UNIVAC

DATA PROCESSING DIVISION

1005

SYSTEM

ASSEMBLER-80

PROGRAMMERS REFERENCE

UP-4084

This manual is published by the UNIVAC Division of Sperry Rand Corporation in loose leaf format as a rapid and complete means of keeping recipients apprised of UNIVAC ® Systems developments. The UNIVAC Division will issue updating packages, utilizing primarily a page-for-page or unit replacement technique. Such issuance will provide notification of hardware and/or software changes and refinements. The UNIVAC Division reserves the right to make such additions, corrections, and/or deletions as in the judgment of the UNIVAC Division, are required by the development of its respective Systems.

PRINTED IN U.S.A.

CONTENTS

UNIVAC 1005 System
ASSEMBLER-80 Programmer's Reference
UP-4084

August 23, 1966

UPDATING PACKAGE "A"

UNIVAC 1005 System P.I.E. Bulletin 4, UP-4072.4, releases and announces the availability of Updating Package "A" for the "UNIVAC 1005 SYSTEM ASSEMBLER-80 Programmer's Reference," UP-4084, 59 pages plus 1 Updating Summary Sheet. This material should be utilized in the following manner:

SECTION	DESTROY FORMER PAGES NUMBERED	FILE NEW PAGE NUMBERED
1	1 & 2	1* & 2 Rev.1
	3 & 4	3 Rev.1 & 4*
3	9 & 10	9 Rev.1 & 10*
	13 & 14	13* & 14 Rev.1
4	1 & 2	1 Rev.1 & 2*
	5 & 6	5 Rev.1 & 6*
	7 & 8	7 Rev.1 & 8 Rev.1
	11 & 12	11* & 12 Rev.1
	15 & 16	15 Rev.1 & 16 Rev.1
	19 & 20	19* & 20 Rev.1
	21 & 22	21 Rev.1 & 22*
	29 & 30	29 Rev.1 & 30*
	35 & 36	35* & 36 Rev.1
	37 & 38	37 Rev.1 & 38*
	39 & 40	39 Rev.1 & 40*
	41 & 42	41 Rev.1 & 42*
	51 & 52	51 Rev.1 & 52*
	53 & 54	53* & 54 Rev.1
	55 & 56	55 Rev.1 & 56*
	61 & 62	61 Rev.1 & 62 Rev.1
	63 & 64	63 Rev.1 & 64 Rev.1
	65 & 66	65 Rev.1 & 66*
	67 & 68	67 Rev.1 & 68*
	73 & 74	73 Rev.1 & 74 Rev.1
	81 & 82	81 Rev.1 & 82 Rev.1
	83 & 84	83 Rev.1 & 84*
	85 & 86	85 Rev.1 & 86*
	97 & 98	97 Rev.1 & 98*
6	1 & 2	1 Rev.1 & 2*
	3	3 Rev.1

*These pages, backups of revised pages, remain unchanged.

1. INTRODUCTION TO THE UNIVAC 1005 AND INTERNALLY STORED PROGRAMMING

The UNIVAC 1005 is a general purpose, stored program, digital computer. The main store consists of either two or four banks of core memory with 1024 locations per bank. In addition to providing storage for instructions and data, two types of Special Registers are provided in core memory to control the operation of the UNIVAC 1005. The special registers are addressable and in some cases can be used as additional data storage.

1.1. ADDRESSING TECHNIQUE

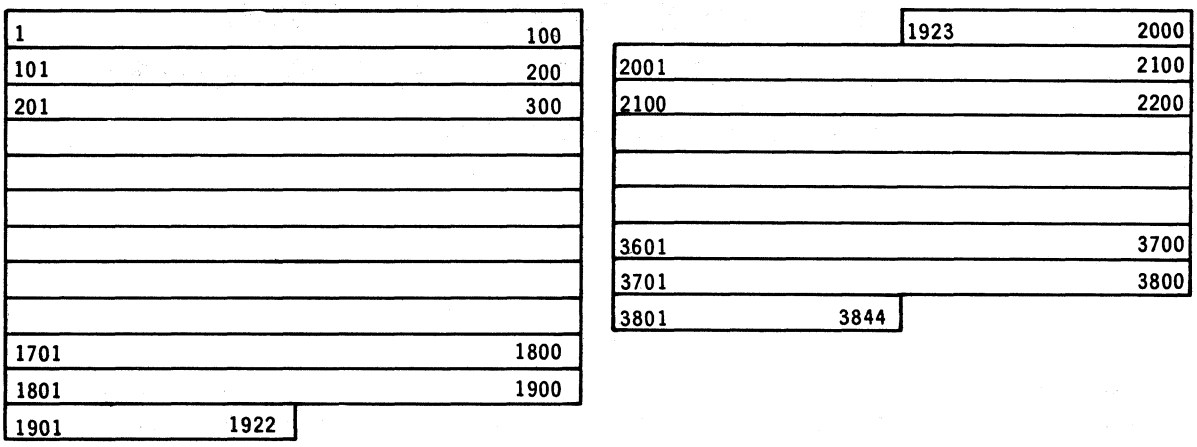
Each bank of core memory consists of a 32 row by 32 column matrix of six-bit memory locations. Each location is addressed by specifying its Row and Column coordinates. For example; the first memory location has an address of Row 1, Column 1; the last memory location has an address of Row 32, Column 32. These address designations are abbreviated to R1/C1 and R32/C32.

In order to store the address of instructions and data in memory, the six bits of two adjacent memory locations are required. Five of the six bits of the left-hand location are used to specify the Row coordinate, and five of the six bits of the right-hand location are used to specify the Column coordinate. A combination of the sixth bits of both locations is used to specify which of the four possible banks of memory is involved.

The UNIVAC 1005 utilizes a special five-bit address concept which operates on a logical rather than a binary arithmetic basis. Special combinations of these five bits are employed for the values 1 to 31 used as row or column coordinates. These five-bit combinations, plus the sixth bit required for bank designation, correspond to the 64 characters of the UNIVAC 1005. Thus, the address of any location in any bank of core memory can be specified by the proper selection of two of these six-bit characters.

The foregoing description indicates the similarity and compatibility of the memory of the UNIVAC 1005 and the UNIVAC 1004.

For the purpose of stored-programming, the main store of the UNIVAC 1005 should be considered as 1922 (or 3844) consecutively numbered, decimally addressed memory locations exclusive of all rows numbered 32 and all Columns numbered 32 of each row. The physical arrangement of main store is then no longer a concern of the programmer. Using this method, the main store of the UNIVAC 1005 takes on the following appearance:



BANKS 1 and 2

BANKS 3 and 4

Decimal addressing would then produce the following facility for programming:

	DECIMAL ADDRESS
First location of Read Input Storage (Card Col. 1)	1
Last location of Read Input Storage (Card Col. 80)	80
First location of Print Storage (Print Pos. 1)	161
Last location of Print Storage (Print Pos. 132)	292
First location of Punch Storage (Card Col. 1)	293
Last location of Punch Storage (Card Col. 80)	372

Since Print Position 1 is located at address 161, Print Position 23, for example, is located 22 positions away at address 183. This same convenience is extended to the addressing of the programmer's data areas as well as to the other reserved areas of Input Output storage. A complete description of decimal addressing as provided by the UNIVAC 1005 Assembly System appears in Section 3.3.3. of this manual.

1.2. BASIC LOGIC AND FORMAT

The UNIVAC 1005 operates on a basic two address instruction logic. A UNIVAC 1005 instruction may contain the address of one or two units of information called Operands, and an operation code for the process to be performed with these Operands. The Operands are defined by specifying the address of the most significant location (abbreviated MSL), or the address of the least significant location (abbreviated LSL), or both, depending on the operation to be performed. Operations are specified by a one character code.

The majority of the UNIVAC 1005 instructions require seven character locations in the following format:

OP	A	B	C

OP is a single character which specifies the operation to be performed.
A is the two character address of the MSL or the LSL of Operand 1.
B is the two character address of the MSL of Operand 2.
C is the two character address of the LSL of Operand 2.

A, B or C may also represent constant information.

The function called for by an instruction is executed in ascending (LSL to MSL) or descending (MSL to LSL) mode. If the operation is performed in ascending mode, the A portion specifies the LSL of Operand 1. If the operation is performed in descending mode, the A portion specifies the MSL of Operand 1. The use of UNIVAC 1005 character codes to specify operation codes and addresses is called absolute coding or machine coding. A table of these codes and their equivalents is shown in Section 6. As will be explained in succeeding sections of this manual, the UNIVAC 1005 Assembly System provides a convenient method of specifying these codes. The output of the Assembler processing is a deck of punched cards containing instructions in machine code. These cards are read by the UNIVAC 1005 and the instructions stored in core memory under the control of a Load program. The Load program supplied by UNIVAC is a special header card which is placed in the front of the instruction cards.

1.3. CONTROL SECTION OPERATION

After the program has been loaded, operation of the UNIVAC 1005 proceeds under the control of the Instruction Control Counter (ICC). The ICC is one of the Special Registers located in core memory. The ICC is a two character register which contains the address of the MSL of the instruction to be accessed next by the UNIVAC 1005. As each instruction is accessed for execution, the contents of the ICC are incremented by the number of characters in the instruction--usually seven. This increment is automatically added to the right-hand character of the ICC, the Column address portion. When the Column count passes 31, an increment of one is added to the Row address portion of the ICC, and the Column portion is returned to 1. The ICC will never create an address of Row 32 or Column 32 because the increment of 1 to "31" produces a result of 1. Memory Bank specification is also advanced as the Row count passes 31. Thus,the access and execution of instructions proceeds in a sequential manner. Instructions are provided to vary this sequential operation when required.

As each instruction is accessed, according to the address in the ICC, it is transferred to the Instruction Register (IR). The IR is a seven character Special Register which is used to examine the instruction. The bit configuration of the single character Operation code is analyzed by the circuitry which establishes and controls the functions necessary to perform the required operation. The address portions of the IR are transferred to the internal storage address controls for OP 1 and OP 2.

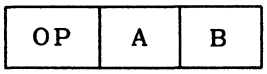
The normal operation of the UNIVAC 1005 when executing instructions is to access the first character from either the LSL or the MSL of OP 1, and the corresponding character from OP 2 (LSL or MSL) and perform the process. The determination of whether the LSL or the MSL is used is based on the mode--ascending or descending --of the instruction. After operating on the first characters from OP 1 and OP 2, the operation proceeds with successive corresponding characters of OP 1 and OP 2 until the processing of the last character location of OP 2 has been completed. This signals the end of the instruction.

During the execution of each instruction, the contents of the ICC are incremented according to the number of characters in the instruction itself. When the end operation signal is generated, the new address in the ICC is used to control the access of the next instruction (NI). The execution of the program proceeds in this manner to direct the UNIVAC 1005 to accomplish the desired results.

1.4. OTHER INSTRUCTION FORMATS

The instruction repertoire of the UNIVAC 1005 includes a complete set of commands for control of the operation of the system. In addition to the type of commands previously mentioned--seven characters, with an OP 1 and an OP 2 address--there is a second type of command which provides for flexibility and ease of control of the programming requirements necessary to internally stored program operation.

The second type of instruction is five characters in length and has the following format:



Where:

OP is a one character operation code.
A is a two character address or constant whose value is used or whose bit configuration is used to perform special functions.
B is a two character address of a location in memory.

As is the case with most stored program computers, there are special commands in the UNIVAC 1005 with a format that does not precisely conform to these two basic formats. The Set Condition Instruction is a unique exception to the general rule in that "B" is used for the normal purpose of "A." These special commands will be either five or seven characters in length, and the format variation will be indicated along with a complete description of the operation.

1.5. STORAGE ALLOCATION AND USE

The main store of the UNIVAC 1005 is separated *by the hardware* into two major areas--input output store, and working store. Input output store consists of selected portions of memory reserved for the information received from and transferred to the input/output devices. When an input or output device is not required by a particular

program, the reserved memory area of that device may be used by the programmer for storage of instructions or data. The working store consists of these portions of memory not reserved for input/output information--the remainder of core memory.

The working store is separated *by the programmer* into two types of areas according to his programming requirements. A portion of working store is established by the programmer to store the information (other than input/output) to be processed. The remainder of working store is used to store the program instructions.

A note should be made at this point that the addressing capability of the instruction address extends to *all* of main store. This means that the contents of any memory location or locations can be accessed as data to be processed. This includes those locations used by the programmer to store instructions. The use of this capability to operate on instructions as if they were data is an important technique of internally stored programming.

1.6. INDIRECT ADDRESSING

Another facility provided by the UNIVAC 1005 is the ability to perform Indirect Addressing. In the UNIVAC 1005, Indirect Addressing is the ability to specify in the address portion of an instruction the address of the memory location which contains not the data to be processed, but a secondary address which specifies the location of the data to be processed. From a hardware standpoint, this is accomplished by using the primary address--the address in the instruction--to control the transfer of the secondary address to the IR where it is then used as the address of the data to be processed.

From a programming standpoint, Indirect Addressing allows the programmer to establish one series of instructions which perform a programming operation on separately stored but related items of information. This is accomplished by programming the instructions once using primary addresses, and changing the secondary address to refer to the proper item for each use of the series of instructions. For example: a detail card contains four twenty-column transaction items. Each item contains four five-column fields. The program is written using primary addresses which refer to a table of secondary addresses. The table is set up to refer to the address of the fields within an item. A series of addresses in the form of constants is created by the programmer and stored in a portion of working store. There is a series of addresses for each of the transaction items as they will appear in Read Input Storage. Before processing the first transaction item, the program transfers the series of addresses which refer to this item, to the secondary address table locations referred to by the primary addresses in the instructions. The first item is then processed. At the conclusion of the processing of the first item, the program then transfers the series of addresses which refer to the second transaction to the table of secondary addresses. The series of instructions is then executed again, only this time, the table of secondary address references the locations of the fields of the second transaction item causing it to be processed. The program action of changing the contents of the secondary address table is then repeated for the processing of the third and fourth transaction items. An explanation of this example is shown in Figure 1–1.

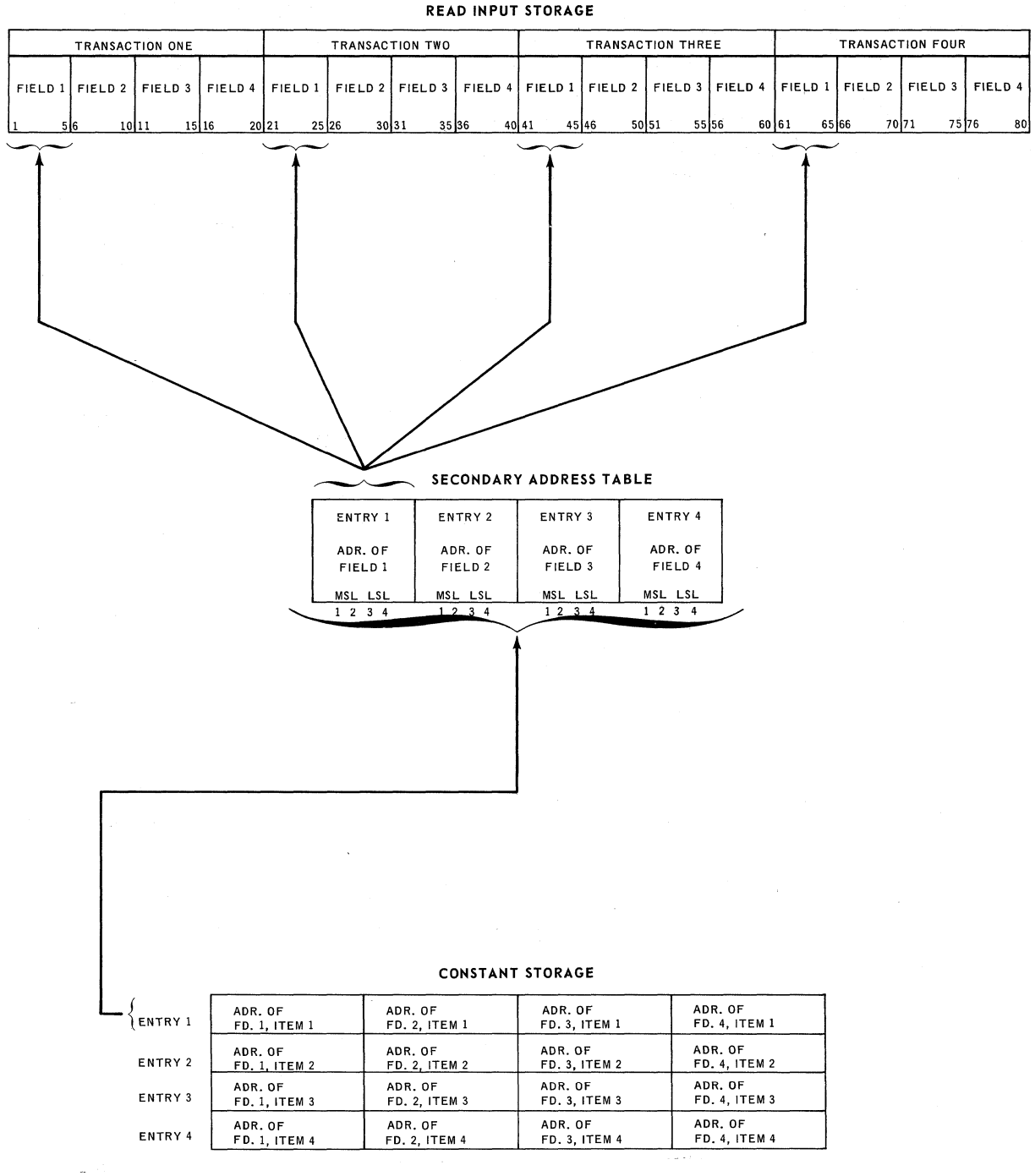


Figure 1–1. Example of Indirect Addressing

INSTRUCTIONS REQUIRED

The * indicates Indirect Addressing

	OPERATION	FIELD A	FIELD B	FIELD C	ACTION
Instr. 1	TRF-D	MSL of Entry 1 in constant storage	MSL of Sec Adr Table	LSL of Sec Adr Table	The MSL and LSL addresses of the Fields of Transaction One are transferred to the Sec Adr Table
2	ADD-ALG	*ADR of Loc 3 Entry 1, Sec Adr Table (LSL Field)	*ADR of Loc 1 Entry 2, Sec Adr (MSL Fd 2)	ADR of Loc 3 Entry 2 Sec Adr (LSL Fd 2)	The primary address in the A position of the instruction refers to the lower 2 characters of Entry 1 in the Sec Adr Table. These two characters are then used as the A address of this instruction. The primary addresses in the B and C positions similarly refer to the Sec Adr Table Entry 2, and address substitution occurs.
3	ADD-ALG	*ADR of Loc 3 Entry 2, Sec Adr (LSL Fd 2)	*ADR of Loc 1 Entry 3 Sec Adr (MSL Fd 3)	ADR of Loc 3 Entry 3 Sec Adr (LSL Fd 3)	Same as for Instruction 2 except that substitutes are made from Entry 2 and Entry 3.
4	SUB-ALG	*ADR of Loc 3 Entry 3, Sec Adr (LSL Fd 3)	*ADR of Loc 1 Entry 4, Sec Adr (MSL Fd 4)	ADR of Loc 3 Entry 4 Sec Adr (LSL Fd 4)	Same as for Instruction 3 except that substitutions are made from Entry 3 and Entry 4.
5	TRF-D	MSL of Entry 2, then 3, then 4, in constant storage	MSL of Sec Adr Table	LSL of Sec Adr Table	The addresses of the next item are set up in the Sec Adr Table.
6	Return to Instruction 2				Transfer control to instruction 2 to initiate processing of the next item.

NOTE: Additional programming techniques are necessary to control the number of times these instructions are to be executed for each card read. They are not involved with the use of Indirect Addressing, and were omitted in order to confine the example to the discussion. They will be covered along with the UNIVAC 1005 Operations which are used for the additional techniques. (See the JL and CC instructions if desired.)

Instructions 2, 3 and 4 will not have anything coded in Field C. The MSL and LSL to be used is defined in Field B.

The normal use of Indirect Addressing is with data which contains a multiple item format similar to the punched card format in the preceding example. The multiple item concept is generally used for magnetic tape master and detail files. The use of the Indirect Addressing capability of the UNIVAC 1005 is not restricted to the multiple item concept.

Other UNIVAC 1005 Operations and capabilities designed to implement stored programming techniques are described in this manual along with an example of the application of the technique.

1.7. OTHER SPECIAL REGISTERS

There are two Special Registers in addition to the Instruction Control Counter and the Instruction Register. These are the Multiply/Divide Register (MDR), and a multi-purpose register called Register X (rX).

The MDR is 31 locations in length, and is used in the process of Multiplication and Division. The most significant 10 positions of the MDR are used by the IR and the ICC during part of each Instruction Cycle. When not required for this purpose, the least significant 21 locations may be used for intermediate results of other programming operations.

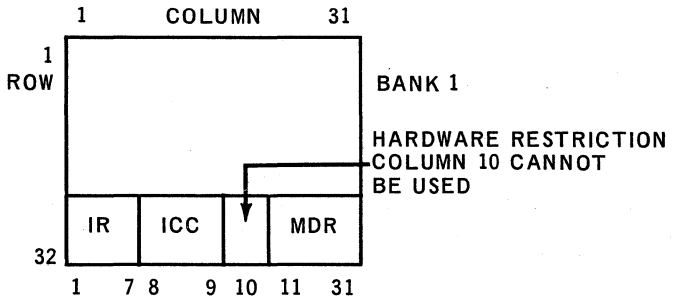
Register X is 31 locations in length and is used primarily with the Edit Mask Operation, otherwise it may be used when Add or Subtract operations involve Operands of unequal length.

A thorough explanation of the use of rX and the MDR is given in this manual where the Operations which reference them are discussed.

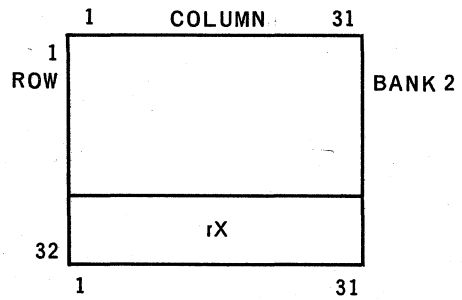
It should be remembered that all Special Registers are located in core memory, are explicitly addressable, and are in addition to the basic two or four banks of main store.

1.8. SPECIAL REGISTER LOCATIONS

The Instruction Register (IR), the Instruction Control Counter (ICC), and the Multiply/Divide Register (MDR) constitute an extra row of core memory--Row 32. This set of special registers is in Row 32 of *BANK 1*



Register X constitutes an extra row of core memory--Row 32 of *BANK 2*.



When the ICC is incremented for sequential access of instructions, it advances from R31/C31 of Bank 1, Bank 2, Bank 3, and Bank 4, to R1/C1 of Bank 2, Bank 3, Bank 4, and Bank 1 respectively, thus bypassing Row 32 and the special registers. The use of decimal addressing does not include the specification of the addresses of the Special Registers.

1.9. ADDRESSING AND USE OF COLUMN 32

Each of the 32 Rows of memory contains 32 columnar positions. The allocation of memory by the Assembler program is made on the basis of 31 Rows and 31 Columns per Bank. As described in Section 1.8, Row 32 of Bank 1, 2, 3, and Bank 4 are excluded from Assembler allocation and are used for the Special Registers. The explanation of the advance of the Row and Column portions of the Instruction Control Counter indicates that not only is Row 32 of memory bypassed, but also Column 32 of each Row of memory is also excluded from Assembler allocation.

Column 32 of each Row in Bank 2, 3, and 4 becomes a series of one character locations which can be used by the programmer for such things as single character constants, control settings, program switches, etc. Decimal addressing does not include the addressing of any Column 32.

NOTE: Column 32 of each of the Rows in *Bank 1* are reserved for hardware/software control purposes and must not be used by the programmer.

Row 32 of Bank 3 and Bank 4 are also not included in the allocation processing of the Assembler program. Row 32 of Bank 3 and Bank 4 can be used by the programmer for the storage of data and intermediate results of processing.

Control of an internally stored program computer is accomplished by providing the computer with a set of instructions which have been designed to produce the desired results. These instructions are created by the programmer according to the specific requirements and capabilities of the computer. The instructions must be entered in the computer memory in a specific sequence using a precise set of characters. This set of characters constitutes the vocabulary or language of the machine. Machine languages are dictated by the design characteristics of the computer and seldom bear any relationship to human language. Furthermore, machine languages seldom follow any logical pattern that a person could use when writing a program. In addition to the language barrier, there are many clerical-type functions which a programmer must perform when writing a program.

2. INTRODUCTION TO ASSEMBLY SYSTEMS

2.1. PURPOSE OF ASSEMBLY SYSTEMS

In order to overcome the language barrier, and to provide the programmer with clerical-type assistance, an assembly system is usually provided as part of the software of an internally stored program computer.

The assembly system allows the programmer to use a machine-oriented language which is also human oriented. The programmer writes the instructions in assembly language according to the rules of the assembly system. These instructions are punched into cards and are read into the computer under the control of a program called the assembler program. The assembler program analyzes the assembly language instructions and translates or converts this language into the precise machine language of the computer. An output deck of punched cards is produced by the assembler processing which contains the machine language instructions. This deck of cards is then read into the computer under the control of a load program which stores the instructions in the required sequence. The computer is then instructed to execute the program.

The deck of cards which contains the instructions written in assembly language is called the source deck. The output deck of cards which contains the instructions in machine language is called the object deck. In some cases, the assembly language is referred to as the source language or source code, and the machine language is referred to as the object language or object code.

2.2. MNEMONIC CODING

The terms mnemonic, symbolic, and relative coding are sometimes erroneously used as synonyms. Each term has a specific meaning, and each one constitutes an important characteristic of an assembler.

An assembly language usually contains a set of mnemonic codes which represent the Operation codes of the computer. These mnemonic codes are established to help the programmer remember the code to be used for the Operation needed.

2.3. SYMBOLIC CODING

Symbolic codes are a usual provision of an assembly language to allow the programmer to assign meaningful names to important information within his program. For example, the fields of data in a payroll card are known to the programmer by the type of information they contain, such as Employee Number, Department, Gross Pay, etc. There is normally a limitation on the length of a symbolic name. However, this length is usually enough to permit meaningful abbreviations or contractions. In the example above, the Employee Number field could be named EMPNO; the Department field could be named DEPT.; the Gross Pay field could be named GRPAY. As will be discussed in the section on memory mapping, the actual addresses of these fields in memory are assigned by the assembler program. When the name or label appears anywhere in the source language instructions, the assembler program will use the assigned actual address in the object language instruction.

Symbolic labels are also assigned to instructions in the program which are referenced by other instructions in the program. When a non-sequential transfer of control is required from one series of instructions to another, the programmer must specify the point to which control is to be transferred. Since actual addresses are assigned by the assembly program, the programmer cannot provide the *actual* address. By labelling the instruction to which control is to be transferred, he can use the symbolic address for the same purpose.

2.4. RELATIVE CODING

Relative coding is another assembly system technique which allows the programmer to specify the location of instructions and data, even though the actual addresses are assigned by the assembly program. To use the relative coding technique, the programmer must have a thorough understanding of the memory mapping operation of the assembler program. Once this is understood, relative coding is a simple yet powerful programming technique. In the preceding section on symbolic coding is an explanation of the assembler program assignment of addresses to symbolic labels. This creates a common fixed point of reference between the programmer and the assembler program. By using this common fixed point of reference (the label) as a base, the programmer can specify other locations by their position relative to the base. For example, if the memory location which is to contain the information from card column 1 has been given a label of DETCD, then the memory location which is to contain the information from card column 5 would be 4 locations away. By specifying an operand of DETCD + 4, the programmer causes the assembler program to assign the actual address of the operand by mathematically adding the increment of 4 to the actual address of DETCD. The assembly system usually provides for decrements to symbolic labels as well as increments.

2.5. MEMORY MAPPING

In order to assign the location of instructions and data, the assembler program must keep track of the locations that are used as the assembler processing is performed. To do this, the assembler program contains an Instruction Location Counter (ILC).

This is a program created device, not a piece of hardware. The loading of the assembler program itself usually sets the ILC to the actual address of the first memory location. The assembler program then causes the reading of the first source language instruction. This instruction is assigned to an actual address according to the present value of the ILC (in this case, the first memory location).

After the machine language for the instruction has been created by the assembler processing, a card is punched containing the machine language instruction. The *number* of locations that will be required to store the instruction is added by the assembler program to the value of the ILC creating a new value in the ILC. The assembler program then causes the next source language instruction card to be read. This instruction is assigned to an actual address according to the present value of the ILC. Assume that the actual address assigned to the preceding instruction had been R25/C1 and that the instruction was seven characters in length. (R25/C1 becomes the address of the MSL of the instruction.) The assembler program would then add seven (the number of locations for the first instruction) to the machine code equivalent stored in the ILC, and arrive at the machine code equivalent of R25/C8. This is the actual address assigned to the second instruction assembled.

The use of this procedure by the assembler program insures that the assignment of addresses to instructions follows the sequential access of the instructions by the computer when the object program is executed.

In addition to an ILC, an assembler program may contain a Data Location Counter (DLC). The DLC provides the programmer with the ability to assign locations to his data in an area of memory other than the area to be used for instructions. Instructions are usually assigned to locations starting with the first memory address (low-numbered locations) and proceeding in ascending sequence. Data is usually assigned to locations starting with the last memory address (high-numbered locations) and proceeding in descending sequence.

2.6. DECLARATIVE INSTRUCTIONS

An assembly system with an ILC and a DLC usually provides a set of pseudo-operations which allow the programmer to establish and modify the value in the location counters. In addition to the pseudo-operations which manipulate the ILC and the DLC, there are usually other pseudo-operations which are required to instruct the assembler program as to the manner in which the assembly processing is to take place. These pseudo-operations are called declarative instructions. Unlike the previously mentioned pseudo-operations, declarative instructions, which are included in the source language deck, do not produce instructions in the object language deck. The declarative instructions are for the use of the assembler program during the assembly processing, and not for the computer as part of the object program.

An example of a declarative instruction would be one that updates the DLC. Assume the problem called for storing the contents of a header card to print headings on each new page. The programmer would label the source language instruction

line, write the mnemonic pseudo-operation code that decrements the DLC, and indicate the value of the decrement (the number of locations to be reserved for the data). Such a line of source code might appear as

Label	OP Code	# of Locations
HDRCD	DA	80

(stands for Define Area)

When this line of source coding is encountered, the DA tells the assembler program to refer to the DLC. The contents of the DLC are decremented by the number of locations to be reserved. The new value of the DLC is assigned as the address of HDRCD. In order to reference the information in the HDRCD area, the programmer can use relative coding.

2.7. ASSEMBLER PROCESSING

An assembler program consists of a set of machine code instructions designed to produce specific results. As is the case with any computer program, the assembler program is designed to receive specific information prepared in a precise format. It is the responsibility of the programmer to prepare the source language program deck according to the rules of the assembly system. Any errors in the source language will produce incorrect results from the assembler processing.

In order to produce a complete object program, the assembler program must read the entire source program before producing any object instruction cards. During the reading of the source cards, the assembler program performs a preliminary analysis and converts or translates from source language to object language wherever possible, eg: the mnemonic operation codes. As each source card is read and the mnemonic operation code is translated, the assembler program determines the length of the instruction. The instruction is assigned to an actual address, and the ILC is updated. If the source language instruction has been assigned a symbolic address by the programmer, the label and the actual address are stored in a table. When these labels appear in the source language instructions as Operand addresses, the assembler program searches the label table using the symbolic address as the key, and secures the actual address assigned to the label. The actual address is substituted for the programmer's symbolic operand address.

The assembler program contains many other tables which it references for conversion of the source language to object language. After complete analysis and conversion of the source language, the assembler program causes the object program to be listed and punched.

3. INTRODUCTION TO THE UNIVAC 1005 ASSEMBLY SYSTEM

Most of the programs for the UNIVAC 1005 will be written in the language of the UNIVAC 1005 Assembly System. The UNIVAC 1005 Assembly System provides the programmer with the necessary functions and convenience described in the preceding section. The use of instruction forms not described in this manual deviates from UNIVAC recommendations and must be the user's responsibility.

3.1. TERMINOLOGY DEFINITIONS

Alphabetic means a letter from the English alphabet (A through Z)

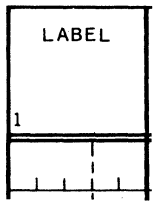
Numeric means an Arabic numeral (0 through 9)

Alphanumeric means the entire 64 character set of the UNIVAC 1005 which includes letters, numbers, and special characters.

3.2. CODING FORM

A coding form to be used to record the programmers instruction for subsequent key punching and processing by the UNIVAC 1005 Assembler program is shown in Appendix A. The coding form is set up in the same format as the punched card, and contains an indication of the card columns to be used for each field.

3.2.1. LABEL Columns 1 through 5

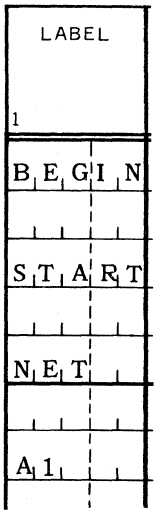


This field is provided for the symbolic Labels assigned to those lines of coding which are referenced by the object program instructions. A Label may consist of from one to five characters (inclusive) and must begin in column 1 of the field. The first (left-most) character of a Label must be an alphabetic character. The remainder of the characters in a Label can be Alphabetic or Numeric. There is no limit to the number of Labels in a source program. However, if the number of labels exceeds the limit of the Assembler (approximately 40 labels for the 2 Bank Assembler and 310 labels for the 4 Bank Assembler) extra processing is required by the Assembler program. This is fully explained in the section on Operating the Assembler System. Five positions are provided in the Label field to allow meaningful assignment of programmer names. However, only the left-most three positions of a Label are significant to Assembler processing. The first three positions of each Label must be unique within a program. Extreme care should be taken when creating Labels.

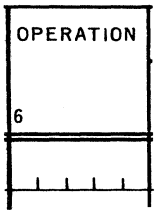
Labels used in a single program must be unique and may appear only once in the Label field. Labels will be used in the Operand address portion of instruction lines and may appear there as often as necessary. The explanations in this manual of the use of the relative coding technique of increments and decrements to Labels should enable the programmer to address the data in his program without an excess of Labels.

The Label of a line of coding becomes the symbolic address for the left-most (MSL) position of the instruction, and is used whenever the instruction is referenced. Labels are also used for the lines of coding which define data areas, and become the symbolic address for the left-most (MSL) position of the area set aside for data.

Since not all lines of coding require a label, the field may be left blank. Some examples of labels are:



3.2.2. OPERATION Columns 6 through 10



This field is for the mnemonic operation codes provided by the Assembler. Operation codes are usually alphabetic. The majority of Assembler Operation codes are two characters in length, and must begin in column 6.

Some examples of Operation codes are:

OPERATION
6
T A
T D
A D
S U
M U
D V
E N D

OPERAND 1			
I. A. *	FIELD A 12	±	INC. 18

3.2.3. OPERAND 1

This is a heading for those columns which are normally used to specify the address of the MSL or LSL of OP 1 depending on the ascending or descending mode of the instruction. This portion of the coding form is also used for other purposes, since not all Operations involve an Operand 1. The following description of the OPERAND 1 fields is based on the normal use to specify OP 1 addresses. A complete explanation of Operand 1 addressing begins in Section *3.3.*

3.2.3.1. IA Column 11

This column is used to indicate Indirect Addressing. When the OP 1 of an instruction is a primary address, an asterisk (*) is placed in this column. It must be left blank at all other times.

3.2.3.2. FIELD A Columns 12 through 16

This field of the form will normally contain a programmer's symbolic address for the location of instructions and data within his program. Any Labels which appear here must also appear in the LABEL field of some line of coding. Field A is a five position field for OP 1 Labels which always begin in column 12.

3.2.3.3. ± INC Columns 17 through 20

These columns are normally used to indicate an increment to the address assigned to a Label. Increments are shown in decimal numbers. If column 17 contains a plus sign (+) the increment is added. If column 17 contains a minus

sign (-) the increment becomes a decrement and is subtracted from the Label address. Increments must be left-justified (begin in column 18). Some examples of OPERAND 1 addresses are:

OPERAND 1			
I.A. *	FIELD A 12	±	INC. 18
	DET CD		
	CAT	-	1
*	IA1		

OPERAND 2							
I.A. *	FIELD B 22	±	INC. 28		FIELD C 32	±	INC. 38

3.2.4. OPERAND 2

This is a heading for those columns which are normally used to specify the MSL and LSL addresses of Operand 2 in the instruction. This portion of the form is also used for other purposes. The following description of the OPERAND 2 fields is based on the normal use to specify OP 2 addresses. A complete description of OPERAND 2 addressing is found in Section 3.3.

3.2.4.1. IA, *FIELD B*, ± INC Columns 21 through 30

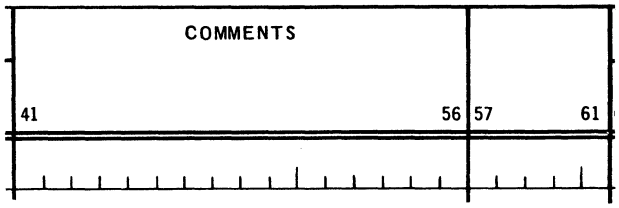
These fields are normally used to specify the most significant location (MSL) of OP 2. The description of the contents of these fields is the same as the description of the contents of OPERAND 1.

3.2.4.2. *FIELD C*, ± INC Columns 32 through 40

These fields are normally used to specify the least significant location (LSL) of OP 2.

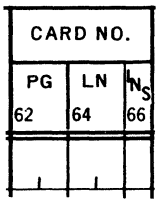
NOTE: The indication for OP 2 Indirect Addressing is in column 21 only. Column 31 is not used as part of the specification for OP 2 LSL.

The description of FIELD C and ± INC is the same as the corresponding fields of OPERAND 1.



3.2.5. COMMENTS Columns 41 through 61

This portion of the form is provided to allow the programmer to include pertinent comments as to the purpose of the line or lines of coding. The comments are not considered by the Assembler processing and merely pass through to the printed and punched output. These columns are also used to indicate constant values which are to be included in the object program. This use is described in the section which covers Constants. Comments used for (1) DC operations, (2) Comma (,) operations, (3) In line constant operations (*), and Comment Card operations may not extend beyond column 61. Comments for all other source code operations may not extend beyond column 56.



3.2.6. CARD NUMBER Columns 62 through 66

This portion of the form is subdivided into three fields--PAGE NUMBER, LINE NUMBER, and INSERT NUMBER. These columns are used to indicate the number of the page of coding, and the number of the line from which the key punched card was produced. The Card Number field will assist the key punching effort as well as provide for the re-sequencing of the cards in the event the original sequence is disturbed. For proper assembly processing, it is necessary that the cards be read by the Assembler program in the sequence in which they appear in the source program. The Card Number field is used for external control purposes only. The Assembler program does not check the sequence as it reads the card.

The INSERT NUMBER column is provided as a facility to insert additional lines of coding in a source program, after the initial effort, without disturbing the sequence established by Page and Line Number.

During the assembly processing, the Assembler program assigns a consecutive number to the output cards in the object program deck. The Assembler assigned card number is punched into columns 62 through 65. Column 66 is blank in the output card.

3.2.7. REMAINDER

The remainder of the card columns are not examined by the Assembler program, and are available for whatever use the programmer may determine. Such things as Job Number, Programmer's Initials, and Date may be included on a repetitive punching basis. These items will not appear in the object deck or on the output listing produced by the Assembler. Card Columns 67 through 73 are used for the object language instructions, and columns 74 through 80 are used for instructions to the Load program.

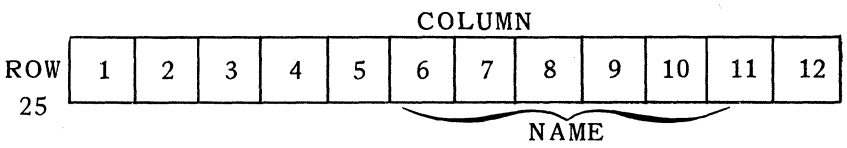
3.3. OPERAND 1 ADDRESS SPECIFICATION

There are several methods of specifying operand addresses in the UNIVAC 1005 Assembly System. A description of the most commonly used methods follows below.

3.3.1. Symbolic Address (Label) Specification

A definition of a symbolic address and some examples of Labels have been given in preceding sections. In the UNIVAC 1005 Assembly System, when a Label is used on a line of coding which defines a data area, that line of coding also includes the length of the area to be allocated to the data. The Label is then used to specify the MSL of the data area. By placing a plus sign (+) as a prefix to the Label when it is used as an operand address, the programmer can specify the LSL of the data area.

Example: A Declarative has been used to establish a data area of six positions with a Label NAME. Assume the data area has been allocated in memory as:



The following use of the Label NAME as an Operand 1 address of a TRF-D instruction would then cause a substitution of R25/C6, since a Label specifies the MSL of the area.

LABEL	OPERATION	OPERAND 1			
		I.A. *	FIELD A	±	INC.
1	6		12		18
	TD		NAME		

The use of the plus sign (+) prefix to the Label NAME as an Operand 1 address in a TRF-A instruction would then cause a substitution of R25/C11, the LSL of the area.

LABEL	OPERATION	OPERAND 1			
		I.A. *	FIELD A	±	INC.
1	6		12		18
	TA		+NAME		

NOTE: The plus sign (+) can be used as a prefix to 5 character Labels by coding the first 4 characters only. The first 3 characters are significant to the Assembler program.

Labels must be coded starting in the left-hand position of Field A (column 12). If the plus sign (+) prefix is used, it must appear in column 12, and the Label starts in column 13.

3.3.2. Increments To Symbolic Addresses

When a Label has been placed on a line of coding, the programmer knows that the Assembler program is going to allocate the number of memory locations required by that line of coding, and will also assign an actual address to the Locations. The programmer does not know the *actual* address which will be assigned, but he knows that the use of the same Label will cause the Assembler program to substitute whatever actual address was assigned. Based on this knowledge, the programmer is then able to specify the address of locations relative to his line of coding. This is accomplished by indicating an increment or decrement to the Label in the ± INC field of the Coding form.

Assume that allocation has been made according to the previous example. The following coding would cause the Assembler program to produce these substitute addresses:

OPERAND 1			
I. A. *	FIELD A 12	±	INC. 18
	NAME	+	1
	NAME	+	5
	+NAME	-	1
	+NAME	-	5
	NAME	-	3
	+NAME	+	1

= address R25/C7 (MSL + 1)

= address R25/C11 (MSL + 5 = LSL)

= address R25/C10 (LSL - 1)

= address R25/C6 (LSL - 5 = MSL)

= address R25/C3 (Outside of area)

= address R25/C12 (Outside of area)

It should be noted that the use of increments is not restricted to addresses within the area allocated by the line of coding.

When an increment is used, a plus or minus sign must appear in column 17, and the amount of the increment (in decimal numbers) must start in column 18.

3.3.3. Decimal Addressing

As mentioned in Section 1.1., decimal addressing is provided for by the UNIVAC 1005 Assembly System. This technique allows the programmer to consider the layout of Input Output, instructions, and data in a consecutive sequential manner. This eliminates the problems associated with advancing that takes place in Row and Column addressing.

When a decimal address is specified to the Assembler program, the decimal number is converted to the two-character address required in the object language.

Decimal addresses take the following form:

¤N through ¤NNNN

¤ = the special character lozenge which indicates to the Assembler program that what follows is a decimal number.

N through NNNN = a decimal number which must be left-justified.

Indirect Addressing is allowed with decimal addressing.

Examples of decimal addressing

OPERAND 1			
I. A. *	FIELD A 12	±	INC. 18
	¤1		
	¤80		
	¤9621		
	¤19212		

= First location of Read Input Storage (R1/C1, Bank 1)

= Last location of Read Input Storage (R3/C18, Bank 1)

= First location in Bank 2 (R1/C1, Bank 2)

= Last location in Bank 2 (R31/C31, Bank 2)

When decimal addressing is used, the lozenge (¤) must appear in column 12. The decimal number must begin in column 13. The decimal number cannot exceed 4 digits, since the maximum address is ¤3844 (four bank system).

3.3.4. Row And Column Addressing

Row and Column addressing is used when the programmer knows the actual address of the data. An actual or absolute address in the UNIVAC 1005 is specified by two 6-bit characters which are converted by the computer circuitry into Row, Column and Bank. Row and Column addressing in the UNIVAC 1005 Assembly System allows the programmer to specify Row, Column, and Bank eliminating the need to memorize or reference tables of the two-character codes required in the object program.

The following format is used to specify Row and Column address:

$RRCCBn

$ is the indication to the Assembler program that what follows is a Row and Column address

RR = the numeric Row Number (1 - 32)

CC = the numeric Column Number (1 - 32)

B must be placed in the ± column of the OPERAND field

n = the numeric Bank Number (1 - 4)

Example:

	OPERAND 1		
I. A. *	FIELD A 12	±	INC. 18
	$01101	B	1
	$03118	B	1
	$32131	B	2

MSL of Read Input

LSL of Read Input

LSL of rX

Indirect Addressing can be specified with machine-oriented addresses by placing an asterisk (*) in the appropriate column (11 or 21) of the form.

The $ must appear in column 12, 22, or 32, and the letter B must appear in column 17, 27, or 37. Increments to Row and Column address are not allowed.

Row and Column address is also used to specify the address of the Special Registers.

3.3.5. Instruction Location Counter (ILC) Addressing

The ILC is the counter in the Assembler program which keeps track of the allocation of memory locations to instructions. The use of the current value of the ILC for addressing purposes is provided by the UNIVAC 1005 Assembly System. Proper use of this technique is based on the programmer's knowledge of the memory mapping process of the Assembler program. (See Section 2.5).

The $ character, alone, in the left-hand position of Field A, Field B, or Field C of the coding form, instructs the Assembler program to use the current value of the MSL of the instruction being assembled as the address for the A, B, or C portion of the object instruction. An increment or decrement to the address currently in the ILC can also be specified in the ± INC fields for each address. This increment or decrement does not change the *value* of the ILC itself. The maximum increment or decrement is 961.

Example 1: Assume the value of the ILC is ¤745 (R25/C1, B1) at the time the following descending transfer instruction is to be assembled.

LABEL	OPERATION	OPERAND 1			
		I.A. *	FIELD A	±	INC.
1	6		12		18
	TD		$	+	7

$ + 7 produces an address of 752 (745 + 7) which is the address which will be assigned to the next instruction.

Example 2: The following coding of a descending transfer instruction will cause the instruction itself to be transferred.

LABEL	OPERATION	OPERAND 1			
		I.A. *	FIELD A	±	INC.
1	6		12		18
	TD		$		

3.4. OPERAND 2 ADDRESS SPECIFICATION

The rules for OPERAND 1 address specification (Section 3.3.) apply to OPERAND 2 address specification.

3.4.1. Operand 2, Field C, Blank Addressing

The majority of UNIVAC 1005 Operations require the specification of an A address (OP 1 MSL or LSL), a B address (OP 2, MSL), and a C address (OP 2 LSL). The UNIVAC 1005 Assembly System allows the programmer to leave the Field C portion of the source language instruction blank when a symbolic address is used in Field B. When a Label is used in the Field B portion of the instruction, the Assembler program references the Label Table to acquire the MSL address of the area it has assigned to the Label. The same reference to the Label Table will also produce the LSL address of the assigned area which the Assembler program will then automatically include in the object instructions as the C address.

If Field B of the instruction does not contain a Label, automatic (blank) addressing will not be performed. If the Field C portion of the instruction contains any information, automatic (blank) addressing will not be performed, and the address specified in Field C will be used. If the LSL of the area indicated by the Label in Field B is not to be used, the programmer must specify the desired address in Field C.

For the following examples, DATA is the Label of a 6 character field assigned by the Assembler program to R25/C1, B1 (MSL) through R25/C6, B1 (LSL).

Example 1:

OPERAND 2							
I. A. *	FIELD B 22	±	INC. 28		FIELD C 32	±	INC. 38
	DATA						

The Assembler program will automatically assign R25/C6, B 1 as the C address.

Example 2:

OPERAND 2							
I. A. *	FIELD B 22	±	INC. 28		FIELD C 32	±	INC. 38
	+DATA						

The coding in Field B specifies that the LSL of DATA (+ prefix) is to be the MSL of the instruction. The blank Field C specifies that the LSL of DATA is to be the LSL of the instruction. This coding produces a one character operand with an MSL and LSL of R25/C6, B 1.

Example 3:

OPERAND 2							
I. A. *	FIELD B 22	±	INC. 28		FIELD C 32	±	INC. 38
	DATA	+	3				

Field B specifies that the MSL of DATA plus 3 locations (R25/C4, B 1) is the MSL of OP 2. Again the blank Field C will automatically produce the LSL of DATA (R25/C6, B1) as the LSL of OP 2. OP 2 is a 3 character operand.

Example 4:

OPERAND 2							
I. A. *	FIELD B 22	±	INC. 28		FIELD C 32	±	INC. 38
	+DATA	-	3		+DATA	-	1

Field B specifies that the LSL of DATA (+ sign) - 3 (R25/C3, B 1) is to be used as OP 2, MSL. The presence of information in Field C prevents the automatic (blank) addressing of OP 2, LSL. Field C specifies that the LSL of DATA (+ sign) - 1 (R25/C5, B 1) is to be used as the LSL of OP 2. OP 2 is a 3 character operand.

3.4.2. Operand 2 Indirect Addressing

As explained in Section 1.6., Indirect Addressing utilizes a primary address in the instruction which specifies the location of a secondary address in memory which contains the address of the data to be used in the Operation.

When Indirect Addressing is used for OP 2 of an instruction, the primary address in the instruction must refer to the MSL of a 4 character location that contains the secondary address. A 4 character secondary address is necessary due to the fact that OP 2 of the object instruction must specify a 2 character MSL address and a 2 character LSL address.

The specification of Indirect Addressing will cause the UNIVAC 1005, at object execution time, to perform a 4 location descending transfer from the address specified in the B portion of the instruction to the B and C portions of the Instruction Register. The OP 2 will then be accessed based on the new addresses in the Instruction Register.

If a Label is used with OP 2 Indirect Addressing, the Label is specified in Field B, and Field C is blank. The Label in Field B must specify an assigned area of 4 characters which contain a B and a C address. The UNIVAC 1005 Assembly System provides a pseudo-operation which is used for this purpose. (See the DI Operation, Section 4.10.4.) OP 2 Indirect Addressing is specified by an asterisk (*) in the IA Field of OPERAND 2 (Column 21). No indication is made in Column 31.

Example: Label JOE has been defined as the primary address for the secondary address DATA. DATA has been defined as a 6 character area. JOE has been assigned to locations ¤801 through ¤804. DATA has been assigned to locations ¤745 through ¤750. Thus, in location ¤801 and ¤802 is the machine code equivalent of ¤745, and in location ¤803 and ¤804 is the machine code equivalent of ¤750.

The following coding will produce a correct Indirect Address reference to DATA as the OP 2 of an instruction.

OPERAND 2							
I. A.	FIELD B	±	INC.		FIELD C	±	INC.
*	22		28		32		38
*	J O E						

When decimal or Row/Column Indirect Addressing is used. Field B must specify the MSL of the location of the 4 character secondary address.

Example 1: Same as above

	OPERAND 2						
	I. A. *	FIELD B 22	±	INC. 28		FIELD C 32	±	INC. 38
*		H801						

Example 2: R25/C25, B1 = 801

	OPERAND 2						
	I. A. *	FIELD B 22	±	INC. 28		FIELD C 32	±	INC. 38
*		$2525	B	1				

3.5. SUMMARY OF FIELD A, FIELD B, AND FIELD C SPECIFICATIONS

3.5.1. FIELD A

Field A of the source language instruction may specify:

1. LSL address of OP 1
2. MSL address of OP 1
3. Decimal digits
4. Octal digits
5. Test bit conditions
6. Destination address of JUMP TEST Operation
7. Two machine language characters

3.5.2. FIELD B

Field B of the source language instruction may specify:

1. MSL address of OP 2
2. MSL address of OP 1
3. Set condition bits
4. Decimal digits
5. Octal digits
6. Destination address of JUMP TEST Operations
7. Two machine language characters

3.5.3. FIELD C

Field C of the source language instruction may specify:

1. LSL address of OP 2
2. Two machine language characters

NOTE: Those specifications not previously explained will be covered in the Section with the instructions that require or allow the specification.

3.6. STANDARD SYSTEM LABELS

In addition to the programmer assigned symbolic Labels previously discussed, the UNIVAC 1005 Assembly System provides 15 Standard Labels for predesignated areas of main store. These Standard Labels are not counted in with the number of Labels assigned by the programmer. The purpose of the Standard Labels is to provide uniformity of assignment of these predesignated areas, and reduce the processing time of the Assembler program for handling repetitive references to these areas.

The Standard Labels and predesignated areas are:

STANDARD LABEL	PREDESIGNATED AREA	DECIMAL ADDRESS	ROW/COLUMN ADDRESS
$R1	80 Column Read Input	¤ 1 - ¤ 80	R1/C1-R3/C18, B1
$R2	2nd Half of 160 Col. Read Code Image	¤ 81 - ¤160	R3/C19-R6/C5, B1
$RC	160 Column Read Code Image	¤ 1 - ¤160	R1/C1-R6/C5, B1
$PR	132 Column Print Storage	¤161 - ¤292	R6/C6-R10/C13, B1
$P1	80 Column PUNCH Storage	¤293 - ¤372	R10/C14-R12/C31, B1
$P1	80 Column Read/Punch Read Storage	¤293 - ¤372	R10/C14-R12/C31, B1
$P2	80 Column Read/Punch Punch Storage	¤373 - ¤452	R13/C1-R15/C18, B1
$PC	160 Column Code Image Punch Storage	¤293 - ¤452	R10/C14-R15/C18, B1
$Z1	160 Column Read/Punch Code Image Read Storage	¤293 - ¤452	R10/C14-R15/C18, B1
$Z2	160 Column Read/Punch Code Image Punch Storage	¤453 - ¤612	R15/C19-R20/C23, B1
$BM	First Location beyond Input Output Storage	¤613	R20/C24, B1
$IR	Instruction Register	None	R32/C1-R32/C7, B1
$CC	Instruction Control Counter	None	R32/C8-R32/C9, B1
$XR	Register X	None	R32/C1-R32/C31, B2
$TR	Translation Table or	¤1828 - ¤1891 ¤3750 - ¤3813 or	R28/C30-R30/C31,B2 R28/C30-R30/C31,B4
$8Ø	First 80 Positions of Print Area		R6/C6-R8/C23
$AR	Arithmetic Register	None	R32/C1-R32/C31,B1

The Assembler processing associated with Standard Labels is the same as for the Labels assigned by the programmer. That is, the use of a Standard Label as an address specification within a line of coding will cause the Assembler program to substitute the MSL of the area identified by the Standard Label. The LSL address is also specified and substituted in the same manner as for programmer's Labels.

The Standard Label $TR refers to the required location for translation tables when the hardware translate option of the UNIVAC 1005 is part of the object system.

4. UNIVAC 1005 ASSEMBLY SYSTEM INSTRUCTIONS

This section of the manual covers the instruction repertoire of the UNIVAC 1005 as programmed through the language of the UNIVAC 1005 Assembly Systems and the declarative instructions which direct the processing of the Assembler program.

4.1. LEGEND

The following abbreviations are used in the description of the UNIVAC 1005 Assembly System.

1L = Operand 1, LSL
1M = Operand 1, MSL
2L = Operand 2, LSL
2M = Operand 2, MSL
3L = least significant location of quotient or product
K = any alphanumeric character
D = any numeric character
CC = characters whose bit positions represent Condition Indicators
NI = MSL Address of the next instruction to be executed
( ) = contents of the area specified within the parentheses
→ = transfer to
Ø = a blank column (space code), used to establish positional notation
IA = Indirect Addressing

NOTE: For a description of Operand 1, Operand 2, and the UNIVAC 1005 machine language instructions, see Section 1.2.

4.2. LENGTH OF OPERANDS

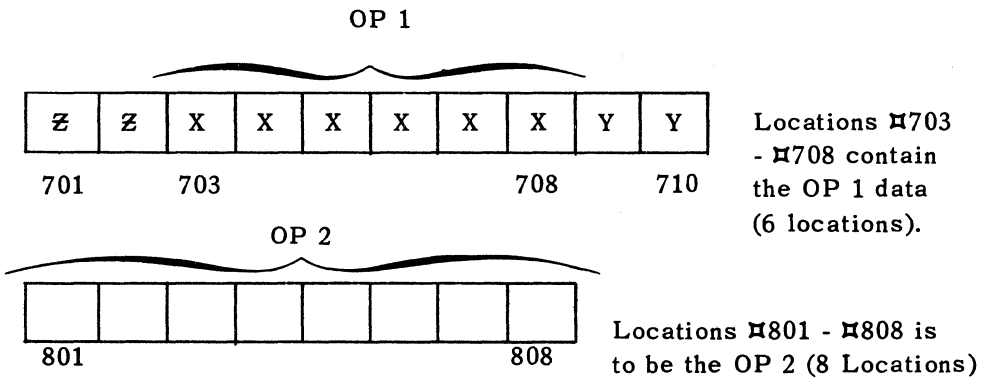
The length of the Operand(s) in a UNIVAC 1005 instruction is normally defined by the addresses of Operand 2. An instruction is normally terminated when the last location (LSL if descending mode, MSL if ascending mode) of OP 2 has been handled. This means that the number of locations in OP 1 *must be the same* as the number of locations in OP 2. The maximum size operand for a transfer, arithmetic, or compare instruction is 961 locations unless otherwise specified.

If the lengths of the Operands (as defined by the programmer) to be processed in an instruction are not the same, special programming involving the use of Register X is required. The "Transfer to rX" (TX) command allows the programmer to specify OP 1, MSL and OP 1, LSL, thus defining the length of OP 1. (See Section 4.3.6. for complete description of TX.) The destination (OP 2) of the TX command is rX (31 positions). The TX instruction performs an Ascending Transfer of OP 1 to rX, beginning at the LSL of each. When the OP 1, MSL (as specified in the instruction) has been transferred to rX, access of OP 1 is terminated. The TX instruction continues, transferring space codes into the excess positions of rX until the MSL of rX has been filled. This signals the end of the instruction.

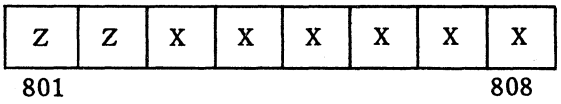
After transferring the smaller of the two Operands to rX, the programmer then uses the appropriate location in rX as the OP 1 address of the instruction which does the required processing. This insures the use of space codes in the locations which are added to make the length of OP 1 equal the length of OP 2. This condition is particularly important for the Arithmetic, the Compare, and the Transfer instructions.

Following is an example of the incorrect use of unequal length Operands and the erroneous result produced, as well as an example of the use of the TX command to produce correct results.

Given:

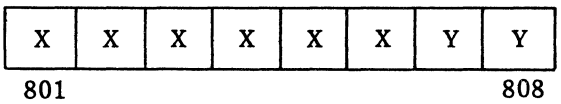


Example 1: If an Ascending Transfer of OP 1 to OP 2 is executed, the results in the OP 2 locations would be



since the length of OP 2 (8 Locations) determines the length of the Operand 1.

Example 2: If a Descending Transfer of OP 1 to OP 2 is executed, the results in the OP 2 locations would be

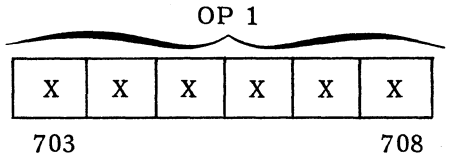


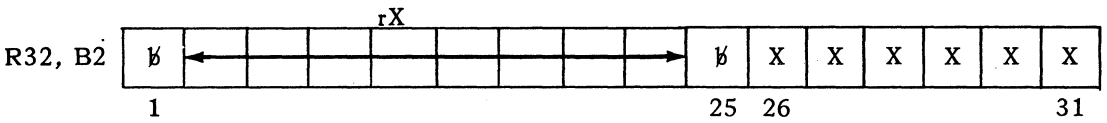
since the length of OP 2 (8 Locations) determines the length of Operand 1.

In Examples 1 and 2, extraneous locations (the Z's and Y's) which are not part of OP 1 would be transferred. If an Arithmetic instruction was executed using the same operands, the locations containing the Z's would become part of the OP 1 and would be combined with the values in ¤801 and ¤802, producing an erroneous result.

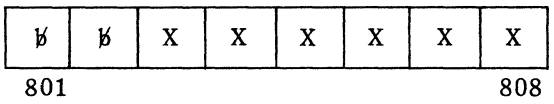
Using the same conditions given for the preceding examples, the following example methods can be used to produce correct results.

Example 3: Instruction 1. Transfer OP 1 to rX using the TX command. The TX command provides for specification of OP 1, MSL *and* LSL.



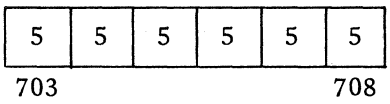


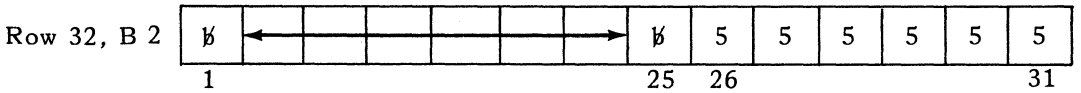
Instruction 2: Ascending Transfer specifying the LSL of rX ($3231B2) as OP 1, LSL to the destination OP 2 (¤801 - ¤808). The length of the OP 2 (8 locations) will cause the low-order 8 locations of rX to be transferred. This will produce the following results in OP 2.



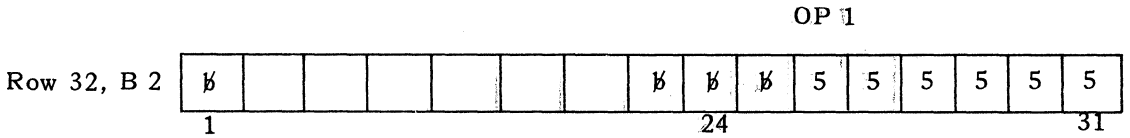
Example 4: In order to perform Arithmetic instructions on Operands of unequal length, the type of operation performed in Instruction 1, Example 3 would establish OP 1 in rX. The length of the OP 2 of the Arithmetic instruction would then control the number of locations to be used from rX (including high order spaces).

Instruction 1: Transfer OP 1 to rX





Instruction 2: Assume an ADD-ALG instruction



Original OP 2 = 2 2 2 2 2 2 2 2

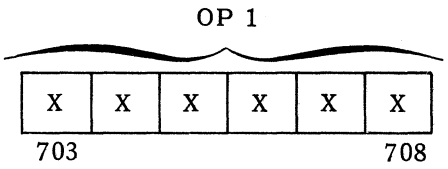
801 808

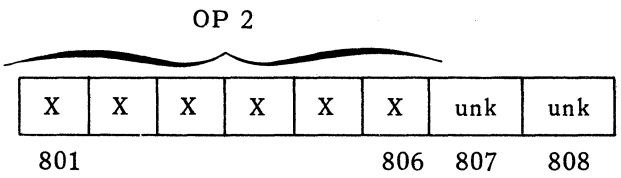
Final OP 2 = 2 2 7 7 7 7 7 7

801 808

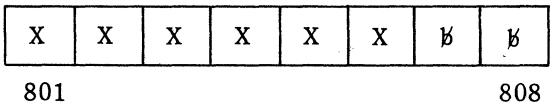
Example 5: In order to perform a Descending Transfer producing a result of the smaller OP 1 left-justified (in the most significant locations) in OP 2 and spaces in the remaining low-order positions of OP 2, two instructions are required.

Instruction 1: Descending Transfer, specifying the portion of the OP 2 locations which are to receive the significant data, as the OP 2 of the TD instruction.





Instruction 2: Use the Transfer Constant (TK) Instruction to transfer space codes to the low-order positions of OP 2. (See Section 4.3.5 for a description of the TK instruction.) This will produce the desired result.



An alternate method to produce the results indicated by Example 5 can be:

(1) A TX of Operand 1 to the XR.

(2) A TD of R32C26 to locations 801 through 808. The TD instruction will "wrap around" and pick up two blank characters from R32C1 a and R32C2, thereby producing a left justified result.

4.3. TRANSFER INSTRUCTIONS

4.3.1. TRANSFER DESCENDING

Mnemonic: TD Mode: DESCENDING Length: 7 IA: YES

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.	FIELD C	±	INC.
1	6		12		18		22		28	32		38
	TD		1M				2M			2L		

Function:

Transfer descending beginning from OP 1 - MSL specified by Field A; to OP 2 - MSL specified by Field B, until OP 2 - LSL specified by Field C has been filled.

Example 1:

Given: A 6 location area with the Label CAT has been allocated to R31/C1 through R31/C6 in Bank 1 (¤931 through ¤936).

Problem: Transfer Descending the data from card columns 4 through 9 to the area CAT.

LABEL	OPERATION	OPERAND 1				OPERAND 2							
		I.A.	FIELD A	±	INC.	I.A.	FIELD B	±	INC.		FIELD C	±	INC.
1	6	*	12		18	*	22		28		32		38
	TD		$R1	+	3		CAT						

FIELD A; OP 1, MSL:

The Standard Label $R1 specifies the MSL of Read Input Storage (the location of card column 1). The increment of +3 causes the MSL of this OP 1 to be the location of card column 4. $0104B1 or ¤4 can be used to specify the same OP 1, MSL since the location is known to the programmer.

FIELD B; OP 2, MSL:

The programmer's Label CAT specifies the MSL of the area assigned to CAT by the Assembler program.

NOTE: $3101B1 or ¤931 could *not* be used since the programmer does not know the actual address which will be assigned to area CAT.

FIELD C: OP 2, LSL:

Blank addressing will cause the Assembler program to use the LSL address of the area specified by the Label in Field B. The programmer can also use CAT + 5, or +CAT in Field C and produce the same result.

Example 2:

Given: The same area with the Label CAT from example 1. Also, a 6 location area with the Label DOG has been allocated to R1/C1 through R1/C6 of Bank 2 (¤962 through ¤967).

Problem: Transfer Descending the (CAT) to DOG.

LABEL	OPERATION	OPERAND 1				OPERAND 2							
		I.A.	FIELD A	±	INC.	I.A.	FIELD B	±	INC.		FIELD C	±	INC.
1	6	*	12		18	*	22		28		32		38
	TD		CAT				DOG						

FIELD A; OP 1, MSL:

The Label CAT specifies the MSL address of the area assigned to CAT.

FIELD B; OP 2, MSL:

The Label DOG specifies the MSL address of the area assigned to DOG.

FIELD C; OP 2, LSL:

Blank addressing will cause the Assembler program to use the LSL address of the area specified by the Label in Field B. The programmer can also use DOG + 5, or + Dog in Field C and produce the same result.

NOTE: Row/Column or Decimal Addressing can not be used for any of the addresses, since the actual locations of the data are not known by the programmer.

Example 3:

Problem: Transfer the entire contents of the card in Read Storage to the first 80 print positions of Print Storage.

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A. *	FIELD A	±	INC.	I.A. *	FIELD B	±	INC.	FIELD C	±	INC.
1	6		12		18		22		28	32		38
	TD		$R1				$PR			$PR	+	79

FIELD A; OP 1, MSL:

The Standard Label $R1 specifies the MSL of the Read Input Storage area. $010 1B1 or ¤1 could have been used.

FIELD B; OP 2, MSL:

The Standard Label $PR specifies the MSL of the Print Storage area $0606B1 or ¤161 could have been used.

FIELD C; OP 2, LSL:

Blank addressing can not be used, since this would cause the Assembler program to use the LSL of $PR, the last column of the Print Storage area. $PR + 79 is the address of Print position 80 which should be the LSL of OP 2. $0823B1 or ¤240 could have been used.

Example 4:

Problem: Transfer the data from column 80 of an input card to column 80 of an output card.

LABEL	OPERATION	OPERAND 1				OPERAND 2							
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	TD		+$R1				+$P1						

FIELD A; OP 1, MSL:

The plus sign (+) prefix to the Standard Label $R1 specifies the LSL of Read Input Storage as the MSL of OP 1. $0318B1 or ¤80 could have been used.

FIELD B; OP 2, MSL:

The plus sign (+) prefix to the Standard Label $P1 specifies the LSL of Punch Storage as the MSL of OP 2. $1231B1 or ¤372 could have been used.

FIELD C; OP 2, LSL:

Blank addressing in Field C specifies the LSL of the area whose Label is in Field B. (The plus sign in Field B does not constitute part of the Label. It instructs the Assembler program to use the LSL address of the labeled area.) $1231B1 or ¤372 could have been used.

NOTE: Since OP 2, MSL and OP 2, LSL specify the same address, a one location Operand is produced.

4.3.2. TRANSFER ASCENDING

Mnemonic: TA Mode: ASCENDING Length: 7 IA: YES

LABEL	OPERATION	OPERAND 1				OPERAND 2							
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	TA		1L				2M				2L		

Function:

Transfer ascending beginning from OP 1 - LSL specified by Field A; to OP 2 - LSL specified by Field C, until OP 2 - MSL specified by Field B has been filled.

Example 1:

Given: A 6 location area with the Label CAT has been allocated to R31/C1 through R31/C6 in Bank 1 (¤931 through ¤936).

Problem: Transfer Ascending the data from card columns 4 through 9 to the area CAT.

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.	FIELD A	±	INC.	I.A.	FIELD B	±	INC.		FIELD C	±	INC.
1	6	*	12		18	*	22		28		32		38
	TA		$R1	+	8		CAT						

FIELD A; OP 1, LSL:

$R1 is the Standard Label for the MSL of Read Input Storage (the location of card column 1). The increment of +8 causes the LSL of this OP 1 to be the location of card column 9. $0109B1 or ¤9 can be used to specify the same OP 1, LSL since the location is known to the programmer.

FIELD B; OP 2, MSL:

The programmer's Label CAT specifies the MSL of the area assigned to CAT by the Assembler program.

NOTE: $3101B1 or ¤931 could *not* be used since the programmer does not know the actual address which will be assigned to area CAT.

FIELD C; OP 1, LSL:

Blank addressing will cause the Assembler program to use the LSL address of the area specified by the Label in Field B. The programmer can also use CAT + 5, or + CAT in Field C and produce the same result.

Example 2:

Given: The same area with the Label CAT from Example 1. Also, a 6 location area with the Label DOG has been allocated to R1/C1 through R1/C6 of Bank 2 (¤962 through ¤967).

Problem: Transfer Ascending the (CAT) to DOG.

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.	FIELD A	±	INC.	I.A.	FIELD B	±	INC.		FIELD C	±	INC.
1	6	*	12		18	*	22		28		32		38
	TA		+CAT				DOG						

FIELD A; OP 1, LSL:

The Label CAT with a plus sign (+) prefix specifies the LSL of the area assigned to CAT.

FIELD B; OP 2, MSL:

The Label DOG specifies the MSL of the area assigned to DOG.

FIELD C; OP 2, LSL:

Blank addressing will cause the Assembler program to use the LSL address of the area specified by the Label in Field B. The programmer can also use DOG + 5, or + DOG in Field C and produce the same result.

NOTE: Row/Column or Decimal Addressing can not be used for any of the addresses, since the actual locations of the data are not known by the programmer.

Example 3:

Given: A 7 position location with the Label XT1 as the last instruction of a subroutine.

Problem: Transfer Ascending the previous 7 character instruction to the exit line XT1 of the subroutine.

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.	FIELD A	±	INC.	I.A.	FIELD B	±	INC.		FIELD C	±	INC.
1	6	*	12		18	*	22		28		32		38
	TA		$	-	1		XT1						

FIELD A; OP 1, LSL:

$-1 instructs the Assembler program to use the current value of the ILC minus one as the address of OP 1, LSL of this instruction. The current value of the ILC is the MSL of this instruction. The MSL of this instruction minus one is the LSL of the previous instruction.

FIELD B; OP 2, MSL:

The Label XT1 specifies the MSL of the locations assigned to that instruction by the Assembler program.

FIELD C; OP 1, LSL:

Blank addressing will cause the Assembler program to use the LSL of the locations assigned to the instruction stored at XT1.

4.3.3. TRANSFER CLEAR

Mnemonic: TC Mode: ASCENDING Length: 7 IA: Yes

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.	FIELD A	±	INC.	I.A.	FIELD B	±	INC.		FIELD C	±	INC.
1	6	*	12		18	*	22		28		32		38
	TC		1L				2M				2L		

Function:

Transfer ascending beginning from OP 1 - LSL specified by Field A; to OP 2 - LSL specified by Field C, until OP 2 - MSL specified by Field B has been filled. *Clear the OP 1 locations to space codes during the process.*

This instruction performs exactly the same as a TA (Transfer Ascending) instruction. The only difference is that as the characters are accessed from the OP 1 locations, they are not returned to the OP 1 locations. The effect of this instruction leaves the OP 1 characters cleared to space codes.

The rules for coding a TC instruction are the same as the rules for coding the TA instruction (See Section 4.3.2.)

4.3.4. TRANSFER NUMERIC

Mnemonic: TN Mode: ASCENDING Length: 7 IA: YES

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.	FIELD C	±	INC.
1	6		12		18		22		28	32		38
	TN		1L				2M			2L		

Function:

Transfer ascending beginning from the OP 1 - LSL specified by Field A; to OP 2 - LSL specified by Field C, until OP 2 - MSL specified by Field B has been filled. *Delete the X and Y bit-positions of the data delivered to OP 2.*

This instruction performs exactly like the TA (Transfer Ascending) instruction. The only difference is that before the characters are stored in the OP 2 locations, the zone bits (X and Y bit-positions) are stripped off (set to binary zero). The contents of OP 1 remain unchanged.

The rules for coding a TN instruction are the same as the rules for coding the TA instruction (See Section 4.3.2.)

4.3.5. TRANSFER CONSTANT

Mnemonic: TK Mode: ASCENDING Length: 7 IA: YES

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.	FIELD C	±	INC.
1	6		12		18		22		28	32		38
	TK		KK				2M			2L		

Function:

Transfer ascending beginning from location 3 of the Instruction Register (IR); to OP 2 - LSL specified by Field C, until OP 2 - MSL specified by Field B has been filled. Transfer a maximum of 2 locations (KK) from the IR. If OP 2 is more than two locations in length, space-fill the unentered high-order locations of OP 2.

When a TK instruction is brought to the IR for execution, the two alphanumeric characters KK will occupy locations 2 and 3 of the IR. These two locations are used similar to an OP 1 in an Ascending Transfer. However, the Operation code (TK) will cause the transfer from OP 1 to cease after the second transfer. The execution of the instruction will continue until OP 2, MSL has been filled. If there are more than two locations in OP 2, the excess high-order locations of OP 2 will be filled with space codes. If there are exactly two locations in OP 2, the instruction will transfer locations 2 and 3 of the IR (the constant KK). If there is only one location in OP 2, only position 3 of the IR will be transferred.

NOTE: The character ¤ (lozenge) may not be used as the first character of a constant in Field A of this instruction. Code in its bit configuration. (See Section 4.3.5.3.)

Example 1:

Problem: Store the letters CR in the last (low-order) two locations of Print Storage.

LABEL	OPERATION	OPERAND 1				OPERAND 2							
		I.A. *	FIELD A	±	INC.	I.A. *	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	TK		CR				+$PR	-	1				

FIELD A; KK:

The letters CR.

FIELD B; OP 2, MSL:

$PR is the Standard Label for the Print Storage area. The plus sign instructs the Assembler program to use the LSL of Print Storage, and the minus one causes the address of Print position 131 to become the MSL of this instruction.

FIELD C; OP 2, LSL:

Blank addressing will cause the Assembler program to use the LSL address of the area specified by the Label in Field B. The use of the plus sign and the increment in Field B have no effect on this Assembler program procedure.

Example 2:

Problem: Store a minus sign (-) in the LSL of the 10 character area assigned to FOX. Clear the high-order 9 locations of FOX to spaces.

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A. *	FIELD A	±	INC.	I.A. *	FIELD B	±	INC.	FIELD C	±	INC.
1	6		12		18		22		28	32		38
	TK		⌀-				FOX					

FIELD A: KK:

The ⌀ (space code) becomes location 2, and the minus sign (-) becomes location 3 of the IR. After they are transferred, the remainder of FOX is space filled.

FIELD B; OP 2, MSL:

The MSL of the area assigned to FOX becomes the MSL of this instruction.

FIELD C; OP 2, LSL:

Blank addressing causes the Assembler program to use the LSL of the area assigned to the Label in Field B (FOX) as the LSL of this instruction.

NOTE: See Example 5 for a description of negative constants.

Example 3:

Given: Several fields of the print line are to have a printed sign of plus (+) or minus (-) based on a condition developed by the program.

Problem: Store the proper sign indication in each of the fields using the TK instruction with Indirect Addressing.

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A. *	FIELD A	±	INC.	I.A. *	FIELD B	±	INC.	FIELD C	±	INC.
1	6		12		18		22		28	32		38
SIGN	TK	*	:SIGN	−	1		+FD1					
	TK	*	:SIGN	−	1		+FD2					
	TK	*	:SIGN	−	1		+FD3					

Solution: The location immediately preceding the instruction labeled SIGN has been established as the secondary address location for the Indirect Addressing to be performed in the instruction SIGN.

The processing in the program will previously determine the sign requirements of the printed fields. This same portion of the program will then transfer the appropriate character to location SIGN - 1, and execute a Jump instruction which transfers control to SIGN.

The instruction in SIGN is then brought to the Instruction Register. Before it is executed the instruction is examined by the hardware to see if it calls for Indirect Addressing. In this case, it does.

The hardware then automatically uses the address in the A portion of the IR as the LSL of a two location transfer from memory to the A portion of the IR. The hardware then examines the Operation code and performs the TK instruction using the "new" contents of the A portion of the IR as the two character constant KK.

Field B specifies that the LSL of FD 1 is to be used as the MSL of the instruction. Blank addressing in Field C will cause the same address to be the LSL of the instruction thus creating a one character OP 2. Each of the instructions will then transfer the appropriate sign indication to the LSL of each of the print fields.

Example 4:

Problem: Clear Bank 2 to space codes.

LABEL	OPERATION	OPERAND 1				OPERAND 2							
		I.A. *	FIELD A	±	INC.	I.A. *	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	TK		¢¢				¤9622				¤1922		

FIELD A; KK:

The blank columns in the Field A will cause KK to become space codes.

FIELD B; OP 2, MSL:

¤962 is the decimal address for the first location in Bank 2, which becomes the MSL of OP 2.

FIELD C; OP 2, LSL:

¤1922 is the decimal address of the last location in Bank 2, which becomes the LSL of OP 2.

Solution: This instruction will first transfer the two space codes, and then space-fill the remainder of OP 2---the rest of Bank 2.

If the KK portion of a TK instruction is to contain a negative constant, the minus sign (-) is used as a prefix to the two decimal digit constant. The Assembler program will place an X-bit over *both* of the numerals in KK. If the negative constant is to be a value from -1 through -9, a zero must be coded between the minus sign and the decimal numeral.

If the KK portion of a TK instruction is to contain a positive constant, no sign indication is required.

Example 5:

Problem: Store a -1 in the 2 location area assigned to COUNT.

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	TK		-Ø1				COUNT						

FIELD A; KK:

The minus sign causes the Assembler program to place a binary 1 in the X bit position of both the Ø and the 1 in the object language instruction. (See Section 4.4.)

FIELD B; OP 2, MSL:

The address of the MSL of the area assigned to COUNT is used as the MSL of this instruction.

FIELD C; OP 2, LSL:

Blank addressing causes the Assembler program to use the address of the LSL of the area assigned to COUNT as the LSL of this instruction.

NOTE: The minus sign (-) symbol cannot be used as the first character of a constant in the TK instruction. Code in its bit configuration. (See 4.5.3.)

4.3.5.1. Symbolic Address Substitution

The TK instruction can also be used to change the A, B, or C address of an instruction. The UNIVAC 1005 Assembly System provides for source language coding of Labels in the Field A portion of a TK instruction. These Labels are converted to the two character machine language address assigned by the Assembler program and stored as KK in the A portion of the object language TK instruction.

The symbol colon (:) is used in column 12 as a prefix to the Label whose assigned address is to become KK.

Example 6:

Problem: Change the A address portion of a TA instruction stored in DOG to refer to CAT.

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	TK		:+CAT				DOG	+	1		DOG	+	2

FIELD A; KK:

The symbol colon (:) informs the Assembler program that a Label appears in Field A of this TK instruction. The plus sign (+) prefix instructs the Assembler to use the LSL address of the area assigned to CAT as KK in this instruction. (The LSL is required due to the ascending mode of the TA instruction.)

FIELD B; OP 2, MSL:

The MSL of the TA instruction stored at DOG contains the Operation code. Therefore, the MSL of the A portion of that instruction is stored at DOG + 1, which becomes the MSL of the TK instruction.

FIELD C; OP 2, LSL:

DOG + 2 is the address of the LSL of the A portion of the TA instruction stored at DOG. This address is used as the LSL of OP 2 of the TK instruction.

NOTE: If the symbol colon (:) is to be used as the constant K, it must be coded in its bit configuration. (See Section 4.3.5.3. below.)

4.3.5.2. Row/Column and Decimal Addressing

Row/Column and Decimal Addressing can also be used to cause a machine language address to be placed in the A portion of the TK instruction.

Example 7:

Problem: Store the machine language of decimal location ¤1000 as the LSL of a 7 character instruction stored in FOX.

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	TK		¤1000				FOX	+	5				

FIELD A; KK:

The Assembler program will use the two character code for address ¤1000 as KK in this instruction.

FIELD B; OP 2, MSL:

FOX is the MSL address of the 7 character instruction. The LSL address portion (the C portion) of FOX is therefore FOX + 5, which will be used as the OP 2 MSL of this instruction.

FIELD C; OP 2, LSL:

Blank addressing will cause the Assembler program to use the LSL of the area assigned to FOX as the OP 2, LSL of this instruction. This is the LSL of the C portion of FOX.

Example 8:

1000 is the decimal address of R2/C8, Bank 2. The coding for Example 7 could have read as follows and produce the same result.

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.	FIELD A	±	INC.	I.A.	FIELD B	±	INC.		FIELD C	±	INC.
1	6	*	12		18	*	22		28		32		38
	TK		$Ø2Ø8	B	2		FOX	+	5				

4.3.5.3. Binary Coded Constants

The basic level of machine code is a series of binary bits. In the UNIVAC 1005, 6 binary digits (bits) are stored in each memory location. The UNIVAC 1005 Assembly System allows the programmer to use binary indications, if necessary, to code his program.

In order to reduce the number of source language columns required for binary indication, the UNIVAC 1005 Assembly System provides for octal coding. An octal code is made up of three adjacent binary digits. Thus,two *octal* digits can be used to express the contents of one UNIVAC 1005 location. To determine the octal equivalent of the 6-bit binary code, the following is suggested.

	(4)	(2)	(1)	(4)	(2)	(1)
UNIVAC 1005 bit position	X	Y	8	4	2	1
	Octal Digit #1			Octal Digit #2		

Add the binary value of the bits in the (4), (2), and (1) bit positions (maximum sum = 7) to create Octal Digit #2. Using the same values (4), (2), (1) add the sum of the X, Y, and 8-bit positions (maximum 7) to create Octal Digit #1. Write as a two place number.

For example: the bit configuration of the letter A equals

X	Y	8	4	2	1
0	1	0	1	0	0

Position 1 = 0
2 = 0
4 = 4
sum 4 equals Octal Digit #2

Position 8 = 0
Y = 2
X = 0
sum 2 equals Octal Digit #1

Thus the octal form of the letter A is 24. In the same manner, any six bit configuration can be shown by using the two digit octal form.

The symbol for number (#) is used as a prefix to indicate to the Assembler program that octal coding has been used. This symbol (#) must precede the four octal digits.

Example 9:

Problem: Store two lozenge symbols (⌑⌑) in the least significant locations of the area assigned to RAT. (Reminder, the lozenge symbol cannot be used for KK in the TK instruction.) = 111 101

LABEL	OPERATION	OPERAND 1				OPERAND 2							
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	TK		#7575				+RAT	−	1				

FIELD A; KK:

The number symbol (#) indicates to the Assembler program that what follows is octal coding. The Assembler program then forms the two characters KK.

FIELD B; OP 2, MSL:

+ RAT -1 is the second least significant location of RAT.

FIELD C; OP 2, LSL:

Blank addressing causes the Assembler program to use the LSL of the area assigned to RAT as the LSL of this instruction.

NOTE: Octal coding can also be used as a method for addressing in the UNIVAC 1005 Assembly System. See Section 6.1 for a complete description.

4.3.6. TRANSFER TO REGISTER X

Mnemonic: TX Mode: ASCENDING Length: 5 IA: NO

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	TX		1L				1M						

Function:

(OP 2 in the TX command is rX). Transfer ascending beginning from OP 1 - LSL; continuing until OP 1 - MSL has been transferred. Space fill any unentered high order positions of rX. Maximum OP 1 operand length is 31 locations. (See Section 4.2 for further information.)

NOTE: 1m in Field "B" applies in this case to OP 1 and not to OP 2.

This instruction has an implied OP 2 of rX, which is indicated by the Operation code. The purpose of this instruction is to provide for the handling of unequal length operands. Complete specification of the length of OP 1 is made in Field A and Field B (two addresses), and rX is OP 2. *The TX instruction is a 5 character instruction.*

Example 1:

Problem: Transfer the 5 characters from card columns 1 thru 5 to the 8 character field assigned to CAT. NOTE: This requires two instructions:

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	TX		$R1	+	4		$R1						
	TA		$3213	B	2		CAT						

Instruction 1 TX

FIELD A; OP 1, LSL:

The Standard Label $R1 + 4 specifies that the address of the fifth position of Read Input storage is to be the LSL of this instruction.

FIELD B; OP 1, MSL:

The Standard Label $R1 specifies that the address of the first position is to be the MSL of this instruction.

FIELD C; *Ignored*

Instruction 1 transfers locations 1 through 5 of Read Input storage to the low order 5 locations of rX (R32/C27, Bank 2 through R32/C31, Bank 2). The remainder of rX (R32/C1, Bank 2 through R32/C26, Bank 2) is filled with space codes.

Instruction 2 TA

FIELD A; OP 1, LSL:

$3231B2 specifies that the LSL of rX is to be used as the OP 1, LSL of this instruction.

FIELD B; OP 2, MSL:

The MSL address of the area assigned to CAT is to be used as the OP 2, MSL of this instruction.

FIELD C; OP 2, LSL:

Blank addressing specifies that the LSL address of the area assigned to CAT is to be used as the OP 2, LSL address of this instruction.

The ascending transfer in Instruction 2 calls for an 8 location transfer (the length of OP 2, CAT). The low order 5 locations of CAT will contain the 5 characters from the card which were transferred to the low order positions of rX by Instruction 1. The 3 high order positions of CAT will contain space codes from the un-entered portion of rX.

4.3.7. TRANSLATE (Optional)

NOTE: The Translate instruction can only be used if the UNIVAC 1005 system for which the program is being assembled has the hardware translate option.

Mnemonic: TR Mode: DESCENDING Length: 7 IA: YES

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	TR						2M				2L		

Function:

Translate each of the characters in the field defined as OP 2 (except the LSL) according to the Translation Table. Return the translated character to OP 2.

The use of the Translate option on the UNIVAC 1005 requires that the translated characters be returned to the same locations from which the characters to be translated were obtained. Thus,2M and 2L not only define OP 2, they also define OP 1. Furthermore, the Translate option also requires that OP 2 be one location longer than OP 1 at the LSL end of OP 2. 2M specifies both OP 1 and OP 2 MSL. 2L specifies the *OP 2* LSL, and the hardware automatically uses 2L-1 as the OP 1, LSL.

The TR instruction replaces each character in the field to be translated with a character selected from the translate table ($TR). The basis for selecting the replacement character is the binary value of the character to be replaced (see Figure 4–1). The binary value of any six-bit character ranges from zero (000000) through 63 (111111). This binary value provides the character address of the particular six-bit configuration within the translate table which is to replace the character. In other words, a character with a binary value of (011111) is replaced by whatever character is pre-stored in R28/C30 of the translate table; a character with a binary value of (111111) is replaced by whatever character is pre-stored in R28/C31 of the translate table; a character with a binary value of (000000) is replaced with whatever character is pre-stored in R29/C1 of the translate table; and so on. The contents of the translate table are not altered by the instruction. The characters to be translated must be in the same bank of storage as the translate table.

The Translation Table must be stored in locations R28/C30, Bank 2 (MSL) through R30/C31, Bank 2 (LSL) of a 2 bank UNIVAC 1005 system; or in R28/C30, Bank 4 (MSL) through R30/C31, Bank 4 of a 4 bank UNIVAC 1005 system. More than one Translation Table can be used in a program provided that the program transfers the proper set of translation codes to the Translation Table locations prior to each change in use.

Example 1:

Given: A field of 80 characters received in a communication code stored in an 80 location area assigned the Label INMSG. (INMSG must be in the Last Bank of storage).

A table of XS-3 codes stored in a 64 location area assigned the Label XS3.

		ROW
		28
		COL
	0 1 1 1 1 1	30

		ROW
		28
		COL
	1 1 1 1 1 1	31

ORIG. CHAR.	ORIGINAL CODE IN 1005 STORAGE — ADDRESS CODE OF TABLE COL	TABLE ROW 29 / TABLE COL	NEW CHAR.
	X Y 8 4 2 1		
	0 0 0 0 0 0	1	
	0 0 0 0 0 1	2	
	0 0 0 0 1 1	3	
	0 0 0 1 1 1	4	
	0 0 1 1 1 0	5	
	0 1 1 1 0 0	6	
	0 1 1 0 0 1	7	
	0 1 0 0 1 0	8	
	0 0 0 1 0 0	9	
	0 0 1 0 0 0	10	
	0 1 0 0 0 1	11	
	0 0 0 0 1 0	12	
	0 0 0 1 0 1	13	
	0 0 1 0 1 0	14	
	0 1 0 1 0 1	15	
	0 0 1 0 1 1	16	
	0 1 0 1 1 1	17	
	0 0 1 1 1 1	18	
	0 1 1 1 1 0	19	
	0 1 1 1 0 1	20	
	0 1 1 0 1 1	21	
	0 1 0 1 1 0	22	
	0 0 1 1 0 1	23	
	0 1 1 0 1 0	24	
	0 1 0 1 0 0	25	
	0 0 1 0 0 1	26	
	0 1 0 0 1 1	27	
	0 0 0 1 1 0	28	
	0 0 1 1 0 0	29	
	0 1 1 0 0 0	30	
	0 1 0 0 0 0	31	
	X Y 8 4 2 1		

ORIG. CHAR.	ORIGINAL CODE IN 1005 STORAGE — ADDRESS CODE OF TABLE COL	TABLE ROW 30 / TABLE COL	NEW CHAR.
	X Y 8 4 2 1		
	1 0 0 0 0 0	1	
	1 0 0 0 0 1	2	
	1 0 0 0 1 1	3	
	1 0 0 1 1 1	4	
	1 0 1 1 1 0	5	
	1 1 1 1 0 0	6	
	1 1 1 0 0 1	7	
	1 1 0 0 1 0	8	
	1 0 0 1 0 0	9	
	1 0 1 0 0 0	10	
	1 1 0 0 0 1	11	
	1 0 0 0 1 0	12	
	1 0 0 1 0 1	13	
	1 0 1 0 1 0	14	
	1 1 0 1 0 1	15	
	1 0 1 0 1 1	16	
	1 1 0 1 1 1	17	
	1 0 1 1 1 1	18	
	1 1 1 1 1 0	19	
	1 1 1 1 0 1	20	
	1 1 1 0 1 1	21	
	1 1 0 1 1 0	22	
	1 0 1 1 0 1	23	
	1 1 1 0 1 0	24	
	1 1 0 1 0 0	25	
	1 0 1 0 0 1	26	
	1 1 0 0 1 1	27	
	1 0 0 1 1 0	28	
	1 0 1 1 0 0	29	
	1 1 1 0 0 0	30	
	1 1 0 0 0 0	31	
	X Y 8 4 2 1		

Figure 4–1. Character Translate Table

A table of communication codes stored in a 64 location area assigned the Label COMCD.

LABEL	OPERATION	OPERAND 1				OPERAND 2							
		I.A. *	FIELD A	±	INC.	I.A. *	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	TD		XS3				$TR						
	TR						INMSG				+INMS	+	1

Instruction 1 TD

FIELD A; OP 1, MSL:

The Label XS3 specifies that the MSL address of the XS-3 table is to be used as OP 1, MSL.

FIELD B; OP 2, MSL:

The Standard Label $TR specifies that the MSL of the area required for the Translation Table is to be used as OP 2, MSL of this instruction.

FIELD C; OP 2, LSL:

Blank addressing specifies that the LSL of the area specified by $TR is to be used as the LSL of this instruction.

Instruction 1 loads the Translation Table with the correct translation characters.

Instruction 2 TR

FIELD A; *Ignored*

FIELD B; OP 1 and OP 2 MSL:

The characters to be translated must be obtained from and replaced in the same locations by the translated characters. The address of the MSL of the area assigned to INMSG will be used as OP 1 and OP 2 MSL of this instruction.

FIELD C; OP 2, LSL:

This must specify the location + 1 of the last character to be translated. The character in this location is not disturbed. + INMS (four characters of the INMSG) specifies the address of the LSL of the characters to be translated. + INMS + 1 specifies the correct OP 2, LSL for this instruction.

If the results of processing are to be translated from XS-3 to Communication code, two similar instructions could be used. Instruction 1 would load the Translation Table locations from the COMCD table, and Instruction 2 would cause the translation.

LABEL	OPERATION	OPERAND 1				OPERAND 2					
		I.A.	FIELD A	±	INC.	I.A.	FIELD B	±	INC.	FIELD C	± INC.
1	6	*	12		18	*	22		28	32	38
	TD		CMCD				$TR				
	TR						INMSG			+INMS	+ 1

4.4. ADDITION AND SUBTRACTION

Addition is accomplished in the UNIVAC 1005 in ascending mode using a one character adder. The LSL of OP 1 is placed in the adder. The LSL of OP 2 is then added, and the sum digit is returned to the LSL of OP 2. The adder circuitry retains the presence of a carry, if any. The next corresponding locations are added from OP 1 and OP 2 (and the preceding carry, if any) and the sum digit is returned to OP 2. This process continues until the sum digit has been returned to the MSL of *OP 2*. Thus OP 2 defines the length of *both* Operands. Subtraction is accomplished by adding the tens complement of OP 1 to OP 2.

There are two types of addition and subtraction in the UNIVAC 1005--Algebraic and absolute.

For Algebraic Add and Subtract operations, the presence of a binary 1 in the X bit position of the LSL of an Operand indicates a negative value. A negative result will have a binary 1 in the X bit position of both the MSL and the LSL. A zero result will have a binary 1 in the Y bit position of the MSL and will have the sign of OP 2. Spaces in OP 1 and OP 2 are treated as zeroes, and zeroes will be placed in result locations which do not contain 1 through 9.

For Absolute Add and Subtract, the signs of OP 1 and OP 2 are ignored. Absolute Add can produce only a positive result. Absolute Subtract is performed by complemented addition and may produce a negative result. However the negative sign indication is not stored. A zero result will have a binary 1 in the Y bit of the MSL.

Associated with the adder circuitry is a Sign Comparator. As a result of every arithmetic operation, the Sign Comparator is set to one of three conditions--plus, minus, or zero. The condition of the Sign Comparator can be tested, for sequential control purposes. (See Section 4.7.1, JC instruction.)

In the event that a carry is produced as a result of adding the OP 1 and OP 2 MSL, an Overflow indicator is set. This condition can also be tested. (See Section 4.7.1, JC instruction.)

The conditions of the Sign Comparator and the Overflow indicator are set following every arithmetic operation, and must be tested, if required, before the next arithmetic operation.

If the Operands for a required arithmetic operation are not of equal length, the shorter of the two must be transferred to rX using the TX instruction. rX can then be used as OP 1.

4.4.1. ADD ALGEBRAIC

Mnemonic: AD Mode: ASCENDING Length: 7 IA: YES

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.	FIELD C	±	INC.
1	6		12		18		22		28	32		38
	AD		1L				2M			2L		

Function:

Condition the adder circuitry according to the sign bits of OP 1 and OP 2. Ascending add the OP 1 - LSL specified by Field A to the OP 2 - LSL specified by Field C; replacing OP 2 - LSL with the sum digit. Continue until a sum digit has been placed in OP 2 - MSL specified by Field B. Set the Sign Comparator. Set the Overflow indicator, if necessary. The arithmetic is performed according to the rules for algebraic addition.

Example 1: EQUAL LENGTH OPERANDS

Problem: Add Quantity 1 from card columns 1 through 5 to Quantity 2 from card columns 6 through 10. Store the result in the Quantity 2 locations.

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.	FIELD C	±	INC.
1	6		12		18		22		28	32		38
	AD		¤5				¤6			¤10		

FIELD A; OP 1, LSL:

¤5 is the decimal address for the LSL of Quantity 1.

FIELD B; OP 2, MSL:

¤6 is the decimal address of the MSL of Quantity 2.

FIELD C; OP 2, LSL:

¤10 is the decimal address of the LSL of Quantity 2.

Example 2: UNEQUAL LENGTH OPERANDS

Given: Input Amount is in card columns 61 through 65. TOTAL is the Label assigned to an area of 10 locations.

Problem: Add Input Amount to TOTAL

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A. *	FIELD A	±	INC.	I.A. *	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	TX		¤65				¤61						
	AD		+$XR				TOTAL						

Instruction 1 TX

FIELD A; OP 1, LSL:

¤65 is the decimal address of the LSL of Input Amount in Read Input Storage.

FIELD B; OP 1, MSL:

¤61 is the decimal address of the MSL of Input Amount in Read Input Storage.

FIELD C; *Ignored*

This instruction transfers the 5 character Input Amount field to the low order 5 locations of rX. The high order locations of rX are space filled.

Instruction 2 AD

FIELD A; OP 1, LSL:

$XR is the Standard Label for the MSL of rX. +$XR specifies the LSL of rX.

FIELD B; OP 2, MSL:

The Label TOTAL specifies the MSL of the area assigned to TOTAL.

FIELD C; OP 2, LSL:

Blank addressing causes the Assembler program to use the LSL of the area assigned to TOTAL.

This instruction specifies a 10 character OP 1 and OP 2. The 10 character OP 1 will consist of the 5 low order locations of rX that were transferred from Input Amount, and the next 5 locations of rX known to contain space codes.

4.4.2. SUBTRACT ALGEBRAIC

Mnemonic: SU Mode: ASCENDING Length: 7 IA: YES

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A. *	FIELD A	±	INC.	I.A. *	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	SU		1L				2M				2L		

Function:

This instruction performs exactly the same as ADD ALGEBRAIC (Section 4.4.1.).

NOTE: OP 1 is subtracted from OP 2 and the result is delivered to OP 2.

4.4.3. ABSOLUTE ADD (ADD MAGNITUDE)

Mnemonic: AM Mode: ASCENDING Length: 7 IA: YES

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	AM		1L				2M				2L		

Function:

Ignore the signs of OP 1 and OP 2. Ascending add the OP 1 - LSL specified by Field A to the OP 2 - LSL specified by Field C, replacing OP 2 - LSL with the sum digit. Continue until a sum digit has been placed in OP 2 - MSL specified by Field B. Set the Sign Comparator. Set the Overflow indicator if necessary.

This instruction performs the same as Algebraic Add except the sign bits of OP 1 and OP 2 are ignored during the process. The X bit of OP 2, LSL is not changed by this instruction.

4.4.4. ABSOLUTE SUBTRACT (SUBTRACT MAGNITUDE)

Mnemonic: SM Mode: ASCENDING Length: 7 IA: YES

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	SM		1L				2M				2L		

Function:

This instruction performs exactly the same as ADD MAGNITUDE (Section 4.4.3.)

NOTE: OP 1 is subtracted from OP 2 and the result is delivered to OP 2. The comparator is set to the results of the subtract (+, or -, or Ø).

Although the signs are ignored for the processing if a negative result is produced, it will be stored in true (not complement) form in OP 2.

For example: OP 1 = 7, OP 2 = 3, 3-7 = 4 which is the result stored in OP 2. The X bit of the original OP 2, LSL remains unchanged.

4.4.5. ADD CONSTANT

Mnemonic: AK Mode: ASCENDING Length: 7 IA: YES

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A. *	FIELD A	±	INC.	I.A. *	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	AK		DD				2M				2L		

Function:

Add algegraic ascending beginning with location 3 of the Instruction Register; to the OP 2 - LSL specified by Field C, until OP 2 - MSL specified by Field B has received a sum digit. Add a maximum of 2 locations (DD) from the IR. If OP 2 is more than two locations in length, spaces are considered as a prefix to DD. Set the Sign Comparator. Set the Overflow indicator if necessary.

DD must always appear as a two digit constant. If the value of DD is less than ten, place a Ø in column 12. The maximum value of DD is 99.

Negative constants are specified by placing a minus sign (-) in column 12 followed by a two digit DD. The Assembler program will use this indication to place a binary 1 in the X bit position of both digits. The X bit over the right hand digit becomes the sign of the constant DD. The X bit over the left hand digit is ignored in the AK instruction.

Example 1:

Problem: Add 1 to the value of a 4 location area assigned to COUNT.

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A. *	FIELD A	±	INC.	I.A. *	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	AK		Ø1				COUNT						

FIELD A; DD:

Ø1 becomes the two characters in locations 2 and 3 of the IR when this instruction is executed.

FIELD B; OP 2, MSL:

The address of the MSL of the area assigned to COUNT is used as the OP 2, MSL of this instruction.

FIELD C; OP 2, LSL:

Blank addressing causes the Assembler program to use the address of the LSL of COUNT as the OP 2, LSL of this instruction.

When the addition is performed, it operates the same as the Add Algebraic (AD) instruction, except that if the OP 2 is more than two locations in length (as in Example 1), space codes are added to the excess locations. The carry, if any, also is added in the excess locations. The AK instruction terminates when a sum digit has been delivered to the MSL of OP 2.

Example 2:

Problem: Subtract 1 from the value of a 4 location area assigned to COUNT.

NOTE: Subtraction is performed by adding a negative constant.

LABEL	OPERATION	OPERAND 1				OPERAND 2							
		I.A.	FIELD A	±	INC.	I.A.	FIELD B	±	INC.		FIELD C	±	INC.
1	6	*	12		18	*	22		28		32		38
	AK		-01				COUNT						

FIELD A; DD:

The minus sign (-) prefix to the constant 0 1 (DD) causes the Assembler program to place a binary 1 in the X bit positions of locations 2 and 3 of this instruction. The X bit of position 3 is the sign of the constant DD. The X bit of position 2 is ignored in the AK instruction.

FIELD B; OP 2, MSL:

The address of the MSL of the area assigned to COUNT is used as the OP 2, MSL of this instruction.

FIELD C; OP 2, LSL:

Blank addressing causes the Assembler program to use the address of the LSL of COUNT as the OP 2, LSL of this instruction.

Space codes will be subtracted from the two high order locations of COUNT, and borrows will occur, if any.

4.5. COMPARE INSTRUCTIONS

Comparison in the UNIVAC 1005 may be considered to consist of two phases--performing the comparison, and testing the result of that comparison. The first phase--performing the comparison is accomplished through use of one of the Compare instructions. The purpose of a Compare instruction is to establish (set) a condition in the Comparator based on the relationship of the Operands which are compared. The condition of the Comparator is then tested by means of a Jump Test instruction. The Comparator which is set and tested by the Compare and Jump Test instructions should not be confused with the Sign Comparator which is set and tested by the Arithmetic and Jump Condition instructions.

The condition of the Comparator is set as a result (the only result) of the execution of a Compare instruction. The contents of OP 1 and OP 2 remain unchanged by a Compare instruction. The condition of the Comparator will not change until another Compare instruction is executed. The Comparator may be tested as often as required.

The Compare instructions operate in ascending mode. In the event of a signed comparison, this enables the circuitry to first examine the sign bits, which are located in the LSL of the Operands. Except for sign considerations, the result condition of the Comparator is based on the last difference, if any, encountered during the comparison.

The Operands in a Compare instruction must be of equal length. For signed comparison of unequal length Operands, the shorter of the two should be transferred to rX using the TX instruction (Section 4.3.6). In signed Compare instructions, space codes are considered equal to zeros.

The maximum Operand length in a Compare instruction is 961 locations.

4.5.1. COMPARE NUMERIC SIGNED COMPARISON

Mnemonic: CN Mode: ASCENDING Length: 7 IA: YES

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A. *	FIELD A	±	INC.	I.A. *	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	CN		1L				2M				2L		

Function:

Compare ascending the OP 1 - LSL (including sign) specified by Field A, to the OP 2 - LSL (including sign) specified by Field C. Continue until the OP 2 - MSL specified by Field B has been compared. Ignore the X and Y bit positions of OP 1 and OP 2 (except sign). Set the Comparator to one of three conditions.

OP 1 > OP 2; OP 1 < OP 2: OP 1 = OP 2.

When a signed comparison is performed, the relationship of OP 1 and OP 2 can be established if the signs (X bit position of LSL) are not alike. If the signs are alike, the values of OP 1 and OP 2 are then automatically compared to determine the result.

If the signs are alike and both plus, the Operand with the larger absolute value is the *greater*. If the signs are alike and both minus, the Operand with the larger absolute value is the *least*. If the signs are not alike, the Operand with the plus sign is the greater. Only if the signs are alike and the absolute values are the same is the result *equal*.

In Compare Numeric (CN), the zone bits (X and Y positions) of OP 1 and OP 2 are ignored (except for the consideration of the sign bits). Space codes are compared as equal to zeros.

The result of a CN instruction is set in the Comparator, and must be tested by a JT instruction (Section 4.7.2) before the execution of any subsequent Compare instruction.

Example 1:

Problem: Compare Total Deductions from card columns 5 through 10 to the Gross Pay in a 6 character area assigned to GRPAY.

LABEL	OPERATION	OPERAND 1				OPERAND 2					
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.	FIELD C	± INC.
1	6		12		18		22		28	32	38
	CN		¤10				GRPAY				

FIELD A; OP 1, LSL:

¤10 is the decimal address of the LSL of Total Deductions (column 10 of Read Input Storage).

FIELD B; OP 2, MSL:

The Label GRPAY specifies the MSL address of the area assigned to GRPAY.

FIELD C; OP 2, LSL:

Blank addressing causes the Assembler program to use the LSL of the area assigned to GRPAY as the LSL of this instruction.

1) If Total Deductions (OP 1) is more than Gross Pay (OP 2), the Comparator is set to Greater Than. OP 1 > OP 2.
2) If Total Deductions (OP 1) is the same as Gross Pay (OP 2), the Comparator is set to Equal. OP 1 = OP 2.
3) If Total Deductions (OP 1) is smaller than Gross Pay (OP 2), the Comparator is set to Less Than. OP 1 < OP 2.

4.5.2. COMPARE ABSOLUTE (MAGNITUDE) UNSIGNED COMPARISON

Mnemonic: CM Mode: ASCENDING Length: 7 IA: YES

LABEL	OPERATION	OPERAND 1				OPERAND 2					
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.	FIELD C	± INC.
1	6		12		18		22		28	32	38
	CM		1L				2M			2L	

Function:

Compare ascending the numeric bits (8, 4, 2, 1) of OP 1 - LSL (excluding sign) specified by Field A, to the OP 2 - LSL specified by Field C. Continue until the OP 2 - MSL specified by Field B has been compared. Ignore the X and Y bit positions of OP 1 and OP 2 including the sign bits. Set the Comparator to one of three conditions:

OP 1 > OP 2; OP 1 < OP 2; OP 1 = OP 2.

The comparison is made on the absolute magnitude of the numeric (8, 4, 2, 1) values of OP 1 and OP 2. The X bit positions of OP 1, LSL and OP 2, LSL (sign bits) are also excluded from consideration. Thus a plus 3 would compare equal to a minus 3. Space codes are compared as equal to zeroes.

The result of a CM instruction is set in the Comparator, and must be tested by a JT instruction (Section 4.7.2) before the execution of any subsequent Compare instruction.

Example 1:

Problem: Compare Actual Tolerance from the area assigned to ACTOL (5 locations) to the Allowed Tolerance in the area assigned to ALTOL (5 locations).

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	CM		+ACTO				ALTOL						

FIELD A; OP 1, LSL:

The plus sign (+) prefix to the Label ACTO causes the Assembler to use the address of the LSL of the area assigned to ACTOL.

FIELD B; OP 2, MSL:

The address of the MSL of ALTOL is used as the MSL of OP 2.

FIELD C; OP 2, LSL:

Blank addressing causes the Assembler program to use the address of the LSL of ALTOL as LSL of OP 2.

Tolerances are usually ± n. If ALTOL contains n, the Actual Tolerance (ACTOL) could have been calculated to a plus or minus value. The CM instruction will ignore the signs, and compare to determine if the absolute value of the Actual Tolerance is greater than the Allowed Tolerance.

1) If the Actual Tolerance (OP 1) is more than the Allowed Tolerance (OP 2), the Comparator is set to Greater Than. OP 1 > OP 2.

2) If the Actual Tolerance (OP 1) is the same as the Allowed Tolerance (OP 2), the Comparator is set to Equal. OP 1 = OP 2.

3) If the Actual Tolerance (OP 1) is smaller than the Allowed Tolerance (OP 2), the Comparator is set to Less Than. OP 1 < OP 2.

If the Allowed Tolerance is 5, an Actual Tolerance of ± 4 would compare Less Than.

4.5.3. COMPARE ALPHANUMERIC UNSIGNED COMPARISON

Mnemonic: CA Mode: ASCENDING Length: 7 IA: YES

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.	FIELD A	±	INC.	I.A.	FIELD B	±	INC.	FIELD C	±	INC.
1	6	*	12		18	*	22		28	32		38
	CA		1 L				2 M			2 L		

Function:

Compare ascending the bit pattern of OP 1 - LSL specified by Field A, to the bit pattern of OP 2 - LSL specified by Field C. Continue until the bit pattern of OP 2 - MSL specified by Field B has been compared. Exclude sign considerations, but include sign bit positions. Set the Comparator to one of *two* conditions. OP 1 = OP 2; OP 1 ≠ (unequal to) OP 2.

The purpose of the CA instruction is to determine if *all* bits in OP 1 are exactly the same as *all* bits in OP 2. There are only two results: all the bits of OP 1 *are* exactly the same (equal condition), or they *are not* the same (unequal condition). Spaces do not equal zeroes.

The comparison is performed on an ascending basis using the X Y 8 4 2 1 bits of each corresponding location of OP 1 and OP 2 beginning with the LSL. The determination of the condition which exists between the two Operands is made as soon as any difference is detected between characters in corresponding locations. If all locations have been compared and no difference is detected, an equal condition exists.

Example 1:

Given: Employee Number (5 locations) and Employee Name, last name first (24 locations) from a Payroll Master Card have been stored in two adjacent areas assigned to MNUM and MNAME.

MNUM	MNAME

Detail cards containing the Employee Number in columns 1 through 5 and the first four letters of the last name of the employee are in columns 6 through 9.

Problem: Compare Employee Number and Name from the detail card to MNUM and the first four locations of MNAME.

LABEL	OPERATION	OPERAND 1				OPERAND 2							
		I.A.	FIELD A	±	INC.	I.A.	FIELD B	±	INC.		FIELD C	±	INC.
1	6	* 12			18	* 22			28		32		38
	CA		¤9				MNUM				MNAME	+	3

FIELD A; OP 1, LSL:

¤9 is the decimal address of the Read Input Storage location which contains the information from card column 9.

FIELD B; OP 2, MSL:

The Label MNUM causes the address of the MSL of the area assigned to MNUM to be used as the MSL of this instruction.

FIELD C; OP 2, LSL:

MNAME + 3 is the address of the location of MNAME that contains the fourth letter of last name stored from the master card. This address becomes the LSL of the CA instruction.

Solution: The 5 locations of MNUM and the first 4 (high order) locations of MNAME become a 9 location OP 2, and are compared to the information from card columns 1 through 9 in Read Input Storage. The assignment of the two Labelled Storage areas (MNUM and MNAME) to adjacent memory locations is accomplished by proper use of Declarative instructions.

The result of the CA instruction is set in the Comparator and can be tested in the next or some subsequent instruction (provided no intervening Compare instruction is executed).

4.5.4. COMPARE CONSTANT UNSIGNED COMPARISON

Mnemonic: CK Mode: ASCENDING Length: 7 IA: YES

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	CK		KK				2M				2L		

Function:

Compare alphanumeric ascending the bit pattern of KK specified in Field A; beginning with the bit pattern stored in the location 2L specified by Field C; continuing until the bit pattern stored in the location 2M has been compared. Compare *all* bits. If 2M and 2L specify the same address, a one character comparison is made. If 2M and 2L specify more than a 2 location OP 2, space codes (binary zeroes) are compared to the excess positions of OP 2. Set the Comparator to one of two conditions: KK = OP 2; KK ≠ OP 2.

When a CK instruction is brought to the IR for execution, the two alphanumeric characters (KK) will occupy locations 2 and 3 of the IR. These two locations are used similar to an OP 1 in a Compare Alphanumeric instruction (Section 4.5.3.) However, the Operation code CK will cause the comparison to continue after the second character comparison has been made. Space codes are compared to any additional locations of OP 2.

The CK instruction is an unsigned bit-for-bit compare instruction. Spaces do not equal zeroes, and sign considerations are ignored. OP 2 will usually be a one or a two character operand.

The instruction is to be used to test for the presence of whole characters in storage location. (The Jump Compare (JK) instruction (Section 4.7.5) can be used to test for the presence of specific bits in a storage location.) Binary coding (Section 4.3.5.3.) may be used, but should not be necessary, since this is a character comparison.

Example 1:

Problem: Test the information in card column 80 for a 3.

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	CK		¢3				¤80				¤80		

FIELD A; KK:

The ⊬ 3 in the KK positions of Field A will be in locations 2 and 3 of the IR when this instruction is executed.

FIELD B; OP 2, MSL:

¤ 80 is the decimal address of the location in Read Input Storage which contains the information from card column 80.

FIELD C; OP 2, LSL:

¤ 80 is the decimal address of the location in Read Input Storage which contains the information from card column 80.

Solution: Since the OP 2 - MSL and OP 2 - LSL specify the same location, a one character comparison is made, using the 3 from location 3 of the IR (the right-hand K). The space code (⊬) is required to position the 3 so that it is in the right-hand K position (column 13 of the form). If card column 80 contained only a 3 punch, the Comparator is set to equal. If card column 80 contained any other punches (or none at all), the Comparator is set to unequal. The Comparator is tested by use of the Jump Test (JT) instruction.

Example 2:

Given: A two location counter is being arithmetically reduced by 1. The counter is stored in the two locations assigned to COUNT.

Problem: Test the value of COUNT to see if it is equal to zero.

LABEL	OPERATION	OPERAND 1				OPERAND 2							
		I.A. *	FIELD A	±	INC.	I.A. *	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	CK		?Ø				COUNT						

FIELD A; KK:

The characters ? Ø in the KK positions of Field A become locations 2 and 3 of the IR when this instruction is executed.

FIELD B; OP 2, MSL:

The address of the MSL of the area assigned to COUNT is used as the OP 2, MSL of this instruction.

FIELD C; OP 2, LSL:

Blank addressing causes the Assembler program to use the address of the LSL of the area assigned to COUNT as the OP 2, LSL of this instruction.

Solution: The contents of COUNT are being arithmetically reduced by 1. When the value of COUNT is reduced to zero, the operation of the Arithmetic unit of the UNIVAC 1005 will cause a Y bit to be placed over the MSL of the result. The internal code for the question mark (?) is the same as an XS-3 zero with a Y bit. The CK instruction performs a bit-for-bit comparison. When COUNT is reduced to zero, this CK instruction will set the Comparator to equal.

NOTE: Addresses can also be specified in the KK portion of a CK instruction by using the same notation described in Sections 4.3.5.1.; 4.3.5.2.; and 4.3.5.3.

4.6. CONDITION INDICATORS

The UNIVAC 1005 provides for two program controlled sensing switches, Sense #1 and Sense #2. By using the Set Condition (SC) instruction, the programmer can turn these switches ON (Set to 1) or OFF (reset to Ø) during the execution of a program. The condition of the Sense switches can be used to control the sequence of the execution of instructions during the program through use of the Jump Condition (JC) instruction. The Jump Condition instruction is used to test for the ON condition of the switches. If the Sense switch being tested is ON (Set), the transfer of control will occur. If the Sense switch being tested is OFF (reset), the program proceeds with the next sequential instruction (NI).

There are other uses for the Set Condition and Jump Condition instruction. A complete description of both instructions is given below.

4.6.1. SET CONDITION

Mnemonic: SC Mode: SPECIAL Length: 5 IA: NO

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	SC		CC										

Function:

Set or reset the Conditions or Controls which correspond to each bit position of CC which contains a binary 1 as specified in Field A.

Each of the bit positions of CC correspond to a Condition Indicator or a Control setting. The presence of a binary 1 in a bit position of CC will cause the Condition or Control to be set or reset by the SC instruction. The presence of a binary zero in a bit position will not change the status of a Condition or Control.

Although coded in Field A (to simplify source coding), the bit patterns of CC must occupy locations 4 and 5 (the B portion) of the object instruction. Locations 2 and 3 of the object instruction are ignored by the UNIVAC 1005, and should be blank. Locations 4 and 5 constitute bit positions 19 through 24 and 25 through 30 of the instruction.

The Conditions and Controls which correspond to the bit position of CC are as follows:

BIT POSITION	*CONDITION/CONTROL*
19 (X)	SET ODD PARITY (See Section 4.13.) (Magnetic Tape)
20 (Y)	SET EVEN PARITY (See Section 4.13.) (Magnetic Tape)
21 (8)	SET SENSE 2 (ON)
22 (4)	SET SENSE 1 (ON)
23 (2)	RESET SENSE 2 (OFF)
24 (1)	RESET SENSE 1 (OFF)
25 (X)	RESET PPT for channel 8 punching
26 (Y)	SET PPT for channel 8 punching
27 (8)	SET SERVO 2 (See Section 4.13.)
28 (4)	SET SERVO 1 (See Section 4.13.)
29 (2)	SET CONSOLE INDICATOR 2 (ON) and HALT
30 (1)	SET CONSOLE INDICATOR 1 (ON) and HALT

The Condition Indicators and Controls can be set (or reset) individually or in multiples as the programmer requires.

NOTE: Caution should be used when coding multiple bits in CC, in order to prevent illogical bit patterns which require the UNIVAC 1005 to establish opposing conditions. The results of such a conflict are unpredictable.

Binary coding is normally used to specify a *multiple* bit pattern for CC (See Section 4.3.5.3.), in which case Field A must always contain a number sign (#) in column 12 followed by four octal digits for binary coding.

The UNIVAC 1005 Assembly System provides the following mnemonic Switch Names if only a *single* Condition or control is to be set (or reset) by the SC instruction. A number sign (#) must appear in column 12 followed by the two-place mnemonic Switch Name in columns 13 and 14.

SWITCH NAME	*BIT POSITION*	*CONDITION/CONTROL*
# SO (Alpha)	19	SET ODD PARITY
# SE	20	SET EVEN PARITY
# + 2	21	SET SENSE 2 (ON)
# + 1	22	SET SENSE 1 (ON)
# - 2	23	RESET SENSE 2 (OFF)
# - 1	24	RESET SENSE 1 (OFF)
# S2	27	SET SERVO 2
# S1	28	SET SERVO 1
# H2	29	CONSOLE INDICATOR 2 and HALT
# H1	30	CONSOLE INDICATOR 1 and HALT
# H3	29, 30	CONSOLE INDICATORS 1 & 2 & HALT

It should be noted that the Switch Names for the Sense switches have a plus sign (+) for set (ON) and a minus sign (-) for reset (OFF); the Controls for magnetic tape operations have a prefix of the letter S; and the Halt and Console Indicators have a prefix of the letter H.

Example 1:

Problem: Set Sense 1 (ON)

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.	FIELD A	±	INC.	I.A.	FIELD B	±	INC.	FIELD C	±	INC.
1	6	*	12		18	*	22		28	32		38
	SC		#+1									

FIELD A; CC:

The Switch Name #+1 causes the Assembler program to create a binary 1 in bit position 22, and the binary zeros in all other bit positions of CC.

NOTE: The octal coded constant #Ø4ØØ would produce the same CC.

FIELD B and C:

Blanks.

Example 2:

Problem: Reset Sense 2 (OFF) and Halt the UNIVAC 1005 with Console Indicator #1 ON.

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.	FIELD A	±	INC.	I.A.	FIELD B	±	INC.	FIELD C	±	INC.
1	6	*	12		18	*	22		28	32		38
	SC		#Ø2Ø1									

FIELD A; CC:

The bit pattern of the octal constant #0201 will cause a binary 1 in bit positions 23 and 30, and binary zeros in all other bit positions of CC.

NOTE: Switch Names cannot be used since multiple bits are required.

4.6.2. STOP (HALT)

Mnemonic: STOP Mode: SPECIAL Length: 5 IA: NO

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.	FIELD C	±	INC.
1	6		12		18		22		28	32		38
	STOP		(See	B	ell	ow	)					

Function:

The STOP command is a variation of the SC instruction (Section 4.6.1.) provided in the UNIVAC 1005 Assembly System to enable the programmer to easily specify and rapidly recognize those instructions which STOP (HALT) the operation of the UNIVAC 1005 during the execution of the object program. Permissible specifications in Field A are Switch Names #H1 or #H2 or #H3. One of these switch names *must* be coded in Field A.

4.7. SEQUENCE CONTROL INSTRUCTIONS

Instructions in the UNIVAC 1005 are stored, accessed, and executed in serial sequence. This sequential operation is used as long as the program does not require branching.

The accessing of instructions is under the control of the Instruction Control Counters. There are two single position counters; one for Row R32/C8/B1, and one for Column R32/C9/B1. The Column Counter is automatically incremented by five or seven as each instruction is transferred to the Instruction Register. The increment is determined by instruction type. The Row Counter is advanced by one each time the Column Counter advances beyond thirty-one and returns to one. Bank specification is also modified when the Row Counter passes 31. The Instruction Control Counters provide the Control Unit with the address of the next instruction (NI).

JUMP instructions are used in the UNIVAC 1005 to vary the normal instruction sequence. The JUMP instructions change the contents of the Instruction Control Counters if conditions specified by the JUMP instruction are present. If not, the contents of the Instruction Control Counters remain unchanged, and the normal execution sequence (NI) is followed.

The UNIVAC 1005 instruction repertoire contains seven Jump instructions for sequence variation.

4.7.1. JUMP CONDITION

Mnemonic: JC Mode: SPECIAL Length: 5 IA: NO

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.	FIELD C	±	INC.
1	6		12		18		22		28	32		38
	JC		CC				JA					

Function:

If *any* of the conditions are met which correspond to binary 1 bits of CC specified by Field A; transfer control (JUMP) to the Jump Address (JA) specified by Field B. Otherwise, execute the next sequential instruction (NI).

NOTE: In some cases, the indicators specified by 1 bits in CC are reset by this instruction.

The JA specified by Field B must be the address of the MSL of the instruction to which control is to be transferred if the Jump occurs. Since instruction addresses are assigned by the Assembler program, Field B will normally contain a programmer's Label. The current value of the ILC maintained by the Assembler program during assembly processing can also be used by specifying the dollar sign symbol ($) with Increment. The JA occupies locations 4 and 5 of the instruction.

The bit patterns of CC occupy locations 2 and 3 of the JC instruction. Locations 2 and 3 constitute bit positions 7 through 12 and 13 through 18 of the instruction. If a single binary 1 bit appears in any of the bit positions of CC *and* the corresponding condition (indicator) is set (ON), the Jump will occur. If multiple 1 bits are present in CC and any *one* of the corresponding conditions (indicators) is set (ON), the Jump will occur. Otherwise, the next sequential instruction is executed.

The conditions and indicators which correspond to the bit positions of CC are as follows:

BIT POSITION	CONDITION/INDICATOR TESTED
7 (X)	*Form Overflow*. Form Overflow is *set* when the Form Overflow position of the Forms Control Tape is sensed by the carriage. Form Overflow *is* reset when tested.
8 (Y)	*Arithmetic Overflow*. Arithmetic Overflow is *set* when the result of an Arithmetic ADD or SUBTRACT instruction exceeds the capacity of OP 2. Arithmetic Overflow *is not* reset when tested. NOTE: Form Overflow and Arithmetic Overflow cannot be tested in the same JC instruction. (See below.)
7 *and* 8 (X, Y)	*End of Tape*. End of Tape is *set* when that condition is detected by a Uniservo. The presence of binary 1's in bits 7 and 8 of CC constitute a specific test for End of Tape. If both bits are present, Form Overflow and Arithmetic Overflow are not tested or changed.

End of Tape *is* reset when tested.

NOTE: EOT is a separate indicator that is set by the Uniservo. It can be tested only through the combination bits 7 and 8 of character *2*. The indicator is reset when tested.

9 (8) *Sense 2 Set.* The Sense 2 Indicator has been *set* by the SC instruction.

Sense 2 *is not* reset when tested.

10 (4) *Sense 1 Set.* The Sense 1 Indicator has been *set* by the SC instruction.

Sense 1 *is not* reset when tested.

11 (2) *Alternate Hold 2 Set.* Alternate Hold 2 Condition Indicator is *set* (ON) when the Alternate Hold Switch # 2 console light is turned ON by depression of the switch.

Alternate Hold 2 *is not* reset when tested.

12 (1) *Alternate Hold 1 Set.* Alternate Hold 1 Condition Indicator is *set* (ON) when the Alternate Hold Switch # 1 console light is turned ON by depression of the switch.

Alternate Hold 1 *is not* reset when tested.

13 (X) *Interrupt.* Interrupt is *set* when the UNIVAC 1005 receives an Interrupt Signal from a peripheral unit.

Interrupt *is not* reset when tested.

14 (Y) *Unit Alert.* Unit Alert is *set* when a peripheral unit is in an abnormal condition.

Unit Alert *is not* reset when tested.

15 (8) *Parity Error.* Parity Error is *set* when a parity error is detected. This may be set as the result of a magnetic tape parity error, mod error (DLT1), invalid card code,or paper tape read (even parity detected).

Parity Error *is* reset when tested.

16 (4) *Sign Comparator Plus.* The Sign Comparator is *set* to Plus when the result of an Arithmetic instruction is positive, and not zero.

The Sign Comparator *is not* reset when tested.

17 (2) *Sign Comparator Zero.* The Sign Comparator is *set* to Zero when the result of an Arithmetic instruction is zero.

The Sign Comparator *is not* reset when tested.

18 (1) *Sign Comparator Minus.* The Sign Comparator is *set* to Minus when the result of an Arithmetic instruction is negative and not zero.

The Sign Comparator *is not* reset when tested.

The conditions may be tested individually or in multiples (except Form Overflow and Arithmetic Overflow) as the programmer requires. Binary coding is normally used to specify a multiple bit pattern for CC (See Section 4.3.5.3.), in which case Field A must always contain a number sign (#) in column 12 followed by four octal digits for binary coding.

The UNIVAC 1005 Assembly System provides the following mnemonic Condition Names if only a *single* condition is to be tested by the JC instruction. A number sign (#) must appear in column 12 followed by the two-place mnemonic Condition Name in columns 13 and 14.

CONDITION NAME	BIT POSITION	CONDITION
#FF	7	Form Overflow
#AF	8	Arithmetic Overflow
#+2	9	Sense 2 Set
#+1	10	Sense 1 Set
#-2	11	Alternate Hold 2 (ON)
#-1	12	Alternate Hold 1 (ON)
#IN	13	Interrupt
#UA	14	Unit Alert
#PE	15	Parity Error
#AP	16	Sign Comparator Plus
#AZ	17	Sign Comparator Zero
#AM	18	Sign Comparator Minus
#ET	7x8	End of Tape

When the JC command is executed by the UNIVAC 1005 at object time, and the jump is to occur, locations 4 and 5 of the IR are transferred to the Instruction Control Counter. Locations 4 and 5 of the IR contain the jump address specified in the JC command. These two characters become the address used by the ICC to control the access of the next instruction.

Example 1:

Problem: Transfer control to the instruction labelled FOF if Form Overflow has occurred.

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	JC		#FF				FOF						

FIELD A; CC:

#FF is the Condition Name for Form Overflow and causes the Assembler program to place a binary 1 in position 7, and binary zeroes in all other positions of CC. #4ØØØ would produce the same pattern for CC.

FIELD B; JA:

The programmer's label FOF causes the Assembler program to use the address of the MSL of that instruction as the JA of this instruction. If Form Overflow has been sensed, the jump will occur.

Example 2:

Problem: Do *not* jump to the instruction labelled ERROR if the last previously executed Arithmetic instruction produced a positive result without Arithmetic Overflow.

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	JC		#2Ø Ø3				ERROR						

FIELD A; CC:

The octal coding will produce binary 1's to test Arithmetic Overflow, Sign Comparator Zero, and Sign Comparator Minus.

FIELD B; JA:

The address of the MSL of the instruction labelled ERROR will be used as the jump address in this instruction.

Solution: The jump *will* occur if *any* of the three conditions tested does exist. The jump will *not* occur if *none* of the three conditions tested exists.

4.7.2. JUMP TEST

Mnemonic: JT Mode: SPECIAL Length: 5 IA: NO

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.	FIELD A	±	INC.	I.A.	FIELD B	±	INC.		FIELD C	±	INC.
1	6	*	12		18	*	22		28		32		38
	JT		JA1				JA2						

Function:

Test the condition established by the last previously executed Compare instruction. If equal, transfer control to instruction JA1 specified by Field A. If less than (or unequal, for alphanumeric comparison), transfer control to instruction JA2 specified by Field B. If greater than, allow control to pass to the next sequential instruction (NI).

The result of a Compare instruction is the setting of the Comparator based on the relationship of OP 1 to OP 2. If the Compare was a numeric comparison, the Comparator is set to one of *three* conditions: equal, less than, or greater than. If the Compare was an alphanumeric comparison, the Comparator is set to one of *two* conditions: equal, or unequal. The Comparator remains set until another Compare instruction is executed.

The purpose of the JT instruction is to test the setting of the Comparator, and transfer control to the addresses specified in the JT instruction, based on the setting. There are three possible settings of the Comparator as a result of a numeric comparison, and only two addresses in the JT instruction. The necessary "third" address is implied, and is the instruction which immediately follows the JT instruction. If the previously executed comparison was an alphanumeric comparison, only *two* settings are possible, so that the JT instruction will always jump to one of the two addresses specified in the instruction, following an alphanumeric comparison.

Since the Assembler program assigns addresses to instructions, the addresses specified in the JT instruction will usually be programmer's labels. The current value of the ILC maintained by the Assembler program during Assembly processing can also be used, by specifying the dollar sign symbol ($) with increment.

Example 1:

Given: A comparison has been made of the Quantity Ordered(OP 1)to the Quantity on Hand (OP 2).

Problem: If the Quantity Ordered is equal to Quantity on Hand, transfer control to the instruction labelled SAME. If the Quantity Ordered is less than the Quantity on Hand, transfer control to the instruction labelled SHIP. (If the Quantity Ordered is greater than the Quantity on Hand, control will automatically pass to the next instruction.)

LABEL	OPERATION	OPERAND 1				OPERAND 2							
		I.A.	FIELD A	±	INC.	I.A.	FIELD B	±	INC.		FIELD C	±	INC.
1	6	*	12		18	*	22		28		32		38
	JT		SAME				SHIP						

FIELD A; JA1 (equal):

If the Comparator is set to equal, the address of the MSL of the instruction labelled SAME, locations 2 and 3 of the JT instruction, is transferred to the ICC.

FIELD B; JA 2 (less than):

If the Comparator is set to less than, the address of the MSL of the instruction labelled SHIP, locations 4 and 5 of the JT instruction, is transferred to the ICC.

Example 2:

Given: The Employee Name from a detail card (OP 1) has been compared to the Employee Name from a Master Card.

Problem: If the Names are equal, transfer control to the address which follows the LSL of the JT instruction. If the names are unequal, transfer control to the instruction labelled ERROR.

LABEL	OPERATION	OPERAND 1				OPERAND 2							
		I.A.	FIELD A	±	INC.	I.A.	FIELD B	±	INC.		FIELD C	±	INC.
1	6	*	12		18	*	22		28		32		38
	JT		$	+	5		ERROR						

FIELD A; JA1 (equal):

The dollar sign symbol ($) causes the Assembler program to use the current value of the Instruction Location Counter with an increment of 5 as the JA1 address of the JT instruction.

FIELD B; JA2 (unequal):

The address of the MSL of the instruction labelled ERROR is used as the JA2 address.

Solution: The *source* language instruction taken from the card which immediately follows the JT instruction during the Assembly processing, will be assigned to the address which follows the LSL of the JT instruction. During the Assembly processing of the JT instruction, the ILC contains the address which is assigned to the JT instruction MSL. The JT instruction will occupy that location, and 4 more. Thus the ILC ($) plus an increment of 5 will be the same address that will be assigned to the instruction which is assembled after the JT.

4.7.3. UNCONDITIONAL JUMP

Mnemonic: J Mode: SPECIAL Length: 5 IA: NO

LABEL	OPERATION	OPERAND 1				OPERAND 2							
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	J		JA										

Function:

Transfer control to instruction JA specified by Field A.

When this instruction is executed, an unconditional transfer is made of the JA to the ICC. Thus the address specified in Field A becomes the address of the next instruction to be executed.

In many cases, a J instruction will be the last instruction of a sub-routine which has been entered through use of the JR instruction.

The JA address occupies locations 4 and 5 in the object instruction.

4.7.4. JUMP RETURN

Mnemonic: JR Mode: SPECIAL Length: 7 IA: NO

LABEL	OPERATION	OPERAND 1				OPERAND 2							
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	JR		JA				2L				RA		

The object language instruction produced from the source language JR above, has the format and will appear in the Instruction Register as follows:

		A	B	C
IR Locations	1	23	45	67
	JR	RA	JA	2L

Function:

Transfer ascending, locations 3 and 2 of the IR (RA) specified by Field C; beginning with the address 2L specified by Field B. Then transfer the JA specified by Field A to the ICC.

The two characters produced from the RA (Return Address) are stored in locations 2 and 3. The two characters produced from the JA are stored in locations 4 and 5. The two characters produced from 2L are stored in locations 6 and 7. These location numbers refer to the positions of the object instruction as it is stored in memory, and to the positions the instruction will occupy in the Instruction Register (IR) when the instruction is executed.

When the instruction is executed, the following operations are automatically performed:

1. Locations 2 and 3 of the IR are transferred (ascending) beginning at the memory location whose address is in positions 6 and 7 of the IR.

2. Locations 4 and 5 of the IR are transferred to the Instruction Control Counter (ICC) and are used to control the access of the instruction to be executed next.

The purpose of the JR instruction is to provide a simple method of interrupting the sequential execution of instructions in order to execute a special subroutine. After the execution of the special subroutine, control is to be returned to the instruction which sequentially follows the JR instruction.

The special subroutine is called a closed subroutine. This means that the entrance (the first instruction executed) is closed as far as the initiation of the subroutine by the sequential advance of the ICC. It also means that the exit (the last instruction executed) is closed to prevent resumption of the program through the sequential advance of the ICC.

A closed subroutine normally has the following form:

(1). The first instruction to be executed (the entrance line) has a label which is the name of the subroutine.

(2). The last instruction to be executed (the exit line) has a label, and is usually an unconditional jump instruction (Operation J).

(3). If there are multiple points within the subroutine from which exit might occur, they must transfer control to the exit line.

Using this form for closed subroutines, the JR instruction is then set up as follows:

FIELD A; JA:

Contains the label of the *entrance* line of the subroutine.

FIELD B; 2L:

Contains the address of the LSL of the exit line of the subroutine.

FIELD C; RA:

Contains the label of the address of the MSL of the instruction to which control is to be transferred *after* the subroutine has been executed.

NOTE: If the return address (RA) of the JR instruction is to be the address of the instruction stored sequentially following the JR instruction ($ + 7), Field C of the JR instruction may be left blank. The Assembler program will automatically insert the address equivalent of $ + 7 in locations 2 and 3 of the object instruction. If the return address (RA) of the JR instruction is to be anything other than $ + 7, the required address must be coded in Field C according to the rules for Assembly System addressing.

Example 1:

Given: A subroutine has been established to calculate square root. The entrance line is labelled SQRT. The exit line is a J instruction labelled EXIT.

Problem: Execute the subroutine, and return control to the next sequential instruction.

LABEL	OPERATION	OPERAND 1				OPERAND 2							
		I.A. *	FIELD A	±	INC.	I.A. *	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	JR		SQRT				+EXIT						

FIELD A; JA:

The address of the MSL of SQRT is used in Locations 4 and 5 of the object instruction.

FIELD B; 2L:

The plus sign (+) prefix to the label EXIT causes the address of the LSL of that instruction to be used as the locations 6 and 7 of this instruction.

FIELD C; RA:

Blanks in Field C cause the Assembler program to use the current value of the ILC (the address of the JR instruction) plus 7 as the address RA in locations 2 and 3 of this instruction.

Solution: At the time the JR instruction is assembled, the ILC contains the address assigned to the JR instruction. The implied $ + 7 is the same as the value which will be in the ILC when the Assembler program assigns an address to the next instruction. At execution time, this two character address of the next instruction is transferred automatically in ascending mode, from locations 3 and 2 of the IR to the LSL and LSL minus one of the instruction EXIT. These two locations of the instruction EXIT constitute the JA of an unconditional jump (J) instruction.

After setting up the RA in the exit line, the two characters from locations 4 and 5 of the JR instruction are automatically transferred to the ICC. This causes the instruction stored at SQRT to become the next instruction executed.

After the execution of the instructions in the subroutine, the instruction EXIT will be executed. The J instruction stored at EXIT now contains a JA address set up by the JR instruction. This JA address is the address of the instruction stored sequentially following the JR instruction. Thus control is returned to the main chain of the program.

When the square root subroutine is reused at another point in the program, the entry is also made by a similar JR instruction. This JR instruction will set up a new RA in the exit line (EXIT) which will return control to the instruction which follows the new JR instruction.

The effect of the JR instruction can be produced by using a TK instruction followed by a J instruction.

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	TK		:$	+	12		+EXIT	−	1				
	J		SQRT										

The instruction which is to follow the execution of the square root subroutine is the one which will be coded and assembled following the J instruction. The address which will be assigned to that instruction is the value of the ILC at the time the TK is assigned, plus 7 for the length of the TK instruction, plus 5 for the length of the J instruction--$ + 12.

The use of the JR instruction thus saves the memory locations required for the J instruction, the access time of the J instruction, and the programmer time to calculate the return address.

4.7.5. JUMP COMPARE

Mnemonic: JK Mode: SPECIAL Length: 7 IA: NO

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	JK		K				JA				2L		

Function:

Using the bit positions of K, specified in Field A, which contain binary 1 bits; test the corresponding bit positions of the memory location whose address is specified by 2L in Field C. If the bit positions of 2L which correspond to binary 1 bit positions of K *all* contain binary 1 bits, transfer control to the JA address specified in Field B. If *any* of the bit positions of 2L which correspond to binary 1 bit positions of K do not contain a binary 1 bit, proceed with the next sequential instruction (NI).

NOTE: Bit positions of 2L which correspond to binary zero positions of K are ignored by the test and may contain binary zero or binary one.

The purpose of the JK instruction is to perform a test for bit(s) present in the contents of a memory location. If K contains only a single binary 1 in the X bit position, *and* the contents of the memory location specified by Field C also contains a binary 1 in the X bit position, the jump to JA will occur. If the memory location specified by Field C does not contain a binary 1 in the X bit position, the jump will not occur, and the program continues with the next sequential instruction (NI). The presence or absence of binary 1's in the other bit positions of 2L are not involved in the operation, and have no effect on the result.

If K contains multiple binary 1 bits, then each corresponding bit position of 2L must also contain a binary 1, or the jump will not occur.

Example 1:

Problem: Test the information stored in location 80 for the presence of a binary 1 in the X bit position. If present, transfer control to CRDT.

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.	FIELD A	±	INC.	I.A.	FIELD B	±	INC.		FIELD C	±	INC.
1	6	*	12		18	*	22		28		32		38
	JK		'				CRDT				¤80		

FIELD A; K:

The apostrophe (') is the UNIVAC 1005 character which contains a binary 1 in only the X bit position.

FIELD B; JA:

The address of the MSL of the instruction labelled CRDT is used as the JA of this instruction.

FIELD C; JA:

¤80 is the decimal address of the location which contains the information from card column 80 of Read Input.

Solution: If the information stored from card column 80 contains a binary 1 in the X bit position, the jump to CRDT will occur.

Example 2:

Problem: If card column 25 contains only the letter D, transfer control to DED. If card column 25 contains anything other than the letter D, transfer control to NOTD.

D = 010 111

LABEL	OPERATION	OPERAND 1				OPERAND 2					
		I.A. *	FIELD A	±	INC.	I.A. *	FIELD B	±	INC.	FIELD C	± INC.
1	6		12		18		22		28	32	38
	JK		'				NOTD			¤25	
	JK		5				NOTD			¤25	
	JK		D				DED			¤25	
NOTD											

Instruction 1:

If card column 25 contains a binary 1 in the X bit position, it does *not* contain only the letter D, and control is transferred to NOTD.

Instruction 2:

If card column 25 contains a binary 1 in the 8-bit position (XS3 code for 5), it does not contain only the letter D, and control is transferred to NOTD.

Instruction 3:

The previous two instructions have eliminated the possibility of the presence of binary 1 in the X bit and the 8-bit positions. If the remaining positions all contain binary 1 bits, this instruction will transfer control to DED. If card column 25 did not contain a D, this instruction will *not* jump, and control will sequentially pass to the next instruction--which is NOTD.

The character K will appear in location 2 of the object instruction. Location 3 is not used. Locations 4 and 5 will contain the JA address. Locations 6 and 7 will contain the address of the location to be tested.

NOTE: Octal coding may be used in Field A to specify K of the JK instruction. Column 12 must contain a number sign (#), and columns 13 and 14 must contain two octal digits whose bit pattern must produce the required K. Columns 15 and 16 of the coding form must contain ØØ. Location 3 of the JK instruction is ignored.

4.7.6. JUMP LOOP

Mnemonic: JL Mode: SPECIAL Length: 7 IA: NO

LABEL	OPERATION	OPERAND 1				OPERAND 2					
		I.A. *	FIELD A	±	INC.	I.A. *	FIELD B	±	INC.	FIELD C	± INC.
1	6		12		18		22		28	32	38
	JL		DD				JA			$	+ 2

Function:

Subtract 1 from locations 2 and 3 of the IR (which will contain DD specified by Field A). Transfer ascending the result from locations 2 and 3 of the IR to 2L specified by Field C. If the result is positive or zero, transfer control to the JA specified in Field B. If the result is negative, proceed with the next sequential instruction.

NOTE: 2L must specify the address of the least significant D in memory, Maximum DD = 99.

$ + 2 in Field C may be left blank, in which case the Assembler will provide the $ + 2 address.

The purpose of the JL instruction is to provide a means to control the number of times a series of instructions are to be repetitively executed. The series of instructions is called a loop.

A loop is established to perform a common operation on each of a set of similar data, thus eliminating the need for a separate series of instructions for each of the set of data.

A loop may consist of four sections

(1) Initialization

(2) Processing

(3) Modification

(4) Control

The Initialization section prepares the loop to be used for the first of the repetitive executions. The Processing section consists of the operations to be performed on the data. The Modification section changes the addresses in the Processing instructions to refer to the next set of data. The Control section determines when the loop has been executed the required number of times.

The Control section of a loop in the UNIVAC 1005 will usually consist of a single JL instruction. The DD portion of the JL instruction must be set to a beginning condition in the Initialization section, due to the fact that DD is changed during the execution of the loop. Assume a loop is to be executed three times. DD will be Ø2 in the JL instruction at load time. After the execution of the Processing and Modification sections, the JL instruction is executed for the first time. 2L specifies the memory address of DD. DD is reduced by 1 in the IR and the result (Ø1) is stored at 2L, replacing the Ø2. The result is positive, so the jump occurs to JA which usually specifies the first instruction in the Processing section. After the execution of the Processing and Modification sections, the JL instruction is executed the second time. DD, which now is Ø1, is again reduced by one in the IR, and the result (ØØ) is stored at 2L, replacing the Ø1. The result is zero, so the jump occurs to JA. After the execution of the Processing and Modification sections, the JL instruction is executed the third time. DD, which now is ØØ, is

again reduced by one in the IR, and the result (−1) is stored at 2L, replacing the ØØ. This time, the result is negative. Therefore,the jump to JA does not occur, and the program continues with the instruction (NI) which sequentially follows the JL instruction. The loop has been executed exactly "DD" + 1 times, three in this example. However, the value of DD in the stored JL instruction has been changed by the execution of the loop and now reads −1. Before the loop is executed again, the value of DD must be re-stored to the correct number of times the loop is to be executed. This is usually accomplished in the Initialization section by using a TK instruction with KK equal to the initial value of DD. The 2M and 2L of the TK instruction specifies the address of the DD portion of the JL instruction.

A skeleton example of the coding for the previously described loop is as follows:

LABEL	OPERATION	OPERAND 1				OPERAND 2						
1	6	I.A.*	FIELD A 12	±	INC. 18	I.A.*	FIELD B 22	±	INC. 28		FIELD C 32	±	INC. 38
INIT	TK		Ø2				CNTRL	+	1		CNTRL	+	2
PROCS													
MODFY													
CNTRL	JL		Ø2				PROCS				CNTRL	+	2

The loop is entered by executing the instruction labeled UNIT. This sets DD equal to Ø2 in the JL instruction, labeled CNTRL. There are usually other operations required in the Initialization section. The fact that DD is Ø2 when loaded, and is reset to Ø2 for the first use of the loop should not be of concern to the programmer. Notice that Field C specifies the address of DD in memory.

NOTE: If Field C is left blank by the programmer, the Assembler automatically provides the value of $ + 2 for Field C.

The Modification section will usually involve the use of the COUNT (CC) instruction, which is explained in Section 4.8.

Section 1.6. on Indirect Addressing contains an example of the use of a loop. In the example, Instruction 6 would be a JL instruction with a DD of Ø3, and a JA of the address of Instruction 1, the first instruction of the Processing section. Instruction 5 would be replaced by the instructions necessary to perform the Modification section requirements. Instruction 1 would be preceded by the Initialization section instructions including a TK instruction which sets the DD of the JL instruction to Ø3.

4.7.7. JUMP INDIRECT

Mnemonic: JI Mode: SPECIAL Length: 7 IA: YES

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.	FIELD A	±	INC.	I.A.	FIELD B	±	INC.		FIELD C	±	INC.
1	6	* 12			18	* 22			28		32		38
	JI		IJA										

Function:

Transfer descending two locations beginning with the IJA (Indirect Jump Address) specified in Field A to the Instruction Control Counter (ICC).

NOTE: If Indirect Addressing is specified (asterisk in column 11) two levels of IJA will occur.

Field A of a Jump instruction (J) specifies the address to which control is to be transferred. Field A of the Jump Indirect instruction (JI) specifies the *address of the address* to which control is to be transferred.

The JI instruction is a pseudo-operation in the UNIVAC 1005 Assembly System. The JI instruction produces a TD command which has an OP 1 of the IJA, and an OP 2 of the ICC. OP 1 contains the address of an instruction (2 characters). When these two characters are transferred to the ICC, they are used to control the access of the next instruction. Thus control is transferred, not to the IJA, but to the address stored in the locations specified by the IJA.

Assume there are several points in a closed subroutine at which the processing is concluded. Each one of these points must return control to the instruction which follows the JR instruction used to enter the closed subroutine. This can be accomplished by coding a Jump (J) instruction at each of the ending points in the subroutine which transfers control to the exit instruction. The JA of the exit instruction was set up by the JR instruction to transfer control to the instruction following the JR instruction. However, by coding a Jump Indirect (JI) instruction at each of the ending points with an IJA that refers to the JA portion of the exit instruction, the *execution* of the second jump is eliminated.

Assume that EXIT is the label of the exit instruction of a closed subroutine. Each of the ending points would conclude with the following instruction:

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.	FIELD A	±	INC.	I.A.	FIELD B	±	INC.		FIELD C	±	INC.
1	6	* 12			18	* 22			28		32		38
	JI		EXIT	+	3								

When the JI instruction is executed, the two character address stored in locations 4 and 5 of EXIT, by the JR instruction, are transferred to the ICC. This effectively transfers control to the instruction following the JR instruction.

4.8. COUNT

Mnemonic: CC Mode: SPECIAL Length: 5 IA: NO

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A. *	FIELD A	±	INC.	I.A. *	FIELD B	±	INC.	FIELD C	±	INC.
1	6		12		18		22		28	32		38
	CC		DD				2M					

Function:

Using address arithmetic, modify by DD, specified in Field A; the two character Row, Column, and Bank address stored beginning at 2M, specified by Field B.

The purpose of the CC instruction is to provide a means of address modification according to the special logic of row, column, and bank addressing employed in the UNIVAC 1005. Addresses are specified by the full 6-bit positions of two adjacent characters. The Arithmetic Unit of the UNIVAC 1005 operates on a 4-bit numeric basis. Thus, the Add and Subtract instructions cannot be used for address modification.

The CC instruction operates on the full 6 bits of the two character address stored in the locations specified by 2M using the decimal value of DD as the modifier. If the address is to be *decremented*, a minus sign (-) is placed in column 12 of Field A, and DD is placed in columns 13 and 14. If the address is to be *incremented*, DD is placed in columns 12 and 13 and assumed to be plus. DD must always be two digits (ØØ through 99 maximum). If DD is less than ten, a Ø is placed in column 12. When DD is a decrement, the Assembler program places an X bit over both of the numeric digits in the object instruction.

2M usually specifies the address of the MSL of the A portion of another instruction which is to be modified to reference a new set of data.

Section 1.6 on Indirect Addressing contains an example of the use of a loop. The Modification section of the loop would consist of a single CC instruction that would replace Instruction 5. The CC instruction would modify the A portion of Instruction 1 by the number of locations required for each entry in constant storage. Since two characters are required for MSL and two for the LSL of each of the four fields in a Transaction, the value of DD must be 16.

Assume that Entry 1 of the constant storage area was labeled ENT1, and the Secondary Address Table was labeled SECAT. Instruction 1 would be:

LABEL	OPERATION	OPERAND 1				OPERAND 2							
		I.A.	FIELD A	±	INC.	I.A.	FIELD B	±	INC.		FIELD C	±	INC.
1	6	*	12		18	*	22		28		32		38
PROCS	TD		ENT1				SECAT				SECAT	+	15

The first execution of this instruction will cause a descending transfer beginning with the MSL of ENT1 (Entry 1 of constant storage) to the MSL of SECAT (the Secondary Address Table). The transfer will continue until the LSL of the Secondary Address Table has been filled. The remainder of the processing will then operate on Transaction 1.

Before Instruction 1 is executed the second time, the two characters in locations 2 and 3 of Instruction 1 must be modified to transfer from *Entry 2* of constant storage to the Secondary Address Table. The following CC instruction would be used to modify PROCS (Instruction 1, locations 2 and 3).

LABEL	OPERATION	OPERAND 1				OPERAND 2							
		I.A.	FIELD A	±	INC.	I.A.	FIELD B	±	INC.		FIELD C	±	INC.
1	6	*	12		18	*	22		28		32		38
	CC		16				PROCS	+	1				

Each time this CC instruction is executed, the OP 1-MSL address in PROCS is incremented by 16. This will cause successive transfers of each of the entries in constant storage. However, after the last execution of the loop, the A portion of PROCS will have an address of the location which follows the last location of constant storage. The Initialization section must then contain the following TK instruction which is executed before PROCS:

LABEL	OPERATION	OPERAND 1				OPERAND 2							
		I.A.	FIELD A	±	INC.	I.A.	FIELD B	±	INC.		FIELD C	±	INC.
1	6	*	12		18	*	22		28		32		38
	TK		:ENT1				PROCS	+	1		PROCS	+	2

The use of the colon (:) will cause the Assembler to use the address of ENT1 as KK in the TK instruction. These two characters are transferred to locations 2 and 3 of PROCS, so that each first execution of PROCS will refer to Entry 1 of constant storage.

The complete coding for the loop is as follows:

LABEL	OPERATION	OPERAND 1				OPERAND 2					
		I.A. *	FIELD A	±	INC.	I.A. *	FIELD B	±	INC.	FIELD C	± INC.
1	6		12		18		22		28	32	38
INIT	TK		Ø3				CNTRL	+	1	CNTRL	+ 2
	TK		:ENT1				PROCS	+	1	PROCS	+ 2
PROCS	TD		ENT 1				SECAT			SECAT	+ 15
	AD	*	SECAT	+	2	*	SECAT	+	4		
	AD	*	SECAT	+	6	*	SECAT	+	8		
	SU	*	SECAT	+	10	*	SECAT	+	12		
MODFY	CC		16				PROCS	+	1		
CNTRL	JL		Ø3				PROCS			CNTRL	+ 2

4.9. EDIT INSTRUCTIONS

The UNIVAC 1005 instruction repertoire includes two types of instructions which perform editing and logical operations. The logical operations provide for the deletion (erasing) of 1 bits and the insertion (superimposing) of 1 bits on the bit positions of a memory location. The edit instruction provides for the preparation of output data for printing and punching. The edit functions include such things as zero suppression, character insertion, asterisk fill, etc.

The erase function performs logical (or binary) multiplication of the corresponding bit positions of a constant and the contents of a memory location and stores the bit by bit product back in the same memory location. A binary zero in either or both corresponding bit positions of the constant and the memory location contents will produce a zero bit in the product. Only if both bit positions contain a binary 1 will the product bit be binary 1. Thus Ø x Ø = Ø, 0 x 1 = Ø, 1 x Ø = Ø, 1 x 1 = 1.

The superimpose function performs logical (or binary) addition, without carry, of the corresponding bit positions of a constant and the contents of a memory location, and stores the bit by bit sum back in the same memory location. A binary 1 in either or both corresponding bit positions of the constant and the memory location contents will produce a binary 1 as the sum. Only if both bit positions contain a binary zero will the sum be a binary zero. Thus Ø + Ø = Ø, Ø + 1 = 1, 1 + Ø = 1, 1 + 1 = 1.

The edit instruction requires the use of a pattern of characters called a *mask* to control the alteration of a field or fields of data for output. The mask pattern must be transferred to rX *before* the execution of the edit instruction.

4.9.1. EDIT LOGICAL

Mnemonic: EL Mode: SPECIAL Length: 7 IA: NO

LABEL	OPERATION	OPERAND 1				OPERAND 2							
		I.A.	FIELD A	±	INC.	I.A.	FIELD B	±	INC.		FIELD C	±	INC.
1	6	*	12		18	*	22		28		32		38
	EL		KK				1M				2M		

Function:

Erase the bit positions of the contents of location 1M specified in Field B which correspond to the bit positions of location 2 of the IR (the left hand K) which contain binary zero. Then, superimpose a binary 1 on the bit positions of 2M specified in Field C which correspond to bit positions of location 3 of the IR (the right hand K) which contain a binary 1.

NOTE: If 1M and 2M specify the same location, the erase precedes the superimpose, on that location.

The two characters (KK) in the A portion are used individually. The instruction uses the left hand character (which appears in location 2 of the IR when the instruction is executed) as the bit pattern for the erase function. The right hand character (which appears in location 3 of the IR) is used as the bit pattern for the superimpose function.

The erase function is performed on the character in the location (1M) specified in Field B. The superimpose function is performed on the character in the location (2M) specified in Field C. The erase and superimpose can be performed on the same character by specifying the same location in 1M and 2M.

NOTE: The UNIVAC 1005 Assembly System provides an instruction to erase only (EE) and an instruction to superimpose only (ES).

The character whose zero bits are used to erase is coded in column 12. The character whose one bits will be superimposed is coded in column 13.

Example 1:

Given: A card field in columns 6 through 9 will contain an X punch over column 9 *if it is negative*. This field will be transferred to print position 66 through 69. All of Print storage has been cleared as a result of the previous PRINT, EXECUTE command.

Problem: Provide the instruction which will print a minus sign from print position 70, and delete the X bit over the information from card column 9.

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A. *	FIELD A	±	INC.	I.A. *	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	EL		=-				¤9				$PR	+	69

FIELD A; KK:

The bit configuration of the equal sign (=) in column 12 is Ø11 111. The bit configuration of the minus sign (-) is ØØØ Ø1Ø.

FIELD B; 1M:

¤9 is the decimal address of the location in Read storage which contains the information from card column 9. This becomes the erase address.

FIELD C; 2M:

$PR is the Standard label for the MSL of Print storage. $PR + 69 is the address of print position 70. This becomes the superimpose address.

Solution: The binary zero in the X bit position of the equal sign will erase the X bit position of Read storage for column 9. The binary 1 in each of the other bit positions of the equal sign will prevent the erase of the corresponding bit positions of Read storage for column 9.

The fact that Print storage is cleared after a PRINT, EXECUTE allows the superimpose of the binary 1 in the 2 bit position of the minus sign, on the binary zero in the 2 bit position of print position 70. If print position 70 was not known to be all binary zeroes, two steps would be required to solve this problem.

NOTE: The TK instruction can be used to *replace* a single character in memory, rather than erase with space and superimpose with the character.

Octal coding can be used to specify the bit configuration of KK in the EL instruction. The number sign (#) is coded in column 12 followed by four octal digits. The coding for the previous solution in octal would be:

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A. *	FIELD A	±	INC.	I.A. *	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	EL		#3702				¤9				$PR	+	69

The binary configuration of octal 37 is Ø11 111. The binary configuration of octal Ø2 is ØØØ Ø1Ø.

4.9.2. EDIT ERASE

Mnemonic: EE Mode: SPECIAL Length: 7 IA: NO

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A. *	FIELD A	±	INC.	I.A. *	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	EE		K				1M						

Function:

Erase the bit positions of the character stored at 1M specified by Field B which correspond to bit positions of K, specified in column 12 of Field A, which contain binary zero.

This instruction is a pseudo-operation provided by the Assembly System and is a variation of the EL instruction. The operation code EE causes the assembler program to automatically use the address of 1M in locations 4 and 5. Characters 6 and 7 (2M of an EL instruction) are assembled as two blanks. The blank column 13 will produce binary zeroes so that no bits are superimposed.

If octal coding is used, the number sign followed by four octal digits must be specified, and the last two must be ØØ.

NOTE: If a memory location is to be cleared to binary zeroes, use a TK with KK equal to ⌀⌀.

4.9.3. EDIT SUPERIMPOSE

Mnemonic: ES Mode: SPECIAL Length: 7 IA: NO

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A. *	FIELD A	±	INC.	I.A. *	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	ES		K				2M						

Function:

Superimpose the binary 1 bit positions of K, specified in column *12* of Field A, on the corresponding bit position of the character stored in 2M specified in Field B.

This instruction is a pseudo-operation provided by the Assembly System and is a variation of the EL instruction. The operation code ES causes the Assembler program to automatically use the address of 2M in locations 6 and 7. Characters

4 and 5 (1M of an EL instruction) are assembled as two blanks. The Assembler program also automatically places K in location 3, and puts all binary 1's in location 2 so that no bits are erased.

If octal coding is used, the number sign followed by four octal digits must be specified, and the first two must be 77.

4.9.4. EDIT

Mnemonic: ED Mode: DESCENDING Length: 7 IA: YES

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A. *	FIELD A	±	INC.	I.A. *	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	ED		1M				2M				2L		

Function:

Initialize a two phase instruction of editing data in X register and transfer the results to Operand 2.

Phase I, edit descending beginning at OP 1-MSL according to the corresponding special characters in the edit mask in rX; when a sentinel (≠) or the LSL of the X register (R32C31B2) is detected in the edit mask, editing is terminated.

Phase II transfers the X register in a descending mode to Operand 2 until OP 2-LSL is sensed.

NOTE: OP 2 should contain the exact number of characters that are in the edit mask excluding the sentinel position. The sentinel position of the mask will be space filled upon completion of Phase I.

The Edit (ED) instruction performs a location by location transfer of the appropriate characters from OP 1 to the locations of rX according to the edit functions specified by special characters placed in corresponding locations of rX and transfers rX to OP 2. The length of OP 2 is dictated by the length of the mask in rX (31 locations maximum). The result character to be stored in OP 2 will be either the character from OP 1 processed according to the edit action specified by the special characters in the mask, or a character to be inserted from the mask. The mask is usually stored as a constant and must be transferred to rX before the execution of the ED instruction. The mask in rX is changed by the ED instruction.

Indirect Addressing can be specified for the address of the data to be edited (OP 1) and for the destination of the edited output (OP 2).

The following symbols are used as special characters in the mask to specify the edit action:

\ (Backward Slash).

When this special character is detected in the mask, it will cause zeros, COMMAS, and spaces from OP 1 to be replaced with asterisks (*) until a character which is not a zero, COMMA,or a space is recognized from OP 1. The asterisk fill operation started by the backward slash can be restarted during the ED instruction by placing additional backward slashes in the mask.

Δ (Delta Symbol).

When this special character is detected in the mask, it will cause zeros, commas and spaces from OP 1 or rX to be replaced with space codes (binary zeroes) until a character which is not a zero, comma,or space is recognized from OP 1. The zero suppress operation started by the Delta symbol can be re-started during the ED instruction by placing additional Delta symbols in the mask.

NOTE: If a comma appears in the mask and a previously started asterisk fill or zero suppress function has not detected a significant character, the comma will be replaced by an asterisk or a space code.

⌑ (Lozenge).

When this character is detected in the mask, it allows the current corresponding character from OP 1 to be transferred to OP 2. If asterisk fill or zero suppress has been previously started, the character from OP 1 is handled accordingly.

≠ (Unequal Symbol).

When this character is sensed in the mask (Phase I), it will be space filled and terminates Phase 1.

All characters other than backward slash, Delta, lozenge, and unequal which appear in the mask will be inserted in OP 2 prior to the current character from OP 1.

The function of the ED instruction when executed is to first examine the MSL of rX. At this precise point in time, the current character from OP 1 is OP 1-MSL. What happens next is based on the character in the MSL of rX. Basically, one of two possibilities exist: the character in the MSL of rX is not one of the four special characters, or it is one of the four special characters.

If the character in the MSL of rX is *not* one of the four special characters, the character in rX will remain in the MSL of rX. The next character in the mask will then be examined. At this point in time, the current character from OP 1 is still the OP 1-MSL. If the second character in the mask is not one of the four special characters, it will also remain. This examination will continue until one of the four special characters is detected. (If the mask does not contain a backward slash, a Delta, or a lozenge, the ED becomes a TD of rX to OP 2.)

When a special character is detected in the mask (other than the unequal symbol), the current character from OP 1 (which in this example, is still the OP 1-MSL) is transferred to the next most significant location of rX. When this occurs, the next most significant character of OP 1 becomes the current character from OP 1. Mask examination is also advanced to the next most significant location of rX. This operation is continued until the first of the following occurs:

(1) The mask examination detects the unequal signal. The EDITING terminates, and the current character from OP 1 is not handled.

(2) The LSL of rX is detected. The LSL position is handled and editing is terminated.

The length of OP 2 cannot be greater than the length of the mask (maximum 31 locations). If OP 2 is smaller than the mask, the excess positions of the mask are not transferred. If the mask is less than 31 characters in length, the unequal sign should be used as the least significant character in the mask, but is not counted in the length of the mask or the length of OP 2.

The length of OP 1 is determined by the number of locations in the mask which contain special characters other than the unequal sign, assuming that the length of OP 2 and the mask are the same. The presence of any of the special characters in OP 1 will not change the operation of the ED instruction.

Example 1:

Given: A ten location operand labeled SUM which contains dollars and cents.

Problem: Print the contents of SUM as follows:

1. Precede with a dollar sign.
2. Zero suppress.
3. Insert commas.
4. Insert the decimal point.
5. Print the word "TOTAL" two print positions to the left of the dollar sign, and starting in print position 1Ø1.

Contents of SUM:

0	0	0	1	2	4	6	8	9	3
1	2	3	4	5	6	7	8	9	10

Constant stored at MASK:

T	O	T	A	L	ъ	ъ	$	Δ	¤	,	¤	¤	¤	,	¤	¤	¤	.	¤	¤	≠
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22

Instructions:

LABEL	OPERATION	OPERAND 1				OPERAND 2							
		I.A. *	FIELD A	±	INC.	I.A. *	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	TD		MASK				$XR				$XR	+	21
	ED		SUM				$PR	+	100		$PR	+	121

Instruction 1.

The mask is transferred from MASK to the high order end of rX.

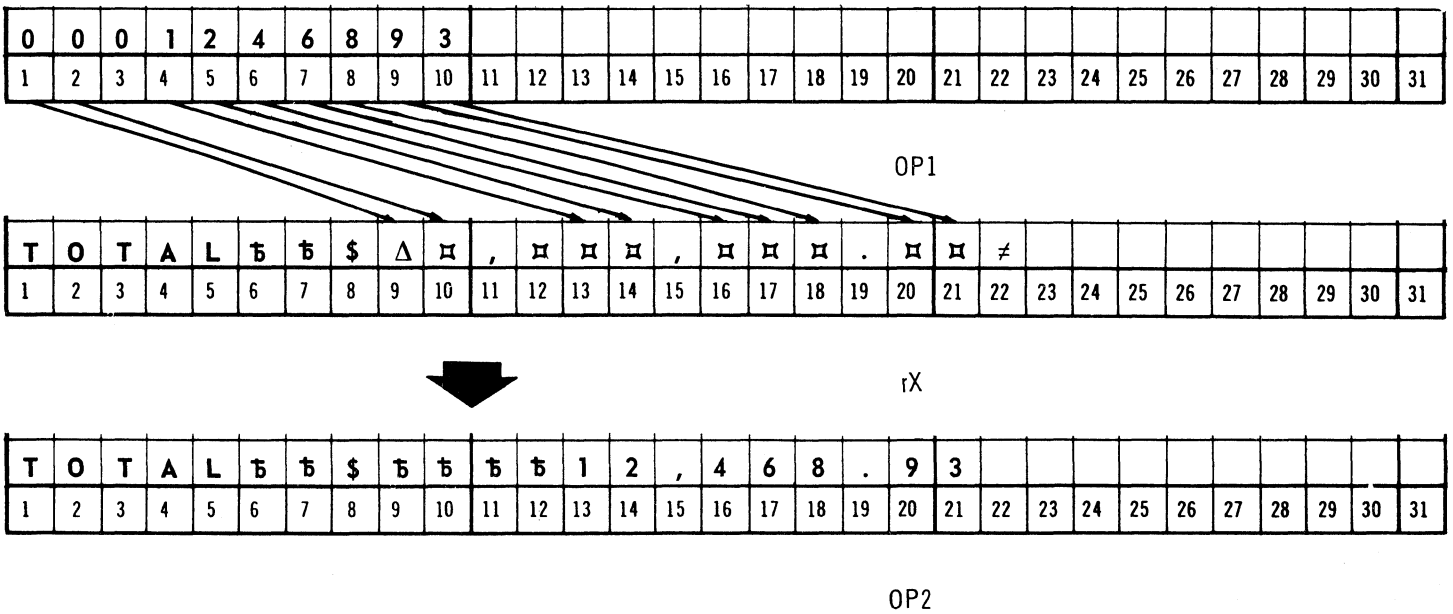
Instruction 2. (See illustration below).

The MSL of SUM becomes the current character from OP 1. The MSL of rX contains the letter "T" which is not a special character. The "T" therefore remains in the MSL of rX. The examination and editing continue until the delta is reached. The MSL of OP 1 is still the current character of OP 1.

Delta (location 9) is the character in the mask. This starts the zero suppress function, and causes the transfer of the current character from OP 1. The Ø from OP 1-MSL is changed to a space code, and transferred to location 9 of rX. The next position of the mask (location 1Ø) contains a lozenge which causes the new current character from OP 1 (location 2) to be transferred to rX, and is found to contain a comma. Since zero suppress is still active, the comma is changed to a space in rX. The next location of the mask (location 12) contains a lozenge which causes the transfer of the current character from OP 1 (location 3) to be transferred to rX (with zero suppression). The next position of the mask (location 13) contains a lozenge which causes the current character from OP 1 (location 4) to be transferred to rX. However, the current character from OP 1 is not a zero or a space. This terminates the zero suppress function. Transfers of OP 1 to rX continue for the next two locations. At this point, the location is the mask (location 15) contains a comma. Since zero suppress is no longer active, the comma remains in rX. The next three locations from OP 1 (locations 6, 7, and 8) are transferred to rX. The next location of the mask (location 15) contains a period, which remains in rX. The next two locations from OP 1 (locations 9 and 10) are transferred to rX.

At this point, the sentinal (≠) in the mask is sensed. The sentinel position is space filled and editing is terminated. The contents of the X register are now transferred to OP 2 in a descending mode until OP 2-LSL is sensed.

NOTE: After editing, the X register contains the result to be placed in Operand 2.



4.10. DECLARATIVE INSTRUCTIONS

Declarative instructions are instructions from the programmer to the Assembler program to control its operation during assembly processing. No object language instructions are produced from a source language Declarative, but the source language information is automatically carried through to the object deck in the event re-assembly is desired at a later time.

Declarative instructions include provision for establishing constants in the object program. The output from a Declarative which sets up a constant will include the constant, and instructions to the Load routine for loading the constant.

The Assembler program maintains two counters which control the assignment of memory locations and addresses of the instructions, and the working storage for the object program. The Instruction Location Counter (ILC) is incremented by the Assembler program when allocating memory for instructions and in-line constants. The Data Location Counter (DLC) is decremented by the Assembler program when allocating memory for working storage and Declarative constants.

These counters are not hardware, but are program locations established and updated by the operation of the Assembler program. When the Assembler program is loaded, the initial value of the ILC is set to the address of the first location following the Read/Punch Punch storage area--Row 15, Column 19, Bank 1. Unless otherwise specified, this will be the address assigned to the MSL of the first object language instruction assembled. A Declarative instruction is provided which allows the programmer to place a new value in the ILC.

The initial value of the DLC is the address of the last location in memory--Row 31, Column 31, Bank 2 for a two bank system, or Row 31, Column 31, Bank 4 for a four bank system. No provision is made for specifically placing a new value in the DLC. The value of the DLC is decremented as working storage areas and Declarative constants are called for by the source language program.

4.10.1. DEFINE INSTRUCTION LOCATION COUNTER

Mnemonic: DL

LABEL	OPERATION	OPERAND 1				OPERAND 2							
		I.A. *	FIELD A	±	INC.	I.A. *	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	DL		XXXXX										

Function:

Set the value of the Instruction Location Counter to the *known* address specified in Field A.

NOTE: A plus or minus increment may also be used.

When the current (or initial) value of the ILC is not the address desired by the programmer for the allocation of the next instruction, the DL Declarative is used to establish a new value in the ILC before the next instruction is allocated. Caution must be exercised by the programmer when changing the value of ILC to insure that an address is not assigned more than once by the Assembler program.

The current value of the ILC (the symbol $) may also be used in Field A.

Example 1:

Given: The Read/Punch Unit is not used by the object program.

Problem: Start the allocation of instructions in the first location following the Print storage area.

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.	FIELD A	±	INC.	I.A.	FIELD B	±	INC.		FIELD C	±	INC.
1	6	*	12		18	*	22		28		32		38
	DL		$P1										

FIELD A:

$P1 is the Standard Label for the MSL of the Punch (Read/Punch Read) storage area.

Solution: This card must be the first source language input card which effects the ILC. The known value of the Standard Label $P1 (R1Ø, C14, B1) is stored by the Assembler program in the ILC, and is used as the address of the first instruction assembled. $1Ø14B1 or ¤293 could also have been used in Field A to produce the same value in the ILC.

NOTE: Programmer's labels cannot be used in Field A.

Example 2:

Given: Code Image and Punch storage are not used by the object program.

Problem: Use these areas for instructions.

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.	FIELD C	±	INC.
1	6		12		18		22		28	32		38
	DL		¤81									
START												
	J		$P1									
	DL		$P1									

Instruction 1;

This instruction sets the value of the ILC to the address of the first location following Read Storage.

Instructions 2 through n.

These lines of coding beginning at START are assigned to addresses beginning with ¤81 and continuing according to the number and length of the instructions. The programmer must determine when the ILC will have been incremented to the point where it contains the value which is six less than the value of the first location of Print storage. At that point,he must code the J instruction to prevent the sequential advance of the ICC at object time from securing instructions from the Print storage area. He must then code the second DL declarative to prevent the ILC from assigning instructions to addresses in Print storage during Assembly processing.

Example 3:

Given: An unconditional jump instruction.

Problem: Store a 2 character constant, XY, in the ignored 2nd and 3rd character of the assembled jump instruction.

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.	FIELD C	±	INC.
1	6		12		18		22		28	32		38
	J		ADDR									
	DL		$	-	4							
	*		2		XY							
	DL		$	+	2							

Line 1: The unconditional jump instruction.
Line 2: Set the ILC to the second character of the jump instruction.
Line 3: Define an In-Line Constant of XY(2 characters) to be assigned in the second and third characters of the jump instruction.
Line 4: Set the ILC to the address following the jump instruction.

NOTE: The value of $ as used in Line 2 differs from the value of $ as used in Line 4.

4.10.2. DEFINE AREA

Mnemonic: DA

LABEL	OPERATION	OPERAND 1				OPERAND 2							
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	DA		XXX										

Function:

Allocate an area of working storage with a length as specified in Field A. Subtract the number of locations in the area from the value of the DLC. Use the new value + 1 of the DLC as the MSL of the area.

NOTE: This instruction will usually have a Label to be used for referencing by the programmer.

The loading of the Assembly program sets the initial value of the DLC to the last location of memory--R31/C31, B2 or R31/C31, B4. When an area of working storage is required, the programmer uses the DA declarative to cause the Assembler program to allocate the required number of locations. This allocation begins at the last location of memory and proceeds toward the first location of memory by decrementing the value of the DLC by the size of the area required.

The number of locations to be allotted by the DA declarative is coded in columns 12, 13, and 14 of Field A. The maximum number of locations which can be allotted by a single DA instruction is 961.

The use of the DA declaratives to allocate working storage does not prevent the same locations from being assigned to instructions. In the event of a lengthy program, the assignment of instructions may increment the value of the ILC until it is greater than the value of the DLC. The programmer can recognize whether or not this situation has occurred by examination of the object printout from the Assembly processing.

The DA declaratives should be coded on a separate page of coding paper to enable the programmer to obtain a "picture" of the full area of working storage. It will also enable the programmer to assess the possibility of exceeding the

total memory capacity for working storage and instructions. Seven times the number of instructions (the maximum number of locations required for instructions) added to the ILC starting value will produce the ILC maximum value. The sum of the numbers in Field A of the DA and DC declaratives when subtracted from the starting DLC value will produce the lowest value of the DLC. The number of locations set aside for in-line constants and Indirect Address constant, if any, should be included.

The Define sub-field declarative is used in conjunction with the DA declarative to provide definition and labelling of sub-fields *within* the locations allocated by a DA declarative (See Section 4.10.2.1.)

The contents of the memory locations allocated by a DA declarative are not entered at object load time, and are not automatically set to blanks or zeroes.

Example 1:

Problem: Establish a working storage area of ten locations, to be used in the accumulation of a total.

LABEL	OPERATION	OPERAND 1				OPERAND 2							
		I.A.	FIELD A	±	INC.	I.A.	FIELD B	±	INC.		FIELD C	±	INC.
1	6	*	12		18	*	22		28		32		38
TOTAL	DA		10										

Solution: Assume that this is the first DA declarative to be assembled for an object program to be run on a two bank UNIVAC 1005. The loading of the Assembly program set the initial value of the DLC to ¤1922. The area TOTAL would be assigned to locations ¤1913 to ¤1922 inclusive. The label TOTAL is then used to specify the MSL of this 10 location area throughout the program. The new value of the DLC is ¤1912 (¤1922 minus 10).

4.10.2.1. DEFINE SUB-FIELD

The Define Sub-field declarative can only be used following a DA declarative. The Define Sub-field declarative does *not* change the value of either the ILC or the DLC. It does *not* allocate memory.

The purpose of the Define Sub-field declarative is to establish labels for fields within an area that has been allocated by the immediately preceding DA declarative. The address assigned to the label of a Define Sub-field declarative is determined by the Assembler program by calculating its location relative to the MSL of the area allocated by the DA declarative.

The Define Sub-field declarative is specified by placing a minus sign (-) in column 6. Complete specification of the Define sub-field declarative is as follows:

LABEL	OPERATION	OPERAND 1				OPERAND 2						
1	6	I.A.*	FIELD A 12	±	INC. 18	I.A.*	FIELD B 22	±	INC. 28		FIELD C 32	±	INC. 38
			XXX				YYY						

Where XXX is the character number within the DA field of the least significant (right most) location of the sub-field.

YYY is the number of locations within the sub-field.

It is recommended that sub-fields be defined in their order of appearance (left to right) within the DA field, however there is no restriction on the sequence of defining sub-fields. Sub-fields may be overlapped, either partially or wholly, without producing inconsistent object code.

Example:

Given: A master card is to be read and transferred to working storage for the subsequent processing of detail cards.

Problem: Define an 80 location area with sub-fields according to the following master card format:

Card Columns	1 through 7	Stock Number
	8 through 30	Description
	31 through 40	Balance on Hand
	41 through 50	Shipped to Date
	51 through 60	Shipped this Week
	61 through 70	On Order
	71 through 80	Minimum Level

LABEL	OPERATION	OPERAND 1				OPERAND 2						
1	6	I.A.*	FIELD A 12	±	INC. 18	I.A.*	FIELD B 22	±	INC. 28		FIELD C 32	±	INC. 38
MASTRD	DA		80										
STKNO	-		7				7						
DESCR	-		30				23						
BALOH	-		40				10						
TPDAT	-		50				10						
THSWK	-		60				10						
ONORD	-		70				10						
MIN	-		80				10						

Solution: The DA instruction allocates an area of 80 locations with an MSL identified by the label MASTR. When the Master card is to be transferred from Read Input storage, the following single instruction can be used to transfer all 80 columns.

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.	FIELD A	±	INC.	I.A.	FIELD B	±	INC.	FIELD C	±	INC.
1	6	*	12		18	*	22		28	32		38
	DD		$R1				MASTR					

Subsequent coding can then refer to the MSL of the fields in the Master card by using the labels of the Define sub-field declaratives.

4.10.2.2. Subfields of Specific Fixed Address Areas

There is a second use of the DA declarative which *does not allocate* working storage area. The purpose of this second use of the DA declarative is to enable the programmer to establish sub-fields for specific areas of memory. Since these areas are "known" to the Assembler program, *no allocation* is made by the DA declarative.

The DA declarative specifies a known address (standard label, row column address or decimal address) in Field A, with or without an increment, and is followed by Define sub-field declaratives. The Define sub-field declaratives are used to define sub-fields within the area of the known address according to the field configuration required by the programmer. The labels assigned by the programmer to the Define sub-field declaratives can then be used to reference these fields. The value of the DLC is not changed. The "leftmost" character (the MSL) of the DA field is considered to be the character whose address is specified by Field A of the DA statement; for this use of the DA statement no "rightmost" (LSL) address is assigned to the DA field.

Example:

Given: A card format as follows:

Card Columns	1 through 10	ID Number
	11 through 20	Quantity 1
	21 through 40	Quantity 2
	41 through 60	Quantity 3
	61 through 80	Quantity 4

Problem: Define Read Input storage giving labels to the card fields.

LABEL	OPERATION	OPERAND 1				OPERAND 2						
1	6	I.A. *	FIELD A 12	±	INC. 18	I.A. *	FIELD B 22	±	INC. 28		FIELD C 32	±	INC. 38
	DA		$R1										
ID	-		10				10						
Q1	-		20				10						
Q2	-		40				20						
Q3	-		60				20						
Q4	-		80				20						

There is no restriction on the number of times a known address may be used to define sub-fields in this manner.

4.10.3. DEFINE CONSTANT

Mnemonic: DC

LABEL	OPERATION	OPERAND 1				OPERAND 2						
1	6	I.A. *	FIELD A 12	±	INC. 18	I.A. *	FIELD B 22	±	INC. 28		FIELD C 32	±	INC. 38
	DC		DDD	±	CCC	C	C						

The purpose of the DC declarative is to enter constant values in the object program at load time. The DC declarative usually has a label that is used to specify the MSL of the constant. The address assigned to the constant is determined by subtracting the number of locations in the constant, specified by Field A, from the current value of the DLC plus 1. The Assembler output card in the object deck will contain the constant, and the instruction to the Load routine to enter the constant in the assigned address.

The constant is coded beginning in column 18 of the coding form. The sign of the constant (plus or minus) is coded in column 17 and is not included in the length of the constant. The maximum length of a negative constant is 25 locations. The maximum number of characters which can be specified in a single card for a positive constant is 44. A blank in column 17 will be interpreted as a plus.

Provision is made for the continuation of a positive constant of more than 44 characters by placing a comma (,) in column 6 of the next card, and continuing the characters in column 18. A maximum of 961 characters can be entered as a single constant in this manner. The length of the constant is specified in only the DC card, and the label (if any) of the DC card refers to the entire constant. A label (if any) present on a constant continuation card will refer to only those characters appearing on the constant continuation card.

The action of the Load program is to use the length specified in Field A to determine the number of the columns, beginning at column 18, which are to be stored in memory. If the number of characters punched as the constant does not agree with the number in Field A, the number of columns specified by Field A is used.

Example 1:

LABEL	OPERATION	OPERAND 1				OPERAND 2							
1	6	I.A. *	FIELD A 12	±	INC. 18	I.A. *	FIELD B 22	±	INC. 28		FIELD C 32	±	INC. 38
MASK	DC		22		TOTA	L	ØØ$Δ¤		,¤¤	¤	,¤¤¤.	¤	¤+
HEAD	DC		76		ØØS	T	OCKØN	O	.ØØ	Ø	DESC		
					RIP	T	ION¤¤¤		etc				
LIMIT	DC		7	-	999	9	999						

MASK is the label of the constant used as the mask in Example 1 of the ED instruction.

HEAD might be used to print the headings on an inventory report. (The actual constant would continue through column 61 before starting again in column 18 of the continuation card.

LIMIT is a 7 digit constant with a value of minus 9999999 which would be stored as 999999R.

If it is desired, the DC declarative can be used to allocate a working storage area whose initial contents are space codes.

LABEL	OPERATION	OPERAND 1				OPERAND 2							
1	6	I.A. *	FIELD A 12	±	INC. 18	I.A. *	FIELD B 22	±	INC. 28		FIELD C 32	±	INC. 38
TOTAL	DC		1Ø										

The area assigned to TOTAL will be entered with the 1Ø space codes from columns 18 through 27 of the DC card. Caution should be exercised by the programmer when using this technique since the area will not be cleared if a restart is required.

If the programmer codes the constants on a separate page and enters the source language cards for constants *after* the cards from the DA page, a single TK instruction can be used to clear all of working storage to spaces.

Example 3:

Given: The above recommended procedure is followed, and the last source language DA card has a label of LAST.

Problem: Clear working storages to space codes.

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.	FIELD C	±	INC.
1	6		12		18		22		28	32		38
	TK		ØØ				LAST			¤1922		

Solution: The allocation of all preceding DA cards began with a DLC value of 1922 (two bank system). The value of the DLC was reduced by each DA card including the card labeled LAST. Thus,the MSL of all working storage is also the MSL of LAST. The 2 space codes from the TK are transferred beginning at the LSL of OP 2 (¤1922). Since OP 2 is larger than two locations, the remainder of OP 2 is cleared to spaces.

NOTE: This procedure may not be used if the number of locations defined by DA statements exceed 961.

4.10.3.1. In-Line Constants

The term in-line constant refers to the allocation of a constant in the portion of memory usually allocated to object language instructions.

The UNIVAC 1005 Assembly System provides for in-line constants through use of the asterisk (*) in column 6 of the coding form. The remainder of the in-line constant format is the same as for a DC declarative. No continuation cards may be used for in-line constants. The in-line constant is allocated based on the current value of the ILC, and usually carries a label for identification.

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.	FIELD C	±	INC.
1	6		12		18		22		28	32		38
	*		DD		CCC	C	C					

The maximum length (DD) of a negative in-line constant is 25 locations. The maximum length (DD) of a positive in-line constant is 44 locations. A blank in column 17 will be interpreted as a "plus."

Example 1:

The in-line constant may also be used to enter constants in the memory locations between Read Input storage and Print storage. The in-line constants should be preceded by a DL with a Field A of ¤81 and followed by a DL with a Field A of the Address for the first instruction to be assembled. These cards must then precede the first instruction when assembling.

NOTE: When in-line constants are assembled along with instructions it is the programmers responsibility to jump his program over the stored constants.

Example 2:

See Example 3 of Section 4.10.1.

4.10.3.2. In-Line Comments

Columns 41 through 56 of the coding form are headed "Comments" and can be used by the programmer to make notes regarding the lines of coding. These comments will also appear on the final printout of the Assembly processing. They provide an excellent assist to good documentation.

As a further means to good documentation, the UNIVAC 1005 Assembly System provides for in-line comments. An in-line comment allows the programmer to use most of the columns of the form for headings and notes. The in-line comment is not allocated to memory in the object program. It is merely printed and punched by the Assembly program. The in-line comment may appear anywhere in the source program.

LABEL	OPERATION	OPERAND 1				OPERAND 2							
		I.A. *	FIELD A	±	INC.	I.A. *	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	. ß C C C	C	C										

The in-line comment is specified by placing a period (.) in column 6 followed by a blank (space) in column 7. The Comment itself may appear anywhere in the coding line from columns 8 through 61. If the required comment is longer than 54 characters, additional in-line Comments may be coded with a period in column 6 and a space in column 7.

In order to assist in the documentation, it is suggested that the following in-line comments be used as the first card or cards of the program to be assembled.

Example:

Problem: Print the Program Name, Author's Name, and Date Written as the heading of the Assembler printout.

LABEL	OPERATION	OPERAND 1				OPERAND 2							
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	.PROGRAM ID:						PAYROLL		GROSS TO				NET
	.AUTHOR:		AB				JONES						
	.DATE WRITTEN:						JANUARY 9, 1966						

4.10.4. DEFINE INDIRECT ADDRESS CONSTANT

Mnemonic: DI

LABEL	OPERATION	OPERAND 1				OPERAND 2							
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	DI		IA-A				IA-B				IA-C		

The purpose of the DI address constant is to provide the programmer with the means to allocate, identify, and establish the initial contents of a secondary address. An instruction which calls for Indirect Addressing contains a primary address for the A or the B and C portions of the instruction, and the locations referred to by the primary address must contain a secondary address which is the address of the data to be processed by the instruction.

Indirect Addressing is specified in an instruction by placing an asterisk (*) in column 11 for OP 1, or column 21 for OP 2 (or both, if required). When an asterisk is coded in an instruction where IA is permitted, it causes the Assembler program to place a binary 1 bit in the X bit position of location 6 for OP 1 IA, or location 7 for OP 2 IA (or both). At object time, when the instruction is transferred to the Instruction Register, the UNIVAC 1005 first examines these two bit positions. Asterisks coded in instructions where IA is not permitted have no effect on the assembler.

If the X bit of location 6 contains a binary 1, a *two* location descending transfer is made beginning at the *address* in the A portion of the IR, to locations 2 and 3 (the A portion) of the IR. If the X bit of location 7 contains a binary 1, a *four* location descending transfer is made beginning at the *address* in the B portion (locations 4 and 5) of the IR, to locations 4 through 7 (the B *and* C portions) of the IR. After performing these functions, the instruction is then executed using the addresses currently in the IR regardless of the reoccurence of IA bits. The contents of the instruction as stored in memory remain unchanged.

In order to provide the locations in the object program for secondary addresses, the DI constant declarative is used as follows:

FIELD A:

The specification of an address in Field A establishes a two location secondary address in the object program. The initial contents of the two location secondary address will be the machine language code for the specified address.

FIELD B:

The specification of an address in Field B establishes a four location secondary address in the object program. The initial contents of the four location secondary address will be established according to the rules for Assembly System OP 2 addressing. If Field B contains a label (Standard or programmer), the two character address of the MSL of the area assigned to the label will be used as the initial contents of the left hand two locations of the secondary address. If Field B contains a label, Field C may be left blank, and the two character address of the LSL of the area assigned to the label will be used as the initial contents of the right hand two locations of the secondary address. Note coding both Fields A and B results in a 6 character pair of secondary addresses.

FIELD C:

If Field B does not contain a label, or if the LSL address of the area assigned to the label in Field B is not to be used as the initial value of the right hand two locations of the secondary address, Field C must contain an address.

NOTE: The DI constant will usually have a label which is used as the primary address in the instruction which calls for Indirect Addressing.

Example 1:

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.	FIELD C	±	INC.
1	6		12		18		22		28	32		38
CAT	DI		DOG									
FOX	DI						RAT					
COW	DI		+PIG				ANT					
MOVE	DI		¤1				¤724			¤733		

CAT becomes the MSL address of a *two* location secondary address which contains the MSL address of the area assigned to DOG. CAT may then be used as the primary address in the A portion of an instruction. When that instruction is executed, the *address* stored in CAT replaces the address of CAT in the IR.

FOX becomes the MSL address of a *four* location secondary address which contains the MSL and the LSL address of the area assigned to RAT.

COW becomes the MSL address of a *six* location constant which contains (from left to right) the address of the LSL of the area assigned to PIG, the address of the MSL of the area assigned to ANT, and the address of the LSL of the area assigned to ANT.

MOVE becomes the MSL address of a *six* location constant which contains (from left to right) the address of decimal location 1, the address of decimal location 724, the address of decimal location 733.

The Assembler processing for the allocation of locations to the DI constants is the same as for the allocation of the DC and the DA declaratives. This means that the groups of locations assigned to the DI constants in the previous example will be in the reverse sequence in which they are assembled. The characters within each constant will appear in the correct sequence.

Assume the value of the DLC is 1700 when CAT is to be assembled. The (MSL) addresses assigned to the labels in the example would be as follows:

CAT	1699
FOX	1695
COW	1689
MOVE	1683

This condition may have an effect on the sequence of coding DI constants when establishing a table of Indirect Addresses. In the example of Indirect Addressing (page 11), a table of secondary addresses was established with a label of SECAT. The coding to establish the table is as follows:

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A. *	FIELD A	±	INC.	I.A. *	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	DI						¤16				¤20		
	DI						¤11				¤15		
	DI						¤6				¤10		
SECAT	DI						¤1				¤5		

Using this sequence causes SECAT to be the address of the MSL of the Secondary Address Table. The LSL of the table is SECAT + 15. Refer to the coding of the loop in Section 4.8 for the use of an Indirect Address table. (The constant storage referred to in the example in Section 1.6 also would be set up in the same manner.)

4.10.5. DEFINE END

Mnemonic: END

LABEL	OPERATION	OPERAND 1				OPERAND 2							
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	E N D		X X X										

The purpose of the Define End declarative is to provide for the automatic jump to the start of the object program after it is loaded by the Load routine.

Field A specifies the address of the MSL of the first instruction to be executed in the object program. Field A usually contains a programmer's label. No allocation is made, and the address in Field A is not stored in memory.

Example 1:

LABEL	OPERATION	OPERAND 1				OPERAND 2							
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	E N D		S T A R T										

START is the label of the source language instruction which is to be executed first after the object program is loaded. The source language instruction START need not be the first instruction assembled.

NOTE: The Assembler output card produced from the END declarative must be the *last* card loaded by the Load routine, and should be followed immediately by the input data to be processed.

4.11. MULTIPLICATION INSTRUCTIONS

Multiplication in the UNIVAC 1005 is performed using fixed length operands for the multiplier, the multiplicand, and the product. When a Multiply instruction is executed by the UNIVAC 1005, the fixed length multiplier and multiplicand are transferred from memory to positions in Row 32, Bank 1 where they are then operated upon to produce the product--also in Row 32, Bank 1. The product is then automatically transferred from Row 32, Bank 1 to memory. During the execution, the locations in Row 32, Bank 1 normally reserved by the IR and the ICC are used in the calculation of the product. The contents of the ICC are preserved and are restored to the ICC automatically, to initiate the next instruction.

Multiplication is performed using the absolute (unsigned) values of the multiplier and multiplicand, producing an absolute (unsigned) value as the product. Locations in the product which do not contain a significant digit (1 through 9) will contain a zero (Ø). Values are treated as integers.

Since there are three operands in multiplication, the multiply commands of the UNIVAC 1005 utilize 3 address logic: OP 1 is the multiplicand, OP 2 is the multiplier, and OP 3 is the product.

There are two multiply instructions in the command structure of the UNIVAC 1005, Multiply (MU) and Multiply Long (ML). The difference between the two is the size of the fixed length multiplier, multiplicand and product.

Multiply utilizes a multiplier of exactly six digits, a multiplicand of exactly four digits, and produces a product of exactly ten digits.

Multiply Long utilizes a multiplier of exactly eleven digits, a multiplicand of exactly nine digits, and produces a product of exactly twenty digits.

If the programmer's multiplier and multiplicand do not conform exactly to the requirements of one or the other of the multiplication instructions, a Multiply Working Storage (MWS) should be established by the programmer (using DA declaratives) for a multiplier, multiplicand, and a product of these lengths. The programmer should then transfer his variable length operands to the MWS before executing a multiply instruction. The transfers to MWS should be made via rX using the TX instruction. An example of the use of rX as MWS is included in the explanation of each of the multiplication instructions.

4.11.1. MULTIPLY

Mnemonic: MU Mode: SPECIAL Length: 7 IA: NO

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.	FIELD A	±	INC.	I.A.	FIELD B	±	INC.		FIELD C	±	INC.
1	6	* 12			18	* 22			28		32		38
	MU		1L				2M				3L		

Function:

Transfer *ascending* the *four* digits beginning at the LSL specified in Field A to the multiplicand positions of Row 32, Bank 1. Transfer *descending* the *six* digits beginning at the MSL specified in Field B to the multiplier positions of Row 32, Bank 1. Multiply. Transfer *ascending* from the product positions of Row 32, Bank 1 to the *ten* locations beginning with the LSL specified in Field C.

NOTE: After completion of the MU instruction, the product is also available in Row 32, Column 21 through Column 30 of Bank 1, until the execution of another multiplication or division instruction.

Example 1:

Given: A four digit Quantity in card columns 1 through 4. A six digit Price in columns 5 through 1Ø.

Problem: Multiply Quantity times Price and store the product in the ten location area labeled AMT.

LABEL	OPERATION	OPERAND 1				OPERAND 2							
		I.A. *	FIELD A	±	INC.	I.A. *	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	MU		¤4				¤5				+AMT		

FIELD A:

¤4 is the decimal address of the LSL of Quantity.

FIELD B:

¤5 is the decimal address of the MSL of Price.

FIELD C:

+AMT is the address of the LSL of the area assigned to AMT.

In the event the size of the operands are not the *exact* length required by the MU command, rX (Row 32, Bank 2) can be used as a Multiply Working Storage. The multiplicand (4 digits or less) is transferred to rX using a TX command. This fills the high-order positions of rX with space codes. A TA command is used to transfer the multiplier to rX as follows: Field A contains the LSL address of the multiplier in memory; Field C specifies $32 27 B2; and Field B specifies a location in rX based on the actual length of the programmer's multiplier. If the multiplier is 5 digits, the Field B address would be $32 23 B2. The high-order positions of the fixed length multiplicand and multiplier will then contain space codes.

$32 21 B2 can be used as the LSL of the fixed length product of the MU command. The programmer can then transfer from rX the actual number of digits in his product.

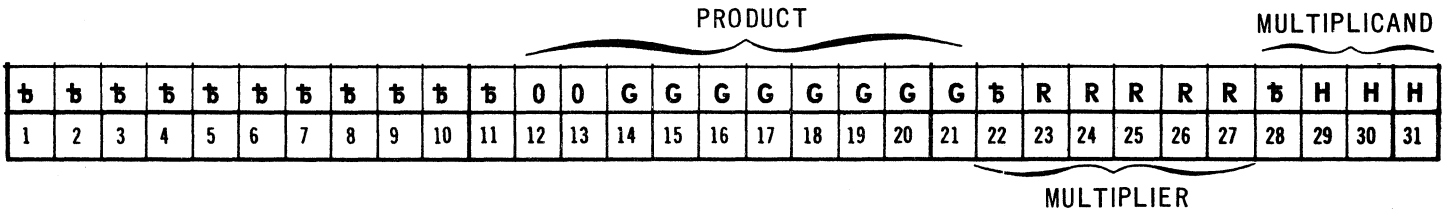
Example 2:

Given: A three digit Hours in columns 9 through 11 in the form XX.X. A five digit Rate in columns 12 through 16 in the form X.XXXX.

Problem: Multiply Hours times Rate and store the half adjusted Gross in Punch storage positions for columns 21 through 25 in the form XXX.XX.

LABEL	OPERATION	OPERAND 1				OPERAND 2							
		I.A. *	FIELD A	±	INC.	I.A. *	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	TX		¤11				¤9						
	TA		¤16				$3223	B	2		$3227	B	2
	MU		$3231	B	2		$3222	B	2		$3221	B	2
	AK		Ø5				$3212	B	2		$3219	B	2
	TA		$3218	B	2		$P1	+	2Ø		$P1	+	24

At the completion of the execution of the MU instruction, rX had the following appearance.



4.11.2. MULTIPLY (LONG)

Mnemonic: ML Mode: SPECIAL Length: 7 IA: NO

LABEL	OPERATION	OPERAND 1				OPERAND 2							
		I.A. *	FIELD A	±	INC.	I.A. *	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	ML		1L				2M				3L		

Function:

Transfer *ascending* the *nine* digits beginning at the LSL specified in Field A to the multiplicand positions of Row 32, Bank 1. Transfer *descending* the *eleven* digits beginning at the MSL specified in Field B to the multiplier positions of Row 32, Bank 1. Multiply. Transfer *ascending* from the product positions of Row 32, Bank 1 to the *twenty* locations beginning with the LSL specified in Field C.

NOTE: After completion of the ML instruction, the product is also available in Row 32, Column 11 through Column 3Ø of Bank 1, until the execution of another multiplication or division instruction.

The ML instruction is performed in the same manner as the MU instruction except for the provision of longer values. The use of rX described in the preceding Section 4.11.1. for the MU must be modified to take into account the longer operands.

The TX instruction specifies the actual length of the multiplicand. The TA instruction has an OP 2-LSL of $32 22 B2 and an OP 2-MSL predicated on the actual length of the multiplier. Since the total number of digits in the ML instruction multiplier, multiplicand,and product exceeds 31 (the capacity of rX), the LSL of OP 3 of the ML instruction can specify $32 3Ø B1. This will cause the product to be transferred to the identical locations in which it was developed.

Example 1:

Given: A six digit Quantity in columns 1 through 6. A seven digit Unit Cost in columns 7 through 13.

Problem: Multiply Quantity times Cost.

LABEL	OPERATION	OPERAND 1				OPERAND 2							
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	TX		$R1	+	5		$R1						
	TA		$R1	+	12		$XR	+	15		$XR	+	21
	ML		+$XR				$XR	+	11		$3230	B	1

4.12. DIVIDE INSTRUCTION

Division in the UNIVAC 1005 is performed using fixed length operands for the divisor, the dividend, and the quotient. When the Divide instruction is executed by the UNIVAC 1005, the fixed length divisor is transferred to certain positions in rX (Row 32, Bank 2), and the fixed length dividend is transferred to positions in Row 32, Bank 1, where they are operated upon to produce the quotient and the remainder --also in Row 32, Bank 1. The fixed length quotient is then automatically transferred from Row 32, Bank 1 to memory. During the execution of the Divide instruction, the locations in Row 32, Bank 1 normally reserved by the IR and the ICC are used in the calculation of the quotient. The contents of the ICC are preserved, and are restored to the ICC automatically, to initiate the next instruction.

Division is performed using the absolute (unsigned) values of the divisor and the dividend, producing an unsigned value as the quotient. Locations in the quotient and remainder which do not contain a significant digit (1 through 9) will contain a zero (Ø). Values are treated as integers.

Since there are three operands in division, the Divide instruction of the UNIVAC 1005 utilizes 3 address logic: OP 1 is the divisor, OP 2 is the dividend, and OP 3 is the quotient.

The Divide instruction utilizes a divisor of exactly six digits, a dividend of exactly eight digits, and produces a quotient of exactly eight digits. If the length of the programmer's divisor, dividend,and quotient do not all conform exactly to the requirements of the Divide instruction, a Divide Working Storage (DWS) should be

established by the programmer for a divisor, dividend, and quotient of these lengths. The programmer should then transfer his variable length operands to the DWS before executing the Divide instruction. An example of the use of Divide Working Storage is given following the explanation of the Divide instruction.

4.12.1. DIVIDE

Mnemonic: DV Mode: SPECIAL Length: 7 IA: NO

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A. *	FIELD A	±	INC.	I.A. *	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	DV		1L				2M				3L		

Function:

Transfer *ascending* the *six* digits beginning at the LSL specified in Field A to the *Divisor* positions of rX (Row 32, Bank 2). Transfer *descending* the *eight* digits beginning at the MSL specified in Field B to the *Dividend* positions of Row 32, Bank 1. Divide. Transfer *ascending* from the *Quotient* positions of Row 32, Bank 1 to the *eight* locations beginning with the LSL specified in Field C.

NOTE: After completion of the Divide instruction, the Remainder is available in Row 32, Column 18 through column 23 of Bank 1, until the execution of another multiply or divide instruction.

Example 1:

Given: An eight digit Total Cost in card columns 9 through 16. A six digit Total Units in card columns 1 through 6.

Problem: Divide Total Cost by Total Units, and store the quotient in the eight location area labeled UNIT.

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A. *	FIELD A	±	INC.	I.A. *	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	DV		¤6				¤9				+UNIT		

FIELD A:

¤6 is the decimal address of the LSL of Total Units (the divisor).

FIELD B:

¤9 is the decimal address of the MSL of Total Cost (the dividend).

FIELD C:

+ UNIT is the address of the LSL of the area assigned to UNIT (the quotient).

NOTE: If the divisor is equal to zero, the quotient is all zeros and the remainder is all zeros. If the dividend is all zeros, the quotient is all zeros and the remainder is all zeros. If the dividend is all spaces, the quotient is all zeros and the remainder is all spaces. If the divisor is greater that the dividend, the quotient is all zeros and the remainder is the least significant six digits of the divident.

In the event the size of the operands in the program are not the *exact* length required by the DV command, rX (Row 32, Bank 2) can be used as a Divide Working Storage. The dividend is transferred to rX using a TX command. This fills the high-order positions of rX with space codes. A TA command is used to transfer the divisor to rX as follows: Field A specifies the LSL address of the divisor in memory; Field C specifies $XR + 15; and Field B specifies a location in rX based on the actual length of the programmer's divisor. For example, if the actual length of the divisor is 5 digits, the Field B address would be $XR + 11. The high-order positions of the fixed length dividend and divisor would then contain space codes. $XR + 7 can be used as the LSL of the fixed length quotient produced by the DV instruction.

NOTE: If rX is used as DWS, the locations indicated above must be used for the divisor.

Example 2:

Given: A seven digit Total Revenue in card columns 11 through 17. A four digit Total Tons in columns 5 through 8.

Problem: Divide Total Revenue by Total Tons and store the maximum 7 digit quotient in a 7 location area labeled RPT.

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A. *	FIELD A	±	INC.	I.A. *	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	TX		¤17				¤11						
	TA		¤8				$XR	+	12		$XR	+	15
	DV		$XR	+	15		$XR	+	23		$XR	+	7
	TA		$XR	+	7		RPT						

4.13. INPUT/OUTPUT INSTRUCTIONS

There are several types of input/output instructions in the UNIVAC 1005 Assembly System. Each type is tailored to the requirements and the operating mode of the input/output device.

One type of input/output instruction is used for the devices which transfer data to or from fixed areas in memory reserved for that purpose, or for commands to an input/output device which do not involve a data transfer (eg. Space 1, Stacker Select, etc.) This type is called General Command.

Each of the devices which does not have a reserved area for input/output data transfers has its own type of command (eg. Magnetic Tape, DLT, etc.) Each type is named for the device it controls.

In the case of the General Command, certain of the uses of this command have been selected as Assembler pseudo-operations in order to provide the programmer with a rapid method of specifying often used commands. These pseudo-operations are called Shortened General Commands.

4.13.1. SHORTENED GENERAL COMMANDS

Mnemonic: GCn Mode: I/O Length: 7 IA: NO

LABEL	OPERATION	OPERAND 1				OPERAND 2							
		I.A.	FIELD A	±	INC.	I.A.	FIELD B	±	INC.		FIELD C	±	INC.
1	6	*	12		18	*	22		28		32		38
	GCn												

Function:

Execute Reader, Printer,and Punch input/output commands specified by n, where n means the following:

1. Read, Execute
2. Test Punch, Punch, Clear
3. Combines GC1 and GC2
4. Space 1, Print, Execute
5. Combines GC1 and GC4
6. Combines GC2 and GC4
7. Combines GC1, GC2, and GC4

The GCn operations are pseudo-operation codes which select specific Reader, Printer, and Punch operations from the structure of the GC coded operations. When any of the specific input operations indicated above are required, the GCn may be used.

Fields A, B, and C must be blank, in the GCn instruction.

NOTE: When using these GCn commands Fields A, B, and C must be left blank.

4.13.2. GENERAL COMMANDS

Mnemonic: GC Mode: 1/O Length: 7 IA: NO

LABEL	OPERATION	OPERAND 1				OPERAND 2							
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	GC		CC				CC				CC		

The bit positions of locations 2 through 7 of the GC instruction correspond to functions of the UNIVAC 1005 input/output devices which have an implied data address, or which do not require a data address.

A thorough knowledge of the various input/output devices of the UNIVAC 1005 is required for proper use of the GC command.

Since the desired input/output functions are indicated by bit positions of the least significant six locations of the GC instruction, octal coding is used to specify CC in Fields A, B, and C. All three Fields must be specified.

Octal coding is specified by the number sign (#) followed by four octal digits. The bit configuration for the required input/output operations is used to determine the four octal digits.

NOTE: Caution should be exercised by the programmer to avoid specification of bits which correspond to conflicting input/output operations.

The following is a brief description of the input/output functions which correspond to the bit positions in the GC instruction. The functions are listed by the type of function. A table is also provided which indicates the corresponding bit position in the GC instruction.

CARD AND PAPER TAPE

Card Read	This bit sets the Read F.F. The card will be read when Execute is given.
Auxiliary Card Read	This bit must occur in the same GC as Execute to cause card reading.
Read Punch Read	This bit sets the Read Punch Read F.F. Read will occur when Punch Hold or Punch Clear is given.
Paper Tape Read-Block	This bit must occur in the same GC as Execute. Will cause the reading of 80 characters from paper tape.
Paper Tape Read-Character	This bit must occur in the same GC as Execute. Will cause the reading of 1 character from paper tape.

End Read on All Bits	This bit must occur in the same GC as Execute and the bits which cause reading. During the read, each character is tested for all 1 bits. If detected, reading is terminated.
Execute	This bit causes the devices to execute those functions signalled, by various of the other bits, either prior to or simultaneous with Execute.

PRINTING

Print	This bit sets the Print F.F. A line will be printed when Execute is given.
End Print at 90 Characters	This bit must occur in the same GC as Execute. The Print function must be signalled prior to or simultaneous with End Print.
Space 1	This bit will cause an immediate line space unless accompanied by Print, Execute. When accompanied by Print, Execute, the line is printed before spacing occurs.
Space 2	This bit will cause two immediate line spaces unless accompanied by Print, Execute. When accompanied by Print, Execute, the line is printed before spacing occurs. If Space 1 and Space 2 are given in the same command only Space 2 is effective.
Skip 1, 2 and 4	Any combination of these three bits will initiate an immediate forms skip, unless accompanied by Print, Execute. When accompanied by Print, Execute, the line is printed before skip occurs.

CARD AND PAPER TAPE PUNCHING

Punch Hold	This bit causes an immediate punch cycle. The punch storage area is *not* cleared following punching.
Punch Clear	This bit causes an immediate punch cycle. The punch storage area is *cleared* following punching.
Punch Test	This bit causes a test to determine if the punch storage area is still in use by the last punch operation. If it is in use, further execution of UNIVAC 1005 instructions is inhibited, until not in use.
Punch Paper Tape-Block	This bit must occur in the same GC as Punch Hold or Punch Clear. When accompanied by Punch Hold or Punch Clear, it causes 80 characters to be punched in paper tape.

Punch Paper Tape-Character Character	This bit must occur in the same GC as Punch Hold or Punch Clear. When accompanied by Punch Hold or Punch Clear it causes 1 character to be punched in paper tape.
Punch Odd Parity-Paper Tape	If this bit occurs in the same GC which causes paper tape punching, Odd Parity punching will occur.

PROCESSER INPUT/OUTPUT MODE

Code Image Read	This bit must occur in the same GC which causes reading. When accompanied by the bit or bits which cause reading, it causes card or paper tape reading in Code Image.
Code Image Punch	This bit must occur in the same GC which causes punching. When accompanied by the bit or bits which cause punching, it causes card or paper tape punching in Code Image.

CARD SELECTION

Punch Stacker Select	If this bit occurs in the same GC which causes *card* punching, it causes the card being punched to be placed in the Select Stacker on the next Punch cycle.
Auxiliary Reader Stacker Select 1	This bit must occur in the same GC which causes reading in the Auxiliary Reader. It causes the previous card read to be placed in Stacker 1.
Auxiliary Reader Stacker Select 2	This bit must occur in the same GC which causes reading in the Auxiliary Reader. It causes the previous card read to be placed in Stacker 2.

DATA LINE CONTROL

Request to Transmit	This bit sets a F.F. that requests data line transmission. The F.F. must be set prior to the execution of the instruction (SD) which causes data transmission.

MAGNETIC TAPE CONTROL

Rewind Magnetic Tape	This bit causes an immediate rewind of tape on the Servo last set by the Set Condition (SC) instruction.

Back Space Magnetic Tape — This bit causes an immediate backward movement of tape, to the preceding inter-record gap, on the Servo last set by the Set Condition (SC) instruction.

Erase Flip Flop — This bit causes the Erase F.F. to be set. When the next write tape (WT) instruction is executed, 5.04 inches of tape will be erased prior to writing the block of tape.

The bit positions which correspond to the previously listed functions are shown below. The bit positions are indicated by the Location within the GC instruction, on the left hand side of the name of the function. On the right hand side of the function is the value of the bit for octal coding purposes. At the bottom of each Value column is a box for indicating the sum of the circled values. These boxes are identical to the position of the octal digits in each of the fields of the GC source language instruction.

LOCATION 2				LOCATION 3			
BITS	VALUE	BITS	VALUE	BITS	VALUE	BITS	VALUE
X NOT USED	4	4 SPACE 1	4	X NOT USED	4	4 READ AUX.	4
Y RESERVED	2	2 READ	2	Y SELECT STACK 2 AUX. READ	2	2 READ PUNCH READ	2
8 PRINT	1	1 EXECUTE	1	8 SELECT STACK 1 AUX. READ	1	1 READ CODE IMAGE PAPER TAPE OR CARD	1
FIELD A CC = #	☐		☐		☐		☐

LOCATION 4				LOCATION 5			
BITS	VALUE	BITS	VALUE	BITS	VALUE	BITS	VALUE
X END READ ON ALL BITS	4	4 PAPER TAPE READ BLOCK	4	X NOT USED	4	4 SKIP 1	4
Y NOT USED	2	2 PAPER TAPE PUNCH CHAR	2	Y SKIP 4	2	2 SPACE 2	2
8 PAPER TAPE PUNCH BLOCK	1	1 PAPER TAPE READ CHAR	1	8 SKIP 2	1	1 END PRINT AT 90	1

FIELD B CC = # [ ] [ ] [ ] [ ]

LOCATION 6				LOCATION 7			
BITS	VALUE	BITS	VALUE	BITS	VALUE	BITS	VALUE
X NOT USED	4	4 BACK SPACE MAGNETIC TAPE	4	X NOT USED	4	4 PUNCH CLEAR	4
Y REWIND MAGNETIC TAPE	2	2 SET REQUEST TO TRANSMIT	2	Y PUNCH STACKER SELECTOR PT PUNCH ODD PARITY	2	2 PUNCH TEST	2
8 SET ERASE FLIP FLOP	1	1 RESERVED	1	8 PUNCH HOLD	1	1 PUNCH CODE IMAGE PAPER TAPE OR CARD	1

FIELD C CC = # [ ] [ ] [ ] [ ]

4.13.3. READ MAGNETIC TAPE

Mnemonic: RT Mode: DESCENDING Length: 7 IA: NO

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	RT						2M						

Function:

Read forward, transfer descending from magnetic tape to memory beginning at 2M specified in Field B. Continue read and transfer until an inter-record gap is detected.

NOTE: The Servo on which reading will occur is the Servo last set by a Set Condition (SC) instruction.

Field B specifies the address of the MSL of an area of memory which is to receive the block of data from tape. Once initiated by the RT command, reading will continue until the inter-record gap, recorded on tape when it was written, is detected by the Servo. The area set aside should be at least as long as the longest block to be read into memory plus one location.

NOTE: The information stored in memory will be the data read from tape plus an additional space code as the last character stored.

Example 1:

Given: TPIN is the label of a DA declarative which established a *251* location area. The longest block of tape expected is *250* characters.

Problem: Read a block of magnetic tape beginning at TPIN.

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	RT						TPIN						

The machine language instruction produced by the RT source language instruction will have the address of the MSL of the input area (TPIN) in locations 4 and 5. Locations 2, 3, 6, and 7 will be blank.

+ TPIN will contain a space code if a 250 character block is read.

NOTE: If the 4 bit position of location 3 contains a binary 1, reading is performed at high gain. This is specified by placing #0004 in Field A. If Field A is blank, reading occurs at normal gain.

4.13.4. WRITE MAGNETIC TAPE

Mnemonic: WT Mode: DESCENDING Length: 7 IA: NO

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	WT		1L				1M / 2M				2L		

Function:

Write forward, transfer descending beginning at OP 1-MSL specified in Field B until OP 1-LSL specified in Field A has been transferred. Also, simultaneously transfer descending beginning at OP 1-MSL to OP 2-MSL *which is the same address as OP 1-MSL*. Transfer ends at the location of OP 2 which coincides with OP 1-LSL. The WT instruction continues until OP 2-LSL specified in Field C has been cycled. *OP 2-LSL must be OP 1-LSL plus 3.*

The OP 1-LSL must be two locations beyond the last data character to be written on tape. The characters in these two extra locations of OP 1 are *not* written on tape and are not disturbed in memory. The contents of the 3 locations to the right of OP 1-LSL are not disturbed. Thus the five locations to the right of the location which contains the last data character to be recorded on tape are included in the machine function, but remain unchanged by the function.

Example 1:

Given: TPOUT is the label of a DA declarative which established a 400 location area.

Problem: Write a block of magnetic tape from TPOUT.

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	WT		+TPOU	+	2		TPOUT				+TPOU	+	5

Consideration need not be given to the contents of the five locations to the right of the LSL of TPOUT, since they are not changed by this instruction.

NOTE: The Servo on which writing will occur is the Servo last set by a Set Condition (SC) instruction.

A Servo must be set by an SC instruction before the first magnetic tape operation is given. Console Clear leaves no Servo set.

4.13.5. RECEIVE DATA LINE

Mnemonic: RD Mode: DESCENDING Length: 7 IA: NO

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	RD		1M				2M/1M				2L		

Function:

Receive from the Data Line. Store descending beginning at OP 2-MSL specified in Field B, continuing until OP 2-LSL specified in Field C has been filled. The last two characters stored in OP 2 will be the End of Message character and the Longitudinal Parity Character. The preceding characters will be the Data characters.

OP 2-MSL is automatically used as OP 1-MSL. Field A which specifies OP 1-LSL should contain the same address as Field B. Thus OP 1 becomes a one character location. There must be an OP 1 in an RD function for control purposes. Blank addressing should not be used in Field C.

Example 1:

Given: RDIN is the label of a DA declarative which established an 82 location area.

Problem: Receive the contents of the card transmitted.

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	RD		RDIN				RDIN				+RDIN		

The contents of the transmitted card will be stored in RDIN through RDIN + 79. RDIN + 80 will contain EOM. RDIN + 81 (+RDIN) will contain the LPC.

4.13.6. SEND DATA LINE

Mnemonic: SD Mode: DESCENDING Length: 7 IA: NO

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	SD		1L				1M/2M				2L		

Function:

Send via the Data Line. Transfer descending to the Data Line, the information beginning at the MSL of OP 1 specified in Field B, continuing until the LSL of OP 1 specified in Field A has been transferred to the Data Line. Simultaneously transfer descending, the same information to OP 2. The OP 1-MSL is used as the OP 2-MSL. OP 2-LSL specified in Field C must be the same as OP 1-LSL specified in Field A. Blank addressing should not be used.

The information to be transferred from memory to the Data Line consists of Transmission Control Characters, and the data; in the following format:

OP 1, MSL to OP 1, MSL + 3	Four synchronizing characters
OP 1, MSL + 4	Space code
OP 1, MSL + 5 to n	Data
OP 1, LSL - 3	Turn off Request to Transmit--right hand parenthesis ( ))
OP 1, LSL - 2	End of Message Code
OP 1, LSL - 1 and OP 1, LSL	Space Codes

Example:

Problem: Transmit the contents of a card.

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
EOM	DC		4	+	)B								
DATA	DA		80										
PREAM	DC		5	+	SSS	S							
	TD		$R1				DATA						
	GC		#0000				#0000				#0200		
	SD		+EOM				PREAM				+EOM		

Solution: The DC and DA declaratives allocate a message area of storage which contains the 5 constant preamble (PREAM), the data (DATA), and the 4 constant trailer. These are allocated in reverse sequence to the order assembled.

The TD transfers the card data from Read Input storage to the data portion of the message area. The GC sets Request to Transmit. The SD does the sending.

4.13.7. RECEIVE INTERFACE

Mnemonic: RF Mode: Descending Length: 7 IA: NO

LABEL	OPERATION	OPERAND 1				OPERAND 2						
		I.A.*	FIELD A	±	INC.	I.A.*	FIELD B	±	INC.	FIELD C	±	INC.
1	6		12		18		22		28	32		38
	RF		CC				M			L		

Function:

Receive data from the Unit selected by bits in CC specified in Field A, and store in the area whose MSL is specified in Field B, and LSL is specified in Field C.

The address specified in Field B becomes OP 1-MSL and OP 2-MSL, and the address specified in Field C becomes OP 1-LSL and OP 2-LSL during the RF function. The M address occupies locations 4 and 5, and the L address occupies location 6 and 7.

The bit positions of CC appear in locations 2 and 3 and are used to indicate Unit Number, Input Control, and Output Control. Octal coding is used to specify the bit pattern of CC. The number sign (#) followed by four octal digits is required to specify octal coding.

The bit positions of CC and corresponding Unit Number and Input/Output Control are as follows:

Location 2	Location 3
X not used	X not used
Y Output Control 3	Y Input Control 3
8 Output Control 2	8 Input Control 2
4 Output Control 1	4 Input Control 1
2 Unit 4	2 Unit 2
1 Unit 3	1 Unit 1

Example 1:

Problem: Receive an 80 column record from Unit 1 through Input Control 1 and store in an 80 location area labeled IN1.

LABEL	OPERATION	OPERAND 1				OPERAND 2							
		I.A. *	FIELD A	±	INC.	I.A. *	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	RF		#0005				IN1				+IN1		

Solution: #0005 is octal for ØØØ ØØØ ØØØ 101, which selects Unit 1 and Input Control 1.

4.13.8. SEND INTERFACE

Mnemonic: SF Mode: Descending Length: 7 IA: NO

LABEL	OPERATION	OPERAND 1				OPERAND 2							
		I.A. *	FIELD A	±	INC.	I.A. *	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	SF		CC				M				L		

Function:

Send data to the Unit Selected by bits in CC specified in Field A; from the area whose MSL is specified in Field B, and LSL is specified in Field C.

The address specified in Field B becomes the OP 1-MSL and OP 2-MSL, and the address specified in Field C becomes the OP 1-LSL and OP 2-LSL during the SF function. The M address occupies locations 4 and 5, and the L address occupies locations 6 and 7.

The bit positions of CC appear in locations 2 and 3 and are used to indicate Unit Number, Input Control, and Output Control. Octal coding is used to specify the bit pattern of CC. The number sign (#) followed by four octal digits is required to specify octal coding.

The bit positions of CC and corresponding Unit Number and Input/Output Control are listed in the section on Receive Interface (Section 4.13.7)

Example :

Problem: Send to Unit 2 through Output Control 1 the information stored in a 160 location area labeled OU2.

LABEL	OPERATION	OPERAND 1				OPERAND 2							
		I.A. *	FIELD A	±	INC.	I.A. *	FIELD B	±	INC.		FIELD C	±	INC.
1	6		12		18		22		28		32		38
	SF		#0402				OU2				+OU2		

Solution: #Ø4Ø2 is octal for ØØØ 1ØØ ØØØ Ø1Ø which selects Unit 2 and Output Control 1.

5. OPERATING PROCEDURES FOR 1005 ASSEMBLY

5.1. LOADING SOURCE PROGRAM

(1) Set Manual Alteration Switch No. 1 on. Place the assembler deck in the input hopper followed by the source code deck. Press Start, Clear, Feed and Run. The line "Univac 1005 IPM Assembler Pass 1" is printed followed by "Reset Alteration Switch No. 1". Operator will reset Alteration Switch No. 1 and press Run. In the event Alteration Switch No. 1 is not reset and Run is pressed, the message "Reset Alteration Switch No. 1" is printed until the switch is reset. During Pass 1, cards with invalid mnemonics will be listed. When the processor halts (hopper empty) press Run and Stop. It is suggested that the cards with invalid mnemonics be repunched and inserted into the source code deck and Pass I rerun from the beginning. However, as an alternate recovery, the cards with invalid mnemonics should be repunched and then rerun by pressing Feed and Run. If Clear or Start was pressed at the end of the Pass, the cards with invalid mnemonics could be rerun by (a) doing a manual start at location 148 or by (b) rerunning Pass I for those cards only. The new cards may then be inserted in the intermediate deck.

(2) Place the assembler deck in the input hopper followed by intermediate deck 1. Press Start, Clear, Feed and Run. The line "Univac 1005 IPM Assembler Pass 2" is printed. During Pass 2, all assembler declaratives are listed except for End. When the processor halts (hopper empty), press Run and Stop. Remove intermediate deck 2 from the punch stacker.

(3) Place the assembler deck in the input hopper followed by intermediate deck 2. Press Start, Clear, Feed and Run. The line "Univac 1005 IPM Assembler Pass 3" is printed. Either (a), (b), or (c) occurs.

(a) Final assembly listing occurs. Continue at (d).

(b) Halt #1 occurs followed by the printing of "Set Alt Switch 2". Operator must set Alteration Switch No. 2 and press Run. In the event Alteration Switch No. 2 is not set and Run is pressed, the message "Set Alt Switch 2" is printed until the switch is set. Continue at (d).

(c) Neither (a) nor (b) occurs. Continue at (d).

(d) When the processor halts (hopper empty), press Run and Stop. Remove the output deck from punch stacker. If (a) occurred, this is the final object deck, and the assembly is complete. If either (b) or (c) occurred, repeat steps (2) and (3) without resetting Alteration Switch No. 2; use the most recent output as intermediate deck 1.

NOTE:

If it is necessary to rerun passes 2 and 3, printouts "Rerunning Univac 1005 IPM Assembler Pass 2" and "Rerunning Univac 1005 IPM Assembler Pass 3" will occur while rerunning these respective passes.

5.2. LOADING OBJECT PROGRAM

A load card is needed to load the object code program when assembly is complete. In normal circumstances the first card of the assembly deck can be used for this purpose. The load card should be reproduced to avoid misplacement. This card inserts the necessary 4 bit into location R31/C32 of bank 1 and commences loading the object program. When using this method for loading, the programmer should not use memory locations 81 thru 92 in this program.

5.3. FINAL LISTING

When assembly is complete, a final listing is produced. This listing contains the following:

- Original Source Code
- Unfound Indicators
- Sequence Numbers
- Object Code Instruction
- Load Instruction
- Diagnostic Message

The columns discussed represent the print positions starting from print position 1.

5.3.1. Original Source Code

Columns 1 to 61 inclusive is exactly the same as coded by the programmer.

1. Instructions	Columns 1 to 40
2. Positive Constants	Columns 18 to 61
3. Negative Constants (minus in Col. 17)	Columns 18 to 42
4. Comment Card	Columns 8 to 61
5. Comment on Instruction Card	Columns 41 to 56

5.3.2. Unfound Indicators

Columns 59, 60 and 61 represent the unfound addresses of Field A, Field B and Field C respectively. The character U will print in the respective column when one or more of the Fields are unfound.

Example:

Transfer descending 80 columns of Read to the first 80 columns of Punch.

	Field A	Field B	Field C
TD	$R1	$P1	$P1+79

If Field B was not specified

	Field A	Field B	Field C
TD	$R1	*Blank*	$P1+79

the character U will print in column 60.

NOTE: If Field C was not specified

	Field A	Field B	Field C
TD	$R1	$P1	*Blank*

the character U will *not* print because in this example Field C is optional (See instruction formats).

5.3.3. Sequence Number

Columns 62 to 65 inclusive is the sequence number of the object program deck.

5.3.4. Object Code Instruction

Columns 67 to 73 is the Source Code Instruction translated to actual machine code.

Example:

Source Code Instruction		67	68 69	70 71	72 73
	Column	TA	$R1+79	$PR	$PR+79
Object Code Instruction		Ø	Ø [	I I	. \

5.3.5. Load Instruction

Columns 74 to 80 places the Object Code Instruction in the assigned storage area.

5.3.6. Diagnostic Message

Column 82 indicated the following types of errors found:

1. ' (Apostrophe, XBIT) Doubly defined label indication.
2. & (Ampersand, YBIT) Continuation of a Define Constant out of sequence.
3. 5 (Five, 8BIT) Define Sub-Field out of sequence.
4. 1 (One, 4BIT) Length of Define Area or Define Constant equal or less than zero.
5. - (Minus, 2BIT) Extra continuation of a Define Constant.

NOTE: A combination of the bit pattern of the above error messages may be produced.

6. PROGRAM TESTING AIDS

The following tables and charts are provided to facilitate Program Testing. Table 6-1 can be used to determine several equivalent forms of actual addresses of assembled instructions and constants. Table 6-2 describes basic characteristics of the assembled UNIVAC 1005 Instruction Repertoire.

ROW OR COLUMN WITHOUT BANK BIT			
ROW OR COLUMN NUMBER	CHAR. EQUIV.	MACHINE CODE X Y 8 4 2 1	OCTAL EQUIV.
1	space	0 0 0 0 0 0	00
2	]	0 0 0 0 0 1	01
3	Ø	0 0 0 0 1 1	03
4	4	0 0 0 1 1 1	07
5	;	0 0 1 1 1 0	16
6	I	0 1 1 1 0 0	34
7	F	0 1 1 0 0 1	31
8	.	0 1 0 0 1 0	22
9	1	0 0 0 1 0 0	04
10	5	0 0 1 0 0 0	10
11	:	0 1 0 0 0 1	21
12	–	0 0 0 0 1 0	02
13	2	0 0 0 1 0 1	05
14	7	0 0 1 0 1 0	12
15	B	0 1 0 1 0 1	25
16	8	0 0 1 0 1 1	13
17	D	0 1 0 1 1 1	27
18	[	0 0 1 1 1 1	17
19	<	0 1 1 1 1 0	36
20	#	0 1 1 1 0 1	35
21	H	0 1 1 0 1 1	33
22	C	0 1 0 1 1 0	26
23	\	0 0 1 1 0 1	15
24	G	0 1 1 0 1 0	32
25	A	0 1 0 1 0 0	24
26	6	0 0 1 0 0 1	11
27	?	0 1 0 0 1 1	23
28	3	0 0 0 1 1 0	06
29	9	0 0 1 1 0 0	14
30	E	0 1 1 0 0 0	30
31	&	0 1 0 0 0 0	20
32	=	0 1 1 1 1 1	37

ROW OR COLUMN WITH BANK BIT			
ROW OR COLUMN NUMBER	CHAR. EQUIV.	MACHINE CODE X Y 8 4 2 1	OCTAL EQUIV.
1	'	1 0 0 0 0 0	40
2	*	1 0 0 0 0 1	41
3	!	1 0 0 0 1 1	43
4	M	1 0 0 1 1 1	47
5	@	1 0 1 1 1 0	56
6	Z	1 1 1 1 0 0	74
7	W	1 1 1 0 0 1	71
8	,	1 1 0 0 1 0	62
9	J	1 0 0 1 0 0	44
10	N	1 0 1 0 0 0	50
11	%	1 1 0 0 0 1	61
12	$	1 0 0 0 1 0	42
13	K	1 0 0 1 0 1	45
14	P	1 0 1 0 1 0	52
15	S	1 1 0 1 0 1	65
16	Q	1 0 1 0 1 1	53
17	U	1 1 0 1 1 1	67
18	Δ	1 0 1 1 1 1	57
19	>	1 1 1 1 1 0	76
20	¤	1 1 1 1 0 1	75
21	Y	1 1 1 0 1 1	73
22	T	1 1 0 1 1 0	66
23	(	1 0 1 1 0 1	55
24	X	1 1 1 0 1 0	72
25	/	1 1 0 1 0 0	64
26	0	1 0 1 0 0 1	51
27	+	1 1 0 0 1 1	63
28	L	1 0 0 1 1 0	46
29	R	1 0 1 1 0 0	54
30	V	1 1 1 0 0 0	70
31	≠	1 1 0 0 0 0	60
32	)	1 1 1 1 1 1	77

Table 6-1. UNIVAC 1005 Address Codes Row, Column, Character and Octal Equivalent.

BANK BITS – (X BIT POSITION)		
BANK	ROW	COLUMN
1	USE TABLE 1	USE TABLE 1
2	USE TABLE 1	USE TABLE 2
3	USE TABLE 2	USE TABLE 1
4	USE TABLE 2	USE TABLE 2

Example:

Convert R16/C6 Bank 4 into Octal.

Bank 4 requires the programmer to look up the Row in Table 2 and the Column in Table 2.

Row 16 in Table 2 is Octal 53. Column 6 in Table 2 is Octal 74.

The octal address for R16/C6 B4 is #5374.

R21/C30 Bank 3 is #7330.

R1/C1 Bank 1 is #0000

LABEL	OPERATION	OPERAND 1				OPERAND 2							
		I. A.	FIELD A	±	INC.	I. A.	FIELD B	±	INC.		FIELD C	±	INC.
1	6	* 12			18	* 22			28		32		38
			#0,0,0,0										

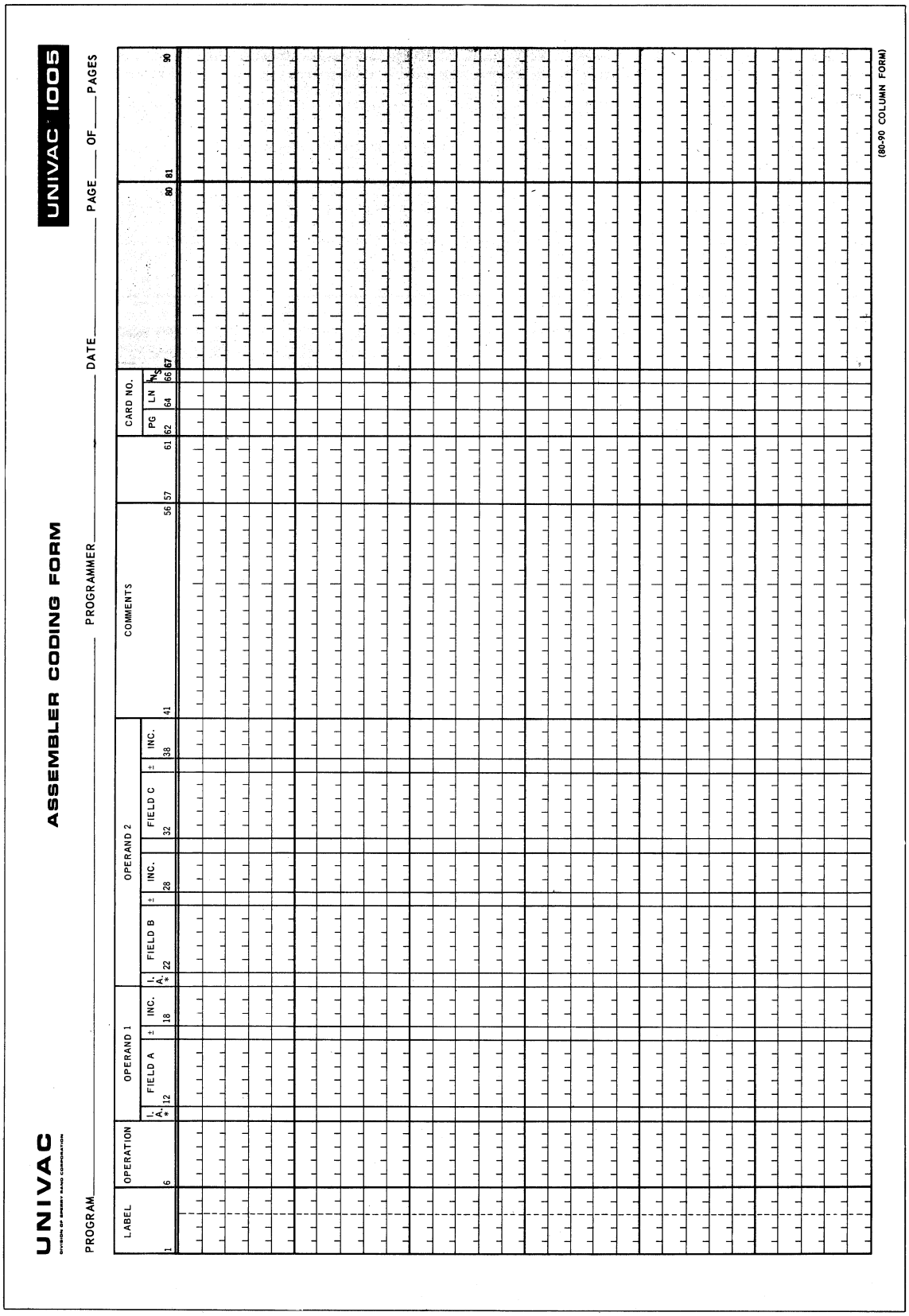
OPERATION MNEMONIC	DESCRIPTION	ACTUAL OPERATION CODE	CHARACTERS 2/3 4/5 6/7 IN OBJECT FORM			LENGTH OF INSTRUCTION	MODE	INDIRECT ADDRESSING
TRANSFER INSTRUCTIONS								
TA	Transfer Asc.	Ø	1L	2M	2L	7	Asc.	Yes
TC	Transfer Clear	4	1L	2M	2L	7	Asc.	Yes
TD	Transfer Desc.	]	1M	2M	2L	7	Desc.	Yes
TK	Transfer Constant	2	KK	2M	2L	7	Asc.	Yes
TN	Transfer Numeric	;	1L	2M	2L	7	Asc.	Yes
TR	Translate	9		2M	2L	7	Desc.	Yes
TX	Transfer to Register X	≠	1L	1M	–	5	Asc.	No
COMPARE INSTRUCTIONS								
CA	Compare Alphanumeric	I	1L	2M	2L	7	Asc.	Yes
CK	Compare Constant	7	KK	2M	2L	7	Asc.	Yes
CM	Compare Magnitude	F	1L	2M	2L	7	Asc.	Yes
CN	Compare Numeric		1L	2M	2L	7	Asc.	Yes
ARITHMETIC INSTRUCTIONS								
AD	Add Algebraic	1	1L	2M	2L	7	Asc.	Yes
AM	Add Magnitude	5	1L	2M	2L	7	Asc.	Yes
DV	Divide	@	1L	2M	3L	7	A+D	No
ML	Multiply Long	M	1L	2M	3L	7	A+D	No
MU	Multiply	*	1L	2M	3L	7	A+D	No
SM	Subtract Magn.	–	1L	2M	2L	7	Asc.	Yes
SU	Subtract Algebr.	:	1L	2M	2L	7	Asc.	Yes
AK	Add Constant	6	DD	2M	2L	7	Asc.	Yes
COUNT INSTRUCTION								
CC	Count	W	±DD	2M	–	5	Spec.	No
EDIT INSTRUCTIONS								
EL	Edit Logical	,	KK	1M	2M	7	Spec.	No
ES	Edit Superimpose	,	–K		2M	7	Spec.	No
EE	Edit Erase	,	K–	1M		7	Spec.	No
ED	Edit	G	1M	2M	2L	7	Desc.	Yes
SET INSTRUCTIONS								
SC	Set Conditions	¤	–	CC	–	5	–	No
STOP	Stop	¤	–	CC	–	5	–	No
JUMP INSTRUCTIONS								
JC	Jump on Conditions	X	CC	JA	–	5	Spec.	No
JK	Jump Compare	(	K–	JA	2L	7	Spec.	No
JL	Jump Loop	J	DD	JA		7	Spec.	No
JR	Jump Return	N	RA	JA	2L	7	Spec.	No
JT	Jump Test	Y	JA=	JA<	–	5	Spec.	No
J	Jump	/	–	JA		5	Spec.	No
JI	Jump Indirect	]	IA			7	Spec.	Yes
INPUT/OUTPUT INSTRUCTIONS								
GC	General Commands	%	CC	CC	CC	7	I/O	No
RT	Read Magnetic Tape	$		2M	2L	7	Desc.	No
WT	Write Magnetic Tape	K	1L	M	2L	7	Desc.	No
RD	Receive Data Line	P	1M	M	2L	7	Desc.	No
SD	Send Data Line	S	1L	M	2L	7	Desc.	No
RF	Receive Interface	Q	CC	M	L	7	Desc.	No
SF	Send Interface	U	CC	M	L	7	Desc.	No

LEGEND

SYMBOL		DESCRIPTION
1L	=	Operand 1 Least Significant Character
1M	=	Operand 1 Most Significant Character
2L	=	Operand 2 Least Significant Character
2M	=	Operand 2 Most Significant Character
3L	=	Least Significant Character of Product or Quotient
–K	=	One Character right justified
K–	=	One Character left justified
KK	=	Two Characters
IA	=	Indirect Jump Address
DD	=	Two Decimal Digits
CC	=	Characters whose bits represent conditions of Flip Flops
JA	=	Address of Next Instruction of a Jump
M	=	Most Significant Position of Both Operand 1 and 2
L	=	Least Significant Position of Both Operand 1 and 2
ASC.	=	Ascending
DESC.	=	Descending
A+D	=	Ascending and Descending
–	=	Not Applicable
RA	=	Return Address

Table 6-2. UNIVAC 1005 Instruction Repertoire

APPENDIX A. UNIVAC 1005 ASSEMBLER CODING FORM



APPENDIX B. UNIVAC 1005 INSTRUCTION TIMING (80)

DESCRIPTION	OPERATION CODE	1005 I	1005 II AND III
Transfer Instructions			
Transfer Descending	TD	176 + 16(n)us	143 + 13(n)us
Transfer Ascending	TA	176 + 16(n)us	143 + 13(n)us
Transfer Clear	TC	176 + 16(n)us	143 + 13(n)us
Transfer Numeric	TN	176 + 16(n)us	143 + 13(n)us
Transfer Constant	TK	176 + 16(n)us	143 + 13(n)us
Transfer to Register X	TX	176 + 16(n)us	143 + 13(n)us
Translate (Optional Feature)	TR	176 + 32(n)us	143 + 26(n)us
		n = number of characters in Operand 2	
Addition and Subtraction			
Add Algebraic	AD	176 + 16(n)us	143 + 13(n)us
Subtract Algebraic	SU	176 + 16(n)us	143 + 13(n)us
Add Magnitude	AM	176 + 16(n)us	143 + 13(n)us
Subtract Magnitude	SM	176 + 16(n)us	143 + 13(n)us
Add Constant	AK	176 + 16(n)us	143 + 13(n)us
		n = number of characters in Operand 2	
Compare Instructions			
Compare Numeric	CN	176 + 16(n)us	143 + 13(n)us
Compare Magnitude	CM	176 + 16(n)us	143 + 13(n)us
Compare Alphanumeric	CA	176 + 16(n)us	143 + 13(n)us
Compare Constant	CK	176 + 16(n)us	143 + 13(n)us
		n = number of characters in Operand 2	
Condition Indicators			
Set Condition	SC	144 us	117 us
Stop	STOP	144 us	117 us
Sequence Control Instructions			
Jump Condition	JC	144 us, 208 if jump	117 us, 169 if jump
Jump Test	JT	144 us, 208 if jump	117 us, 169 if jump
Unconditional Jump	J	208 us	169 us
Jump Return	JR	272 us	221 us
Jump Compare	JK	256 us, 320 if jump	208 us, 260 if jump
Jump Loop	JL	208 us, 336 if jump	169 us, 273 if jump
Jump Indirect	JI	208 us	169 us
Count	CC	240 + 112(DD)us*	195 + 91(DD)us**
Edit Instructions			
Edit Logical	EL	240 us	195 us
Edit Erase	EE	240 us	195 us
Edit Superimpose	ES	240 us	195 us
Edit	ED	208 + 32(n)us	169 + 26(n)us
		n = number of characters in mask	
Multiply Instructions			
Multiply	MU	6.4 ms Avg.	5.2 ms Avg.
Multiply Long	ML	18.5 ms Avg.	15.0 ms Avg.
Divide			
Divide	DV	13.7 ms Avg.	11.1 ms Avg.
I/O			
General	GC	144 us + I/O Time	117 us + I/O Time
Read Tape	RT	144 us + I/O Time	117 us + I/O Time
Write Tape	WT	144 us + I/O Time	117 us + I/O Time
Receive Data Line	RD	144 us + I/O Time	117 us + I/O Time
Send Data Line	SD	144 us + I/O Time	117 us + I/O Time
Receive Interface	RF	144 us + I/O Time	117 us + I/O Time
Send Interface	SF	144 us + I/O Time	117 us + I/O Time
Indirect Addressing			
OP1 - Add to normal time		64 us	52 us
OP2 - Add to normal time		80 us	65 us

* Add 80 us for each change of row.
Add 80 us for each change of bank.

** Add 65 us for each change of row.
Add 65 us for each change of bank.

UNIVAC

DIVISION OF SPERRY RAND CORPORATION

UP-4084