

ComputerAutomation

NAKED MINI. Division

18651 Von Karman, Irvine, California 92713

Telephone: (714) 833-8830 TWX: 910-595-1767

REAL-TIME EXECUTIVE (RTX)

USER'S MANUAL

90-94500-00F2

April 1977

REVISION HISTORY

<u>Revision</u>	<u>Issue Date</u>	<u>Comments</u>
A0		Original issue.
A1 to E6		Misc. RTX/IOX updates.
F0	November 1976	Adds Magnetic Tape Intelligent Cable, Storage Module Disk, and IEEE Intelligent Cable IOX Handlers. Adds block diagrams for IOB, UAT, DIB and CIB. Adds IOX Handler listing.
F2	April 1977	Adds File Manager to IOX, and overall documentation cleanup.

TABLE OF CONTENTS

PART I. REAL-TIME EXECUTIVE (RTX)

<u>Section</u>		<u>Page</u>
1	INTRODUCTION TO RTX	
1.1	WHAT IS RTX?	I/1-1
1.2	WHEN SHOULD RTX BE USED?	I/1-2
1.3	WHAT DOES AN APPLICATION PROGRAM LOOK LIKE?	I/1-3
1.4	DEFINITIONS	I/1-3
2	RTX ORGANIZATION	
2.1	WORK AREA (USER BLOCKS)	I/2-2
2.2	RTX FUNCTIONS	I/2-2
2.2.1	Initialize Work Area (RTX:)	I/2-2
2.2.2	Initiate New Task (BEGIN:)	I/2-3
2.2.3	Terminate Current Task (END:)	I/2-6
2.2.4	Suspend Current Task (PAUSE:)	I/2-6
2.2.5	Coordinate Numbers	I/2-7
2.2.6	Inter Task Coordination (PUT:/GET:)	I/2-7
2.2.7	Delay Current Task (DELAY:)	I/2-8
3	INTERRUPT PROCESSING	
3.1	SAVE ENVIRONMENT (INTSV:)	I/3-2
3.2	RESTORE ENVIRONMENT (INTRS:)	I/3-2
3.3	INITIATE A NEW TASK FROM AN INTERRUPT SUBROUTINE (INTAC:).	I/3-2
3.4	INTERRUPT SERVICE AND QUEUE TASK (INTQ:)	I/3-3
3.5	COMMON SUBROUTINES (REENTRANCE)	I/3-4
3.6	WRITING COMMON SUBROUTINES	I/3-5
3.7	CALL A COMMON SUBROUTINE (SUBR:)	I/3-6
3.8	EXIT FROM COMMON SUBROUTINE (SUBX:)	I/3-6
3.9	PROTECT A FACILITY (PROT:)	I/3-7
3.10	RELEASE A PROTECTED FACILITY (UNPR:)	I/3-8
3.11	LOCK OUT A FACILITY (LOCK:)	I/3-10
3.12	UNLOCK A LOCKED FACILITY (UNLK:)	I/3-11

TABLE OF CONTENTS (Cont'd)

<u>Section</u>	<u>Page</u>
3.13 ABORT A TASK (ABORT:)	I/3-11
3.14 OBTAIN CURRENT PRIORITY (GETPR:)	I/3-11
3.15 SET TASK PRIORITY (SETPR:)	I/3-12
3.16 INCREMENT TASK PRIORITY (INCPR:)	I/3-12
3.17 DECREMENT TASK PRIORITY (DECPR:)	I/3-12
4 ADDITIONAL RTX REATURES	
4.1 RTX DEBUG REATURE (ZBG)	I/4-1
4.2 PROGRAM LOAIDNG WITH ZBG	I/4-4
4.3 POWER-FAIL, AUTO-RESTART (PWREL:)	I/4-4
4.4 TELETYPE INPUT/OUTPUT	I/4-4
4.5 LSI-3/05 SOFTWARE CONSOLE ROUTINE (CNSOL3)	I/4-4
5 RTX OPERATING PROCEDURES	
6 A SAMPLE RTX PROGRAM - RTX DEMO	
6.1 PROGRAM DESCRIPTION	I/6-1
6.2 PROGRAM MODULE FUNCTIONS	I/6-2
6.2.1 BEGIN	I/6-3
6.2.2 TASK1	I/6-3
6.2.3 TASK2	I/6-4
6.2.4 TASK3	I/6-4
6.2.5 IOTASK	I/6-4
6.2.6 ADD1	I/6-4

TABLE OF CONTENTS

PART II. INPUT/OUTPUT EXECUTIVE (RTX)

<u>Section</u>		<u>Page</u>
1	IOX GENERAL DESCRIPTION	
1.1	GENERAL DESCRIPTION	II/1-2
1.2	CALLING SEQUENCES	II/1-3
1.3	DEVICE DEDICATION	II/1-5
1.4	LOADING	II/1-5
1.5	RESTARTABILITY	II/1-7
2	IOB AND UAT ORGANIZATION	
2.1	INPUT/OUTPUT BLOCK (IOB) - 10 words	II/2-1
2.2	UNIT ASSIGNMENT TABLE (UAT)	II/2-6
2.3	STANDARD DIB NAMES	II/2-7
2.4	SAMPLE UAT	II/2-8
3	I/O HANDLER ORGANIZATION	II/3-1
3.1	THE STANDARD HANDLERS	II/3-1
3.1.1	Character-Oriented Device Handler (Non-Fortran)	II/3-1
3.1.2	Fortran List Device Handler	II/3-1
3.1.3	Card Reader Handler	II/3-1
3.1.4	Magnetic Tape Handler	II/3-2
3.1.5	Disk and Storage Module Disk Handler (Non-Fortran).	II/3-2
3.1.6	Floppy Disk Handler (Non-Fortran)	II/3-3
3.1.7	Disk, Storage Module Disk, and Floppy Disk Handler (Fortran)	II/3-3
3.1.8	Magnetic Tape Intelligent Cable (MTIC) Handler	II/3-4
3.2	I/O HANDLER REQUIREMENTS	II/3-4
3.2.1	SINT: (Set up an instruction at the Word Interrupt Location)	II/3-5
3.2.2	SIO: (Start I/O and Watchdog Timer)	II/3-5
3.2.3	INTP: (End of Block Interrupt Return Point)	II/3-7
3.2.4	WAIT: (End of Record Delay Routine)	II/3-9
3.2.5	EOFQ: (End of File Check Routine)	II/3-9
3.2.6	EOF: (End of File Routine)	II/3-10
3.2.7	EOR: (End of Record Routine)	II/3-10
3.2.8	EORST: (Alternate Entry Point to EOR:)	II/3-10
3.2.9	FETCH: (Input one character from an I/O device)	II/3-11
3.2.10	BUFFQ: (Store input character into buffer)	II/3-12
3.2.11	UNRES: (Unresponsive Device Routine)	II/3-12
3.2.12	IORTN: (Return to I/O Scheduler)	II/3-13
3.3	CHARACTER-ORIENTED DEVICE HANDLER LISTING	II/3-13



TABLE OF CONTENTS (Cont'd)

<u>Section</u>	<u>Page</u>
DIB AND CIB DESCRIPTIONS	
4.1	DEVICE INFORMATION BLOCK (DIB) - 11 to 18 words II/4-1
4.2	REGULAR DIB CONFIGURATION (ALL HANDLERS) - WORDS 0 TO 10 . . . II/4-3
4.3	ADDITIONAL DIB CONFIGURATIONS - UP TO 18 WORDS II/4-6
4.3.1	Distributed I/O DIB II/4-6
4.3.2	Magnetic Tape Intelligent Cable DIB II/4-7
4.3.3	Disk DIB II/4-9
4.3.4	Fortran Disk DIB II/4-11
4.3.5	Storage Module Disk DIB (Fortran and Non-Fortran) . . . II/4-12
4.4	SAMPLE DISK DIB II/4-14
4.5	CONTROLLER INFORMATION BLOCK (CIB) - 38 WORDS (47 WORDS FOR STORAGE MODULE DISK) II/4-14
4.6	STANDARD CIB NAMES II/4-17
5 FILE MANAGER	
5.1	FILE ORGANIZATION II/5-1
5.1.1	Sequential File Access II/5-5
5.1.2	File Opening and Closing II/5-5
5.1.3	File Positioning II/5-6
5.1.4	File Functions II/5-7
5.2	TABLE ORGANIZATION II/5-9
5.2.1	File Device Information Block (DIB) II/5-9
5.2.2	Controller Information Block (CIB) II/5-15
5.3	RTX FILE LABEL UTILITY II/5-18
5.3.1	Environment II/5-18
5.3.2	Program Operation II/5-18
6 DEVICE DEPENDENT CONSIDERATIONS II/6-1	
6.1	STANDARD CHARACTER DEVICE HANDLERS II/6-1
6.1.1	Line Printer II/6-1
6.1.2	Teletype Keyboard (TK) II/6-1
6.1.3	Teletype Console (TY) (implies tape reader or keyboard for input, whichever is ready) II/6-2
6.1.4	Teletype Reader (TR) II/6-2
6.1.5	Teletype Punch (TP) II/6-2
6.1.6	Card Reader (CR) II/6-3
6.1.7	High Speed Reader (PR) II/6-3
6.1.8	High Speed Punch (PP) II/6-3



LIST OF ILLUSTRATIONS (PART I)

<u>Figure</u>		<u>Page</u>
1-1	Typical Example of RTX	I/1-4
1-2	RTX Software Configuration	I/1-5
6-1	RTX Demo Program - Flow Diagram (Sheet 1)	I/6-5

LIST OF ILLUSTRATIONS (PART II)

2-1	IOB Configuration	II/2-2
2-2	UAT Configuration	II/2-6
4-1	DIB Configuration	II/4-2
4-2	CIB Configuration	II/4-15
5-1	Disk Directory Structure	II/5-2
5-2	Disk Description Table (DDT) in Volumn Table of Contents.	II/5-3
5-3	Disk File Linkage	II/5-4
5-4	Sequential File Positioning Examples	II/5-8
5-5	Table Organization	II/5-10
5-6	DIB Definition When used With the File Manager	II/5-11
5-7	CIB Definition When Used With the File Manager	II/5-16
7-1	IEC IOB Configuration -- 9 to 12 words	II/7-2
7-2	IEC Status Byte Configuration	II/7-6
7-3	IEC Set Mode Command Word Format	II/7-7
7-4	IEC CIB Configuration	II/7-9

LIST OF TABLES

<u>Table</u>		<u>Page</u>
2-1	User Blocks for RTX Functions	I/2-4



PART I

REAL-TIME EXECUTIVE (RTX)

SECTION 1

INTRODUCTION TO RTX

This section presents an overview of Computer Automation's Real Time-Executive (RTX) program which operates on all ALPHA-16 and LSI processors. The following discussion is concerned with three basic questions:

1. What is RTX?
2. When should RTX be used?
3. What does an application program look like?

1.1 WHAT IS RTX?

RTX is a modular package of service routines that handles both the overhead functions and the scheduling services associated with a real-time environment. Modular construction allows you to select only the portions of RTX required for your application. Real-time environment means that if your application requires that certain tasks be performed at selected intervals or in response to an external signal or event, then RTX will manage the orderly interruption and resumption of your program. RTX does all the overhead functions to maintain and direct the execution of your application during both normal and real-time processing.

RTX is also a powerful multi-task executive that controls all tasks of the overall application. These tasks include priority scheduling, response and assignment, interrupt servicing, and communication among RTX tasks and user-developed handlers. Overall task control:

1. Allows the application program to be designed as a number of either inter-related or subordinate tasks. The nature of the application determines the task relationships. RTX will completely handle the switching from task to task as required.
2. Allows the application program to dynamically define (and redefine) the priority level of the various tasks in the application using RTX service routines. This is a software priority which is then used by the RTX scheduler function to direct the sequence of task execution.
3. Allows RTX priority scheduling, response and assignment to share the computer among tasks with equal priority. When all tasks of the highest priority are temporarily waiting for some event to occur, the next highest priority level is scheduled in the same manner.
4. Allows response to interrupts, as generated, because the user provides the interrupt instructions which transfer control to an interrupt service routine. This interrupt service routine will save status (using an RTX function), perform the necessary instructions to assure no data loss, and then restore status (using an RTX function). This routine can also cause a lower priority routine to be

Scheduled if additional processing of the interrupt data is required; the lower priority routine can be temporarily deferred until any higher priority tasks have had their turn at executing.

Allows the various tasks in the application to communicate between themselves (or with RTX) through RTX communication routines. These routines allow a task to uniquely identify the communication request and then post it. Posting consists of presenting information to, or requesting information from, another task. This facility may be used to operate simply as a signaling device, or it may be as complex as both a signaling and parametric (pointer-passing) function.

All of these RTX features combine to produce a multi-tasking, real-time scheduling executive that is, despite its small size, the most powerful and easy to use system of its kind on the market. Figure 1-1 illustrates a typical example of RTX.

2. WHEN SHOULD RTX BE USED?

The most significant reason for using RTX is that your application program requires a real-time environment. Real-time environments are found in many circumstances, ranging from high speed data acquisition to occasional sampling of an electro-mechanical device such as a relay. The basic criterion is that a need exists for the application to communicate with some external device or event in a time-dependent manner. If this criterion is met, then RTX is a suitable vehicle for defining the relationship between the external device or event and the application programming tasks which control and service that device or event. Some of the more obvious applications are:

1. Communications
 - Message Switching
 - Store-and-Forward
 - Networks
 - Reservation Systems
2. Process Control
 - Plant Operations
 - Flow Monitoring
 - Equipment Direction
 - X-Y Positioning
 - Petro-chemical Applications
3. Data Acquisition
 - Test cells, such as automotive or airframe/aircraft
 - Traffic Control
 - Instrumentation Control
 - Source Data Entry
 - Oil Field Data Monitoring
4. Medical Data Processing
 - EKG/EEG Analysis
 - Patient Monitoring
 - Cardiac Monitoring
 - Patient Billing



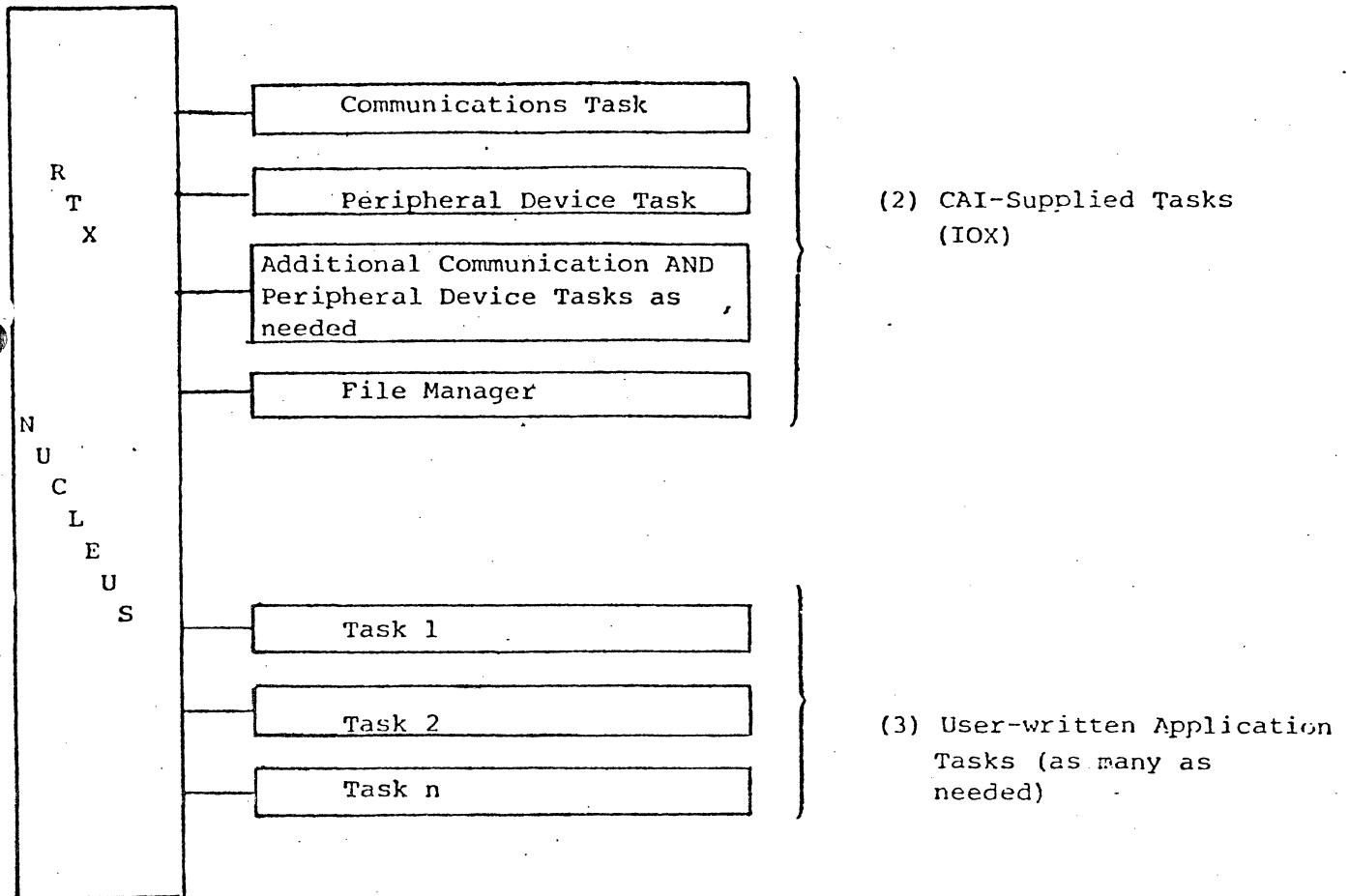
5. Security Systems
 - Plant/Facility Security
 - X-Ray Security Systems
 - Video Transmission Systems
6. Financial Transactions
 - Point-of-Sale
 - Automatic Banking
 - Inventory Control

1.3 WHAT DOES AN APPLICATION PROGRAM LOOK LIKE?

RTX allows the user to construct his application in modules. These modules are then combined with RTX during the loading process to produce the final application program. The user may choose any arrangement of his program into modules that suit his needs. Figure 1-1 shows a general diagram of this type of arrangement. This modularity concept applies not only to the user's application, but also to RTX itself. The RTX package is simply a library of separate subroutines which may be referenced by the user's modules; certain of the RTX subroutines in turn reference others, and the linking of all required modules (performed by the LAMBDA loader or by the OS:LNK program) results in a configuration consisting of only those modules needed for the application. Figure 1-2 shows how the modules and user programs are loaded into memory and the size of the individual RTX modules. Keep in mind that the only RTX modules actually loaded for a given program will be the ones required by the particular program.

1.4 DEFINITIONS

1. Activity: A task which has been initialized (via BEGIN: for example) and is receiving support from RTX.
2. Common Subroutines: Subroutines which may be used by two or more different activities concurrently. These require special coding to provide reentrant capability.
3. Coordination Number: A decimal integer used to identify a task to RTX. It is analogous to a telephone number in that it is used to "connect" a task to another task or to the DELAY: process.
4. Interrupt Data Processing: That portion of code that processes the data obtained by an Interrupt Service routine.
5. Interrupt Service: That portion of code that must be executed immediately after the interrupt occurs (so as not to lose data). It should be limited to only that code which is necessary to assure no data loss.
6. Inter-Task Coordination: A method for tasks to communicate and pass parameters using two 16-bit computer words. These words may contain any information, such as a table address, a pointer to a list of values, or a value itself.



- (1) RTX Nucleus provides control, scheduling, priority handling.
- (2) CAI-supplied tasks provide handlers for I/O (such as printers, tapes, etc.), for communications (such as BISYNC, ASYNC, etc.), and others.
- (3) The user need only supply tasks which perform his application's work, while utilizing the CAI-supplied software for support.

Figure 1-1. Typical Example of RTX

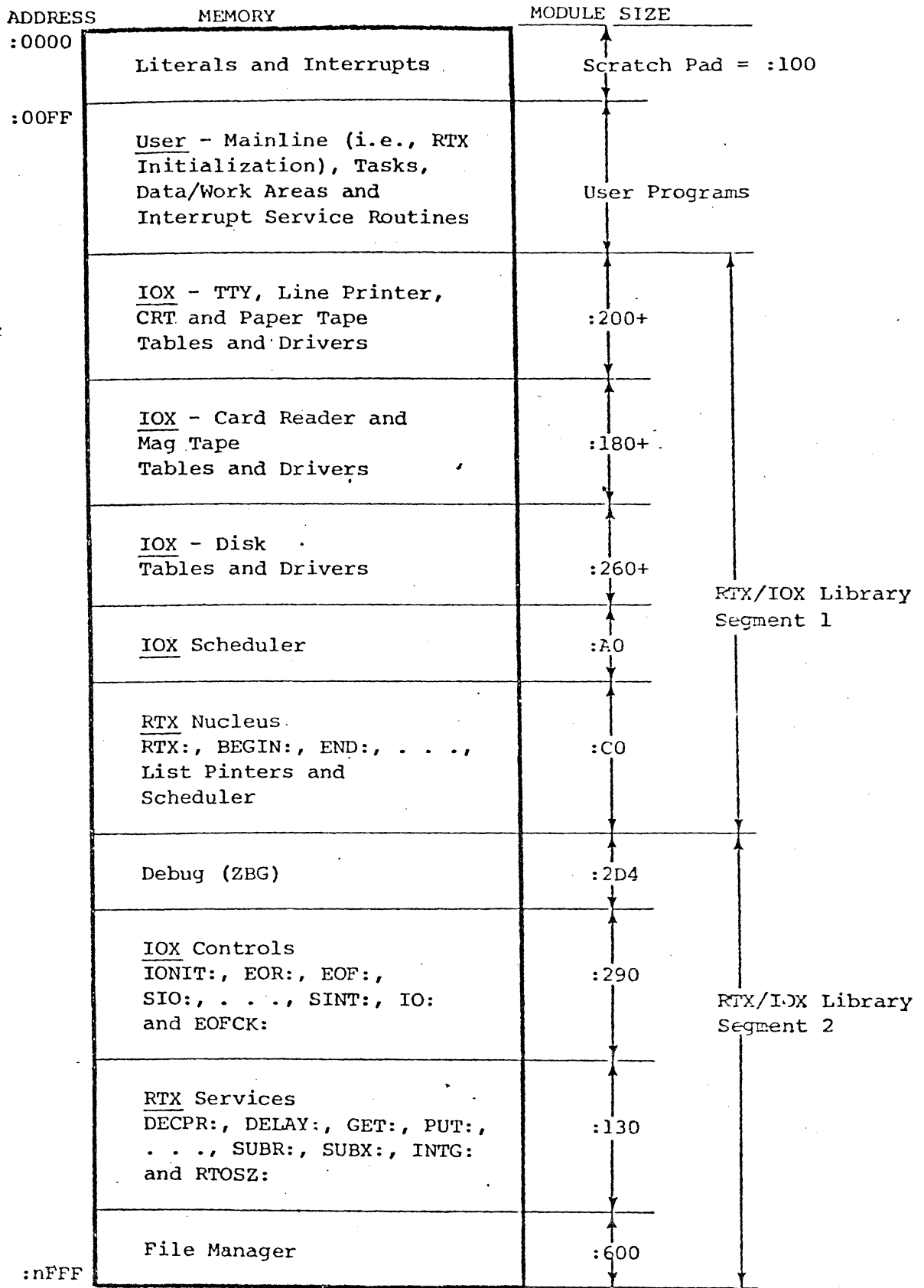


Figure 1-2. RTX Software Configuration

7. Main Line: A short initializing sequence which resets all task table pointers, and then begins one or more tasks. (Tasks may also be begun by other tasks, or upon an interrupt from an external device.)
8. Priority: A software defined method for assigning (and re-assigning) the relative importance of a task to RTX.
9. Re-entrant Programs: A program specifically written such that it may be directly entered by more than one program, concurrently. Under RTX, this is necessary only if two or more Interrupt Service routines require immediate use of the same program. For example, Interrupt Service A calls routine C. While C is executing, Interrupt Service B becomes active and also calls routine C. If C were not re-entrant, this second call to C would replace the return address at C's entry point, causing the return address for routine A to be lost.
10. Task: A program or set of programs which operate to perform a specific function within the real-time application.
11. Work Area: An area of storage dedicated to table space for RTX. This table contains all the necessary information for RTX to perform its functions. Its usage is dynamic and is dependent upon the maximum concurrent usage of RTX functions.

SECTION 2

RTX ORGANIZATION

RTX is basically a collection of functions (subroutines) and a user-supplied work area, which are linked to the user's Mainline sequence and tasks prior to execution. Each RTX function may be called as a subroutine by the user as it is needed, to perform a specific job. (See below for descriptions and calling sequences of these functions.) RTX also includes a task scheduler (SCHED:) which is used to execute the task of highest priority. The priority of a task is defined when the task is begun, and may be changed by the task, using the SETPR:, INCPR:, and DECPR: functions. Priorities may range from 1 to 8191, with larger numbers representing the higher priority.

The scheduler maintains a "Ready" list of each task in order of priority. The highest priority task is executed until it suspends itself by calling any of the following RTX functions:

- DELAY: (unless altering or cancelling a previous delay)
- GET: (if no corresponding PUT: yet, and not a cancel call)
- SUBR: (if the common subroutine is busy)
- PAUSE: (essentially reschedules the pausing task at the same priority)
- IO: (BEGINS, at I/O completion time, the normal or abnormal return at the same priority)
- SETPR: (if the new priority is lower than that of another task)
- DECPR: (if the new priority is lower than that of another task)

Once the task has been suspended, RTX executes the new highest priority task. The rule for determining the highest of equal priority tasks is, "first in - first out". Thus, if a task suspends itself, it thereby becomes "last in" within its priority.

In addition to the user-invoked suspends listed above, occurrence of an interrupt will cause a task to be suspended, if the new priority is higher than that of the current task. An interrupt is defined to be:

1. A hardware (external) interrupt, with INTQ: or INTAC: attached, or
2. A software (internal) interrupt:
 - a DELAY: expiring
 - a PUT: which satisfies an outstanding GET:
 - a SUBX, UNLK:, or UNPR:, with a higher priority task waiting



In addition, an Input Output Executive package (IOX) is available, which may be linked to run in conjunction with RTX. Its function is to perform I/O operations to the standard CAI I/O devices (teletype, high speed paper tape reader and punch, card reader, magnetic tape units, and disk) and resolve conflicts of concurrent I/O utilization.

A File Manager operates in conjunction with IOX. It enables the user to communicate with data files by name, independent of the physical medium storing the file. Requests for access are made through IOX using Logical Units (LUNs).

2.1 WORK AREA (USER BLOCKS)

The user must supply a contiguous work area for RTX to build its tables. The address and length of this work area is specified in the call to the RTX: function. It is grouped by RTX into blocks of five words each, and there must be at least two of these blocks (10 words) reserved; otherwise an error return will be made from the initialization routine. Table 2-1 gives a list of the RTX functions which allocate and de-allocate this area. The left hand column denotes the number of blocks allocated (+) or de-allocated (-) by the function in the right-hand column. The user must supply sufficient work area for the maximum number of five-word blocks which may be allocated at any one time.

2.2 RTX FUNCTIONS:

2.2.1 Initialize Work Area (RTX:)

Calling Sequence:

N	EQU	(NUMBER OF TASK BLOCKS)	
WKAREA	RES	N+N+N+N+N,0	AREA FOR BLOCKS
	JST	RTX:	
	DATA	N	# OF CONCURRENT ACTIVITIES
	DATA	WKAREA	
	ERROR	RETURN	WORK AREA EXCEEDED
	NORMAL	RETURN	

Returns With:

INTERRUPTS ENABLED
 OVERFLOW RESET
 WORD MODE
 A REGISTER --- CURRENT RTX REVISION NUMBER IN ASCII
 X REGISTER --- CURRENT RTX REVISION NUMBER IN ASCII

This subroutine is called in the user's Mainline sequence to initialize the working area of RTX. The work area is broken into N blocks of five words each, which are then used by the remainder of RTX during system operation. The number N must be large enough to allow for all concurrent activities. Work area overflow will cause a jump to the RTX: routine's error return at any subsequent time during the running of the program, not just during the call to RTX:.

**NOTE**

A call to this subroutine causes activation of the RTX Scheduler. Upon return, the calling program (normally the user's Mainline sequence) is thenceforth considered a task with a priority of 8172.

In addition to initializing the work area, the RTX: subroutine can also reset all I/O tables, if desired; this feature will insure restartability of a user's program. The feature may be referenced in the user program, if restart capability is required; otherwise it may be omitted, thereby shortening the overall length of the program. (Upon initial loading, I/O reset is not required before execution.)

To include this feature in the RTX: subroutine, simply reference the module "IONIT:" in the Mainline sequence; either of the following directives:

```
IONIT: REF  
or  
LOAD IONIT:
```

will serve this purpose.

2.2.2 Initiate New Task (BEGIN:)

Calling Sequence:

```
JST      BEGIN:  
DATA    (*) START ADDRESS OF NEW TASK  
DATA    PRIORITY OF NEW TASK
```

Returns With:

```
INTERRUPTS --- ENABLED  
OV --- UNCHANGED  
A REGISTER --- UNCHANGED  
X REGISTER --- UNCHANGED
```

NOTE

When the new task starts executing, the A and X registers will contain the values at the time of the JST to BEGIN:, OV will be reset, and the computer will be in word mode.

This subroutine is called to initiate a new task. The task is scheduled and BEGIN: then exits to the task Scheduler. This means that the calling program will not receive control back immediately if the new ("begun") activity is of higher priority, or if another task of higher priority is ready to begin execution.



Table 2-1. User Blocks for RTX Functions

No. of Blocks	Function
+1	RTX:
+1	BEGIN:
-1	END:
0	PAUSE:
+1	PUT: (If a new, unique PUT: and no corresponding GET: is waiting for it)
0	PUT: (If a new unique PUT: and the corresponding GET: is already waiting for it)
0	PUT: (To change the information in a previous PUT:)
-1	PUT: (To cancel an outstanding PUT:)
0	GET: (If a new, unique GET: and no corresponding PUT: is waiting for it)
-1	GET: (If a new, unique GET:, and the corresponding PUT: is already waiting for it)
-1	GET: (To replace a previous task currently waiting for a PUT: with the current task; the new GET: must be called with the same coordination number as the task to be replaced)
-1	GET: (To cancel an outstanding GET:)
0	DELAY: (To initiate a new delay)
0	DELAY: (To change the length of an outstanding delay)
-1	DELAY: (To cancel an outstanding delay)
0	INTSV:
0	INTRS:
+1	INTAC:
+1	INTQ:
+1	SUBR: (If the common subroutine is not already in use)
0	SUBR: (If the common subroutine is already in use)



Table 2-1. User Blocks For RTX Functions (Continued)

No. of Blocks	Function				
-1	SUBX: (If no other tasks are waiting to use the common sub-routine)				
0	SUBX: (If one or more tasks are waiting to use the common sub-routine)				
+1	PROT: (If the facility is not already protected)				
0	PROT: (If the facility is already protected)				
-1	UNPR: (If no other tasks are waiting to protect the facility)				
0	UNPR: (If one or more tasks are waiting to protect the facility)				
+1	LOCK: (If the facility is not already locked)				
0	LOCK: (If the facility is already locked)				
-1	UNLK: (If no other tasks are waiting to LOCK: the facility)				
-1	ABORT: (In addition, -1 for each resultant SUBX: call where no other tasks are waiting to use the common subroutine, and -1 for each resultant UNPR: and UNLK: call where no other tasks are waiting to PROT: or LOCK: the facility)				
0	GETPR:				
0	SETPR:				
0	INCPR:				
0	DECPR:				
0	IOREL:				
0	IOWAT:				
3 or 4	IO: (as follows:) <table data-bbox="560 1661 1593 1768" style="margin-left: 20px;"> <tr> <td>+1 For the immediate return</td> <td>+1 For setting a watchdog timer</td> </tr> <tr> <td>+1 For scheduling</td> <td>+1 If I/O completes before scheduling completes</td> </tr> </table>	+1 For the immediate return	+1 For setting a watchdog timer	+1 For scheduling	+1 If I/O completes before scheduling completes
+1 For the immediate return	+1 For setting a watchdog timer				
+1 For scheduling	+1 If I/O completes before scheduling completes				

NOTE

Priorities are integers from 0 (lowest) to 8191 (highest). Users should limit priority to less than 7000 because certain RTX functions use those of 7000 and higher.

2.3 Terminate Current Task (END:)

Calling Sequence:

JST END:

The current task may terminate itself with a call to END:. No arguments are required and control will not return.

NOTE

The Mainline sequence (as a result of the JST to RTX:) has a priority of 8172. This sequence should begin other necessary tasks and then terminate itself by a call to END:. If it does not terminate, no tasks of a lower priority can execute.

2.4 Suspend Current Task (PAUSE:)

Calling Sequence:

JST PAUSE:

Returns With:

INTERRUPTS --- ENABLED
STATUS --- UNCHANGED
A REGISTER --- UNCHANGED
X REGISTER --- UNCHANGED

This subroutine is called by a program which desires to allow other tasks at the same priority level to get service. This is useful if a program is unusually long or is a closed loop. PAUSE: is essentially similar to a BEGIN:, END: pair, but is less demanding on work area space in RTX.

NOTE

Programs which loop indefinitely are permissible, but should be used carefully since they will block execution of all activities of a lower priority. Tasks should begin in response to a stimulus, generate the appropriate reaction, and end.



2.2.5 Coordination Numbers

Before discussing GET:, PUT:, and DELAY: the concept of coordination number must be understood. A coordination number is a 16-bit value which is supplied as an argument to GET:, PUT:, DELAY:, PROT:, UNPR:, LOCK:, UNLK:, IO: and IOREL:. This number serves to identify the activity so that it may be referenced by a later call.

For GET:, PUT: AND DELAY:, the same coordination number used in the same type of call supersedes the previous call. The negative (2's complement) of a coordination number cancels the previous call. FORTRAN uses the following coordination numbers, and the designer should avoid their re-use:

F:RBPG address (for LOCK:)
:FFDC (for LOCK:)

In addition, all DELAYS performed in IOX and COMX use memory addresses as coordination numbers. These memory addresses fall within the IOX or COMX boundaries, or their associated tables (CIB's). Thus, it is strongly suggested that the system designer follow this practice, and use as coordination numbers, only memory addresses of locations within his program. Basically, it is the system designer's responsibility to allocate coordination numbers so that no conflicts arise.

NOTE

Zero has no separate identifiable two's complement, and therefore a coordination number of zero should not be used.

2.2.6 Inter Task Coordination (PUT:/GET:)

These two facilities are generally used together as a pair. In general, PUT: passes 32 bits (the A and X registers) to a GET:. Coordination numbers are used to insure proper reference. There are no timing restrictions on associated PUT:/GET: pairs. (If a task calls GET: before another task has made the corresponding PUT: call, the GETting task will suspend until the PUT: is made.)

PUT:

Calling Sequence:

JST	PUT:
DATA	COORDINATION NUMBER

Returns With:

INTERRUPTS	---	ENABLED
STATUS	---	UNCHANGED
A REGISTER	---	UNCHANGED
X REGISTER	---	UNCHANGED

This subroutine is called to do one of three things:

1. Pass 32 bits to another task; call PUT: with the same (positive) coordination number which will be used in the call to GET;



Change the information in a previous PUT:; call PUT: with the same coordination number used previously.

Delete an outstanding PUT:; call PUT: with the 2's complement of the coordination number of the PUT: to be deleted.

NOTE

If a PUT: is issued before the associated GET: is called, one block is used from the work area in RTX. If the GET: is called first no additional demands are made on the work area.

ET:

Calling Sequence:

JST	GET:
DATA	COORDINATION NUMBER

Returns With:

INTERRUPTS --- ENABLED
 STATUS --- UNCHANGED
 A REGISTER --- FROM ASSOCIATED PUT
 X REGISTER --- FROM ASSOCIATED PUT

This subroutine is called for one of three reasons:

To obtain 32 bits (A and X registers) from another task: call GET: with the positive coordination number to be used with PUT:.

To delete a task currently in a GET: waiting for the associated PUT:; call GET: with the 2's complement of the coordination number.

To replace a task currently waiting for a PUT: with the current task; call GET: with the same coordination number as the task to be replaced.

If GET: is called, control will not be returned until the associated PUT: is issued.

2.7 Delay Current Task (DELAY:) (Requires Real-Time Clock Option)

Calling Sequence:

JST	DELAY:
DATA	# OF TICKS ON THE CLOCK FOR DELAY
DATA	COORDINATION NUMBER

Returns with:

INTERRUPTS --- ENABLED
 STATUS --- UNCHANGED



If deleting or changing an outstanding delay:

A REGISTER --- UNCHANGED
X REGISTER --- UNCHANGED

If actually executing a delay:

A REGISTER --- COORDINATION NUMBER
X REGISTER --- UNDEFINED

This subroutine is called for one of three reasons:

1. To delay the current task for a specified period of time. (The number of ticks referred to above is the number of time interrupts from the Real-Time Clock. These interrupts normally occur every 10 msec but may be changed by a jumper wire. (See the appropriate ALPHA-16 or ALPHA LSI Computer Reference Manual). For this call, supply a currently unused positive coordination number.
2. To delete an outstanding delay. A call to DELAY: with the 2's complement of the coordination number of any current delay will delete the delay request (and the task that called it). This is useful for deleting a watchdog routine.
3. To change an outstanding delay. A call to DELAY: with the coordination number of a currently active delay will change the outstanding delay. This is equivalent to deleting a task in a delay and immediately starting the same task with a new delay.



SECTION 3

INTERRUPT PROCESSING

Most interrupt service routines can be divided into two sections. First, the recognition that the requesting device usually has an immediate need which will result in data being lost if it is not met. Second, a subsequent need to perform some processing upon that data. In the case of output, the device may not continue to operate at full speed if its request is not answered within a certain interval. After meeting this very high-speed requirement, the need for continued rapid servicing diminishes considerably, until the next request is made.

RTX provides two alternative methods for interrupt service. One is the INTQ: service, which combines the functions of saving status, queueing or scheduling of support tasks, and then dismissing the interrupt since it has been honored. The second is to use the INTSV:, INTAC:, and INTRS: services to provide each of those three functions separately. Use of these three functions is described below.

Upon receiving control after an interrupt, the interrupt handler should immediately call INTSV:, to preserve the register status. When control returns, the handler may utilize the registers as required. Processing, at this point, should be restricted to the very high speed "lost data" requirements. The handler may then schedule other activities, by calling INTAC:, with the start address and priority as arguments. Processing is ended for this phase, by issuing a call to INTRS:, which resumes processing. Normally, the newly scheduled activity will have a high priority. Note, however, that the programmer may assign this priority, as distinct from those systems where the hardware has the device priorities wired in. When the scheduled processing activity receives control, it will be considered a normal activity, and may make use of all RTX functions. Interrupts will be enabled, so that other devices which require service may receive control during their "lost data" intervals, after which the System Scheduler will return control to the highest priority processing program.

The A and X register are passed between the scheduling and the scheduled routines, so that word or byte transfer devices can pass the data itself to the processing programs. After the processing program has finished its task, it may terminate, or it may schedule other responding tasks.

By using INTSV: and INTRS: to save and restore status, the user is relieved of one of the most important and error-prone types of coding. With INTAC:, he can schedule routines which are normal, interruptable programs, and which can utilize all of RTX's capabilities.

Note that the INTSV:, INTRS:, INTAC:, and INTQ: routines are necessary only for the user who is using RTX in conjunction with his own special (non-standard) device and has written his own interrupt handler for it. The RTX I/O Executive (IOX), discussed in Chapter 2 of this manual, contains the necessary I/O handler routines for the standard CAI-supplied I/O devices (card reader, teletype, high speed paper tape punch and reader, magnetic tape, disk and floppy disk). These standard handlers within IOX make use of the INTQ: routine internally.



3.1 SAVE ENVIRONMENT (INTSV:)

Calling Sequence:

JST	INTSV:	INTERRUPTS MUST BE DISABLED
DATA	*PLOC	LOCATION OF ENTRY POINT TO INTERRUPT ROUTINE

Returns With:

INTERRUPTS---STILL DISABLED
 STATUS---OV RESET, WORD MODE
 A REGISTER---SAVED P REGISTER
 X REGISTER---UNCHANGED

This subroutine must be called by an interrupt subroutine to save the current environment.

3.2 RESTORE ENVIRONMENT (INTRS:)

Calling Sequence:

JST	INTRS:
---	DOES NOT RETURN

This subroutine is called by an interrupt subroutine to exit. If RTX was interrupted, control is returned to RTX. Otherwise, task control is moved to the block at the top of the scheduler ready chain and the system Scheduler is entered.

3.3 INITIATE A NEW TASK FROM AN INTERRUPT SUBROUTINE (INTAC:)

Calling Sequence:

JST	INTAC: (MUST BE IN WORD MODE)
DATA	(*) START ADDRESS
DATA	PRIORITY

Returns With:

INTERRUPTS---UNCHANGED
 OV---INDETERMINATE
 A REGISTER---DESTROYED
 X REGISTER---DESTROYED



3.4 INTERRUPT SERVICE AND QUEUE TASK (INTQ:)

This service may be used in place of the INTSV:, INTRS:, INTAC:, sequence. It is functionally identical to the combination of those three services when they are used as follows:

SUBENT	ENT		
	JST	INTSV:	SAVE ENVIRONMENT
	DATA	*PLOC	
	JST	INTAC:	QUEUE "TASKC" AT "PRIOR"
	DATA	TASKC, PRIOR	
	JST	INTRS:	DISMISS INTERRUPT AND GO TO RTX SCHEDULER

The advantage to using INTQ: is that it is faster; i.e., it shortens the period of time during which interrupts are disabled.

Calling Sequence:

JST	INTQ:	
DATA	\$,0,0,0	CALLING LOCATION, 3 TEMPS REQUIRED
DATA	TASK-ADDRESS	FOR TASK WHICH IS QUEUED
DATA	PRIORITY	FOR QUEUED TASK
DATA	A-REGISTER	VALUE PASSED TO QUEUED TASK IN A
DATA	X-REGISTER	VALUE PASSED TO QUEUED TASK IN X
DATA	P-LOC	LOCATION OF SAVED P-REGISTER AT TIME OF INTERRUPT

Returns With:

DOES NOT RETURN. QUEUES TASK FOR SCHEDULER AND DISMISSES INTERRUPT.

Sample Usage

1. Interrupt for End-of-Block

EOBENT	ENT		VECTORED INTERRUPT
	JST	INTQ:	
	DATA	\$,0,0,0	
	DATA	TASKB, PRIORB, 0, 0, EOBENT	



2. Interrupt for Data (Input) Ready

DATENT	ENT		VECTORED INTERRUPT
	SIN	3	BLOCK BYTE MODE
	STA	AREG	SAVE A-REG
	INA	ADDR,FCN	INPUT THE DATA VALUE
	EMA	AREG	RESTORE PROPER A-REG AND PASS INPUT VALUE TO QUEUED TASK
	JST	INTQ:	
	DATA	\$,0,0,0	
	DATA	TASKA,PRIORA	
AREG	DATA	0	A-REG VALUE FOR TASK
XREG	DATA	0	X-REG VALUE FOR TASK
	DATA	DATENT	RETURN POINTER FROM INTERRUPT

3.5 COMMON SUBROUTINES (REENTRANCE)

Normally, different activities are independent of each other. However, it is not unusual to have two unrelated programs use the same utility subroutines, therefore defining a "common" subroutine. One example would be mathematical functions library routines. Rather than duplicating copies in each using program, a single copy is loaded, and entered with subroutine calls (JST instructions). If control is within the common subroutine when an interrupt occurs, and another program gains control and re-calls the subroutine, the second call will destroy the return location of the first. When control finally returns to the middle of the interrupted subroutine (clearing the interrupt), it will complete its execution, and again return to the second caller. The original caller never sees control come back. The later caller gets two returns from one call. This dilemma is referred to as the common subroutine problem, and it occurs in any system which allows interrupt processing. It is solved in different ways. Most simply, common subroutines can be forbidden. Alternatively, push-down stacks are utilized, scratch storage is forbidden, (except in the stack), and the programming task is made significantly more imposing.

RTX has implemented an alternative solution to this problem, that of a "shared" facility. In our context a shared facility is a body of code which may be called concurrently from more than one task. In this sense, a shared facility is then common to several tasks.

This implementation consists of two services which are contained in RTX. These are:

- SUBR: To initiate the execution of a shared facility
- SUBX: To return from a shared facility

To illustrate usage of these services, consider the following example. If the subroutine CUP is a common subroutine to two tasks (named COFFEE and TEA), then it is possible that an interrupt could occur which causes task COFFEE to execute before task TEA Finished. This means that subroutine CUP could be entered from COFFEE before it completed the processing due to its prior entry from TEA. In this case, subroutine CUP is in common usage. It is designated as a shared facility and must be

Designed to accomodate that condition. The method here is to use the following sequence of code in both COFFEE and TEA whenever it is desired to call subroutine CUP:

```

JST      SUBR:      ACTUALLY CALL SUBR: SERVICE
DATA    CUP        NAME OF COMMON SUBROUTINE
    
```

instead of the usual method

```

JST      CUP
    
```

NOTE

NEVER call a common subroutine directly; that is, with a JST name.
ALWAYS call a common subroutine using

```

JST      SUBR:      CALL THE SUBR: SERVICE
DATA    NAME       NAME OF COMMON SUBROUTINE
    
```

(or using the LOCK: or PROT: routines described below).

3.6 WRITING COMMON SUBROUTINES

The rules for writing a common subroutine are very simple. They apply to the subroutine exit instruction. There are two rules:

1. Instead of the traditional RTN instruction, use a JMP to the location directly before the subroutine entry point.
2. In the location directly before the subroutine entry point, place a JST SUBX:.

Use of these two rules will allow an orderly exit from the common subroutine. In our previous example, subroutine CUP looks like this:

```

                NAM      CUP
                EXTR     SUBX:
                JST      SUBX:
CUP             ENT      ENTRY TO COMMON ROUTINE CUP
                JMP      CUP-1      EXIT COMMON ROUTINE
    
```

When SUBR: and SUBX: are used, all subsequent calls to the common subroutine are "locked out" until the current call to the subroutine has completed and the jump to SUBX: has been made. Then, each subsequent call (made while the common subroutine was busy) is completed in priority order.

If this procedure is not followed, the system behavior will appear to be very erratic. Although the system will probably correct itself, when the doubly-returned task finally terminates, one activity has been lost, and one has been duplicated, probably incorrectly. If the user understands this section thoroughly, he can have the convenience of library subroutines, without the difficulty of accidental re-entry.



3. CALL A COMMON SUBROUTINE (SUBR:)

This subroutine is called by a user task to schedule a subroutine which may be used by more than one task.

NOTE

This subroutine does not return directly to the calling program. It exits through the Scheduler (SCHED:).

Calling Sequence:

JST	SUBR:
DATA	(*) ADDRESS OF COMMON SUBROUTINE

Enters Subroutine With:

INTERRUPTS---ENABLED
 STATUS---UNCHANGED
 A REGISTER---UNCHANGED
 X REGISTER---UNCHANGED

NOTE

The return address put in the entry point of the common subroutine is the location following the data in the above call. That is, it appears to the subroutine as if it were called from the location of its address (Not the location of the "JST SUBR:").

3.8. EXIT FROM COMMON SUBROUTINE (SUBX:)

This subroutine is called from within a common subroutine to return to the calling task.

NOTE

This subroutine does not return directly to the calling program. It exits through the Scheduler (SCHED:).

Calling Sequence

	JST	SUBX:
SUB	ENT	

where: SUB is the entry point of the common subroutine. This call must immediately precede the entry so that RTX can keep its chains straight.

	JMP	SUB-1	RETURN
--	-----	-------	--------

Returns to calling task with:

INTERRUPTS--ENABLED
 STATUS---UNCHANGED
 A REGISTER---UNCHANGED
 X REGISTER---UNCHANGED

NOTE

Each SUBR: call made must have a corresponding call made to SUBX: once the routine has completed. If a call to END: (to terminate the calling task) is made from within a subroutine called by SUBR:, all other tasks will be permanently denied the user of that routine. To terminate a task from within a SUBR'd subroutine, the ABORT: routine should be used.

3.9 PROTECT A FACILITY (PROT:)

PROT: is called by a user's subroutine to protect itself from usage by other tasks. It is in a way similar to SUBR: in that reentrance to a common subroutine is prevented during its usage; however, in SUBR:, the determination to protect the subroutine is made by the calling program, while in PROT:, the determination is made by the subroutine itself.

Calling sequence:

	DATA	0
SUB	ENT	
	JST	PROT:
	DATA	\$-3

The call to PROT: must be the first instruction following the entry point. The temp cell SUB-1 is used by PROT: to store the contents of SUB (the return address from the caller). Note that exiting from the routine SUB must be done via the return address in SUB-1, not the address in SUB.

Returns with:

INTERRUPTS---ENABLED
 STATUS---UNCHANGED
 A-REGISTER---UNCHANGED
 X-REGISTER---UNCHANGED

PROT: may be called more than once using the same coordination number by the same task. However, a different task is effectively locked out of the subroutine until it is released by executing a call to UNPR:.



NOTE

The INTR: and INTQ: subroutines contain logic to preclude task-switching caused by an interrupt occurring immediately before a JST LOCK: or JST PROT: instruction. This involves checking the interrupted instruction to see if it is a JST LOCK: or JST PROT:. This check is effective only if the instruction is a JST indirect through a base page pointer to LOCK: or PROT:; that is, an :F9xx instruction. To insure this protection feature, reference LOCK: or PROT: by means of an EXTR directive, rather than a REF directive. This also implies that if EXTR directives are used in conjunction with the LPOOL directive, then an EXTR LOCK: or EXTR PROT: must be accompanied by a SPAD LOCK: or SPAD PROT: directive to insure that the pointer remains in the base page.

3. RELEASE A PROTECTED FACILITY (UNPR:)

UNPR: is called by a common subroutine to delete its protected condition caused by a previous call to PROT:

Calling Sequence:

JST	UNPR:
DATA	Coordination Number

Returns with:

INTERRUPTS---ENABLED
 STATUS---UNCHANGED
 A-REGISTER---UNCHANGED
 X-REGISTER---UNCHANGED

In effect, RTX treats the address of a common subroutine (as used in SUBR: and SUBX:) as a coordination number. These are shared with the coordination numbers used by PROT: and UNPR:. That is, the list in which the common subroutine addresses are saved for SUBR: is the same list that saves the coordination numbers for PROT: and LOCK:. Results will be unpredictable (and probably disastrous) if the coordination number used by PROT:, UNPR:, LOCK: or UNLK: is also the address of a common subroutine (called by SUBR:).

Because RTX maintains a single list for PROT: and LOCK: coordination numbers and SUBR: common subroutine addresses, an alternative method for writing common subroutines exists. The rules for this type of common subroutine are:

1. Instead of the standard "RTN SUB" instruction, use a "JMP SUB-2".
2. In the 2 locations directly before the subroutine entry point, place:

JST	SUBX:
RES	1



3. In the two locations immediately following the subroutine entry point, place:

```

JST      PROT:
DATA     SUB-1

```

4. Because PROT: moves the return address from SUB to SUB-1, references to parameters must be made through SUB-1, rather than SUB. For example, a typical routine, that adds the arguments presented to it and returns the sum in the A register, would normally be coded as follows:

Calling Sequence:

```

                                JST      ADDM
                                DATA     3
                                DATA     4

ADDM                               ENT
                                LDA      *ADDM
                                IMS      ADDM
                                ADD      *ADDM
                                IMS      ADDM
                                RTN      ADDM

```

NOTE

This may not be used as a common subroutine because it has no protection from re-entrance.

Using the SUBR: common subroutine feature, the routine would appear as follows:

Calling Sequence:

```

                                JST      SUBR:
                                DATA     ADDM
                                DATA     3
                                DATA     4

ADDM                               JST      SUBX:
                                ENT
                                LDA      *ADDM
                                IMS      ADDM
                                ADD      *ADDM
                                IMS      ADDM
                                JMP      ADDM-1

```




The alternative method, using the PROT: common subroutine feature, is as follows:

Calling Sequence:

	JST	ADDM
	DATA	3
	DATA	4
	JST	SUBX:
	RES	1
ADDM	ENT	
	JST	PROT:
	DATA	ADDM-1
	LDA	*ADDM-1
	IMS	ADDM-1
	ADD	*ADDM-1
	IMS	ADDM-1
	JMP	ADDM-2

The advantages of the last example, using the PROT:/SUBX: sequence, are:

1. The calling sequence is shorter than that calling SUBR: (the standard JST SUB is used).
2. The burden for insuring that the subroutine is common (re-entrance protected) lies solely with the subroutine writer, not the subroutine caller.
3. If the subroutine is capable of stacking multiple return addresses (not shown in this example), the subroutine is then recursive, and may call itself. (Note that if recursive, SUBX: should only be called on the last return (use RTN SUB-1 for all returns but the last)).

3.11 LOCK OUT A FACILITY (LOCK:)

LOCK: was designed for use by Real Time FORTRAN, and is similar to PROT:. The only difference between them is that the return address from the subroutine is stored in the location following the coordination number, instead of the location in front of the entry point, e.g.:

Calling Sequence:

SUB	ENT	
	JST	LOCK:
	DATA	Coordination Number
	DATA	0 (Return address stored here)

Returns With:

```

INTERRUPTS---ENABLED
STATUS---UNCHANGED
A-REGISTER---UNCHANGED
X-REGISTER---UNCHANGED

```

The JST to LOCK: does not need to be placed immediately following the subroutine entry point, although JST to PROT: does.

The user should reference the LOCK: or PROT: subroutine with an EXTR directive, rather than a REF directive. See the note in the PROT: description regarding this.

Note that the PROT:/SUBX: example shown above does not apply to LOCK:.

3.12 UNLOCK A LOCKED FACILITY (UNLK:)

UNLK: is similar to UNPR:. However, UNLK: permits the common subroutine to complete processing, then returns control to the calling task, while UNPR: returns through the Scheduler to the Ready list for the next task on the list.

3.13 ABORT A TASK (ABORT:)

ABORT: is called from within a common subroutine to terminate the task which called the subroutine.

In addition to performing the END: function, ABORT: also deletes any PROT:, LOCK: or SUBR: conditions previously set by the aborted task.

Calling Sequence:

JST ABORT:

ABORT: exits to the scheduler (SCHED:).

NOTE

The duration of an ABORT: call is significantly longer than an END: call, and therefore it should be called only if in a common subroutine, or in a PROTECTED or LOCKED condition.

3.14 OBTAIN CURRENT PRIORITY (GETPR:)

Calling Sequence:

JST GETPR:

Returns With:

INTERRUPTS---ENABLED
STATUS---UNCHANGED
A REGISTER CONTAINS TASK PRIORITY
X REGISTER---UNCHANGED

The subroutine is called to get the current priority of a task. It is usually called so that a task's priority may be restored after it is temporarily altered.



3. SET TASK PRIORITY (SETPR:)

Calling Sequence:

```
LDA      DESIRED PRIORITY
JST      SETPR:
```

Returns With:

```
INTERRUPTS---ENABLED
STATUS---OV RESET, WORD MODE
A REGISTER---UNCHANGED
X REGISTER---UNCHANGED
```

This subroutine is called whenever a task desires to alter its priority.

3.16 INCREMENT TASK PRIORITY (INCPR:)

Calling Sequence:

```
JST      INCPR:
```

Returns With:

```
INTERRUPTS---ENABLED
STATUS---UNCHANGED
A REGISTER---UNCHANGED
X REGISTER---UNCHANGED
```

This subroutine will increment the priority of the calling task by 1. No range checking is performed.

3.17 DECREMENT TASK PRIORITY (DECPR:)

Calling Sequence:

```
JST      DECPR:
```

Returns With:

```
INTERRUPTS---ENABLED
STATUS---UNCHANGED
A REGISTER---UNCHANGED
X REGISTER---UNCHANGED
```

This subroutine will decrement the calling task's priority by 1. No range checking is performed.



SECTION 4

ADDITIONAL RTX FEATURES

4.1 RTX DEBUG FEATURE (ZBG)

The standard CAI DEBUG program is included in the RTX library tape (Segment 1) under the name ZBG. (Detailed descriptions of DEBUG are included in LSI-2 AutoMagic, CA document 96045-00, or LSI-3/05 AutoMagic, CA document 93001-00). When this module is linked, Relocation Register RF points to the RTX Linked list pointers for use with Z function; the corresponding length required by the Z function is set to five words, which is the length of each block used in the RTX Linked lists. When displaying a particular list with the Z function, the first printed line is not an entry in the list, but simply the pointer to the top of the list, followed by the next four higher words in memory; this first line may therefore be ignored.

There are eight lists maintained by RTX, and the pointers to the top of each of these lists reside within the RTX nucleus in eight consecutive memory locations, in the following order:

0RF	Pointer to the list of tasks awaiting execution (READY)
1RF	Pointer to the list of INTQ: and INTAC: tasks awaiting execution (FIFO)
2RF	Pointer to the list of tasks currently awaiting completion of a DELAY (DLYCH)
3RF	Pointer to the list of common subroutines currently requested (COMN)
4RF	Pointer to the list of tasks currently awaiting I/O execution (IOCH)
5RF	Pointer to the list of tasks awaiting a PUT: response to a requested GET: (GETCH)
6RF	Pointer to the list of PUT: requests awaiting a GET: response (PUTCH)
7RF	Pointer to the list of currently unused blocks (FREE)

The following is a description of the contents and manipulation of a user block within each of the lists:

1. READY List (RF) Ready to Run (used by BEGIN:)

RTX maintains a list of all tasks which are ready to execute in the READY list. This list is sorted into priority order, so that RTX simply executes the task at the top of the list. The format for a READY block is as follows:

<u>Word</u>	<u>Contents</u>
0	Word address pointer to next block entry in the list. (The last element in the list contains a zero).
1	Bits 15-3. Task priority number. Bits 2-0. (LSI-2 only) Bit 2. EIN indicator, for reference only. (RTX always allows interrupts.) Bit 1. BYTE mode indicator upon next resumption of task. Bit 0. OVERflow indicator upon next resumption of task.



<u>Word</u>	<u>Contents</u>
	Bits 2-0. (LSI-3/05 only)
	Bit 2. BYTE mode indicator upon next resumption of task.
	Bit 1. OVerflow indicator upon next resumption of task.
	Bit 0. Unused
2	P register contents upon next resumption of task.
3	A register contents upon next resumption of task.
4	X register contents upon next resumption of task.

2. FIFO list (1RF) Ready to Run (used by INTAC: and INTQ:)

In order to avoid the problems of interrupting a linked list processor, INTQ; and INTAC: put the entries for their tasks in the FIFO list. (BEGIN: operates directly on the READY list). The RTX scheduler (which is never run as an interrupt routine) empties the FIFO list into the READY list and sorts the READY list. The format of a FIFO block is the same as a READY block.

3. DLYCH List (2RF) Delay (used by DELAY:)

A call to DELAY: (with a unique positive coordination number) causes the block for the currently executing task to be deleted from the READY list and put on top of the DLYCH list. The format of a DLYCH block is as follows:

<u>Word</u>	<u>Contents</u>
0	Word address pointer to next block in the list.
1	Status & Priority. Same as READY list entry.
2	The P register. Points to address of return from DELAY:
3	The coordination number.
4	Working number of ticks left in Delay.

Upon return, the A register will contain the coordination number. The X register will contain the number of Real Time Clock "ticks" remaining (normally zero).

4. COMN List (3RF) Common Subroutine (used by SUBR:, SUBX:, LOCK:, UNLK:, PROT:, UNPR:)

A call to SUBR:, LOCK: or PROT: causes the COMN list to be searched for a block for the common subroutine. If none is found, a block is deleted from the FREE list and put on top of the COMN list. The format for a COMN block is as follows:

<u>Word</u>	<u>Contents</u>
0	Pointer to the next block in the list
1	Busy flag (zero = not busy)
2	Pointer to the block of the highest priority task waiting to use the common subroutine (0 = no task waiting)
3	Address of the common subtoutine (or coordination number)
4	Unused

If SUBR: is called and a block for the common subroutine is found with the Busy flag set, the block for the currently executing task is deleted from the READY list, and inserted into a secondary list pointed to by Word 2 above. At the same time, the P register is set so that the task will again call SUBR: when RTX next executes the task.



5. IOCH List (4RF) I/O Suspend (used by IOX:, Fortran Interface)

A call to IO: or IOWAT: when the busy flag is set in the IOB, or a Fortran call for I/O when no parameter block is currently available, will cause the task block to be deleted from the READY list and put on the top of the IOCH list. The P register is set so the task will repeat the call when RTX next executes the task. The format of an IOCH block is the same as for a READY block. The IOCH list is emptied into the READY list each time any I/O completes.

6. GETCH List (5RF) Get (used by GET:)

A call to GET: with a unique positive coordination number (and no matching PUT: yet) causes the block for the currently executing task to be deleted from the READY list and put on top of the GETCH list.

<u>Word</u>	<u>Contents</u>
0	Pointer to next block in the list
1	Status & Priority (same as Ready)
2	P register. Points to return from GET:
3	Coordination No.
4	Unused

When the associated PUT: is done, the block is deleted from the GETCH list, the A and X register contents are stored into words 3 and 4, and the block is inserted into the READY list in priority order.

7. PUTCH List (6RF) Put (used by PUT:)

A call to PUT with a unique positive coordination number (and no waiting GET:) causes a block to be deleted from the FREE list (see below) and added to the top of the PUTCH list. The format for a PUTCH block is as follows:

<u>Word</u>	<u>Contents</u>
0	Pointer to next block in the list
1	Unused
2	A register contents to be passed
3	Coordination No.
4	X register contents to be passed

When the associated GET is processed, the block is deleted from the PUTCH list and put on top of the FREE list.

8. FREE List (7RF) Available Storage

This list is initialized to contain the entire work space during a call to RTX:. As blocks are required, they are taken from the top of the FREE list. As blocks are no longer required, they are deleted from the appropriate list and put onto the tail of the FREE list. A FREE block has no specific format. It will simply contain data from the function which last used the block.

4.2 PROGRAM LOADING WITH ZBG

ZBG resides in the RTX library; to make use of ZBG, it is necessary to include a

ZBG REF

instruction within the user's program. Thus ZBG is entered immediately upon execution, and may then be used to breakpoint through the mainline sequence and any particular task.

4.3 POWER-FAIL, AUTO-RESTART (PWRFL:)

If the computer being used has the Power Fail option, the user may utilize the RTX program module which provides service for that device. The loader will cause the routine to be loaded if the user has a REF to PWRFL:. He must, however, not actually call that program at execution time. Instead, if a power failure begins, the interrupt hardware will force control into that routine, saving the computer's register status, and halt, to prevent loss of information from core storage. When the power is restored, the program will schedule a user-supplied routine, which must be named PWRUP:, and must occur in a NAM directive. Re-initiation of the activity which was in process (at the time of the power failure) will also be scheduled and control will be passed to the system Scheduler.

RTX will schedule PWRUP: as a task at priority 8184 with the contents of the A register nonzero if the power failure was detected. If power failure was not detected (e.g., the computer was halted), RTX will transfer control to PWRUP: with the contents of the A register equal to zero. Note that RTX cannot resume the activity in progress at the time of the power failure if the power failure was not detected.

4.4 TELETYPE INPUT/OUTPUT

RTX provides decimal, octal, and hexadecimal I/O on the standard Teletype, by using a software interface to CAI's Teletype Utility Package (TUP). The calls and usage are identical to the standard version.

TUP also provides the capability to read and print strings of text, (for headings, labels, etc.), and this capability is retained in the RTX version.

Refer to the standard TUP documentation (#96014) for a complete description of each routine. Additionally, a specific limitation exists with respect to TUP usage through RTX:. TUP must not be called concurrently by more than one task, because TUP itself calls subroutines within it with JST instructions, and these subroutines are not protected from re-entrance.

TUP resides on the RTX Segment 2 library tape, and its routines should be referenced with the REF or EXTR directive.

4.5 LSI-3/05 SOFTWARE CONSOLE ROUTINE (CNSOL3)

The LSI-3/05 version of RTX includes CNSOL3, the Software Console Routine, which may be linked by a reference to CNSOL3 in the user program module. Usage of the Software Console Routine is described in the LSI-3/05 Software Manual (90-20010-00).

SECTION 5

RTX OPERATING PROCEDURES

1. Assemble each of your application program modules. Be sure to reference each RTX function that a module uses in either an EXTR or a REF directive.
2. When you have a useful object tape for each of your modules, you are ready to create the executable application program. This requires that you first load LAMBDA, the relocating, linking loader.

Using LAMBDA, force load the initializer task module of your application.

Then using LAMBDA, load the remainder of your group of application program modules. You can use the Selective Load feature of LAMBDA to include only the modules your program actually requires.

5. Still using LAMBDA, selectively load the RTX Library object modules from the two RTX Library Tapes (70-93300-01 and 70-93300-02).

NOTE

If the user program does not reference PROT: and LOCK:, LAMBDA and OS:LNK will declare these subroutines as undefined. This declaration can be ignored since INTRS: and INTQ: (loaded after PROT: and LOCK:) check to see if a call to either subroutine is the next instruction after an interrupt is serviced.

NOTE

When operating under the IOX File Manager, disk devices must be labeled prior to their use. Labeling is done with the stand-alone program, RTX File Label Utility (tape Nos. 70-93324-40A1 and -41A1). Subsection II/5.3 gives a complete description of this utility.

6. Start execution of your program so that the initializer module (Mainline Sequence) or ZBG, if used, is executed first.

Section 6

A SAMPLE RTX PROGRAM - RTX DEMO

6.1 PROGRAM DESCRIPTION

The RTX Demo Program (00-93300-13) demonstrates the basic functions of RTX in a simple, straightforward manner. It consists of three main tasks (TASK1, TASK2, TASK3). The function of each of these tasks is to delay a specific amount of time, and then call a routine to output a message to the teletype. The message consists of the task name followed by the elapsed time in seconds since the start of the program.

An actual user's application of RTX might very well use the interrupt from some external device to initiate a task. This example simulates the effect of three such devices which interrupt every 5, 7, and 11 seconds, respectively; that is, the delays themselves simulate external devices.

Each task delays a different amount of time than the other tasks, before printing.

```
TASK1 delay: 5 seconds
TASK2 delay: 7 seconds
TASK3 delay: 11 seconds
```

Thus TASK1 will output

```
"TASK1 0005"
"TASK1 0010"
"TASK1 0015"
etc.
```

TASK2 will output

```
"TASK2 0007"
"TASK2 0014"
"TASK2 0021"
etc.
```

And TASK3 will output

```
"TASK3 0011"
"TASK3 0022"
"TASK3 0033"
etc.
```

Because of teletype timing, each message takes more than one second to complete. Thus the three tasks will contend with each other for the use of the teletype.



In addition; a fourth task called "IOTASK" outputs the actual teletype messages. This task is begun by each of the three main tasks whenever their delays expire, at the following various priorities:

- TASK1 begins IOTASK at priority 5
- TASK2 begins IOTASK at priority 7
- TASK3 begins IOTASK at priority 11

This means that if TASK1 and TASK3 both begin IOTASK at the same time (which they will, at 55 seconds), TASK3's message will be output first, since its priority to begin IOTASK is higher than TASK1's.

To be more specific, and to demonstrate the priority sequence more fully, the actual teletype output after 55 seconds appears as:

TASK3 0055, TASK2 0056, TASK1 0055,...because each message takes slightly more than one second to print, thus causing the following sequence:

<u>TIME</u>	<u>ACTION</u>
55 seconds after start	TASK1 and TASK3 both begin IOTASK with a "55 seconds" message. Since TASK3 has the higher priority, its message is printed first.
56 seconds after start	TASK2 begins IOTASK with a "56 seconds" message. TASK3's "55 seconds" message is still printing, and TASK1's "55 seconds message" is queued up. Since TASK2 has a higher priority than TASK1, the TASK2 "56 seconds" message gets output when TASK3's message completes.
57+ seconds after start	TASK1's "55 seconds" message is output after TASK2's "56 seconds" message is completed.

After 80 seconds, the teletype listing should appear as:

```
TASK1 0005, TASK2 0007, TASK1 0010, TASK3 0011
TASK2 0014, TASK1 0015, TASK1 0020, TASK2 0021, TASK3 0022
TASK1 0025, TASK2 0028, TASK1 0030, TASK3 0033
TASK2 0035, TASK1 0035, TASK1 0040, TASK2 0042, TASK3 0044
TASK1 0045, TASK2 0049, TASK1 0050, TASK3 0055
TASK2 0056, TASK1 0055, TASK1 0060, TASK2 0063, TASK1, 0065, TASK3 0066
TASK2 0070, TASK1 0070, TASK1 0075, TASK3 0077
TASK2 0077, TASK1 0080,
```

(TASK3's message contains carriage return and line feed control characters).

6.2 PROGRAM MODULE FUNCTIONS

Let us now examine the RTX functions used in this program (refer to the flowchart in figure 6-1 and the program listing at the end of this section). There are six basic modules comprising the program:

```
BEGIN      TASK3
TASK1      IOTASK
TASK2      ADD1
```



6.2.1 BEGIN (Initialize and Begin Tasks)

The program start occurs at the BEGIN section of the flowchart. The first step is to initialize RTX. This is performed using the RTX: function to define the maximum number of RTX tasks which may be in concurrent operation and the required table space for RTX management of those tasks. If insufficient table space is found or other peculiarities occur during initialization, the error return is taken. In our example, we halt the computer to remedy the problem. Using the BEGIN: function of RTX defines the task name (TASK1, TASK2 and TASK3 in our example) and its software priority number (100 for each in our example).

No other tasks have begun their activity at this point. This is because the first task following the RTX: call (the initialization sequence itself) is automatically scheduled at the highest software priority. When the END: function is called, this task is deleted and the Scheduler can then schedule the other tasks in relation to their priority.

Since the three tasks all have priority 100 and priority 100 is the highest active priority value, the Scheduler will arrange each task in sequence according to the order in which it was initiated by the BEGIN: call, and will then start execution of the first task in that sequence. The sequence is determined by a first-in, first-out rule. Therefore, TASK1 executes until it requests an RTX service which causes it to be suspended.

When the task is re-scheduled (on completion of one of the above function calls), it is put back in sequence at the end of all other equal priority tasks.

This type of organization allows for true priority scheduling within an application, while also allowing the tasks themselves to be executed, interrupted, and resumed in an orderly fashion.

6.2.2 TASK1 (Delay 5 seconds, Then Output Name and Elapsed Time)

When TASK1 is begun, it first performs a five second delay. This is done by a call to DELAY: with parameters of 500 (number of 1/10 millisecond real time clock "ticks" to delay) and 1 (a specific coordination number for this particular task's delay calls). The coordination number is necessary mainly for identifying a delay to be changed or deleted; however, it is also required when beginning a new delay, as in this example. When the delay is completed, control is returned to TASK1, which then calls the subroutine ADD1, which increments the elapsed time in the TASK1 message by five seconds. Note that ADD1 is called via SUBR:, because it is a common subroutine used by all three tasks, and is not re-entrant; thus SUBR: prevents another task from entering ADD1 until this call is completed.

Upon return from ADD1, the message is ready for output to the teletype. This is done by a call to BEGIN: to initialize the common task called "IOTASK," which in turn makes the actual call to the I/O executive (IOX) to perform the output. Note that "IOTASK" is a task, not a subroutine; this means that TASK1 may now continue with its next 5-second delay while the I/O is in progress rather than upon its completion, which would invalidate the elapsed time count. Also, the initiation of the common task is made with a priority of 5. IOTASK is also initiated by TASK2 and TASK3, with priorities of 7 and 11 respectively, so that a predictable ordering of outputs is achieved when two or three tasks are vying for the teletype at the same time.



6.2.3 TASK2 (Delay 7 Seconds, Then Output Name and Elapsed Time)

TASK2 is identical to TASK1 in its logical functioning. The only difference between them is in the parameters passed in their calls to DELAY:, ADD1, and IOTASK. TASK2 calls DELAY: with a 7 second count and a coordination number of 2 (to differentiate it from TASK1's delay call). The common subroutine ADD1 is called to increment the elapsed time by seven instead of five, and the common task IOTASK is begun at a higher priority (7).

6.2.4 TASK3 (Delay 11 Seconds, Then Output Name and Elapsed Time)

TASK3 is similar to TASK1 and TASK2. TASK3 calls DELAY: with an 11 second count and a coordination number of 3. It calls ADD1 to increment the count by eleven, and begins IOTASK at priority 11.

6.2.5 IOTASK (Call IOX To Output A Message On The Teletype)

IOTASK is a common task begun as a task by BEGIN: calls in TASK1, TASK2 and TASK3. Upon entry, the X register contains an address pointer to the I/O Information Block (IOB) of the calling task. A call is then made to the IOX package (at its entry point named IO:) passing the IOB address as a parameter. An error status from the I/O operation will cause the computer to halt. Otherwise, the task terminates itself with a call to END:.

6.2.6 ADD1 (Common Subroutine To Increment The Elapsed Time for Printing)

ADD1 is a common subroutine called by TASK1, TASK2 and TASK3 prior to printing their messages. Upon entry, the A register contains the amount by which to increment the elapsed time tally, which is pointed to by an address in the X register. The routine performs the addition, and then returns to the calling task through SUBX:. This is because the subroutine was called via SUBR: to avoid re-entrance.

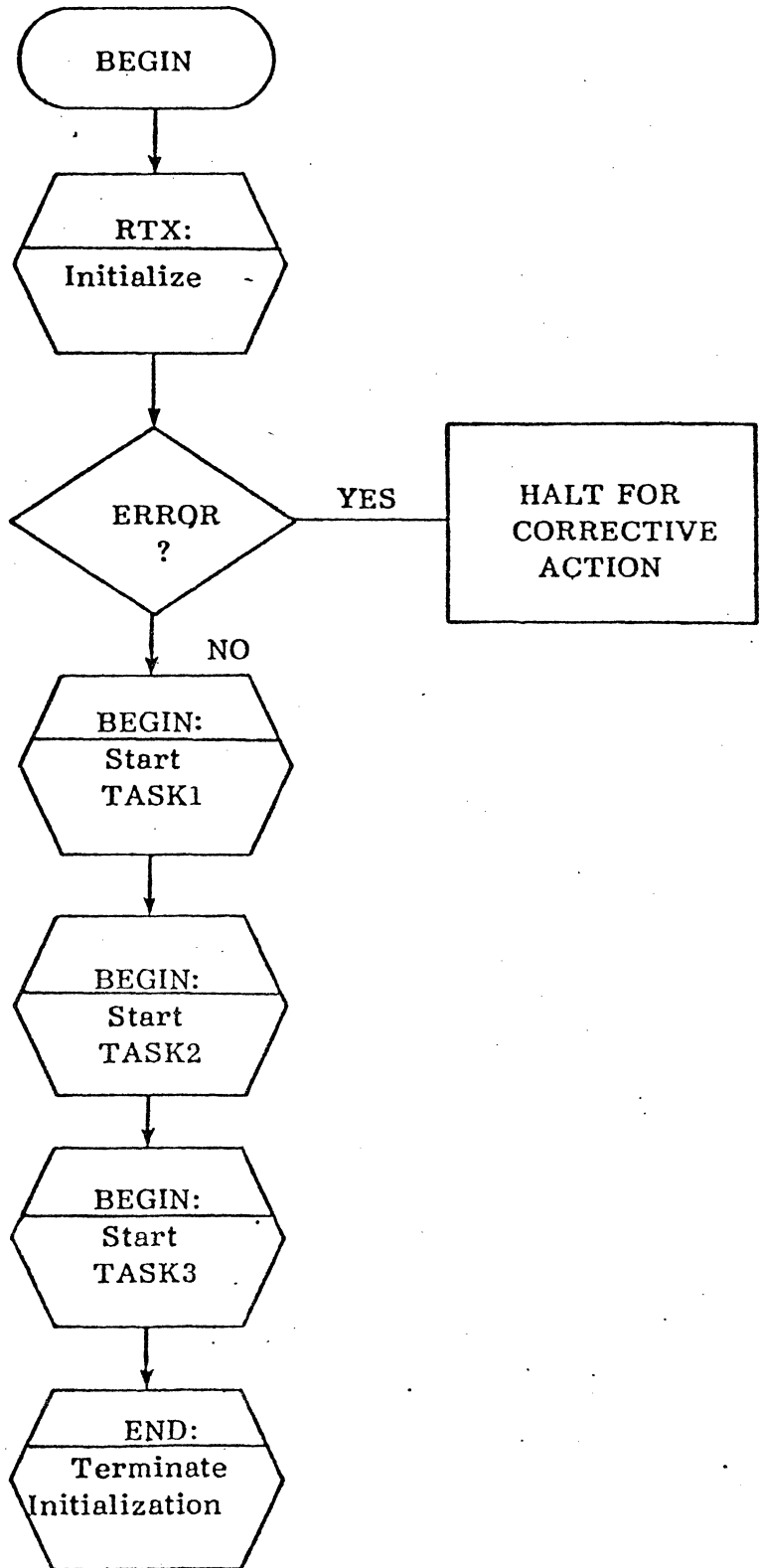


Figure 6-1. RTX Demo Program - Flow Diagram (Sheet 1)

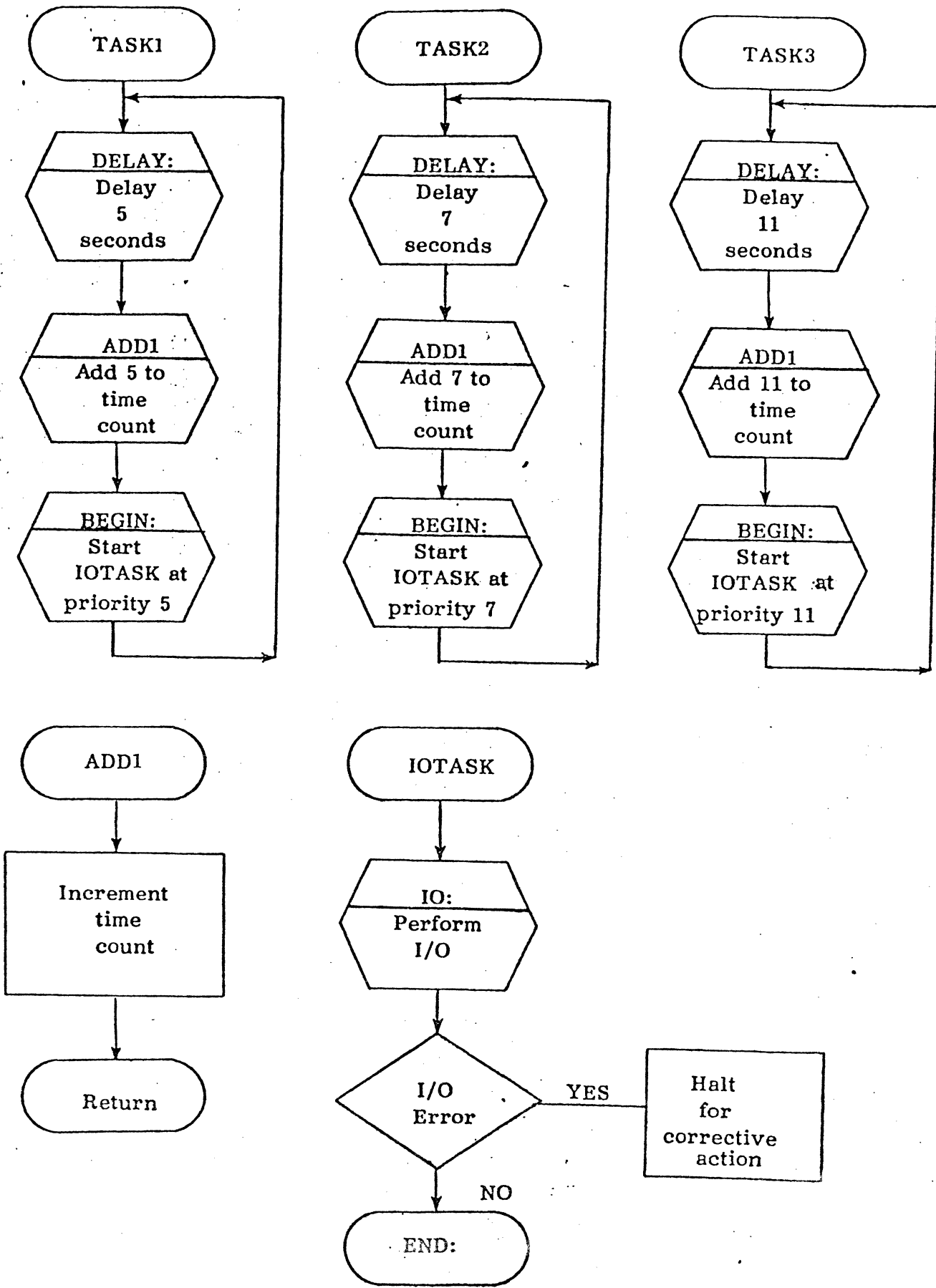


Figure 6-1. RTX Demo Program - Flow Diagram (Cont'd)

```

0002      0000      IFF      LSI305
0004      ENDC
0008      * THIS PROGRAM CONCURRENTLY EXECUTES THREE (3)
0009      * TASKS (TASK1, TASK2, & TASK3) WHICH DELAY
0010      * THEMSELVES FOR 5, 7, & 11 SECONDS RESPECTIVELY
0011      * AND THEN TYPE OUT THEIR IDENTIFICATION FOLLOWED
0012      * BY THE NUMBER OF SECONDS THAT HAVE ELAPSED
0013      * SINCE THE PROGRAM WAS STARTED. THE THREE TASKS
0014      * ARE ALL RUNNING AT THE SAME PRIORITY AND CALL
0015      * A COMMON SUBROUTINE TO UPDATE THE NUMBER OF
0016      * SECONDS IN THEIR OUTPUT MESSAGES. A COMMON
0017      * TASK (IOTASK) IS THEN QUEUED TO OUTPUT THE
0018      * APPROPRIATE MESSAGE. THIS TASK IS QUEUED
0019      * AT THREE (3) DIFFERENT PRIORITIES (TASK1=5,
0020      * TASK2=7, & TASK3=11) SO THAT, FOR EQUAL TIMES,
0021      * THE MESSAGES SHOULD APPEAR IN THE FOLLOWING
0022      * ORDER: TASK3, TASK2, TASK1.
0023      *
  
```

```

0024      000D      NAM      PWRUP:
0025      00F5      NAM      I:UAT
0026      EXTR      RTX: , BEGIN: , END:
0027      EXTR      SUBR: , SUBX: , DELAY: , IO:
0028      EXTR      PAUSE:
0029      0000      IFF      LSI305
0030      EXTR      D:TY00
0031      ENDC
0035      0014      NOACTS EQU      20
0036      0000      REL      0
0037      BEGIN EQU      $
0038      0000 8000 B0B0 LDA      ='00'      RESET
0039      0001 9A89 0088 STA      T1      .
0040      0002 9A89 008C STA      T1+1    . ALL
0041      0003 9AA3 00A7 STA      T2      .
0042      0004 9AA3 00AB STA      T2+1    . ELAPSED
0043      0005 9ABD 00C3 STA      T3      .
0044      0006 9ABD 00C4 STA      T3+1    . TIMES
  
```

I/6-7

PAGE 0002 03/30/77 11:25:27
MACRO2 (A2) S1= DEMOS B0=

RTX DEMO PROGRAM # 93300701-1SE.
LSI-2 RTX DEMO #00-93300-1SE1

0045	0007	F900	0000	JST	RTX:	START RTX
0046	0008	0014		DATA	NOACTS	NUMBER OF ACTIVITIES
0047	0009	000E		DATA	WRAREA	RTX BUFFER AREA
0048	000A	0800		HLT		
0049	000B	F266	0072	JMP	START	GO START INITIALIZE ROUTINE
0050	000C			PWRFL: REF		FORCE LOAD THE POWER FAIL
0051				*		ROUTINE
0052	000D	F900	0000	PWRUP: JST	END:	IGNORE POWER UP RESTART TASK
0053	000E	0000		WRAREA RES	NOACTS+NOACTS+NOACTS+NOACTS+NOACTS,0	



PAGE 003 05/30/77 11:25:27 -IX DEMO PROGRAM 93300/01-13E1
 WALKUP (AC) SIE DEMON DIE LSI-2 -IX F40 #00-93300-13E1

```

0055 *
0056 *   INITIALIZE TASKS
0057 *
0058 0072 F900 0000 START JST   BEGIN:
0059 0073 007C      DATA  TASK1   START TASK 1
0060 0074 0064      DATA  100     PRIORITAET VON TASK 1
0061 *
0062 * * * * *
0063 *
0064 0075 F900 0000      JST   BEGIN:
0065 0076 0048      DATA  TASK2   START TASK 2
0066 0077 0064      DATA  100
0067 *
0068 * * * * *
0069 *
0070 0078 F900 0000      JST   BEGIN:
0071 0079 0044      DATA  TASK3
0072 007A 0064      DATA  100
0073 *
0074 * * * * *
0075 *
0076 007B F900 0000      JST   END:   HALT INITIALIZE ROUTINE
  
```

I/6-9

I/6-10

```

0078 *
0079 007C F900 0000 TASK1 JST DELAY: WAIT FOR
0080 007D 01F4 DATA 500,1 5 SECONDS
007E 0001

0081 *
0082 * * * * *
0083 *
0084 007F C705 LAM 5 5 TIMES THROUGH ADD
0085 0080 E000 008B LDX =T1 POINTER TO NUMBER
0086 0081 F900 0000 JST SUBR: CALL COMMON SUBROUTINE
0087 0082 00DA DATA ADD1 TO ADD IN BCD
0088 *
0089 * * * * *
0090 *
0091 0083 E000 008E LDX =10B1 ADDRESS OF 10B1
0092 0084 F900 0000 JST BEGIN: START COMMON TASK
0093 0085 00CF DATA 10TASK,5 TO DO I/O (PRIORITY IS 5)
0086 0005

0094 *
0095 * * * * *
0096 *
0097 0087 F60B 007C JMP TASK1 KEEP GOING
0098 *
0099 * * * * *
0100 *
0101 0088 EQU $
0102 0088 04C1 TEXT 'TASK1 '
0089 03CB
008A 81A0

0103 0088 8080 T1 TEXT '0000'
008C E000

0104 008D ACA0 TEXT ' , '
0105 008E IOB1 EQU $ IOB BLOCK
0106 008E 0000 DATA 0,0,0,0
008F 0000
0090 0000
  
```

PAGE 0005 03/30/77 11:25:27 RIX DEMO PROGRAM 95300/01-13E1
MACRO2 (A2) SIE DEMOS NOE LSI-2 RIX DEMO #00-95300-13E1

0091	0000		
0107	0092	C3CF	DATA 'CO' LUN
0108	0093	0005	DATA 5 FUNC CODE (UNFORMATTED WRT)
0109	0094	000C	DATA 12 MESSAGE BYTE LENGTH
0110	0095	0088	DATA BUFF1 MSG BUFFER
0111	0096	0000	DATA 0,0
	0097	0000	

```

0113 *
0114 0098 F900 0000 TASK2 JST DELAY: WAIT FOR
0115 0099 028C DATA 700,2 7 SECONDS
      009A 0002

0116 *
0117 * * * * *
0118 *
0119 009B C707 LAM 7 7 TIMES THROUGH ADD
0120 009C F000 00A7 LDX =T2 POINTER TO NUMBER
0121 009D F900 0000 JST SUBR: CALL COMMON SUBROUTINE
0122 009E 00DA DATA ADD1 TO ADD IN BED
0123 *
0124 * * * * *
0125 *
0126 009F F000 00AA LDX =JOB2 ADDRESS OF JOB BLOCK
0127 00A0 F900 0000 JST BEGIN: START COMMON TASK
0128 00A1 00CF DATA LOTASK,7 TO DO I/O (PRIORITY IS 7)
      00A2 0007

0129 *
0130 * * * * *
0131 *
0132 00A3 F60B 0098 JMP TASK2 KEEP GOING
0133 *
0134 * * * * *
0135 *
0136 00A4 BUFF2 EQU $
0137 00A4 D4C1 TEXT 'TASK2 '
      00A5 D3C8
      00A6 B2A0
0138 00A7 B0B0 T2 TEXT '0000'
      00A8 B0B0
0139 00A9 ACA0 TEXT ' '
0140 00AA EQU $ JOB BLOCK
0141 00AA 0000 DATA 0,0,0,0
      00AB 0000
      00AC 0000

```

I/6-12

PAGE 0007 03/30/77 11:25:27 RTX DEMO PROGRAM 93300/01-15E1
MACRO2 (A2) SI= DEMO NO= LSI-2 RTX DEMO ADD-93300-15E1

0142	00AD 0000	DATA	'CU'	LUN
0143	00AE C3CF	DATA	5	FUNC CODE (UNFORMATTED WRT)
0144	00B0 000C	DATA	12	MESSAGE BYTE LENGTH
0145	00B1 00A4	DATA	HUFF2	MSG BUFFER
0146	00B2 0000	DATA	0,0	
	00B3 0000			

1/6-13

```

0148
0149 0084 F900 0000 TASKS JST DELAY: WAIT FOR
0150 0085 044C DATA 1100,3 11 SECONDS
      0086 0003

0151
0152 * * * * *
0153 *
0154 0087 C708 LAM 11 11 TIME THROUGH ADD
0155 0088 F900 0003 LDX =T5 POINTER TO NUMBER
0156 0089 F900 0000 JST SUBR: CALL COMMON SUBROUTINE
0157 008A 000A DATA ADD1 10 ADD IN BCD
0158
0159 * * * * *
0160 *
0161 008B E000 0003 LDX =I0B5 ADDRESS OF I0B
0162 008C F900 0000 JST BEGIN: START COMMON TASK
0163 008D 00CF DATA I0TASK,11 TO DO I/O (PRIORITY IS 11)
      008E 000B

0164
0165 * * * * *
0166 *
0167 008F F608 0084 JMP TASKS KEEP GOING
0168
0169 * * * * *
0170 *
0171 00C0 BUFFS EQU $
0172 00C0 04C1 TEXT 'TASKS '
      00C1 03C4
      00C2 03A0
0173 00C3 00B0 IS TEXT '0000'
      00C4 00B0
0174 00C5 I0B5 EQU $ I0B BLOCK
0175 00C5 0000 DATA 0,0,0,0
      00C6 0000
      00C7 0000
      00C8 0000
  
```

I/6-1A

PAGE 0009 03/30/77 11:25:27
MACRO2 (A2) SI= DEMUS BU=

R1X DEMO PROGRAM 93500/01-15E1
LSI-2 R1X DEMO #00-93500-15E1

0176 00C4 C3CF
0177 00CA 0006
0178 00CB 000A
0179 00CC 00C0
0180 00CD 0000
00CE 0000

DATA 'C0' LUN
DATA 6 FUNC CODE (ASCII WRITE)
DATA 10 MESSAGE BYTE LENGTH
DATA BUFF3 MESSAGE BUFFER ADDRESS
DATA 0,0

1/6-15

PAGE 0010 03/30/77 11:25:27 RTX DEMO PROGRAM 93300/01-13E1
MACRO2 (A2) SI= DEMOS NO= LSI-2 RTX DEMO #00-93300-13E1

```
0182 *
0183 * THIS IS THE COMMON TASK "IOTASK" QUEUED
0184 * BY TASK1, TASK2 AND TASK3 SO THAT THEY WILL
0185 * NOT BE DELAYED WAITING FOR THE COMPLETION
0186 * OF THE I/O. NOTE THAT THE IOX PACKAGE (IO:)
0187 * IS USED.
0188 *
0189 IOTASK EQU $ COMMON. TASK ENTRY POINT
0190 00CF 6803 SIN 2 AVOID INTERRUPTS HERE
0191 0000 F401 0002 STX IOB STORE IOB ADDRESS INTO CALL
0192 0001 F900 0000 JST IO: CALL IOX
0193 0002 0000 IOB DATA $-$ IOB ADDRESS STORED HERE
0194 0003 F900 0000 JST END: IMMEDIATE RETURN
0195 0004 0000 NOP IGNORE ERROR RETURN
0196 0005 F900 0000 JST END: TERMINATE THE COMMON TASK
```

I/6-16

0198 *
 0199 * THIS IS THE COMMON SUBROUTINE CALLED BY TASK1,
 0200 * TASK2, AND TASK3 TO UPDATE THE NUMBER OF
 0201 * ELAPSED SECONDS IN THE APPROPRIATE OUTPUT
 0202 * MESSAGE. ALL TASKS CALLING THIS SUBROUTINE
 0203 * HAVE THE SAME PRIORITY, SO THE CALL TO PAUSE:
 0204 * WILL CAUSE SOME ATTEMPTS AT RE-ENTRY.
 0205 *

0206	0005	0F00	NEXT	SWM		
0207	0007	0A18		IMS	COUNT	DONE?
0208	0008	F206		JMP	LX	NO, CONTINUE LOOP
0209	0009	F900		JST	SUBX:	YES, RETURN FROM COMMON
0210			*			SUBROUTINE
0211	000A	0800	ADD1	ENT		ENTRY POINT
0212	000B	9A14		STA	COUNT	
0213	000C	1326		LLX	1	BYTE ADDRESS OF NUMBER
0214	000D	C203		AXI	3	ADDRESS OF LEAST
0215			*			SIGNIFICANT DIGIT
0216	000E	EA12		STX	SAVEX	SAVE IT
0217	000F	C704	LX	LAM	4	DO ONLY 4 DIGITS
0218	00F0	9A11		STA	FOUR	
0219	00E1	E20F		LDX	SAVEX	GET ADDRESS OF LSD
0220	00E2	0E00		SBM		
0221	00E3	8400	ADD2	LDAB	00	GET DIGIT
0222	00F4	F900		JST	PAUSE:	ALLOW RE-ENTRY ATTEMPT
0223	00F5	0150		IAR		ADD ONE (1)
0224	00E6	9C00		STAB	00	PUT IT BACK
0225	00E7	C08A		CAI	'9'+1	WAS IT '9'?
0226	00E8	F201		JHP	\$+2	YES, GOTTA DO NEXT DIGIT
0227	00E9	F613		JHP	NEXT	NO, CHECK FOR DONE
0228	00FA	C680		LAP	'0'	CHANGE TO ZERO ('0')
0229	00FB	9C00		STAB	00	PUT IN DIGIT
0230	00FC	00A8		DXR		POINT TO PREVIOUS DIGIT
0231	00ED	0A04		IMS	FOUR	BUMP FOUR DIGIT COUNT
0232	00EE	F60B		JMP	ADD2	DO NEXT DIGIT
0233	00EF	F619		JMP	NEXT	CONTINUE

I/6-17



PAGE 0012 05/50/77 11:25:27
 MACHIN? (A2) 51= DEMON KUE
 COUNT DATA 0
 SAVEX DATA 0
 FDUR DATA 0

DEMO PROGRAM 95500/01-15E1
 (S) - 2 MIX OF MC #00-95300-1 SE1

PAGE 0013 03/30/77 11:25:27
MACRO2 (A?) SIE DEMOS

RLX DEMO PROGRAM 93300/01-13E1
LSI-2 RLX DEMO #00-93300-13E1

0238
0239
0240
0241
0242
0243
0244 00F3
0245 00F3 C3CF
0246 0000
0247 00F4 0000
0248
0252 00F5 FFFC
0253
0254 0000

0000 ERRORS
0000 WARNING

*
* THIS IS THE UNIT ASSIGNMENT TABLE REQUIRED
* BY IOX. THERE IS ONLY ONE ENTRY, SINCE ONLY
* ONE I/O DEVICE (TTY) IS USED IN THE
* PROGRAM.
*
UATTOP EQU \$ TOP OF UAT
 DATA 'CO' LUN
 IFF LSI305
 DATA D:TY00 DIB ADDRESS FOR STD TTY
 ENDC
I:UAT DATA UATTOP-5-2 LENGTH OF UAT
*
 END BEGIN

1/6-19



PART II

THE INPUT/OUTPUT EXECUTIVE (IOX)



SECTION 1

IOX GENERAL DESCRIPTION

IOX is a subsystem of RTX which operates under RTX control, and provides the user with a complete, modular method of input/output device management and support. Application programming is faster since time-consuming input/output programming for standard peripherals and communications devices need no longer be done by the user. Since IOX is open-ended, the user can add capability for virtually any kind of device unique to his application and program it under IOX control. All I/O performed by IOX is interrupt-driven and allows other tasks in the system to execute even though I/O is in progress.

Working in conjunction with IOX is the File Manager that enables the user to communicate with data files by name, independent of the physical medium storing the file. Requests for file access are made through IOX using Logical Units (LUNs).

IOX can perform one operation at a time for each peripheral device. Operations requiring the use of the same device are done in I/O task priority order (i.e., the highest priority request is honored whenever the device is available to be used). Operations performed on different devices are done concurrently. All calls to IOX specify a Logical Unit (LUN) on which to perform the I/O rather than physical units. This feature allows a program to be debugged using one set of I/O assignments and executed using another.

IOX satisfies the following I/O requirements of the system:

1. Selects the proper commands for communicating with external devices.
2. Processes device interrupts in the following manner:
 - a. Saves the status of the currently executing task.
 - b. Determines the task priority of the interrupt. (Must it be serviced immediately or can it wait for the completion of a higher priority task and if so, is the higher priority task ready for execution?)
 - c. Determines whether the task processing the interrupt is a re-entrant task, or that the interrupt may not be serviced until each prior interrupt has been fully processed.
 - d. Determines which of the I/O tasks awaiting execution has the highest priority, then restores the CPU status to the environment of that highest priority and gives control to that task.
 - e. Ensures that no task may access a device while it is controlled (dedicated) by another task.
 - f. Ensures that the interrupt system is not disabled for a period of time which would prevent a high speed device from performing I/O successfully.

1.1 GENERAL DESCRIPTION

Because of the likelihood of having several similar devices attached to the computer (including identical units) in a real time environment, IOX has been designed to make it easy to support several similar devices (differing only by device address) using "shareable" code. IOX requires some space for flags, device addresses, etc. Since the types of flags depend on the device, as well as the interface to which the device is connected (there may be more than one device per controller), IOX maintains flags in two separate locations depending on whether the information is unique to the device or to the controller. In order to utilize the minimum space in memory for these flags and temporary cells, and to facilitate the allocation of these cells, IOX does most of its interfacing by means of tables which define the type of device and interface to which it is connected.

IOX is primarily concerned with four tables:

IOB	Input Output Block
UAT	Unit Assignment Table
DIB	Device Information Block
CIB	Controller Information Block

(These tables are more fully described in section 2 (IOB and UAT) and section 4 (DIB and CIB). The IOB is created by the user (task) and resides within the calling task. It contains the Logical Unit Name or Number (LUN) as well as specifications for the I/O operation to be performed.

The UAT is also created by the user. It is a series of two-word entries, each of which equates the LUN (specified in the IOB) to a specific device.

The DIB and CIB are tables which are used in communication between IOX and a particular handler. IOX contains within it DIB's and CIB's for each standard device. Additionally, the user may create his own tables if he desires; for example, he may reserve an extent on a disk by specifying its boundaries in his own disk DIB, or he may create a DIB and CIB (and a handler) for a non-standard device.

In general, the usage of these tables by IOX is as follows: The user constructs the IOB within his program and calls IOX, giving as the sole argument the address of this IOB. IOX must then transfer control to the handler associated with this request. To do so, it first obtains the logical unit number (LUN) from within the IOB, and compares it to each entry in the UAT until a match is found. The UAT is simply a list of each possible Logical Unit Name/Number (LUN), associated with the address of the DIB which defines the device assigned to that LUN. Thus for each LUN the UAT contains a pointer to the appropriate DIB. In turn, each DIB contains a pointer to the CIB which defines the interface to which the device is connected. Finally, the CIB contains a jump table which points to the particular handlers (procedures) for processing the specific request. Therefore, given an IOB and a UAT, IOX can find the procedure to handle the request made in the IOB.

The following steps are performed during a normal call from the user to IOX:

1. The user calls IOX carrying the word address (may be indirect) of his IOB.
2. IOX examines the status within the IOB. If the IOB is busy (from a previous call to IOX), the calling task is suspended and control is passed to the RTX task scheduler.

- ... message takes more than one second to complete.
- If the IOB is not busy, it is then flagged as busy, and the UAT is searched to find a LUN which matches the LUN in the IOB. If not found, an abnormal return is made to the caller after setting the "Invalid LUN" status bit in the IOB.
4. If a matching UAT entry is found, the correct DIB is located (the DIB is referenced within the UAT entry) and the requested function code is compared to the permissible function code(s) within the DIB. If the requested function code is found to be illegal, an abnormal return is made to the caller after setting the "Error" status bit in the IOB.
 5. IOX next queues the I/O request with any previously pending I/O requests for the requested device according to the priority of the calling task and passes control to its internal I/O scheduling routine.
 6. The scheduling routine then monitors the request queue in each DIB; whenever it becomes physically possible to begin an I/O request (the I/O device is available and no higher priority request is pending), the scheduler calls the appropriate I/O handler routine (driver) according to the handler entry address within the CIB.
 7. In general, the I/O handler routine will set up the required interrupt locations, select the device, and initiate a watchdog timer, and then return control to the I/O scheduler.
 8. The I/O scheduler continues monitoring the I/O request queues and calling the applicable I/O handler routine(s) until each DIB has been examined once. Then the I/O scheduler terminates with a call to END:.
 9. When an end-of-block I/O interrupt occurs, it causes a return to the I/O handler which initiated the I/O operation. The handler will normally at this time, call an end-of-block routine within IOX, which stores the I/O status and record count into the IOB, releases the device from dedication (if desired), returns to the calling task through either the normal or the abnormal return location, depending on the status, and begins the I/O scheduler.
 10. If an I/O error should cause the watchdog timer to expire prior to I/O completion, it causes a return to the applicable handler, which will then normally execute an initialize function to the device, store an "Unresponsive Device" status into the IOB and return to the caller's abnormal return location.

1.2 CALLING SEQUENCES

The three entry points to IOX are:

- IO: To perform an I/O operation or special function
- IOREL: To release a dedicated device
- IOWAT: To wait for completion of an I/O operation



Each of these entries requires a parameter list (IOB). IOB format is described in detail in section 2. The IOB specifies the type and mode of operation, data area, data length, and the Logical Unit Name/Number. It also provides room for status information to be returned to the calling task. All calls to IOX return with the registers as follows:

A Register	Undefined
X Register	Pointing to the IOB
OV Register	Undefined
Word Mode	
LSI Console Data Register	Unchanged

The format of a call to IOX to perform an I/O operation is:

JST	IO:	Call the IOX perform I/O routine
DATA	(*)IOB	Address of the Input/Output Block
---		Immediate Return
---		Operation complete---abnormal return
---		Operation complete---normal return

Note that there are three exits from IO: -- two are always taken. As soon as the first is processed, IOX BEGIN: is a new task whose starting address is the immediate return location. When the I/O operation is completed, IOX returns to either the abnormal or normal return depending on the success of the operation. Having an immediate exit as well as a complete exit from IOX provides the user with the option of concurrently executing his program while the I/O is in progress. If he does not wish to continue execution until the I/O has completed, he simply codes:

```

( .T      END:

```

by the location of the Immediate Return.

Alternatively, if a certain amount of concurrent processing can take place during the I/O operation, the immediate return location should contain a jump to the processing routine. When the intermediate processing has finished, and it is necessary to await I/O completion before continuing, a call to the IOWAT: routine is made, as in the following example:

	JST	IO:	Initiate the I/O operation
	DATA	(*)IOB	IOB address
	JMP	TAG	Immediate return - continue processing
	JST	END:	Ignore complete return
	JST	END:	Ignore complete return
TAG	EQU	\$	
	.		Concurrent processing
	.		during I/O
	.		
	JST	IOWAT:	Wait until I/O completion
	DATA	(*)IOB:	IOB address
	---		Operation complete - abnormal return
	---		Operation complete - normal return

Note that a call to END: must be made at the "complete" returns from the call to IO:, in order to terminate the I/O task. One of these two returns will be made if I/O completes before the call to IOWAT: is executed.



NOTE

A call to IO: is equivalent to a call to BEGIN: (see chapter 1, RTX Functions) with a starting address of the immediate return and a priority of the task which calls IO: except that the new task is queued before all tasks of equal priority.

An abnormal return may result due to the following:

- LUN not in UAT
- Illegal Operation Request
- Device Error
- File Mark Input
- End-of-Device

A normal exit will result from all other conditions.

1.3 DEVICE DEDICATION

If desired, the user may dedicate a device to specific IO: calls only. Word 3 of the IOB provides the capability of establishing a specific (non-zero) coordination number for an I/O call. Once such a call has established the dedication of a device, all future I/O requests for that device will be held off (queued) until the device is released, unless they contain the established coordination number.

A device is released from dedication by a call to the IOREL: subroutine, as follows:

JST	IOREL:
DATA	(*)IOB
---	Return

On return the A register will be zero if the device was released; otherwise, one or more of the following A register bits will be set:

Bit 0 set:	the LUN entry in the IOB could not be found in the UAT.
Bit 1 set:	the IOB contains a coordination number of zero.
Bit 2 set:	the coordination number in the DIB does not match the coordination number in the IOB and no queued IOB has a matching coordination number.

1.4 LOADING

The user is supplied with two standard relocatable object segments, each residing on two separate paper tapes:

Segment 1 (paper tape 70-93300/1-01):

This segment contains the following program modules, in the order shown:

1. Character I/O Drivers
2. Card Reader Drivers
3. Magnetic Tape Drivers
4. Disk Drivers



5. I/O Scheduler
6. RTX Nucleus
7. ZBG
8. CNSOL3 (if LSI-3 version)

Segment 2 (paper tape 70-93300/1-02):

This segment contains, in the following order:

1. IOX Control
2. RTX Services

In addition to these two modules, the user will require:

1. An RTX Mainline sequence, which makes a call to RTX: to initialize the RTX environment, and to BEGIN: for each task he wishes to initiate immediately.
2. One or more "task" programs to be run simultaneously under RTX (See chapter 1, RTX Description)
3. Special device handler program(s) and the associated DIB and CIB tables, for use in communicating with any device(s) for which a standard handler does not currently exist in IOX (see section 3, I/O Handler Organization below). These handler programs are not necessary if using only the standard devices (teletype, CRT, high speed paper tape reader and punch, line printer, card reader, magnetic tape, disk, floppy disk).

NOTE

The user's special DIB's will each contain a CHAN directive to permit chaining to the other DIB's referenced during linking. The user who does not have an OS system will need version D0 or higher of the OMEGA assembler in order to correctly assemble the DIB tables, because lower versions do not recognize the CHAN directive.

4. A Unit Assignment Table module (UAT) containing entries for each I/O unit to be accessed (see section 2, UAT Description).

The user may either load each module using LAMBDA, or produce a binary tape via the OS Link Editor. The order of input of the object modules is as follows:

1. User's main line sequence.
2. User's various tasks.
3. Unit Assignment Table (UAT).
4. Special user-coded DIBs and CIBs, if any.
5. User-coded I/O handlers, if any.
6. RTX/IOX tape, Segment 1.
7. RTX/IOX tape, Segment 2.

The RTX/IOX tapes, Segments 1 and 2, are organized in library format. Each routine on these tapes is loaded conditionally until the last module of the tape is read. The routines are organized so that only one pass through the loader is necessary.

**NOTE**

Fortran tasks to be run under RTX control require additional library modules to be linked. Refer to the Fortran Operations Manual for a complete description.

1.5 RESTARTABILITY

In general, if some I/O error occurs during execution for which the operator wishes to abort the program, it may not be restartable if the abort condition (e.g., the operator halts the processor through the console) occurs during the period of any I/O request (either pending or being serviced). This is because various "busy" flags within the I/O tables must be reset upon restarting the program. To insure resetting of these flags, reference the "IONIT:" module from the Mainline sequence (see chapter 1, section 2: description of the RTX: initialization routine).



SECTION 2

IOB AND UAT ORGANIZATION

The IOB (Input/Output Block) is created by the user and resides within the calling task. It contains the Logical Unit Name or Number (LUN) as well as specifications for the I/O operation to be performed.

The UAT is also created by the user. It is a series of two-word entries, each of which equate the LUN (specified in the IOB) to a specific device.

The following IOB description applies to all standard IOX handlers. The description is annotated to include File Manager functions. IOB organization for non-standard handlers (for example, the IEEE Intelligent Cable Handler) is described in Section 7.

2.1 INPUT/OUTPUT BLOCK (IOB) - 10 WORDS

The IOB must be set up by the user within his own program. Word 0 is temporary storage and will be destroyed by IOX each time IO: is called. Words 1 and 2 are set to the device name by IO:. Words 3-7 are parameters passed by the user on calls to IO:. Words 5 (bits 8-15) and 8 contain information returned to the user from IOX. Word 9 is used only on devices which support direct access I/O (i.e., disk, floppy disk). (Note that IOB tables are not required for Fortran tasks. Refer to the Fortran Operations Manual). Figure 2-1 illustrates the IOB configuration.

Sample IOB's are included in TASK1, TASK2, and TASK3 of the RTX Demo Program. Refer to Chapter 1, Section 6.

- Word 0 Temporary Storage for Use by IOX. This word is used by IOX as a pointer to queue requests for each device. It must NOT be altered by the user.
- Word 1 Device Type (Two ASCII Characters). This word is set by IO:. It contains the two character mnemonic for the device type.
- Word 2 Device Number. This word is set by IO:. By convention it contains two ASCII digits (0-9) and is used to distinguish between multiple devices of the same type.

CAUTION

Words 1 and 2 are used for temporary storage during calls to IO: and are only valid after one of the complete exits has been taken. These locations must not be changed when the busy bit in word 5 is set.

INPUT/OUTPUT BLOCK

Standard Name*

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 word

	CHAIN POINTER (RESERVED FOR USE BY IOX)												0
IDT	DEVICE TYPE												1
ICUN or ITCB	DEVICE NUMBER												2
	COORDINATION NUMBER												3
ILUN	LOGICAL UNIT NAME/NUMBER												4
ISTA, IOP	BUSY	ERROR	NO I/O	BAD LUN	FORM	DEV. UN-RESP	DEVICE POS.	INT. USE	RES.	S O	OP CODE	OP MOD.	5
IRCNT	REQUESTED COUNT												6
IBUFF	BUFFER ADDRESS												7
IACNT	FLAG	ACTUAL COUNT/PROMPT CHARACTERS											8
AAA	DIRECT ACCESS ADDRESS												9

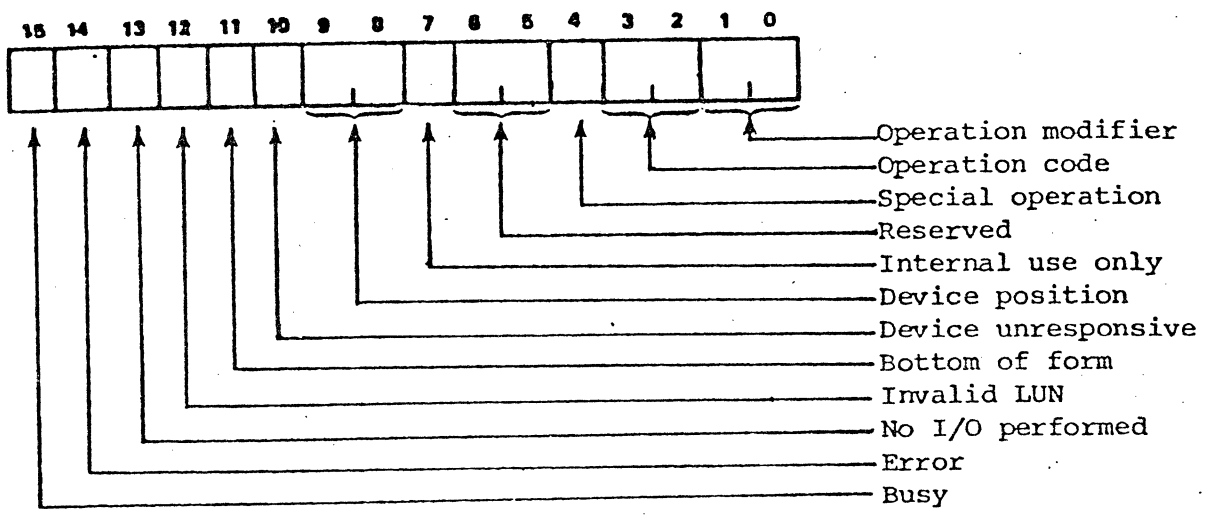
Figure 2-1. IOB Configuration

* refer to the I/O Handler listing at the end of Section 3.

Word 3 I/O Coordination Number. This word is supplied by the user to coordinate his I/O requests. If this word is non-zero, the device on which the call is being made will be dedicated to the coordination number supplied. When a device is dedicated to a specific coordination number, only those requests with matching numbers will be honored. All others will be queued until the device is released. If device dedication is not required, this word should be set to zero.

Word 4 Logical Unit Name/Number (LUN). This word is supplied by the user and it describes the Logical Unit on which the I/O should take place. Although the LUN may be any 16-bit value, by convention all negative numbers are considered to be ASCII character pairs (e.g., SI, LO). All positive numbers are considered to be FORTRAN unit numbers (e.g., 5,6,10).

Word 5 Status, Function Code. This word uses the following format:



Bits 15-8 Status returned to the user by IOX. The breakdown of bits is as follows:

- Bit 15** Busy (the operation has not been completed)
- Bit 14** Error (an unrecoverable error has occurred); or bit 11 or 12 is set for the File Manager.
- Bit 13** No I/O performed (e.g., LUN is assigned to dummy device, device cannot perform the requested operation, LUN not in assignment table, Read or Write with zero (0) count).
- Bit 12** Invalid LUN (LUN cannot be found in Unit Assignment Table); or File Manager access mode error.



- Bit 11 Bottom of form (listing device only); or File Manager end of medium, directory full, directory error, device not labeled or partition busy.
- Bit 10 Device unresponsive (the device has not responded to the request in a reasonable length of time); not used by the File Manager.
- Bits 9 and 8 Position of device:
 00 Indeterminate
 01 Beginning of device
 10 File mark found
 11 End of device (disk and Magnetic tape only). For tape, the EOT reflective marker was encountered. For disk, the last sector in the extent was accessed. This status does not necessarily mean that no data was transferred.
- Bit 7 This bit is for INTERNAL use only. Initialize to zero and do not ALTER.
- Bits 6 and 5 Reserved for future expansion
- Bits 4-0 Requested Function Code. This is supplied by the user and defines the operation to be performed on the device. The breakdown of bits is as follows:
- Bit 4 Special Operation - If this bit is set, bits 3-0 are ignored. This is to allow users to supply drivers for devices which perform special functions.
- Bits 3 and 2 Operation Code
 00 Read
 01 Write
 10 Position
 11 Function
- Bits 1 and 0 Operation Modifier - These bits define the specific type of operation to be performed. Their meaning depends on the operation code. (Some operation modifiers vary for certain Handlers. These differences are noted accordingly.)

For read:	<u>File Manager</u>
00 Direct Access (MTIC only, Read Reverse)	Random Access
01 Unformatted, Sequential	Sequential
10 Formatted ASCII, Sequential	Sequential
11 Formatted Binary, Sequential	Sequential

For write:	<u>File Manager</u>
00 Direct Access	Random Access
01 Unformatted, Sequential	Sequential
10 Formatted ASCII, Sequential	Sequential
11 Formatted Binary, Sequential	Sequential

For position:	<u>File Manager</u>
00 Absolute, Records	No change
01 Absolute, Files	No change
10 Relative, Records	No change
11 Relative, Files	No change
 For function:	 <u>File Manager</u>
00 Write File Mark	No change
01 Punch leader	Reserved
10 MTIC only, Control Edit; Line Printer only, Eject to Top-of-Form	Set file deleted bit in DIB
11 MTIC only, Control Erase	Update directory (New files only)

Word 6

Requested Count. This word is supplied by the user to specify the I/O length, which is defined as follows:
For read or write functions, this word is the number of bytes to be transmitted (1 to 65,535). (If the operation is Write Formatted ASCII, IOX will alter the requested count to remove trailing blanks before calling the handler. This is done with an intermediate counter. IOB Word 6 is not altered.)

For relative record or relative file positioning, this word is the number of records or files to skip. (A positive count means skip forward, a negative count means skip backward).

For absolute record or absolute file positioning, this word is the actual record or file number to skip to. (For MTIC Handlers, the unit is rewound and placed offline if this word is equal to minus one.)
NOTE: Positioning a file to absolute -1 (file marks or records) is a close file operation for the File Manager (refer to Section 5.1.3).

Word 7

Buffer Address. This word is supplied by the user to specify the start address of the I/O buffer. Note that this address is always a word address and that indirect addressing is not allowed.

Word 8

Actual Count/Prompt Characters. This word is returned to the user by the File Manager. It contains the number of records or files actually skipped (for relative position), the actual record or file skipped to (for absolute position), or the actual record length in bytes (for read or write). The File Manager will NOT read more bytes into the user's buffer than requested, but will continue to count characters to establish the physical record length.

On devices which are capable of prompting, this word is used to hold up to two prompt characters.

NOTE

Word 8 contents will be assumed to be prompt characters if negative (bit 15 set). Bits 7-0 not equal to zero indicate two prompt characters; bits 7-0 equal to zero indicate only one prompt character (in bits 15-8).



Word 9

Direct Access Address. This word is the direct access data address within the device (current record number), for devices capable of supporting direct access. For sequential access, this word will be incremented to the current logical record number after each access. For random access, the user stores the logical record number here.

2.2 UNIT ASSIGNMENT TABLE (UAT)

The Unit Assignment Table is not part of the standard IOX library; it must be "tailor-made" by the user for the particular configuration of devices he requires. Figure 2-2 illustrates the UAT configuration.

UNIT ASSIGNMENT TABLE

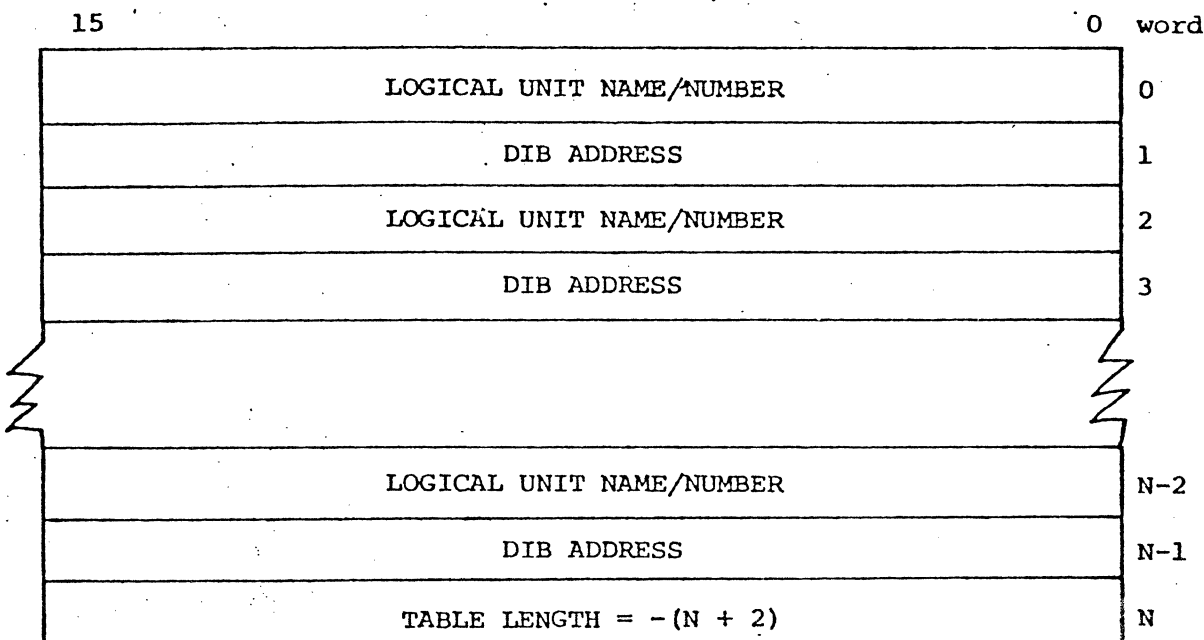


Figure 2-2. UAT Configuration

The UAT is a table of two-word entries for each logical unit which can be referenced in calls to IOX, plus a terminating word containing the UAT word length. The first word of the entry is the Logical Unit Name/Number (LUN) which is referenced in the user's IOB. It may be any value from 0 to 65535.

The second word of the entry is the address of the corresponding DIB table.

The last word in the table is the count word. It is a negative quantity representing the number of words in the table, plus one; that is, two words for each entry, plus the count word itself, plus one. Thus, if there exist four two-word entries, the contents of the count would be minus 10, or $-(4 \times 2 + 1 + 1)$. The count word must be the last word in the table, and must be labeled I:UAT, because this is the name used by IOX when referencing the UAT. (Refer to the sample UAT at the end of this section).



2.3 STANDARD DIB NAMES

The following table shows the DIB names for all devices for which standard and non-standard handlers exist within IOX. The label is to be used as the second word of the UAT entry for each device the user wishes to include.

	<u>Non-DIO</u>	<u>Fortran Non-DIO</u>	<u>DIO</u>	<u>Fortran DIO</u>
Teletype Console	D:TY00	D:TYF0	D:TYOD	D:TYFD
Teletype Keyboard	D:TK00	D:TKF0	D:TKOD	D:TKFD
Teletype Tape Reader	D:TR00	D:TR00	D:TROD	D:TROD
Teletype Punch	D:TP00	D:TP00	D:TPOD	D:TPOD
CRT Console	D:TY00	D:TYF0	D:TVOD	--
CRT Keyboard	D:TK00	D:TKF0	D:TVOD	--
High Speed Paper Tape Reader	D:PRO0	D:PRO0	D:PROD	D:PROD
High Speed Paper Tape Punch	D:PP00	D:PP00	D:PPOD	D:PPOD
Centronics Line Printer	D:LP00	D:LPF0	D:LPOD	D:LPPD
Tally Line Printer	D:LP10	D:LPF1	--	--
Data Products Line Printer	D:LP20	D:LPF2	--	--
Card Reader	D:CR00	D:CR00	D:CR0D	D:CR0D
Disk (43 series, fixed platter), unit 0	D:DK00	D:DKF0	--	--
Disk (43 series, fixed platter), unit 1	D:DK02	D:DKF2	--	--
Disk (43 series, fixed platter), unit 2	D:DK04	D:DKF4	--	--
Disk (43 series, fixed platter), unit 3	D:DK06	D:DKF6	--	--
Disk (43 series, removable platter), unit 0	D:DK01	D:DKF1	--	--
Disk (43 series, removable platter), unit 1	D:DK03	D:DKF3	--	--
Disk (43 series, removable platter), unit 2	D:DK05	D:DKF5	--	--
Disk (43 series, removable platter), unit 3	D:DK07	D:DKF7	--	--
Storage Module Disk, unit 0 (cylinders 0-201)	D:SM00	D:SMF0	--	--
Storage Module Disk, unit 0 (cylinders 202-403)	D:SM01	D:SMF1	--	--
Floppy Disk, unit 0	D:FD00	D:FDF0	--	--
Floppy Disk, unit 1	D:FD01	D:FDF1	--	--
Floppy Disk, unit 2	D:FD02	D:FDF2	--	--
Floppy Disk, unit 3	D:FD03	D:FDF3	--	--
Magnetic Tape, unit 0	D:MT00	D:MT00	D:MC00	--
Magnetic Tape, unit 1	D:MT01	D:MT01	D:MC01	--
Magnetic Tape, unit 2	D:MT02	D:MT02	D:MC02	--
Magnetic Tape, unit 3	D:MT03	D:MT03	D:MC03	--
IEEE Intelligent Cable	--	--	D:IEOD	--

2.4 SAMPLE UAT

When creating the UAT, the user must declare I:UAT in a NAM directive, and any of the Standard DIB names in an EXTR directive, e.g.:

	NAM	I:UAT	
	EXTR	D:CR00, D:LP00, D:TK00, D:LPF0	
UATTOP	DATA	'CR'	Card Reader Entry
	DATA	D:CR00	
	DATA	'LP'	Centronics Line Printer Entry
	DATA	D:LP00	
	DATA	'CI'	Command Input Entry
	DATA	D:TK00	
	DATA	'CO'	Command Output Entry
	DATA	D:TK00	
	DATA	5	FORTRAN Unit 5
	DATA	D:CR00	
	DATA	6	FORTRAN Unit 6
	DATA	D:LPF0	
I UAT	DATA	UATTOP-I:UAT-2	Table Length

SECTION 3

I/O HANDLER ORGANIZATION

The purpose of an I/O handler routine is to set up and execute the actual I/O instructions (normally interrupt-driven Auto-I/O instructions) necessary to perform an input or output operation to a specified device. The I/O operation and the Logical Unit Name/Number are specified in the user's IOB, and the I/O must be performed within the constraints of the device as specified in the CIB and DIB. (These tables are described fully in section 4.) A listing of the Character-oriented I/O handler is included at the end of this section.

3.1 THE STANDARD HANDLERS

Each standard IOX handler is described below. Refer to Section 7 for descriptions of non-standard handlers and to Publication No. 93325-00 for the A/D, D/A Handler.

3.1.1 Character-oriented Device Handler (non-Fortran)

This handler performs I/O, according to specifications within the applicable CIB, for the teletype, high speed reader and punch, and line printer. (A complete listing of this handler is found at the end of this section.)

3.1.2 Fortran List Device Handler

This handler exists for I/O to the teletype console, teletype keyboard and line printer when used as a list output device under Fortran. It differs from the previously described handler in that it recognizes and processes Fortran carriage control characters; i.e., a "1" character as the first print character signifies top-of-form, and a "0" signifies double spacing before printing. (A top-of-form function to the teletype consists of six consecutive line feeds).

Note that the Fortran task does not use an IOB, but rather Fortran I/O statements; these are passed through the Fortran/RTX I/O Interface routine which sets up an internal IOB for the user, according to the DIB's he has included in his Unit Assignment Table. The Fortran I/O handler is entered because the third character of the device name in DIB Words 5 and 6 is an "F"; thus "LPF0" will be processed by the Fortran handler, and "LP00" will be processed by the standard character handler.

3.1.3 Card Reader Handler

The card reader handler is similar to the standard character handler except that input characters are converted to ASCII before returning.

3.1.4 Magnetic Tape Handler

The Magnetic tape handler processes I/O for magnetic tape devices, and will perform read, write, write end-of-file and reposition functions.

3.1.5 Disk and Storage Module Disk Handler (Non-Fortran)

The IOX disk handler allows the RTX user to communicate with the disk. The communication takes place through IOX and the standard calling sequence is used.

The user calls the IOX disk handler by making a standard call to IOX with an IOB which contains a LUN assigned to a disk DIB. The op-code must be either read-direct access or write-direct access.

Data Formats

The IOX disk handler supplies no formatting information of its own. It just reads (or writes) the number of bytes requested by the user. The length of each "record" is unknown (supplied by calling program) and therefore the disk handler is unable to read variable length records without some form of external formatting routines.

The IOX disk handler can support multiple "extents" on each disk and can allow access to them as if each were a separate disk unit. Extents are simply regions on the disk which may be defined by the user to be handled separately. Without any outside action by the user, IOX will process contiguous records throughout the extent. Each record contains the number of bytes requested in the I/O call, and each record starts at the beginning of a sector. Therefore, for fixed length records, each extent may be considered as a sequential file.

In order to allow "direct access", each sector has a "relative sector number". The user may direct the IOX disk handler to process a particular record by initializing IOB Word 9 (IOB Direct Access Address) in the IOB used for the I/O call. At the completion of each request, this address is appropriately incremented by the IOX disk handler so that the next request will process the next record. If the record contains 1-512 bytes, the address will be incremented by one; 513-1024 bytes, the address will be incremented by two; etc. Note that the relative sector number and relative record number may not agree (in fact they will not agree if the records are larger than 512 bytes).

Extents are defined in the disk DIB's. The standard Disk DIB's (DK00 and DK01) define an extent as an entire platter (200 cylinders, 2 heads). The user who wishes to utilize several extents on a single platter may do so by creating his own disk DIB's, using the following variables within each DIB to define the parameters of the desired extent:

- a. The number of sectors per track (may be less than the physical number).
- b. The starting sector number (when added to the number of sectors per track must be less than or equal to the physical number per track).
- c. The number of heads per cylinder (may be less than the physical number).
- d. The starting head number (when added to the number of heads per cylinder must be less than or equal to the physical).



- e. The number of cylinders the extent occupies.
- f. The starting cylinder (when added to the number of cylinders must be equal to or less than the physical).
- g. The drive number.

The IOX disk handler does not check for validity of the resulting sector, head, and cylinder numbers. It assumes that the dimensions and offsets supplied in the DIB are valid. This allows the user to take advantage of the "flag" bits described in the Disk Interface Manual.

Contiguous sectors occur in the following sequence:

- a. Consecutive sectors on a single track (up to the number of sectors per track).
- b. The same sectors on the next head (up to the number of heads per cylinder).
- c. The same sectors and heads on the next cylinder (up to the number of cylinders).

The disk handler requires four additional words (five if under Fortran) in the DIB which are not required for the other handlers. These are DIB words 11-14, (11-15 if under Fortran) and are described in section 4.

3.1.6 Floppy Disk Handler (Non-Fortran)

An "extent" on a floppy disk is constructed as described for the disk handler, taking into account the size limitations in the number of cylinders, heads, and sectors:

Cylinders per Floppy Disk platter = 77 (00-76)
Heads per platter = 1 (single surface)
Sectors per track = 26 (00-25)
Words per sector = 64

There exists within RTX a standard Floppy Disk DIB (D:FD00) whose extent is defined as an entire platter. The user may define his own DIB's as described in the disk handler description.

3.1.7 Disk, Storage Module Disk, and Floppy Disk Handler (Fortran)

Fortran tasks require a certain minimum amount of file management to be performed by the disk handler. The Fortran disk handler differs from the standard disk handler as follows:

- a. The random access address within the IOB is maintained by the Fortran disk handler itself, rather than the user, since the Fortran task does not create its own IOB.
- b. The Fortran disk handler can write and recognize an end-of-file mark. This is a 2-character ASCII record comprised of "/"* characters.

The determination as to whether a Fortran or a non-Fortran disk handler is to be used is made on the basis of the device name in Words 5 and 6 in the DIB. If the third character is an "F", it signifies Fortran, and the Fortran disk handler is used.

In addition, a sixteenth word (Word 15) is required in a Fortran DIB. This word is used for storage of the current relative record number, which would normally be maintained in IOB Word 9. Since the Fortran user does not have access to the IOB, the Fortran/RTX I/O Interface routine keeps this information in the DIB.

1.8 Magnetic Tape Intelligent Cable (MTIC) Handler

The MTIC handler controls data transfers between Pertec or Pertec-compatible formatters and tape transports and the central processor. The handler performs read, write, write filemark, rewind and offline, control edit, control erase, and reposition functions.

3.2 I/O HANDLER REQUIREMENTS

The user may write his own handler routine for any type of I/O device he wishes. The requirements for any I/O handler to be run under control of IOX are as follows:

- 1. Since all I/O under RTX must be done under interrupts, the word and block interrupt locations must be set up prior to I/O.
- 2. A time-out sequence must be included to avoid the possibility of the device "hanging-up" indefinitely without completing its operations. The real time clock, via the RTX DELAY: call is normally used for this purpose.

NOTE

The user must not attempt to manipulate the real time clock by any means other than through the DELAY: call, as this will adversely affect the operation of RTX.

- 3. Once I/O is initiated, the handler should pass control back to the IOX scheduler. This permits other I/O operations to be executed simultaneously if requested.
- 4. The I/O handler should resume control upon either an end-of-block interrupt or upon watchdog time-out, to check the status and return to the caller at either the normal or the abnormal return location.

Several IOX- internal subroutines (described below) currently exist to aid the standard handlers in accomplishing the above requirements. The user-written handler may use any of these routines he wishes. The names of any of these routines must be declared in EXTR or RFF directives within the user's handler.



3.2.1 SINT: (Set up an Instruction at the Word Interrupt Location)

Calling sequence:

```

EXTR      SINT:
.
.
.
LDX      CIB Address
JST      SINT:
DATA     :XXXX

```

Returns with:

```

INTERRUPTS---UNCHANGED
STATUS---UNCHANGED
A-REGISTER---UNDEFINED
X-REGISTER---UNCHANGED

```

where :XXXX represents a constant which is added to CIB Word 1 to form an interrupt instruction:

SINT: does the following:

1. It determines the word interrupt location of the device. This address must reside in CIB Word 21.
2. It calculates and stores an instruction into the word interrupt location. The actual instruction stored is the arithmetic sum of (contents of CIB Word 1) + (:XXXX), where :XXXX may be any positive or negative value.

NOTE

The standard CIB's contain a "SEL DA,7" instruction in word 1.

3. Preparation is then made for a subsequent call by the handler to the SIO: routine (the handler need not call SIO:, however). This preparation consists of transferring the contents of DIB Word 8 into CIB Word 12.

(In the standard DIB's Word 8 will contain various function codes which are required for SELECT instructions in order to initiate an Auto I/O sequence during the SIO: routine. If the specific handler does not call SIO:, DIB Word 8 need not be preset.)

3.2.2 SIO: (Start I/O and Watchdog Timer)

Calling sequence:

```

EXTR      SIO:
.
.
.
LDA      DPTR
LDX      CIB address
JST      SIO:

```




Returns with:

Does not return directly; if the INTP: subroutine is used, a return will ultimately be made in the following state:

INTERRUPTS---ENABLED
 STATUS---WORD MODE OV RESET
 A-REGISTER---UNDEFINED
 X-REGISTER---CIB Address

DPTR is an address pointer to a two-word information block:

Word 1: Positive number of bytes to be transferred.
 Word 2: Word address of I/O buffer.

(Note that the standard handlers use CIB Words 26 and 27 for this information).

I/O SIO: routine does the following:

1. Negates the byte count pointed to by the A register, and stores it into the Word interrupt location plus one.
2. Shifts the Buffer address pointed to by the A register to the left by one bit (converts to a byte address), then decrements the byte address and stores it into the word interrupt location plus two.

NOTE

(Steps 1 and 2 above complete the three-word Auto I/O sequence.
 The AIN/AOT instruction itself may be generated by a call to SINT:)

3. Calculates the delay count required for the watchdog timer, as follows (assume a ten millisecond Real Time Clock rate):
 - a. The negative byte count created in step 1 is loaded into the A register.
 - b. The contents of CIB Word 20 are stored in-line and executed as an instruction.
 - c. The contents of the A register are then negated (converted to positive) and incremented by 1000.

Steps a, b and c above compute the number of RTC "ticks" (normally 10 milliseconds each) to delay during the I/O operation. Since the number is constructed beginning with the byte count (step a) and incremented by 1000 (step c) the minimum delay possible is ten seconds, plus ten milliseconds for each data byte to be transferred. The purpose of step b is to permit a larger delay, if necessary. For example, CIB Word 20 can be set up by the user, when constructing the CIB prior to execution, to be a shift instruction (e.g., "LLA 1") which would double the value in the A register, and thus cause a twenty millisecond delay for each data byte (plus the ten second constant). Note that the instruction in CIB Word 20 is executed before the byte count in the A register has been converted from negative to positive, and before the constant 1000 is added. If the minimum delay (ten seconds, plus 10 milliseconds for each byte to be transferred) is adequate, then the instruction in CIB Word 20 should be zero (a no-op instruction). It is the responsibility of the user when creating the CIB table for his handler to determine how large a delay is required to permit completion of an I/O operation, and thus what instruction (normally LLA K, where K must be determined) is to be stored into CIB Word 20.



4. Sets up and executes the following I/O instructions:

SEL DA,X	Handler-determined function
SEL DA,5	Set word transfer mask
SEL DA,6	Set block transfer mask
SEL DA,Y	Handler-determined function

X and Y represent the function codes in bits 15 through 13 and 12 through 10, respectively, of CIB Word 12. (These function codes were originally copied from DIB Word 8 in a prior call to SINT:.) Note that if Select instructions of function X and/or Y are not required by the device, they can be organized in the DIB so that X=5 and Y=6, so that each is executed twice, or they can be set to a function code which has no meaning to the device, if such a code exists.

NOTE

If these function codes are all zero, it indicates an operation under Distributed I/O.

If the device uses function codes 5 and 6 for other purposes than to set the transfer masks, the user may wish to perform the Select functions within the handler itself, rather than calling SIO:.

5. Once the Select instructions have been executed, a call to RTX DELAY: is made, carrying the calculated delay time described in step 3 above.

If the Watchdog Timer expires before an end-of-block interrupt occurs, the instruction in CIB Word 1 (normally "SEL DA 7") is executed to disable interrupts for the device, and the "Error" and "Device Unresponsive" status bits are set in the DIB, and control is then passed to the EOR: routine at EORST:.

NOTE

SIO: does not set up the end-of-block interrupt location. This must be done in the handler.

3.2.3 INTP: (End of Block Interrupt Return Point)

The INTP: routine cancels the watchdog timer upon end-of-block interrupt, and passes Control to the return address of SIO:. Thus INTP: is an extension of SIO:, and is intended to be used only in conjunction with SIO:.



To call INTP: at end-of-block, the handler should, prior to calling SIO:, set up the following sequence at the end-of-block interrupt location:

```
JST    *$+1
DATA   TAG
```

Example:

```
TAG      EXTR    INTQ:,INTP:
        ENT
        JST      INTQ:
        DATA    $,0,0,0
        DATA    INTP:,8180,0
        DATA    CIB Address
        DATA    TAG
```

where TAG is a short calling sequence to the RTX INTQ: subroutine, which points to INTP: as the task to be queued.

(The user should first familiarize himself with the RTX INTQ: description in chapter 1 (RTX Functions)).

The above description is the method used by the standard I/O handlers for end-of-block interrupts. For this purpose, the first 12 words of the applicable CIB may be used to contain the calling sequence to INTQ:.

For example, the following is a representation of the first twelve locations within the CIB for the line printer:

```
      C : L P 0 --- L I N E P R I N T E R

LOC  INST ADDR LABEL MNEM OPERAND COMMENT
0000                NAM C:LP0
                EXTR INTQ:,INTP:,I:READ,I:RITE,I:FUN
                *
                * * * * *
                *
0004      DA      EQU 4
0042      INTAD   EQU :42
                *
                * * * * *
                *
0000      REL 0
0000      C:LP0 EQU $
0000      CIB   ENT
0001      4027   SEL DA,7      SELECT --- FC = 7
0002      F900   JST INTQ:
0003      0003   DATA $,0,0,0,INTP:,8180,0,CIB,CIB
0004      0000
0005      0000
0006      0000
0007
0008      1FF4
0009      0000
000A      0000
000B      0000
```



Note that the end-of-block interrupt location contains a JST into the CIB itself; Word 1 of the CIB is the SEL DA,7 instruction used by the SIO: routine. It is also executed at end of block, thus serving as a convenient method to turn off the interrupt masks following an I/O operation.

Following this instruction is a JST to INTQ: followed by the required parameters, of which INTP: is the task to be executed. Note also that this sequence will automatically cause the X register to be loaded with the CIB address upon entry to INTP:.

3.2.4 WAIT: (End of Record Delay Routine)

Calling sequence:

LDX	CIB Address
JST	WAIT:

returns with:

INTERRUPTS---ENABLED
 STATUS---UNCHANGED
 A-REGISTER---UNDEFINED
 X-REGISTER---CIB Address

The WAIT: routine utilizes the delay length specified in DIB Word 7 to delay a sufficient length of time at end-of-record to ensure that the device is physically ready to perform the next I/O request. (Generally, one character time is sufficient for this delay.)

The routine loads the delay count from DIB Word 7 depending on the I/O instruction at the Word interrupt location; i.e., if bit 13 of the I/O instruction is on, it is assumed to be an output instruction, and bits 0-7 of DIB Word 7 are used as the delay count. If bit 13 of the I/O instruction is off, it is assumed to be an input instruction, and bits 8-15 of DIB Word 7 are used as the delay count. Once the delay count is established, a call to RTX DELAY: is made; upon return from the delay, the routine exits to the caller.

3.2.5 EOFQ: (End of File Check Routine)

Once an end-of-block interrupt has occurred, EOFQ: may be called as follows:

LDX	CIB Address
JMP	EOFQ:

This routine does the following:

1. Examines the first two input characters in the buffer to determine whether they are '/'*
2. If so, control is passed to the EOF: routine.
3. If not, control is passed to the EOR: routine.

3.2.6 EOF: (End of File Routine)

Calling sequence:

```
LDX      CIB Address
JMP      EOF:
```

The EOF: routine is entered when it has been determined that an end-of-file has been encountered (the routine EOFQ: may be used to determine this).

The routine stores a zero value into CIB Word 28, loads the A register with an end-of-file status, and transfers control to the EOR: routine at EORST:.

3.2.7 EOR: (End of Record Routine)

Calling sequence:

```
LDX      CIB Address
JMP      EOR:
```

This routine is entered when the handler has completed the requested I/O operation and wishes to return to the calling task.

The routine loads the A register with the current status from CIB Word 32, and continues at EORST:.

3.2.8 EORST: (Alternate Entry Point to EOR:)

EORST: and EOR: are alternate entry points to the same end-of-record routine. The difference between the two is that EOR: loads the I/O status word into the A register from the CIB. EORST: assumes that the status is already in the A register.

Calling sequence:

```
LDX      CIB Address
LDA      I/O status (from handler)

JMP      EORST:
```

The routine does the following:

1. It copies the actual transfer count of the I/O operation from the CIB into Word 8 of the IOB.
2. It stores the status of the I/O operation (in the A register upon entry) into bits 15-8 of IOB Word 5.
3. It performs an RTX BEGIN: call, passing as a parameter the normal or abnormal return address of the caller, depending on the status. The abnormal return address is taken if any of bits 9, 10, 11, or 14 are set in word 5 of the IOB.
4. It calls WAIT: to perform an end-of-record delay.



5. It loads CIB Word 1 (assumed to be "SEL DA,7), masks off the low order two bits (to make it a SEL DA,4 or initialize instruction) and executes it in-line.
6. It empties the IOCH (I/O suspend) list into the READY list.
7. It then transfers to the IOX request scheduler routine to check to see if another request is pending for any device on the controller just used.

3.2.9 FETCH: (Input one character from an I/O device)

Calling sequence:

```

EXTR      FETCH:
.
.
.
LDA              CIB Address
JST              FETCH:

```

Returns with:

```

INTERRUPTS---ENABLED
STATUS---UNCHANGED
A-REGISTER---CONTAINS INPUT BYTE
X-REGISTER---UNCHANGED

```

The FETCH: routine calls WAIT: to wait one character time, then calls SIO: to perform a one-character I/O operation. Upon input of the character, it is checksummed, and the subroutine exits back to the caller.

The following assumptions are made by FETCH:.

1. The handler has previously zeroed out the checksum word (CIB Word 13) at the start of the record.

There exists in CIB words 34 through 37 the following sequence:

DATA	\$+1	Pointer to byte count
DATA	1	Byte count (1 character)
DATA	\$+1	Buffer address
DATA	0	One-character input buffer

which are required for FETCH:'s call to SIO:.

Upon return from FETCH:, the input character is in CIB word 37 as well as in the A register, and the cumulative checksum is in CIB word 13.



3.2.10 BUFFQ: (Store input character into buffer)

Calling sequence:

```
( EXTR      BUFFQ:
  .
  .
  .
  LDX      CIB Address
  JST      BUFFQ:
```

Returns with:

```
INTERRUPTS---ENABLED
WORD MODE
OVERFLOW---RESET (unless buffer filled)
A-REGISTER---CONTAINS INPUT BYTE
X-REGISTER---UNCHANGED
```

The BUFFQ: routine is designed to be used following a call to FETCH:, in that it moves CIB word 37 (stored into by FETCH:) into the user's buffer. The step-by-step procedure is:

1. The overflow register is reset.
2. The actual transfer count (CIB Word 28) is incremented.
3. The actual transfer count is compared to the requested count (CIB word 26).
4. If the actual count is greater (indicating that the buffer is already full), the buffer address (CIB Word 27) is incremented and the subroutine exits.
5. If the actual count is less, CIB Word 37 is copied into the user's buffer pointed to by CIB Word 27. Then Word 27 is incremented and the subroutine exits.
6. If the actual count is equal (indicating that this character will cause the buffer to be full), overflow is set and CIB Word 37 is copied into the user's buffer pointed to by CIB Word 27. Then Word 27 is incremented and the subroutine exits.

3.2.11 UNRES: (Unresponsive Device Routine)

Calling sequence:

```
EXTR      UNRES:
  .
  .
  .
  LDX      CIB Address
  JMP      UNRES:
```



3.2.12 IORTN: (Return to I/O Scheduler)

Calling sequence:

```
EXTR      IORTN:
.
.
.
LDX      CIB Address
JMP      IORTN:
```

In practice, an I/O handler is a subroutine with an abnormal calling sequence (a JMP instruction is used, rather than a JST). This is because I/O handlers are only "called" from one location, and thus the return is known. This return address is IORTN:. Therefore, once an I/O operation has been initiated, a jump to IORTN: must be made. Note that if the SIO: routine is called, it will exit to IORTN:.

3 CHARACTER-ORIENTED DEVICE HANDLER LISTING

The following listing illustrates the standard Character-oriented Device Handler (non-Fortran) written for an LSI-2 processor. The code also includes a table of equates used by RTX, its subexecutives, and its library modules, as well as a listing of the TTY console DIB (D:TY00) and TTY CIB (C:TY0). CONCORDANCE listings provide an alphabetized map of all symbols.

GE 0001 09/01/76 U9:4 94500-10 TX, IOX EQUATES
 MACRO2 (A?) SI= MACRUS BU= RTXEQU --- EQUATES USED IN RTX

```

0003 * * * * *
0004 *   THE EQUATES CONTAINED IN THIS ASSEMBLY
0005 *   ARE USED BY RTX AND ITS SUBEXECUTIVES AND
0006 *   ITS LIBRARY MODULES
0007 *
0008 *   IT MUST BE ASSEMBLED AND THE SYMBOL TABLE
0009 *   GENERATED BE PASSED TO THE RTX MODULE
0010 *   BEING ASSEMBLED
0011 *
0012 * * * * *
0013 *
0014 *   EQUATES COMMON TO SEVERAL BLOCK TYPES
0015 *
0016 * * * * *
0017 *
0018 0000 CHAIN EQU 0 PUNIER TO NEXT BLOCK
0019 0001 PRI EQU 1 PRIORITY (BITS 15-3)
0020 0003 CN EQU 3 COORDINATION NUMBER
0021 0002 QUEUE EQU 2 TOP OF QUEUE
0022 *
0023 * * * * *
0024 *
0025 *   T C B   E Q U A T E S
0026 *
0027 * * * * *
0028 *
0029 0001 STAPRI EQU PRI STATUS (BITS 0-2) & PRIORITY (BITS 15-3)
0030 0002 PREG EQU 2 PROGRAM REGISTER
0031 0003 AREG EQU 3 ACCUMULATOR REGISTER
0032 0004 XREG EQU 4 INDEX REGISTER
0033 *
0034 * * * * *
0035 *
0036 *   I O B   E Q U A T E S
0037 *
0038 * * * * *

```



PAGE 0002 09/01/76 09:46:43 94500-10 RTX, IOX EQUATES
MACR02 (A2) SI= MACROS HO= R1XEQU --- EQUATES USED IN RTX

0039		*				
0040	0001	IDT	EQU	1	DEVICE TYPE	
0041	0002	ICUN	EQU	2	UNIT NUMBER	
0042	0002	ITCB	EQU	2	ADDRESS OF USER'S TCH	
0043	0004	ILUN	EQU	4	LOGICAL UNIT NAME/NUMBER	
0044	0005	ISTA	EQU	5	STATUS	
0045	0005	IOP	EQU	5	OP-CODE	
0046	0006	IRCNT	EQU	6	REQUESTED COUNT	
0047	0007	IBUFF	EQU	7	BUFFER ADDRESS	
0048	0008	IACNT	EQU	8	ACTUAL COUNT TRANSMITTED	
0049	0009	IDAA	EQU	9	DIRECT ACCESS ADDRESS	

II/3-15

```

0051      * * * * *
0052      *
0053      *       C I B       E Q U A T E S
0054      *
0055      * * * * *
0056      *
0057      0000      CHOR      EQU      0          BEGINNING OF RECORD FLAG
0058      0001      CSEL7    EQU      1          SELECT FC = 7
0059      0004      CTMP1    EQU      4          TEMP CELL 1
0060      0005      CTMP2    EQU      5          TEMP CELL 2
0061      0006      CTMP3    EQU      6          TEMP CELL 3
0062      0007      CEHISK   EQU      7          END OF BLOCK TASK POINTER
0063      0009      CNEWA    EQU      9          A REGISTER FOR EBTSK
0064      000A      CNEWX    EQU     10          X REGISTER FOR EBTSK
0065      000C      CFUN     EQU     12          TEMP CELL FOR I/O INSTRUCT
0066      000D      CCSUM    EQU     13          CHECKSUM          TEMP
0067      000E      RENCNT   EQU     14
0068      000F      CJTBL    EQU     15          JUMP TABLE
0069      0013      CSPLOP   EQU     19          POINTER TO SPECIAL OP PROC
0070      0014      CDEL     EQU     20          DELAY MODIFICATION
0071      0015      CINTR    EQU     21          POINTER TO INTERRUPT ADDRE
0072      0016      EXCESS   EQU     22
0073      0017      CEOF     EQU     23
0074      *
0075      *       FILLED FROM IOB
0076      *
0077      0018      CIOB     EQU     24          IOB POINTER
0078      0019      COP      EQU     25          OPERATION CODE
0079      001A      CRCNT    EQU     26          REQUESTED COUNT
0080      001B      CBUFF    EQU     27          BUFFER ADDRESS
0081      001C      CTCNT    EQU     28          TRANSFER COUNT
0082      001D      CDAA     EQU     29          DIRECT ACCESS ADDRESS
0083      *
0084      *       FILLED FROM DIB
0085      *
0086      001E      CDIB     EQU     30          DIB POINTER
    
```

II/3-16



PAGE 0004 09/01/76 09:46:43 94500-10 RTX, IOX EQUATES
MACRO2 (A2) S1= MACRUS H0= RTXEQU --- EQUATES USED IN RTX

0087	001F	CFUN1 EQU	31	TEMP CELL 2 FOR FUNCTIONS
0088		*		
0089		*		TEMP STORAGE USED BY IOX AND ITS DRIVERS
0090		*		
0091	0020	STATUS EQU	32	DEVICE STATUS WORD
0092	0021	CRTN EQU	33	RETURN ADDRESS FROM I:SIO
0093	0022	CDCHN EQU	34	START OF DATA CHAIN
0094	0023	CDCHN1 EQU	CDCHN+1	
0095	0024	CDCHN2 EQU	CDCHN+2	
0096	0025	CDCHN3 EQU	CDCHN+3	

```

0098      * * * * *
0099      *
0100      *   D I R   E Q U A T E S
0101      *
0102      * * * * *
0103      *
0104      0001   DCIB   EQU   1       CIB POINTER
0105      0004   DSW    EQU   4       DEVICE SPECIFICATION WORD
0106      0005   DT     EQU   5       DEVICE TYPE
0107      0006   DCUN   EQU   6       CONTROLLER & UNIT NUMBERS
0108      0007   DDEL   EQU   7       END OF HLUCK DELAY TIMES
0109      0008   DFUN   EQU   8       FUNCTION CODES & FLAGS
0110      0009   DULS   EQU   9       UPPER LIMITS
0111      000A   DERRC  EQU  10      ERROR COUNTER
0112      000B   DSIRT  EQU  11      DIU START ADDRESSES & MODES
0113      000B   DSECT  EQU  11      VERIFY FLAG, DRIVE #, STARTING SECTOR
0114      000C   DHEAD  EQU  12      SECTORS/TRACK & STARTING HEAD
0115      000D   DCYL   EQU  13      SECTORS/CYLINDER & STARTING CYLINDER
0116      000E   DEOD   EQU  14      NUMBER OF SECTORS IN FILE
0117      000F   DCSECT EQU  15      FORMATTED SECTOR NO
0118      *
0119      * * * * *
0120      *
0121      *   I N T E R R U P T   B L O C K   E Q U A T E S
0122      *
0123      * * * * *
0124      0000   NTAIO  EQU   0       I/O INSTRUCTION
0125      0001   NTCNT  EQU   1       COUNT FOR AUTO I/O
0126      0002   NTBUFF EQU   2       BUFFER ADDRESS - 1
0127      0004   NTEOB  EQU   4       END-OF-BLOCK INTERRUPT
0128      0005   NTEOBA EQU   5       ADDRESS FOR EOB INSTRUCTION
0129      *
0130      * * * * *
0131      *
0132      *   M I S C E L L A N E I O U S   E Q U A T E S
0133      *
    
```

II/3-18



PAGE 0006 09/01/76 09:46:43 94500-10 RTX, IOX EQUATES
MACR02 (A2) SI= MACROS BU= RTXEQU --- EQUATFS USED IN RTX

```
0134          * * * * *
0135          *
0136      0010      ARROW EQU      29      BACKARROW FLAG SAME AS CDA
0137      0003      EORMSK EQU      3      END OF RECORD MASK
0138      0004      EOFMSK EQU      4      END OF FILE MASK
0139      4000      I:ERR EQU      :4000
0140      0800      I:BOP EQU      :800
0141      0400      I:RES EQU      :400
0142      0200      I:EOF EQU      :200
0143      0100      I:BOV EQU      :100      BEGINNING OF DEVICE STATUS BIT
0144      0300      I:EOD EQU      :300
0145      2000      I:NOID EQU      :2000
0146      4000      ERORR EQU      :4000
0147      001F      OPMSK EQU      31
0148      0080      IOREL EQU      :80
0149      0003      EORTYP EQU      3
0150      0004      PROMPT EQU      4
0151      0008      EUFTYP EQU      8
0152      0000      IFF      LSI305
0153      0005      IOREQ EQU      5
0154          ENDC
```

II/3-19



```

0218          MACRO  LLL
0219          IFF    LSI305
0220          LLL::  :1R0,#1-1
0221          ENDC
0222          IFT    LSI305
0223          REPT   #1
0224          LLL:
0225          ENDC
0226          ENUM

0228          MACRO  LLL:
0229          LLX    1
0230          RLA    1
0231          ENDM
0232          FORM   LLL::,12,4
0233          MACRO  INISIF  CIB INTERRUPT & DEVICE ADDRESS STUFF
0234          SPACE  1
0235          DA     EQU    #1      DEFINE DEVICE ADDRESS
0236          INTAD EQU    #2      DEFINE INTERRUPT ADDRESS (DATA)
0237          ABS   INTAD  ORG TO AUTO I/R LOCATION
0238          DATA 0,0,0
0239          ABS   INTAD+4  ORG TO END-OF-BLOCK INTERRUPT
0240          JST   *3+1    GO TO CIB FOR
0241          DATA CIB     . END OF BLOCK
0242          REL   0       ORG TO RELATIVE ZERO (0)
0243          SPACE 1
0244          ENDM

0246          MACRO  SINT      GENERATE CALL TO SETUP INTERRUPTS
0247          JST   SINT:      CALL SUBROUTINE
0248          IFF   #1(S)
0249          DATA :3800     MAKES STOP
0250          NOTE U,FIRST.PARAMETER
0251          ENDC
0252          IFF   #2(I)
0253          IFF   #2(O)
  
```

0010

DA
INTAD

II/3-20

PAGE 0010 09/01/76 09:46:43 94500-10 RTX, IOX EQUATES
MACRO2 (A2) S1= MACRUS BD= M A C R O S

0254 DATA :3800 MAKES STOP
0255 NOTE U,SECOND.PARAMETER
0256 ENDC
0257 ENDC
0258 IF1 LSI305
0259 IFT #2(I)
0260 DATA #3+:40F9
0261 ENDC
0262 IFT #2(O)
0263 DATA #3+:60F9
0264 ENDC
0265 ENDC
0266 IFF LSI305
0267 IFT #2(I)
0268 DATA #3+:13F9
0269 ENDC
0270 IFT #2(O)
0271 DATA #3+:23F9
0272 ENDC
0273 ENDC
0274 ENDM

0276 MACRO CIB
0277 TITL 'RTX/IOX CHARACTER I/O DRIVERS 93302-1XE0
0278 TITL C:#1...CONTROLLER.INFORMATION.BLOCK
0279 NAM C:#1
0280 EXTR SCH:,INTQ:,INTP:
0281 XDEF #4
0282 XDEF #5
0283 XDEF #6
0284 INTSTF #2,:#3
0285 CIB EQU \$
0286 C:#1 DATA SCH:
0287 IFT LSI305
0288 SEA DAX1+1
0289 ENDC

TI/3-21


```
0290          IFF      LS1305
0291          SEL      DAY1+1
0292          ENDC
0293          JST      INTQ:
0294          DATA    %,0,0,0,INIP:,,8180,0,C:#1,C:#1,0,0,0
0295          XREF      #4
0296          XREF      #5
0297          DATA    0
0298          XREF      #6
0299          DATA    0
0300          LLA      #7
0301          DATA    INTAD
0302          RES      12,0
0303          DATA    $+1,1,$+1,0
0304          END
0305          ENDM
0306          MACRO    DIH
0307          TITL     'RIX/IOX CHARACTER I/O DRIVERS 93302-1XE0
0308          TITL     D:#1...DEVICE.INFORMATION.BLOCK
0309          NAM      D:#1
0310          EXTR     C:#3
0311          CHAN     X::
0312          DATA    C:#3,0,0,:#4
0313          TEXT     #2
0314          DATA    0,:#5,:#6,0,:#7
0315          END
0316          ENDM
0317          MACRO    XDEF
0318          IFF      #1(0)
0319          EXTR     I:#1
0320          ENDC
0321          ENDM
0322          MACRO    XREF
0323          IFF      #1(0)
0324          DATA    I:#1
0325          ENDC
```

D:#1

PAGE 0012 09/01/76 09:46:43 94500-10 RTX, IOX
MACR02 (A2) SI= MACROS BO= M A C R O S

EQUATES

0326 IFT #1(0)
0327 DATA 0
0328 ENDC
0329 ENDM
0330 SAVE
0331 END

0000 ERRORS
0000 WARNING

PAGE 0001 09/01/76 09:47:57
MACRO2 (A2) SI= CHRDS RO=

RTX/IO CHARACTER I/O UNIVER 93302-1XE0
D : T Y 0 0 --- TELETYPE CONSOLE

```
0125 *
0126 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
0127 *
0128 0000          NAM      D:TY00
0129          EXTR      C:TY0
0130 *
0131 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
0132 *
0133 0000          D:TY00 CHAN  X::
0134 0001 0000          DATA  C:TY0,0,0,:5066,'TY','00'
          0002 0000
          0003 0000
          0004 5066
          0005 0409
          0006 80H0
* 0135 0007 020C          DATA  :20C,:A6E,:4800,0
          0008 0A6E
          0009 4800
          000A 0000

0136          END

0000 ERRORS
0000 WARNING
```

II/3-24

PAGE 0001 09/01/76 09:54:16 RTX/IOX CHARACTER I/O DRIVERS 93302-1X

X 0000 C:TY0 0129 0134
N 0133 D:TY00 0128
U 0000 X:: 0133
0136 SOURCE LINES

PAGE 0001 09/01/76 09:47:37
MACRO2 (A2) SI= CHRDS BU=

RTX/IOX CHARACTER I/O DRIVERS 93302-1XE0
C: T-Y 0 --- TELETYPE

0251 0000 NAM C:TYO
0252 EXTR INTQ:,INIP:,I:READ,I:RITE,I:FUN
0253 EXTR SCH:

0254 0002 INTSTF 7,2
0254+ 0002 0000
0254+ 0003 0000
0254+ 0004 0000
0254+ 0006
0254+ 0006 F907 0007
0254+ 0007 0000
0254+ 0000

0255 0000 C:TYO EQU \$
0256 0000 0000 CIB DATA SCH:
0257 0000 IFF LSI305
0258 0001 403F SEL DA,7 SELECT --- FC = 7
0259 ENDC
0263 0002 F900 0000 JST INTQ:
0264 0003 0003 DATA \$,0,0,0,INTP:,8180,0,CIB,CIB

0004 0000
0005 0000
0006 0000
0007 0000
0008 1FF4
0009 0000
000A 0000
000B 0000

0265 000C 0000 RES 3,0
0266 000F 0000 DATA I:READ,I:RITE,0,I:FUN,0
0010 0000
0011 0000
0012 0000
0013 0000

0267 0014 1353 LLA 4
0268 0015 0002 DATA INTAD

PAGE 0002 09/01/76 09:47:37
MACR02 (A2) SI= CHRDS HO=

RTX/IOX CHARACTER I/O DRIVERS 93302-1XED
C : I Y 0 --- TELETYPE

0269 0016 0000
0270 0022 0023
0023 0001
0024 0025
0025 0000

RES 12,0
DATA \$+1,1,\$+1,0

0271

END

0000 ERRORS
0000 WARNING



PAGE 0001 09/01/76 09:54:23 RTX/IOX CHARACTER I/O DRIVERS 93302-1X

N	0255	C:TYO	0251
	0256	CIH	0264
U	0000	DA	0258
X	0000	I:FUN	0252
X	0000	I:READ	0252
X	0000	I:RITE	0252
U	0000	INTAD	0268
X	0000	INTP:	0252
X	0000	INTU:	0252
U	0000	LSI305	0257
X	0000	SCH:	0253

0271 SOURCE LINES

PAGE 0001 09/01/76 09:47:37
MACR02 (A2) SI= CHRDS BO=

RTX/IOX CHARACTER I/O DRIVERS 93302-1XE0
CHARACTER READ-WRITE PROCEDURES

```
0427      0000      MACH  0      MUST WORK ON LSI AND ALPHA/NM-16
0428      * * * * *
0429      *
0430      *      ORIGINATING NAMES
0431      *
0432      * * * * *
0433      *
0434      0000      NAM      I:READ  INPUT REQUEST
0435      0074      NAM      I:RITE  OUTPUT REQUEST
0436      0085      NAM      I:FUN   SPECIAL FUNCTIONS PROCESSOR
0437      0083      NAM      RITE2   OUTPUT END OF RECORD (CR,LF,ETC)
0438      * * * * *
0439      *
0440      *      EXTERNAL NAMES
0441      *
0442      * * * * *
0443      *
0444      EXTR      BEGIN:   BEGIN TASK SERVICE
0445      EXTR      END:     END TASK SERVICE
0446      EXTR      SUBR:   BEGIN COMMON SUBROUTINE
0447      EXTR      SUBX:   END COMMON SUBROUTINE
0448      EXTR      DELAY:  DELAY SERVICE ROUTINE
0449      EXTR      EOF:    END OF FILE TASK
0450      EXTR      EOR:    END OF RECORD TASK
0451      EXTR      EORST:  SET STATUS AND END OF RECORD
0452      EXTR      SINT:   SET INTERRUPTS TASK
0453      EXTR      SIO:    START I/O
0454      EXTR      CKSUM:  COMPUTE CHECKSUM TASK
0455      EXTR      FETCH:  GET CHARACTER
0456      EXTR      EOFCK:  CHECK FOR END-OF-FILE
0457      EXTR      BUFFQ:  PUT BYTE INTO BUFFER
0458      EXTR      WAIT:   WAIT FOR DEVICE
0459      EXTR      EOFW:   CHECK FOR ASCII EOF ('/*')
```

IX/3-29


```

0461 * * * * *
0462 *
0463 * THIS ROUTINE WILL PROCESS ANY REQUESTS
0464 * TO INPUT FROM A CHARACTER DEVICE.
0465 *
0466 * ALL REQUESTED WILL BE ISSUED FOR ONE (1)
0467 * CHARACTER AT A TIME.
0468 * IF THE TRANSFERRED COUNT CONTAINED IN THE
0469 * CIB IS NEGATIVE THE DEVICE WILL BE
0470 * PROMPTED WITH THE CHARACTER OR CHARACTERS
0471 * CONTAINED IN THE CTCNT.
0472 *
0473 * * * * *
0474 *
0475 0000 REL 0
0476 0000 I:READ EQU $
0477 *
0478 0000 H41B 001B LDA @CBUFF
0479 0001 1350 LLA 1
0480 0002 9C0E 000E STA @REQCNT SAVE FOR A RESTART
0481 *
0482 0003 RFA2 EQU $ RESTART
0483 *
0484 0003 9C1B 001B STA @CBUFF SET BUFFER ADDRESS TO BYTE
0485 0004 C601 LAP 1
0486 0005 9C23 0023 STA @CCHN1 SET BYTE COUNT TO ONE
0487 0006 0110 ZAR
0488 0007 9C1D 001D STA @ARROW CLEAR BACK ARROW FLAG
0489 0008 9C1C 001C STA @CTCNT CLEAR TRANSFER COUNT
0490 0009 H416 0016 LDA @EXCESS PROMPT CHARACTERS
0491 000A 308E 0019 JAP RFAX IF NONE
0492 000B 1357 LLA 8 CHECK FOR #
0493 000C 2101 000E JAZ $+2 IF ONLY ONE
0494 000D C601 LAP 1 TWO PROMPT CHARACTERS
0495 000E 0150 IAR ADJUST COUNT
0496 000F 9C04 0004 STA @CTMP1 PUT IN MINI-IOH
    
```

II/3-30

PAGE 0003 09/01/76 09:47:37
MACRO2 (A2) SI= CHRDS BU=

R1X/IOX CHARACTER I/O DRIVERS 93302-1XE0
I:READ --- CHARACTER READ PROCEDURE

0497	0010	C608		LAP	IACNT	
0498	0011	8C18	0018	ADD	@CIOB	ADDRESS OF PROMPT CHARACTERS
0499	0012	9C05	0005	STA	@CTMP1+1	PUT IN MINI-IOB
0500	0013	FBDC	00F0	SINT		STANDARD, OUTPUT, 0
0500+	0014	23F9				
0501	0015	0030		TXA		CIB ADDRESS
0502		0000		IFF	LSI305	
0503	0016	8ADA	00F1	ADD	=CTMP1	ADDRESS OF MINI-IOB
0504				ENDC		
0508	0017	FBDA	00F2	JST	SIO:	DO OUTPUT
0509	0018	FBDA	00F3	JST	WAIT:	FOR DELAY.
0510		0019		EQU	\$	
0511	0019	FB06	00F0	SINT		STANDARD, INPUT, 0
0511+	001A	13F9				
0512	001B	C607		LAP	7	OP CODE MASK
0513	001C	8419	0019	AND	@COP	MASK OFF OP CODE
0514	001D	D203	0021	CMS	TWO	COMPARE TO TWO
0515	001E	F203	0022	JMP	UNFMTI	UNFORMATTED
0516	001F	F234	0054	JMP	BININ	BINARY
0517	0020	F20C	002D	JMP	RFA3	FORMATTED ASCII
0518	0021	0002		TWO	DATA	2 CONSTANT TWO (2)
0519				*		
0520				*		
0521				*	TITL UNFMTI	--- UNFORMATTED INPUT
0522				*		
0523		0022		UNFMTI	EQU	\$
0524				*		
0525	0022	FB01	00F4	JST	FETCH:	GET NEXT CHARACTER
0526	0023	FB01	00F5	JST	BUFFQ:	GO PUT INTO BUFFER
0527	0024	3242	0022	JOR	UNFMTI	GU GET NEXT CHARACTER
0528	0025	F3D0	00F6	JMP	EOR:	END OF RECORD

RFAX

TWO

*

*

*

*

*

*

II/3-31

PAGE 0004 09/01/76 09:47:57
MACR02 (A2) SI= CHRDS BU=

R1X/IOX CHARACTER I/O PARAMETERS 93302-1XE0
RFA --- READ FORMATTED ASCII

```
0530
0531      0026      * RFA10 EQU $
0532      *
0533 0026 FBCC 00F3      JST WAIT:
0534 0027 B40E 000E      LDA @REQCNT
0535 0028 F625 0003      JMP RFA2 GO RESTART
0536      *
0537      0029      * RFA4 EQU $ MAYBE GOOD
0538      *
0539 0029 9C1D 001D      STA @ARROW SET FLAG
0540 002A C0DF      CAI :DF IS IT BACK ARROW?
0541 002B F220 004C      JMP RFA9 YES, BACK UP
0542 002C FBC8 00F5      JST BUFF0: PUT INTO BUFFER
0543      *
0544      002D      * RFA3 EQU $ LOOP
0545      *
0546 002D FBC6 00F4      JST FETCH: GET CHARACTER
0547 002E FBCH 00F7      JST EOFCK: IF FILE MARK, GOODBYE
0548 002F A2C8 00F8      IOR =:B0
0549 0030 9C25 0025      STA @CDCHN3 HIGH-ORDER BIT ON
0550 0031 C0FF      CAI :FF IS IT RUBOUT?
0551 0032 F605 002D      JMP RFA3 YES, IGNORE IT
0552 0033 D2C5 00F9      CMS =:B0 HOW ABOUT A CARRIAGE RETURN?
0553 0034 F607 002D      JMP RFA3 TOO SMALL
0554 0035 F60C 0029      JMP RFA4 GOT A LIVE ONE
0555      *
0556      * FOUND CARRIAGE RETURN
0557      *
```

II/3-32

PAGE 0005 09/01/76 09:47:37
MACRO2 (A2) SI= CHRDS HD=

RTX/IOX CHARACTER I/O DRIVERS 93302-1XE0
RFA --- READ FORMATTED ASCII

```
0559          *
0560          *   FOUND CARRIAGE RETURN
0561          *
0562 0036 C604      LAP   PROMPT   PROMPTABLE BIT
0563 0037 B41F 001F AND    @CFUN1  IS IT?
0564 0038 210C 0045 JAZ   RFA1    IF NOT, GET OUT
0565 0039 C603      LAP   EORMSK  MASK FOR EOR TYPE
0566 003A 841F 001F AND    @CFUN1  GET EOR TYPE
0567 003B C002      CAI    2        IS IT CR/LF ?
0568 003C F201 003E JMP   $+2    YES, ECHO LINE FEED
0569 003D F207 0045 JMP   RFA1   NO, FORGET LINE FEED
0570 003E F8B4 00F3 JST   WAIT:  DO A HICCUP
0571 003F F8B0 00F0 SINT  STANDARD, OUTPUT, 0
0571+ 0040 23F9
0572 0041 B250 0092 LDA   CRLF+1  LINE FEED
0573 0042 9C25 0025 STA   @CDCHN3 DATA CHARACTER
0574 0043 B422 0022 LDA   @CDCHN  POINTER TO MINI-IOB
0575 0044 FBAD 00F2 JST   SIU:   OUTPUT
0576
0577          *   RFA1 EQU   $        CHECK FOR VALID RECORD
0578          *
0579 0045 BC1D 001D EMA   @ARROW  CORRECTION FLAG
0580 0046 C0DF      CAI    :DF     LAST CHARACTER BACK ARROW
0581 0047 F621 0026 JMP   RFA10  YES, DO IT OVER
0582 0048 B40E 000E LDA   @REQCNT RESTART ADDRESS
0583 0049 13D0      LRA    1      MAKE IT WORD
0584 004A 9C1B 001B STA   @CBUFF
0585 004B F3AE 00FA JMP   EOF0:  CHECK FOR END OF FILE
0586
0587          *   RFA9 EQU   $        BACK ARROW FOUND
0588          *
0589 004C B41C 001C LDA   @CICNT  CURRENT COUNT
0590 004D 2160 002D JAZ   RFA3   IF AT BEGINNING
0591 004E 00D0      DAR    DOWN ONE
0592 004F 9C1C 001C STA   @CICNT  RESTORE IT
0593 0050 B41B 001B LDA   @CBUFF  * *
```

PAGE 0006 09/01/76 09:47:37
MACRO2 (A2) SI= CHRDS BU=

RTX/IOX CHARACTER I/O DRIVERS 93302-1XE0
RFA --- READ FORMATTED ASCII

0594	0051	00D0	DAR		* DECREMENT BUFFER ADDRESS
0595	0052	9C1B 001B	STA	@CBUFF	* *
0596	0053	F626 002D	JMP	RFA3	GO GET NEXT

II/3-34

PAGE 0007 09/01/76 09:47:37
MACRO2 (A2) SI= CHRDS BU=

RTX/IOX CHARACTER I/O DRIVERS 93302-1XE0
BININ --- INPUT BINARY ROUTINE

```
0598      *      READ A BINARY RECORD
0599      *      FIND THE RECORD HEADER
0600      BININ EQU      $
0601      0054 FB9F 00F4      JST      FETCH:      GET A CHARACTER
0602      0055 C0FF          CAI      :FF      IS IT A RUBOUT
0603      0056 F202 0059      JMP      $+3      YES GO GET BYTE COUNT
0604      0057 FB9F 00F7      JST      EOFCK:      CHECK FOR /*
0605      0058 F604 0054      JMP      BININ
0606      *
0607      *      GET THE BYTE COUNT
0608      *
0609      0059 0110          ZAR
0610      005A 9C0D 000D      STA      @CCSUM      CLEAR THE CHECKSUM
0611      005B FB98 00F4      JST      FETCH:      GET FIRST CHARACTER
0612      005C 1357          LLA      8          SHIFT TO HIGH ORDER BYTE
0613      005D 9C1D 001D      STA      @CDAA      SAVE IN TEMP CELL
0614      005E FB95 00F4      JST      FETCH:      GET SECOND CHARACTER
0615      005F A41D 001D      IOR      @CDAA      MERGE THE TWO BYTES
0616      0060 3101 0062      JAM      $+2      NOT AN END OF FILE
0617      0061 F399 00FB      JMP      EOF:      AN END-OF-FILE
0618      *
0619      *      READ THE INPUT DATA
0620      *
0621      0062 0310          NAR
0622      0063 9C0E 000E      STA      @REQCNT      SAVE REQUIRED COUNT
0623      *
0624      0064          NEXT1 EQU      $      GET NEXT ONE
0625      *
0626      0064 FB8F 00F4      JST      FETCH:      GO GET NEXT BYTE
0627      0065 FB8F 00F5      JST      BUFFQ:      GO WUFUE INTO BUFFER
0628      0066 DC0E 000E      IMS      @REQCNT      INCREMENT NUMBER OF BYTES
0629      0067 F603 0064      JMP      NEXT1
0630      *
0631      *      PERFORM CHECKSUM
0632      *
0633      0068 C6FF          LAP      :FF
```

PAGE 0008 09/01/76 09:47:57
MACRO2 (A2) SI= CHRDS BO=

HTX/IOX CHARACTER I/O DRIVERS 93302-1XE0
BININ --- INPUT BINARY ROUTINE

0634	0069	840D	0000	AND	~CCSUM	MASK OFF LOW ORDER BITS
0635	006A	9C1D	001D	STA	~CDAA	SAVE RECORD CHECKSUM
0636	006B	F88B	00F4	JST	FETCH:	FIRST BYTE
0637	006C	1357		LLA	8	SHIFT TO HIGH ORDER BYTE
0638	006D	9C0E	000E	STA	@REQCNT	SAVE
0639	006E	F885	00F4	JST	FETCH:	SECOND CHARACTER
0640	006F	A40E	000E	IOR	@REQCNT	MERGE TWO BYTES
0641	0070	941D	001D	SUB	~CDAA	SUBTRACT COMPUTED CHECKSUM
0642	0071	2101	0073	JAZ	\$+2	IF EVERYTHING OK
0643	0072	H289	00FC	LDA	=ERROR	ERROR CODE
0644	0073	F389	00FD	JMP	EORST:	ERROR EXIT

II/3-36

```
0649 * * * * *
0650 *
0651 * THIS ROUTINE WILL PROCESS ANY REQUESTS
0652 * TO OUTPUT TO A CHARACTER DEVICE.
0653 *
0654 * AFTER THE REQUEST HAS BEEN STARTED, CONTROL
0655 * WILL BE RETURNED TO THE IO SCHEDULER
0656 *
0657 * * * * *
0658 *
0659 0074 I:RITE EQU $
0660 *
0661 0074 FH78 00F0 SINT STANDARD,OUTPUT,0
0661+ 0075 23F9
0662 0076 R41A 001A LDA WCRcnt REQUESTED COUNT
0663 0077 9C1C 001C STA WTCNT SET TRANSFERED COUNT
0664 *
0665 * WHAT WAS REQUEST
0666 *
0667 0078 C603 LAP 3
0668 0079 8419 0019 AND WCOPI MASK OFF OP-CODE
0669 007A 9C19 0019 STA WCOPI REPLACE NEW ONE
0670 007B C003 CAI 3 IF BINARY
0671 007C F21E 009B JMP RITE10 FORMATTED BINARY
0672 007D 0030 IXA
0673 0000 IFF LSI305
0674 007E 8A7F 00FE ADD =CRCNT ADDRESS OF DATA CHAIN
0675 ENDC
0679 *
0680 007F RITE1 EQU $ DUIT TUIT
0681 *
0682 007F FB72 00F2 JST SIO: START I/O
0683 0080 H419 0019 LDA WCOPI
0684 0081 C001 CAI 1 IF UNFORMATTED
0685 0082 F373 00F6 JMP EOR: GO TO END OF RECORD
0686 *
```

II/3-37


```
0687 * OUTPUT TRAILER RECORD
0688 *
0689 0083 RITE2 EQU $
0690 0083 FB6F 00F3 JST WAIT: WAIT AWHILE
0691 0084 C603 LAP EORMSK END OF RECORD MASK
0692 0085 841F 001F AND @CFUN1 MASK OFF THE EOR FLAG
0693 0086 9C1B 001B STA @CBUFF
0694 0087 8A08 0090 ADD EORBAD ADD START OF BUFFER ADDRESS
0695 0088 BC1B 001B EMA @CBUFF PUT IT AND PICK UP EOR FLAG
0696 0089 13D0 LRA 1
0697 008A 0150 IAR CORRECT # OF CHARACTERS
0698 008B 9C1A 001A STA @CRCNT PUT INTO DATA CHAIN
0699 008C 0030 TXA
0700 0000 IFF LSI305
0701 008D 8A70 00FE ADD =CRCNT DATA CHAIN ADDRESS
0702 ENDC
0706 *
0707 008E LEAVE EQU $ CALL SIO: AND GO TO EOR:
0708 *
0709 008E FB63 00F2 JST SIO: START I/O
0710 008F F366 00F6 JMP EOR: ALL DONE
0711 0090 0091 EORBAD DATA CRLF
0712 0091 8DBA CRLF DATA :8DBA,:8ABD,:8DBA,:A08D
0092 8ABD
0093 8DBA
0094 A08D
0713 0095 0000 NULLS DATA 0,0,0
0096 0000
0097 0000
0714 0098 FF00 EOF1 DATA :FF00,0
0099 0000
0715 009A AFAA EOF2 TEXT '/*'
0716 *
0717 009B RITE10 EQU $ FORMATTED BINARY
0718 *
0719 009B 0110 ZAR
```

II/3-38

PAGE 0011 09/01/76 09:47:37
MACR02 (A2) SI= CHRDS BU=

R1X/IOX CHARACTER I/O DRIVERS 93302-1XE0
I:RITE --- CHARACTER WRITE PROCEDURE

0720	009C	9C0D	000D	STA	WCCSUM	CLEAR THE CHECKSUM BYTE
0721	009D	H214	00B2	LDA	RHEDAD	RECORD HEADER TASK
0722	009E	FB53	00F2	JST	SIO:	START I/O
0723	009F	FB53	00F3	JST	WAIT:	WAIT AWHILE
0724				*		
0725		00A0		RITE11 EQU	\$	OUTPUT BYTE COUNT
0726				*		
0727	00A0	H41A	001A	LDA	WCRCNT	REQUESTED COUNT
0728	00A1	9C25	0025	STA	WCDCHN3	PUT INTO CIB
0729	00A2	C602		LAP	2	TRANSFER COUNT
0730	00A3	9C23	0023	STA	WCDCHN1	PUT INTO DATA CHAIN
0731	00A4	B422	0022	LDA	WCDCHN	DATA CHAIN ADDRESS
0732	00A5	FA2E	00D4	JST	I:OCS	GO COMPUT CHECKSUM
0733	00A6	FB48	00F2	JST	SIO:	START I/O
0734	00A7	FB48	00F3	JST	WAIT:	WAIT AWHILE
0735				*		
0736		00A8		RITE12 EQU	\$	OUTPUT BINARY RECORD
0737				*		
0738	00A8	0030		IXA		
0739		0000		IFF	LSI305	
0740	00A9	8A54	00FE	ADD	WCRCNT	DATA CHAIN ADDRESS
0741				ENDC		
0745	00AA	FA29	00D4	JST	I:OCS	COMPUTE CHECKSUM
0746	00AB	FB46	00F2	JST	SIO:	START I/O
0747	00AC	FB46	00F3	JST	WAIT:	WAIT A BIT
0748				*		
0749		00AD		RITE13 EQU	\$	OUTPUT CHECKSUM
0750				*		
0751	00AD	C6FF		LAP	:FF	
0752	00AE	840D	000D	AND	WCCSUM	MASK OFF CHECKSUM
0753	00AF	9C25	0025	STA	WCDCHN3	PUT INTO CIB
0754	00B0	H422	0022	LDA	WCDCHN	DATA CHAIN ADDRESS
0755	00B1	F623	008E	JMP	LEAVE	LEAVE BECAUSE YOU'RE DONE
0756	00B2	00B3		RHEDAD DATA	RHEAD	
0757	00B3	0005		RHEAD DATA	5, NULLS+1	
	00B4	0096				

II/3-39

Computer Automation



```

0759          * * * * *
0760          *
0761          *   THIS ROUTINE WILL PROCESS THE SPECIAL
0762          *   FUNCTIONS REQUESTED FOR THE I/O DEVICES
0763          *
0764          * * * * *
0765          *
0766          00B5      I:FUN  EQU    $
0767          *
0768          00B5 FB3A 00F0      SINT  STANDARD,OUTPUT,0
0768+ 00B6 23F9
0769          00B7 C60F          LAP    :F
0770          00B8 8419 0019      AND    @COP    MASK OFF OP CODE
0771          00B9 9C19 0019      STA    @COP
0772          00BA C00C          CAI    :C      IF A C
0773          00BB F203 00BF      JMP    FMARK  JMP TO WRITE FILE MARK
0774          00BC C00D          CAI    :D      IF A D
0775          00BD F208 00C6      JMP    PLEAD  JMP TO PUNCH LEADER
0776          00BE F337 00F6      JMP    EOR:   GO TO END OF RECORD
0777          *
0778          00BF      FMARK  EQU    $      WRITE FILE MARK
0779          *
0780          00BF H41F 001F      LDA    @CFUN1  SPECIAL FLAGS
0781          00C0 13D2          LRA    3      MOVE EOF BIT TO DV
0782          00C1 B20F 00D1      LDA    SEOF   '/' FILE MARK
0783          00C2 3201 00C4      JOR    $+2
0784          00C3 F644 007F      JMP    RITE1  IF THAT'S IT, DUIT TUIT
0785          00C4 B612 00B2      LDA    FEOF   RUBOUT-NULL-NULL
0786          00C5 FH2C 00F2      JST    SIO:   WRITE IT
0787          *
0788          00C6      PLEAD  EQU    $      PUNCH LEADER
0789          *
0790          00C6 C714          LAM    20
0791          00C7 9C16 0016      STA    @EXCESS  SET COUNT FOR 20 TIMES
0792          00C8 FH2A 00F3      JST    WAIT:   WAIT A BIT
0793          00C9 B204 00CE      LDA    LEADER  ADDRESS OF LEADER CHAIN
  
```

II/3-40

PAGE 0013 09/01/76 09:47:37
MACRO2 (A2) SI= CHRDS BO=

R1X/IOX CHARACTER I/O DRIVERS 93302-1XE0
I : FUN --- FUNCTIONS

0794	00CA	F827	00F2	JST	SIO:	OUTPUT 6 NULLS
0795	00CB	DC16	0016	IMS	WEXCESS	ARE WE DONE
0796	00CC	F604	00C8	JMP	\$-4	NO
0797	00CD	F328	00F6	JMP	EOR:	JMP END OF RECORD
0798	00CE	00CF		LEADER DATA	\$+1,6, NULLS	
	00CF	0006				
	00D0	0095				
0799		00B2		FEOF	EQU	RHFDAD
0800	00D1	00D2		SEOF	DATA	\$+1,2, EOF2
	00D2	0002				
	00D3	009A				

```

0802      * * * * *
0803      *
0804      *   THIS ROUTINE WILL SEARCH THRU THE OUTPUT
0805      *   DATA CHAIN AND CREATE THE CHECKSUM FOR
0806      *   THE ENTIRE CHAIN
0807      *
0808      *   CALLING SEQUENCE:
0809      *
0810      *   JST   I:OCS
0811      *
0812      *   A REGISTER MUST CONTAIN THE ADDRESS OF
0813      *   THE FIRST PORTION OF THE CHAIN
0814      *   THE CHECKSUM IS TO BE COMPUTED
0815      *   X REGISTER MUST CONTAIN THE CIB ADDRESS
0816      *
0817      *   RETURN STATUS:
0818      *
0819      *   A REGISTER CONTAINS A 8 BIT CHECKSUM
0820      *   X REGISTER UNCHANGED
0821      *   STATUS:
0822      *   OVERFLOW --- RESET
0823      *   REMAINDER IS UNCHANGED
0824      *
0825      *   THE COMPUTED CHECKSUM IS PLACED BACK IN
0826      *   THE CIB
0827      *
0828      * * * * *
0829      *
0830      0004 0800      I:OCS  ENT
0831      0005 9A18 00EE  STA    TMP3    SAVE DATA CHAIN ADDRESS
0832      0006 EA16 00ED  STX    TMP2    SAVE CIB ADDRESS
0833      0007 B400 0000  LDA    @CCSUM  CHECKSUM BYTE
0834      0008 9A0A 00E3  STA    I:OCS5  INITIALIZE CHECKSUM
0835      0009 B314 00EE  LDA    *TMP3    NUMBER OF BYTES IN RECORD
0836      000A 0310      NAR
0837      000B 9A13 00EF  STA    COUNT   SET BYTE COUNTER

```

II/3-42

PAGE 0015 09/01/76 09:47:37
MACR02 (A2) SI= CHRDS BO=

RTX/IOX CHARACTER I/O DRIVERS 93302-1XE0
I : O C S --- OUTPUT CHECKSUM

```
0838 00DC DA11 00EE      IMS      TMP3      BUMP CHAIN POINTER
0839 00DD E310 00EE      LDX      *TMP3     BUFFER ADDRESS
0840 00DE 1328              LLX      1         SET TO BYTE ADDRESS
0841
0842      00DF      *      I:OCS4 EQU $      COMPUTE CHECKSUM FOR NEXT
0843      *
0844 00DF 0E00              SBM              SET BYTE MODE
0845 00E0 R400 0000      LDAB     @0      LOAD OUTPUT BYTE
0846 00E1 0F00              SWM              SET WORD MODE
0847 00E2 F81C 00FF      JSI      CKSUM:  GO COMPUTE CHECKSUM
0848 00E3 0000      I:OCS5 DATA $-1  CHECKSUM DATA CELL
0849 00E4 0128              IXR              INCREMENT BUFFER ADDRESS
0850 00E5 DA09 00EF      IMS      COUNT     INCREMENT COUNT DONE
0851 00E6 F607 00DF      JMP      I:OCS4  NOT DONE
0852      *
0853      00E7      I:OCS9 EQU $      ALL DONE SO CLEAN HOUSE
0854      *
0855 00E7 B604 00E3      LDA      I:OCS5  COMPUTED CHECKSUM
0856 00E8 E204 00ED      LDX      TMP2     RESTORE X REGISTER
0857 00E9 9C0D 000D      SIA      @CCSUM  PUT CHECKSUM IN CIB
0858 00EA B203 00EE      LDA      TMP3
0859 00EB 00D0      DAR              RESTORE A REGISTER
0860 00EC F718 00D4      RTN      I:OCS   R E T U R N
0861 00ED 0000      TMP2 RES 1,0
0862 00EE 0000      TMP3 RES 1,0
0863 00EF 0000      COUNT RES 1,0
0864      0010      LPOOL
      00F0 0000
      00F1 0004
      00F2 0000
      00F3 0000
      00F4 0000
      00F5 0000
      00F6 0000
      00F7 0000
      00F8 0080
```

II/3-43



PAGE 0016 09/01/76 09:47:37
MACR02 (A2) S1= CHRDS HO=

RTX/IOX CHARACTER I/O DRIVERS 93302-1XE0
I : O C S --- OUTPUT CHECKSUM

00F9 008D
00FA 0000
00FB 0000
00FC 4000
00FD 0000
00FE 001A
00FF 0000

0865

END

0000 ERRORS
0000 WARNING

II/3-44

U	0000	ARROW	0488*	0539*	0579*				
X	0000	BEGIN:	0444						
	0600	HININ	0516	0605					
X	0000	BUFFQ:	0457	0526*	0542*	0627*			
U	0000	CBUFF	0478	0484*	0584*	0593	0595*	0693*	0695*
U	0000	CCSUM	0610*	0634	0720*	0752	0833	0857*	
U	0000	CDA A	0613*	0615	0635*	0641			
U	0000	CDCHN	0574	0731	0754				
U	0000	CDCHN1	0486*	0730*					
U	0000	CDCHN3	0549*	0573*	0728*	0753*			
U	0000	CFUN1	0563	0566	0692	0780			
U	0000	CI0B	0498						
X	0000	CKSUM:	0454	0847*					
U	0000	CUP	0513	0668	0669*	0683	0771	0771*	
	0863	COUNT	0837*	0850*					
U	0000	CRCNT	0662	0674	0677	0698*	0701	0704	0727
			0743						
	0712	CRLF	0572	0711					
U	0000	CTCNT	0489*	0589	0592*	0663*			
U	0000	CTMP1	0496*	0499*	0503	0506			
U	0000	D	0500	0511	0571	0661	0768		
X	0000	DELAY:	0448						
X	0000	END:	0445						
	0714	EOF1							
	0715	EOF2	0800						
X	0000	EOF:	0449	0617					
X	0000	EOFCK:	0456	0547*	0604*				
X	0000	EOFQ:	0459	0585					
X	0000	EOR:	0450	0528	0685	0710	0776	0797	
	0711	EORBAD	0694						
U	0000	EURMSK	0565	0691					
X	0000	EORST:	0451	0644					
U	0000	ERROR	0643						
U	0000	EXCESS	0490	0791*	0795*				
	0799	FE0F	0785						
X	0000	FETCH:	0455	0525*	0546*	0601*	0611*	0614*	0520*
			0639*						0556*

II/3-45

	0778	FMARK	0773						
N	0766	I:FUN	0436						
	0830	I:OCS	0732*	0745*	0860				
	0842	I:OCS4	0851						
	0848	I:OCS5	0834*	0855					
	0853	I:OCS9							
N	0476	I:READ	0434						
N	0659	I:RITE	0435						
U	0000	IACNT	0497						
U	0000	INPUT	0511						
	0798	LEADER	0793						
	0707	LEAVE	0755						
U	0000	LSI305	0502	0505	0645	0673	0676	0700	0705 0739
			0742						
	0624	NEXT1	0629						
	0713	NULLS	0757	0798					
U	0000	OUTPUT	0500	0571	0661	0768			
	0788	PLEAD	0775						
U	0000	PROMPT	0562						
U	0000	REQCNT	0480*	0534	0582	0622*	0628*	0634*	0670
	0577	RFA1	0564	0569					
	0531	RFA10	0581						
	0482	RFA2	0535						
	0544	RFA3	0517	0551	0553	0590	0596		
	0537	RFA4	0554						
	0587	RFA9	0541						
	0510	RFAX	0491						
	0757	RHEAD	0756						
	0756	RHEAD	0721	0799					
	0680	RITE1	0784						
	0717	RITE10	0671						
	0725	RITE11							
	0736	RITE12							
	0749	RITE13							
N	0689	RITE2	0437						
	0800	SEOF	0782						
X	0000	SINI:	0452						

PAGE 0003 09/01/76 09:54:48 RTX/IUX CHARACTER I/O DRIVERS 93302-1X

X 0000 SIU: 0453 0508* 0575* 0682* 0709* 0722* 0733* 0746*
0786* 0794*
U 0000 STANDA 0500 0511 0571 0661 0768
X 0000 SUHR: 0446
X 0000 SUBX: 0447
0861 IMP2 0832* 0856
0862 TMP3 0831* 0835 0838* 0839 0858
0518 TWO 0514
0523 UNFMTI 0515 0527
X 0000 WAIT: 0458 0509* 0533* 0570* 0690* 0723* 0734* 0747*
0792*

0865 SOURCE LINES



SECTION 4

DIB AND CIB DESCRIPTIONS

The DIB and CIB are tables which are used in communication between IOX and a particular I/O handler or the File Manager.

The following DIB and CIB descriptions apply to all standard IOX handlers. DIB and CIB descriptions for non-standard handlers (for example, the IEEE Intelligent Cable handler) are included in Section 7 and for the File Manager, in Section 5.

4.1 DEVICE INFORMATION BLOCK (DIB) - 11 TO 18 WORDS

Words 0 to 10 are used by all IOX device handlers. Words 11 to 17 are used by specific handlers and the File Manager.

Figure 4-1 illustrates the DIB configuration.



DEVICE INFORMATION BLOCK

STANDARD NAME *	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	word
CHAIN	DIB CHAIN ADDRESS																0
DCIB	CIB ADDRESS																1
QUEUE	USED BY IOX TO QUEUE REQUESTS																2
CN	COORDINATION NUMBER																3
DSW	DEVICE SPECIFICATION WORD																4
DT	DEVICE NAME																5
CON	CONTROLLER NUMBER								UNIT NUMBER								6
DEL	INPUT RTC TICKS								OUTPUT RTC TICKS								7
DFUN	FUNCTION CODE	FUNCTION CODE	FUNCTION CODE	FUNCTION CODE	FUNCTION CODE	FUNCTION CODE	FUNCTION CODE	FUNCTION CODE	FUNCTION CODE	FILE MARK	PF	END OF RECORD					8
DULS	MAX BYTES-ASCII								MAX BYTES-BINARY								9
RC	HARDWARE ERROR COUNT (except MTIC)																10
	<p style="text-align: center;">ADDITIONAL WORDS USED BY SPECIFIC HANDLERS</p>																11
																	17

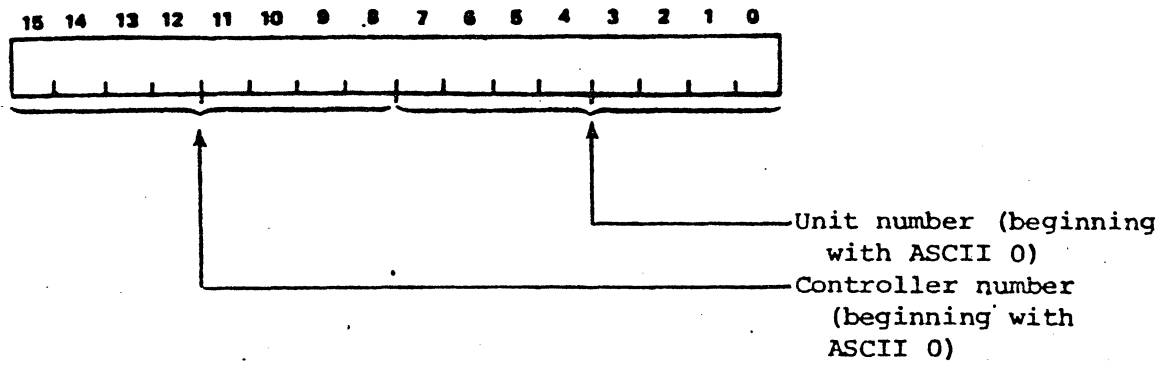
} IOB
Words 1
and 2

* refer to the I/O Handler listing at the end of Section 3.

Figure 4-1. DIB Configuration

4.2 REGULAR DIB CONFIGURATION (ALL HANDLERS) - WORDS 0 TO 10

- Word 0 Chain pointer to next DIB (CHAN directive). Last DIB contains 0. The DIB CHAN operand is X::.
- Word 1 Associated CIB address. (See list of standard CIB names at the end of Section 4.)
- Word 2 Used by IOX as a pointer to queue requests for this DIB. Initialize to zero.
- Word 3 Device coordination number. Initialize to zero.
- Word 4 Device Specification Word (DSW). Each of the 16 bits corresponds to the equivalent binary value described for bits 0-3 of IOB Word 5 (opcode); e.g., if the device is capable of reading Formatted ASCII (which function, if requested by the IOB, would appear as 0010 in bits 0-3 in IOB Word 5) then bit 2 should be set on in the DSW. If the device can punch leader (1101 in bits 0-3 in IOB Word 5), then bit 13 (:D) should be set on in the DSW.
- Word 5 and 6 Device Name. These words are copied into IOB words 1 and 2, respectively, upon finishing a call to IO:. These words contain four ASCII characters. Word 5 contains the first two characters which specify the device ("CR" for card reader, for example). Word 6 uses the following format for the third and fourth characters:

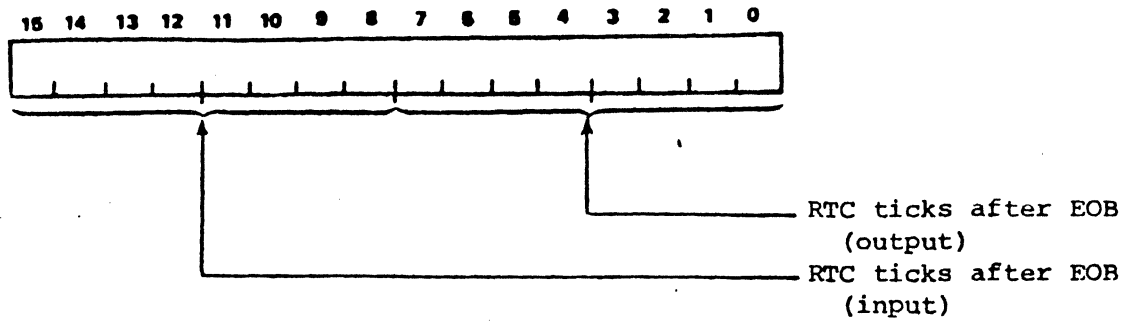


For Fortran tasks using the teletype or lineprinter as a list device with carriage control character recognition, or for a disk with end-of-file capability, the third character of the device name must be an "F", to serve as a flag that the Fortran handler is to be used.



Word 7

This word uses the following format:



Bits 8-15. A binary value representing the number of Real-Time Clock ticks to delay after an end-of-block interrupt for an input operation, before the device is considered available for the next I/O operation.

Bits 0-7. A binary value representing the number of Real-Time Clock ticks to delay after an end-of-block interrupt for an output operation, before the device is considered available for the next I/O operation.

Word 8

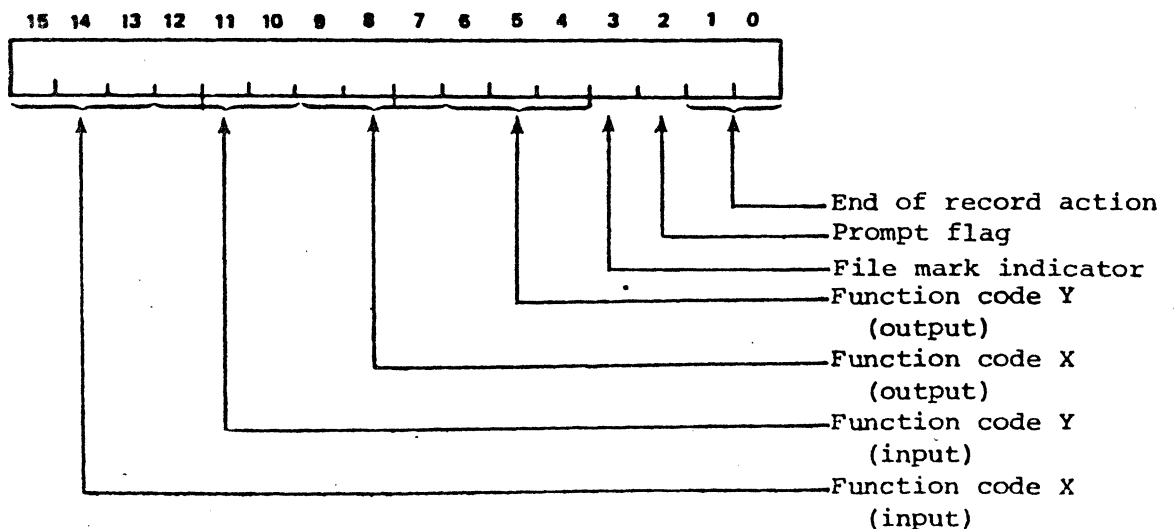
This word contains function codes which are executed in Select instructions to initiate an I/O operation if SIO: is called.

The order of execution of the Select instruction within SIO: is:

```

SEL      DA,X
SEL      DA,5
SEL      DA,6
SEL      DA,Y
    
```

This word uses the following format:



Bits 13-15. Contains function code X in the above sequence, for an input operation.

Bits 10-12. Contains function code Y in the above sequence, for an input operation.

Bits 7-9. Contains function code X in the above sequence, for an output operation.

Bits 4-6. Contains function code Y in the above sequence, for an output operation.

Bit 3. A flag signifying the type of file mark to be used for the device.

- 1 = slash/asterisk
- 0 = rubout/nll/null

Bit 2. A flag signifying whether the device is to be prompted before an input operation.

- 1 = Prompt the device
- 0 = Do not prompt the device

Bits 0-1. These bits represent the end of record action to be taken for Formatted ASCII output:

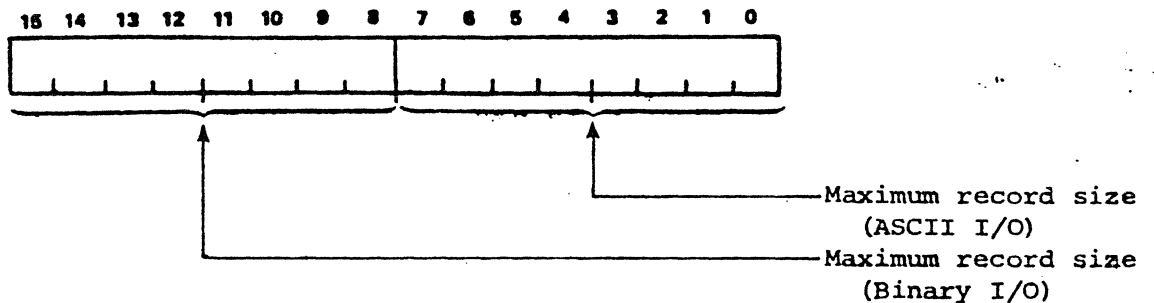
- 00 = Output carriage return only
- 01 = Output line feed only
- 10 = Output carriage return and line feed
- 11 = Output space and carriage return

NOTE

Word 8 is set to zero for Distributed I/O and Disk DIB's.

Word 9

This word uses the following format:



Bits 8-15. Maximum record size (in bytes) for formatted ASCII I/O operations. (Zero signifies unlimited record size.)

Bits 0-7. Maximum record size (in bytes) for binary I/O operations. (Zero signifies unlimited record size.)



Word 10 Cumulative hardware error count (must be incremented by the individual handler). Initialize to zero.

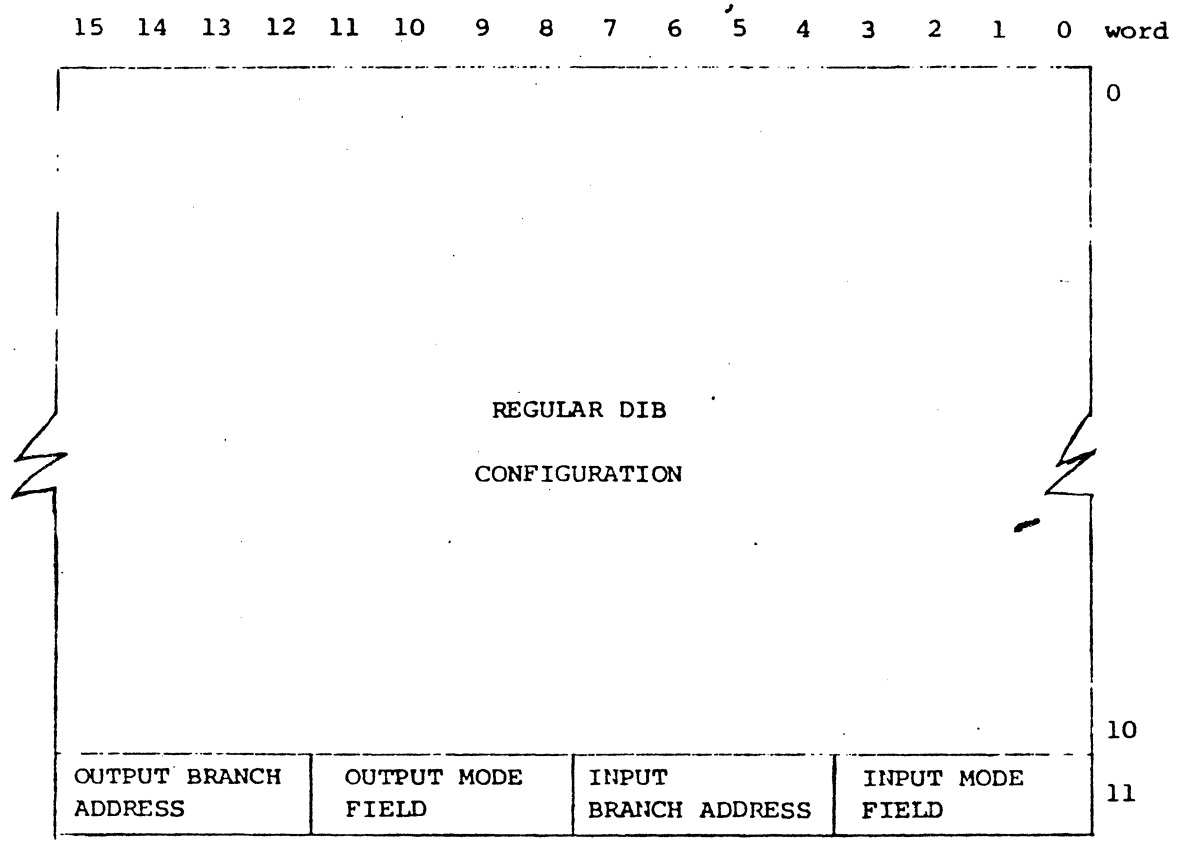
NOTE

Word 10 is used differently by the Magnetic Tape Intelligent Cable DIB. See the additional DIB configurations section.

4.3 ADDITIONAL DIB CONFIGURATIONS - UP TO 18 WORDS

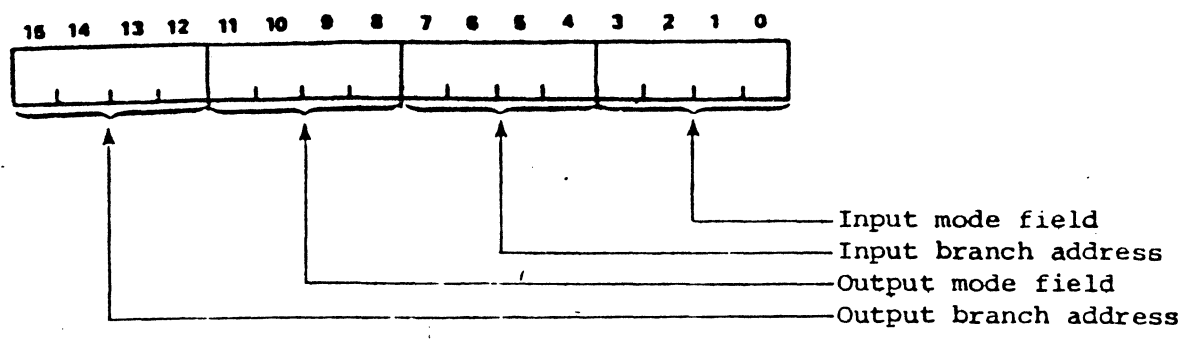
The following DIB configurations require additional words which are not required in the regular DIB configuration.

4.3.1 Distributed I/O DIB



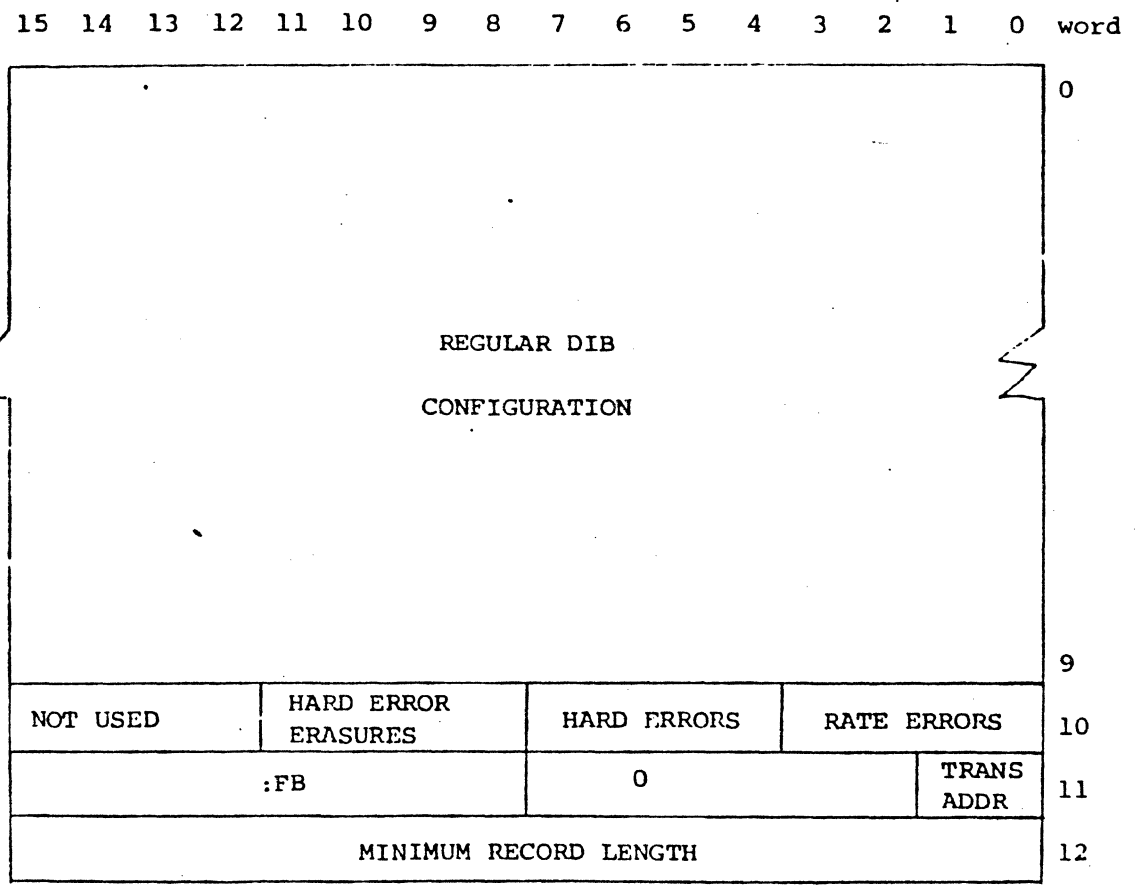
Word 11

DIO command fields. This word uses the following format:



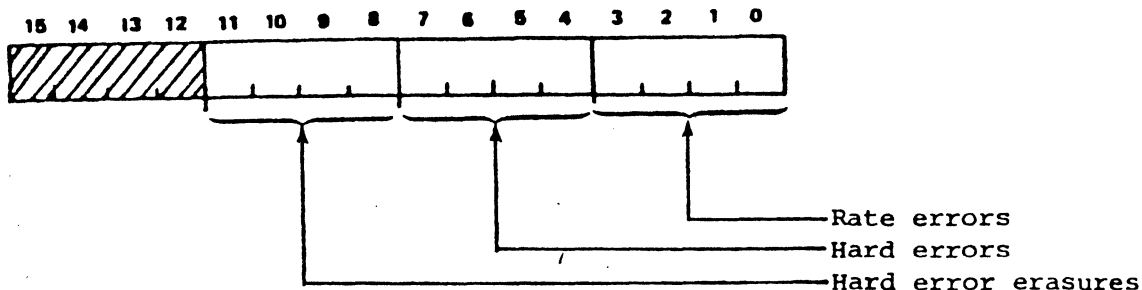
- Bits 15-12. Branch Address Field of DIO Command Word for output.
- Bits 11-8. Mode Field of DIO Command Word for output.
- Bits 7-4. Branch Address field of DIO Command Word for input.
- Bits 3-0. Mode Field of DIO Command Word for input.

4.3.2 Magnetic Tape Intelligent Cable DIB



Word 10

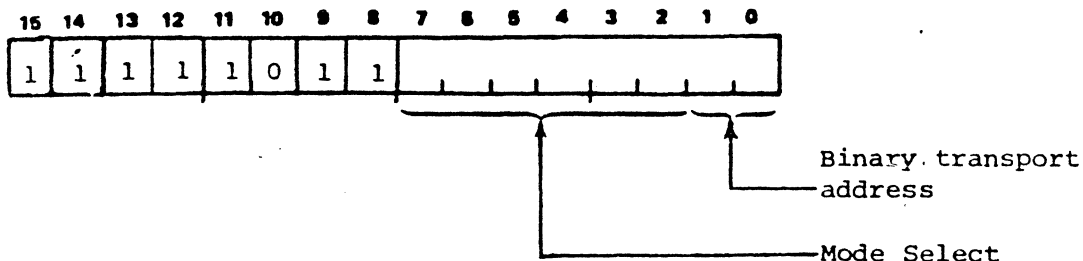
Three MTIC error counters. This word uses the following format:



Bits 11-8. The number of erasures due to hard errors.
 Bits 7-4. The number of hard errors. This counter is reset whenever an erasure occurs.
 Bits 3-0. The number of rate errors. This counter is reset whenever a hard error occurs.

Word 11

Basic mode select word. This word uses the following format:



Examples - :FB00 indicates transport 0
 :FB03 indicates transport 3

Word 12

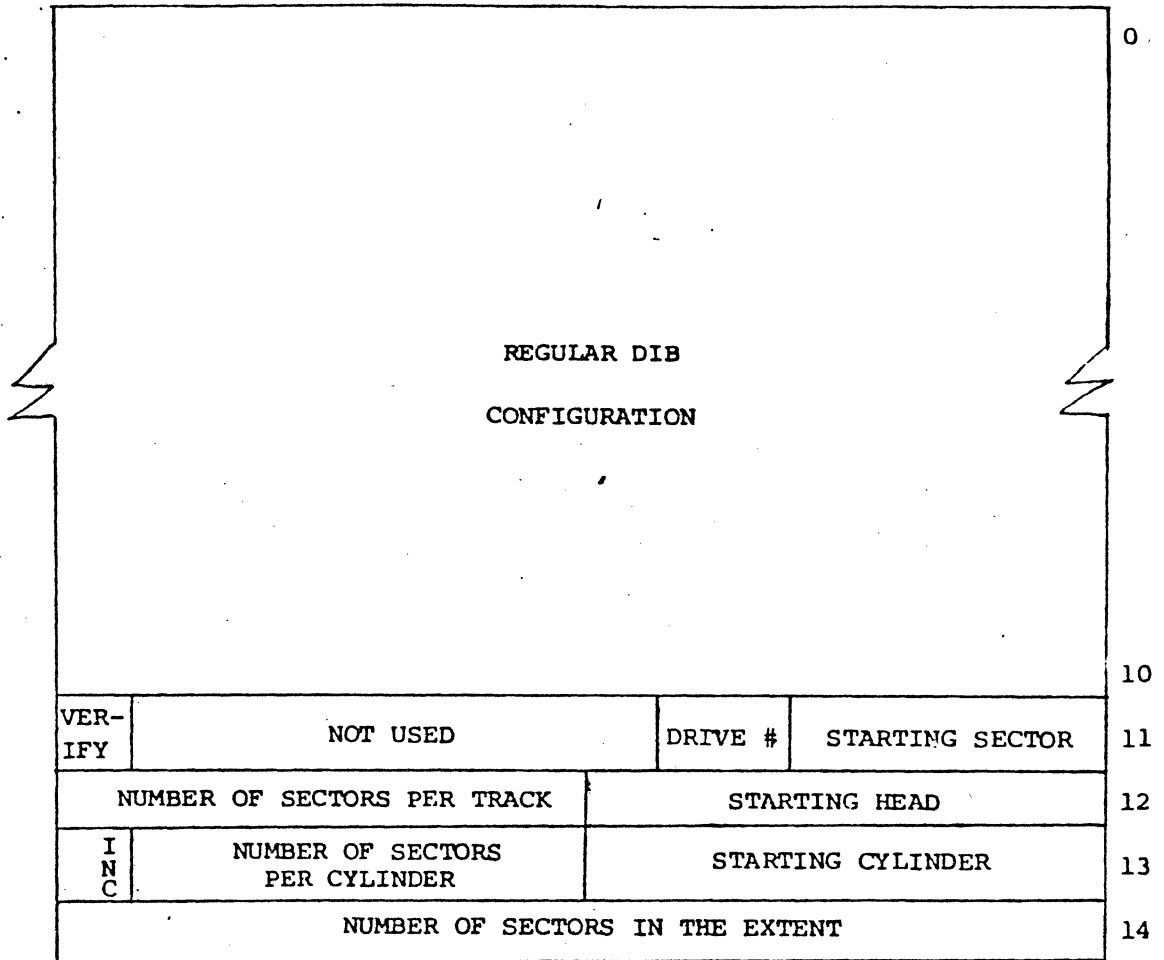
This word contains the minimum record length (in bytes). Records smaller than this byte count are considered noise records. CIB word 35 must specify the word address of a buffer with a size greater than or equal to the minimum record length. The standard minimum record length for the MTIC handler is 12 bytes.

Write requests with a byte count less than the minimum record length will have additional characters appended to the record until the byte count equals the contents of word 12. Blanks are appended to ASCII records and zeros are appended to Binary records.

Read request will return only the number of characters requested.

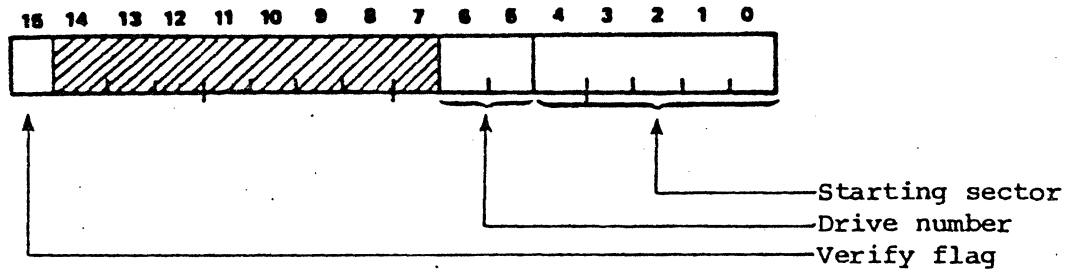
4.3.3 Disk DIB

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 word



Word 11

This word uses the following format:



Bit 15. Verify flag

If this bit is set, a verify operation will be performed after each write. Two additional attempts will be made to re-write the record before the error bit in the status is set.

Bits 14-7. Not used.



Bits 6-5. Drive Number

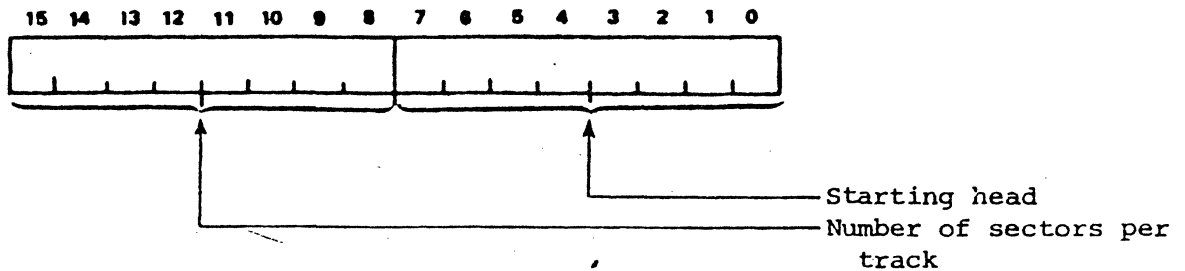
This is the number of the drive attached to the controller. Its range is from 0 through 3 inclusive.

Bits 4-0. Starting Sector

This is the sector number where the extent is to start. Its range is from 0 through the number of physical sectors -1 per track.

Word 12

This word uses the following format:



Bits 15-8. Number of Sectors per Track

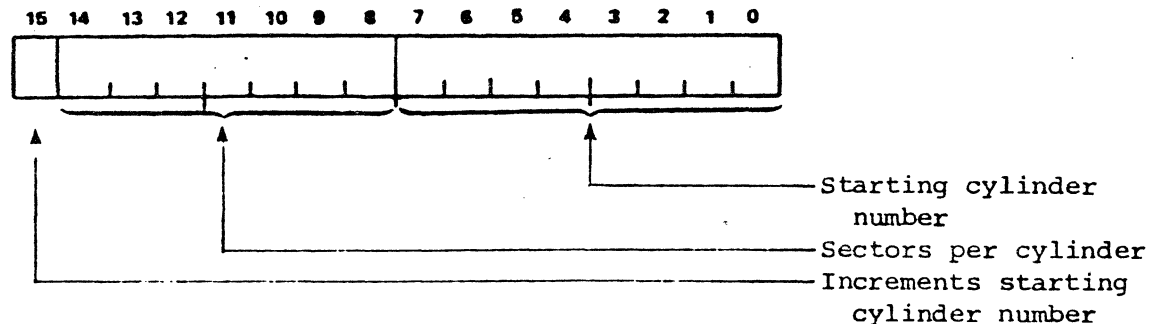
This number defines the number of sectors on each track that this extent is to occupy. The sum of the number and the starting sector may not exceed the physical number of sectors per track.

Bits 7-0. Starting Head

This number defines the starting head number of the extent. Its range is from 0 through the number of heads -1 on the disk drive.

rd 13

This word uses the following format:



Bit 15. If this bit is set, the contents of bits 7-0 are incremented by 256.

Bits 14-8. Number of Sectors per Cylinder

This number equals the number of sectors per cylinder times the number of read/write heads. This is the maximum value of any extent.

Bits 7-0. Starting cylinder

This number is the first cylinder that the extent is to occupy.



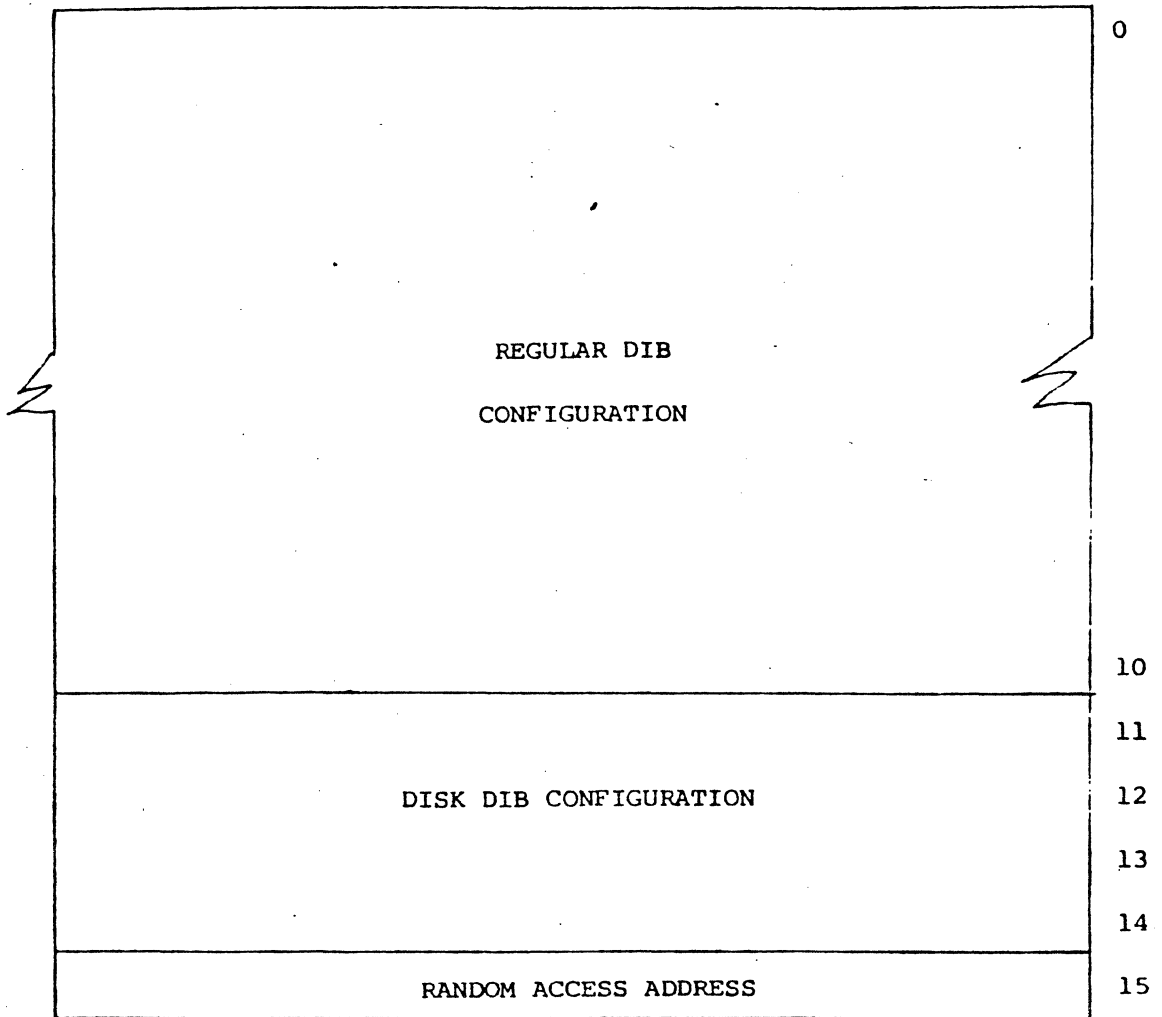
Word 14

Number of Sectors in the Extent

This number is used to detect the end of the extent and to allow the IOX disk handler to set the end-of-device status if access to the last sector of the extent or beyond is requested. This number is equal to the number of cylinders times the number of heads per cylinder times the number of sectors per track.

4.3.4 Fortran Disk DIB

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 word



Word 15

Random Access Address

This word provides a location other than the user's IOB to store the record number.

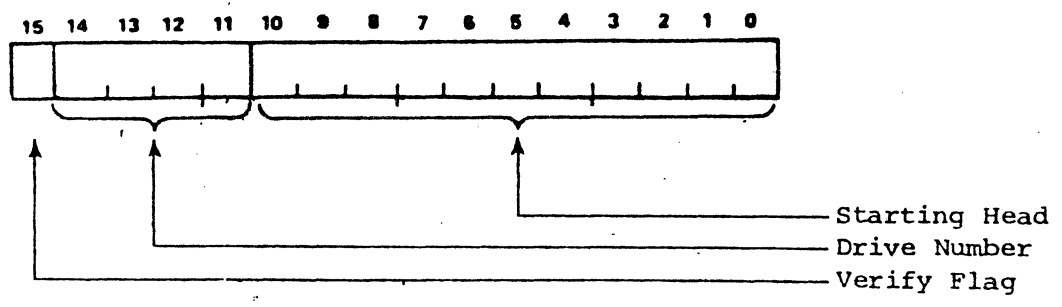
3.5 Storage Module Disk DIB (Fortran and Non-Fortran)

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 word

																10
VER- IFY	DRIVE NUMBER														11	
STARTING SECTOR NUMBER																12
STARTING CYLINDER																13
NUMBER OF SECTORS																14
RECORD NUMBER (FORTRAN ONLY)																15
NUMBER OF SECTORS PER TRACK																16
NUMBER OF SECTORS PER CYLINDER																17

Word 11

This word uses the following format:



**Bit 15. Verify Flag**

If this bit is set, a verify operation will be performed after each write. Two additional attempts will be made to re-write the record before the error bit in the status is set.

Bits 14-11. Drive Number

This is the number of the drive attached to the controller. Its range is from 0 to 15 inclusive.

Bits 10-0. Starting Head

This number defines the starting head number of the extent. Its range is from 0 through the number of heads -1 on the disk drive.

Word 12

Bits 15-0. Starting Sector

This is the sector number where the extent is to start. Its range is from 0 through the number of physical sectors -1 per track.

Word 13

Bits 15-0. Starting Cylinder

This number defines the starting cylinder number of the extent. Its range is from 0 through the number of physical cylinders -1 on the disk drive.

Word 14

Bits 15-0. Number of Sectors

This number is used to detect the end of the extent to allow the IOX Storage Module handler to set the end-of-device status if access to the last sector of the extent or beyond is requested. This number is equal to the number of heads per cylinder times the number of sectors per track.

Word 15

Bits 15-0. Fortran Record Number

This word is only required for Fortran to provide a location other than the user's IOB to store the record number.

Word 16

Bits 15-0. Number of Sectors per Track

This number defines the number of sectors on each track that this extent is to occupy. The sum of this number and the starting sector number may not exceed the physical number of sectors per track.

Word 17

Bits 15-0. Number of Sectors per Cylinder

This number defines the number of sectors on each cylinder that this extent is to occupy. It is numerically equal to the number of sectors per track times the number of heads per cylinder. Note that the number of heads per cylinder plus the starting head number must not exceed the physical number of heads on the drive.



SAMPLE DISK DIB

This DIB defines an extent on disk unit 0 of cylinders 0 through 10, heads 2 and 3, sectors 0-11; that is, all sectors of the first eleven cylinders of the removable platter:

NAM	D:DKXX	DIB NAM
EXTR	C:DKO	CIB Reference
* D:DKXX	EQU	\$
	CHAN	X::
	DATA	C:DKO
	DATA	0
	DATA	0
	DATA	0
	DATA	:0011
	DATA	'DK, 'XX'
	DATA	0
	DATA	0
	DATA	0
	DATA	0
	DATA	0
	DATA	0
	DATA	:C02
	DATA	:1800
	DATA	:108

Chain link to other DIB's
CIB Address
IOX temp cell
Coordination number
DSW: Direct access Read/Write
Device name
EOB delay (none required)
FC's, flags (none required)
Max record size
Error count
Drive 0, starting sector 0
Sectors per track = 12
Starting head number = 2
Sectors per cylinder = 24
Starting cylinder number = 0
Sectors per extent = 264
(24 sectors x 11 cylinders)

CONTROLLER INFORMATION BLOCK (CIB) - 38 WORDS (47 WORDS FOR STORAGE MODULE DISK)

The CIB is used for storing and/or transferring information between IOX and the I/O handler. Words 15-19 must contain the described information upon initial entry to IOX. Words 22-31 have data stored in them while in IOX. All other words are used by the standard I/O handlers and IOX routines, but may not be required by the user's specially written handler. Figure 4-2 illustrates the CIB configuration.

Each word location and its usage is described below:

- Word 0 Temp cell. Set to zero by the scheduler to be used for beginning of record flag. Set to -1 by IORTN: or SIO:. Set to a number greater than zero by an interrupt.
- Word 1 Temp cell. If the subroutine SIO: or EORST: is called, this word should contain a SEL DA, 7 instruction where DA=the device address of the device being accessed.
- Words 2-11 Temp cells. CIB's for standard I/O handlers contain a calling sequence to the RTX INTQ: routine, which is executed upon an end-of-block interrupt. (See INTQ: description.)
- Word 12 Temp Cell. The special function codes from DIB Word 8 are stored here by SINT:, and used by SIO: in setting up the I/O select instruction sequence.



CONTROLLER INFORMATION BLOCK

Standard Name*

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	word
CBOR	TC FOR SIO:--BEGINNING OF RECORD FLAG																0
CSEL7	TC FOR SIO: OR EORST:--SPL DA, 7																1
	JST INTQ:																2
	DATA \$ CALLING LOC																3
CTMP1	DATA 0 TEMP 1																4
CTMP2	DATA 0 TEMP 2																5
CTMP3	DATA 0 TEMP 3																6
CEBTSK	DATA TASK ADDRESS																7
	DATA PRIORITY FOR QUEUED TASK																8
CNEWA	DATA AREG																9
CNEWX	DATA XREG																10
	DATA P-LOC CIB ADDRESS																11
CFUN	TC FOR SINT: AND SIO:--FUNCT. CODES (DIB wd 8)																12
CCSUM	TC FOR CHECKSUM																13
REOCNT	TC																14
CJTBL	ENTRY POINT TO READ																15
	ENTRY POINT TO WRITE																16
	ENTRY POINT TO POSITION																17
	ENTRY POINT TO FUNCTION																18
CSPLOP	ENTRY POINT TO SPECIAL OPERATION																19
CDEL	TC FOR SIO:--WATCHDOG TIMER INSTR.																20
CINTR	TC FOR SINT: AND SIO:--DEVICE WORD INTERRUPT ADDR.																21
EXCESS	TC FOR IOX--PROMPT CHARS (IOB wd 8)																22
CEOF	TC FOR IOX--EOF, AND FOR MAG TAPE--RETRY CTR																23
CIOB	TC FOR IOX--IOB ADDR																24
COP	TC FOR IOX SCHED--OP CODE AND STATUS (IOB wd 5)																25
CRCNT	TC FOR IOX SCHED--REQUESTED COUNT (IOB wd 6)																26
CBUFF	TC FOR IOX SCHED--BUFFER ADDR (IOB wd 7)																27
CTCNT	TC FOR IOX--ACTUAL BYTE COUNT																28
CDA A	TC FOR IOX SCHED--DIRECT ACCESS ADDR (IOB wd 9)																29
CDIB	TC FOR IOX SCHED--DIB ADDR AND BUSY FLAG																30
CFUN1	TC FOR IOX SCHED--FUNCT. CODES (DIB wd 8)																31
STATUS	TC FOR EOR:--STATUS																32
CRTN	TC FOR SIO: AND WAIT:--RETURN ADDRESSES																33
CDCHN	DATA \$+1 POINTER TO BYTE COUNT																34
CDCHN1	DATA 1 BYTE COUNT																35
CDCHN2	DATA \$+1 BUFFER ADDRESS																36
CDCHN3	DATA 0 1 CHAR INPUT BUFF																37

NOTE:
TC = Temp Cell

*refer to the I/O Handler listing at the end of Section 3.
Figure 4-2. CIB Configuration



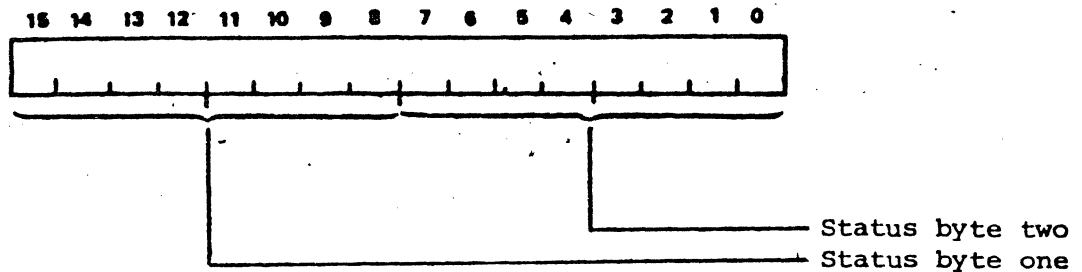
- Word 13 Temp Cell. Used by the standard I/O handlers for a checksum storage cell.
- Word 14 Temp Cell.
- Words 15-18 IOX requires these words to be set up as a jump table to various entry points in the I/O handler, as follows:
- | | |
|---------|--------------------------|
| Word 15 | Entry point to READ. |
| Word 16 | Entry point to WRITE. |
| Word 17 | Entry point to POSITION. |
| Word 18 | Entry point to FUNCTION. |
- If any of the above functions have no meaning to the handler, the corresponding cell (Words 15-18) should be zero.
- Word 19 Entry point to SPECIAL OPERATION. If the handler does not perform a special operation, this word should be zero.
- Word 20 Temp Cell. This cell is assumed by SIO: to be an instruction (e.g., LLA or NOP) to be used in calculating the watchdog timer. (See SIO: routine description.)
- Word 21 Temp Cell. SINT: and SIO: routines expect this word to contain the device's word interrupt address.
- Word 22 Temp Cell. Used by IOX to store prompt characters from IOB Word 8, if any.
- Word 23 Temp Cell. Used by IOX character handler in checking for end of file, and by the magnetic tape handler as a retry counter.
- Word 24 Temp Cell. IOX scheduler routine stores IOB address here.
- Word 25 Temp Cell. IOX scheduler routine stores IOB Word 5 (op code and status) here.
- Word 26 Temp Cell. IOX scheduler routine stores IOB Word 6 (requested count) here.
- Word 27 Temp Cell. IOX scheduler routine stores IOB Word 7 (buffer address) here.
- Word 28 Temp Cell. Used by IOX routines to count actual byte transfers.
- Word 29 Temp Cell. IOX scheduler routine stores IOB Word 9 (direct access address) here, if any.
- Word 30 Temp Cell. IOX scheduler routine stores DIB address here, and later uses it for a busy flag. (If non-zero, IOX assumes the device to be busy.)
- Word 31 Temp Cell. IOX scheduler routine stores DIB Word 8 (function codes) here.

- Word 32 Temp Cell. Used by FOR: routine for storage of status.
- Word 33 Temp Cell. Used by the SIO: and WAIT: routines to store their return addresses.
- Words 34-37 Temp Cells. Used by the standard I/O handlers as a byte count/buffer address/1-character buffer sequence for 1-character I/O calls to SIO:. (See FETCH: description.)

NOTE

MTIC Handlers use CIB words 34 and 35 in the following manner:

Word 34 Temp Cell. Used to store the MTIC Hardware Status. This word uses the following format:



Word 35 Minimum Record Length Buffer Address. This word contains a word address of a buffer with a size greater than or equal to DIB word 12.

4.6 STANDARD CIB NAMES

The following table shows the CIB names for all devices for which standard and non-standard handlers exist within IOX. The label is to be used as the second word of the associated DIB(s). (A table of DIB names is shown in section 2 - Unit Assignment Table description.)

	<u>Non-DIO</u>	<u>Fortran Non-DIO</u>	<u>DIO</u>	<u>Fortran DIO</u>
Teletype	C:TYO	C:TYF	C:TYD	C:TYFD
CRT	C:TYO	C:TYF	C:TVD	--
High Speed Paper Tape Reader	C:PRO	C:PRO	C:PRD	C:PRD
High Speed Paper Tape Punch	C:PP0	C:PP0	C:PPD	C:PPD
Line Printer	C:LPO	C:LPF	C:LPD	C:LPPD
Card Reader	C:CRO	C:CRO	--	--
Disk	C:DKO	C:DKF	--	--
Storage Module Disk	C:SMO	C:SMFO	--	--
Floppy Disk	C:FDO	C:FDO	--	--
Magnetic Tape	C:MT0	C:MT0	C:MCO	--
IEEE Intelligent Cable	--	--	C:IEOD	--



SECTION 5

FILE MANAGER

The File Manager provides directory and data management for file-oriented devices. The devices supported by the File Manager are the moving head disk and the floppy disk. It operates as a driver working in conjunction with RTX/IOX. By using the File Manager, an application program may communicate directly with the data files by name, independent of the physical medium storing the file.

All requests for file access are made through IOX (IO:) using Logical Units (LUNs). The File Manager calls standard IOX device drivers using Logical Units for the required physical I/O. LUN assignments for files as well as LUNs for use by the File Manager for physical I/O are made in the Unit Assignment Table (UAT). (See Section 2, IOB and UAT Organization.) File information (name, file attributes, etc.) is contained in a Device Information Block (DIB) for that file. The file DIB is not to be confused with the device DIB described in Section 4 although the first ten words are the same. The file DIB is described in this section.

The File Manager requires that all File-oriented devices be labeled prior to use. This involves the creation of a Volume Table of Contents (VTOC) and directories on each individual unit to allow later file processing by name. Do not confuse "labeling" with the "formatting" of disk packs; the latter must be done with stand-alone programs before labeling. The RTX File Label Utility (93324-40A1 and -41A1) is a stand-alone program for labeling file-oriented devices. The device labeled using this utility is compatible with the Computer Automation OS file format.

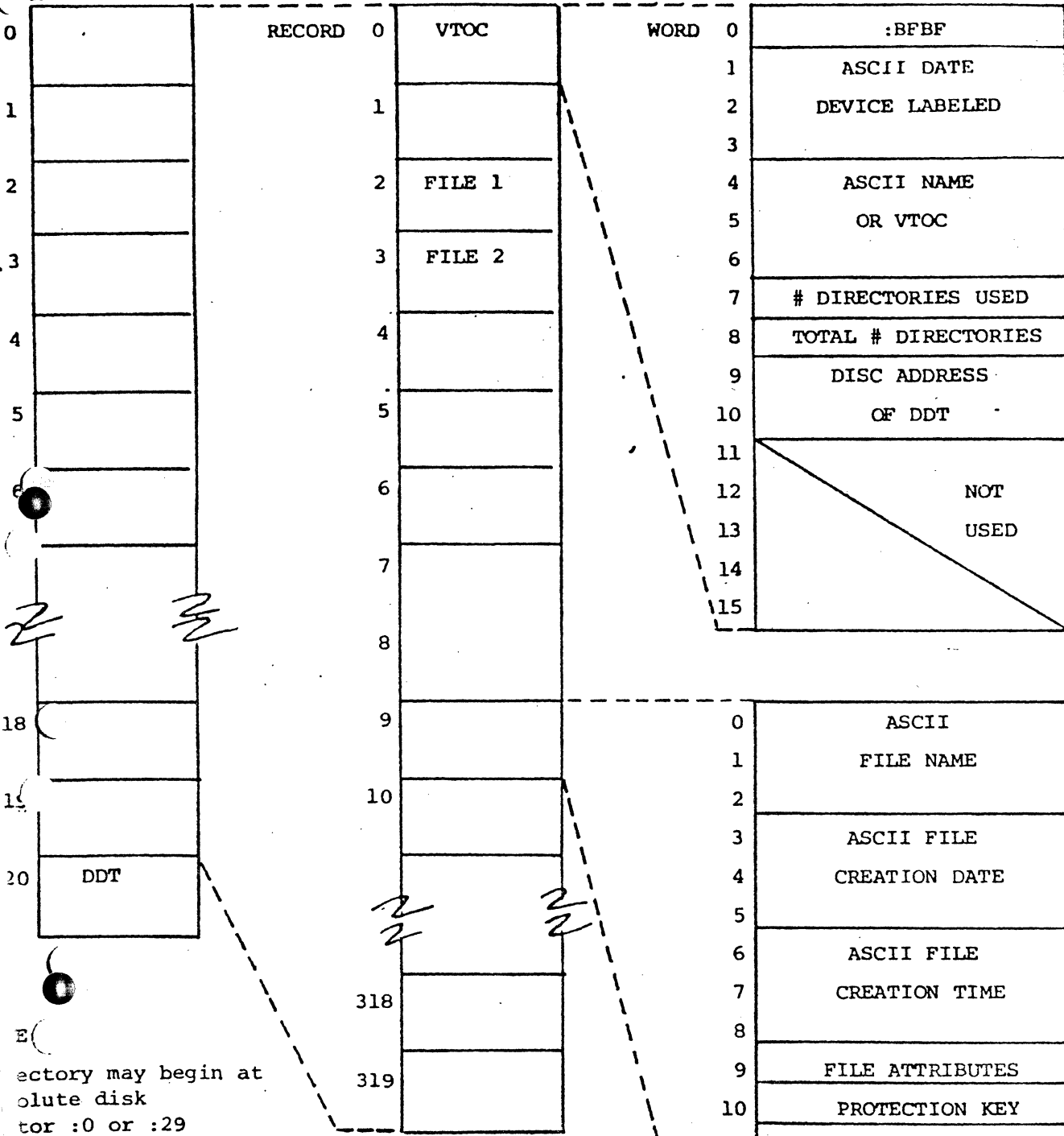
5.1 FILE ORGANIZATION

File organization in the File Manager is compatible with the Computer Automation OS file format. Any file-oriented device accessed through the File Manager must contain a directory. The directory describes by name all data files which reside on the device. The physical medium containing a directory and files is called a Volume. The first entry in the directory is the Volume Table of Contents (VTOC). This entry contains information for the File Manager as well as volume name and creation date. The remainder of the directory is segmented into file description entries, one for each file on that volume. An entry contains the file name, creation date and time, and File Manager information such as record size, block size and file length. See Figure 5-1 for directory structure. Figure 5-2, the Disk Descriptor Table, defines the disk partition limits.

For disk volumes, multiple new file writes are supported through disk partitioning. The disk is divided into as many as eight partitions, each of which may have a new file open. If a file extends past the end of a partition, the file is linked to the next available partition. File linkage is supported for forward sequential reading or for positioning only (in either direction). Any number of old files may be open. (See Figure 5-3 for file linkage.)

RELATIVE
SECTOR

VTOCRD: SEQUENTIAL FILE
(blocked 32 bytes per record, 510 bytes per block)



VTOC

FILE DIRECTORY ENTRY

FILE LINKED FLAG

ATTRIBUTES:

- Bit 15 - 1 - Deleted
- BIT 14 - 1 = Random, 0 = Sequential
- BIT 13 - 1 = Blocked, 0 = Unblocked

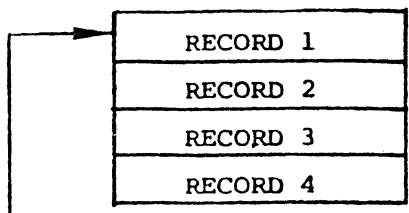
Figure 5-1. Disk Directory Structure
II/5-2



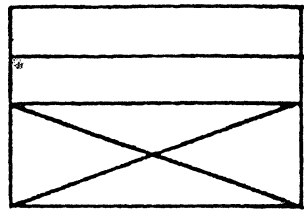
WORD 0	LAST PHYSICAL SECTOR	1 PARTITION
1	NUMBER IN PARTITION	
2		2
3		NOTE:
4		Entry is zero if
5		that partition
6		does not exist
7		
8		4
9		5
10		6
11		7
12		8
13		
14		1 PARTITION
15		
16	NEXT RELATIVE RECORD (ACTUAL)	2
17	NEXT RELATIVE RECORD (WORKING)	3
18		4
19		5
20		6
21		7
22		8
23		
24		
25		
26		
27		
28		
29		
30		
31		

Figure 5-2. Disk Description Table (DDT) in Volume Table of Contents

PARTITION #1

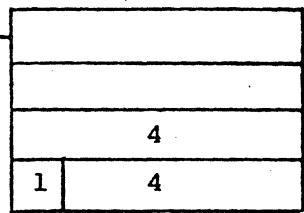


NOTE: Each Block Represents a Single Disk Record

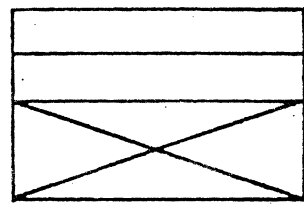
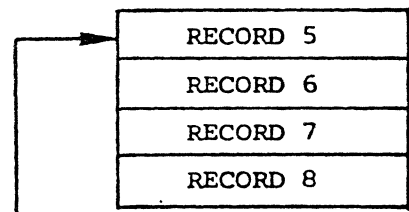


FORWARD LINK:
RECORD NUMBER (2 WORDS)

PARTITION #2



LINK BACK POINTER:
RECORD NUMBER (2 WORDS)
PREVIOUS RECORD COUNT
NEW RECORD COUNT AND
LINK AGAIN FLAG



PARTITION #3

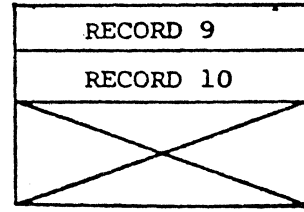
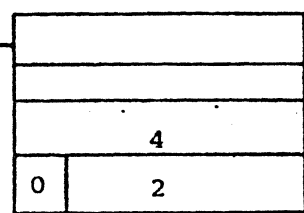


Figure 5-3. Disk File Linkage
II/5-4



5.1.1 Sequential File Access

Sequential file processing is available to the user on the moving head disk and the floppy disk. Sequential files are uniquely ordered by the File Manager: Given logical record N, the next READ request will always return logical record N+1. A READ or WRITE operation automatically advances the file to the next logical record. However, records may be accessed out of order by using the POSITION operation.

The File Manager provides automatic blocking and deblocking of logical records under sequential access. All I/O requests access a single logical record whose position in the physical record is controlled by the File Manager and need not be known by the user.

If the data security bit is set in the DIB, every sequential WRITE operation on that file will cause a directory update on the disk.

For blocked files, the user must provide a record buffer and a blocking buffer. The size and address of each is in the appropriate DIB and IOB. The record buffer may be smaller than the file record size; however, the blocking buffer must be the block size plus two bytes.

Only a record buffer is required for unblocked files. The record buffer may be smaller than the file record size. The user MUST reserve a word (two bytes) at address BUFFER -1 that is required for use by the File Manager.

Random Access

With the File Manager, random access file processing is available only for disk devices. Random files are accessed by physical records; automatic blocking/deblocking is not provided. A random file must reside within a single partition. The number of data bytes contained in each record is fixed at 512. The medium-capacity disk sector size is 512 bytes. When using a floppy disk, four sectors are used for each random file record; each sector has 128 bytes.

Although the record size of a random file is fixed, any number of bytes may be read or written. The specified record number is relative to the beginning of the file.

NOTE

The record number is used to test for end-of-file. If more than 512 bytes are written, the sector(s) beyond the end-of-file will be destroyed.

To access a file in the random mode, the file must have been created as a random file. When a new file is opened with the random file type bit set in the DIB, a random file is created. When closed, the file size is equal to the largest relative record number accessed +1.

5.1.2 File Opening and Closing

The File Manager provides automatic file opening. On the first access (read, write, position, function) of the file, the File Manager will attempt to open the file. If the file name is found in the directory, the open and first access is completed. If the file name is not found, a new file is created. When creating a new file, the



Partition number for placement of the file may be specified in the DIB. If not supplied (zero), the File Manager will use the partition having the largest unused space. Position to absolute file -1 to close the file.

5.1.3 File Positioning

File positioning is provided for use with sequential files. It allows the user to access logical records out of sequence. There are four basic types of positioning. With each type of positioning a count is specified by the user in the Input/Output Block (IOB word 6). (The IOB is defined in Section 2.)

Note that counting of records or file marks begins at zero. See Figure 5-4 for examples of sequential file positioning.

1. Absolute by file mark. The count is the number of file marks to skip from the beginning of the file. The next READ or WRITE will access the logical record following the file mark. Note that a position to absolute zero is equivalent to a rewind. Positioning a file to absolute -1 will close the file. If the count exceeds the number of file marks in the file, an "end-of-media" status is returned with the file positioned after the last logical record.
2. Absolute by logical record. The count is the number of logical records to skip from the beginning of the file (the count must be positive). If a file mark is encountered before the count is exhausted, a "file-mark-found" status is returned and the file is left positioned at the file mark. If the end-of-file is encountered before the count is exhausted, an "end-of-file" status is returned and the file is left positioned after the last logical record.
3. Relative by file marks. The count is the number of file marks to skip from the current file position. A positive count means skip forward; a negative count means skip backwards. While skipping forward, if the end-of-file is encountered, and "end-of-file" status is returned and the file is left positioned after the last logical record. In like manner, when skipping backward, a "beginning-of-file" status is returned and the file is positioned at the first logical record.
4. Relative by logical record. The count is the number of logical records to skip from the current file position. While skipping forward, if a file mark is encountered, a "file-mark-found" status is returned and the file is positioned at the file mark.

For backwards skips, if a file mark is found, a "file-mark-found" status is returned and the file is positioned after the file mark. As with relative positioning by file marks, the File Manager will not allow the position to go beyond the beginning and end of file limits.

With a normal completion, the actual number of records/file marks skipped is returned to the user in IOB word 8. For an error completion, the count returned is the number successfully skipped when the error occurred. For a retry, the requested count should be set to the REQUESTED count in the IOB minus the ACTUAL count.



5.1.4 File Functions

The File Manager provides the functions described below. They are set by the user in the IOB (see Section 2).

Write File Mark

This function writes a sequential record (blocked or unblocked) that contains a :80 in the first byte. When read, this record will cause a file-mark-found status to be returned. Note that this is a data separator, not an end-of-file.

Delete File

This function sets the file-deleted bit in the file DIB and in the directory when the file is closed. Note that this does not free the space on the file device; it only enables a new file to be created with the same name.

Update Directory

This function causes the directory to be updated with the current end of file. This function is valid only for new files. This enables the user to secure the data without performing a close on the file.

TE: The number indicates the count supplied by the user.

ABSOLUTE POSITIONING

RELATIVE POSITIONING

FILE MARK OR RECORD -1 → CLOSE FILE

FILE MARK OR RECORD 0 →

← BEGINNING OF DEVICE

FILE MARK -100

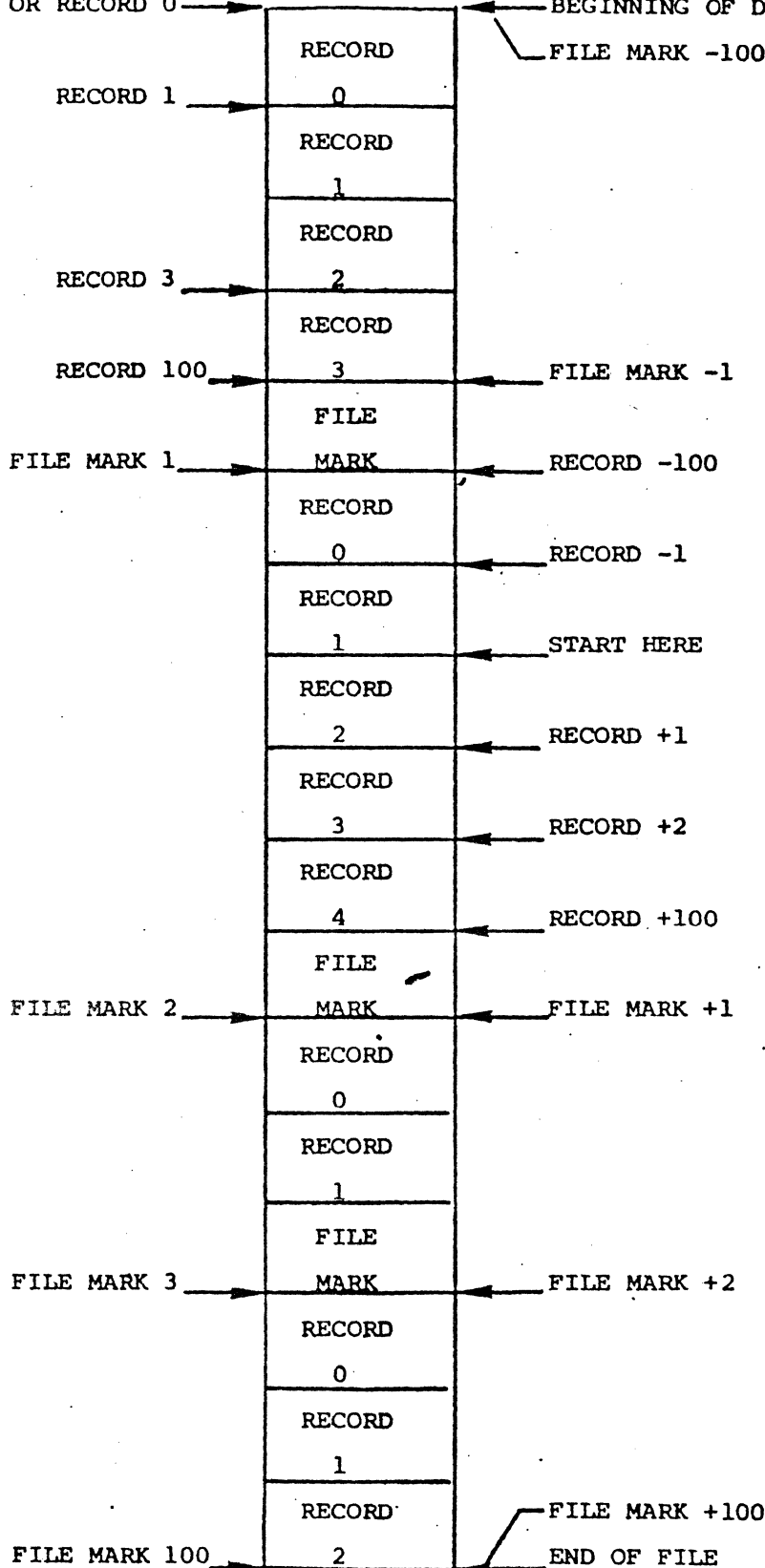


Figure 5-4. Sequential File Positioning Examples



5.2 TABLE ORGANIZATION

The File Manager may be considered as a "dummy" IOX driver in that it is a "data" driver as opposed to a device driver. The File Manager is only concerned with the data contained on the device and not the device itself. Since the File Manager is independent of the file device, it calls a standard IOX device driver to access data on the device. These calls are made to IO: using the logical units associated with the device.

Since the File Manager operates under IOX as a driver, it requires the same type driver tables (i.e. DIBs and CIBs). If the File Manager was equated to a device driver, then a VTOC (directory) would be equivalent to a device controller, and a file would be equivalent to a device unit. The File Manager requires that one CIB for each VTOC, and one DIB for each file be concurrently active (open).

A device containing a VTOC to be accessed by the File Manager must have a unique logical unit associated with it. This logical unit is contained in the File Manager CIB for that VTOC and is used to access the device.

Each File Manager DIB must have a logical unit associated with it. This logical unit is used by the user to access the file described by the DIB.

Logical unit associations are made in the Unit Assignment Table (UAT). A description of the UAT, as well as of the Input/Output Block (IOB) that contains the LUN, is given in Section 2.

Figure 5-5 gives an example of a table configuration. In this example, the file device is a moving head disk with two platters (unit 0 and unit 1). Each unit contains an independent Volume Table of Contents (VTOC) and file directory for that unit.

The standard IOX moving head disk driver requires one controller information block (CIB) C:DKO and two device information blocks (DIBs) D:DK00 and D:DK01 for disk units 0 and 1 respectively.

The File Manager requires two CIBs, C:FM0 and C:FM1, for VTOC 1 and VTOC 2, respectively. Since three files are to be active (open) concurrently, three DIBs are required: D:FM00 for FILE 1, D:FM01 for FILE 2 and D:FM02 for FILE 3.

Each file device (VTOC) has a logical unit associated with it which is used by the File Manager to access the device (LUN X for VTOC 1 and LUN Y for VTOC 2).

The user accesses the files through a standard IOX call to IO: using the logical unit associated with the file DIB. (LUN A for FILE 1, LUN B for FILE 2 and LUN C for FILE 3.)

2.1 File Device Information Block (DIB)

The first ten words of the Device Information Block (DIB) have essentially the same functions for the File Manager as they have for IOX. These standard functions are described in Section 4, DIB and CIB Descriptions. The functions for words 10 through 19 are given below. (Refer to Figure 5-6.)

Words 0-6 Standard for IOX.

Words 7-9 Standard for IOX, but must be set to zero for File Manager.

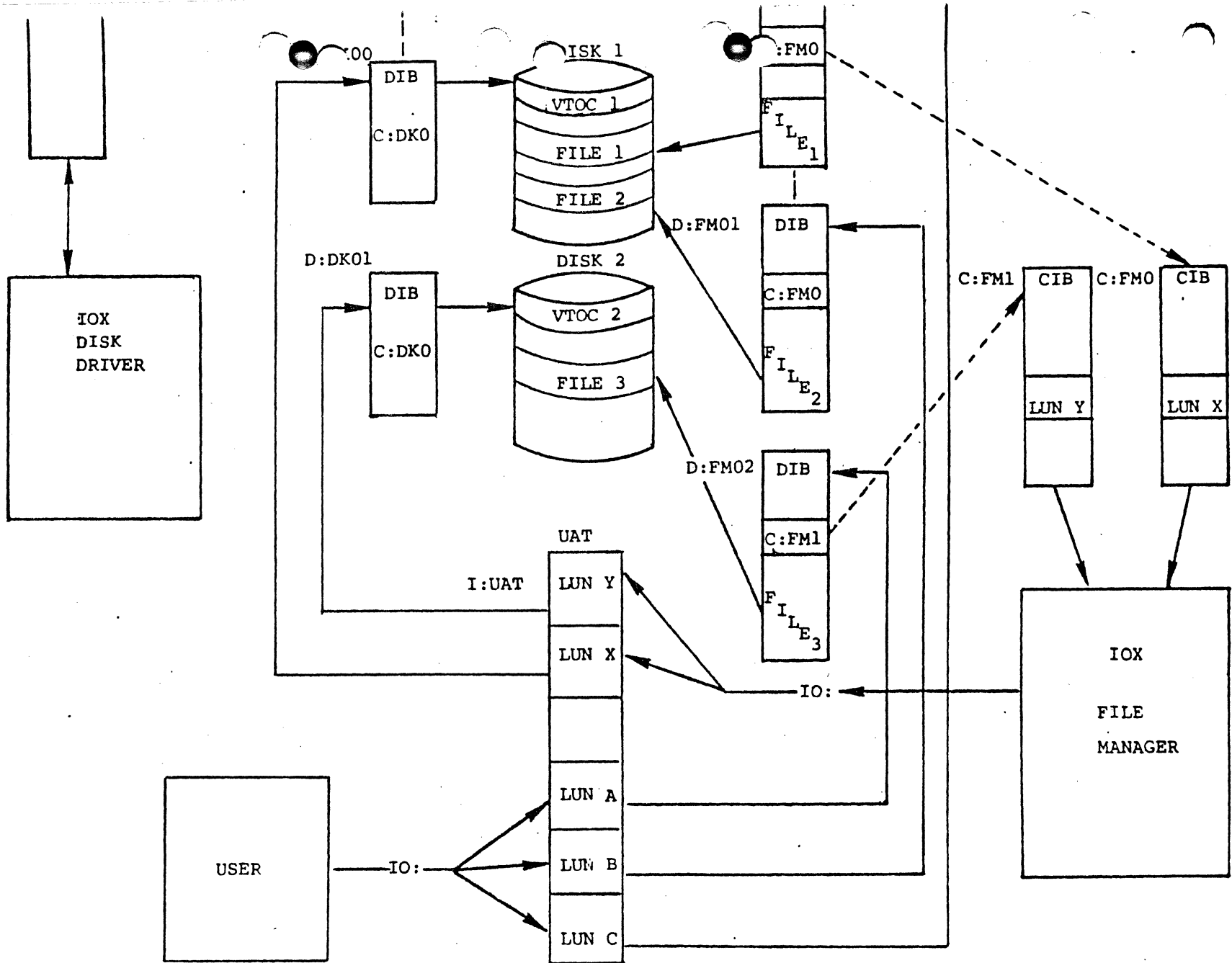


Figure 5-5. Table Organization





0	STANDARD FOR IOX	
9		
10	PHYSICAL I/O ERROR STATUS	DHST
11	FILE STATUS WORD	DFST
12		DFNAM
13	FILE NAME	
14		
15	RELATIVE RECORD NUMBER	DRRN
16	ABSOLUTE RECORD NUMBER	DARN
17	RECORD SIZE	DRS
18	BLOCK SIZE	DBKS
19	PHYSICAL RECORDS PER BLOCK	DPRB
20	PHYSICAL RECORD NUMBER	DPRN
21	TOTAL RECORDS	DTREC
22	DIRECTORY ENTRY NUMBER	DDEN
23	CURRENT BLOCK ADDRESS	DCBA
24	BLOCKING BUFFER ADDRESS	DBBA
25	LAST PHYSICAL RECORD	DLPR
26	COMPLETION STATUS	DCST

Figure 5-6. DIB Definition when Used with the File Manager

- Word 10 Physical I/O error status. The status (word 5) of the physical I/O IOB is stored here after each operation.
- Word 11 File Status word. For old files, all bits are supplied by the File Manager from the directory; therefore, all bits of word 11 are initialized to zero. When creating a new file, those bits flagged with an asterisk (*) must be supplied by the user before the first access. The data security bit may be modified at any time to enable or disable this function. After the first access of a file (new or old), if the file delete bit (15) is set, the file will be deleted when the file is closed. Bits 15-13 correspond to the file attribute bits in the directory entry and are transferred to the entry when a new file is closed.
- Bit 15. 0 = keep file, 1 = delete file
- Bit 14*. 0 = sequential file, 1 = random file
- Bit 13*. 0 = unblocked records, 1 = blocked records
- Bit 12. 0 = file closed, 1 = file open
- Bit 11. 0 = file open for sequential access
1 = file open for random access
- Bit 10. 0 = old file, 1 = new file
- Bit 9. 0 = current block not modified
1 = current block modified (blocked files only)
- Bit 8. Data security bit. When set the directory is updated after each sequential write (unblocked files) or after a block is written (blocked files).
- Bit 7. 0 = file not linked, 1 = file linked
- Bits 6-4. Reserved for future expansion.
- Bits 3-0*. Partition number. For old files, contains the number. For new files, specifies where the new file is to be created. If zero, the available partition with the greatest unused space is used and its number is stored here.
- Words 12-14* ASCII file name. Supplied by the user.
- Word 15 Relative record number. Relative to the beginning of the current file segment for linked files. With unlinked files, this word is the same as the absolute record number.
- Word 16 Absolute record number. The current file position relative to the beginning of the file. Note that the first record is record zero.
- Word 17* Record size in bytes. Set to 512 for random files. Supplied by user for new files.

* Information supplied by user.



- Word 18* Block size in bytes. Used for blocked files only. Supplied by user for new files.
- Word 19 Number of physical records/block. Contains the number of 512 byte physical records required for a file block (blocked files) or record (unblocked files). Supplied by the file manager. Referred to as "tach ratio" under CAI OS.
- Word 20 Physical record number of first record in file. Supplied by the File Manager.
- Word 21 Total records in the file. For linked files, contains total records in current segment. Supplied by the file manager.
- Word 22 Directory entry number for this file. Supplied by the file manager.
- Word 23 Current block address. Contains the physical record number of the last block read. Supplied by the File Manager.
- Word 24* Blocking buffer address, (Word address, no indirect). Supplied by user when accessing blocked files. Buffer size must be block size plus 2 bytes. Not required for unblocked or random files.
- Word 25 Last physical record in partition. For new files, contains the last available record number. Not used for old files. Supplied by the File Manager.
- Word 26 Completion status. Cleared upon entry into file manager and set when operation is complete. A bit is active when it is set to 1.
- Bit 15. Physical I/O error. An abnormal status was returned from physical I/O. The detail physical I/O status (DHST) word in the DIB contains word 5 of the CIB IOB used for the physical I/O.
- Bit 14. Device not labeled. A valid VTOC identifier was not found. This error can only occur during a file open.
- Bit 13. Directory full. No unused entries are available in the directory for the creation of a new file.
- Bit 12. Directory error. An error was returned from physical I/O during a direction read or write. Detail physical I/O status (DHST) word in DIB is set. This error can occur during a file open, close, or directory update.
- Bit 11. End of Media. The end of a partition was reached during write on a new file. It is valid for both sequential and random access modes.
- Bit 10. Partition(s) busy. The required partition for a new file creation already has a new file currently open, (partition is busy), or required partition is full. If no partition was specified, then all partitions are busy.

* Information supplied by user.



Bit 9-8. Reserved for future expansion.

Bit 7. Access mode error. A sequential access was made on a random file or a random access was made on a sequential file. The access type did not match file type in a new file open.

Bits 6-2. Reserved for future expansion.

Bit 1. Unable to close. Indicates a close was in process when an error occurred (file remains open).

Bit 0. Unable to open. Indicated an open was in progress when an error occurred, (file remains closed).



2 Manager Controller Information Block (CIB)

The Controller Information Block (CIB) is used for storing and/or transferring information between the File Manager and the IOX I/O handler. Words 15-19 must contain the described information upon initial entry to the File Manager (actually to IOX).

Figure 5-7 illustrates the CIB configuration. The functions of each CIB word are described below. Word 0, words 15-19 and words 24-33 are defined the same for the File Handler as they are for IOX.

- Word 0 SIO: beginning of record flag.
- Words 1-10 IOB used by the File Manager for physical I/O; includes user-supplied LUN for the file device (IOB word 4 = CIB word 5). All other data in IOB is supplied by the File Manager. The IOB status word is transferred to the DIB physical I/O error status word after each IO: call.
- Word 11 Number of physical sectors per physical record (supplied by the File Manager).
- Word 12 Physical sector address of Volume Table of Contents (VTOC). Initialize to zero. The File Manager determines the VTOC address (0 or :29) on first open.
- Word 13 Address of Disk Descriptor Table (DDT) (supplied by the File Manager after first open). This is a physical record address.
- Word 14 Open/close buffer address. This word contains the word address (no indirection) of a 256-word buffer supplied by the user. This buffer is used by the File Manager for directory searching during open or close processing.
- Words 15-18 Entry point jump table.
- | | | |
|---------|----------|--------|
| Word 15 | Read | FM:REA |
| Word 16 | Write | FM:WRT |
| Word 17 | Position | FM:POS |
| Word 18 | Function | FM:FUN |
- Word 19 Special operation entry point. Not used; set to zero.
- Word 20 Current direction record number during open, or operation code during position/function processing (supplied by the File Manager).
- Word 21 Number of directory entries used during open, or absolute file position count during position processing (supplied by the File Manager).
- Word 22 Number of directory entries available during open or current file position during position processing. Supplied by the File Manager.



0	SIO: BEGINNING OF RECORD FLAG	DBOR
1	PHYSICAL I/O	
10	IOB	
11	NUMBER OF SECTORS/RECORD	CSPR
12	VTOC ADDRESS	CVTOC
13	DDT ADDRESS	CDDT
14	OPEN/CLOSE BUFFER	COCB
*15	STANDARD	CJTB
*17	FOR	
*18	CIB	
*19		
20	CURRENT DIRECTORY RECORD	CCDR/CPFC
21	DIRECTORY ENTRIES USED	CVND/CAPC
22	DIRECTORY ENTRIES AVAILABLE	CVDA/CCPC
23	PARTITION BUSY FLAGS	CPBS
*24		
*25		
*26		
*27	STANDARD	
*28	FOR	
*29	CIB	
*30		
*31		
*32		
*33		
34	SUBROUTINE LEVEL 5	CR:xxx WHERE
35	RETURN LEVEL 4	
36	ADDRESS LEVEL 3	xxx = SUBROUTINE NAME
37	SAVE LEVEL 2	
38	AREA LEVEL 1	
39	FILE LINKAGE	CLKBF
40	BUFFER	
41		
42		

Figure 5-7. CIB Definition When Used With the File Manager

Word 23 Partition busy flags. Each partition on disk is represented by a single bit. The bit position is equal to the partition number. With a maximum of 8 partitions numbered 1-8, only bits 1-8 are used. Bits 0 and 9-15 are unused. A partition busy flag is set when a new file is open in that partition and cleared when it is closed. Only one new file may be open in any one time. Supplied by the File Manager.

*Words 24-33 Standard CIB definition.

Words 34-38 Subroutine return address save area.

- Word 34 Level 5 subroutines: FM:PS
FM:FN
- Word 35 Level 4 subroutines: FM:RE
FM:WR
FM:OP
FM:CL
- Word 36 Level 3 subroutines: FM:DM
FM:EOF
- Word 37 Level 2 subroutines: FM:WBK
FM:RBK
FM:RLK
FM:RLR
- Word 38 Level 1 subroutines: FM:PIO

Words 39-42 Buffer for processing partition file linkage.



5.3 RTX FILE LABEL UTILITY

The RTX File Label Utility is a stand-alone program for labeling file-type devices. The RTX/IOX File Manager requires that all file-type devices be labeled prior to use. This involves the creation of a Volume Table of Contents (VTOC) and directories on each individual unit to allow later file processing by name. Do not confuse "labeling" with "formatting" of disk packs; the latter must be done with stand-alone programs before labeling. The labeled device is compatible with Computer Automation OS File Format.

5.3.1 Environment

The Label Utility requires an LSI-2 or LSI-3/05 CPU with a minimum of 4K words of memory. The tape numbers (binary paper tape) are 93324-40A1 and -41A1 for LSI-2 and LSI-3/05, respectively.

5.3.2 Program Operation

After loading and executing, the Label Utility halts with P=:0100 and waits for the user to specify TTY I/O type:

1. Standard option board TTY, set Sense switch OFF.
2. DIO TTY, set Sense switch ON.

To continue execution, depress RUN after setting the desired I/O Option. The Label Program will then query the user for its variable information. When responding, certain keys on the keyboard have special functions.

1. Return. The Return key indicates the end of a line of input and causes a carriage return and line feed to be generated.
2. Back arrow (←). The back arrow causes the previous character input to be replaced by the next character typed. Multiple characters may be replaced by typing the appropriate number of back arrows followed by the correction characters.
3. Back arrow (←)/Return. A back arrow followed immediately by Return causes the entire current line to be ignored and replaced by the next line input. The Return causes a carriage return and line feed to be generated.



NOTE

An invalid response to a query will result in the query being repeated.

The Label Utility begins with the first query:

DATE? (MMDDYY)

The user should respond with a Volume Identification. It must consist of one to six characters, normally alphanumeric, although any characters are allowed.

Example: Feb. 4, 1977 would be input as 020477.

TIME? (HHMMSS)

Enter the current time of day (hours, minutes, seconds). This time is saved in the VTOC. This time is NOT incremented by a real-time clock. This is a 24 hour clock.

Example: 1:23 PM would be input at 132300.

VOLUME NAME?

The user should respond with a Volume Identification. It must consist of 1-6 characters, normally alphanumeric, although any characters are allowed.

TYPE AND UNIT NUMBER?

The response is a two-character specifier of the physical device which is to be labeled. The specifiers are:

DEVICE	SPECIFIER
Moving Head Disk, Unit 0	D0
Moving Head Disk, Unit 1	D1
Moving Head Disk, Unit 2	D2
Moving Head Disk, Unit 3	D3
Floppy Disk, Unit 0	F0
Floppy Disk, Unit 1	F1
Floppy Disk, Unit 2	F2
Floppy Disk, Unit 3	F3

DOES xx CONTAIN OS?

If the device to be labeled (xx) contains a copy of the Computer Automation Operation System (OS) the user responds with "YES". Otherwise, the user's response is "NO", causing the next query to be suppressed. OS must be on the device before labeling.



SAVE OS?

If an operating system exists on the Unit and is to be saved, the user responds with "YES", otherwise "NO".

If the device to be labeled is a disk, the next query is:

NUMBER OF PARTITIONS? (1-8)

The user now selects the number of partitions (1-8) into which the disk is to be divided and enters that value. Only the first digit entered is used. The number of partitions selected is the limit to the number of new files which may be open simultaneously (new file creation).

The labeling process then begins. When successfully completed, the following message is output:

LABEL COMPLETE

If the selected device is off-line, not ready, write protected, or otherwise malfunctions during the labeling process, the following message is output:

HARDWARE ERROR
RETRY?

If the user responds with "YES", the program will retry the label process. If the device still fails, the error message is repeated. If the user responds with "NO", the labeling process is aborted and the Program continues with the next query.

LABEL MORE?

The user is offered the option of labeling another device or terminating the process. A "YES" response will cause a restart with the query "VOLUME NAME?". If the user wishes to change the date and time and continue, the response is "NO". A "NO" response will halt the CPU. Depressing RUN will restart the program at the beginning. At this point, a new I/O option may be selected.

NOTE

The restart entry point is :0101. The LSI 3/05 version contains a software console routine for restarting (CNSOL3).

PAGE 0001 03/11/77 15:36:32
MACRO2 (A?) SI= EX:S BO=

EXAMPLE FILE MANAGER APPLICATION
** UNIT ASSIGNMENT TABLE **

```
0003      *
0004      *      UNIT ASSIGNMENT TABLE
0005      *
0006      0001      NEW      EQU      1      NEW FILE LOGICAL UNIT
0007      0002      OLD      EQU      2      OLD FILE LOGICAL UNIT
0008      0003      PTO      EQU      3      PHYSICAL I/O LOGICAL UNIT
0009      SAVE
0010      *
0011      0006      NAM      I:UAT
0012      EXTR      D:FM0
0013      EXTR      D:FM1
0014      EXTR      D:DK01
0015      0000      UATTOP EQU      1
0016      0000 0002      DATA      OLD      OLD FILE LOGICAL UNIT
0017      0001 0000      DATA      D:FM0      OLD FILE DTB
0018      0002 0001      DATA      NEW      NEW FILE LOGICAL UNIT
0019      0003 0000      DATA      D:FM1      NEW FILE DTB
0020      0004 0003      DATA      PTO      PHYSICAL I/O LOGICAL UNIT
0021      0005 0000      DATA      D:DK01      PHYSICAL I/O LOGICAL UNIT
0022      0006 FFF8      I:UAT      DATA      UATTOP-1-2
0023      FND
```

0000 ERRORS
0000 WARNING

II/5-21

PAGE 0001 03/11/77
MACRO2 (AP) ST=FX:S

15:36:32
BN=

EXAMPLE FILE MANAGER APPLICATION
** RTX MAINLINE CODE **

```
0026 *
0027 * RTX MAINLINE CODE
0028 *
0029 0000 NAM MAIN
0030 FXTR TASK
0031 FXTR RTX:
0032 EXTR BEGIN:
0033 EXTR END:
0034 000A N EQU 10 NUMBER OF WORKING BLOCKS
0035 0000 MAIN EQU $
0036 0000 FB07 0008 JST RTX: INITIALIZE RTX
0037 0001 000A DATA N
0038 0002 0008 DATA WKAREA
0039 0003 0800 HLT FRPOP
0040 0004 FB04 0009 JST BEGIN: START TASK
0041 0005 0000 DATA TASK
0042 0006 0064 DATA 100 AT PRIORITY 100
0043 0007 FB02 000A JST END:
0044 0003 LPOOL
000P 0000
000Q 0000
000A 0000
0045 000R 0000 WKAREA RES 5*N,0 RTX WORK AREA
0046 0000 END MAIN

0000 ERRORS
0000 WARNING
```

II/5-22

PAGE 0001 03/11/77
MACR112 (A2) ST= EX:S

15:36:32
RD=

EXAMPLE FILE MANAGER APPLICATION
** APPLICATION PROGRAM **

```
0049 *
0050 * APPLICATION PROGRAM. THIS COPIES EVERY THIRD RECORD FROM
0051 * THE EXISTING FILE "OLD" (BLOCKED 72,510) TO A NEW
0052 * FILE "NEW" (UNBLOCKED, 80 BYTE RECORDS).
0053 *
0054 0000 NAM TASK
0055 0033 NAM DATE:
0056 0036 NAM TIME:
0057 EXTR IO:
0058 EXTR END:
0059 0000 TASK EQU $
0060 0000 C648 LAP 72 RECORD BUFFER SIZE IN BYTES
0061 0001 9A43 0045 STA IOB+6 SET UP BYTE COUNT IN IOB
0062 0002 C602 LAP OLD LOGICAL UNIT OF FILE "OLD"
0063 0003 9A3F 0043 STA IOB+4
0064 0004 C601 LAP :0001 OP CODE FOR SEQUENTIAL READ
0065 0005 9A3E 0044 STA IOB+5
0066 0006 FB32 0039 JST IO: READ A RECORD FROM FILE "OLD"
0067 0007 003F DATA IOB THE FIRST READ WILL OPEN THE FILE
0068 0008 FB31 003A JST END:
0069 0009 F213 0010 JMP CHECK ABNORMAL RETURN, TEST FOR END OF FILE
0070 *
0071 0004 C601 LAP NEW LOGICAL UNIT FOR FILE "NEW"
0072 0008 9A37 0043 STA IOB+4 SET UP IOB
0073 000C C605 LAP :0005 OP CODE FOR SEQUENTIAL WRITE
0074 000D 9A36 0044 STA IOB+5
0075 000F FB2A 0039 JST IO: WRITE THE RECORD TO FILE "NEW"
0076 000F 003F DATA IOB THE FIRST WRITE WILL CREATE A NEW FILE
0077 0010 FB29 003A JST END: AND OPEN IT
0078 0011 F20E 0020 JMP ERROR ABNORMAL RETURN
0079 *
0080 0012 C602 LAP OLD LOGICAL UNIT OF FILE "OLD"
0081 0013 9A2F 0043 STA IOB+4 SET UP IOB
0082 0014 C60A LAP :000A OP CODE FOR POSITION RELATIVE RECORDS
0083 0015 9A2E 0044 STA IOB+5
0084 0016 C602 LAP ? FORWARD RECORD COUNT TO SKIP
```

II/5-23

PAGE 0002 03/11/77
MACRO2 (AP) ST= EX:S

1.30
BQ=

EXAMPLE FILE MANAGER APPLICATION
** APPLICATION PROGRAM **

```
0085 0017 9A2D 0045 STA IOB+6
0086 0018 FB20 0039 JST IO: SKIP TWO RECORDS ON FILE "OLD"
0087 0019 003F DATA IOB
0088 001A FB1F 003A JST END:
0089 001B F201 001D JMP CHECK ABNORMAL RETURN, TEST FOR END OF FILE
0090 001C F61C 0000 JMP TASK GO BACK TO READ ANOTHER RECORD
0091
0092 001D 001D * CHECK EQU $
0093 001D B276 0044 LDA IOB+5 GET JOB COMPLETION STATUS
0094 001E B21C 003B AND =:0200 TEST FOR FILE MARK FOUND OR END OF FILE
0095 001F 3102 0022 JAN DONE YES, COPY COMPLETE
0096 * NO, SOME OTHER ERROR OCCURED
0097 0020 0020 * EPROR EQU $
0098 0020 0800 HLT ERROR HALT
0099 0021 F601 0020 JMP $-1
0100
0101 0022 * DONE EQU $ COPY COMPLETE, CLOSE FILES
0102 *
0103 0022 C701 I AM 1 COUNT = -1
0104 0023 9A21 0045 STA IOB+6 SET UP IOB
0105 0024 C609 LAP :0009 OP CODE FOR POSITION ABSOLUTE FILES
0106 0025 9A1E 0044 STA IOB+5
0107 0026 C602 LAP UID LOGICAL UNIT OF FILE "OLD"
0108 0027 9A18 0043 STA IOB+4
0109 0028 FB10 0039 JST IO: CLOSE READ FILE "OLD"
0110 0029 003F DATA IOB
0111 002A FB0F 003A JST END:
0112 002B F608 0020 JMP ERROR ABNORMAL RETURN
0113 *
0114 002C C601 LAP NEW LOGICAL UNIT OF FILE "NEW"
0115 002D 9A15 0043 STA IOB+4 SET UP IOB
0116 002E FB0A 0039 JST IO: CLOSE WRITE FILE "NEW"
0117 002F 003F DATA IOB
0118 0030 FB09 003A JST END:
0119 0031 F611 0020 JMP ERROR ABNORMAL RETURN
0120 *
```

II/5-24

PAGE 0003 03/11/77 15:36:32
MACRO? (A?) ST= EY:S RO=

EXAMPLE FILE MANAGER APPLICATION
** APPLICATION PROGRAM **

0121 0032 FB07 003A .IST END: ALL DONE
0122 0033 C0C0 DATE: TEXT 'MMDDYY'

0034 C4C4
0035 D9D9
0123 0036 C8C8 TIME: TEXT 'HHMMSS'

0037 C0C0
0038 D3D3
0124 0006 LP001

0039 0000
003A 0000
003B 0200
003C
003D
003E

0125 *
0126 * INPUT OUTPUT BLOCK (TOP)

0127 *
0128 IOB DATA 0,0,0,0,3-1,3-3,3-3,BUFFER,0,0

003F 0000
0040 0000
0041 0000
0042 0000
0043 0000
0044 0000
0045 0000
0046 004A
0047 0000
0048 0000

0129 004A BUFFER EQU \$+1
0130 0049 0000 DATA 0 REQUIRED FOR UNBLOCKED FILES
0131 004A 0000 RES 36,0 72 BYTE RECORD BUFFER
0132 END

0000 ERRORS
0000 WARNING

II/5-25

PAGE 0001 03/11/77
MACRO2 (A?) ST= EX:S

15:32
HO=

EXAMPLE FILE MANAGER APPLICATION
** FILE MANAGER CTB **

0135 0000
0136
0137
0138
0139
0140
0141 0000
0142 0000 0000
0143 0005 0003
0144 0006 0000
0145 000E 002B
0146 000F 0000
0147 0010 0000
0148 0011 0000
0149 0012 0000
0150 0013 0000
0151
0152 002B 0000
0153

NAM C:FMO
EXTR FM:REA
EXTR FM:WRT
EXTR FM:POS
EXTR FM:FUN

*
C:FMO EQU 5 FILE MANAGER CTB
RES 5,0
DATA PTO PHYSICAL I/O LOGICAL UNIT
RES 8,0
DATA OCBUF OPEN/CLOSE BUFFER ADDRESS
DATA FM:REA READ ENTRY POINT
DATA FM:WRT WRITE ENTRY POINT
DATA FM:POS POSITION ENTRY POINT
DATA FM:FUN FUNCTION ENTRY POINT
RES 24,0
*
OCBUF RES 256,0 OPEN/CLOSE BUFFER
END

0000 ERRORS
0000 WARNING

II/5-26

PAGE 0001 03/11/77
MACRO? (A?) ST= EY:S

15:36:32
BN=

EXAMPLE FILE MANAGER APPLICATION
** FILE DTR **

```
0156 *
0157 * OLD FILE DTR. DEVICE STATUS WORD IS SET TO ALLOW FILE READ
0158 * OR POSITION OPERATIONS ONLY.
0159 *
0160 0000 NAM D:FMO
0161 FATH C:FMO
0162 *
0163 0000 D:FMO CHAN X::
0164 0001 0000 DATA C:FMO FILE MANAGER CTR ADDRESS
0165 0002 0000 RES 2,0
0166 0004 0F0F DATA :0F0F DEVICE STATUS WORD
0167 0005 C6C0 TEXT 'FMO0' DEVICE NAME
      0006 B0B0
0168 0007 0000 RES 4,0
0169 0008 0000 DATA :0000 FILE STATUS WORD
0170 000C CFCC TEXT 'OLD' FILE NAME
      000D C4A0
      000F A0A0
0171 000F 0000 RES 9,0
0172 0018 001C DATA BRUF BLOCKING BUFFER ADDRESS
0173 0019 0000 RES 3,0
0174 *
0175 001C 0000 BRUF RES 510+2/2,0 BLOCKING BUFFER, SIZE = BLOCK SIZE+2 BYTES
0176 END

0000 ERRORS
0000 WARNING
```

II/5-27

```
0179 *
0180 * NEW FILE DIB. DEVICE SPECIFICATION WORD IS SET TO ALLOW
0181 * ANY FILE OPERATION. THE FILE IS UNBLOCKED WITH 80 BYTE RECORDS.
0182 * NO BLOCKING BUFFER IS REQUIRED.
0183 *
0184 0000 NAM D:FM1
0185 EXTR C:FM0
0186 *
0187 0000 D:FM1 CHAN X::
0188 0001 0000 DATA C:FM0 FILE MANAGER CIB ADDRESS
0189 0002 0000 RES 2,0
0190 0004 FFFF DATA :FFFF DEVICE STATUS WORD
0191 0005 C6CU TEXT 'FM01' DEVICE NAME
0192 0006 B0P1
0193 0007 0000 RES 4,0
0194 0008 0000 DATA :0000 FILE STATUS WORD
0195 000C CECS TEXT 'NEW' FILE NAME
0196 000D 07A0
0197 000F 00A0
0198 000F 0000 RES 2,0
0199 0011 0050 DATA 80 RECORD SIZE
0200 0012 0000 DATA 0 BLOCK SIZE (UNBLOCKED)
0201 0013 0000 RES 5,0
0202 0018 0000 DATA 0 BLOCKING BUFFER ADDRESS (NONE REQUIRED)
0203 0019 0000 RES 3,0
0204 *
0205 END
0000 ERRORS
0000 WARNING
```

II/5-28





SECTION 6

DEVICE-DEPENDENT CONSIDERATIONS

The device-dependent functions of IOX are the responsibility of the individual device handlers. Initially IOX performs all parameter validation and error checking before control is transferred to the appropriate device handler. The device handler will execute the data transfer and perform the device testing. Note that the bit configuration for each function (bits 3-0 of IOB word 5) is listed below each operation in parentheses.

6.1 STANDARD CHARACTER DEVICE HANDLERS

6.1.1 Line Printer (LP)

Write (formatted ASCII) (0110)	Outputs up to 132 (or less if the printer is not that wide) characters.
Write file mark (1100)	Outputs /* in columns 1 and 2.
All other function codes	No I/O

6.1.2 Teletype Keyboard (TK)

Write (formatted ASCII) (0110)	Outputs up to 72 characters. Carriage return, line feed are appended to the end of each record.
Write (unformatted) (0101)	Outputs up to 65,535 characters exactly as in the user's buffer.
Read (formatted ASCII) (0010)	Inputs from the keyboard until a carriage return is read. Standard character editing is active.
Read (unformatted) (0001)	Inputs from the keyboard until the number of characters requested is input.
Write File Mark (1100)	/* is output followed by carriage return, line feed.
All other function codes	No I/O



6.1.3 Teletype Console (TY) (implies tape reader or keyboard for input, whichever is ready)

Write (formatted ASCII) (0110)	Outputs up to 72 characters. Carriage return and line feed are appended to the end of each record.
Write (unformatted) (0101)	Outputs up to 65,535 characters exactly as in the user's buffer.
Read (formatted ASCII) (0010)	Inputs (from the tape reader, if ready, otherwise from the keyboard) until a carriage return is read. Standard character editing is active.
Read (unformatted) (0001)	Inputs (from the tape reader, if ready; otherwise from the keyboard) until the requested number of characters is input.
Write File mark (1100)	/* is output, followed by carriage return, line feed.
All other function codes	No I/O

1. Teletype Reader (TR)

Read (formatted ASCII) (0010)	Inputs up to 256 ASCII characters from the reader (does NOT echo on printer) until a carriage return is read. Standard character editing is active.
Read (unformatted) (0001)	Inputs from the reader (does NOT echo on printer) until the number of characters requested is input.
Read (formatted binary) (0011)	Reads one binary record and checks the checksum. If a checksum error is detected, the error status will be set.
All other function codes	No I/O

1. Teletype Punch (TP)

Write (formatted ASCII) (0110)	Outputs up to 256 ASCII characters. Carriage return line feeds are supplied at the end of each record.
Write (formatted binary) (0111)	Outputs up to 65,535 bytes in IOX binary format.
Write (unformatted) (0101)	Outputs up to 65,535 bytes exactly as in the user's buffer.
Write File Mark (1100)	Outputs: Rubout-Null-Null on the paper tape.
Punch Leader (1101)	Outputs 12 inches of leader.
All other function codes	No I/O

6.1.6 Card Reader (CR)

Read (formatted ASCII) (0010)	One card will be read. The maximum number of bytes transferred is 80. If the first two columns contain /* an end-of-file is assumed.
Read (formatted binary) (0011)	To be specified . . . if the first two columns contain /* an end-of-file is assumed.
All other function codes	No I/O

6.1.7 High Speed Reader (PR)

Read (formatted ASCII) (0010)	Inputs from the reader until a carriage return is read. Standard character editing is active.
Read (unformatted) (0001)	Inputs from the reader until the number of characters requested is input.
Read (formatted binary) (0011)	Reads one binary record and checks the checksum. If the checksum is in error the error status is set.
All other function codes	No I/O

6.1.8 High Speed Punch (PP)

Write (formatted ASCII) (0110)	Outputs up to 256 ASCII characters. Carriage return line feeds are supplied at the end of each record.
Write (formatted binary) (0111)	Outputs up to 65,535 bytes in the IOX binary format.
Write (unformatted). (0101)	Outputs up to 65,535 bytes exactly as in the user's buffer.
Write File Mark (1100)	Outputs Rubout, Null, Null on the paper tape.
Punch Leader (1101)	Outputs 12 inches of leader.
All other function codes	No I/O

6.2 FORTRAN LIST DEVICE HANDLER

6.2.1 Line Printer (LPF)

Write (formatted ASCII) (0110)	Outputs up to 132 characters, preceded by a carriage control character ("1" = top of form, "0" = double upspace, any other = single upspace).
-----------------------------------	---



Write file mark
(1100)

Outputs "/"* in columns 1 and 2.

All other function codes

No I/O

6.2.2 Teletype Keyboard (TKF)

Write (formatted ASCII)
(0110)

Outputs up to 72 characters, preceded by carriage control character ("1" = top of form = 6 upspaces, "0" = double upspace, any other = single upspace).

Write (unformatted)
(0101)

Outputs up to 65,535 characters exactly as in the user's buffer.

Read (formatted ASCII)
(0010)

Inputs from the keyboard until a carriage return is read. Standard character editing is active.

Read (unformatted)
(0101)

Inputs from the keyboard until the number of characters requested is input.

Write File Mark
(1100)

/* is output followed by carriage return, line feed.

All other function codes

No I/O

6.2 Teletype Console (TYF) (implies tape reader or keyboard for input, whichever is ready)

Write (formatted ASCII)
(0110)

Outputs up to 72 characters, preceded by carriage control character ("1" = top of form = 6 upspaces, "0" = double upspace, any other = single upspace).

Write (unformatted)
(0101)

Outputs up to 65,535 characters exactly as in the user's buffer.

Read (formatted ASCII)
(0010)

Inputs (from the tape reader, if ready, otherwise from the keyboard) until a carriage return is read. Standard character editing is active.

Read (unformatted)
(0001)

Inputs (from the tape reader, if ready; otherwise from the keyboard) until the requested number of characters is input.

Write File mark
(1100)

/* is output, followed by carriage return, line feed.

All other function codes

No I/O



6.3 MAGNETIC TAPE HANDLER

6.3.1 Magnetic Tape (MT)

Write (formatted ASCII,
formatted binary, or
unformatted)
(0110, 0111, or 0101)

Outputs 1 to 65535 bytes as a single record.

Read (formatted ASCII,
formatted binary, or
unformatted)
(0010, 0011, or 0001)

Inputs one record up to 65,535 bytes. If the actual record is longer than the requested number of bytes, only the requested number will be input. If the actual record is shorter than the requested input, only the actual number of bytes are input. Up to ten retries will be made in the event of a parity error before an error status is returned to the caller.

Position Relative Records
(1010)

Skips the number of records in the requested count. A positive count indicates forward skips. A negative count indicates backward skips. If a file mark is encountered during the positioning, the operation is terminated, and the number of records actually skipped (not including the file mark) is returned along with an end-of-file status. The tape is left positioned prior to the file mark (the file mark is never actually crossed and movement is effectively bounded within a pair of file marks). If an end of tape or beginning of tape marker is found during positioning, the operation is terminated with the actual count returned and an end-of-device status.

Position Relative Files
(1011)

Skips the number of file marks in the requested count. A positive count indicates forward skips. A negative count indicates backward skips. Upon return, the tape is positioned past the last file mark skipped. If an end-of-tape or beginning-of-tape mark is encountered, the operation is terminated with the actual skip count returned, along with the appropriate end-of-device status.

Position Absolute Records
(1000)

The tape is first rewound to load point, then skipped forward the number of records requested. The requested count must be positive. If the count is zero, the tape is left at load point.

Position Absolute Files
(1001)

The tape is first rewound, then skipped forward the number of files requested. The requested count must be positive. If the count is zero, the tape is left at load point.

Write File Mark
(1100)

A write file mark function is issued to the tape unit.

All other operations

No I/O



6.4 DISK, STORAGE MODULE DISK, AND FLOPPY DISK HANDLER

6.4.1 Disk (DK), Storage Module Disk (SM), and Floppy Disk (FD)

Write Direct Access (0100)	Writes to the disk the number of bytes specified by the user in IOB Word 6, to the relative record number specified in IOB Word 9. Upon completion of the operation, this record number is incremented.
Read Direct Access (0000)	Reads from the disk the number of bytes specified by the user in IOB Word 6, from the relative record number specified in IOB Word 9. Upon completion of the operation, this record number is incremented.
All other function codes	No I/O

NOTE

The Floppy Disk Handler supports only one floppy disk controller. The handler must not be used concurrently with a storage module disk controller.

The Storage Module Disk Handler supports only one storage module disk controller. The handler must not be used concurrently with a floppy disk controller.

6.4.2 Fortran Disk (DKF), Storage Module Disk (SMF), and Floppy Disk (FDF)

Write (formatted ASCII, formatted binary) (0110 or 0111)	Outputs to the disk the number of bytes specified by the user, to the relative record number maintained in DIB Word 15. Upon completion of the operation, this record number is incremented and stored into IOB Word 9.
Read (formatted ASCII, formatted binary) (0010 or 0011)	Inputs from the disk the number of bytes specified by the user, from the relative record number maintained in DIB Word 15. Upon completion of the operation, this record number is incremented and stored into IOB Word 9.
Position Relative Records (1010)	The requested count (positive or negative) is added to the current relative record number maintained in DIB Word 15. (No actual I/O occurs). The new record number is also copied into IOB Word 9. If the resultant relative record number is greater than the highest sector number in the extent, the highest sector number is stored, and the end-of-device status is returned. If the resultant relative record number is negative, a zero (representing the first record of the extent) is stored, and a beginning-of-device status is returned.



Position Absolute Records
(1000)

The requested count (which represents the actual record number to be positioned to), is stored into DIB Word 15 and IOB Word 9. No actual I/O occurs. If the record number is greater than the highest sector number in the extent, the highest sector number is stored, and the end-of-device status is returned. If the record number is negative, a zero (representing the first record of the extent) is stored, and a beginning-of-device status is returned.

Write File Mark
(1100)

A two character record containing "/"* is written into the record pointed to by the Relative Record Count, then this count is incremented and copied into IOB Word 9.

All other function codes

No I/O.

NOTE

The Floppy Disk Handler supports only one floppy disk controller. The handler must not be used concurrently with a storage module disk controller.

The Storage Module Disk Handler supports only one storage module disk controller. The handler must not be used concurrently with a floppy disk controller.

6.5 MAGNETIC TAPE INTELLINET CABLE (MTIC) HANDLER

Write forward
(ASCII or Binary)
(0110 or 0111)

Outputs 1 to 65,535 bytes as a single record. Records containing a byte count less than the minimum record length (DIB word 12) will have additional characters appended to the record until the byte count is equal to the minimum record length. Blanks are appended to ASCII records and zeros are appended to Binary records.

During write operation error recovery, the tape is backspaced one record and another write is attempted. Up to ten retries are made in the event of a rate error (processor workload error). Up to three retries are made in the event of a hard error (tape error); subsequently, a fixed length erase function is used to erase the hard error region and three more retries are executed. This erase procedure is executed up to ten times, at which point an error status is returned. (Note: Hard error recovery is modified if the Control Edit function is on. Refer to the Control Edit description.)

Error counts for each type of recovery are returned to DIB word 10.

Read (forward, reverse)
(ASCII, Binary),
Read Reverse
(0010, 0011, 0000)

Inputs one record up to 65,535 bytes. If the actual record is longer than the requested number of bytes, only the requested number is input. If the actual record is shorter than the requested input, only the actual number of bytes are input. Up to ten retries are made before an error status is returned.

Position Relative Records
(1010)

Skips the number of records in the requested count. A positive count indicates forward skips. A negative count indicates backward skips. If a file mark is encountered during the positioning, the operation is terminated, and the number of records actually skipped (not including the file mark) is returned along with an end-of-file status. The tape is left positioned prior to the file mark (the file mark is never actually crossed and movement is effectively bounded within a pair of file marks). If an end of tape or beginning of tape marker is found during positioning, the operation is terminated with the actual count returned with an end-of-device status.

Position Relative Files
(1011)

Skips the number of file marks in the requested count. A positive count indicates forward skips. A negative count indicates backward skips. Upon return, the tape is positioned past the last file mark skipped. If an end-of-tape or beginning-of-tape mark is encountered, the operation is terminated with the actual skip count returned with the appropriate end-of-device status.

Position Absolute Records
(1000)

The tape is first rewound to load point, then skipped forward the number of records requested. The requested count must be positive. If the count is zero, the tape is left at load point. If the count is minus one, the unit is placed offline.

Position Absolute Files
(1001)

The tape is first rewound, then skipped forward the number of files requested. The requested count must be positive. If the count is zero, the tape is left at load point. If the count is minus one, the unit is placed offline.

Write File Mark
(1100)

A write file mark function is issued to the tape unit.

Control Edit
(1110)

This function causes the formatter to implement special head positioning to allow record updating.

NOTE

Control Edit needs to be used with caution because of possible "tape creep". Refer to the Distributed I/O System User's Manual, Publication No. 91-53629-00B2, for a more detailed explanation.

Control Edit requires five calls to IO:. Call one positions the tape at the end of the record to be updated. (An inter-record gap containing an erasure or noise record might be found between the end of this record and the beginning of the next record.) Call two sets the edit function on. Call three performs a skip or read reverse function for the current record. Call four performs a write forward function for the new record. The byte counts for the new and old records must be equal. Call five set the edit function off.

Hard error recovery for write operations is modified when Control Edit is on. Up to three retries are made in the event of a hard error; subsequently, an error status is returned.

Control Erase
(1111)

This function performs a fixed length (filemark) or variable length erase. The erase mode bit is set to override a write operation. This function can be used with Control Edit to erase a record in place.

Control Erase requires three calls to IO:. Call one sets Control Erase on. Call two performs a write or write file mark function. Call three sets Control Erase off.

All other function codes No I/O.

6.6 STANDARD CHARACTER EDITING

In order to facilitate input from an operator, IOX supports character editing on input from all keyboard and paper tape devices. Three editing functions are supported by IOX.

1. Backspace. Character backspace is implemented using the back arrow (\leftarrow) character. One character is erased for each back arrow character input. Since it is impossible to physically backspace on a teletype, the back arrows are echoed on the printer. Note that the character editing will take place over the length of the entire physical record, not just until the number of currently valid characters equals the requested count.



2. Ignore entire input. Occasionally the operator decides it would be easier to start over rather than backspace and correct all of the errors on the current input. IOX supports this by deleting the entire input and restarting whenever the back arrow is typed followed immediately by a carriage return.
3. Ignore this character. This is useful when the input is from a paper tape which was prepared off-line on a teletype. The punch on a teletype has a local backspace feature, and the most common means of correcting a tape such that it prints properly when read off line is to backspace the punch over the offending character(s) and punch rubout(s) on top of them. IOX will read such tapes properly by ignoring all rubouts. In addition, IOX will read such tapes properly by ignoring all rubouts. In addition, IOX ignores all line feeds and all other characters whose ASCII code is less than :0D (e.g., bell, leader).

Since an end-of-file is defined as a Rubout, Null, Null on paper tape, and since it is difficult to enter Rubout, Null, Null on a keyboard, IOX recognizes two different end-of-file marks in the standard character editing mode for formatted ASCII input. These file marks are Rubout, Null, Null or /*. Either of these character sequences input at the beginning of a record will cause an end-of-file to be recognized.



SECTION 7

NON-STANDARD HANDLER DESCRIPTIONS

Some IOX handlers do not conform to the standard IOB, DIB, and CIB configurations described in sections 2 and 4. This section describes the software tables and device-dependent functions of these IOX handlers. (The A/D, D/A handler is described in Publication No. 93325-00.)

7.1 IEEE INTELLIGENT CABLE (IEC) HANDLER

The IEC Handler controls the operation of the IEEE Intelligent Cable. The IEC Handler and the IEEE Intelligent Cable together conform to the requirements for an IEEE (STD 488-1975) interface system controller. The IEEE Intelligent Cable provides the hardware to drive the IEEE interface bus and the firmware to conduct both the Source Handshake and the Acceptor Handshake. It also senses the state of the IEEE Interface Bus. The IEC Handler implements the remaining IEC functions. The interfaced devices must have no controller capabilities.

Refer to the Distributed I/O System User's Manual (revision B2 or higher) and IEEE document 488-1975, "IEEE Standard Digital Interface for Programmable Instrumentation" for detailed IEEE function descriptions.

Note that an arbitrary distinction is made between the terms "control" and "data" with respect to IEC handler message transfers. "Control" refers to bytes which are sent over the interface bus while ATN is true. "Data" refers to bytes which are sent or received over the interface bus while ATN is false.



7.1.1 IEC IOB Configuration -- 9 to 12 words.

Figure 7-1 illustrates the IOB configuration for the IEC Handler.

INPUT/OUTPUT BLOCK
FOR THE IEEE INTELLIGENT CABLE HANDLER

Standard
Name

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0 word

IDT

IOBN or
IOB

CN

ILUN

ISTA, IOP

IRCNT

IBUFF

IRCNT

IRCNTU

IBUFFU

ITIME

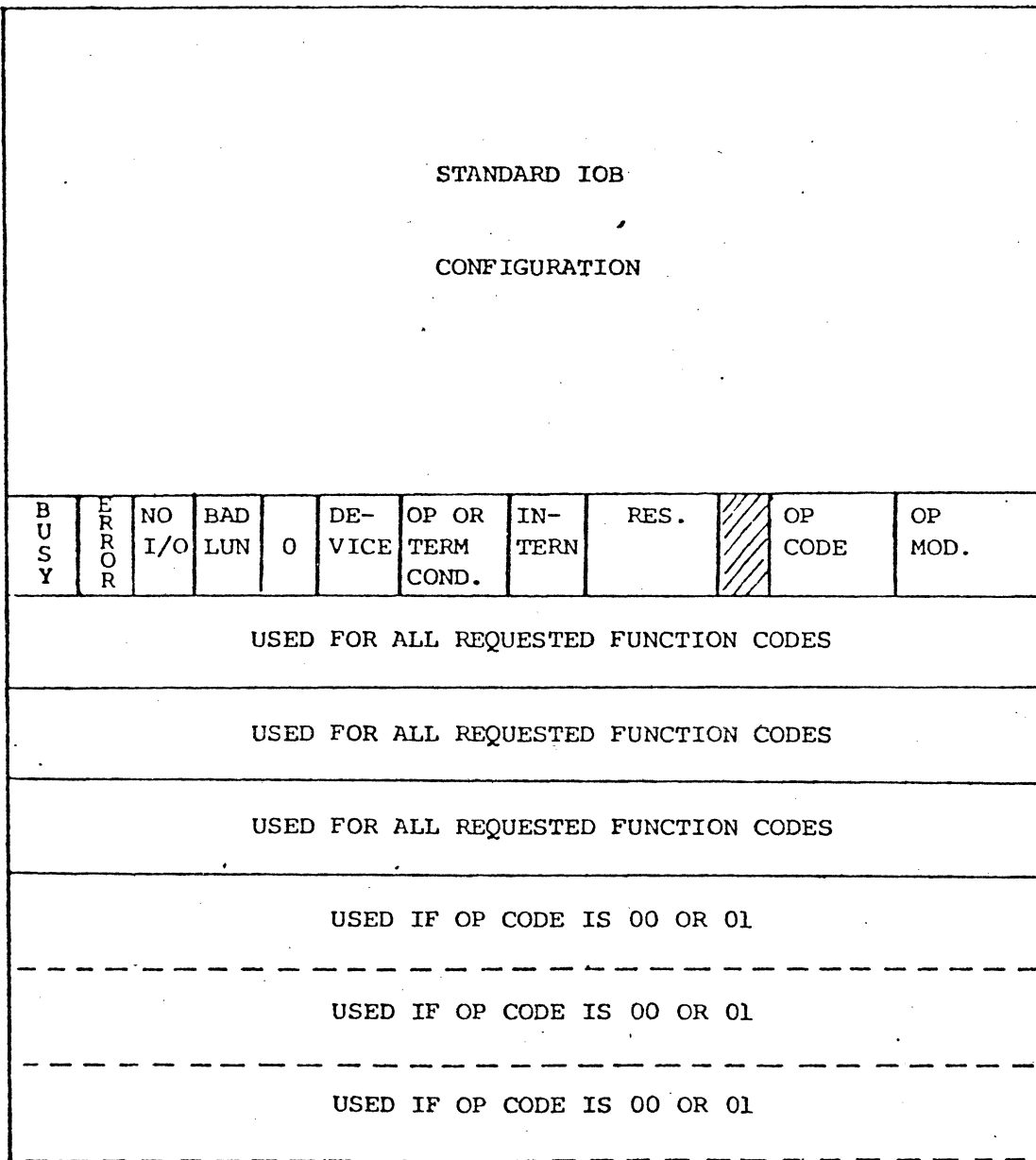
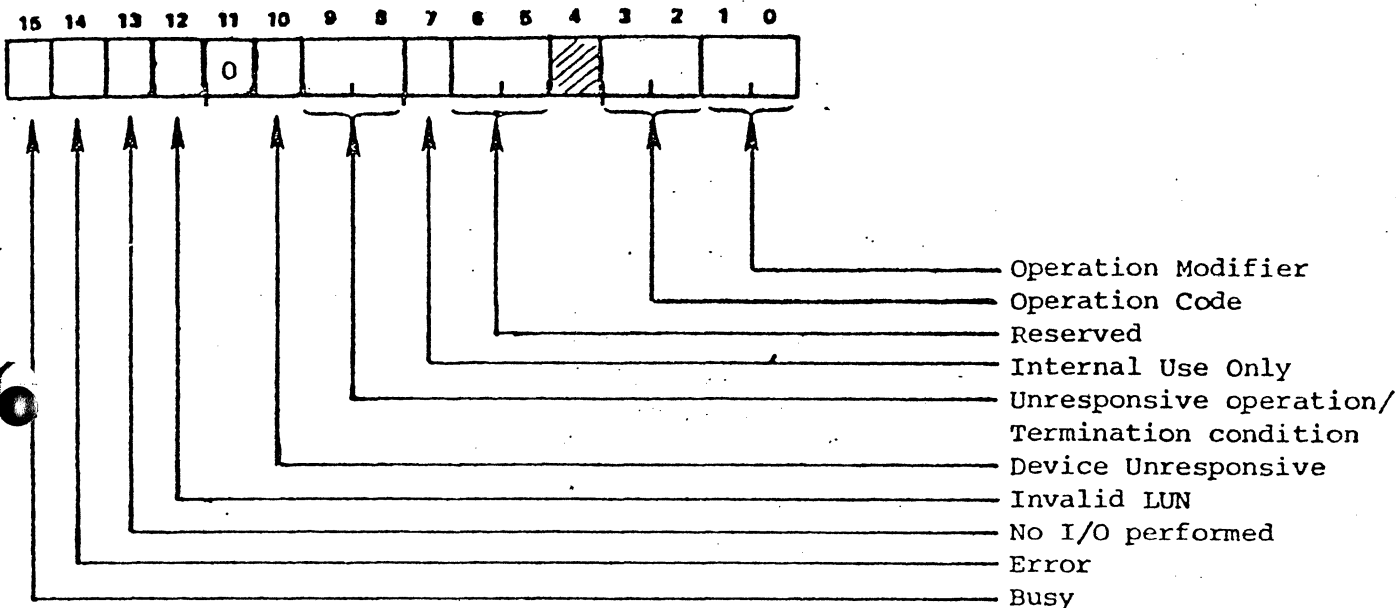


Figure 7-1. IOB Configuration
IEC Handler

Words 0 through 4 are the same as the standard IOB configuration. Refer to Section 2 for detailed descriptions of these words.

Word 5 Status, Function Code. This word uses the following format:



All bit positions, with the exception of bits 9 and 8, are described in Section 2.

Bits 9 and 8. These bits can have two meanings, as follows:

- 1) Unresponsive operation. If an error has occurred (bit 14 set), bits 9 and 8 indicate what operation was being performed when the error occurred, as follows:
 - 01 while taking control of the IEEE interface
 - 10 while writing control
 - 11 while reading or writing data
- 2) Termination condition. For a read data operation, if bits 10 and 14 are zero, bits 9 and 8 indicate the reason for terminating the read data operation:
 - 00 END message detected
 - 10 Byte count reached zero (abnormal return)

Bits 9 and 8 are zero when all other operations are terminated.



Format of the IOB after the first six words is determined by bits 3-0 of word 5.

Format 1

<u>Op Code</u> (bits 3 and 2)	<u>Op Modifier</u> (bits 1 and 0)	<u>Function</u>
00	01	Write control and read data to END.
00	11	Same as 0001 with parity standardization
01	00	Write control and write data.
01	01	Write control and write data with END.
01	10	Write control only.
01	11	Write control and ignore data.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	word
DATA REQUEST COUNT																6
DATA BUFFER ADDRESS																7
ACTUAL DATA TRANSFER COUNT																8
CONTROL REQUEST COUNT																9
CONTROL BUFFER ADDRESS																10
TIME LIMIT																11

Word 6 Data Request Count. This word is supplied by the user to specify the number of data bytes to be transferred. This word must not be zero.

Word 7 Data Buffer Address. This word is supplied by the user to specify the starting address of the data buffer. Note that this address is always a word address and that indirect addressing is not allowed.

Word 8 Actual Data Transfer Count. This word contains the number of data bytes transferred when the operation is completed. This word is returned by IOX at the completion of I/O.

- Word 9 Control Request Count. This word is supplied by the user to specify the number of control bytes to be transferred. No control bytes are transferred if this word is zero.
- Word 10 Control Buffer Address. This word is supplied by the user to specify the starting address of the control buffer. Note that this address is always a word address and that indirect addressing is not allowed.
- Word 11 Time Limit. This word is supplied by the user to specify the operation time limit. If negative, there is no time limit. If positive, a "device unresponsive" error will occur if the read or write operation has not completed within the number of clock ticks specified. If zero, the operation time limit will equal the number of data bytes (IOB Word 6) modified by the delay modification instruction stored into CIB word 20. (Refer to the SIO: description in Section 3).

Note that the specified number of clock ticks (word 11 positive) applies to data transfers only. The time limit for control transfers is always determined by the byte count and CIB word 20.

Format 2

Op Code Op Modifier
 (Bits 3 and 2) (bits 1 and 0)

10 00 Wait for SRQ

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	word
CLOCK TICKS															6	
NOT USED															7	
NOT USED															8	

Word 6 Clock Ticks. This word is supplied by the user to specify the number of clock ticks before SRQ is found. No time limit is applied if this word is negative. If positive, a "device unresponsive" error will occur if SRQ is not found within the number of clock ticks specified. This word may not be zero.

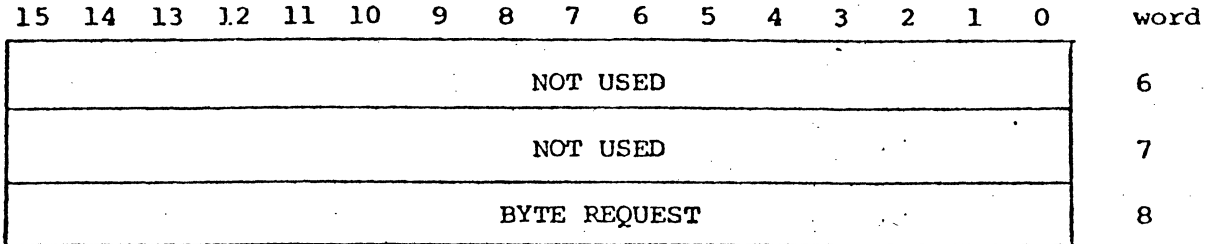
Words 7 and 8 are not used but must be provided.



Format 3

Op Code (bits 3 and 2) Op Modifier (bits 1 and 0)

11	00	Get IEC status
11	10	Get parallel poll response



rd(6 and 7 are not used but must be provided.

or 3 Byte Request. This word is returned to the user by IOX. It contains the requested byte (either status or parallel poll response). Figure 7-2 illustrates the IEC status byte configuration. The parallel poll response will be returned in the low order byte.

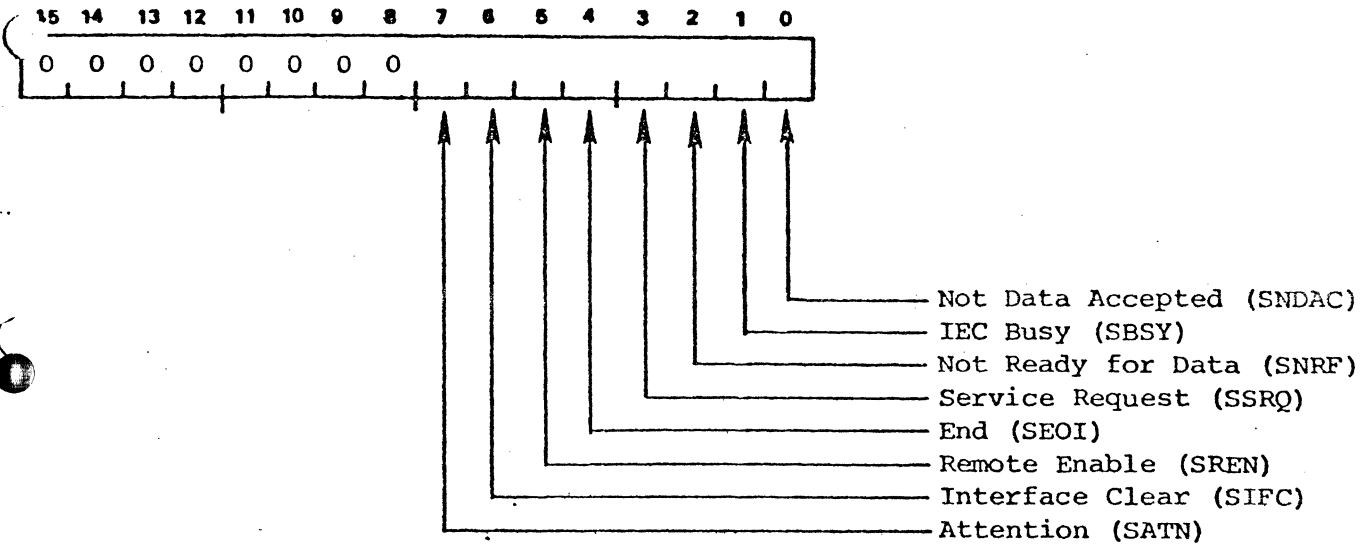


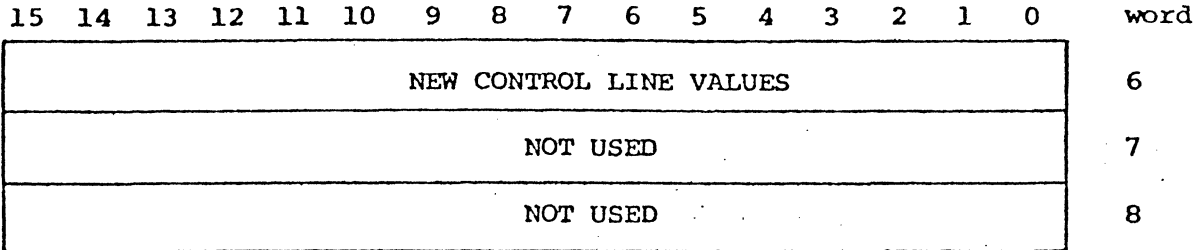
Figure 7-2. IEC Status Byte Configuration



Format 4

Op Code (bits 3 and 2) Op Modifier (bits 1 and 0)

11 01 Set IEC control lines



Word 6 New Control Line Values. This word is supplied by the user to specify the new value of the IEEE control lines. Only lines ATN, REN, IFC, EOI, and SRQ can be changed. A "Get IEC status" operation should be performed prior to a "Set IEC control lines" operation to ensure that the values of other lines are not changed inadvertently. Figure 7-3 illustrates the IEC Set Mode Command Word Format.

Words 7 and 8 are not used but must be provided.

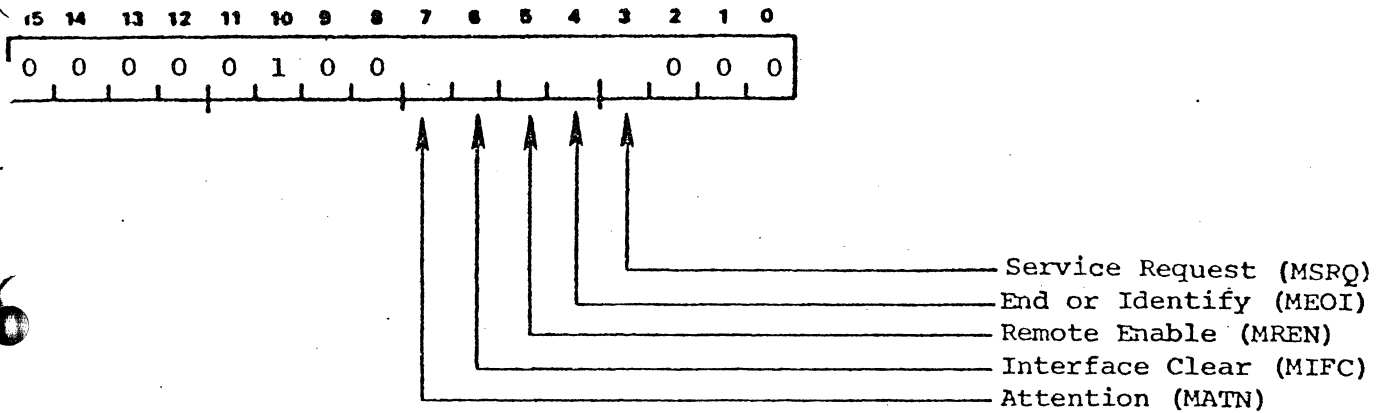
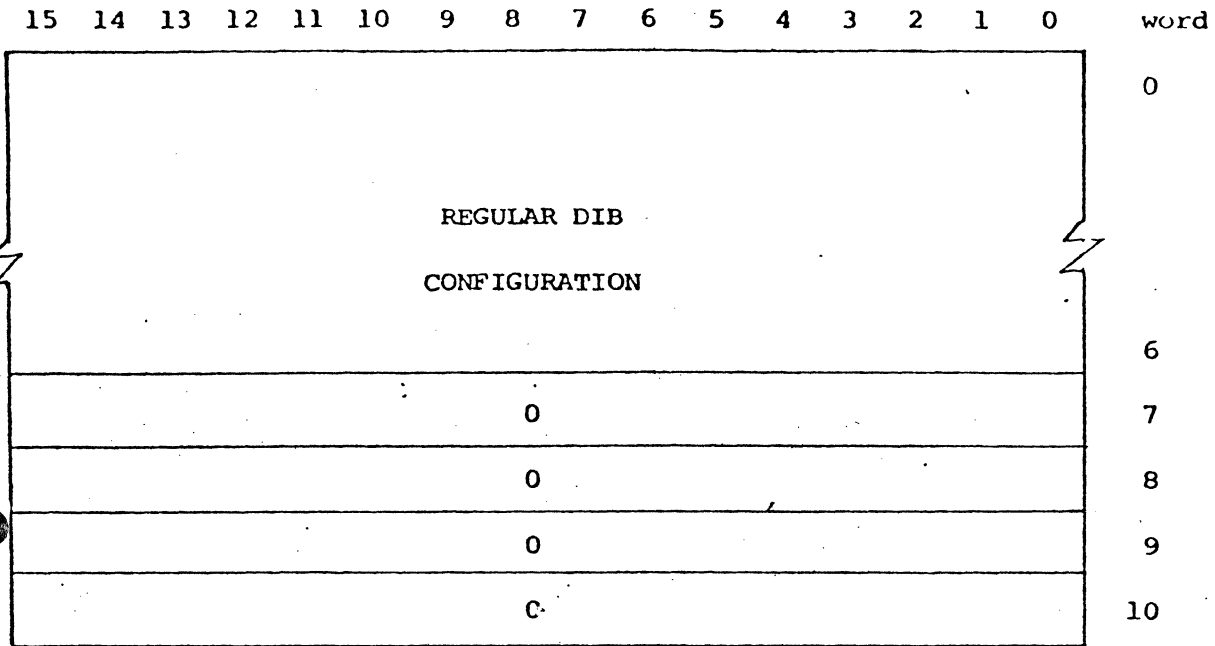


Figure 7-3. IEC Set Mode Command Word Format

The following function codes are undefined:

Op Code	Op Modifier
00	00
00	10
10	10
10	11
11	10

7.1.2 IEC DIB Configuration -- 11 words



Words 0-6 correspond to the regular DIB configuration described in Section 4. Words 7-10 are zeros.

7.1.3 IEC CIB Configuration -- 34 words

Figure 7-4 illustrates the IEC Controller Information Block.

IEC CONTROLLER INFORMATION BLOCK

Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Word
CBOR	TC FOR IE:SIO--BEGINNING OF RECORD FLAG																0
CSEL7	TC FOR IE:SIO--SEL DA,7																1
	JST								INTQ:								2
	DATA \$								CALLING LOC								3
CTMP1	DATA 0								TEMP 1								4
CTMP2	DATA 0								TEMP 2								5
CTMP3	DATA 0								TEMP 3								6
CZBTSK	DATA INTP:								TASK ADDRESS								7
	DATA B180								PRIORITY								8
CNEWA	DATA 0																9
CNEWX	DATA \$-10																10
	DATA \$-11																11
CAIDL	CURRENT PICO IDLE STATE--INITIALIZE TO 0																12
CDOG	WATCHDOG FLAG FOR IE:SIO																13
REQCNT	TC																14
CJTEL	DATA IECR:																15
	DATA IECW:																16
	DATA 0																17
	DATA IECP:																18
CSPLOP	DATA 0																19
CDEL	TC FOR IE:SIO--WATCHDOG TIMER INSTR. (NOP)																20
CINTR	TC FOR IE:SIO--IEC WORD INTERRUPT ADDR.																21
CADDLY	TC FOR IOX--TIME DELAY FROM IOB																22
CSTPCH	TC FOR IOX--STOP CHARACTER																23
CIOB	TC FOR IOX--IOB ADDR.																24
COP	TC FOR IOX SCHED--OP CODE AND STATUS (IOB wd 5)																25
CRCNT	TC FOR IOX SCHED--REQUESTED DATA COUNT (IOB wd 6)																25
CBUFF	TC FOR IOX SCHED--DATA BUFFER ADDR (IOB wd 7)																27
CTCNT	TC FOR IOX--ACTUAL DATA BYTE COUNT (IOB wd 8)																28
CRCNTU	TC FOR IOX SCHED--REQUESTED CONTROL COUNT (IOB wd 9)																29
CDIB	TC FOR IOX SCHED--DIB ADDR AND BUSY FLAG																30
CBUFFU	TC FOR IOX SCHED--CONTROL BUFFER ADDR (IOB wd 10)																31
STATUS	TC FOR EOR:--STATUS																32
CRTN	TC FOR IE:SIO--RETURN ADDRESS																33

NOTE:
TC = Temp Cell

Figure 7-4. IEC CIB Configuration



7.1.4 IEC Device-Dependent Considerations

Write Control and Read
Data to END
(0001)

The contents of the control buffer (IOB word 10) are sent with the source handshake while ATN is true. The control buffer will usually contain interface commands which address a peripheral as the talker. The IEC then initiates the acceptor handshake. The peripheral will transmit data when ATN is false. The IEC receives data until an END message was received with a byte or the Auto I/O byte count reached zero.

The handler will set bits 9 and 8 of IOB word 5 to the appropriate termination condition. The number of bytes transferred is returned to IOB word 6.

A read data to END only operation is performed by issuing requested function code 0001 with the control request count (IOB word 9) equal to zero.

Write Control and Read
Data to END with Parity
Standardization
(0011)

This function is the same as function code 0001; during input, however, the I/O Distributor performs parity standardization on all data bytes.

Write Control and
Write Data
(0100)

The control buffer is transmitted using the source handshake while ATN is true. The control bytes will generally address a peripheral to accept device programming. ATN is driven false following termination of the control sequence, and the data buffer is transmitted using the source handshake.

A write data only operation is performed by issuing requested function code 0100 with the control request count (IOB word 9) equal to zero.

Write Control and Write
Data with END
(0101)

This operation is the same as function code 0100 except that the last byte of data is sent with the EOI control line true, indicating an END message.

A write data with END only operation is performed by issuing function code 0101 with the control request count (IOB word 9) equal to zero.

Write Control Only
(0110)

The control buffer is transmitted using the source handshake while ATN is true. The data request count (IOB word 6) must not be set to zero. The IEC will maintain control of the IEEE interface after the transfer by setting NRFD true.



Write Control and
Ignore Data
(0111)

The control buffer is transmitted using the source handshake while ATN is true. ATN is driven false and remains false following termination of the control sequence. Peripheral to peripheral data transfers may occur with the IEC in this state. Any read or write request following this function will be prefaced with a Take Control Synchronously operation so the IEC will regain control of the IEEE interface.

Wait for SRQ
(1000)

The IEC is instructed to wait until the IEEE control line SRQ is found true. The handler will return immediately if SRQ is true when the request is made.

Get IEC Status
(1100)

The IEC status is returned to IOB word 8.

Set IEC Control Lines
(1101)

IEEE control lines ATN, REN, IFC, EOI, and SRQ assume the values contained in IOB word 6. This function allows the transmission of interface messages which involve these control lines, such as "interface clear" and "remote enable". Note that the handler changes the values of all these lines when performing other operations.

Get Parallel Poll Response
(1110)

An IDY remote message is sent for parallel polling. When the IEC is ready, the handler returns the result of the parallel poll to IOB word 8.

All Other Function Codes

No I/O

Original

SUBJECT: THE RTX SCHEDULER

VERTEILT	
Bryant	
Duch	
Emery	
Kerck	
Masters	X
Sievers	
Uitz	
Ulbrich	X
Verbeck	
(H.S.P.)	

In a typical application system based on LSI-Series computers, several independent external activities must be processed in the same time frame (for instance, a system may perform a test operation while the line printer is printing the results of the previous test, and the teletype is inputting parameters for the next test).

This implies that the system will be able to recognize events (probably via interrupts) and schedule appropriate service activities to process the events in a timely fashion. Generally, the most effective mechanism to accomplish this recognition and scheduling process is the Real-Time Executive (RTX). TAB/DB104 (TRN 93300-03-01-XX) discusses the recognition process and the insertion of new activities into the stream of ongoing activities (via INTQ:). This TAB discusses the actual scheduling mechanism, SCHED:.

The function of SCHED: can be described quite simply: If there are no activities to be performed, wait until there is one; if there are, merge the list of new activities (created by INTQ:) into the list of ongoing activities, according to priority, and cause the highest priority activity to be executed.

The term "activity" is nebulous, describing a whole class of "things to be processed," including interrupt service subroutines, Auto I/O, DMA, and tasks. The term "task" is much more definable, and in the context of RTX means precisely: "A program or set of programs which operates to perform a specific function within the Real-Time application."

A discussion of the difference between the two terms will clarify the operation of the system. When a user starts the execution of a task (by a call to BEGIN: or INTQ:), he is starting one or more activities, depending on the operation of the task.

For example:

1. A task is currently executing--this is one activity. Somewhere along the line an interrupt occurs, and the execution of the activities required to service the interrupt temporarily suspends the current task--and terminates the current activity. When the task is resumed, it will still be the same task, but of course, it is now a different activity.



THE RTX SCHEDULER (continued)

2. A task--one activity--calls DELAY:, suspending the task, and terminating the activity. There is still an activity associated with the task though--the "active" delay.

Some activities such as an interrupt service subroutine, occur without intervention by RTX. Other activities, like those described above, must each be regarded by RTX as an entity, and kept track of in some manner.

The RTX work area contains a user-defined number of 5-word blocks (see section 2 of the RTX User's Manual). One of these blocks is used to record each activity known to the system. The table in the RTX manual shows the number of work area blocks allocated for each call for RTX service. This number is also the net gain or loss in number of activities in the system.

The user's initial call to RTX: causes his defined work area to be broken up into 5-word blocks. Each of these blocks contains, in the first word (word 0), a pointer to the next block, thus forming a "linked list" of available blocks-- the FREE list. A pointer to the first of these available blocks is maintained at the location called FREE (at 7RF--see section 4 of the RTX User's Manual). One of these blocks is immediately allocated to contain information about the current activity-- the initialization task. (This block is placed at the top of the READY list, see below.)

Besides the FREE list, RTX maintains a number of other lists. This TAB is concerned with only two of them, the READY list and the FIFO list. The READY list (0RF) contains blocks, linked in priority order, describing activities that are "ready" for service by the processor. The first of these (the "top" block) is the activity currently being processed, and is always the highest priority activity on the list. If there are no blocks on the READY list, this implies that--as far as RTX is concerned--the processor is idle (actually it is always doing something--note the "wait loop" in the attached flow chart).

Since these lists are linked, they must be maintained carefully. If an RTX service routine were in the process of changing the links in one of these lists, and an interrupt occurred, the interrupt service activities could try to use the same list, which would be a disaster.

It is the responsibility of INTQ: to prevent this from happening. This imposes two requirements: if an RTX

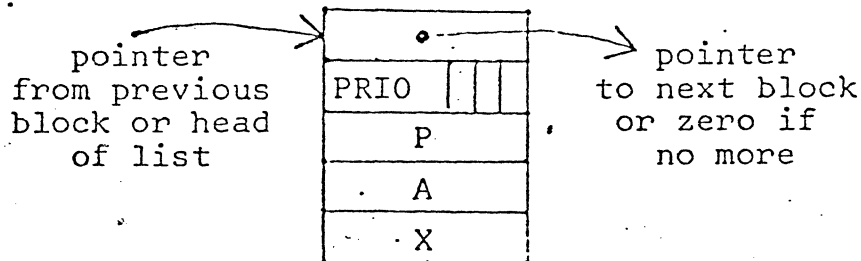


THE RTX SCHEDULER (continued)

service routine is interrupted, it must be completed before any further service is performed; and INTQ: must not alter the linkage in any list that is also altered in a routine that may have been interrupted.

Therefore, a special list is used for the "handshake" between INTQ: and SCHED:-- the FIFO list (LRF, the name means nothing). This list is maintained very carefully. INTQ: puts any new activities to be queued at the top of this list, and SCHED: carefully removes these, one at a time, and places them in the READY list in priority order. Once all of these have been placed, the scheduler is ready to set up the highest priority activity to be processed. (Once it has done this, it must check the FIFO list once more to see if an interrupt occurred during the scheduling process.)

Each of the blocks on the READY and FIFO lists has the following format:



Where:

- PRIO. is the priority of the activity (exclusive or'd with :1000 and shifted left three bits). Bit 2 of this word contains EIN indicator from processor status--always enabled. Bit 1 contains the byte/word indicator. Bit 0 contains the OV indicator.
- P is the contents of the program counter for the activity (i.e., where it is to be entered).
- A is the A register contents for the activity.
- X is the X register contents for the activity.

Let's go through an example of the operation of the scheduler. The system is initialized by the following call:

```

WKAREA RES 25,0      Reserve 5 five-word blocks
MAIN   JST RTX:      Initialize RTX
        DATA 5        ... using 5 blocks
        DATA WKAREA   ... allocated here
        HLT           Return to here if
                    1)we run out of blocks, or
                    2)an RTX service other than INTQ:
                       is called when
    
```

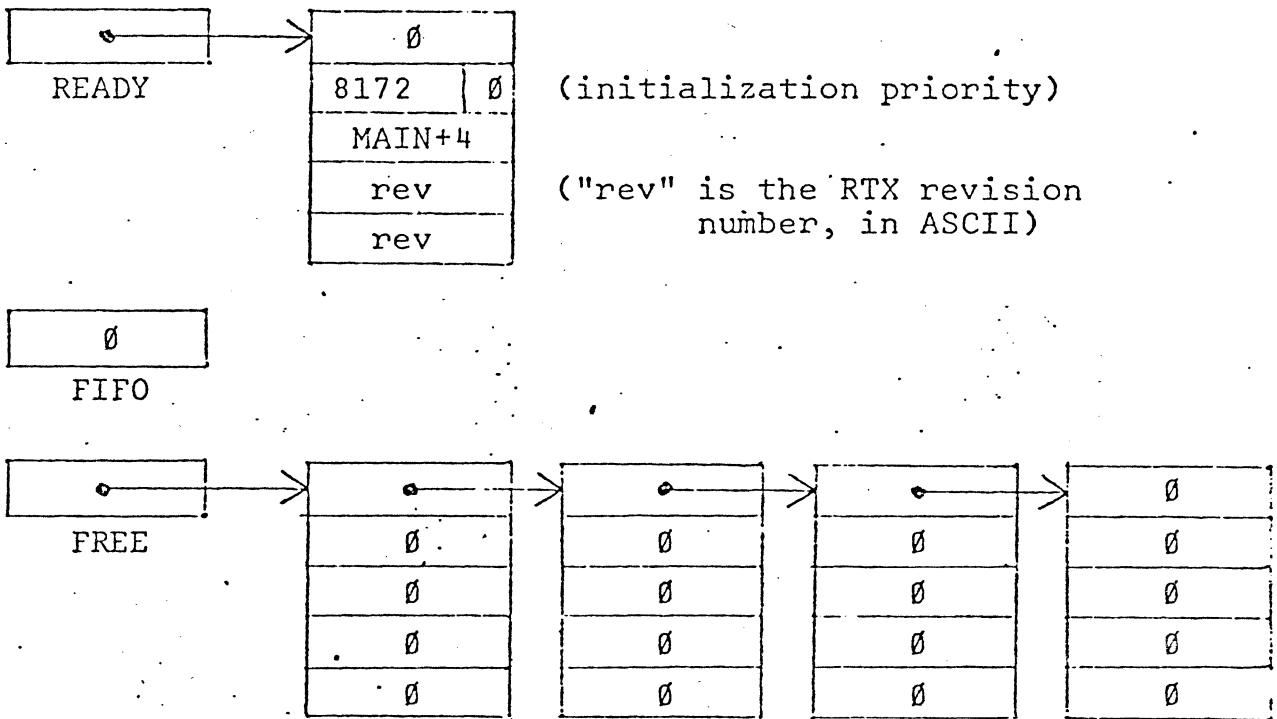


THE RTX SCHEDULER (continued)

there is no current activity
at the top of the READY
list.

Return to here to continue
initialization

After completion of this call, the work area has the following
contents:





THE RTX SCHEDULER (continued)

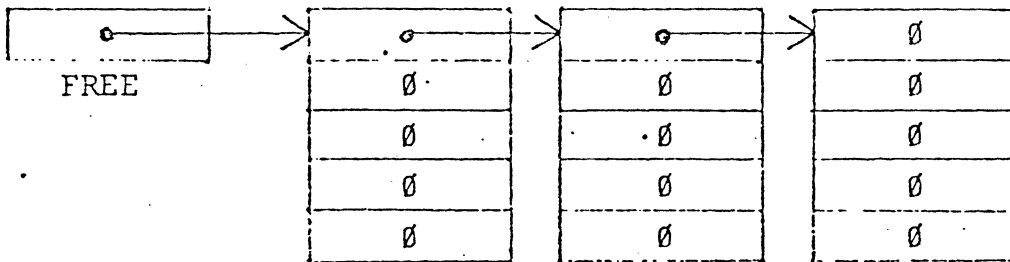
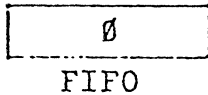
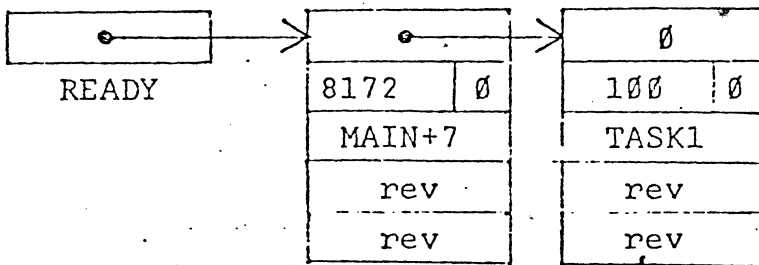
Now suppose the next thing in the initialization task, (at MAIN +4) is the following call to BEGIN:

```

:
JST  BEGIN:  Queue a task
DATA  TASK1  ... to be entered here
DATA  100    ... at this priority
:
... and return to here

```

After completion of this call, the work area has the following contents:



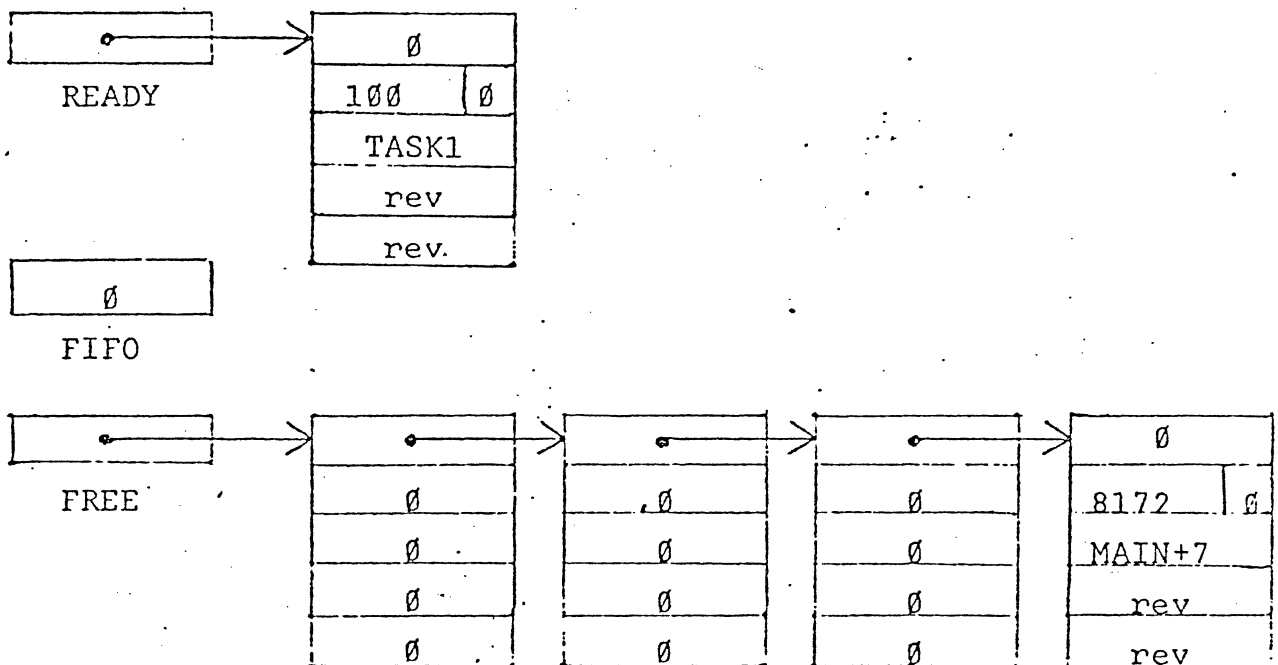


THE RTX SCHEDULER

The highest priority activity on the READY list is still the initialization task. Next (at MAIN +7) it does:

JST END: Terminate initialization

The work area now looks like this:



Note that TASK1 is now the highest priority activity, and will be executed next. Note also that blocks are returned to the end of the FREE list to allow the user to examine the history of the system if something goes wrong.

Therefore, we will begin executing at TASK1. Now suppose, for purposes of illustration, that the application program has a service subroutine for the console interrupt as follows:

```

ABS CONINT  interrupt location for console int
JST *$+1   call service subroutine
DATA CONIS at this location
REL 0      relocatable portion
CONIS ENT  save P here
CID       turn off interrupt or switch bounces
JST INTQ: call INTQ:
DATA $,0,0,0 ...see TAB DB/104
DATA CONTRK,200

```

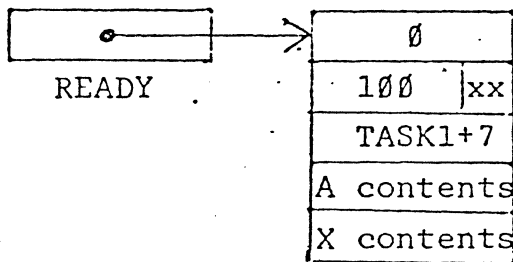


THE RTX SCHEDULER

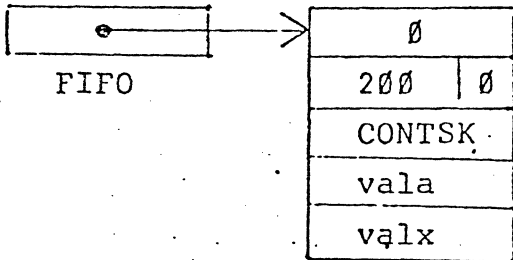
DATA vala, valx ...pass these values in A & X
DATA CONIS ...address of entry point

Now suppose that after seven instructions of TASK1, someone pushes the console interrupt switch. The interrupt service subroutine above will run, and call INTQ:, which will suspend the current activity (at TASK1 +7) and queue the console service task.

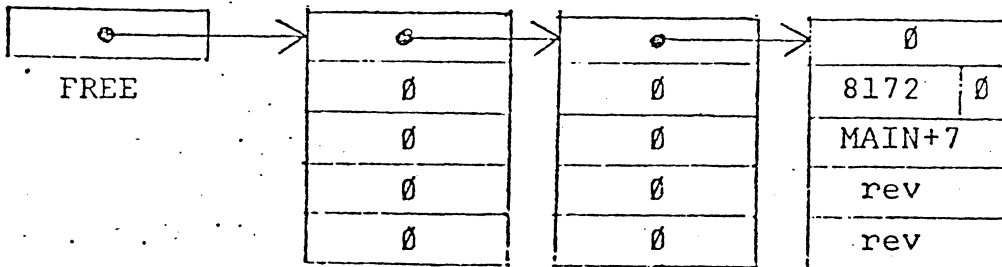
When INTQ: calls the scheduler, the work area looks like this:



(A, X, & status have changed, and TASK1 has been suspended)



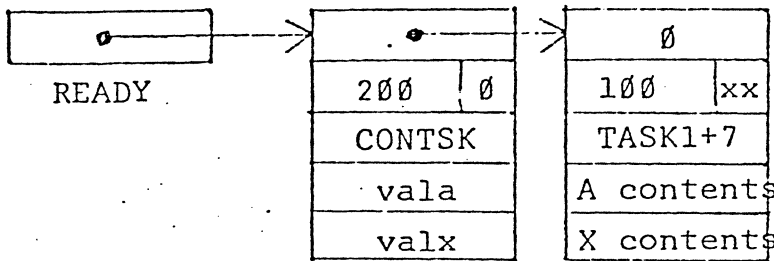
(CONTSK has been queued)



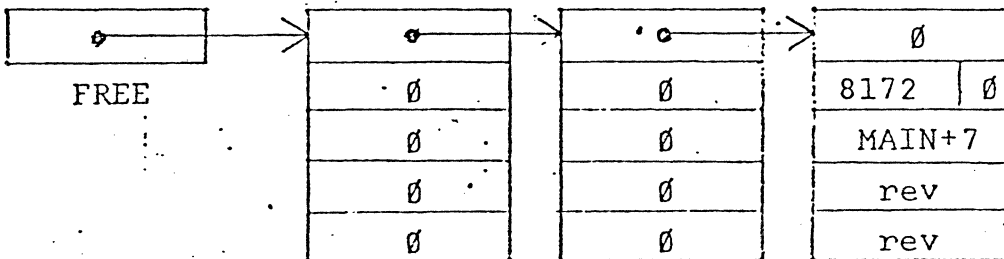
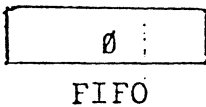


THE RTX SCHEDULER

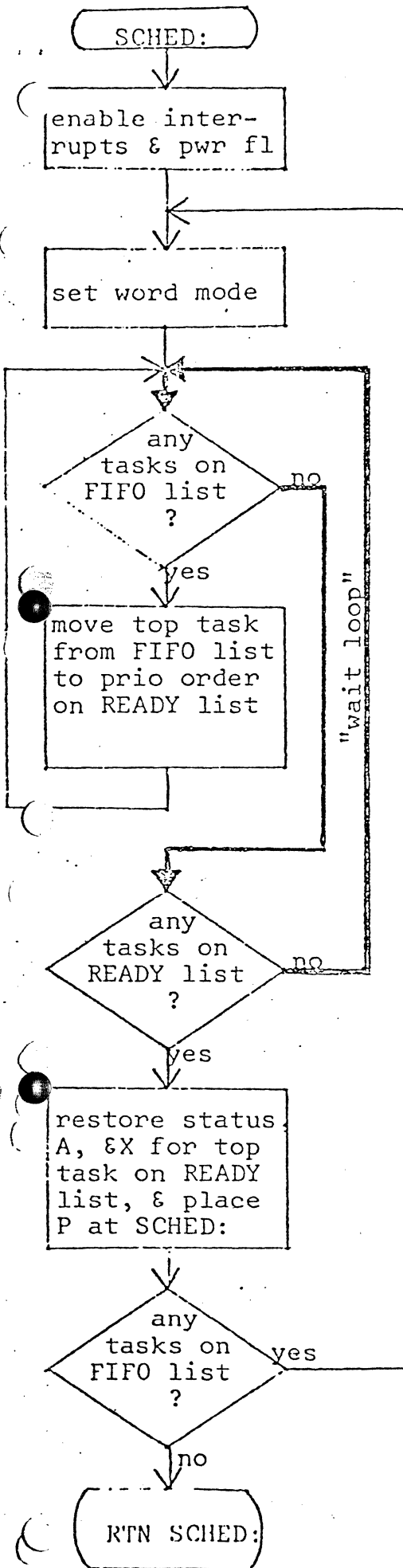
The scheduler will now merge the FIFO list into the ready list, leaving the work area as follows:



(the new task, CONTSK, is higher in priority than TASK1)



Now there are two activities on the READY list. The higher priority of the two, CONTSK, will be executed first. The other activity, TASK1 +7, will be executed when CONTSK suspends or terminates (unless something else of a higher priority comes in first).



flow of SCHED:
 RTX task
 scheduling
 routine

KUNDENINFORMATION

CIB NO 1128 a

Real Time Executive (RTX) Version F2

Mit dieser Information erhalten Sie die neue Version F2 von RTX. Es enthält einen RTX Basis File Manager für die Handhabung von Files im Standard Computer Automation OS File Format (siehe Section 5) und ein RTX File Label Utility.

Der File Manager, eingegliedert im IOX, enthält eine Directory und File Verwaltung für sequentielle oder random disc storage devices, welche dem Anwenderprogramm erlauben, mit Hilfe von Namen mit Daten Files zu korrespondieren.

Zusätzlich enthält der File Manager ein automatisches Blocken und Nichtblocken von Datensätzen mit Zugriff in Speicherreihenfolge.

Das RTX Label Utility ist ein binäres "stand-alone" Programm zum Labeln von dateiorientierten Geräten. Das Labeln mit diesen Utility ist kompatibel zum Computer Automation OS Datei Format.

Folgende Dokumentation und Lochstreifen sind beigelegt:

Dokumentation : RTX Users Manual Version F2

Lochstreifen : LSI 2 RTX/IOX Sequent 1 & 2
93300 - 30 F2 / 31 F2
LSI 2 RTX File Label Utility
93324 - 40 A1
LSI - 2 RTX Demo
93300 - 33 E1

Technischer Support
Rohde