLOYEE	AA			Ø1	96		C	S																					
02	I	OR				Ø2	96		C	O																					
03	I																				1	3	Ø	CLASS							
04	I																				8	9	Ø	DSTRCT							
05	I																				25	27	Ø	EMPNUM							
06	I																				14	15	2	COMM			Ø1				
07	I																				13	17	2	SALARY			Ø2				

Figure 12-14. Describing Fields with Field Record Relation

DSTRCT (M3), CLASS (M2), and EMPNUM (M1) are the three match fields to be used in sequence checking the EMPLOYEE file. Since all match fields are common to all record types, they have been described only once on the Input Sheet without any field record relation entries. Therefore, the match field entries need be assigned only once (Figure 12-15). When record types are described in an OR relationship and a match field entry is assigned to a field without any field record relation entry, the match field will be used for all record types.



IBM

International Business Machines Corporation

Form X21-9094
Printed in U.S.A.

RPG INPUT SPECIFICATIONS

Date ______________

Program ______________

Programmer ______________

Punching Instruction	Graphic							
	Punch							

Page (1 2) □□

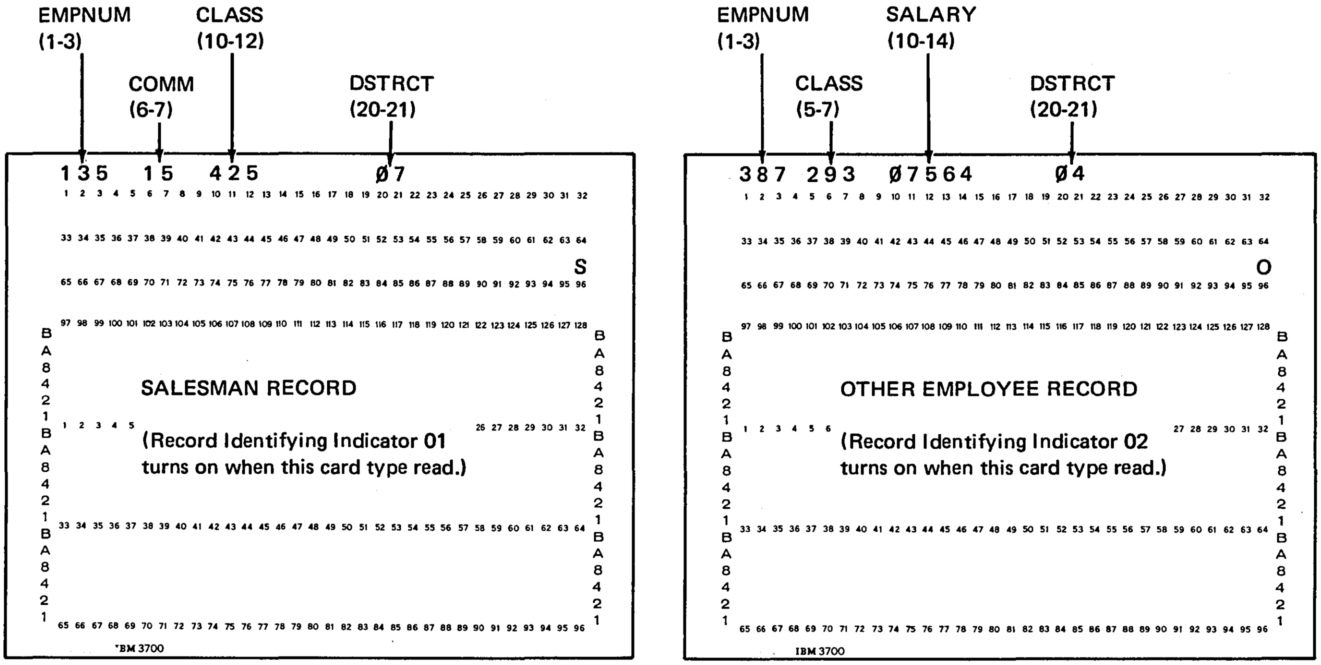
Program Identification (75 76 77 78 79 80) □□□□□□

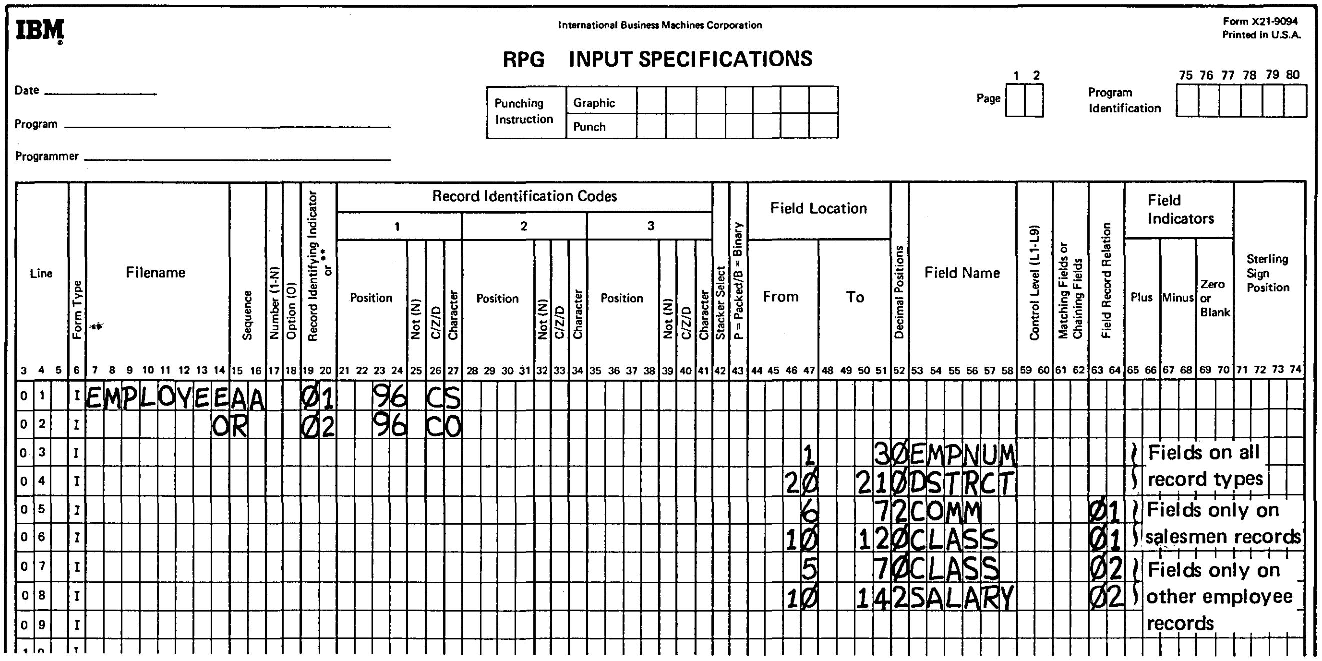
Line (3-5)	Form Type (6)	Filename (7-14)	Sequence (15-16)	Number (1-N) (17)	Option (O) (18)	Record Identifying Indicator or ** (19-20)	Record Identification Codes 1: Position (21-24)	Not (N) (25)	C/Z/D (26)	Character (27)	2: Position (28-31)	Not (N) (32)	C/Z/D (33)	Character (34)	3: Position (35-38)	Not (N) (39)	C/Z/D (40)	Character (41)	Stacker Select (42)	P = Packed/B = Binary (43)	Field Location From (44-47)	To (48-51)	Decimal Positions (52)	Field Name (53-58)	Control Level (L1-L9) (59-60)	Matching Fields or Chaining Fields (61-62)	Field Record Relation (63-64)	Field Indicators Plus (65-66)	Minus (67-68)	Zero or Blank (69-70)	Sterling Sign Position (71-74)
01	I	EMPLOYEE	AA			Ø1	96		C	S																					
02	I	OR				Ø2	96		C	O																					
03	I																				1	3	Ø	CLASS		M2					
04	I																				8	9	Ø	DSTRCT		M3					
05	I																				25	27	Ø	EMPNUM		M1					
06	I																				14	15	2	COMM			Ø1				
07	I																				13	17	2	SALARY			Ø2				
08	I																														

Figure 12-15. Assigning Match Fields Once for Two Record Types

Although some fields (Figure 12-15) differed between record types, all the fields to be used in matching were the same (same name, format, and card columns). Suppose that one of the match fields, CLASS, is in different record columns for each record type. The two record types in the EMPLOYEE file might then appear as in Figure 12-16.

The two record types can be described either by coding separate input specifications for each record type or by combining the entries using field record relation. Both methods have advantages. If most fields are common, describing the record types with field record relation usually reduces the number of specifications you must write and also reduces the amount of storage necessary to hold the instructions. However, when match fields differ between record types, assigning field record relation entries to the match fields can be somewhat confusing at first.





IBM

International Business Machines Corporation

RPG INPUT SPECIFICATIONS

Form X21-9094
Printed in U.S.A.

Date

Program

Programmer

Punching Instruction: Graphic / Punch

Page 1 2

Program Identification 75 76 77 78 79 80

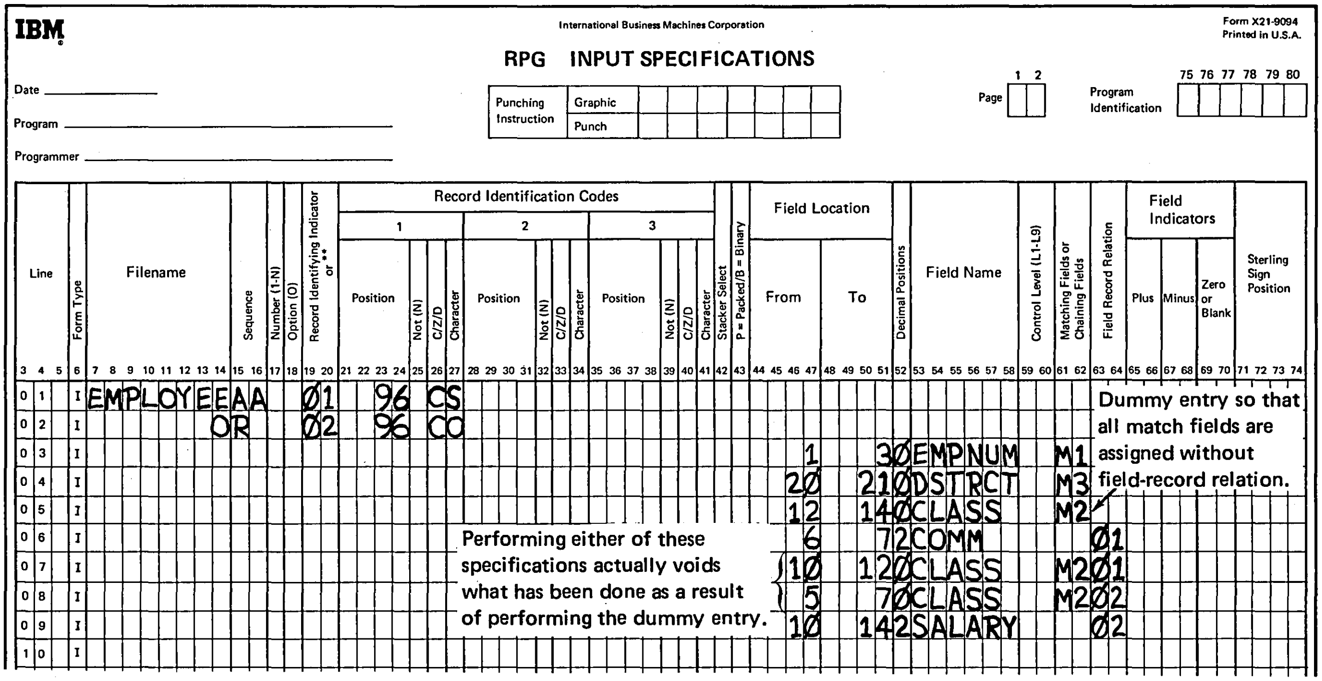
Line	Form Type	Filename	Sequence	Number (1-N)	Option (O)	Record Identifying Indicator or **	Record Identification Codes 1 Position	Not (N)	C/Z/D	Character	2 Position	Not (N)	C/Z/D	Character	3 Position	Not (N)	C/Z/D	Character	Stacker Select	P = Packed/B = Binary	Field Location From	To	Decimal Positions	Field Name	Control Level (L1-L9)	Matching Fields or Chaining Fields	Field Record Relation	Field Indicators Plus	Minus	Zero or Blank	Sterling Sign Position
01	I	EMPLOYEE	AA			Ø1	96		C	S																					
02	I	OR				Ø2	96		C	O																					
03	I																				1	3	Ø	EMPNUM				Fields on all			
04	I																				2Ø	21	Ø	DSTRCT				record types			
05	I																				6	7	2	COMM			Ø1	Fields only on			
06	I																				1Ø	12	Ø	CLASS			Ø1	salesmen records			
07	I																				5	7	Ø	CLASS			Ø2	Fields only on			
08	I																				1Ø	14	2	SALARY			Ø2	other employee			
09	I																											records			

Figure 12-16. Match Fields Differ Between Record Types

The two record types in Figure 12-16 are described using field record relation. To sequence check the EMPLOYEE file, match field entries must also be entered for the appropriate fields (EMPNUM, DSTRCT, and CLASS). Figure 12-17 shows correct specifications for this job. Two of the match fields (EMPNUM and DSTRCT) are found on all record types, and consequently do not have field record relation entries. Notice, however, that the M2 match field (CLASS) is not the same for all record types since the location of the field varies. Therefore, the CLASS match field differs on each record type and must be assigned field record relation indicators. When such a situation occurs (some match fields on all record types and some match fields which differ between record types), the match fields assigned field record relation indicators must also be assigned to *all* record types as EMPNUM and DSTRCT are. This is done by assigning a dummy match field entry that is not conditioned by field record relation indicators.

Figure 12-17, line 05, shows the dummy entry for the CLASS match field. You know which entries to make in the Field Location columns of this dummy match field because you know that any one match field is always the same length, regardless of record type of column location. In this case, CLASS is three columns long. Any numbers which give the correct length of the match field can be specified.

As shown in Figure 12-17, columns 12-14 are specified for the dummy match field. When this specification line is performed, the program reads the three columns of the data (columns 12-14) on whichever record type was read. Of course, the M2 match field is not in columns 12-14 on either record type so you do not want this incorrect data to be used in sequence checking. It will not be, because all of the match field entries are checked before the sequence checking is performed. If the card read is record type 01, line 07 is performed. This entry tells the computer that the data in columns 10-12 should be used for the M2 field. On the other hand, if record type 02 is read, line 08 is performed and the data in columns 5-7 is used instead of that in columns 12-14. Therefore, when either of the specifications in lines 07 or 08 is performed (depending on record type), the data used as the match field is changed, as if the dummy entry has never been specified.



IBM

International Business Machines Corporation

Form X21-9094
Printed in U.S.A.

RPG INPUT SPECIFICATIONS

Date

Program

Programmer

Punching Instruction: Graphic / Punch

Page 1 2

Program Identification 75 76 77 78 79 80

Line	Form Type	Filename	Sequence	Record Identifying Indicator	Position	C/Z/D	Character	From	To	Decimal Positions	Field Name	Matching Fields or Chaining Fields	Field Record Relation
01	I	EMPLOYEE	AA	Ø1	96	C	S						
02	I	OR		Ø2	96	C	O						
03	I							1	3	Ø	EMPNUM	M1	
04	I							2Ø	21	Ø	DSTRCT	M3	
05	I							12	14	Ø	CLASS	M2	
06	I							6	7	2	COMM		Ø1
07	I							1Ø	12	Ø	CLASS	M2	Ø1
08	I							5	7	Ø	CLASS	M2	Ø2
09	I							1Ø	14	2	SALARY		Ø2
10	I												

Dummy entry so that all match fields are assigned without field-record relation.

Performing either of these specifications actually voids what has been done as a result of performing the dummy entry.

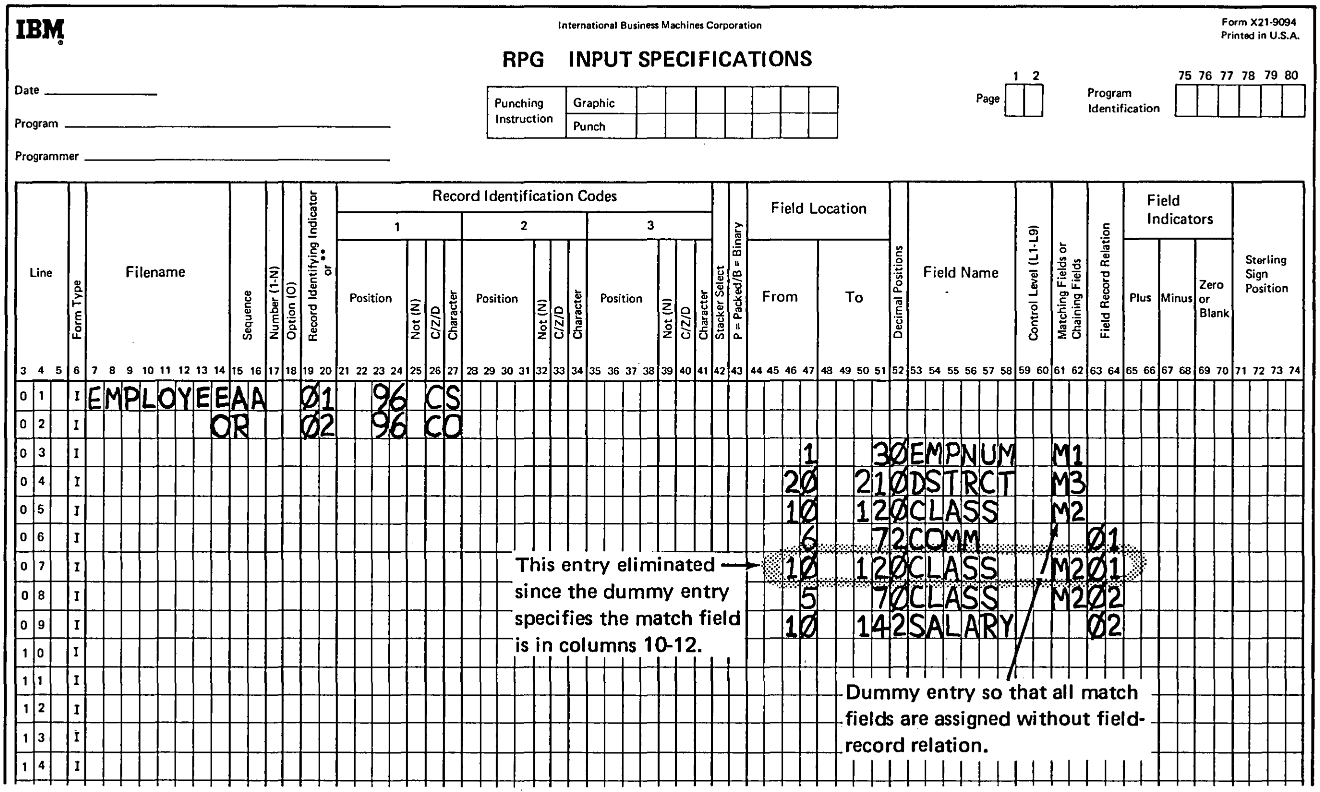
Figure 12-17. Assigning Match Fields for Records Described with Field Record Relation

Although the specifications in Figure 12-17 will cause the EMPLOYEE file to be sequence checked correctly, there is a way to reduce the number of specifications required. As mentioned, for a dummy entry you can specify any columns which give the correct length for the match field. However, if you specify the actual columns associated with that match field on one of the record types, there is no need for the specification which relates those columns to the match field for that record type (Figure 12-18). By entering 10 to 12 (line 05, Figure 12-18) as the columns for the M2 field, you can eliminate the match field entry (line 07) in which the M2 field is described for record type 01.

After performing the dummy entry (line 05), the computer knows the M2 match field is to be found in columns 10-12. If record type 01 is read, the M2 field actually is in columns 10-12. Thus, line 07 does not have to be performed because it would not change anything. Of course, if record type 02 were read, the specification in line 08 is performed.

This says, for record type 02, use columns 5-7 for the M2 field instead of columns 10-12.

The Field Name specified for the dummy entry can be any name, also, since field names are ignored in selecting match fields. In this case, CLASS was specified since the M2 fields on both record types have the same name. If the names differ, it is still a good practice to use a name given to the match field on one of the record types.



IBM

International Business Machines Corporation

Form X21-9094
Printed in U.S.A.

RPG INPUT SPECIFICATIONS

Date
Program
Programmer

Punching Instruction | Graphic | Punch

Page 1 2

Program Identification 75 76 77 78 79 80

Line	Form Type	Filename	Sequence	Record Identifying Indicator	Record Identification Codes 1: Position	C/Z/D	Character	Field Location From	To	Decimal Positions	Field Name	Matching Fields or Chaining Fields	Field Record Relation
01	I	EMPLOYEE	AA	Ø1	96	C	S						
02	I		OR	Ø2	96	C	O						
03	I							1	3	Ø	EMPNUM	M1	
04	I							2Ø	21	Ø	DSTRCT	M3	
05	I							1Ø	12	Ø	CLASS	M2	
06	I							6	7	2	COMM		Ø1
07	I							1Ø	12	Ø	CLASS	M2	Ø1
08	I							5	7	Ø	CLASS	M2	Ø2
09	I							1Ø	14	2	SALARY		Ø2
10	I												
11	I												
12	I												
13	I												
14	I												

This entry eliminated since the dummy entry specifies the match field is in columns 10-12.

Dummy entry so that all match fields are assigned without field-record relation.

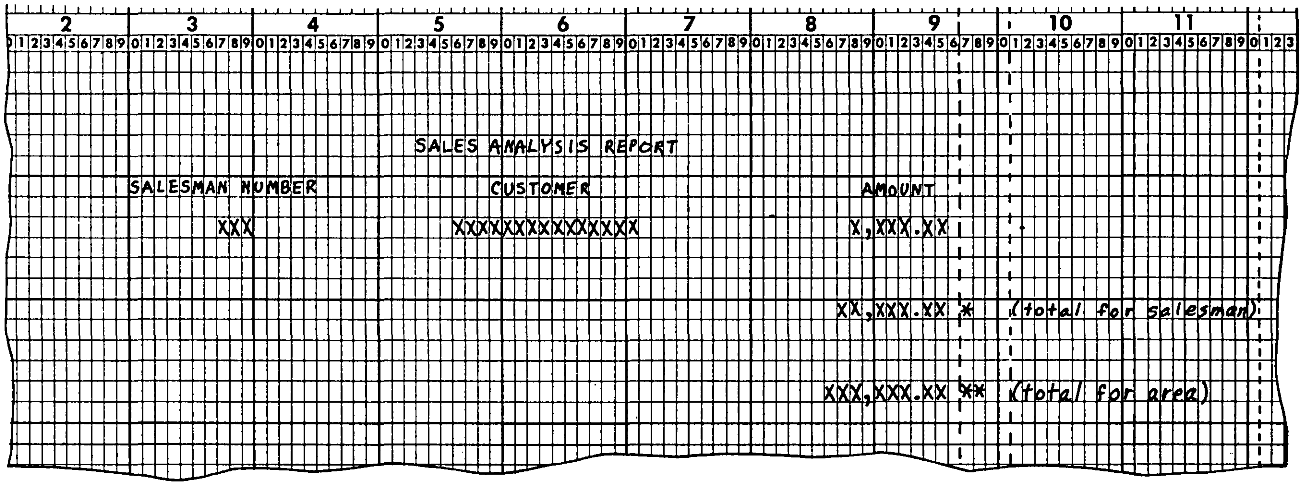
Figure 12-18. Eliminating Specifications in Assigning Match Field Entries

Review 12

1. A sales analysis report is to be group-indicated by salesman number as shown on the following printer spacing chart. The fields on input file records are arranged as follows:

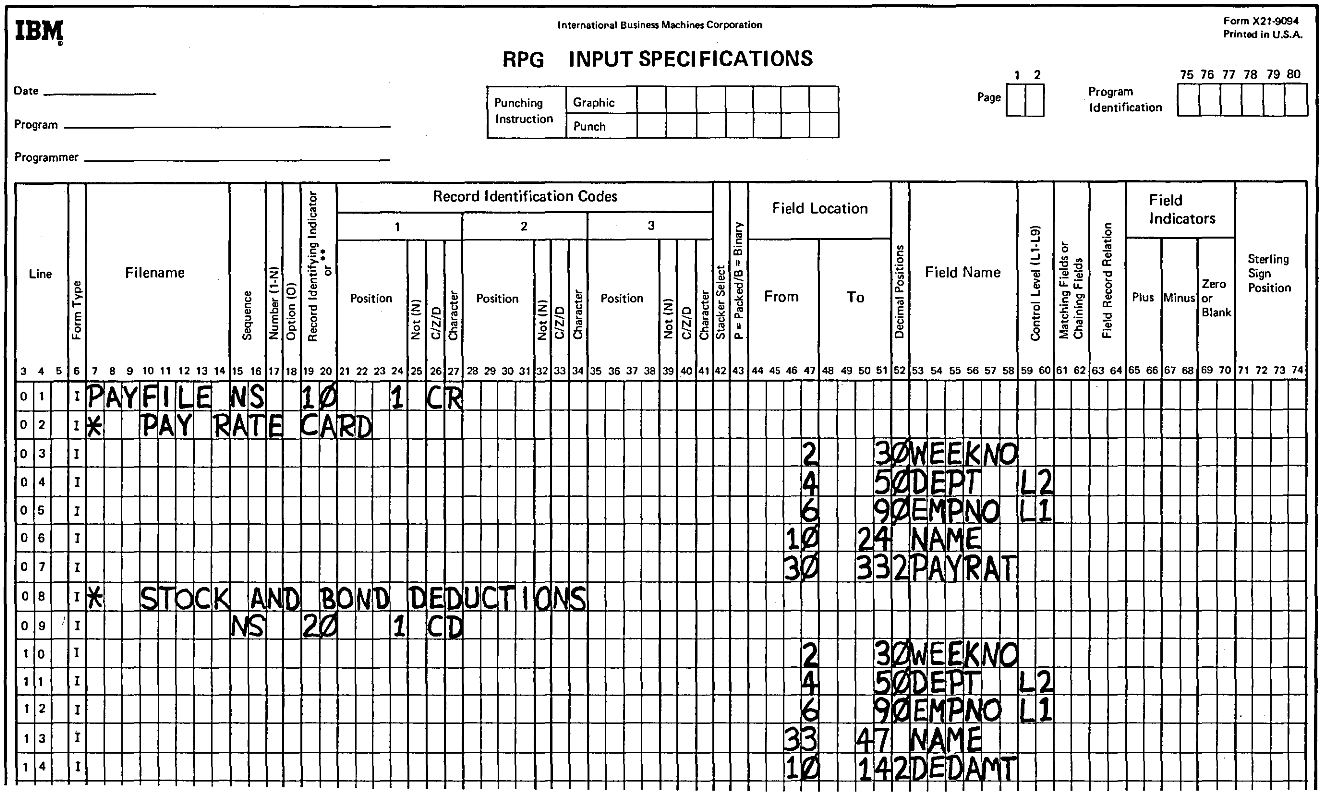
Columns	*Entries*
1-2	Salesman number (last two digits of the three possible)
3-8	Amount of sale
9-23	Customer name
30	Salesman number (first digit of the three possible)
96	Indicates the record type

Fill in the input specifications for this job choosing your own file and field names.



```
                                 SALES ANALYSIS REPORT

          SALESMAN NUMBER                       CUSTOMER                    AMOUNT
                 XXX                    XXXXXXXXXXXXXXXX                   X,XXX.XX

                                                                          XX,XXX.XX *    (total for salesman)

                                                                         XXX,XXX.XX **   (total for area)
```

2. Rewrite the Input specifications shown below using field record relation entries.



IBM

International Business Machines Corporation

RPG INPUT SPECIFICATIONS

Form X21-9094
Printed in U.S.A.

Date ____________

Program ____________

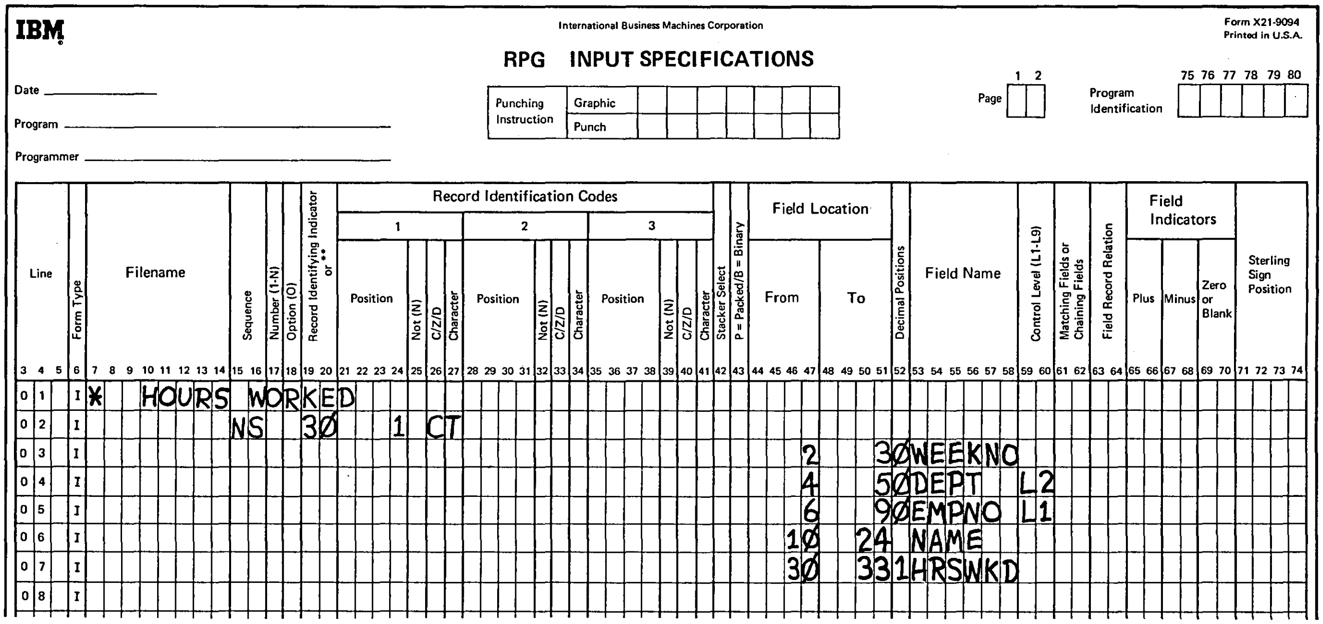
Programmer ____________

Punching Instruction: Graphic / Punch

Page 1 2

Program Identification 75 76 77 78 79 80

Line	Form Type	Filename	Sequence	Number (1-N)	Option (O)	Record Identifying Indicator or **	1 Position	1 Not (N)	1 C/Z/D	1 Character	2 Position	2 Not (N)	2 C/Z/D	2 Character	3 Position	3 Not (N)	3 C/Z/D	3 Character	Stacker Select	P = Packed/B = Binary	Field Location From	Field Location To	Decimal Positions	Field Name	Control Level (L1-L9)	Matching Fields or Chaining Fields	Field Record Relation	Field Indicators Plus	Field Indicators Minus	Field Indicators Zero or Blank	Sterling Sign Position
01	I	PAYFILE	NS			1Ø	1		C	R																					
02	I	* PAY RATE CARD																													
03	I																				2	3	Ø	WEEKNO							
04	I																				4	5	Ø	DEPT	L2						
05	I																				6	9	Ø	EMPNO	L1						
06	I																				1Ø	24		NAME							
07	I																				3Ø	33	2	PAYRAT							
08	I	* STOCK AND BOND DEDUCTIONS																													
09	I		NS			2Ø	1		C	D																					
10	I																				2	3	Ø	WEEKNO							
11	I																				4	5	Ø	DEPT	L2						
12	I																				6	9	Ø	EMPNO	L1						
13	I																				33	47		NAME							
14	I																				1Ø	14	2	DEDAMT							



IBM

International Business Machines Corporation

RPG INPUT SPECIFICATIONS

Form X21-9094
Printed in U.S.A.

Date ____________

Program ____________

Programmer ____________

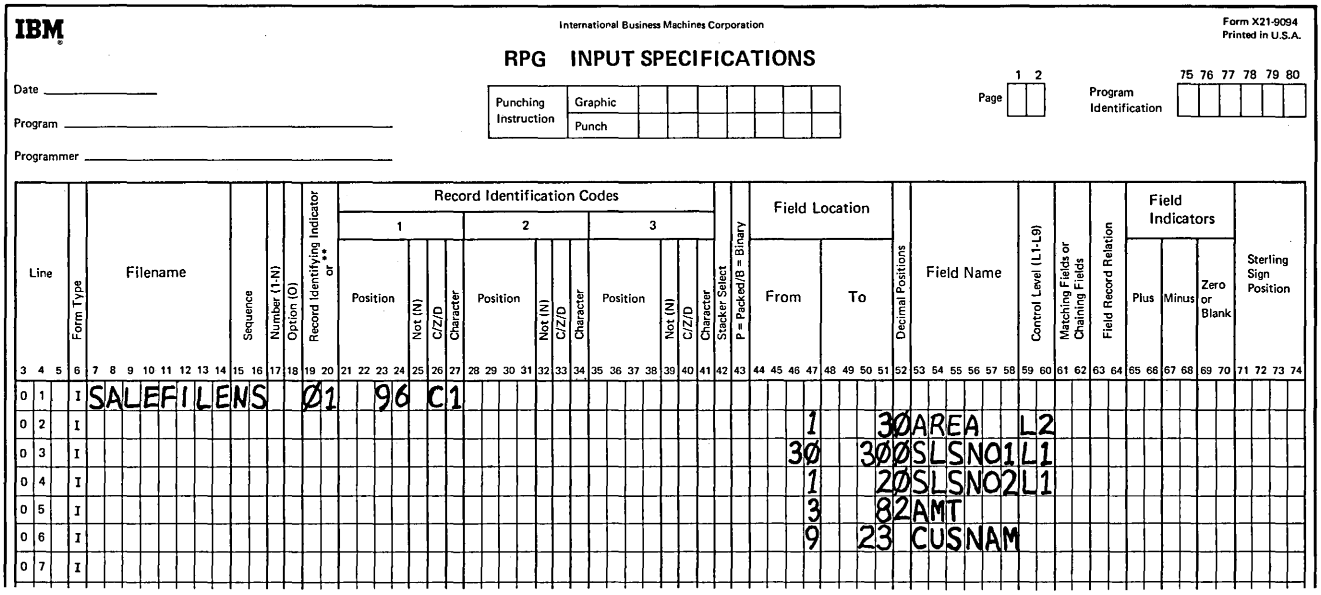
Punching Instruction: Graphic / Punch

Page 1 2

Program Identification 75 76 77 78 79 80

Line	Form Type	Filename	Sequence	Number (1-N)	Option (O)	Record Identifying Indicator or **	1 Position	1 Not (N)	1 C/Z/D	1 Character	2 Position	2 Not (N)	2 C/Z/D	2 Character	3 Position	3 Not (N)	3 C/Z/D	3 Character	Stacker Select	P = Packed/B = Binary	Field Location From	Field Location To	Decimal Positions	Field Name	Control Level (L1-L9)	Matching Fields or Chaining Fields	Field Record Relation	Field Indicators Plus	Field Indicators Minus	Field Indicators Zero or Blank	Sterling Sign Position
01	I	* HOURS WORKED																													
02	I		NS			3Ø	1		C	T																					
03	I																				2	3	Ø	WEEKNO							
04	I																				4	5	Ø	DEPT	L2						
05	I																				6	9	Ø	EMPNO	L1						
06	I																				1Ø	24		NAME							
07	I																				3Ø	33	1	HRSWKD							
08	I																														

1.



IBM

International Business Machines Corporation

Form X21-9094
Printed in U.S.A.

RPG INPUT SPECIFICATIONS

Date

Program

Programmer

Punching Instruction	Graphic							
	Punch							

Page (1 2)

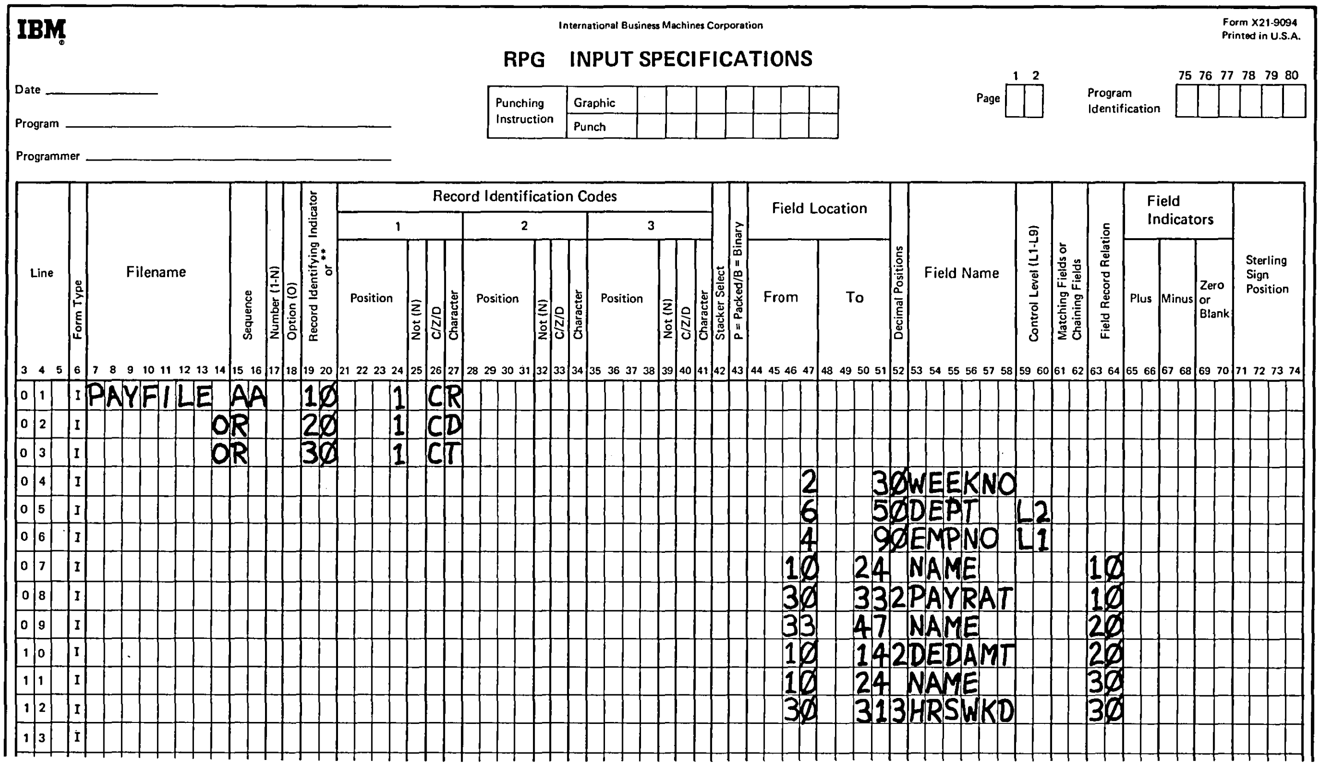
Program Identification (75 76 77 78 79 80)

Line (3-5)	Form Type (6)	Filename (7-14)	Sequence (15-16)	Number (1-N) (17)	Option (O) (18)	Record Identifying Indicator or ** (19-20)	Record Identification Codes 1: Position (21-24)	Not (N) (25)	C/Z/D (26)	Character (27)	Record Identification Codes 2: Position (28-31)	Not (N) (32)	C/Z/D (33)	Character (34)	Record Identification Codes 3: Position (35-38)	Not (N) (39)	C/Z/D (40)	Character (41)	Stacker Select (42)	P = Packed/B = Binary (43)	Field Location From (44-47)	Field Location To (48-51)	Decimal Positions (52)	Field Name (53-58)	Control Level (L1-L9) (59-60)	Matching Fields or Chaining Fields (61-62)	Field Record Relation (63-64)	Field Indicators Plus (65-66)	Minus (67-68)	Zero or Blank (69-70)	Sterling Sign Position (71-74)
01	I	SALEFILE	NS			Ø1	96		C	1																					
02	I																				1	3	Ø	AREA	L2						
03	I																				3Ø	3Ø	Ø	SLSNO1	L1						
04	I																				1	2	Ø	SLSNO2	L1						
05	I																				3	8	2	AMT							
06	I																				9	23		CUSNAM							
07	I																														

The two fields which make up the salesman number should be assigned the same control level indicator to indicate that both fields are to be considered as one.

The split control fields must be specified on two adjacent lines. Since the first digit of the salesman number is in column 30, this single digit field should be specified before the field containing the last two digits of the number. The computer determines the order in which the digits are to be arranged by the order in which the fields are specified.

2.



IBM

International Business Machines Corporation

Form X21-9094
Printed in U.S.A.

RPG INPUT SPECIFICATIONS

Date

Program

Programmer

Punching Instruction: Graphic / Punch

Page 1 2

Program Identification 75 76 77 78 79 80

Line	Form Type	Filename	Sequence	Number (1-N)	Option (O)	Record Identifying Indicator or **	Record Identification Codes 1: Position	Not (N)	C/Z/D	Character	2: Position	Not (N)	C/Z/D	Character	3: Position	Not (N)	C/Z/D	Character	Stacker Select	P = Packed/B = Binary	Field Location: From	To	Decimal Positions	Field Name	Control Level (L1-L9)	Matching Fields or Chaining Fields	Field Record Relation	Field Indicators: Plus	Minus	Zero or Blank	Sterling Sign Position
01	I	PAYFILE	AA			1Ø	1		C	R																					
02	I		OR			2Ø	1		C	D																					
03	I		OR			3Ø	1		C	T																					
04	I																				2	3	Ø	WEEKNO							
05	I																				6	5	Ø	DEPT	L2						
06	I																				4	9	Ø	EMPNO	L1						
07	I																				1Ø	24		NAME			1Ø				
08	I																				3Ø	33	2	PAYRAT			1Ø				
09	I																				33	47		NAME			2Ø				
10	I																				1Ø	14	2	DEDAMT			2Ø				
11	I																				1Ø	24		NAME			3Ø				
12	I																				3Ø	31	3	HRSWKD			3Ø				
13	I																														

Because these card types contain common fields, the OR relationship may be used to describe them. However, since not all fields are common to all card types, field record relation entries must also be used. All common fields – WEEKNO, EMPNO, and DEPT – are described first. The NAME field, although found on all card types, is in different locations. Thus, it must be related to all card types by specifying it and its end position three times and using the record identifying indicator in columns 63-64 to indicate the record type with which it is associated. PAYRAT is found in only card type 10. Thus 10 is placed in the Field Record Relation columns (64-64). DEDAMT and HRSWKD are related to the card type on which they are found in the same way. Remember that all fields related to one card type must be grouped together.

CHAPTER

USING THE PRINTER

13

CHAPTER 13 DESCRIBES:

RPG II overflow.

RPG II fetch overflow.

Use of the special RPG II word *PLACE.

Use of the dual feed carriage feature and coding for the two output files.

Use of the 5471 Printer-Keyboard as a second printer and coding of the File Description Sheet to describe it as an output device.

BEFORE READING THIS CHAPTER YOU SHOULD BE ABLE TO DESCRIBE THE FOLLOWING RPG II CONCEPTS:

Object cycle.

Function of RPG II indicators.

DEBUG operation code.

Automatic page formatting.

DSPLY operation code.

Disk system IPL statements and procedures.

AFTER READING THIS CHAPTER YOU SHOULD BE ABLE TO DESCRIBE:

Overflow.

Fetch overflow.

Effects of overflow on the RPG II object cycle.

*PLACE.

Use of the dual feed carriage feature and coding for the two output files.

Use of the 5471 Printer-Keyboard as a second printer and coding of the File Description Sheet to describe it as an output device.

INTRODUCTION

The most important part of any job is the result of the job – the output. This chapter describes five programming methods for controlling printed output:

1. Overflow signals that the end of a page has been reached and allows you to advance to a new page and print special lines such as headings.

2. Fetch overflow prevents records from printing over the perforation.

3. *PLACE is a special RPG II function that allows you to print duplicate information.

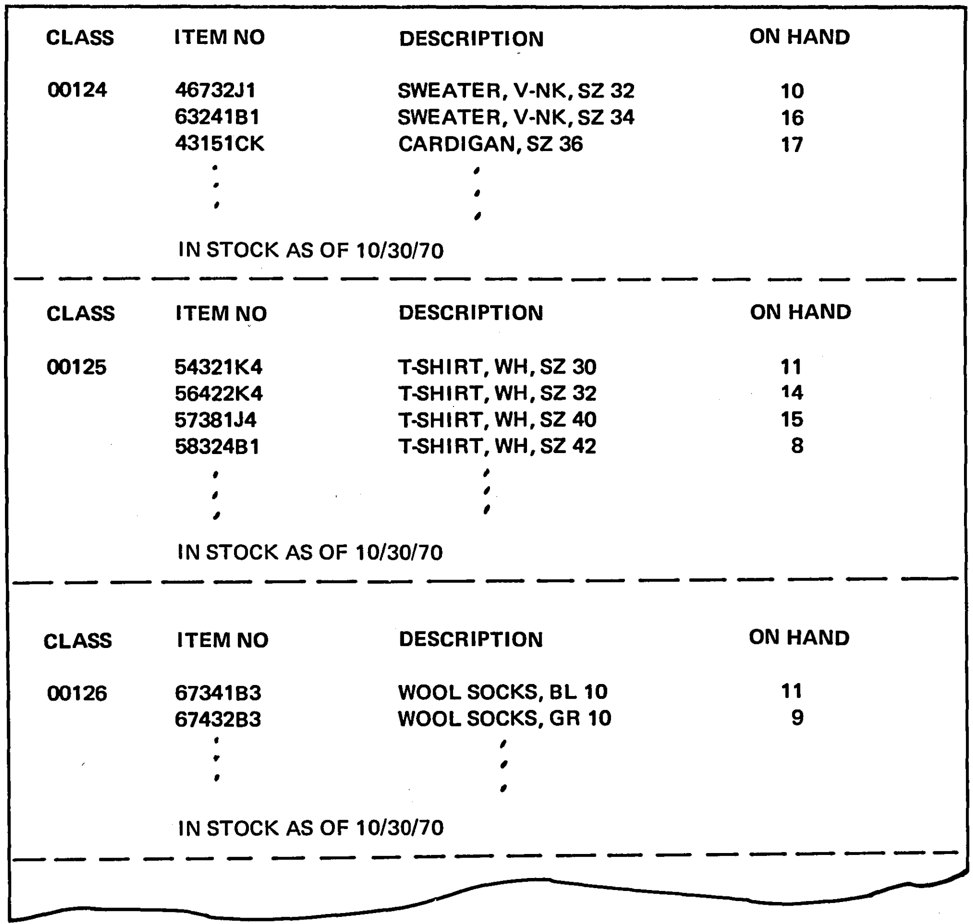
4. Using the dual feed carriage feature, you can produce two printer output files in the same program.

5. Using the 5471 Printer-Keyboard, you can communicate with the operator and have a second output device for RPG II programs.

USING OVERFLOW AND FETCH OVERFLOW TO CONTROL PAGE FORMATTING

RPG II performs automatic page formatting. It leaves five blank lines at the top of a page and six at the bottom. (Six lines are printed per inch.) Automatic page formatting may not always meet your needs. If you want control over page formatting, you can use an overflow indicator (OA-OG, OV). For instance, assume that at the end of every month you prepare an inventory report which consists of a list of the quantity of all items in stock by product class. Items are listed by product class, and each product class should start on a new page (Figure 13-1).

Suppose the heading were to start on line 11 of each page. To have an equal margin (ten spaces) on top and bottom, line 56 should be the last printed line on the page (assuming 66 lines per page). For this report, you must use an overflow indicator to control page format.



```
CLASS    ITEM NO     DESCRIPTION                ON HAND

00124    46732J1     SWEATER, V-NK, SZ 32          10
         63241B1     SWEATER, V-NK, SZ 34          16
         43151CK     CARDIGAN, SZ 36               17
            .              .
            .              .
            .              .

         IN STOCK AS OF 10/30/70
— — — — — — — — — — — — — — — — — — — — — — — — — — — —
CLASS    ITEM NO     DESCRIPTION                ON HAND

00125    54321K4     T-SHIRT, WH, SZ 30            11
         56422K4     T-SHIRT, WH, SZ 32            14
         57381J4     T-SHIRT, WH, SZ 40            15
         58324B1     T-SHIRT, WH, SZ 42             8
            .              .
            .              .
            .              .

         IN STOCK AS OF 10/30/70
— — — — — — — — — — — — — — — — — — — — — — — — — — — —

CLASS    ITEM NO     DESCRIPTION                ON HAND

00126    67341B3     WOOL SOCKS, BL 10             11
         67432B3     WOOL SOCKS, GR 10              9
            .              .
            .              .
            .              .

         IN STOCK AS OF 10/30/70
— — — — — — — — — — — — — — — — — — — — — — — — — — — —
```

Figure 13-1. End-of-Month Inventory Report

Overflow Indicators

Overflow indicators, like other indicators, are used to do two things:

- Signal a certain condition.

- Control when specific operations (including those which control page format) are performed.

For example, in the monthly inventory report, items in stock are listed by product class. The report consists of 46 lines per page (starting line is 11 and ending line 56). Some product classes are going to have more than 46 different items in stock. For these classes, additional pages (overflow pages) are required to list the items.

The *overflow line* is the last line you want to print on the page. For this report, the overflow line would be line 56. When this line is printed, the overflow indicator (if one is assigned) turns on to signal that the last line you wished printed on the page has been reached.

When the overflow indicator is on, you know that the overflow line has been reached. At the end of the page, operations, such as advancing to a new page (the overflow page) and printing headings on the new page, are performed. By assigning and using overflow indicators, RPG II allows you to print special lines at the bottom of the page and at the top of the new page. Because you do these operations only when the overflow indicator is on, you will have to condition these operations by the overflow indicator.

Specifications for Using Overflow Indicators

You must specify to the RPG II compiler how reports should be printed. To tell it what/to do, you make line counter, file description, and output-format specifications.

Line Counter Specifications

Line counter specifications, found on the bottom half of the Extension and Line Counter Sheet (Figure 13-2), are used exclusively for defining the number of lines you wish printed on each page.

Every time you use an overflow indicator to control formatting, you should prepare line counter specifications. Otherwise, a page length of 66 lines will be assumed with line 60 as overflow line.

Note: Standard forms are 11 inches long. The printer prints 6 lines to an inch. This provides 66 lines of printing per page.

Figure 13-3 is a sample Line Counter Sheet for the inventory report. Columns 7-14 are for filename. Only a *printer* file name can be used here. Columns 15-22 contain the entries for report formatting:

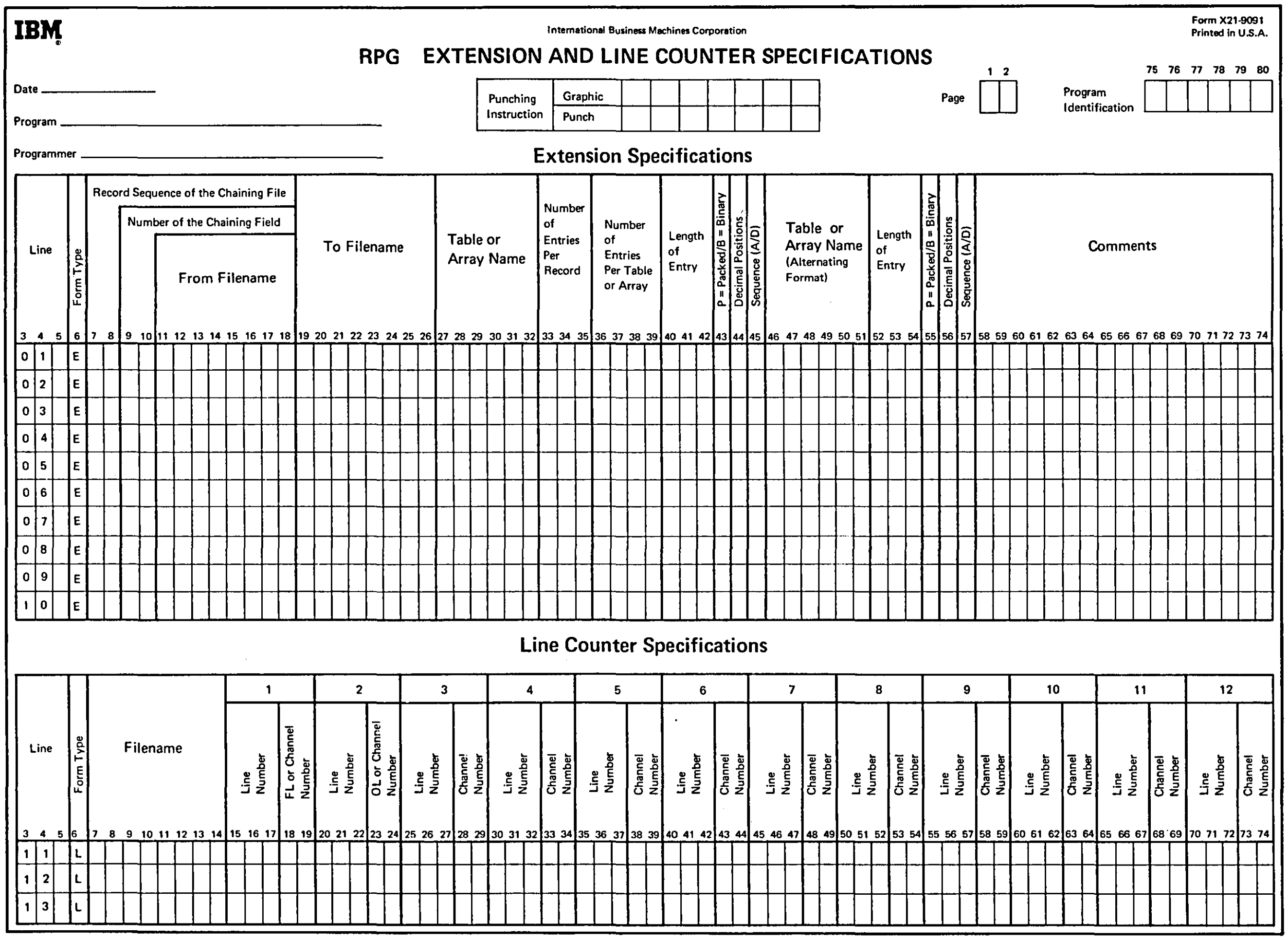
- Columns 15-17: Place in these columns the number of available lines per page. Your page can contain a maximum of 112 lines. Therefore, the maximum length of a page is slightly more than 18 inches. The inventory report uses standard 11 inch paper, providing 66 lines per page.

- Columns 18-19: Put the letters FL in these columns to show that the previous specifications gave form length.

- Columns 20-22: Enter in these columns the number of the overflow line, when you want the overflow indicator to be turned on. In the example given, it was 56. You can use any number from 1-112.

- Columns 23-24: Enter the letters OL in these columns to show that the previous specification was the overflow line.

Notice that columns 25-80 are not used.



IBM

International Business Machines Corporation

Form X21-9091
Printed in U.S.A.

RPG EXTENSION AND LINE COUNTER SPECIFICATIONS

Date

Program

Programmer

Punching Instruction	Graphic								
	Punch								

Page 1 2

Program Identification 75 76 77 78 79 80

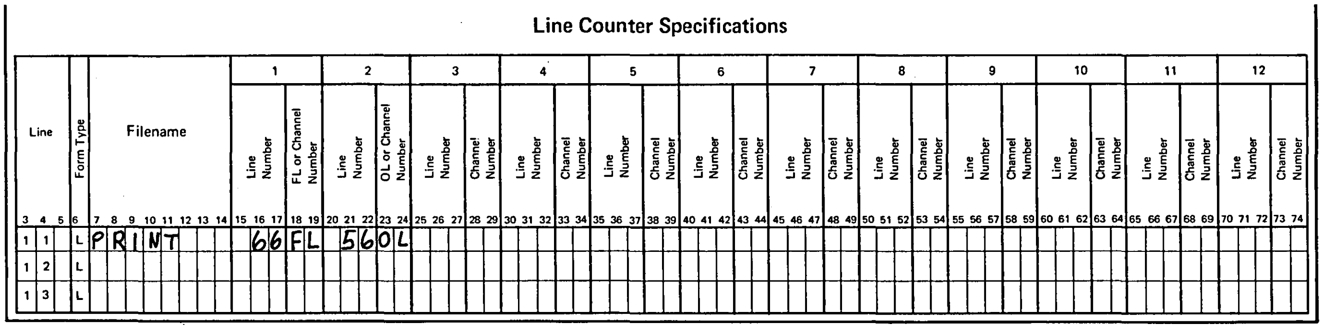
Extension Specifications

Line	Form Type	Record Sequence of the Chaining File	Number of the Chaining Field	From Filename	To Filename	Table or Array Name	Number of Entries Per Record	Number of Entries Per Table or Array	Length of Entry	P = Packed/B = Binary	Decimal Positions	Sequence (A/D)	Table or Array Name (Alternating Format)	Length of Entry	P = Packed/B = Binary	Decimal Positions	Sequence (A/D)	Comments
3 4 5	6	7 8	9 10	11 12 13 14 15 16 17 18	19 20 21 22 23 24 25 26	27 28 29 30 31 32	33 34 35	36 37 38 39	40 41 42	43	44	45	46 47 48 49 50 51	52 53 54	55	56	57	58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74
0 1	E																	
0 2	E																	
0 3	E																	
0 4	E																	
0 5	E																	
0 6	E																	
0 7	E																	
0 8	E																	
0 9	E																	
1 0	E																	

Line Counter Specifications

			1		2		3		4		5		6		7		8		9		10		11		12	
Line	Form Type	Filename	Line Number	FL or Channel Number	Line Number	OL or Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number
3 4 5	6	7 8 9 10 11 12 13 14	15 16 17	18 19	20 21 22	23 24	25 26 27	28 29	30 31 32	33 34	35 36 37	38 39	40 41 42	43 44	45 46 47	48 49	50 51 52	53 54	55 56 57	58 59	60 61 62	63 64	65 66 67	68 69	70 71 72	73 74
1 1	L																									
1 2	L																									
1 3	L																									

Figure 13-2. RPG II Extension and Line Counter Specification Sheet



Line Counter Specifications

			1		2		3		4		5		6		7		8		9		10		11		12	
Line	Form Type	Filename	Line Number	FL or Channel Number	Line Number	OL or Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number
3 4 5	6	7 8 9 10 11 12 13 14	15 16 17	18 19	20 21 22	23 24	25 26 27	28 29	30 31 32	33 34	35 36 37	38 39	40 41 42	43 44	45 46 47	48 49	50 51 52	53 54	55 56 57	58 59	60 61 62	63 64	65 66 67	68 69	70 71 72	73 74
1 1	L	PRINT	66	FL	56	OL																				
1 2	L																									
1 3	L																									

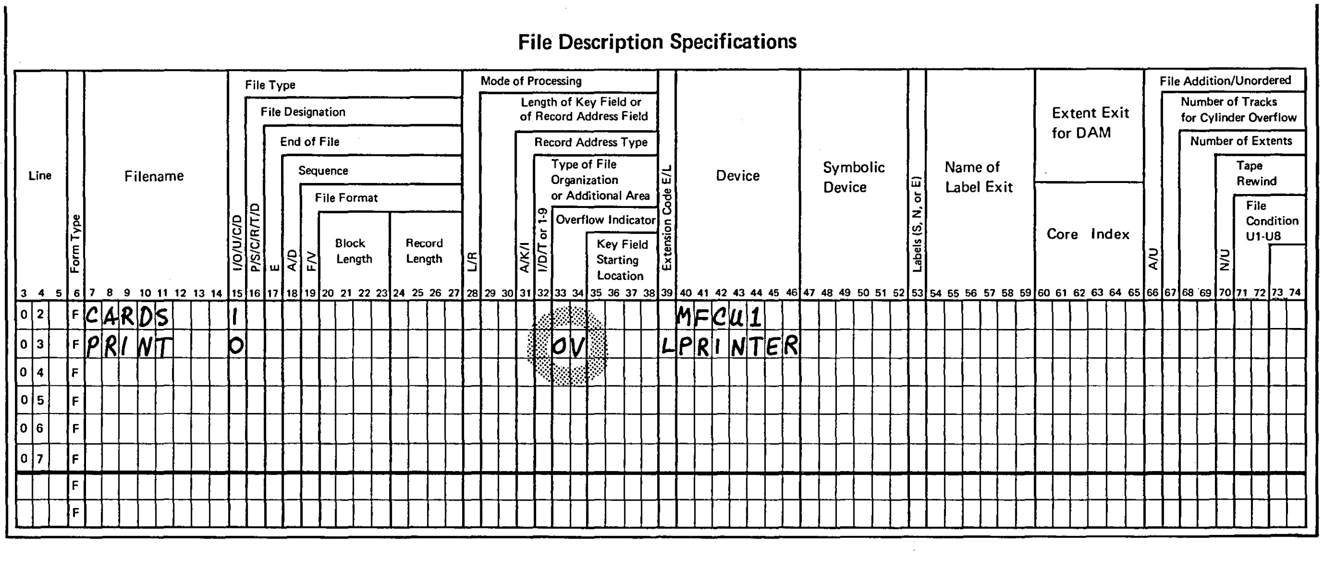
Figure 13-3. Line Counter Specifications

You must assign an overflow indicator to the *printer file* when you want to control the format of printed reports. This is done by an entry in columns 33-34 of the File Description Sheet (Figure 13-4). You may choose to enter any of the following overflow indicators: OA, OB, OC, OD, OE, OF, OG, or OV. The one you choose, however, must be used throughout the program. *L* must also be entered in column 39 to indicate that line counter specifications are used.

These two entries indicate to the RPG II compiler that it should not provide automatic page formatting, but should format according to your specifications.

When RPG II handles overflow, pages are advanced automatically. When you handle overflow, you must specify that forms should advance. This is done by specifying a skip to the first printing line on the page, line 11 in this case. For this job, the heading line would be first. Figure 13-5 shows the correct specification for forms advancement. Remember to make a skip specification on a line conditioned by the overflow indicator (Figure 13-5). If you forget, a continuous listing will be the result.

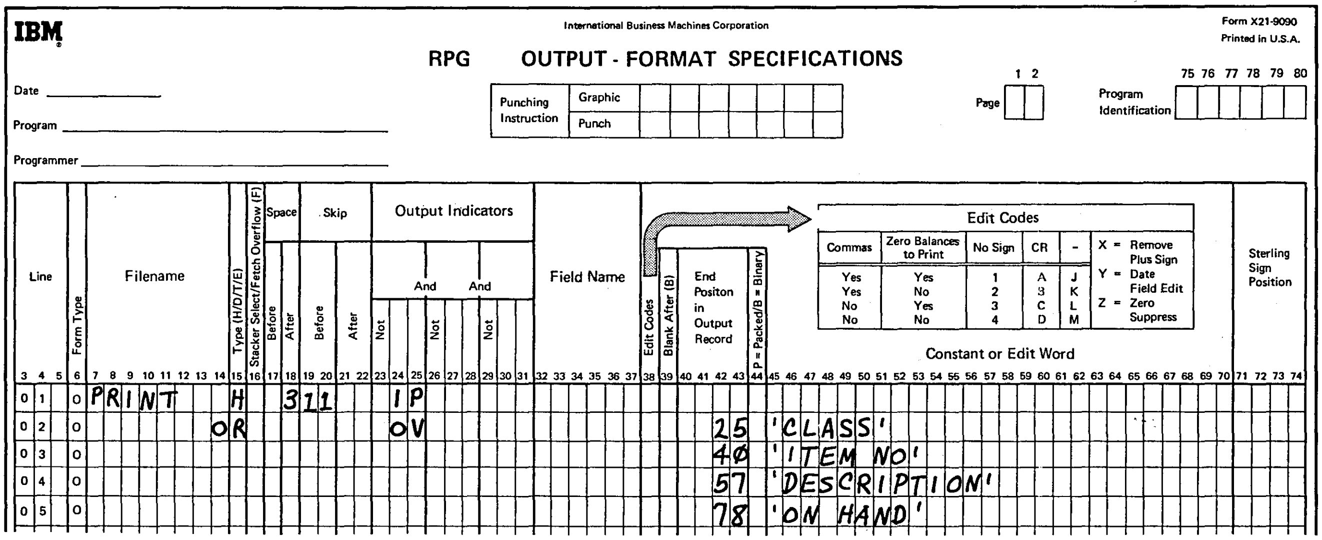
When the printer reaches the end of a printed page, RPG II also allows you to ignore that the end of the page has been reached and continue printing. You do this by assigning



File Description Specifications

Line (3-5)	Form Type (6)	Filename (7-14)	File Type (15)	Overflow Indicator (33-34)	Extension Code E/L (39)	Device (40-46)
02	F	CARDS	I			MFCU1
03	F	PRINT	O	OV	L	PRINTER
04	F					
05	F					
06	F					
07	F					
	F					
	F					

Figure 13-4. Assigning an Overflow Indicator to the Printer



IBM International Business Machines Corporation Form X21-9090 Printed in U.S.A.

RPG OUTPUT - FORMAT SPECIFICATIONS

Line (3-5)	Form Type (6)	Filename (7-14)	Type H/D/T/E (15)	Space After (18)	Skip Before (19-20)	Output Indicators (23-25)	End Position in Output Record (40-43)	Constant or Edit Word (45-70)
01	O	PRINT	H	3	11	IP		
02	O	OR				OV	25	'CLASS'
03	O						4Ø	'ITEM NO'
04	O						57	'DESCRIPTION'
05	O						78	'ON HAND'

Figure 13-5. Specification for Forms Advancement

an overflow indicator and never using it to condition output files. Lines will be printed from the top line to the bottom line of each page, even over the perforation. If you do not want this to happen, remember to use an overflow indicator to condition the output operations which are to be done when the end of the page is reached.

Overflow Logic

Figure 13-6 shows one page of a report printed according to the line counter and output-format specifications shown in Figure 13-7. The heading line is printed on line 11 as was specified, but the last line printed was 57. This happened as a result of RPG II overflow logic (Figure 13-8).

According to the logic, there are two times at which the overflow indicator can be turned on:

1. At total time when total records are printed.

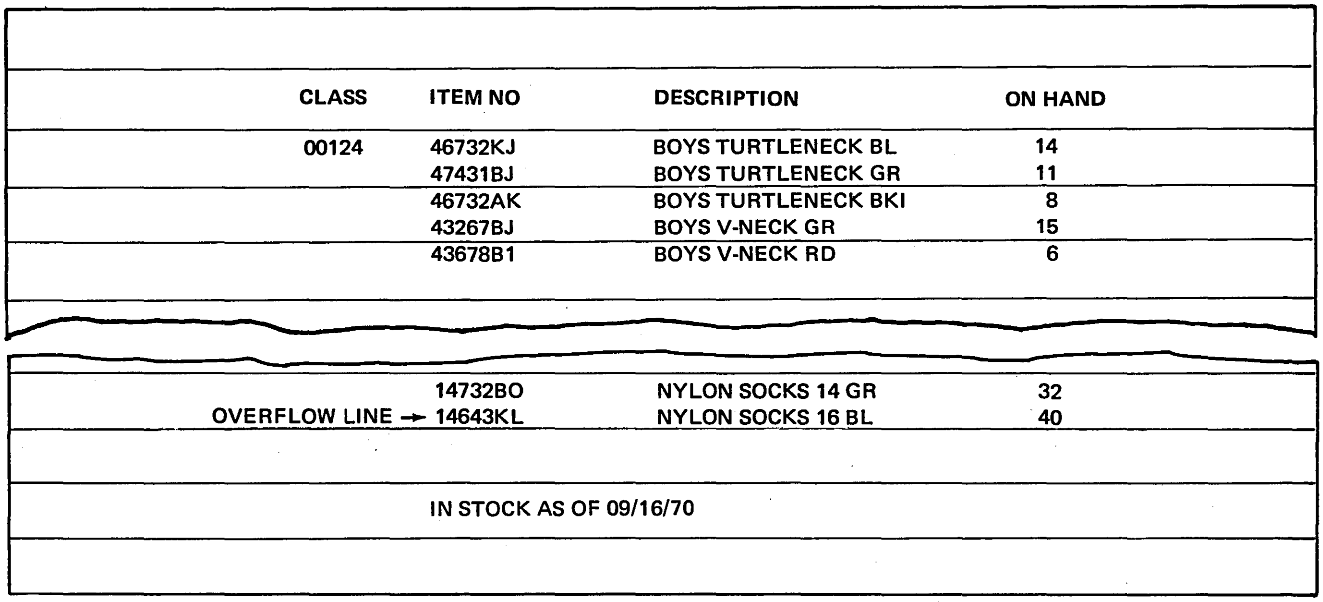
2. At detail time when detail records are printed.

There is, however, only one time during each cycle when the program checks to see if the overflow indicator is on. This is immediately before detail calculations are performed.

Follow this logic, as shown in Figure 13-8, step by step. First, a record is read. If this card causes a control break, total calculations and total output operations are performed. If a record is printed on the overflow line, the indicator turns on. A test is then made to determine if the overflow indicator is on. If it is on, all operations conditioned by the overflow indicator are performed, and the indicator is turned off. Detail calculations and output operations are performed last. If a detail record is printed on the overflow line, the overflow indicator will turn on, but the program will not check to see if the indicator is on until after total output in the next cycle. Thus, lines can be printed past the overflow line simply because the computer does not as yet know that the overflow indicator is on.

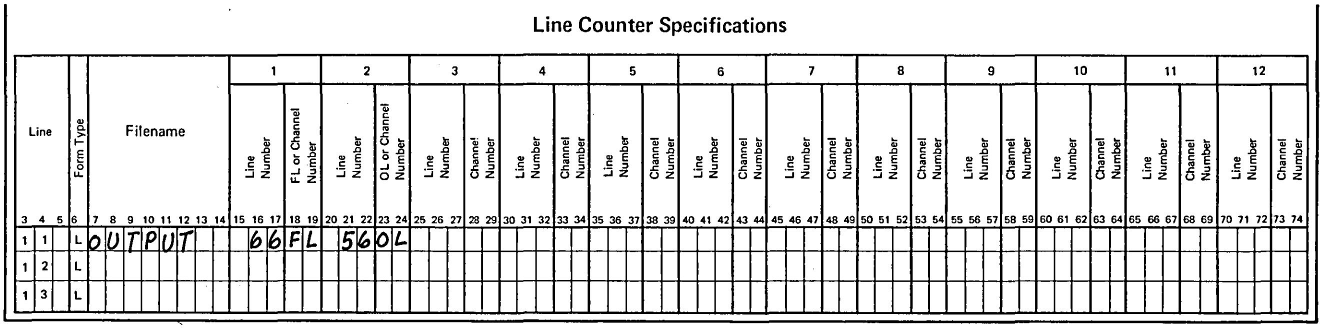
The overflow indicator was turned on at detail time for the inventory report shown in Figure 13-6, because a detail record was printed on the overflow line.

The program did not know that the overflow line was reached, since no check was made at this point. Therefore, another record is read. In this case, assume that the record is from department 00125. A control break occurs because the previous record was from department 00124. Thus, total operations are performed. The total record specified in lines 12 and 13 of the Output-Format Sheet shown in Figure 13-7 is printed on line 57, one line past the overflow line. It is only *after* total records are printed that the computer knows overflow has occurred (Figure 13-8). Therefore, the total record was printed after the overflow line has been reached.



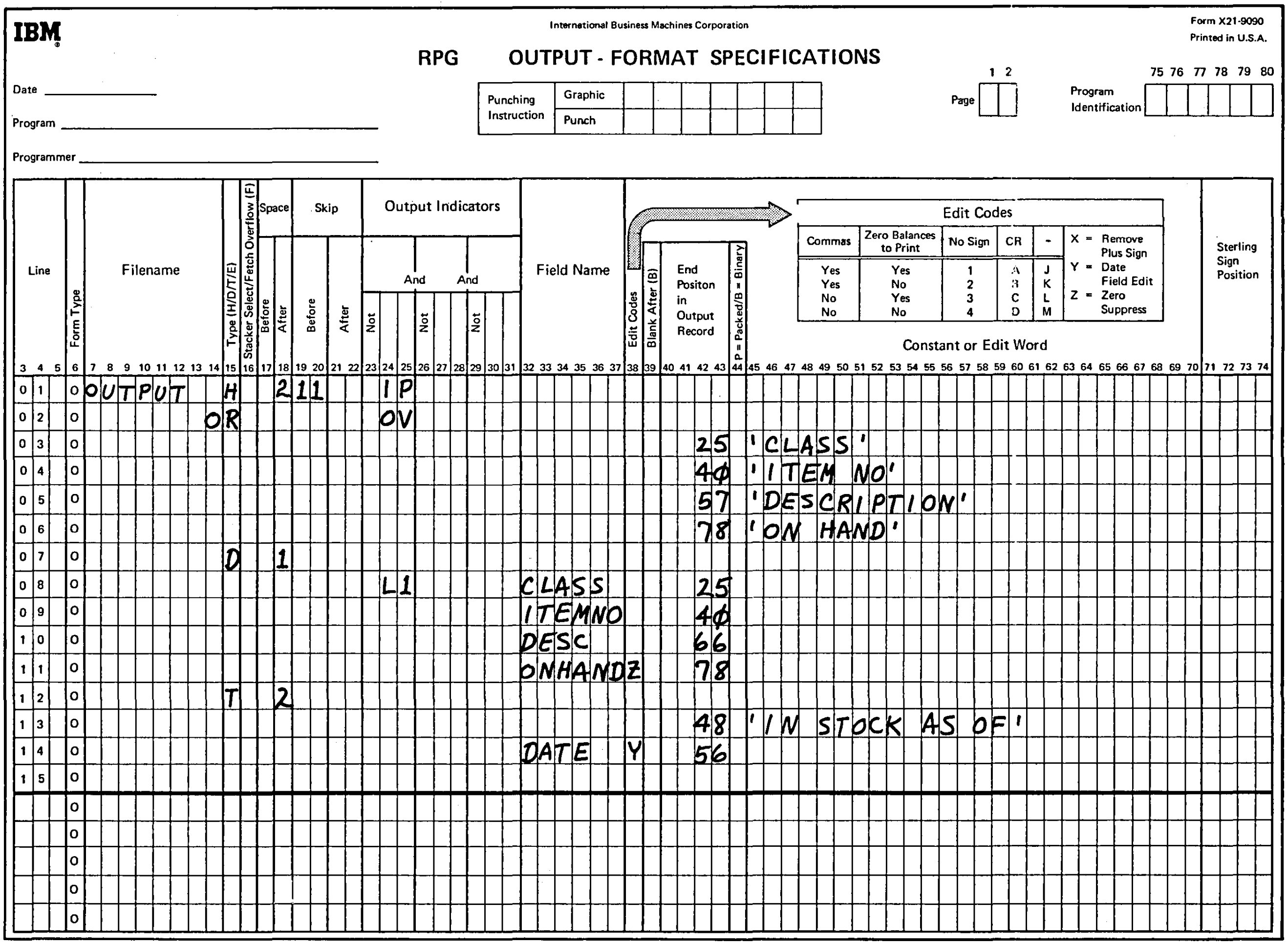
```
                    CLASS      ITEM NO          DESCRIPTION                 ON HAND

                    00124      46732KJ          BOYS TURTLENECK BL            14
                               47431BJ          BOYS TURTLENECK GR            11
                               46732AK          BOYS TURTLENECK BKI            8
                               43267BJ          BOYS V-NECK GR                15
                               43678B1          BOYS V-NECK RD                 6

                               14732BO          NYLON SOCKS 14 GR             32
            OVERFLOW LINE -->  14643KL          NYLON SOCKS 16 BL             40


                               IN STOCK AS OF 09/16/70
```

Figure 13-6. Inventory Report Showing Overflow Line



Line Counter Specifications

Line (3-5)	Form Type (6)	Filename (7-14)	1 Line Number (15-17)	1 FL or Channel Number (18-19)	2 Line Number (20-22)	2 OL or Channel Number (23-24)	3 Line Number	3 Channel Number	4 Line Number	4 Channel Number	5 Line Number	5 Channel Number	6 Line Number	6 Channel Number	7 Line Number	7 Channel Number	8 Line Number	8 Channel Number	9 Line Number	9 Channel Number	10 Line Number	10 Channel Number	11 Line Number	11 Channel Number	12 Line Number	12 Channel Number
1 1	L	OUTPUT	66	FL	56	OL																				
1 2	L																									
1 3	L																									



IBM

International Business Machines Corporation

Form X21-9090
Printed in U.S.A.

RPG OUTPUT - FORMAT SPECIFICATIONS

Date ______________

Program ______________

Programmer ______________

Punching Instruction: Graphic □□□□□□□□ Punch □□□□□□□□

Page (1 2) □□ Program Identification (75 76 77 78 79 80) □□□□□□

Edit Codes

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Line (3-5)	Form Type (6)	Filename (7-14)	Type (H/D/T/E) (15)	Stacker Select/Fetch Overflow (F) (16)	Space Before (17)	Space After (18)	Skip Before (19-20)	Skip After (21-22)	Output Indicators (23-31)	Field Name (32-37)	Edit Codes (38)	Blank After (B) (39)	End Position in Output Record (40-43)	P = Packed/B = Binary (44)	Constant or Edit Word (45-70)	Sterling Sign Position (71-74)
0 1	O	OUTPUT	H			2	11		1P							
0 2	O	OR							OV							
0 3	O												25		'CLASS'	
0 4	O												4Ø		'ITEM NO'	
0 5	O												57		'DESCRIPTION'	
0 6	O												78		'ON HAND'	
0 7	O		D			1										
0 8	O								L1	CLASS			25			
0 9	O									ITEMNO			4Ø			
1 0	O									DESC			66			
1 1	O									ONHAND	Z		78			
1 2	O		T			2										
1 3	O												48		'IN STOCK AS OF'	
1 4	O									DATE	Y		56			
1 5	O															
	O															
	O															
	O															
	O															
	O															

Figure 13-7. Specifications for Controlling the Format of the Inventory Report

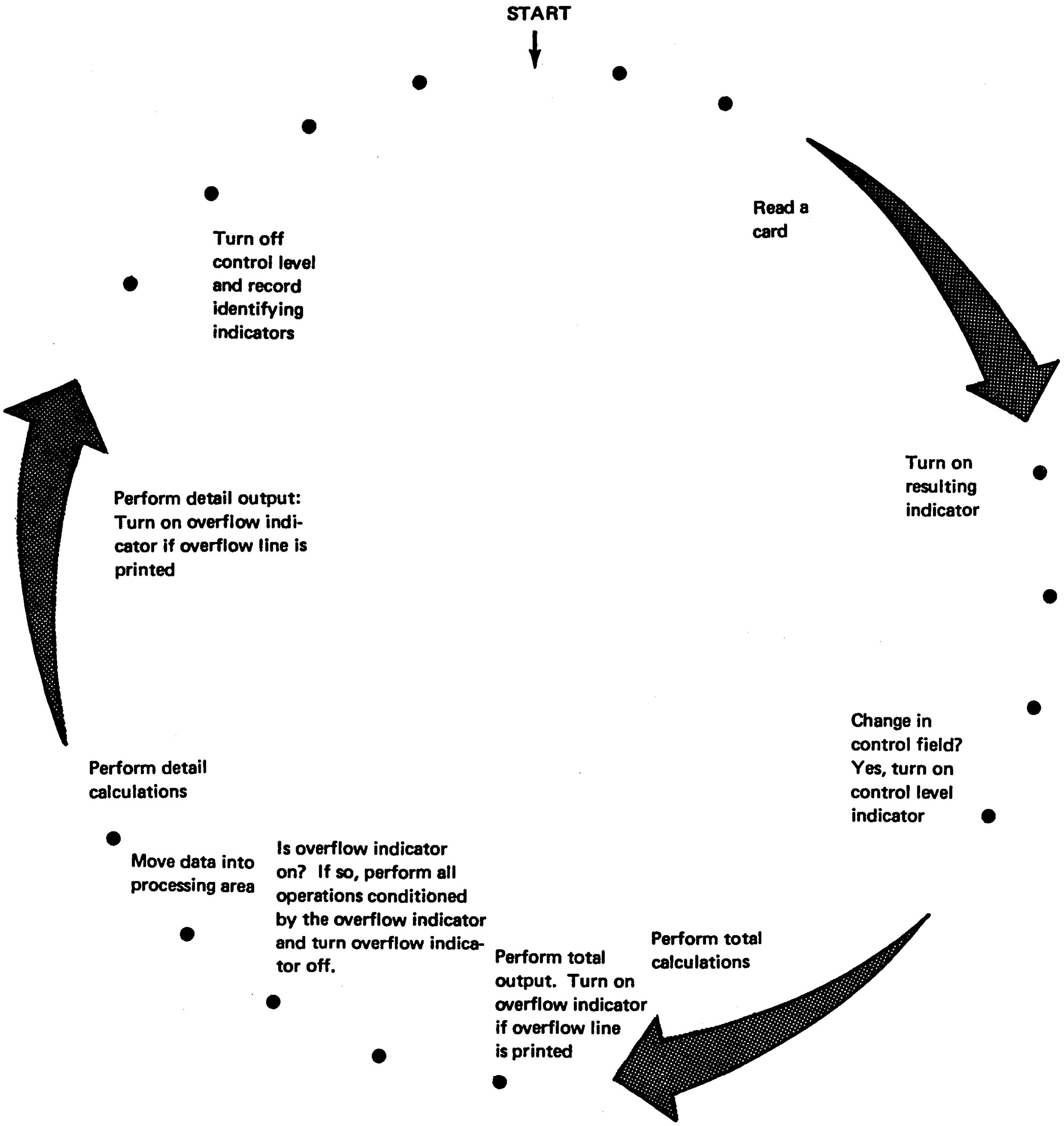
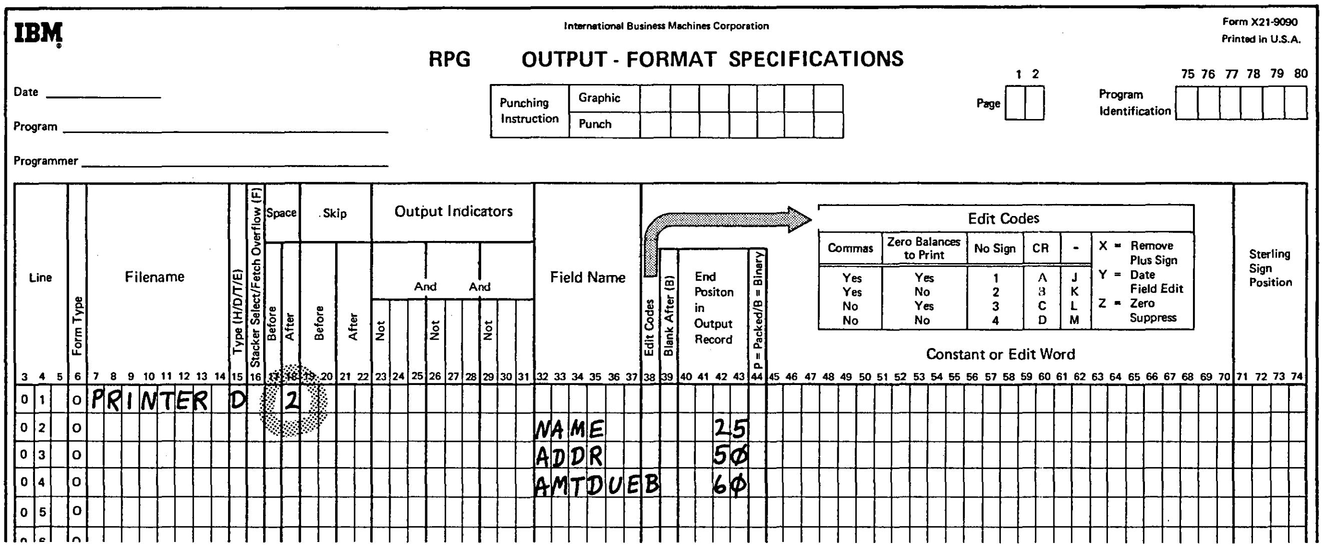

START
Read a card
Turn on resulting indicator
Change in control field? Yes, turn on control level indicator
Perform total calculations
Perform total output. Turn on overflow indicator if overflow line is printed
Is overflow indicator on? If so, perform all operations conditioned by the overflow indicator and turn overflow indicator off.
Move data into processing area
Perform detail calculations
Perform detail output: Turn on overflow indicator if overflow line is printed
Turn off control level and record identifying indicators

Figure 13-8. Logic for Overflow

The Effect of Skipping and Spacing On Overflow

So far, you have learned that the overflow indicator is turned on when a record is *printed* on the overflow line. Actually, the overflow indicator is turned on whenever the overflow line is reached. This means that spacing or skipping to a line past the overflow line will also cause the overflow indicator to turn on. However, if the skip specification skips past the overflow line to the next page, the indicator does not turn on. In this case there is no need for it to be on since a new page was advanced anyway.

For example, assume the overflow line for a job is line 58. Assume also that the detail record specified in Figure 13-9 was printed on line 57. This would not turn the overflow indicator on. However, this same detail specification allows for two spaces. Spacing two lines moves line 59 into printing position. Although line 58 has not been printed, it has been passed, and the overflow indicator will be turned on to indicate this.



IBM

International Business Machines Corporation

Form X21-9090
Printed in U.S.A.

RPG OUTPUT - FORMAT SPECIFICATIONS

Date ______

Program ______

Programmer ______

Punching Instruction	Graphic								
	Punch								

Page (1 2) □□

Program Identification (75 76 77 78 79 80) □□□□□□

Line (3-5)	Form Type (6)	Filename (7-14)	Type (H/D/T/E) (15)	Stacker Select/Fetch Overflow (F) (16)	Space Before (17)	Space After (18)	Skip Before (19-20)	Skip After (21-22)	Output Indicators: Not (23-25)	And: Not (26-28)	And: Not (29-31)	Field Name (32-37)	Edit Codes (38)	Blank After (B) (39)	End Positon in Output Record (40-43)	P = Packed/B = Binary (44)	Constant or Edit Word (45-70)	Sterling Sign Position (71-74)
01	O	PRINTER	D			2												
02	O											NAME			25			
03	O											ADDR			5Ø			
04	O											AMTDUEB			6Ø			
05	O																	
06	O																	

Edit Codes

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Figure 13-9. Space Specification

Printing Over the Perforation

Sometimes lines are printed over the perforation because the overflow indicator has not been tested. To eliminate this situation you can:

- Specify the overflow line high enough on the page to ensure that the overflow indicator will be sensed (Figure 13-10).
- Specify fetch overflow.

Preventing Records From Printing Over the Perforation By Fetch Overflow

Fetch overflow specifications allow you to alter the basic RPG II overflow logic by checking the overflow indicator before printing records. You can cause forms to advance at the time that total or detail records are printed. Figure 13-11 shows the two additional times when operations conditioned by the overflow indicator may be performed. (Remember that forms advance at this time.)

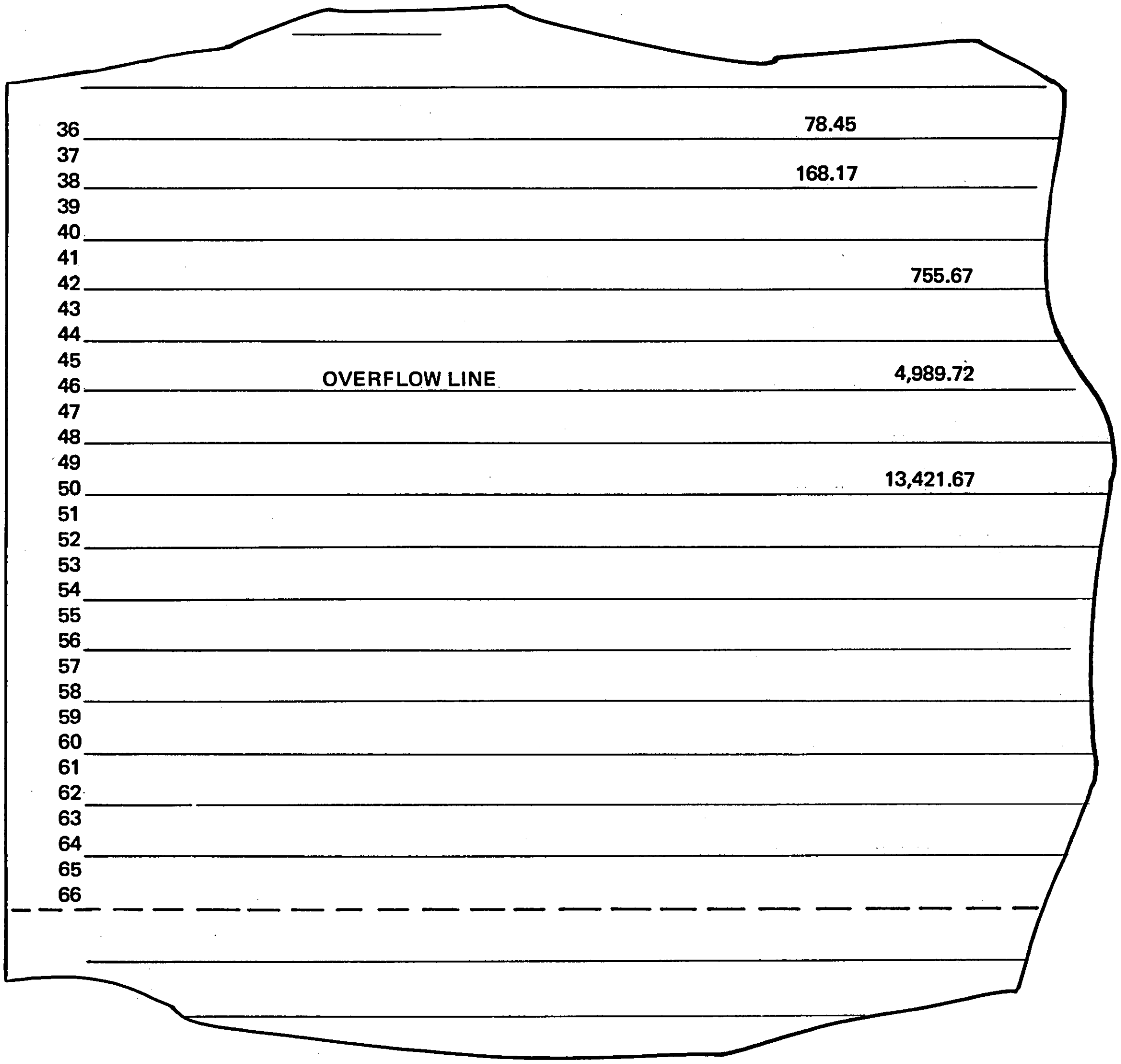


Figure 13-10. Specifying the Overflow Line High on the Page

During the regular program cycle, RPG II tests to see if the overflow indicator is on after a total or detail record is printed. By using the fetch overflow specification, you can tell the computer to determine if the overflow indicator is on *before* it prints total or detail records. You do this by entering an *F* in column 16 of the Output-Format Sheet for any detail or total record. When an *F* is encountered, a test is made.

If the overflow indicator is on when the tests is made, all operations conditioned by the overflow indicator are immediately performed. These operations usually include forms advancement and the printing of headings.

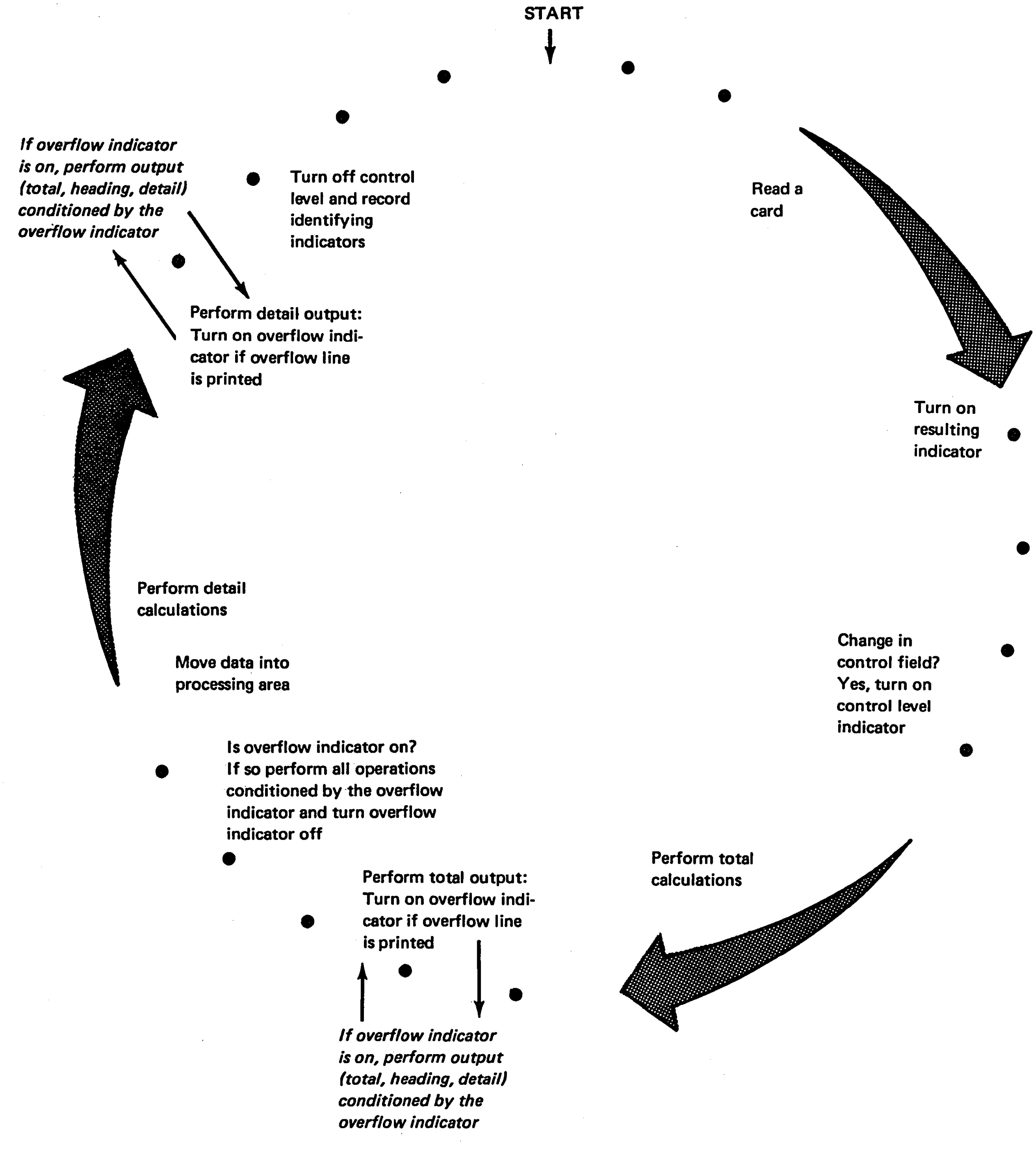

START
Read a card
Turn on resulting indicator
Change in control field? Yes, turn on control level indicator
Perform total calculations
Perform total output: Turn on overflow indicator if overflow line is printed
If overflow indicator is on, perform output (total, heading, detail) conditioned by the overflow indicator
Is overflow indicator on? If so perform all operations conditioned by the overflow indicator and turn overflow indicator off
Move data into processing area
Perform detail calculations
Perform detail output: Turn on overflow indicator if overflow line is printed
If overflow indicator is on, perform output (total, heading, detail) conditioned by the overflow indicator
Turn off control level and record identifying indicators

Figure 13-11. Logic for Fetch Overflow

Figure 13-12 shows two fetch overflow specifications (lines 07 and 09). Consider how these operations are performed. When it is time for the specifications in line 07 to be done, a test is made to see if the overflow routine is fetched; if the overflow indicator is on, the following operations are performed:

1. All total lines conditioned by the overflow indicator are printed.

2. Forms are advanced (provided a skip to 01 has been specified in a line conditioned by the overflow indicator).

3. Heading lines conditioned by the overflow indicator are printed.

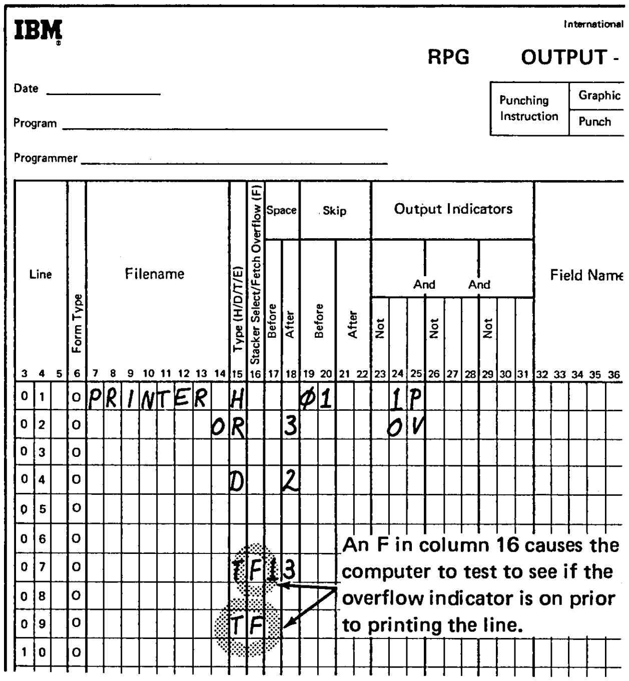
4. The overflow indicator is turned off.

5. The record specified in line 07 is printed.

Another test is made to see if the overflow indicator is on because of the specification (F) in line 09. If line 07 causes forms to advance, the overflow indicator would not be on at this time. The total record specified in specification line 09 would print normally.

However, if the record specified in line 07 were printed on the overflow line, the overflow indicator would be on, and the specification in line 09 would cause the overflow routine to be performed.

To determine where to place the *F* that will fetch the overflow routine (provided the overflow indicator is on), you should study all possible overflow situations. By counting spaces and lines, you can calculate what would happen if overflow occurred on each detail and total line.



IBM International

RPG OUTPUT -

Date ____________

Program ____________

Programmer ____________

Punching Instruction | Graphic | Punch

Line (3-5)	Form Type (6)	Filename (7-14)	Type (H/D/T/E) (15)	Stacker Select/Fetch Overflow (F) (16)	Space Before (17)	Space After (18)	Skip Before (19-20)	Skip After (21-22)	Output Indicators (23-25)	And (26-28)	And (29-31)	Field Name (32-36)
01	O	PRINTER	H				Ø1		1P			
02	O	O	R			3			OV			
03	O											
04	O		D			2						
05	O											
06	O											
07	O		T	F	1	3						
08	O											
09	O		T	F								
10	O											

An F in column 16 causes the computer to test to see if the overflow indicator is on prior to printing the line.

Figure 13-12. Fetch Overflow Specifications

Review 13. Overflow and Fetch Overflow

1. When you are not using overflow indicators or line counter specifications but are allowing RPG II to handle overflow automatically, how many lines are assumed per page? What is the first line printed? What is the overflow line?

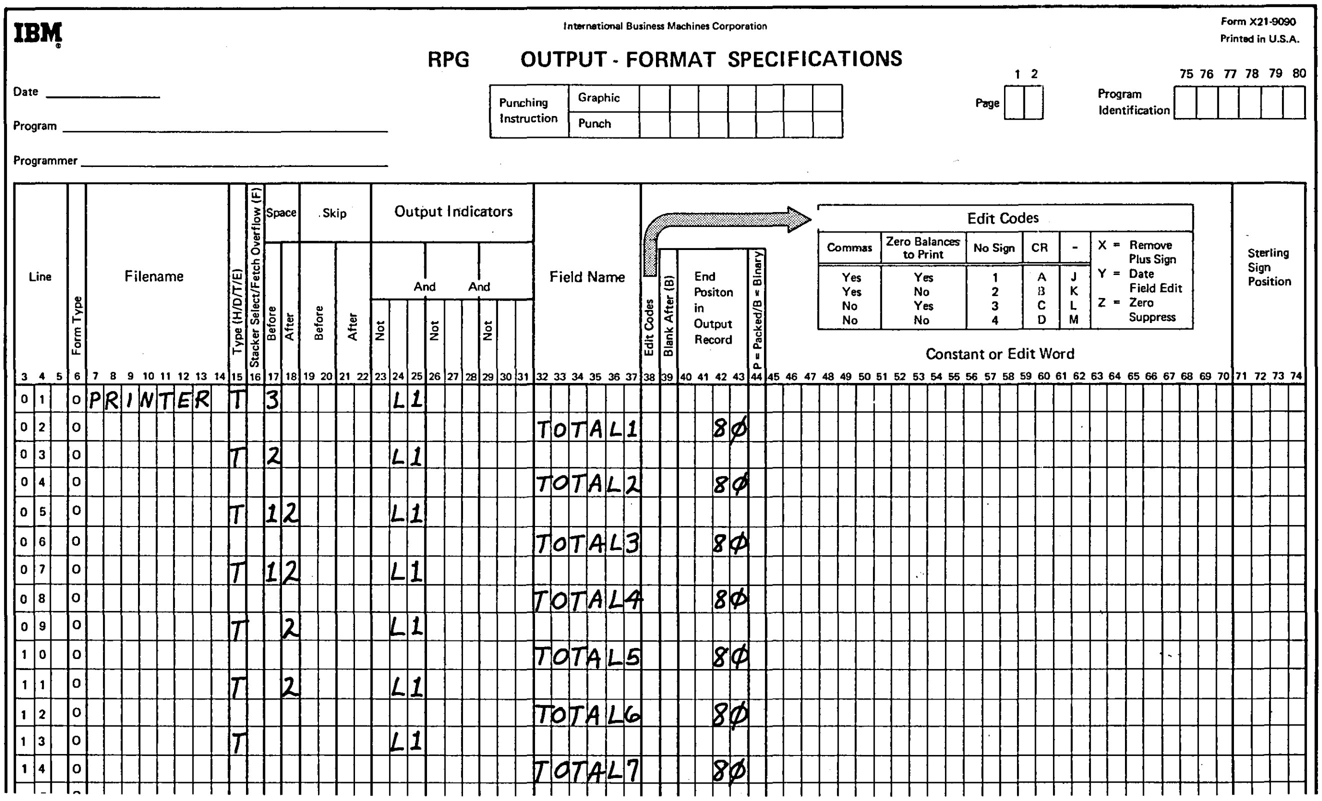
2. Code the line counter specifications which are necessary to define a form of 50 lines with the overflow line eight lines from the bottom? What entry must also be made on the File Description Sheet?

3. Describe a situation where printing can occur below the overflow line.

4. How does the fetch overflow specification alter the normal program cycle?

5. Given the following information, supply the fetch specifications for the job shown in the following output-format, which will prevent printing records on or over the perforation.

   - Number of printing lines per page is 66. The overflow line is 58.
   - There are seven total lines in all. Since all are conditioned by the same control level indicator, they will all print when a level 1 control break occurs.
   - Overflow should be forced if the overflow line is printed *prior* to beginning total output.
   - Total lines 1, 2, and 3 must print on the same page. Total lines 4 through 7 must print on the same page.



IBM

International Business Machines Corporation

Form X21-9090
Printed in U.S.A.

RPG OUTPUT - FORMAT SPECIFICATIONS

Date ______

Program ______

Programmer ______

Punching Instruction: Graphic □□□□□□□□□□ Punch □□□□□□□□□□

Page (1 2) □□

Program Identification (75 76 77 78 79 80) □□□□□□

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Line (3-5)	Form Type (6)	Filename (7-14)	Type (H/D/T/E) (15)	Stacker Select/Fetch Overflow (F) (16)	Space Before (17)	Space After (18)	Skip Before (19-20)	Skip After (21-22)	Output Indicators (23-31)	Field Name (32-37)	Edit Codes (38)	Blank After (B) (39)	End Position in Output Record (40-43)	P = Packed/B = Binary (44)	Constant or Edit Word (45-70)	Sterling Sign Position (71-74)
01	O	PRINTER	T		3				L1							
02	O									TOTAL1			8∅			
03	O		T		2				L1							
04	O									TOTAL2			8∅			
05	O		T		1	2			L1							
06	O									TOTAL3			8∅			
07	O		T		1	2			L1							
08	O									TOTAL4			8∅			
09	O		T			2			L1							
10	O									TOTAL5			8∅			
11	O		T			2			L1							
12	O									TOTAL6			8∅			
13	O		T						L1							
14	O									TOTAL7			8∅			

Answers to Review 13. Overflow and Fetch Overflow

1. Sixty-six lines are assumed per page. First line printed is 06 and the overflow line is 60.

2.

File Description Specifications

Line 3-5	Form Type 6	Filename 7-14	File Type I/O/U/C/D 15	File Designation P/S/C/R/T/D 16	End of File E 17	Sequence A/D 18	File Format F/V 19	Block Length 20-23	Record Length 24-27	Mode of Processing L/R 28	Length of Key Field or of Record Address Field 29-30	Record Address Type A/K/I 31	Type of File Organization or Additional Area I/D/T or 1-9 32	Overflow Indicator 33-34	Key Field Starting Location 35-38	Extension Code E/L 39	Device 40-46	Symbolic Device 47-52	Labels (S, N, or E) 53	Name of Label Exit 54-59	Extent Exit for DAM / Core Index 60-65	A/U 66	File Addition/Unordered 67	Number of Tracks for Cylinder Overflow 68-69	Number of Extents N/U 70	Tape Rewind 71-72	File Condition U1-U8 73-74
02	F	PRINT	O						96					OF			PRINTER										

Line Counter Specifications

Line 3-5	Form Type 6	Filename 7-14	1 Line Number 15-17	1 FL or Channel Number 18-19	2 Line Number 20-22	2 OL or Channel Number 23-24	3 Line Number 25-27	3 Channel Number 28-29	4 Line Number 30-32	4 Channel Number 33-34	5 Line Number 35-37	5 Channel Number 38-39	6 Line Number 40-42	6 Channel Number 43-44	7 Line Number 45-47	7 Channel Number 48-49	8 Line Number 50-52	8 Channel Number 53-54	9 Line Number 55-57	9 Channel Number 58-59	10 Line Number 60-62	10 Channel Number 63-64	11 Line Number 65-67	11 Channel Number 68-69	12 Line Number 70-72	12 Channel Number 73-74
11	L	PRINT	Ø5Ø	FL	Ø42	OL																				
12	L																									
13	L																									

Form length is 50 lines and overflow is 42. Any overflow indicator OA-OG or OV must be entered in columns 33-34 of the File Description Sheet.

3. a. Printing can occur below the overflow line when more than one detail or total line is printed during one cycle and a line other than the last total line is printed on the overflow line.

   b. When the last detail line for a control group prints on the overflow line, the total lines for that group will print below the overflow line.

4. The overflow indicator is tested prior to printing each line specified with fetch overflow. If the indicator is on when tested, overflow output is performed immediately and then the line specified is printed. Normally, the indicator will not be tested until after total output.

5. *F* in column 16 of lines 1 and 4 of the Output-Format Sheet. *F* in column 16 of line 1 will assure that total 1 will not print below the overflow line, thus causing total 3 to fall over the perforation. Since 1, 2, and 3 must all be on the same page, no fetch should be specified for lines 2 and 3. Since totals 4 through 7 must all print on the same page and will not all fit below the overflow line, enter *F* in the specifications for total 4 to cause a skip to the next page if the overflow indiaator is on. Since totals 5 through 7 must print on the same page as total 4, no fetch specification should be entered for them.

USING *PLACE TO PRINT DUPLICATE INFORMATION

Using *PLACE, you can tell the RPG II compiler to print duplicate information. When you specify *PLACE on the Output-Format Sheet, the fields listed above it will be printed in a different position on the same line. This eliminates much duplicate coding.

For example, assume that your distribution firm prepares invoices on their data processing system. The invoice (Figure 13-13) sent to each customer consists of two parts: one part the customer keeps, the other he tears off and sends along with his payment. Many fields are common to both parts of the invoice. For example, NAME and CUSTNO (customer number) are printed on the first line of each part. All fields in the fourth line of the report, except for the description (DESC) fields, and all fields in the total line are found in both parts of the invoice. The second part is almost a duplicate of the first.

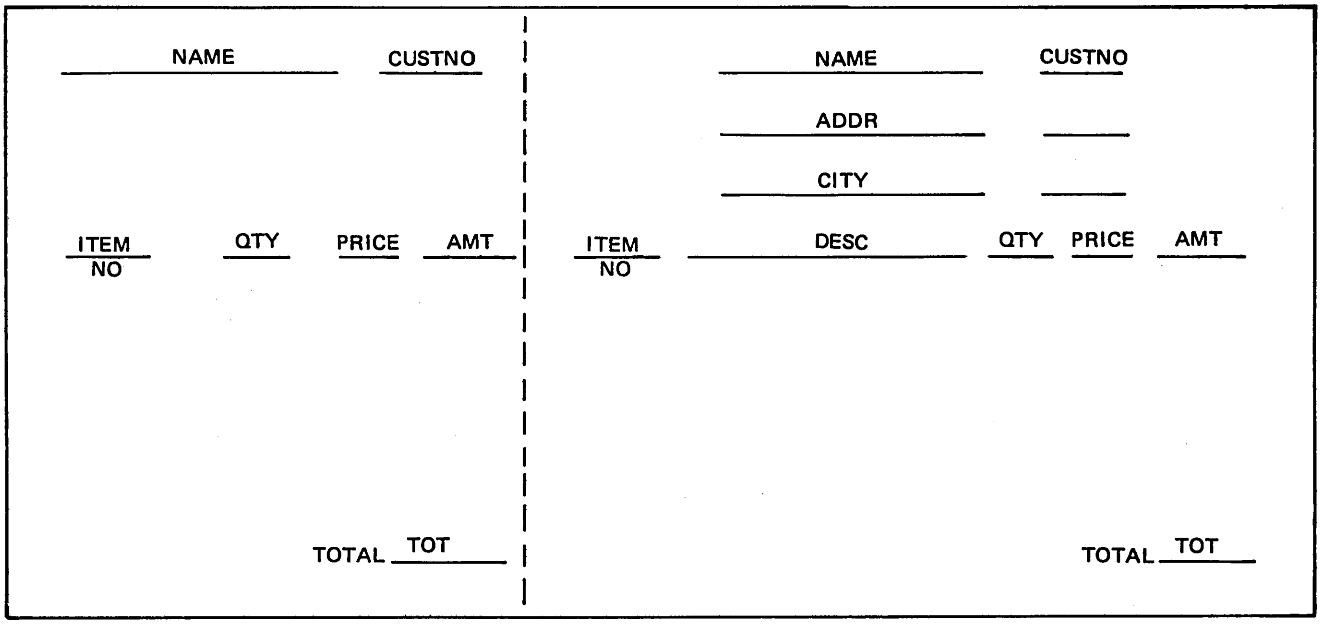


Figure 13-13. Invoice Form

Figure 13-14 shows the printer spacing chart for the invoice. What output-format specifications would you write to print fields twice on the same line? You could define the field and give the end position for it each time you wanted to print the field. Figure 13-15 shows the coding necessary using this method. There is an easier way to do this, however. This is through the use of *PLACE.

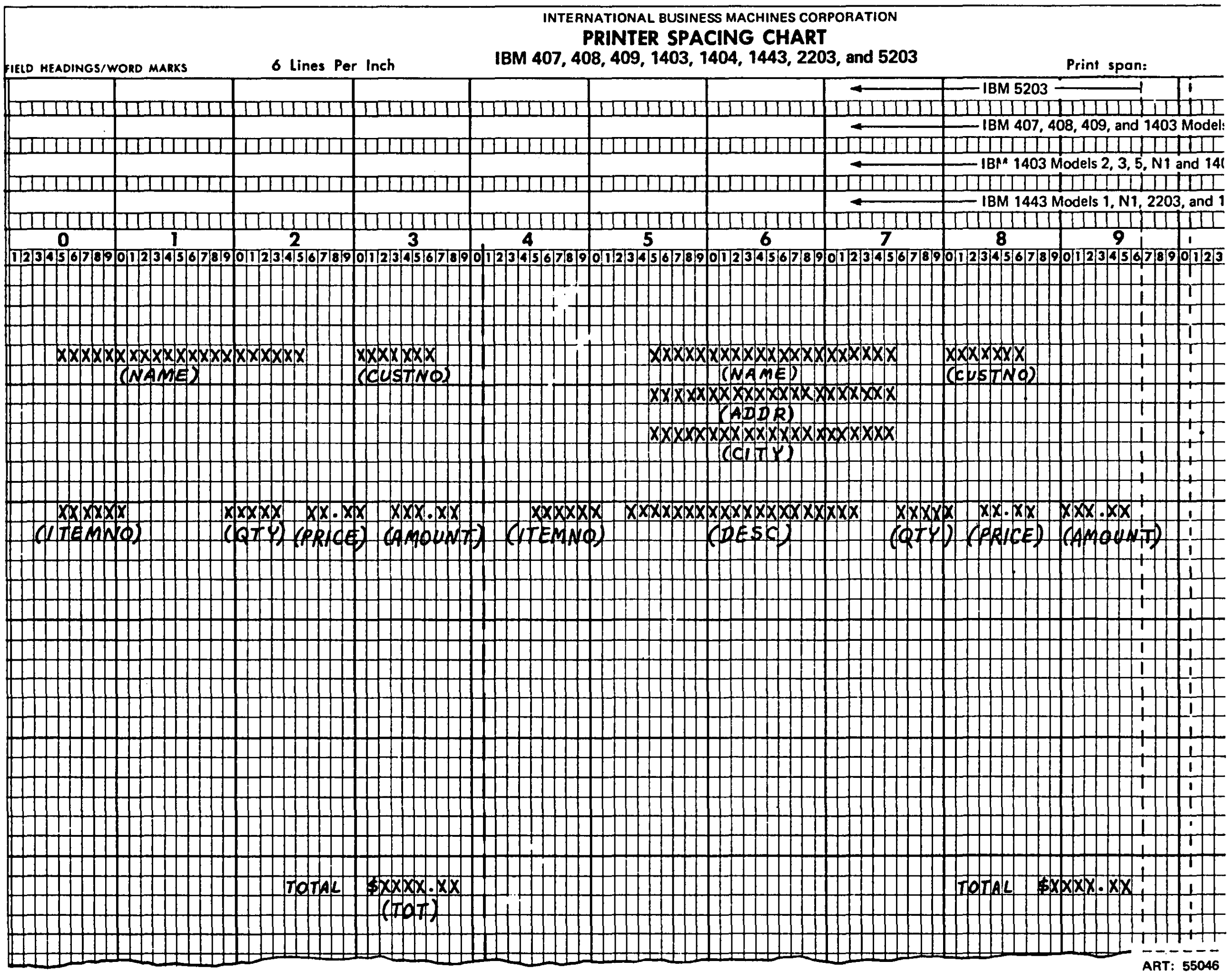
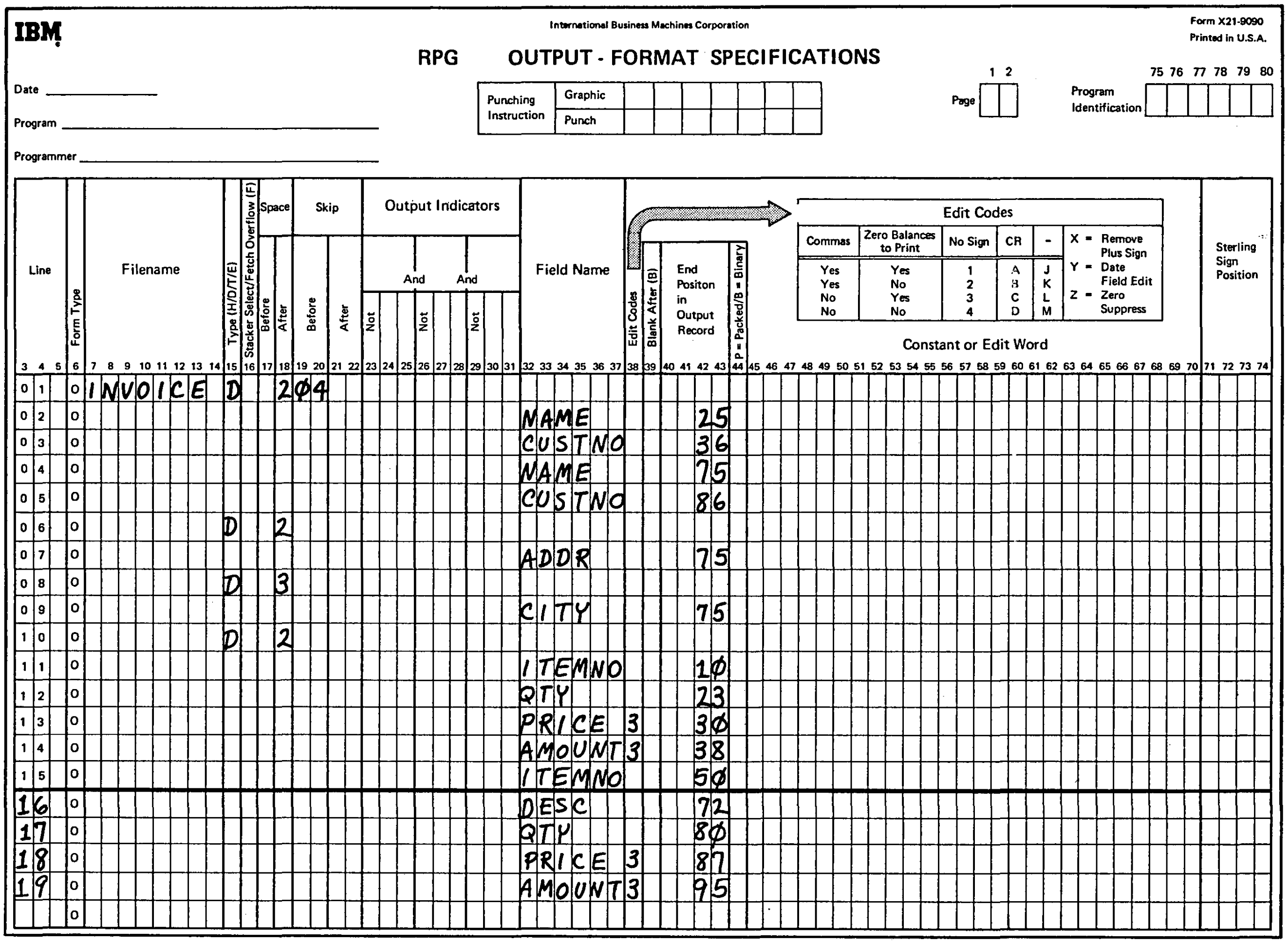

INTERNATIONAL BUSINESS MACHINES CORPORATION
PRINTER SPACING CHART
IBM 407, 408, 409, 1403, 1404, 1443, 2203, and 5203
FIELD HEADINGS/WORD MARKS
6 Lines Per Inch
Print span:
IBM 5203
IBM 407, 408, 409, and 1403 Model
IBM 1403 Models 2, 3, 5, N1 and 14
IBM 1443 Models 1, N1, 2203, and 1
(NAME)
(CUSTNO)
(NAME)
(CUSTNO)
(ADDR)
(CITY)
(ITEMNO)
(QTY)
(PRICE)
(AMOUNT)
(ITEMNO)
(DESC)
(QTY)
(PRICE)
(AMOUNT)
XX.XX
XXX.XX
TOTAL
$XXXX.XX
(TOT)
TOTAL
$XXXX.XX
ART: 55046

Figure 13-14. Printer Spacing Chart for Invoice



IBM

International Business Machines Corporation

Form X21-9090
Printed in U.S.A.

RPG OUTPUT - FORMAT SPECIFICATIONS

Date ____________

Program ____________

Programmer ____________

Punching Instruction: Graphic / Punch

Page 1 2

Program Identification 75 76 77 78 79 80

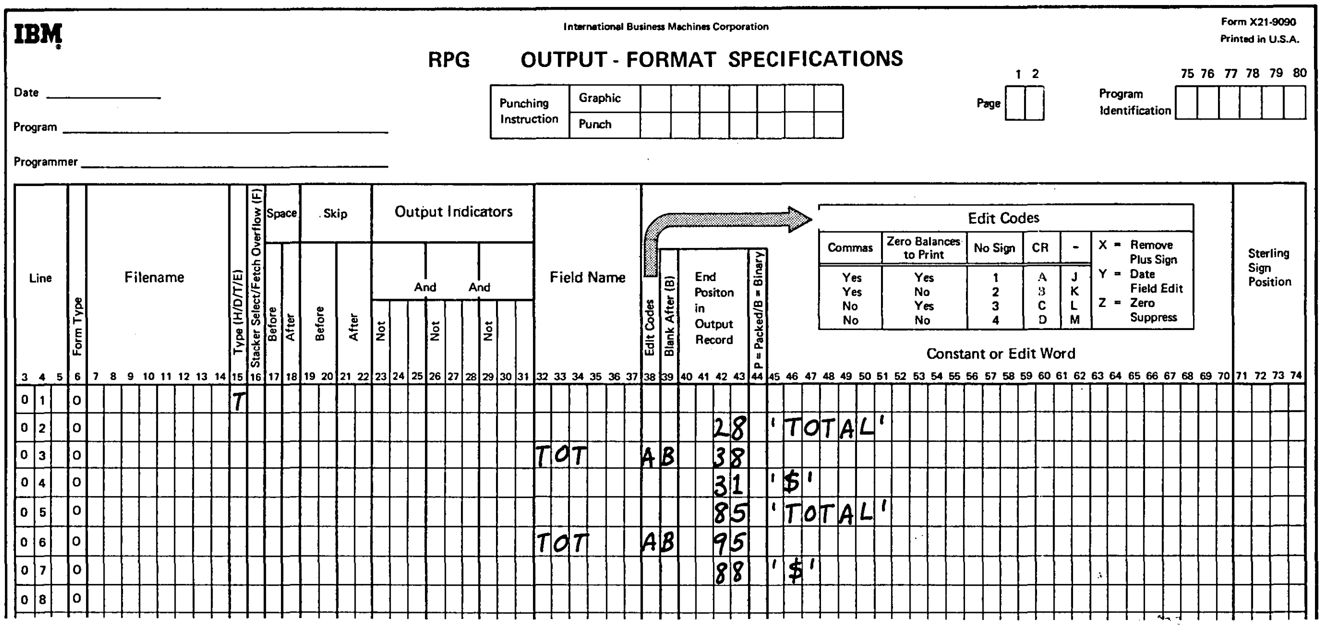
Edit Codes

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Sterling Sign Position

Line (3-5)	Form Type (6)	Filename (7-14)	Type (H/D/T/E) (15)	Stacker Select/Fetch Overflow (F) (16)	Space Before (17)	Space After (18)	Skip Before (19-20)	Skip After (21-22)	Output Indicators (23-31)	Field Name (32-37)	Edit Codes (38)	Blank After (B) (39)	End Position in Output Record (40-43)	P = Packed/B = Binary (44)	Constant or Edit Word (45-70)	Sterling Sign Position (71-74)
01	O	INVOICE	D			2	Ø4									
02	O									NAME			25			
03	O									CUSTNO			36			
04	O									NAME			75			
05	O									CUSTNO			86			
06	O		D			2										
07	O									ADDR			75			
08	O		D			3										
09	O									CITY			75			
10	O		D			2										
11	O									ITEMNO			1Ø			
12	O									QTY			23			
13	O									PRICE	3		3Ø			
14	O									AMOUNT	3		38			
15	O									ITEMNO			5Ø			
16	O									DESC			72			
17	O									QTY			8Ø			
18	O									PRICE	3		87			
19	O									AMOUNT	3		95			
	O															



IBM

International Business Machines Corporation

Form X21-9090
Printed in U.S.A.

RPG OUTPUT - FORMAT SPECIFICATIONS

Date ____________

Program ____________

Programmer ____________

Punching Instruction: Graphic / Punch

Page 1 2

Program Identification 75 76 77 78 79 80

Edit Codes

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Sterling Sign Position

Line (3-5)	Form Type (6)	Filename (7-14)	Type (H/D/T/E) (15)	Stacker Select/Fetch Overflow (F) (16)	Space Before (17)	Space After (18)	Skip Before (19-20)	Skip After (21-22)	Output Indicators (23-31)	Field Name (32-37)	Edit Codes (38)	Blank After (B) (39)	End Position in Output Record (40-43)	P = Packed/B = Binary (44)	Constant or Edit Word (45-70)	Sterling Sign Position (71-74)
01	O		T													
02	O												28		'TOTAL'	
03	O									TOT	A	B	38			
04	O												31		'$'	
05	O												85		'TOTAL'	
06	O									TOT	A	B	95			
07	O												88		'$'	
08	O															

Figure 13-15. Output-Format Specifications for Invoice (Coding Each Field Twice)

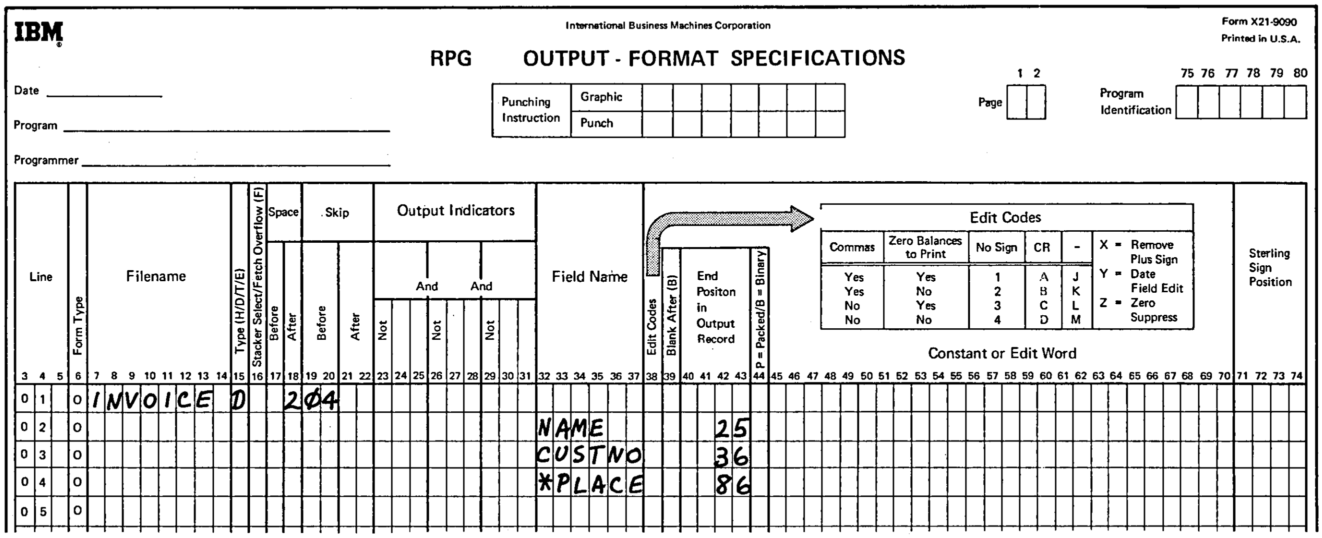
Specifications for Using *PLACE

*PLACE is a special RPG II function which can be used to accomplish duplicate printing with less coding. To the RPG II compiler the specification *PLACE means: Duplicate that part of the line which has been specified and place the duplicated information in a different position on the same line. *PLACE means a special function is to be performed. You should not use this specification as a field name, since the RPG II compiler will assume you want the preceding field duplicated. When using *PLACE you first define, for each record, all the fields which are to be duplicated. Give the end position for each field as you normally do. Then enter the word *PLACE on the line below the fields which are to be duplicated. Figure 13-16 shows the entries for the first detail line of the invoice.

The compiler does not know where to print unless you specify an end position on the *PLACE entry. In Figure 13-16, the end position given for the *PLACE entry was 86.

The *PLACE specification duplicates not only letters but also blank spaces. It will duplicate all the characters (including blanks) from position 1 to the end position specified for a field. These duplicated characters are then placed so that they end in the end position specified for the *PLACE entry.

When specifying an end position for the *PLACE entry, you must know exactly where you wish the fields to print. You must also consider the amount of space needed for the printing of *all* characters to be duplicated. Always specify an end position which allows room for the printing of duplicated fields.



IBM

International Business Machines Corporation

Form X21-9090
Printed in U.S.A.

RPG OUTPUT - FORMAT SPECIFICATIONS

Date ____________

Program ____________

Programmer ____________

Punching Instruction	Graphic							
	Punch							

Page (1 2) □□

Program Identification (75 76 77 78 79 80) □□□□□□

Edit Codes

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Line (3-5)	Form Type (6)	Filename (7-14)	Type (H/D/T/E) (15)	Stacker Select/Fetch Overflow (F) (16)	Space Before (17)	Space After (18)	Skip Before (19-20)	Skip After (21-22)	Output Indicators: Not (23)	(24-25)	And Not (26)	(27-28)	And Not (29)	(30-31)	Field Name (32-37)	Edit Codes (38)	Blank After (B) (39)	End Positon in Output Record (40-43)	P = Packed/B = Binary (44)	Constant or Edit Word (45-70)	Sterling Sign Position (71-74)
01	O	INVOICE	D			2	Ø4														
02	O														NAME			25			
03	O														CUSTNO			36			
04	O														*PLACE			86			
05	O																				

Figure 13-16. Output-Format Specifications for First Line of Invoice (Using *PLACE to Print Fields Twice)

Formation of Print Lines

When System/3 performs printer output, a whole line is printed at once, regardless of how many fields are in that line. Before printing, the whole line is moved to an area of storage exactly as it is to be printed. Data is placed in this storage area one field at a time.

The sequence in which data enters the storage area depends on the sequence that field names are specified on the RPG II Output-Format Sheet. The first field recorded on the Output-Format Sheet is entered first, then the second, etc. Each field is inserted into the storage area according to its end-position entry on the Output-Format Sheet. If you have made conflicting entries in your specifications (for example, one field overlapping another) the last field mentioned is the one that will print in its entirety.

*PLACE operates in the same way as normal field names. The operations associated with *PLACE are performed in the sequence *PLACE is specified on the Output-Format Sheet in relation to other output entries.

Follow the formation of the first line to be printed on the invoice. According to the specifications in Figure 13-16, the NAME field ends in position 25 and CUSTNO in 36. The first part of the line is completed with these specifications (Figure 13-17, insert A). Because of the way lines are formed, the end position for the *PLACE entry must be at least two times the higher end position specified for a field that is to be duplicated. This ensures that the last field mentioned will not overlap the field preceding. In this case the same fields are to be printed again on the second part of the same line. Since the end position was 36, the second part of the same line must end at least in position 72 (two times higher than the end position for the field to be duplicated). It is decided they are to end in position 86. The second part of the line is formed by the *PLACE entry (Figure 13-17, insert B).

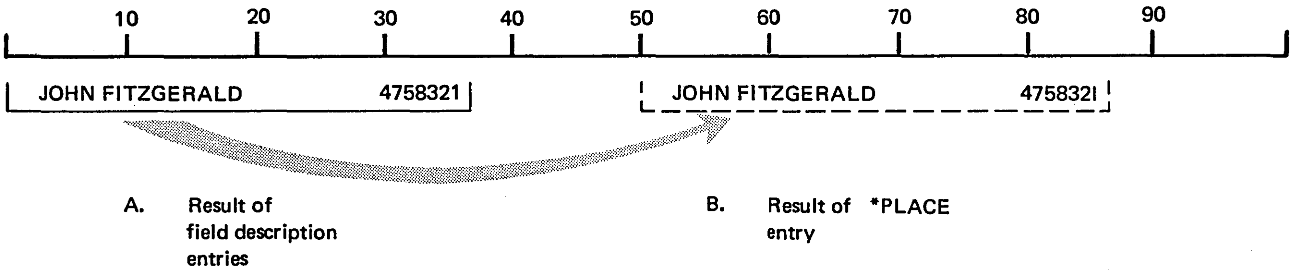

10
20
30
40
50
60
70
80
90
JOHN FITZGERALD
4758321
JOHN FITZGERALD
4758321
A. Result of field description entries
B. Result of *PLACE entry

Figure 13-17. Line Formation (First Line of Invoice)

Using Different Spacing for Duplicated Fields. The second and third lines of the invoice do not have fields to be duplicated. However, the fourth line of the invoice requires that all fields be duplicated. Notice that different spacing is required for the duplicated fields because a field called DESC must be inserted between ITEMNO and QTY.

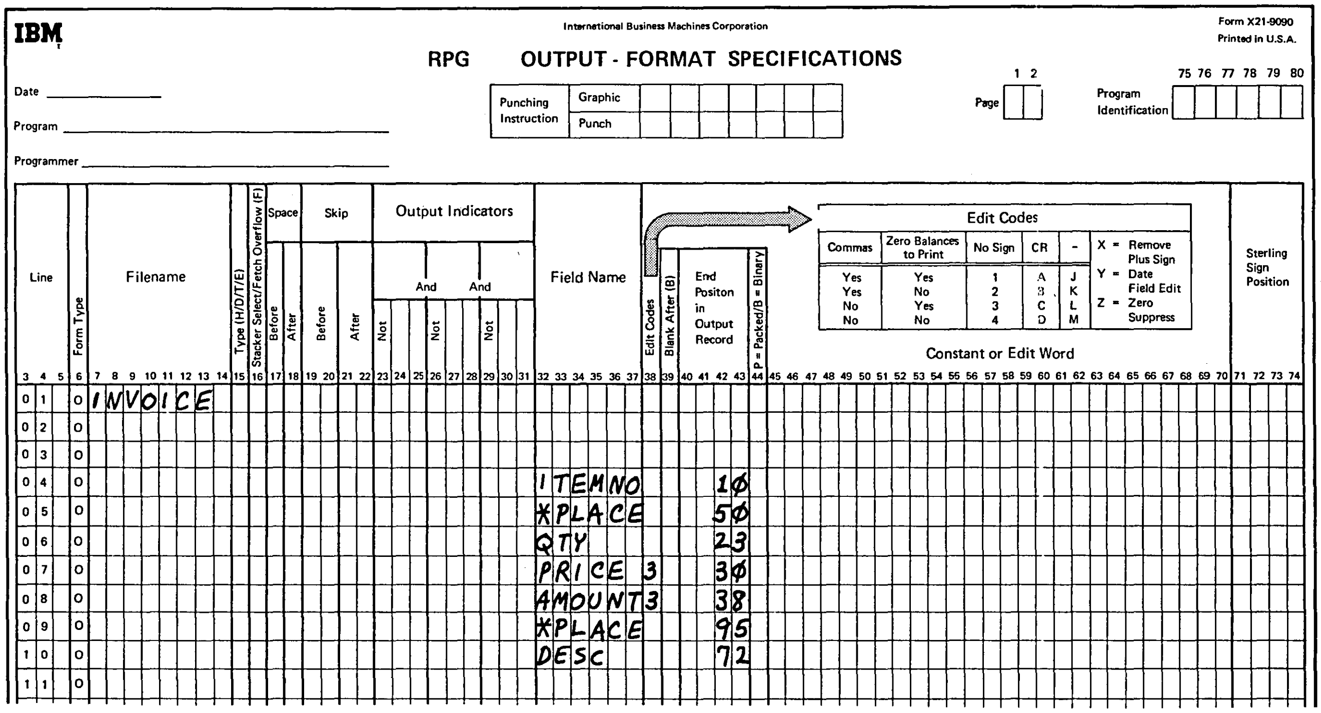
Figure 13-18 shows correct specifications. You want to start with ITEMNO since it is the first field. ITEMNO is specified as usual; the end position is given. Then *PLACE is specified with the correct end position, 50 in this case. These specifications cause the line to look like that in Figure 13-19, insert A.

Now, the remaining three fields are specified and an end position is given for each. *PLACE is entered after them to signify that the above three fields should be duplicated. Remember that when fields are duplicated, all information from position 1 to the highest end position specified for a field is used. In this case, positions 1 through 38 are duplicated and placed so that they end in position 95.

QTY, PRICE, and AMOUNT are in positions 1 through 38, but ITEMNO is also there since it ends in position 10. Thus, all four fields are duplicated and placed so that they end in 96. Figure 13-19, insert B shows resulting formation of the line. ITEMNO now appears three times, once in the DESC field area where it should not be.

In this example, we can specify the field DESC to end in position 75. It will overlay the unwanted ITEMNO field and thus get rid of it. Figure 13-19, insert C shows the line as it will be printed.

For each job you do using *PLACE, you will have to calculate exactly what happens when lines are formed.



IBM

International Business Machines Corporation

Form X21-9090
Printed in U.S.A.

RPG OUTPUT - FORMAT SPECIFICATIONS

Date ____________

Program ____________

Programmer ____________

Punching Instruction	Graphic							
	Punch							

Page 1 2 □□

Program Identification 75 76 77 78 79 80 □□□□□□

Edit Codes

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Line (3-5)	Form Type (6)	Filename (7-14)	Type (H/D/T/E) (15)	Stacker Select/Fetch Overflow (F) (16)	Space Before (17)	Space After (18)	Skip Before (19-20)	Skip After (21-22)	Output Indicators (23-31)	Field Name (32-37)	Edit Codes (38)	Blank After (B) (39)	End Positon in Output Record (40-43)	P = Packed/B = Binary (44)	Constant or Edit Word (45-70)	Sterling Sign Position (71-74)
01	O	INVOICE														
02	O															
03	O															
04	O									ITEMNO			1Ø			
05	O									*PLACE			5Ø			
06	O									QTY			23			
07	O									PRICE	3		3Ø			
08	O									AMOUNT	3		38			
09	O									*PLACE			95			
10	O									DESC			72			
11	O															

Figure 13-18. Correct Specifications for Fourth Line of Invoice

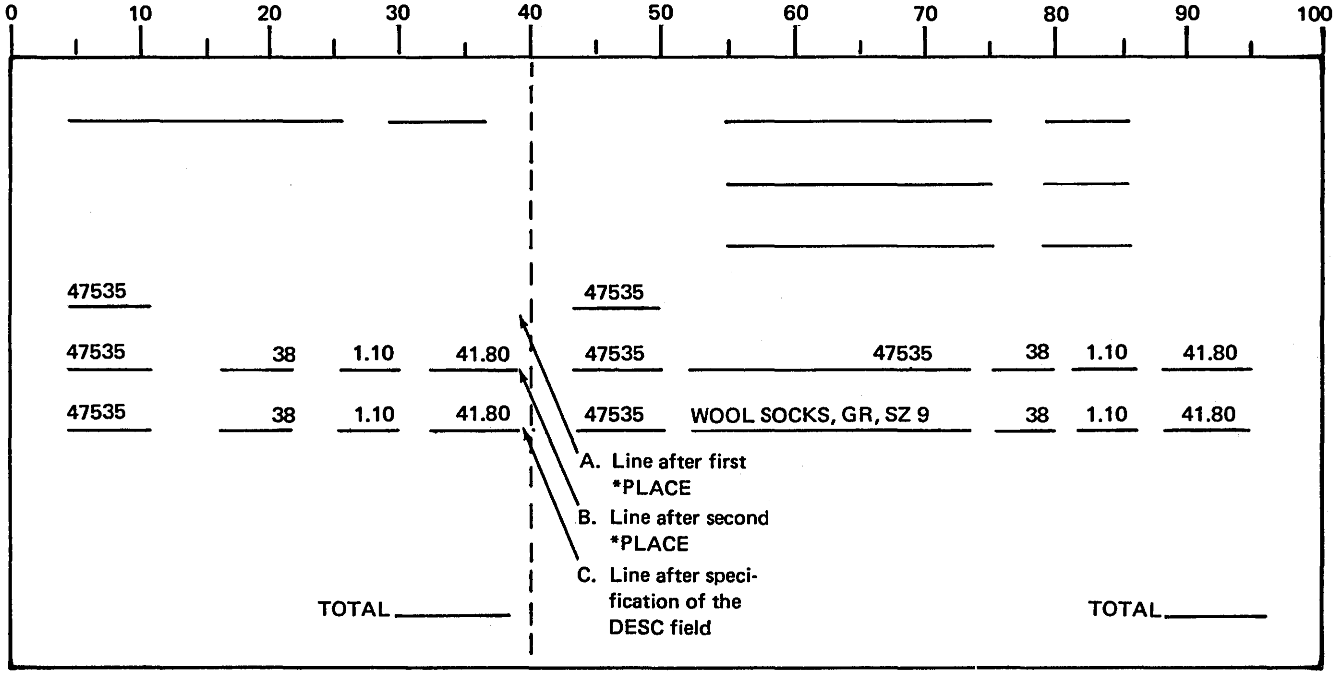
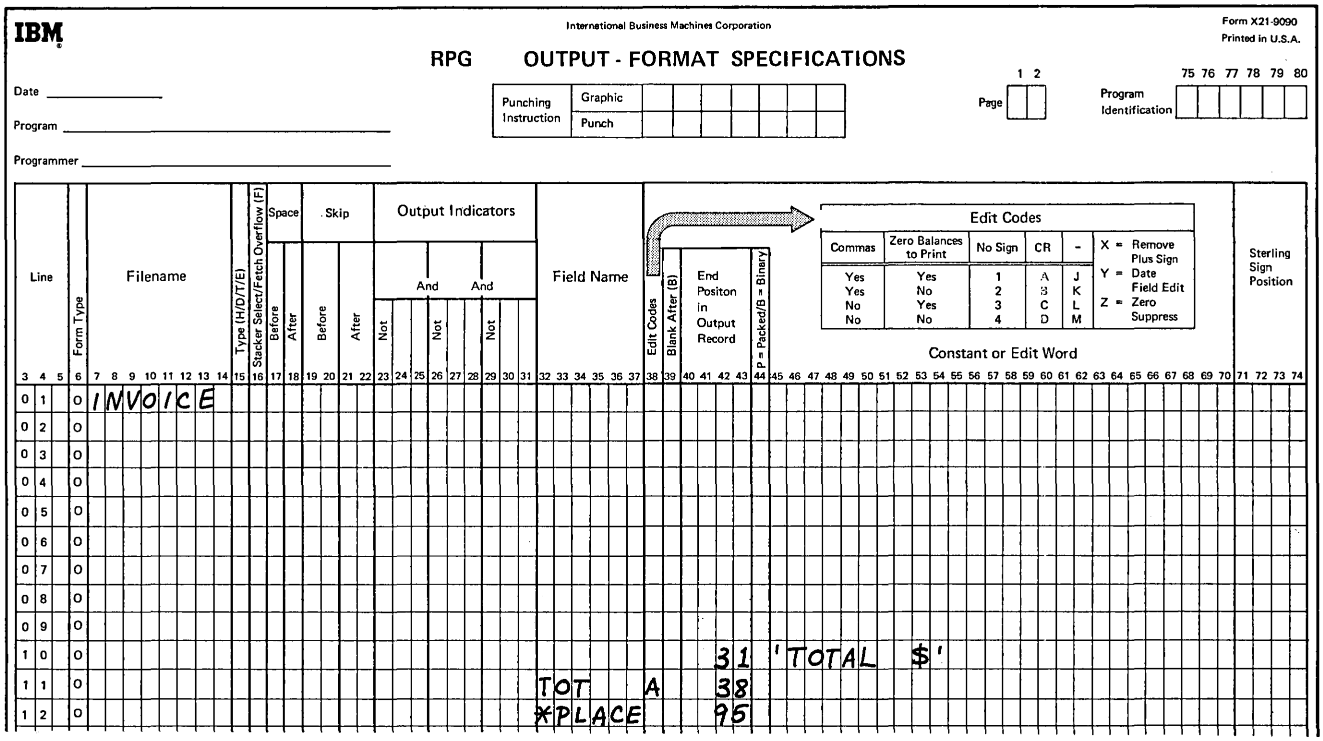


Figure 13-19. Fourth Printed Line (Using Correct Specifications)

Duplicating Constants: *PLACE can duplicate constants as well as fields. The same specifications are used for both. Figure 13-20 shows the specifications for the last line of the invoice. In this case *PLACE duplicates a field and two constants. As you can see, using *PLACE eliminates duplicate coding.

**PLACE Used With Other Entries:* For the *PLACE entry, columns 7-22 and 44-74 must be blank. However, *PLACE may be conditioned by any indicators specified in columns 23-31. When indicators are used, fields will be duplicated only when the condition set by the indicators is met. For example, the specification in line 3 of Figure 13-21 will be done only when matching records are found.



IBM — International Business Machines Corporation — Form X21-9090, Printed in U.S.A.

RPG OUTPUT - FORMAT SPECIFICATIONS

Date
Program
Programmer

Punching Instruction: Graphic / Punch — Page 1 2 — Program Identification 75 76 77 78 79 80

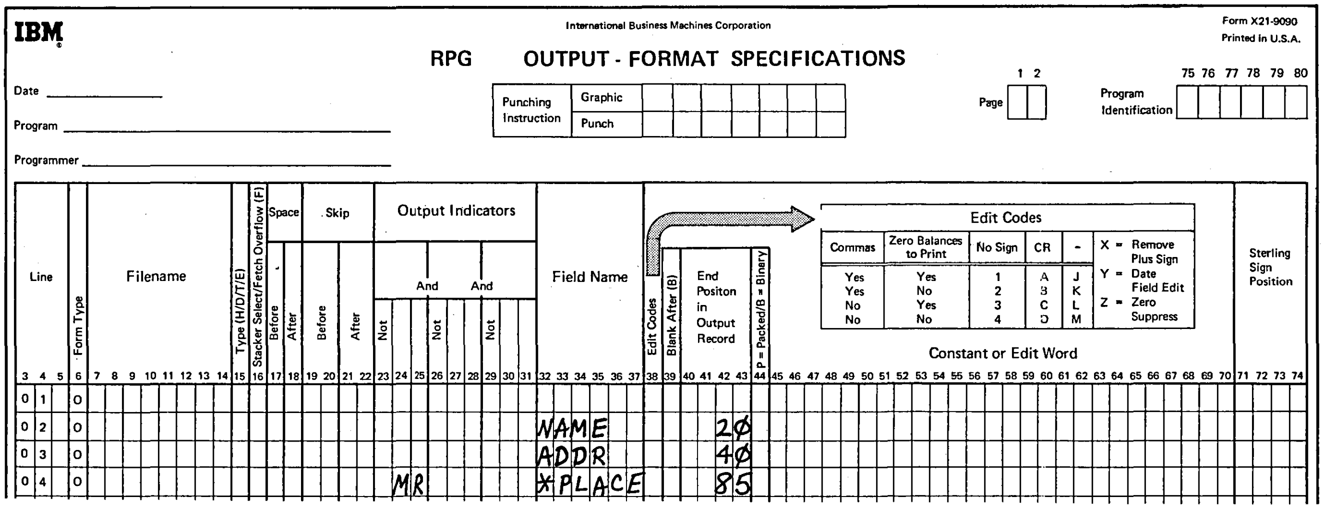
Edit Codes

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Line (3-5)	Form Type (6)	Filename (7-14)	Output Indicators (23-31)	Field Name (32-37)	Edit Codes (38)	End Position in Output Record (40-43)	Constant or Edit Word (45-70)
01	O	INVOICE					
02	O						
03	O						
04	O						
05	O						
06	O						
07	O						
08	O						
09	O						
10	O					31	'TOTAL $'
11	O			TOT	A	38	
12	O			*PLACE		95	

Figure 13-20. Using *PLACE to Duplicate Constants



IBM — International Business Machines Corporation — Form X21-9090, Printed in U.S.A.

RPG OUTPUT - FORMAT SPECIFICATIONS

Date
Program
Programmer

Punching Instruction: Graphic / Punch — Page 1 2 — Program Identification 75 76 77 78 79 80

Line (3-5)	Form Type (6)	Filename (7-14)	Output Indicators (23-31)	Field Name (32-37)	Edit Codes (38)	End Position in Output Record (40-43)	Constant or Edit Word (45-70)
01	O						
02	O			NAME		2Ø	
03	O			ADDR		4Ø	
04	O		MR	*PLACE		85	

Figure 13-21. Conditioning the *PLACE Entry

Printing a Field Several Times on the Same Line

*PLACE can be used to print the same field several times in the line. All you have to do is enter *PLACE along with an end position for each time you want the fields duplicated. If you want the field duplicated twice, you need two *PLACE entries.

Assume that periodically a store prepares mailing labels for each customer who has an account with them. They use the labels when they send out special advertisements. The mailing label has only name, address, and zip code on it. Since the label has to be only a few inches wide, the manager found he could print three labels side by side on his 120-print position printer (Figure 13-22).

You can see that each field needs to be printed three times on each line. In the examples discussed so far, *PLACE was used to duplicate fields only once.

Figure 13-23 shows the specifications for the first line. NAME needs to be entered three times per line. The original field specification prints it one time: the two *PLACE entries cause it to be printed two more times.

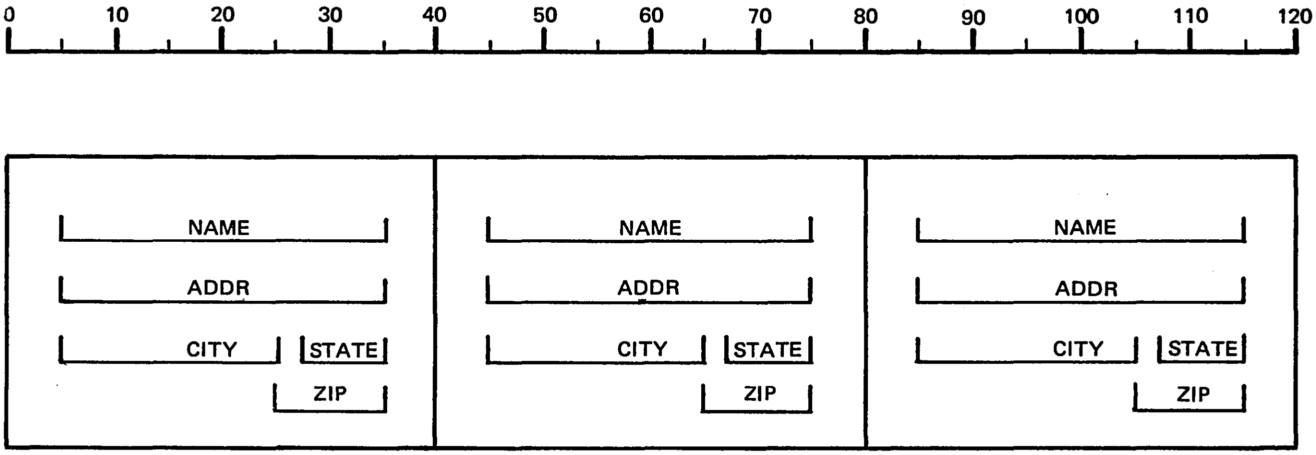
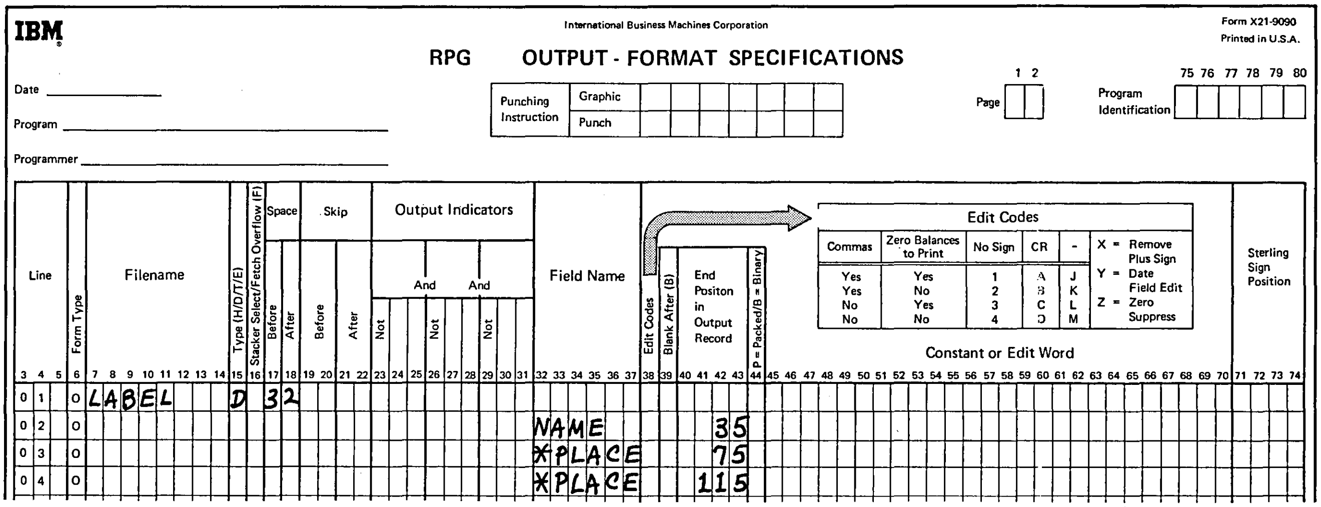


Figure 13-22. Mailing Labels



IBM

International Business Machines Corporation

Form X21-9090
Printed in U.S.A.

RPG OUTPUT - FORMAT SPECIFICATIONS

Date

Program

Programmer

Punching Instruction	Graphic								
	Punch								

Page (1 2)

Program Identification (75 76 77 78 79 80)

Edit Codes

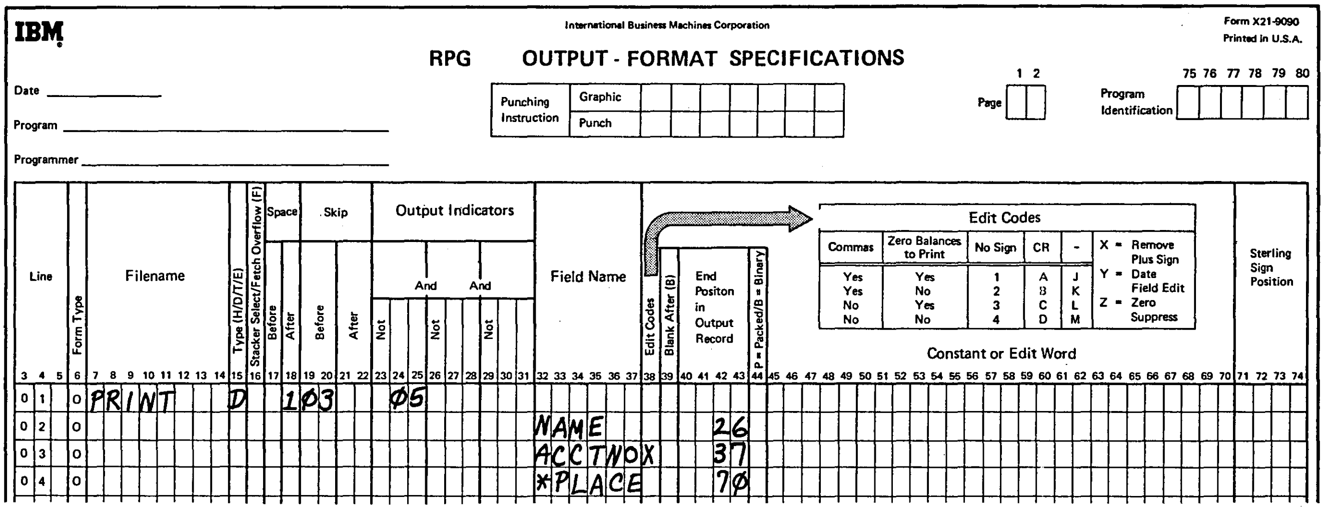
Commas	Zero Balances to Print	No Sign	CR	-	
Yes	Yes	1	A	J	X = Remove Plus Sign
Yes	No	2	B	K	Y = Date Field Edit
No	Yes	3	C	L	Z = Zero Suppress
No	No	4	D	M	

Line (3-5)	Form Type (6)	Filename (7-14)	Type (H/D/T/E) (15)	Stacker Select/Fetch Overflow (F) (16)	Space Before (17)	Space After (18)	Skip Before (19-20)	Skip After (21-22)	Output Indicators (23-31)	Field Name (32-37)	Edit Codes (38)	Blank After (B) (39)	End Position in Output Record (40-43)	P = Packed/B = Binary (44)	Constant or Edit Word (45-70)	Sterling Sign Position (71-74)
01	O	LABEL	D		3	2										
02	O									NAME			35			
03	O									*PLACE			75			
04	O									*PLACE			115			

Figure 13-23. Using *PLACE for Producing Mailing Labels

1. What is the function of *PLACE?

2. In the example shown, is *PLACE used correctly? If not, why not?



IBM

International Business Machines Corporation

Form X21-9090
Printed in U.S.A.

RPG OUTPUT - FORMAT SPECIFICATIONS

Date
Program
Programmer

Punching Instruction: Graphic / Punch

Page 1 2

Program Identification 75 76 77 78 79 80

Edit Codes					
Commas	Zero Balances to Print	No Sign	CR	-	
Yes	Yes	1	A	J	X = Remove Plus Sign
Yes	No	2	B	K	Y = Date Field Edit
No	Yes	3	C	L	Z = Zero Suppress
No	No	4	D	M	

Line	Form Type	Filename	Type (H/D/T/E)	Space After	Skip Before	Output Indicators	Field Name	Edit Codes	End Position in Output Record
01	O	PRINT	D	1	Ø3	Ø5			
02	O						NAME		26
03	O						ACCTNO	X	37
04	O						*PLACE		7Ø

3. Write the output specifications to print three mailing labels in a row using *PLACE. The first label ends in print position 25. The second in 60, the third in 95. Each mailing label will have three lines and look like this:

NAME (25 characters)

ADDR (25)

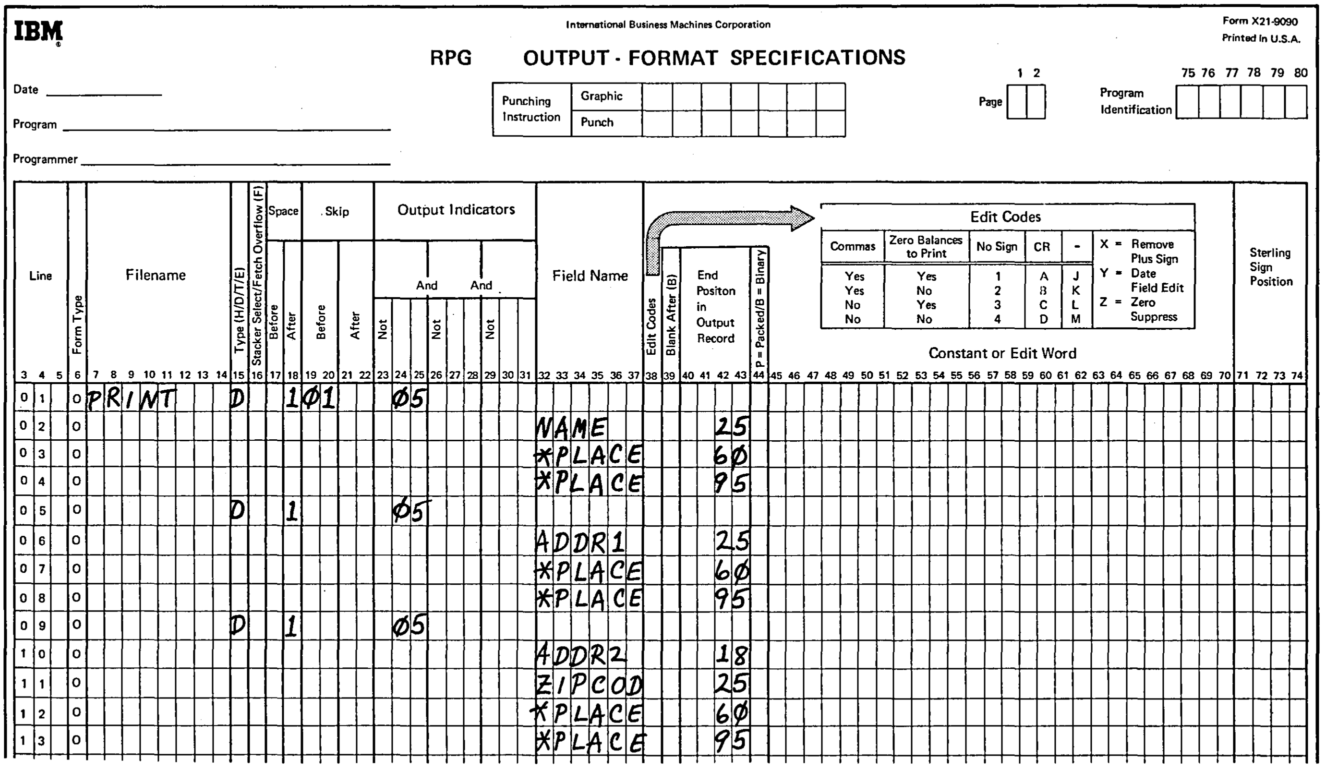
ADDR (18) ZIPCODE (5)

Answers to Review 13. *PLACE

1. The function of *PLACE is to easily code the printing of duplicate information on the same output line. *PLACE places information from print position 1, through the highest end position previously defined for a field into the print positions indicated by the end position in the *PLACE entry.

2. It is not correct. The end position in the *PLACE specification is not high enough. The duplicated information will overlay the field called ACCTNO. The end position on the *PLACE line should be at least twice the highest end position previously specified for that record.

3. Three labels must be printed. Therefore, for each line you must specify the original field and two *PLACE entries which will cause the contents of the original field to be duplicated twice.



IBM

International Business Machines Corporation

Form X21-9090
Printed in U.S.A.

RPG OUTPUT - FORMAT SPECIFICATIONS

Date

Program

Programmer

Punching Instruction	Graphic	
	Punch	

Page (1 2)

Program Identification (75 76 77 78 79 80)

Commas	Zero Balances to Print	No Sign	CR	-	
Yes	Yes	1	A	J	X = Remove Plus Sign
Yes	No	2	B	K	Y = Date Field Edit
No	Yes	3	C	L	Z = Zero Suppress
No	No	4	D	M	

Line (3-5)	Form Type (6)	Filename (7-14)	Type (H/D/T/E) (15)	Stacker Select/Fetch Overflow (F) (16)	Space Before (17)	Space After (18)	Skip Before (19-20)	Skip After (21-22)	Output Indicators (23-31)	Field Name (32-37)	Edit Codes (38)	Blank After (B) (39)	End Position in Output Record (40-43)	P = Packed/B = Binary (44)	Constant or Edit Word (45-70)	Sterling Sign Position (71-74)
01	O	PRINT	D			1	Ø1		Ø5							
02	O									NAME			25			
03	O									*PLACE			6Ø			
04	O									*PLACE			95			
05	O		D			1			Ø5							
06	O									ADDR1			25			
07	O									*PLACE			6Ø			
08	O									*PLACE			95			
09	O		D			1			Ø5							
10	O									ADDR2			18			
11	O									ZIPCOD			25			
12	O									*PLACE			6Ø			
13	O									*PLACE			95			

USING THE DUAL FEED CARRIAGE FEATURE TO PRINT TWO OUTPUT FILES FOR ONE PROGRAM

The dual feed carriage feature of the 5203 Printer (Figure 13-24) allows you to produce two separate printer output files for one program. Note that two forms are placed on the same printer. The forms used for the two output files are special forms for your printer such as checks or invoices which are narrower than the standard forms. Each form is held in place by its own set of carriage tractors. One form is controlled by the left carriage of the printer and the other form is controlled by the right carriage. Note the space between the right and left carriage tractors that contains no form. When you are printing on two forms you will lose 17 print positions since no forms can be place in this space.

To print two output files for one program, each of the two printer files are considered separate output files and must be described as such. These output files require special descriptions on the File Description Sheet and the Output-Format Sheet.

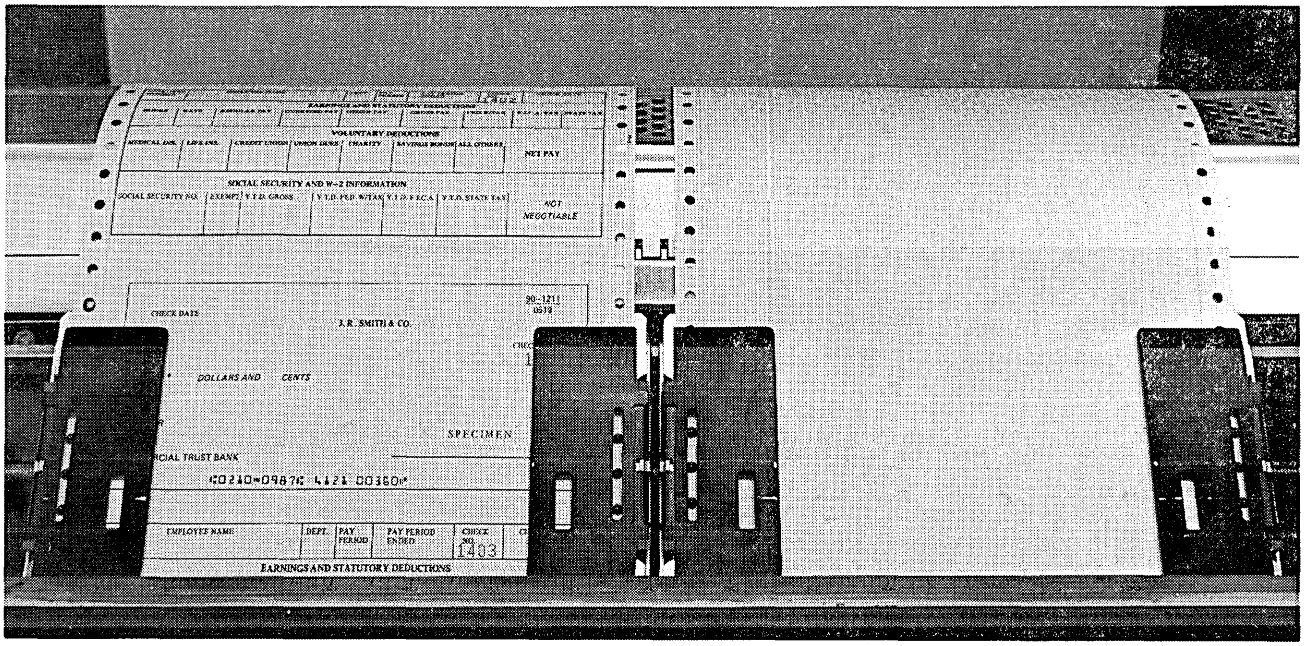


Figure 13-24. 5203 Printer with the Dual Feed Carriage Feature

53958

File Description Specifications

The two output files to be printed must be assigned to the device names PRINTER and PRINTR2 on the File Description Sheet (columns 40-46). PRINTER is the device name for the left carriage of the printer. The form controlled by the right carriage of the printer is assigned the device name PRINTR2. Figure 13-25 is a sample File Description Sheet for the dual feed carriage feature.

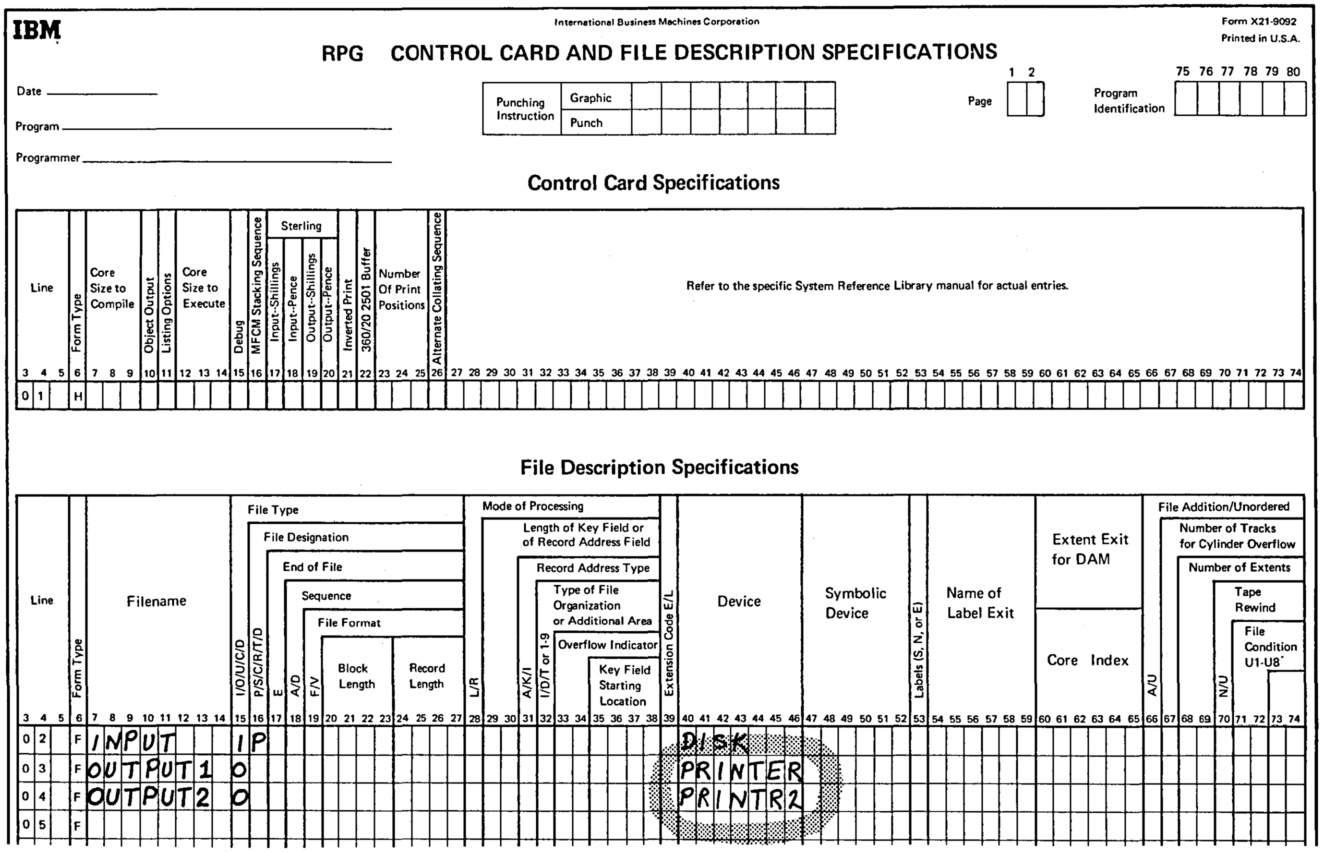
Output-Format Specifications

Spacing and skipping on the two forms are completely independent. You can specify different spacing and skipping for each output file. Spacing and skipping are entered in columns 17-22 of the Output-Format Sheet.

Remember, there are 17 print positions you cannot use, because there is a space between the left and right carriage tractors which cannot contain a form. This is important when you are planning where to position your printing on each form. The first character to be printed on the form in the right carriage must be *at least* 17 positions away from the last character on the form in the left carriage (PRINTER) to have 80 print positions. You decide to use print positions 1 through 80. Now, since the first character in the right carriage must be at least 17 positions away from the last character in the left carriage, print position 98 is the first available position:

```
 80  (End position of the form in the left carriage)
+17  (Number of print positions you cannot use)
 97
```

Therefore, 98 is the first available position. If the length of your print line will be 35 characters, the end position for the second form will be 132. Recall that you must specify the end position for each form in columns 40-43 on the Output-Format Sheet. You would specify 80 as the end position of the left form, and 132 for the right form.



IBM

International Business Machines Corporation

Form X21-9092
Printed in U.S.A.

RPG CONTROL CARD AND FILE DESCRIPTION SPECIFICATIONS

Date
Program
Programmer

Punching Instruction: Graphic / Punch

Page 1 2

Program Identification 75 76 77 78 79 80

Control Card Specifications

Line (3-5) | Form Type (6) | Core Size to Compile (7-9) | Object Output (10) | Listing Options (11) | Core Size to Execute (12-14) | Debug (15) | MFCM Stacking Sequence (16) | Sterling: Input-Shillings (17), Input-Pence (18), Output-Shillings (19), Output-Pence (20) | Inverted Print (21) | 360/20 2501 Buffer (22) | Number Of Print Positions (23-25) | Alternate Collating Sequence (26) | 27-74: Refer to the specific System Reference Library manual for actual entries.

Line	Form Type
01	H

File Description Specifications

Line (3-5) | Form Type (6) | Filename (7-14) | File Type: I/O/U/C/D (15) | File Designation: P/S/C/R/T/D (16) | End of File: E (17) | Sequence: A/D (18) | File Format: F/V (19) | Block Length (20-23) | Record Length (24-27) | Mode of Processing: L/R (28) | Length of Key Field or of Record Address Field (29-30) | Record Address Type: A/K/I (31) | Type of File Organization or Additional Area: I/D/T or 1-9 (32) | Overflow Indicator (33-34) | Key Field Starting Location (35-38) | Extension Code E/L (39) | Device (40-46) | Symbolic Device (47-52) | Labels (S, N, or E) (53) | Name of Label Exit (54-59) | Extent Exit for DAM / Core Index (60-65) | File Addition/Unordered: A/U (66) | Number of Tracks for Cylinder Overflow (67) | Number of Extents (68-69) | Tape Rewind: N/U (70) | File Condition U1-U8 (71-72) | (73-74)

Line	Form Type	Filename	File Type	File Designation	Device
02	F	INPUT	I	P	DISK
03	F	OUTPUT1	O		PRINTER
04	F	OUTPUT2	O		PRINTR2
05	F				

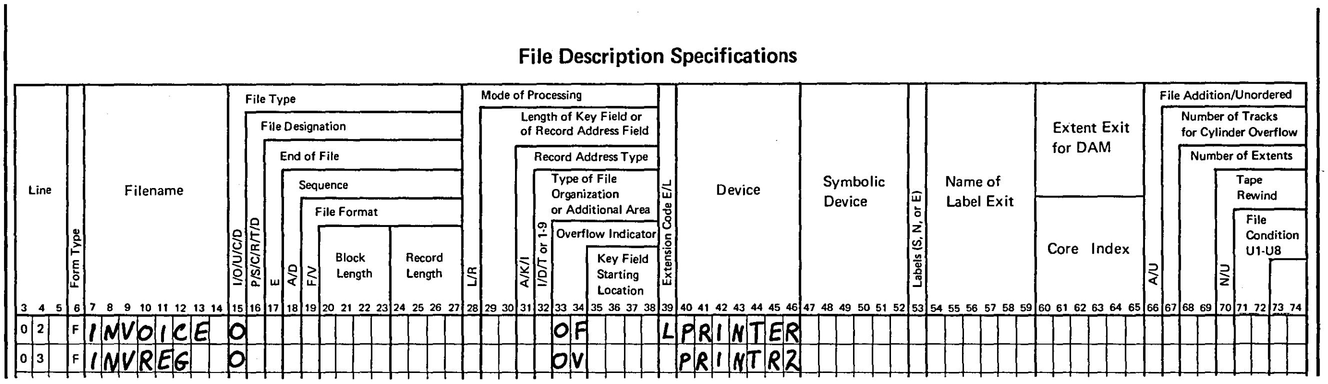
Figure 13-25. Device Names for Two Output Files Using the Dual Feed Carriage

Example: End-of-the-Month Billing

Assume that your company invoices its customers using your data processing system. It is your responsibility to prepare and print the invoices to be sent to the customers. You are also going to keep an invoice register; a record of every invoice that is set out. Since you have the dual feed carriage feature, you will print both the invoice and the invoice register at the same time.

Since you are printing an invoice and an invoice register, you name your two output files INVOICE and INVREG. Recall that the form in the left carriage is controlled by the device name PRINTER, and the form in the right carriage is controlled by the device name PRINTR2. Figure 13-26 shows a sample File Description Sheet.

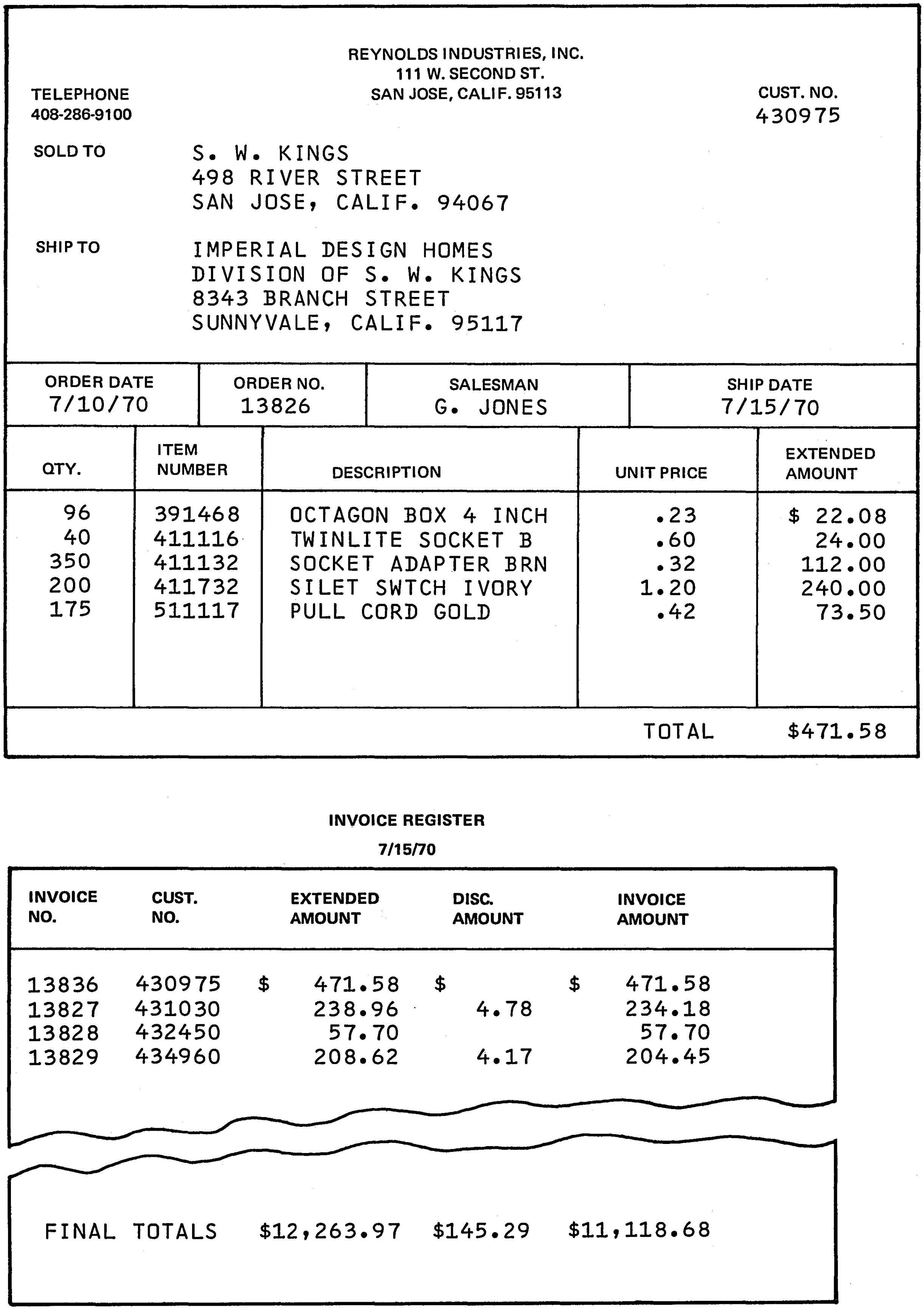
Next, the format of your output must be determined. In this case, INVREG will have the standard length of 66 lines, while INVOICE will have a non-standard form length of 50. Headings must be printed on the top of each page. INVOICE has a 63 print position line and INVREG has a 50 print position line. Figure 13-27, insert A is a sample invoice and Figure 13-27, insert B is a sample invoice register.



File Description Specifications

Line (3-5)	Form Type (6)	Filename (7-14)	I/O/U/C/D (15)	P/S/C/R/T/D (16)	E (17)	A/D (18)	F/V (19)	Block Length (20-23)	Record Length (24-27)	L/R (28)	Length of Key Field or of Record Address Field (29-30)	A/K/I (31)	I/D/T or 1-9 (32)	Overflow Indicator (33-34)	Key Field Starting Location (35-38)	Extension Code E/L (39)	Device (40-46)	Symbolic Device (47-52)	Labels (S, N, or E) (53)	Name of Label Exit (54-59)	Extent Exit for DAM / Core Index (60-65)	A/U (66)	Number of Tracks for Cylinder Overflow (67-69)	N/U (70)	File Condition U1-U8 (71-72)	(73-74)
02	F	INVOICE	O											OF		L	PRINTER									
03	F	INVREG	O											OV			PRINTR2									

Figure 13-26. Assigning Device Names to INVOICE and INVREG



REYNOLDS INDUSTRIES, INC.
111 W. SECOND ST.
SAN JOSE, CALIF. 95113

TELEPHONE
408-286-9100

CUST. NO.
430975

SOLD TO

S. W. KINGS
498 RIVER STREET
SAN JOSE, CALIF. 94067

SHIP TO

IMPERIAL DESIGN HOMES
DIVISION OF S. W. KINGS
8343 BRANCH STREET
SUNNYVALE, CALIF. 95117

ORDER DATE	ORDER NO.	SALESMAN	SHIP DATE
7/10/70	13826	G. JONES	7/15/70

QTY.	ITEM NUMBER	DESCRIPTION	UNIT PRICE	EXTENDED AMOUNT
96	391468	OCTAGON BOX 4 INCH	.23	$ 22.08
40	411116	TWINLITE SOCKET B	.60	24.00
350	411132	SOCKET ADAPTER BRN	.32	112.00
200	411732	SILET SWTCH IVORY	1.20	240.00
175	511117	PULL CORD GOLD	.42	73.50
			TOTAL	$471.58

INVOICE REGISTER
7/15/70

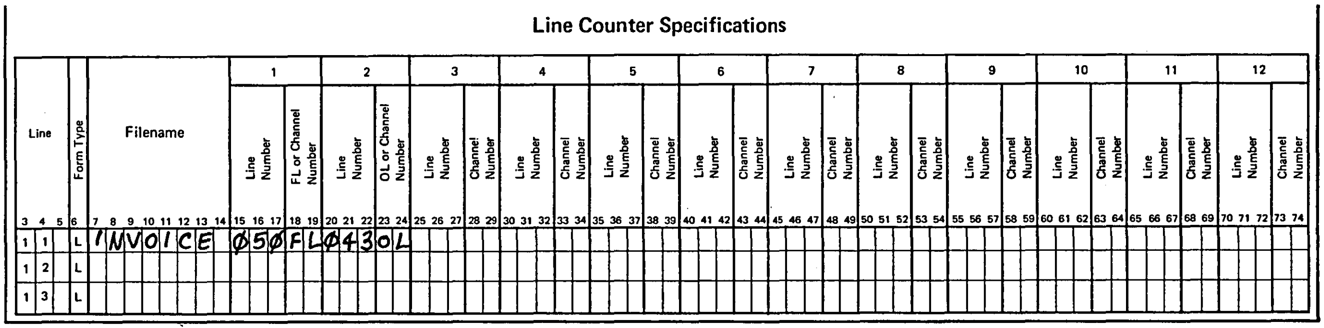
INVOICE NO.	CUST. NO.	EXTENDED AMOUNT	DISC. AMOUNT	INVOICE AMOUNT
13836	430975	$ 471.58	$	$ 471.58
13827	431030	238.96	4.78	234.18
13828	432450	57.70		57.70
13829	434960	208.62	4.17	204.45
FINAL	TOTALS	$12,263.97	$145.29	$11,118.68

Figure 13-27. Sample Invoice and Invoice Register

Since INVOICE has a non-standard form length, it must be defined on the Line Counter Sheet. You will use line 43 as the overflow line. Figure 13-28 shows a sample Line Counter Sheet. (Note that since INVREG has a standard form length, it does not have to be defined on the Line Counter Sheet.)

You also want to have headings printed on the top of every page. Because you do these operations only when an overflow indicator is on, you have to condition these operations by the overflow indicator. Figure 13-29 is the File Description Sheet with overflow indicators. Figure 13-30 shows the Output-Format Sheet to print headings at the top of every page. (Remember that skipping and spacing on the two carriages are independent.)

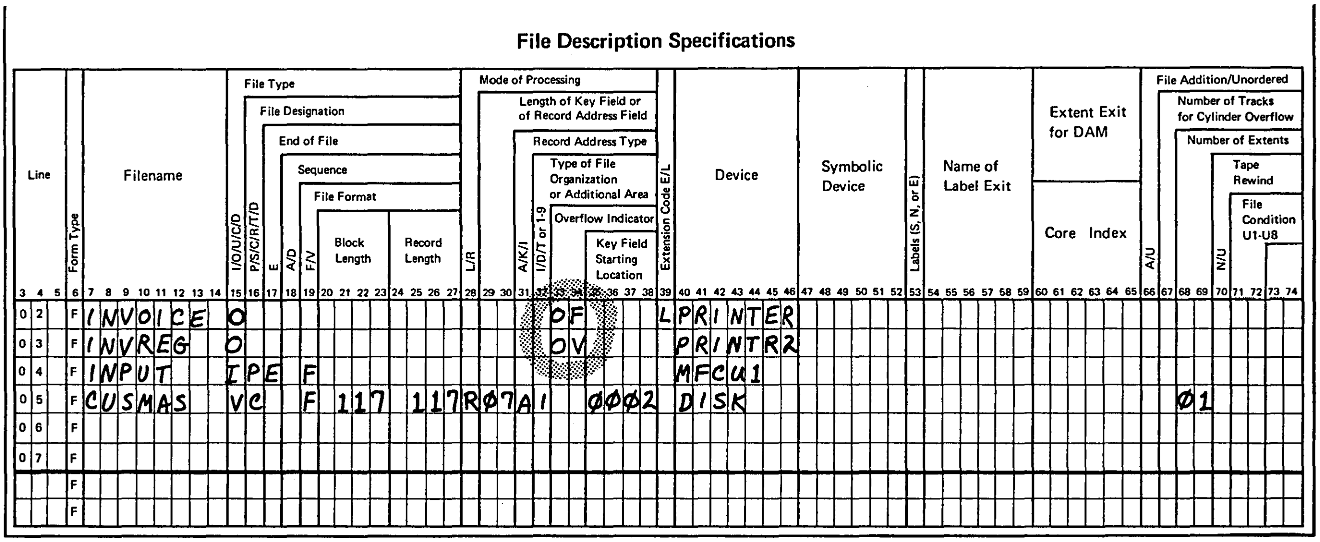
Also, recall that the form in the right carriage (in this case INVREG) must be at least 17 positions away from the form in the left carriage. Since INVOICE will have a 63 print position line, you assign positions 1-63 to it. INVREG is a 50 print position line and you assign it to positions 81 through 131.



Line Counter Specifications

Line	Form Type	Filename	1		2		3		4		5		6		7		8		9		10		11		12	
			Line Number	FL or Channel Number	Line Number	OL or Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number	Line Number	Channel Number
3 4 5	6	7 8 9 10 11 12 13 14	15 16 17	18 19	20 21 22	23 24	25 26 27	28 29	30 31 32	33 34	35 36 37	38 39	40 41 42	43 44	45 46 47	48 49	50 51 52	53 54	55 56 57	58 59	60 61 62	63 64	65 66 67	68 69	70 71 72	73 74
1 1	L	INVOICE	Ø5Ø	FL	Ø43	OL																				
1 2	L																									
1 3	L																									

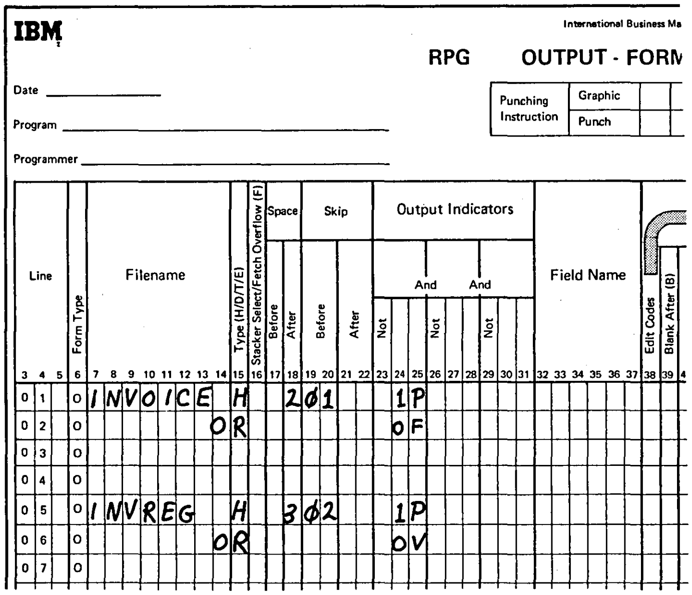
Figure 13-28. Line Counter Specifications for INVOICE



File Description Specifications

Line	Form Type	Filename	File Type I/O/U/C/D	File Designation P/S/C/R/T/D	End of File E	Sequence A/D	File Format F/V	Block Length	Record Length	Mode of Processing L/R	Length of Key Field or of Record Address Field	Record Address Type A/K/I	Type of File Organization or Additional Area I/D/T or 1-9	Overflow Indicator	Key Field Starting Location	Extension Code E/L	Device	Symbolic Device	Labels (S, N, or E)	Name of Label Exit	Extent Exit for DAM Core Index	A/U		Number of Tracks for Cylinder Overflow Number of Extents	Tape Rewind N/U	File Condition U1-U8	
3 4 5	6	7 8 9 10 11 12 13 14	15	16	17	18	19	20 21 22 23	24 25 26 27	28	29 30	31	32	33 34	35 36 37 38	39	40 41 42 43 44 45 46	47 48 49 50 51 52	53	54 55 56 57 58 59	60 61 62 63 64 65	66	67	68 69	70	71 72	73 74
0 2	F	INVOICE	O											OF		L	PRINTER										
0 3	F	INVREG	O											OV			PRINTR2										
0 4	F	INPUT	I	P	E		F										MFCU1										
0 5	F	CUSMAS	V	C			F	117	117	R	Ø7	A	I		ØØØ2		DISK							Ø1			
0 6	F																										
0 7	F																										
	F																										
	F																										

Figure 13-29. File Description Sheet for End-of-Month Billing



IBM

International Business Ma

RPG OUTPUT - FORM

Date ____________

Program ____________

Programmer ____________

Punching Instruction	Graphic		
	Punch		

Line 3 4 5	Form Type 6	Filename 7 8 9 10 11 12 13 14	Type (H/D/T/E) 15	Stacker Select/Fetch Overflow (F) 16	Space Before 17	Space After 18	Skip Before 19 20	Skip After 21 22	Output Indicators Not 23	Output Indicators 24 25	And Not 26	And 27 28	And Not 29	And 30 31	Field Name 32 33 34 35 36 37	Edit Codes 38	Blank After (B) 39
0 1	O	INVOICE	H			2	Ø1			1P							
0 2	O	O	R							OF							
0 3	O																
0 4	O																
0 5	O	INVREG	H			3	Ø2			1P							
0 6	O	O	R							OV							
0 7	O																

Figure 13-30. Heading Specifications for INVOICE and INVREG

1. What does the use of the dual feed carriage allow you to do?
2. What relationship exists between the two feeds of the dual feed carriage?
3. What limitations exist when designing forms for use with the dual feed carriage?
4. How do you distinguish between the two feeds on the RPG II specification sheets?

Answers to Review 13. Dual Feed Carriage

1. The dual feed carriage allows the printing of two independent reports simultaneously.

2. None. It is as though there were two separate printers.

3. Seventeen print positions must be left blank between the two output forms.

4. The only difference is that the device name on the File Description Sheet for the left carriage is PRINTER and for the right carriage is PRINTR2.

USING THE PRINTER-KEYBOARD AS A SECOND PRINTER

The IBM 5471 Printer-Keyboard is an input/output device. It accepts data into storage and/or prints data from storage. You will want most of your output printed on the 5203 Printer, but there are certain times when you may want output printed on the printer-keyboard. For example, OCL statements and halt messages are printed on the printer-keyboard, if you have one. If you do not have a printer-keyboard, they may be printed on the printer. If you are printing a report on the printer, you would not want halt messages from the system printed among the figures of your report. In this case, the printer-keyboard is a device which will print this output for you. There are two major uses for printing on the printer-keyboard:

- As a logging device for OCL statements and halt messages to the operator.
- As an output device for certain RPG II programs.

Using the Printer-Keyboard to Communicate With the Operator

OCL statements and message codes can be printed on the printer-keyboard. OCL statements and halt messages are printed in order to communicate the system's status to the operator. If a halt occurs, the operator will want to know why, so he can correct the situation. For example, if a wrong disk pack is mounted, the operator needs to mount the right pack before the program can execute.

You can also place comments among your OCL statements to give the operator certain set-up information. You might tell him which pack to mount, so your program will not be interrupted by a halt because a wrong pack is mounted. If your system has no printer-keyboard, the OCL statements and halt messages are printed on the printer. If you have a printer-keyboard and do not want messages printed on the printer, you can tell the system to print them on the printer-keyboard. Whether you use the printer-keyboard or the printer for these messages, the LOG statement is used to tell the system whether or not statements are to be printed and where you want to print them.

LOG Statement

The device used to print OCL statements and message codes is called the logging device. If you want to change the logging device or specify if the statements and codes are to be printed, you must use a LOG statement. You can use the LOG statement anywhere among the IPL statements or within any of the sets of OCL statements for your programs. In the job stream, a LOG statement appearing within a set of statements for a program must precede the RUN statement. If a LOG statement appears in a procedure, it must follow the LOAD statement and precede the RUN statement.

The LOG statement (Figure 13-31) can be coded to perform the following four functions:

Code	*Meaning*
CONSOLE	Use printer-keyboard as logging device.
PRINTER	Use printer as logging device.
OFF	Stop printing.
ON	Start printing.

```
// LOG   code
```

Figure 13-31. LOG Statement Format

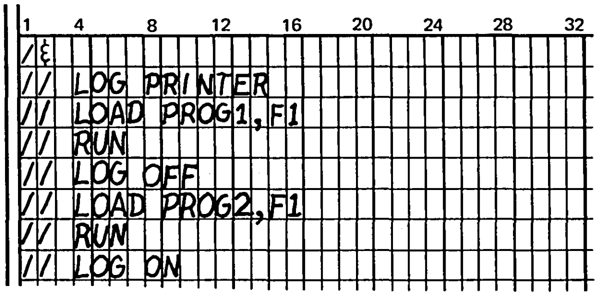
When disk system management reads a LOG statement that contains the OFF code, it stops printing OCL statements and message codes. The only way you can instruct the system to start printing them again is by using a LOG statement that contains the ON, PRINTER, or CONSOLE code. The ON code causes disk system management to resume printing on the last used logging device. If your system has no printer-keyboard, printing is resumed on the printer.

For example, assume you are using the OCL statements shown in Figure 13-32. Also assume that the system has a printer-keyboard.

Because of the first LOG statement, the printer is used as the logging device while program PROG1 is being run. OCL statements and error messages are not printed for program PROG2 because of the second LOG statement. The third LOG statement causes the logging device to be used again but the print-keyboard, not the printer, is the device used.

Using the Printer-Keyboard as an Output Device for RPG II Programs

The 5471 Printer-Keyboard can be used as the only output printer or as the second output printer for an RPG II program. The DSPLY operation code uses the printer-keyboard as the only output device, and the printer-keyboard can be used as a second printer for exception reports. Following are three applications in which the printer-keyboard can be used as the output device.



```
1  4   8   12  16  20  24  28  32
/&
// LOG PRINTER
// LOAD PROG1,F1
// RUN
// LOG OFF
// LOAD PROG2,F1
// RUN
// LOG ON
```

Figure 13-32. LOG Statement Example

Example 1

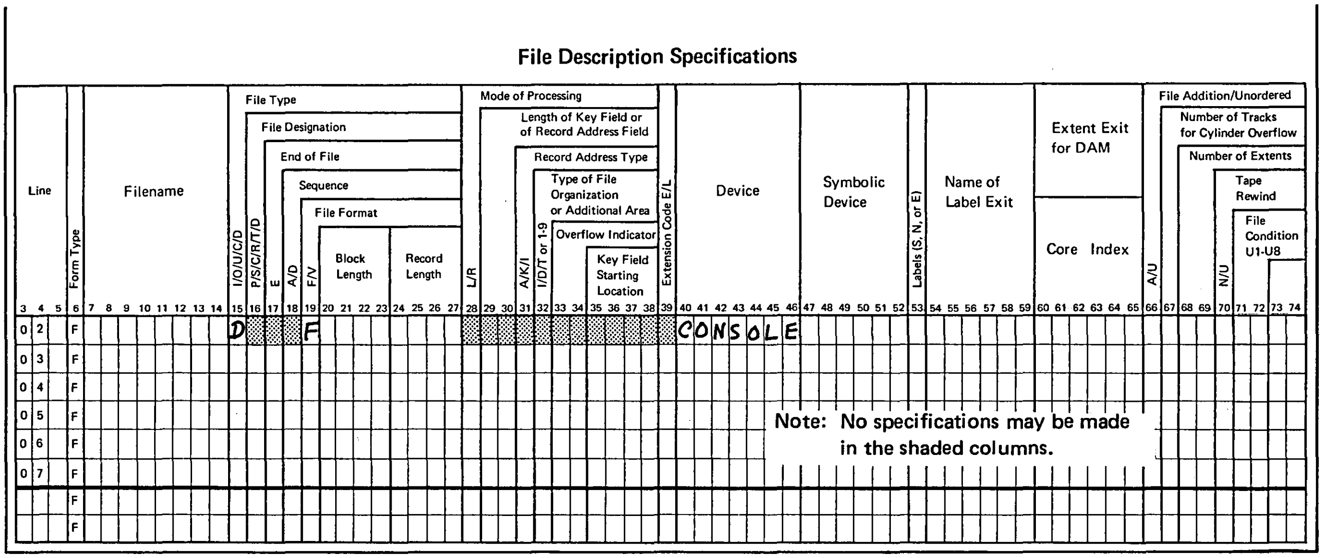
Remember that the printer-keyboard can be used to print a field. For example, if you were processing an inventory program, you might want to print all item numbers for which no disk records were found. The printer-keyboard can print a field if you use the DSPLY operation code. Output from the DSPLY operation code (either a field or array element with or without a halt) can only be printed on the printer-keyboard. This field will not be displayed on the printer. Figure 13-33 is a sample File Description Sheet for using the DSPLY operation code. Note that the file must be defined as a display file in column 15 and the output device must be CONSOLE (the symbolic device name for the 5471 Printer-Keyboard).

Example 2

Remember that the DEBUG operation is an RPG II function that you may use to help you find errors in a program which is not working properly. This code causes either one or two output records to be printed containing information which is helpful for finding programming errors. If you have a report being printed on the printer, and you find you must use the DEBUG operation code, you would not want these output records from DEBUG among your own output records. This would be a good time to use the printer-keyboard for the output records from DEBUG. Figure 13-34 shows how to define the printer-keyboard as the output device.

Example 3

The printer-keyboard can also be used as the right carriage is used on the dual feed carriage feature. Recall that the dual feed carriage feature will give you two separate output files for the same program. Just as the right carriage was used as a second output device to print the invoice register (INVREG) in the dual feed carriage application example, the printer-keyboard could also have been used as the second output device. The only difference is that the printer-keyboard must be defined as the output device on the File Description Sheet as it was in Figure 13-34.

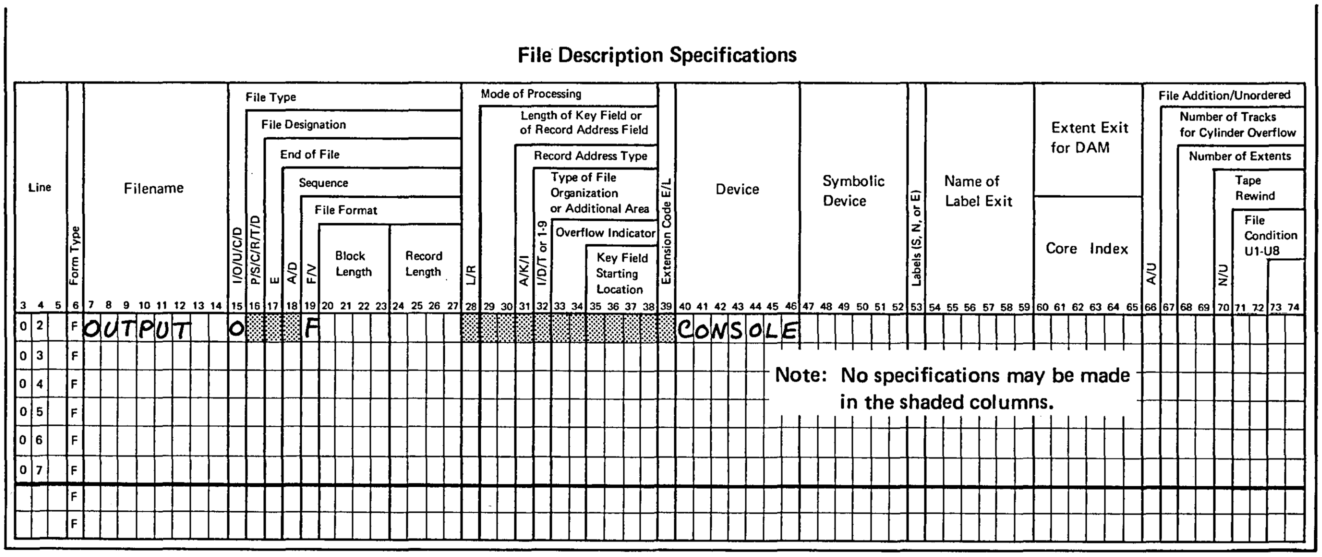


File Description Specifications

Line	Form Type	Filename	File Type	File Designation	End of File	Sequence	File Format	Block Length	Record Length	L/R	Mode of Processing	Length of Key Field or of Record Address Field	Record Address Type	Type of File Organization or Additional Area	Overflow Indicator	Key Field Starting Location	Extension Code E/L	Device	Symbolic Device	Labels (S, N, or E)	Name of Label Exit	Extent Exit for DAM / Core Index	A/U	File Addition/Unordered	Number of Tracks for Cylinder Overflow	Number of Extents	Tape Rewind	File Condition U1-U8
02	F		D				F											CONSOLE										
03	F																											
04	F																											
05	F																											
06	F																											
07	F																											
	F																											
	F																											

Note: No specifications may be made in the shaded columns.

Figure 13-33. Specifications for Using the Printer-Keyboard for Display Files



File Description Specifications

Line	Form Type	Filename	File Type	File Designation	End of File	Sequence	File Format	Block Length	Record Length	L/R	Mode of Processing	Length of Key Field or of Record Address Field	Record Address Type	Type of File Organization or Additional Area	Overflow Indicator	Key Field Starting Location	Extension Code E/L	Device	Symbolic Device	Labels (S, N, or E)	Name of Label Exit	Extent Exit for DAM / Core Index	A/U	File Addition/Unordered	Number of Tracks for Cylinder Overflow	Number of Extents	Tape Rewind	File Condition U1-U8
02	F	OUTPUT	O				F											CONSOLE										
03	F																											
04	F																											
05	F																											
06	F																											
07	F																											
	F																											
	F																											

Note: No specifications may be made in the shaded columns.

Figure 13-34. Specifications for Using the Printer-Keyboard as an Output File

1. What are the two general uses for the printer-keyboard?

2. Name the three types of situations when you may want to use the printer-keyboard as an output device.

3. Define the printer-keyboard files on the File Description Sheet for these three situations.

Answers to Review 13. Printer-Keyboard

1. The printer-keyboard can be used to log OCL statements and operator messages and as an output device for RPG II programs.

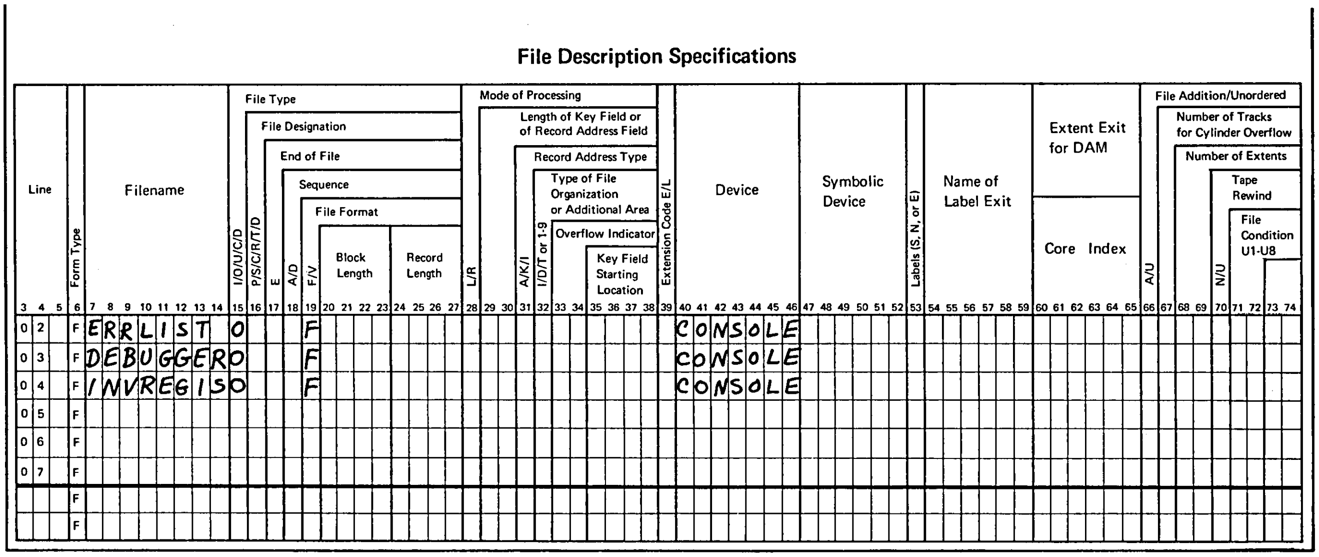
2. You may want to define the printer-keyboard as an output device for:

   1. The DSPLY operation code.

   2. The DEBUG operation code.

   3. Specifying normal output on an output sheet.

3.



File Description Specifications

Line	Form Type	Filename	File Type I/O/U/C/D	File Designation P/S/C/R/T/D	End of File E	Sequence A/D	File Format F/V	Block Length	Record Length	Mode of Processing L/R	Length of Key Field or of Record Address Field	Record Address Type A/K/I	Type of File Organization or Additional Area I/D/T or 1-9	Overflow Indicator	Key Field Starting Location	Extension Code E/L	Device	Symbolic Device	Labels (S, N, or E)	Name of Label Exit	Extent Exit for DAM / Core Index	A/U	Number of Tracks for Cylinder Overflow	Number of Extents	N/U	Tape Rewind	File Condition U1-U8
02	F	ERRLIST	O				F										CONSOLE										
03	F	DEBUGGER	O				F										CONSOLE										
04	F	INVREGIS	O				F										CONSOLE										
05	F																										
06	F																										
07	F																										
	F																										
	F																										

CHAPTER

ALTERING THE ORDER OF PROCESSING FILES

14

CHAPTER 14 DESCRIBES:

RPG II FORCE operation code.

Forcing records from a file.

FORCE and the look-ahead feature.

BEFORE READING THIS CHAPTER YOU SHOULD BE ABLE TO DESCRIBE:

Multi-file processing.

Look-ahead.

Matching records logic and match fields.

RPG II object cycle.

End-of-file condition.

AFTER READING THIS CHAPTER YOU SHOULD BE ABLE TO DESCRIBE:

Functions and coding of the FORCE operation code.

How to force records from a file.

Use and coding of FORCE with the look-ahead feature.

INTRODUCTION

RPG II uses two methods to determine the order in which records are processed in a multi-file job.

- If match fields are not specified for either file, all records in the primary file are processed, followed by those in the secondary files in the order defined on the File Description Sheets.

- When match fields are assigned, the RPG II logic of matching records determines from which file the next record is to be processed.

The order of processing determined by RPG II logic is appropriate for most of your multi-file jobs. However, for certain jobs, it may be necessary to have some of the records in the two files processed in an order other than that in which RPG II logic would select the records.

A record can be processed out of order only if you indicate to the program that the file containing that record is to be forced. To do this, you must code additional specifications.

Regardless of how your files are organized, the following situations require that you alter the order of processing:

1. Match fields cannot be assigned to the files and you wish to:

   a. Process a primary file record followed by a particular number of secondary file records.

   b. Process a secondary file record only when it matches a primary file record.

2. Match fields are assigned to both input files. You wish to process one primary file record, followed by matching secondary file records, then the rest of the matching primary records.

3. A particular file is to be used in a job only if certain conditions occur.

To alter the order of processing, you must first determine which file is to be processed – when and under which conditions. Once you know the order, the next step is to determine, for a particular programming cycle, whether the RPG II logic will select the appropriate record or if you must force the processing of that record.

The first record to be processed in any job can only be selected by RPG II logic in the usual way. Thereafter, to alter the order of processing, you tell the program to force a record from a file which would not ordinarily be processed next. Once the forced record is processed, and providing another record is not forced, the RPG II logic selects the next record in the usual way. This is the record which would have ordinarily been processed if the other file had not been forced.

FORCE: SPECIFYING THE NEXT FILE TO PROCESS

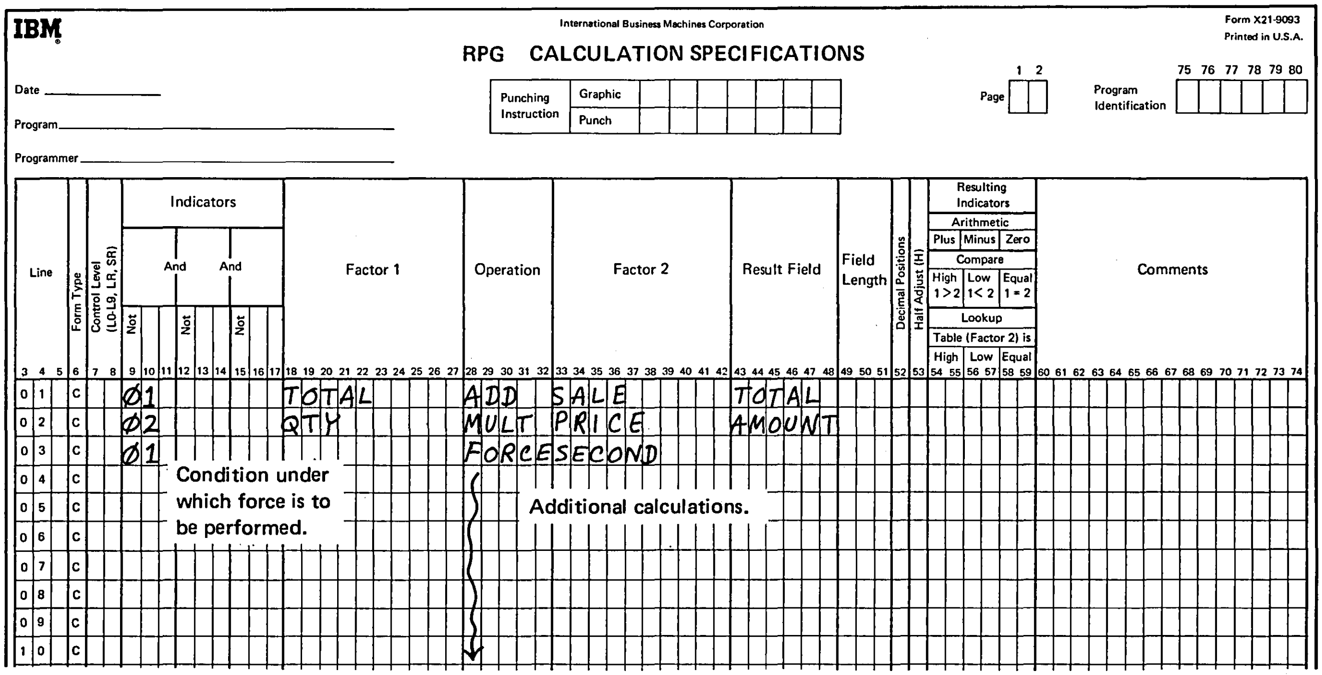
To process a record out of its normal sequence, you specify on the Calculation Sheet the FORCE operation code and the name of the file which is to be forced in the *next* program cycle (Figure 14-1).

Assuming a record type 01 from the primary file is being processed, the calculation on line 01 is performed. The next detail-time calculation specification for record type 01 indicates that the secondary file (SECOND) is to be forced. The FORCE does not occur immediately, however. This specification only tells the program to remember that a record from the file SECOND is to be processed next. Any additional calculations and/or output for the record being processed are performed first to complete the present program cycle.

At the beginning of a normal program cycle, RPG II logic looks at the two records available to select the one to process during that cycle. However, if the record from the file which would not normally be selected is to be processed, this must be indicated to the program before the beginning of the cycle. If a file is to be forced, there is no need for RPG II logic to compare the records and make a selection. This is the reason that, if a file is to be forced, the FORCE must be indicated during the program cycle immediately before the cycle in which the FORCE is to occur.

Depending on your job, you may not have to force a record in every program cycle of a job. For such situations, you must indicate when the FORCE is to be done by specifying conditioning indicators in columns 9-17 of the Calculation Sheet (Figure 14-1). Whether the FORCE is to be performed in the next cycle or not may depend on any of several conditions:

- The type of file or record type being processed at the time.
- The number of records which have been processed.
- The result of a calculation performed.
- The contents of a field on the record being processed.
- The contents of a field on a record which has not been processed yet.



IBM

International Business Machines Corporation

Form X21-9093
Printed in U.S.A.

RPG CALCULATION SPECIFICATIONS

Date ____
Program ____
Programmer ____

Punching Instruction: Graphic / Punch

Page 1 2

Program Identification 75 76 77 78 79 80

Line	Form Type	Control Level (L0-L9, LR, SR)	Indicators	Factor 1	Operation	Factor 2	Result Field	Field Length	Decimal Positions	Half Adjust (H)	Resulting Indicators	Comments
01	C		Ø1	TOTAL	ADD	SALE	TOTAL					
02	C		Ø2	QTY	MULT	PRICE	AMOUNT					
03	C		Ø1		FORCE	SECOND						
04	C											
05	C											
06	C											
07	C											
08	C											
09	C											
10	C											

Condition under which force is to be performed.

Additional calculations.

Figure 14-1. Specifying the File to be Processed in the Next Program Cycle

Forcing a Number of Records from a File

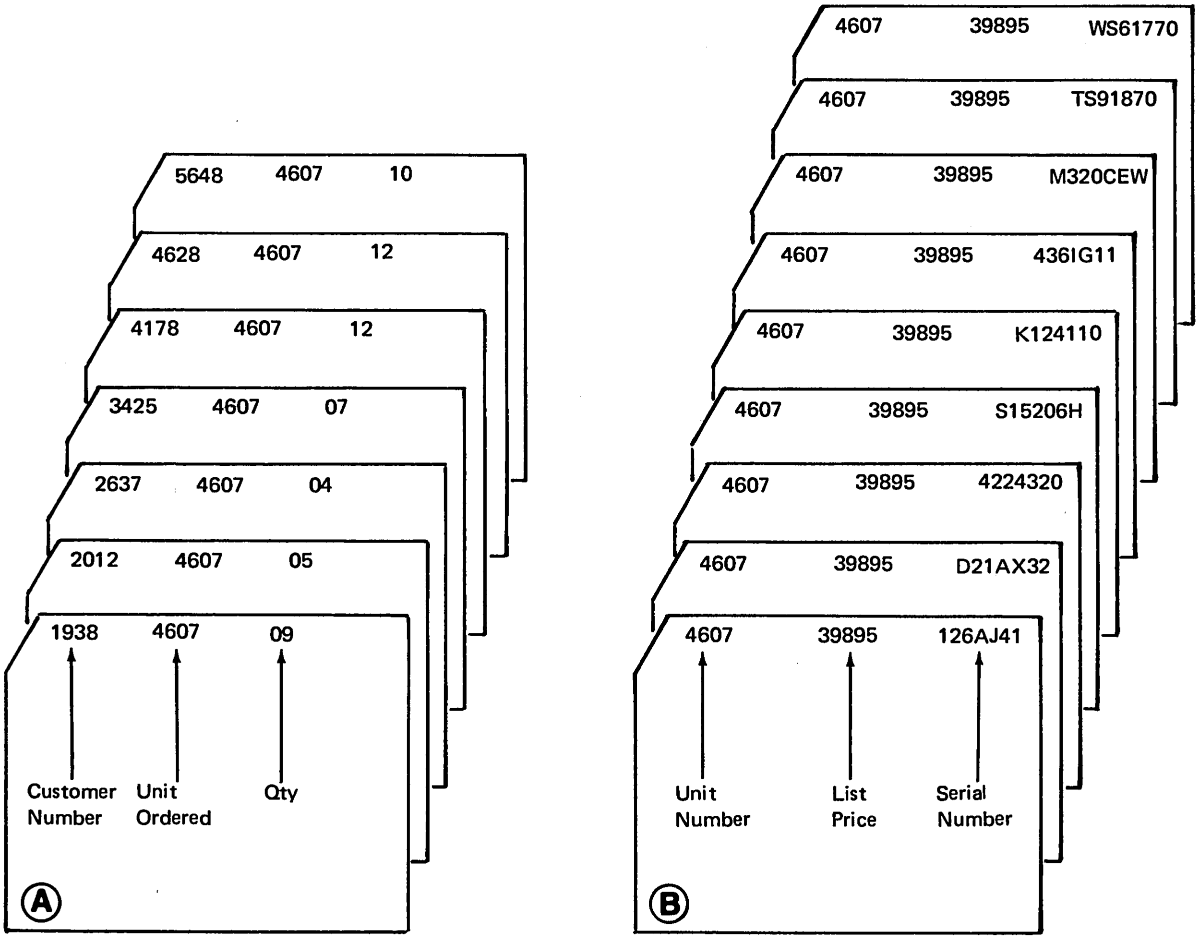
Let's consider a case in which you condition the FORCE operation on the basis of whether a resulting indicator is on or off.

Suppose you have a number of customers who periodically order items to be delivered from a central warehouse. One record is kept for each unit in stock in the warehouse, and another record for each customer's order of a particular unit.

Orders are processed according to the type of unit ordered. Therefore, for a particular run, the primary file (ORDER) contains all order records for only one type of unit, and the secondary file (STOCK) contains all in-stock records for that type of unit.

For this run, the ORDER file (Figure 14-2, insert A) contains the week's order records for television sets, unit number 4607. The records show which customer placed the order, and the quantity of television sets wanted. The STOCK file consists of a separate record for each television set (unit 4607) in stock (Figure 14-2, insert B). Each record provides the unit number, list price, and serial number of the item.

There are two purposes for processing the files. First, you want an indication of which orders can be filled and which orders cannot be filled. Secondly, the STOCK file is to be kept up-to-date so it only contains as many records as there are television sets available.



ORDER file

STOCK file

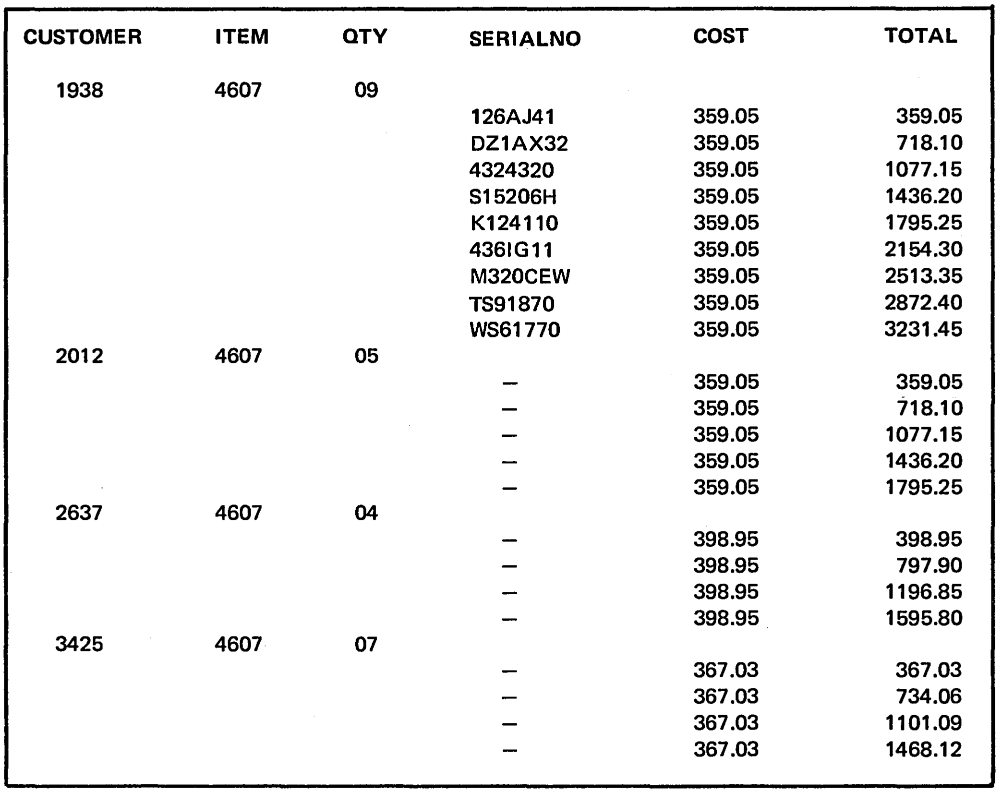
Figure 14-2. Files for Processing Customer Orders

The job should produce a printed report showing which orders can be filled, and the amount each customer owes (Figure 14-3). Thus, you must determine the amount due for each item and the total amount for each order. A record from each file must be available before you can calculate the information.

Files for this job must be processed in a specific order. The quantity ordered (QTY) from an ORDER record must be available first. This quantity is used to determine how many STOCK records are to be processed. When enough STOCK records for an order have been processed, the next ORDER card is selected to repeat the process.

Looking at the two files, you can see that every record has a common field containing the unit number. It does no good to assign a match field to control processing order, because the unit number is always the same for every record. All records in the primary file would be processed before any secondary file records.

Remember, there is no way you can control selection of the first record to be processed in a job. RPG II logic always selects a primary file record first when match fields are not specified. Since an ORDER record must be available first for this job, the ORDER file should be designated as the primary file.



```
CUSTOMER      ITEM      QTY      SERIALNO      COST          TOTAL

  1938        4607      09
                                  126AJ41      359.05         359.05
                                  DZ1AX32      359.05         718.10
                                  4324320      359.05        1077.15
                                  S15206H      359.05        1436.20
                                  K124110      359.05        1795.25
                                  4361G11      359.05        2154.30
                                  M320CEW      359.05        2513.35
                                  TS91870      359.05        2872.40
                                  WS61770      359.05        3231.45
  2012        4607      05
                                     –         359.05         359.05
                                     –         359.05         718.10
                                     –         359.05        1077.15
                                     –         359.05        1436.20
                                     –         359.05        1795.25
  2637        4607      04
                                     –         398.95         398.95
                                     –         398.95         797.90
                                     –         398.95        1196.85
                                     –         398.95        1595.80
  3425        4607      07
                                     –         367.03         367.03
                                     –         367.03         734.06
                                     –         367.03        1101.09
                                     –         367.03        1468.12
```

Figure 14-3. Printed Report Showing Customer Orders Processed

Controlling the Number of Times FORCE is Performed

After RPG II selects and processes an ORDER record, you must FORCE the processing of a number of STOCK records. The quantity ordered (QTY) from the ORDER record is used to control the number of times you force secondary file records. The quantity is stored in a field, called COUNT, to keep track of how many records are left to be forced for an order. Each time a STOCK record is forced, the number in COUNT is reduced by one. When COUNT reaches zero, enough STOCK records have been processed for that particular order. Then RPG II logic can again take over to process the next ORDER record (Figure 14-4).

The calculation specifications shown for this job (Figure 14-5) only determine if a record is to be forced in the next cycle.

Assume the first ORDER record (record type 01) has been selected, making the quantity ordered (QTY) available. The first calculation specification (line 01) for this record type stores the quantity in the COUNT field. Then the program determines if any STOCK records are to be processed for this order (line 05). If COUNT is greater than zero, indicator 27 turns on. With 27 on, line 06 is performed, indicating a STOCK record must be forced in the next program cycle.

At the beginning of the second cycle then, the first STOCK record (record type 02) is selected (by being forced). Line 03 is performed to reduce the COUNT by one for this record being processed. The COUNT is then compared to zero again (line 05) to determine if any more STOCK records are to be processed for this order. If COUNT is still greater than zero (27 set on again), line 06 is performed again, indicating another STOCK record is to be forced at the beginning of the next cycle.

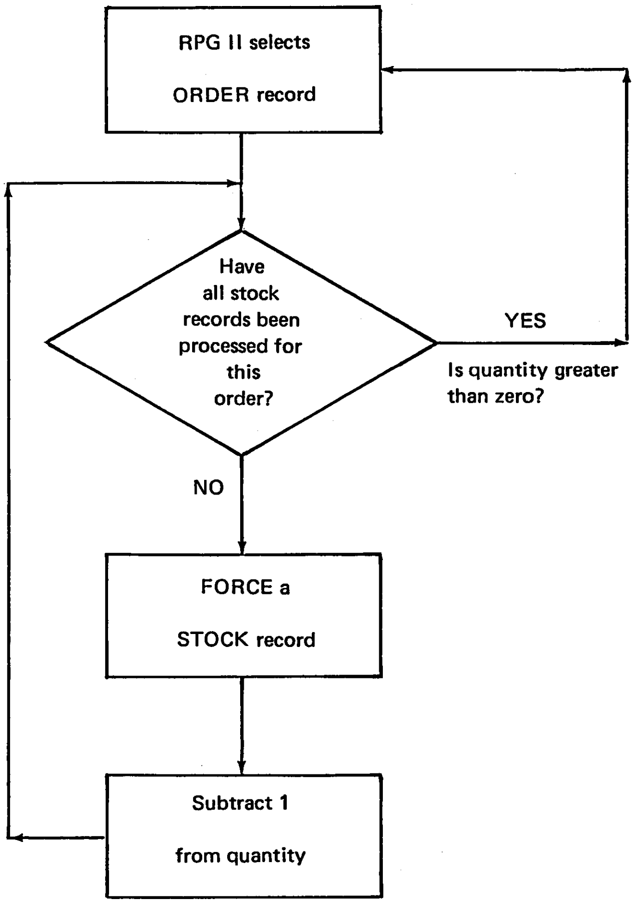
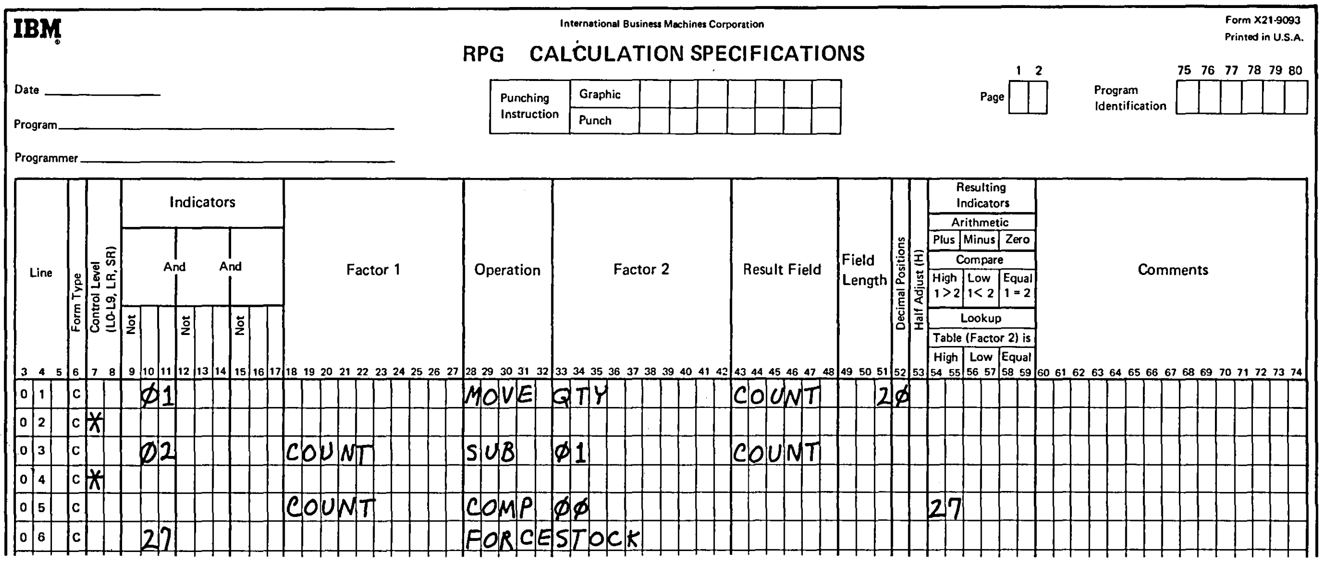

RPG II selects
ORDER record
Have all stock records been processed for this order?
YES
Is quantity greater than zero?
NO
FORCE a
STOCK record
Subtract 1
from quantity

Figure 14-4. Determining When Stock Records Must be Forced to Fill an Order



IBM

International Business Machines Corporation

RPG CALCULATION SPECIFICATIONS

Form X21-9093
Printed in U.S.A.

Date

Program

Programmer

Punching Instruction: Graphic □□□□□□□□ Punch □□□□□□□□

Page □□ (1 2)

Program Identification □□□□□□ (75 76 77 78 79 80)

Line (3-5)	Form Type (6)	Control Level (L0-L9, LR, SR) (7-8)	Indicators (9-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field (43-48)	Field Length (49-51)	Decimal Positions (52)	Half Adjust (H) (53)	Resulting Indicators: Plus / High 1>2 (54-55)	Minus / Low 1<2 (56-57)	Zero / Equal 1=2 (58-59)	Comments (60-74)
01	C		Ø1		MOVE	QTY	COUNT	2	Ø					
02	C	*												
03	C		Ø2	COUNT	SUB	Ø1	COUNT							
04	C	*												
05	C			COUNT	COMP	ØØ					27			
06	C		27		FORCE	STOCK								

Figure 14-5. Controlling the Number of Times a File is Forced

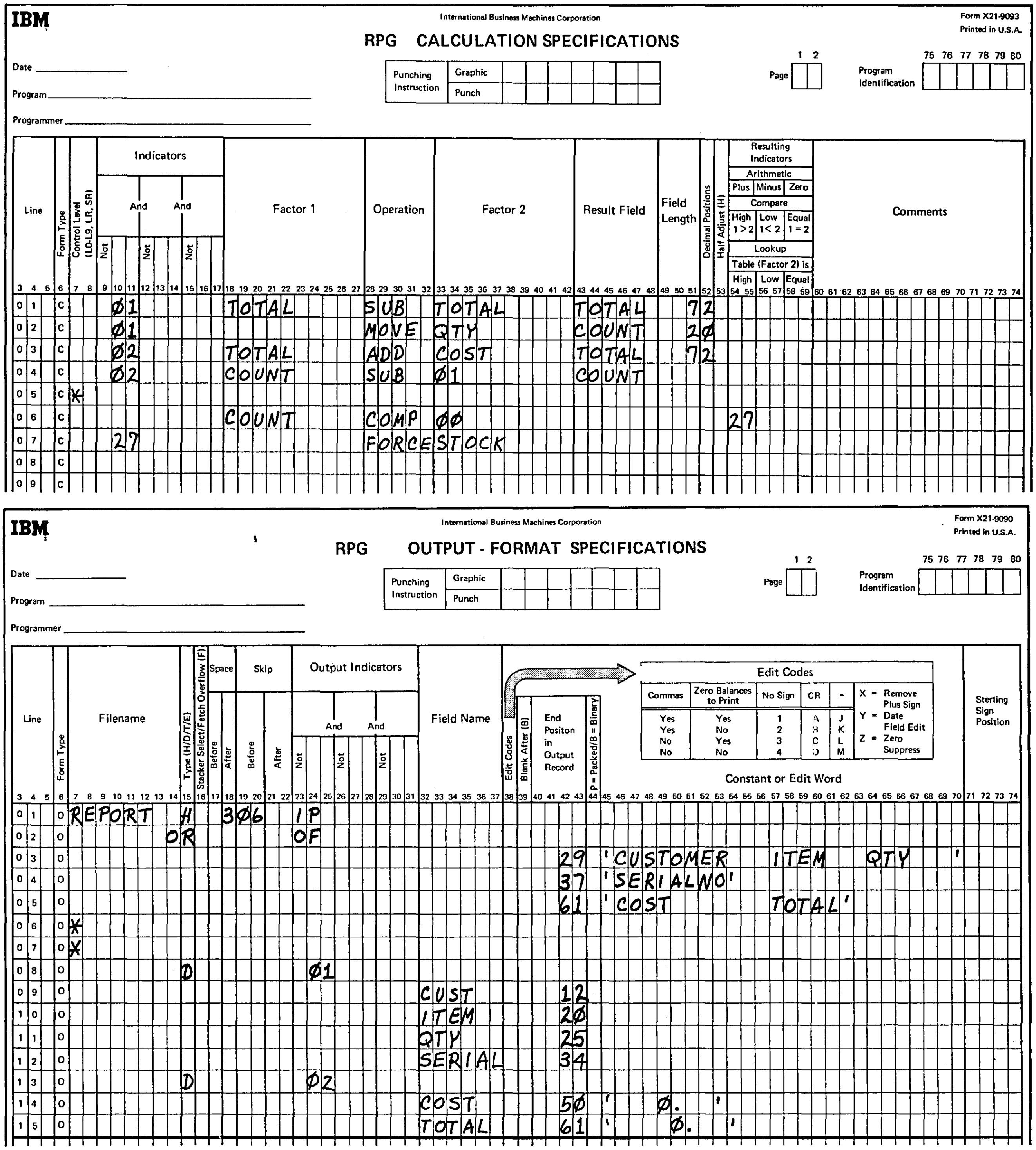
During processing of the second STOCK record, COUNT is again reduced by one (line 03). Assuming COUNT is now at zero, the COMPARE operation on line 05 sets indicator 27 off. With 27 off, the FORCE operation on line 06 is not performed during this cycle. At the beginning of the next program cycle then, RPG II selects the next ORDER record from the primary file in the usual way.

At this point, add the specifications for calculating the amount due and for printing the report (Figure 14-6). An ORDER record is selected first, making the QTY available. Calculation lines 01, 02, and 06 in Figure 14-6 are performed for this record (record type 01). First, a TOTAL field, to be printed for each group of customers, is set to zero (line 01). Next, the quantity ordered is moved into the COUNT field (line 02). If COUNT is greater than zero, indicator 27 is turned on (line 06), and line 07 is performed; the program is instructed to force a STOCK record at the beginning of the next cycle. Before forcing, however, the output specifications (Figure 14-6, lines 08-12) are performed to print data from the ORDER record.

Following output, the next cycle begins with a forced STOCK record (record type 02) being processed. The cost is added to a TOTAL field (line 03) to accumulate the total amount due on the order.

Then, COUNT is reduced by one for the record being processed (line 04). Once again COUNT is compared to zero (line 06) to determine if line 07 should be performed; that is, to determine if another STOCK record is to be forced for the next cycle. The COST and TOTAL calculated for the STOCK record are then printed, by performing the output specifications on lines 13-15.

The record selected at the beginning of the next program cycle depends on whether a FORCE operation was indicated in the previous cycle. If calculation line 09 had been performed, another STOCK record would be processed (by being forced). If not, RPG II would select the next ORDER record from the primary file.



IBM

International Business Machines Corporation

Form X21-9093
Printed in U.S.A.

RPG CALCULATION SPECIFICATIONS

Date ______ Program ______ Programmer ______

Punching Instruction: Graphic / Punch

Page 1 2 Program Identification 75 76 77 78 79 80

Line	Form Type	Control Level (L0-L9, LR, SR)	Indicators (Not, And, And)	Factor 1	Operation	Factor 2	Result Field	Field Length	Decimal Positions	Half Adjust (H)	Resulting Indicators: Plus / High 1>2	Minus / Low 1<2	Zero / Equal 1=2	Comments
0 1	C		Ø1	TOTAL	SUB	TOTAL	TOTAL	7	2					
0 2	C		Ø1		MOVE	QTY	COUNT	2	Ø					
0 3	C		Ø2	TOTAL	ADD	COST	TOTAL	7	2					
0 4	C		Ø2	COUNT	SUB	Ø1	COUNT							
0 5	C	*												
0 6	C			COUNT	COMP	ØØ					27			
0 7	C		27		FORCE	STOCK								
0 8	C													
0 9	C													

IBM

International Business Machines Corporation

Form X21-9090
Printed in U.S.A.

RPG OUTPUT - FORMAT SPECIFICATIONS

Date ______ Program ______ Programmer ______

Punching Instruction: Graphic / Punch

Page 1 2 Program Identification 75 76 77 78 79 80

Edit Codes

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Line	Form Type	Filename	Type (H/D/T/E)	Stacker Select/Fetch Overflow (F)	Space Before	Space After	Skip Before	Skip After	Output Indicators (Not, And, And)	Field Name	Edit Codes	Blank After (B)	End Position in Output Record	P = Packed/B = Binary	Constant or Edit Word	Sterling Sign Position
0 1	O	REPORT	H			3	Ø6		1P							
0 2	O	OR							OF							
0 3	O												29		'CUSTOMER ITEM QTY '	
0 4	O												37		'SERIALNO'	
0 5	O												61		'COST TOTAL'	
0 6	O*															
0 7	O*															
0 8	O		D						Ø1							
0 9	O									CUST			12			
1 0	O									ITEM			2Ø			
1 1	O									QTY			25			
1 2	O									SERIAL			34			
1 3	O		D						Ø2							
1 4	O									COST			5Ø		' Ø. '	
1 5	O									TOTAL			61		' Ø. '	

Figure 14-6. Specifications to Process Customer Orders

Controlling Processing After Reaching End of File

In a multifile job, end-of-file entries can be specified for one, both, or neither of the two input files. If an end-of-file entry is specified for only one file, processing stops after all records from that file have been processed. (Remember, however, if match fields are assigned to the files, the program continues to process the records from the other file which match the last record processed or which have no match fields.) If end-of-file entries are specified for both or for neither of the input files, processing continues until all records in both files have been processed.

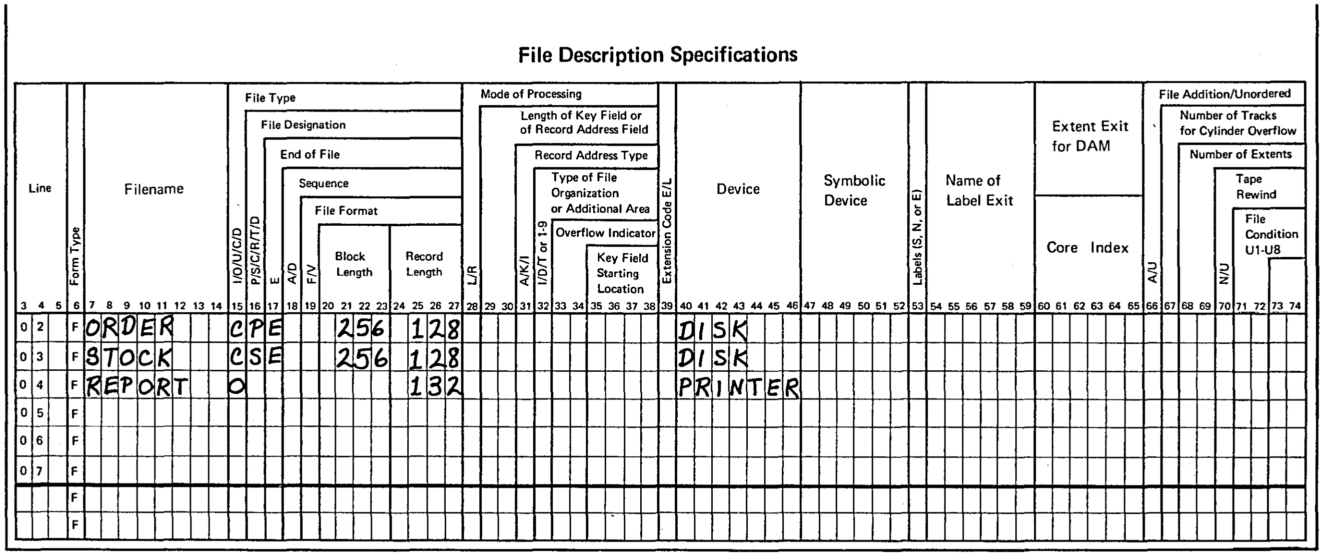
For this job, processing must continue until both files reach end of file. Actually, if one file runs out before the other, you do not want to perform the usual calculations and output for the remaining file.

To continue the job until both files have been completely processed, specify end-of-file entries on the File Description Sheet either for both or for neither of the two input files (Figure 14-7). By doing this, the LR indicator will not be set on to end the job until the last record of the last file has been processed.

With end of file specified for both files, consider what will happen when only one of the files reaches end of file (Figure 14-6). First, suppose the STOCK file reaches end of file before all ORDER records are processed. Since both files are not at end of file, processing will not stop. Instead, after processing the next ORDER record, the program will try to force the appropriate number of records from the STOCK file. With no more STOCK records to force, another ORDER record will be selected. Once again, the program will try to force STOCK records for that order. The process continues until all primary file ORDER records have been processed. Furthermore, every time a new ORDER record is processed, it is printed on the report as if the order were being filled.

A similar problem arises if the ORDER file reaches end of file first. Suppose the last order has just been filled.

After processing the last STOCK record for that order, no more STOCK records are to be forced, so RPG II logic tries to select the next primary file ORDER record. In doing so, the end of file of the ORDER file is reached. Since the job is to continue until both files are processed, RPG II logic automatically processes the remaining records in the other file (STOCK). As each remaining STOCK record is processed, the calculations and output for that record type are performed as if the record were being used to fill an order.



File Description Specifications

Line (3-5)	Form Type (6)	Filename (7-14)	File Type I/O/U/C/D (15)	File Designation P/S/C/R/T/D (16)	End of File E (17)	Sequence A/D (18)	File Format F/V (19)	Block Length (20-23)	Record Length (24-27)	Device (40-46)	Symbolic Device (47-52)
02	F	ORDER	C	P	E			256	128	DISK	
03	F	STOCK	C	S	E			256	128	DISK	
04	F	REPORT	O						132	PRINTER	
05	F										
06	F										
07	F										
	F										
	F										

Figure 14-7. Continuing Processing Until Both Files Reach End of File

LOOK-AHEAD TO DETERMINE WHETHER A FILE IS TO BE FORCED

For some jobs, you cannot determine whether a record is to be forced in the next cycle until you know something about the records which have not been processed yet. Thus, you must look ahead in one or both files at the next record which is not yet available for processing. In looking ahead, you may be checking to determine what record type is next, to see what data is on the next record, or to determine if the next record has the same match field as the record being processed. What you find in looking ahead can determine which file is to be processed next and, also, whether the file must be forced or not.

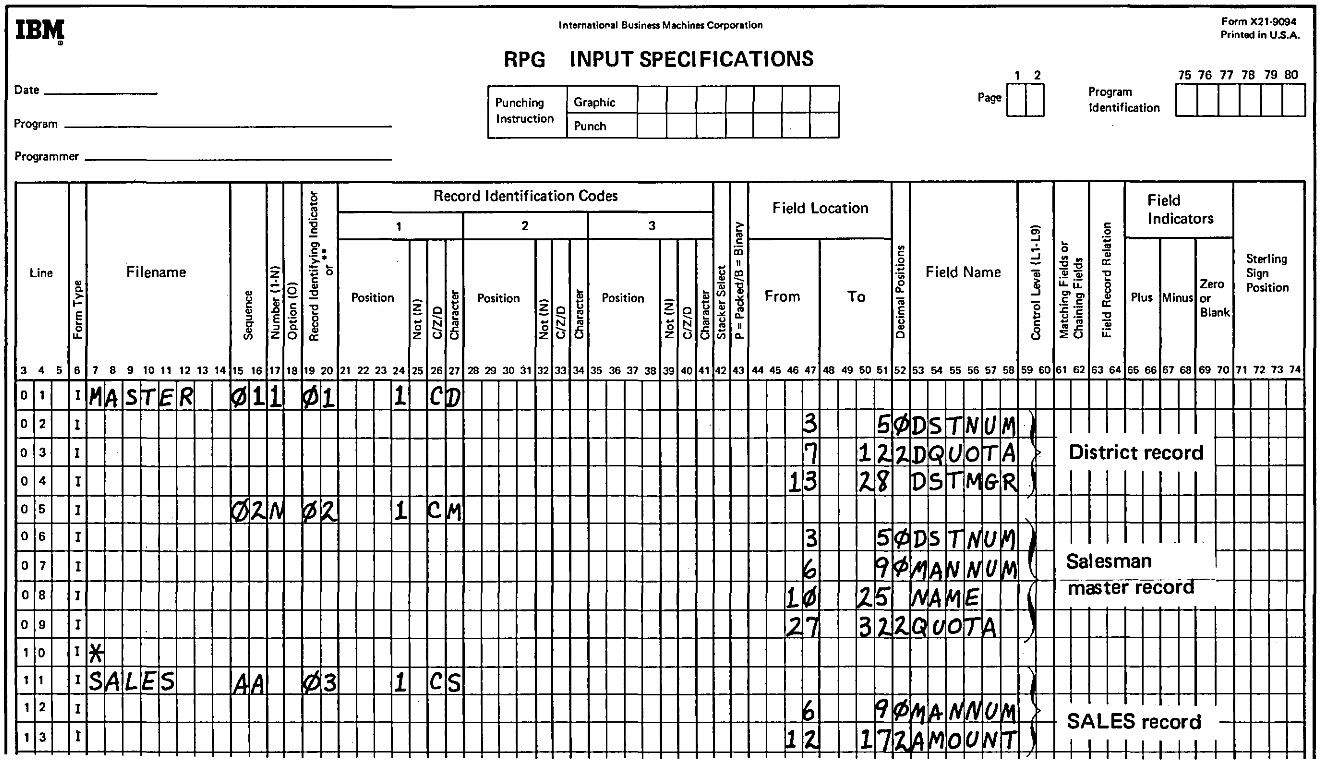
Before considering the use of FORCE with look-ahead, however, you should evaluate your system design. If at all possible, you should organize your files in such a way that the RPG II logic can determine the appropriate order of file processing. In this way, you do not have to code additional specifications to control the order. Of course, from time to time you may have jobs in which you must use FORCE and look-ahead.

Performing a Matching Records Job Without Match Fields

If two files are organized such that the same match fields cannot be assigned to the two files, you can still process the matching records together by using look-ahead fields and the FORCE operation. (This cannot be done, however, if the look-ahead file is defined as a combined or update file.) Look-ahead can be used to determine if certain fields (not assigned as match fields) on an unprocessed record match those on the record being processed. If they match, FORCE is performed to cause the matching unprocessed record to be selected next.

As an example, assume a report is to be prepared showing the amount of each salesman's sales and his quota. The report should also compare the total of district sales with the district quota.

The two files available for this job are described in Figure 14-8. The primary file (MASTER) contains a district record (record type 01), followed by all salesmen's MASTER records (type 02) associated with the district. These are in turn followed by the next district record (01) and its related salesmen MASTER records (02) and so on. Although the records are grouped by district, all salesman MASTER records in the file are still in ascending sequence by salesman number. The secondary file (SALES) contains only one record type (03), a record for each individual sale. The SALES file is also in ascending sequence by salesman number. While the MASTER file contains a record for every salesman (assume there are no MASTER records missing), the SALES file may contain only one, several, or even no records for a particular salesman.



IBM

International Business Machines Corporation

Form X21-9094
Printed in U.S.A.

RPG INPUT SPECIFICATIONS

Date

Program

Programmer

Punching Instruction | Graphic | Punch

Page 1 2

Program Identification 75 76 77 78 79 80

Line	Form Type	Filename	Sequence	Number (1-N)	Option (O)	Record Identifying Indicator or **	Position	Not (N)	C/Z/D	Character	From	To	Decimal Positions	Field Name
01	I	MASTER	Ø1	1		Ø1	1		C	D				
02	I										3	5	Ø	DSTNUM
03	I										7	12	2	DQUOTA
04	I										13	28		DSTMGR
05	I		Ø2	N		Ø2	1		C	M				
06	I										3	5	Ø	DSTNUM
07	I										6	9	Ø	MANNUM
08	I										1Ø	25		NAME
09	I										27	32	2	QUOTA
10	I	*												
11	I	SALES	AA			Ø3	1		C	S				
12	I										6	9	Ø	MANNUM
13	I										12	17	2	AMOUNT

District record

Salesman master record

SALES record

Figure 14-8. Describing Files to be Matched Without Match Fields

To produce the report, the records should be processed in the following order:

1. District record (record type 01).
2. Salesman MASTER record (record type 02).
3. All SALES records for that salesman (record type 03).
4. Next salesman MASTER record.
5. SALES records for that salesman.
6. Next district record (after all salesman MASTER records associated with the first district have been processed).

There is no common field on all three record types which can be assigned as a match field to cause the records to be processed in this order. Totals are to be accumulated by salesman number; but the MANNUM field is contained only on the salesman MASTER and SALES records. The district records do not contain this information.

If the MANNUM fields are assigned as M1 match fields for only two of the record types, the records will be processed in an incorrect order. Following the last salesman MASTER record (02 record type) for a particular district, the next record in the same file is another district card. Since district records have no M1 match field entry assigned (no MANNUM field), the district record is processed immediately before the SALES records for the last salesman.

Although this job cannot be done using match fields, you can match the records yourself by using the look-ahead capability to compare fields (Figure 14-9). The object is to match a salesman's SALES record with this salesman MASTER record. Thus, the MANNUM field on a SALES record is defined as a look-ahead field. While processing a salesman MASTER record, you then look ahead (in calculations) at the unprocessed SALES record to determine if the salesman number is the same as on the record being processed. If they match, the FORCE operation code is used to process the SALES record as if RPG II logic were performing a matching records job. You then continue to force the rest of the SALES records which contain the same salesman number. For each record, look-ahead is used to check the MANNUM field to determine if the SALES record should be forced. When look-ahead indicates that the next SALES record is for a different salesman number, the SALES record is not forced. Instead, RPG II takes over to select the next primary file record which is either another salesman MASTER record or the next district record.

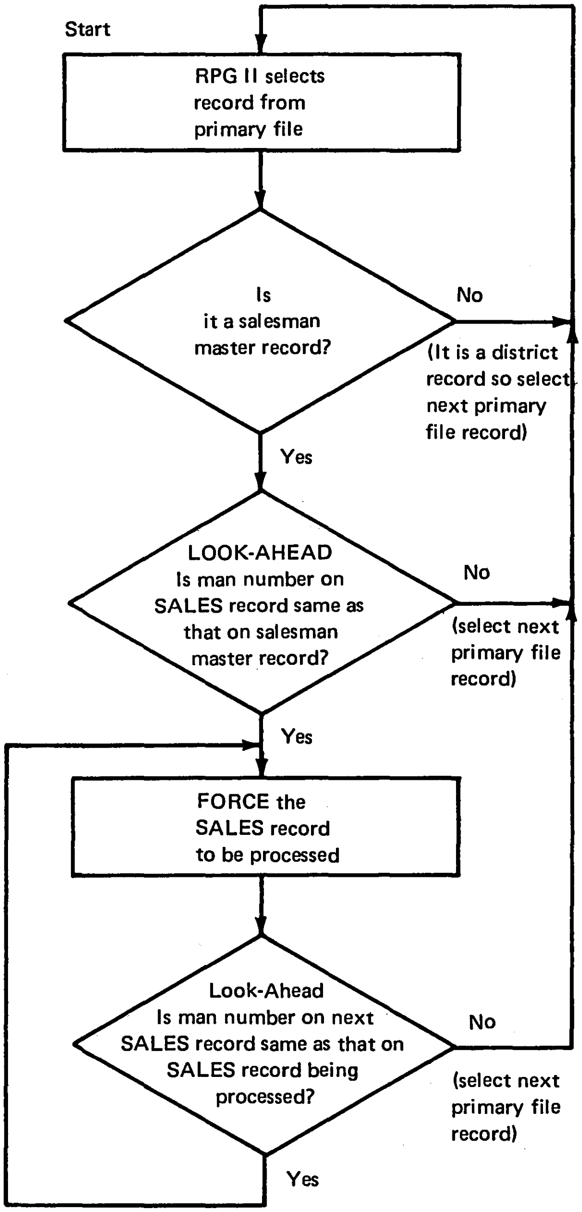
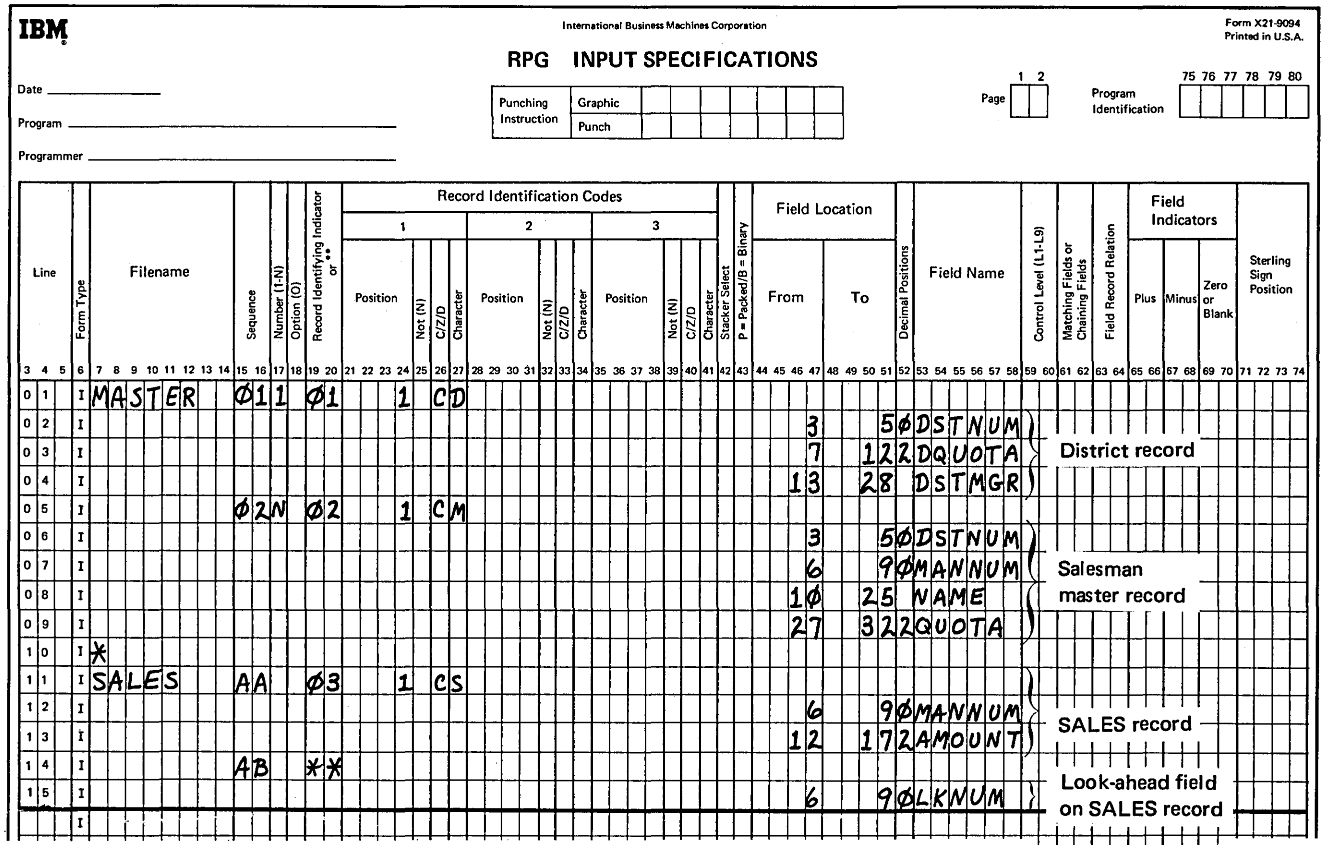

Start
RPG II selects record from primary file
Is it a salesman master record?
No
(It is a district record so select next primary file record)
Yes
LOOK-AHEAD Is man number on SALES record same as that on salesman master record?
No
(select next primary file record)
Yes
FORCE the SALES record to be processed
Look-Ahead Is man number on next SALES record same as that on SALES record being processed?
No
(select next primary file record)
Yes

Figure 14-9. Using Look Ahead to Determine if Records Match

Now that you understand the steps involved in this job, look at the specifications in Figure 14-10. The Input Sheet describes the records in each file and defines the look-ahead field for the SALES records. Only the calculations necessary to determine which record is to be processed next are shown. To actually prepare the report, you would need file description specifications to define the files and additional calculation and output specifications to accumulate totals and print the data.



IBM

International Business Machines Corporation

RPG INPUT SPECIFICATIONS

Form X21-9094
Printed in U.S.A.

Date

Program

Programmer

Punching Instruction: Graphic / Punch

Page (1 2)

Program Identification (75 76 77 78 79 80)

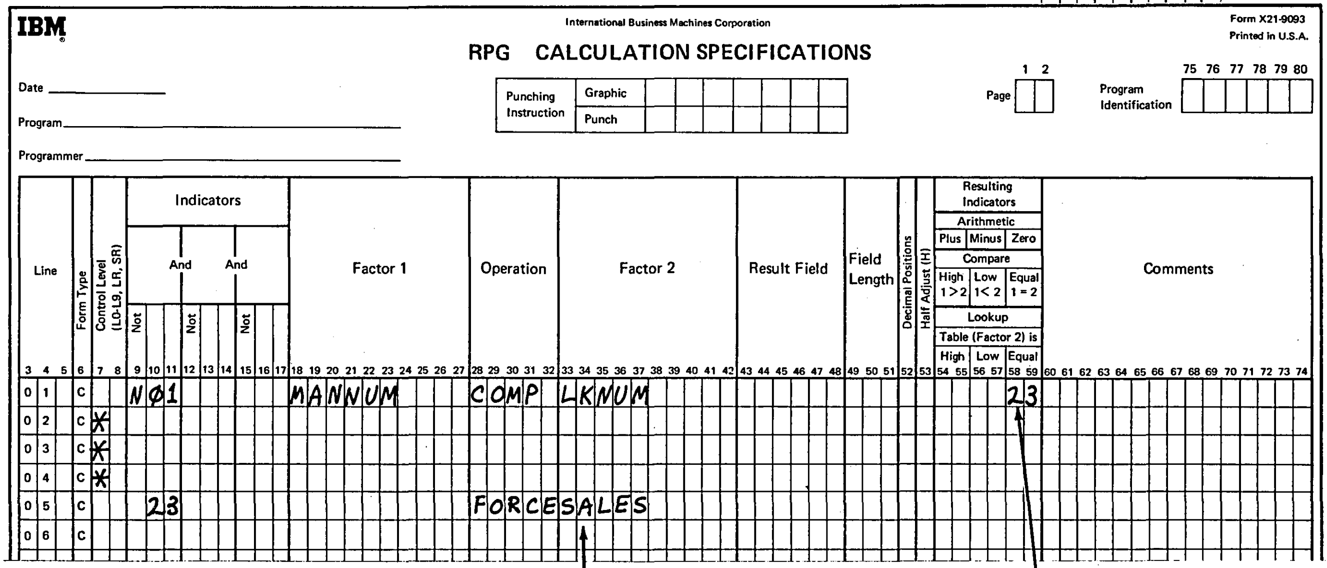
Line	Form Type	Filename	Sequence	Number (1-N)	Option (O)	Record Identifying Indicator or **	Code 1 Position	Not (N)	C/Z/D	Character	Code 2 Position	Not (N)	C/Z/D	Character	Code 3 Position	Not (N)	C/Z/D	Character	Stacker Select	P = Packed/B = Binary	Field Location From	Field Location To	Decimal Positions	Field Name	Control Level (L1-L9)	Matching Fields or Chaining Fields	Field Record Relation	Field Indicators Plus	Minus	Zero or Blank	Sterling Sign Position
01	I	MASTER	Ø1	1		Ø1	1		C	D																					
02	I																				3	5	Ø	DSTNUM							
03	I																				7	12	2	DQUOTA							
04	I																				13	28		DSTMGR							
05	I		Ø2	N		Ø2	1		C	M																					
06	I																				3	5	Ø	DSTNUM							
07	I																				6	9	Ø	MANNUM							
08	I																				1Ø	25		NAME							
09	I																				27	32	2	QUOTA							
10	I*																														
11	I	SALES	AA			Ø3	1		C	S																					
12	I																				6	9	Ø	MANNUM							
13	I																				12	17	2	AMOUNT							
14	I		AB			**																									
15	I																				6	9	Ø	LKNUM							
	I																														

District record

Salesman master record

SALES record

Look-ahead field on SALES record



IBM

International Business Machines Corporation

RPG CALCULATION SPECIFICATIONS

Form X21-9093
Printed in U.S.A.

Date

Program

Programmer

Punching Instruction: Graphic / Punch

Page (1 2)

Program Identification (75 76 77 78 79 80)

Line	Form Type	Control Level (L0-L9, LR, SR)	Not	Indicator	And Not	Indicator	And Not	Indicator	Factor 1	Operation	Factor 2	Result Field	Field Length	Decimal Positions	Half Adjust (H)	Plus / High 1>2	Minus / Low 1<2	Zero / Equal 1=2	Comments
01	C		N	Ø1					MANNUM	COMP	LKNUM							23	
02	C	*																	
03	C	*																	
04	C	*																	
05	C			23						FORCE	SALES								
06	C																		

If salesman numbers are the same (23 on), force the next SALES record.

If salesman master card (02) or SALES card (03) being processed, check salesman number on the next SALES card.

Figure 14-10. Using Look Ahead to Match Records

Review 14

1. What job situations would require you to alter the order of processing?

2. What does FORCE do?

3. If you must alter the order of processing, how do you control this in your program?

Answers to Review 14

1. A. Match fields cannot be assigned to the files, and you need to:

    a. Process a primary file record followed by a number of secondary file records.

    b. Process a secondary file record only when it matches a primary record.

   B. Match fields are assigned to both files. You need to alter the order of matching record logic to process a primary file record, then matching secondary file records before matching primary file records.

   C. A particular file is to be processed in a job only if certain conditions occur.

2. No action occurs when the specification is encountered. At the beginning of the next program cycle, the next record from the file specified as Factor 2 of the FORCE operation is selected (by being forced) for processing.

3. You can control the order of processing by describing your situation on the Calculation Sheet and conditioning the FORCE statements with indicators, branching, or looping.

CHAPTER ARRAYS 15

CHAPTER 15 DESCRIBES:

Use of arrays and RPG II coding to reference an entire array or individual elements of the arrays.

XFOOT operation code.

LOKUP operation code.

BEFORE READING THIS CHAPTER YOU SHOULD BE ABLE TO DESCRIBE:

Use of and coding for tables.

Exception output.

RPG II object cycle.

OR relationship.

AFTER READING THIS CHAPTER YOU SHOULD BE ABLE TO:

Determine the use of arrays as opposed to the use of tables.

Define an array.

Code problems that reference all fields in an array.

Code problems referencing individual fields in an array.

Define and code the LOKUP operation code with arrays.

Describe data and store it in an array.

INTRODUCTION

An *array* is a continuous series of data fields stored side by side so they can be referenced as a group. Figure 15-1 shows an array of 12 fields containing the total sales for each month of the year. Each field of the array has the same characteristics; that is, each contains data in the same format (alphmeric or numeric), of the same length, and with the same number of decimal positions.

An array is very similar in concept to a table. Both arrays and tables are set up by coding extension specifications. The type of data which you can put in an array is the same as that which you can put in a table. The data to be read into the computer and placed in an array can be punched on cards or written on disk. Otherwise, the data can be produced during the program, as a result of calculations, and then stored in the array. The way in which the data is stored inside the computer is also the same; that is, one field of data immediately follows another. The uses, however, of tables and arrays differ considerably.

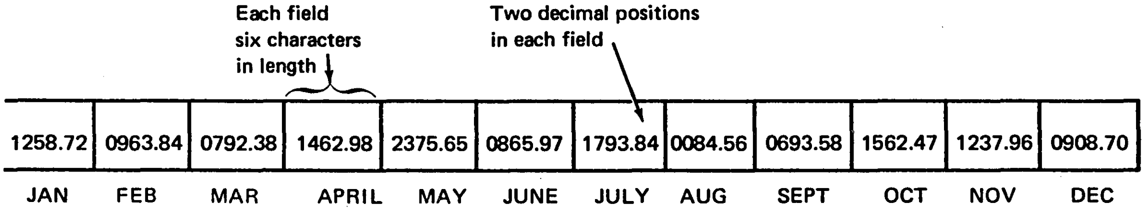
WHEN TO USE AN ARRAY INSTEAD OF A TABLE

Two factors determine when to use an array instead of a table:

- How the data is to be loaded into storage.
- How you want to use the data once it is stored.

In most cases, tables contain constant data such as tax rates, shipping instructions, or discount rates. The constant data is then used for calculations or printing with variable transaction data. Arrays are generally used for variable data and totals which are used independently of the variable transaction data.

You should usually use arrays instead of tables when you want to reference all fields at one time. Arrays can reduce the number of RPG II specifications you must code for a job, as well as the time required to reference the entries.


Each field six characters in length
Two decimal positions in each field

JAN	FEB	MAR	APRIL	MAY	JUNE	JULY	AUG	SEPT	OCT	NOV	DEC
1258.72	0963.84	0792.38	1462.98	2375.65	0865.97	1793.84	0084.56	0693.58	1562.47	1237.96	0908.70

Figure 15-1. 12-Field Numeric Array

DEFINING AN ARRAY

You tell the RPG II compiler to set up an array in the same way that you indicate a table is to be created; that is, by coding extension specifications. One line is coded for each array to be used in a program. As you can see in Figure 15-2, the entries made on the Extension Sheet indicate:

- The name of the array.
- The number of fields in the array.
- The length of each field.
- The number of decimal positions, if any, in a field.

All arrays used in your program must be assigned a name of six characters or less which is entered in columns 27-32. The rules for naming arrays are similar to those for naming tables; an array name can consist of any combination of alphabetic characters and numbers. However, while the first character must be an alphbetic character, an array name *cannot* begin with the letters TAB. This is the way the compiler distinguishes between an array and a table.

Columns 33-35 of the Extension Sheet should be blank when defining an array. An entry is made in these columns only if the group of data items is to be considered a table.

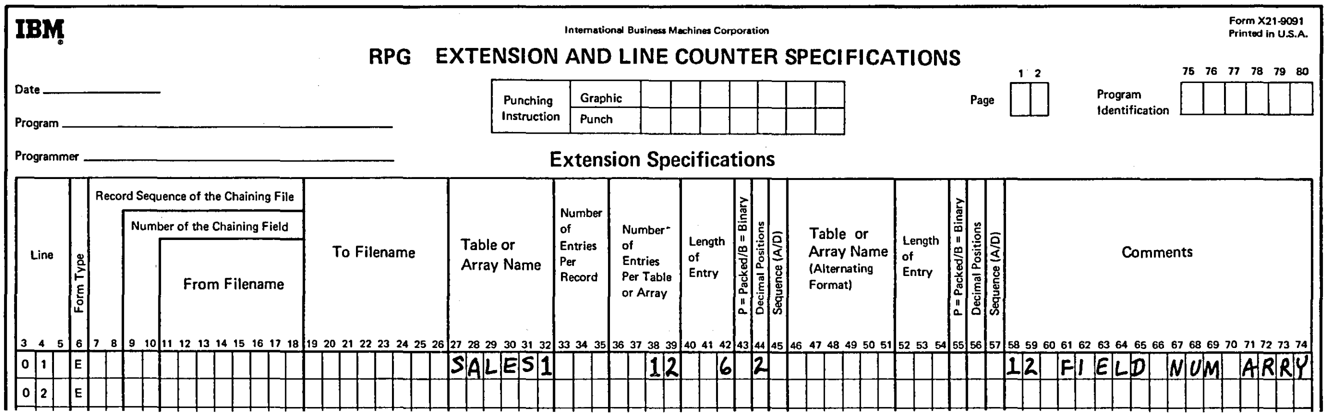
Columns 36-39 are used to enter the number of fields in the array (from 1 to 9999). This number should be entered so that the last digit is in column 39.

The length of each field (number of characters, including blanks) should be specified in columns 40-42, with the number ending in column 42. The length, which must be the same for every field in the array, cannot be greater than 255.

If the fields in an array are numeric, the number (0-9) of digits to the right of the decimal point should be entered in column 44. Even if no decimal positions are present, a zero must be specified if the field is numeric. A blank in column 44 indicates that the fields are to contain alphameric data. Remember, however, that if arithmetic operations are to be performed on the fields, the array must be defined as numeric.

Columns 46 through 57 are not used in defining an array. These columns are used if two related tables are set up in an alternating format on table input records. Two arrays cannot be used in the same way that two related tables are.

The extension specifications only reserve the appropriate space in storage for the array. In a following section, you will learn how data is stored in the array.



IBM

International Business Machines Corporation

Form X21-9091
Printed in U.S.A.

RPG EXTENSION AND LINE COUNTER SPECIFICATIONS

Date ____________

Program ____________

Programmer ____________

Punching Instruction	Graphic							
	Punch							

Page (1 2) □□

Program Identification (75 76 77 78 79 80) □□□□□□

Extension Specifications

Line	Form Type	Record Sequence of the Chaining File	Number of the Chaining Field	From Filename	To Filename	Table or Array Name	Number of Entries Per Record	Number of Entries Per Table or Array	Length of Entry	P = Packed/B = Binary	Decimal Positions	Sequence (A/D)	Table or Array Name (Alternating Format)	Length of Entry	P = Packed/B = Binary	Decimal Positions	Sequence (A/D)	Comments
3 4 5	6	7 8	9 10	11 12 13 14 15 16 17 18	19 20 21 22 23 24 25 26	27 28 29 30 31 32	33 34 35	36 37 38 39	40 41 42	43	44	45	46 47 48 49 50 51	52 53 54	55	56	57	58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74
0 1	E					SALES1		12	6		2							12 FIELD NUM ARRY
0 2	E																	

Figure 15-2. Defining an Array

REFERENCING ALL FIELDS IN AN ARRAY

Suppose a company employs 15 sales clerks whose daily sales are recorded on a punched card (SALES). As Figure 15-3 shows, field 1 contains sales for clerk #1, field 2 for clerk #2, and so on. There is one SALES record for each day. In addition to a daily amount, the company wishes to have a monthly sales total for each clerk. Therefore, at the end of the month, the daily sales amounts for a clerk must be accumulated.

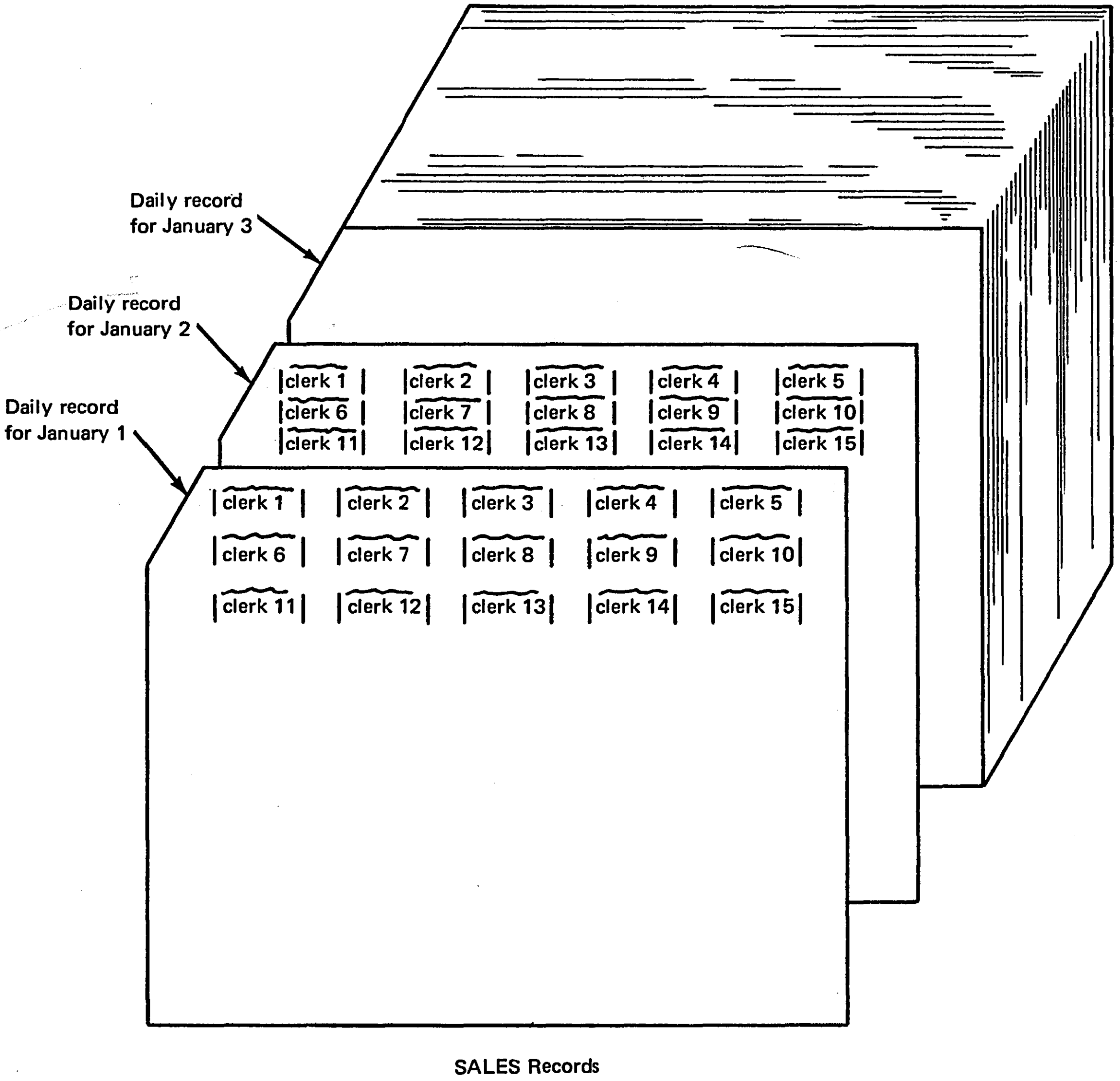


Figure 15-3. SALES Records

As shown in Figure 15-4, an array (MONTH) of 15 fields is set up to contain the monthly totals. The monthly sales record is read and each clerk's total is placed in the appropriate array element. Another array, called DAY, could be set up to contain the 15 sales amounts for any particular day. In this way, as the SALES card for one day is read into the computer, the 15 fields of data would be placed in the array DAY.

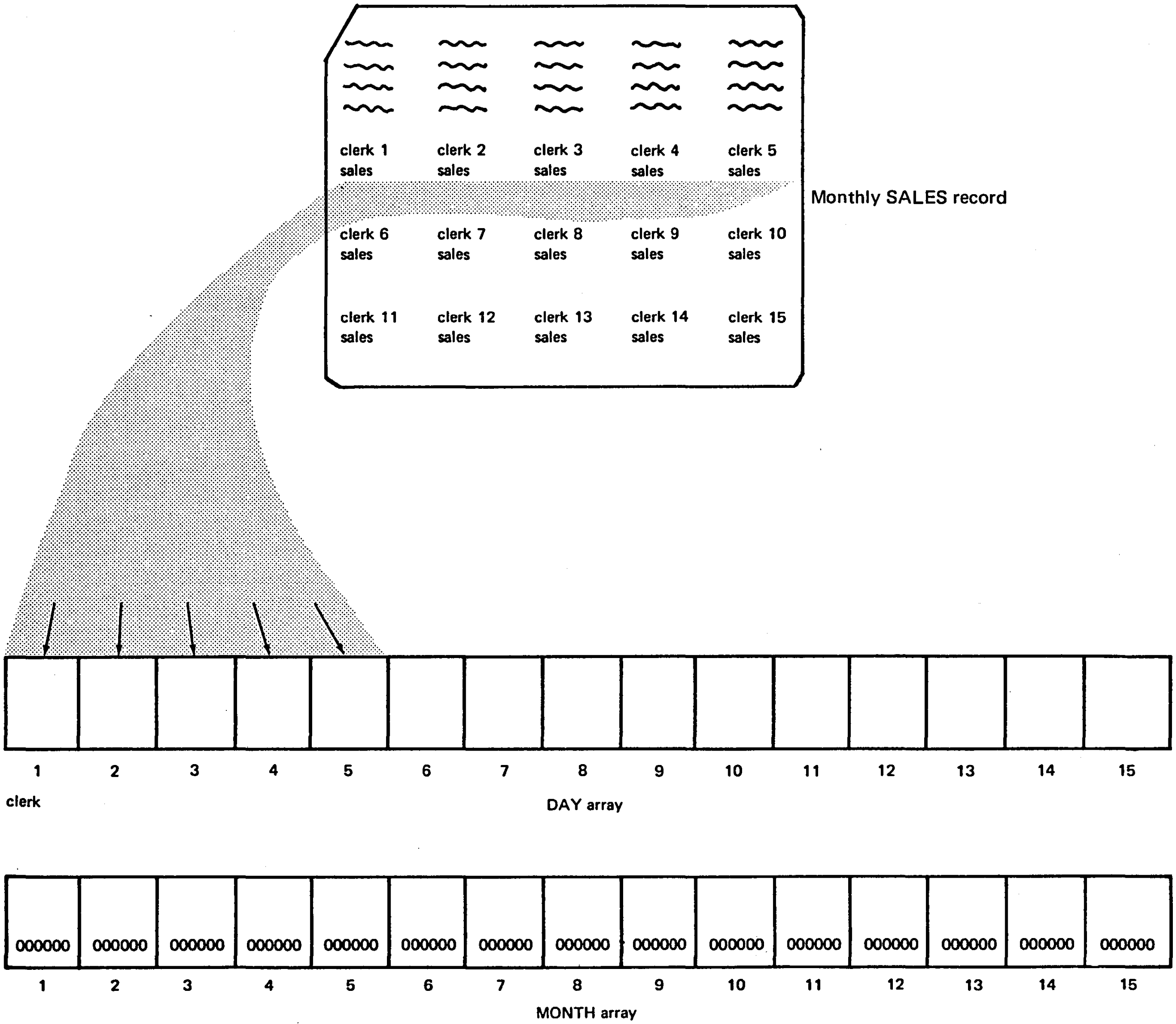


Figure 15-4. Using Arrays to Contain Sales Data

Array to Array Calculations

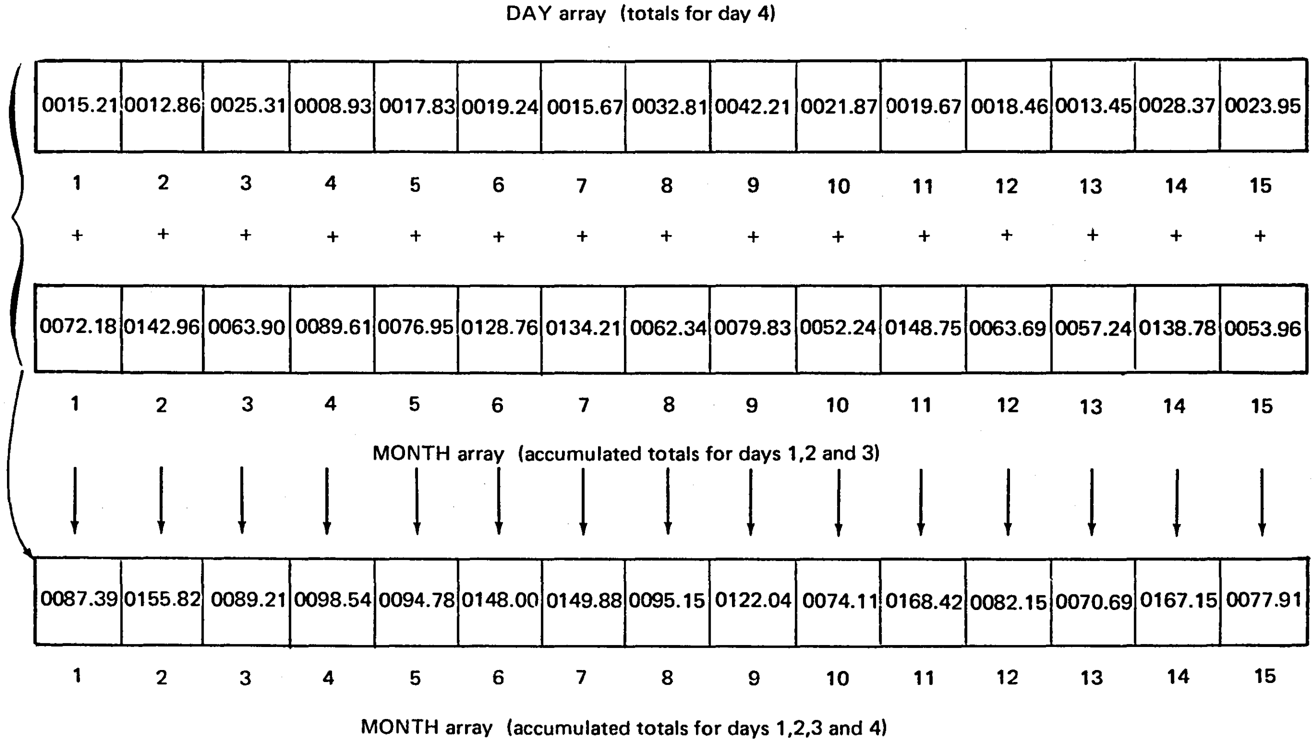
Once the first SALES record is read and the data stored in the DAY array, the 15 fields of DAY are added to the 15 fields of MONTH. In other words, field 1 of DAY is added to field 1 of MONTH, field 2 to field 2, and so on (Figure 15-5).

The 15 accumulated sale amounts (results of the additions) are stored in MONTH. Then, another SALES card is read into the DAY array. The new DAY fields are then added again to the accumulated totals in MONTH.

This method is similar to using two tables and adding an entry from one table to an entry in the other table. However, performing the operations for a table requires more specifications than to do the job using arrays.

With tables, you must reference each element (sales amount for a clerk) separately. First, you must perform a table lookup to find the appropriate sale amount from the day table. Of course, since you do not know the amount of each sale, you cannot search the day table directly. A related table of sales clerk numbers must be set up and searched. Only after you find the appropriate salesclerk entry is the corresponding sale amount in the day table made available. Then you must lookup the corresponding element of the month table. At this point, use of the table names in calculations or output would finally refer to each of the entries looked up. An addition operation would then be required to add the two entries and place the result in the month table. After all this, you have accumulated a total for only one of the sales clerks.

To repeat the same procedure 14 more times for the other sales clerks' entries, the program must read 14 cards and go through 14 program cycles. This occurs when you use a table name in specifications. The name refers to only one element, the entry just looked up.



DAY array (totals for day 4)

0015.21	0012.86	0025.31	0008.93	0017.83	0019.24	0015.67	0032.81	0042.21	0021.87	0019.67	0018.46	0013.45	0028.37	0023.95
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
+	+	+	+	+	+	+	+	+	+	+	+	+	+	+

0072.18	0142.96	0063.90	0089.61	0076.95	0128.76	0134.21	0062.34	0079.83	0052.24	0148.75	0063.69	0057.24	0138.78	0053.96
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

MONTH array (accumulated totals for days 1,2 and 3)

0087.39	0155.82	0089.21	0098.54	0094.78	0148.00	0149.88	0095.15	0122.04	0074.11	0168.42	0082.15	0070.69	0167.15	0077.91
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

MONTH array (accumulated totals for days 1,2,3 and 4)

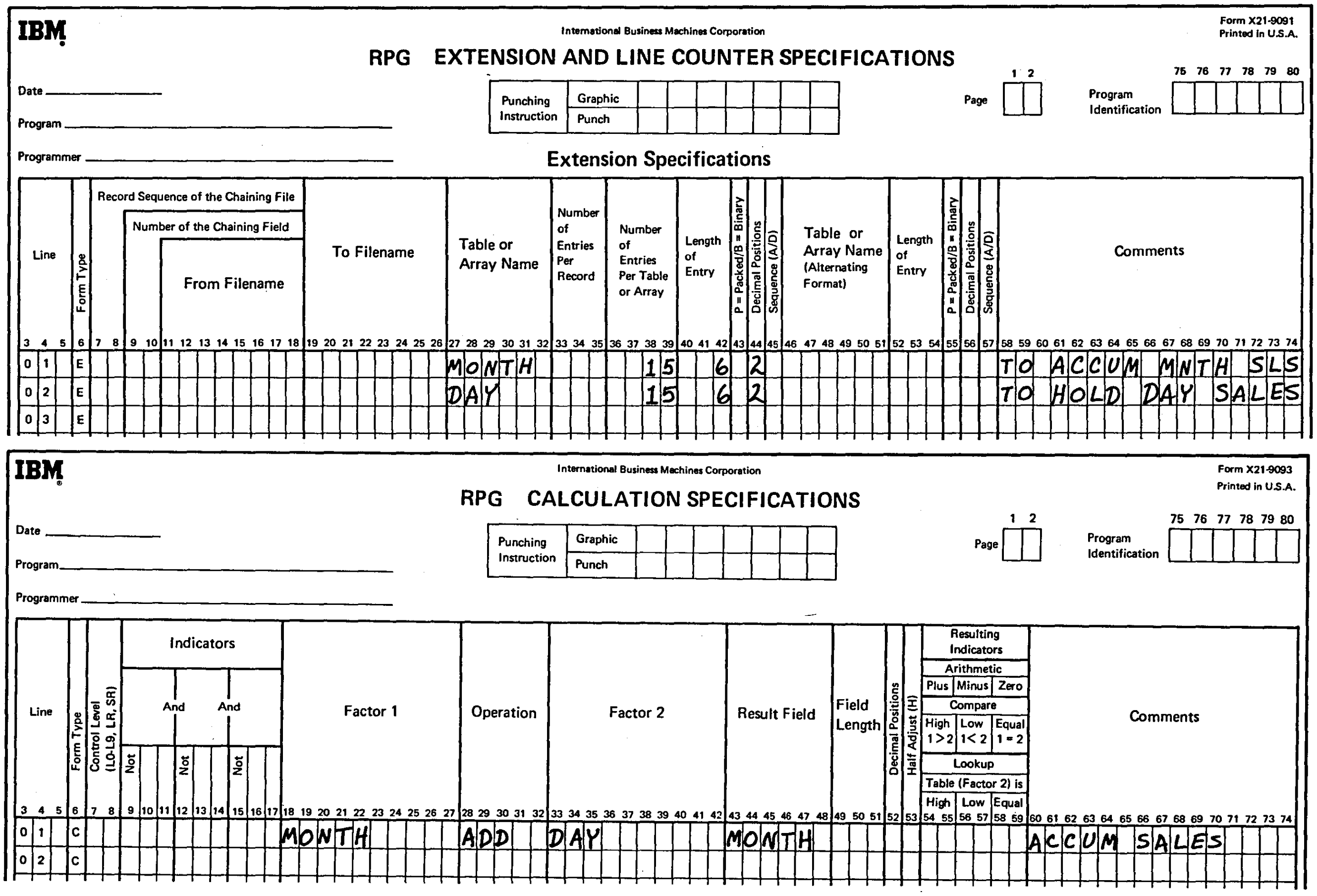
Figure 15-5. Adding One Array to Another Array

On the other hand, if you have defined your groups of data as arrays rather than tables, only one calculation specification is necessary. The name of an array actually refers to all of the fields in that array. Adding the array DAY to the array MONTH causes every field of one array to be added to corresponding fields of the other array (1 to 1, 2 to 2, 3 to 3, etc.). Since the MONTH array is specified under Result Field, the result of each addition is placed back into the appropriate field of MONTH (Figure 15-6).

Notice on the Calculation Sheet in Figure 15-6 that no resulting indicators have been specified for this arithmetic operation. When an array name is specified in a calculation, the operation is performed on every field of the array. Therefore, there are a multiple number of results; in this case, 15 sales totals. A resulting indicator can indicate the condition of only a single result. Thus, resulting indicators are usually not used when referencing an entire array. There are two exceptions when resulting indicators can be used, as explained under *Adding All Fields Within An Array* and *Searching An Array For A Particular Field.*

Operations Which Can be Performed on Arrays

As mentioned, an operation to be performed on an array is performed for every field in the array. A result is then produced for each field operated on. For this reason, certain operations cannot be performed on arrays, because the results have no meaning. The operation codes COMP (compare), TESTZ (test zone), and MVR (move remainder) should not be used.



IBM International Business Machines Corporation Form X21-9091 Printed in U.S.A.

RPG EXTENSION AND LINE COUNTER SPECIFICATIONS

Date ______ Program ______ Programmer ______

Punching Instruction: Graphic / Punch

Page 1 2 Program Identification 75 76 77 78 79 80

Extension Specifications

Line	Form Type	Record Sequence of the Chaining File	Number of the Chaining Field	From Filename	To Filename	Table or Array Name	Number of Entries Per Record	Number of Entries Per Table or Array	Length of Entry	P = Packed/B = Binary	Decimal Positions	Sequence (A/D)	Table or Array Name (Alternating Format)	Length of Entry	P = Packed/B = Binary	Decimal Positions	Sequence (A/D)	Comments
3 4 5	6	7 8	9 10	11 12 13 14 15 16 17 18	19 20 21 22 23 24 25 26	27 28 29 30 31 32	33 34 35	36 37 38 39	40 41 42	43	44	45	46 47 48 49 50 51	52 53 54	55	56	57	58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74
0 1	E					MONTH		15	6		2							TO ACCUM MNTH SLS
0 2	E					DAY		15	6		2							TO HOLD DAY SALES
0 3	E																	

IBM International Business Machines Corporation Form X21-9093 Printed in U.S.A.

RPG CALCULATION SPECIFICATIONS

Date ______ Program ______ Programmer ______

Punching Instruction: Graphic / Punch

Page 1 2 Program Identification 75 76 77 78 79 80

Line	Form Type	Control Level (L0-L9, LR, SR)	Indicators: Not		And: Not		And: Not		Factor 1	Operation	Factor 2	Result Field	Field Length	Decimal Positions	Half Adjust (H)	Resulting Indicators — Arithmetic: Plus / Compare: High 1>2 / Lookup Table (Factor 2) is: High	Minus / Low 1<2 / Low	Zero / Equal 1=2 / Equal	Comments
3 4 5	6	7 8	9	10 11	12	13 14	15	16 17	18 19 20 21 22 23 24 25 26 27	28 29 30 31 32	33 34 35 36 37 38 39 40 41 42	43 44 45 46 47 48	49 50 51	52	53	54 55	56 57	58 59	60 61 62 63 64 65 66 67 68 69 70 71 72 73 74
0 1	C								MONTH	ADD	DAY	MONTH							ACCUM SALES
0 2	C																		

Figure 15-6. Referencing All Fields of an Array

Performing Operations on Arrays of Different Lengths

In this last example, all arrays used in an operation were of the same length; Factor 1, Factor 2, and the result array each contained 15 fields. Thus the operations were carried out until all fields were processed.

Suppose, as shown in Figure 15-7, that DAY only contains 12 while the MONTH array contains 15 fields. In such a case, the operations are performed only until the last field in the shortest array has been processed. Thus, the 12 fields of DAY are added to the first 12 fields of MONTH, and the 12 results are placed in the first 12 fields of MONTH. The remaining three fields of the result field (MONTH) remain unchanged. Likewise, if the result array is shorter than any of the factors (arrays), the operation is repeated only for the number of fields in the shortest (result) array.

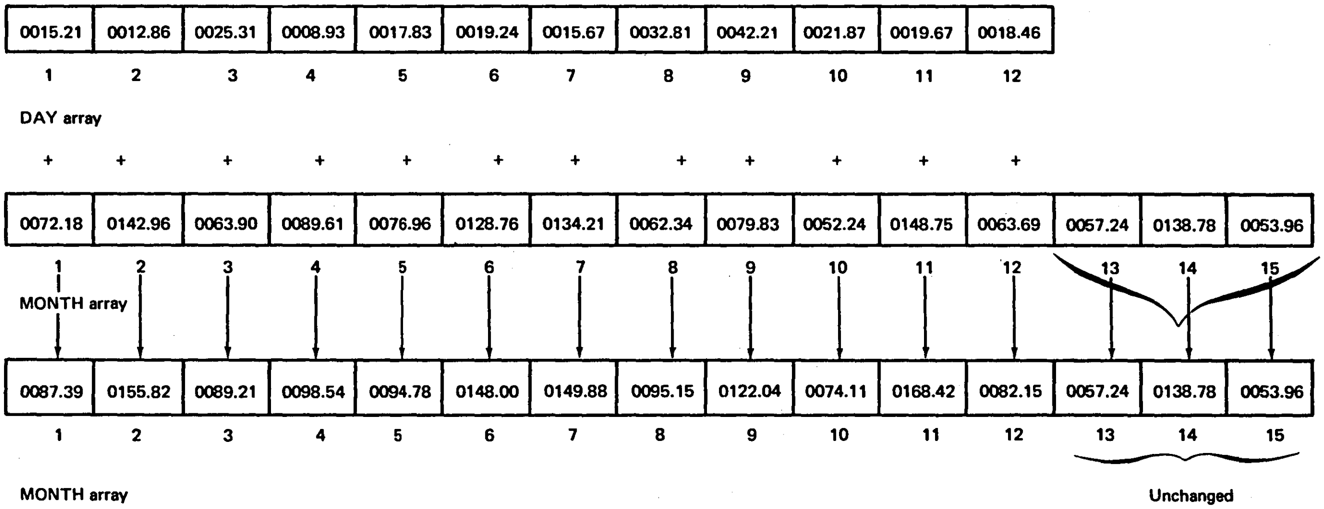


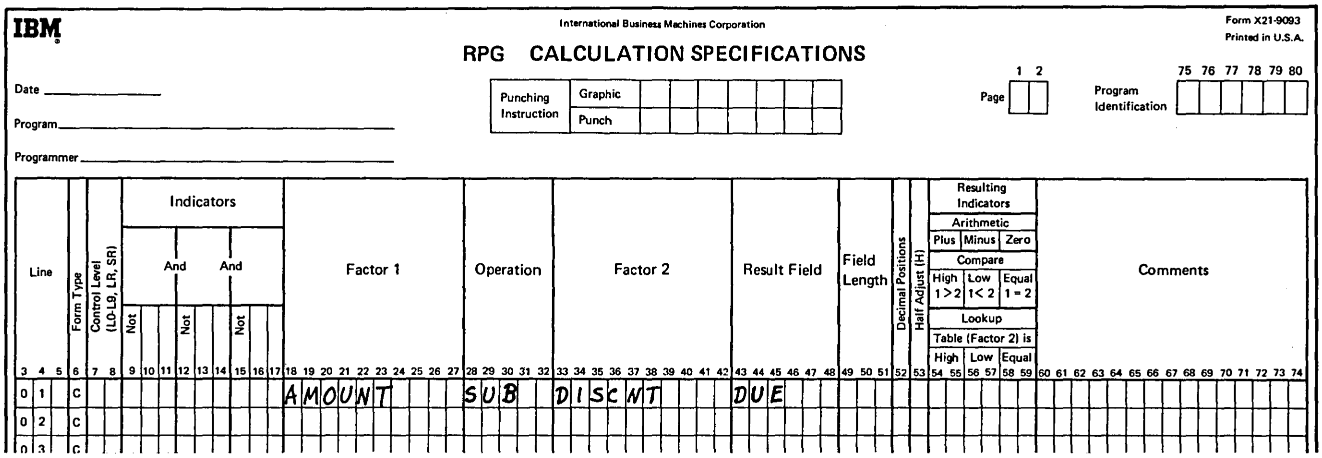
Figure 15-7. Operations on Arrays of Different Lengths

Calculations Using Arrays and Single Fields (or Constants)

Another way in which you can perform calculations on an entire array is by adding (or multiplying, etc.) the same value to every field in the array. For example, suppose the sales clerks are to receive a commission of 10 percent of their sales, to be paid at the end of the month. After all daily sales have been accumulated into the MONTH array, you want to multiply each of the 15 fields in MONTH by the value .10 and to place the commission amounts in another 15-field array called COMMIS.

To perform this job, it is not necessary to set up a 15-field array for the commission rates, with each field containing the value .10. In an array operation, when one of the factors is a field (containing a value) or a constant, the operation is performed using the same field or constant on every field in the array.

You can also use a field or constant as both factors to place the same result in every field of an array. The calculation specifications in Figure 15-8 show the single field named DISCNT being subtracted from the single field AMOUNT, with the result placed in a 5-field array named DUE. The value (017) in DISCNT is subtracted from the value (243) in AMOUNT, and the result (226) is placed in the first field of DUE. The single specification then causes the operation to be performed four more times until the result (226) has been placed in each remaining field of the DUE array.



IBM — International Business Machines Corporation — Form X21-9093 Printed in U.S.A.

RPG CALCULATION SPECIFICATIONS

Date ______ Program ______ Programmer ______

Punching Instruction: Graphic / Punch — Page 1 2 — Program Identification 75 76 77 78 79 80

Line	Form Type	Control Level (L0-L9, LR, SR)	Indicators (Not, And, Not, And, Not)	Factor 1	Operation	Factor 2	Result Field	Field Length	Decimal Positions	Half Adjust (H)	Resulting Indicators	Comments
01	C			AMOUNT	SUB	DISCNT	DUE					
02	C											
03	C											

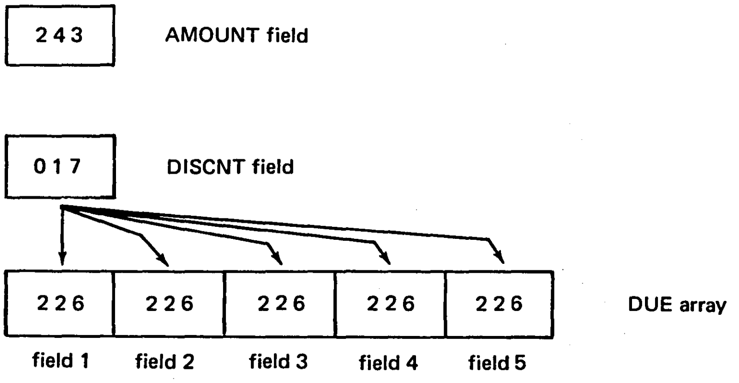


Figure 15-8. Storing the Same Data in All Array Fields

Adding All Fields Within An Array

In the job to accumulate a monthly sales total for each clerk, the amounts that each clerk sold was determined. In addition, the company also wants to know the total of all sales each day.

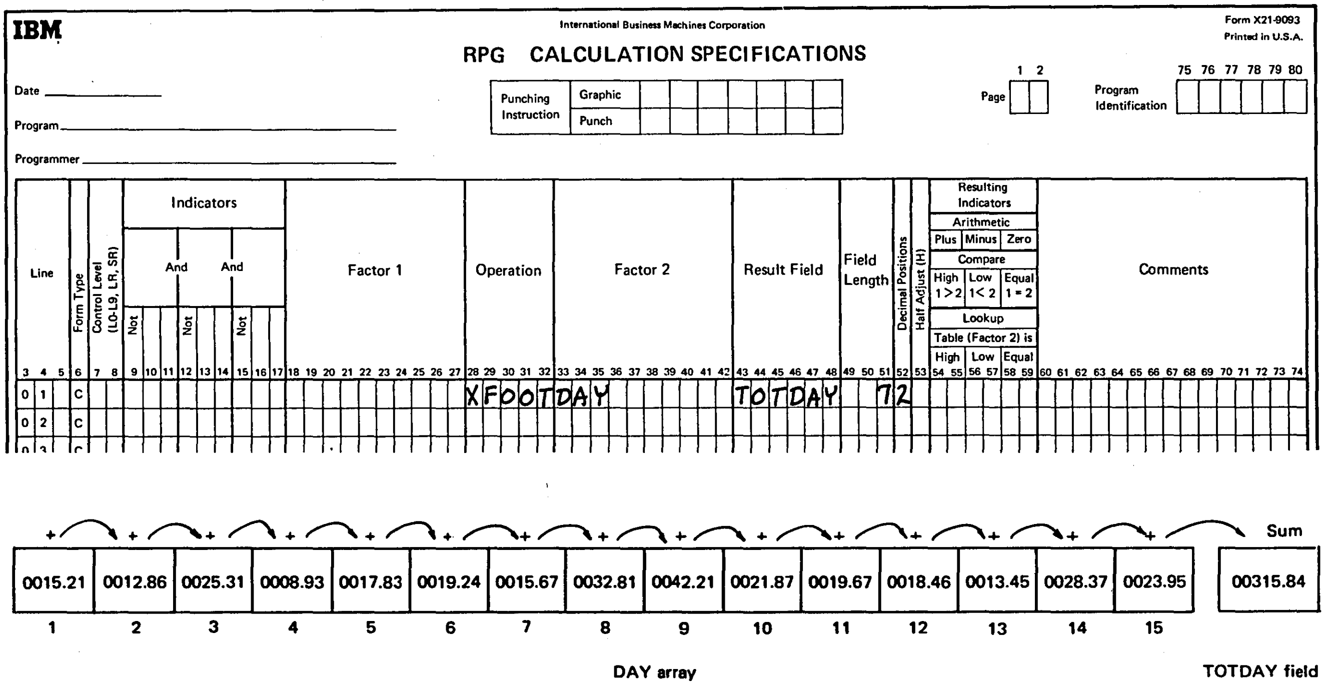
As mentioned before, each clerk's daily sales are stored in a separate field of a 15-field array named DAY. To obtain a total of all sales for the day, you must add together the contents of all fields in the array. The sum can then be placed in a single field.

The XFOOT operation code (Figure 15-9, columns 28-32) tells the computer to sum the contents of every field in the array named in Factor 2. Columns 18 through 27 (Factor 1) of the Calculation Sheet are left blank since the XFOOT operation involves only the values in one array. The sum of the DAY array fields is then placed in the single field named in columns 43 through 48 (Result Field).

In all other types of array calculations, multiple results are produced in accordance with the number of fields in an array. However, performing an XFOOT operation provides only one result, the total of all fields. For this reason, XFOOT is the only operation referencing an entire array in which you specify a single field name rather than an array name, under Result Field. Furthermore, since there is only one result, a resulting indicator may be assigned in columns 54-59 to determine if the sum is plus, minus, or zero. In this case, a resulting indicator was not specified (Figure 15-9) because the sales amounts will always be positive.

Output of An Entire Array

For many jobs, you will want to put out array data as well as perform calculations on the fields. The output of all fields in an array is accomplished as easily as referencing an entire array in calculations, that is, by specifying the name of the array. By specifying the array name under Field Name (columns 32-37) on the Output Sheet, all fields within the name array will be punched, printed, or written on the indicated output file (Figure 15-10).



IBM International Business Machines Corporation

RPG CALCULATION SPECIFICATIONS

Form X21-9093
Printed in U.S.A.

Date
Program
Programmer

Line	Form Type	Control Level	Indicators	Factor 1	Operation	Factor 2	Result Field	Field Length	Decimal Positions	Half Adjust (H)	Resulting Indicators	Comments
01	C				XFOOT	DAY	TOTDAY	7	2			
02	C											
03	C											

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Sum
0015.21	0012.86	0025.31	0008.93	0017.83	0019.24	0015.67	0032.81	0042.21	0021.87	0019.67	0018.46	0013.45	0028.37	0023.95	00315.84

Figure 15-9. Adding All Fields of an Array

Any output conditioning indicators specified in columns 23 through 31 of the Output Sheet determine when during the program the array fields will be printed or punched. If no indicators are specified, the entire array is printed or punched every time a record is processed. Indicators can be specified to put out array data during detail cycles or at total time. You may want to put out array data at total time by customer or inventory item, for example, to be used as input to subsequent update runs.

RPG II determines where the array data is to be put out on a card, printer file, or disk by the end position column you specify in columns 40 through 43 of the Output Sheet. The array fields are put out such that the last field of the named array ends in the column indicated. Note, however, that if all fields in the array cannot be put out on one output record, the array fields must be referenced separately on the Output Sheet. Output of individual fields will be discussed later.

The output of an entire array requires only one specification. An entire array can be printed or punched at any time during the run or at the end of job, depending on how the output specification is conditioned.

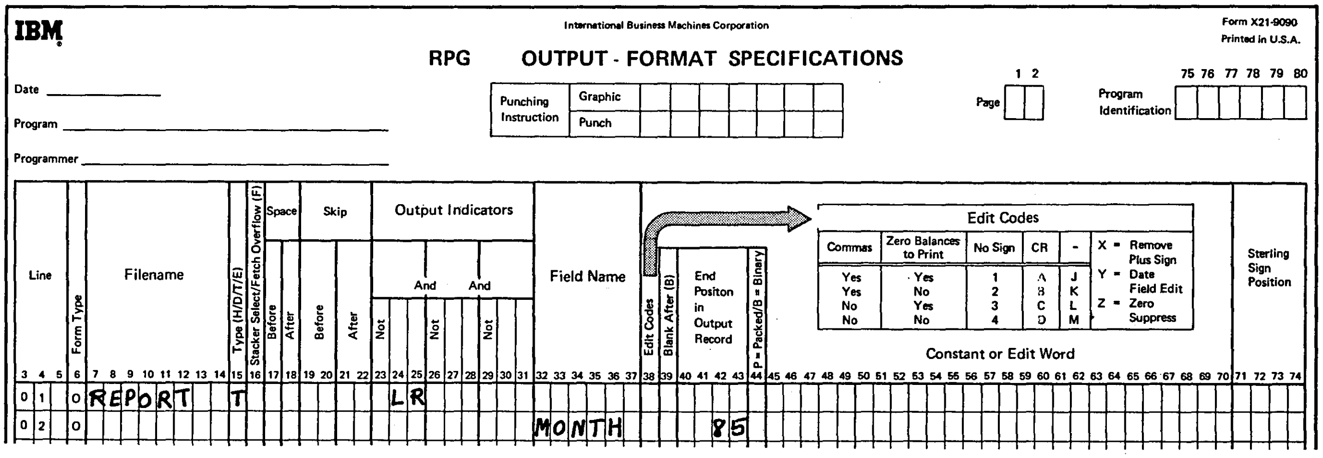
You must specify how you want the data fields to appear on the output record. Alphameric fields appear on an output record just as they appear in storage; however, numeric array fields may be edited or unedited. If no editing is specified, the fields will be printed or punched just as they appear in storage, with the last field ending in the end position column of the output file. In other words, one field will immediately follow another with no punctuation and no blanks between fields.

Usually, printed array output is easier to read and has a better appearance if edit codes or edit words are used to punctuate the data and insert spaces between fields. If punched output or disk output is desired, generally the array output records are used as input the next time the job is run. Therefore, editing is usually not specified, so the fields will be in the appropriate format to be used as input.

Editing affects the column which is specified as the end position of the output record. If each field in a 5-field array contains seven characters, 35 positions would be necessary to output the entire array in unedited form. On the other hand, if punctuation and blanks are inserted for each field, the number of positions required increases. When specifying an end position, you must allow enough positions to output all edited fields.

Regardless of whether editing is specified or not, when output of an entire array is performed, every field of that array is put out in the same format. If an edit code or edit word is specified, every field is edited in the same way. Since all fields of an array contain the same type of information, ordinarily you want the fields punctuated in the same way. If, however, one field must be edited differently from another field in the same array, you must put out the fields separately. The means of referencing individual fields of an array is discussed later in this section.

When an edit code is specified in column 38 of the Output Sheet, every field of the named array will be punctuated accordingly. Furthermore, any edit code specified for an entire array also causes two blank spaces to be inserted between each field. The insertion of blanks is taken into consideration by the program so that the last field ends in the end position specified.



IBM

International Business Machines Corporation

Form X21-9090
Printed in U.S.A.

RPG OUTPUT - FORMAT SPECIFICATIONS

Date
Program
Programmer

Punching Instruction: Graphic / Punch

Page 1 2

Program Identification 75 76 77 78 79 80

Line (3-5)	Form Type (6)	Filename (7-14)	Type (H/D/T/E) (15)	Output Indicators (23-31)	Field Name (32-37)	End Position in Output Record (40-43)
01	O	REPORT	T	LR		
02	O				MONTH	85

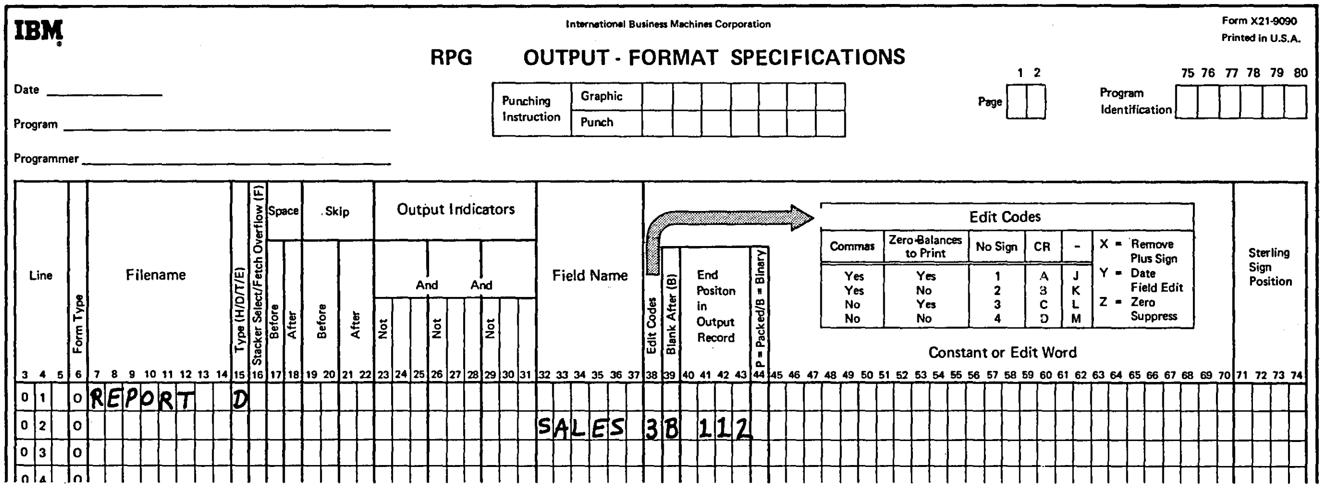
Edit Codes

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Figure 15-10. Output of an Entire Array

As shown in Figure 15-11, the edit code *3* causes all five fields of the SALES array to be printed with decimal points inserted, leading zeros suppressed, and zero balances present. In addition, two blanks are automatically printed before each field since an edit code was specified.



IBM

International Business Machines Corporation

Form X21-9090
Printed in U.S.A.

RPG OUTPUT - FORMAT SPECIFICATIONS

Date ____________

Program ____________

Programmer ____________

Punching Instruction: Graphic / Punch

Page 1 2

Program Identification 75 76 77 78 79 80

Edit Codes

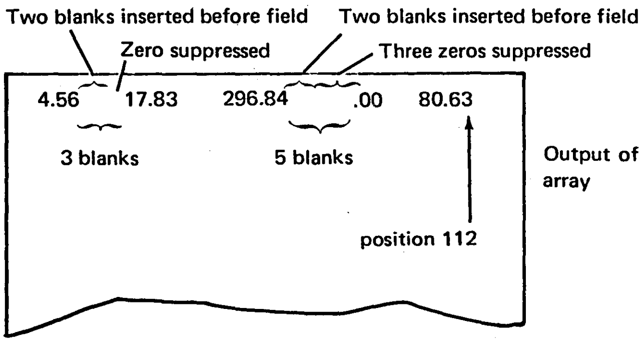
Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Line (3-5)	Form Type (6)	Filename (7-14)	Type (H/D/T/E) (15)	Field Name (32-37)	Edit Codes (38)	Blank After (B) (39)	End Position in Output Record (40-43)
01	O	REPORT	D				
02	O			SALES	3	B	112
03	O						
04	O						

00456	01783	29684	00000	08063

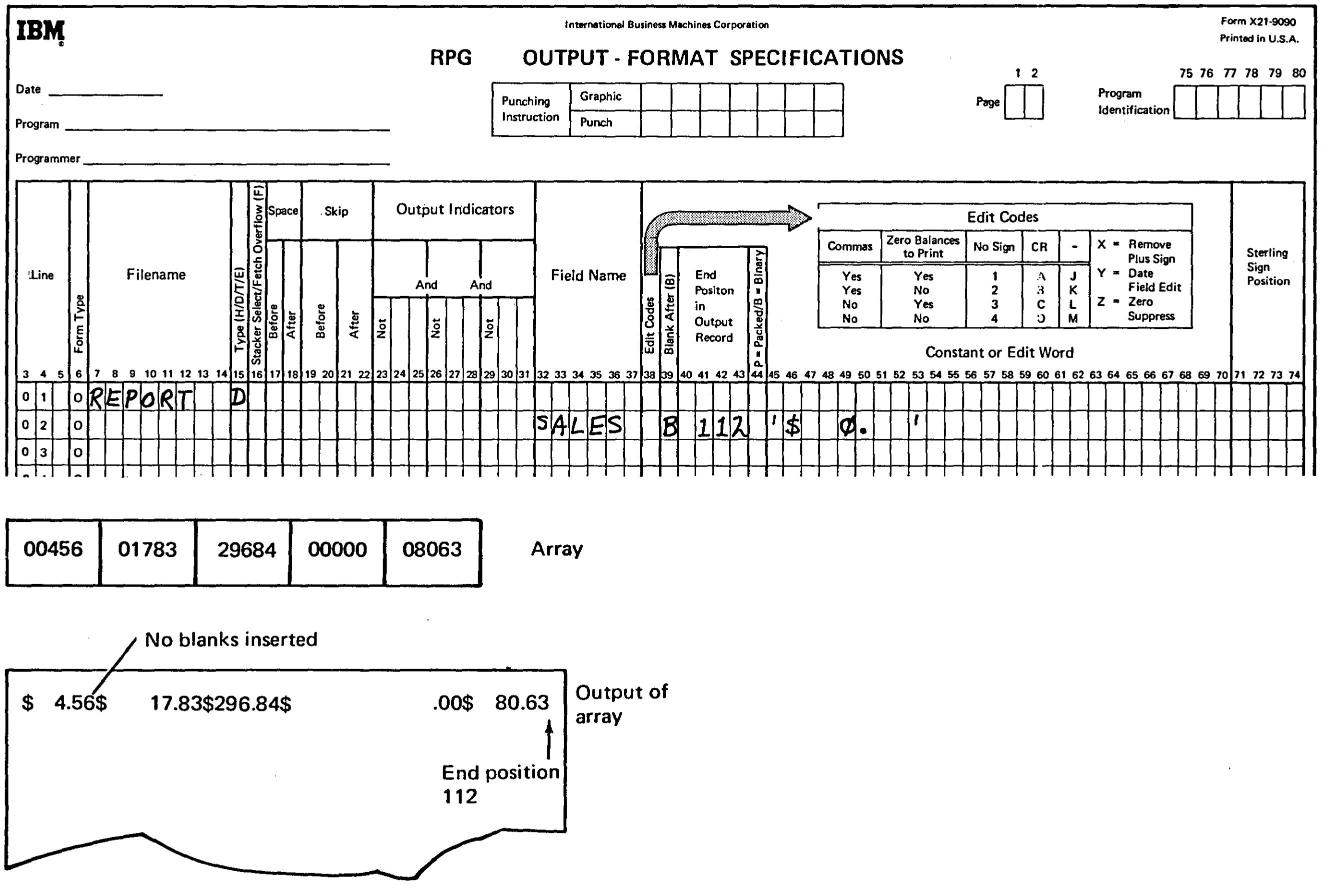
Array



Output of array

Figure 15-11. Output of an Entire Array With Edit Codes

If no edit code specifies exactly how you want the array fields to be edited, you can specify the punctuation by using an edit word (columns 45-70). In this way, you can edit array fields with dollar signs, zero suppression, blanks, constant words, or any combination of punctuation desired. When edit words are used, all punctuation must be specified. Unlike edit codes, edit words do not cause two blanks to be automatically inserted in the output record before each array field. Figure 15-12 shows an edit word specified without blanks; one field of SALES immediately follows the next on the output record. Any extra blanks which are to appear must be indicated in the edit word by an &. Notice, in Figure 15-13, that the two blanks specified are to be printed as part of every field. Thus, the second blank following the last field will be the character which ends in the end position column. Notice that an additional two columns have been allowed for each field (five fields). The end position column has thus been increased by ten over that in Figure 15-12.



IBM International Business Machines Corporation Form X21-9090 Printed in U.S.A.

RPG OUTPUT - FORMAT SPECIFICATIONS

Date ____ Program ____ Programmer ____

Punching Instruction: Graphic / Punch Page 1 2 Program Identification 75 76 77 78 79 80

Line (3-5)	Form Type (6)	Filename (7-14)	Type (H/D/T/E) (15)	Field Name (32-37)	Edit Codes (38)	Blank After (B) (39)	End Position in Output Record (40-43)	P = Packed/B = Binary (44)	Constant or Edit Word (45-70)	Sterling Sign Position (71-74)
01	O	REPORT	D							
02	O			SALES		B	112		'$ Ø. '	
03	O									

Edit Codes

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

00456	01783	29684	00000	08063

Array

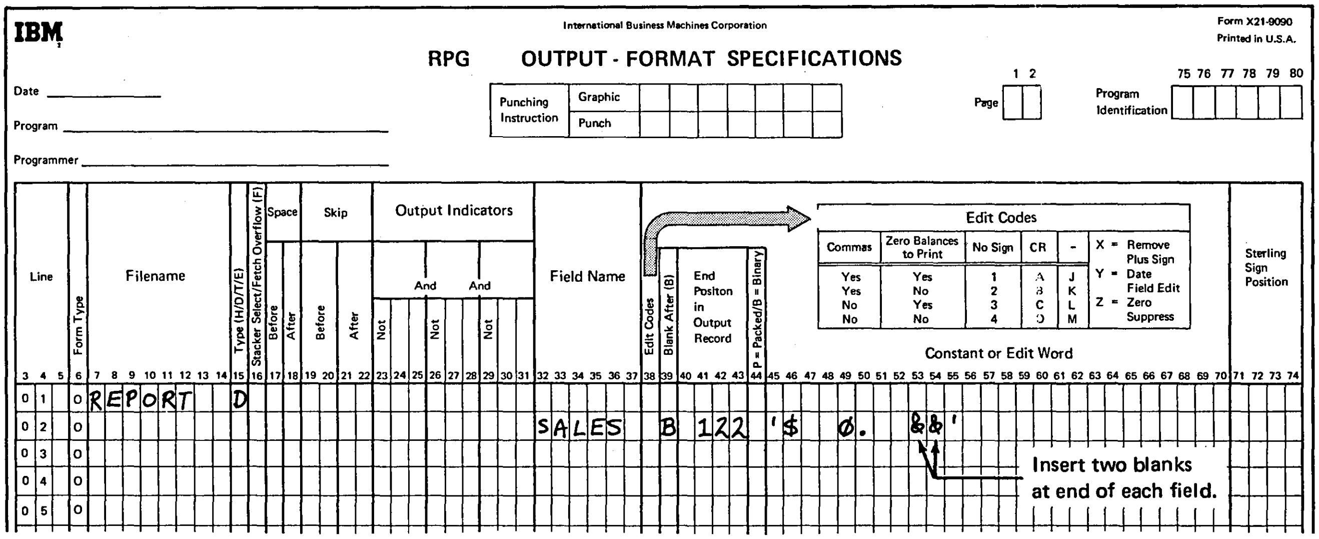
No blanks inserted

```
$  4.56$     17.83$296.84$               .00$  80.63
```

Output of array

End position 112

Figure 15-12. Output of an Entire Array With Edit Words



IBM

International Business Machines Corporation

Form X21-9090
Printed in U.S.A.

RPG OUTPUT - FORMAT SPECIFICATIONS

Date ____________

Program ____________

Programmer ____________

Punching Instruction: Graphic / Punch

Page 1 2

Program Identification 75 76 77 78 79 80

Edit Codes					
Commas	Zero Balances to Print	No Sign	CR	-	X = Remove Plus Sign Y = Date Field Edit Z = Zero Suppress
Yes	Yes	1	A	J	
Yes	No	2	B	K	
No	Yes	3	C	L	
No	No	4	D	M	

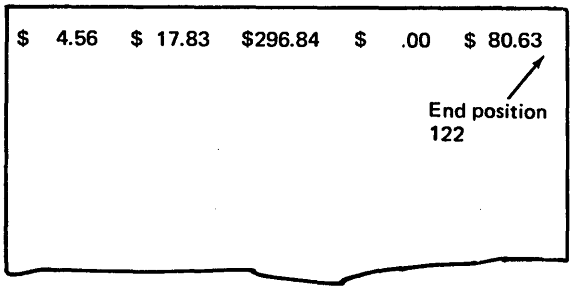
Line (3-5)	Form Type (6)	Filename (7-14)	Type (H/D/T/E) (15)	Stacker Select/Fetch Overflow (F) (16)	Space Before (17)	Space After (18)	Skip Before (19-20)	Skip After (21-22)	Output Indicators (23-31)	Field Name (32-37)	Edit Codes (38)	Blank After (B) (39)	End Position in Output Record (40-43)	P = Packed/B = Binary (44)	Constant or Edit Word (45-70)	Sterling Sign Position (71-74)
0 1	O	REPORT	D													
0 2	O									SALES		B	122		'$ Ø. &&'	
0 3	O															
0 4	O															
0 5	O															

Insert two blanks at end of each field.

00456	01783	29684	00000	08063

Array

SALES



```
$   4.56    $  17.83    $296.84    $    .00    $  80.63
```

End position 122

Output of array

Figure 15-13. Editing Every Field of an Array

Accumulating Groups of Totals

As you have seen, arrays can be used to accumulate a total. In a previous example, fields containing daily sales were added to obtain monthly totals, which were stored in the MONTH array.

To carry this concept further, one of the most common uses of arrays is accumulating more than one group of totals. Such a procedure is called *rolling of totals,* since one total is used to obtain a greater total, which is then used to calculate an even larger total, and so on. Each total is *rolled into* or accumulated into the next total.

Figure 15-14 shows the organization of the Nelson Company. The company's two regions are each divided into three branches, which are in turn made up of three to six stores each.

Company sales data is recorded on cards, as shown in Figure 15-15. For each store there is a separate record providing the 12 sales amounts for each month of the year. The sales records are organized such that stores are grouped within a branch and branches grouped within a region. Three one-column fields on each record identify a card with a particular store, a particular branch, and a particular region.

A sales report must be produced showing the monthly sales for each store, for each branch, for all branches within a region, and for both regions (the total monthly sales of the entire company). The report, a series of accumulated totals, should look like the one in Figure 15-16.

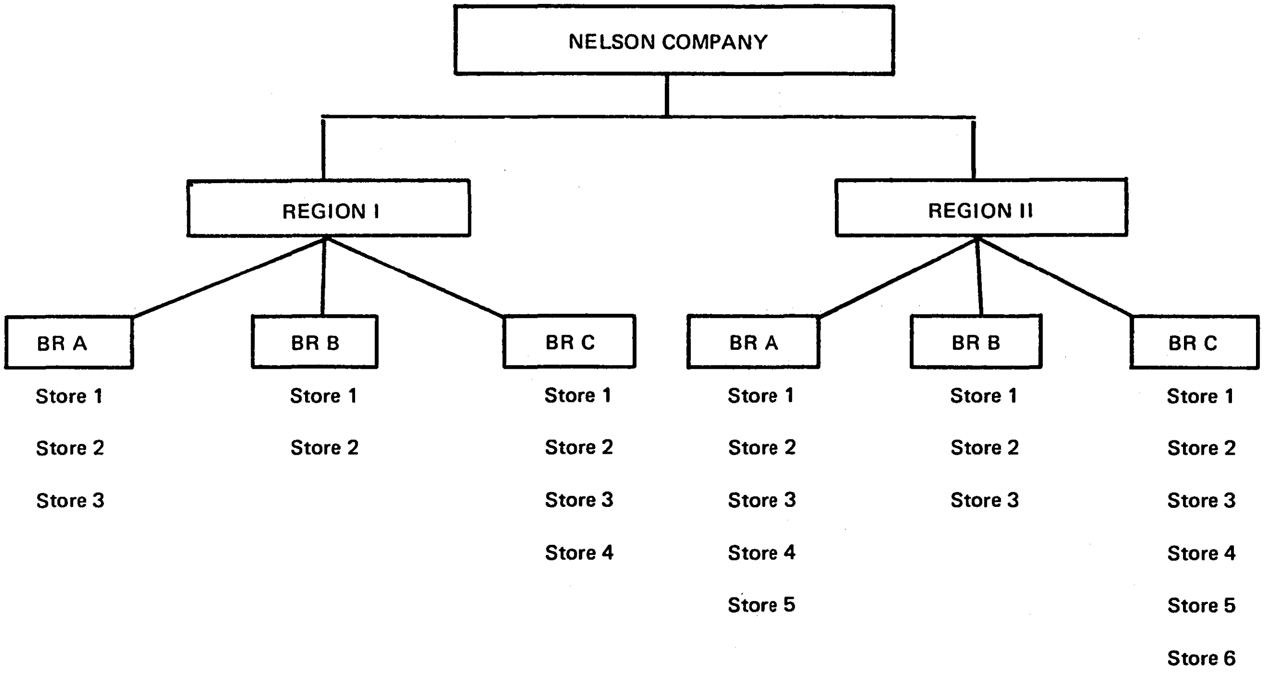


Figure 15-14. Company Organization by Groups

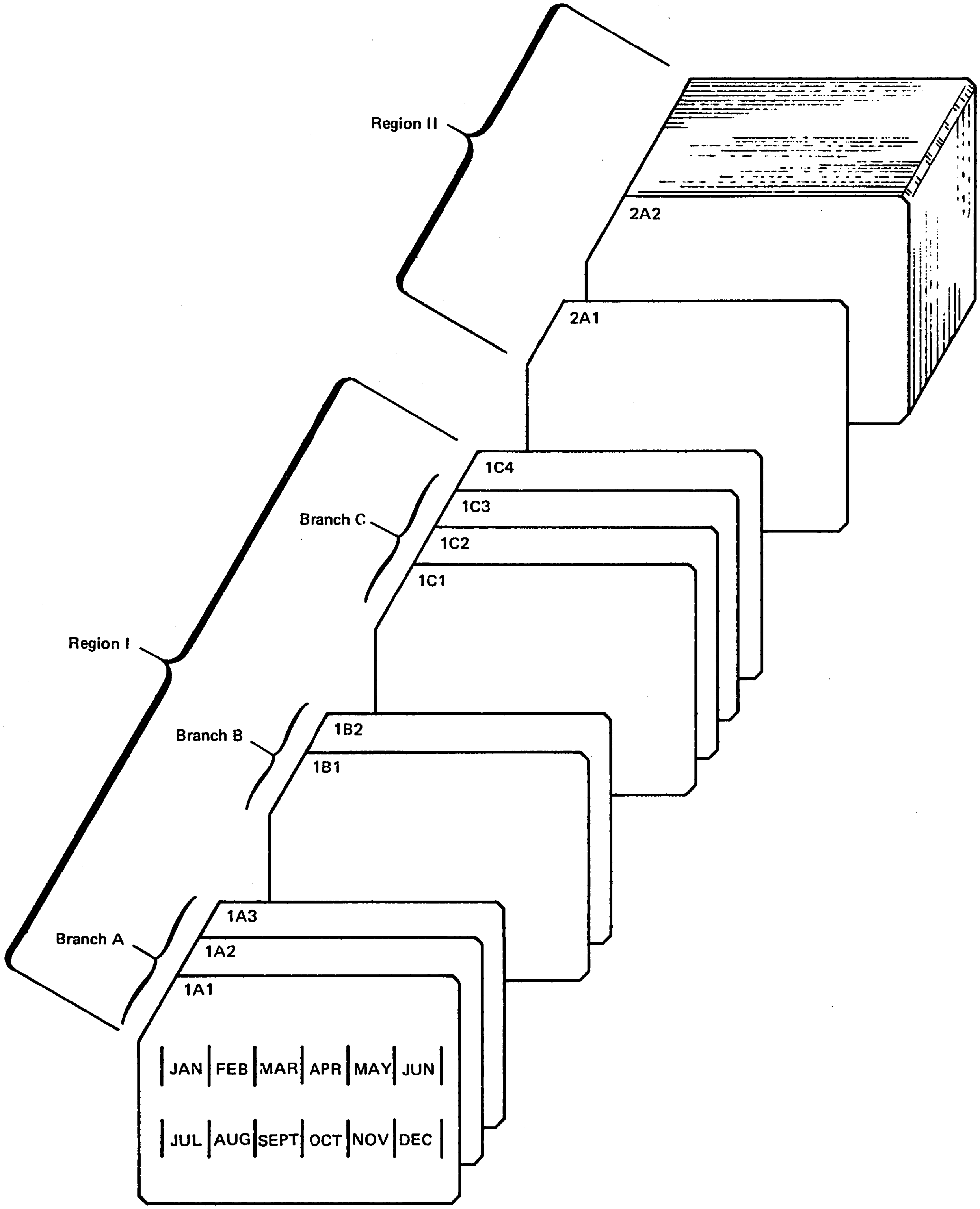
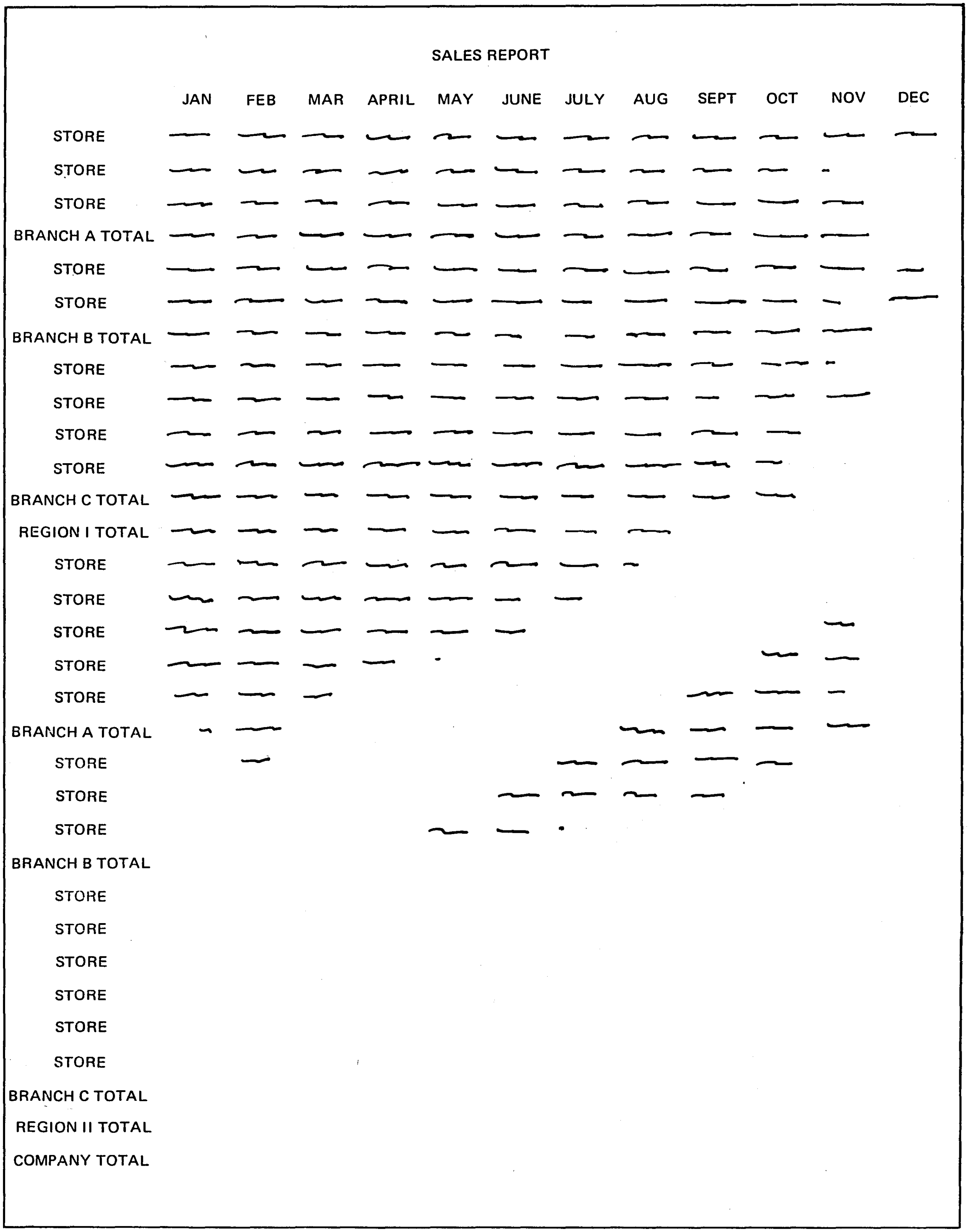


Figure 15-15. Sales Records Organized by Groups Within Groups



SALES REPORT

JAN FEB MAR APRIL MAY JUNE JULY AUG SEPT OCT NOV DEC

STORE
STORE
STORE
BRANCH A TOTAL
STORE
STORE
BRANCH B TOTAL
STORE
STORE
STORE
STORE
BRANCH C TOTAL
REGION I TOTAL
STORE
STORE
STORE
STORE
STORE
BRANCH A TOTAL
STORE
STORE
STORE
BRANCH B TOTAL
STORE
STORE
STORE
STORE
STORE
STORE
BRANCH C TOTAL
REGION II TOTAL
COMPANY TOTAL

Figure 15-16. Sales Report by Groups Within Groups

To produce the report, four arrays of 12 fields each should be set up, as shown in Figure 15-17. The first array, STR, will be used to hold the 12 sales amounts read from the sales records. The other three arrays will be used to accumulate the necessary totals for each branch, each region, and the entire company.

In general, this job should accumulate store totals into the BRNCH array, branch totals into the REG, array, and region totals into the COMP array. Thus, the specifications must perform two functions:

- Add all fields of one array to all fields of another array.
- Print all fields of each array.

To have the program produce the correct totals, you must specify that one array is to be added to another array and printed. To do this, the two fields which identify a record with a particular branch and region should be specified as control fields. A change in the branch (or region) control field will cause a control break, indicating the records for all stores in a particular branch (or region) have been processed.

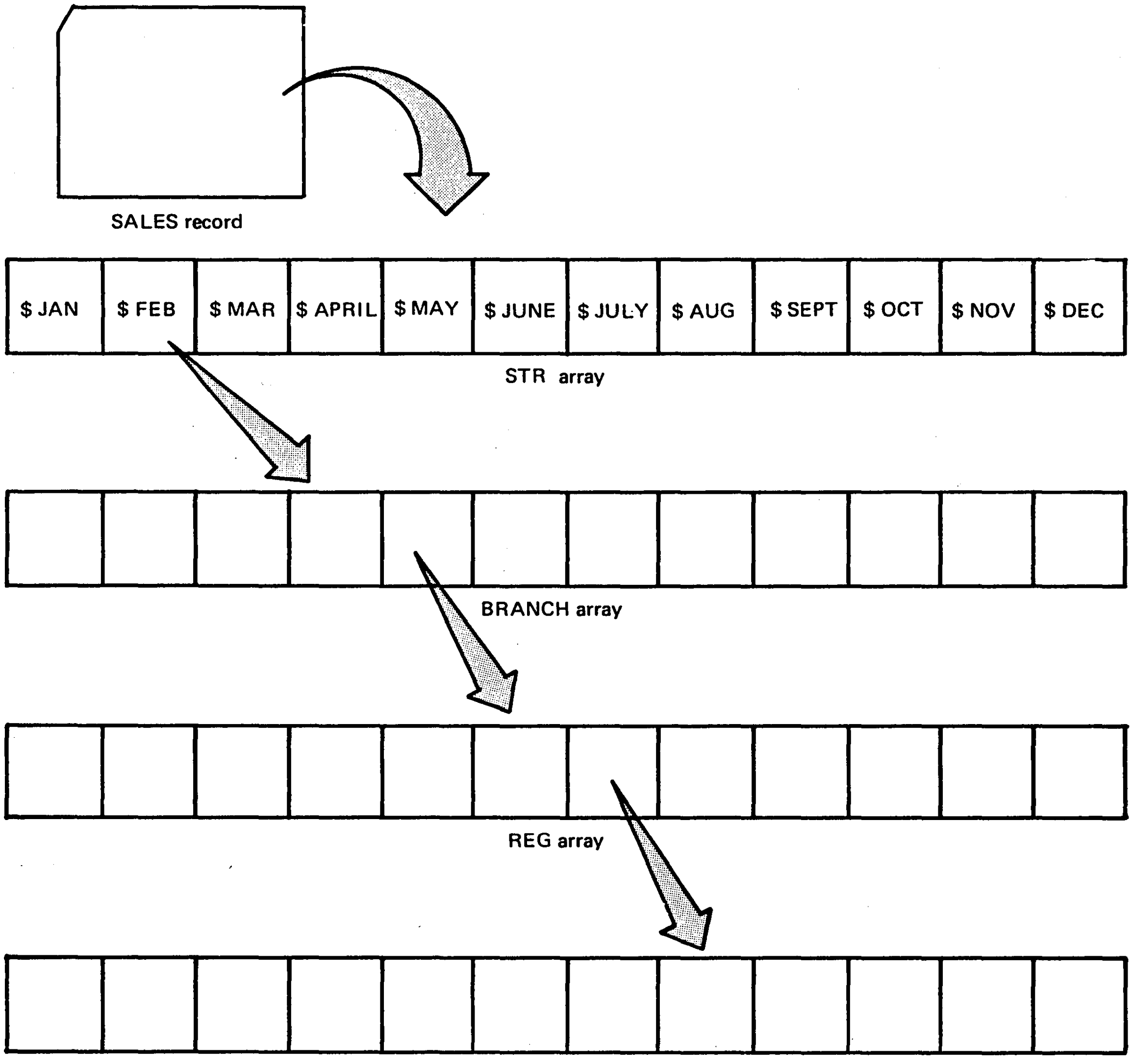
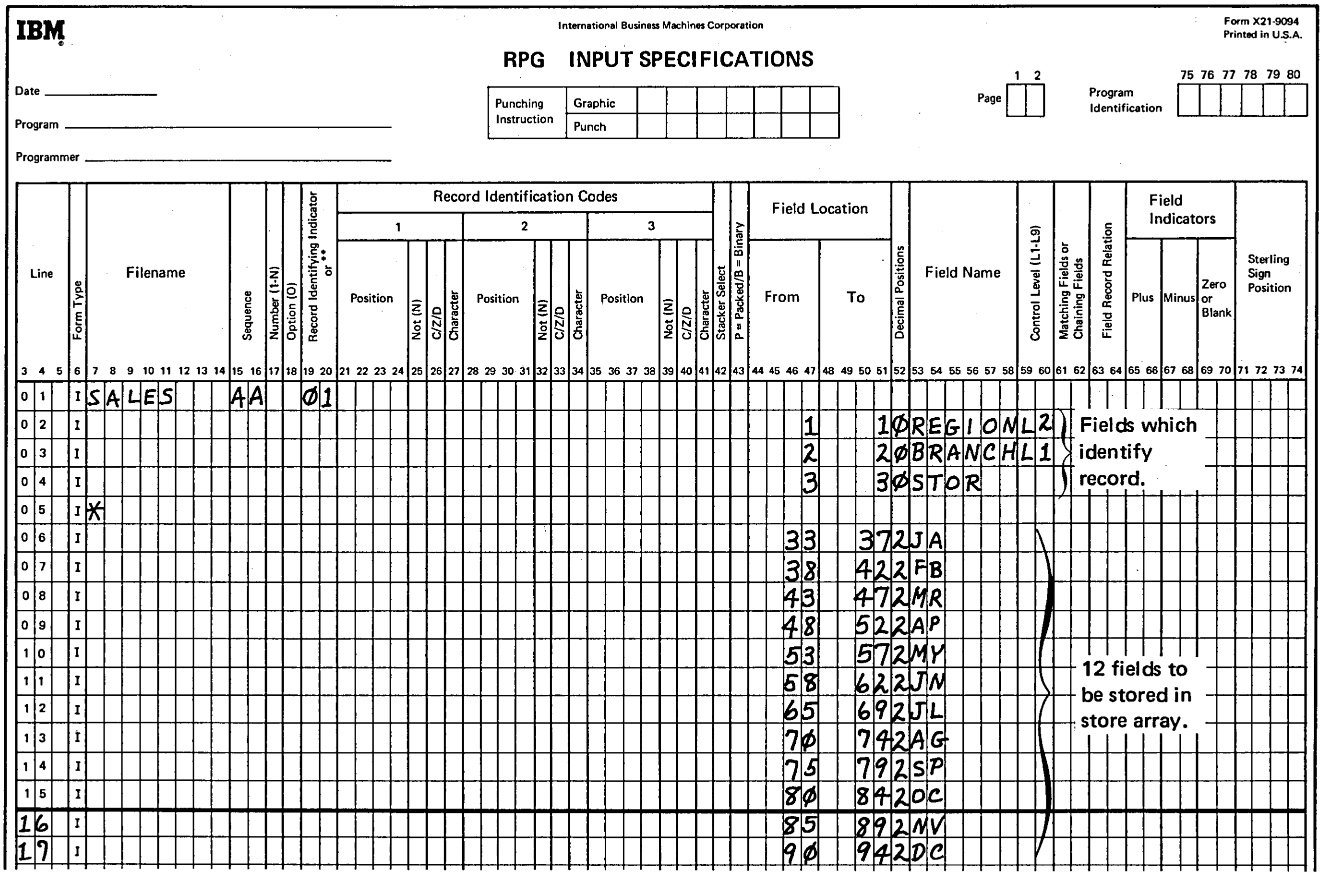


Figure 15-17. Four Arrays for Group Totals

As shown in Figure 15-18, control level indicator L1 is turned on when the first record is read for a store in a different branch. Likewise, L2 is turned on (and thus L1 is automatically turned on) when the first record is read for a store in a different region. The specifications on lines 06-17 merely describe the store sales data for each month of the year.



IBM

International Business Machines Corporation

Form X21-9094
Printed in U.S.A.

RPG INPUT SPECIFICATIONS

Date
Program
Programmer

Punching Instruction | Graphic | Punch

Page 1 2

Program Identification 75 76 77 78 79 80

Line	Form Type	Filename	Sequence	Record Identifying Indicator	Field Location From	Field Location To	Decimal Positions	Field Name	Control Level (L1-L9)
01	I	SALES	AA	Ø1					
02	I				1	1Ø		REGION	L2
03	I				2	2Ø		BRANCH	L1
04	I				3	3Ø		STOR	
05	I	*							
06	I				33	37	2	JA	
07	I				38	42	2	FB	
08	I				43	47	2	MR	
09	I				48	52	2	AP	
10	I				53	57	2	MY	
11	I				58	62	2	JN	
12	I				65	69	2	JL	
13	I				7Ø	74	2	AG	
14	I				75	79	2	SP	
15	I				8Ø	84	2	OC	
16	I				85	89	2	NV	
17	I				9Ø	94	2	DC	

Fields which identify record.

12 fields to be stored in store array.

Figure 15-18. Identifying Groups by Assigning Control Level Indicators

The specifications in Figure 15-19, insert A illustrate how control level indicators are used to control the performance of calculations and output.

As a sales record is read, the 12 monthly totals for that store are placed in the array STR. (How data gets into an array will be discussed later.) Each time new data is placed in STR (every time a card is processed), the fields are added to the BRNCH array to accumulate totals for the branch (Calculation Sheet, line 01). The store totals are printed as each record is processed, because every time a new card is read the data previously in the STR array is replaced by the totals for the next store (output lines 01-02).

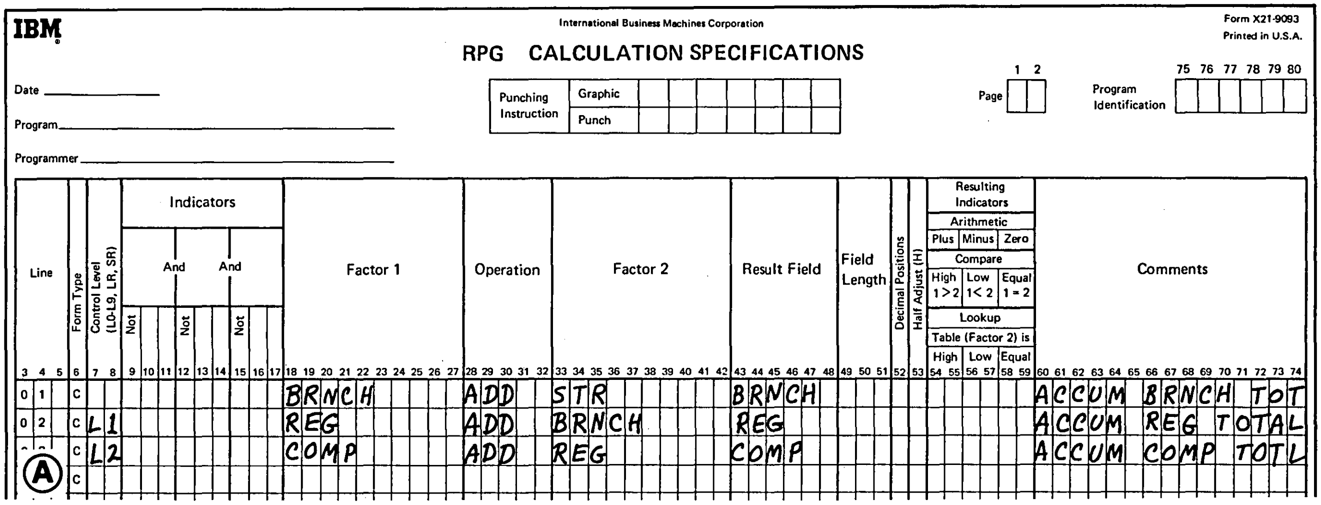
When all store records for a particular branch have been read and their totals printed and accumulated in the BRNCH array, an L1 control break occurs. The control break is indicated by reading the first store record in the next branch. Before processing this next record, the branch totals are printed and the BRNCH array is filled with zeros to prepare for accumulating the next branch totals (Figure 15-19, insert B, lines 03-04). Before printing and zeroing the BRNCH array, however, the branch totals are added to the REG array (Calculation Sheet, line 02).

The same program cycles are repeated for the rest of the records in region I. Remember, however, that data is accumulated into the REG array only when processing for a branch is complete (L1 on).

Once records for all branches within region I have been processed, L2 is turned on, indicating the first record in the next region has been read, but not yet processed. With L2 on, the 12 accumulated region totals are printed (Output Sheet, lines 05-06). Before output of the REG array, however, calculations conditioned by L2 on are performed; that is, the region totals are added to the company array COMP (Calculation Sheet, line 03). The calculation is done before output so the region totals can be saved before REG is filled with zeros.

The same procedure is followed for all store records in region II. During every program cycle, the store totals are printed and accumulated to form a branch total; when L1 is on, the branch total is printed and accumulated into a region total.

When the end of file is reached, the LR indicator is turned on. Automatically, all control level indicators assigned (L1 and L2) are also turned on. Therefore, after the last store record has been printed and the store totals added to BRNCH (Figure 15-19, insert A, line 01), any specifications conditioned by L1, L2, or LR are performed. In other words, the totals for the last branch are added to REG (Figure 15-19, insert A); then the region totals are added to COMP. Following the calculations, three sets of totals are printed; totals for the last branch (Figure 15-19, insert B, lines 03-01); then the region II totals; and, finally, the company totals.



IBM

International Business Machines Corporation

Form X21-9093
Printed in U.S.A.

RPG CALCULATION SPECIFICATIONS

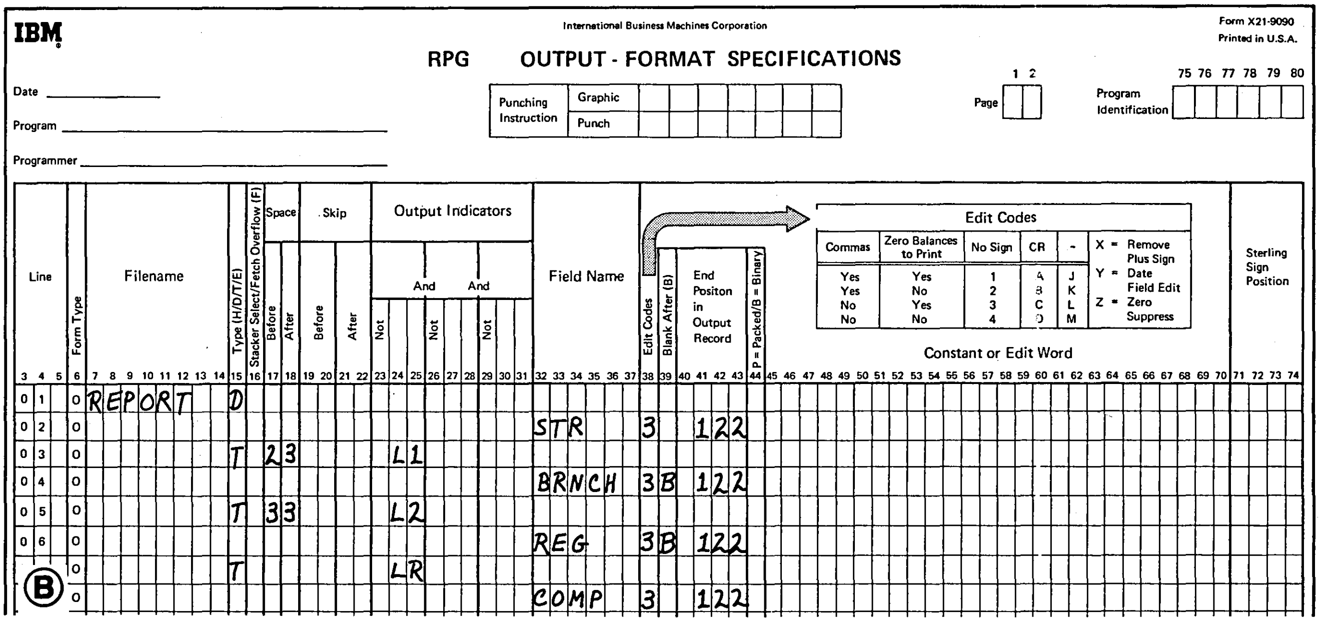
Date ____
Program ____
Programmer ____

Punching Instruction	Graphic	
	Punch	

Page 1 2 — Program Identification 75 76 77 78 79 80

Line (3-5)	Form Type (6)	Control Level (L0-L9, LR, SR) (7-8)	Indicators: Not, And Not, And Not (9-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field (43-48)	Field Length (49-51)	Decimal Positions (52)	Half Adjust (H) (53)	Resulting Indicators: Arithmetic Plus / Compare High 1>2 / Lookup Table (Factor 2) is High (54-55)	Minus / Low 1<2 / Low (56-57)	Zero / Equal 1=2 / Equal (58-59)	Comments (60-74)
01	C			BRNCH	ADD	STR	BRNCH							ACCUM BRNCH TOT
02	C	L1		REG	ADD	BRNCH	REG							ACCUM REG TOTAL
	C	L2		COMP	ADD	REG	COMP							ACCUM COMP TOTL
	C													

A



IBM

International Business Machines Corporation

Form X21-9090
Printed in U.S.A.

RPG OUTPUT - FORMAT SPECIFICATIONS

Date ____
Program ____
Programmer ____

Punching Instruction	Graphic	
	Punch	

Page 1 2 — Program Identification 75 76 77 78 79 80

Edit Codes

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Line (3-5)	Form Type (6)	Filename (7-14)	Type (H/D/T/E) (15)	Stacker Select/Fetch Overflow (F) (16)	Space Before (17)	Space After (18)	Skip Before (19-20)	Skip After (21-22)	Output Indicators: Not, And Not, And Not (23-31)	Field Name (32-37)	Edit Codes (38)	Blank After (B) (39)	End Position in Output Record (40-43)	P = Packed/B = Binary (44)	Constant or Edit Word (45-70)	Sterling Sign Position (71-74)
01	O	REPORT	D													
02	O									STR	3		122			
03	O		T		2	3			L1							
04	O									BRNCH	3	B	122			
05	O		T		3	3			L2							
06	O									REG	3	B	122			
	O		T						LR							
	O									COMP	3		122			

B

Figure 15-19. Accumulation and Output of Group Totals Using Arrays

REFERENCING INDIVIDUAL FIELDS OF AN ARRAY

In addition to referencing all fields of an array, you can use an individual array field in calculations or output. Suppose you have an array with each field containing the quantity in stock of a particular part manufactured by your company. Field 1 contains the quantity in stock for part #1, field 2 for part #2, and so on. When a shipment of ordered parts is received, the quantity in stock must be updated to reflect the current inventory. This means you should reference (add to) only particular fields of the array.

Indexing an Array

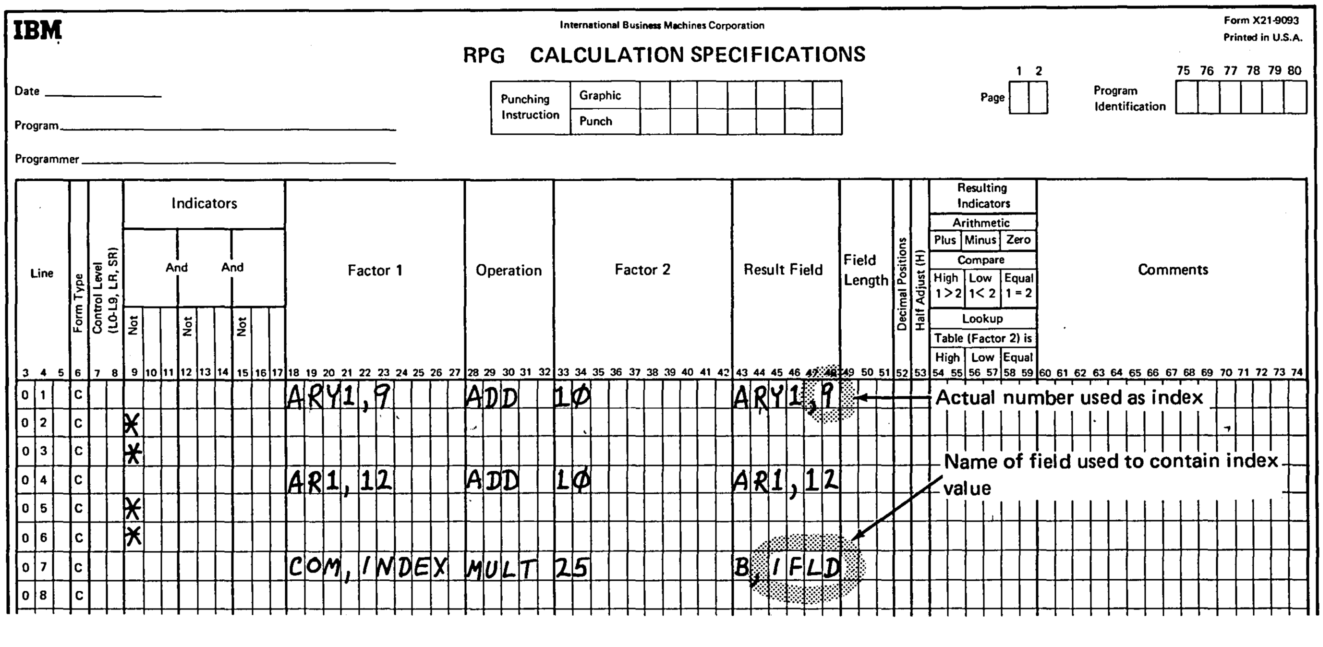
As you learned, if a calculation or output specification contains an array name alone, that specification is automatically performed for every field of the array. To reference only a single field of an array, you must identify that field for the computer. This is done by placing a comma after the array name, followed by an index which points to the particular field (Figure 15-20). This *index* can be either the actual number of the field to be referenced or the name of a field containing the number of the field to be used.

If you recall *Defining an Array,* the name used to refer to an array cannot exceed six characters in length. When referencing individual fields, both the array name and an index are necessary to refer to the data. Therefore, usually the array name, plus the comma, plus the index cannot exceed six characters. The name used to refer to an individual array field must be one to six characters long unless the name (with index) used to refer to an individual array field is specified *only* as Factor 1 or Factor 2 on the Calculation Sheet. In this case, the array name plus comma plus index may be as long as ten characters. However, the array name portion of the reference still cannot exceed six characters.

Figure 15-20, line 01 shows a valid reference to the ninth field of an array named ARY1. However, if the array contains ten or more fields, some of which may have to be referenced, the name of this array would have to be shortened to provide enough positions for the index (Figure 15-20, line 04). The limit of six characters applies even if the name of a field is used as an index. As line 07 shows, if an index field IFLD is specified, only one character (B) can be used as the name of the array because the indexed name is specified under Result Field. However, COM, INDEX on line 07 is valid, even though longer than six characters, because it is specified only under factor columns.

Specifying an Index Which Does Not Change

If you know exactly which field is to be used in a calculation or output operation and the specification is to reference the same field in every program cycle of the job, you should use a constant as the index. Assume a 7-field array (SLS) is defined to contain a salesman's six daily commission amounts and his total commission for the week. The six daily amounts from one of the salesmen's input records are read into fields 1-6 of the array. The seventh field on the input record contains zeros and is read into field 7 of the array (Figure 15-21, insert A).



IBM

International Business Machines Corporation

RPG CALCULATION SPECIFICATIONS

Form X21-9093
Printed in U.S.A.

Date ____________

Program ____________

Programmer ____________

Punching Instruction: Graphic / Punch

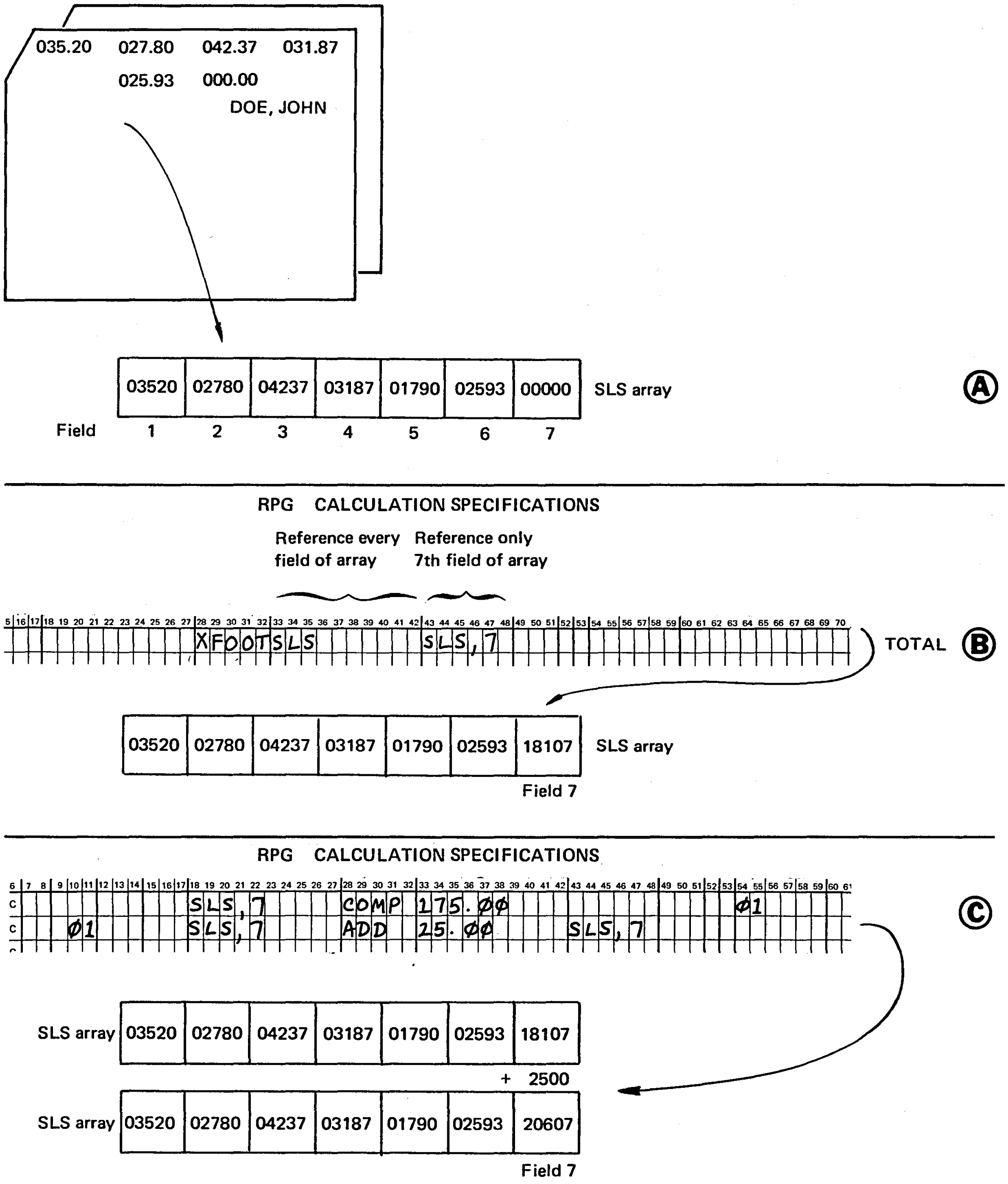
Page □□ (1 2)

Program Identification □□□□□□ (75 76 77 78 79 80)

Line	Form Type	Control Level (L0-L9, LR, SR)	Indicators: Not	Factor 1	Operation	Factor 2	Result Field	Field Length	Decimal Positions	Half Adjust (H)	Resulting Indicators	Comments
01	C			ARY1,9	ADD	1Ø	ARY1,9					Actual number used as index
02	C		*									
03	C		*									
04	C			AR1,12	ADD	1Ø	AR1,12					Name of field used to contain index value
05	C		*									
06	C		*									
07	C			COM,INDEX	MULT	25	B,IFLD					
08	C											

Figure 15-20. Referencing a Particular Field of an Array

The array fields are defined as 5-digit numbers with two decimal positions. Once the data is in the array, the XFOOT calculation operation is performed to add all fields of the array and place the total in the seventh field (Figure 15-21, insert B). The weekly total for every salesman is always stored in the seventh field. Therefore, the actual number 7 can be specified as the index. In addition, a $25 bonus is to be added to a salesman's total if his weekly commission exceeds $175 (Figure 15-21, insert C). Thus, in every program cycle, field 7 must first be compared to $175 to determine if the bonus is to be added to the contents of field 7.


035.20 027.80 042.37 031.87
025.93 000.00
DOE, JOHN
03520 02780 04237 03187 01790 02593 00000
SLS array
Field 1 2 3 4 5 6 7
A
RPG CALCULATION SPECIFICATIONS
Reference every field of array
Reference only 7th field of array
TOTAL
B
03520 02780 04237 03187 01790 02593 18107
SLS array
Field 7
RPG CALCULATION SPECIFICATIONS
C
SLS array 03520 02780 04237 03187 01790 02593 18107
+ 2500
SLS array 03520 02780 04237 03187 01790 02593 20607
Field 7

```
                      XFOOTSLS       SLS,7
```

```
C            SLS,7     COMP 175.ØØ                Ø1
C    Ø1      SLS,7     ADD  25.ØØ      SLS,7
```

Figure 15-21. Specifying a Number as an Index

Specifying an Index Which Can Be Changed

On the other hand, if the array field will vary when a particular specification is performed, the index should be a field name rather than an actual number. In this way, the number stored in the index field can be changed during the program to indicate which array field is to be referenced.

An array (STK) is used to contain the quantities in stock of all parts manufactured by a company. Field 1 of the array contains the quantity for part #1, field 2 for part #2, and so on. When additional parts are manufactured, the values in the appropriate fields must be updated. Therefore, records are punched daily for each type of part produced. Each record contains the part number (NM) and the quantity of that part produced (QTY).

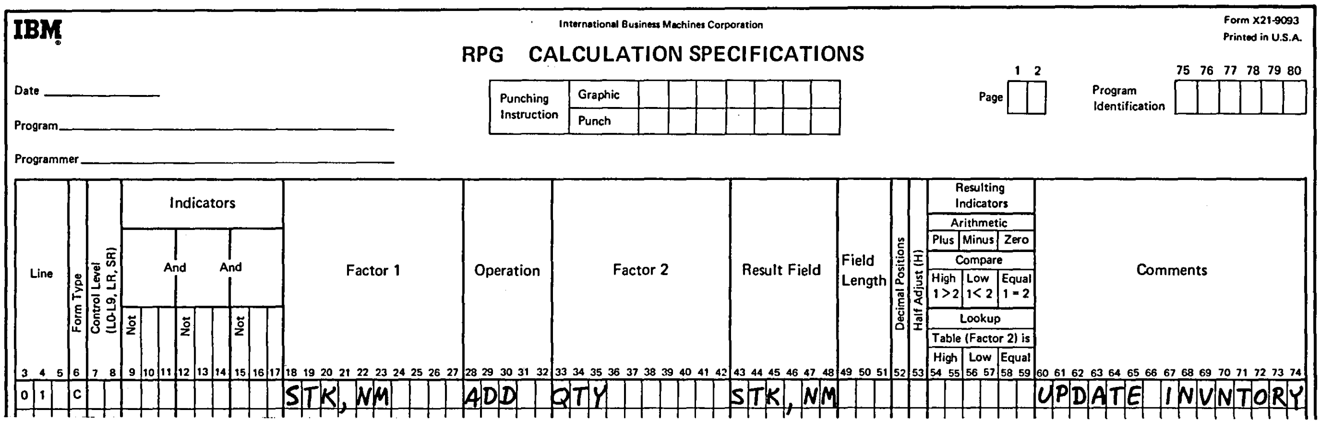
To perform the update job, the contents of the QTY field must be added to one of the array fields for every record processed. Thus, an index must be used in order to reference only the individual field to be updated. Since each daily record is for a different part number, the array field to be increased will vary each time the specification is performed. For this reason, an actual number cannot be specified as the index, because QTY would be added to the same field for every part number. Instead, the NM field, which contains the part number for each record, can be specified as the index (Figure 15-22). Then, every time the addition specification is performed, the part number just stored in NM indicates which number field of the array is to be referenced.

Output of Individual Fields of an Array

To put out individual fields of an array, you code the same output specifications you would for normal fields. The only difference is that under Field Name on the Output Sheet you must specify the array name followed by a comma and an index. The index then points to the particular field to be put out (Figure 15-23).

Thus, referencing individual array fields for output is the same as referencing them for calculations. If the same field is to be put out every time the output specification is performed, an actual number can be used as an index. Otherwise, if different fields are to be put out individually, a field should be specified which contains the changing index value. In any case, the array field (array name plus comma plus index) on the Output Sheet cannot exceed six characters in length.

Edit codes and edit words can be used to punctuate an individual numeric array field. If an entire array is to be put out but the fields require different punctuation, each field and its editing should be specified individually. Editing to be done on an individual array field is specified and performed just as it would be for any normal field. This means that, if an edit code is specified for an individual array field, two blanks are *not* automatically inserted before the field, as was the case with an entire array. Furthermore, although any type of output can be edited, editing is generally not specified for an array field which is to be punched on a card or written on disk to be used as input to another run.



IBM

International Business Machines Corporation

Form X21-9093
Printed in U.S.A.

RPG CALCULATION SPECIFICATIONS

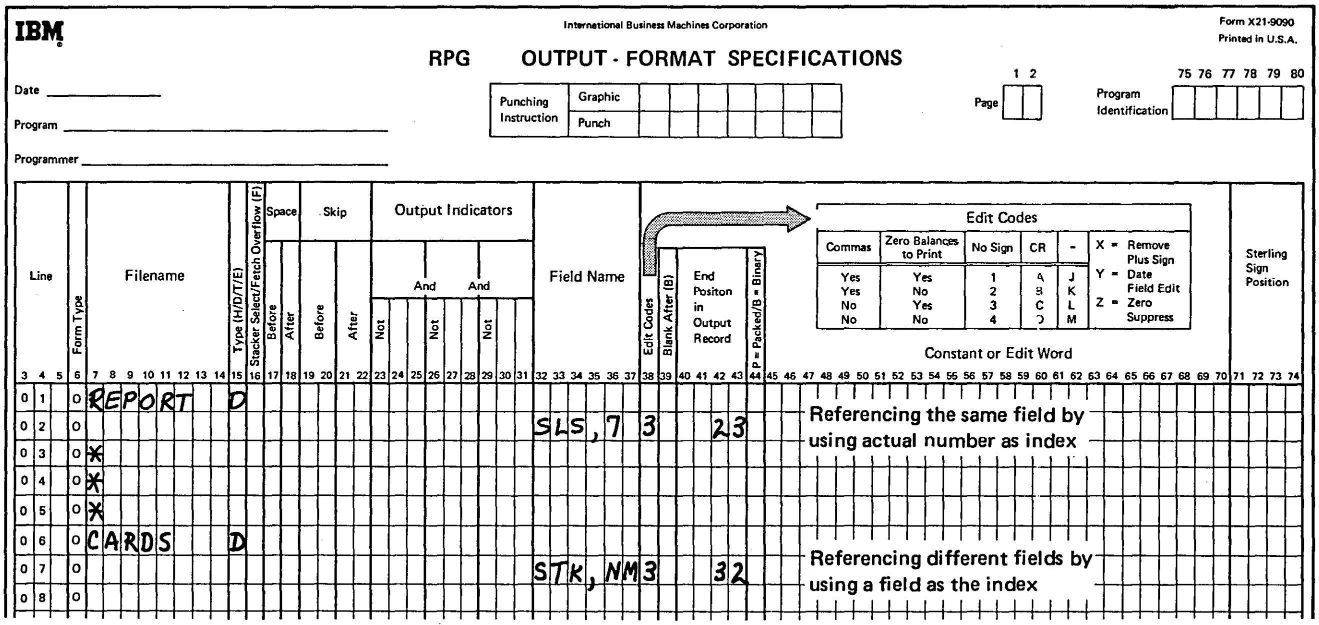
Date ____
Program ____
Programmer ____

Punching Instruction	Graphic							
	Punch							

Page 1 2 □□ Program Identification 75 76 77 78 79 80 □□□□□□

Line (3-5)	Form Type (6)	Control Level (L0-L9, LR, SR) (7-8)	Indicators: Not / And / Not / And / Not (9-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field (43-48)	Field Length (49-51)	Decimal Positions (52)	Half Adjust (H) (53)	Resulting Indicators: Arithmetic Plus Minus Zero / Compare High 1>2 Low 1<2 Equal 1=2 / Lookup Table (Factor 2) is High Low Equal (54-59)	Comments (60-74)
01	C			STK,NM	ADD	QTY	STK,NM					UPDATE INVNTORY

Figure 15-22. Specifying the Name of a Field as an Index



IBM

International Business Machines Corporation

Form X21-9090
Printed in U.S.A.

RPG OUTPUT - FORMAT SPECIFICATIONS

Date ____
Program ____
Programmer ____

Punching Instruction	Graphic							
	Punch							

Page 1 2 □□ Program Identification 75 76 77 78 79 80 □□□□□□

Edit Codes

Commas	Zero Balances to Print	No Sign	CR	-	
Yes	Yes	1	A	J	X = Remove Plus Sign
Yes	No	2	B	K	Y = Date Field Edit
No	Yes	3	C	L	Z = Zero Suppress
No	No	4	D	M	

Line (3-5)	Form Type (6)	Filename (7-14)	Type (H/D/T/E) (15)	Stacker Select/Fetch Overflow (F) (16)	Space Before (17)	Space After (18)	Skip Before (19-20)	Skip After (21-22)	Output Indicators Not / And / Not / And / Not (23-31)	Field Name (32-37)	Edit Codes (38)	Blank After (B) (39)	End Positon in Output Record (40-43)	P = Packed/B = Binary (44)	Constant or Edit Word (45-70)	Sterling Sign Position (71-74)
01	O	REPORT	D													
02	O									SLS,7	3		23		Referencing the same field by using actual number as index	
03	O	*														
04	O	*														
05	O	*														
06	O	CARDS	D													
07	O									STK,NM	3		32		Referencing different fields by using a field as the index	
08	O															

Figure 15-23. Output of Individual Array Fields

Referencing Only Part of a Field

When a field is referenced in a specification, all characters within that field are used in the calculation or output. However, you may wish to reference only some of the data stored in a field. For example, consider the case during address printing where the zip code is within the same field as the city and state on an input record but must be printed on a separate line on the output record (Figure 15-24).

The indexing capability of arrays can be used to enable you to reference specific characters from an input field. This is accomplished by setting up two arrays; one to contain the entire field of data and one to hold only the specific characters you want to reference.

First, the entire field from which you wish to use data is stored in an array (Figure 15-25). This array is previously defined as containing as many fields as there are characters in the field to be referenced. Thus, each character of the one field is actually stored in a separate field of the array. The array fields can then be referenced one at a time (using an index) until a field containing a specific character is located. This process of checking the fields of an array for particular data is referred to as *field scanning.*

After scanning the fields and locating a specific character, you can then move that character and any characters (fields) on either side of it to a smaller array. This array will then contain the portion of the original input field which you wish to reference separately in calculations or output.

For an address printing job, let's assume the input records are defined as shown in Figure 15-26. The CTYSTA field contains the city/state and zip code. Although names of the city and state may vary in length, the zip code is always five digits long. Any righthand, unused positions of the field will contain blanks.

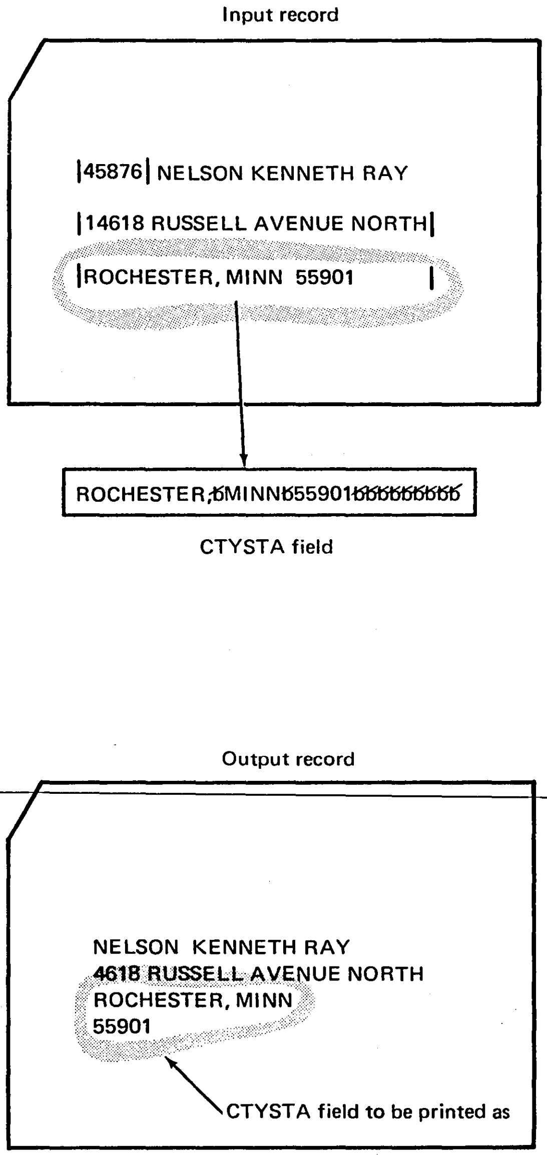
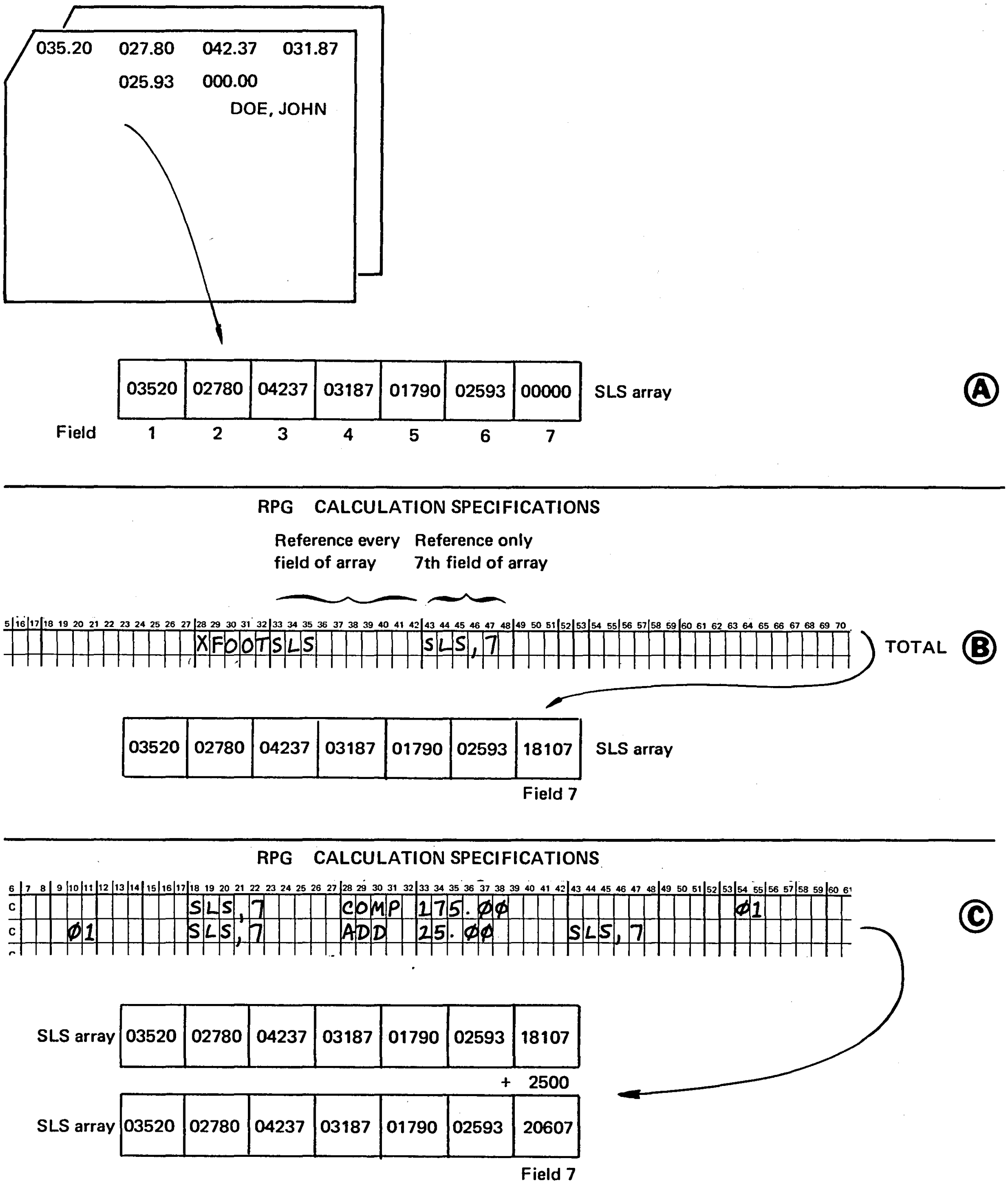

Input record
|45876| NELSON KENNETH RAY
|14618 RUSSELL AVENUE NORTH|
|ROCHESTER, MINN 55901 |
ROCHESTER,ƀMINNƀ55901ƀƀƀƀƀƀƀƀƀ
CTYSTA field
Output record
NELSON KENNETH RAY
4618 RUSSELL AVENUE NORTH
ROCHESTER, MINN
55901
CTYSTA field to be printed as

Figure 15-24. Referencing Parts of a Field Separately

The array fields are defined as 5-digit numbers with two decimal positions. Once the data is in the array, the XFOOT calculation operation is performed to add all fields of the array and place the total in the seventh field (Figure 15-21, insert B). The weekly total for every salesman is always stored in the seventh field. Therefore, the actual number 7 can be specified as the index. In addition, a $25 bonus is to be added to a salesman's total if his weekly commission exceeds $175 (Figure 15-21, insert C). Thus, in every program cycle, field 7 must first be compared to $175 to determine if the bonus is to be added to the contents of field 7.



035.20 027.80 042.37 031.87
025.93 000.00
DOE, JOHN
03520 02780 04237 03187 01790 02593 00000 SLS array
Field 1 2 3 4 5 6 7
A
RPG CALCULATION SPECIFICATIONS
Reference every field of array
Reference only 7th field of array
TOTAL
B
03520 02780 04237 03187 01790 02593 18107 SLS array
Field 7
RPG CALCULATION SPECIFICATIONS
C
SLS array 03520 02780 04237 03187 01790 02593 18107
+ 2500
SLS array 03520 02780 04237 03187 01790 02593 20607
Field 7

```
XFOOTSLS      SLS,7
```

```
C          SLS,7     COMP 175.ØØ                  Ø1
C   Ø1     SLS,7     ADD  25.ØØ      SLS,7
```

Figure 15-21. Specifying a Number as an Index

Specifying an Index Which Can Be Changed

On the other hand, if the array field will vary when a particular specification is performed, the index should be a field name rather than an actual number. In this way, the number stored in the index field can be changed during the program to indicate which array field is to be referenced.

An array (STK) is used to contain the quantities in stock of all parts manufactured by a company. Field 1 of the array contains the quantity for part #1, field 2 for part #2, and so on. When additional parts are manufactured, the values in the appropriate fields must be updated. Therefore, records are punched daily for each type of part produced. Each record contains the part number (NM) and the quantity of that part produced (QTY).

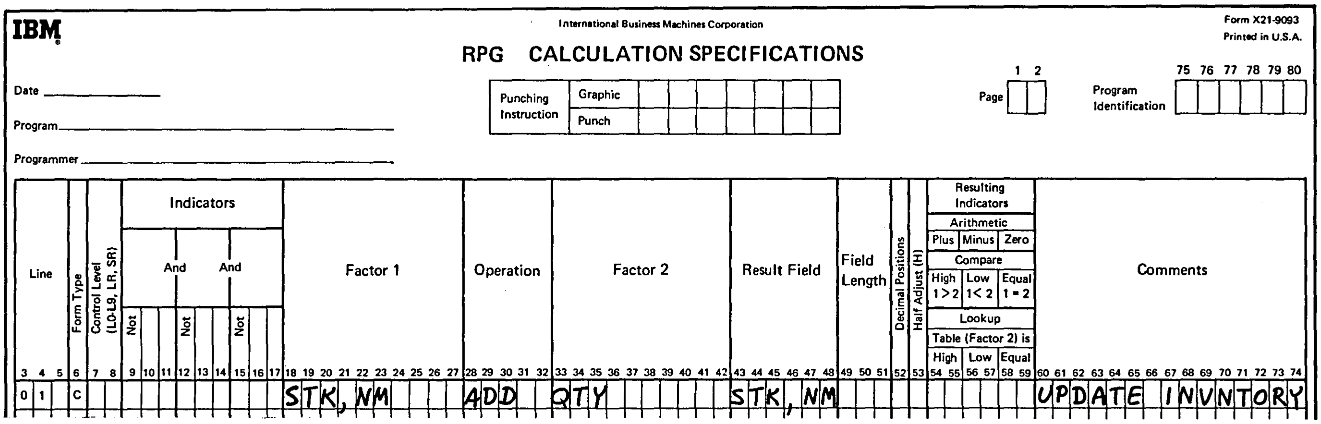
To perform the update job, the contents of the QTY field must be added to one of the array fields for every record processed. Thus, an index must be used in order to reference only the individual field to be updated. Since each daily record is for a different part number, the array field to be increased will vary each time the specification is performed. For this reason, an actual number cannot be specified as the index, because QTY would be added to the same field for every part number. Instead, the NM field, which contains the part number for each record, can be specified as the index (Figure 15-22). Then, every time the addition specification is performed, the part number just stored in NM indicates which number field of the array is to be referenced.

Output of Individual Fields of an Array

To put out individual fields of an array, you code the same output specifications you would for normal fields. The only difference is that under Field Name on the Output Sheet you must specify the array name followed by a comma and an index. The index then points to the particular field to be put out (Figure 15-23).

Thus, referencing individual array fields for output is the same as referencing them for calculations. If the same field is to be put out every time the output specification is performed, an actual number can be used as an index. Otherwise, if different fields are to be put out individually, a field should be specified which contains the changing index value. In any case, the array field (array name plus comma plus index) on the Output Sheet cannot exceed six characters in length.

Edit codes and edit words can be used to punctuate an individual numeric array field. If an entire array is to be put out but the fields require different punctuation, each field and its editing should be specified individually. Editing to be done on an individual array field is specified and performed just as it would be for any normal field. This means that, if an edit code is specified for an individual array field, two blanks are *not* automatically inserted before the field, as was the case with an entire array. Furthermore, although any type of output can be edited, editing is generally not specified for an array field which is to be punched on a card or written on disk to be used as input to another run.



IBM

International Business Machines Corporation

RPG CALCULATION SPECIFICATIONS

Form X21-9093
Printed in U.S.A.

Date ____

Program ____

Programmer ____

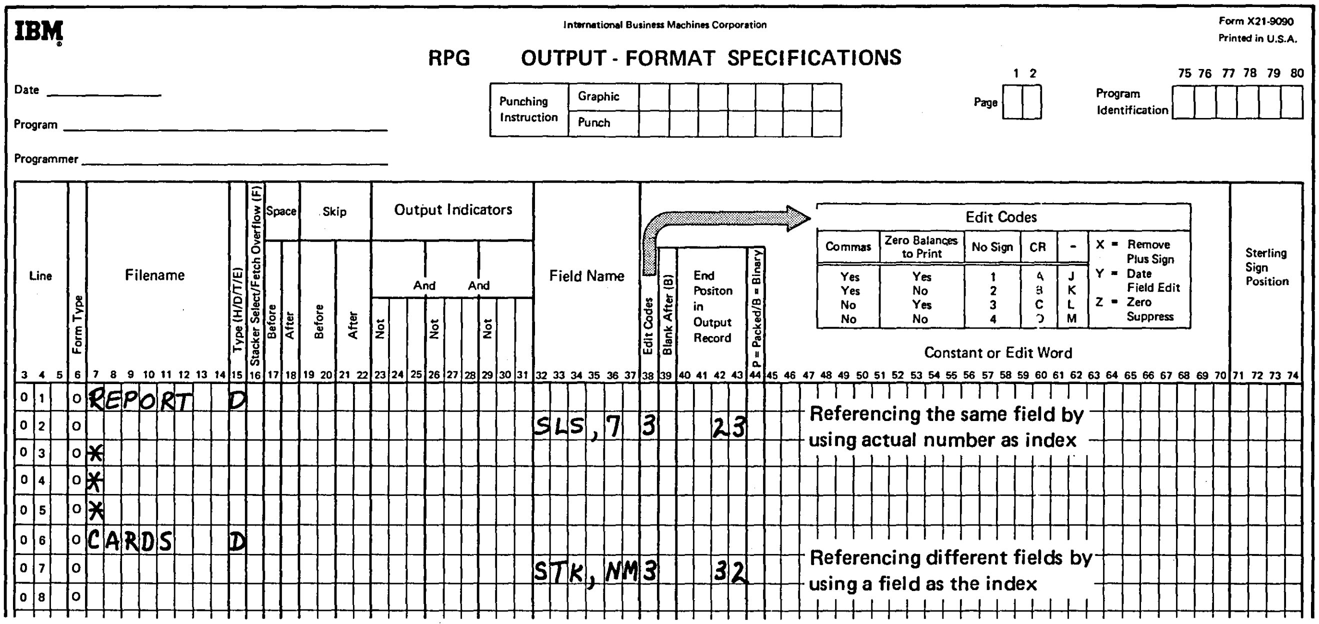
Punching Instruction: Graphic / Punch

Page 1 2

Program Identification 75 76 77 78 79 80

Line	Form Type	Control Level (L0-L9, LR, SR)	Indicators	Factor 1	Operation	Factor 2	Result Field	Field Length	Decimal Positions	Half Adjust (H)	Resulting Indicators	Comments
0 1	C			STK,NM	ADD	QTY	STK,NM					UPDATE INVNTORY

Figure 15-22. Specifying the Name of a Field as an Index



IBM

International Business Machines Corporation

RPG OUTPUT - FORMAT SPECIFICATIONS

Form X21-9090
Printed in U.S.A.

Date ____

Program ____

Programmer ____

Punching Instruction: Graphic / Punch

Page 1 2

Program Identification 75 76 77 78 79 80

Edit Codes

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Line	Form Type	Filename	Type (H/D/T/E)	Field Name	Edit Codes	End Position in Output Record	Constant or Edit Word
0 1	O	REPORT	D				
0 2	O			SLS,7	3	23	Referencing the same field by using actual number as index
0 3	O	*					
0 4	O	*					
0 5	O	*					
0 6	O	CARDS	D				
0 7	O			STK,NM	3	32	Referencing different fields by using a field as the index
0 8	O						

Figure 15-23. Output of Individual Array Fields

Referencing Only Part of a Field

When a field is referenced in a specification, all characters within that field are used in the calculation or output. However, you may wish to reference only some of the data stored in a field. For example, consider the case during address printing where the zip code is within the same field as the city and state on an input record but must be printed on a separate line on the output record (Figure 15-24).

The indexing capability of arrays can be used to enable you to reference specific characters from an input field. This is accomplished by setting up two arrays; one to contain the entire field of data and one to hold only the specific characters you want to reference.

First, the entire field from which you wish to use data is stored in an array (Figure 15-25). This array is previously defined as containing as many fields as there are characters in the field to be referenced. Thus, each character of the one field is actually stored in a separate field of the array. The array fields can then be referenced one at a time (using an index) until a field containing a specific character is located. This process of checking the fields of an array for particular data is referred to as *field scanning.*

After scanning the fields and locating a specific character, you can then move that character and any characters (fields) on either side of it to a smaller array. This array will then contain the portion of the original input field which you wish to reference separately in calculations or output.

For an address printing job, let's assume the input records are defined as shown in Figure 15-26. The CTYSTA field contains the city/state and zip code. Although names of the city and state may vary in length, the zip code is always five digits long. Any righthand, unused positions of the field will contain blanks.

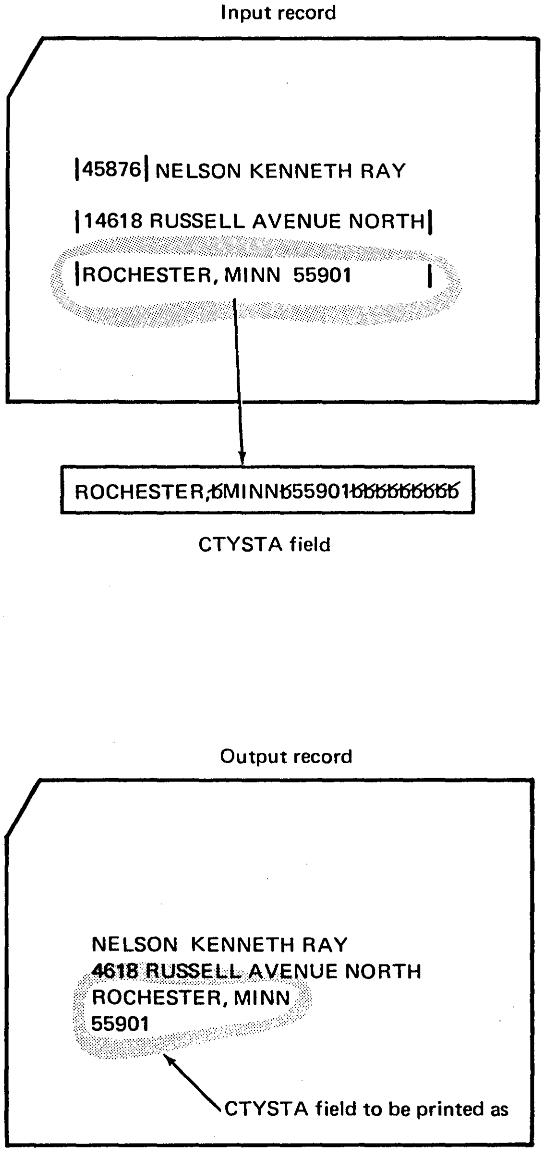

Input record
|45876| NELSON KENNETH RAY
|14618 RUSSELL AVENUE NORTH|
|ROCHESTER, MINN 55901 |
ROCHESTER,ƀMINNƀ55901ƀƀƀƀƀƀƀƀƀ
CTYSTA field
Output record
NELSON KENNETH RAY
4618 RUSSELL AVENUE NORTH
ROCHESTER, MINN
55901
CTYSTA field to be printed as

Figure 15-24. Referencing Parts of a Field Separately

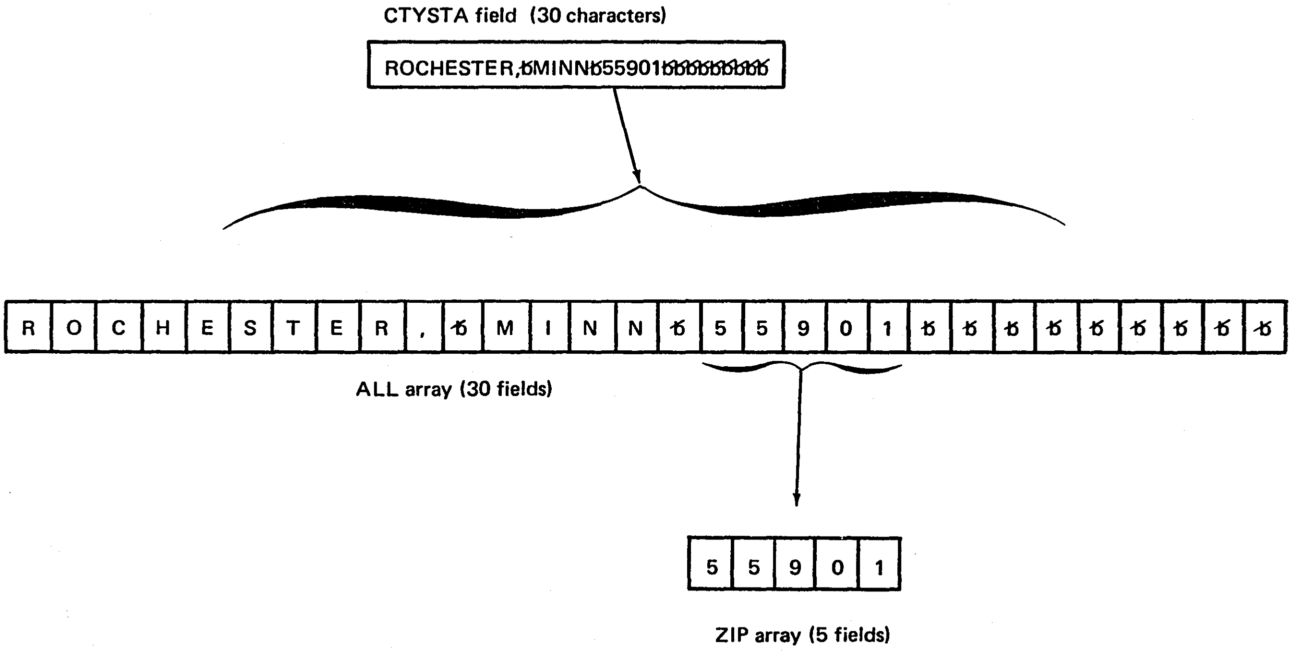
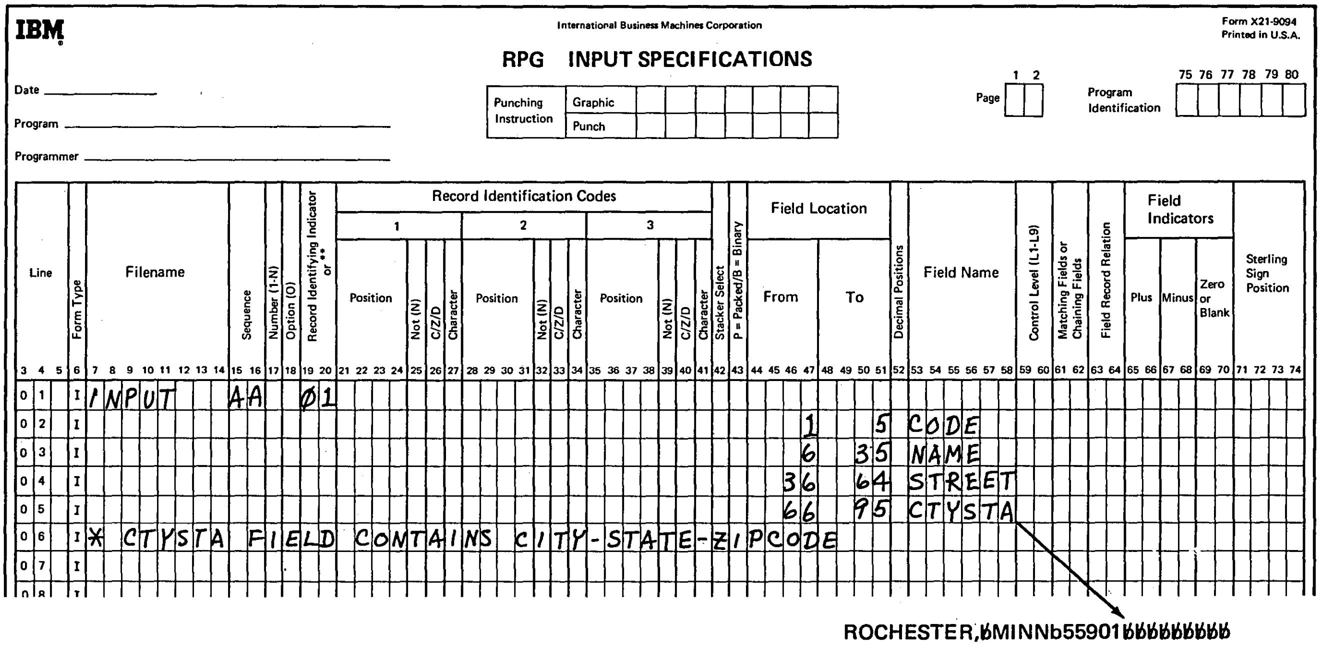

CTYSTA field (30 characters)
ROCHESTER,ƀMINNƀ55901ƀƀƀƀƀƀƀƀƀ
R O C H E S T E R , ƀ M I N N ƀ 5 5 9 0 1 ƀ ƀ ƀ ƀ ƀ ƀ ƀ ƀ ƀ
ALL array (30 fields)
5 5 9 0 1
ZIP array (5 fields)

Figure 15-25. Isolating Part of a Field


IBM

International Business Machines Corporation

Form X21-9094
Printed in U.S.A.

RPG INPUT SPECIFICATIONS

Date ______

Program ______

Programmer ______

Punching Instruction	Graphic							
	Punch							

Page 1 2 □□

Program Identification 75 76 77 78 79 80 □□□□□□

Line (3-5)	Form Type (6)	Filename (7-14)	Sequence (15-16)	Number (1-N) (17)	Option (O) (18)	Record Identifying Indicator or ** (19-20)	Record Identification Codes (21-41)	Stacker Select (42)	P = Packed/B = Binary (43)	Field Location From (44-47)	Field Location To (48-51)	Decimal Positions (52)	Field Name (53-58)	Control Level (L1-L9) (59-60)	Matching Fields or Chaining Fields (61-62)	Field Record Relation (63-64)	Field Indicators Plus (65-66)	Minus (67-68)	Zero or Blank (69-70)	Sterling Sign Position (71-74)
0 1	I	INPUT	AA			Ø1														
0 2	I									1	5		CODE							
0 3	I									6	35		NAME							
0 4	I									36	64		STREET							
0 5	I									66	95		CTYSTA							
0 6	I	* CTYSTA FIELD CONTAINS CITY-STATE-ZIPCODE																		
0 7	I																			
0 8	I																			

ROCHESTER,ƀMINNb55901ƀƀƀƀƀƀƀƀƀ

CTYSTA field

Figure 15-26. Defining a Field to be Scanned

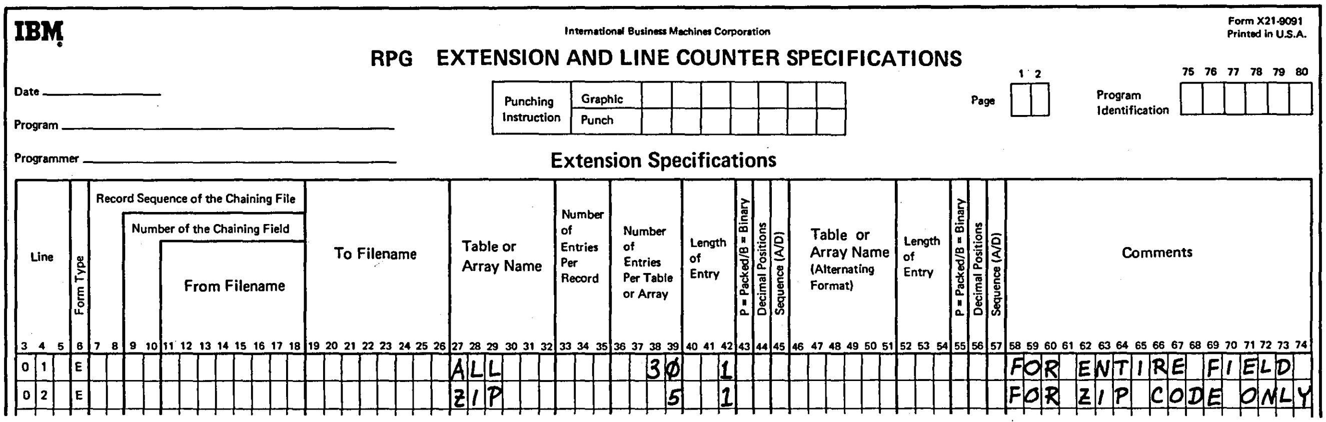
The two arrays for this job are defined with the extension specifications in Figure 15-27. ALL is set up to contain 30 fields, one for each character of the CTYSTA field from the input record. The five fields of the ZIP array will be used to contain the zip code portion of the CTYSTA field.

To locate the zip code in the ALL array, the fields must be scanned one at a time, beginning with the last (rightmost) field of the array. Thus, the index field (A) for referencing the individual fields of ALL is initially set up in the Calculation Sheet to contain the value 30 (Figure 15-28, line 01). When the last (rightmost) character of the zip code is located, it should be moved to the rightmost (fifth) field of the ZIP array. Therefore, a *5* is initially set up in the index field (Z) which will reference a particular field of ZIP (Figure 15-28, line 02).

With the index fields set up, the computer can begin scanning ALL for the zip code. The fields of ALL are checked, from right to left, until the first nonblank character is located. As line 04 shows, a character is compared to a blank. If it is blank, the index value is decreased (line 05) so the next character to the left can be compared to a blank. When one of the characters checked is not a blank (indicator 20 off), the last character of the zip code has been located. This ends the field scanning.

The computer can now proceed to perform the next group of calculations which move the located character to the rightmost position of the ZIP array (line 09). The character in the ALL array which was moved to the ZIP array is now made blank (line 10) so the city and state line can be printed without the zip code. Line 11 checks the index value of *Z* to determine if all five characters of the zip code have been moved to the ZIP array and made blank in the ALL array. If *Z* has a value of *1,* all five characters have been moved. If not (indicator 21 is off), the index values for both arrays are decreased by *1,* so the next zip code character (to the left of the last one moved) can be moved from ALL to the next portion in the ZIP array.

After calculations, the output specifications in Figure 15-29 cause the name and address to be printed. The NAME and STREET fields are printed exactly as they appear on the input record. City and state, on the other hand, are printed from the ALL array rather than the input field, because the zip code has been blanked out. The zip code, which was moved to the ZIP array, is then printed alone on the next output line.



IBM

International Business Machines Corporation

Form X21-9091
Printed in U.S.A.

RPG EXTENSION AND LINE COUNTER SPECIFICATIONS

Date ______

Program ______

Programmer ______

Punching Instruction: Graphic / Punch

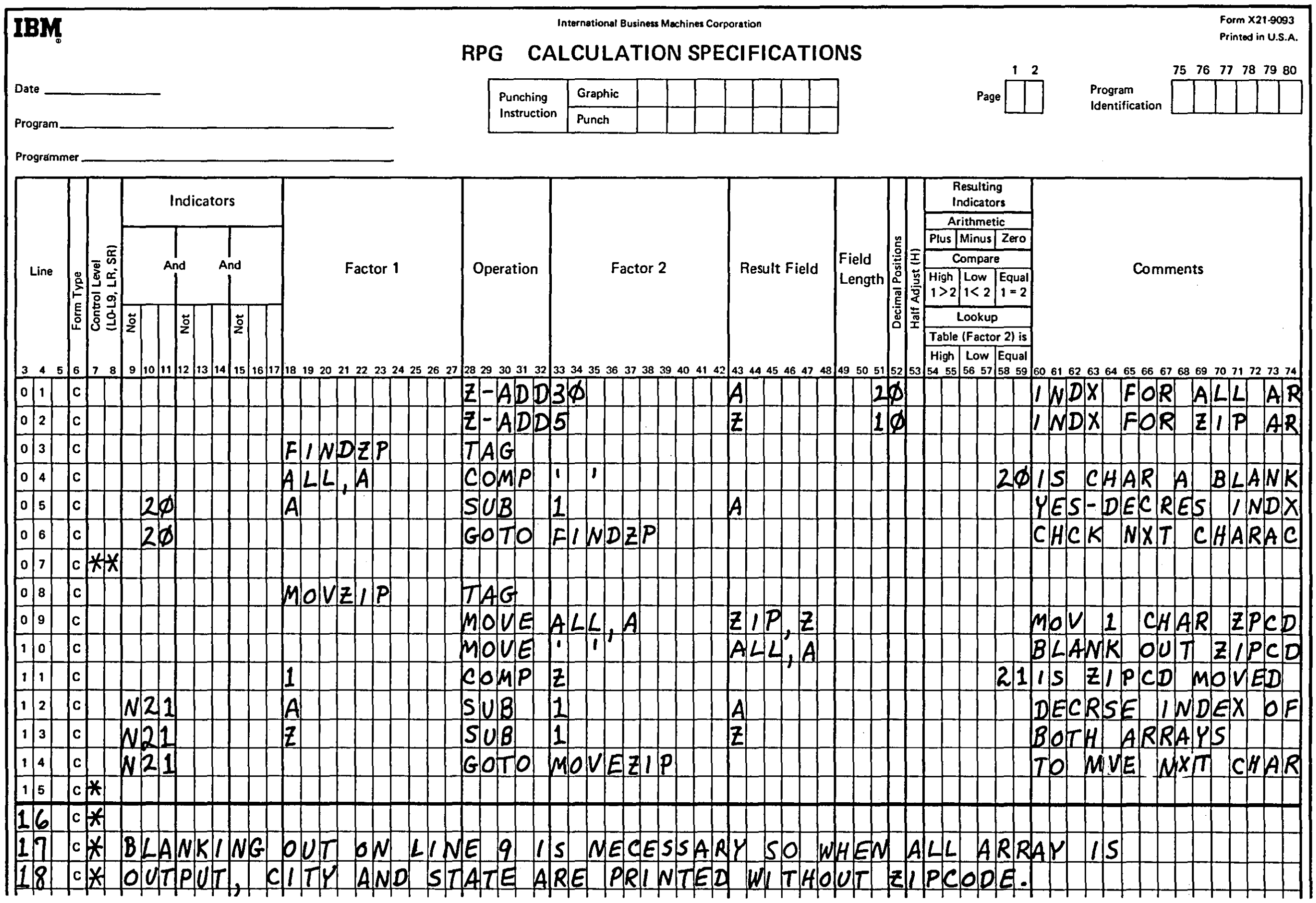
Page 1 2

Program Identification 75 76 77 78 79 80

Extension Specifications

Line (3-5)	Form Type (6)	Record Sequence of the Chaining File (7-8)	Number of the Chaining Field (9-10)	From Filename (11-18)	To Filename (19-26)	Table or Array Name (27-32)	Number of Entries Per Record (33-35)	Number of Entries Per Table or Array (36-39)	Length of Entry (40-42)	P = Packed/B = Binary (43)	Decimal Positions (44)	Sequence (A/D) (45)	Table or Array Name (Alternating Format) (46-51)	Length of Entry (52-54)	P = Packed/B = Binary (55)	Decimal Positions (56)	Sequence (A/D) (57)	Comments (58-74)
01	E					ALL		3∅	1									FOR ENTIRE FIELD
02	E					ZIP		5	1									FOR ZIP CODE ONLY

Figure 15-27. Defining Arrays for Field Scanning



IBM | International Business Machines Corporation | Form X21-9093 Printed in U.S.A.

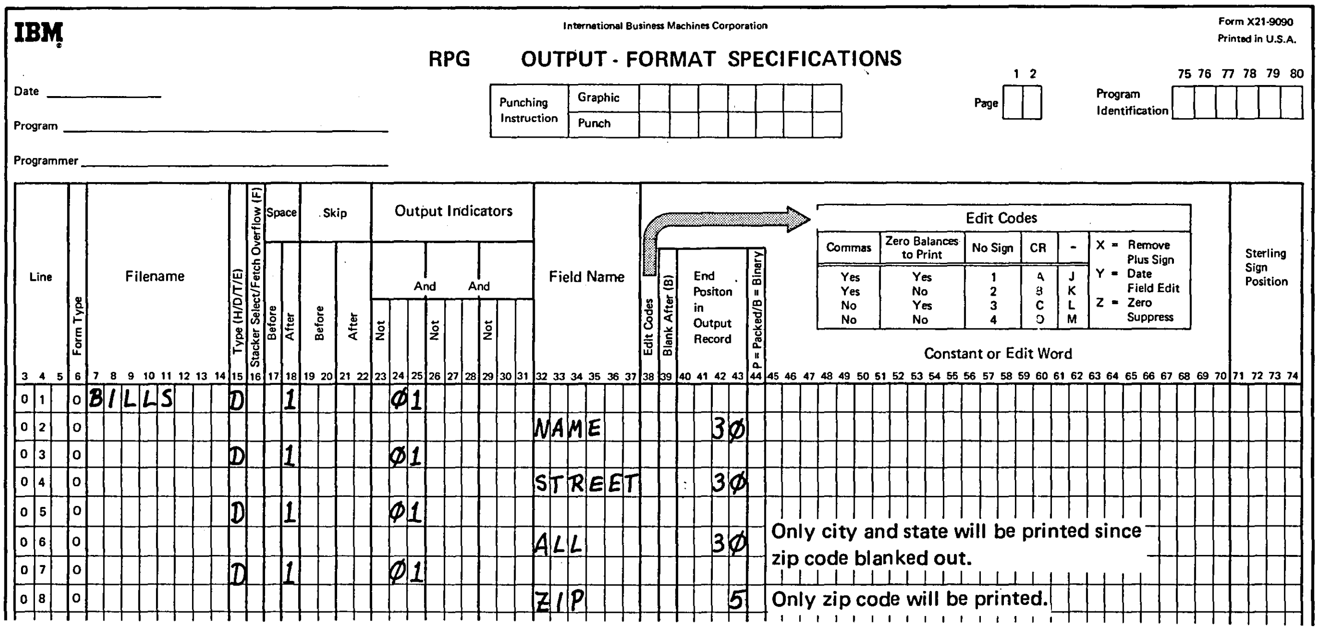
RPG CALCULATION SPECIFICATIONS

Date ______ Program ______ Programmer ______

Punching Instruction: Graphic / Punch | Page 1 2 | Program Identification 75 76 77 78 79 80

Line (3-5)	Form Type (6)	Control Level (LO-L9, LR, SR) (7-8)	Indicators (9-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field (43-48)	Field Length (49-51)	Decimal Positions (52)	Half Adjust (H) (53)	Resulting Indicators Plus / High 1>2 (54-55)	Minus / Low 1<2 (56-57)	Zero / Equal 1=2 (58-59)	Comments (60-74)
01	C				Z-ADD	3Ø	A	2	Ø					INDX FOR ALL AR
02	C				Z-ADD	5	Z	1	Ø					INDX FOR ZIP AR
03	C			FINDZP	TAG									
04	C			ALL,A	COMP	' '							2Ø	IS CHAR A BLANK
05	C		2Ø	A	SUB	1	A							YES-DECRES INDX
06	C		2Ø		GOTO	FINDZP								CHCK NXT CHARAC
07	C	**												
08	C			MOVZIP	TAG									
09	C				MOVE	ALL,A	ZIP,Z							MOV 1 CHAR ZPCD
10	C				MOVE	' '	ALL,A							BLANK OUT ZIPCD
11	C			1	COMP	Z							21	IS ZIPCD MOVED
12	C		N21	A	SUB	1	A							DECRSE INDEX OF
13	C		N21	Z	SUB	1	Z							BOTH ARRAYS
14	C		N21		GOTO	MOVEZIP								TO MVE NXT CHAR
15	C	*												
16	C	*												
17	C	* BLANKING OUT ON LINE 9 IS NECESSARY SO WHEN ALL ARRAY IS												
18	C	* OUTPUT, CITY AND STATE ARE PRINTED WITHOUT ZIPCODE.												

Figure 15-28. Field Scanning



IBM | International Business Machines Corporation | Form X21-9090 Printed in U.S.A.

RPG OUTPUT - FORMAT SPECIFICATIONS

Date ______ Program ______ Programmer ______

Punching Instruction: Graphic / Punch | Page 1 2 | Program Identification 75 76 77 78 79 80

Edit Codes

Commas	Zero Balances to Print	No Sign	CR	-	
Yes	Yes	1	A	J	X = Remove Plus Sign
Yes	No	2	B	K	Y = Date Field Edit
No	Yes	3	C	L	Z = Zero Suppress
No	No	4	D	M	

Line (3-5)	Form Type (6)	Filename (7-14)	Type (H/D/T/E) (15)	Stacker Select/Fetch Overflow (F) (16)	Space Before (17)	Space After (18)	Skip Before (19-20)	Skip After (21-22)	Output Indicators (23-31)	Field Name (32-37)	Edit Codes (38)	Blank After (B) (39)	End Position in Output Record (40-43)	P = Packed/B = Binary (44)	Constant or Edit Word (45-70)	Sterling Sign Position (71-74)
01	O	BILLS	D			1			Ø1							
02	O									NAME			3Ø			
03	O		D			1			Ø1							
04	O									STREET			3Ø			
05	O		D			1			Ø1							
06	O									ALL			3Ø		Only city and state will be printed since zip code blanked out.	
07	O		D			1			Ø1							
08	O									ZIP			5		Only zip code will be printed.	

Figure 15-29. Output of Part of a Field

LOKUP OF AN ARRAY

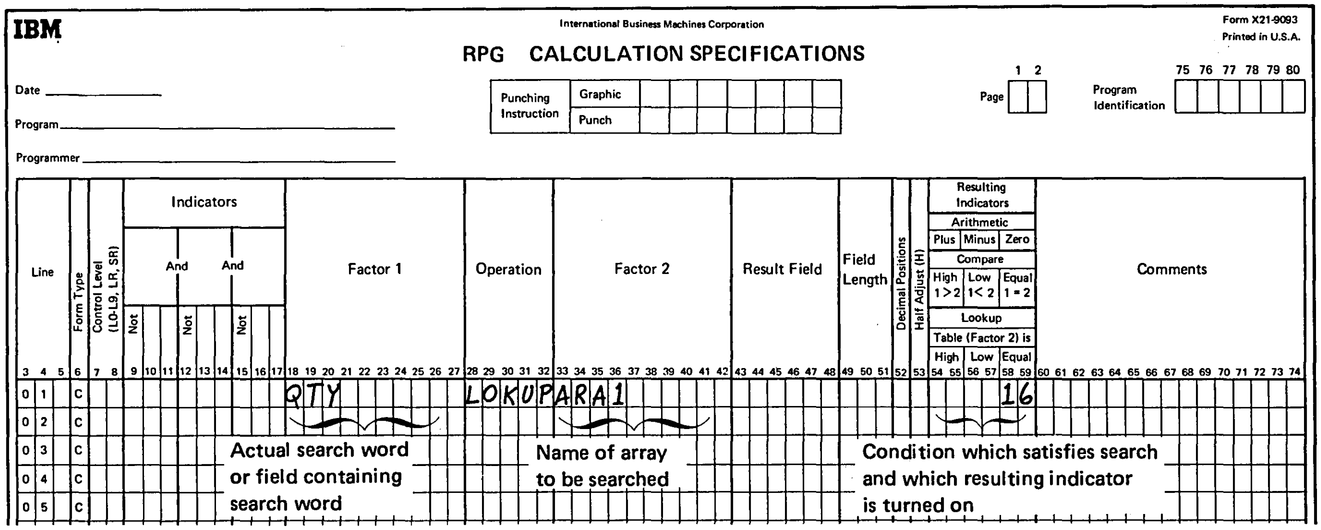
Searching an Array for a Particular Field

An array can be searched to determine if a particular field of data is stored in the array. Actually, the array lookup is coded and performed in almost the same way as a single table lookup. As the Calculation Sheet in Figure 15-30 shows, you specify:

1. The search word to be used.
2. The LOKUP operation code.
3. The array to be searched.
4. The condition which must be satisfied.
5. The resulting indicator which turns on if the condition is met.

The array lookup continues, one field at a time, until the search condition is satisfied or the end of the array is reached, whichever occurs first. As is the case for table lookups, array fields must be in sequence (A or D) if searching for either a low or high condition.

Although array and table searches are similar, there is an important difference you must be aware of. Remember, the array lookup is similar to a *single* table lookup, not a two-table lookup. Only one array is specified in the lookup operation. Only one array is specified in the lookup operation. Any field which is referenced as the result of a successful search can only be from the array actually searched. In other words, the array can not be searched to make a field from another array available, as is the case when two related tables are used in a lookup operation. For this reason, no result field is ordinarily specified in an array lookup operation.



IBM International Business Machines Corporation Form X21-9093 Printed in U.S.A.

RPG CALCULATION SPECIFICATIONS

Date ____ Program ____ Programmer ____

Punching Instruction: Graphic / Punch

Page 1 2 Program Identification 75 76 77 78 79 80

Line	Form Type	Control Level (L0-L9, LR, SR)	Indicators (Not, And Not, And Not)	Factor 1	Operation	Factor 2	Result Field	Field Length	Decimal Positions	Half Adjust (H)	Resulting Indicators: High 1>2	Low 1<2	Equal 1=2	Comments
3 4 5	6	7 8	9–17	18–27	28–32	33–42	43–48	49–51	52	53	54 55	56 57	58 59	60–74
0 1	C			QTY	LOKUP	ARA1							16	
0 2	C													
0 3	C			Actual search word		Name of array		Condition which satisfies search						
0 4	C			or field containing		to be searched		and which resulting indicator						
0 5	C			search word				is turned on						

Figure 15-30. Searching an Array for a Particular Field

Starting the Search at a Particular Field

Another very important difference between tables and arrays concerns where the search can begin. In a table search, only the name of the table to be searched can be specified as Factor 2 of the lookup operation. As a result, a table search always begins at the first table element. Likewise, if only an array name is specified as Factor 2 of a lookup operation, the search will automatically begin at the first field of the named array.

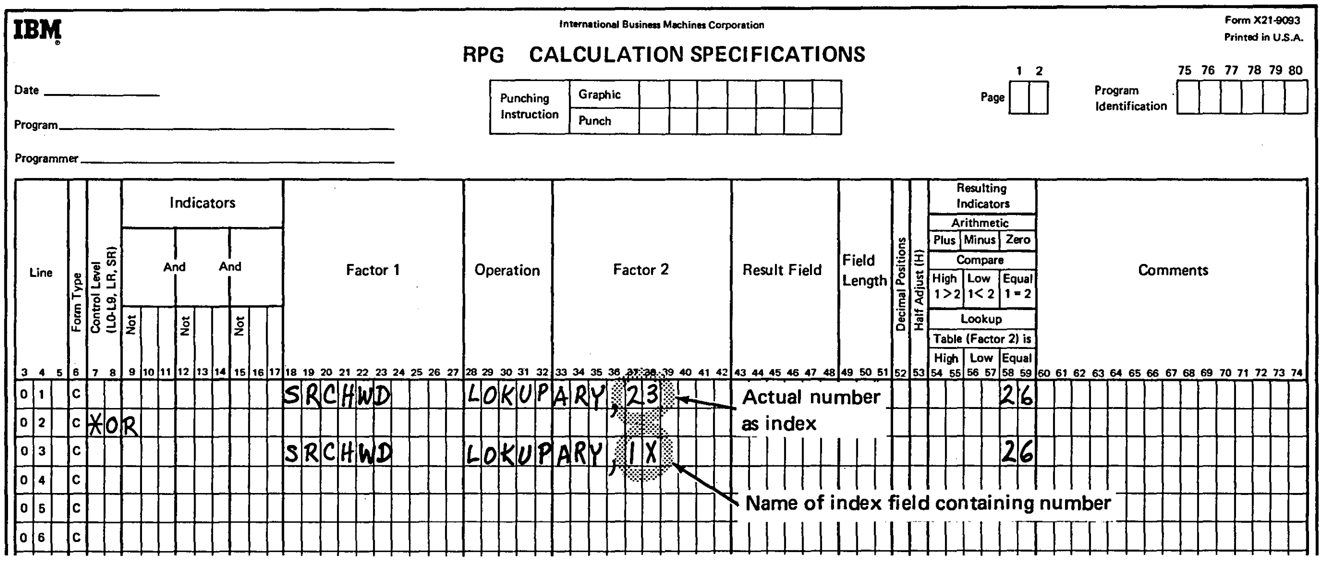
With arrays, however, you also have the capability of beginning an array search at any field you specify. Under Factor 2, you specify the array name, followed by a comma and an index. The index, whether an actual number or the name of a field containing a number, points to the array field where the search is to begin (Figure 15-31).

In a large array where you know that the value you are searching for is not in a particular section of the array, search time can be greatly decreased by beginning the lookup at a particular field. Suppose you have a 300-field array name ARY containing the values 001 through 300 in ascending sequence. To locate a value of 047, only 47 fields would have to be checked before the search condition was satisfied. However, to locate the value 289, 289 fields would have to be checked, *if* the search began at the first array field.

Now, divide the array into three parts of 100 fields each:

Fields 1-100:	values 001-100.
Fields 101-200:	values 101-200.
Fields 201-300:	values 201-300.

For any value of less than 101, the first third of the array is searched, beginning at field 1. For values greater than 100, but less than 201, the second third of the array is searched, beginning at field 101. Likewise, a search is started at field 201 to locate any value greater than 200. In any case, no more than 100 fields have to be checked to satisfy the search condition.



IBM

International Business Machines Corporation

RPG CALCULATION SPECIFICATIONS

Form X21-9093
Printed in U.S.A.

Date ____________

Program ____________

Programmer ____________

Punching Instruction: Graphic / Punch

Page 1 2

Program Identification 75 76 77 78 79 80

Line (3-5)	Form Type (6)	Control Level (L0-L9, LR, SR) (7-8)	Indicators (9-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field (43-48)	Field Length (49-51)	Decimal Positions (52)	Half Adjust (H) (53)	Resulting Indicators High 1>2 (54-55)	Low 1<2 (56-57)	Equal 1=2 (58-59)	Comments (60-74)
01	C			SRCHWD	LOKUP	ARY,23							26	
02	C	*OR												
03	C			SRCHWD	LOKUP	ARY,IX							26	
04	C													
05	C													
06	C													

Figure 15-31. Starting an Array Search at a Particular Field

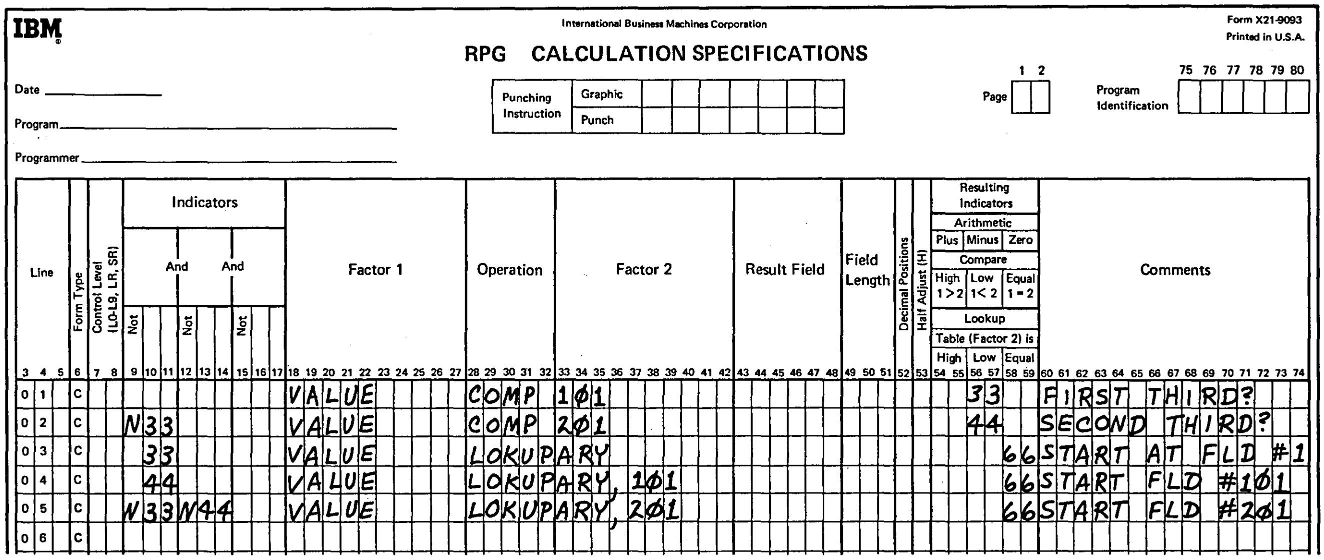
For this example, the number of the array field at which the search is to begin will vary, depending on the value being searched for. Figure 15-32 shows that three LOKUP's have been coded. Only one of the lookup operations is performed for a particular value.

To determine which LOKUP (line 03, 04, or 05) is performed, you must first determine in which part of the array the value is located. The first COMP (compare) operation (line 01) checks for a value in the first 100 fields. If the value is less than 101, indicating the first one third of the array, indicator 33 is set on. If 33 is on, the LOKUP beginning at field 1 is performed (line 03). However, if the value is not in the first third of the array (33 off), another compare (line 02) is necessary to determine if the value is in the second third of the array (indicator 44 set on). Thus, the LOKUP beginning at field 101 (line 04) is performed with indicator 44 on. If neither indicator (33 or 44) was set on, the value must be in the last third of the array, if it is in the array at all. Therefore, with both 33 and 44 off, the LOKUP beginning at field 201 is performed.

For the first LOKUP (line 03), it is not necessary to actually specify the numeric value 1 as the index, in the same manner as 101 is specified for the second LOKUP. When no index is specified with the array name, the search automatically begins at the first field, as if the index were 1.

If the value of the index changes, as in this case, you can use an index field to contain the number of the array field, rather than using the actual number. In this way, it is necessary to code only one LOKUP. Of course, you must place the appropriate number in the index field every time before the lookup operation is performed. Thus an index field will not always reduce the number of specifications required.

As shown in Figure 15-33, first the compare operations are performed to determine whether the value is in the first, second, or last third of the array. The results of the compare operations determine which number should be zero-added into the index field, ISFLD, before the lookup is performed.



IBM International Business Machines Corporation

RPG CALCULATION SPECIFICATIONS

Form X21-9093
Printed in U.S.A.

Date

Program

Programmer

Punching Instruction: Graphic / Punch

Page (1 2)

Program Identification (75 76 77 78 79 80)

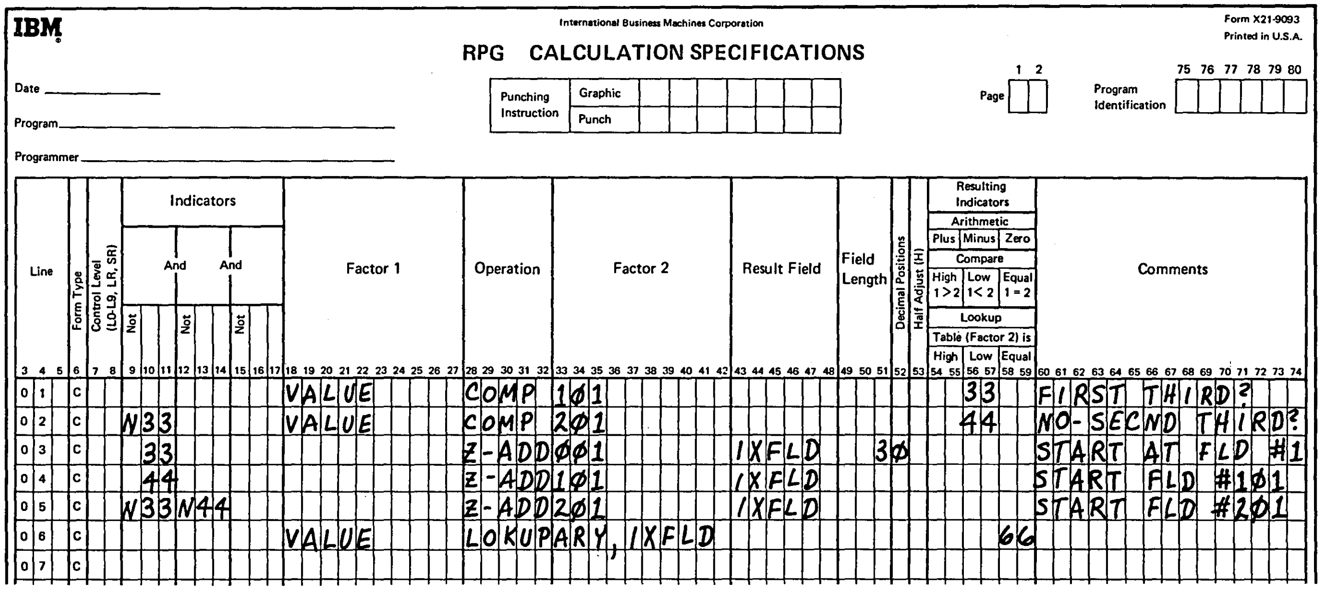
Line (3-5)	Form Type (6)	Control Level (L0-L9, LR, SR) (7-8)	Indicators: Not (9-11)	And: Not (12-14)	And: Not (15-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field (43-48)	Field Length (49-51)	Decimal Positions (52)	Half Adjust (H) (53)	Resulting Indicators: Arithmetic Plus / Compare High 1>2 / Lookup High (54-55)	Minus / Low 1<2 / Low (56-57)	Zero / Equal 1=2 / Equal (58-59)	Comments (60-74)
01	C					VALUE	COMP	1Ø1						33		FIRST THIRD?
02	C		N33			VALUE	COMP	2Ø1						44		SECOND THIRD?
03	C		33			VALUE	LOKUP	ARY							66	START AT FLD #1
04	C		44			VALUE	LOKUP	ARY,1Ø1							66	START FLD #1Ø1
05	C		N33	N44		VALUE	LOKUP	ARY,2Ø1							66	START FLD #2Ø1
06	C															

Figure 15-32. LOKUP with an Actual Index

Determining if a Search Is Successful

At this point, we should discuss the index field and how its contents are changed as a result of the lookup operation. Before the lookup is performed, you determine the value which is to be placed in the index field. The array search then begins at the field number specified. The array lookup continues, one field at a time, until the search condition is satisfied or the end of the array has been reached, whichever occurs first. If an index field is specified, the number of the array field first satisfying the search condition is stored in the index field. However, if the end of the array is reached and none of the fields satisfy the search, a *1* is placed in the index field. In any case, if an actual number, not an index field, is specified as the index, the actual index is not changed to reflect the success of the search.

The way in which you determine a successful search is whether the resulting indicator assigned has been turned on or off. Thus, if the resulting indicator is off and an index field had been specified, the index field should contain the value *1,* the result of an unsuccessful search. Note, however, that the contents of the index field alone cannot indicate that a search was unsuccessful. If the first field of an array satisfied the search condition, the index field would also contain the value *1;* however, in such a case, the resulting indicator would be on.



IBM

International Business Machines Corporation

Form X21-9093
Printed in U.S.A.

RPG CALCULATION SPECIFICATIONS

Date

Program

Programmer

Punching Instruction: Graphic / Punch

Page (1 2)

Program Identification (75 76 77 78 79 80)

Line (3-5)	Form Type (6)	Control Level (L0-L9, LR, SR) (7-8)	Indicators (9-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field (43-48)	Field Length (49-51)	Decimal Positions (52)	Half Adjust (H) (53)	Resulting Indicators: Plus / High 1>2 / High (54-55)	Minus / Low 1<2 / Low (56-57)	Zero / Equal 1=2 / Equal (58-59)	Comments (60-74)
01	C			VALUE	COMP	1Ø1						33		FIRST THIRD?
02	C		N33	VALUE	COMP	2Ø1						44		NO-SECND THIRD?
03	C		33		Z-ADD	ØØ1	IXFLD	3	Ø					START AT FLD #1
04	C		44		Z-ADD	1Ø1	IXFLD							START FLD #1Ø1
05	C		N33N44		Z-ADD	2Ø1	IXFLD							START FLD #2Ø1
06	C			VALUE	LOKUP	ARY,IXFLD							66	
07	C													

Figure 15-33. LOKUP with an Index Field

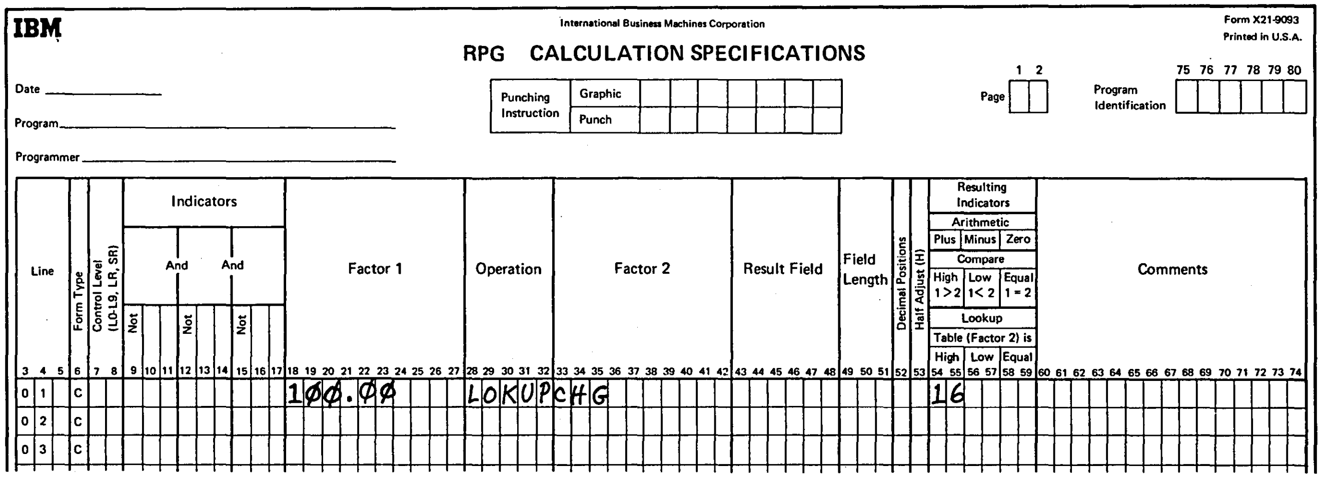
Referencing a Field Which Satisfies a Search

After a successful search, you can use the data from the field which satisfied the condition only if the array name *with an index field* is specified in the LOKUP specifications. If an index field is specified, the number of the field which satisfied the search is stored in the index field. Therefore, specifying the array name with the index field in a subsequent calculation or output specification refers to the field which satisfied the search.

However, if no index field is available (array name specified alone or with a numeric index), the number of the field cannot be determined and, therefore, the data cannot be referenced. You can only determine if one of the array fields does contain the data for which you searched, according to the on-off status of the resulting indicator.

The ability to reference a data item which satisfies a search is one of the major differences between an array lookup and a table lookup. During a table lookup, when an element is found which satisfies the search, the element is saved in a special hold area related to the table. Thereafter, using the table name alone refers to the contents of the hold area (in other words, to the data item which satisfied the search). There is no hold area associated with an array, however. Thus, following a lookup, specifying the array name alone refers to the entire array, rather than to any particular field. The only way an individual array field can be referenced is by specifying the array name with an index.

Assume you wish to search an array CHG to check for amounts over $100. If you only want to determine if there are any fields containing a greater amount, the search can be coded as shown in Figure 15-34. If indicator 16 is on, indicating a successful search, you can then print a message stating there is a charge over $100. Otherwise, if indicator 16 is off, you can print a message stating all charges are under $100. With the LOKUP specification shown, however, you would have no way of knowing how many fields or which fields satisfied the search condition.



IBM

International Business Machines Corporation

RPG CALCULATION SPECIFICATIONS

Form X21-9093
Printed in U.S.A.

Date ____________

Program ____________

Programmer ____________

Punching Instruction: Graphic / Punch

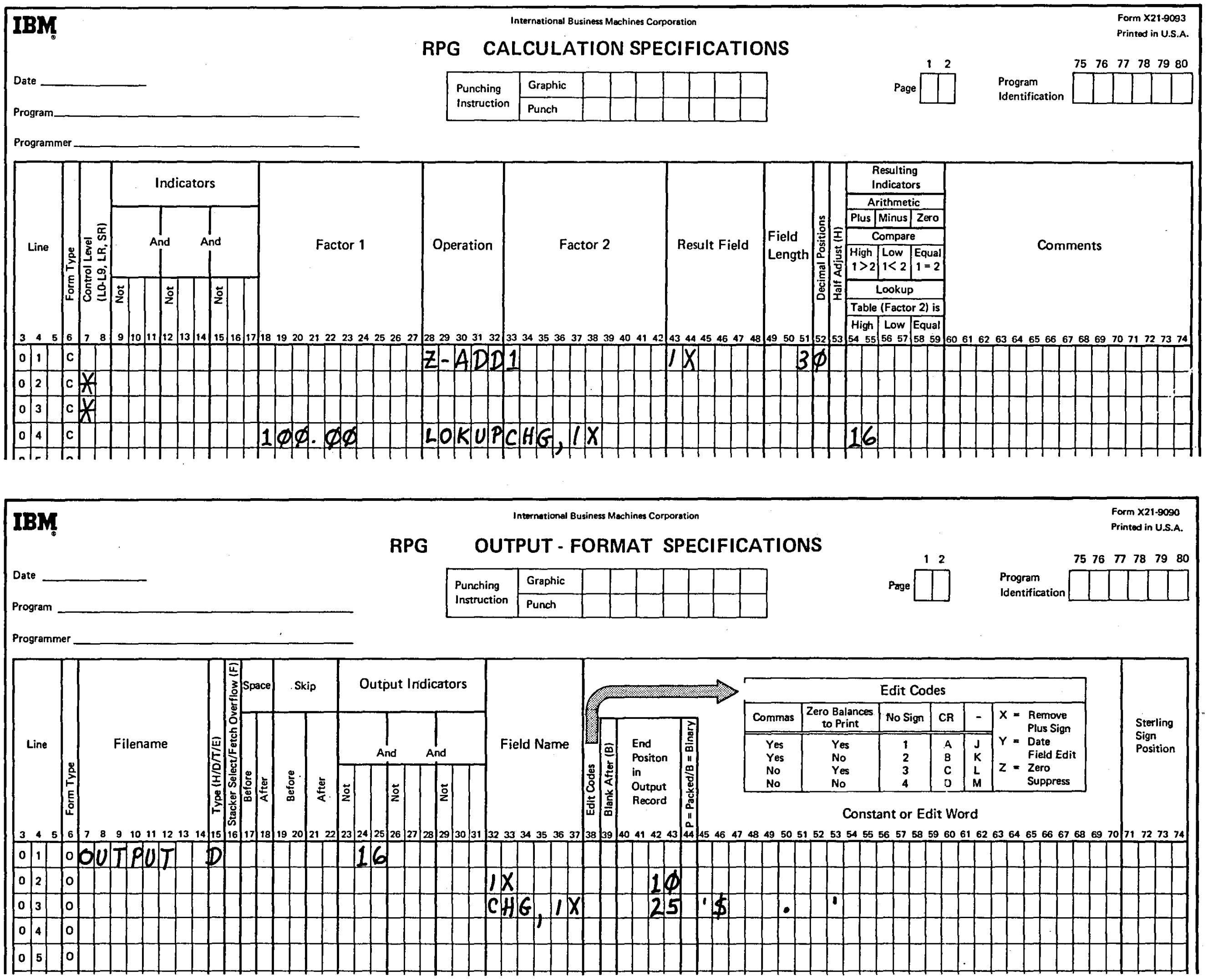
Page 1 2

Program Identification 75 76 77 78 79 80

Line (3-5)	Form Type (6)	Control Level (L0-L9, LR, SR) (7-8)	Indicators (9-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field (43-48)	Field Length (49-51)	Decimal Positions (52)	Half Adjust (H) (53)	Resulting Indicators: Plus / Compare High 1>2 / Lookup High (54-55)	Minus / Low 1<2 / Low (56-57)	Zero / Equal 1=2 / Equal (58-59)	Comments (60-74)
01	C			1ØØ.ØØ	LOKUP	CHG					16			
02	C													
03	C													

Figure 15-34. Determining Only if a Search Is Successful

If you wish to know which field satisfied the search or, perhaps, how much over $100 the amount is, the array lookup should be coded with an index field (Figure 15-35). The index field can be preset to contain the value *1,* so the search begins at the first field of the array. If the search is satisfied, IX will contain the number of the first field over $100; and the resulting indicator will be turned on. The contents of IX can then be printed to indicate which field satisfied the search. The actual contents of that field can be printed by specifying the array name with the index field (Figure 15-35, Output Sheet).



IBM International Business Machines Corporation Form X21-9093 Printed in U.S.A.

RPG CALCULATION SPECIFICATIONS

Date ______ Program ______ Programmer ______ Punching Instruction: Graphic / Punch. Page 1 2. Program Identification 75 76 77 78 79 80

Line (3-5)	Form Type (6)	Control Level (L0-L9, LR, SR) (7-8)	Indicators (9-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field (43-48)	Field Length (49-51)	Decimal Positions (52)	Half Adjust (H) (53)	Resulting Indicators (54-59)	Comments (60-74)
01	C				Z-ADD	1	IX	3	0			
02	C	X										
03	C	X										
04	C			100.00	LOKUP	CHG,IX					16	

IBM International Business Machines Corporation Form X21-9090 Printed in U.S.A.

RPG OUTPUT - FORMAT SPECIFICATIONS

Date ______ Program ______ Programmer ______ Punching Instruction: Graphic / Punch. Page 1 2. Program Identification 75 76 77 78 79 80

Line (3-5)	Form Type (6)	Filename (7-14)	Type (H/D/T/E) (15)	Space / Skip (17-22)	Output Indicators (23-31)	Field Name (32-37)	Edit Codes (38)	Blank After (B) (39)	End Position in Output Record (40-43)	P = Packed/B = Binary (44)	Constant or Edit Word (45-70)	Sterling Sign Position (71-74)
01	O	OUTPUT	D		16							
02	O					IX			10			
03	O					CHG,IX			25		'$. '	
04	O											
05	O											

Edit Codes: Commas / Zero Balances to Print / No Sign / CR / -: Yes Yes 1 A J; Yes No 2 B K; No Yes 3 C L; No No 4 D M. X = Remove Plus Sign; Y = Date Field Edit; Z = Zero Suppress

Figure 15-35. Determining Which Array Field Satisfies the Search

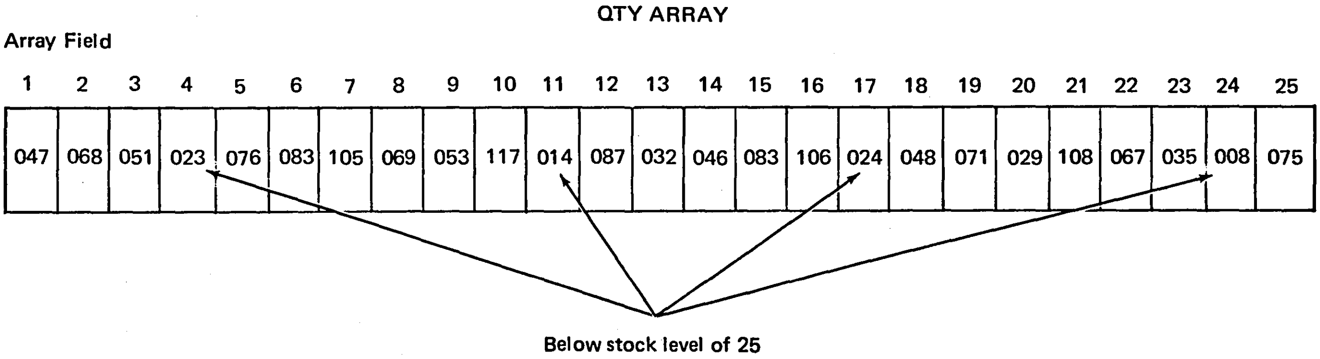
Searching An Array for More Than One Field

The previous example points out an important consideration: an array LOKUP operation is completed when the first field is found which satisfies the search condition. If you wish to find all fields which satisfy the condition, you must code additional specifications which cause the program to loop back in calculations to repeat the lookup operation from the point where the last search was successful.

As example, assume your company manufactures 25 different items, identified by item codes 1-25. A 25-field array QTY (Figure 15-36) is used to keep track of the quantity in stock of each item. The first field contains the quantity of item code *1,* the second field contains the quantity of item code *2,* and so on.

Whenever the quantity of an item falls below 25, 100 items are to be manufactured and added to stock. To determine which items are to be manufactured, every week the QTY array is searched, comparing the array fields with a search word (MFGPT) of 25 from a data card. When a quantity is found to be less than 25 (search condition Low), the item code and quantity in stock are printed.

From Figure 15-36 you can see that four items must be manufactured. The specifications in Figure 15-37 will not locate all of the items with quantities less than 25. The lookup operation shown will locate only the first quantity below 25. If only one data card (containing the search word) is read, the specifications are performed once. As you know, for every data card read, the program cycle is repeated. However, even if several data cards with the same search word are read, every time the lookup is repeated, the search begins again at field *1* of the array. Therefore, the same array field satisfies the search every time, and the other three quantities are never found.



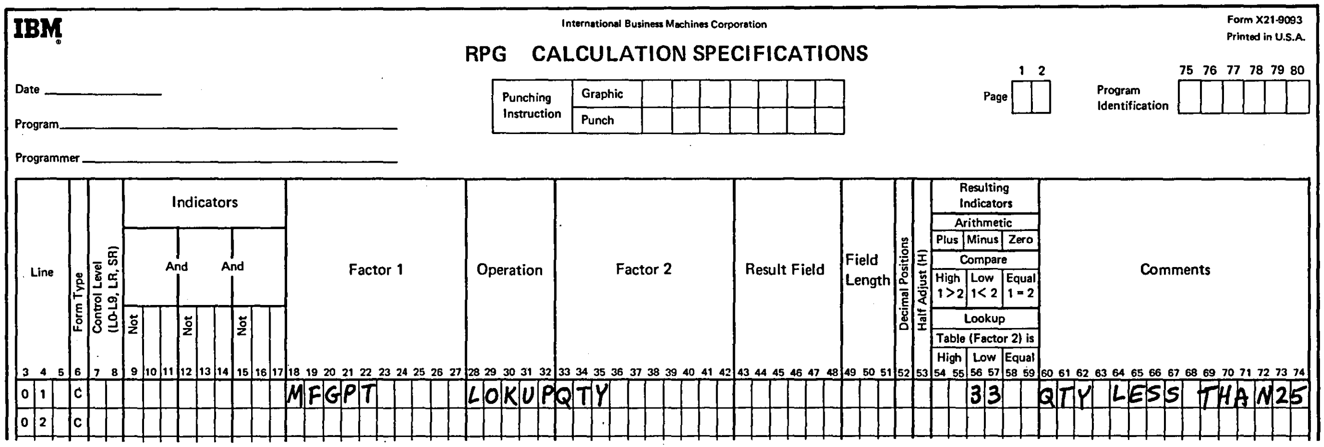
QTY ARRAY

Array Field

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
047	068	051	023	076	083	105	069	053	117	014	087	032	046	083	106	024	048	071	029	108	067	035	008	075

Below stock level of 25

Figure 15-36. More than One Array Field Which Satisfies the Search Condition



IBM

International Business Machines Corporation

RPG CALCULATION SPECIFICATIONS

Form X21-9093
Printed in U.S.A.

Date

Program

Programmer

Punching Instruction: Graphic / Punch

Page 1 2

Program Identification 75 76 77 78 79 80

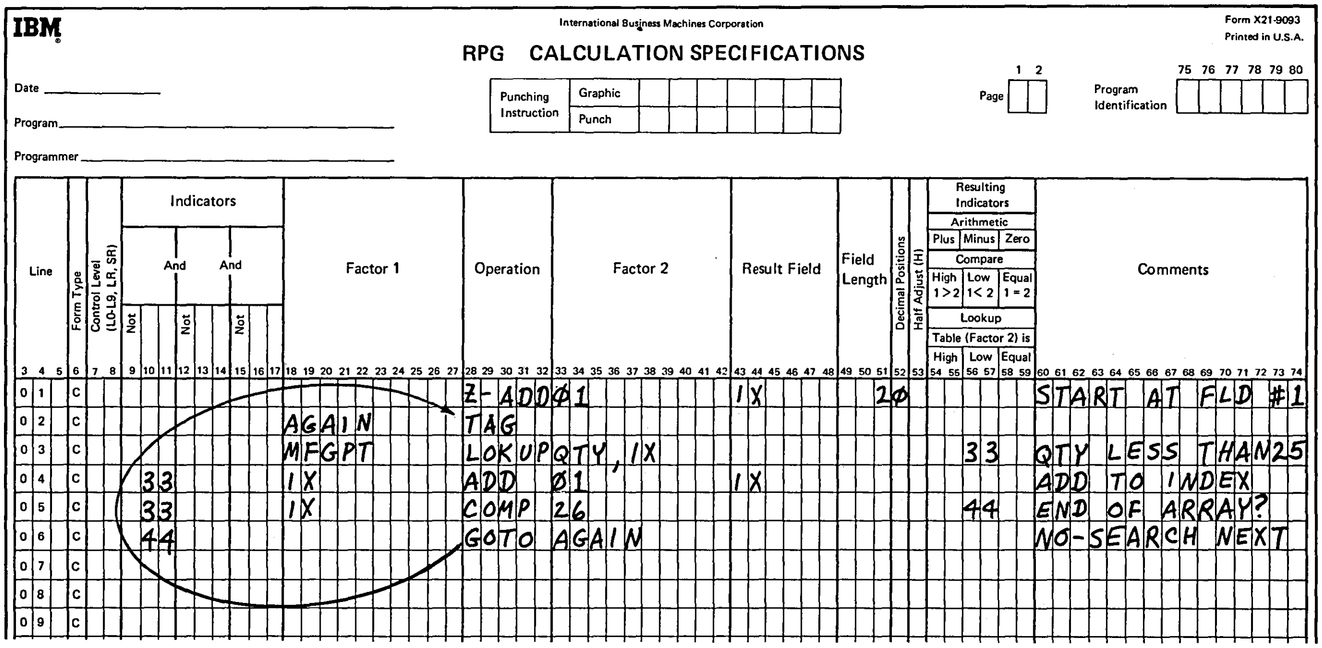
Line	Form Type	Control Level (L0-L9, LR, SR)	Indicators	Factor 1	Operation	Factor 2	Result Field	Field Length	Decimal Positions	Half Adjust (H)	Resulting Indicators High	Low	Equal	Comments
01	C			MFGPT	LOKUP	QTY						33		QTY LESS THAN25
02	C													

Figure 15-37. Array Search Which Locates Only One Field

To locate more than one field satisfying the same search condition, the LOKUP must be repeated within a single program cycle. Not only must the LOKUP be repeated, but the search must begin at the point where the previous search ended. You can repeat the LOKUP using the GOTO and TAG operations, as shown in Figure 15-38. To make sure the repeated search begins where the last search left off, you must specify the array name *with an index field* in the LOKUP specification. The contents of the index field is then updated after each successful search to indicate at which array field the next search should begin.

The first search should begin at field *1*. Thus, as Figure 15-38 shows, the index field IX is initially set up to contain the value *1*. (The field is zeroed before adding *1*, since you have no way of knowing the contents of IX at the beginning of the program run.) The TAG operation is not performed; therefore, the computer skips this specification and performs the LOKUP.

When the first QTY field less than 25 is found, the number of the field (04) is placed in the index field. Providing the LOKUP was successful (33 on), a *1* is added to the value in the index field, to indicate at which field the next search should begin. The value in the index field is then compared to 26 to see if the entire array (25 fields) has been searched. If there are still array fields to be checked (indicator 44 on), the program branches back (GOTO) to perform the LOKUP again. The search would then begin again, only at the field following the last field which satisfied the search. The calculation specifications would be repeated over and over until all items to be manufactured are located and until the end of the array is reached.



IBM

International Business Machines Corporation

RPG CALCULATION SPECIFICATIONS

Form X21-9093
Printed in U.S.A.

Date

Program

Programmer

Punching Instruction: Graphic / Punch

Page 1 2

Program Identification 75 76 77 78 79 80

Line (3-5)	Form Type (6)	Control Level (L0-L9, LR, SR) (7-8)	Indicators: Not (9)	(10-11)	And: Not (12)	(13-14)	And: Not (15)	(16-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field (43-48)	Field Length (49-51)	Decimal Positions (52)	Half Adjust (H) (53)	Resulting Indicators: Arithmetic Plus / Compare High 1>2 / Lookup High (54-55)	Minus / Low 1<2 / Low (56-57)	Zero / Equal 1=2 / Equal (58-59)	Comments (60-74)
01	C									Z-ADD	Ø1	IX	2	Ø					START AT FLD #1
02	C								AGAIN	TAG									
03	C								MFGPT	LOKUP	QTY,IX						33		QTY LESS THAN25
04	C			33					IX	ADD	Ø1	IX							ADD TO INDEX
05	C			33					IX	COMP	26						44		END OF ARRAY?
06	C			44						GOTO	AGAIN								NO-SEARCH NEXT
07	C																		
08	C																		
09	C																		

Figure 15-38. Repeating a LOKUP to Locate to Locate All Array Fields Satisfying the Search Condition

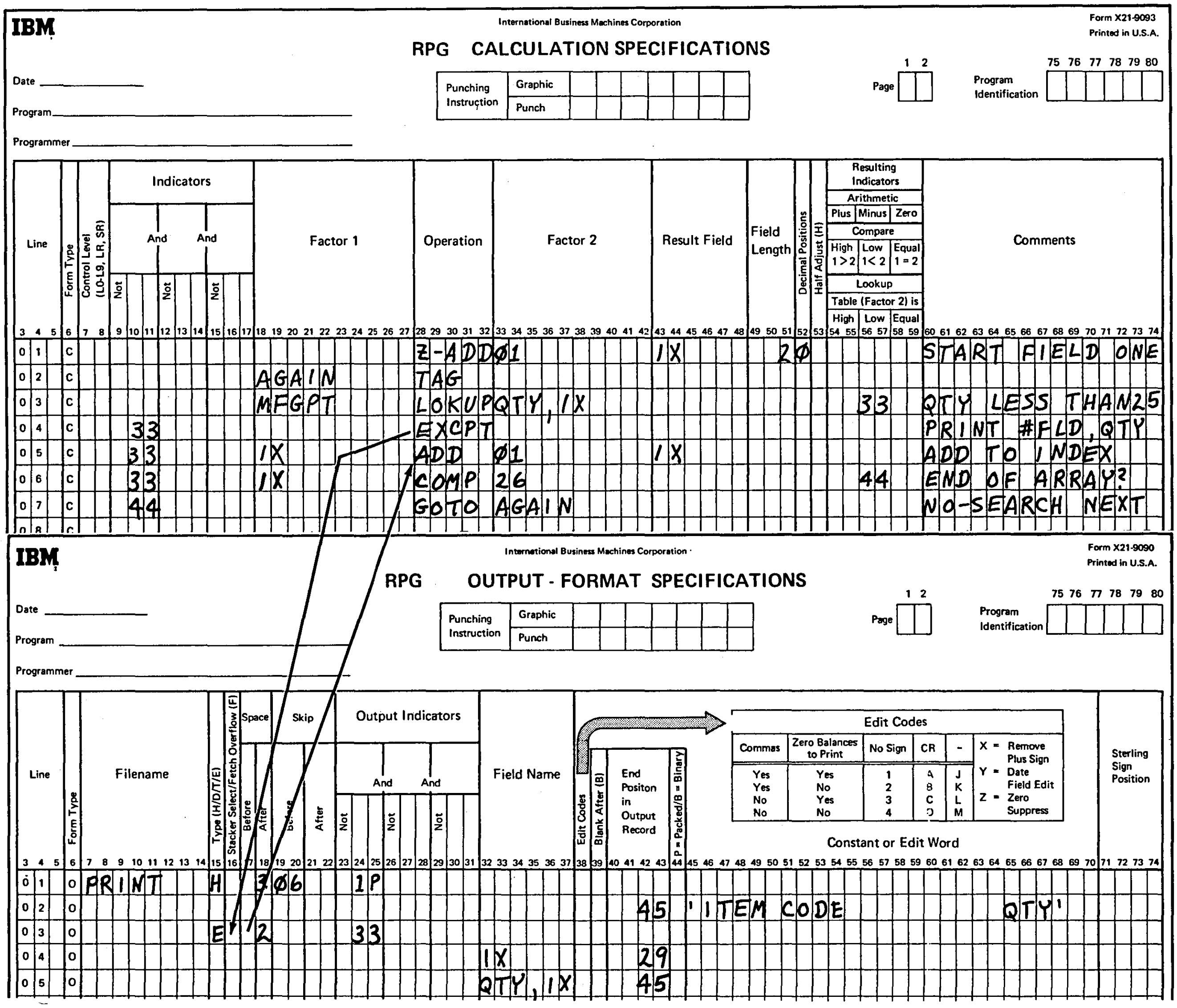
Output During an Array Search

The specifications in Figure 15-39 search through the QTY array to locate more than one field. In this case, it does no good to search through an array unless you know what data was found. For this reason, each quantity less than 25 and its related item code are printed. Following each successful search, the item code number (same as the number of the array field containing the quantity) is stored in the index field IX. Thus, the field IX can be printed. The acutal quantity which satisfied the search can be printed by specifying the array name with the index field in the output specification.

Since the output specifications usually are not performed until all calculations are done, normal output would cause only the last field number located plus the value one (Figure 15-38, line 04) and the contents of that field (quantity) to be printed.

In order to print each item code (field number) located in the array search (and its quantity), output must be done before the contents of the index field are changed.

You have learned that using the EXCPT operation on the Calculation Sheet makes it possible to perform output specifications before calculations are finished and then to



IBM International Business Machines Corporation Form X21-9093 Printed in U.S.A.

RPG CALCULATION SPECIFICATIONS

Line	Form Type	Indicators	Factor 1	Operation	Factor 2	Result Field	Field Length	Decimal Positions	Resulting Indicators High	Low	Equal	Comments
01	C			Z-ADD	Ø1	IX	2	Ø				START FIELD ONE
02	C		AGAIN	TAG								
03	C		MFGPT	LOKUP	QTY,IX					33		QTY LESS THAN25
04	C	33		EXCPT								PRINT #FLD,QTY
05	C	33	IX	ADD	Ø1	IX						ADD TO INDEX
06	C	33	IX	COMP	26					44		END OF ARRAY?
07	C	44		GOTO	AGAIN							NO-SEARCH NEXT
08	C											

IBM International Business Machines Corporation Form X21-9090 Printed in U.S.A.

RPG OUTPUT - FORMAT SPECIFICATIONS

Line	Form Type	Filename	Type (H/D/T/E)	Space After	Skip Before	Output Indicators	Field Name	End Position in Output Record	Constant or Edit Word
01	O	PRINT	H	3	Ø6	1P			
02	O							45	'ITEM CODE QTY'
03	O		E	2		33			
04	O						IX	29	
05	O						QTY,IX	45	

Figure 15-39. Output of Array Field as it is Located in the Search

branch back to finish the calculation operations. As Figure 15-39 shows, following a successful LOKUP, the EXCPT operation then causes the data placed in the index field to be printed, followed by the contents of the array field which satisfied the search. After the exception output has been performed (output lines identified by an E in column 15), the program continues with the calculation specifications, by performing the array lookup again.

DESCRIBING DATA AND STORING IT IN AN ARRAY

Now that you understand how arrays can be used, you must be able to get the data into an array so that you can reference the information. Once an array is defined, a place is reserved in storage for the fields, and the array data can be stored. Data to be stored in an array can be either punched on cards which are read into the computer at the time your object program is executed, or created from calculations performed during the job and then stored in the array. Creation of array information during a program has been discussed under referencing fields of an array.

Fields of array data to be read from input records must be described on the Input Sheet. The input specifications indicate where the data is located on the record. The way in which the data is described determines whether the data is automatically stored in the array fields or whether you must code calculation specifications to MOVE the data into the array. How the array information is described, and thus stored, depends on two factors:

1. How the array data is organized on a record.

2. Whether the data for an array is contained in one or more records.

An input record containing array data can contain only data for that array or can contain both array data and other data fields to be used in the program. In either case, the array data is organized in one of two ways:

1. All array fields may occupy consecutive positions on the record; that is, each field immediately following another with no blanks or other data between the fields.

2. The array fields may be scattered on the record, in any order, with blanks or normal input fields placed between the array fields.

The way in which the data is organized and the size of the array generally determines the number of input records required to contain the array data.

Entire Array Data On One Record

Array Data in Consecutive Positions

If array fields are in order in consecutive positions on a record, describing and storing the data is very easy. All of the array data on the one record may be described on the Input Sheet as if it were a single field. Thus, only one input specification is necessary to indicate a name for the field and the columns on the record where the array data begins and ends (Figure 15-40). By specifying the name of the array as the field name, the data is automatically stored in the appropriate fields of the array as the input record is read.

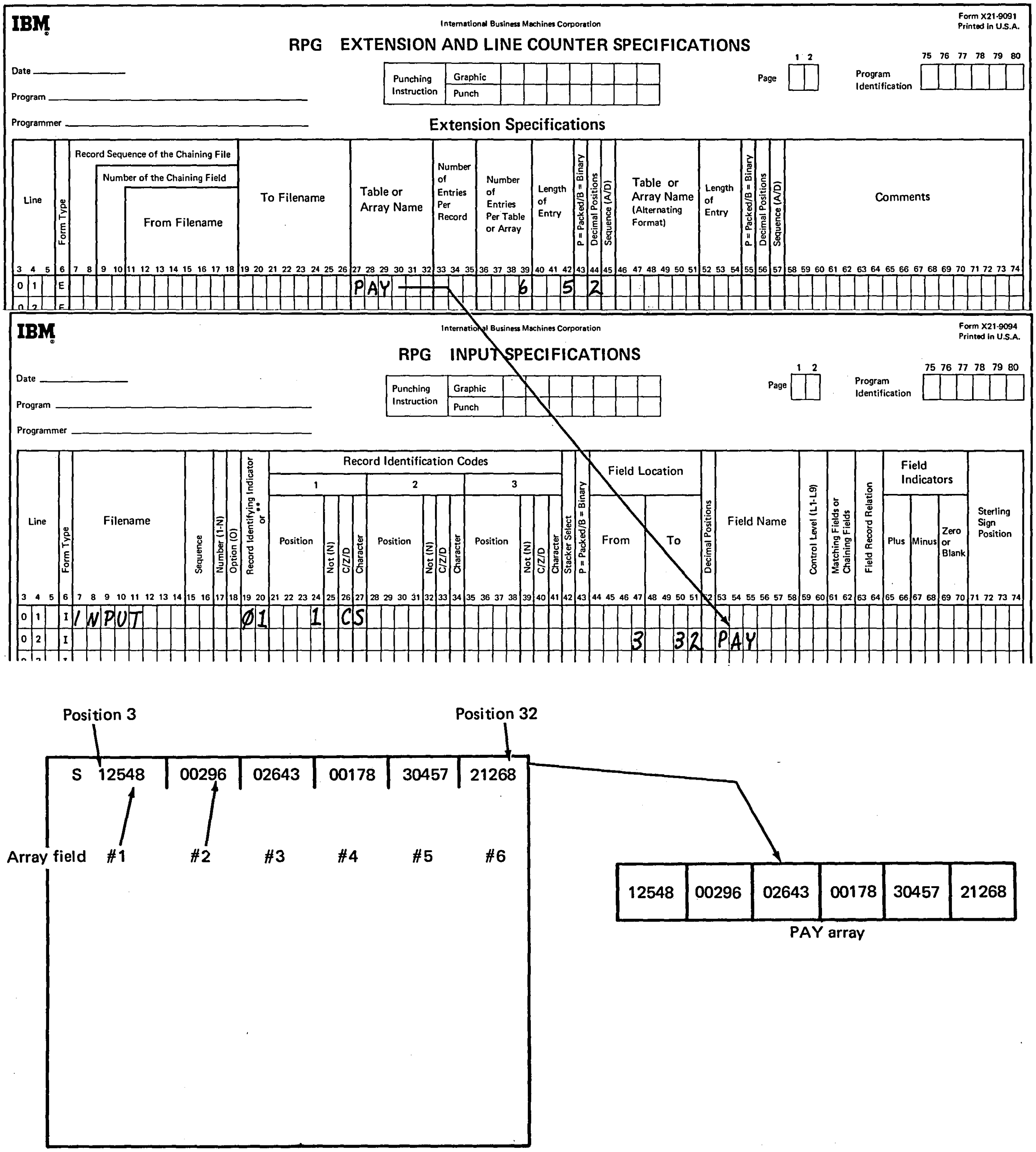
When you describe an input record of array data, you specify no entry in column 52 (Decimal Positions) of the Input Sheet. Since the array name and characteristics have been previously defined on the Extension Sheet, the entry in column 44 of the Extension Sheet indicates the number of decimal positions in each array field.

Array Data Scattered on a Record

When array fields are scattered on an input record, the data cannot be automatically stored in an array. Each field must be described separately on the Input Sheet to indicate where each item of array data begins and ends. Calculation specifications are then necessary to individually move each field of data into the appropriate field of the array.

Assume that a 6-field array named EMP is set up by coding extension specifications. The six fields of data for the array are scattered on a record, as shown in Figure 15-41. Additional input information (blanks and other input fields) is recorded between the array fields. Furthermore, the array fields are not in the order in which they are to be stored.

When you describe the array data, you must identify each field by a separate line of input specifications, with each field given a unique field name. This is necessary because the array data is not continuous. As Figure 15-41 shows, normal input fields can be described along with the array fields.



IBM International Business Machines Corporation Form X21-9091 Printed in U.S.A.

RPG EXTENSION AND LINE COUNTER SPECIFICATIONS

Date ____ Program ____ Programmer ____

Punching Instruction: Graphic / Punch   Page 1 2   Program Identification 75 76 77 78 79 80

Extension Specifications

Line (3-5)	Form Type (6)	Record Sequence of the Chaining File (7-8)	Number of the Chaining Field (9-10)	From Filename (11-18)	To Filename (19-26)	Table or Array Name (27-32)	Number of Entries Per Record (33-35)	Number of Entries Per Table or Array (36-39)	Length of Entry (40-42)	P = Packed/B = Binary (43)	Decimal Positions (44)	Sequence (A/D) (45)	Table or Array Name (Alternating Format) (46-51)	Length of Entry (52-54)	P = Packed/B = Binary (55)	Decimal Positions (56)	Sequence (A/D) (57)	Comments (58-74)
01	E					PAY		6	5		2							
02	E																	

IBM International Business Machines Corporation Form X21-9094 Printed in U.S.A.

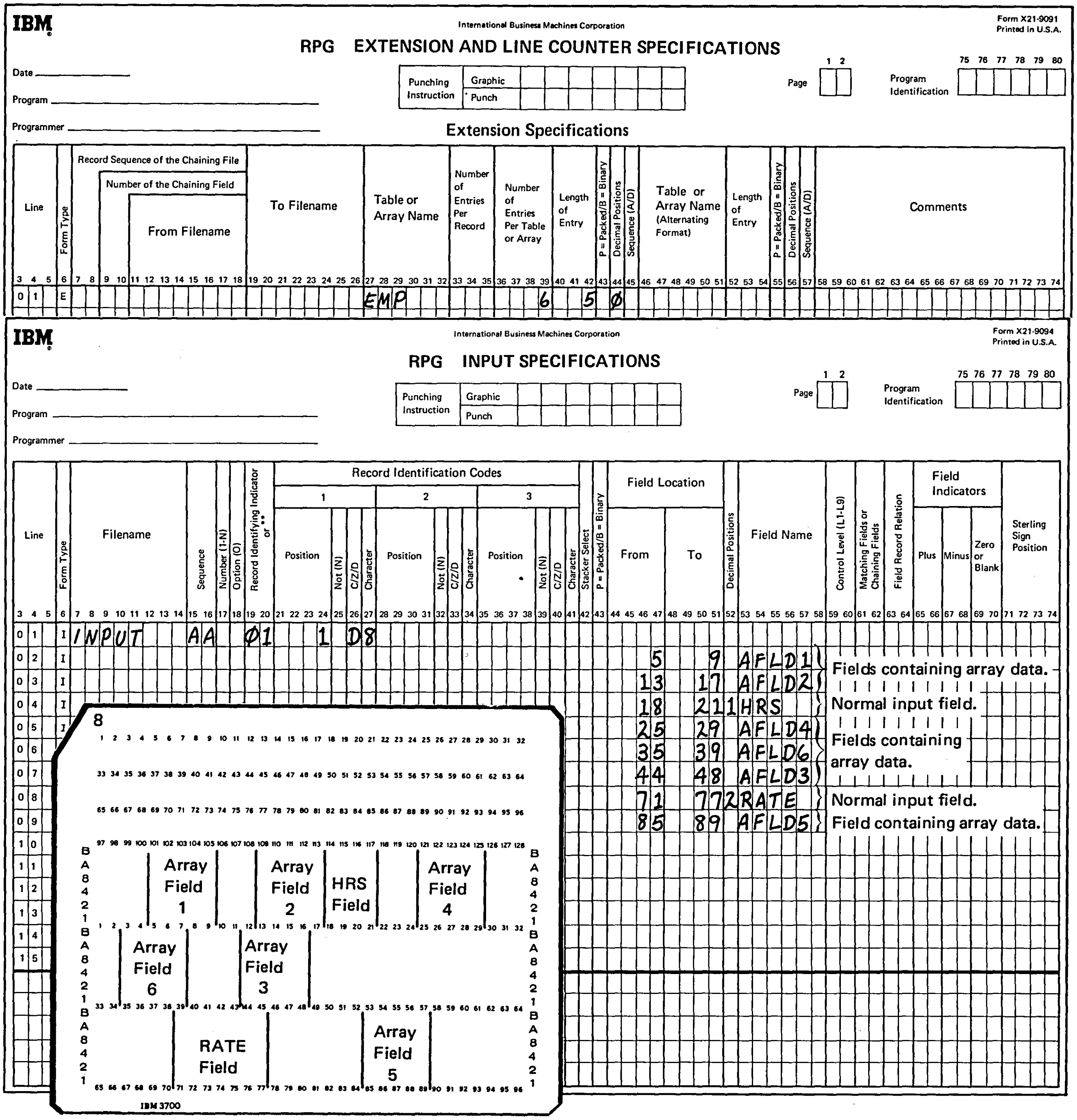
RPG INPUT SPECIFICATIONS

Date ____ Program ____ Programmer ____

Punching Instruction: Graphic / Punch   Page 1 2   Program Identification 75 76 77 78 79 80

Line (3-5)	Form Type (6)	Filename (7-14)	Sequence (15-16)	Number (1-N) (17)	Option (O) (18)	Record Identifying Indicator or ** (19-20)	Position (21-24)	Not (N) (25)	C/Z/D (26)	Character (27)	Position (28-31)	Not (N) (32)	C/Z/D (33)	Character (34)	Position (35-38)	Not (N) (39)	C/Z/D (40)	Character (41)	Stacker Select (42)	P = Packed/B = Binary (43)	Field Location From (44-47)	Field Location To (48-51)	Decimal Positions (52)	Field Name (53-58)	Control Level (L1-L9) (59-60)	Matching Fields or Chaining Fields (61-62)	Field Record Relation (63-64)	Field Indicators Plus (65-66)	Minus (67-68)	Zero or Blank (69-70)	Sterling Sign Position (71-74)
01	I	INPUT				Ø1	1		C	S																					
02	I																				3	32		PAY							

Figure 15-40. Storing Data in an Array



IBM International Business Machines Corporation Form X21-9091 Printed in U.S.A.

RPG EXTENSION AND LINE COUNTER SPECIFICATIONS

Date
Program
Programmer

Extension Specifications

Line	Form Type	Record Sequence of the Chaining File	Number of the Chaining Field	From Filename	To Filename	Table or Array Name	Number of Entries Per Record	Number of Entries Per Table or Array	Length of Entry	P = Packed/B = Binary	Decimal Positions	Sequence (A/D)	Table or Array Name (Alternating Format)	Length of Entry	P = Packed/B = Binary	Decimal Positions	Sequence (A/D)	Comments
01	E					EMP		6	5		Ø							

IBM International Business Machines Corporation Form X21-9094 Printed in U.S.A.

RPG INPUT SPECIFICATIONS

Date
Program
Programmer

Line	Form Type	Filename	Sequence	Number (1-N)	Option (O)	Record Identifying Indicator or **	Position	Not (N)	C/Z/D	Character	Field Location From	Field Location To	Decimal Positions	Field Name	
01	I	INPUT	AA			Ø1	1		D	8					
02	I										5	9		AFLD1	Fields containing array data.
03	I										13	17		AFLD2	
04	I										18	21	1	HRS	Normal input field.
05	I										25	29		AFLD4	Fields containing array data.
06											35	39		AFLD6	
07											44	48		AFLD3	
08											71	77	2	RATE	Normal input field.
09											85	89		AFLD5	Field containing array data.
10															
11															
12															
13															
14															
15															

Figure 15-41. Array Data Scattered on a Record

Once the scattered fields have been described on the Input Sheet, each field of array data is stored in the array using a MOVE calculation (Figure 15-42). Since each field has a unique field name and must be stored in a specific array field, a separate move specification must be coded for each field to be stored.

The specifications which move the array data into the array fields should generally be specified first on the Calculation Sheet. This is necessary to ensure that the data will be in the array when any calculations on the array (specified later on the Calculation Sheet) are performed.

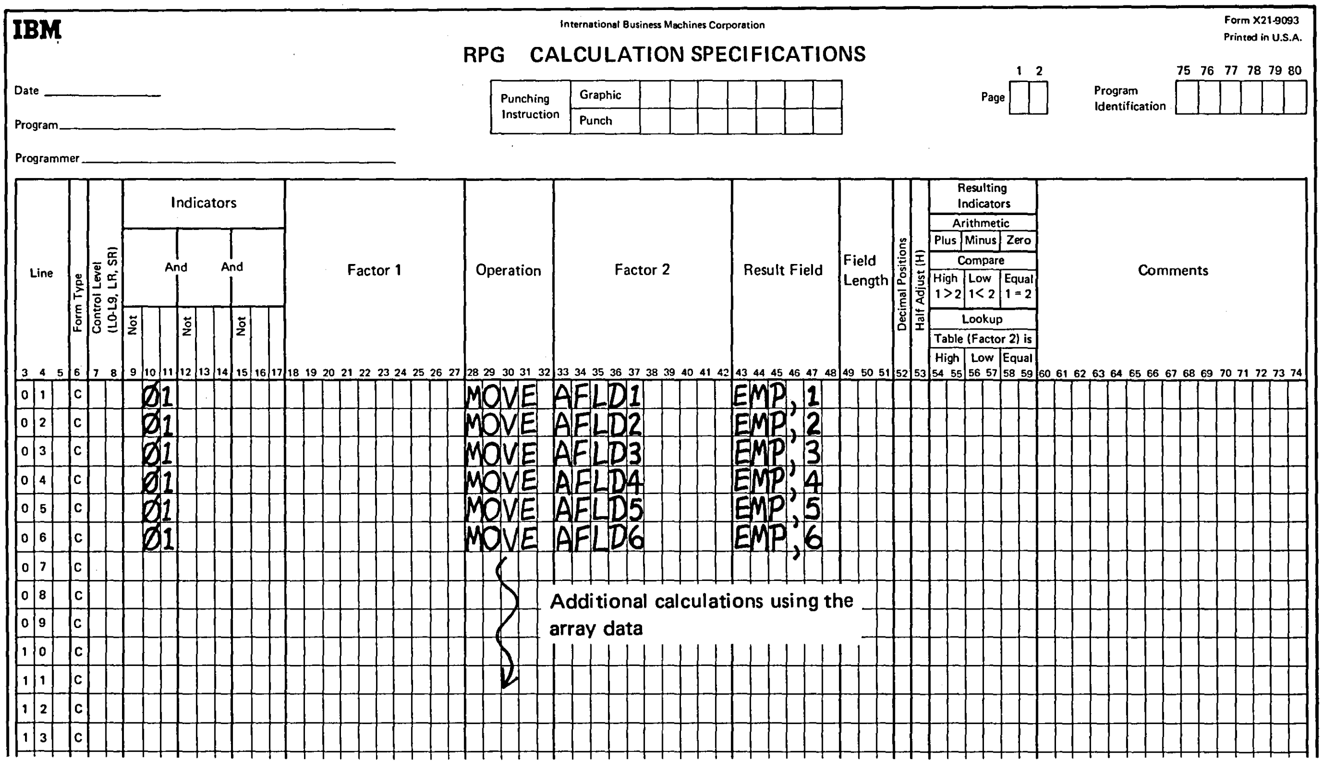
Note that in the input specifications in Figure 15-42, the consecutive array fields were stored directly into the PAY array by specifying the array name (PAY) as the field name. The array fields are not stored directly into the individual fields of the EMP array (Figure 15-43), because an ordinary field name (AFLD1, ALFD2, etc.) was assigned to each field of array data. The field was then moved into the array position. This occurred because an array name *with an index* cannot be used in input specifications. Thus, the only time the array fields can be stored directly is when the array data is together on one input record.

Array Data On More Than One Record

In some cases, the data for an array cannot be contained on one input record. First, consider a case in which the array data on all input records is organized consecutively. Data for a 17-field array, named TAX, is contained on four input records (Figure 15-44). The first three records contain five fields each, and the fourth record contains the data for the last two fields of the array. Each numeric field is five characters long. The data is organized on the records in the order in which it is to be stored in the array.

It is important to note that *when data is stored in an array by specifying the array name as the field name the information is placed at the beginning of the array.* Thus, the 25 columns of data from this first input record are stored in fields 1-5 of the array (Figure 15-44).

Although the data on the next three records is also arranged consecutively, each field is defined separately in the input specifications. Each record is not defined as a single array field and stored automatically in the array because every time another record is read, the data would be stored at the beginning of the array. Thus, the data previously stored at the beginning of the array would be destroyed.



IBM International Business Machines Corporation Form X21-9093 Printed in U.S.A.

RPG CALCULATION SPECIFICATIONS

Date ____ Program ____ Programmer ____

Punching Instruction: Graphic / Punch

Page 1 2 — Program Identification 75 76 77 78 79 80

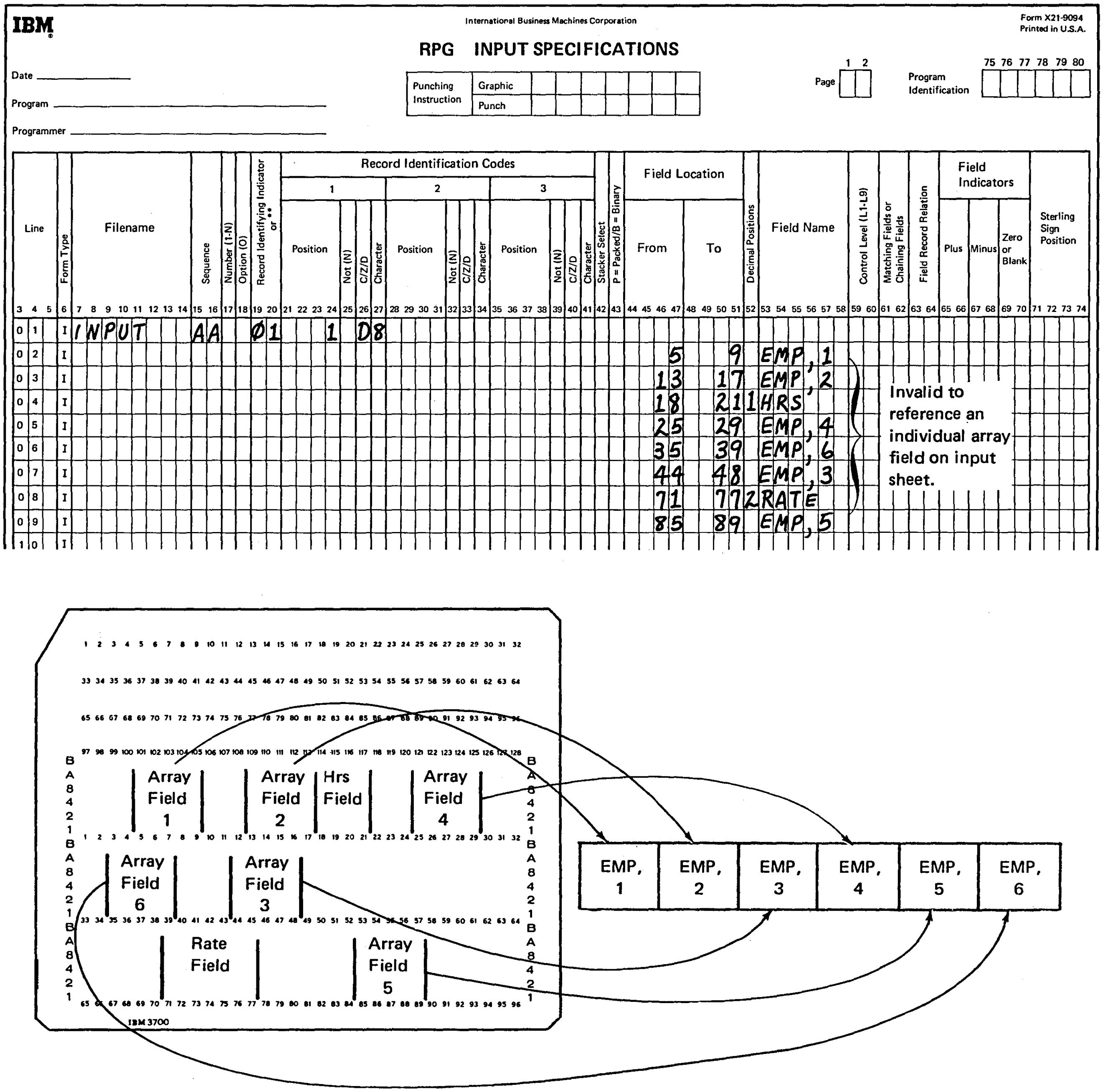
Line	Form Type	Control Level	Indicators	Factor 1	Operation	Factor 2	Result Field	Field Length	Decimal Positions	Half Adjust (H)	Resulting Indicators	Comments
01	C		Ø1		MOVE	AFLD1	EMP,1					
02	C		Ø1		MOVE	AFLD2	EMP,2					
03	C		Ø1		MOVE	AFLD3	EMP,3					
04	C		Ø1		MOVE	AFLD4	EMP,4					
05	C		Ø1		MOVE	AFLD5	EMP,5					
06	C		Ø1		MOVE	AFLD6	EMP,6					
07	C											
08	C											
09	C											
10	C											
11	C											
12	C											
13	C											

Additional calculations using the array data

Figure 15-42. Moving Data Fields into Individual Fields of an Array

Instead, the data from record types 2, 3, and 4 is stored by moving each field directly into the appropriate field of the array. To store the data in this way requires a move operation for each field of data on every record but the first record.

In this example, the method of defining and storing data in the TAX array is relatively simple. However, if there is a large number of data fields contained on records other than the first, storing the data can require a great number of move instructions. As an example, the TAX array consists of 32 fields, requiring seven input records to contain the array data. Records 1-6 each contain five fields of data, while the seventh record contains the remaining two fields. Storing the data in the way described would require 27 separate move instructions for the fields on records two through seven.



IBM

International Business Machines Corporation

Form X21-9094
Printed in U.S.A.

RPG INPUT SPECIFICATIONS

Date
Program
Programmer

Punching Instruction: Graphic / Punch

Page 1 2

Program Identification 75 76 77 78 79 80

Line	Form Type	Filename	Sequence	Number (1-N)	Option (O)	Record Identifying Indicator or **	Record Identification Codes 1: Position	Not (N)	C/Z/D	Character	2: Position	Not (N)	C/Z/D	Character	3: Position	Not (N)	C/Z/D	Character	Stacker Select	P = Packed/B = Binary	Field Location: From	To	Decimal Positions	Field Name	Control Level (L1-L9)	Matching Fields or Chaining Fields	Field Record Relation	Field Indicators: Plus	Minus	Zero or Blank	Sterling Sign Position
0 1	I	INPUT	AA			Ø1	1		D	8																					
0 2	I																				5	9		EMP,1							
0 3	I																				13	17		EMP,2							
0 4	I																				18	21	1	HRS							
0 5	I																				25	29		EMP,4							
0 6	I																				35	39		EMP,6							
0 7	I																				44	48		EMP,3							
0 8	I																				71	77	2	RATE							
0 9	I																				85	89		EMP,5							
1 0	I																														

Invalid to reference an individual array field on input sheet.

Figure 15-43. Incorrect Description of Array Data Fields

When the fields in an array are numerous and many of them are contained on records other than the first, it is more efficient to directly store each record as it is read. This can be done by arranging the array records so the data fields which are to be stored at the end of the array are read first. With this in mind, assume that the data has been organized such that record 1 is stored at the end of the array while the two fields on the last record are stored at the beginning of the array.

The fields from record 1 are initially read and stored at the beginning of the array. The stored data is then moved within the array to the appropriate fields before the next array input record is read and stored at the beginning of the array (Figure 15-45).

Although the array fields are read in reverse order, notice in Figure 15-45 that the fields of data on any one record are still in proper sequence. Thus, the last five fields of the TAX array are in order on record 1 from field 28 through field 32, not field 28. This is because the fields of data are moved in the order in which they are read into the array. Therefore, the last field on record 1, after being moved, becomes the last field of the array.

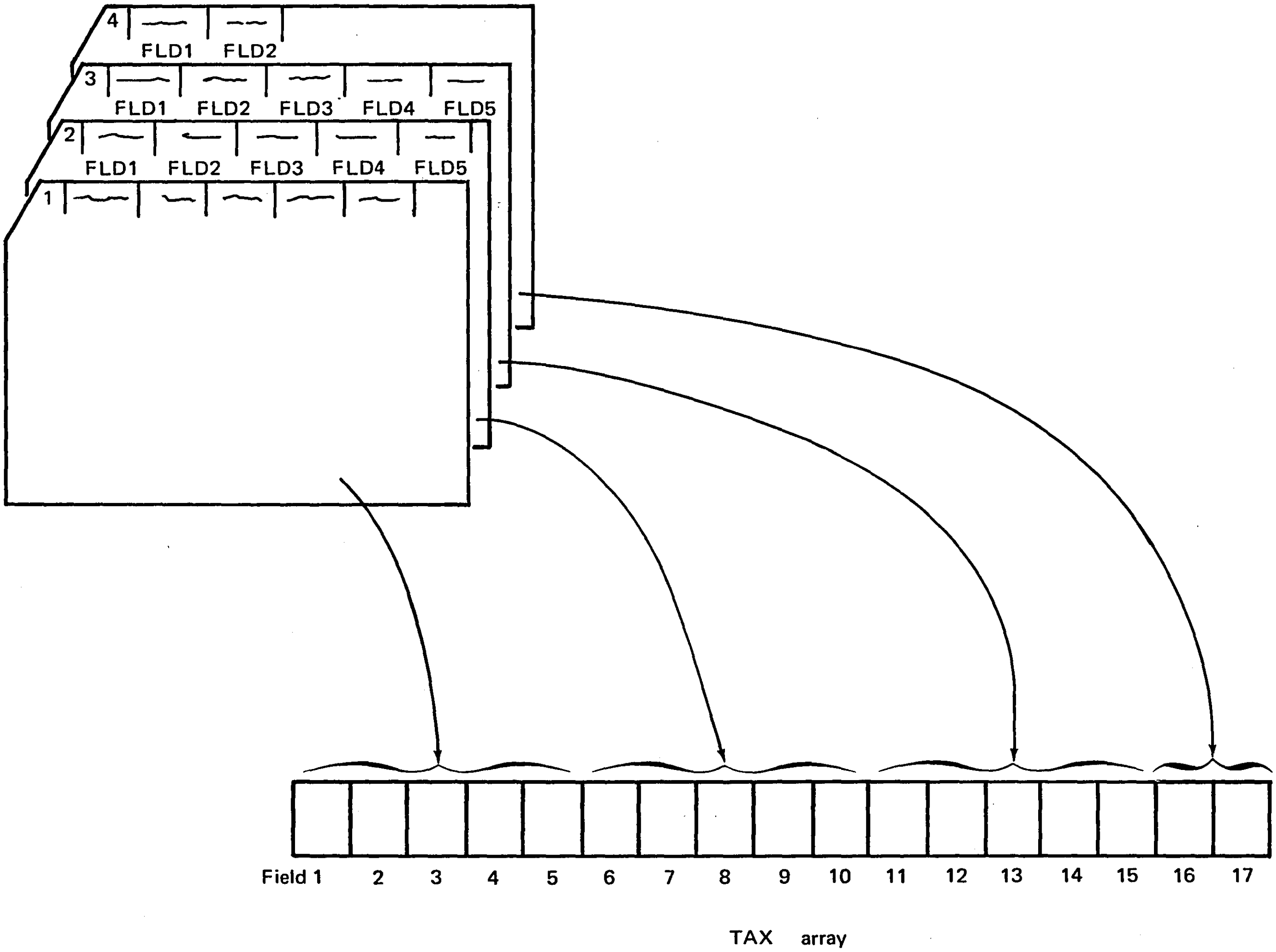


Figure 15-44. Array Data Consecutive on More than One Record

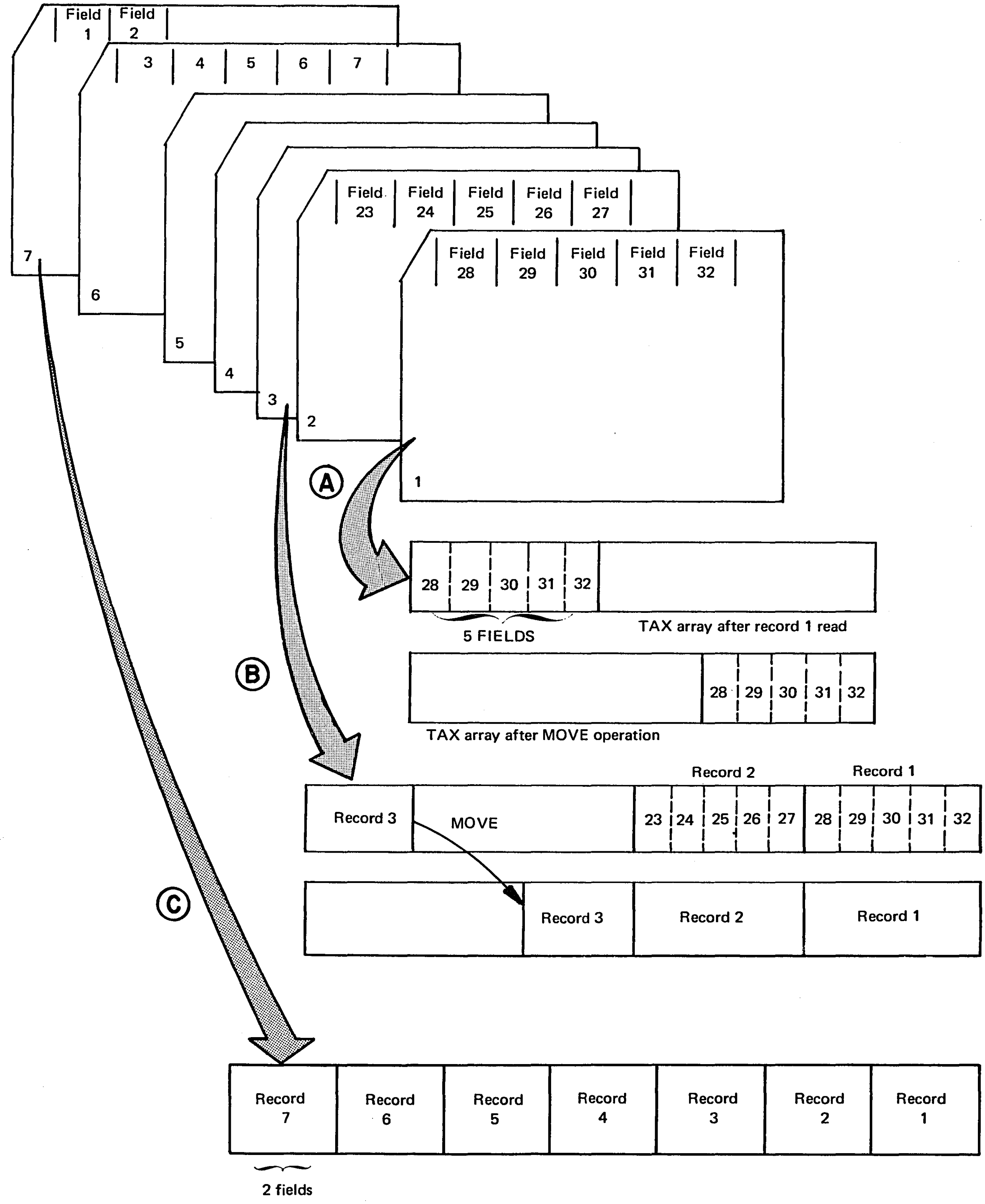


Figure 15-45. Storing Array Data in Reverse Order

Array Data Scattered on More Than One Record

Regardless of how many records are used to contain array data, if the fields are scattered on the records, each field must be individually moved into its appropriate position in the array. However, a separate move specification is not always necessary for each field of data to be moved. In some cases, the same move specification can be used for all the records. This depends on whether all the input records for a single array are organized in the same format and whether the fields from different records can be assigned the same name.

Assume that a 22-field array, named ARA, is defined. The data for the array is scattered on six input records, as shown in Figure 15-46. Although the array data is not consecutive, the four fields on each of the first five records are in the same format on each record. The remaining two fields on the sixth record are in the same format as the first two fields on all other records.

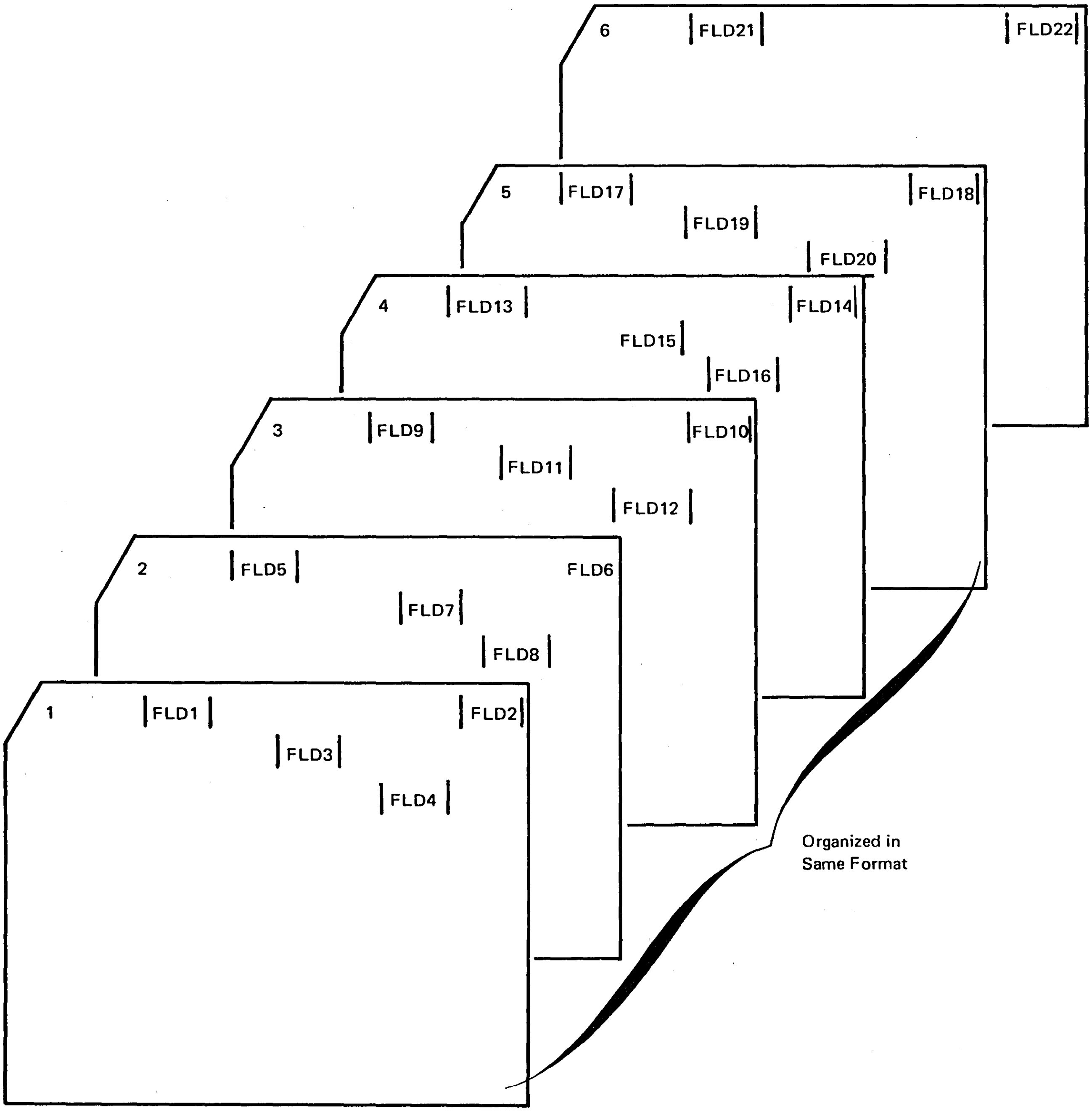
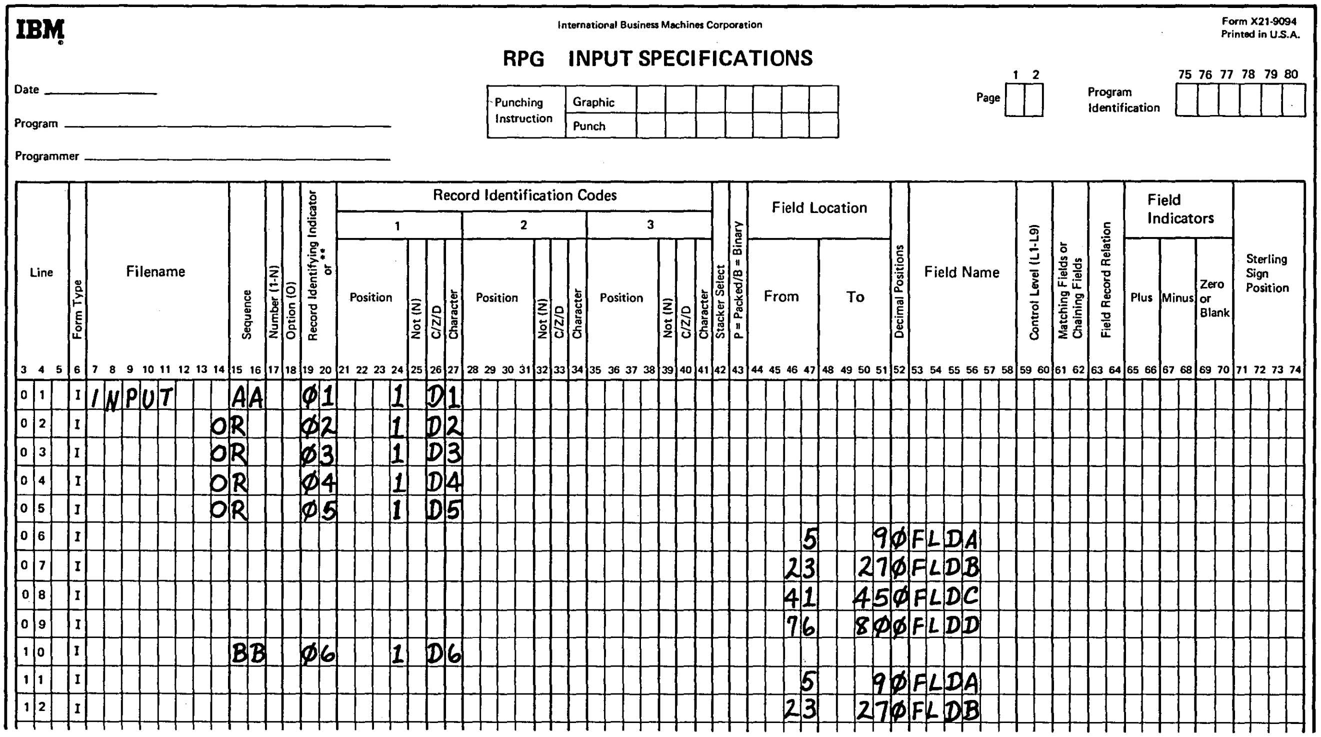


Figure 15-46. Array Data Scattered on More than One Record

Since the array data follows the same organization on all records, describing one set of fields (Figure 15-47) actually describes the fields on all records, except the last. A separate input specification must be coded to indicate that record 6 only contains two of the fields. (Note on the Input Sheet that records 1-5 are described in an OR relationship. Therefore, a specific card sequence cannot be specified in columns 15 through 16. You can assume that the array input records are in sequence. Record type 1 is the first record read and record type 6 is the last.)

Because the fields on the different records have the same field names, only one MOVE specification is necessary for each unique field name. The specification on line 07 of Figure 15-48, when repeated for each record, moves FLDA of that record to the appropriate field of the ARA array. Lines 09 and 10 are performed for every record except the last, which does not have fields FLDC and FLDD.

Since the fields on the input records are in the same order as they are to be stored in ARA, a definite pattern is established as to where the data is to be moved. Fields from record 1 are stored in array fields one through four, fields from record 2 in array fields 5 through 8, fields from record 3 in array fields 9 through 12, and so on.



IBM

International Business Machines Corporation

Form X21-9094
Printed in U.S.A.

RPG INPUT SPECIFICATIONS

Date

Program

Programmer

Punching Instruction | Graphic | Punch

Page 1 2

Program Identification 75 76 77 78 79 80

Line (3-5)	Form Type (6)	Filename (7-14)	Sequence (15-16)	Number (1-N) (17)	Option (O) (18)	Record Identifying Indicator or ** (19-20)	Record Identification Codes 1: Position (21-24)	Not (N) (25)	C/Z/D (26)	Character (27)	2: Position (28-31)	Not (N) (32)	C/Z/D (33)	Character (34)	3: Position (35-38)	Not (N) (39)	C/Z/D (40)	Character (41)	Stacker Select (42)	P = Packed/B = Binary (43)	Field Location From (44-47)	To (48-51)	Decimal Positions (52)	Field Name (53-58)	Control Level (L1-L9) (59-60)	Matching Fields or Chaining Fields (61-62)	Field Record Relation (63-64)	Field Indicators Plus (65-66)	Minus (67-68)	Zero or Blank (69-70)	Sterling Sign Position (71-74)
01	I	INPUT	AA			ø1	1		D	1																					
02	I		OR			ø2	1		D	2																					
03	I		OR			ø3	1		D	3																					
04	I		OR			ø4	1		D	4																					
05	I		OR			ø5	1		D	5																					
06	I																				5	9	ø	FLDA							
07	I																				23	27	ø	FLDB							
08	I																				41	45	ø	FLDC							
09	I																				76	8ø	ø	FLDD							
10	I		BB			ø6	1		D	6																					
11	I																				5	9	ø	FLDA							
12	I																				23	27	ø	FLDB							

Figure 15-47. Describing One Set of Array Fields for Several Records

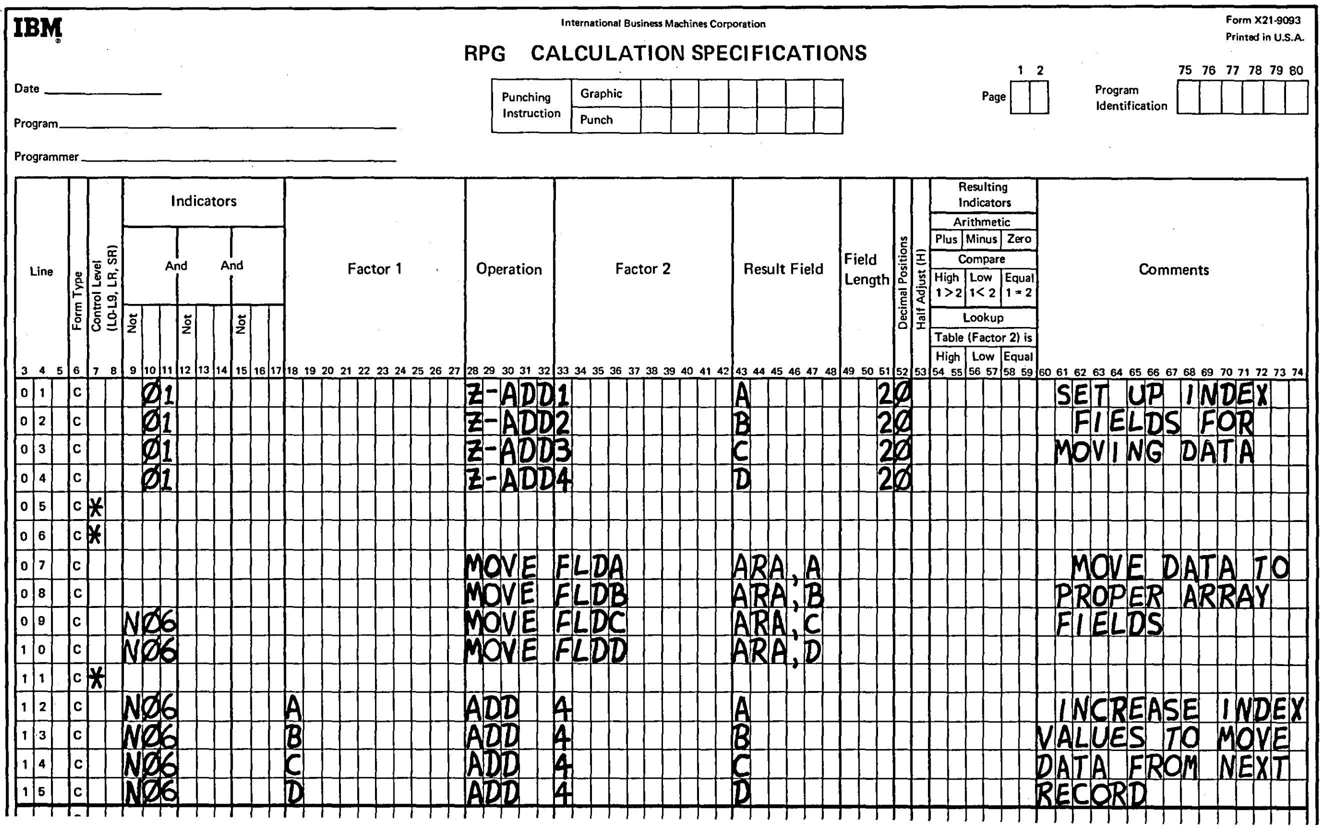
Array index fields can be used to indicate to which array fields the data is to be moved. For each unique field name, an individual index field should be set up. In this way, the values in the index fields only have to be changed every time another array input record is processed. When the first record is read, the index fields *A, B, C,* and *D* are initialized to *1, 2, 3,* and *4,* respectively, to prepare for moving the fields from record type 1 (Figure 15-48, lines 01-04). After the four fields are moved, the value *4* is added to each of the index fields so they point to where the four fields on the next record should be stored (Figure 15-48, lines 12-15). The same calculation specifications are repeated until fields FLDA and FLDB from the sixth record have been moved to the last two array fields.

Conditioning Operations Until All Array Data is Stored

All information must be stored in an array before you can reference the data by specifying the array name or array name with an index. If array data is contained on only one record, this means that any calculations to move the data into the array must be specified before any calculations which use the array information. On the other hand, when array data is contained on more than one record, some operations cannot be performed until all the array data is stored.

For every record of array data, RPG II goes through a complete program cycle, just as it does to process any other data card. This means that input, calculation, and output specifications can be performed every time an array input record is processed. You want input specifications to be performed to describe the array record to the system. Likewise, calculation specifications which move the data from the record to the array should also be performed. However, if there are still some array records which have not been processed (thus, not stored in the array), calculations and output which reference the array must not be performed. For example, if only five fields of data have been moved into a 10-field array, adding all fields of the array or printing all fields will certainly not provide the results you want.

Once the last array input record has been stored, any specifications referencing the array fields can be performed. Thus, you must specify a conditioning indicator (columns 9 through 17 on Calculation Sheet and columns 23 through 31 on Output Sheet) which indicates when the last array record has been processed.



IBM

International Business Machines Corporation

Form X21-9093
Printed in U.S.A.

RPG CALCULATION SPECIFICATIONS

Date ____

Program ____

Programmer ____

Punching Instruction: Graphic / Punch

Page 1 2

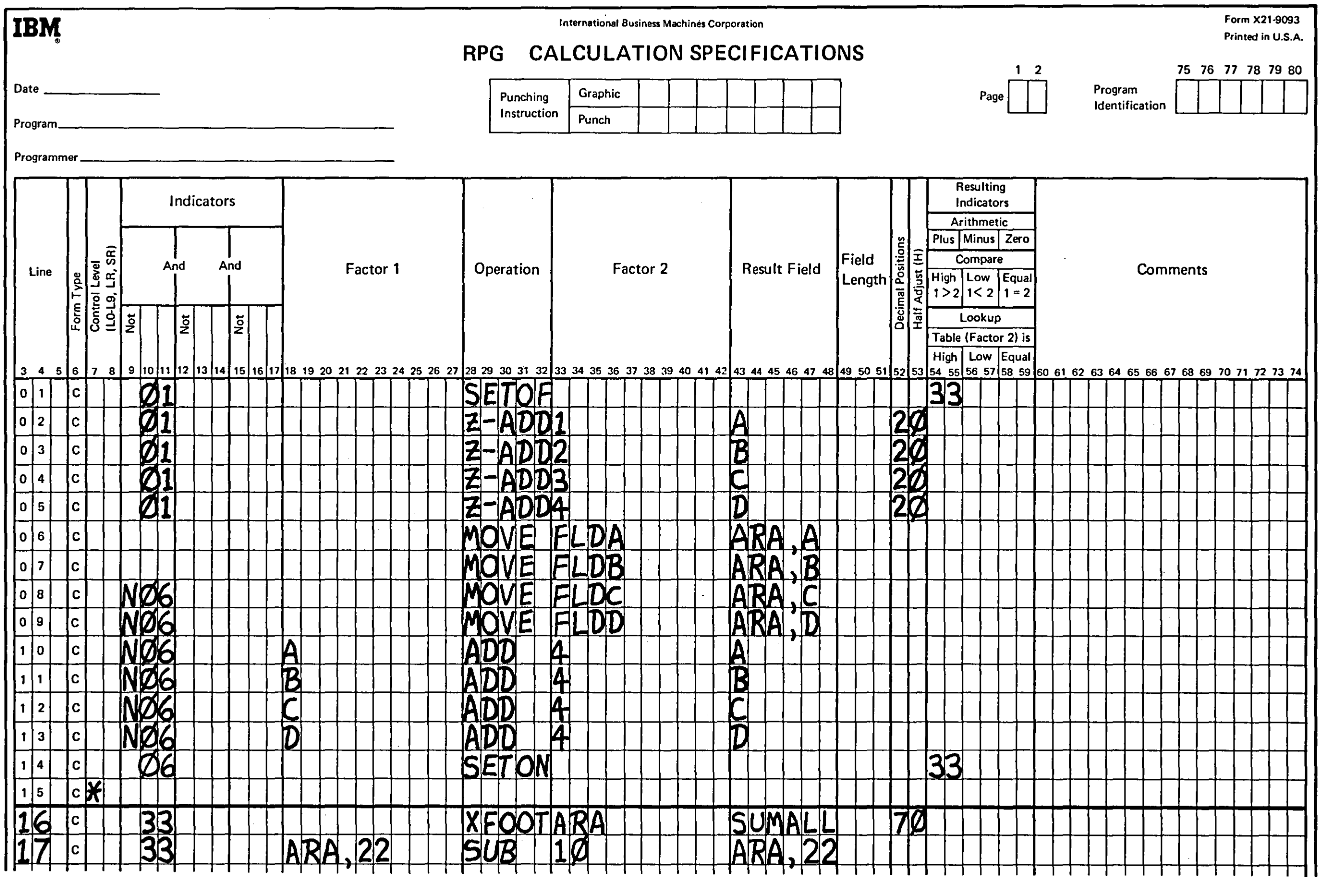
Program Identification 75 76 77 78 79 80

Line (3-5)	Form Type (6)	Control Level (LO-L9, LR, SR) (7-8)	Indicators (9-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field (43-48)	Field Length (49-51)	Decimal Positions (52)	Half Adjust (H) (53)	Resulting Indicators (54-59)	Comments (60-74)
01	C		Ø1		Z-ADD	1	A	2	Ø			SET UP INDEX
02	C		Ø1		Z-ADD	2	B	2	Ø			FIELDS FOR
03	C		Ø1		Z-ADD	3	C	2	Ø			MOVING DATA
04	C		Ø1		Z-ADD	4	D	2	Ø			
05	C	*										
06	C	*										
07	C				MOVE	FLDA	ARA,A					MOVE DATA TO
08	C				MOVE	FLDB	ARA,B					PROPER ARRAY
09	C		NØ6		MOVE	FLDC	ARA,C					FIELDS
10	C		NØ6		MOVE	FLDD	ARA,D					
11	C	*										
12	C		NØ6	A	ADD	4	A					INCREASE INDEX
13	C		NØ6	B	ADD	4	B					VALUES TO MOVE
14	C		NØ6	C	ADD	4	C					DATA FROM NEXT
15	C		NØ6	D	ADD	4	D					RECORD

Figure 15-48. Using the Same MOVE for Fields from Several Records

Lines 02 through 31 of the Calculation Sheet in Figure 15-49 are performed to move data from the six array input records into the array ARA. When the last record is processed (record identifying indicator 06 on), the two array operations on lines 16 and 17 can be performed during that program cycle. Therefore, when record type 6 has been stored, indicator 33 is set on (Figure 15-49, line 14). Indicator 33 (or any other indicator which is set on) can then condition the XFOOT and SUB operations to be performed in a program cycle.

The record identifying indicator 06 was not specified to condition the array operations, because 06 is on only for the cycle in which the sixth array record is processed. Since the array operations on lines 16 through 17 must be performed in the following program cycles also (for example, if normal data records follow the array records), they must be conditioned by an indicator which is on during the following cycles. Once indicator 33 has been set on, it remains on through following program cycles, until set off (line 01) when another group of array records are processed.



IBM

International Business Machines Corporation

RPG CALCULATION SPECIFICATIONS

Form X21-9093
Printed in U.S.A.

Date

Program

Programmer

Punching Instruction: Graphic / Punch

Page 1 2

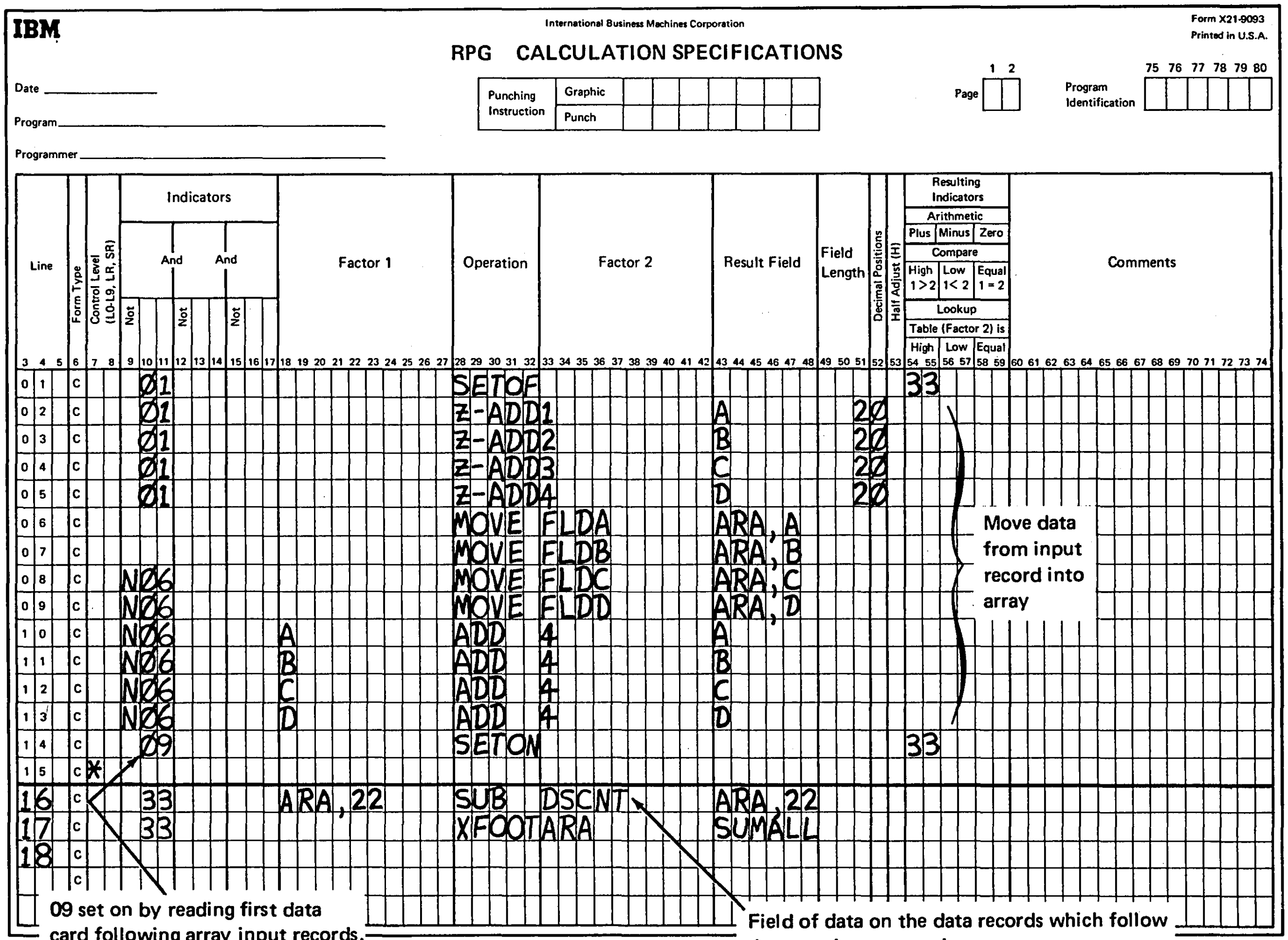
Program Identification 75 76 77 78 79 80

Line (3-5)	Form Type (6)	Control Level (7-8)	Indicators (9-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field (43-48)	Field Length (49-51)	Decimal Positions (52)	Half Adjust (53)	Resulting Indicators (54-59)	Comments (60-74)
01	C		Ø1		SETOF						33	
02	C		Ø1		Z-ADD	1	A	2	Ø			
03	C		Ø1		Z-ADD	2	B	2	Ø			
04	C		Ø1		Z-ADD	3	C	2	Ø			
05	C		Ø1		Z-ADD	4	D	2	Ø			
06	C				MOVE	FLDA	ARA,A					
07	C				MOVE	FLDB	ARA,B					
08	C		NØ6		MOVE	FLDC	ARA,C					
09	C		NØ6		MOVE	FLDD	ARA,D					
10	C		NØ6	A	ADD	4	A					
11	C		NØ6	B	ADD	4	B					
12	C		NØ6	C	ADD	4	C					
13	C		NØ6	D	ADD	4	D					
14	C		Ø6		SETON						33	
15	C	*										
16	C		33		XFOOT	ARA	SUMALL	7	Ø			
17	C		33	ARA,22	SUB	1Ø	ARA,22					

Figure 15-49. Conditioning Operations Until All Array Data is Stored

Figure 15-50 shows how the array operations must be conditioned for another situation. In this case, record identifying indicator 06 does not set on indicator 33 because information (DSCNT) from data records following the array records must be available before the array operations can be performed (line 16). If indicator 06 caused indicator 33 to be set on, the array operations would be performed during the program cycle in which the sixth array record is stored. At that point, the DSCNT data is not available. Therefore, record identifying indicator 09 (the first type of data card following the array records) sets on the conditioning indicator 33 instead.

At this point, we must mention a problem which can come up if array fields are contained on more than one record (or the same record type), and the records contain normal input data as well as array data. Assume three cards contain the array data and all the data must be stored in the array prior to performing any calculation or output operations. This means the three records must be read before processing. As each new record (of the same record type) is read, the data from the previous record is destroyed, unless it has been moved or stored in a special place, such as an array. Since normal input data (non-array fields) from the first two records is no longer available once the third record has been read, any calculation or output specifications which reference this input data might give incorrect results.



IBM

International Business Machines Corporation

RPG CALCULATION SPECIFICATIONS

Form X21-9093
Printed in U.S.A.

Date ______

Program ______

Programmer ______

Punching Instruction: Graphic / Punch

Page 1 2

Program Identification 75 76 77 78 79 80

Line	Form Type	Control Level (L0-L9, LR, SR)	Indicators (Not, And, Not, And, Not)	Factor 1	Operation	Factor 2	Result Field	Field Length	Decimal Positions	Half Adjust (H)	Resulting Indicators: Plus/High (54-55)	Minus/Low (56-57)	Zero/Equal (58-59)	Comments
01	C		Ø1		SETOF						33			
02	C		Ø1		Z-ADD	1	A	2	Ø					
03	C		Ø1		Z-ADD	2	B	2	Ø					
04	C		Ø1		Z-ADD	3	C	2	Ø					
05	C		Ø1		Z-ADD	4	D	2	Ø					
06	C				MOVE	FLDA	ARA,A							
07	C				MOVE	FLDB	ARA,B							
08	C		NØ6		MOVE	FLDC	ARA,C							
09	C		NØ6		MOVE	FLDD	ARA,D							
10	C		NØ6	A	ADD	4	A							
11	C		NØ6	B	ADD	4	B							
12	C		NØ6	C	ADD	4	C							
13	C		NØ6	D	ADD	4	D							
14	C		Ø9		SETON						33			
15	C	*												
16	C		33	ARA,22	SUB	DSCNT	ARA,22							
17	C		33		XFOOT	ARA	SUMALL							
18	C													
	C													

Move data from input record into array

09 set on by reading first data card following array input records.

Field of data on the data records which follow the array input records.

Figure 15-50. Conditioning Operations Until All Array Data Is Stored and Input Data Is Available

Review 15

1. An array is like a table in which of the following ways (state true or false and the reasons for your answer):

   a. Each can be referenced as one group of information.

   b. Each is a continuous series of data fields (elements) stored side by side.

   c. A particular item of data can be individually referenced in either a table or an array.

   d. Each is defined by coding extension specifications.

2. Can one array be compared to another array to determine which is greater or less? State the reason for your answer.

3. Explain what happens if an array:

   a. of 18 fields is added to an array.

   b. of three fields, with the result placed in an array.

   c. of 18 fields.

4. The following array (ARASIX) is to be set up at the beginning of a program run:

1	2	72	5	20	15

   a. Define the array on an Extension Sheet.

   b. If ARASIX is multiplied by *3,* what data will be placed in the result array RESARA?

   c. Should the result array RESARA be defined on the Extension Sheet also?

   d. If so, code the necessary extension specifications to define RESARA.

   e. What is accomplished by defining an array on the Extension Sheet?

5. a. Explain what happens when an XFOOT operation is performed.

   b. If ARASIX (refer to question 3) is specified on the Calculation Sheet as Factor 2 of an XFOOT operation, what data would be placed in the result field?

6. How does a programmer specify that an entire array is to be printed or punched?

7. How does a programmer specify whether an entire array or only a particular array field is to be operated upon or used for output?

8. Data for a SALES array is recorded on one card (record type 01 of INFILE) in the following format:

Field	*Columns*	*Field*	*Columns*
Clerk1	1-10	Clerk6	51-60
Clerk2	11-20	Clerk7	61-70
Clerk3	21-30	Clerk8	71-80
Clerk4	31-40	Clerk9	81-90
Clerk5	41-50	card code	91 (not array data)

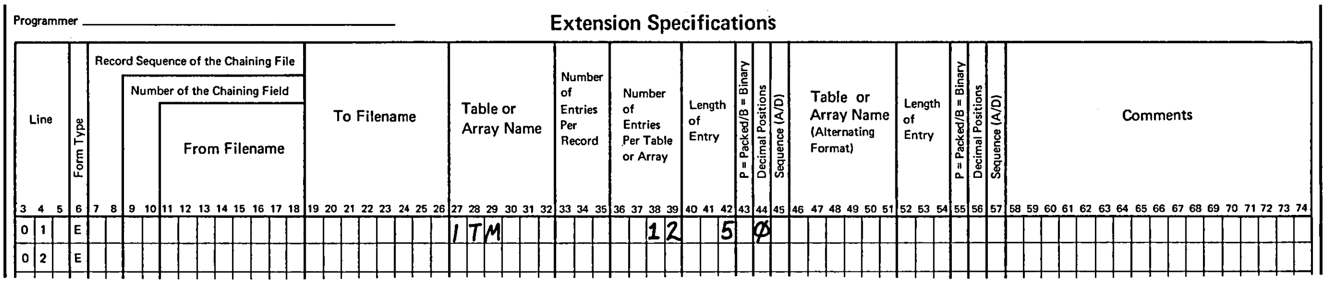
   Each of the clerk fields contains data with two decimal positions. Code the specifications necessary to:

   a. Define the array.

   b. Describe the input record.

   c. Store the data in the SALES array.

9. A 12-field array ITM is defined as follows:

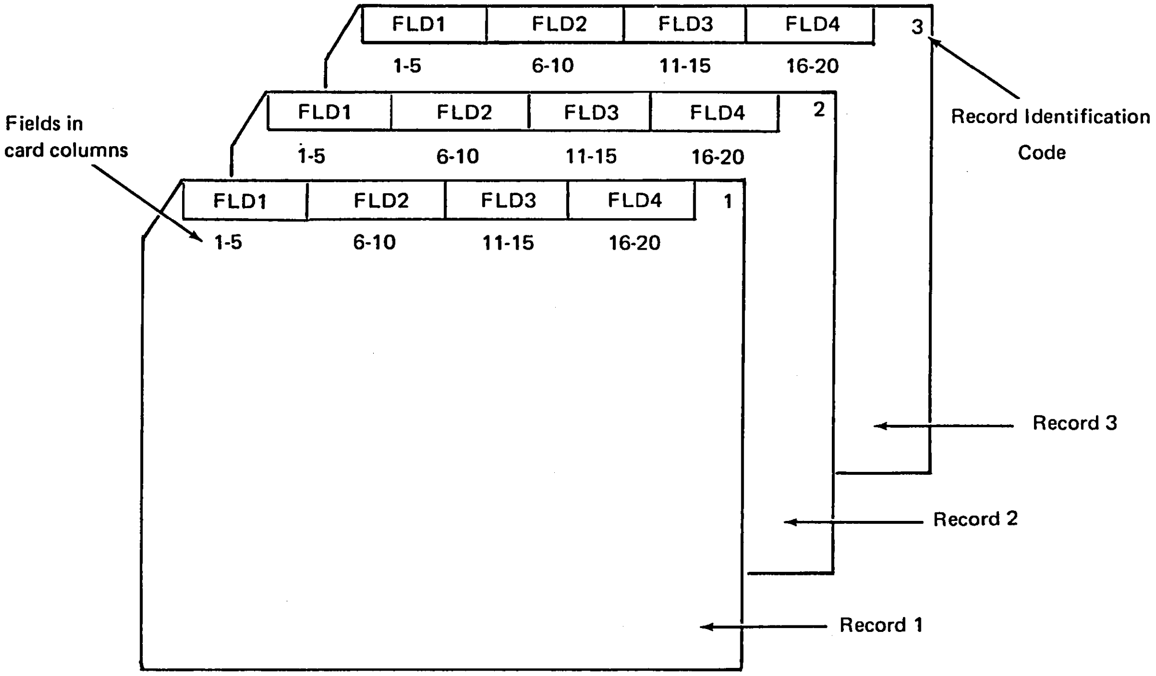


Programmer ______________________

Extension Specifications

Line	Form Type	Record Sequence of the Chaining File	Number of the Chaining Field	From Filename	To Filename	Table or Array Name	Number of Entries Per Record	Number of Entries Per Table or Array	Length of Entry	P = Packed/B = Binary	Decimal Positions	Sequence (A/D)	Table or Array Name (Alternating Format)	Length of Entry	P = Packed/B = Binary	Decimal Positions	Sequence (A/D)	Comments
3 4 5	6	7 8	9 10	11 12 13 14 15 16 17 18	19 20 21 22 23 24 25 26	27 28 29 30 31 32	33 34 35	36 37 38 39	40 41 42	43	44	45	46 47 48 49 50 51	52 53 54	55	56	57	58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74
0 1	E					ITM		12	5		Ø							
0 2	E																	

Data for the array is contained on three records from the input file INPT:



a. Code the specifications to describe the records and store the data in ITM.

b. How could you simplify the method of storing the array data?

10. Explain how data from the following record would be stored into the SET array (15 fields, three numeric characters each, no decimal positions), and code the specifications to describe and store the first five fields:

Field	Columns	Field	Columns	Field	Columns
Fld1	2-4	Fld6	22-24	Fld11	42-44
Fld2	6-8	Fld7	26-28	Fld12	46-48
Fld3	10-12	Fld8	30-32	Fld13	50-52
Fld4	14-16	Fld9	34-36	Fld14	54-56
Fld5	18-20	Fld10	38-40	Fld15	58-60

Column *1* contains a *P* as the record identifying code.

11. a. Code the output specifications to print the 13th field of the array SET (output filename PRINT).

    b. Code the specifications to print the entire array SET at end of job (output filename PRINT).

12. SEARCH is the name of a field containing data you wish to locate in the 6-field PAY array. Code the specifications to search the array to determine if the data is present. If present, print the number and contents of the array field.

13. Code specifications to:

    a. Add ARA1 to ARA2 and place the result in ARA2.

    b. Sum all fields of ARA2 and place the result in TOTAL.

    c. Print both results.

1. a. False. Only one table element can be referenced at one time.

   b. True.

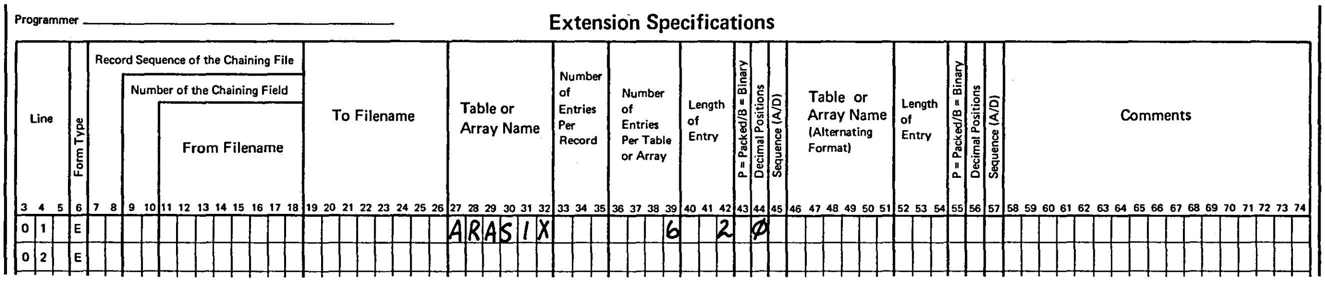
   c. True.

   d. True.

2. No. An operation to be performed on an array is repeated for each field in the array. Therefore, a compare (COMP) operation cannot give a meaningful result for the entire array.

3. The first three fields of array *a* would be added to the three fields of array *b,* with the three results placed in the first three fields of array *c.* The remaining 15 fields of array *c* (result array) remain unchanged.

4. a. Entries shown are required; no other entries should be made. The entry in column 44 is necessary to indicate the fields are numeric for arithmetic operations.



Programmer

Extension Specifications

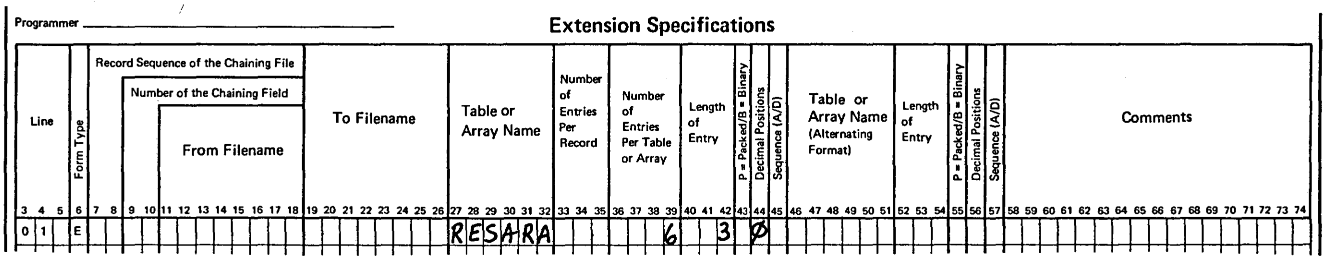
Line	Form Type	Record Sequence of the Chaining File	Number of the Chaining Field	From Filename	To Filename	Table or Array Name	Number of Entries Per Record	Number of Entries Per Table or Array	Length of Entry	P = Packed/B = Binary	Decimal Positions	Sequence (A/D)	Table or Array Name (Alternating Format)	Length of Entry	P = Packed/B = Binary	Decimal Positions	Sequence (A/D)	Comments
3 4 5	6	7 8	9 10	11 12 13 14 15 16 17 18	19 20 21 22 23 24 25 26	27 28 29 30 31 32	33 34 35	36 37 38 39	40 41 42	43	44	45	46 47 48 49 50 51	52 53 54	55	56	57	58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74
0 1	E					ARASIX		6	2		Ø							
0 2	E																	

b.

3	6	216	15	60	45

c. Yes, all arrays to be used in a program must be defined on the Extension Sheet.

d. Entries shown are required; no other entries should be made. Length of array field (columns 40-42) must be 3 to contain the largest addition result.



Programmer

Extension Specifications

Line	Form Type	Record Sequence of the Chaining File	Number of the Chaining Field	From Filename	To Filename	Table or Array Name	Number of Entries Per Record	Number of Entries Per Table or Array	Length of Entry	P = Packed/B = Binary	Decimal Positions	Sequence (A/D)	Table or Array Name (Alternating Format)	Length of Entry	P = Packed/B = Binary	Decimal Positions	Sequence (A/D)	Comments
3 4 5	6	7 8	9 10	11 12 13 14 15 16 17 18	19 20 21 22 23 24 25 26	27 28 29 30 31 32	33 34 35	36 37 38 39	40 41 42	43	44	45	46 47 48 49 50 51	52 53 54	55	56	57	58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74
0 1	E					RESARA		6	3		Ø							

e. An area in storage sufficient to contain the array data is reserved. The actual array data may be stored in the array later, when input records are read or during calculations.

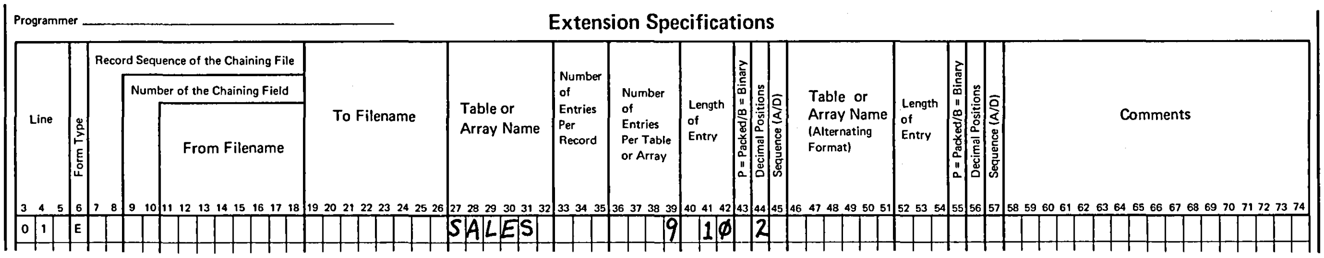
5. a. The XFOOT operation causes all data fields in the array specified as Factor 2 to be added together. The single result of the additions is placed in the result field specified with the XFOOT operation.

   b. The total of all fields in the ARASIX array (115) would be placed in the result field.

6. The name of the array is specified uner Field Name (columns 32 through 37) on the Output Sheet. The filename (columns 7 through 14) must also be specified, as for output of any field.

7. The array name is specified alone (on the Calculation or Output Sheet) to reference the entire array. The array name must be followed by a comma and an index number or index field to reference only a particular array field.

8. a. Extension specifications to define the array:

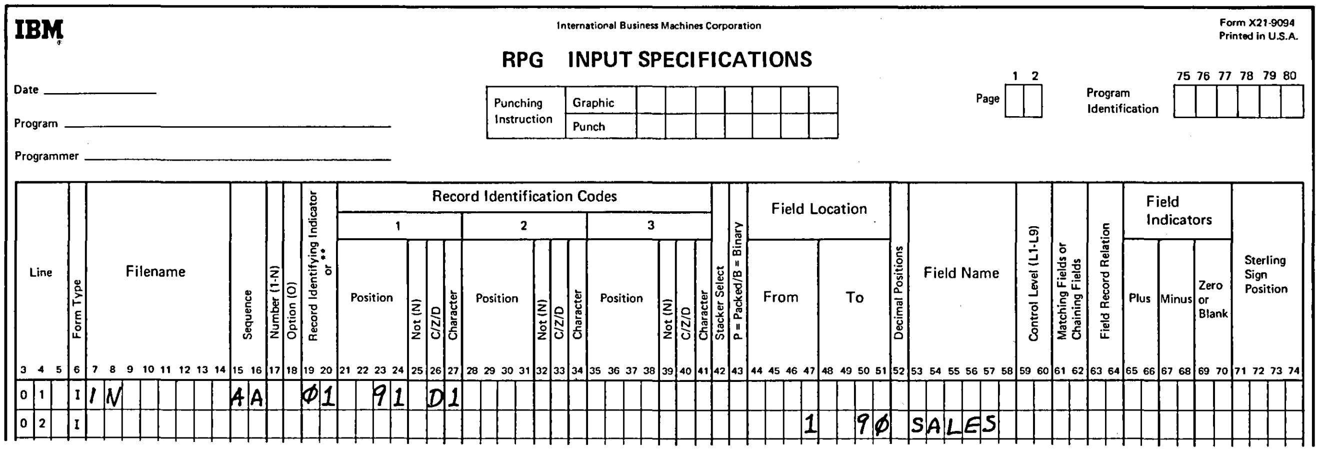


Programmer

Extension Specifications

Line (3-5)	Form Type (6)	Record Sequence of the Chaining File (7-8)	Number of the Chaining Field (9-10)	From Filename (11-18)	To Filename (19-26)	Table or Array Name (27-32)	Number of Entries Per Record (33-35)	Number of Entries Per Table or Array (36-39)	Length of Entry (40-42)	P = Packed/B = Binary (43)	Decimal Positions (44)	Sequence (A/D) (45)	Table or Array Name (Alternating Format) (46-51)	Length of Entry (52-54)	P = Packed/B = Binary (55)	Decimal Positions (56)	Sequence (A/D) (57)	Comments (58-74)
01	E					SALES		9	1Ø		2							

b. Input specifications to describe the input record and store the data in the SALES array:



IBM

International Business Machines Corporation

Form X21-9094
Printed in U.S.A.

RPG INPUT SPECIFICATIONS

Date

Program

Programmer

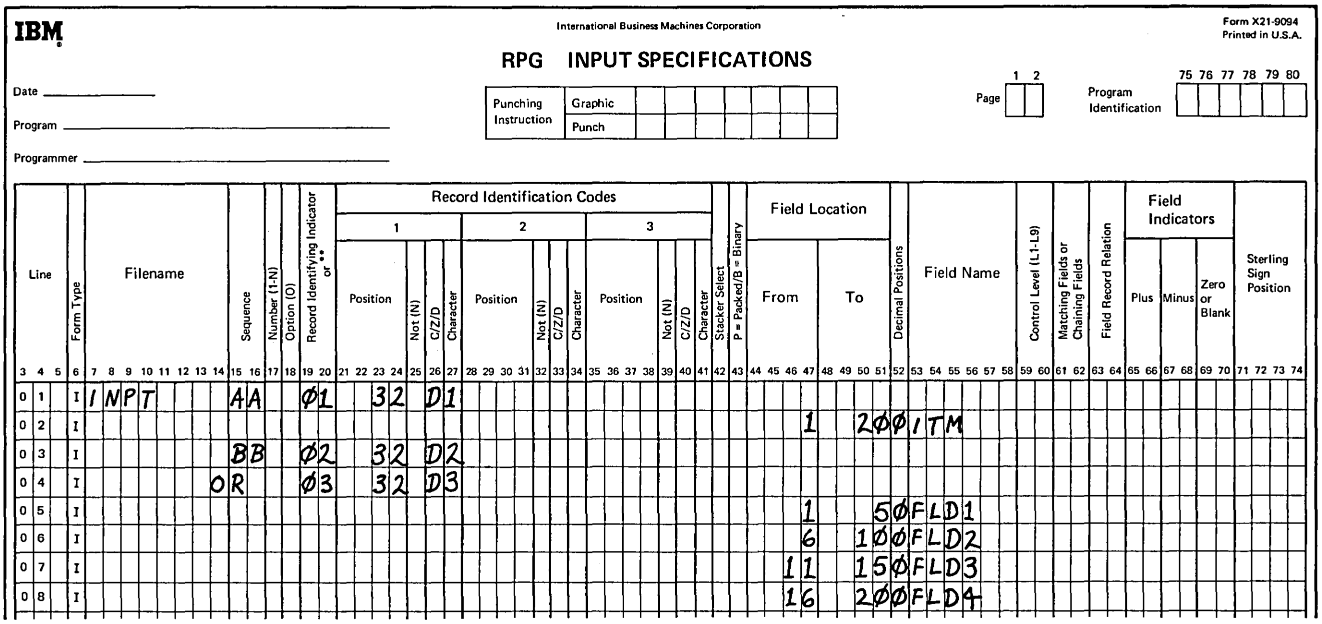
Punching Instruction	Graphic							
	Punch							

Page (1 2) □□

Program Identification (75 76 77 78 79 80) □□□□□□

Line (3-5)	Form Type (6)	Filename (7-14)	Sequence (15-16)	Number (1-N) (17)	Option (O) (18)	Record Identifying Indicator or ** (19-20)	Record Identification Codes 1: Position (21-24)	Not (N) (25)	C/Z/D (26)	Character (27)	2: Position (28-31)	Not (N) (32)	C/Z/D (33)	Character (34)	3: Position (35-38)	Not (N) (39)	C/Z/D (40)	Character (41)	Stacker Select (42)	P = Packed/B = Binary (43)	Field Location From (44-47)	Field Location To (48-51)	Decimal Positions (52)	Field Name (53-58)	Control Level (L1-L9) (59-60)	Matching Fields or Chaining Fields (61-62)	Field Record Relation (63-64)	Field Indicators Plus (65-66)	Minus (67-68)	Zero or Blank (69-70)	Sterling Sign Position (71-74)
01	I	IN	AA			Ø1	91		D	1																					
02	I																				1	9Ø		SALES							

9. a.



IBM International Business Machines Corporation Form X21-9094 Printed in U.S.A.

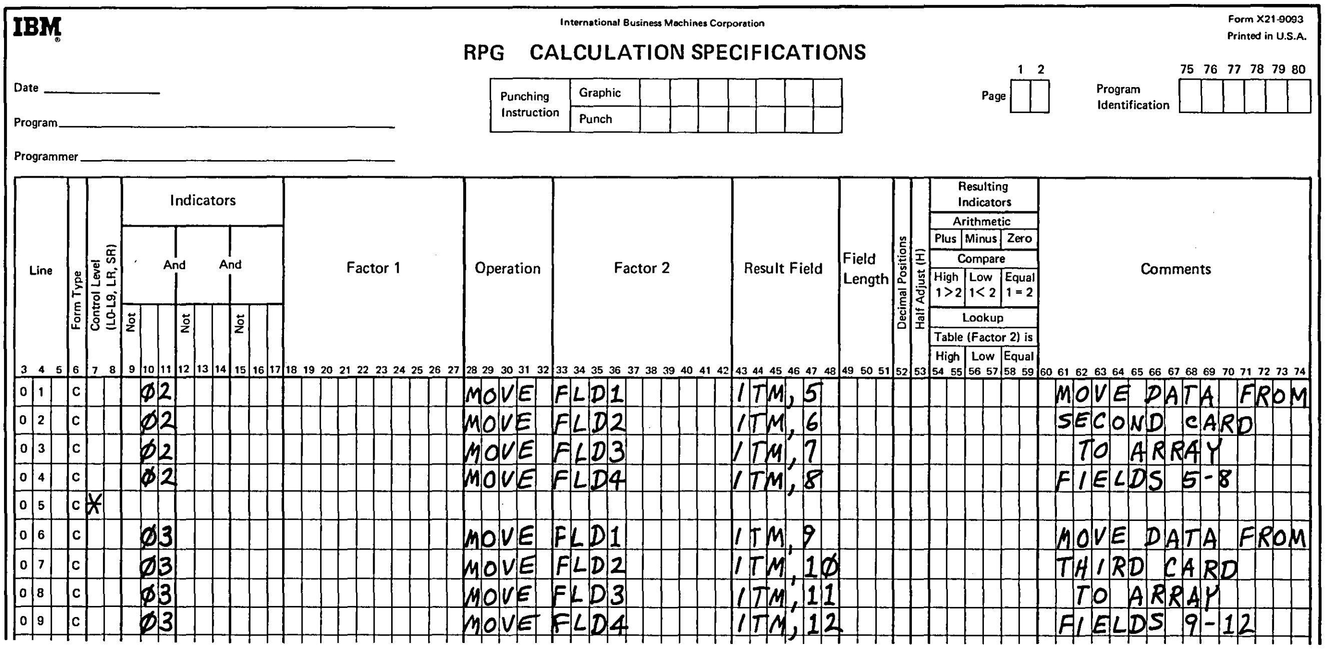
RPG INPUT SPECIFICATIONS

Date ______ Program ______ Programmer ______

Punching Instruction: Graphic / Punch — Page 1 2 — Program Identification 75 76 77 78 79 80

Line | Form Type | Filename | Sequence | Number (1-N) | Option (O) | Record Identifying Indicator or ** | Record Identification Codes 1, 2, 3: Position, Not (N), C/Z/D, Character | Stacker Select | P = Packed/B = Binary | Field Location: From, To | Decimal Positions | Field Name | Control Level (L1-L9) | Matching Fields or Chaining Fields | Field Record Relation | Field Indicators: Plus, Minus, Zero or Blank | Sterling Sign Position

```
  01 IINPT    AA  Ø1  32 D1
  02 I                                        1  20ØITM
  03 I        BB  Ø2  32 D2
  04 I       OR   Ø3  32 D3
  05 I                                        1   5ØFLD1
  06 I                                        6  10ØFLD2
  07 I                                       11  15ØFLD3
  08 I                                       16  20ØFLD4
```



IBM International Business Machines Corporation Form X21-9093 Printed in U.S.A.

RPG CALCULATION SPECIFICATIONS

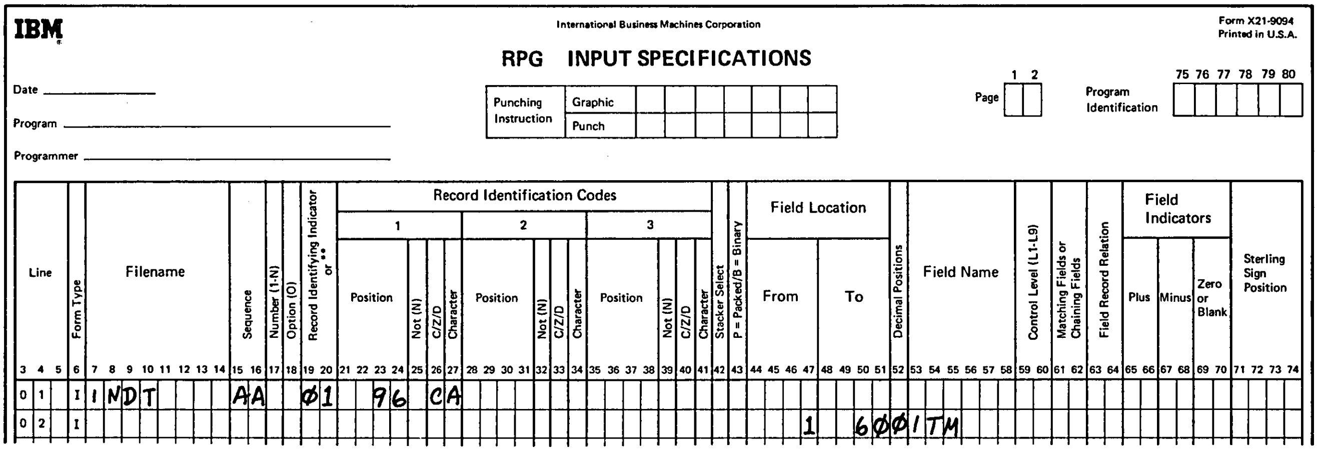
Date ______ Program ______ Programmer ______

Punching Instruction: Graphic / Punch — Page 1 2 — Program Identification 75 76 77 78 79 80

Line | Form Type | Control Level (L0-L9, LR, SR) | Indicators: Not, And Not, And Not | Factor 1 | Operation | Factor 2 | Result Field | Field Length | Decimal Positions | Half Adjust (H) | Resulting Indicators: Arithmetic Plus Minus Zero; Compare High 1>2 Low 1<2 Equal 1=2; Lookup Table (Factor 2) is High Low Equal | Comments

```
  01 C   Ø2                MOVE FLD1      ITM,5            MOVE DATA FROM
  02 C   Ø2                MOVE FLD2      ITM,6            SECOND CARD
  03 C   Ø2                MOVE FLD3      ITM,7             TO ARRAY
  04 C   Ø2                MOVE FLD4      ITM,8            FIELDS 5-8
  05 C*
  06 C   Ø3                MOVE FLD1      ITM,9            MOVE DATA FROM
  07 C   Ø3                MOVE FLD2      ITM,1Ø           THIRD CARD
  08 C   Ø3                MOVE FLD3      ITM,11            TO ARRAY
  09 C   Ø3                MOVE FLD4      ITM,12           FIELDS 9-12
```

b. All 12 fields could be recorded consecutively on one record (columns 1-60) and then stored directly into the array by specifying the array name (ITM) as the Field Name on the Input Sheet. Calculations would not be necessary to move data into the array.



IBM

International Business Machines Corporation

Form X21-9094
Printed in U.S.A.

RPG INPUT SPECIFICATIONS

Date ______

Program ______

Programmer ______

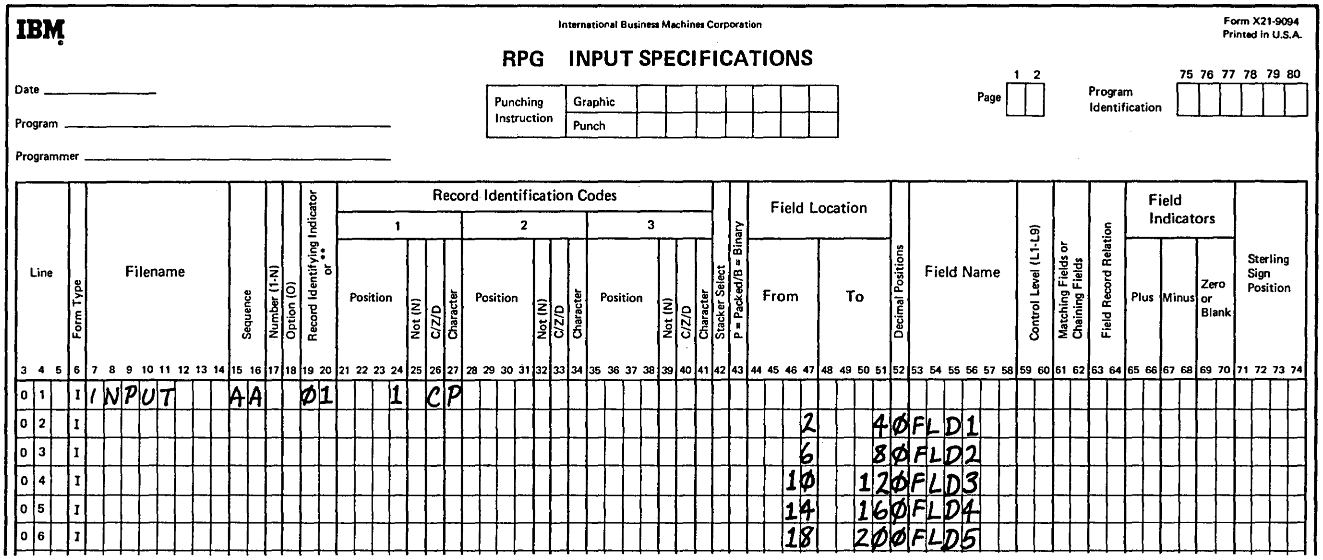
Punching Instruction: Graphic ______ Punch ______

Page (1 2) □□

Program Identification (75 76 77 78 79 80) □□□□□□

Line (3-5)	Form Type (6)	Filename (7-14)	Sequence (15-16)	Number (1-N) (17)	Option (O) (18)	Record Identifying Indicator or ** (19-20)	Record Identification Codes 1: Position (21-24)	Not (N) (25)	C/Z/D (26)	Character (27)	2: Position (28-31)	Not (N) (32)	C/Z/D (33)	Character (34)	3: Position (35-38)	Not (N) (39)	C/Z/D (40)	Character (41)	Stacker Select (42)	P = Packed/B = Binary (43)	Field Location From (44-47)	To (48-51)	Decimal Positions (52)	Field Name (53-58)	Control Level (L1-L9) (59-60)	Matching Fields or Chaining Fields (61-62)	Field Record Relation (63-64)	Field Indicators Plus (65-66)	Minus (67-68)	Zero or Blank (69-70)	Sterling Sign Position (71-74)
01	I	INDT	AA			Ø1	96		C	A																					
02	I																				1	6Ø	Ø	ITM							

10.



IBM

International Business Machines Corporation

Form X21-9094
Printed in U.S.A.

RPG INPUT SPECIFICATIONS

Date ____________

Program ____________

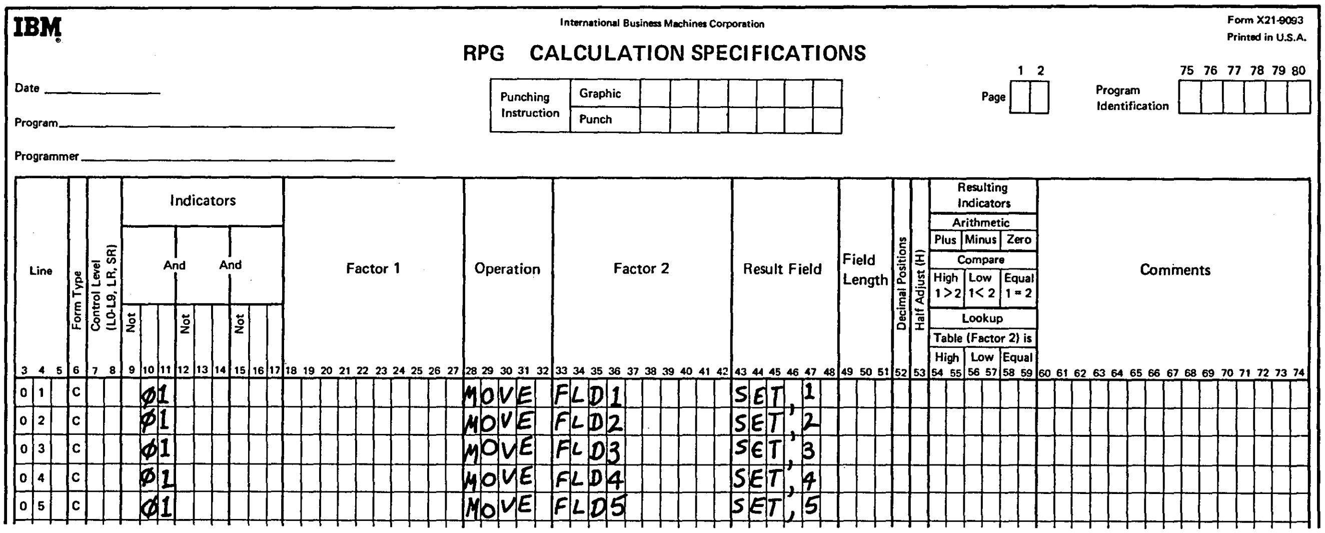
Programmer ____________

Punching Instruction: Graphic / Punch

Page 1 2

Program Identification 75 76 77 78 79 80

Line	Form Type	Filename	Sequence	Number (1-N)	Option (O)	Record Identifying Indicator or **	Record Identification Codes 1: Position	Not (N)	C/Z/D	Character	2: Position	Not (N)	C/Z/D	Character	3: Position	Not (N)	C/Z/D	Character	Stacker Select	P = Packed/B = Binary	Field Location: From	To	Decimal Positions	Field Name	Control Level (L1-L9)	Matching Fields or Chaining Fields	Field Record Relation	Field Indicators: Plus	Minus	Zero or Blank	Sterling Sign Position
3 4 5	6	7 8 9 10 11 12 13 14	15 16	17	18	19 20	21 22 23 24	25	26	27	28 29 30 31	32	33	34	35 36 37 38	39	40	41	42	43	44 45 46 47	48 49 50 51	52	53 54 55 56 57 58	59 60	61 62	63 64	65 66	67 68	69 70	71 72 73 74
0 1	I	INPUT	AA			Ø1	1		C	P																					
0 2	I																				2	4	Ø	FLD1							
0 3	I																				6	8	Ø	FLD2							
0 4	I																				1Ø	12	Ø	FLD3							
0 5	I																				14	16	Ø	FLD4							
0 6	I																				18	2Ø	Ø	FLD5							



IBM

International Business Machines Corporation

Form X21-9093
Printed in U.S.A.

RPG CALCULATION SPECIFICATIONS

Date ____________

Program ____________

Programmer ____________

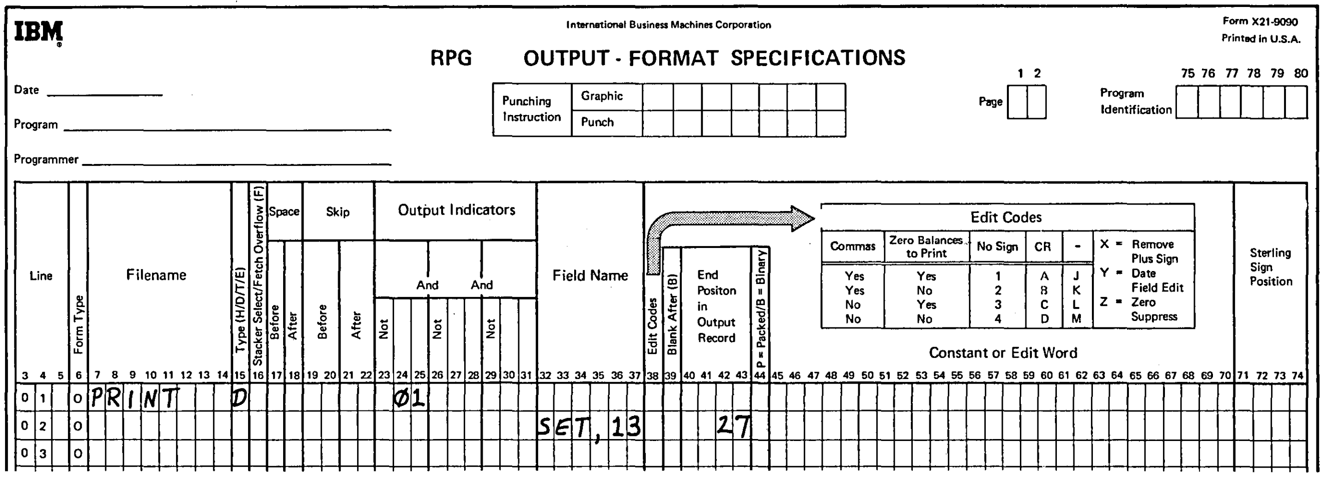
Punching Instruction: Graphic / Punch

Page 1 2

Program Identification 75 76 77 78 79 80

Line	Form Type	Control Level (L0-L9, LR, SR)	Indicators: Not		And: Not		And: Not		Factor 1	Operation	Factor 2	Result Field	Field Length	Decimal Positions	Half Adjust (H)	Resulting Indicators: Arithmetic Plus / Compare High 1>2 / Lookup Table (Factor 2) is High	Minus / Low 1<2 / Low	Zero / Equal 1=2 / Equal	Comments
3 4 5	6	7 8	9	10 11	12	13 14	15	16 17	18 19 20 21 22 23 24 25 26 27	28 29 30 31 32	33 34 35 36 37 38 39 40 41 42	43 44 45 46 47 48	49 50 51	52	53	54 55	56 57	58 59	60 61 62 63 64 65 66 67 68 69 70 71 72 73 74
0 1	C			Ø1						MOVE	FLD1	SET,1							
0 2	C			Ø1						MOVE	FLD2	SET,2							
0 3	C			Ø1						MOVE	FLD3	SET,3							
0 4	C			Ø1						MOVE	FLD4	SET,4							
0 5	C			Ø1						MOVE	FLD5	SET,5							

11. a. Output of 13th field of SET array:



IBM

International Business Machines Corporation

Form X21-9090
Printed in U.S.A.

RPG OUTPUT - FORMAT SPECIFICATIONS

Date ______________

Program ______________

Programmer ______________

Punching Instruction: Graphic / Punch

Page (1 2)

Program Identification (75 76 77 78 79 80)

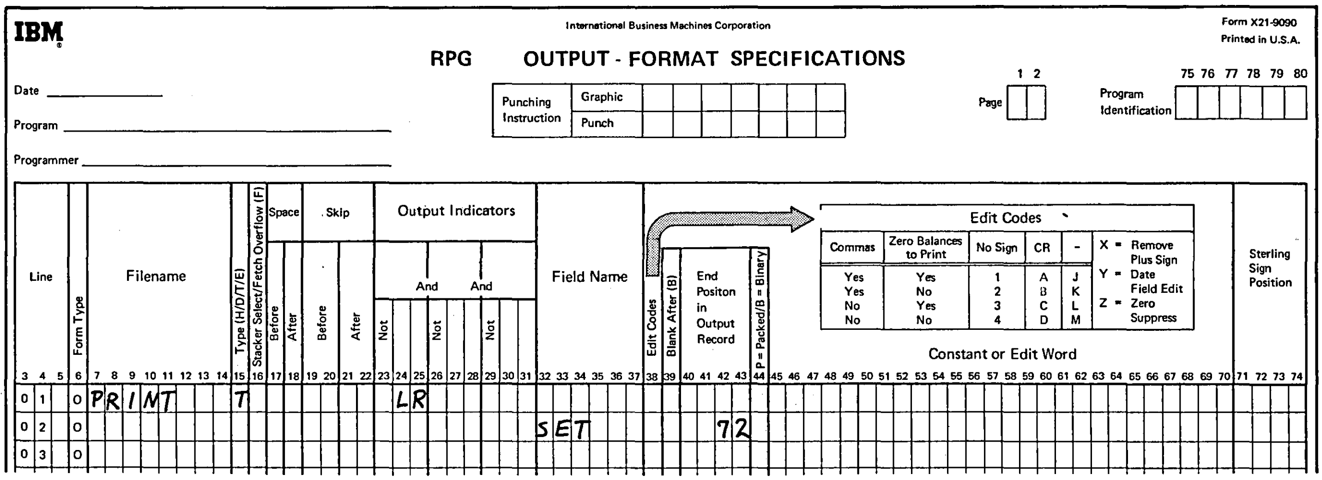
Edit Codes:

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Line (3-5)	Form Type (6)	Filename (7-14)	Type (H/D/T/E) (15)	Stacker Select/Fetch Overflow (F) (16)	Space Before (17)	Space After (18)	Skip Before (19-20)	Skip After (21-22)	Output Indicators (23-31)	Field Name (32-37)	Edit Codes (38)	Blank After (B) (39)	End Position in Output Record (40-43)	P = Packed/B = Binary (44)	Constant or Edit Word (45-70)	Sterling Sign Position (71-74)
01	O	PRINT	D						Ø1							
02	O									SET,13			27			
03	O															

b. Output of entire SET array at end of job:



IBM

International Business Machines Corporation

Form X21-9090
Printed in U.S.A.

RPG OUTPUT - FORMAT SPECIFICATIONS

Date ______________

Program ______________

Programmer ______________

Punching Instruction: Graphic / Punch

Page (1 2)

Program Identification (75 76 77 78 79 80)

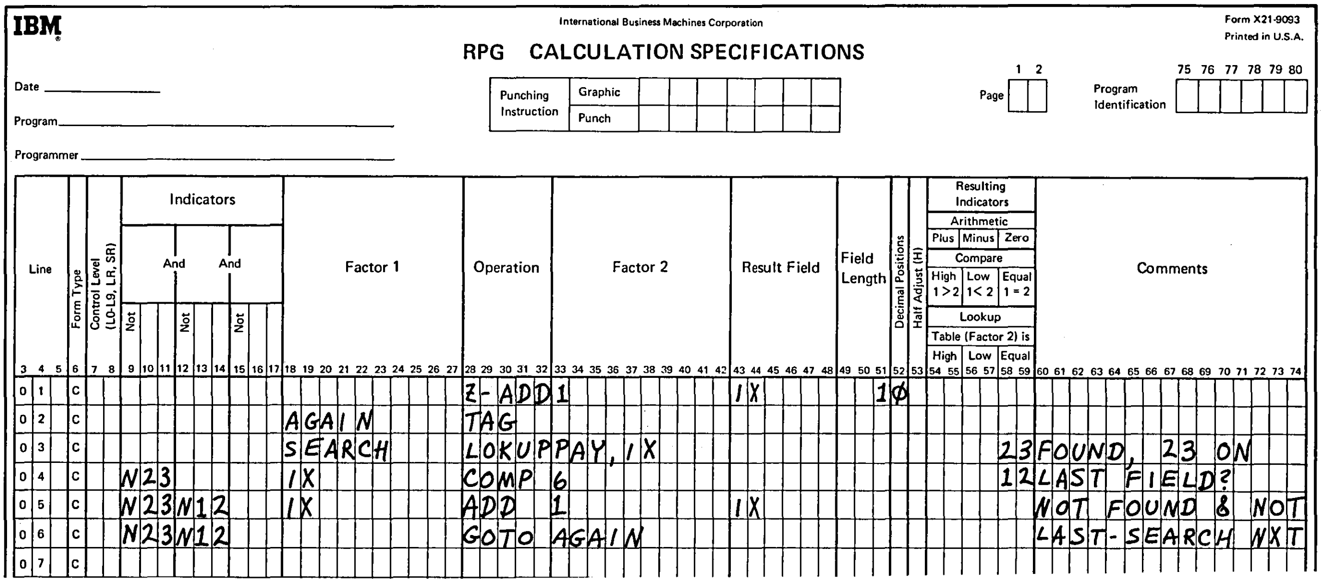
Edit Codes:

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

Line (3-5)	Form Type (6)	Filename (7-14)	Type (H/D/T/E) (15)	Stacker Select/Fetch Overflow (F) (16)	Space Before (17)	Space After (18)	Skip Before (19-20)	Skip After (21-22)	Output Indicators (23-31)	Field Name (32-37)	Edit Codes (38)	Blank After (B) (39)	End Position in Output Record (40-43)	P = Packed/B = Binary (44)	Constant or Edit Word (45-70)	Sterling Sign Position (71-74)
01	O	PRINT	T						LR							
02	O									SET			72			
03	O															

12.



IBM

International Business Machines Corporation

Form X21-9093
Printed in U.S.A.

RPG CALCULATION SPECIFICATIONS

Date

Program

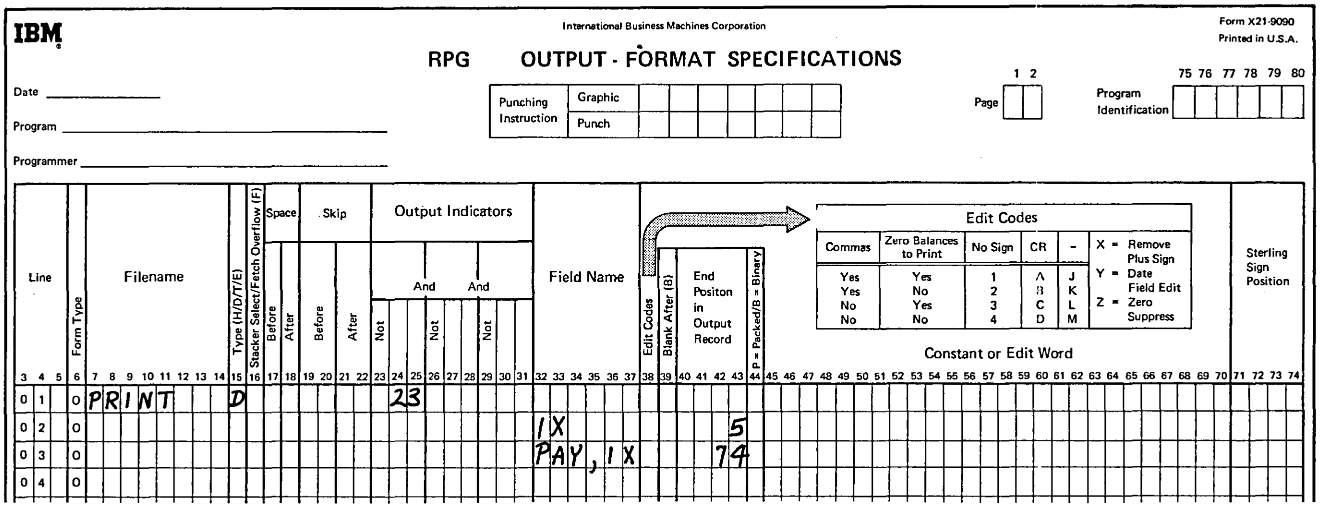
Programmer

Punching Instruction: Graphic / Punch

Page 1 2

Program Identification 75 76 77 78 79 80

Line (3-5)	Form Type (6)	Control Level (7-8)	Indicators (9-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field (43-48)	Field Length (49-51)	Decimal Positions (52)	Half Adjust (53)	Resulting Indicators High (54-55)	Low (56-57)	Equal (58-59)	Comments (60-74)
01	C				Z-ADD	1	IX	1	Ø					
02	C			AGAIN	TAG									
03	C			SEARCH	LOKUP	PAY,IX							23	FOUND, 23 ON
04	C		N23	IX	COMP	6							12	LAST FIELD?
05	C		N23N12	IX	ADD	1	IX							NOT FOUND & NOT
06	C		N23N12		GOTO	AGAIN								LAST-SEARCH NXT
07	C													



IBM

International Business Machines Corporation

Form X21-9090
Printed in U.S.A.

RPG OUTPUT - FORMAT SPECIFICATIONS

Date

Program

Programmer

Punching Instruction: Graphic / Punch

Page 1 2

Program Identification 75 76 77 78 79 80

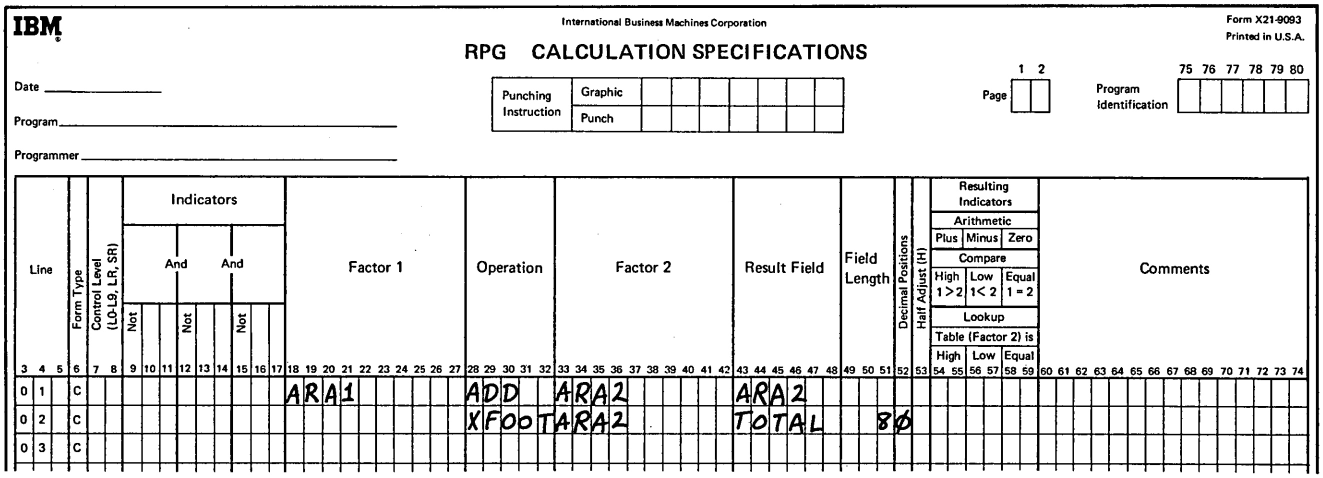
Line (3-5)	Form Type (6)	Filename (7-14)	Type (H/D/T/E) (15)	Stacker Select/Fetch Overflow (F) (16)	Space Before/After (17-18)	Skip Before/After (19-22)	Output Indicators (23-31)	Field Name (32-37)	Edit Codes (38)	Blank After (B) (39)	End Position in Output Record (40-43)	P = Packed/B = Binary (44)	Constant or Edit Word (45-70)	Sterling Sign Position (71-74)
01	O	PRINT	D				23							
02	O							IX			5			
03	O							PAY,IX			74			
04	O													

Edit Codes

Commas	Zero Balances to Print	No Sign	CR	-
Yes	Yes	1	A	J
Yes	No	2	B	K
No	Yes	3	C	L
No	No	4	D	M

X = Remove Plus Sign
Y = Date Field Edit
Z = Zero Suppress

13.



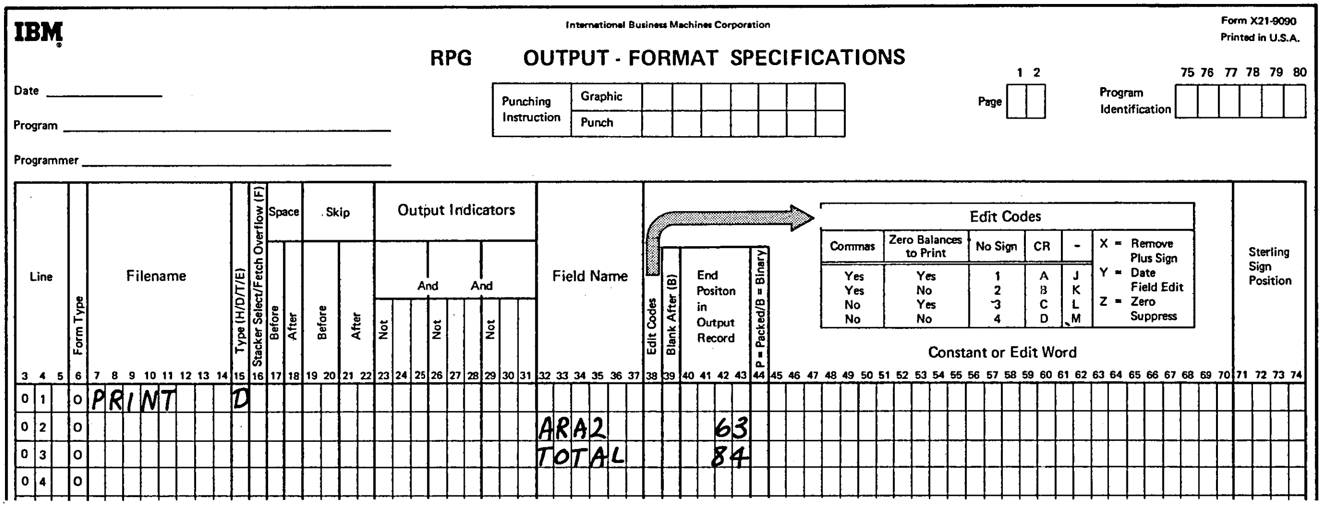
IBM International Business Machines Corporation Form X21-9093 Printed in U.S.A.

RPG CALCULATION SPECIFICATIONS

Date ______ Program ______ Programmer ______

Punching Instruction: Graphic □□□□□□□□ Punch □□□□□□□□ Page (1 2) □□ Program Identification (75 76 77 78 79 80) □□□□□□

Line (3-5)	Form Type (6)	Control Level (L0-L9, LR, SR) (7-8)	Indicators: Not (9-11)	And: Not (12-14)	And: Not (15-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field (43-48)	Field Length (49-51)	Decimal Positions (52)	Half Adjust (H) (53)	Resulting Indicators: Arithmetic Plus / Compare High 1>2 / Lookup Table (Factor 2) is High (54-55)	Minus / Low 1<2 / Low (56-57)	Zero / Equal 1=2 / Equal (58-59)	Comments (60-74)
0 1	C					ARA1	ADD	ARA2	ARA2							
0 2	C						XFOOT	ARA2	TOTAL	8	Ø					
0 3	C															



IBM International Business Machines Corporation Form X21-9090 Printed in U.S.A.

RPG OUTPUT - FORMAT SPECIFICATIONS

Date ______ Program ______ Programmer ______

Punching Instruction: Graphic □□□□□□□□ Punch □□□□□□□□ Page (1 2) □□ Program Identification (75 76 77 78 79 80) □□□□□□

Edit Codes

Commas	Zero Balances to Print	No Sign	CR	-	
Yes	Yes	1	A	J	X = Remove Plus Sign
Yes	No	2	B	K	Y = Date Field Edit
No	Yes	3	C	L	Z = Zero Suppress
No	No	4	D	M	

Line (3-5)	Form Type (6)	Filename (7-14)	Type (H/D/T/E) (15)	Stacker Select/Fetch Overflow (F) (16)	Space Before (17)	Space After (18)	Skip Before (19-20)	Skip After (21-22)	Output Indicators: Not (23-25)	And: Not (26-28)	And: Not (29-31)	Field Name (32-37)	Edit Codes (38)	Blank After (B) (39)	End Positon in Output Record (40-43)	P = Packed/B = Binary (44)	Constant or Edit Word (45-70)	Sterling Sign Position (71-74)
0 1	O	PRINT	D															
0 2	O											ARA2			63			
0 3	O											TOTAL			84			
0 4	O																	

CHAPTER

CHANGING DATA STRUCTURE

16

CHAPTER 16 DESCRIBES:

Representation of characters on cards.

Representation of characters in storage (disk and inside the computer).

Packed and binary data.

Collating sequence of characters.

Move zone operations.

File translation.

AFTER READING THIS CHAPTER YOU SHOULD BE ABLE TO:

Describe the representation of characters and negative numbers on cards.

Describe the representation of characters on disk and inside the computer.

Define byte, bit, zone portion and digit portion.

Compare the storing of characters on cards to the storing of characters in storage.

Identify bit combinations with numerical values.

Assign numerical values to zone and digit portions.

Define unpacked decimal format, packed decimal format, and binary format.

Describe the hexadecimal numbering system.

Describe the collating sequence of characters.

Code specifications to change the collating sequence.

Alter the structure of characters in storage by using move zone operations.

Translate characters by coding the Translation Table and Alternate Collating Sheet.

CHARACTER STRUCTURE

Punched cards provide data the computer is to work with. Each of the 96 columns of a card can contain punches for a single character. Therefore, up to 96 characters of information can be represented on a single record.

Each column of a card consists of six punch positions, labeled *B, A, 8, 4, 2,* and *1,* from the top of a column to the bottom. Characters are represented by a combination of from zero to six holes punched in the punch positions of a single column.

Since there are six punch positions available, the number and positions of the holes may be varied to form 64 different punch combinations. Each unique combination of punches is associated with a particular character. Therefore, you can represent 64 different characters in the computer (Figure 16-1).

			Numeric Characters									
			0	1	2	3	4	5	6	7	8	9
Punch Positions	Zone	B										
		A	A									
	Digit	8									8	8
		4					4	4	4	4		
		2			2	2			2	2		
		1		1		1		1		1		1

			Alphabetic Characters																									
			A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Punch Positions	Zone	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B								
		A	A	A	A	A	A	A	A	A	A										A	A	A	A	A	A	A	A
	Digit	8								8	8								8	8							8	8
		4				4	4	4	4						4	4	4	4					4	4	4	4		
		2		2	2			2	2				2	2			2	2			2	2			2	2		
		1	1		1		1		1		1	1		1		1		1		1		1		1		1		1

			Special Characters																										
			}	¢	.	<	(	+	|	!	$	*	)	;	¬	-	/	&	,	%	—	>	?	:	#	@	'	=	"	b̸
Punch Positions	Zone	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B														
		A	A	A	A	A	A	A	A								A	A	A	A	A	A	A							
	Digit	8		8	8	8	8	8	8	8	8	8	8	8	8			8	8	8	8	8	8	8	8	8	8	8	8	
		4				4	4	4	4			4	4	4	4					4	4	4	4			4	4	4	4	
		2		2	2			2	2	2	2			2	2			2	2			2	2	2	2			2	2	
		1			1		1		1		1		1		1		1		1		1		1		1		1		1	

S1708

Figure 16-1. Character Set and Punch Combinations

A card column consists of both a zone portion and a digit portion. *B* and *A* are referred to as zone punch positions, while *8, 4, 2,* and *1* are digit punch positions. The combinations of zone and digit punches make it possible to separate the characters into three groups (Figure 16-1):

- Alphabetic letters are represented by at least one punch in both the zone and digit portions of a column.
- The 28 special characters can consist of no punches, only zone punches, only digit punches, or both zone and digit punches.
- Positive numbers are represented by holes only in the digit punch positions. The one exception if the number *0* which is represented by a single punch in the *A* zone punch position.

Representation of Negative Numbers

Note that *positive* numbers are represented only by digit punches. Negative numbers (*–1* through *–9*) can also be represented to the computer. However, to indicate that a number is negative, a column must contain both the punch combination for the number and the punch combination for the minus sign. As column *7* of Figure 16-2 shows, the *8* and *1* digit-punch positions are punched to represent the number *9.* A *–9* is represented in column 12 by the same digit punches plus a hole in the *B* zone-punch position.

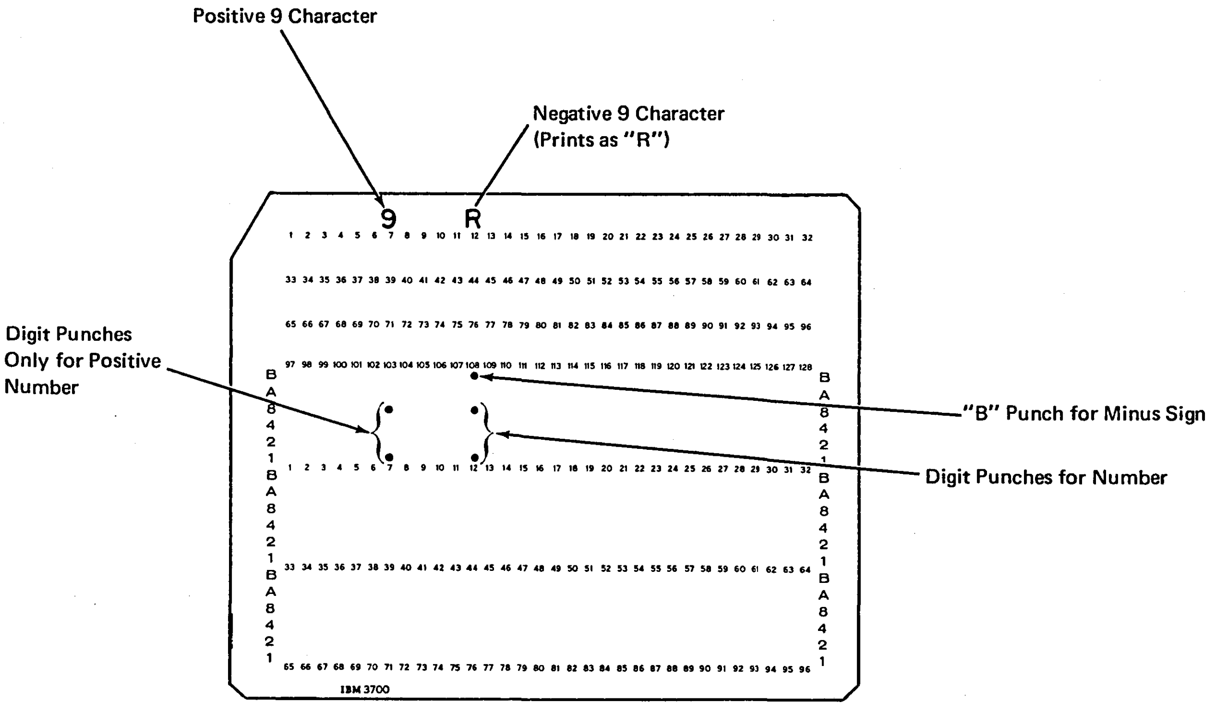


Figure 16-2. Punches for Negative Numbers

As mentioned, all 64 possible punch combinations are associated with a character. Therefore, adding a *B* zone punch to the punch combination of a number means the punch combination for any negative number is the same as the punch combination already assigned to one of the 64 characters. The negative numbers *–1* through *–9* are represented by the same punch combinations as the letters *J* through *R* (Figure 16-3).

The computer determines whether the punch combination is a letter or a number according to whether an entry has been made in column 52 of the input specifications. Column 52 is used to specify the number of decimal positions in a field. If an entry is present, the computer assumes any character in that field to be numeric. Absence of an entry in column 52 tells the computer that it is reading either a letter or a special character in an alphameric field. By examining column 52, the computer recognizes when the *B* zone punch is associated with the punch combination of one of the letters *J* through *R* or the punch combination of a negative number.

In the discussion so far, you have learned how data is recorded in a form which the computer can understand. The data is represented as punched holes on 96-column cards. Before the computer can use the data, as in calculations or output operations, it must store the information. The data is then available in computer storage whenever it is needed during the run of a program.

CHARACTERS

Punch Position	–1	–2	–3	–4	–5	–6	–7	–8	–9
B	●	●	●	●	●	●	●	●	●
A									
8								●	●
4				●	●	●	●		
2		●	●			●	●		
1	●		●		●		●		●

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

Punch Position	J	K	L	M	N	O	P	Q	R
B	●	●	●	●	●	●	●	●	●
A									
8								●	●
4				●	●	●	●		
2		●	●			●	●		
1	●		●		●		●		●

Figure 16-3. Negative Number Punch Combinations the Same as Punch Combinations for J-R

Representation of Characters in Storage

When you look at the punch area of a card, you cannot immediately determine which characters are stored on the card. First, you have to determine which character is associated with a particular punch combination. The punched holes, then, are the means of representing characters on a card. Similarly, a character such as the letter *A* is not stored on disk or inside the computer in a form you would recognize as the letter *A*. On disk or inside the computer, there is also a means of representing characters.

Information from each of the 96 columns of punched cards can be transferred to disk. Data from each column is stored in corresponding positions on disk in the form of magnetized spots.

Characters are represented electronically in computer storage. The storage area of the computer consists of a number of magnetic *bits,* which can be turned on or off by passing an electric current through them. The exact details of how this is done is not important to this discussion; what is important is that each bit can be in either an on state or an off state. We use a *1* to show a bit that is on; while a *0* represents a bit that is off (Figure 16-4).

The magnetic bits inside the computer or on disk are arranged in groups, called *bytes,* just as the punch positions on a card are arranged in groups called columns (Figure 16-4). Just as each column on a card can contain a character, each byte in storage can also contain a character. A particular combination of on and off bits in a byte represent a certain character inside the computer, just as a particular combination of punched and unpunched positions in a column represent that character on a card.

Data is represented on a card, character by character; likewise, data is stored inside the computer, character by character. Just as you can look at a punched card and refer to a character by the particular column containing that character's punch combination, the computer can reference a character by the particular byte in storage which contains that character's bit combination.

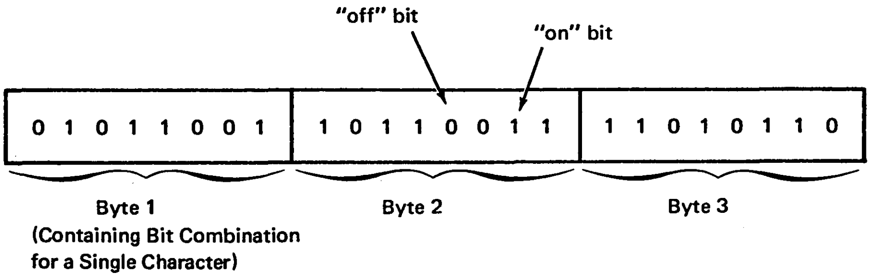


Figure 16-4. Representation of Characters in Storage

Difference Between Character Representation on Cards and in Storage

Although there are many similarities in the way a character is represented in storage and on a punched card, it is important to note one difference. While a card column consists of six positions, a byte consists of eight positions or bits. Thus, within the computer, eight positions are used to represent a single character, whereas only six positions are available on a card.

A byte is divided into a zone portion and a digit portion, just as a card column is (Figure 16-5). The four digit positions, in both a byte and a column, are labeled *1, 2, 4,* and *8.* However, since there are two additional positions (bits) in storage which are used to represent a character, a byte contains four zone positions, whereas a card column contains only two zone positions. The additional two zone positions in each byte are labeled *C* and *D* as shown.

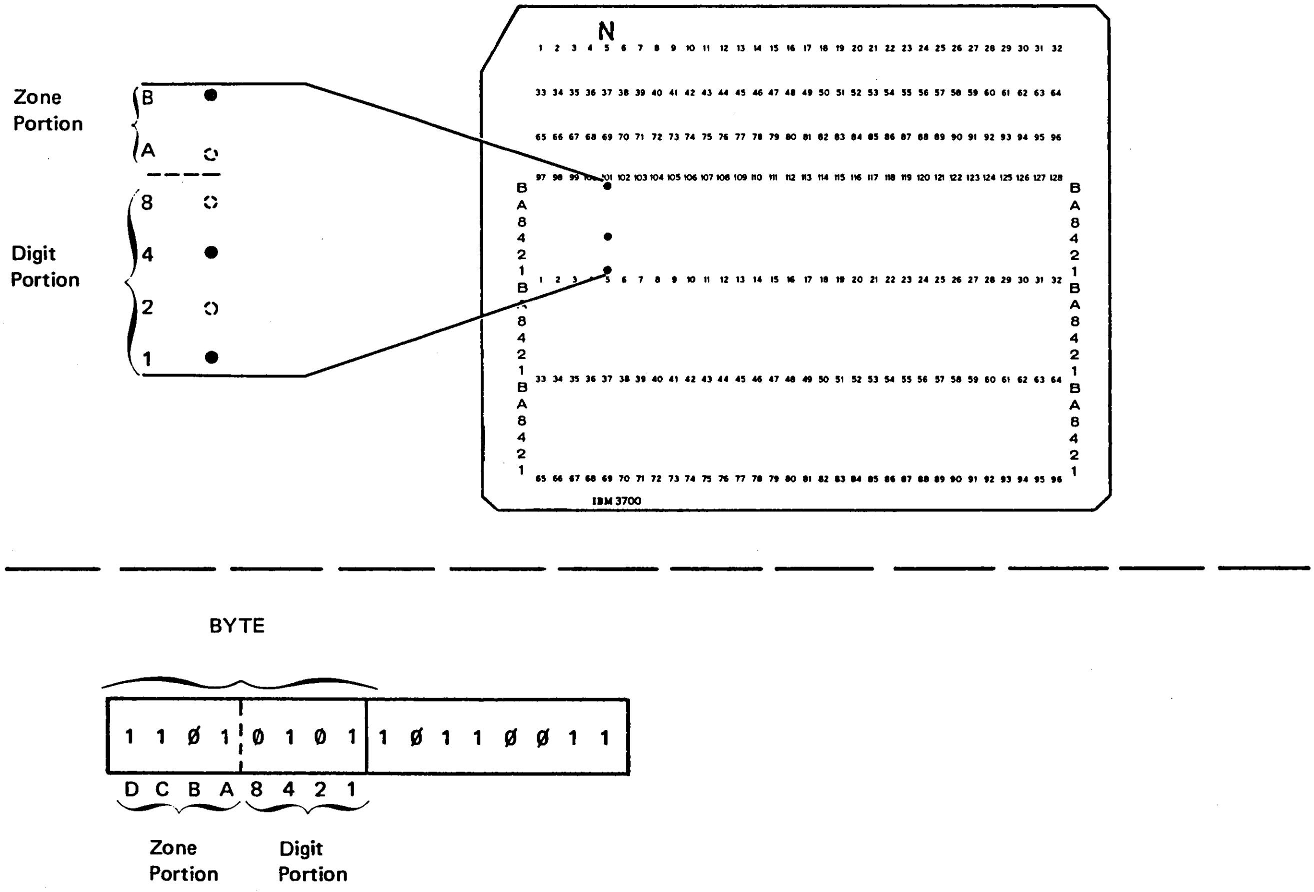


Figure 16-5. Correspondence Between a Byte and a Card Column

Since there are four digit positions in both a byte and a card column, the digit portion of a byte corresponds one-for-one with the digit portion of that character's punch combination. That is, if a digit punch position is punched, the corresponding digit bit is set on *(1)* in storage. Likewise, a digit bit is set off *(0)* if the corresponding punch position does not contain a punch. To check this, note how the digit portion of the plus sign (+) character is represented on a card and in storage (Figure 16-6).

The zone portions of a card column and a byte do not correspond one-to-one, however. This is because there are four zone bits in storage for each character, while there are only two zone positions in a card column. Looking at Figure 16-6 again, you can see that even though *A* and *B* punch positions contain punches for the plus sign character, the *A* and *B* bits in storage are not on.

Since the zone portions differ, a translation must take place when a card is read and the data (characters) is stored inside the computer. This is exactly what happens. The machine reads the punch combination on the card and electronically produces the appropriate bit combinations in storage. Such translations (shown in Figure 16-7) are automatic; therefore, you need not be concerned with how the computer knows which bits to turn on and off.

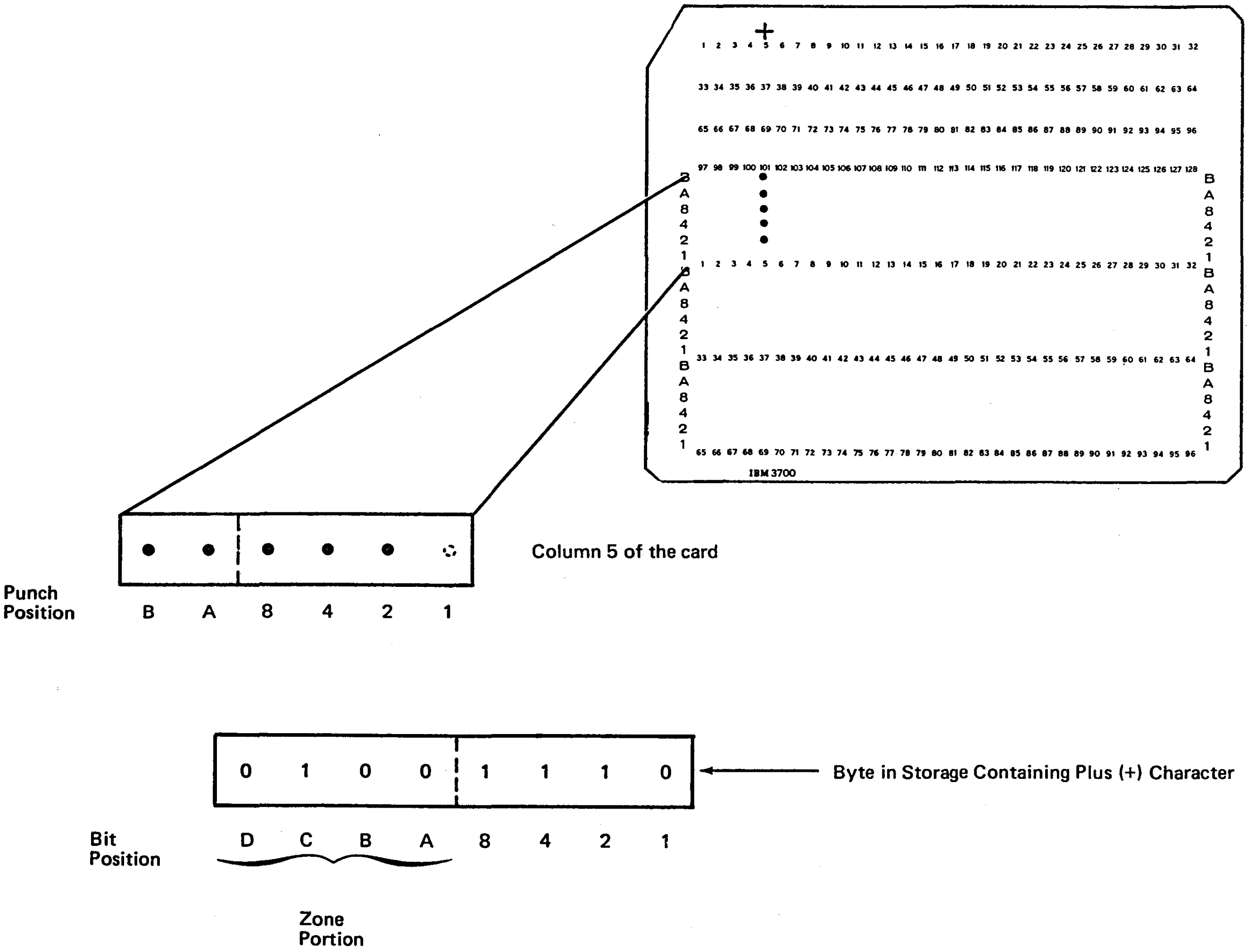


Figure 16-6. Similarity in Digit Portion of Byte and Card Column

Character	Punch Combination						Bit Combination	
	Zone		Digit					
	B	A	8	4	2	1	Zone	Digit
b̸ (blank)							0100	0000
¢	●	●	●		●		0100	1010
. (period)	●	●	●		●	●	0100	1011
<	●	●	●	●			0100	1100
(	●	●	●	●		●	0100	1101
+	●	●	●	●	●		0100	1110
|	●	●	●	●	●	●	0100	1111
&		●	●		●		0101	0000
!	●		●		●		0101	1010
$	●		●		●	●	0101	1011
*	●		●	●			0101	1100
)	●		●	●		●	0101	1101
;	●		●	●	●		0101	1110
¬	●		●	●	●	●	0101	1111
- (minus)	●						0110	0000
/		●				●	0110	0001
,		●	●		●	●	0110	1011
%		●	●	●			0110	1100
_ (underscore)		●	●	●		●	0110	1101
>		●	●	●	●		0110	1110
?		●	●	●	●	●	0110	1111
:			●		●		0111	1010
#			●		●	●	0111	1011
@			●	●			0111	1100
' (apostrophe)			●	●		●	0111	1101
=			●	●	●		0111	1110
"			●	●	●	●	0111	1111

Character	Punch Combination						Bit Combination	
	Zone		Digit					
	B	A	8	4	2	1	Zone	Digit
A	●	●				●	1100	0001
B	●	●			●		1100	0010
C	●	●			●	●	1100	0011
D	●	●		●			1100	0100
E	●	●		●		●	1100	0101
F	●	●		●	●		1100	0110
G	●	●		●	●	●	1100	0111
H	●	●	●				1100	1000
I	●	●	●			●	1100	1001
	●	●					1101	0000
J or -1	●					●	1101	0001
K or -2	●				●		1101	0010
L or -3	●				●	●	1101	0011
M or -4	●			●			1101	0100
N or -5	●			●		●	1101	0101
O or -6	●			●	●		1101	0110
P or -7	●			●	●	●	1101	0111
Q or -8	●		●				1101	1000
R or -9	●		●			●	1101	1001
S		●			●		1110	0010
T		●			●	●	1110	0011
U		●		●			1110	0100
V		●		●		●	1110	0101
W		●		●	●		1110	0110
X		●		●	●	●	1110	0111
Y		●	●				1110	1000
Z		●	●			●	1110	1001
+0		●					1111	0000
1						●	1111	0001
2					●		1111	0010
3					●	●	1111	0011
4				●			1111	0100
5				●		●	1111	0101
6				●	●		1111	0110
7				●	●	●	1111	0111
8			●				1111	1000
9			●			●	1111	1001

Figure 16-7. Bit and Punch Combinations for Characters

When programming in RPG II, however, you do have to be concerned with zones and digits as they are represented inside the computer. The division of the card column into zone and digit portions is only for convenience.

On the Input Sheet, you can specify record identification codes. If you choose to use only the zone portion of a character, you will be using the zone positions as they are in storage. Assume that a record identification code with the zone of a $ character in column *1* is to turn on resulting indicator 21. If an input record is read with the character *J* in column *1,* indicator 21 will not turn on. Even though the card zone punches for $ and *J* are the same (both have the *B* zone punched), the bit combinations in storage for the $ and *J* do not have identical zone portions (Figure 16-8).

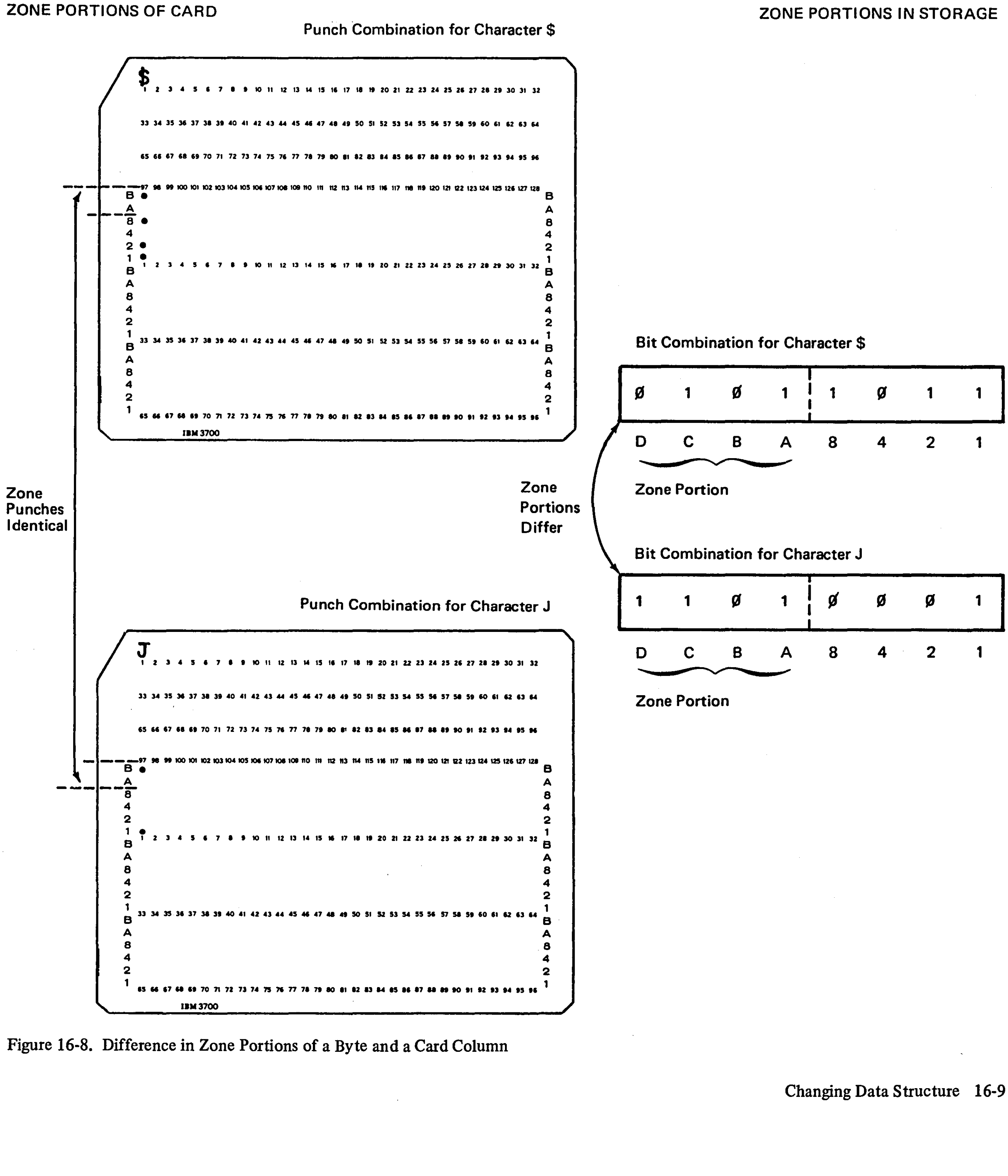


Figure 16-8. Difference in Zone Portions of a Byte and a Card Column

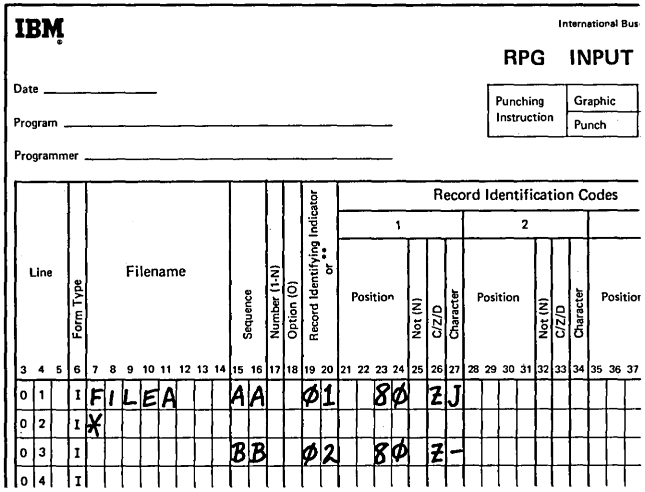
There is one exception which should be noted in specifying that the zones of characters be used to identify records. According to Figure 16-9, the zone of the letter *J* is used to identify record type 01, while the zone of a minus sign is used to identify record type 02. Recorded on cards, both characters have a *B* zone punch. However, inside the computer, their zone representations differ:

– = 0110
DCBA

J = 1101
DCBA

Although the zones differ, the computer considers the two the same. Thus, a card with a minus punched might turn on either the 01 or 02 indicator. Likewise, a card with the letter *J* could turn on either indicator.

The reason the computer treats both zone representations the same is to aid programmers whose files were set up for former systems. To avoid any confusion, we suggest that you not specify both (zones of the letters *J* and –) for identification of record types which are to be used in the same program.



IBM

International Bus

RPG INPUT

Date ______

Program ______

Programmer ______

Punching Instruction	Graphic
	Punch

							Record Identification Codes								
							1				2				
Line	Form Type	Filename	Sequence	Number (1-N)	Option (O)	Record Identifying Indicator or **	Position	Not (N)	C/Z/D	Character	Position	Not (N)	C/Z/D	Character	Positior
3 4 5	6	7 8 9 10 11 12 13 14	15 16	17	18	19 20	21 22 23 24	25	26	27	28 29 30 31	32	33	34	35 36 37
0 1	I	FILEA	AA			Ø1	8Ø		Z	J					
0 2	I	*													
0 3	I		BB			Ø2	8Ø		Z	–					
0 4	I														

Figure 16-9. Exception: Zone Representation Considered the Same

Consider another example which points out the difference in how negative numbers are stored and how you may think they are stored. The minus sign alone is represented on a card and in storage as shown in Figure 16-10, insert A. Only the zone portion of the card contains a punch. Figure 16-10, insert B shows how a positive *5* is represented on a card and in storage. In this case, only the digit portion of the card contains punches. Note Figure 16-10, insert C for the punch and bit combinations which represent a *–5*.

When checking the cards you can see that the digit punches for the positive *5* and the negative *5* are the same. Furthermore, the digit bits in storage for the two characters are also the same. The zone punch for *–5* is the same as the zone punch for the minus sign character. Therefore, you would not always assume that, in storage, the zone bits for the *–5* should be identical to the zone bits for the minus sign (Figure 16-10).

The reason is that the computer checks the *entire* punch combination (both zone and digit portions) of a column to determine which *zone* bits are to be on or off. Since the entire punch combinations (not only zone punches) for the minus character and the negative *5* are different, their zone bits in storage are also different.

A conclusion can be drawn from the previous examples: each unique punch combination is associated with a different bit combination. Of course, in discussing negative numbers before, we stated that the punch combinations of *–1* through *–9* are the same as the punches for the letters *J* through *R*. Thus, the bit combinations for the negative numbers are also the same as those for *J* through *R*.

Consider again the number of positions available to represent a character. A characteristic of codes involving different combinations (such as bits on and off or punches) is that the greater the number of positions available to represent any one combination, the greater the number of combinations that are possible. As mentioned, with six punch positions, 64 unique punch combinations can be made, and, therefore, 64 different characters can be represented on a card. With eight bits (positions), 256 unique combinations of on-off bits can be created and, therefore, 256 different characters could be represented inside the computer. However, you only need 64 characters to program the computer; therefore, only 64 of its 256 bit combinations are associated with a printable character.

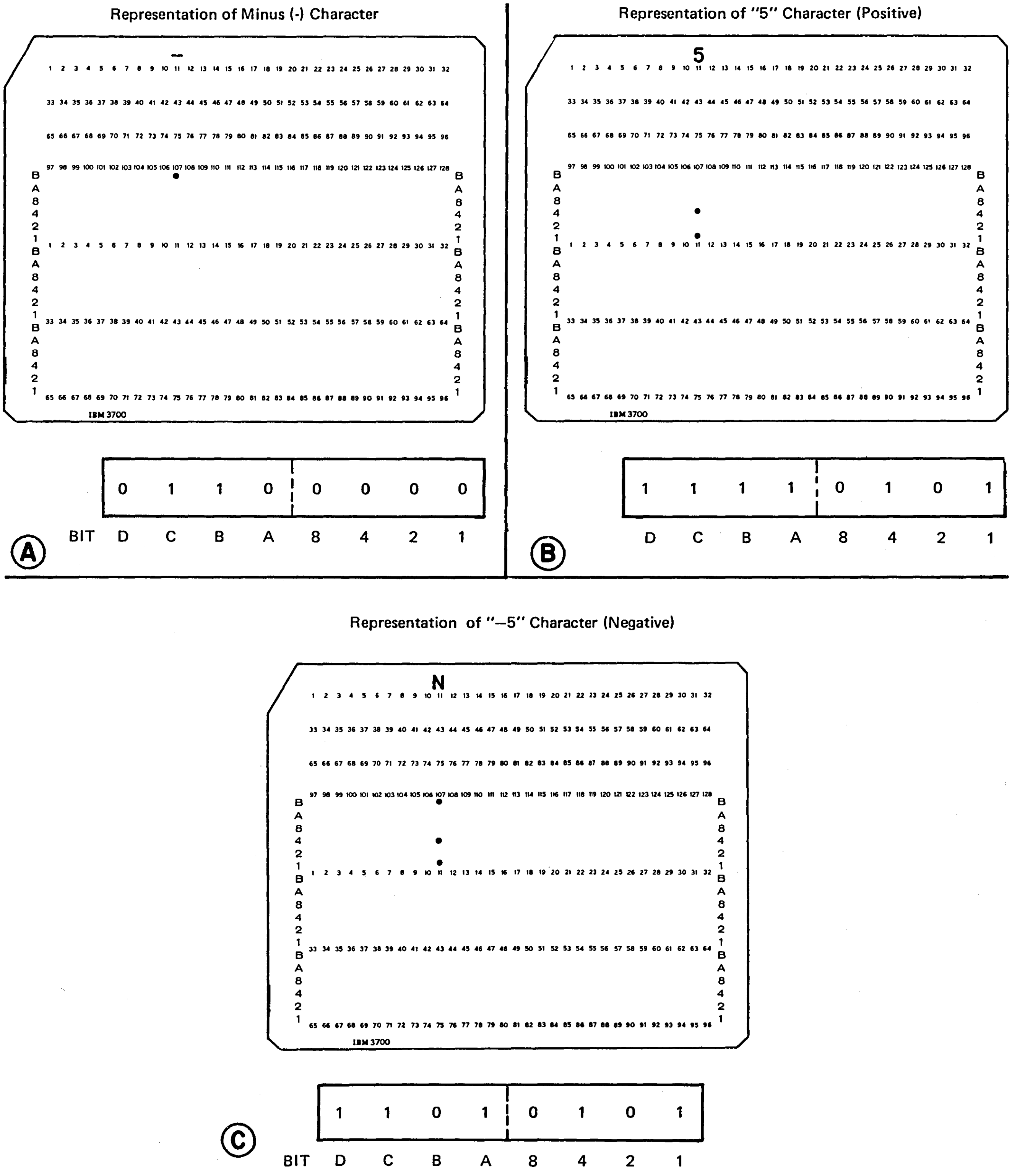


Figure 16-10. Representation of a Negative Number

Identifying Bit Combinations with Numerical Values

Each unique combination of eight bits can be associated with a numerical value. Before discussing how the numerical value is determined for a character, perhaps first you would like to know why numeric values are assigned.

As mentioned before, data is represented on punched cards because the computer can read punched holes. Actually, after reading a card, the computer does not immediately determine what character is punched. It can, however, distinguish one punch combination from another punch combination. Furthermore, the particular combination of punches indicates to the computer which bits should be set on and off to represent that punch combination inside the machine. At this point, the representation on the card is just a particular group of punches and the representation in storage is merely a particular combination of on and off bits.

To use the byte of data for output, the computer must know what character to punch or print. This is done by associating a numerical value with each unique bit combination. The computer automatically knows that a certain value is related to a particular character, such as the value 209 indicates the character *J.*

Consider how a numerical value and how the character are determined. Each of the eight bits in a byte are assigned a number. The values begin with *1* for the *1* bit and are doubled for each of the next bits (Figure 16-11). By adding only the numbers which correspond to bits which are on *(1),* a numerical value is obtained for a byte. As Figure 16-11 shows, first the punch combination (for the character *F*) in column 7 is translated into the bit combination in storage. The bits *on* result in a numerical value of 198, which the computer associates with the character *F.*

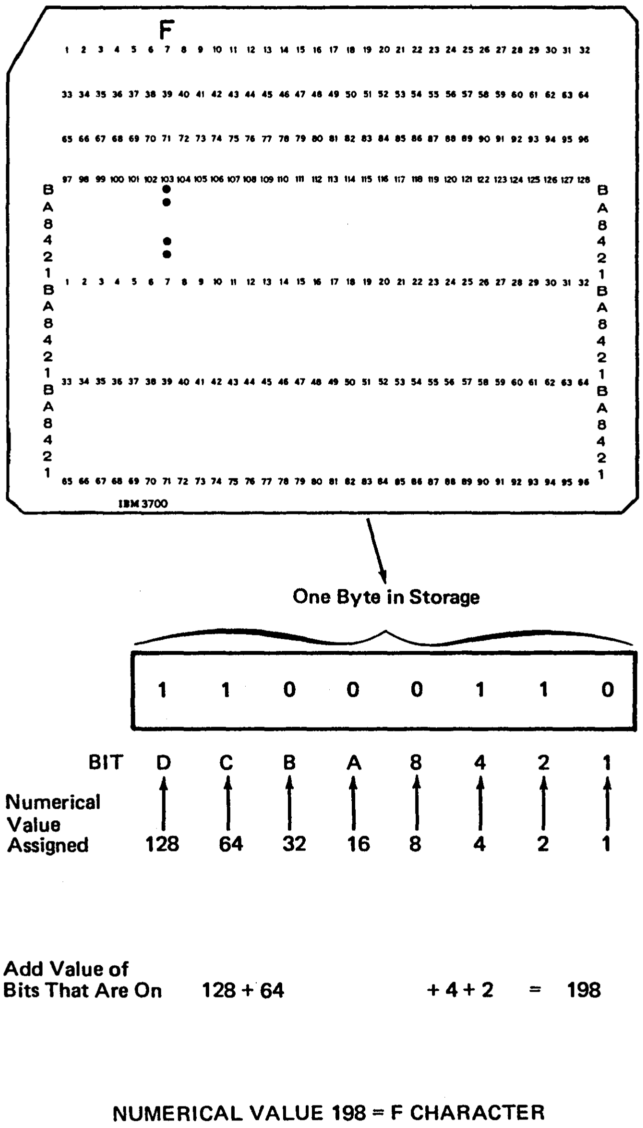

F
B
A
8
4
2
1
IBM 3700
One Byte in Storage
1 1 0 0 0 1 1 0
BIT D C B A 8 4 2 1
Numerical Value Assigned 128 64 32 16 8 4 2 1
Add Value of Bits That Are On 128 + 64 + 4 + 2 = 198
NUMERICAL VALUE 198 = F CHARACTER

Figure 16-11. Determining a Numerical Value for a Character

Any difference in the bit combination results in a difference in numerical value. Therefore, every character is associated with a different numerical value. The greatest numerical value which can be associated with a bit combination is 255 (all eight bits on), while the lowest numerical value is 0 (all eight bits off). This results in a total of 256 possible numerical values. Only 64 different characters can be represented on a card; therefore, we are concerned with only 64 of the different numerical values. However, as Figure 16-12 shows, the 64 numerical values associated with the characters can range anywhere from 0 through 255. The numerical values missing from the chart are not related to any printable character.

Bit Combination	Numerical Value	Character
00000000	0	
00000001	1	
00000010	2	
00000011	3	
00000100	4	
00000101	5	
00000110	6	
00000111	7	
00001000	8	
00001001	9	
00001010	10	
00001011	11	
00001100	12	
00001101	13	
00001110	14	
00001111	15	
00010000	16	
00010001	17	
00010010	18	
00010011	19	
00010100	20	
00010101	21	
00010110	22	
00010111	23	
00011000	24	
00011001	25	
00011010	26	
00011011	27	
00011100	28	
00011101	29	
00011110	30	
00011111	31	
00100000	32	
00100001	33	
00100011	34	
00100011	35	
00100100	36	
00100101	37	
00100110	38	
00100111	39	
00101000	40	
00101001	41	
00101010	42	
00101011	43	
00101100	44	
00101101	45	
00101110	46	
00101111	47	
00110000	48	
00110001	49	
00110010	50	

Figure 16-12. Numerical Values Associated with Characters (part 1 of 3)

Bit Combination	Numerical Value	Character
00110011	51	
00110100	52	
00110101	53	
00110110	54	
00110111	55	
00111000	56	
00111001	57	
00111010	58	
00111011	59	
00111100	60	
00111101	61	
00111110	62	
00111111	63	
01000000	64	Blank
01000001	65	
01000010	66	
01000011	67	
01000100	68	
01000101	69	
01000110	70	
01000111	71	
01001000	72	
01001001	72	
01001010	74	¢
01001011	75	.
01001100	76	<
01001101	77	(
01001110	78	+
01001111	79	|
01010000	80	&
01010001	81	
01010010	82	
01010011	83	
01010100	84	
01010101	85	
01010110	86	
01010111	87	
01011000	88	
01011001	89	
01011010	90	!
01011011	91	$
01011100	92	*
01011101	93	)
01011110	94	;
01011111	95	¬
01100000	96	-
01100001	97	/
01100010	98	
01100011	99	
01100100	100	
01100101	101	

Bit Combination	Numerical Value	Character
01100110	102	
01100111	103	
01101000	104	
01101001	105	
01101010	106	
01101011	107	,
01101100	108	%
01101101	109	_
01101110	110	>
01101111	111	?
01110000	112	
01110001	113	
01110010	114	
01110011	115	
01110100	116	
01110101	117	
01110110	118	
01110111	119	
01111000	120	
01111001	121	
01111010	122	:
01111011	123	#
01111100	124	@
01111101	125	'
01111110	126	=
01111111	127	"
10000000	128	
10000001	129	
10000010	130	
10000011	131	
10000100	132	
10000101	133	
10000110	134	
10000111	135	
10001000	136	
10001001	137	
10001010	138	
10001011	139	
10001100	140	
10001101	141	
10001110	142	
10001111	143	
10010000	144	
10010001	145	
10010010	146	
10010011	147	
10010100	148	
10010101	149	
10010110	150	
10010111	151	
10011000	152	

Figure 16-12. Numerical Values Associated with Characters (part 2 of 3)

Bit Combination	Numerical Value	Character
10011001	153	
10011010	154	
10011011	155	
10011100	156	
10011101	157	
10011110	158	
10011111	159	
10100000	160	
10100001	161	
10100010	162	
10100011	163	
10100100	164	
10100101	165	
10100110	166	
10100111	167	
10101000	168	
10101001	169	
10101010	170	
10101011	171	
10101100	172	
10101101	173	
10101110	174	
10101111	175	
10110000	176	
10110001	177	
10110010	178	
10110011	179	
10110100	180	
10110101	181	
10110110	182	
10110111	183	
10111000	184	
10111001	185	
10111010	186	
10111011	187	
10111100	188	
10111101	189	
10111110	190	
10111111	191	
11000000	192	
11000001	193	A
11000010	194	B
11000011	195	C
11000100	196	D
11000101	197	E
11000110	198	F
11000111	199	G
11001000	200	H
11001001	201	I
11001010	202	
11001011	203	

Bit Combination	Numerical Value	Character
11001100	204	
11001101	205	
11001110	206	
11001111	207	
11010000	208	}
11010001	209	J or -1
11010010	210	K or -2
11010011	211	L or -3
11010100	212	M or -4
11010101	213	N or -5
11010110	214	O or -6
11010111	215	P or -7
11011000	216	Q or -8
11011001	217	R or -9
11011010	218	
11011011	219	
11011100	220	
11011101	221	
11011110	222	
11011111	223	
11100000	224	
11100001	225	
11100010	226	S
11100011	227	T
11100100	228	U
11100101	229	V
11100110	230	W
11100111	231	X
11101000	232	Y
11101001	233	Z
11101010	234	
11101011	235	
11101100	236	
11101101	237	
11101110	238	
11101111	239	
11110000	240	0
11110001	241	1
11110010	242	2
11110011	243	3
11110100	244	4
11110101	245	5
11110110	246	6
11110111	247	7
11111000	248	8
11111001	249	9
11111010	250	
11111011	251	
11111100	252	
11111101	253	
11111110	254	
11111111	255	

Figure 16-12. Numerical Values Associated with Characters (part 3 of 3)

Assigning Numerical Values to Zone and Digit Portions

You have seen how a single numerical value is determined for a combination of eight bits. The numerical value of a character in storage can also be expressed as a pair of numbers, rather than a single value. One number designates the value of only the four zone bits; the other number represents the value of the four digit bits.

You may be wondering why a character would ever be associated with two paired numbers, since it can be associated with just a single number. In certain jobs, you may be concerned with only the digit portion or only the zone portion of a character. For example, if records within a group are to be sequence checked only on the basis of the zone of a character, the computer must look at only the zone bits and determine a numerical value for the zone portion alone in order to make the comparison. Also, if you want to alter the collating sequence or translate a file, both to be discussed later, you must know the separate values for the zone and digit portions of a character.

Determining separate zone and digit values is similar to determining a single value for an entire bit combination; that is, values are assigned to each of the bit positions. The values which correspond to *on* bits *(1)* are then added to obtain a value.

To determine separate values, the zone and digit portions are each treated as separate 4-bit combinations. The four bits in each portion are assigned the values *1, 2, 4,* and *8* (Figure 16-13). The rightmost zone and digit bits each have the value *1*; while the leftmost bits in each portion are assigned the value *8.* A value for the zone portion of a byte is determined by adding only the values corresponding to zone bits which are on *(1).* Likewise, a digit value is obtained by considering only digit bits which are on.

As Figure 16-14, insert A shows, the bit combination for the slash (/) character produces a zone value of 6 and a digit value of 1. Putting the two values together, the entire character can be expressed as the value 61. Note, however, that this is not the same as the numerical value 61 in our decimal numbering system. If we were to determine a numerical value for the *entire* 8-bit combination, we would obtain the value 97 (Figure 16-14, insert B).

	Zone				Digit			
	1	0	1	0	0	0	1	1
Bit	D	C	B	A	8	4	2	1
Assigned Value	8	4	2	1	8	4	2	1

Figure 16-13. Assigning Values for Zone and Digit Portions of a Character

VALUE BY ZONE AND DIGIT

	Zone				Digit			
Bit Combination for "/" Character	0	1	1	0	0	0	0	1
Bit	D	C	B	A	8	4	2	1
Values For "ON" Bits	4 + 2							+ 1
Ⓐ	Zone Value 6				Digit Value 1			

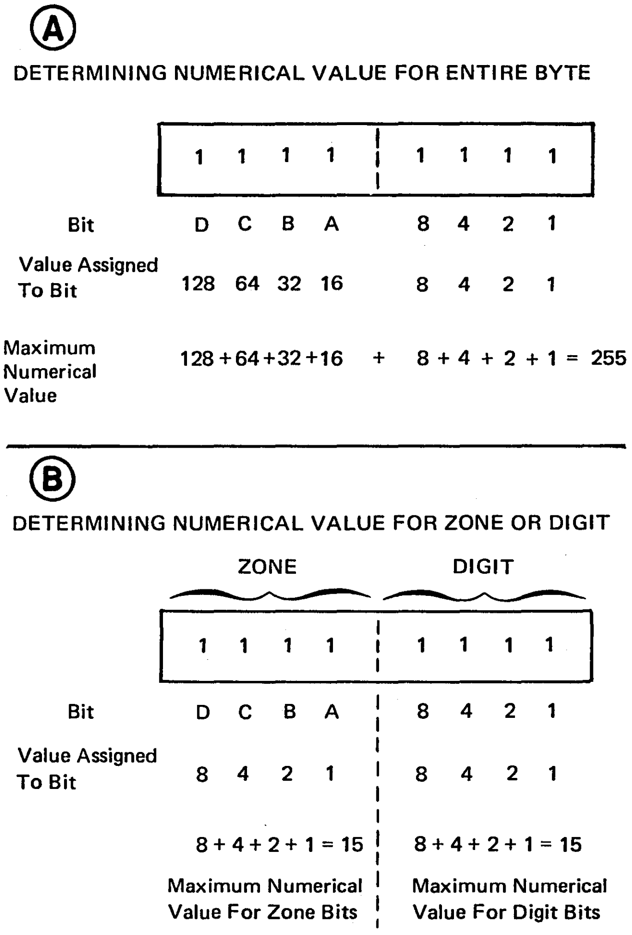
VALUE FOR ENTIRE BIT COMBINATION

Bit Combination For "/" Character	0	1	1	0	0	0	0	1
Bit	D	C	B	A	8	4	2	1
Values For "ON" Bits	64 + 32							+ 1
Ⓑ	Value 97							

Figure 16-14. Difference in Value of Entire Character and Value of Zone and Digit Portions of Character

As mentioned before, with eight bits or positions in a byte, 256 different 8-bit combinations can be formed. The 256 combinations can be associated with the numerical values 0-255 (Figure 16-15, insert A). If either the zone or digit portion are considered separately, however, only four bits or positions are available. Therefore, a maximum of 16 different 4-bit combinations can be represented in either the zone or digit portion of a byte. The 16 zone or digit combinations can be associated with the values 0 through 15 (Figure 16-15, insert B).

The value obtained for a zone or digit bit combination is referred to as *hexadecimal* number. Hex means 6, while decimal refers to 10. Hexadecimal, then, means 6 + 10, or 16. A hexadecimal number can be any one of 16 possible values (0-15). Putting the two hexadecimal numbers for the zone and digit portions together gives a hexadecimal value for the entire character. *61* is the hexadecimal value for the / character. Keep in mind that this hexadecimal value is actually two separate values, one for the zone and one for the digit portion.



Ⓐ

DETERMINING NUMERICAL VALUE FOR ENTIRE BYTE

	1	1	1	1		1	1	1	1
Bit	D	C	B	A		8	4	2	1
Value Assigned To Bit	128	64	32	16		8	4	2	1

Maximum Numerical Value 128+64+32+16 + 8 + 4 + 2 + 1 = 255

Ⓑ

DETERMINING NUMERICAL VALUE FOR ZONE OR DIGIT

	ZONE				DIGIT			
	1	1	1	1	1	1	1	1
Bit	D	C	B	A	8	4	2	1
Value Assigned To Bit	8	4	2	1	8	4	2	1

8 + 4 + 2 + 1 = 15 Maximum Numerical Value For Zone Bits

8 + 4 + 2 + 1 = 15 Maximum Numerical Value For Digit Bits

Figure 16-15. Maximum Values for Entire Character and for Zone and Digit Portions

All of the 256 possible 8-bit combinations can be represented by a hexadecimal value; that is, two hexadecimal numbers. However, each hexadecimal number can take up only one position. If a zone portion has the value 15 and a digit portion has the value 12, the hexadecimal value for the character cannot be expressed as 1512. Consequently, a zone or digit portion whose numerical value is 10 or greater (2-position number) must be represented in a slightly different form. This is done by assigning a single letter as a substitute for the number. The letters *A* through *F* serve as the hexadecimal forms of the values 10 through 15 as shown in Figure 16-16.

An 8-bit combination with a zone value of 15 and a digit value of 12 is expressed as having a hexadecimal value of FC. Because the complete numbering series is composed of numbers *0* through *9* followed by letters *A* through *F*, a hexadecimal value for the zone and digit portion of an 8-bit combination can appear as a pair of numbers (61), a letter and a number (C4, 4F), or a pair of letters (DB).

Since zone and digit values are determined separately, a single combination of eight bits can have the same hexadecimal number for both the zone and digit portion. Thus 11, 22, 33, AA, and other such values represent 8-bit combinations which have the same bits on in both their zone and digit portions.

Entirely different 8-bit combinations can have identical zone hexadecimal numbers *or* identical digit hexadecimal numbers, but not both. That is, the zone portion of one character can contain the same bits on and off as the zone portion of another character. In such a case, the identical zone bit combinations would give identical zone hexadecimal values. However, if they are different characters and the zone values are identical, the digit bits and, thus, the digit values, must differ. This is because no two characters can have the same 8-bit combination.

Figure 16-17 shows the hexadecimal values associated with the 64 printable characters the computer recognizes. The hexadecimal values are in sequence just as the regular numerical values are. Furthermore, the hexadecimal value associated with a character is equivalent to the numerical value associated with that character. However, there is no need for you to be able to translate back and forth between numerical values and hexadecimal values. If you must use a character's hexadecimal value in your programming, you can refer to the chart showing the appropriate value.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
Hexadecimal Number	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Figure 16-16. Hexadecimal Values of Numbers 0-15

Character	Hexadecimal Value	Numerical Value
Blank	40	64
c	4A	74
.	4B	75
<	4C	76
(	4D	77
+	4E	78
|	4F	79
&	50	80
!	5A	90
$	5B	91
*	5C	92
)	5D	93
;	5E	94
¬	5F	95
-	60	96
/	61	97
,	6B	107
%	6C	108
—	6D	109
>	6E	110
?	6F	111
:	7A	122
#	7B	123
@	7C	124
'	7D	125
=	7E	126
"	7F	127
A	C1	193
B	C2	194
C	C3	195
D	C4	196
E	C5	197
F	C6	198
G	C7	199
H	C8	200
I	C9	201
}	D0	208
J or -1	D1	209
K or -2	D2	210
L or -3	D3	211
M or -4	D4	212
N or -5	D5	213
O or -6	D6	214
P or -7	D7	215
Q or -8	D8	216
R or -9	D9	217
S	E2	226
T	E3	227
U	E4	228
V	E5	229
W	E6	230
X	E7	231
Y	E8	232
Z	E9	233
0	F0	240
1	F1	241
2	F2	242
3	F3	243
4	F4	244
5	F5	245
6	F6	246
7	F7	247
8	F8	248
9	F9	249

Figure 16-17. Hexadecimal Values Associated with Characters

Packed Decimal Format

As you have learned, each byte of storage, whether on disk or in the computer, can contain one character. That character can be a decimal number or an alphabetic or special character. The format of the characters is known as *unpacked decimal format.* Each byte of storage is divided into a 4-bit zone and a 4-bit digit part. Figure 16-18 shows the unpacked decimal format.

The zone part of the low-order (rightmost) byte indicates whether the decimal number is positive or negative. In unpacked decimal format, the zone part is included for each digit in a decimal number; however, only the zone over the low-order digit serves as the sign. The low-order digit is the only digit which makes use of the zone portion. Figure 16-19 shows the unpacked decimal format for decimal number 28,191.

Packed decimal format means that one byte of storage can contain two decimal numbers. A decimal number will occupy the zone portion which is unused in unpacked decimal format. This format allows you to put almost twice as much data into a byte as you can using the unpacked decimal format.

The low-order byte in packed decimal format is also divided into two 4-bit parts. Each byte, except the low-order byte, contains one decimal digit in each 4-bit part. The low-order byte contains a decimal digit in the leftmost 4-bit part (bits 0-3) and the sign of the decimal field in the rightmost 4-bit part (bits 4-7). Figure 16-20 shows packed decimal format.

The sign part of the low-order byte is used to indicate whether the numeric value represented in the digit parts is positive or negative. Compare how the decimal number 28,191 is represented in packed decimal format (Figure 16-21) with its unpacked representation (Figure 16-19).

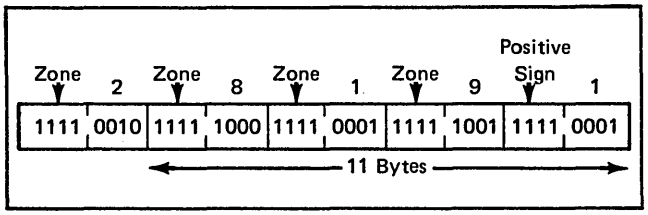


Figure 16-19. Unpacked Format of Decimal Number 28,191

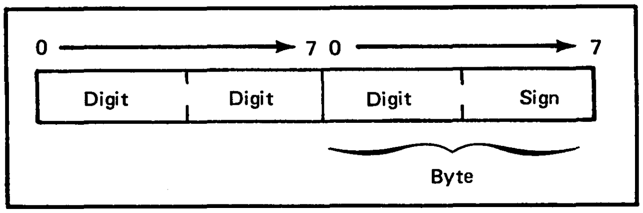


Figure 16-20. Packed Decimal Format

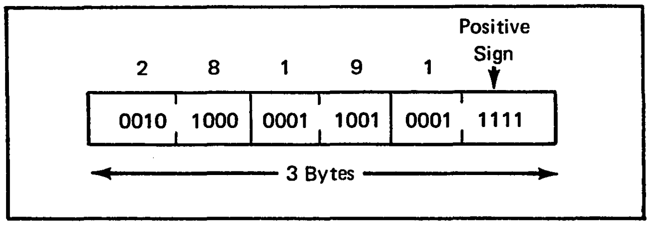


Figure 16-21. Packed Format of Decimal Number 28,191

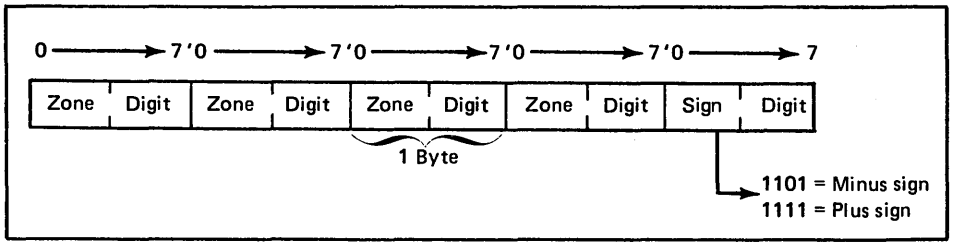


Figure 16-18. Unpacked Decimal Format

You can specify packed input, output, table, or array fields:

- Packed input fields. Enter a *P* in column 43 of the Input Sheet. This causes the data to be unpacked before it is stored.

- Packed output fields. Enter a *P* in column 44 of the Output-Format Sheet. This causes the data to be packed before it is written out.

- Packed table or array fields. Enter a *P* in column 55 of the Extension Sheet. The data will be unpacked before it is stored. Packed tables or arrays are not allowed at compile time.

Since data must be represented in unpacked decimal format once it is inside the computer, you must give the RPG program an indication when input fields are in a different format.

Because data must be represented in unpacked decimal format before it can be processed, it is correct to store data on disk and read it into the computer in the unpacked decimal format. This eliminates converting the input field.

Binary Format

To save a maximum amount of space on disk, you can store numeric fields in *binary format.* Binary format means that two bytes of disk storage can contain a maximum of four decimal numbers, and that four bytes of disk storage can contain a maximum of nine decimal numbers. In the binary format, each field on disk must be either two or four bytes long.

Each 2-byte binary field consists of a 1-bit sign followed by a 15-bit numeric value. In binary format, a decimal number as large as 9,999 requires two bytes of disk storage. For each 2-byte binary field stored on disk, the system automatically sets aside four bytes of storage to accomodate the field when it is converted to zoned decimal. Figure 16-22 shows a 2-byte field in binary format.

Each 4-byte binary field consists of a 1-bit sign followed by a 31-bit numeric value. In binary format, a decimal number as large as 999,999,999 requires four bytes of disk storage. For each 4-byte binary field stored on disk, the system sets aside nine bytes of main storage to accomodate the field when it is converted to zoned decimal. Figure 16-23 shows a 4-byte field in binary format.

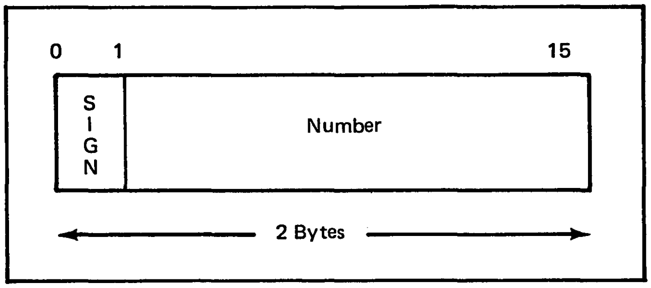


Figure 16-22. Two-byte Field in Binary Format

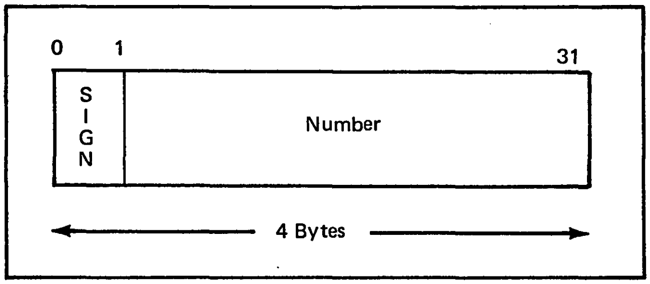
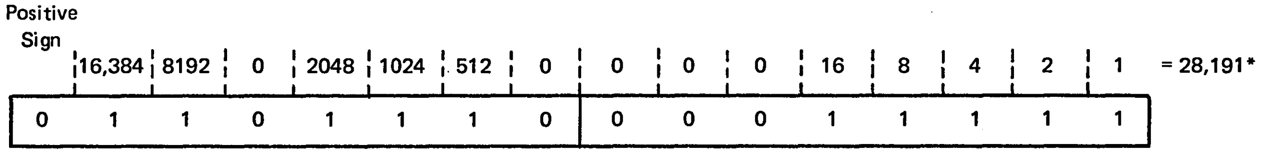


Figure 16-23. Four-byte Field in Binary Format

In each case, the sign portion of the high-order byte (leftmost) is used to indicate whether the numeric value is positive or negative. Notice that in the binary format the zone portion of the decimal number is not given. Compare how the decimal number 28,191 is represented in binary format (Figure 16-24) with its packed and unpacked representation (Figure 16-19 and 16-21).

Since data must be represented in unpacked decimal format when it is inside the computer, you must indicate to the RPG II program when fields are in another format. You can specify binary input, output, table, or array fields:

- Binary input fields. Enter a *B* in column 43 of the Input Sheet. The data is then converted into decimals before it is stored.
- Binary output fields. Enter a *B* in column 44 of the Output-Format Sheet. The data is then converted into binary before it is stored.
- Binary table or array fields. Enter a *B* in column 55 of the Extension Sheet. The data will be converted to decimals before it is stored. Binary tables or arrays are not allowed at compile time.



Positive Sign	16,384	8192	0	2048	1024	512	0	0	0	0	16	8	4	2	1	= 28,191*
0	1	1	0	1	1	1	0	0	0	0	1	1	1	1	1	

* The numeric value for each binary byte is obtained by adding the numbers which correspond to the bits that are on. (Bits that are on are represented as 1's.) The sign bit is not included in the value of the number.

Figure 16-24. Binary Format of Decimal Number 28,191

COLLATING SEQUENCE OF CHARACTERS

To perform jobs efficiently, you usually organize your information in some order or sequence. Imagine trying to locate a person's phone number if the names in a telephone book were not in alphabetical order. Of course, before using this order or sequence, you must know what it is. Through the learning process, you know that alphabetical order means that *A* comes before *B, B* before *C,* and so on. Likewise, in numerical sequence, *1* is less than *2, 2* less than *3,* and so on.

In most of your data processing jobs, the computer must be able to recognize an order or sequence of data. For example, if you instruct the computer to sequence check a file according to an alphabetic department code on each record, it must be able to determine if the department *T* record should appear before the department *X* record, or vice versa. In another instruction, perhaps the computer is to compare two quantities, such as *3* and *8,* and turn on an indicator if the first quantity is less than the second quantity. The computer must determine if *3* is less than *8* or if *8* is less than *3.*

The previous two tasks would be easy for you to perform because, through memorization or habit, you know the natural order of the alphabetic characters *A* through *Z* and the numbers *0* through *9.* However, a computer has not memorized such orders. For the computer to perform these tasks, an order or sequence of characters must be established.

To further point out the need for a set order of characters, assume the computer must check to make sure records in a file are in proper order according to a department field. Some department codes are alphabetic, as department *A;* while some department codes are numbers, such as department *8.* Should the numeric department records appear before the alphabetic department records, or vice versa? It is likely that the answer would depend on who is asked. However, for efficient data processing, you certainly do not want records sorted one way one time and another way the next time. Thus, the computer must use *one* set order of characters.

Every character recognized by the computer must hold a certain position in this order in relation to the position of the rest of the characters. Such an order is referred to as a *collating sequence* of characters. By definition, *to collate* means to arrange or verify that data appears in proper order or sequence.

There can be any number of collating sequences. The sequence used depends on the particular order in which characters are to be recognized. In any case, the computer should use only one collating sequence at a time.

The standard collating sequence of 64 characters is shown in Figure 16-25. The blank, which is the first character, is considered as the lowest in the sequence while the number *9,* the last character, is the highest in the sequence. Note that all of the special characters (except the }) are first in this sequence, followed by the alphabetic characters *A* through *Z* in their natural order, and then the numbers *0* through *9* (in their natural order). The only character which you might not expect to be in its position is the } which comes between the letters *I* and *J.*

This collating sequence is the order used by the computer for the purpose of sorting cards, comparing numbers to determine which is greater or less, checking the sequence of records in a file, and matching records from two files to determine which record should be processed next. According to the collating sequence, the computer compares two characters to determine if one comes before or after the other, or is less than or greater than the other. Of course, you specify which characters (or fields of characters) are to be compared.

1	Blank
2	c
3	.
4	<
5	(
6	+
7	|
8	&
9	!
10	$
11	*
12	)
13	;
14	¬
15	- (minus)
16	/
17	,
18	%
19	___(underscore)
20	>
21	?
22	:
23	#
24	@
25	'
26	=
27	"
28	A
29	B
30	C
31	D
32	E
33	F
34	G
35	H
36	I
37	}
38	J
39	K
40	L
41	M
42	N
43	O
44	P
45	Q
46	R
47	S
48	T
49	U
50	V
51	W
52	X
53	Y
54	Z
55	0
56	1
57	2
58	3
59	4
60	5
61	6
62	7
63	8
64	9

Figure 16-25. Standard Collating Sequence

For sorting, sequence checking, and matching, you can specify an ascending or descending collating sequence. For example, if records are to be in ascending sequence, the characters being checked should be in the order shown in Figure 16-25. That is, a card with a blank should come before a card with the letter *K.* (The blank characters is lower in sequence than the letter *K.*) Likewise, a card with the letter *K* should come before any records containing one of the numbers *0* through *9.* If you specify descending sequence, the computer compares to make sure they are in the opposite order, the characters higher in sequence coming first.

As mentioned, a computer cannot memorize the order of characters; it must use another method for *remembering* the collating sequence. To do this, it uses the values associated with characters to determine each character's relation to another character in the sequence.

In a previous discussion, a value is calculated for each bit combination in storage. The value can be thought of as a single numerical value for the entire 8-bit combination or as a 2-digit hexadecimal value, which is actually one hexadecimal number for each 4-bit combination (zone and digit). A hexadecimal value is another way of representing a numerical value.

Once a value is calculated, the computer uses it to determine which character is represented. Thus, the numerical value 193 (same as hexadecimal value C1) is associated with the character *A* while the numerical value 243 (hexadecimal value F3) is associated with the numeric character *3.* Perhaps you wonder why a particular value, such as 193 (C1) is related to the letter *A,* rather than a different value.

The values associated with the 64-characters were originally assigned such that the natural sequence of the values corresponds with the positions characters are to hold within the collating sequence. For example, the character *A* is associated with value 193 (hexadecimal C1), *B* with 194 (C2), *C* with 195 (C3), and so on. Just as *A* is lower than *B* and *B* is lower than *C* in the collating sequence, 193 (C1) is less than 194 (C2), and 194 (C2) is less than 195(C3).

Figure 16-26 shows the 256 possible bit combinations, their numerical and hexadecimal values, and the characters associated with each. In this list of bit combinations, the numerical values are in order from 0 through 255 (hexadecimal values 00 through FF), and the associated characters are in the standard ascending collating sequence.

As you can readily see, the value associated with a character does not always *immediately* follow the value associated with the previous character in the sequence. For example, the character *S* follows the character *R* in the collating sequence of characters. However, the numerical value of *S,* 226 (hexadecimal E2), does not immediately follow the numerical value of *R,* 217 (hexadecimal D9). The reason for the gap is because the bit combinations with the numerical values 218 through 225 are not associated with any of the 64-printable characters. Regardless, the computer determines that *R* is lower in sequence than (comes before) *S* because the value of *R* (217 or hexadecimal D9) is less than the value of *S* (226 or hexadecimal E2).

Bit Combination	Character	Hexadecimal Value	Numerical Value
00000000		00	0
00000001		01	1
00000010		02	2
00000011		03	3
00000100		04	4
00000101		05	5
00000110		06	6
00000111		07	7
00001000		08	8
00001001		09	9
00001010		0A	10
00001011		0B	11
00001100		0C	12
00001101		0D	13
00001110		0E	14
00001111		0F	15
00010000		10	16
00010001		11	17
00010010		12	18
00010011		13	19
00010100		14	20
00010101		15	21
00010110		16	22
00010111		17	23
00011000		18	24
00011001		19	25
00011010		1A	26
00011011		1B	27
00011100		1C	28
00011101		1D	29
00011110		1E	30
00011111		1F	31
00100000		20	32
00100001		21	33
00100010		22	34
00100011		23	35
00100100		24	36
00100101		25	37
00100110		26	38
00100111		27	39
00101000		28	40
00101001		29	41
00101010		2A	42
00101011		2B	43
00101100		2C	44
00101101		2D	45
00101110		2E	46
00101111		2F	47
00110000		30	48
00110001		31	49
00110010		32	50
00110011		33	51
00110100		34	52
00110101		35	53
00110110		36	54
00110111		37	55
00111000		38	56
00111001		39	57
00111010		3A	58
00111011		3B	59
00111100		3C	60
00111101		3D	61
00111110		3E	62
00111111		3F	63
01000000	Blank	40	64
01000001		41	65
01000010		42	66
01000011		43	67
01000100		44	68
01000101		45	69
01000110		46	70
01000111		47	71
01001000		48	72
01001001		49	73
01001010	c	4A	74
01001011	.	4B	75
0100110	<	4C	76
01001101	(	4D	77
01001110	+	4E	78
01001111	|	4F	79
01010000	&	50	80
01010001		51	81
01010010		52	82
01010011		53	83
01010100		54	84
01010101		55	85
01010110		56	86
01010111		57	87
01011000		58	88
01011001		59	89
01011010	!	5A	90
01011011	$	5B	91
01011100	*	5C	92
01011101	)	5D	93
01011110	;	5E	94
01011111	¬	5F	95
01100000	-	60	96
01100001	/	61	97
01100010		62	98
01100011		63	99
01100100		64	100
01100101		65	101

Figure 16-26. Characters and Values Associated with the 256 Bit Combinations (part 1 of 3)

Bit Combination	Character	Hexadecimal Value	Numerical Value
01100110		66	102
01100111		67	103
01101000		68	104
01101001		69	105
01101010		6A	106
01101011	,	6B	107
01101100	%	6C	108
01101101	–	6D	109
01101110	>	6E	110
01101111	?	6F	111
01110000		70	112
01110001		71	113
01110010		72	114
01110011		73	115
01110100		74	116
01110101		75	117
01110110		76	118
01110111		77	119
01111000		78	120
01111001		79	121
01111010	:	7A	122
01111011	#	7B	123
01111100	@	7C	124
01111101	'	7D	125
01111110	=	7E	126
01111111	"	7F	127
10000000		80	128
10000001		81	129
10000010		82	130
10000011		83	131
10000100		84	132
10000101		85	133
10000110		86	134
10000111		87	135
10001000		88	136
10001001		89	137
10001010		8A	138
10001011		8B	139
10001100		8C	140
10001101		8D	141
10001110		8E	142
10001111		8F	143
10010000		90	144
10010001		91	145
10010010		92	146
10010011		93	147
10010100		94	148
10010101		95	149
10010110		96	150
10010111		97	151
10011000		98	152
10011001		99	153
10011010		9A	154
10011011		9B	155
10011100		9C	156
10011101		9D	157
10011110		9E	158
10011111		9F	159
10100000		A0	160
10100001		A1	161
10100010		A2	162
10100011		A3	163
10100100		A4	164
10100101		A5	165
10100110		A6	166
10100111		A7	167
10101000		A8	168
10101001		A9	169
10101010		AA	170
10101011		AB	171
10101100		AC	172
10101101		AD	173
10101110		AE	174
10101111		AF	175
10110000		B0	176
10110001		B1	177
10110010		B2	178
10110011		B3	179
10110100		B4	180
10110101		B5	181
10110110		B6	182
10110111		B7	183
10111000		B8	184
10111001		B9	185
10111010		BA	186
10111011		BB	187
10111100		BC	188
10111101		BD	189
10111110		BE	190
10111111		BF	191
11000000		C0	192
11000001	A	C1	193
11000010	B	C2	194
11000011	C	C3	195
11000100	D	C4	196
11000101	E	C5	197
11000110	F	C6	198
11000111	G	C7	199
11001000	H	C8	100
11001001	I	C9	101
11001010		CA	202
11001011		CB	203

Figure 16-26. Characters and Values Associated with the 256 Bit Combinations (part 2 of 3)

Bit Combination	Character	Hexadecimal Value	Numerical Value
11001100		CC	204
11001101		CD	205
11001110		CE	206
11001111		CF	207
11010000		D0	208
11010001	J	D1	209
11010010	K	D2	210
11010011	L	D3	211
11010100	M	D4	212
11010101	N	D5	213
11010110	O	D6	214
11010111	P	D7	215
11011000	Q	D8	216
11011001	R	D9	217
11011010		DA	218
11011011		DB	219
11011100		DC	220
11011101		DD	221
11011110		DE	222
11011111		DF	223
11100000		E0	224
11100001		E1	225
11100010	S	E2	226
11100011	T	E3	227
11100100	U	E4	228
11100101	V	E5	229
11100110	W	E6	230
11100111	X	E7	231
11101000	Y	E8	232
11101001	Z	E9	233
11101010		EA	234
11101011		EB	235
11101100		EC	236
11101101		ED	237
11101110		EE	238
11101111		EF	239
11110000	0	F0	240
11110001	1	F1	241
11110010	2	F2	242
11110011	3	F3	243
11110100	4	F4	244
11110101	5	F5	245
11110110	6	F6	246
11110111	7	F7	247
11111000	8	F8	248
11111001	9	F9	249
11111010		FA	250
11111011		FB	251
11111100		FC	252
11111101		FD	253
11111110		FE	254
11111111		FF	255

Figure 16-26. Characters and Values Associated with the 256 Bit Combinations (part 3 of 3)

Collating By Zone Or Digit

You learned from a previous discussion that the zone and digit portions of characters can be treated as separate and distinct groups of four bits, each with its own hexadecimal number.

Different characters may have identical zone bits or identical digit bits, but not both. Consequently, different characters may be associated with the same zone hexadecimal number or the same digit hexadecimal number but not both. As an example, the character *A* is associated with the value C1, *B* is associated with C2, and *K* is associated with D2. *A* has the same zone value (C) as *B,* while *K* has the same digit value (2) as *B.* However, *B* is the only character with both a zone value of *C* and a digit value of *2.*

In most data processing tasks, the computer uses entire characters or the values of those characters to make comparisons, to determine which is greater or less, and so on. However, for certain purposes, such as sorting cards, you may wish to have the computer check only the zone or only the digit portion of characters. In such a case, the computer must use a collating sequence based on zone or digit values rather than the standard collating sequence based on the entire value.

If the computer uses a collating sequence based on the zone portions of characters, any differences in the digit bits are ignored. Only the value of the zone bits are considered. The reverse occurs if a collating sequence based on the digit portions of characters is to be used.

The fact that certain characters have the same zone or digit values can be used to group characters within a collating sequence. On the basis of zone values, the 64 printable characters are divided into eight groups (Figure 16-27). The zone bits (and values) are identical for all characters within a group. If collating is to be on the basis of digit values, the characters can be divided into 16 groups (Figure 16-28). In such a case, digit bits (and values) are identical for all characters within a particular group.

Using the standard collating sequence, the computer considers each character to hold a specific position in the sequence. Therefore, no two characters can be considered equal; one must come before another or be less than another character.

On the other hand, using a collating sequence based on zones or digits, one *group* or characters follows another *group* of characters. The characters within a group can occupy any position within that group. Thus, there is an order of groups but no particular order of characters within a group.

Note that in the collating sequence by zone shown in Figure 16-27, any character in group *5* is considered lower in sequence than any character in group *6.* If records are sorted in ascending order by zone, a record with the letter D (group 5) comes before a record with the letter N (group 6).

Now consider a case in which characters from the same zone group are to be compared. Assume one record contains the letter *D* (group 5) and the next record contains the letter *F* (group 5). Which should be sorted first according to a collating sequence based on zones? Since the computer ignores the digit bits of each character, they are considered equal because they both have the same zone value. Therefore, no one character must come earlier in the sequence than another character *from the same group.* The sequence of the characters is the same order in which the records are read. Thus, if the *D* card is read first in a sort job, the *D* record comes before the *F* record. On the other hand, if the *F* card is read first, the *F* record comes before the *D* record. In either case, the records are in proper sequence based on zones.

Character	Bit Combination		Collating Sequence of Zones
	Zone	Digit	
b (blank)	0100	0000	1
¢	0100	1010	
. (period)	0100	1011	
<	0100	1100	
(	0100	1101	
+	0100	1110	
|	0100	1111	
&	0101	0000	2
!	0101	1010	
$	0101	1011	
*	0101	1100	
)	0101	1101	
;	0101	1110	
¬	0101	1111	
- (minus)	0110	0000	3
/	0110	0001	
,	0110	1011	
%	0110	1100	
_ (underscore)	1101	1101	
>	0110	1110	
?	0110	1111	
:	0111	1010	4
#	0111	1011	
@	0111	1100	
' (apostrophe)	0111	1101	
=	0111	1110	
"	0111	1111	

Character	Bit Combination		Collating Sequence of Zones
	Zone	Digit	
A	1100	0001	5
B	1100	0010	
C	1100	0011	
D	1100	0100	
E	1100	0101	
F	1100	0110	
G	1100	0111	
H	1100	1000	
I	1100	1001	
}	1101	0000	6
J or -1	1101	0001	
K or -2	1101	0010	
L or -3	1101	0011	
M or -4	1101	0100	
N or -5	1101	0101	
O or -6	1101	0110	
P or -7	1101	0111	
Q or -8	1101	1000	
R or -9	1101	1001	
S	1110	0010	7
T	1110	0011	
U	1110	0100	
V	1110	0101	
W	1110	0110	
X	1110	0111	
Y	1110	1000	
Z	1110	1001	
+0	1111	0000	8
1	1111	0001	
2	1111	0010	
3	1111	0011	
4	1111	0100	
5	1111	0101	
6	1111	0110	
7	1111	0111	
8	1111	1000	
9	1111	1001	

Figure 16-27. Collating Sequence by Zone

Character	Bit Combination		Collating Sequence of Digits
	Zone	Digit	
b̸ (blank)	0100	0000	1
&	0101	0000	
- (minus)	0110	0000	
}	0111	0000	
+0	1111	0000	
/	0110	0001	2
A	1100	0001	
J or -1	1101	0001	
1	1111	0001	
B	1100	0010	3
K or -2	1101	0010	
S	1110	0010	
2	1111	0010	
C	1100	0011	4
L or -3	1101	0011	
T	1110	0011	
3	1111	0011	
D	1100	0100	5
M or -4	1101	0100	
U	1110	0100	
4	1111	0100	
E	1100	0101	6
N or -5	1101	0101	
V	1110	0101	
5	1111	0101	
F	1100	0110	7
O or -6	1101	0110	
W	1110	0110	
6	1111	0110	
G	1100	0111	8
P or -7	1101	0111	
X	1110	0111	
7	1111	0111	
H	1100	1000	9
Q or -8	1101	1000	
Y	1110	1000	
8	1111	1000	
I	1100	1001	10
R or-9	1101	1001	
Z	1110	1001	
9	1111	1001	
¢	0100	1010	11
!	0101	1010	
:	0111	1010	
. (period)	0100	1011	12
$	0101	1011	
,	0110	1011	
#	0111	1011	
<	0100	1100	13
*	0101	1100	
%	0110	1100	
@	0111	1100	
(	0100	1101	14
)	0101	1101	
— (underscore)	0110	1101	
' (apostrophe)	0111	1101	
+	0100	1110	15
;	0101	1110	
>	0110	1110	
=	0111	1110	
|	0100	1111	16
¬	0101	1111	
?	0110	1111	
"	0111	1111	

Figure 16-28. Collating Sequence by Digit

ALTERING THE COLLATING SEQUENCE

A collating sequence is the order in which characters are arranged. As you know, all characters are associated with different numerical values in order that the computer may recognize them. The sequence of numerical values (ascending or descending sequence) determines the order in which characters associated with the values are recognized.

The association of a particular character with a numerical value is an arbitrary decision. Thus, the collating sequence itself is arbitrary. System/3 is programmed to expect the collating sequence discussed previously in the section *Collating Sequence Of Characters.* This does not mean, however, that you must always use this sequence. You can change it and there may be times when you desire to do so.

For example, you may want alphabetic characters to follow the numbers instead of preceding them. Suppose that a company originally started with a few departments. The departments were assigned numbers from 01-99. Two columns were devoted to department numbers in various records. The company expanded and departments increased. Soon there were more than 99 departments. To avoid having to change the department field from two to three characters in all records, the manager decided to use the letters of the alphabet to represent department numbers: 99, A0, A1, etc. In this case, *A* must follow the number *9* in the sequence. Thus it is necessary to alter the collating sequence so that numbers come before alphabetic characters.

There can be other reasons than the one just explained for altering the collating sequence. Suppose you wish to have the computer consider two characters equal (having the same place in the sequence). For example, it is often desirable to have the computer consider a blank and zero equal. This eliminates having to punch leading zeros in numeric fields since blanks will be the same as zeros in the new sequence. Or your language may demand that you have characters such as *Ä, A', Ö, Ë, E'* included in the alphabetic sequence (*A, Ä, B*). Since the 64 graphics do not include these characters, other seldom used characters can be substituted for them and repositioned in the collating sequence. For example, a number-symbol (#) repositioned between the letters *A* and *B* can substitute for an *Ä,* an at-symbol (@) repositioned between *O* and *P* substitutes for an *Ö.*

These are only a few reasons for altering the collating sequence. You may have others. Just remember that you can alter the collating sequence in any way that fits your needs.

Specifying Changes in Collating Sequence

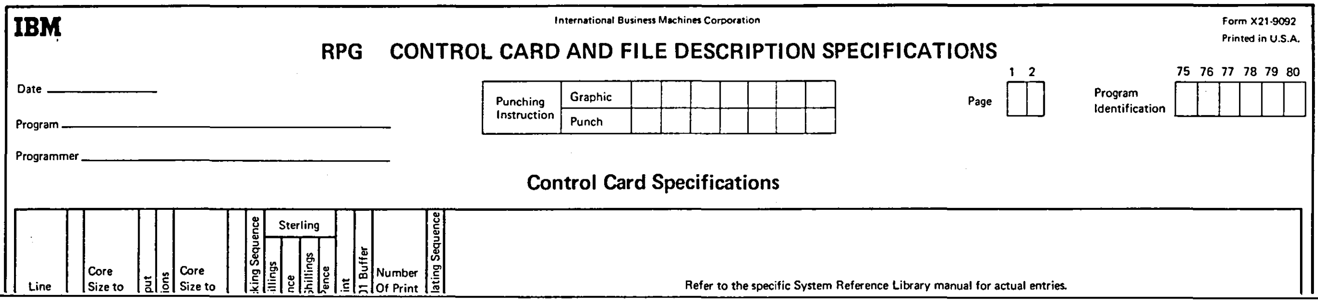
To change the collating sequence, you must associate characters with different numerical values. The following sections will explain how this is done.

Forms for Altering the Collating Sequence

Figure 16-29 illustrates two forms on which you must specify changes to the collating sequence. One form is the RPG II Control Card and File Description Sheet; the other is the Translation Table and Alternate Collating Sequence Coding Sheet which is used for listing the actual changes in sequence. Both forms are used in conjunction with the RPG II Input, Output and Calculation Sheets.

A letter *S* entered in column 26 of the RPG II control card notifies the computer that additional information will be furnished as part of the job so that the collating sequence can be altered. All other columns contain the information that must normally be entered to process job.

The Alternate Collating Sequence Coding Sheet lists 256 bit combinations along with their hexadecimal numerical values. As you learned from discussions of character structure, hexadecimal values are written in the form of two character values. One value represents the numerical value of the character's zone; the other represents the numerical value of the character's digit. The 64 printable graphics are listed beside the bit combinations and numerical values with which they are associated.



IBM

International Business Machines Corporation

Form X21-9092
Printed in U.S.A.

RPG CONTROL CARD AND FILE DESCRIPTION SPECIFICATIONS

Date ____________

Program ______________________________

Programmer ______________________________

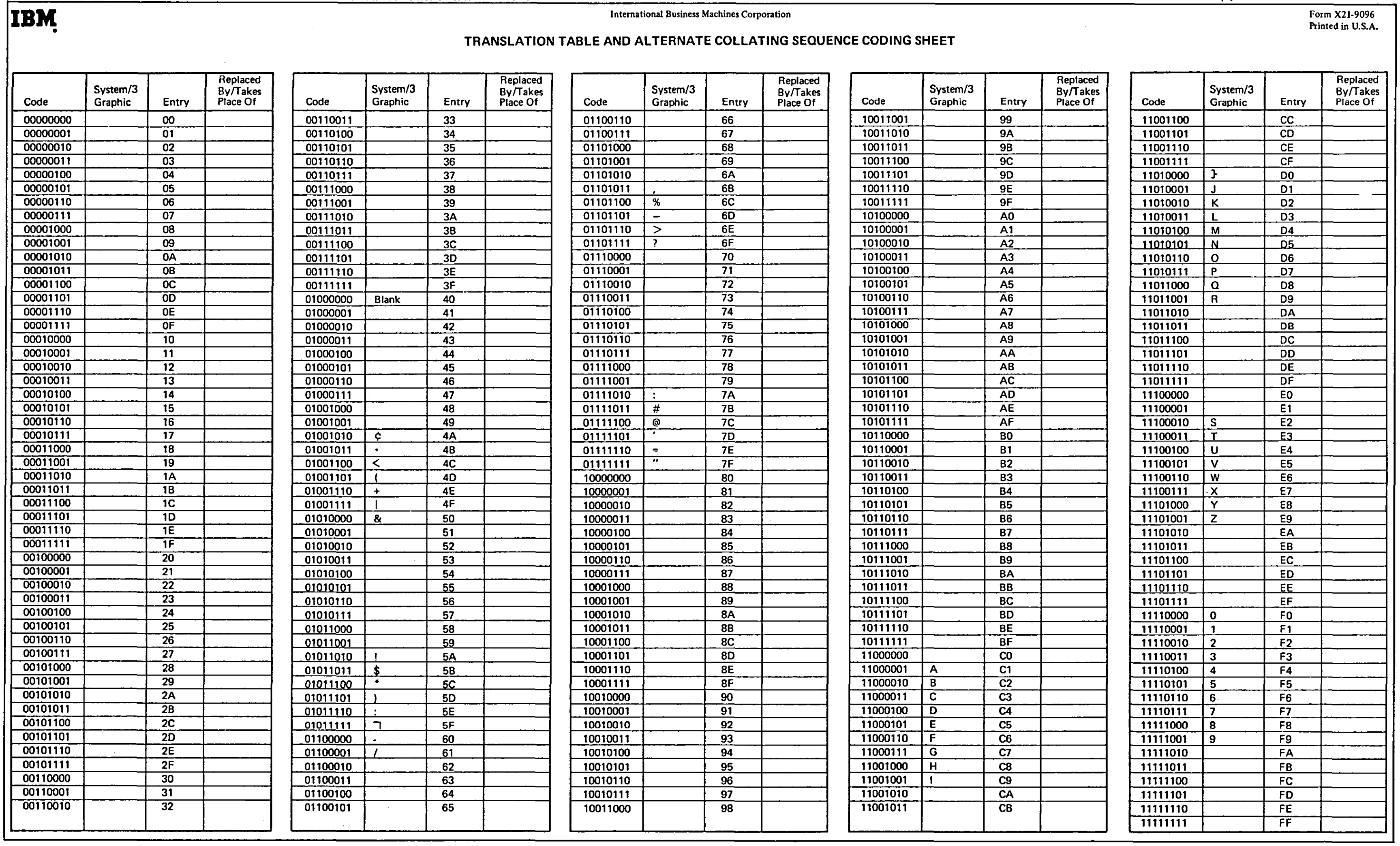
Punching Instruction	Graphic								
	Punch								

Page 1 2 □□

Program Identification 75 76 77 78 79 80 □□□□□□

Control Card Specifications

Line		Core Size to	put	ions	Core Size to	king Sequence	Sterling illings	nce	hillings	ence	int	1 Buffer	Number Of Print	lating Sequence	Refer to the specific System Reference Library manual for actual entries.



IBM

International Business Machines Corporation

Form X21-9096
Printed in U.S.A.

TRANSLATION TABLE AND ALTERNATE COLLATING SEQUENCE CODING SHEET

Code	System/3 Graphic	Entry	Replaced By/Takes Place Of
00000000		00	
00000001		01	
00000010		02	
00000011		03	
00000100		04	
00000101		05	
00000110		06	
00000111		07	
00001000		08	
00001001		09	
00001010		0A	
00001011		0B	
00001100		0C	
00001101		0D	
00001110		0E	
00001111		0F	
00010000		10	
00010001		11	
00010010		12	
00010011		13	
00010100		14	
00010101		15	
00010110		16	
00010111		17	
00011000		18	
00011001		19	
00011010		1A	
00011011		1B	
00011100		1C	
00011101		1D	
00011110		1E	
00011111		1F	
00100000		20	
00100001		21	
00100010		22	
00100011		23	
00100100		24	
00100101		25	
00100110		26	
00100111		27	
00101000		28	
00101001		29	
00101010		2A	
00101011		2B	
00101100		2C	
00101101		2D	
00101110		2E	
00101111		2F	
00110000		30	
00110001		31	
00110010		32	
00110011		33	
00110100		34	
00110101		35	
00110110		36	
00110111		37	
00111000		38	
00111001		39	
00111010		3A	
00111011		3B	
00111100		3C	
00111101		3D	
00111110		3E	
00111111		3F	
01000000	Blank	40	
01000001		41	
01000010		42	
01000011		43	
01000100		44	
01000101		45	
01000110		46	
01000111		47	
01001000		48	
01001001		49	
01001010	¢	4A	
01001011	.	4B	
01001100	<	4C	
01001101	(	4D	
01001110	+	4E	
01001111	|	4F	
01010000	&	50	
01010001		51	
01010010		52	
01010011		53	
01010100		54	
01010101		55	
01010110		56	
01010111		57	
01011000		58	
01011001		59	
01011010	!	5A	
01011011	$	5B	
01011100	*	5C	
01011101	)	5D	
01011110	:	5E	
01011111	¬	5F	
01100000	-	60	
01100001	/	61	
01100010		62	
01100011		63	
01100100		64	
01100101		65	
01100110		66	
01100111		67	
01101000		68	
01101001		69	
01101010		6A	
01101011	,	6B	
01101100	%	6C	
01101101	–	6D	
01101110	>	6E	
01101111	?	6F	
01110000		70	
01110001		71	
01110010		72	
01110011		73	
01110100		74	
01110101		75	
01110110		76	
01110111		77	
01111000		78	
01111001		79	
01111010	:	7A	
01111011	#	7B	
01111100	@	7C	
01111101	'	7D	
01111110	=	7E	
01111111	"	7F	
10000000		80	
10000001		81	
10000010		82	
10000011		83	
10000100		84	
10000101		85	
10000110		86	
10000111		87	
10001000		88	
10001001		89	
10001010		8A	
10001011		8B	
10001100		8C	
10001101		8D	
10001110		8E	
10001111		8F	
10010000		90	
10010001		91	
10010010		92	
10010011		93	
10010100		94	
10010101		95	
10010110		96	
10010111		97	
10011000		98	
10011001		99	
10011010		9A	
10011011		9B	
10011100		9C	
10011101		9D	
10011110		9E	
10011111		9F	
10100000		A0	
10100001		A1	
10100010		A2	
10100011		A3	
10100100		A4	
10100101		A5	
10100110		A6	
10100111		A7	
10101000		A8	
10101001		A9	
10101010		AA	
10101011		AB	
10101100		AC	
10101101		AD	
10101110		AE	
10101111		AF	
10110000		B0	
10110001		B1	
10110010		B2	
10110011		B3	
10110100		B4	
10110101		B5	
10110110		B6	
10110111		B7	
10111000		B8	
10111001		B9	
10111010		BA	
10111011		BB	
10111100		BC	
10111101		BD	
10111110		BE	
10111111		BF	
11000000		C0	
11000001	A	C1	
11000010	B	C2	
11000011	C	C3	
11000100	D	C4	
11000101	E	C5	
11000110	F	C6	
11000111	G	C7	
11001000	H	C8	
11001001	I	C9	
11001010		CA	
11001011		CB	
11001100		CC	
11001101		CD	
11001110		CE	
11001111		CF	
11010000	}	D0	
11010001	J	D1	
11010010	K	D2	
11010011	L	D3	
11010100	M	D4	
11010101	N	D5	
11010110	O	D6	
11010111	P	D7	
11011000	Q	D8	
11011001	R	D9	
11011010		DA	
11011011		DB	
11011100		DC	
11011101		DD	
11011110		DE	
11011111		DF	
11100000		E0	
11100001		E1	
11100010	S	E2	
11100011	T	E3	
11100100	U	E4	
11100101	V	E5	
11100110	W	E6	
11100111	X	E7	
11101000	Y	E8	
11101001	Z	E9	
11101010		EA	
11101011		EB	
11101100		EC	
11101101		ED	
11101110		EE	
11101111		EF	
11110000	0	F0	
11110001	1	F1	
11110010	2	F2	
11110011	3	F3	
11110100	4	F4	
11110101	5	F5	
11110110	6	F6	
11110111	7	F7	
11111000	8	F8	
11111001	9	F9	
11111010		FA	
11111011		FB	
11111100		FC	
11111101		FD	
11111110		FE	
11111111		FF	

Figure 16-29. Forms Needed for Alternate Collating Sequence Specifications

Coding a Change in Sequence

Each change in the collating sequence is specified in the *Replaced By* column on the coding sheet. In this column, place the hexadecimal value of the graphic whose position in the normal sequence is to be changed. The *character* corresponding to the hexadecimal value entered in the *Replaced By* column replaces the character which is presently associated with the bit combination shown on the same line.

Figure 16-30 illustrates entries made to change the normal collating sequence. Hexadecimal values entered on the second and third lines of the sample coding sheet reverse the order in which the numbers *1* and *2* are recognized by the computer.

Numerical values entered on the second line of the sample specify that the number *2* (hexadecimal value F2) replaces the number *1*. In other words, in the new sequence the number *2* is associated with the value F1 instead of the number *1*. Hexadecimal values on the third line specify that the number *1* (hexadecimal value F1) replaces the number *2*. These two specification lines cause *2* to come before *1* in the collating sequence (*0, 2, 1, 3*).

Effect of the Coded Change in Sequence

Any alternate collating sequence you specify is used temporarily. It is used only for the program which contains the alternate collating sequence specifications. Even more specifically, it is used in that program for operations which involve sequencing, such as checking sequence of records, comparing fields, or matching records.

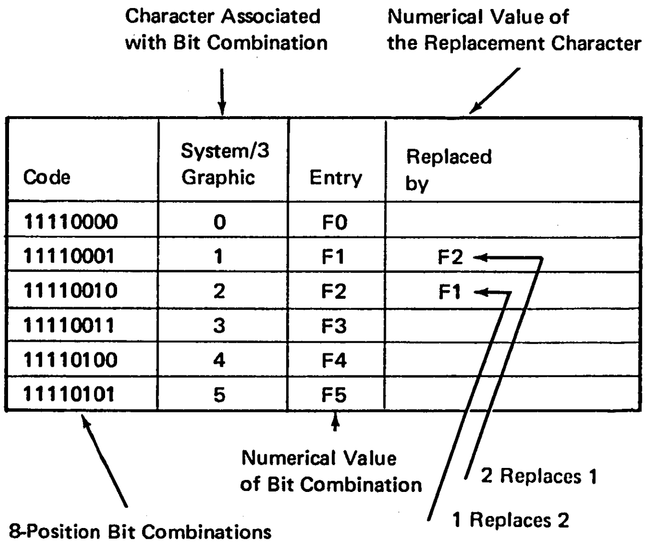
You may think, according to specifications in Figure 16-30, that the character 2 read into the computer is *always* replaced by a 1. This is not true. The computer associates characters with the values you specify only before sequencing operations involving:

1. Compare operations on alphameric fields.

2. Matching or sequence checking match fields.

How does the computer keep track of the collating sequence to use? The computer keeps all your instructions for altering the sequence in storage. The area in storage which holds this information may be pictured as shown in Figure 16-31. These instructions combined with the pattern for normal sequence give the computer the correct collating sequence to use.

Consider the use of an altered sequence when determining which record to select for processing in a multi-file job. The collating sequence has been changed so that *2* comes before *1*.



Code	System/3 Graphic	Entry	Replaced by
11110000	0	F0	
11110001	1	F1	F2
11110010	2	F2	F1
11110011	3	F3	
11110100	4	F4	
11110101	5	F5	

Character Associated with Bit Combination
Numerical Value of the Replacement Character
Numerical Value of Bit Combination
8-Position Bit Combinations
2 Replaces 1
1 Replaces 2

Figure 16-30. Explanation of Alternate Collating Sequence Sheet

Numerical Value of Bit Combinations	Associated Characters	
	Normal Collating Sequence	Altered Collating Sequence
F0	0	0
F1	1	2
F2	2	1
F3	3	3
F4	4	4
F5	5	5
F6	6	6
F7	7	7

Figure 16-31. Storage Area Holding Alternate Collating Sequence Instruction

Figure 16-32 illustrates how the computer uses the alternate sequence. Two cards are read into the read area. Just before the compare operation which is done to determine which match field has a lower value, the computer checks to see if the characters used in the compare are affected by the alternate collating sequence instructions. They are. The character *1* normally associated with the value F1 is replaced by the character *2;* the character *2* normally associated with the value F2 is replaced by the character *1.*

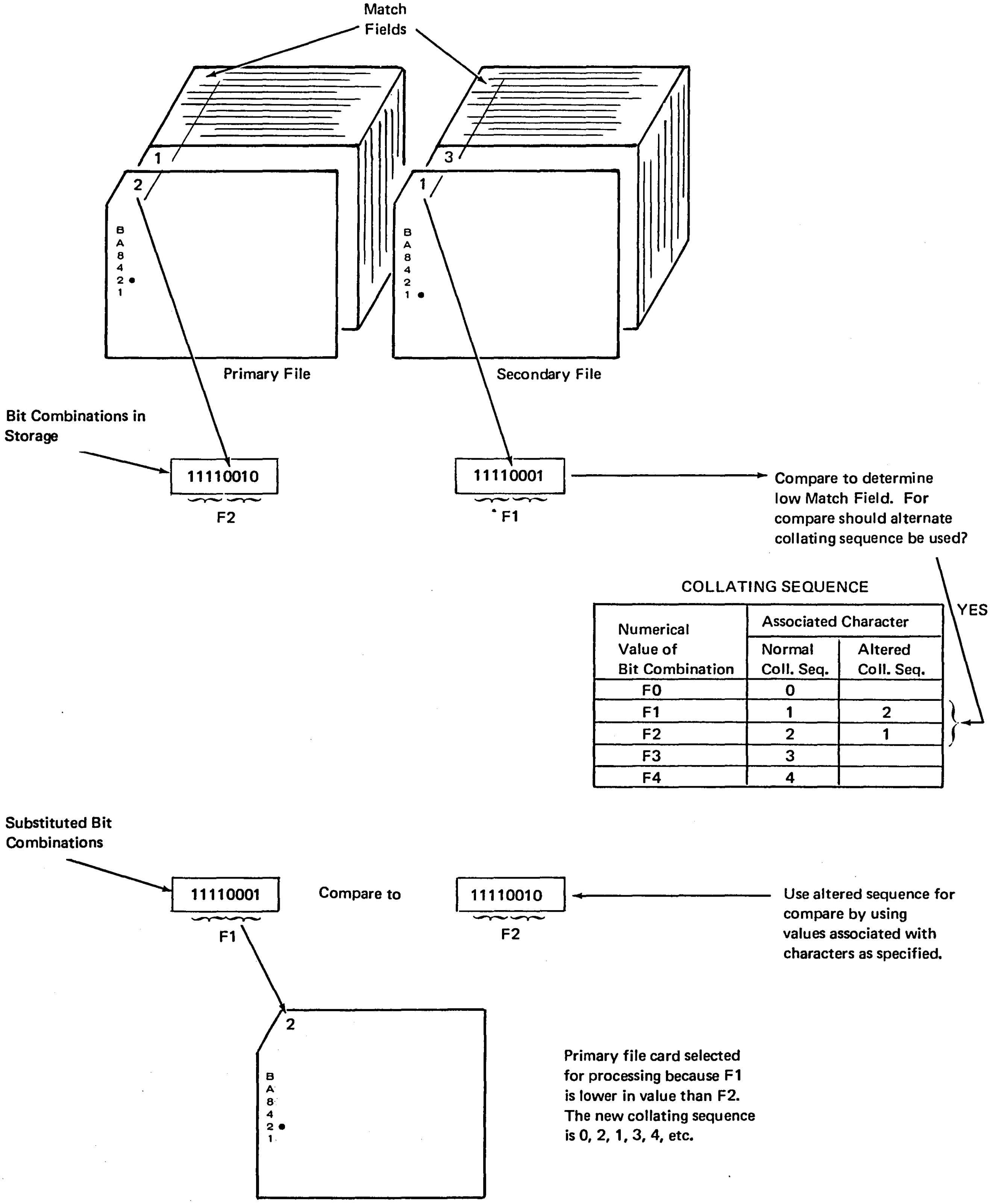
F1 and 1 → F1 and 2

F2 and 2 → F2 and 1

F3 and 3 → F3 and 3

When doing the compare, the, the computer substitutes these values. For the match field having the character *2,* the computer uses the bit combination whose value is F1 instead of the bit combination for F2. Similarly for the match field containing a *1,* the computer uses the bit combination of F2 instead of the bit combination for F1. As a result of the compare, the primary card containing a 2 in the match field is chosen for processing. This card was chosen because F1 (now associated with character *2*) is lower in sequence than F2 (now associated with the character *1*).

After the compare, characters are again associated with values as assigned in the normal collating sequence.



Compare to determine low Match Field. For compare should alternate collating sequence be used?

COLLATING SEQUENCE

Numerical Value of Bit Combination	Associated Character	
	Normal Coll. Seq.	Altered Coll. Seq.
F0	0	
F1	1	2
F2	2	1
F3	3	
F4	4	

Use altered sequence for compare by using values associated with characters as specified.

Primary file card selected for processing because F1 is lower in value than F2. The new collating sequence is 0, 2, 1, 3, 4, etc.

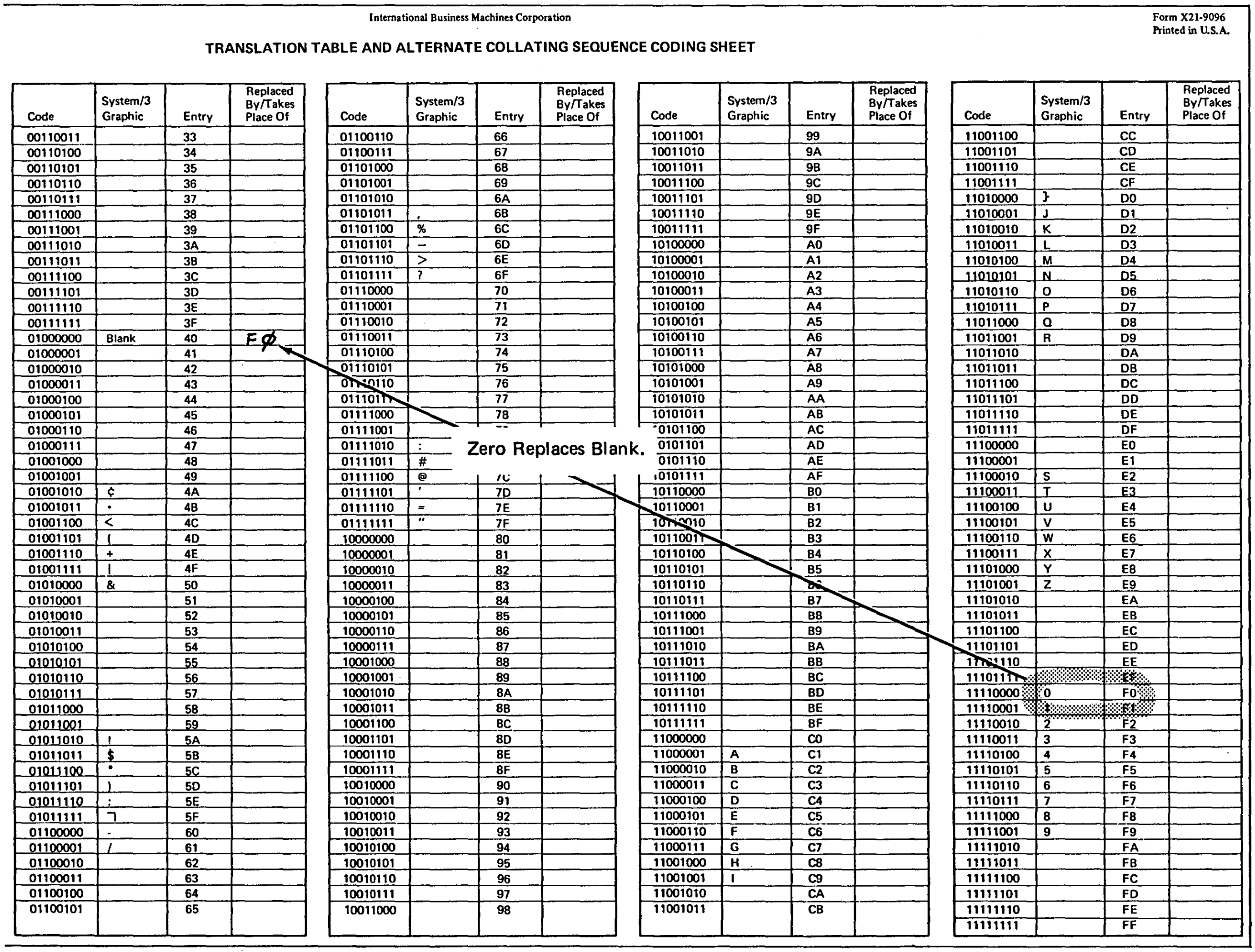
Figure 16-32. Using Alternate Collating Sequence (0, 2, 1, 3, 4, 9)

Coding Characters to be Equal

Entries can be made to allow two characters to occupy the same position in the collating sequence; that is, they are associated with the same numeric value. When two characters occupy the same position in the sequence, the computer recognizes one character as being the same as the other.

Figure 16-33 illustrates the specifications which allow a blank and zero to occupy the same position in an altered sequence. The hexadecimal value associated with the character blank is replaced by the hexadecimal value (F0) which is already associated with the zero. Because the zero and blank are associated with the same numerical value, they are recognized as the same character. Figure 16-34 shows why a field containing a blank is equal to a field containing a zero when the altered sequence is used.

If a blank is equal to zero, then ƀƀ43 is the same as 0043. You can see from this example that altering the sequence in this way saves time because leading zeros do not have to be recorded on the card. Blanks can be left instead since blanks are considered to be the same as zeros.



International Business Machines Corporation

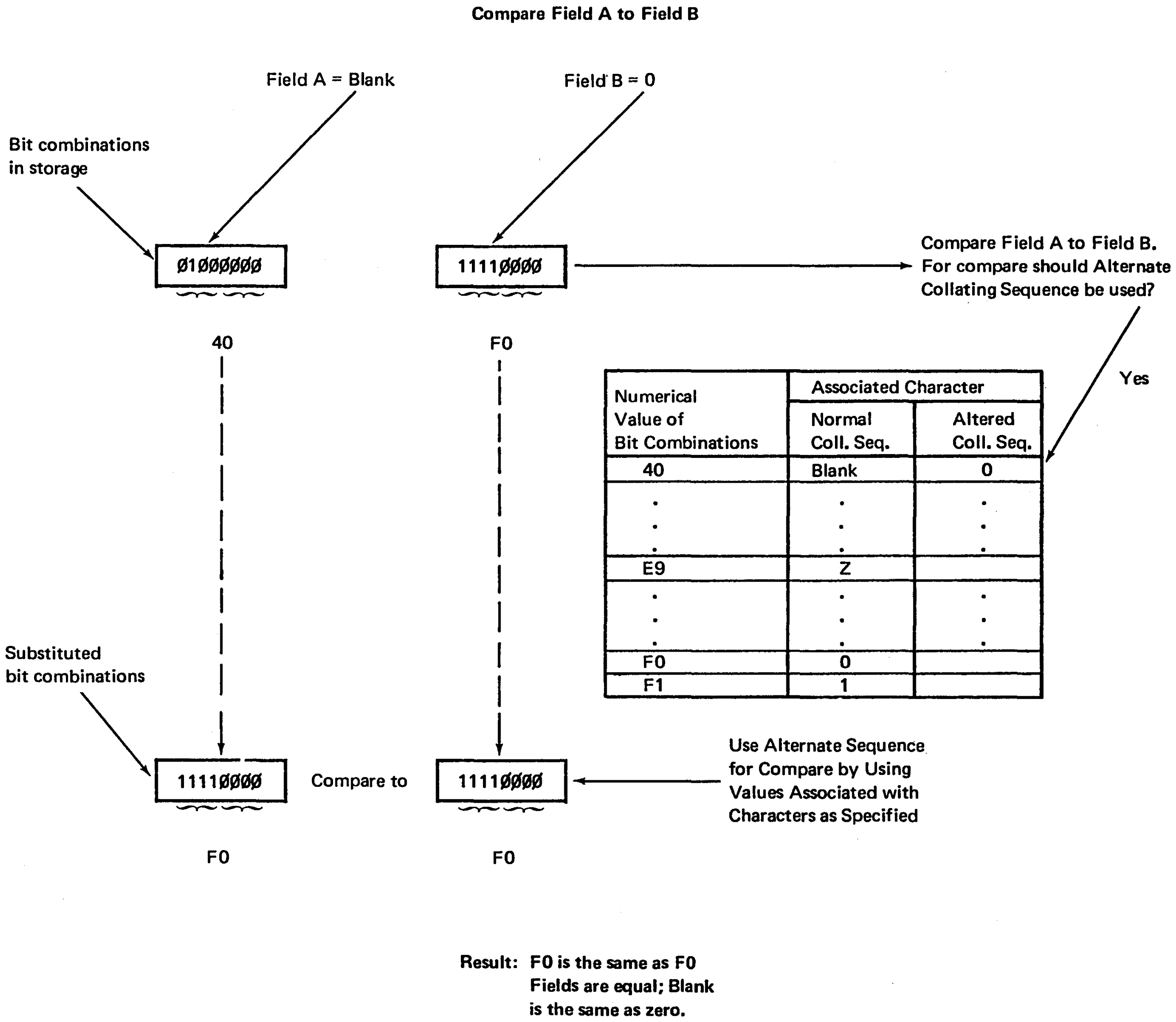
Form X21-9096
Printed in U.S.A.

TRANSLATION TABLE AND ALTERNATE COLLATING SEQUENCE CODING SHEET

Code	System/3 Graphic	Entry	Replaced By/Takes Place Of	Code	System/3 Graphic	Entry	Replaced By/Takes Place Of	Code	System/3 Graphic	Entry	Replaced By/Takes Place Of	Code	System/3 Graphic	Entry	Replaced By/Takes Place Of
00110011		33		01100110		66		10011001		99		11001100		CC	
00110100		34		01100111		67		10011010		9A		11001101		CD	
00110101		35		01101000		68		10011011		9B		11001110		CE	
00110110		36		01101001		69		10011100		9C		11001111		CF	
00110111		37		01101010		6A		10011101		9D		11010000	}	D0	
00111000		38		01101011	,	6B		10011110		9E		11010001	J	D1	
00111001		39		01101100	%	6C		10011111		9F		11010010	K	D2	
00111010		3A		01101101	–	6D		10100000		A0		11010011	L	D3	
00111011		3B		01101110	>	6E		10100001		A1		11010100	M	D4	
00111100		3C		01101111	?	6F		10100010		A2		11010101	N	D5	
00111101		3D		01110000		70		10100011		A3		11010110	O	D6	
00111110		3E		01110001		71		10100100		A4		11010111	P	D7	
00111111		3F		01110010		72		10100101		A5		11011000	Q	D8	
01000000	Blank	40	FØ	01110011		73		10100110		A6		11011001	R	D9	
01000001		41		01110100		74		10100111		A7		11011010		DA	
01000010		42		01110101		75		10101000		A8		11011011		DB	
01000011		43		01110110		76		10101001		A9		11011100		DC	
01000100		44		01110111		77		10101010		AA		11011101		DD	
01000101		45		01111000		78		10101011		AB		11011110		DE	
01000110		46		01111001		[illegible]		10101100		AC		11011111		DF	
01000111		47		01111010	:	[illegible]		10101101		AD		11100000		E0	
01001000		48		01111011	#	[illegible]		10101110		AE		11100001		E1	
01001001		49		01111100	@	7C		10101111		AF		11100010	S	E2	
01001010	¢	4A		01111101	'	7D		10110000		B0		11100011	T	E3	
01001011	.	4B		01111110	=	7E		10110001		B1		11100100	U	E4	
01001100	<	4C		01111111	"	7F		10110010		B2		11100101	V	E5	
01001101	(	4D		10000000		80		10110011		B3		11100110	W	E6	
01001110	+	4E		10000001		81		10110100		B4		11100111	X	E7	
01001111	|	4F		10000010		82		10110101		B5		11101000	Y	E8	
01010000	&	50		10000011		83		10110110		B6		11101001	Z	E9	
01010001		51		10000100		84		10110111		B7		11101010		EA	
01010010		52		10000101		85		10111000		B8		11101011		EB	
01010011		53		10000110		86		10111001		B9		11101100		EC	
01010100		54		10000111		87		10111010		BA		11101101		ED	
01010101		55		10001000		88		10111011		BB		11101110		EE	
01010110		56		10001001		89		10111100		BC		11101111		EF	
01010111		57		10001010		8A		10111101		BD		11110000	0	F0	
01011000		58		10001011		8B		10111110		BE		11110001	1	F1	
01011001		59		10001100		8C		10111111		BF		11110010	2	F2	
01011010	!	5A		10001101		8D		11000000		C0		11110011	3	F3	
01011011	$	5B		10001110		8E		11000001	A	C1		11110100	4	F4	
01011100	*	5C		10001111		8F		11000010	B	C2		11110101	5	F5	
01011101	)	5D		10010000		90		11000011	C	C3		11110110	6	F6	
01011110	;	5E		10010001		91		11000100	D	C4		11110111	7	F7	
01011111	¬	5F		10010010		92		11000101	E	C5		11111000	8	F8	
01100000	-	60		10010011		93		11000110	F	C6		11111001	9	F9	
01100001	/	61		10010100		94		11000111	G	C7		11111010		FA	
01100010		62		10010101		95		11001000	H	C8		11111011		FB	
01100011		63		10010110		96		11001001	I	C9		11111100		FC	
01100100		64		10010111		97		11001010		CA		11111101		FD	
01100101		65		10011000		98		11001011		CB		11111110		FE	
												11111111		FF	

Zero Replaces Blank.

Figure 16-33. Specifying Blank Equal to Zero in New Collating Sequence


Compare Field A to Field B
Field A = Blank
Field B = 0
Bit combinations in storage
Ø1ØØØØØØ
1111ØØØØ
Compare Field A to Field B. For compare should Alternate Collating Sequence be used?
Yes
40
F0
Substituted bit combinations
1111ØØØØ
Compare to
1111ØØØØ
Use Alternate Sequence for Compare by Using Values Associated with Characters as Specified
F0
F0

Numerical Value of Bit Combinations	Associated Character	
	Normal Coll. Seq.	Altered Coll. Seq.
40	Blank	0
• • •	• • •	• • •
E9	Z	
• • •	• • •	• • •
F0	0	
F1	1	

Result: F0 is the same as F0
Fields are equal; Blank
is the same as zero.

Figure 16-34. Using Alternate Collating Sequence (Blank Equals Zero)

Example of the Coding of an Altered Sequence

Figure 16-35 shows a part of the normal collating sequence, and one of several ways in which the sequence can be changed. Arrows depict changes required in the positions of characters to alter the sequence as shown at the right side of the figure.

In like manner, arrows in Figure 16-36 show entries on the coding sheet which must be specified to alter the sequence. Note that letters *B* through *I* are repositioned to allow the at-symbol (@) to appear between letters *A* and *B*. Identical results could be achieved by repositioning the value for the letter *A* to the line above, making it correspond to bit combination 1100000.

To produce the sequence shown in Figures 16-35 and 16-36, the appropriate hexadecimal values must be specified in the *Replaced By* column beside each graphic involved in the change. Figure 16-37 shows the actual coding required to alter the sequence.

Notice that each number which is to be collated before the alphabetic character is assigned a hexadecimal value which has no graphic associated with it. These values have no associated graphics that could have been assigned to the values previously associated with numbers. This is not necessary, however, because these values have no associated graphics. When two graphics are involved in the change, then both must be assigned different values except when they are to be considered equal.

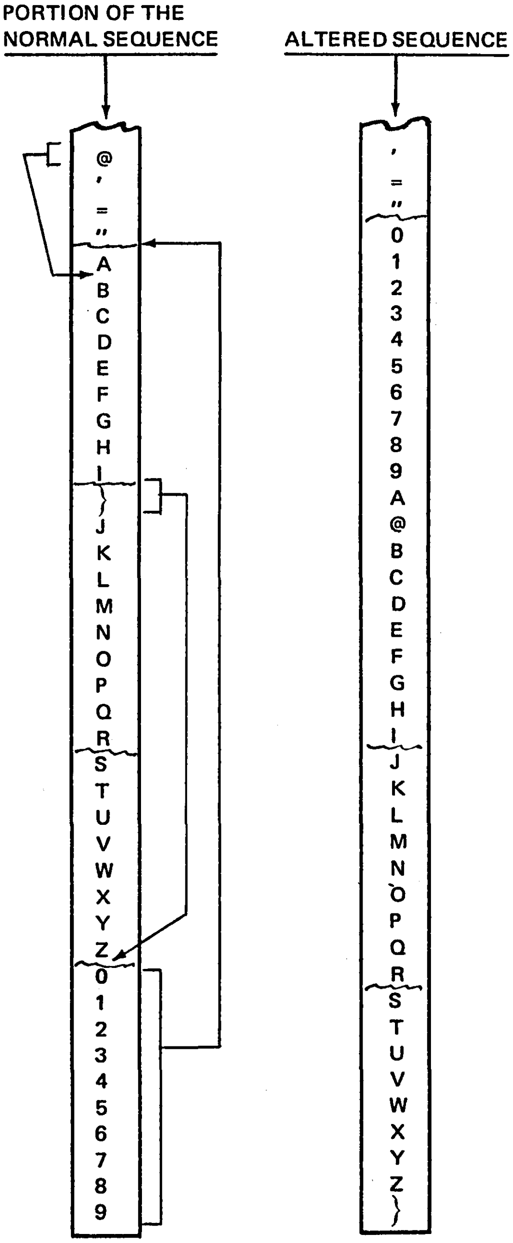
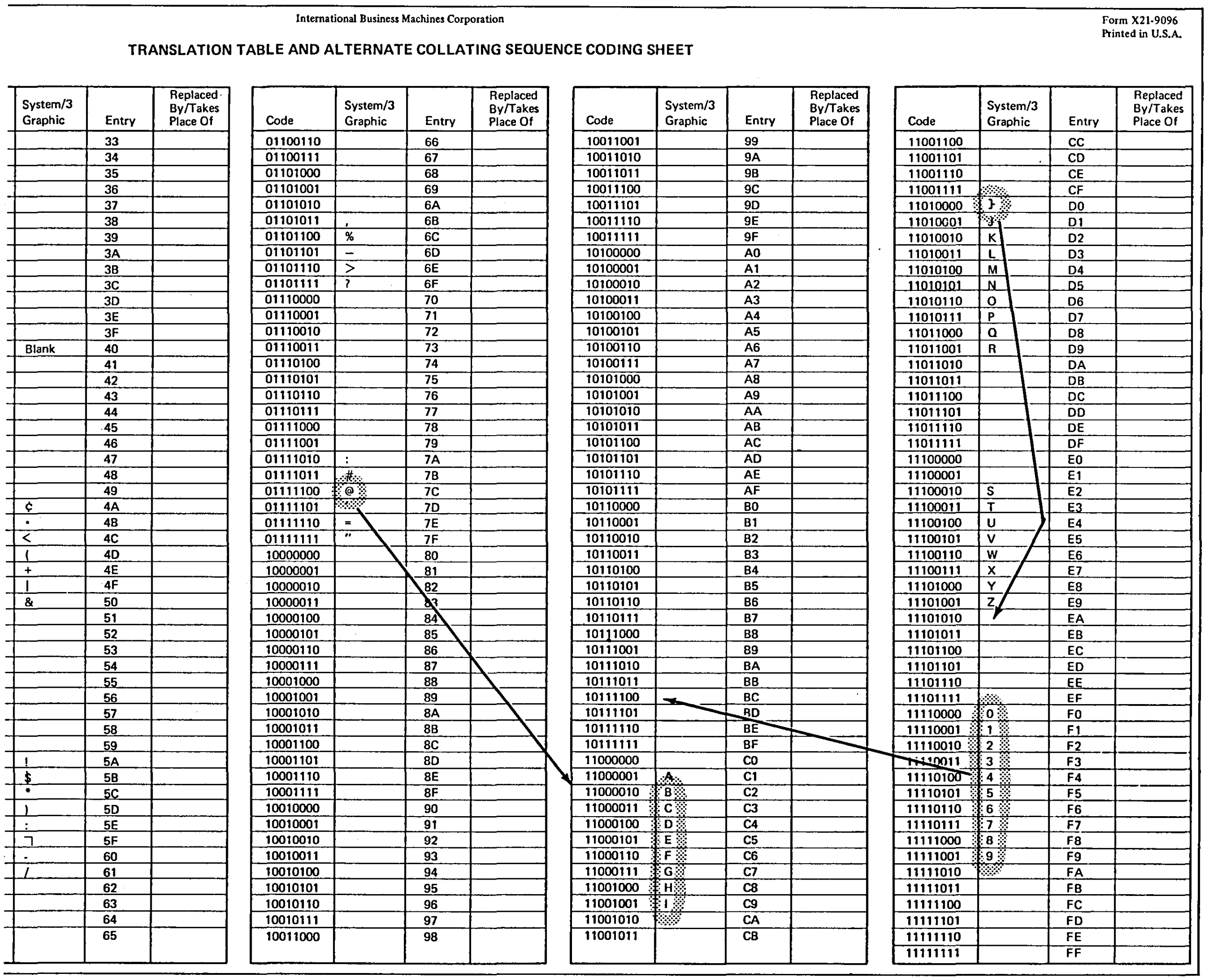

PORTION OF THE NORMAL SEQUENCE
ALTERED SEQUENCE
@ , = " A B C D E F G H I } J K L M N O P Q R S T U V W X Y Z 0 1 2 3 4 5 6 7 8 9
, = " 0 1 2 3 4 5 6 7 8 9 A @ B C D E F G H I J K L M N O P Q R S T U V W X Y Z }

Figure 16-35. Normal Sequence Versus Altered Sequence



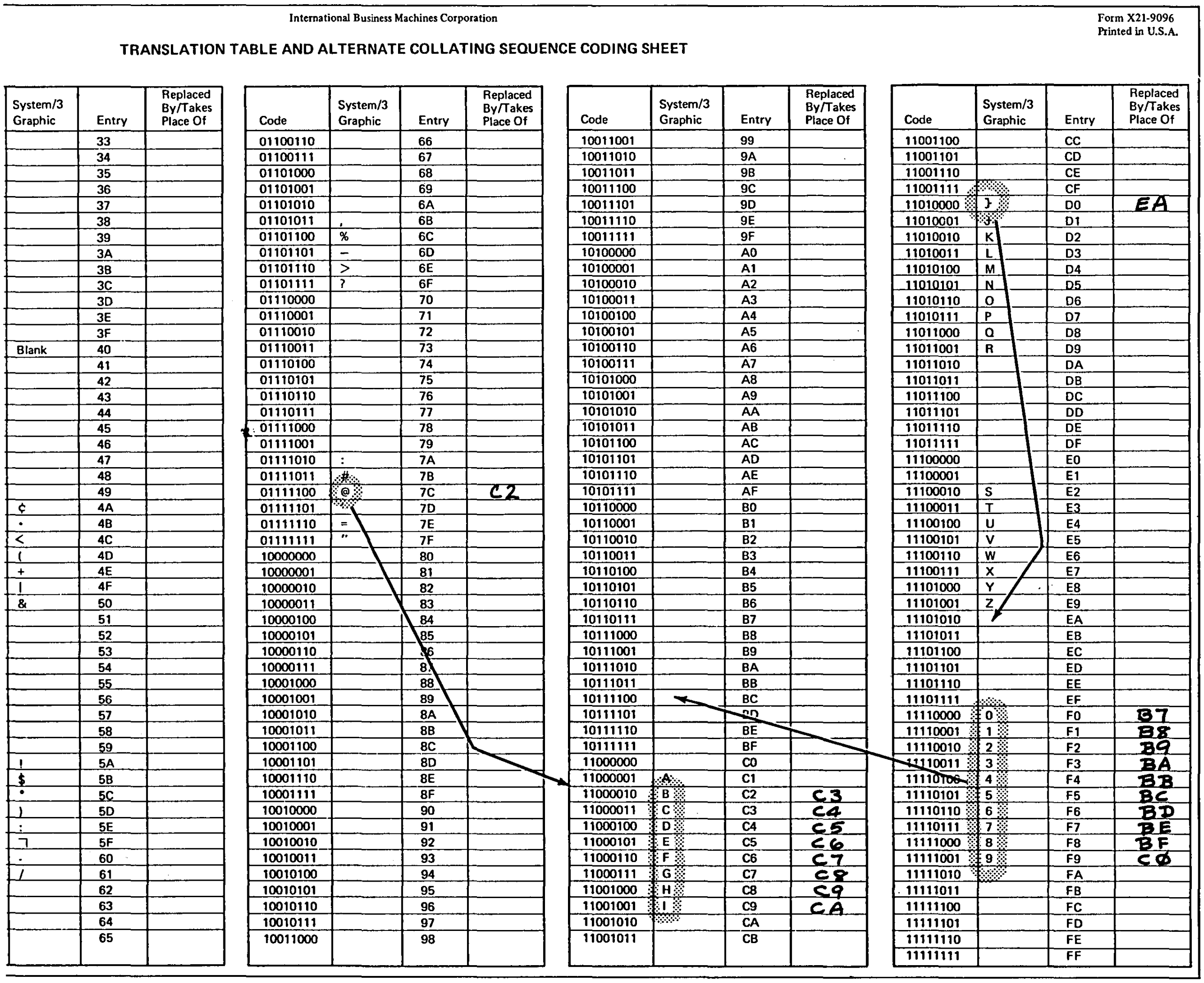
International Business Machines Corporation

Form X21-9096
Printed in U.S.A.

TRANSLATION TABLE AND ALTERNATE COLLATING SEQUENCE CODING SHEET

System/3 Graphic	Entry	Replaced By/Takes Place Of	Code	System/3 Graphic	Entry	Replaced By/Takes Place Of	Code	System/3 Graphic	Entry	Replaced By/Takes Place Of	Code	System/3 Graphic	Entry	Replaced By/Takes Place Of
	33		01100110		66		10011001		99		11001100		CC	
	34		01100111		67		10011010		9A		11001101		CD	
	35		01101000		68		10011011		9B		11001110		CE	
	36		01101001		69		10011100		9C		11001111		CF	
	37		01101010		6A		10011101		9D		11010000	}	D0	
	38		01101011	,	6B		10011110		9E		11010001	J	D1	
	39		01101100	%	6C		10011111		9F		11010010	K	D2	
	3A		01101101	–	6D		10100000		A0		11010011	L	D3	
	3B		01101110	>	6E		10100001		A1		11010100	M	D4	
	3C		01101111	?	6F		10100010		A2		11010101	N	D5	
	3D		01110000		70		10100011		A3		11010110	O	D6	
	3E		01110001		71		10100100		A4		11010111	P	D7	
	3F		01110010		72		10100101		A5		11011000	Q	D8	
Blank	40		01110011		73		10100110		A6		11011001	R	D9	
	41		01110100		74		10100111		A7		11011010		DA	
	42		01110101		75		10101000		A8		11011011		DB	
	43		01110110		76		10101001		A9		11011100		DC	
	44		01110111		77		10101010		AA		11011101		DD	
	45		01111000		78		10101011		AB		11011110		DE	
	46		01111001		79		10101100		AC		11011111		DF	
	47		01111010	:	7A		10101101		AD		11100000		E0	
	48		01111011	#	7B		10101110		AE		11100001		E1	
	49		01111100	@	7C		10101111		AF		11100010	S	E2	
¢	4A		01111101	'	7D		10110000		B0		11100011	T	E3	
.	4B		01111110	=	7E		10110001		B1		11100100	U	E4	
<	4C		01111111	"	7F		10110010		B2		11100101	V	E5	
(	4D		10000000		80		10110011		B3		11100110	W	E6	
+	4E		10000001		81		10110100		B4		11100111	X	E7	
|	4F		10000010		82		10110101		B5		11101000	Y	E8	
&	50		10000011		83		10110110		B6		11101001	Z	E9	
	51		10000100		84		10110111		B7		11101010		EA	
	52		10000101		85		10111000		B8		11101011		EB	
	53		10000110		86		10111001		B9		11101100		EC	
	54		10000111		87		10111010		BA		11101101		ED	
	55		10001000		88		10111011		BB		11101110		EE	
	56		10001001		89		10111100		BC		11101111		EF	
	57		10001010		8A		10111101		BD		11110000	0	F0	
	58		10001011		8B		10111110		BE		11110001	1	F1	
	59		10001100		8C		10111111		BF		11110010	2	F2	
!	5A		10001101		8D		11000000		C0		11110011	3	F3	
$	5B		10001110		8E		11000001	A	C1		11110100	4	F4	
*	5C		10001111		8F		11000010	B	C2		11110101	5	F5	
)	5D		10010000		90		11000011	C	C3		11110110	6	F6	
;	5E		10010001		91		11000100	D	C4		11110111	7	F7	
¬	5F		10010010		92		11000101	E	C5		11111000	8	F8	
-	60		10010011		93		11000110	F	C6		11111001	9	F9	
/	61		10010100		94		11000111	G	C7		11111010		FA	
	62		10010101		95		11001000	H	C8		11111011		FB	
	63		10010110		96		11001001	I	C9		11111100		FC	
	64		10010111		97		11001010		CA		11111101		FD	
	65		10011000		98		11001011		CB		11111110		FE	
											11111111		FF	

Figure 16-36. Changes Necessary for Altered Sequence



International Business Machines Corporation

Form X21-9096
Printed in U.S.A.

TRANSLATION TABLE AND ALTERNATE COLLATING SEQUENCE CODING SHEET

System/3 Graphic	Entry	Replaced By/Takes Place Of
	33	
	34	
	35	
	36	
	37	
	38	
	39	
	3A	
	3B	
	3C	
	3D	
	3E	
	3F	
Blank	40	
	41	
	42	
	43	
	44	
	45	
	46	
	47	
	48	
	49	
¢	4A	
.	4B	
<	4C	
(	4D	
+	4E	
|	4F	
&	50	
	51	
	52	
	53	
	54	
	55	
	56	
	57	
	58	
	59	
!	5A	
$	5B	
*	5C	
)	5D	
;	5E	
¬	5F	
-	60	
/	61	
	62	
	63	
	64	
	65	

Code	System/3 Graphic	Entry	Replaced By/Takes Place Of
01100110		66	
01100111		67	
01101000		68	
01101001		69	
01101010		6A	
01101011	,	6B	
01101100	%	6C	
01101101	–	6D	
01101110	>	6E	
01101111	?	6F	
01110000		70	
01110001		71	
01110010		72	
01110011		73	
01110100		74	
01110101		75	
01110110		76	
01110111		77	
01111000		78	
01111001		79	
01111010	:	7A	
01111011	#	7B	
01111100	@	7C	C2
01111101		7D	
01111110	=	7E	
01111111	"	7F	
10000000		80	
10000001		81	
10000010		82	
10000011		83	
10000100		84	
10000101		85	
10000110		86	
10000111		87	
10001000		88	
10001001		89	
10001010		8A	
10001011		8B	
10001100		8C	
10001101		8D	
10001110		8E	
10001111		8F	
10010000		90	
10010001		91	
10010010		92	
10010011		93	
10010100		94	
10010101		95	
10010110		96	
10010111		97	
10011000		98	

Code	System/3 Graphic	Entry	Replaced By/Takes Place Of
10011001		99	
10011010		9A	
10011011		9B	
10011100		9C	
10011101		9D	
10011110		9E	
10011111		9F	
10100000		A0	
10100001		A1	
10100010		A2	
10100011		A3	
10100100		A4	
10100101		A5	
10100110		A6	
10100111		A7	
10101000		A8	
10101001		A9	
10101010		AA	
10101011		AB	
10101100		AC	
10101101		AD	
10101110		AE	
10101111		AF	
10110000		B0	
10110001		B1	
10110010		B2	
10110011		B3	
10110100		B4	
10110101		B5	
10110110		B6	
10110111		B7	
10111000		B8	
10111001		B9	
10111010		BA	
10111011		BB	
10111100		BC	
10111101		BD	
10111110		BE	
10111111		BF	
11000000		C0	
11000001	A	C1	
11000010	B	C2	C3
11000011	C	C3	C4
11000100	D	C4	C5
11000101	E	C5	C6
11000110	F	C6	C7
11000111	G	C7	C8
11001000	H	C8	C9
11001001	I	C9	CA
11001010		CA	
11001011		CB	

Code	System/3 Graphic	Entry	Replaced By/Takes Place Of
11001100		CC	
11001101		CD	
11001110		CE	
11001111		CF	
11010000	}	D0	EA
11010001	J	D1	
11010010	K	D2	
11010011	L	D3	
11010100	M	D4	
11010101	N	D5	
11010110	O	D6	
11010111	P	D7	
11011000	Q	D8	
11011001	R	D9	
11011010		DA	
11011011		DB	
11011100		DC	
11011101		DD	
11011110		DE	
11011111		DF	
11100000		E0	
11100001		E1	
11100010	S	E2	
11100011	T	E3	
11100100	U	E4	
11100101	V	E5	
11100110	W	E6	
11100111	X	E7	
11101000	Y	E8	
11101001	Z	E9	
11101010		EA	
11101011		EB	
11101100		EC	
11101101		ED	
11101110		EE	
11101111		EF	
11110000	0	F0	B7
11110001	1	F1	B8
11110010	2	F2	B9
11110011	3	F3	BA
11110100	4	F4	BB
11110101	5	F5	BC
11110110	6	F6	BD
11110111	7	F7	BE
11111000	8	F8	BF
11111001	9	F9	C0
11111010		FA	
11111011		FB	
11111100		FC	
11111101		FD	
11111110		FE	
11111111		FF	

Figure 16-37. Coding for Altered Sequence

Punched Cards for the Altered Sequence

After you have coded all specifications for the alternate collating sequence, you can record them so that they can be used by the computer. Records describing the alternate sequence are to be formatted as follows:

Columns	*Entries*
1-6	ALTSEQ (This entry allows the computer to recognize that this card is describing an alternate sequence.)
7-8	Blank
9-96	The hexadecimal values involved in changing the sequence

In columns 9-96, there are 22 groups of 4 columns. Each group (9-12, 13-16, etc.) must contain two hexadecimal values involved in changing the sequence. The first two columns of a group are for the hexadecimal value taken from the *Entry* column of the Alternate Collating Sequence Coding Sheet. The last two columns in a group are for the hexadecimal value taken from the *Replaced By* column of the coding sheet (Figure 16-38).

More than one record may be used to specify changes in collating sequence. However, each additional record must be formatted in the same way as the first.

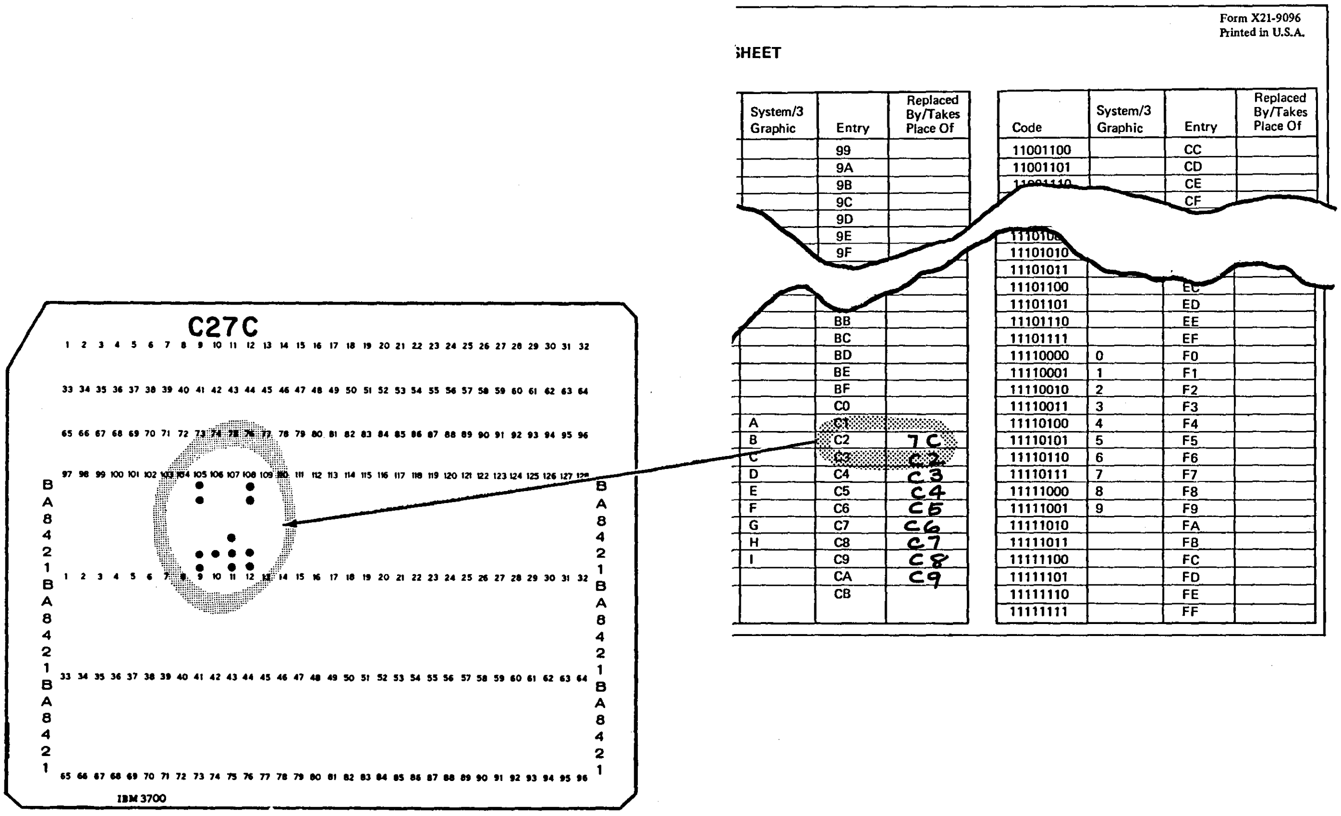
The first blank appearing in columns 9-96 is recognized by the computer as the end of the record. Consequently, blanks must not appear between pairs of hexadecimal values.

Two additional records must be included along with the alternate collating sequence records. One record containing **ƀ (two asterisks and a blank) in columns 1 through 3 must precede the sequence records; the other record, containing /*ƀ (slash, asterisk, and blank) in columns 1 through 3 must follow the sequence records.

All records (except the RPG II control card) used for altering the collating sequence must follow RPG II specifications (or file translation specifications, when used) and must precede any tables being entered.

Figure 16-39 shows the cards containing the different types of information that you must supply to alter the collating sequence. Reference numbers appearing in parentheses identify the order in which the cards must be placed.

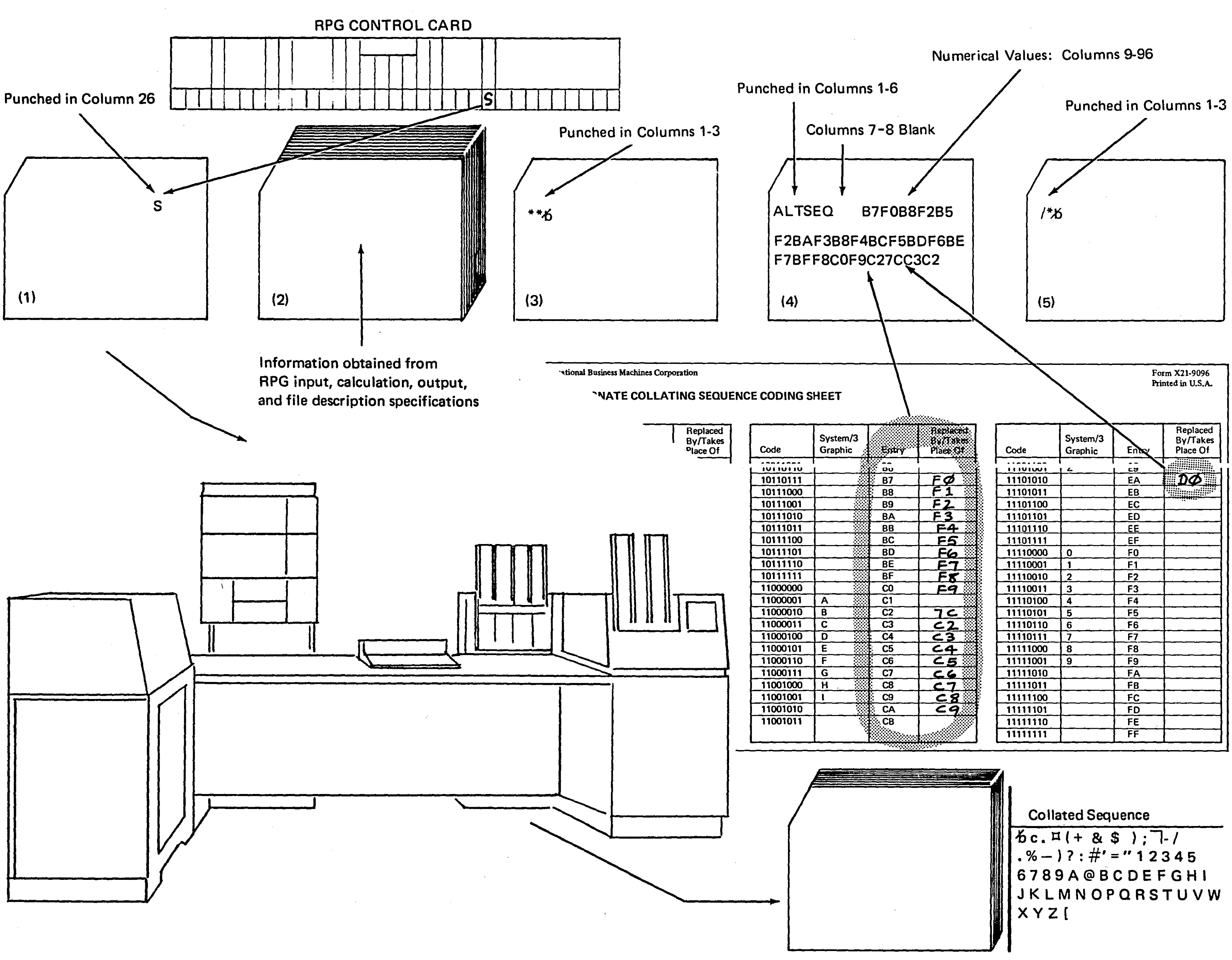
Information presented in the lower right-hand corner of the figure shows the altered collating sequence that will be recognized by the computer during the processing of the job. After the job has been processed, the computer automatically resumes using the normal collating sequence.



System/3 Graphic	Entry	Replaced By/Takes Place Of
	99	
	9A	
	9B	
	9C	
	9D	
	9E	
	9F	
	BB	
	BC	
	BD	
	BE	
	BF	
	C0	
A	C1	
B	C2	7C
C	C3	C2
D	C4	C3
E	C5	C4
F	C6	C5
G	C7	C6
H	C8	C7
I	C9	C8
	CA	C9
	CB	

Code	System/3 Graphic	Entry	Replaced By/Takes Place Of
11001100		CC	
11001101		CD	
11001110		CE	
		CF	
1110100			
11101010			
11101011			
11101100		EC	
11101101		ED	
11101110		EE	
11101111		EF	
11110000	0	F0	
11110001	1	F1	
11110010	2	F2	
11110011	3	F3	
11110100	4	F4	
11110101	5	F5	
11110110	6	F6	
11110111	7	F7	
11111000	8	F8	
11111001	9	F9	
11111010		FA	
11111011		FB	
11111100		FC	
11111101		FD	
11111110		FE	
11111111		FF	

Figure 16-38. Punching Alternate Collating Sequence Cards



ational Business Machines Corporation

Form X21-9096
Printed in U.S.A.

NATE COLLATING SEQUENCE CODING SHEET

Code	System/3 Graphic	Entry	Replaced By/Takes Place Of
10110110		B6	
10110111		B7	FØ
10111000		B8	F1
10111001		B9	F2
10111010		BA	F3
10111011		BB	F4
10111100		BC	F5
10111101		BD	F6
10111110		BE	F7
10111111		BF	F8
11000000		C0	F9
11000001	A	C1	
11000010	B	C2	7C
11000011	C	C3	C2
11000100	D	C4	C3
11000101	E	C5	C4
11000110	F	C6	C5
11000111	G	C7	C6
11001000	H	C8	C7
11001001	I	C9	C8
11001010		CA	C9
11001011		CB	

Code	System/3 Graphic	Entry	Replaced By/Takes Place Of
11101001	Z	E9	
11101010		EA	DØ
11101011		EB	
11101100		EC	
11101101		ED	
11101110		EE	
11101111		EF	
11110000	0	F0	
11110001	1	F1	
11110010	2	F2	
11110011	3	F3	
11110100	4	F4	
11110101	5	F5	
11110110	6	F6	
11110111	7	F7	
11111000	8	F8	
11111001	9	F9	
11111010		FA	
11111011		FB	
11111100		FC	
11111101		FD	
11111110		FE	
11111111		FF	

Collated Sequence

b̸c.¤(+ & $);¬-/
.%—)?:#'=" 12345
6789A@BCDEFGHI
JKLMNOPQRSTUVW
XYZ[

Figure 16-39. Summary of Alternate Collating Sequence Specifications

ALTERING THE STRUCTURE OF CHARACTERS

You learned in the discussion of character structure that each System/3 graphic is represented in the machine by a unique setting of eight bits; four zone bits and four digit bits. If any change is made to either the zone or digit bits, the entire character is changed. For example, if the *A* bit of the letter *M* is changed from on to off, the letter *M* becomes the letter *D* (Figure 16-40).

You can, of course, change a character before it is read into the computer by punching different zone punches on the card. But you can also change a character after it has been read. This is done by changing the zones of characters through the use of move zone operation codes.

Why would you ever want to change the zone of a character after it has been read? One common reason for changing zones is to deliberately change the sign of a field from positive to negative, or vice verasa.

This is necessary when a numeric field read in from a special file has its sign in the high-order (leftmost) position of the field. Numeric fields are required to have the sign in the low-order (rightmost) position of the field. Thus, a numeric input field having its sign in the high-order position must have its sign moved to the low-order position. The move zone operations allow you to do this.

How Move Zone Operations Work

Move zone operations involve only the zone portion of characters. The computer does not actually move the zone of one character to the zone portion of another. Rather, it changes a character by making its zone identical to the zone of the character which you indicate should serve as the model. The character serving as a model is not changed by the operation.

Thus, in order to use the move zone operations you must have:

1. A character which needs to be changed.

2. A character that has the zone you want the changed character to have.

For example, if you want the low-order (rightmost) position of the field AMOUNT to be changed from a positive *5* to a negative *5* you must have a character to serve as a model whose zone portion is the same as the zone of a negative five.

Coding a Move Zone Operation

Figure 16-41 illustrates the way in which a move zone operation is coded. The name of the field containing the character to be changed must be entered in the Result Field. Either a constant or the name of the field which contains the model character must be entered in Factor 2. The move zone operation code is specified in the Operation columns (28-32). Any conditioning indicators you wish to use can be specified, but resulting indicators cannot be used.

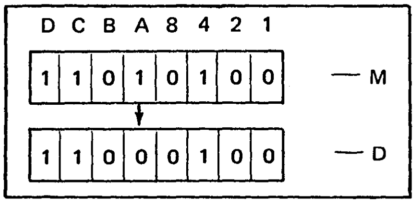
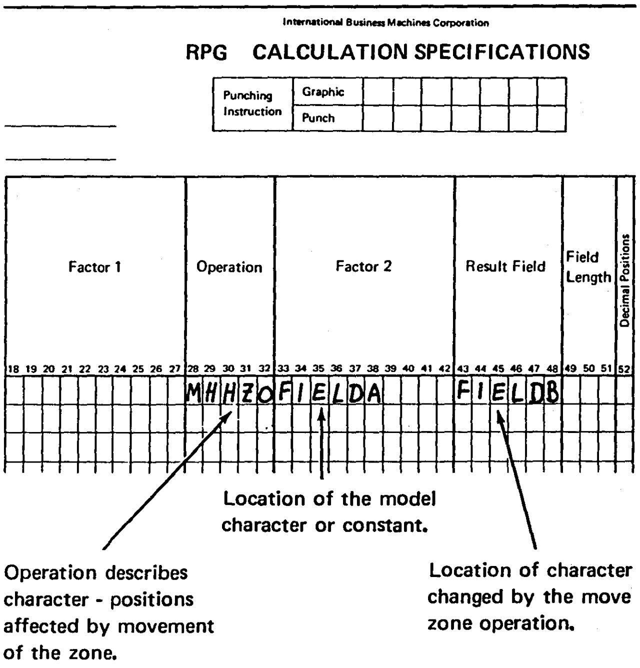


Figure 16-40. Changing Zones Changes Characters



Location of the model character or constant.

Operation describes character - positions affected by movement of the zone.

Location of character changed by the move zone operation.

Figure 16-41. Coding for a Move Zone Instruction

Differences in the Move Zone Operations

There are four different move zone operation codes available. Each code involves the zones of characters located in different positions; namely:

1. High-order positions in both Factor 2 and the Result Field.
2. High-order position in Factor 2 and low-order in the Result Field.
3. Low-order positions in both Factor 2 and the Result Field.
4. Low-order position in Factor 2 and high-order in the Result Field.

Since only the zones of high and low-order characters in a field or constant are involved in the move zone operations, only the high or low-order positions of a field can be changed.

Figure 16-42 illustrates the ways in which the four operation codes affect the zone of a character in the Result Field.

Move From High-Order Zone to High-Order Zone (MHHZO)

This operation code moves the zone of the high-order alphameric character in the constant or field entered in Factor 2 to the high-order alphameric character in the Result Field.

Move From Low-Order Zone to High-Order Zone (MLHZO)

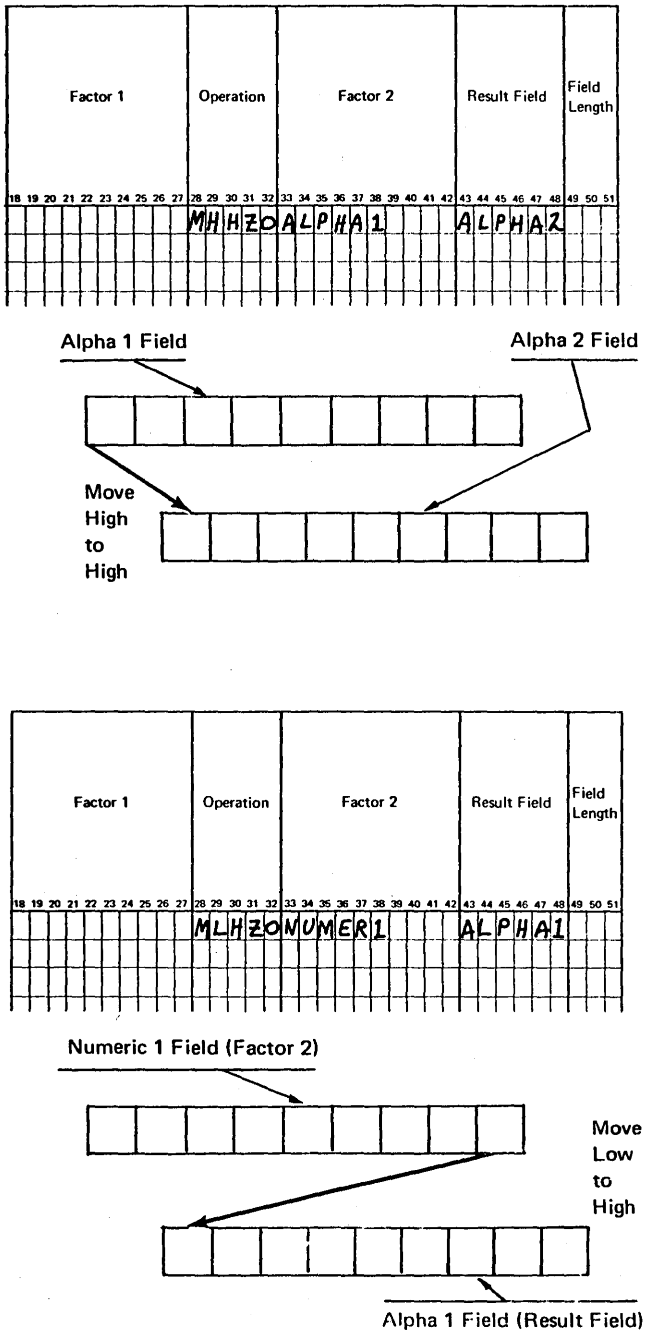
This operation code moves the zone of the low-order character in the field or constant entered in Factor 2 to the high-order alphameric character in the Result Field. The Result Field must be alphameric; Factor 2 can be either numeric or alphameric.

Move From High-Order Zone to Low-Order Zone (MHLZO)

This operation code moves the zone of the high-order alphameric character in the constant or field entered in Factor 2 to the low-order rightmost character in the Result Field. Because of its high-order zone, Factor 2 must be an alphameric field. The Result Field can be either alphameric or numeric.

Move From Low-Order Zone to Low-Order Zone (MLLZO)

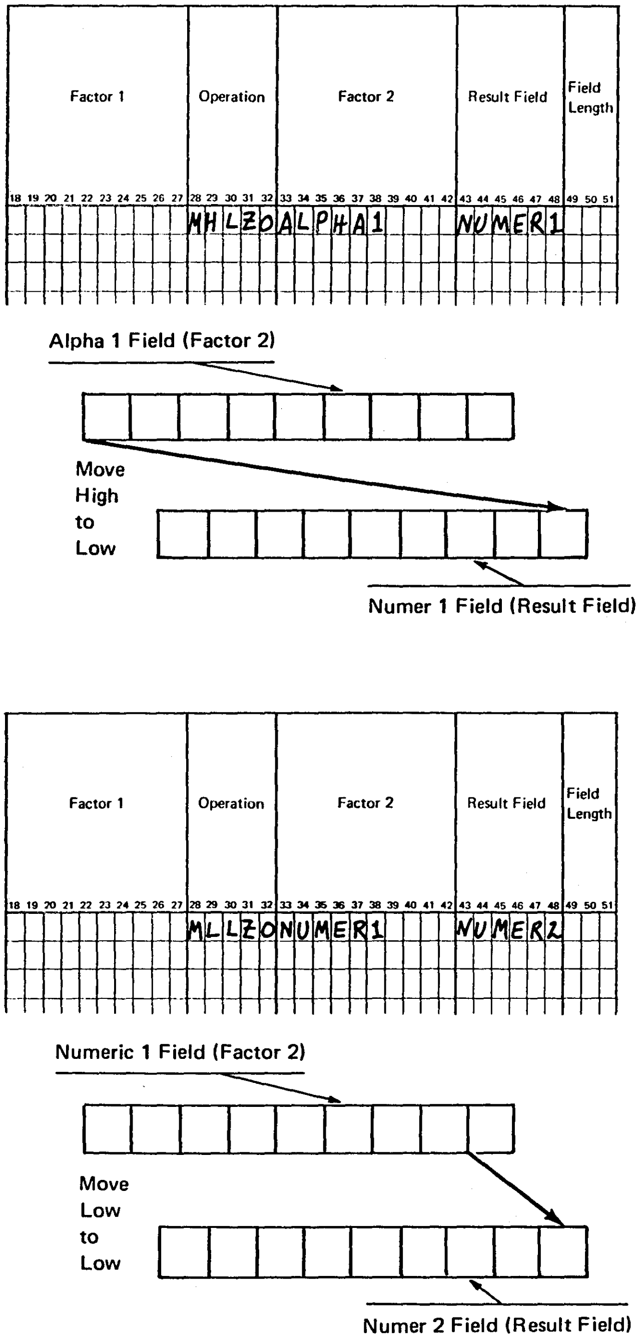
This operation code moves the zone of the low-order character in the field or constant entered in Factor 2 field to the low-order character in the Result Field. Both Factor 2 and the Result Field can be either numeric or alphameric.



Factor 1	Operation	Factor 2	Result Field	Field Length
18 19 20 21 22 23 24 25 26 27	28 29 30 31 32	33 34 35 36 37 38 39 40 41 42	43 44 45 46 47 48	49 50 51
	MHHZO	ALPHA1	ALPHA2	

Factor 1	Operation	Factor 2	Result Field	Field Length
18 19 20 21 22 23 24 25 26 27	28 29 30 31 32	33 34 35 36 37 38 39 40 41 42	43 44 45 46 47 48	49 50 51
	MLHZO	NUMER1	ALPHA1	

Figure 16-42. Move Zone Operations (part 1 of 2)



Factor 1	Operation	Factor 2	Result Field	Field Length
18 19 20 21 22 23 24 25 26 27	28 29 30 31 32	33 34 35 36 37 38 39 40 41 42	43 44 45 46 47 48	49 50 51
	MHLZO	ALPHA1	NUMER1	

Alpha 1 Field (Factor 2)
Move
High
to
Low
Numer 1 Field (Result Field)

Factor 1	Operation	Factor 2	Result Field	Field Length
18 19 20 21 22 23 24 25 26 27	28 29 30 31 32	33 34 35 36 37 38 39 40 41 42	43 44 45 46 47 48	49 50 51
	MLLZO	NUMER1	NUMER2	

Numeric 1 Field (Factor 2)
Move
Low
to
Low
Numer 2 Field (Result Field)

Figure 16-42. Move Zone Operations (part 2 of 2)

Field Format and Move Zone Operations

As you read the description of each move zone operation, you probably noticed that special attention was given to the types of fields which can be used with each operation. Keep in mind that you cannot move from or to the high-order positions of a numeric field because the computer does not use the high-order zone of fields defined as numeric.

Which of the following move zone operations can be done if the two fields involved have formats as given below?

1. Alphameric to Alphameric: MHLZO

2. Alphameric to Numeric: MHHZO

3. Numeric to Alphameric: MLHZO

4. Numeric to Alphameric: MHHZO

5. Numeric to Numeric: MLHZO

6. Numeric to Numeric: MLLZO

Items *1, 3,* and *6* can be done. Items *2, 4,* and *5* cannot be done. Item *2* suggests that the zone of the high-order position in the numeric field be changed. The computer does not use high-order zone of numeric fields. Item *4* suggests that the zone of the high-order character is to serve as a model. It cannot because the computer does not work with the zones of high-order characters in a numeric field. Item *5* cannot be done because again it involves high-order positions of numeric fields.

Example of a Move Zone Operation

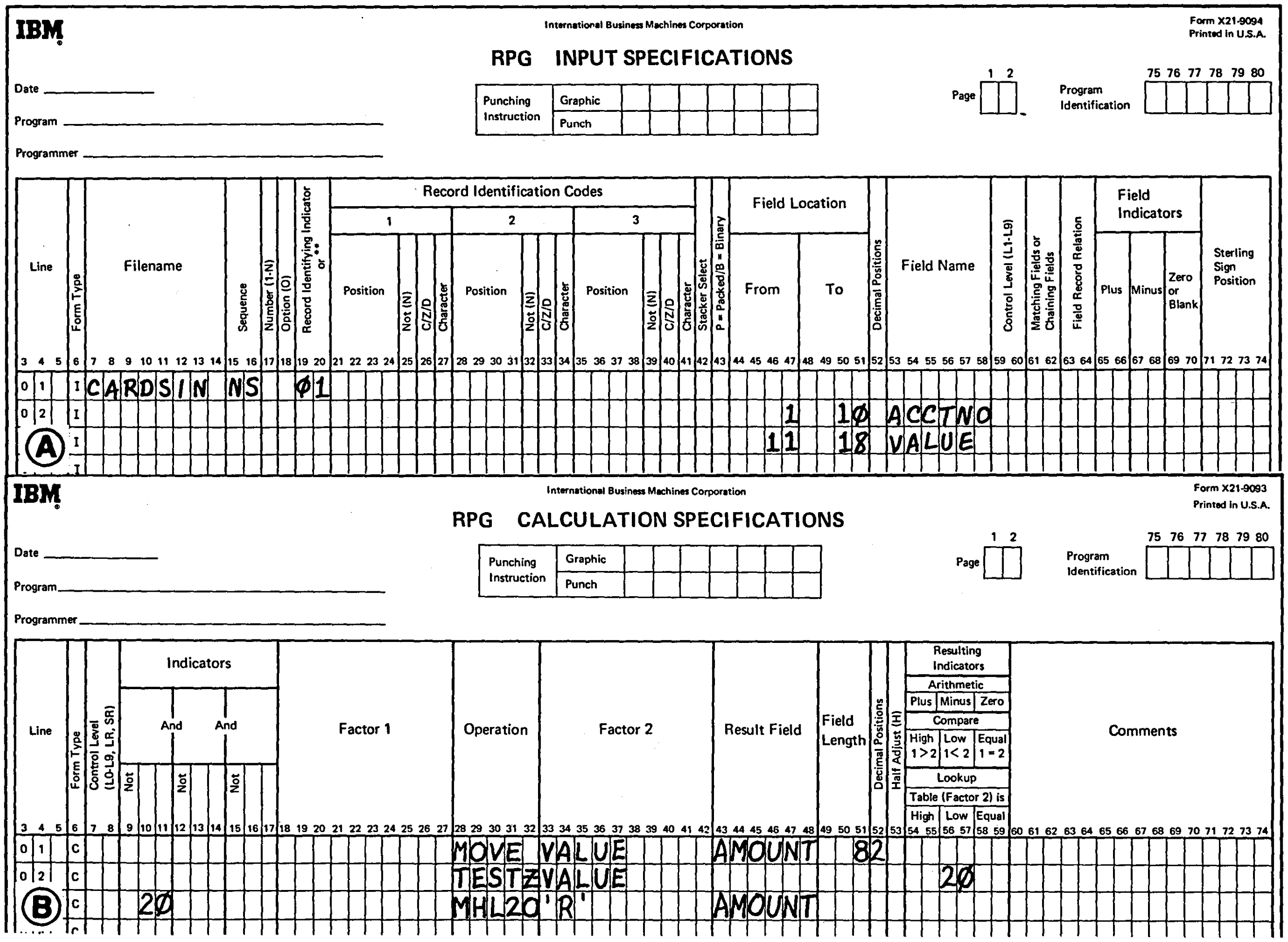
Now that you know how the various move zone operation codes work, let's see how they can be used to change the sign of the field, VALUE, from the high-order to the low-order position.

Naturally any field that has zones other than in the low-order position must be defined as alphameric if those zones are to be used by the computer. But if the field is to be involved in/an arithmetic operation, it must be numeric.

To allow for both possibilities, you could define the field twice; once as alphameric and once as numeric. (Two unique field names are needed.) Another possibility is to define the field once as alphameric and then change it into a numeric field by moving it into a numeric field. This is what is done in the example (Figure 16-43).

Before doing any arithmetic operation, you must get the sign in the low-order position of a *numeric* field. First, you must determine what the sign is. This is done by the TESTZ operation. Remember that TESTZ turns on the minus indicator when it finds the characters –, }, or *J* through *R*. The specification in Figure 16-43, insert B, line 02 causes indicator 20 to turn on if the sign of the field is minus. If indicator 20 is on, the zone of an *R*, which is the minus sign to the computer, is moved to the low-order position of the AMOUNT field. If the field tested is plus, no zone is moved because a numeric field having no minus sign is automatically assumed to be positive.

Notice that the MHLZO (Move High to Low Zone) operation code was used to change the zone of the low-order position of the AMOUNT field by giving it the same zone as the constant *R*. MLLZO (Move Low to Low Zone) could also have been used because the one and only character in the alphameric constant specified serves as both the high and low-order character.



IBM International Business Machines Corporation Form X21-9094 Printed in U.S.A.

RPG INPUT SPECIFICATIONS

Date ______ Program ______ Programmer ______

Line	Form Type	Filename	Sequence	Number (1-N)	Option (O)	Record Identifying Indicator	Field Location From	Field Location To	Decimal Positions	Field Name
01	I	CARDSIN	NS			Ø1				
02	I						1	1Ø		ACCTNO
Ⓐ	I						11	18		VALUE
	I									

IBM International Business Machines Corporation Form X21-9093 Printed in U.S.A.

RPG CALCULATION SPECIFICATIONS

Date ______ Program ______ Programmer ______

Line	Form Type	Control Level	Indicators	Factor 1	Operation	Factor 2	Result Field	Field Length	Decimal Positions	Resulting Indicators Plus/High	Minus/Low	Zero/Equal	Comments
01	C				MOVE	VALUE	AMOUNT	8	2				
02	C				TESTZ	VALUE					2Ø		
Ⓑ	C		2Ø		MHL2O	'R'	AMOUNT						
	C												

Figure 16-43. Using Move Zone Operations to Change the Sign of a Field

Choosing the Model Character for Factor 2

Before specifying a move zone operation, you must have a character designated in Factor 2 whose zone will give the desired zone in the Result Field.

Usually you will use move zone operations to change the signs of fields. Using any numbers in Factor 2 will produce a positive character in the Result Field. Using any one of the characters –, } , or *J-R* in Factor 2 will give you a negative character. Remember that negative numbers are punched with a *B* punch (minus sign) over the number. The punch combinations of negative numbers have the same numeric value in the computer as *J-R.* Thus, when you specify that the zone of a character should be made like the zone of *J-R,* you will get a minus character. See *Character Structure* for more information.

Use Figure 16-27 as a guide for selecting the zone which will produce the desired change.

TRANSLATING CHARACTERS

In the previous discussion, you learned that the computer can alter the structure of characters by moving zones. But, through the file translation function of the RPG II language, it can do even more. It can translate one character into another.

The translating function is known as file translation because characters can be translated either when they are read in or before they are recorded in the output file. The computer acts like an interpreter. Just as a human interpreter translates languages (a word in German for a word in English), the computer translates characters by replacing one character with another.

Need for File Translation

Think of the use for file translation when translating codes. Codes are often used as a security measure to prevent access to classified information. Information is recorded on cards in coded form. In order to process the information, it must be decoded. A coded character must be replaced by the corresponding decoded character.

For example, a firm which keeps all information classified uses the characters in the word FITZGERALD as a code for the numbers 0 through 9. *F* is the code for zero, *I* for one, etc. When recorded on a card, the number 1432 appears as IGZT. If a field containing IGZT is read into the computer and used in arithmetic operations, results received are wrong. IGZT must first be decoded, or translated into 1432.

Specifying File Translation

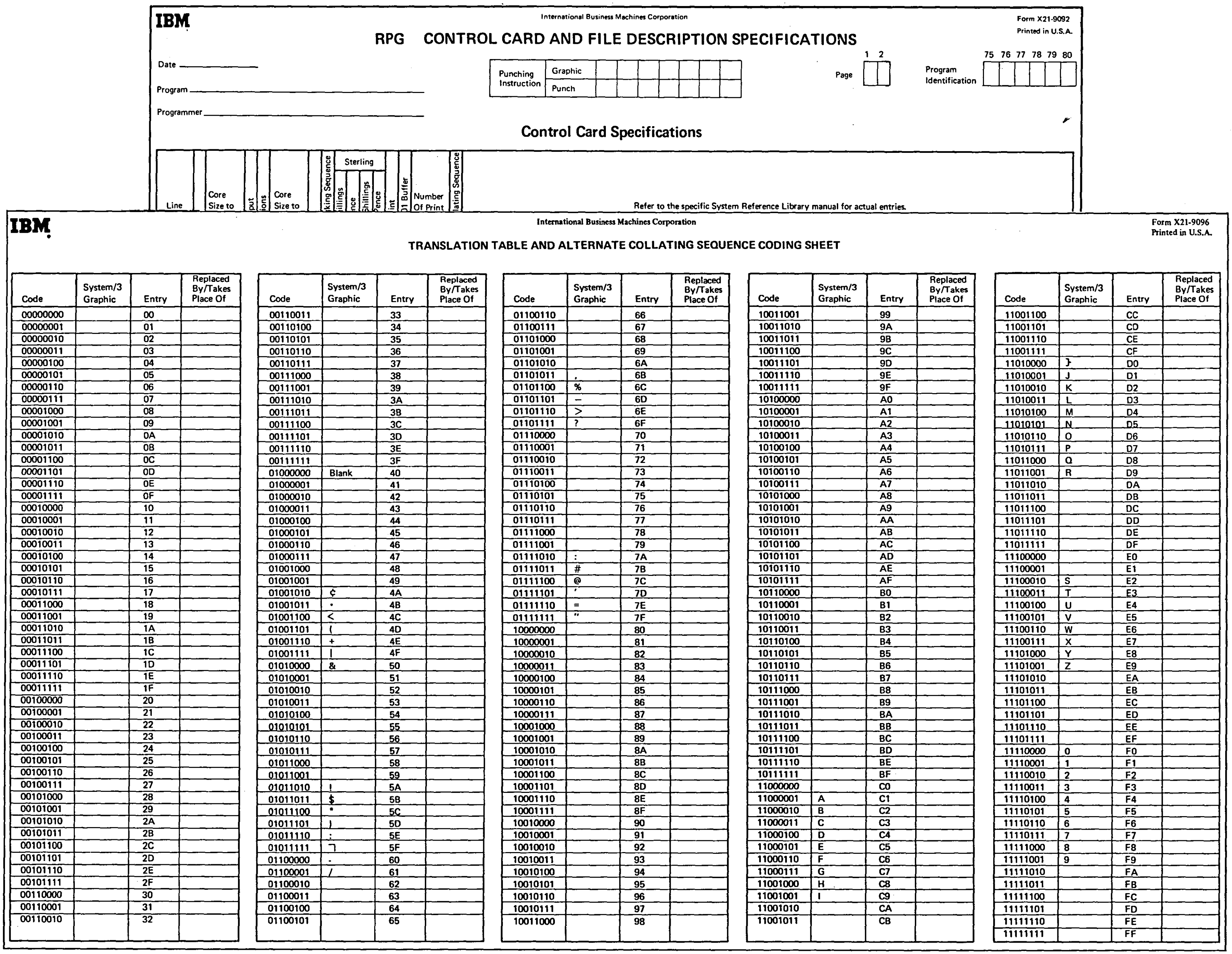
Specifications for file translation are identical to those used to alter the collating sequence.

Forms Used for a File Translation

Figure 16-44 shows the forms on which you must specify the way in which files are to be translated. One form consists of the RPG II Control Card and File Description Sheet; the other consists of the Translation Table and Alternate Collating Sequence Coding Sheet for listing the characters to be translated. Both forms are used in conjunction with the RPG II Input, Output, and Calculation Sheets.

Only column 43 in the RPG II control card relates to the change in sequence. A letter *F* entered in column 43 notifies the computer that additional information furnished as part of the job relates to translating files. All other columns contain the information that must normally be entered to process a job.

The Translation Table and Alternate Collating Sequence Coding Sheet lists 256 bit combinations along with their hexadecimal numerical values. You learned from discussions of character structure that the first number in the hexadecimal value represents the numerical value of the character's digit and the second number represents the numerical value of the character's zone. The 64 printable characters are listed beside the bit combination and hexadecimal values with which they are associated.



IBM

International Business Machines Corporation

Form X21-9092
Printed in U.S.A.

RPG CONTROL CARD AND FILE DESCRIPTION SPECIFICATIONS

Date ____________

Program ____________

Programmer ____________

Punching Instruction	Graphic							
	Punch							

Page 1 2

Program Identification 75 76 77 78 79 80

Control Card Specifications

Line | Core Size to | put | ions | Core Size to | king Sequence | Sterling: illings, nce, Shillings, Pence | int | 01 Buffer | Number Of Print | ating Sequence

Refer to the specific System Reference Library manual for actual entries.

IBM

International Business Machines Corporation

Form X21-9096
Printed in U.S.A.

TRANSLATION TABLE AND ALTERNATE COLLATING SEQUENCE CODING SHEET

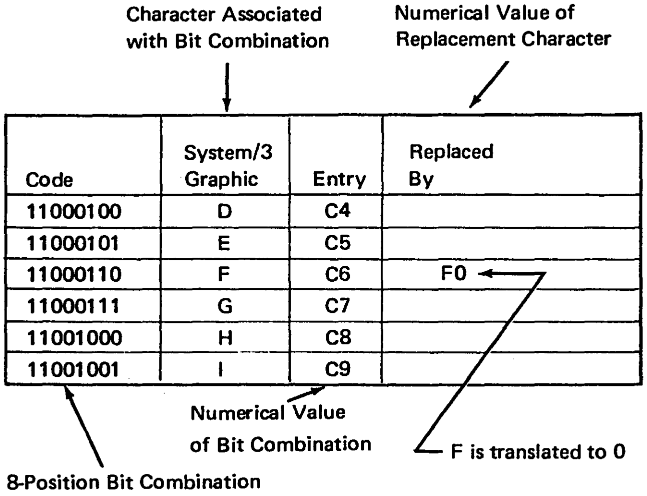
Code	System/3 Graphic	Entry	Replaced By/Takes Place Of
00000000		00	
00000001		01	
00000010		02	
00000011		03	
00000100		04	
00000101		05	
00000110		06	
00000111		07	
00001000		08	
00001001		09	
00001010		0A	
00001011		0B	
00001100		0C	
00001101		0D	
00001110		0E	
00001111		0F	
00010000		10	
00010001		11	
00010010		12	
00010011		13	
00010100		14	
00010101		15	
00010110		16	
00010111		17	
00011000		18	
00011001		19	
00011010		1A	
00011011		1B	
00011100		1C	
00011101		1D	
00011110		1E	
00011111		1F	
00100000		20	
00100001		21	
00100010		22	
00100011		23	
00100100		24	
00100101		25	
00100110		26	
00100111		27	
00101000		28	
00101001		29	
00101010		2A	
00101011		2B	
00101100		2C	
00101101		2D	
00101110		2E	
00101111		2F	
00110000		30	
00110001		31	
00110010		32	
00110011		33	
00110100		34	
00110101		35	
00110110		36	
00110111		37	
00111000		38	
00111001		39	
00111010		3A	
00111011		3B	
00111100		3C	
00111101		3D	
00111110		3E	
00111111		3F	
01000000	Blank	40	
01000001		41	
01000010		42	
01000011		43	
01000100		44	
01000101		45	
01000110		46	
01000111		47	
01001000		48	
01001001		49	
01001010	¢	4A	
01001011	.	4B	
01001100	<	4C	
01001101	(	4D	
01001110	+	4E	
01001111	|	4F	
01010000	&	50	
01010001		51	
01010010		52	
01010011		53	
01010100		54	
01010101		55	
01010110		56	
01010111		57	
01011000		58	
01011001		59	
01011010	!	5A	
01011011	$	5B	
01011100	*	5C	
01011101	)	5D	
01011110	;	5E	
01011111	¬	5F	
01100000	-	60	
01100001	/	61	
01100010		62	
01100011		63	
01100100		64	
01100101		65	
01100110		66	
01100111		67	
01101000		68	
01101001		69	
01101010		6A	
01101011	,	6B	
01101100	%	6C	
01101101	–	6D	
01101110	>	6E	
01101111	?	6F	
01110000		70	
01110001		71	
01110010		72	
01110011		73	
01110100		74	
01110101		75	
01110110		76	
01110111		77	
01111000		78	
01111001		79	
01111010	:	7A	
01111011	#	7B	
01111100	@	7C	
01111101	'	7D	
01111110	=	7E	
01111111	"	7F	
10000000		80	
10000001		81	
10000010		82	
10000011		83	
10000100		84	
10000101		85	
10000110		86	
10000111		87	
10001000		88	
10001001		89	
10001010		8A	
10001011		8B	
10001100		8C	
10001101		8D	
10001110		8E	
10001111		8F	
10010000		90	
10010001		91	
10010010		92	
10010011		93	
10010100		94	
10010101		95	
10010110		96	
10010111		97	
10011000		98	
10011001		99	
10011010		9A	
10011011		9B	
10011100		9C	
10011101		9D	
10011110		9E	
10011111		9F	
10100000		A0	
10100001		A1	
10100010		A2	
10100011		A3	
10100100		A4	
10100101		A5	
10100110		A6	
10100111		A7	
10101000		A8	
10101001		A9	
10101010		AA	
10101011		AB	
10101100		AC	
10101101		AD	
10101110		AE	
10101111		AF	
10110000		B0	
10110001		B1	
10110010		B2	
10110011		B3	
10110100		B4	
10110101		B5	
10110110		B6	
10110111		B7	
10111000		B8	
10111001		B9	
10111010		BA	
10111011		BB	
10111100		BC	
10111101		BD	
10111110		BE	
10111111		BF	
11000000		C0	
11000001	A	C1	
11000010	B	C2	
11000011	C	C3	
11000100	D	C4	
11000101	E	C5	
11000110	F	C6	
11000111	G	C7	
11001000	H	C8	
11001001	I	C9	
11001010		CA	
11001011		CB	
11001100		CC	
11001101		CD	
11001110		CE	
11001111		CF	
11010000	}	D0	
11010001	J	D1	
11010010	K	D2	
11010011	L	D3	
11010100	M	D4	
11010101	N	D5	
11010110	O	D6	
11010111	P	D7	
11011000	Q	D8	
11011001	R	D9	
11011010		DA	
11011011		DB	
11011100		DC	
11011101		DD	
11011110		DE	
11011111		DF	
11100000		E0	
11100001		E1	
11100010	S	E2	
11100011	T	E3	
11100100	U	E4	
11100101	V	E5	
11100110	W	E6	
11100111	X	E7	
11101000	Y	E8	
11101001	Z	E9	
11101010		EA	
11101011		EB	
11101100		EC	
11101101		ED	
11101110		EE	
11101111		EF	
11110000	0	F0	
11110001	1	F1	
11110010	2	F2	
11110011	3	F3	
11110100	4	F4	
11110101	5	F5	
11110110	6	F6	
11110111	7	F7	
11111000	8	F8	
11111001	9	F9	
11111010		FA	
11111011		FB	
11111100		FC	
11111101		FD	
11111110		FE	
11111111		FF	

Figure 16-44. Forms Needed for File Translation Specifications

Coding the Translation

Each character that will be affected during the translation of a specified file must be identified on the coding sheet. In the column entitled *Replaced By,* enter the hexadecimal value of the character which is to replace the character presently associated with the bit combination shown. This means that the character associated with the value found in the *Entry* column will be translated into the character associated with the value entered in the *Replaced By* column.

Figure 16-45 illustrates the entry made on the coding sheet to translate a character. If an input file is to be translated, this entry means that the letter *F* will be translated as the number *0* (F0 is the hexadecimal value associated with 0). If the output file is to be translated, this entry means that the number *0* will be translated back into an *F* before being written out. You can think of the character associated with the value in the *Entry* column as being the character read in or printed out. On the other hand, the character associated with the value in the *Replaced By* column is the character represented in the machine (Figure 16-46).


Character Associated with Bit Combination
Numerical Value of Replacement Character
Numerical Value of Bit Combination
F is translated to 0
8-Position Bit Combination

Code	System/3 Graphic	Entry	Replaced By
11000100	D	C4	
11000101	E	C5	
11000110	F	C6	F0
11000111	G	C7	
11001000	H	C8	
11001001	I	C9	

Figure 16-45. Explanation of File Translation Coding Sheet

Differences Between File Translation and Alternate Collating Sequence

Because of the similarity of entries used in coding an alternate collating sequence and a file translation, these functions may seem identical. They are not, however. The difference occurs in the way the computer works with the characters involved.

When alternate collating sequence is used, the characters are altered only temporarily for sequencing operations. The original bit combination of the character, obtained from the punch combination for that character, is not changed. Temporary substitution of another bit combination is done instead.

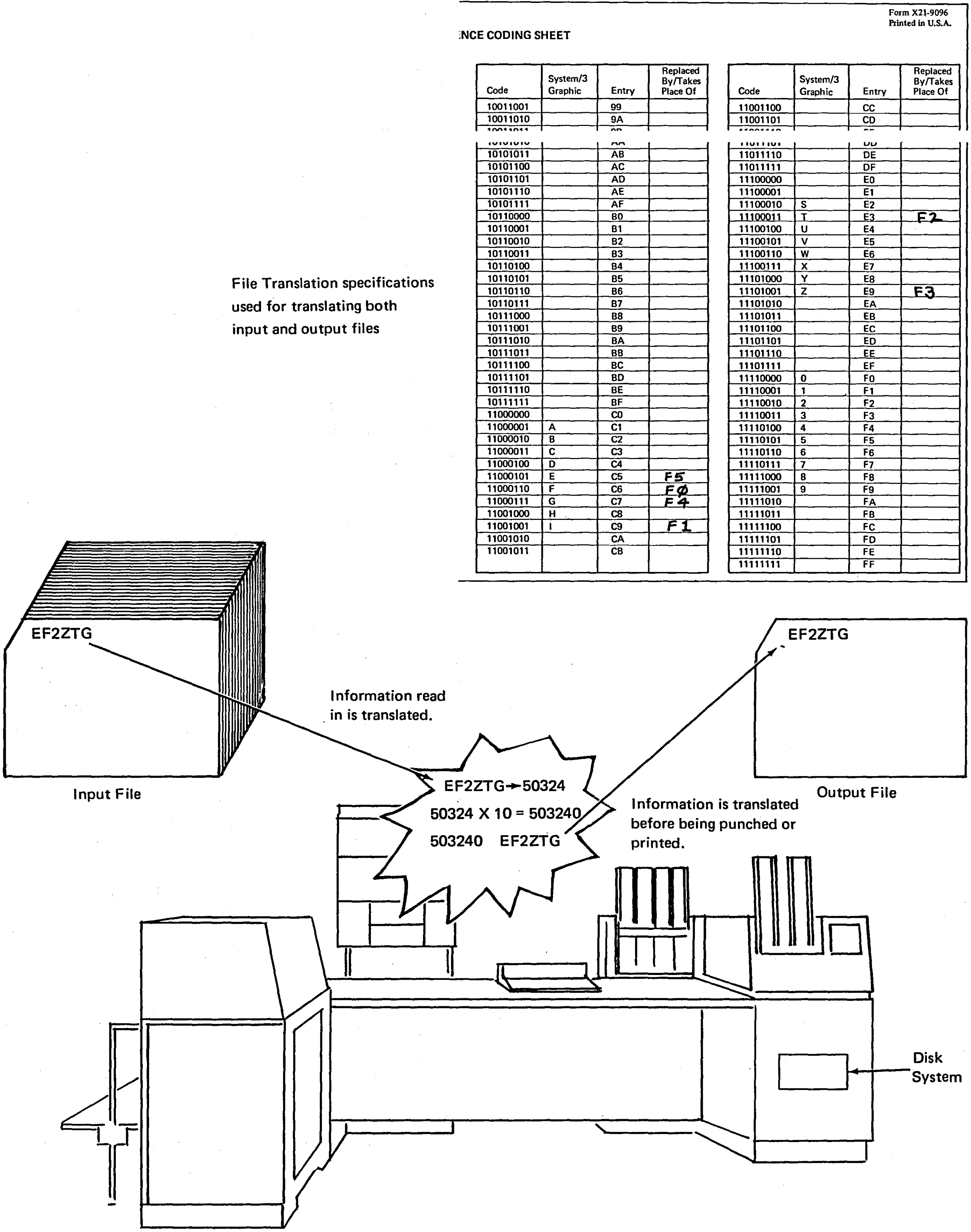
For file translation, bit combinations are actually changed. As a result, one character is changed (translated) into another. This translation occurs before your program instructions are executed.

What Files Should Be Translated?

Any input files which contain information recorded in coded form should be translated if correct results are to be obtained. All characters which you specify to be translated are translated whenever they are encountered. This means if you specify an *F* to be translated to *0,* all *F*'s read in will be translated. When there are several other fields on the cards in addition to the one containing coded information, remember all characters specified to be translated are translated regardless of fields.

When printing or punching information out, you may or may not find it necessary to specify file translation for the output files. If you have translated (decoded) your input file, you should translate information back into coded form before it is written or punched out. If all *F*'s are translated as *0*'s when read in, then all *0*'s should be translated to *F*'s before they are put out. Keep in mind that only characters which you specify are involved in the retranslation.

If you do not specify file translation for output files, information is put out exactly as it is in the machine. If you do not intend to translate output files, be certain that all characters from the input file are translated into a value associated with a printable graphic. Any hexadecimal value which does not have an associated graphic cannot be written or punched out (Figure 16-47). If an unprintable graphic is specified to be put out, a blank appears in its place.



Form X21-9096
Printed in U.S.A.

:NCE CODING SHEET

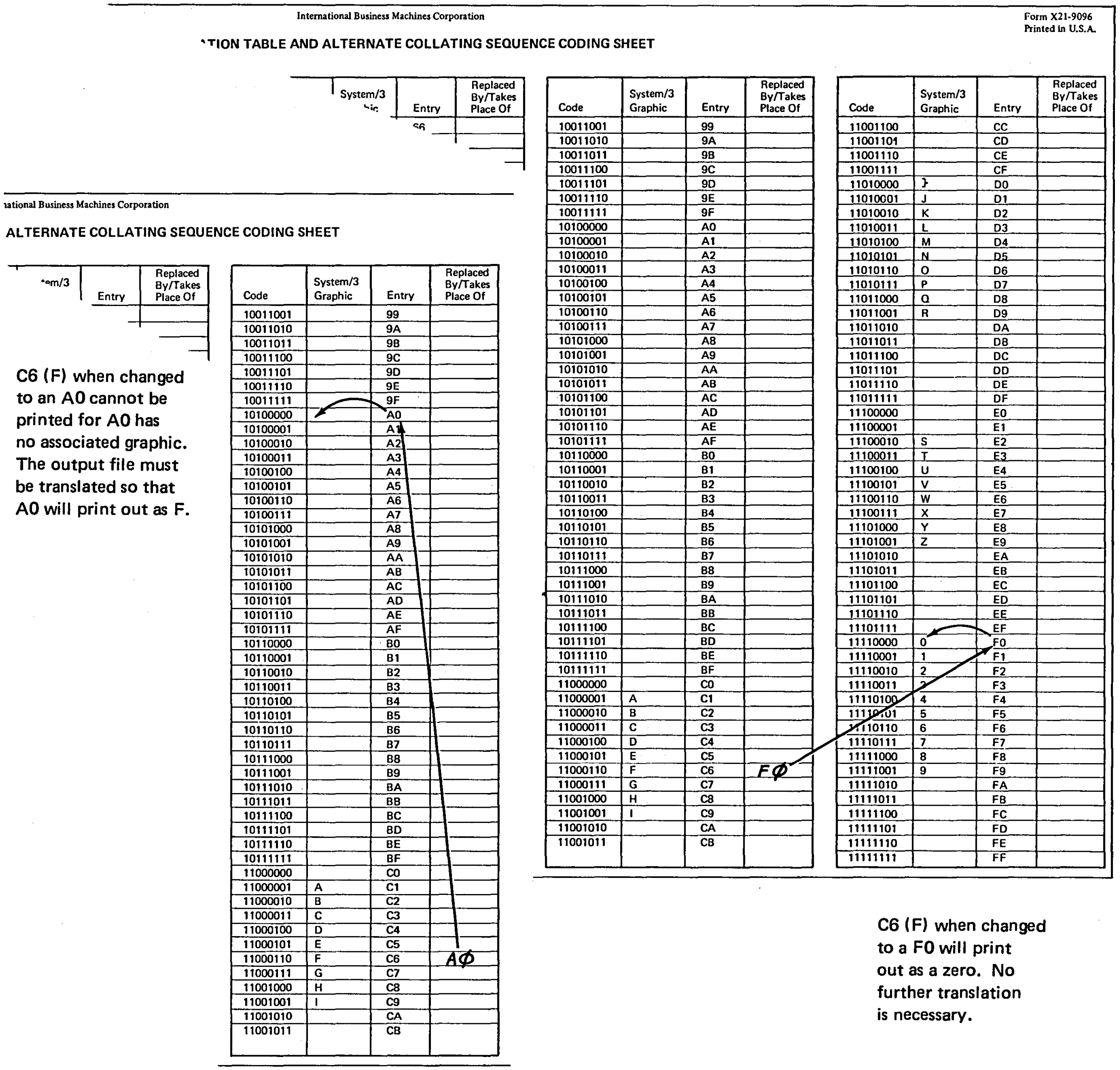
Code	System/3 Graphic	Entry	Replaced By/Takes Place Of	Code	System/3 Graphic	Entry	Replaced By/Takes Place Of
10011001		99		11001100		CC	
10011010		9A		11001101		CD	
10101011		AB		11011110		DE	
10101100		AC		11011111		DF	
10101101		AD		11100000		E0	
10101110		AE		11100001		E1	
10101111		AF		11100010	S	E2	
10110000		B0		11100011	T	E3	F2
10110001		B1		11100100	U	E4	
10110010		B2		11100101	V	E5	
10110011		B3		11100110	W	E6	
10110100		B4		11100111	X	E7	
10110101		B5		11101000	Y	E8	
10110110		B6		11101001	Z	E9	F3
10110111		B7		11101010		EA	
10111000		B8		11101011		EB	
10111001		B9		11101100		EC	
10111010		BA		11101101		ED	
10111011		BB		11101110		EE	
10111100		BC		11101111		EF	
10111101		BD		11110000	0	F0	
10111110		BE		11110001	1	F1	
10111111		BF		11110010	2	F2	
11000000		C0		11110011	3	F3	
11000001	A	C1		11110100	4	F4	
11000010	B	C2		11110101	5	F5	
11000011	C	C3		11110110	6	F6	
11000100	D	C4		11110111	7	F7	
11000101	E	C5	F5	11111000	8	F8	
11000110	F	C6	FØ	11111001	9	F9	
11000111	G	C7	F4	11111010		FA	
11001000	H	C8		11111011		FB	
11001001	I	C9	F1	11111100		FC	
11001010		CA		11111101		FD	
11001011		CB		11111110		FE	
				11111111		FF	

File Translation specifications used for translating both input and output files

Information read in is translated.

Information is translated before being punched or printed.

Figure 16-46. Summary of File Translation



International Business Machines Corporation

Form X21-9096
Printed in U.S.A.

TION TABLE AND ALTERNATE COLLATING SEQUENCE CODING SHEET

Code	System/3 Graphic	Entry	Replaced By/Takes Place Of
10011001		99	
10011010		9A	
10011011		9B	
10011100		9C	
10011101		9D	
10011110		9E	
10011111		9F	
10100000		A0	
10100001		A1	
10100010		A2	
10100011		A3	
10100100		A4	
10100101		A5	
10100110		A6	
10100111		A7	
10101000		A8	
10101001		A9	
10101010		AA	
10101011		AB	
10101100		AC	
10101101		AD	
10101110		AE	
10101111		AF	
10110000		B0	
10110001		B1	
10110010		B2	
10110011		B3	
10110100		B4	
10110101		B5	
10110110		B6	
10110111		B7	
10111000		B8	
10111001		B9	
10111010		BA	
10111011		BB	
10111100		BC	
10111101		BD	
10111110		BE	
10111111		BF	
11000000		C0	
11000001	A	C1	
11000010	B	C2	
11000011	C	C3	
11000100	D	C4	
11000101	E	C5	
11000110	F	C6	FØ
11000111	G	C7	
11001000	H	C8	
11001001	I	C9	
11001010		CA	
11001011		CB	

Code	System/3 Graphic	Entry	Replaced By/Takes Place Of
11001100		CC	
11001101		CD	
11001110		CE	
11001111		CF	
11010000	}	D0	
11010001	J	D1	
11010010	K	D2	
11010011	L	D3	
11010100	M	D4	
11010101	N	D5	
11010110	O	D6	
11010111	P	D7	
11011000	Q	D8	
11011001	R	D9	
11011010		DA	
11011011		DB	
11011100		DC	
11011101		DD	
11011110		DE	
11011111		DF	
11100000		E0	
11100001		E1	
11100010	S	E2	
11100011	T	E3	
11100100	U	E4	
11100101	V	E5	
11100110	W	E6	
11100111	X	E7	
11101000	Y	E8	
11101001	Z	E9	
11101010		EA	
11101011		EB	
11101100		EC	
11101101		ED	
11101110		EE	
11101111		EF	
11110000	0	F0	
11110001	1	F1	
11110010	2	F2	
11110011	3	F3	
11110100	4	F4	
11110101	5	F5	
11110110	6	F6	
11110111	7	F7	
11111000	8	F8	
11111001	9	F9	
11111010		FA	
11111011		FB	
11111100		FC	
11111101		FD	
11111110		FE	
11111111		FF	

C6 (F) when changed to a F0 will print out as a zero. No further translation is necessary.

ational Business Machines Corporation

ALTERNATE COLLATING SEQUENCE CODING SHEET

C6 (F) when changed to an A0 cannot be printed for A0 has no associated graphic. The output file must be translated so that A0 will print out as F.

Code	System/3 Graphic	Entry	Replaced By/Takes Place Of
10011001		99	
10011010		9A	
10011011		9B	
10011100		9C	
10011101		9D	
10011110		9E	
10011111		9F	
10100000		A0	
10100001		A1	
10100010		A2	
10100011		A3	
10100100		A4	
10100101		A5	
10100110		A6	
10100111		A7	
10101000		A8	
10101001		A9	
10101010		AA	
10101011		AB	
10101100		AC	
10101101		AD	
10101110		AE	
10101111		AF	
10110000		B0	
10110001		B1	
10110010		B2	
10110011		B3	
10110100		B4	
10110101		B5	
10110110		B6	
10110111		B7	
10111000		B8	
10111001		B9	
10111010		BA	
10111011		BB	
10111100		BC	
10111101		BD	
10111110		BE	
10111111		BF	
11000000		C0	
11000001	A	C1	
11000010	B	C2	
11000011	C	C3	
11000100	D	C4	
11000101	E	C5	
11000110	F	C6	AØ
11000111	G	C7	
11001000	H	C8	
11001001	I	C9	
11001010		CA	
11001011		CB	

Figure 16-47. Printable Graphics

Punched Cards for the Translation Table

After you have written all specifications for file translation, you can record them on cards so that they can be entered into the system. Cards containing these specifications must be formatted as follows:

Columns	*Entry*
1-6	*FILES
or	
1-8	a filename
7-8	Blank if not required
9-96	Numerical values involved in translating characters

If all files (both input and output) are to be translated, use the entry *FILES in columns 1 through 6. If only one file is translated, use that filename in columns 1 through 8. If several, but not all files, are to be translated, you must format separate cards for each file.

In columns 9 through 96, there are 22 groups of four columns. Each group (9-12, 13-16, etc.) must contain two hexadecimal values involved in the translating of *one* character. The first two columns of the group are for the hexadecimal value taken from the *Entry* column of the Translation Table and Alternate Collating Sequence Coding Sheet. The last two columns are for the hexadecimal value taken from the *Replaced By* column of the coding sheet.

More than one card can be used to specify the characters which must be translated. However, each additional card must be formatted in the same way as the first. All cards for one file must be grouped together. An error will occur if four cards are entered in the following order:

1. FILEA
2. FILEA
3. FILEB
4. FILEA

Also, the first blank appearing in columns 9 through 96 is recognized by the computer as the end of the translation specifications. Consequently blanks should not appear between pairs of hexadecimal values.

Two additional cards must be included along with the file translation table cards. One card containing **ƀ (two asterisks and a blank) in columns 1 through 3 must precede the file translation cards. The other card, containing /*ƀ (slash, asterisk, and a blank) in columns 1 through 3, follows the translation table cards.

All cards used for file translation except the RPG II control card must follow RPG II input, calculation, and output specifications and must precede any tables or alternate collating sequence cards used.

1. Into what two portions may every card column and every byte in storage be divided?

2. Do all characters that have an *A* zone punched in the zone portion of a card have the same zone representation in storage? Why or why not?

3. Calculate the numerical value of each of the following binary numbers as recorded in one byte of storage:

   a. 11000100.

   b. 11010101.

   c. 11101000.

   d. 11110011.

4. Express the numerical value of the bytes shown in Question 4 as a pair of numbers (hexadecimal value), rather than as a single value.

5. What does the computer use to determine the collating sequence of characters?

6. Arrange the following characters in ascending collating sequence. Arrange the same characters in ascending collating sequence by zone and digit.

Character	Hexadecimal Value	Numerical Value
C	C3	195
/	61	97
P	D7	215
J	D1	209
*	5C	92
T	E3	227
R	D9	217
4	F4	245
&	50	80
9	F9	249
0	F0	240

7. Fill in the Alternate Collating Sequence Coding Sheet to:

   a. Insert a *Ü* between *U* and *V* (use the # sign to represent *U*).

   b. Make a blank fall in the same sequence as zero.

   Show how this information would be punched into a card.

8. In what RPG II operations is the alternate collating sequence used?

9. Where is the sign located in a numeric field?

10. The TESTZ operation checks the zones of:

    a. any position in a field.

    b. only the low order position in the field.

    c. only the high-order position in a field.

11. A field may be alphameric for *any* move zone operations. Check those fields (Factor 2, Result) which *can be* numeric for the following move zone operations:

	Operation	Factor 2	Result
a.	MHLZO		
b.	MLHZO		
c.	MLLZO		
d.	MHHZO		

12. Code the calculation specifications to make the contents of a positive numeric AMTDUE field negative.

13. What is the difference between the way the computer works with characters involved in an alternate collating sequence and the way it works with characters involved in file translation?

14. Fill in the coding forms to translate *A*'s to *1*'s and *B*'s to *3*'s. Show how these specifications would be punched in cards when all files are to be translated.

Answers to Review 16

1. Zone and digit.

2. All characters which have the same zone punch in a card do not necessarily have the same zone representation in storage. There are four zone bits for each character in storage and only two zone positions in a card column. Therefore a translation must take place when the character is read. The computer checks the entire punch combination (both zone and digit) of a character to determine which bits are turned on or off in order to represent the character in storage.

3. a. 196

```
    1   1 0 0       0 1 0 0
    128+64+0+0      0+4+0+0 = 196
```

   b. 213

   c. 232

   d. 243

4. a. C4

```
        1 1 0 0     0 1 0 0
    C = 8+4+0+0     0+4+0+0 = 4
```

   b. D5

   c. E8

   d. F3

5. The computer uses the numerical values associated with characters to determine the collating sequence of characters.

6. When characters are collated by zone and digit, they are collated in this order:

Character	Numerical Value
&	80
*	92
/	97
C	195
J	209
P	215
R	217
T	227
0	240
4	245
9	249

When characters are collated by zone the left half of the hexadecimal value is used to determine the order; when collating by digit the right half of the hexadecimal value is used. When characters are collated by zone or digit, several may hold the same position in the sequence and thus belong in the same group. Within that group they may hold any position.

Characters collated by zone are in this order:

Character	Hexadecimal Value Used
& } *	50
* }	5C
/	61
C	C3
P }	D7
J } *	D1
R }	D9
T	E3
4 }	F4
9 } *	F9
0 }	F0

* Characters within brackets may be in any order since they are in the same group.

Characters collated by digit are in this order:

Character	Hexadecimal Value Used
0 } *	F0
& }	50
/ } *	61
J }	D1
C } *	C3
T }	E3
4	F4
P	D7
R } *	D9
9 }	F9
*	5C

* Characters within brackets may be in any order since they are in the same group.

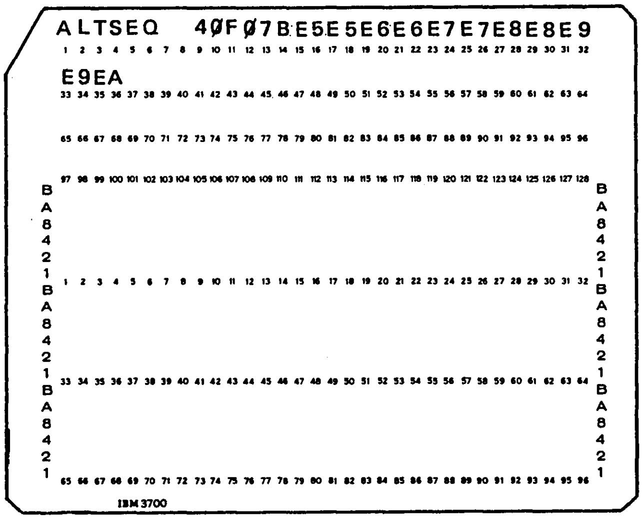
7.

International Business Machines Corporation

Form X21-9096
Printed in U.S.A.

TRANSLATION TABLE AND ALTERNATE COLLATING SEQUENCE CODING SHEET

Code	System/3 Graphic	Entry	Replaced By/Takes Place Of	Code	System/3 Graphic	Entry	Replaced By/Takes Place Of	Code	System/3 Graphic	Entry	Replaced By/Takes Place Of	Code	System/3 Graphic	Entry	Replaced By/Takes Place Of
00110011		33		01100110		66		10011001		99		11001100		CC	
00110100		34		01100111		67		10011010		9A		11001101		CD	
00110101		35		01101000		68		10011011		9B		11001110		CE	
00110110		36		01101001		69		10011100		9C		11001111		CF	
00110111		37		01101010		6A		10011101		9D		11010000	}	D0	
00111000		38		01101011	,	6B		10011110		9E		11010001	J	D1	
00111001		39		01101100	%	6C		10011111		9F		11010010	K	D2	
00111010		3A		01101101	–	6D		10100000		A0		11010011	L	D3	
00111011		3B		01101110	>	6E		10100001		A1		11010100	M	D4	
00111100		3C		01101111	?	6F		10100010		A2		11010101	N	D5	
00111101		3D		01110000		70		10100011		A3		11010110	O	D6	
00111110		3E		01110001		71		10100100		A4		11010111	P	D7	
00111111		3F		01110010		72		10100101		A5		11011000	Q	D8	
01000000	Blank	40	FØ	01110011		73		10100110		A6		11011001	R	D9	
01000001		41		01110100		74		10100111		A7		11011010		DA	
01000010		42		01110101		75		10101000		A8		11011011		DB	
01000011		43		01110110		76		10101001		A9		11011100		DC	
01000100		44		01110111		77		10101010		AA		11011101		DD	
01000101		45		01111000		78		10101011		AB		11011110		DE	
01000110		46		01111001		79		10101100		AC		11011111		DF	
01000111		47		01111010		7A		10101101		AD		11100000		E0	
01001000		48		01111011	#	7B	E5	10101110		AE		11100001		E1	
01001001		49		01111100	@	7C		10101111		AF		11100010	S	E2	
01001010	¢	4A		01111101	'	7D		10110000		B0		11100011	T	E3	
01001011	.	4B		01111110	=	7E		10110001		B1		11100100	U	E4	
01001100	<	4C		01111111	"	7F		10110010		B2		11100101	V	E5	E6
01001101	(	4D		10000000		80		10110011		B3		11100110	W	E6	E7
01001110	+	4E		10000001		81		10110100		B4		11100111	X	E7	E8
01001111	|	4F		10000010		82		10110101		B5		11101000	Y	E8	E9
01010000	&	50		10000011		83		10110110		B6		11101001	Z	E9	EA
01010001		51		10000100		84		10110111		B7		11101010		EA	
01010010		52		10000101		85		10111000		B8		11101011		EB	
01010011		53		10000110		86		10111001		B9		11101100		EC	
01010100		54		10000111		87		10111010		BA		11101101		ED	
01010101		55		10001000		88		10111011		BB		11101110		EE	
01010110		56		10001001		89		10111100		BC		11101111		EF	
01010111		57		10001010		8A		10111101		BD		11110000	0	F0	
01011000		58		10001011		8B		10111110		BE		11110001	1	F1	
01011001		59		10001100		8C		10111111		BF		11110010	2	F2	
01011010	!	5A		10001101		8D		11000000		C0		11110011	3	F3	
01011011	$	5B		10001110		8E		11000001	A	C1		11110100	4	F4	
01011100	*	5C		10001111		8F		11000010	B	C2		11110101	5	F5	
01011101	)	5D		10010000		90		11000011	C	C3		11110110	6	F6	
01011110	:	5E		10010001		91		11000100	D	C4		11110111	7	F7	
01011111	¬	5F		10010010		92		11000101	E	C5		11111000	8	F8	
01100000	-	60		10010011		93		11000110	F	C6		11111001	9	F9	
01100001	/	61		10010100		94		11000111	G	C7		11111010		FA	
01100010		62		10010101		95		11001000	H	C8		11111011		FB	
01100011		63		10010110		96		11001001	I	C9		11111100		FC	
01100100		64		10010111		97		11001010		CA		11111101		FD	
01100101		65		10011000		98		11001011		CB		11111110		FE	
												11111111		FF	



ALTSEQ 4ØFØ7BE5E5E6E6E7E7E8E8E9

E9EA

IBM 3700

8. An alternate collating sequence is used only for compare operations and matching or sequence checking operations done on match fields.

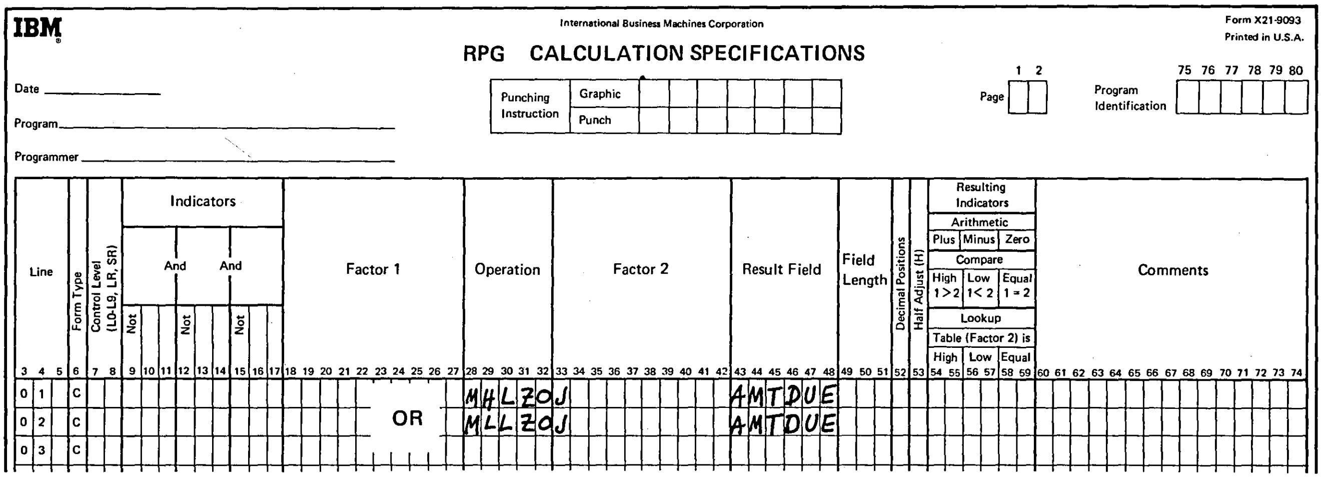
9. The sign of a numeric field must be in the low-order (rightmost) position of the low-order byte.

10. C.

11. *Factor 2* *Result Field*

	Factor 2	*Result Field*
a.		X
b.	X	
c.	X	X
d.		

12.



IBM International Business Machines Corporation Form X21-9093 Printed in U.S.A.

RPG CALCULATION SPECIFICATIONS

Date ____ Program ____ Programmer ____

Punching Instruction: Graphic ____ Punch ____

Page (1 2) ____ Program Identification (75 76 77 78 79 80) ____

Line (3-5)	Form Type (6)	Control Level (L0-L9, LR, SR) (7-8)	Indicators: Not (9)	(10-11)	And: Not (12)	(13-14)	And: Not (15)	(16-17)	Factor 1 (18-27)	Operation (28-32)	Factor 2 (33-42)	Result Field (43-48)	Field Length (49-51)	Decimal Positions (52)	Half Adjust (H) (53)	Resulting Indicators: Arithmetic Plus / Compare High 1>2 / Lookup Table (Factor 2) is High (54-55)	Minus / Low 1<2 / Low (56-57)	Zero / Equal 1=2 / Equal (58-59)	Comments (60-74)
01	C									MHLZO	J	AMTDUE							
02	C								OR	MLLZO	J	AMTDUE							
03	C																		

13. In file translation a character is actually translated into another character because the computer changes bit combinations. All affected characters are changed for the entire job. The bit combinations for characters involved in an alternate collating sequence are not changed. Bit combinations are substituted for others during sequencing operations only.

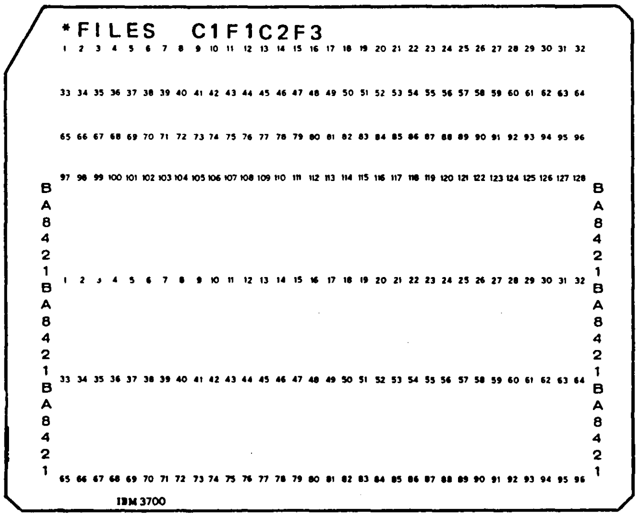
14.

International Business Machines Corporation

Form X21-9096
Printed in U.S.A.

TRANSLATION TABLE AND ALTERNATE COLLATING SEQUENCE CODING SHEET

Code	System/3 Graphic	Entry	Replaced By/Takes Place Of
00110011		33	
00110100		34	
00110101		35	
00110110		36	
00110111		37	
00111000		38	
00111001		39	
00111010		3A	
00111011		3B	
00111100		3C	
00111101		3D	
00111110		3E	
00111111		3F	
01000000	Blank	40	
01000001		41	
01000010		42	
01000011		43	
01000100		44	
01000101		45	
01000110		46	
01000111		47	
01001000		48	
01001001		49	
01001010	¢	4A	
01001011	.	4B	
01001100	<	4C	
01001101	(	4D	
01001110	+	4E	
01001111	|	4F	
01010000	&	50	
01010001		51	
01010010		52	
01010011		53	
01010100		54	
01010101		55	
01010110		56	
01010111		57	
01011000		58	
01011001		59	
01011010	!	5A	
01011011	$	5B	
01011100	*	5C	
01011101	)	5D	
01011110	:	5E	
01011111	¬	5F	
01100000	-	60	
01100001	/	61	
01100010		62	
01100011		63	
01100100		64	
01100101		65	
01100110		66	
01100111		67	
01101000		68	
01101001		69	
01101010		6A	
01101011	,	6B	
01101100	%	6C	
01101101	–	6D	
01101110	>	6E	
01101111	?	6F	
01110000		70	
01110001		71	
01110010		72	
01110011		73	
01110100		74	
01110101		75	
01110110		76	
01110111		77	
01111000		78	
01111001		79	
01111010	:	7A	
01111011	#	7B	
01111100	@	7C	
01111101	'	7D	
01111110	=	7E	
01111111	"	7F	
10000000		80	
10000001		81	
10000010		82	
10000011		83	
10000100		84	
10000101		85	
10000110		86	
10000111		87	
10001000		88	
10001001		89	
10001010		8A	
10001011		8B	
10001100		8C	
10001101		8D	
10001110		8E	
10001111		8F	
10010000		90	
10010001		91	
10010010		92	
10010011		93	
10010100		94	
10010101		95	
10010110		96	
10010111		97	
10011000		98	
10011001		99	
10011010		9A	
10011011		9B	
10011100		9C	
10011101		9D	
10011110		9E	
10011111		9F	
10100000		A0	
10100001		A1	
10100010		A2	
10100011		A3	
10100100		A4	
10100101		A5	
10100110		A6	
10100111		A7	
10101000		A8	
10101001		A9	
10101010		AA	
10101011		AB	
10101100		AC	
10101101		AD	
10101110		AE	
10101111		AF	
10110000		B0	
10110001		B1	
10110010		B2	
10110011		B3	
10110100		B4	
10110101		B5	
10110110		B6	
10110111		B7	
10111000		B8	
10111001		B9	
10111010		BA	
10111011		BB	
10111100		BC	
10111101		BD	
10111110		BE	
10111111		BF	
11000000		C0	
11000001	A	C1	F1
11000010	B	C2	F3
11000011	C	C3	
11000100	D	C4	
11000101	E	C5	
11000110	F	C6	
11000111	G	C7	
11001000	H	C8	
11001001	I	C9	
11001010		CA	
11001011		CB	
11001100		CC	
11001101		CD	
11001110		CE	
11001111		CF	
11010000	}	D0	
11010001	J	D1	
11010010	K	D2	
11010011	L	D3	
11010100	M	D4	
11010101	N	D5	
11010110	O	D6	
11010111	P	D7	
11011000	Q	D8	
11011001	R	D9	
11011010		DA	
11011011		DB	
11011100		DC	
11011101		DD	
11011110		DE	
11011111		DF	
11100000		E0	
11100001		E1	
11100010	S	E2	
11100011	T	E3	
11100100	U	E4	
11100101	V	E5	
11100110	W	E6	
11100111	X	E7	
11101000	Y	E8	
11101001	Z	E9	
11101010		EA	
11101011		EB	
11101100		EC	
11101101		ED	
11101110		EE	
11101111		EF	
11110000	0	F0	
11110001	1	F1	
11110010	2	F2	
11110011	3	F3	
11110100	4	F4	
11110101	5	F5	
11110110	6	F6	
11110111	7	F7	
11111000	8	F8	
11111001	9	F9	
11111010		FA	
11111011		FB	
11111100		FC	
11111101		FD	
11111110		FE	
11111111		FF	



*FILES C1F1C2F3

B
A
8
4
2
1
IBM 3700

Index

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
[U.S.A. only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

READER'S COMMENT FORM

IBM System/3
Disk System
RPG II and System Additional Topics
Programm's Guide

GC21-7511-0

Your answers to the questions on this sheet will help us produce better manuals for your use. If any of your answers require comments, or if you have additional information you think would be helpful, please use the space provided. All comments and suggestions become the property of IBM.

	Yes	No
1. Is the manual easy to read?		
2. Is any of the information unclear?		
3. Is additional information needed?		
4. Is any of the information unnecessary?		
5. Did you read the Preface?		
6. Did you use the Table of Contents?		
7. Did you use the Index? *		
8. Did you take the tests? *		

* Not included in all manuals

9. How did you use the manual:

Instructor in a class ________
Student in a class ________
Reference material ________
Self-Training ________
Other (Explain) ________

Have you had previous computer or programming training? ________

What is your present job? ________

What business is your company engaged in? ________

COMMENTS

- Thank you for your cooperation. No postage necessary if mailed in the U.S.A.

GC21-7511-0

YOUR COMMENTS, PLEASE...

Your answers to the questions on the back of this form, together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Cut Along Line

Fold Fold

FIRST CLASS
PERMIT NO. 387
ROCHESTER, MINN.

BUSINESS REPLY MAIL

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY . . .

IBM Corporation
General Systems Division
Development Laboratory
Rochester, Minnesota 55901

Attention: Programming Publications, Dept. 425

Fold Fold

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
[U.S.A. only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

READER'S COMMENT FORM

IBM System/3
Disk System
RPG II and System Additional Topics
Programm's Guide

GC21-7511-0

Your answers to the questions on this sheet will help us produce better manuals for your use. If any of your answers require comments, or if you have additional information you think would be helpful, please use the space provided. All comments and suggestions become the property of IBM.

	Yes	No
1. Is the manual easy to read?		
2. Is any of the information unclear?		
3. Is additional information needed?		
4. Is any of the information unnecessary?		
5. Did you read the Preface?		
6. Did you use the Table of Contents?		
7. Did you use the Index? *		
8. Did you take the tests? *		

* Not included in all manuals

9. How did you use the manual:

Instructor in a class ________
Student in a class ________
Reference material ________
Self-Training ________
Other (Explain) ________

Have you had previous computer or programming training? ________

What is your present job? ________

What business is your company engaged in? ________

COMMENTS

● Thank you for your cooperation. No postage necessary if mailed in the U.S.A.

YOUR COMMENTS, PLEASE...

Your answers to the questions on the back of this form, together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Cut Along Line

Fold Fold

FIRST CLASS
PERMIT NO. 387
ROCHESTER, MINN.

BUSINESS REPLY MAIL
NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY . . .

IBM Corporation
General Systems Division
Development Laboratory
Rochester, Minnesota 55901

Attention: Programming Publications, Dept. 425

Fold Fold

IBM®

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
[U.S.A. only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

GC21-7511-0

IBM System/3 Printed in U.S.A. GC21-7511-0

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
[U.S.A. only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]