REAL-TIME EXECUTIVE (RTX)
USER'S MANUAL

90-94500-00G1

DECEMBER 1978

ComputerAutomation ®

NAKED MINI® Division
18651 Von Karman, Irvine, CA 92713
tel. (714) 833-8830 TWX: (910) 595-1767

REVISION HISTORY

Revision	Issue Date	Comments
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.
F3	July 1977	Add A/D, D/A Handler Use Description to Manual.
F4	October 1977	Misc. RTX revision errors. Description of capability of IOX Handler for IEEE Intelligent Cable.
G0	November 1977	Adds COMX as part III of RTX Manual.
G1	December 1978	Removes COMX; corrects one documentation error.

TABLE OF CONTENTS

PART I. REAL-TIME EXECUTIVE (RTX)

PART I. TABLE OF CONTENTS (Cont'd)

LIST OF ILLUSTRATIONS

LIST OF TABLES

TABLE OF CONTENTS

PART II. INPUT/OUTPUT EXECUTIVE (RTX)

PART II. TABLE OF CONTENTS (Cont'd)

PART II. TABLE OF CONTENTS (Cont'd)

LIST OF ILLUSTRATIONS

TABLE OF CONTENTS

PART III. COMMUNICATIONS EXECUTIVE (COMX)

PART III. TABLE OF CONTENTS (Cont'd)

PART III. TABLE OF CONTENTS (Cont'd)

Page

LIST OF ILLUSTRATIONS

Figure

LIST OF TABLES

Table

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 the ALPHA-16, LSI-2, and LSI-305 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.

5. 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.

1.2 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, varying 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.

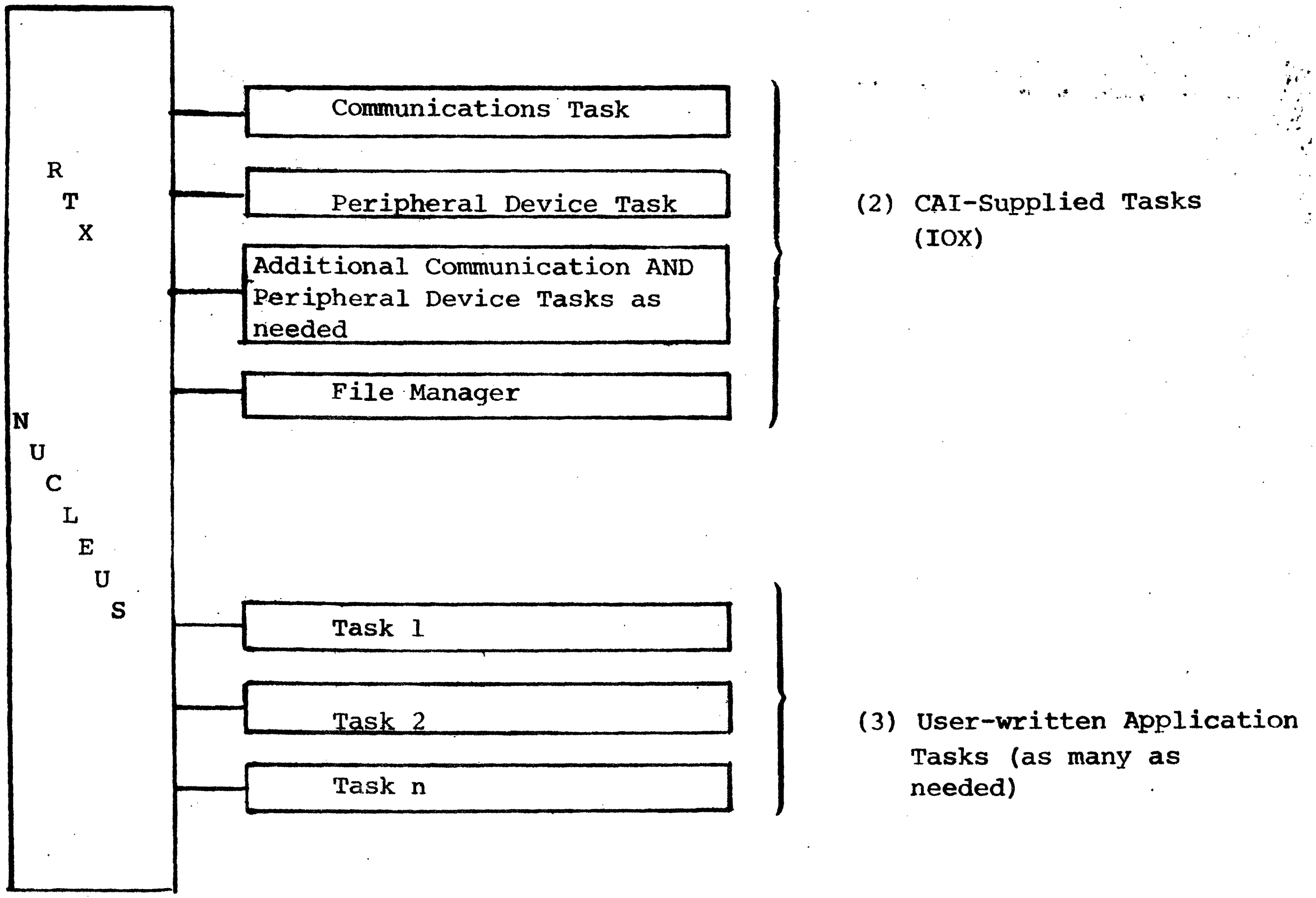
2. Common Subroutines: Subroutines which may be used by two or more different activities concurrently. These require special coding to provide reentrant capability.

3. Coordination Number: A decimal integer used to identify a task to RTX. It is analagous 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

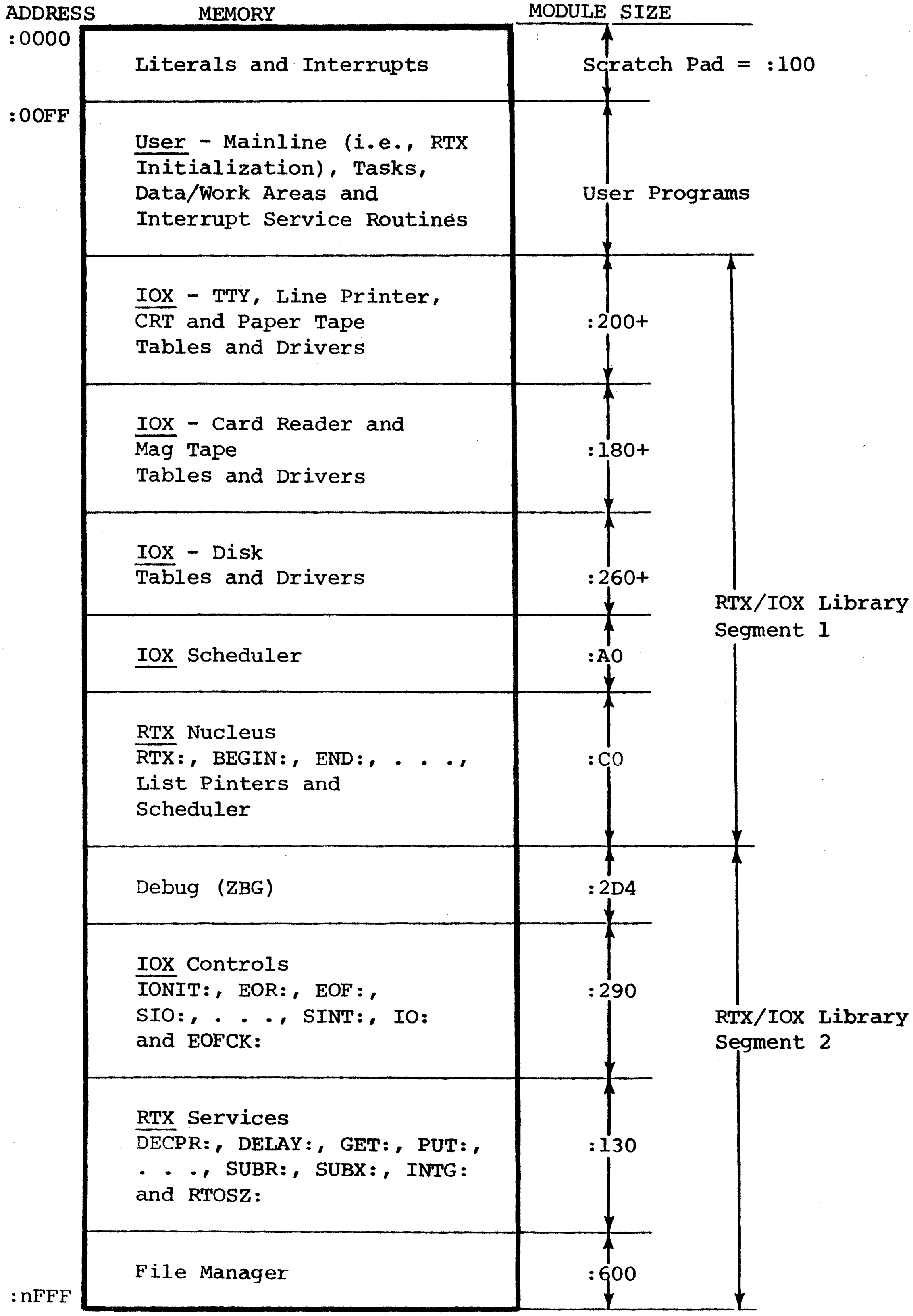


Figure 1-2. RTX Software Configuration

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 <u>equal priority tasks</u> 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 confilicts 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 subroutine)
0	SUBX: (If one or more tasks are waiting to use the common subroutine)
+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:) +1 For the immediate return +1 For scheduling +1 For setting a watchdog timer +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.

2.2.3 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.

2.2.4 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;

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

3. 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.

GET:

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:

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

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

3. 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.

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

2.2.7 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      SUBENT         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    SUBENT
            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.

3.7 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 INTRS: and INTQ: subroutines contain logic to preclude task-switching caused by an interrupt occuring 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.

3.10 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

                        JST            SUBX:
          ADDM          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: must be the first instruction following the subroutine entry point.

Revised 78

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 an alternate name for UNPR:, and is provided solely for symmetry in documentation.

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.

3.15 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:

Word	Contents
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.

Word	Contents
	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:

Word	Contents
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:

Word	Contents
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.

Word	Contents
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:

Word	Contents
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.

3. Using LAMBDA, <u>force</u> load the initializer task module of your application.

4. 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, <u>selectively</u> load the RTX Library object modules from the three RTX Library Tapes. The first tape applies only to COMX and is not needed unless COMX is to be used. The tapes are:

	LSI-2	LSI-3/05
Segment 0 - COMX	70-93300-32	70-93301-32
Segment 1 - RTX/IOX	70-93300-30	70-93301-30
Segment 2 - RTX/IOX	70-93300-31	70-93301-31

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 (00-93300-33) 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:

TIME	ACTION
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
TASK1
TASK2
TASK3
IOTASK
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 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.

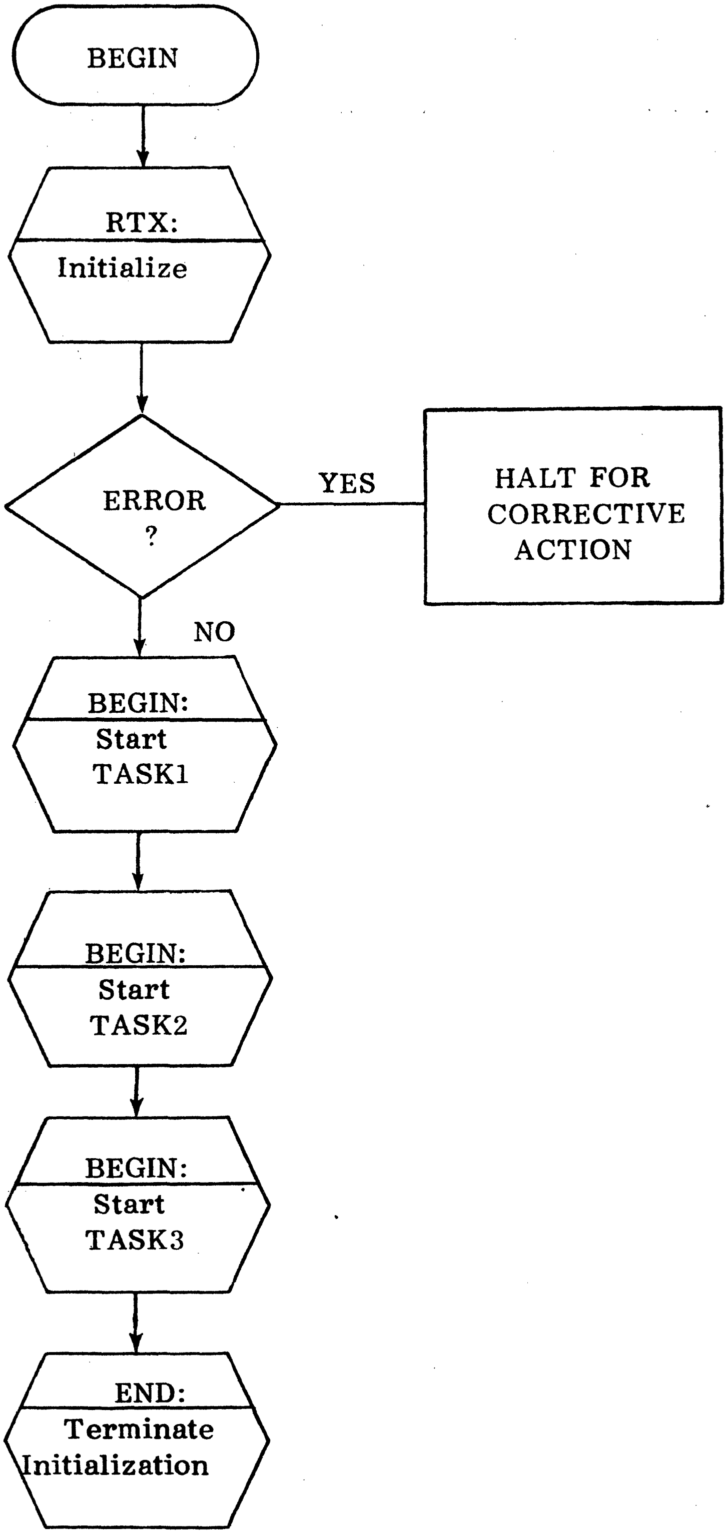


Figure 6-1. RTX Demo Program - Flow Diagram (Sheet 1)

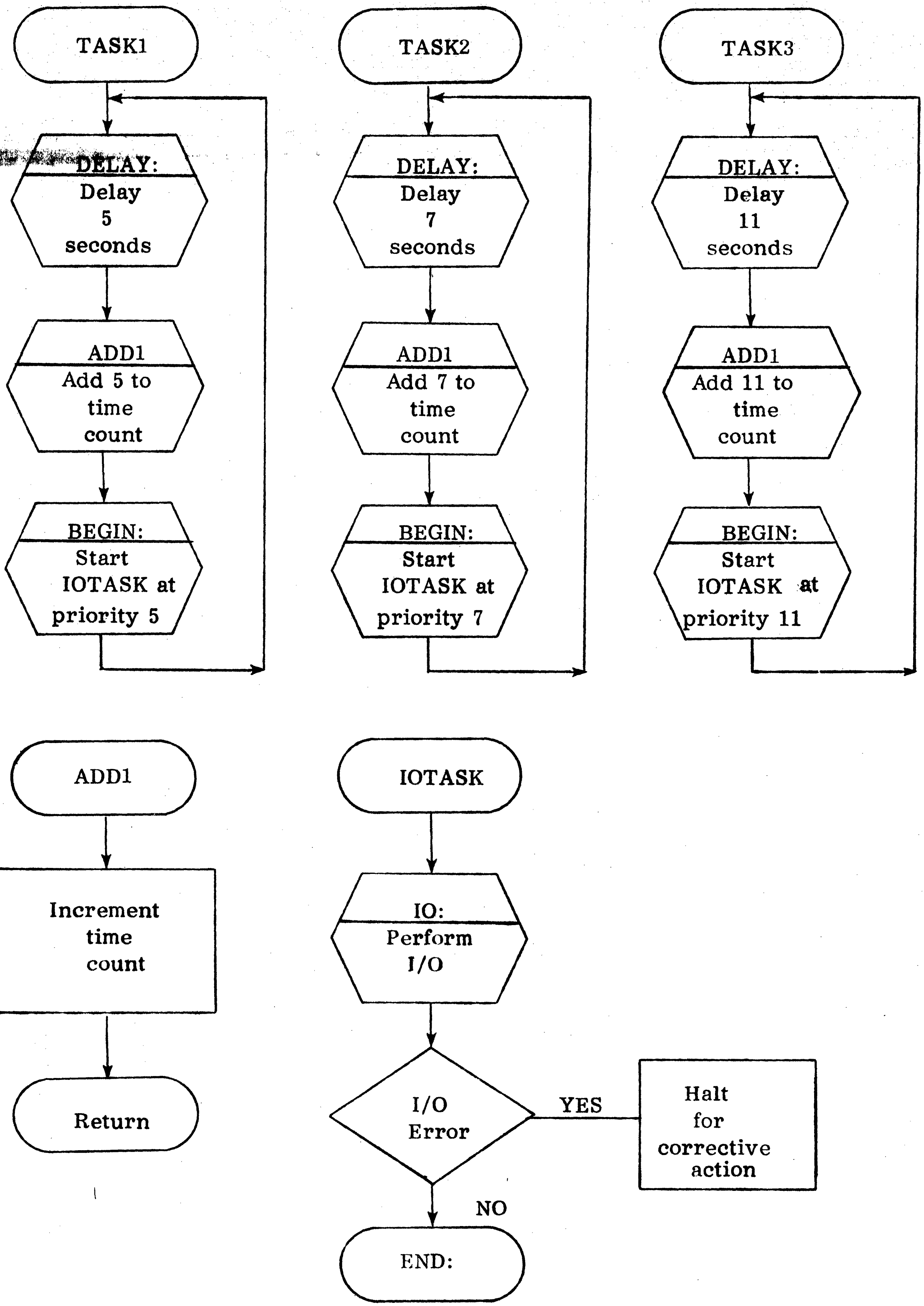


Figure 6-1. RTX Demo Program - Flow Diagram (Cont'd)

```
PAGE  0001  03/30/77  11:25:27     RTX DEMO PROGRAM 93300/01-13F1
MACRO2 (A2) SI= DE[illegible]    BO=

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:OAT
0026                             EXTR    RTX:,BEGIN:,END:
0027                             EXTR    SUBR:,SUBX:,DELAY:,IO:
0028                             EXTR    PAUSE:
0029        0000                 IFF     LSI305
0030                             EXTR    O:TY00
0031                             ENDC
0035        0014         NOACTS  EQU     20
0036   0000                      REL     0
0037        0000         BEGIN   EQU     $
0038   0000 B000 B0B0            LDA     ='00'     RESET
0039   0001 9A89 008B            STA     T1        .
0040   0002 9A89 008C            STA     T1+1      . ALL
0041   0003 9AA3 00A7            STA     T2        .
0042   0004 9AA3 00A8            STA     T2+1      . ELAPSED
0043   0005 9AB0 00C3            STA     T3        .
0044   0006 9AB0 00C4            STA     T3+1      . TIMES
```

```
PAGE  0002  03/30/77  11:25:27    RTX DEMO PROGRAM 93300/01-13E1
MACRO2 (A2) SI= DEMOS   BO=            LSI-2 RTX DEMO #00-93300-13E1

0045  0007 F900 0000             JST     RTX:      START RTX
0046  0008 0014                  DATA    NOACTS    NUMBER OF ACTIVIETES
0047  0009 000E                  DATA    WKAREA    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        WKAREA RES     NOACTS+NOACTS+NOACTS+NOACTS+NOACTS,0
```

```
PAGE  0003  03/30/77  11:25:27     RTX DEMO PROGRAM 93300/01-13E1
MACRO2 (A2) SI= DEMOS   BO=            LSI-2 RTX DEMO #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
0061                      *
0062                      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
0063                      *
0064  0075 F900 0000           JST    BEGIN:
0065  0076 0098                DATA   TASK2    START TASK 2
0066  0077 0064                DATA   100
0067                      *
0068                      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
0069                      *
0070  0078 F900 0000           JST    BEGIN:
0071  0079 00B4                DATA   TASK3
0072  007A 0064                DATA   100
0073                      *
0074                      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
0075                      *
0076  007B F900 0000           JST    END:     HALT INITIALIZE ROUTINE
```

```
PAGE  0004  03/30/77   11:25:27     RTX DEMO PROGRAM 93300/01-13E1
MACRO2 (A2) SI= DEMO5    BO=             LSI-2 RTX DEMO #00-93300-13E1

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    =I1       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    =IOB1     ADDRESS OF IOB
0092  0084 F900 0000           JST    BEGIN:    START COMMON TASK
0093  0085 00CF                DATA   IOTASK,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        BUFF1   EQU    $
0102  0088 D4C1                TEXT   'TASK1 '
      0089 D3CB
      008A B1A0
0103  008B B0B0        I1      TEXT   '0000'
      008C B0B0
0104  008D ACA0                TEXT   ', '
0105       008E        IOB1    EQU    $         IOB BLOCK
0106  008E 0000                DATA   0,0,0,0
      008F 0000
      0090 0000
```

```
PAGE  0005  05/30/77  11:25:27     RTX DEMO PROGRAM 93300/01-13E1
MACRO2 (A2) SI= DEMOS  BO=               LST-2 RTX 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 0086                  DATA    BUFF1     MSG BUFFER
0111   0096 0000                  DATA    0,0
       0097 0000
```

```
PAGE 0006  03/30/77  11:25:27     RTX DEMO PROGRAM 93300/01-13E1
MACRO2 (A2) SI= DEMOS   BO=           LSI-2 RTX DEMO #00-93300-13E1

0113                    *
0114  0098 F900 0000    TASK2   JST     DELAY:    WAIT FOR
0115  0099 02BC                 DATA    700,2     7 SECONDS
      009A 0002
0116                    *
0117                    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
0118                    *
0119  009B C707                 LAM     7         7 TIMES THROUGH ADD
0120  009C E000 00A7            LDX     =T2       POINTER TO NUMBER
0121  009D F900 0000            JST     SUBR:     CALL COMMON SUBROUTINE
0122  009E 00DA                 DATA    ADD1      TO ADD IN BCD
0123                    *
0124                    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
0125                    *
0126  009F E000 00AA            LDX     =IOB2     ADDRESS OF IOB BLOCK
0127  00A0 F900 0000            JST     BEGIN:    START COMMON TASK
0128  00A1 00CF                 DATA    IOTASK,7  TO DO I/O (PRIORITY IS 7)
      00A2 0007
0129                    *
0130                    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
0131                    *
0132  00A3 F608 0098            JMP     TASK2     KEEP GOING
0133                    *
0134                    * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
0135                    *
0136       00A4         BUFF2   EQU     $
0137  00A4 D4C1                 TEXT    'TASK2 '
      00A5 D3CB
      00A6 B2A0
0138  00A7 B0B0         T2      TEXT    '0000'
      00A8 B0B0
0139  00A9 ACA0                 TEXT    ', '
0140       00AA         IOB2    EQU     $         IOB BLOCK
0141  00AA 0000                 DATA    0,0,0,0
      00AB 0000
      00AC 0000
```

```
      00AD 0000
0142  00AE C3CF               DATA    'CO'      LUN
0143  00AF 0005               DATA    5         FUNC CODE (UNFORMATTED WRT)
0144  00B0 000C               DATA    12        MESSAGE BYTE LENGTH
0145  00B1 00A4               DATA    BUFF2     MSG BUFFER
0146  00B2 0000               DATA    0,0
      00B3 0000
```

```
0148                        *
0149  0084 F900 0000  TASK3  JST    DELAY:   WAIT FOR
0150  0085 044C              DATA   1100,3   11 SECONDS
      0086 0003
0151                        *
0152                        * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
0153                        *
0154  0087 C70B              LAM    11       11 TIME THROUGH ADD
0155  0088 E000 00C3         LDX    =T3      POINTER TO NUMBER
0156  0089 F900 0000         JST    SUBR:    CALL COMMON SUBROUTINE
0157  008A 000A              DATA   ADD1     TO ADD IN BCD
0158                        *
0159                        * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
0160                        *
0161  008B E000 00C5         LDX    =IOB3    ADDRESS OF IOB
0162  008C F900 0000         JST    BEGIN:   START COMMON TASK
0163  008D 00CF              DATA   IOTASK,11 TO DO I/O (PRIORITY IS 11)
      008E 000B
0164                        *
0165                        * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
0166                        *
0167  008F F60B 0084         JMP    TASK3    KEEP GOING
0168                        *
0169                        * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
0170                        *
0171       00C0       BUFF3  EQU    $
0172  00C0 D4C1              TEXT   'TASK3 '
      00C1 D3CB
      00C2 B3A0
0173  00C3 B0B0       T3     TEXT   '0000'
      00C4 B0B0
0174       00C5       IOB3   EQU    $             IOB BLOCK
0175  00C5 0000              DATA   0,0,0,0
      00C6 0000
      00C7 0000
      00C8 0000
```

```
PAGE  0009  03/30/77  11:20:27     RTX DEMO PROGRAM 93500/01-15F1
MACRO2 (A2) SI= DEMOS   SO=            LSI-2 RTX DEMO #00-93500-15F1

0176  00C9 C3CF               DATA   'CO'     LUN
0177  00CA 0006               DATA   6        FUNC CODE (ASCII WRITE)
0178  00CB 000A               DATA   10       MESSAGE BYTE LENGTH
0179  00CC 00C0               DATA   BUFF3    MESSAGE BUFFER ADRESS
0180  00CD 0000               DATA   0,0
      00CE 0000
```

```
PAGE  0010  03/30/77  11:25:27     RTX DEMO PROGRAM 93300/01-13E1
MACRO2 (A2) SI= DEMO3   BO=             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       00CF              IOTASK EQU    $         COMMON TASK ENTRY POINT
0190  00CF 6803                     SIN    2         AVOID INTERRUPTS HERE
0191  0000 EA01 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
```

```
PAGE  0011  03/30/77  11:25:27     RTX DEMO PROGRAM 93500/01-13E1
MACRO2 (A2) SI= DEMO         0=             LSI-2 RTX DEMO #00-93500-13E1

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  00D6 0F00              NEXT   SWM
0207  00D7 DA18 00F0                IMS    COUNT      DONE?
0208  00D8 F206 00DF                JMP    LX         NO, CONTINUE LOOP
0209  00D9 F900 0000                JST    SUBX:      YES, RETURN FROM COMMON
0210                         *                       SUBROUTINE
0211  00DA 0800              ADD1   ENT               ENTRY POINT
0212  00DB 9A14 00F0                STA    COUNT
0213  00DC 1326                     LLX    1          BYTE ADDRESS OF NUMBER
0214  00DD C203                     AXI    3          ADDRESS OF LEAST
0215                         *                       SIGNIFICANT DIGIT
0216  00DE EA12 00F1                STX    SAVEX      SAVE IT
0217  00DF C704              LX     LAM    4          DO ONLY 4 DIGITS
0218  00E0 9A11 00F2                STA    FOUR
0219  00E1 E20F 00F1                LDX    SAVEX      GET ADDRESS OF LSD
0220  00E2 0E00                     SBM
0221  00E3 B400 0000         ADD2   LDAB   @0         GET DIGIT
0222  00E4 F900 0000                JST    PAUSE:     ALLOW RE-ENTRY ATTEMPT
0223  00E5 0150                     IAR               ADD ONE (1)
0224  00E6 9C00 0000                STAB   @0         PUT IT BACK
0225  00E7 C0BA                     CAI    '9'+1      WAS IT '9'?
0226  00E8 F201 00EA                JMP    *+2        YES, GOTTA DO NEXT DIGIT
0227  00E9 F613 00D6                JMP    NEXT       NO, CHECK FOR DONE
0228  00EA C6B0                     LAP    '0'        CHANGE TO ZERO ('0')
0229  00EB 9C00 0000                STAB   @0         PUT IN DIGIT
0230  00EC 00A6                     DXR               POINT TO PREVIOUS DIGIT
0231  00ED DA04 00F2                IMS    FOUR       BUMP FOUR DIGIT COUNT
0232  00EE F606 00E3                JMP    ADD2       DO NEXT DIGIT
0233  00EF F619 00D6                JMP    NEXT       CONTINUE
```

```
PAGE  0012  03/30/77  11:25:27     RTX DEMO PROGRAM 93300/01-13E1
MACRO2 (A2) SI= DEMOS   SO=               LSI-2 RTX DEMO 400-93300-13E1

0234   00F0 0000          COUNT  DATA    0
0235   00F1 0000          SAVEX  DATA    0
0236   00F2 0000          FOUR   DATA    0
```

```
PAGE  0013  03/30/77  11:25:27     RTX DEMO PROGRAM 93500701-13E1
MACRO2 (A2) SI= DEMOS   BO=            LSI-2 RTX DEMO #00-93500-13E1

0238                          *
0239                          *  THIS IS THE UNIT ASSIGNMENT TABLE REQUIRED
0240                          *  BY IOX.  THERE IS ONLY ONE ENTRY, SINCE ONLY
0241                          *  ONE I/O DEVICE (TTY) IS USED IN THE
0242                          *  PROGRAM.
0243                          *
0244          00F3            UATTOP EQU    $          TOP OF UAT
0245     00F3 C3CF                   DATA   'CO'       LUN
0246          0000                   IFF    LSI305
0247     00F4 0000                   DATA   D:TY00   DIB ADDRESS FOR STD TTY
0248                                 ENDC
0252     00F5 FFFC            I:UAT  DATA   UATTOP-$-2  LENGTH OF UAT
0253                          *
0254          0000                   END    BEGIN

0000  ERRORS
0000  WARNING
```

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.

3. 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 request is processed, IOX BEGIN:'s 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:

```
    JST        END:
```

in 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

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

This segment contains, in the following order:

1. ZBG
2. IOX Control
3. RTX Services
4. CNSOL3 (If LSI-3 Version)

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.

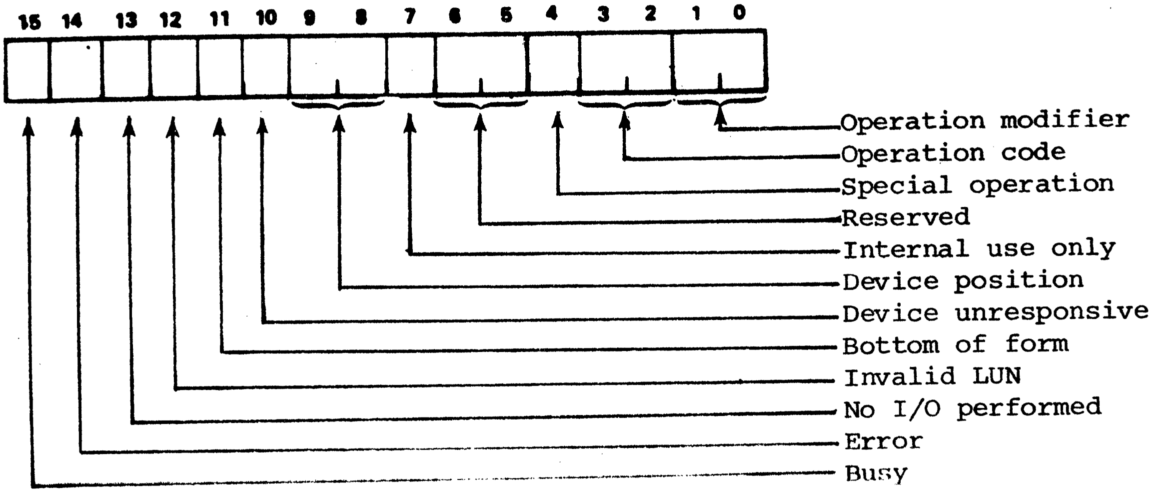
INPUT/OUTPUT BLOCK

<table>
<tr><th>Standard Name*</th><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th><th>word</th></tr>
<tr><td></td><td colspan="16">CHAIN POINTER (RESERVED FOR USE BY IOX)</td><td>0</td></tr>
<tr><td>IDT</td><td colspan="16">DEVICE TYPE</td><td>1</td></tr>
<tr><td>ICUN or ITCB</td><td colspan="16">DEVICE NUMBER</td><td>2</td></tr>
<tr><td>CN</td><td colspan="16">COORDINATION NUMBER</td><td>3</td></tr>
<tr><td>ILUN</td><td colspan="16">LOGICAL UNIT NAME/NUMBER</td><td>4</td></tr>
<tr><td>ISTA,IOP</td><td>BUSY</td><td>ERROR</td><td>NO I/O</td><td>BAD LUN</td><td>FORM</td><td>DEV. UN-RESP</td><td colspan="2">DEVICE POS.</td><td>INT. USE</td><td colspan="2">RES.</td><td>SO</td><td colspan="2">OP CODE</td><td colspan="2">OP MOD.</td><td>5</td></tr>
<tr><td>IRCNT</td><td colspan="16">REQUESTED COUNT</td><td>6</td></tr>
<tr><td>IBUFF</td><td colspan="16">BUFFER ADDRESS</td><td>7</td></tr>
<tr><td>IACNT</td><td>FLAG</td><td colspan="15">ACTUAL COUNT/PROMPT CHARACTERS</td><td>8</td></tr>
<tr><td>IDAA</td><td colspan="16">DIRECT ACCESS ADDRESS</td><td>9</td></tr>
</table>

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:		File Manager
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:		File Manager
00	Direct Access	Random Access
01	Unformatted, Sequential	Sequential
10	Formatted ASCII, Sequential	Sequential
11	Formatted Binary, Sequential	Sequential

For position:		File Manager
00	Absolute, Records	No change
01	Absolute, Files	No change
10	Relative, Records	No change
11	Relative, Files	No change

For function:		File Manager
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

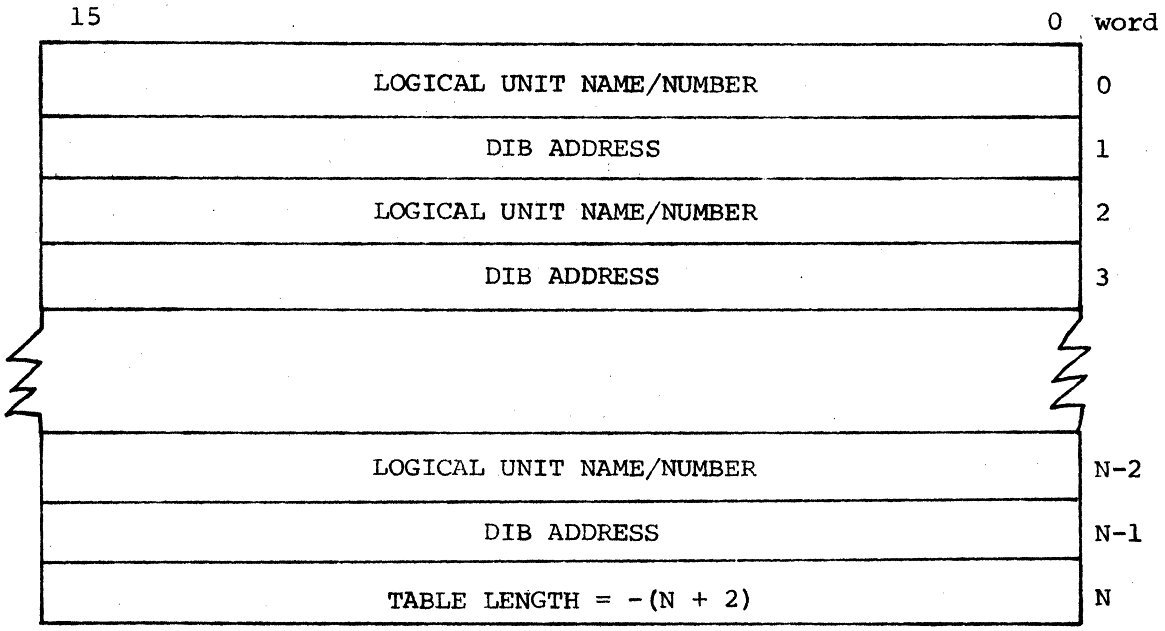
Word 8 contents will be assumed to be prompt characters if negative (bit 15 set). Bits 7-0 not equal to zero indicate two prompt characters; bits 7-0 equal to zero indicate only one prompt character (in bits 15-8).

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



LOGICAL UNIT NAME/NUMBER	0
DIB ADDRESS	1
LOGICAL UNIT NAME/NUMBER	2
DIB ADDRESS	3
LOGICAL UNIT NAME/NUMBER	N-2
DIB ADDRESS	N-1
TABLE LENGTH = -(N + 2)	N

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 x 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.

	Non-DIO	Fortran Non-DIO	DIO	Fortran DIO
Teletype Console	D:TY00	D:TYF0	D:TY0D	D:TYFD
Teletype Keyboard	D:TK00	D:TKF0	D:TK0D	D:TKFD
Teletype Tape Reader	D:TR00	D:TR00	D:TR0D	D:TR0D
Teletype Punch	D:TP00	D:TP00	D:TP0D	D:TP0D
CRT Console	D:TY00	D:TYF0	D:TV0D	--
CRT Keyboard	D:TK00	D:TKF0	D:TV0D	--
High Speed Paper Tape Reader	D:PR00	D:PR00	D:PR0D	D:PR0D
High Speed Paper Tape Punch	D:PP00	D:PP00	D:PP0D	D:PP0D
Centronics Line Printer	D:LP00	D:LPF0	D:LP0D	D:LPFD
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:IE0D	--

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.

3.1.8 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 REF 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).

The 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.

6. 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:

```
                         EXTR      INTQ:,INTP:
     TAG                 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:LP0 --- LINE PRINTER

     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 0800       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.

2. 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:
```

The UNRES: routine may be called when a "Device Unresponsive" status is returned. This routine masks off the lower two bits of CIB Word 1, which converts it to a SEL 4, or initialize instruction, and executes it to prevent any further interrupts, stores a zero count into CIB Word 28, (the actual transfer count), loads the A register with an Error/Unresponsive status, and exits to the EORST: routine.

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:.

3.3 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.

```
PAGE  0001  11/03/77  10:04:27   94500-10 RTX, IOX, COMX EQUATES
MACRO2 (A2) SI= MACROS  BO=          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         POINTER 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 (BTIS 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  11/03/77  10:04:27   94500-10 RTX, IOX, COMX EQUATES
MACRO2 (A2) SI= MACROS  BO=         RTXEQU  ---  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 TCB
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
```

```
PAGE  0003  11/03/77  10:04:27   94500-10 RTX, IOX, COMX EQUATES
MACRO2 (A2) SI= MACROS  BO=         RTXEQU  ---  EQUATES USED IN RTX

0051              * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
0052              *
0053              *     C  I  B     E Q U A T E S
0054              *
0055              * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
0056              *
0057     0000     CBOR   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     CEBTSK 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     REQCNT EQU  14
0068     000F     CJTBL  EQU  15    JUMP TABLE
0069     0013     CSPLOP EQU  19    POINTER TO SPECIAL OP PROC
0070     0014     CDFL   EQU  20    DELAY MODIFICATION
0071     0015     CINTR  EQU  21    POINTER TO INTERRUPT ADDRE
0072     0016     EXCESS EQU  22
0073     0017     CEOF   EQU  23
0074     0022     CHDST  EQU  34    HARDWARE STATUS
0075     0023     CNRS   EQU  35    NOISE RECORD BUFFER ADDRESS IN CIB
0076              *
0077              *     FILLED FROM IOB
0078              *
0079     0018     CIOB   EQU  24    IOB POINTER
0080     0019     COP    EQU  25    OPERATION CODE
0081     001A     CRCNT  EQU  26    REQUESTED COUNT
0082     001B     CBUFF  EQU  27    BUFFER ADDRESS
0083     001C     CTCNT  EQU  28    TRANSFER COUNT
0084     001D     CDAA   EQU  29    DIRECT ACCESS ADDRESS
0085              *
0086              *     FILLED FROM DIB
```

```
PAGE  0004  11/03/77  10:04:27   94500-10 RTX, IOX, COMX EQUATES
MACRO2 (A2) SI= MACROS  BO=          RTXEQU  ---  EQUATES USED IN RTX

0087                 *
0088       001E      CDIB   EQU    30        DIB POINTER
0089       001F      CFUN1  EQU    31        TEMP CELL 2 FOR FUNCTIONS
0090                 *
0091                 *      TEMP STORAGE USED BY IOX AND ITS DRIVERS
0092                 *
0093       0020      STATUS EQU    32        DEVICE STATUS WORD
0094       0021      CRTN   EQU    33        RETURN ADDRESS FROM I:SIO
0095       0022      CDCHN  EQU    34        START OF DATA CHAIN
0096       0023      CDCHN1 EQU    CDCHN+1
0097       0024      CDCHN2 EQU    CDCHN+2
0098       0025      CDCHN3 EQU    CDCHN+3
```

```
PAGE  0005  11/03/77  10:04:27   94500-10 RTX, ICX, COMX EQUATES
MACRO2 (A2) SI= MACROS  BO=           RTXEQU  ---  EQUATES USED IN RTX

0100                     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
0101                     *
0102                     *      D I B    E Q U A T E S
0103                     *
0104                     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
0105                     *
0106       0001          DCIB    EQU    1         CIB POINTER
0107       0004          DSW     EQU    4         DEVICE SPECIFICATION WORD
0108       0005          DT      EQU    5         DEVICE TYPE
0109       0006          DCUN    EQU    6         CONTROLLER & UNIT NUMBERS
0110       0007          DDEL    EQU    7         END OF BLOCK DELAY TIMES
0111       0008          DFUN    EQU    8         FUNCTION CODES & FLAGS
0112       0009          DULS    EQU    9         UPPER LIMITS
0113       000A          DERRC   EQU    10        ERROR COUNTER
0114       000B          DSTRT   EQU    11        DIO START ADDRESSES & MODES
0115       000B          DSECT   EQU    11        VERIFY FLAG, DRIVE #, STARTING SECTOR
0116       000C          DHEAD   EQU    12        SECTORS/TRACK & STARTING HEAD
0117       000C          DNRSZ   EQU    12        NOISE RECORD SIZE IN DIB
0118       000D          DCYL    EQU    13        SECTORS/CYLINDER & STARTING CYLINDER
0119       000D          DMOD    EQU    13        LAST MODE IN DIB
0120       000E          DEOD    EQU    14        NUMBER OF SECTORS IN FILE
0121       000F          DCSECT  EQU    15        FORMATTED SECTOR NO
0122                     *
0123                     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
0124                     *
0125                     *      INTERRUPT BLOCK  E Q U A T E S
0126                     *
0127                     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
0128       0000          NTAIO   EQU    0         I/O INSTRUCTION
0129       0001          NTCNT   EQU    1         COUNT FOR AUTO I/O
0130       0002          NTBUFF  EQU    2         BUFFER ADDRESS - 1
0131       0004          NTEOB   EQU    4         END-OF-BLOCK INTERRUPT
0132       0005          NTEOBA  EQU    5         ADDRESS FOR EOB INSTRUCTION
0133                     *
0134                     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
0135                     *
```

```
0136                       *      MISCELLANEOUS EQUATES
0137                       *
0138                       * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
0139                       *
0140           001D        ARROW  EQU    29        BACKARROW FLAG SAME AS CDA
0141           0003        EORMSK EQU    3         END OF RECORD MASK
0142           0004        EOFMSK EQU    4         END OF FILE MASK
0143           4000        I:ERR  EQU    :4000
0144           0800        I:BOP  EQU    :800
0145           0400        I:RES  EQU    :400
0146           0200        I:EOF  EQU    :200
0147           0100        I:BOD  EQU    :100      BEGINNING OF DEVICE STATUS BIT
0148           0300        I:EOD  EQU    :300
0149           2000        I:NOIO EQU    :2000
0150           4000        ERROR  EQU    :4000
0151           001F        OPMSK  EQU    31
0152           000C        OPMSK1 EQU    :C        OP-CODE MASK - NOT SPECIAL BIT
0153           001C        OPMSK2 EQU    :1C       FULL OP-CODE MASK
0154           0080        IOREL  EQU    :80
0155           0003        EORTYP EQU    3
0156           0004        PROMPT EQU    4
0157           0008        EOFTYP EQU    8
0158           0000               IFF    LSI305
0159           0005        IO:TMP EQU    5         TEMP USED BY IOX
0160                              ENDC
0164                       *
0165                       *      TITLE GENERATING MACRO
0166                       *       ACCEPTS TITLE AS TEXT STRING IN FIRST PARAMETER
0167                       *        THEN PART NUMBER AS SECOND PARAMETER
0168                       *      GENERATES PART NUMBER BASED ON LSI2/LSI305 FLAG
0169                       *
0170                              MACRO  TITLE
0171                              IFT    LSI305
0172                              TITL   #1,#2-11
0173                              ENDC
0174                              IFT    LSI2
```

```
PAGE  0007  11/03/77  10:04:27   94500-10 RTX, IOX, COMX EQUATES
MACRO2 (A2) SI= MACROS  BO=          RTXEQU  ---  EQUATES USED IN RTX

0175                        TITL   #1,#2-10
0176                        ENDC
0177                        ENDM
0178                  *
0179                  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
0180                  *
0181       0009       INITC  EQU    9        DEVICE INTTIALIZE BITS
0182                  *
```

```
PAGE  0008  11/03/77  10:04:27   94500-10 RTX, IOX, COMX EQUATES
MACRO2 (A2) SI= MACROS  BO=               COMX GENERAL EQUATES

0184                      *
0185                      *      PRIORITIES
0186                      *
0187        1FF2          CPDPRI EQU    8178     PROTOCOL DRIVER PRIORITY
0188        1FF8          CRPPRI EQU    8184     RECEIVE PROC PRIO
0189        1FF8          CFNPRI EQU    CRPPRI   FUNCTION PROCESSOR PRIO
0190        1FF8          CTDPRI EQU    CRPPRI   TRANSMIT DATA PROC PRIO
0191        1FF8          CTLPRI EQU    CRPPRI   TRANSMIT LINE PROC PRIO
0192        1FFE          CINPRI EQU    8190     INTERRUPT TASK PRIO
0193        1FF9          C2IPRI EQU    CRPPRI+1 INTERRUPT TASK LOWER PRIORITY
0194        1FF9          CSEPRI EQU    CRPPRI+1 SPECIAL EXCEPTION PROCESSING PRIO
0195        1FFA          STIPRI EQU    8186     START INPUT PRIO
0196        1FFA          STOPRI EQU    STIPRI   START OUTPUT PRIO
0197                      *
0198                      *      STOP TABLE FLAGS
0199                      *
0200        8000          STP1ST EQU    :8000    WORD 0  BIT(S)  15   1ST STOP CHAR OF 2 FOUND
0201        4000          STPEPN EQU    :4000    WORD 0  BIT(S)  14   ERROR IF 1ST FOUND,PAIR NOT
0202        2000          STPECN EQU    :2000    WORD 0  BIT(S)  13   ERROR IF CHAR NOT STOP CHAR
0203        0100          STPESF EQU    :0100    WORD 0  BIT(S)  12  ERROR IF STOP CHAR FOUND
0204        0800          STPTSC EQU    :0800    WORD 0  BIT(S)  11   TRANSFER STOP CHAR(S)
0205        0400          STPCKS EQU    :0400    WORD 0  BIT(S)  10   ACCUMULATE CHECKSUM ON SC
0206        00FF          STPCNT EQU    :00FF    WORD 0  BIT(S)  0-7  COUNT
0207                      *
0208                      *      DATA SET SIGNALS ROUTINES PARAMETER WORD
0209                      *
0210        0001          DSSTM  EQU    :0001    BIT  0      TRANSMIT MODE
0211        0002          DSSRM  EQU    :0002    BIT  1      RECEIVE MODE
0212        0004          DSSEM  EQU    :0004    BIT  2      ECHO MODE
0213        0008          DSSLB  EQU    :0008    BIT  3      LOOP-BACK MODE
0214        0010          DSSDTR EQU    :0010    BIT  4      DATA TERMINAL READY
0215        0020          DSSRTS EQU    :0020    BIT  5      REQUEST TO SEND
0216        0100          DSSDSR EQU    :0100    BIT  8      DATA SET READY
0217        0200          DSSCTS EQU    :0200    BIT  9      CLEAR TO SEND
0218        0400          DSSCD  EQU    :0400    BIT 10      CARRIER DETECT
0219        0800          DSSRI  EQU    :0800    BIT 11      RING INDICATOR
```

```
 7  10:04:27   94500-10 RTX, IOX, COMX EQUATES
 OS  RO=              COMX GENERAL EQUATES

                    DSSSQ  EQU    :1000     BIT 12      SIGNAL QUALITY
                    DSSSCR EQU    :2000     BIT 13      SECONDARY CHANNEL RECEIVE
                    DSSRL  EQU    :003F     SIGNALS TO RAISE OR LOWER
                    DSSWT  EQU    :3F00     SIGNALS TO WAIT FOR
                    *
                    *      DEVICE DEPENDENT ROUTINES TABLE DISPLACEMENTS
                    *
                    DDRLNP EQU    1         SET LINE PARAMETERS
                    DDRSYN EQU    2         SET SYNCH CHARACTER
                    DDRSPC EQU    3         SET SPECIAL CHARACTER
                    DDRINT EQU    4         INITIALIZE LINE
                    DDRENL EQU    5         ENABLE LINE
                    DDRDSL EQU    6         DISABLE LINE
                    DDRSTI EQU    7         START INPUT
                    DDRSPI EQU    8         STOP INPUT
                    DDRSTO EQU    9         START OUTPUT
                    DDRSPO EQU    10        STOP INPUT
                    DDRIC1 EQU    11        INPUT CHARACTER PROCESSOR 1
                    DDRIC2 EQU    12        INPUT CHARACTER PROCESSOR 2
                    DDROC1 EQU    13        OUTPUT CHARACTER PROCESSOR 1
                    DDROC2 EQU    14        OUTPUT CHARACTER PROCESSOR 2
                    DDRRON EQU    15        RAISE AND/OR WAIT FOR DATA SET SIGNALS ON
                    DDRLOF EQU    16        LOWER AND/OR WAIT FOR DATA SET SIGNALS OFF
                    DDRRSY EQU    17        RESYNC LINE
                    DDRNSY EQU    18        ISSUE NEW SYNC
                    DDRBRK EQU    19        ISSUE BREAK
                    DDRRVC EQU    20        REVERSE CHANNEL
                    DDRLTP EQU    21        LINE TRANSMIT PROCEDURE
                    *
                    *      SYNCH MODEM CONTROLLER COMMAND WORD
                    *
                    SCCRTS EQU    :01       BIT  0     REQUEST TO SEND
                    SCCTM  EQU    :02       BIT  1     TRANSMIT MODE
                    SCCRM  EQU    :04       BIT  2     RECEIVE MODE
                    SCCIE  EQU    :08       BIT  3     INTERRUPTS ENABLE
          0010      SCCDTR EQU    :10       BIT  4     DATA TERMINAL READY
```

```
PAGE  0010  11/03/77  10:04:27   94500-10 RTX, IOX, COMX EQUATES
MACRO2 (A2) SI= MACROS  BO=                COMX GENERAL EQUATES

0256        0020         SCCSSC EQU    :20       BIT  5      SEARCH SPECIAL CHARACTER
0257        0040         SCCALS EQU    :40       BIT  6      ALLOW SYNCH CHARACTERS
0258        0080         SCCLB  EQU    :80       BIT  7      LOOP-BACK MODE
0259                     *
0260                     *     SYNCH MODEM CONTROLLER STATUS WORD
0261                     *
0262        0001         SCSDSR EQU    :01       BIT  0      DATA SET READY
0263        0002         SCSCTS EQU    :02       BIT  1      CLEAR TO SEND
0264        0004         SCSCD  EQU    :04       BIT  2      CARRIER DETECT
0265        0008         SCSRI  EQU    :08       BIT  3      RING INDICATOR
0266        0010         SCSSQ  EQU    :10       BIT  4      SIGNAL QUALITY
0267        0060         SCSERC EQU    :60       BIT 5-6     ERROR CODE
0268        0020         SCSPE  EQU    :20       PARITY ERROR CODE
0269        0040         SCSRDO EQU    :40       RECEIVE DATA OVERRUN CODE
0270        0060         SCSTDO EQU    :60       TRANSMIT DATA OVERRUN CODE
0271        0080         SCSSCD EQU    :80       BIT  7      SPECIAL CHARACTER DETECT
0272        001F         SCSLST EQU    :1F       SMC STATUS BITS COMMON WITH LSTNS
0273        0003         SCSPES EQU    3         SHIFT COUNT TO ALIGN PARITY ERROR IN LSTNS
0274        0004         SCSROS EQU    4         SHIFT COUNT TO ALIGN RECV DATA OVERRUN IN LSTNS
0275        0006         SCSSCS EQU    6         SHIFT COUNT TO ALIGN SP CHAR DETECT IN LSTNS
```

```
0277                   *
0278                   *      BISYNC PROTOCOL MASTER TABLE DISPLACEMENTS
0279                   *
0280        0001       CRLTBL EQU     1
0281                   *      CONTROL CHARACTER TABLE DISPACEMENTS
0282        0001       ACK0   EQU     1
0283        0002       ACK1   EQU     2
0284        0003       NAK    EQU     3
0285        0004       WACK   EQU     4
0286        0005       RVI    EQU     5
0287        0006       ENQ    EQU     6
0288        0007       STX    EQU     7
0289        0008       ITB    EQU     8
0290        0009       ETB    EQU     9
0291        000A       ETX    EQU     10
0292        000B       EOT    EQU     11
0293        000C       DLEENQ EQU     12
0294        000D       DLESTX EQU     13
0295        000E       DLEITB EQU     14
0296        000F       DLEETB EQU     15
0297        0010       DLEETX EQU     16
0298        0011       DLEEOT EQU     17
0299        0012       TTD    EQU     18
0300                   *
0301        0002       ENQTBL EQU     2
0302        0003       STXTBL EQU     3
0303        0004       ITBTBL EQU     4
0304        0005       ETBTBL EQU     5
0305        0006       ENDTBL EQU     6
0306        0007       AKOTBL EQU     7
0307        0008       EAKTBL EQU     8
0308        0009       UAKTBL EQU     9
0309        000A       PADTBL EQU     10
0310                   *
0311                   *      CIB EQUATES
0312                   *
```

```
PAGE  0012  11/03/77  10:04:27   94500-10 RTX, IOX, COMX EQUATES
MACRO2 (A2) SI= MACROS  BO=               COMX PROTOCOL DRIVER TABLES EQUATES

0313        0014        CPFLGS EQU    20        WORD 20 BIT(S)  0-15 PROTOCOL DRIVER FLAGS
0314        0007        CSTATE EQU    :0007     WORD 20 BIT(S)  0-2 CURRENT PROTOCOL STATE
0315        0008        CRDRQ  EQU    :0008     WORD 20 BIT(S)   3   READ REQUEST OUTSTANDING
0316        0010        CWRRQ  EQU    :0010     WORD 20 BIT(S)   4   WRITE REQUEST OUTSTANDIT
0317        0020        CACKFG EQU    :0020     WORD 20 BIT(S)   5   ACK FLAG
0318        0040        CXPRNT EQU    :0040     WORD 20 BIT(S)   6   TRANSPARENT MODE FLAG
0319        0080        CITBFG EQU    :0080     WORD 20 BIT(S)   7   INTER BLOCK FLAG
0320        0300        CTOCNT EQU    :0300     WORD 20 BIT(S) 8-9  TIMEOUT COUNTER
0321        0100        CTOINC EQU    :0100     INCREMENT FOR TIMEOUTS
0322        0300        CTOMAX EQU    :0300     MAXIMUM TIMEOUTS
0323        1C00        CAKCNT EQU    :1C00     WORD 20 BIT(S) 10-12 INVALID ACK COUNTER
0324        0400        CAKINC EQU    :0400     INCREMENT FOR INV ACKS
0325        0C00        CAKMAX EQU    :0C00     MAXIMUM INVALID ACKS
0326        E000        CNKCNT EQU    :E000     WORD 20 BIT(S) 13-15 NAK COUNTER
0327        2000        CNKINC EQU    :2000     INCREMENT FOR NAK COUNTER
0328        6000        CNKMAX EQU    :6000     MAXIMUM NAK COUNT
0329        0015        CPTBL  EQU    21        WORD 21 BIT(S)  0-15 PROTOCOL MASTER TBL ADDR
0330        0017        CLSTAT EQU    23        WORD 23 BIT(S)       LINE STATUS WORD
0331        0001        CLSTDO EQU    :0001     WORD 23 BIT(S)   0   TRANSMIT DATA OVERRUN
0332        0008        CLSTFE EQU    :0008     WORD 23 BIT(S)   3   TRANSMIT FRAMING ERROR
0333        0020        CLSTTO EQU    :0020     WORD 23 BIT(S)   5   TRANSMIT TIMEOUT
0334        0040        CLSCTS EQU    :0040     WORD 23 BIT(S)   6   LOSS OF CTS
0335        0080        CLSDSR EQU    :0080     WORD 23 BIT(S)   7   LOSS OF DSR
0336        0100        CLSCD  EQU    :0100     WORD 23 BIT(S)   8   LOSS OF CD
0337        0200        CLSSTO EQU    :0200     WORD 23 BIT(S)   9   SPEACIAL TIMEOUT
0338        0400        CLSRTO EQU    :0400     WORD 23 BIT(S)  10   RECEIVE TIMEOUT
0339        1000        CLSRFE EQU    :1000     WORD 23 BIT(S)  12   RECEIVE FRAMING ERROR
0340        4000        CLSPE  EQU    :4000     WORD 23 BIT(S)  14   PARITY ERROR
0341        8000        CLSRDO EQU    :8000     WORD 23 BIT(S)  15   RECEIVE DATA OVERRUN
0342        03FF        CLSOER EQU    :03FF     WORD 23 BIT(S)  0-9   OUTPUT ERRORS
0343        FF80        CLSIER EQU    :FF80     WORD 23 BIT(S)  7-15  INPUT ERRORS
0344        0FF8        CLSTRM EQU    :0FF8     WORD 23 BIT(S)  3-11  TERMINATION ERRORS
0345        03F8        CLSOTE EQU    :03F8     WORD 23 BIT(S)  3-9   OUTPUT TERMINATION ERRORS
0346        0F80        CLSITE EQU    :0F80     WORD 23 BIT(S)  7-11  INPUT TERMINATION ERRORS
0347        001F        CFLGS  EQU    31        WORD 31 BIT(S)  0-15 CIB WDS, PROTOCOL FLAGS
0348        0022        CPDIB  EQU    34        WORD 34 BIT(S)  0-15 CURRENT DIB ADDR FOR [illegible]
```

```
PAGE  0013  11/03/77  10:04:27   94500-10 RTX, IOX, COMX EQUATES
MACRO2 (A2) SI= MACROS  BO=           COMX PROTOCOL DRIVER TABLES EQUATES

0349        0023        CURCNT EQU    35        WORD 35 BIT(S)  0-15 TEMP FOR USERS REC COUNT
0350        0023        CPST1  EQU    35        WORD 35 BIT(S)  0-15 TEMP FOR SUBROUTINES
0351        0024        CUBUFF EQU    36        WORD 36 BIT(S)  0-15 TEMP FOR USER BUFF ADDR
0352        0024        CPST2  EQU    36        WORD 36 BIT(S)  0-15 TEMP FOR SUBROUTINES
0353        0025        LPCIB  EQU    37        LENGTH OF PROT CIB - CPST2+1
0354                    *
0355                    *     DIB EQUATES
0356                    *
0357        0007        DCTMR  EQU    7         WORD 7  BIT(S)  0-15 INPUT CHAR TIMER VALUE
0358        0008        DFLGS  EQU    8         WORD 8  BIT(S) 0-15  PROTOCOL FLAGS
0359        000A        DDFLGS EQU    10        WORD 10 BIT(S)  0-7  DEVICE DEPENDENT FLAGS
0360        0001        DDFMAS EQU    :0001     WORD 10 BIT(S)   0   MASTER STATION
0361        0004        DDFACU EQU    :0004     WORD 10 BIT(S)   3   AUTO-CALL THIS LINE
0362        000A        DPFLGS EQU    10        WORD 10 BIT(S)  8-15 PROTOCOL DEPENDENT FLAGS
0363        0100        DPFUWX EQU    :0100     WORD 10 BIT(S)   8   UNFORMATTED WRITES XPRNT
0364        0200        DPFAWX EQU    :0200     WORD 10 BIT(S)   9   ALL WRITES XPRNT
0365        0400        DPFUSR EQU    :0400     WORD 10 BIT(S)  10   USER LINE CONTROL
0366        0800        DPFDIS EQU    :0800     WORD 10 BIT(S)  11   DISCONNECT ON WRITE FILEMAR
0367        000B        DBLKS  EQU    11        WORD 11 BIT(S)  0-15 MAX BLOCK SIZE
0368        000C        DLPARM EQU    12        WORD 12 BIT(S)       LINE PARAMETERS
0369        0001        DLPPEN EQU    :0001     WORD 12 BIT(S)   0   PARITY ENABLE
0370        0002        DLPPSL EQU    :0002     WORD 12 BIT(S)   1   PARITY SELECT,1=EVEN
0371        000C        DLPWSL EQU    :000C     WORD 12 BIT(S)  2-3  WORD LENGTH SELECT
0372        000D        DIXLTR EQU    13        WORD 13 BIT(S)  0-15 INPUT XLATE ROUTINE ADDR
0373        000E        DOXLTR EQU    14        WORD 14 BIT(S)  0-15 OUTPUT XLATE ROUTINE ADDR
0374        000F        DLUN   EQU    15        WORD 15 BIT(S)  0-15 DEVICE LUN
0375        0010        DDIB   EQU    16        WORD 16 BIT(S)  0-15 LINE DIB THREAD
0376        0011        DSRTN  EQU    17        WORD 17 BIT(S)  0-15 SUBROUTINE RTN ADDR
0377        0012        DTMP1  EQU    18        WORD 18 BIT(S)  0-15 TEMP CELL 1
0378        0013        DTMP2  EQU    19        WORD 19 BIT(S)  0-15 TEMP CELL 2
0379        0014        DCKSR  EQU    20        WORD 20 BIT(S)  0-15 CHECKSUM ROUTINE ADDR
0380        0015        DCKSF  EQU    21        WORD 21 BIT(S)       CHECKSUM FLAGS
0381        00FF        DCKSM  EQU    :00FF     WORD 21 BIT(S)  0-7  CHECKSUM CHAR MASK
0382        8000        DCKSL  EQU    :8000     WORD 21 BIT(S)  15   CHECKSUM LENGTH,1=2 CHARS
0383        0016        DCKSP  EQU    22        WORD 22 BIT(S)  0-15 CHECKSUM POLYNOMIAL
0384        0017        DCKSA  EQU    23        WORD 23 BIT(S)  0-15 CHECKSUM ACCUMULATION
```

```
PAGE  0014  11/03/77  10:04:27   94500=10 BTX, IOX, CCMX EQUATES
MACRO2 (A2) SI= MACROS  BO=           CCMX PROTOCOL DRIVER TABLES EQUATES

0385         0018         DHDRCT EQU    24        WORD 24 BIT(S)   0-15 HEADER BYTE COUNT
0386         0019         DHDC12 EQU    25        WORD 25 BIT(S)   0-15 HEADER BYTES 1 + 2
0387         001A         DHDC34 EQU    26        WORD 26 BIT(S)   0-15 HEADER BYTES 3 + 4
0388         001B         DHDC56 EQU    27        WORD 27 BIT(S)   0-15 HEADER BYTES 5 + 6
0389         001C         DHDC78 EQU    28        WORD 28 BIT(S)   0-15 HEADER BYTES 7 + 8
0390         001D         DHDC9A EQU    29        WORD 29 BIT(S)   0-15 HEADER BYTES 9 + 10
0391                      *
0392                      *      IOB EQUATES
0393                      *
0394         0003         IOPMOD EQU    :0003     WORD 5  BIT(S)   0-1  OP CODE MODIFIER
0395         0000         IOPMR0 EQU    :0000     WORD 5  BIT(S)   0-3  READ MOD 00, READ CHARS
0396         0001         IOPMR1 EQU    :0001     WORD 5  BIT(S)   0-3  READ MOD 01, DELETE CHARS
0397         0002         IOPMR2 EQU    :0002     WORD 5  BIT(S)   0-3  READ MOD 10, CHECKSUM VERFY
0398         0001         IOPMW1 EQU    :0001     WORD 5  BIT(S)   0-3  WRITE MOD 01, WRITE CHARS
0399         000A         IOPMF2 EQU    :000A     WORD 5  BIT(S)   0-3  FUNCTION OP CODE, [illegible]
0400         000C         IOPOPC EQU    :000C     WORD 5  BIT(S)   2-3  OPERATION CODE
0401         0000         IOPOPR EQU    :0000     WORD 5  BIT(S)   2-3  READ OP CODE
0402         0004         IOPOPW EQU    :0004     WORD 5  BIT(S)   2-3  WRITE OP CODE
0403         0008         IOPOPF EQU    :0008     WORD 5  BIT(S)   2-3  FUNCTION OP CODE
0404         000C         IOPOPP EQU    :000C     WORD 5  BIT(S)   2-3  POSITION OP CODE
0405         0010         IOPSPO EQU    :0010     WORD 5  BIT(S)    4   SPECIAL OPERATION
0406         0020         IOPSTR EQU    :0020     WORD 5  BIT(S)    5   0= STORE, 1= DISPLAY
0407         0040         IOPTXT EQU    :0040     WORD 5  BIT(S)    6   0= CONTROL, 1= TEXT
0408         0300         IOPTMD EQU    :0300     WORD 5  BIT(S)   8-9  OP CODE TRANSFER MODIFIER
0409         0100         IOPTRI EQU    :0100     WORD 5  BIT(S)   8-9  READ INITIAL TRANSFER
0410         0100         IOPTWI EQU    :0100     WORD 5  BIT(S)   8-9  WRITE INTERMEDIATE TRANSFER
0411         0200         IOPTWT EQU    :0200     WORD 5  BIT(S)   8-9  WRITE TERMINATING TRANSFER
0412         0300         IOPTWF EQU    :0300     WORD 5  BIT(S)   8-9  WRITE ENDING TRANSFER
0413         0400         IOPXLT EQU    :0400     WORD 5  BIT(S)   10   TRANSLATE
0414         0800         IOPCKS EQU    :0800     WORD 5  BIT(S)   11   ACCUMULATE CHECKSUM
0415         1000         IOPHDR EQU    :1000     WORD  5 BIT(S)   12   HEADER REQUESTED
0416         2000         IOPSCK EQU    :2000     WORD 5  BIT(S)   13   CHECK CHAR FOR STOP
0417                      *   NOTE: IOPSCK IS IN THE SAME POSITION AS STPFCN IN STOP TABLE FLAG
0418         0000         ISTAOK EQU    :0000     WORD 5  BIT(S) 10-14 NO ERRORS, ACK
0419         0000         ISTETB EQU    :0000     WORD 5  BIT(S)   8-9  BSC- ETB RECEIVED
0420         0100         ISTETX EQU    :0100     WORD 5  BIT(S)   8-9  BSC- ETX RECEIVED
```

```
PAGE  0015  11/03/77  10:04:27   90500-10 RTX, IOX, COMX EQUATES
MACRO2 (A2) SI= MACROS  BO=            COMX PROTOCOL DRIVER TABLES EQUATES

0421         0200         ISTEOT EQU     :0200     WORD 5  BIT(S)  8-9  ESC- EOT RECEIVED
0422         0300         ISTITB EQU     :0300     WORD 5  BIT(S)  8-9  ESC- ITB RECEIVED
0423         4000         ISTLER EQU     :4000     WORD 5  BIT(S) 10-14 ERROR- LINE ERROR
0424         4400         ISTDUN EQU     :4400     WORD 5  BIT(S) 10-14 ERROR- DEVICE UNRESPONSIVE
0425         4C00         ISTICS EQU     :4C00     WORD 5  BIT(S) 10-14 ERROR- INVALID COMMAND
0426         4800         ISTHLT EQU     :4800     WORD 5  BIT(S) 10-14 ERROR-HALT I/O
0427         5000         ISTILU EQU     :5000     WORD 5  BIT(S) 10-14 ERROR- INVALID LU
0428         5400         ISTRRJ EQU     :5400     WORD 5  BIT(S) 10-14 ERROR- REQUEST REJECTED
0429         5800         ISTBFL EQU     :5800     WORD 5  BIT(S) 10-14 ERROR- INPUT BUF FULL, [illegible]
0430         5C00         ISTDRJ EQU     :5C00     WORD 5  BIT(S) 10-14 ERROR- DATA REJECTED
0431         3C00         ISTERC EQU     :3C00     WORD 5  BIT(S) 10-13 ERROR CODE
0432         0007         ICCHAR EQU     IBUFF     COMX CURRENT/LAST CHAR(S) SENT BY DEVICE [illegible]
0433         0008         ICTBL  EQU     IACNT     COMX CONTROL SEQUENCE TABLE ADDR
0434         0009         IPCMR  EQU     IDAA      COMX PROTOCOL CHAR MODIF ROUTINE
0435         000A         ICTMR  EQU     10        COMX CHARACTER TIMER
```

```
PAGE  0016  11/03/77  10:04:27   94500-10 ETX, ICX, COMX EQUATES
MACRO2 (A2) SI= MACROS  BO=            COMX DEVICE DRIVER EQUATES

0437                      *
0438                      *   CIB DEFINITIONS
0439                      *
0440       0006           CINTSK EQU    6        WORD  6 BIT(S)  0-15 INPUT INT TASK ADDR
0441       0007           COTTSK EQU    7        WORD 7 BIT(S)   0-15 OUTPUT INT TASK ADDR
0442       000D           CTIMER EQU    13       WORD 13 BIT(S)  0-15 TIMER/INTRPT STATE FLAG
0443       000E           CLIB   EQU    14       WORD 14 BIT(S)  0-15 LIB ADDRESS
0444       0014           CINRTN EQU    20       WORD 20 BIT(S)  0-15 INPUT INT ROUTINE ADDR
0445       0015           CDSRTN EQU    21       WORD 21 BIT(S)  0-15 SUBR RETURN ADDR
0446       0016           CSTPTB EQU    22       WORD 22 BIT(S)  0-15 IOB WD 8- STOP TABLE ADDR
0447       001B           CFUNP  EQU    CBUFF    WORD 27 BIT(S)  0-15 FUNCTION PARAMETER WORD
0448       001D           CCMR   EQU    29       WORD 29 BIT(S)  0-15 IOB WD 9- CHAR PROC ROUTN
0449       0022           CCHAIN EQU    34       WORD 34 BIT(S)  0-15 DEVICE CIB CHAIN (OC::)
0450       0023           CCCADR EQU    35       WORD 35 BIT(S)  0-15 CURRENT CHAR ADDR
0451       0024           CCCHAR EQU    36       WORD 36 BIT(S)  0-15 CURRENT CHAR(S)
0452       0025           CTMP21 EQU    37       WORD 37 BIT(S)  0-15 TEMP CELL 1
0453       0026           CTMP22 EQU    38       WORD 38 BIT(S)  0-15 TEMP CELL 2
0454       0027           CSTPFG EQU    39       WORD 39 BIT(S)       STOP TABLE FLAGS
0455       0028           CXLTR  EQU    40       WORD 40 BIT(S)  0-15 TRANSLATE ROUTINE ADDR
0456       0029           CAFOR  EQU    41       WORD 41 BIT(S)  0-15 A REG RETURNED TO CALLER AT
0457       002A           CINTBF EQU    42       WORD 42 BIT(S)  0-15 CURRENT INTERRUPT BUFFER AD
0458       002B           CINTCT EQU    43       WORD 43 BIT(S)  0-15 ADDITIONAL INTERRUPT BYTE C
0459       002C           CCHMSK EQU    44       WORD 44 BIT(S)  0-15 CHARACTER BIT MASK
0460       002D           LDICIB EQU    45       LENGTH OF INPUT CIB - CCHMSK+1
0461       002D           LDOCIB EQU    45       LENGTH OF OUTPUT CIB - CCHMSK+1
0462                      *
0463                      *   LIB (LINE INFORMATION BLOCK) DEFINITIONS
0464                      *
0465       0000           LSTNS  EQU    0        WORD 0  BIT(S)       STANDARD LINE STATUS WORD
0466       003F           LSTLSG EQU    :003F    WORD 0  BIT(S)  0-5  LINE SIGNALS
0467       0001           LSTDSR EQU    :0001    WORD 0  BIT(S)   0   DATA SET READY
0468       0002           LSTCTS EQU    :0002    WORD 0  BIT(S)   1   CLEAR TO SEND
0469       0004           LSTCD  EQU    :0004    WORD 0  BIT(S)   2   CARRIER DETECT
0470       0008           LSTRI  EQU    :0008    WORD 0  BIT(S)   3   RING INDICATOR
0471       0010           LSTSQ  EQU    :0010    WORD 0  BIT(S)   4   SIGNAL QUALITY
0472       0020           LSTSCR EQU    :0020    WORD 0  BIT(S)   5   SECONDARY CHANNEL RECEIVE
```

```
PAGE  0017  11/03/77  10:04:27   94500-1C BTX, IOX, COMX EQUATES
MACRO2 (A2) SI= MACROS  BO=              COMX DEVICE DRIVER EQUATES

0473        0040        LSTTFE EQU    :0040     WORD 0  BIT(S)    6   TRANSMIT FRAMING ERROR
0474        0080        LSTRFE EQU    :0080     WORD 0  BIT(S)    7   RECEIVE FRAMING ERROR
0475        0100        LSTPE  EQU    :0100     WORD 0  BIT(S)    8   PARITY ERROR
0476        0200        LSTTDO EQU    :0200     WORD 0  BIT(S)    9   TRANSMIT DATA OVERRUN
0477        0400        LSTRDO EQU    :0400     WORD 0  BIT(S)   10   RECEIVE DATA OVERRUN
0478        0800        LSTSCD EQU    :0800     WORD 0  BIT(S)   11   SECONDARY CHANNEL RECEIVE
0479        1000        LSTDRD EQU    :1000     WORD 0  BIT(S)   12   DATA READY
0480        2000        LSTTMT EQU    :2000     WORD 0  BIT(S)   13   TRANSMITTER EMPTY
0481        0001        LSOES  EQU    1         WORD 1  BIT(S)        SOFTWARE LINE STATUS
0482        0001        LSFTM  EQU    :0001     WORD 1  BIT(S)    0   TRANSMIT MODE
0483        0002        LSFRM  EQU    :0002     WORD 1  BIT(S)    1   RECEIVE MODE
0484        0004        LSFEM  EQU    :0004     WORD 1  BIT(S)    2   ECHO MODE
0485        0008        LSFLB  EQU    :0008     WORD 1  BIT(S)    3   LOOP BACK MODE
0486        0010        LSFPDX EQU    :0010     WORD 1  BIT(S)    4   PROTOCOL DUPLEX
0487        0020        LSFLDX EQU    :0020     WORD 1  BIT(S)    5   LINE DUPLEX
0488        0040        LSFMTY EQU    :0040     WORD 1  BIT(S)    6   MODEM/CONTROLLER TYPE
0489        0080        LSFANM EQU    :0080     WORD 1  BIT(S)    7   ANSWER MODE
0490        0100        LSFCTY EQU    :0100     WORD 1  BIT(S)    8   CARRIER TYPE
0491        0200        LSFNCD EQU    :0200     WORD 1  BIT(S)    9   -CD REQUIRED BEFORE XMIT
0492        3000        LSFCOD EQU    :3000     WORD 1  BIT(S) 12-13 LINE CODE SET
0493        4000        LSFENL EQU    :4000     WORD 1  BIT(S)   14   LINE ENABLED
0494        8000        LSFINT EQU    :8000     WORD 1  BIT(S)   15   LINE INITIALIZED
0495        0002        LDADR  EQU    2         WORD 2  BIT(S)  0-15 DEVICE ADDRESS
0496        0003        LDFLG  EQU    3         WORD 3  BIT(S)       DEVICE DRIVER FLAGS
0497        0004        LDRTBL EQU    4         WORD 4  BIT(S)  0-15 DEVICE TRANFER VECTOR
0498        0005        LICIB  EQU    5         WORD 5  BIT(S)  0-15 INPUT CIB ADDR
0499        0006        LOCIB  EQU    6         WORD 6  BIT(S)  0-15 OUTPUT CIB ADDR
0500        0007        LOBIR  EQU    7         WORD 7  BIT(S)  0-15 OUTPUT FOR INT ROUTINE ADDR
0501        0008        LIINT  EQU    8         WORD 8  BIT(S)  0-15 INPUT WORD INTRPT LOC
0502        0009        LOINT  EQU    9         WORD 9  BIT(S)  0-15 OUTPUT WORD INT LOC
0503        000A        LISIN  EQU    10        WORD 10 BIT(S)  0-15 START INPUT INSTRUCTION
0504        000B        LIBFS  EQU    11        WORD 11 BIT(S)  0-15 INPUT INT BUF START A
0505        000C        LIBFE  EQU    12        WORD 12 BIT(S)  0-15 INPUT INT BUF END ADDR
0506        000D        LOBFS  EQU    13        WORD 13 BIT(S)  0-15 OUTPUT DEVICE BUFFER ADDR
0507        000E        LOBUF  EQU    14        WORD 14 BIT(S)  0-15 OUTPUT INT BUFFER ADDR
0508        000F        LOBCT  EQU    15        WORD 15 BIT(S)  0-15 OUTPUT INT BYTE COUNT
```

```
PAGE  0018  11/03/77  10:04:27   94500-10 RTX, IOX, COMX EQUATES
MACRO2 (A2) SI= MACROS  BO=          COMX DEVICE DRIVER EQUATES

0509        0010        LFCIB   EQU     16         WORD 16BIT(S)   0-15 FUNCTION PROC CURR CIB
0510        0011        LFWCIB  EQU     17         WORD 17 BIT(S)  0-15 FUNCTION PROC NEXT CIB
0511        0012        LLFUN   EQU     18         WORD 18 BIT(S)  0-15 LAST FUNCTION ISSUED
0512        0013        LHDWS   EQU     19         WORD 19 BIT(S)       LAST HARDWARE STATUS WORD
0513        0014        LIPARM  EQU     20         WORD 20 BITS  0-15 CURRENT LINE PARAMETER WORD
0514        0015        LLSDLY  EQU     21         WORD 21 BIT(S)  0-15 LINE (MODEM) SIGNAL DELAY
0515        0016        LDDWD1  EQU     22         WORD 22 BIT(S)       DEVICE DEPENDENT WORD 1
0516        8000        LDWTMD  EQU     :8000      WORD 22 BIT(S)  15   TRANSMIT MODIFICATION
0517        4000        LDWTXP  EQU     :4000      WORD 22 BIT(S)  14   TRANSMIT TRANSPARENT
0518        2000        LDWRXP  EQU     :2000      WORD 22 BIT(S)  13   RECEIVE TRANSPARENT
0519        0001        LDWDLE  EQU     :0001      WORD 22 BIT(S)   0   DLE SEEN
0520        0017        LDDWD2  EQU     23         WORD 23 BIT(S)       DEVICE DEPENDENT WORD 2
0521        0018        LDDWD3  EQU     24         WORD 24 BIT(S)       DEVICE DEPENDENT WORD 3
0522        0019        LPAD    EQU     25         WORD 25 BIT(S)  0-15 PAD CHARACTER INFORMATION
0523        00FF        LPDCNT  EQU     :00FF      WORD 25 BIT(S)  0-7  PAD CHARACTER COUNT
0524        FF00        LPDCHR  EQU     :FF00      WORD 25 BIT(S)  8-15  PAD CHARACTER
0525        001A        LSPCH   EQU     26         WORD 26 BIT(S)  0-7  SMC SYNC/SPEC CHAR
0526        001B        LEXCPT  EQU     27         WORD 27 BIT(S)  0-15 EXCEP INT POINT OF INT
0527        0021        LEXTSK  EQU     LEXCPT+6   WORD    BIT(S)  0-15 EXCEPTION INT TASK ADDR
0528        0023        LEXARG  EQU     LEXCPT+8   WORD    BIT(S)  0-15 EXCEP INT A REG - STATUS
0529        0028        LIMSK1  EQU     LEXCPT+13 WORD     BIT(S)  0-15 EXCEP INPUT MASK 1
0530        0029        LIMSK2  EQU     LEXCPT+14 WORD     BIT(S)  0-15 EXCEP INPUT MASK 2
0531        002A        LOMSK1  EQU     LEXCPT+15 WORD     BIT(S)  0-15 EXCEP OUTPUT MASK1
0532        002B        LOMSK2  EQU     LEXCPT+16 WORD     BIT(S)  0-15 EXCEP OUTPUT MASK2
0533        002C        LGMSK1  EQU     LEXCPT+17 WORD     BIT(S)  0-15 EXCEP GENERAL MASK 1
0534        002D        LGMSK2  EQU     LEXCPT+18 WORD     BIT(S)  0-15 EXCEP GENERAL MASK 2
0535        002E        LGEXCP  EQU     LEXCPT+19 WORD     BIT(S)  0-15 EXCEP GENERAL MASK PROC
0536        002F        LGEPRI  EQU     LEXCPT+20 WORD     BIT(S)  0-15 EXCEP GENERAL MASK PRIORITY
0537        0030        LSMSK1  EQU     LEXCPT+21 WORD     BIT(S)  0-15 EXCEP GENERAL MASK 1
0538        0031        LSMSK2  EQU     LEXCPT+22 WORD     BIT(S)  0-15 EXCEP GENERAL MASK 2
0539        0032        LSEXCP  EQU     LEXCPT+23 WORD     BIT(S)  0-15 EXCEP GENERAL MASK PROC
0540        0033        LEXRTN  EQU     LEXCPT+24 WORD     BIT(S)  0-15 EXCEPTION INT ROUTINE ADDR
```

```
PAGE  0019  11/03/77  10:41:27   94500-10 RTX, IOX, COMX EQUATES
MACRO2 (A2) SI= MACROS  ==              COMX PROTOCOL SERVICE MACROS

0542                          MACRO  GENXIO
0543                          IFT    #1+1     SKIP IF #1 = -1
0544                          LAP    #1       TABLE ORDINAL
0545                          ENDC
0546                          EXTR   CP:XIO
0547                          JST    CP:XIO   EXECUTE I/O CALL
0548                          ENDM


0550                   *
0551                   *  OPCODE DEFINITIONS
0552                   *
0553         0000      OPREAD EQU    IOPOPR   READ OPCODE
0554         0004      OPWRIT EQU    IOPOPW   WRITE OPCODE
0555         0008      OPFUNC EQU    IOPOPF   FUNCTION OPCODE
0556         0001      OPDELT EQU    IOPMR1   DELETE CHAR(S)
0557         0040      OPTEXT EQU    IOPTXT   TEXT MODE
0558         0800      OPAC-* EQU    IOPCKS   ACCUMULATE CHECKSUM
0559         0002      OPCC-* EQU    IOPMR2   CHECK CHECKSUM
0560         0400      OPXLAT EQU    IOPXLT   TRANSLATE
0561         0100      OPINIT EQU    IOPTRT   INITIAL
0562         1000      OPHDR  EQU    IOPHDR   HEADER REQUESTED
0563         2000      OPCH*  EQU    IOPSCK   CHECK INCOMING DATA
0564         8000      OPWCS  EQU    :8000    WRITE CONTROL SEQUENCE - FLAG ONLY (I CALL)
0565         0100      OPINTS EQU    IOPTMI   INTERMEDIATE STOP
0566         0300      OPENDS EQU    IOPTWE   ENDING STOP
0567                   *
0568                   *     FUNCTION DEFINITIONS
0569                   *
0570         0001      LNP    EQU    1        SET LINE PARAMETERS
0571         0002      SPC    EQU    2        SET SPECIAL CHARACTER
0572         0003      SYN    EQU    3        SET SYNCH CHARACTER
0573         0004      INT    EQU    4        INITIALIZE LINE
0574         0005      ENL    EQU    5        ENABLE LINE
0575         0006      DSL    EQU    6        DISABLE LINE
0576         0007      RST    EQU    7        READ STATUS
```

```
PAGE  0020  11/03/77  10:04:27   94500-10 RTX, IOX, COMX EQUATES
MACRO2 (A2) SI= MACROS  BO=            COMX PROTOCOL SERVICE MACROS

0577          0008        RON     EQU     8        RAISE +/OR WAIT FOR DATA SET SIGNALS ON
0578          0009        LOF     EQU     9        LOWER +/OR WAIT FOR DATA SET SIGNALS OFF
0579          000A        RVC     EQU     10       REVERSE CHANNEL
0580          000B        BRK     EQU     11       ISSUE BREAK
0581          000C        NSY     EQU     12       ISSUE NEW SYNC
0582          000D        RSY     EQU     13       RE-ESTABLISH SYNCHRONIZATION
0583          000E        SDW     EQU     14       SET DEVICE DEPENDENT WORD 1 BIT(S)
0584          000F        RDW     EQU     15       RESET DEVICE DEPENDENT WORD 1 BIT(S)



0586                              MACRO   CPREAD
0587                      OP:     SET     OPREAD   SET READ OPCODE
0588                      K:      SET     1
0589                              IFT     #?>1
0590                              REPT    #?-1     DO ALL OPTIONS
0591                              OPSET   #2,#3,#4,#5,#6,#7,#8,#9,#10
0592                              ENDC
0593                              GENXIO  #1       SET PARAM AND CALL XIO
0594                              IFT     OP:&OPTEXT
0595                              DATA    OP:      ACCUMULATED OPCODE
0596                              DATA    0,0,0    USE REQUESTORS BUFFER IF TEXT
0597                              ENDC
0598                              IFF     OP:&OPTEXT
0599                              DATA    OP:;IOPSTR ACCUMULATED OPCODE + SKIP BIT
0600                              DATA    0,-1,2   USE NO BUFFER OTHERWISE
0601                              ENDC
0602                              ENDM

0604                              MACRO   CPWRIT
0605                      OP:     SET     OPWRIT   SET WRITE OPCODE
0606                      K:      SET     1
0607                              IFT     #?>1
0608                              REPT    #?-1     DO ALL OPTIONS
```

```
0609                               OPSET  #2,#3,#4,#5,#6,#7,#8,#9,#10
0610                               ENDC
0611                               GENXIO #1         SET UP PARAM AND CALL XIO
0612                               DATA   OP:K\OPWCS  ACCUMULATED OPCODE
0613                               IFT    OP:KOPWCS
0614                               DATA   1          TABLE 1 IF WCS
0615                               ENDC
0616                               IFF    OP:KOPWCS
0617                               DATA   0,0,0      USERS BUFFER IF NOT WCS
0618                               ENDC
0619                               ENDM


0621                               MACRO  OPFUNC
0622                               GENXIO -1         CALL XIO - NO PARAM
0623                               IFF    #?=2
0624                               DATA   OPFUNC,0,%,#1     SELECTED FUNCTION
0625                               ENDC
0626                               IFT    #?=2
0627                               DATA   OPFUNC,0,#2,#1
0628                               ENDC
0629                               ENDM


0631                               MACRO  OPSET
0632                        OP:    SET    OP:!OP#K: ACCUMULATE OPTION INTO OPCODE
0633                        K:     SET    K:+1       BUMP CTR
0634                               ENDM
```

```
PAGE  0022  11/03/77  10:04:27   94500-10 PTY, IOX, COMX EQUATES
MACRO2 (A2) SI= MACROS  RO=            NAMDFF MACRO

                 MACRO   NAMDFF
        K:       SET     1
                 REPT    #?
                 ND      #1,#2,#3,#4,#5,#6,#7,#8,#9,#10
        PROT     SET     0          PROTOCOL DRIVER SPECIFICATION
        ECHO     SET     0          TTY ECHO OPTION
        EVEN     SET     0          TTY EVEN PARITY OPTION
        ODD      SET     0          TTY ODD PARITY OPTION
        DISCON   SET     0          DISCONNECT ON FILE MARK OPTION
        ASCII    SET     0          BSC-ASCII CHARACTER SET
        EBCDIC   SET     0          BSC-EBCDIC CHARACTER SET
        XLATE    SET     0          BSC-EBCDIC-ASCII TRANSLATION OPTION
        XPRNT    SET     0          BSC-FULL TRANSPARENT OPTION
        UXPRNT   SET     0          BSC-UNFORMATTED WRITE=TRANSPARENT OPTION
        ULC      SET     0          BSC-USER LINE CONTROL OPTION
        SLAVE    SET     0          BSC-SLAVE STATION OPTION
        WAITCO   SET     0          WAIT FOR CARRIER OFF OPTION
        CONSTC   SET     0          CONSTANT CARRIER OPTION
        SMC1     EQU     1          SYNCHRONOUS MODEM CONTROLLER - 513
        SMC2     EQU     2          SYNCHRONOUS MODEM CONTROLLER - 617
        DMODEM   EQU     3          DIO MODEM CABLE
        DTTY     EQU     4          DIO TTY CABLE
        DCRT     EQU     5          DIO CRT CABLE
        TTY      EQU     1          TTY PROTOCOL DRIVER
        BSC      EQU     2          BSC PROTOCOL DRIVER
        DMAD     EQU     :40        DMA IOD FLAG
        HDR:CT   SET     0
                 ENDM
        *
        *    NAM DECLARE MACRO - ND
        *
                 MACRO   ND
                 NAM     D:#K:
        K:       SET     K:+1
                 ENDM
```

```
              MACRO   CLINE
       *   GET PARAMS
              IFT     #?<3
              NOTE    F,'CLINE MACRO REQUIRES AT LEAST 3 PARAMETERS'
              ENDC
       DLUN:  SET     #2          DEVICE LUN
       *   GET PROPER DEVICE ADDRESS, INT ADDRESS AND LINE SPEED
              IFT     #4+0
       DEVAD: SET     #4          USE SUPPLIED DEVICE ADDRESS
              ENDC
              IFT     #5+0
       IADD:  SET     #5          USE SUPPLIED INTERRUPT ADDRESS
              ENDC
              IFT     #6+0
       LINE:  SET     #6
              ENDC
              IFT     #7+0
       SPEED: SET     #7          USE SUPPLIED BAUD RATE
              ENDC
       *   SET UP LINE IDENTIFIER PARAMETERS
              PLID    .           PROCESS LINE IDENTIFIER
       *    CHECK FOR LEGALITY
              IFF     DEVAD:
              NOTE    F,'DEVICE ADDRESS 0 ILLEGAL'
              ENDC
              IFF     IADD:
              NOTE    F,'INTERRUPT ADDRESS 0 ILLEGAL'
              ENDC
              IFF     SPEED:
              NOTE    F,'LINE  SPEED 0 ILLEGAL'
              ENDC
       *   NOW SET UP LIB SOFTWARE STATUS
       P:HDX  SET     P:FDX-1K1
              POSBIT  P:LCS,LSFCOD,T1:   SET LINE CODE
              POSBIT  WAITCO,LSFNCD,T2:  WAIT FOR CARRIER DETECT FLAG
       CC:    SET     CONSTC-1K1
```

```
       POSBIT CC:,LSFCTY,T3:     CONTROLLED CARRIER
       POSBIT SWITCH,LSFANM,T4:  SWITCHED LINE
       POSBIT SYNC:,LSFMTY,T5:   SYNC MODEM TYPE
       POSBIT HDX:,LSFLDX,T6:    HALF DUPLEX LINE
       POSBIT P:HDX,LSFPDX,T7:   HALF DUPEX PROTOCOL
SOFS:  SET    T1::;T2::;T3::;T4::;T5::;T6::;T7:
*
*   NOW GENERATE PROTOCOL TABLES
*
*    CIB
*
PICIB  SET    $
       DATA   0        CBOR
       RES    9,0      IOB
       RES    10,0     ENTRY ADDRESSES, ETC.
       DATA   P:FLGS   FLAGS WORD
       DATA   0        MASTER TABLE ADDRESS
       RES    P:LCIB-CPTBL-1,0
*   END OF CIB
*
*   PROTOCOL DIB
*
PIDIB  SET    $
D:#1   EQU    $
       CHAN   X::      DIB CHAIN
       DATA   PICIB,0,0 DCIB, QUEUE, CN
       DATA   P:DSW    DSW
       DATA   P:DVTP   DT
       DATA   '00'     DCUN
       DATA   0        DDEL
       DATA   0,0      DFUN, DULS
       DATA   P:DFLG,P:BLKS,P:LPRM      DPFLGS,DBLKSZ,DLPARM
       DATA   0,0      DIXLTR,DOXLTR
       CLUN   DLUN:    DLUN
       CHAN   Q:#1     DDIB
       RES    3,0      DSRTN, DTMP1, DTMP2
```

```
                    DATA    P:CKSF   DCKSF
                    IFT     P:CKFG
                    DATA    P:CKFG,P:CKSP,0      DCKSF,DCKSP,DCKSA
                    ENDC
                    IFF     P:CKFG
                    DATA    0,0,0
                    ENDC
                    DATA    HDR:CT
                    IFT     HDR:CT>0
                    FORM    BYTES,8,8
                    BYTES   HDR:C1,HDR:C2
                    IFT     HDR:CT>2
                    BYTES   HDR:C3,HDR:C4
                    IFT     HDR:CT>4
                    BYTES   HDR:C5,HDR:C6
                    IFT     HDR:CT>6
                    BYTES   HDR:C7,HDR:C8
                    IFT     HDR:CT>8
                    BYTES   HDR:C9,CA
                    ENDC    CT>0
                    ENDC    CT>2
                    ENDC    CT>4
                    ENDC    CT>6
                    ENDC    CT>8
            *  END OF PROTOCOL TABLES
            *
            *       DEVICE TABLES
            *
            *  LINE BUFFER
            *
            TRUE    SET     1
            FALSE   SET     0
            T1:     SET     SPEED:<1800
            T2:     SET     PROT=TTY
            T3:     SET     P:OCM-1&1
                    IFT     T1:&T2:&T3:
```

```
              RES    2,0         2 WORD BUFFER IF SLOW TTY
              ENDC
              IFF    T1:&T2:&T3:
              RES    P:MAXT/2,0  FULL BUFFER
              ENDC
       EBUFF  SET    $
              DATA   0           BUFFER OVERFLOW WORD
              IFT    P:FDX&FDX:&P:OCM
       OBUF   SET    $
              RES    P:MAXT/2,0  OUTPUT BUFFER FOR FULL DUX
              ENDC
       *
       *
       *  OUTPUT CIB
       *
       OBIRA  SET    $
              XD     SCH:        CHOR
              EXTR   INTQS:
              SPAD   INTQS:
              JST    INTQS:
              DATA   $,0,0,0
              DATA   0
              DATA   CINPRI
              DATA   0,OBIRA,OBIRA
              LDX    $-2
              JMP    *OBIRA+CINRTN   JUMP TO INTERRUPT ROUTINE
              DATA   -1          CTIMER
              DATA   L:#3        CLIB
              RES    5,0         DRIVER ENTRIES
              DATA   0           CINRTN
       DOCIB  SET    $           OUTPUT CIB ADDR = INPUT CIB
              IFT    P:FDX&FDX:
       DOCIB  SET    OBIRA       REAL OUTPUT CIB
              RES    L:DOCB-CDSRTN,0
              ENDC
       *  END OF OUPUT CIB
```

```
                *
                *   INPUT CIB
                *
                DICIB   SET     $
                        XD      SCH:       CPUR
                        EXTR    INTQS:
                        SPAD    INTQS:
                        JST     INTQS:
                        DATA    $,0,0,0
                        DATA    0
                        DATA    CINPRI
                        DATA    0,DICIB,DICIB
                        LDX     $-2
                        JMP     *DICIB+CINRTN     JUMP TO INTERRUPT ROUTINE
                        DATA    -1         CTIMER
                        DATA    L:#3       CLIP
                        RES     5,0        DRIVER ENTRIES
                        DATA    0,0        CINRTN, CDSRTN
                        RES     CCHAIN-CDSRTN-1,0
                        CHAN    C:CIB      CIB CHAIN - CCHAIN
                        DATA    IBUF       CCCADR
                        RES     L:DICB-CCCHAR-1,0
                *   END OF INPUT CIB
                *
                        IFT     P:FDXNFDX:
                *   OUTPUT DIB
                *
                DODIB   SET     $
                        CHAN    X::        DIB CHAIN
                        DATA    DOCIB      DCIB
                        DATA    0,0        QUEUE, CA
                        DATA    D:DDSW     DSW
                        DATA    D:DVTP     DT
                        DATA    '00'       DCUN
                        DATA    0,4,0      DDFL, DFUN, DULS
                *   END OF OUPUT DIB
```

```
PAGE  0028  11/03/77  10:04:27   94500-10 RTX, IOX, COMX EQUATES
MACRO2 (A2) SI= MACROS  BO=             CLINE MACRO

                       *
                             ENDC
                       *  INPUT (MAIN) DIB
                       *
                       DIDIB SET    $
                       D:#3  SET    $
                             CHAN   X::      DIB CHAIN
                             DATA   DICIB    DCIB
                             DATA   0,0      QUEUE, CN
                             IFF    P:FDX&FDX:
                       D:IDSW SET   D:IDSW;D:ODSW   OR INPUT & OUTPUT DSW FOR HDX
                             ENDC
                             DATA   D:IDSW   DSW
                             DATA   D:DVTP   DT
                             DATA   '00'     DCUN
                             DATA   0,4,0    DDEL, DFUN, DULS
                       * END OF DIB
                       *
                       *  DEVICE DRIVER LIB
                       *
                       DLIB  SET    $
                       L:#3  SET    $
                             DATA   0        LSTNS
                             DATA   SOFS:    LSOFS
                             DATA   DEVAD:   LDADR
                             DATA   D:DFLG   LDFLG
                             DATA   0        LDRTBL
                             DATA   DICIB    LICIB
                             DATA   DOCIB    LOCIB
                             DATA   OBIRA    LOBIR
                             DATA   0        LIINT
                             DATA   0        LOINT
                             DATA   0        LISIN
                             BAC    IBUF     LIBFS
                             BAC    EBUFF    LIBFF
                             BAC    OBUF     LOBFS
```

```
PAGE  0029  11/03/77  10:04:27   94500-10 RTX, IOX, COMX EQUATES
MACRO2 (A2) SI= MACROS  BO=          CLINE MACRO

0888                         BAC    0,0      LOBUF, LOBCT
0889                         DATA   0,0      LFCIB, LFWCTB
0890                         DATA   0        LLFUN
0891                         DATA   0        LHDWS
0892                         DATA   P:LPRM   LLPARM
0893                         DATA   D:SDLY   LLSDLY
0894                         RES    3,0      LODW01,2,3
0895                         DATA   P:PADS   LPADS
0896                         DATA   0        LSPCH
0897                         IFT    D:EXIP
0898                         DATA   0        LEXCPT
0899                         EXTR   INTQS:
0900                         SPAD   INTQS:
0901                         JST    INTQS:
0902                         DATA   $,0,0,0
0903                         DATA   0        TASK TO PROCESS INT
0904                         DATA   CINPRI   PRIORITY
0905                         DATA   0        LEXARG
0906                         DATA   L:#3     X-REG
0907                         DATA   $-10     P-REG STORAGE ADDR
0908                         LDX    $-2      LIB ADDR TO X
0909                         JMP    *DLIB+LEXRTN     JUMP TO PROCESSOR
0910                         RES    12,0     IN,OUT,GENERAL MASKS AND PROCESSORS
0911                         ENDC
0912                  *
0913                  *  END OF LIB
0914                  *
0915                  *
0916                  *   END OF DEVICE TABLES
0917                         LPOOL
0918                  *
0919                  * ESCAPE MACROS
0920                  *
0921                  BACK   SET    $
0922                         DEVFIX .        FIX UP DEVICE INFO
0923                         INTSET .        SET UP INTERRUPT VECTORS
```

```
PAGE  0030  11/03/77  10:04:27   94500-10 RTX, IOX, COMX EQUATES
MACRO2 (A2) SI= MACROS  BO=             CLINE MACRO

0924                          IFT    PROT=TTY
0925                          TTYFIX .           FIX UP TBLS FOR TTY
0926                          ENDC
0927                          IFT    PROT=BSC
0928                          BSCFIX .           FIX UP TBLS FOR BSC
0929                          ENDC
0930                          REL    BACK
0931                          ENDM
```

```
0933                      MACRO  CLUN
0934                      DATA   #1+:4000 COMX STANDARD LUN GENERATION
0935                      ENDM
```

```
                MACRO  DEVICE
        DEV:    SET    0          INITIAL DEVICE ID
        LINE:   SET    0          INITIAL LINE ID
        SPEED:  SET    0          INITIAL SPEED (BAUD RATE)
        DEVAD:  SET    0          INITIAL DEVICE ADDRESS
        IADD:   SET    0          INITIAL INTERRUPT ADDRESS
        *
        *  PULL PARAMETERS OFF CALL
        *
        DEV:    SET    #1+0
                IFT    #?>=2
        LINE:   SET    #2
                ENDC
                IFT    #?>=3
        SPEED:  SET    #3
                ENDC
                IFT    #?>=4
        DEVAD:  SET    #4+0
        IADD:   SET    #5+0
                ENDC
                IFT    #?>5
        K:      SET    1
                REPT   #?-5
                SETONE #6,#7,#8,#9,#10
                ENDC
        *
        *   NOW START DEVICE ORIENTED GENERATION
        *     SYNC MODEM CONTROLLER
        T1:     SET    DEV:=617
        T2:     SET    DEV:=513
        SMC:    SET    T1:;T2:
                IFT    SMC:
        L:DICB  SET    LDICIB     STANDARD LENGTH CIB
        L:DOCB  SET    LDOCIB     STANDARD LENGTH CIB
        D:DFLG  SET    0          LDFLG WORD
        D:EXIP  SET    1          EXCEPT INT DEVICE
```

```
0973              D:SDLY SET    2         TICK COUNT FOR MODEM SIGNAL DELAY
0974              D:IDSW SET    :4007     DSW = FUNC 2, READ 0,1,2
0975              D:ODSW SET    :4019     DSW = FUNC 2, WRITE 0
0976              D:DVTP SET    'SC'
0977                     ENDC
0978                     ENDM


0980              *
0981              *      DEVICE FIXUP MACRO
0982              *
0983                     MACRO  DEVFIX   FIX UP DEVICE AFTER CLINE
0984                     IFT    SMC:
0985              *
0986              *  SET UP INTERRUPT ADDRESSES
0987              *   EXCEPTION INTERRUPT
0988              *
0989                     ORG    DLTB+LEXRIN
0990                     XD     SC:EXR
0991                     ORG    DLTB+LEXTSK
0992                     XD     SC:EXX
0993              *   INPUT INTERRUPTS
0994                     ORG    DICTB+CINTSK
0995                     XD     SC:IEX
0996                     ORG    DICTB+CINRIN
0997                     XD     SC:IEB
0998              *   OUTPUT INTERRUPTS
0999                     ORG    OBTRA+CINTSK
1000                     XD     CD:OEX
1001                     ORG    OBTRA+CINRIN
1002                     XD     SC:OER
1003              *  SET SYNC DETECTED EXCEPTION PROCESSOR ADDR
1004                     ORG    DLTB+LSEXCP
1005                     IFT    PROT=BSC
1006                     XD     SC:BSY
1007                     ENDC
```

```
*
*   SHIFT DEVICE TO PROPER BIT POSITTION
*
DEVAD: SET    DEVAD:%3 SHIFT LEFT 3 FOR SMC
       ENDC
* END OF SMC DEVICE DEPENDENT SETUP
*
*  DEVICE DEPENDENT XFER VECTOR ADDRESS
*
       ORG    DLIB+LDRTBL
       IFT    DEV:=513
       IFT    PROT=BSC
       IFT    ASCII
       EXTR   SC:BA5
       DATA   *SC:BA5
       ENDC
       IFT    EBCDIC
       EXTR   SC:BE5
       DATA   *SC:BE5
       ENDC
       ENDC
       IFF    PROT=BSC
       XD     SC:513
       ENDC
       ENDC
       IFT    DEV:=617
       IFT    PROT=BSC
       IFT    ASCII
       EXTR   SC:BA6
       DATA   *SC:BA6
       ENDC
       IFT    EBCDIC
       EXTR   SC:BE6
       DATA   *SC:BE6
       ENDC
       ENDC
```

```
                  IFF     PRNT=BSL
                  XD      SC:617
                  ENDC
                  ENDC
          *   SET SPECIAL/SYNC CHAR
                  ORG     DLIB+LSPCH
                  IFT     DEV:=513
                  DATA    :00FF%P:CLSC;P:SYNC   513 SYNC CHAR
                  ENDC
                  IFT     DEV:=617
                  DATA    P:SYNC    SYNC CHAR
                  ENDC
          *    START INPUT INSTRUCTION
                  ORG     DLIB+LISTN
                  IFT     SMC:
                  SEA     DEVAD:    LISTN
                  ENDC
          *    START INPUT INSTRUCTION
                  IFF     SMC:
                  OTA     DEVAD:    LISTN
                  ENDC
          *    LIB DEVICE ADDR
                  ORG     DLIB+LDADR
                  DATA    DEVAD:
          *   SET UP DRIVER ENTRY ADDRESSES
                  ORG     DICTB+CJTBL  DRIVER ENTRIES
                  XD      CD:RDE    READ COMMAND
                  XD      CD:WRE    WRITE COMMAND
                  DATA    0         POSITION COMMAND
                  XD      CD:FNE    FUNCTION COMMAND
                  ENDM
```

```
PAGE  0036  11/03/77  10:04:27   94500-10 RTX, IOX, COMX EQUATES
MACRO2 (A2) SI= MACROS  BO=           INTERRUPT VECTOR SETUP MACRO

                    MACRO  INTSET    SET UP INTERRUPT VECTORS
               *  SET UP INTERRUPT VECTOR
                    IFT    SMC:
                    ABS    IADD:
                    JST    *$+1
                    DATA   DLIB+LEXCPT   EXCEPTION INT
                    AIB    DEVAD:   INPUT BYTE
                    DATA   0,0,0
                    JST    *$+1     INPUT EOB
                    DATA   DICIB
                    AOB    DEVAD:   OUTPUT BYTE
                    DATA   0,0,0
                    JST    *$+1     OUTPUT EOB
                    DATA   OBIRA
                    REL    DLIB+LIINT
                    DATA   IADD:+2  INPUT BYTE INTERRUPT ADDR
                    ORG    DLIB+LOINT
                    DATA   IADD:+8  OUTPUT BYTE INTERRUPT ADDR
                    ENDC
                    ENDM
```

```
          MACRO  RLID
*
*    FIRST SET UP LN SYMBOLS TO TRUTH OF LINE=N
*    THESE SYMBOLS ARE USED LATER FOR CONVENIENCE
*
K:        SET     0
SUM:      SET     0
          REPT    9          9=NO OF POSSIBLE VALUES FOR LINE
          SETMUX 0,1,2,3,4,5,6,7,8     ALL POSSIBLE VALUES FOR LINE
*
*    DETERMINE ANSWER MODE
*
SWITCH SET    LINE:>=5
*
*    CHECK VALIDITY OF LINE
*
          IFF     SUM:=1
          NOTE    E,'LINE  SPECIFICATION INVALID'
          ENDC
*
*    DETERMINE LINE DUPLEX
*
HDX:      SET     L2;L4;L6;L8
FDX:      SET     L0;L1;L3;L5;L7
*
*    DETERMINE SYNC OR ASYNC
*
SYNC:     SET     L3;L4;L7;L8
ASYNC:    SET     L0;L1;L2;L5;L6
*
*    DETERMINE  MODEM OR CURRENT LOOP INTERFACE
*
MODEM:    SET     L0-1*1
          ENDM
```

```
*
*   POSBIT MACRO - CREATES A VALUE IN PROPER BIT POSITION FROM
*      A TRUE/FALSE FLAG AND A MASK EQUATE
*    CALLED AS:
*      POSBIT FLAG,MASK,OPTIONAL-NEW-SYMBOL
*        THE FLAG IS SHIFTED INTO THE BIT POSITION SPECIFIED BY THE
*        MASK AND THEN IS EQUATED TO EITHER THE FIRST PARAMETER
*        SYMBOL OR THE THIRD PARAMETER SYMBOL (IF SPECIFIED)
*
*    EXAMPLE
*      POSBIT 1,:0400,MYFLG
*        RESULTS IN MYFLG BEING SET (NOT EQUATED) TO :0400
*
*FLAG  SET    0
*      POSBIT FLAG,:8,LABEL
*        RESULTS INLABEL BEING SET TO 0
*
*FLAG  SET    1
*      POSBIT FLAG,:8000
*        RESULTS IN FLAG BEING SET TO :8000
*
*
       MACRO  POSBIT   POSITION FLAG BIT IN WORD
       SHFCNT #2       CALCULATE SHIFT COUNT
       IFT    #?=3     SET NEW RESULT IF 3 PARMS
#3     SET    #1%SHFCNT SHIFT VALUE TO PROPER POSITION
       ENDC
       IFT    #?=2
       IFF    SHFCNT=0 SKIP IF NO SHIFTING
#1     SET    #1%SHFCNT SHIFT VALUE TO PROPER POSITION
       ENDC
       ENDC
       ENDM


*
```

```
*       SYMBOL SETUP MULTIPLEXOR
*
        MACRO   SETMUX
T1:     SET     K:+1        ACCOUNT FOR PARAMETER ZERO
L#T1:   SET     LINE:=#T1:  SET EACH LX EITHER TRUE OR FALSE
SUM:    SET     SUM:+L#T1: SUM RESULTS
K:      SET     K:+1
        ENDM


        MACRO   SHFCNT    DETERMINE SHIFT COUNT TO GET MASK
SHFCNT  SET     0
VAL:    SET     #1          VALUE OF MASK
        REPT    16
        SHF1:   .           CALCULATE SHIFT COUNT
        ENDM


        MACRO   SHF1:
        IFF     VAL:X1=1 SKIP IF RIGHT JUSTIFIED
VAL:    SET     VAL:%-1  SHIFT
SHFCNT  SET     SHFCNT+1 AND COUNT
        ENDC
        ENDM


*
*       THE SETONE MACRO SETS THE PROVIDE PARAMETER
*          TO A ONE AND INCREMENTS K:
*       THIS IS USED FOR SCANNING OFF PARAMETERS
*
        MACRO   SETONE
#K:     SET     1
K:      SET     K:+1
        ENDM
*
```

```
1201                  *      THE XD MACRO DECLARES AN EXTERNAL DATA ITEM
1202                  *
1203                         MACRO   XD
1204                         EXTR    #1
1205                         DATA    #1
1206                         ENDM
1207                  *
1208                  *     HEADER MACRO
1209                  *
1210                         MACRO   HEADER
1211                  *
1212                  HDR:CT SET     #1+0      COUNT OF CHARS
1213                  HDR:C1 SET     #2+0      CHAR 1
1214                  HDR:C2 SET     #3+0      CHAR 2
1215                  HDR:C3 SET     #4+0      CHAR 3
1216                  HDR:C4 SET     #5+0      CHAR 4
1217                  HDR:C5 SET     #6+0      CHAR 5
1218                  HDR:C6 SET     #7+0      CHAR 6
1219                  HDR:C7 SET     #8+0      CHAR 7
1220                  HDR:C8 SET     #9+0      CHAR 8
1221                  HDR:C9 SET     #10+0     CHAR 9
1222                  HDR:CA SET     #11+0     CHAR 10
1223                  *
1224                         ENDM
```

```
1226                MACRO  BSC
1227         PROT   SET    BSC
1228         ASCII  SET    0
1229         EBCDIC SET    0
1230         XLATE  SET    0
1231         XPRNT  SET    0
1232         UXPRNT SET    0
1233         DLE    SET    0
1234         SLAVE  SET    0
1235         DISCON SET    0
1236         K:     SET    1
1237                REPT   #?-1
1238                SETONE #2,#3,#4,#5,#6,#7,#8,#9,#10
1239                IFT    ASCII&EBCDIC
1240                NOTE   E,'BOTH  ASCII AND EBCDIC SPECIFIED'
1241                ENDC
1242                IFT    XPRNT&UXPRNT
1243                NOTE   E,'BOTH  TRANSPARENT OPTIONS CANT BE GIVEN'
1244                ENDC
1245                IFF    ASCII;EBCDIC
1246                NOTE   E,'MUST  SPECIFY CHARACTER SET ASCII OR EBCDIC'
1247                ENDC
1248         P:MAXT SET    #1
1249         P:PADS SET    :FF06     BSC PAD CHARACTERS
1250         P:FLGS SET    0         CPFLGS FLAG WORD
1251         P:LCIB SET    LPCIB     LENGTH OF CIB
1252         P:NCIB SET    1         NO OF CIBS PER PROT LINE
1253         P:FDX  SET    0         HALF DUPLEX PROTOCOL
1254         P:DSW  SET    :50FF     DSW - ALLOW READ, WRITE & FUNC
1255         P:CKSR SET    1         CHECKSUM PROTOCOL FLAG
1256         P:DVTP SET    'BS'
1257         P:CLSC SET    8         BSC 8 BIT CHARS
1258         P:BLKS SET    0
1259                IFT    ASCII
1260         P:LPRM SET    :F        ASCII, 8-BIT, EVEN PARITY
1261         P:CKFG SET    :807F     CHECKSUM FLAGS
```

```
1262          P:OCM  SET    0          NO OUTPUT CHAR MODIFICATION
1263          P:SYNC SET    :16        ASCII SYNC CHAR
1264          P:LCS  SET    1          LINE CHAR SET USASCII
1265          T1:    SET    XPRNT=1
1266          T2:    SET    UXPRNT=1
1267                 IFT    T1::T2:    DO IF SOME FORM OF TRANSPARENT
1268          P:CKSP SET    :A001      CRC POLYNOMIAL - X**16+X**15+X**2+1
1269                 ENDC
1270                 IFF    T1::T2:
1271          P:CKSP SET    :0080      LRC POLYNOMIAL - X**8+1
1272                 ENDC
1273                 ENDC
1274                 IFT    EBCDIC
1275          P:LPRM SET    :00C       EBCDIC, 8-BIT, NO PARITY
1276          P:CKFG SET    :80FF      CHECKSUM FLAGS
1277          P:CKSP SET    :A001      CRC POLYNOMIAL - X**16+X**15+X**2+1
1278          P:SYNC SET    :32        EBCDIC SYNC CHARACTER
1279          P:OCM  SET    0          OUTPUT CHAR MODIFICATION
1280          P:LCS  SET    2          LINE CHAR SET EBCDIC
1281                 ENDC
1282          ULC    SET    ULC%10
1283          XPRNT  SET    XPRNT%9
1284          UXPRNT SET    UXPRNT%8
1285          MASTER SET    SLAVE-1&1
1286          P:DFLG SET    MASTER+UXPRNT+XPRNT+ULC
1287                 ENDM


1289          *
1290          *   BSC FIXUP MACRO
1291          *
1292                 MACRO  BSCFIX   FIX UP BSC AFTER CLINE
1293          *  SET UP DRIVER ENTRY ADDRESSES
1294                 ORG    PICIB+CJTBL
1295                 XD     BS:RRE
1296                 XD     BS:WRE
```

```
          DATA   0
          XD     BS:ERF
*   SET UP INPUT CHAR TIMER
          ORG    PICIB+ICTMR+1
          IFT    LSI305
          DATA   1200      10SECS
          ENDC
          IFT    LSI2
          DATA   1000      10 SECS
          ENDC
*   SET UP PROTOCOL MASTER TABLE ADDRESS
          ORG    PICIB+CPTBL
          IFT    ASCII
          IFT    DEV:=513;DEV:=617
          XD     BT:SCA
          ENDC
          IFF    DEV:=513;DEV:=617
          XD     BT:ASC
          ENDC
          ENDC
          IFT    EBCDIC
          XD     BT:EBC
          ENDC
*   SET UP ASCII TRANSLATE TABLES
          IFT    XLATE
          IFT    ASCII
          ORG    PIDIB+DIXLTB
          XD     CD:A2I
          XD     CD:I2A
          ENDC
*   SET UP EBCDIC TRANSLATE TABLES
          IFT    EBCDIC
          ORG    PIDIB+DIXLTB
          XD     CD:E2I
          XD     CD:I2E
          ENDC
```

```
1333                         ENDC
1334                  *  SET UP BCC GENERATOR
1335                         ORG    PIDIB+DCKSR
1336                         XD     CD:CKS
1337                  *  SET UP BLOCK SIZE PER LINE SPEED
1338                         ORG    PIDIB+DBLKS
1339                  T3:    SET    SPEED:/8
1340                         IFT    T3:<P:MAXT
1341                         DATA   T3:
1342                         ENDC
1343                         ENDM
```

```
PAGE  0045  11/03/77  10:04:27   94500-10 RTX, IOX, CONX EQUATES
MACRO2 (A2) SI= MACROS  BO=              R T X   M A C R O S

1346                             MACRO  RTX        INITIALIZE RTX
1347                             EXTR   RTX:       REAL TIME EXECUTIVE INITIALIZE
1348                             JST    RTX:
1349                             DATA   #1,#2
1350                             ENDM

1352                             MACRO  WKAREA     DEFINE WORK AREA HERE
1353                             RES    #1*5
1354                             ENDM
```

```
1356                    MACRO  BEGIN     BEGIN TASK
1357                    EXTR   BEGIN:    BEGIN TASK
1358                    JST    BEGIN:
1359                    PARAMS BEGIN,#?,2,2  CHECK NUMBER OF PARAMETERS
1360                    RTX:1  #1,#2     DATA TASKADR,PRIORITY
1361                    ENDM

1363                    MACRO  PAUSE     ALLOW ROUND ROBIN
1364                    EXTR   PAUSE:    ALLOW ROUND ROBIN
1365                    JST    PAUSE:    CALL RTX TO ALLOW ROUND ROBIN
1366                    ENDM

1368                    MACRO  TERM      TERMINATE TASK
1369                    EXTR   END:      TERMINATE TASK
1370                    JST    END:
1371                    ENDM

1373                    MACRO  ABORT     ABORT A TASK
1374                    EXTR   ABORT:    ABORT A TASK
1375                    JST    ABORT:
1376                    ENDM
```

```
PAGE  0047  11/03/77  10:04:27   94500-10 RTX, IOX, COMX EQUATES
MACRO2 (A2) SI= MACROS  BO=              P R I O R I T Y   C O N T R O L

1378                          MACRO   GETPR     GET CURRENT PRIORITY
1379                          EXTR    GETPR:    GET CURRENT PRIORITY
1380                          JST     GETPR:
1381                          ENDM

1383                          MACRO   SETPR     ALTER CURRENT PRIORITY
1384                          EXTR    SETPR:    ALTER CURRENT PRIORITY
1385                          JST     SETPR:
1386                          ENDM

1388                          MACRO   INCPR     INCREMENT CURRENT PRIORITY
1389                          EXTR    INCPR:    INCREMENT CURRENT PRIORITY
1390                          JST     INCPR:
1391                          ENDM

1393                          MACRO   DECPR     DECREMENT CURRENT PRIORITY
1394                          EXTR    DECPR:    DECREMENT CURRENT PRIORITY
1395                          JST     DECPR:
1396                          ENDM
```

```
         MACRO  GET       GET REGISTERS
         EXTR   GET:      GET REGISTERS
         JST    GET:
         PARAMS GET,#?,1,1
         DATA   #1
         ENDM

         MACRO  PUT       PUT REGISTERS
         EXTR   PUT:      PUT REGISTERS
         JST    PUT:
         PARAMS PUT,#?,1,1
         DATA   #1
         ENDM
```

```
1412                          MACRO  INTSV     SAVE ENVIRONMENT (INTERRUPT)
1413                          EXTR   INTSV:    SAVE ENVIRONMENT (INTERRUPT)
1414                          JST    INTSV:
1415                          PARAMS INTSV,#?,1,1
1416                          DATA   *#1       DATA *RETURN ADDRESS
1417                          ENDM

1419                          MACRO  INTRS     RESTORE ENVIRONMENT (INTERRUPT)
1420                          EXTR   INTRS:    RESTORE ENVIRONMENT (INTERRUPT)
1421                          JST    INTRS:
1422                          ENDM

1424                          MACRO  INTAC     ACTIVATE TASK (INTERRUPT)
1425                          EXTR   INTAC:    ACTIVATE TASK (INTERRUPT)
1426                          JST    INTAC:
1427                          PARAMS INTAC,#?,2,2
1428                          RTX:1  #1,#2     DATA TASKADR,PRIORITY
1429                          ENDM

1431                          MACRO  INTQ      QUEUE TASK (INTERRUPT)
1432                          EXTR   INTQ:     QUEUE TASK (INTERRUPT)
1433                          JST    INTQ:
1434                          PARAMS INTQ,#?,3,5
1435                          DATA   3,0,0,0   TEMP CELLS & RETURN ADDRESS
1436                          RTX:1  #2,#3     DATA TASKADR,PRIORITY
1437                          IFT    #?=3
1438                          DATA   0,0       NO  A  OR  X  REGISTERS
1439                          ENDC
1440                          IFT    #?=4
1441                          DATA   #4,0      A  REGISTER ONLY
1442                          ENDC
1443                          IFT    #?=5
1444                          DATA   #4,#5     A & X  REGISTERS
1445                          ENDC
1446                          DATA   #1        RETURN ADDRESS
1447                          ENDM
```

11/3-62

```
PAGE  0050  11/03/77  10:04:27   94500-10 RTX, IOX, CORX EQUATES
MACRO2 (A2) SI= MACROS  BO=                IOB GENERATOR MACRO

1449                         MACRO  IOB      LUN,BUFFER,LENGTH,OPERATION,MODIF,DA,COORD
1450                   *
1451                   *     GENERATE IOB
1452                   *
1453                         RES    3,0
1454                         DATA   #7+0     COORDINATION NO, IF ANY
1455                         DATA   #1       LUN
1456                   IOBM:1 SET   #5+0     MODIFIER, IF ANY
1457                         DATA   #4%2+IOBM:1 OPCODE & MODIFIER
1458                         DATA   #3+0     LENGTH
1459                         DATA   #2+0     BUFFER
1460                         DATA   0        ACTUAL COUNT
1461                         DATA   #6+0     DIRECT ACCESS ADDRESS, IF ANY
1462                         ENDM
```

```
1464                         MACRO   ENT       DEFINE SUBROUTINE
1465                  RTX:T  SET     $         DEFINE LOCATION OF RETURN ADDRESS
1466                         STOP    :3C       :3C3C IS FLAG FOR NOT CALLED YET
1467                  SUBX:: SET     0         DEFINE RTN AS EXIT TYPE
1468                         ENDM

1470                         MACRO   RETURN    RETURN FROM LAST DEFINED SUBROUTINE
1471                         IFT     SUBX::
1472                         JMP     RTX:T-1   IF SUBX IS EXIT TYPE
1473                         ENDC
1474                         IFF     SUBX::
1475                         RTN     RTX:T     IF RTN IS EXIT TYPE
1476                         ENDC
1477                         ENDM

1479                         MACRO   ARG
1480                         LD#1    *RTX:T
1481                         IMS     RTX:T
1482                         ENDM

1484                         MACRO   COMSUB    DEFINE COMMON SUBROUTINE
1485                         SPACE   1
1486                         EXTR    SUBX:     EXIT FROM COMMON SUBROUTINE
1487                         JST     SUBX:     CALL SUBX TO RETURN
1488                  #1     ENT               DEFINE ENTRY & RETURN ADDRESS LOCATION
1489                  SUBX:: SET     1         DEFINE SUBX AS EXIT TYPE
1490                         SPACE   1
1491                         ENDM

1493                         MACRO   PROSUB    DEFINE PROTECTED SUBROUTINE
1494                         SPACE   1
1495                         EXTR    SUBX:     EXIT FROM COMMON SUBROUTINE
1496                         EXTR    PROT:
1497                         SPAD    PROT:
1498                         JST     SUBX:     CALL SUBX TO RETURN
1499                         ENT               DEFINE RETURN ADDRESS LOCATION
```

```
1500                    #1      HLT                DEFINE ENTRY POINT
1501                    SUBX::  SET     1          DEFINE SUBX AS EXIT TYPE
1502                            JST     PROT:      CALL PROT
1503                            DATA    #1-1       USE RETURN LOC AS COORDINATION #
1504                            SPACE   1
1505                            ENDM

1507                            MACRO   LOCSUB     DEFINE LOCKED SUBROUTINE
1508                            EXTR    LOCK:
1509                            SPAD    LOCK:
1510                            SPACE   1
1511                    #1      HLT                DEFINE ENTRY POINT
1512                            JST     LOCK:      CALL LOCK
1513                            DATA    #2         COORDINATION NUMBER
1514                            ENT                DEFINE RETURN ADDRESS LOCATION
1515                            SPACE   1
1516                            ENDM

1518                            MACRO   CALCOM     CALL COMMON SUBROUTINE
1519                            EXTR    SUBR:      CALL COMMON SUBROUTINE
1520                            JST     SUBR:      CALL SUBR
1521                    CNT:    SET     1
1522                            REPT    #?
1523                            RTX:2   #1,#2,#3,#4,#5,#6,#7,#8,#9,#10,#11,#12,#13,#14,#15,#16
1524                            ENDM

1526                            MACRO   CALPRO     CALL PROTECTED SUBROUTINE
1527                            JST     #1
1528                    CNT:    SET     1
1529                            REPT    #?-1
1530                            RTX:2   #2,#3,#4,#5,#6,#7,#8,#9,#10,#11,#12,#13,#14,#15,#16
1531                            ENDM

1533                            MACRO   CALOCK     CALL LOCKED SUBROUTINE
1534                            JST     #1
1535                    CNT:    SET     1
```

```
PAGE  0053  11/03/77  10:04:27   94500-10 RTX, IOX, COMX EQUATES
MACRO2 (A2) ST= MACROS  RU=            S U B R O U T I N E S   ETC.

1536                    REPT    #?-1
1537                    RTX:2   #2,#3,#4,#5,#6,#7,#8,#9,#10,#11,#12,#13,#14,#15,#16
1538                    ENDM

1540                    MACRO   SUBX      EXIT FROM COMMON OR PROTECTED SUBROUTINE
1541                    EXTR    SUBX:     EXIT FROM COMMON SUBROUTINE
1542                    JST     SUBX:
1543                    ENDM

1545                    MACRO   LOCK      LOCK A FACILITY
1546                    EXTR    LOCK:
1547                    SPAD    LOCK:
1548                    JST     LOCK:     CALL LOCK
1549                    PARAMS LOCK,#?,1,1
1550                    DATA    #1,0      COORDINATION # & ROOM FOR RETURN
1551                    ENDM

1553                    MACRO   UNLOCK    UNLOCK A FACILITY
1554                    EXTR    UNLK:     UNLOCK A FACILITY
1555                    JST     UNLK:
1556                    PARAMS UNLOCK,#?,1,1
1557                    DATA    #1
1558                    ENDM

1560                    MACRO   UNPROT    UN-PROTECT A FACILITY
1561                    EXTR    UNPR:     UN-PROTECT A FACILITY
1562                    JST     UNPR:
1563                    PARAMS UNPROT,#?,1,1
1564                    DATA    #1
1565                    ENDM

1567                    MACRO   PROT      PROTECT A FACILITY
1568                    EXTR    PROT:     PROTECT A FACILITY
1569                    SPAD    PROT:
1570                    JST     PROT:
1571                    PARAMS PROT,#?,1,1
```

```
PAGE  0054  11/03/77  10:04:27   94500-10 RTX, IOX, COMX EQUATES
MACRO2 (A2) SI= MACROS  BO=           S U B R O U T I N E S   ETC.

1572                         DATA    #1
1573                         ENDM
```

```
1575                   MACRO  DELAY
1576                   EXTR   DELAY:    DELAY FOR SPECIFIED TIME
1577                   JST    DELAY:
1578                   PARAMS DELAY,#?,1,2
1579                   IFT    #1<0
1580                   NOTE   X,NEGATIVE.DELAY.ILLEGAL
1581                   ENDC
1582                   IFT    #1=0
1583                   NOTE   X,ZERO.DELAY.ILLEGAL
1584                   ENDC
1585                   IFT    #1>32767
1586                   NOTE   X,TIME.TOO.LARGE
1587                   ENDC
1588                   IFT    #?=1
1589                   DATA   #1,0
1590                   ENDC
1591                   IFT    #?=2
1592                   DATA   #1,#2
1593                   ENDC
1594                   ENDM
```

```
PAGE  0056  11/03/77  10:04:27   94500-10 RTX, IOX, COMX EQUATES
MACRO2 (A2) SI= MACROS  BO=              INTERNAL MACROS

1596                           MACRO   PARAMS
1597                           IFT     #2<#3
1598                           IFT     #3=1
1599                           NOTE    X,#1.NEEDS.AT.LEAST.1.PARAMETER
1600                           ENDC
1601                           IFF     #3=1
1602                           NOTE    X,#1.NEEDS.AT.LEAST.#3.PARAMETERS
1603                           ENDC
1604                           ENDC
1605                           IFT     #2>#4
1606                           NOTE    W,#1.USES.ONLY.#3.PARAMETERS
1607                           ENDC
1608                           ENDM

1610                           MACRO   RTX:2     TO DO REPETITIVE DATA
1611                           DATA    #CNT:
1612                   CNT:    SET     CNT:+1
1613                           ENDM

1615                           MACRO   RTX:1
1616                           DATA    #1
1617                           IFT     #2<0
1618                           NOTE    W,PRIORITY.#2.NEGATIVE,USING.ZERO.(0)
1619                           DATA    0
1620                           ENDC
1621                           IFT     #2>8191
1622                           NOTE    W,PRIORITY.#2.TOO.LARGE,USING.8000
1623                           DATA    8000
1624                           ENDC
1625                           IFT     0<=#2<=8191
1626                           IFT     #2>8000
1627                           NOTE    W,PRIORITY.#2.IN.SYSTEM.RANGE
1628                           ENDC
1629                           DATA    #2
1630                           ENDC
1631                           ENDM
```

```
PAGE  0057  11/03/77  10:04:27   94500-10 RTX, IOX, CONX EQUATES
MACRO2 (A2) SI= MACROS. BO=             INTERNAL MACROS

1632                         MACRO  QUIT
1633                         IFT    LSI305
1634                         JST    *:7F
1635                         ENDC
1636                         IFF    LSI305
1637                         JST    *:FF
1638                         ENDC
1639                         ENDM
```

```
1641                      MACRO  LLL
1642                      IFF    LSI305
1643                      LLL::  :180,#1-1
1644                      ENDC
1645                      IFT    LSI305
1646                      REPT   #1
1647                      LLL:
1648                      ENDC
1649                      ENDM

1651                      MACRO  LLL:
1652                      LLX    1
1653                      RLA    1
1654                      ENDM
1655      0010            FORM   LLL::,12,4
1656                      MACRO  INTSTF   CIB INTERRUPT & DEVICE ADDRESS STUFF
1657                      SPACE  1
1658                DA    EQU    #1       DEFINE DEVICE ADDRESS
1659                INTAD EQU    #2       DEFINE INTERRUPT ADDRESS (DATA)
1660                      ABS    INTAD    ORG TO AUTO I/O LOCATION
1661                      DATA   0,0,0
1662                      ABS    INTAD+4  ORG TO END-OF-BLOCK INTERRUPT
1663                      JST    *$+1     GO TO CIB FOR
1664                      DATA   CIB      . END OF BLOCK
1665                      REL    0        ORG TO RELATIVE ZERO (0)
1666                      SPACE  1
1667                      ENDM

1669                      MACRO  SINT     GENERATE CALL TO SETUP INTERRUPTS
1670                      JST    SINT:    CALL SUBROUTINE
1671                      IFF    #1(S)
1672                      DATA   :3800    MAKES STOP
1673                      NOTE   U,FIRST.PARAMETER
1674                      ENDC
1675                      IFF    #2(I)
1676                      IFF    #2(0)
```

```
                 DATA    :3800     MAKES STOP
                 NOTE    U,SECOND,PARAMETER
                 ENDC
                 ENDC
                 IFT     LSI305
                 IFT     #2[I]
                 DATA    #3+:40E9
                 ENDC
                 IFT     #2[O]
                 DATA    #3+:60E9
                 ENDC
                 ENDC
                 IFF     LSI305
                 IFT     #2[I]
                 DATA    #3+:13E9
                 ENDC
                 IFT     #2[O]
                 DATA    #3+:23E9
                 ENDC
                 ENDC
                 ENDM

                 MACRO   CIB
                 TITL    'RTX/IOX CHARACTER I/O DRIVERS 93302-1XE0
                 TITL    C:#1...CONTROLLER.INFORMATION.BLOCK
                 NAM     C:#1
                 EXTR    SCH:,INTQ:,INTF:
                 XDEF    #4
                 XDEF    #5
                 XDEF    #6
                 INTSIE  #2,:#3
         CIB     EQU     $
         C:#1    DATA    SCH:
                 IFT     LSI305
                 SEA     DA%1+1
                 ENDC
```

11/3-72

```
1713                               IFF     LSI305
1714                               SEL     DA%1+1
1715                               ENDC
1716                               JST     INTQ:
1717                               DATA    $,0,0,0,INTP:,B180,0,C:#1,C:#1,0,0,0
1718                               XREF    #4
1719                               XREF    #5
1720                               DATA    0
1721                               XREF    #6
1722                               DATA    0
1723                               LLA     #7
1724                               DATA    INTAD
1725                               RES     12,0
1726                               DATA    $+1,1,$+1,0
1727                               END
1728                               ENDM
1729                               MACRO   DIB
1730                               TITL    'RTX/IOX CHARACTER I/O DRIVERS 93302-1XE0
1731                               TITL    D:#1...DEVICE.INFORMATION.BLOCK
1732                               NAM     D:#1
1733                               EXTR    C:#3
1734                       D:#1    CHAN    X::
1735                               DATA    C:#3,0,0,:#4
1736                               TEXT    #2
1737                               DATA    0,:#5,:#6,0,:#7
1738                               END
1739                               ENDM
1740                               MACRO   XDEF
1741                               IFF     #1101
1742                               EXTR    I:#1
1743                               ENDC
1744                               ENDM
1745                               MACRO   XREF
1746                               IFF     #1101
1747                               DATA    I:#1
1748                               ENDC
```

```
PAGE  0061  11/03/77  10:04:27   94500-10 WIX, ICX, IMAX EQUATES
MACRO2 (A2) SI= MACROS  BO=              M A C R O S

1749                                     TFT    #1101
1750                                     DATA   0
1751                                     ENDC
1752                                     ENDM
1753                                     SAVE
1754                                     END

0000  ERRORS
0000  WARNING
```

```
PAGE  0001  09/01/76  09:47:37    RTX/IOX CHARACTER I/O DRIVERS 93302-1XE0
MACRO2 (A2) SI= CHRDS   BO=        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 D4D9
      0006 B0B0
0135  0007 020C                    DATA    :20C,:A6E,:4800,0
      0008 0A6E
      0009 4800
      000A 0000
0136                               END

0000  ERRORS
0000  WARNING
```

```
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    RTX/IOX CHARACTER I/O DRIVERS 93302-1XE0
MACRO2 (A2) SI= CHRDS   BO=           C:TY0  ---  TELETYPE

0251  0000                      NAM    C:TY0
0252                            EXTR   INTQ:,INIP:,I:READ,I:RITE,I:FUN
0253                            EXTR   SCH:

0254  0002                      INISTF 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:TY0  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            JSI    INTQ:
0264  0003 0003                 DATA   $,0,0,0,INIP:,8160,0,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
```

```
0269  0016 0000          RES    12,0
0270  0022 0023          DATA   $+1,1,$+1,0
      0023 0001
      0024 0025
      0025 0000
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  CIB        0264  0264
U    0000  DA         0258  0261
X    0000  I:FUN      0252  0266
X    0000  I:READ     0252  0266
X    0000  I:RITE     0252  0266
U    0000  INTAD      0268
X    0000  INTP:      0252  0264
X    0000  INTQ:      0252  0263*
U    0000  LSI305     0257  0260
X    0000  SCH:       0253  0256
0271 SOURCE LINES
```

```
     0000              MACH   0        MUST WORK ON LSI AND ALPHA/NM-16
                * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                *
                *      ORIGINATING NAMES
                *
                * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                *
0000                   NAM    I:READ   INPUT REQUEST
0074                   NAM    I:RITE   OUTPUT REQUEST
0085                   NAM    I:FUN    SPECIAL FUNCTIONS PROCESSOR
0083                   NAM    RITE2    OUTPUT END OF RECORD (CR,LF,ETC)
                * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                *
                *      EXTERNAL NAMES
                *
                * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                *
                       EXTR   BEGIN:   BEGIN TASK SERVICE
                       EXTR   END:     END TASK SERVICE
                       EXTR   SUBR:    BEGIN COMMON SUBROUTINE
                       EXTR   SUBX:    END COMMON SUBROUTINE
                       EXTR   DELAY:   DELAY SERVICE ROUTINE
                       EXTR   EOF:     END OF FILE TASK
                       EXTR   EOR:     END OF RECORD TASK
                       EXTR   EORST:   SET STATUS AND END OF RECORD
                       EXTR   SINT:    SET INTERUPTS TASK
                       EXTR   SIO:     START I/O
                       EXTR   CKSUM:   COMPUTE CHECKSUM TASK
                       EXTR   FETCH:   GET CHARACTER
                       EXTR   EOFCK:   CHECK FOR END-OF-FILE
                       EXTR   BUFFQ:   PUT BYTE INTO BUFFER
                       EXTR   WAIT:    WAIT FOR DEVICE
                       EXTR   EOFQ:    CHECK FOR ASCII EOF ('/*')
```

11/7-80

```
                        * * * * * * * * * * * * * * * * * * * * * * * * * * *
                        *
                        *      THIS ROUTINE WILL PROCESS ANY REQUESTS
                        *      TO INPUT FROM A CHARACTER DEVICE.
                        *
                        *      ALL REQUESTED WILL BE ISSUED FOR ONE (1)
                        *      CHARACTER AT A TIME.
                        *      IF THE TRANSFERRED COUNT CONTAINED IN THE
                        *      CIB IS NEGATIVE THE DEVICE WILL BE
                        *      PROMPTED WITH THE CHARACTER OR CHARACTERS
                        *      CONTAINED IN THE CTCNT.
                        *
                        * * * * * * * * * * * * * * * * * * * * * * * * * * *
                        *
0000                           REL    0
          0000          I:READ EQU    $
                        *
0000 B41B 001B                 LDA    @CBUFF
0001 1350                      LLA    1
0002 9C0E 000E                 STA    @REQCNT  SAVE FOR A RESTART
                        *
          0003          RFA2   EQU    $        R E S T A R T
                        *
0003 9C1B 001B                 STA    @CBUFF   SET BUFFER ADDRESS TO BYTE
0004 C601                      LAP    1
0005 9C23 0023                 STA    @CDCHN1  SET BYTE COUNT TO ONE
0006 0110                      ZAR
0007 9C1D 001D                 STA    @ARROW   CLEAR BACK ARROW FLAG
0008 9C1C 001C                 STA    @CTCNT   CLEAR TRANSFER COUNT
0009 B416 0016                 LDA    @EXCESS  PROMPT CHARACTERS
000A 308E 0019                 JAP    RFAX     IF NONE
000B 1357                      LLA    8        CHECK FOR #
000C 2101 000E                 JAZ    $+2      IF ONLY ONE
000D C601                      LAP    1        TWO PROMPT CHARACTERS
000E 0150                      IAR             ADJUST COUNT
000F 9C04 0004                 STA    @CIMP1   PUT IN MINI-IOB
```

```
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           RFAX   EQU    $
0511   0019 FBD6 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 FBD1 00F4             JST    FETCH:    GET NEXT CHARACTER
0526   0023 FBD1 00F5             JST    BUFFQ:    GO PUT INTO BUFFER
0527   0024 3242 0022             JOR    UNFMTI    GO GET NEXT CHARACTER
0528   0025 F3D0 00F6             JMP    EOR:      END OF RECORD
```

```
PAGE  0004  09/01/76   09:47:37    RTX/IOX CHARACTER I/O DRIVERS 93302-1XE0
MACRO2 (A2) SI= CHRDS    BO=          RFA --- READ FORMATTED ASCII

                          *
          0026            RFA10  EQU    $
                          *
0026 FBCC 00F3                   JST    WAIT:
0027 B40E 000E                   LDA    @REQCNT
0028 F625 0003                   JMP    RFA2      GO RESTART
                          *
          0029            RFA4   EQU    $         MAYBE GOOD
                          *
0029 9C1D 001D                   STA    @ARROW    SET FLAG
002A C0DF                        CAI    :DF       IS IT BACK ARROW?
002B F220 004C                   JMP    RFA9      YES, BACK UP
002C FBC8 00F5                   JST    BUFFO:    PUT INTO BUFFER
                          *
          002D            RFA3   EQU    $         L O O P
                          *
002D FBC6 00F4                   JST    FETCH:    GET CHARACTER
002E FBC8 00F7                   JST    EOFCK:    IF FILE MARK, GOODBYE
002F A2C8 00F8                   IOR    =:80
0030 9C25 0025                   STA    @CDCHN3   HIGH-ORDER BIT ON
0031 C0FF                        CAI    :FF       IS IT RUBOUT?
0032 F605 002D                   JMP    RFA3      YES, IGNORE IT
0033 D2C5 00F9                   CMS    =:8D      HOW ABOUT A CARRIAGE RETURN?
0034 F607 002D                   JMP    RFA3      TOO SMALL
0035 F60C 0029                   JMP    RFA4      GOT A LIVE ONE
                          *
                          *      FOUND CARRIAGE RETURN
                          *
```

```
0559                      *
0560                      *       FOUND CARRIAGE RETURN
0561                      *
0562   0036 C604                  LAP     PROMPT    PROMPTABLE BIT
0563   0037 841F 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 FBB4 00F3             JST     WAIT:     DO A HICCUP
0571   003F FBB0 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    POINER TO MINI-IOB
0575   0044 FBAD 00F2             JST     SIO:      OUTPUT
0576                      *
0577        0045          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     EOFQ:     CHECK FOR END OF FILE
0586                      *
0587        004C          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     RTX/IOX CHARACTER I/O DRIVERS 93302-1XF0
MACRO2 (A2) SI= CHRDS   BO=          RFA --- READ FORMATTED ASCII

0594  0051 0000                DAR                   * DECREMENT BUFFER ADDRESS
0595  0052 9C1B 001B           STA      @CBUFF       * *
0596  0053 F626 002D           JMP      RFA3         GO GET NEXT
```

```
0598                     *       READ A BINARY RECORD
0599                     *       FIND THE RECORD HEADER
0600         0054        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            JAN     $+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 QUEUE 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
```

```
0594   0051 0000                 DAR                    * DECREMENT BUFFER ADDRESS
0595   0052 9C1B 001B            STA     @CBUFF         * *
0596   0053 F626 002D            JMP     RFA3           GO GET NEXT
```

```
                        *      READ A BINARY RECORD
                        *      FIND THE RECORD HEADER
      0054              BININ  EQU    $
0054  FB9F  00F4               JST    FETCH:   GET A CHARACTER
0055  C0FF                     CAI    :FF      IS IT A RUBOUT
0056  F202  0059               JMP    $+3      YES GO GET BYTE COUNT
0057  FB9F  00F7               JST    EOFCK:   CHECK FOR /*
0058  F604  0054               JMP    BININ
                        *
                        *      GET THE BYTE COUNT
                        *
0059  0110                     ZAR
005A  9C0D  000D               STA    @CCSUM   CLEAR THE CHECKSUM
005B  FB98  00F4               JST    FETCH:   GET FIRST CHARACTER
005C  1357                     LLA    8        SHIFT TO HIGH ORDER BYTE
005D  9C1D  001D               STA    @CDAA    SAVE IN TEMP CELL
005E  FB95  00F4               JST    FETCH:   GET SECOND CHARACTER
005F  A41D  001D               IOR    @CDAA    MERGE THE TWO BYTES
0060  3101  0062               JAN    $+2      NOT AN END OF FILE
0061  F399  00FB               JMP    EOF:     AN END-OF-FILE
                        *
                        *      READ THE INPUT DATA
                        *
0062  0310                     NAR
0063  9C0E  000E               STA    @REQCNT  SAVE REQUIRED COUNT
                        *
      0064              NEXT1  EQU    $        GET NEXT ONE
                        *
0064  FB8F  00F4               JST    FETCH:   GO GET NEXT BYTE
0065  FB8F  00F5               JST    BUFFQ:   GO QUEUE INTO BUFFER
0066  DC0E  000E               IMS    @REQCNT  INCREMENT NUMBER OF BYTES
0067  F603  0064               JMP    NEXT1
                        *
                        *      PERFORM CHECKSUM
                        *
0068  C6FF                     LAP    :FF
```

```
0634  0069 8400 000D         AND   @CCSUM    MASK OFF LOW ORDER BITS
0635  006A 9C1D 001D         STA   @CDAA     SAVE RECORD CHECKSUM
0636  006B F888 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 9410 001D         SUB   @CDAA     SUBTRACT COMPUTED CHECKSUM
0642  0071 2101 0073         JAZ   $+2       IF EVERYTHING OK
0643  0072 B289 00FC         LDA   =ERROR    ERROR CODE
0644  0073 F389 00FD         JMP   EORST:    ERROR EXIT
```

```
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 FB7B 00F0                 SINT   STANDARD,OUTPUT,0
0661+ 0075 23F9
0662 0076 B41A 001A                 LDA    @CRCNT    REQUESTED COUNT
0663 0077 9C1C 001C                 STA    @CTCNT    SET TRANSFERED COUNT
0664                         *
0665                         *  WHAT WAS REQUEST
0666                         *
0667 0078 C603                      LAP    3
0668 0079 8419 0019                 AND    @COP      MASK OFF OP-CODE
0669 007A 9C19 0019                 STA    @COP      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 F872 00F2                 JST    SIO:      START I/O
0683 0080 B419 0019                 LDA    @COP
0684 0081 C001                      CAI    1         IF UNFOPMATTED
0685 0082 F373 00F6                 JMP    EOR:      GO TO END UF RECORD
0686                         *
```

TT/3-88

```
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 8D8A        CRLF    DATA    :8D8A,:8A8D,:8D8A,:A08D
       0092 8A8D
       0093 8D8A
       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
```

```
0720  009C 9C0D 000D          STA    @CCSUM    CLEAR THE CHECKSUM BYTE
0721  009D B214 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 B41A 001A          LDA    @CRCNT    REQUESTED COUNT
0728  00A1 9C25 0025          STA    @CDCHN3   PUT INTO CIB
0729  00A2 C602               LAP    2         TRANSFER COUNT
0730  00A3 9C23 0023          STA    @CDCHN1   PUT INTO DATA CHAIN
0731  00A4 B422 0022          LDA    @CDCHN    DATA CHAIN ADDRESS
0732  00A5 FA2E 00D4          JST    I:OCS     GO COMPUT CHECKSUM
0733  00A6 FB4B 00F2          JST    SIO:      START I/O
0734  00A7 FB4B 00F3          JST    WAIT:     WAIT AWHILE
0735                    *
0736       00A8         RITE12 EQU    $         OUTPUT BINARY RECORD
0737                    *
0738  00A8 0030               TXA
0739       0000               IFF    LSI305
0740  00A9 8A54 00FE          ADD    =CRCNT    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    @CCSUM    MASK OFF CHECKSUM
0753  00AF 9C25 0025          STA    @CDCHN3   PUT INTO CIB
0754  00B0 B422 0022          LDA    @CDCHN    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
```

```
                         * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                         *
                         *      THIS ROUTINE WILL PROCESS THE SPECIAL
                         *      FUNCTIONS REQUESTED FOR THE I/O DEVICES
                         *
                         * * * * * * * * * * * * * * * * * * * * * * * * * * * *
                         *
     00B5                I:FUN  EQU    $
                         *
00B5 FB3A 00F0                  SINT   STANDARD,OUTPUT,0
00B6 23F9
00B7 C60F                       LAP    :F
00B8 B419 0019                  AND    @COP      MASK OFF OP CODE
00B9 9C19 0019                  STA    @COP
00BA C00C                       CAI    :C        IF A C
00BB F203 00BF                  JMP    FMARK     JMP TO WRITE FILE MARK
00BC C00D                       CAI    :D        IF A D
00BD F208 00C6                  JMP    PLEAD     JMP TO PUNCH LEADER
00BE F337 00F6                  JMP    EOR:      GO TO END OF RECORD
                         *
     00BF                FMARK  EQU    $         WRITE FILE MARK
                         *
00BF B41F 001F                  LDA    @CFUN1    SPECIAL FLAGS
00C0 1302                       LRA    3         MOVE EOF BIT TO OV
00C1 B20F 00D1                  LDA    SEOF      '/*' FILE MARK
00C2 3201 00C4                  JOR    $+2
00C3 F644 007F                  JMP    RITE1     IF THAT'S IT, QUIT TUIT
00C4 B612 00B2                  LDA    FEOF      RUBOUT-NULL-NULL
00C5 FB2C 00F2                  JST    SIO:      WRITE IT
                         *
     00C6                PLEAD  EQU    $         PUNCH LEADER
                         *
00C6 C714                       LAM    20
00C7 9C16 0016                  STA    @EXCESS   SET COUNT FOR 20 TIMES
00C8 FB2A 00F3                  JST    WAIT:     WAIT A BIT
00C9 B204 00CE                  LDA    LEADER    ADDRESS OF LEADER CHAIN
```

```
PAGE  0013  09/01/76  09:47:37    RTX/IOX CHARACTER I/O DRIVERS 93302-1XE0
MACRO2 (A2) SI= CHRDS   BO=          I : FUN --- FUNCTIONS

0794   00CA F827 00F2              JST    SIO:      OUTPUT 6 NULLS
0795   00CB DC16 0016              IMS    @EXCESS   ARE WE DONE
0796   00CC F604 00C8              JMP    $-4       NO
0797   00CD F328 00F6              JMP    EOR:      JMP END OF RECORD
0798   00CF 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 B40D 000D               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
```

```
PAGE  0015  09/01/76  09:47:37    RTX/IOX CHARACTER I/O DRIVERS 93302-1XE0
MACRO2 (A2) SI= CHRDS   BO=          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    $         COMPUT CHECKSUM FOR NEXT
0843                        *
0844   00DF 0E00                 SBM              SET BYTE MODE
0845   00E0 B400 0000            LDAB   @0        LOAD OUTPUT BYTE
0846   00E1 0F00                 SWM              SET WORD MODE
0847   00E2 FB1C 00FF            JST    CKSUM:    GO COMPUTE CHECKSUM
0848   00E3 0000            I:OCS5 DATA   $-$       CHECKSUM DATA CELL
0849   00E4 0128                 IXR              INCREMENT BUFFER ADDRESS
0850   00E5 DA09 00EF            IMS    COUNT     INCREMENT COUNT DONE
0851   00E6 F607 000F            JMP    I:OCS4    NOT DONE
0852                        *
0853        00E7            I:OCS9 EQU    $         ALL DONE SO CLEAN HOUSE
0854                        *
0855   00E7 8604 00E3            LDA    I:OCS5    COMPUTED CHECKSUM
0856   00E8 E204 00ED            LDX    TMP2      RESTORE X REGISTER
0857   00E9 9C0D 000D            STA    @CCSUM    PUT CHECKSUM IN CIB
0858   00EA 8203 00EE            LDA    TMP3
0859   00EB 000D                 DAR              RESTORE A REGISTER
0860   00EC F718 0004            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
```

```
PAGE  0016  09/01/76  09:47:37    RTX/IOX CHARACTER I/O DRIVERS 93302-1XE0
MACRO2 (A2) SI= CHRDS    RO=        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
```

```
PAGE 0001  09/01/76  09:54:40    RTX/IOX CHARACTER I/O DRIVERS 93302-1X

U    0000   ARROW      0488* 0539* 0579*
X    0000   BEGIN:     0444
     0600   BININ      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   CDAA       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   CIOB       0498
X    0000   CKSUM:     0454  0847*
U    0000   CUP        0513  0668  0669* 0683  0770  0771*
     0863   COUNT      0837* 0850*
U    0000   CRCNT      0662  0674  0677  0698* 0701  0704  0727  0740
                       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   EORMSK     0565  0691
X    0000   EORST:     0451  0644
U    0000   ERROR      0643
U    0000   EXCESS     0490  0791* 0795*
     0799   FEOF       0785
X    0000   FETCH:     0455  0525* 0546* 0601* 0611* 0614* 0620* 0636*
                       0639*
```

```
      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   0706   0735   0759
                        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*  0640
      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   RHEDAD     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/IOX 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   SUBR:       0446
X    0000   SUBX:       0447
     0861   TMP2        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

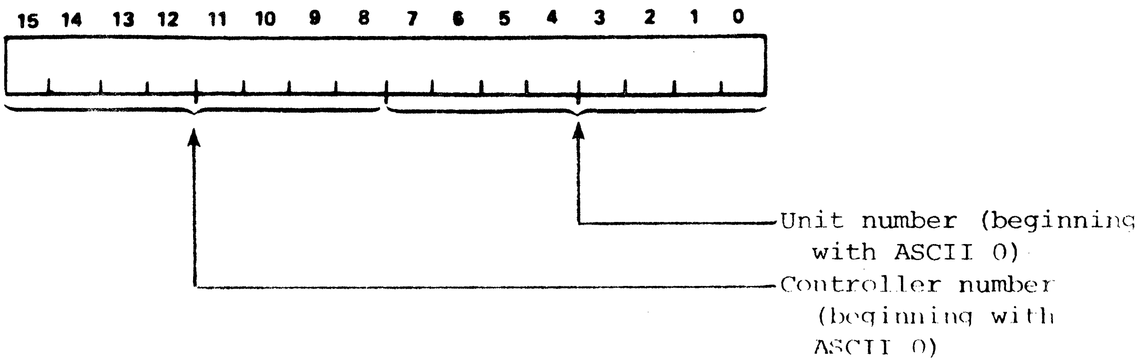
<table>
<tr><th>STANDARD NAME *</th><th colspan="7">15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0</th><th>word</th><th></th></tr>
<tr><td>CHAIN</td><td colspan="7">DIB CHAIN ADDRESS</td><td>0</td><td></td></tr>
<tr><td>DCIB</td><td colspan="7">CIB ADDRESS</td><td>1</td><td></td></tr>
<tr><td>QUEUE</td><td colspan="7">USED BY IOX TO QUEUE REQUESTS</td><td>2</td><td></td></tr>
<tr><td>CN</td><td colspan="7">COORDINATION NUMBER</td><td>3</td><td></td></tr>
<tr><td>DSW</td><td colspan="7">DEVICE SPECIFICATION WORD</td><td>4</td><td></td></tr>
<tr><td>DT</td><td colspan="7">DEVICE NAME</td><td>5</td><td rowspan="2">IOB Words 1 and 2</td></tr>
<tr><td>DCUN</td><td colspan="3">CONTROLLER NUMBER</td><td colspan="4">UNIT NUMBER</td><td>6</td></tr>
<tr><td>DDEL</td><td colspan="3">INPUT RTC TICKS</td><td colspan="4">OUTPUT RTC TICKS</td><td>7</td><td></td></tr>
<tr><td>DFUN</td><td>FUNCTION CODE</td><td>FUNCTION CODE</td><td>FUNCTION CODE</td><td>FUNCTION CODE</td><td>FILE MARK</td><td>PF</td><td>END OF RECORD</td><td>8</td><td></td></tr>
<tr><td>DULS</td><td colspan="3">MAX BYTES-ASCII</td><td colspan="4">MAX BYTES-BINARY</td><td>9</td><td></td></tr>
<tr><td>DERRC</td><td colspan="7">HARDWARE ERROR COUNT (except MTIC)</td><td>10</td><td></td></tr>
<tr><td></td><td colspan="7">ADDITIONAL WORDS USED BY SPECIFIC HANDLERS</td><td>11
17</td><td></td></tr>
</table>

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

Figure 4-1. DIB Configuration

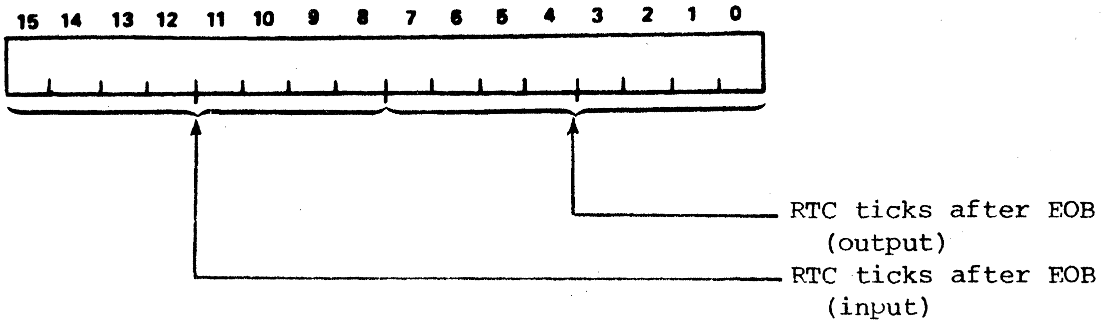
4.2 REGULAR DIB CONFIGURATION (ALL HANDLERS) - WORDS 0 TO 10

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.

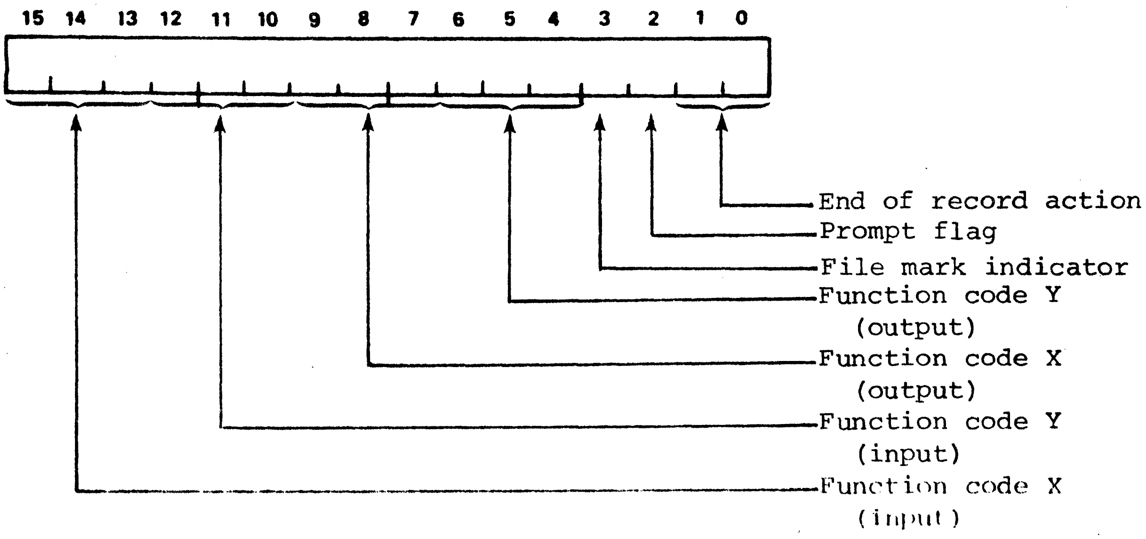
Bits 0-7. A binary value representing the number of Real-Time Clock ticks to delay after an end-of-block interrupt for an output operation, before the device is considered available for the next I/O operation.

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

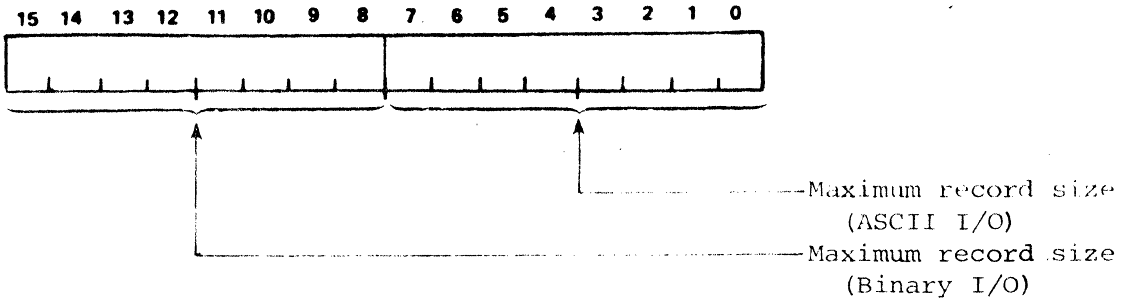
Bits 0-1. These bits represent the end of record action to be taken for Formatted ASCII output:

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.

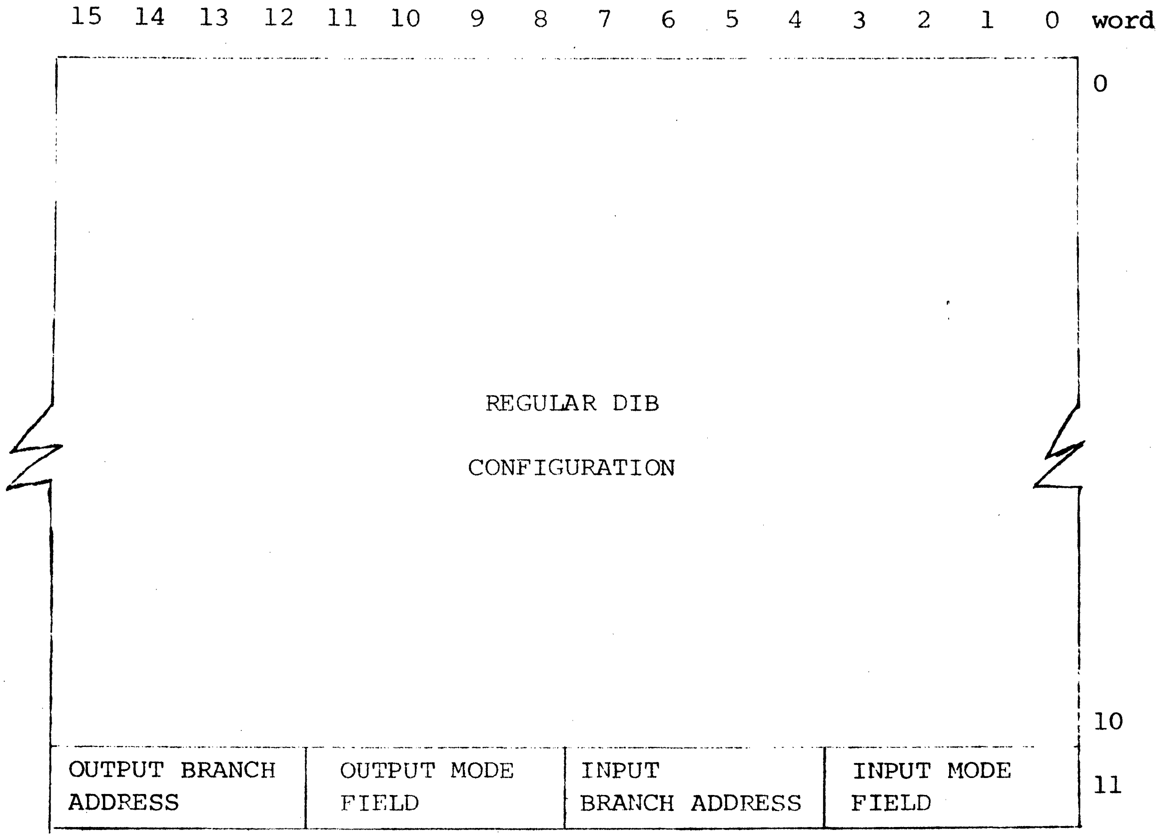
NOTE

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

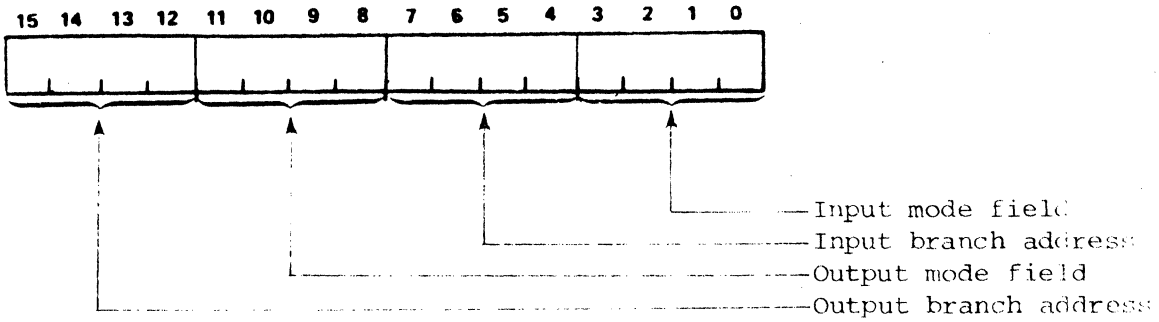
4.3 ADDITIONAL DIB CONFIGURATIONS - UP TO 18 WORDS

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:



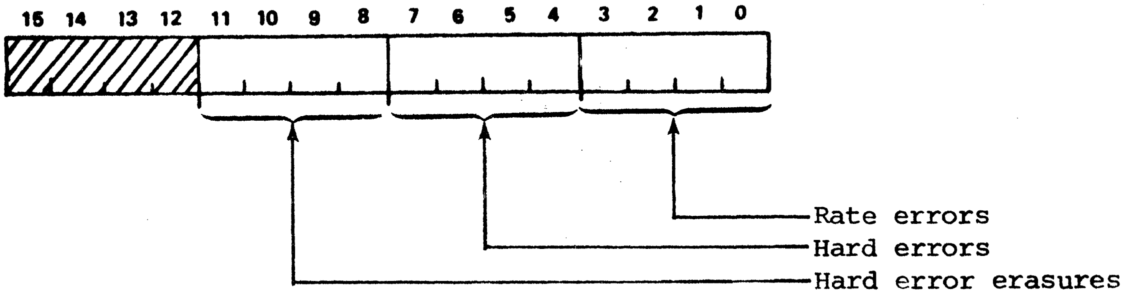
Bits 15-12. Branch Address Field of DIO Command Word for output.
Bits 11-8. Mode Field of DIO Command Word for output.
Bits 7-4. Branch Address field of DIO Command Word for input.
Bits 3-0. Mode Field of DIO Command Word for input.

4.3.2 Magnetic Tape Intelligent Cable DIB

<table>
<tr><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th><th>word</th></tr>
<tr><td colspan="16">REGULAR DIB
CONFIGURATION</td><td>0
9</td></tr>
<tr><td colspan="4">NOT USED</td><td colspan="4">HARD ERROR ERASURES</td><td colspan="4">HARD ERRORS</td><td colspan="4">RATE ERRORS</td><td>10</td></tr>
<tr><td colspan="8">:FB</td><td colspan="7">0</td><td>TRANS ADDR</td><td>11</td></tr>
<tr><td colspan="16">MINIMUM RECORD LENGTH</td><td>12</td></tr>
</table>

Word 10

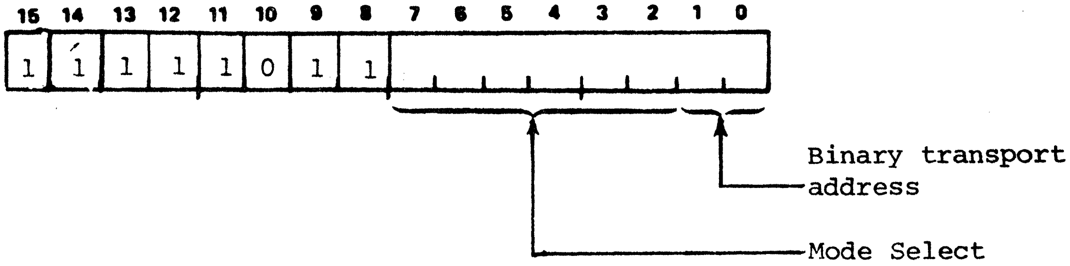
Three MTIC error counters. This word uses the following format:



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.

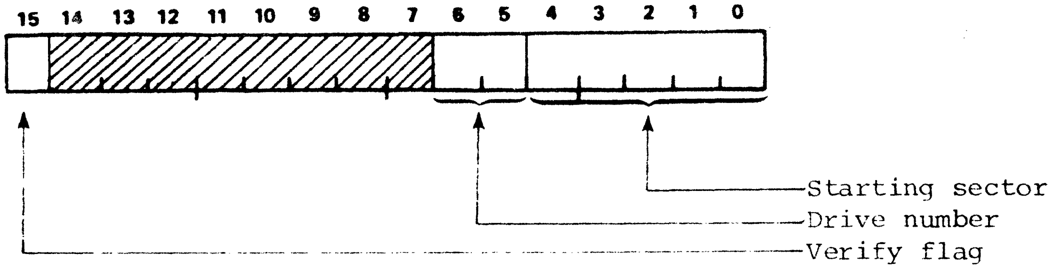
Write requests with a byte count less than the minimum record length will have additional characters appended to the record until the byte count equals the contents of word 12. Blanks are appended to ASCII records and zeros are appended to Binary records.

Read request will return only the number of characters requested.

4.3.3 Disk DIB

<table>
<tr><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th><th>word</th></tr>
<tr><td colspan="16">REGULAR DIB
CONFIGURATION</td><td>0
10</td></tr>
<tr><td>VER-
IFY</td><td colspan="8">NOT USED</td><td colspan="2">DRIVE #</td><td colspan="5">STARTING SECTOR</td><td>11</td></tr>
<tr><td colspan="8">NUMBER OF SECTORS PER TRACK</td><td colspan="8">STARTING HEAD</td><td>12</td></tr>
<tr><td>I
N
C</td><td colspan="7">NUMBER OF SECTORS
PER CYLINDER</td><td colspan="8">STARTING CYLINDER</td><td>13</td></tr>
<tr><td colspan="16">NUMBER OF SECTORS IN THE EXTENT</td><td>14</td></tr>
</table>

Word 11 This word uses the following format:



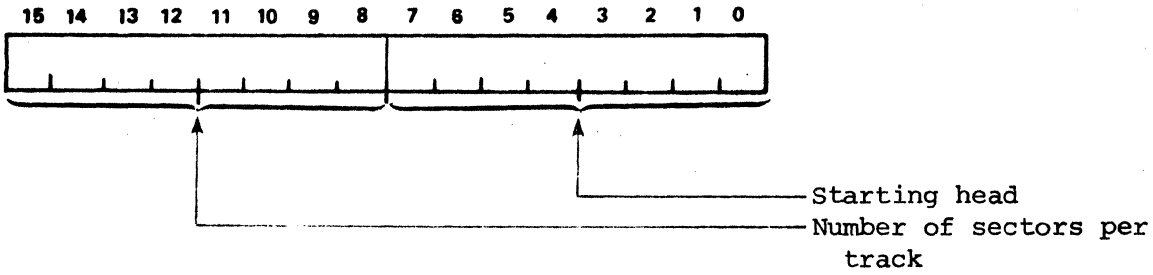
Bit 15. Verify flag
If this bit is set, a verify operation will be performed after each write. Two additional attempts will be made to re-write the record before the error bit in the status is set.

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.

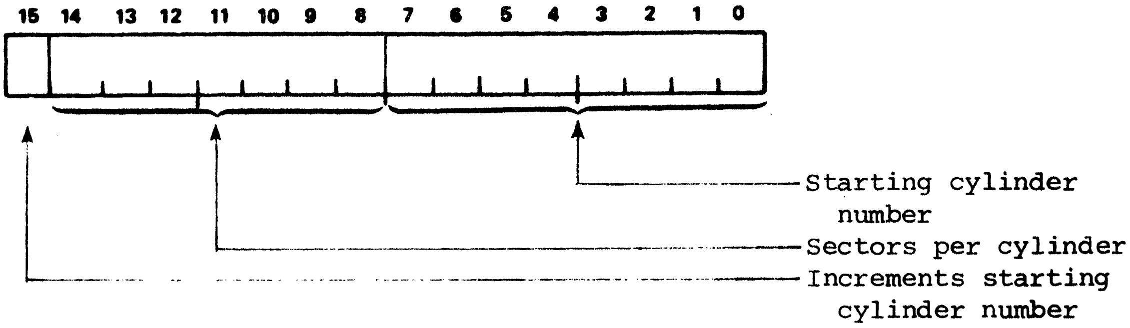
Word 12 This word uses the following format:



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.

Word 13 This word uses the following format:



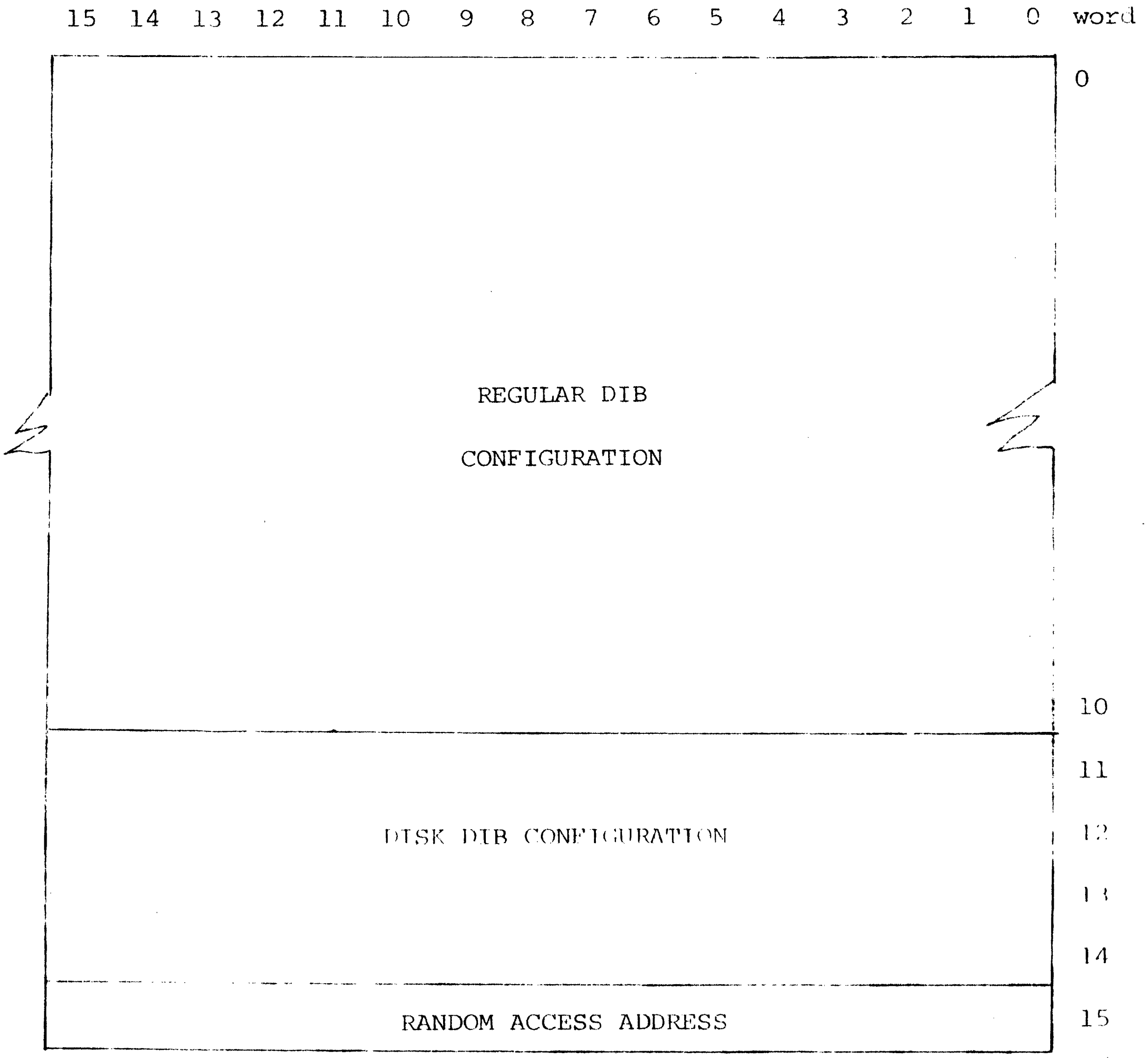
Bit 15. If this bit is set, the contents of bits 7-0 are incremented by 256.

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

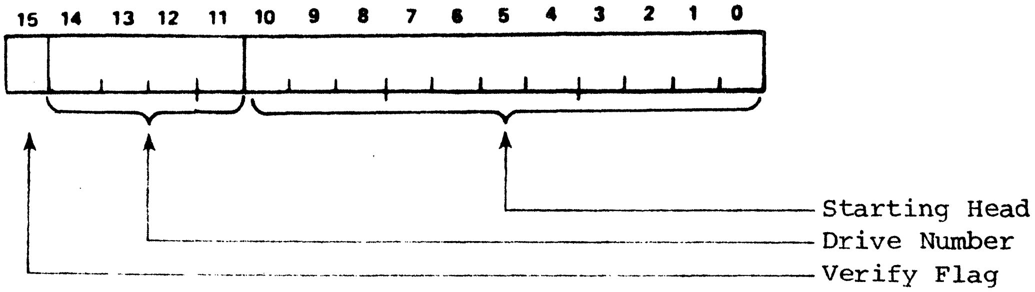


Word 15 Random Access Address
This word provides a location other than the user's IOB to store the record number.

4.3.5 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	STARTING HEAD	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.

4.4 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:DK0          CIB Reference
*
D:DKXX        EQU         $
              CHAN        X::            Chain link to other DIB's
              DATA        C:DK0          CIB Address
              DATA        0              IOX temp cell
              DATA        0              Coordination number
              DATA        :0011          DSW: Direct access Read/Write
              DATA        'DK, 'XX'      Device name
              DATA        0              EOB delay (none required)
              DATA        0              FC's, flags (none required)
              DATA        0              Max record size
              DATA        0              Error count
              DATA        0              Drive 0, starting sector 0
              DATA        :C02           Sectors per track = 12
*                                        Starting head number = 2

              DATA        :1800          Sectors per cylinder = 24
*                                        Starting cylinder number = 0
              DATA        :108           Sectors per extent = 264
                                         (24 sectors x 11 cylinders)
```

4.5 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 CIB 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:--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
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
REQCNT	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
CDAA	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 EOR: routine for storage of status.

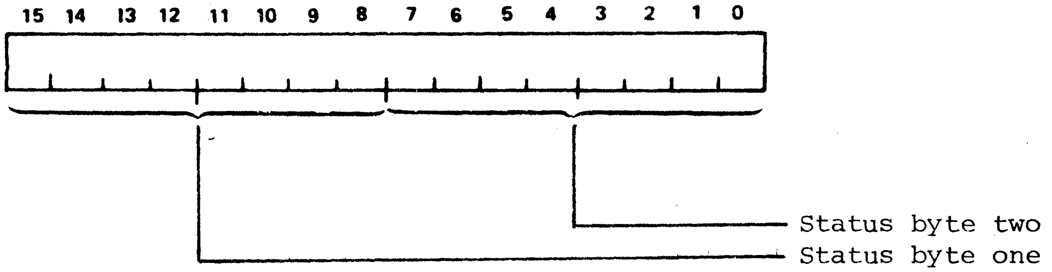
Word 33 Temp Cell. Used by the SIO: and WAIT: routines to store their return addresses.

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.)

	Non-DIO	Fortran Non-DIO	DIO	Fortran DIO
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:PPO	C:PPO	C:PPD	C:PPD
Line Printer	C:LPO	C:LPF	C:LPD	C:LPFD
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:MTO	C:MTO	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.

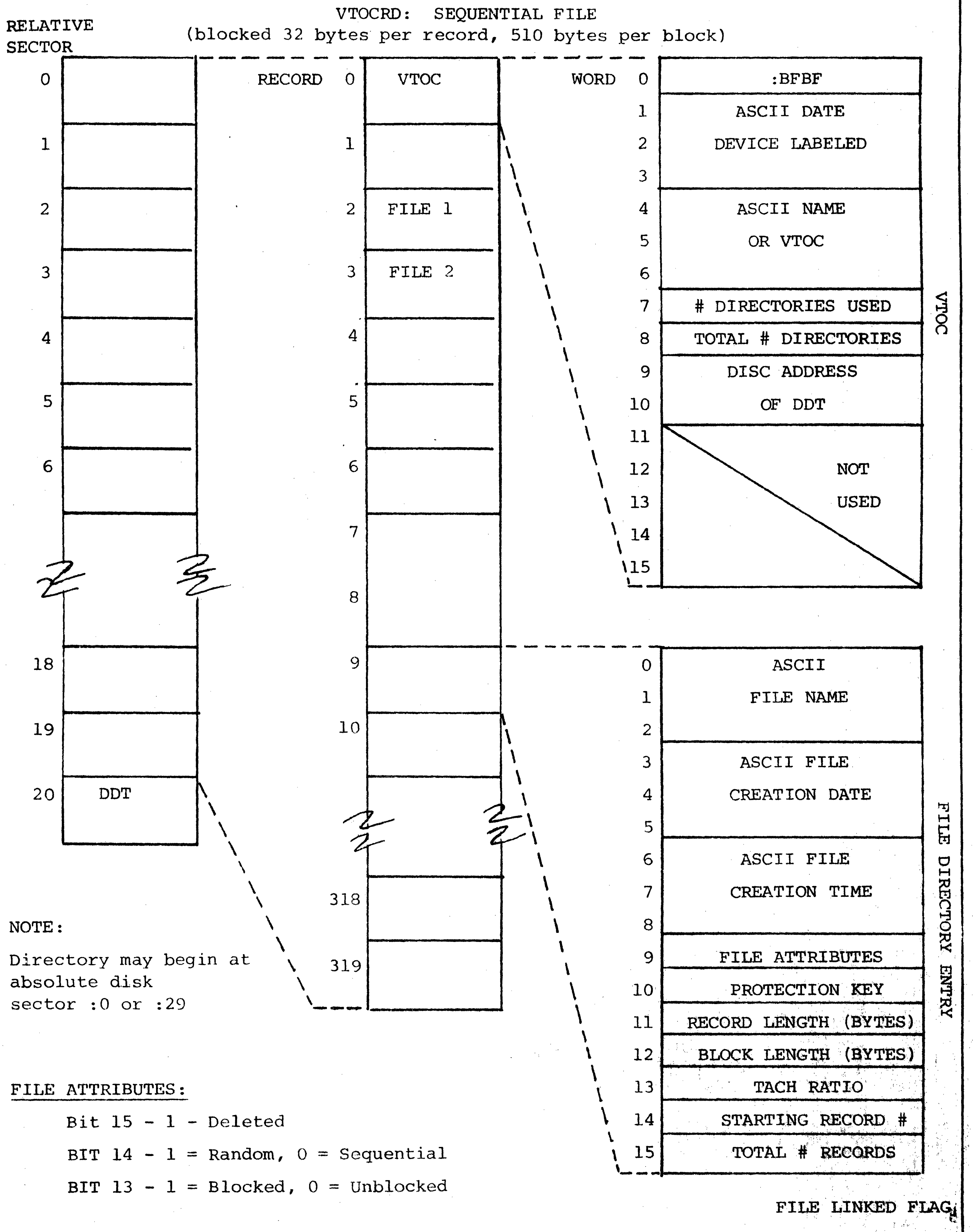
All requests for file access are made through IOX (IO:) using Logical Units (LUNs). The File Manager calls standard IOX device drivers using Logical Units for the required physical I/O. LUN assignments for files as well as LUNs for use by the File Manager for physical I/O are made in the Unit Assignment 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. This utility is described in Section 5.3.

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.)



NOTE:

Directory may begin at absolute disk sector :0 or :29

FILE ATTRIBUTES:

Bit 15 - 1 - Deleted

BIT 14 - 1 = Random, 0 = Sequential

BIT 13 - 1 = Blocked, 0 = Unblocked

Figure 5-1. Disk Directory Structure

WORD	0	LAST PHYSICAL SECTOR NUMBER IN PARTITION	1 PARTITION
	1		
	2		2
	3		
	4		3
	5		
	6		4
	7		
	8		5
	9		
	10		6
	11		
	12		7
	13		
	14		8
	15		
	16	NEXT RELATIVE RECORD (ACTUAL)	1 PARTITION
	17	NEXT RELATIVE RECORD (WORKING)	
	18		2
	19		
	20		3
	21		
	22		4
	23		
	24		5
	25		
	26		6
	27		
	28		7
	29		
	30		8
	31		

NOTE:
Entry is zero i
that partition
does not exist

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

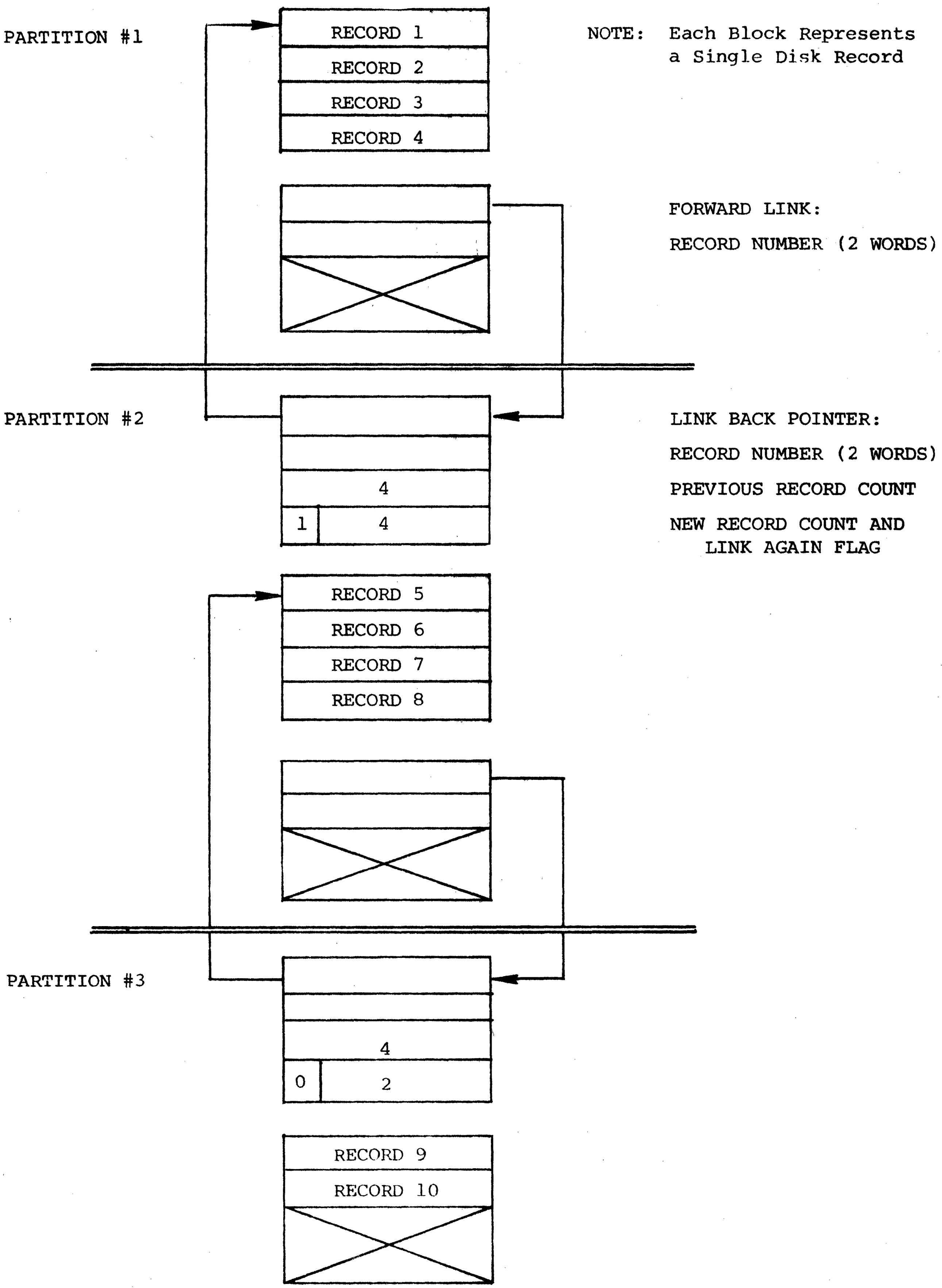


Figure 5-3. Disk File Linkage

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.

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

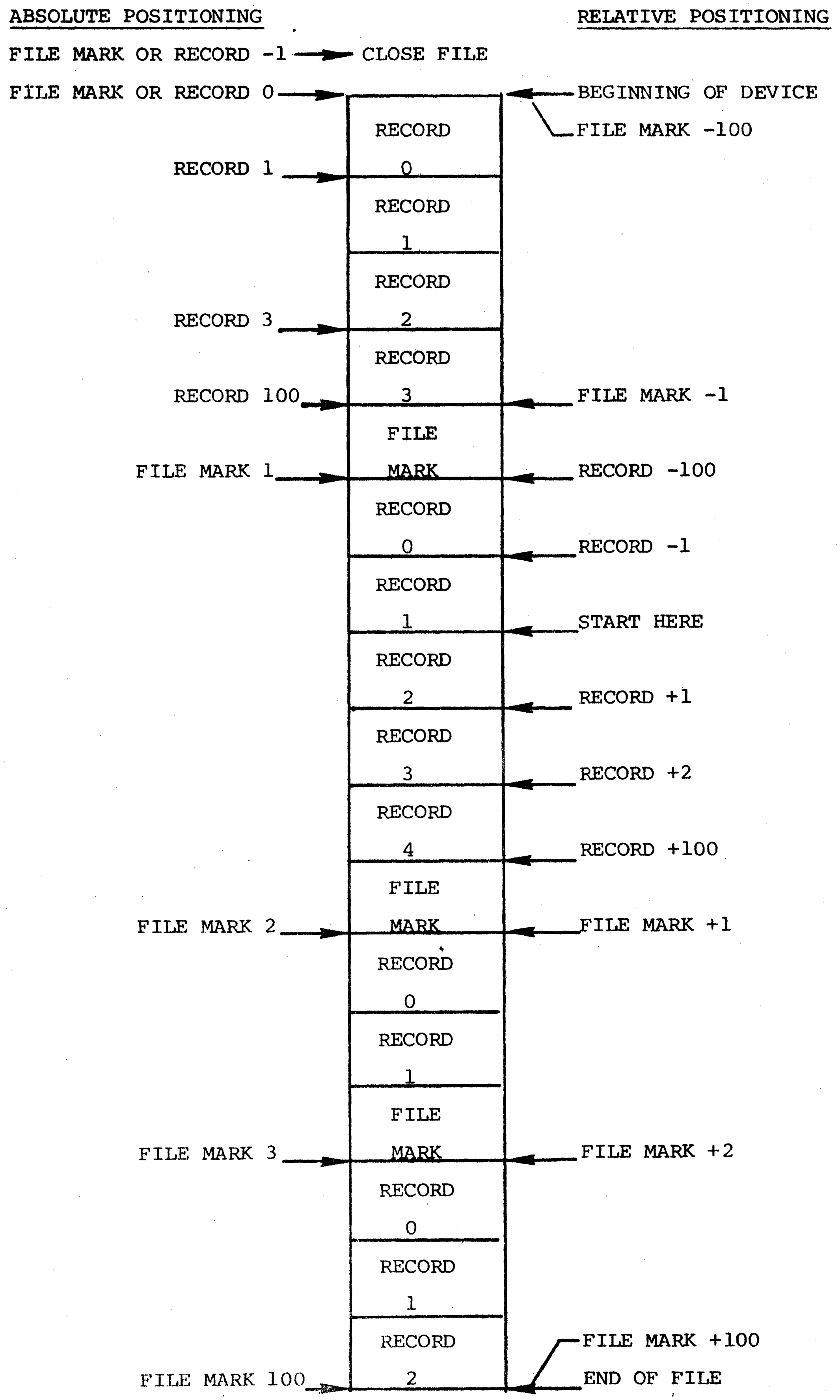


Figure 5-4. Sequential File Positioning Examples

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:DK0 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.

5.2.1 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 26 are given below. (Refer to Figure 5-6.)

Words 0-6	Standard for IOX.
Words 7-9	Standard for IOX, but <u>must</u> be set to zero for File Manager.

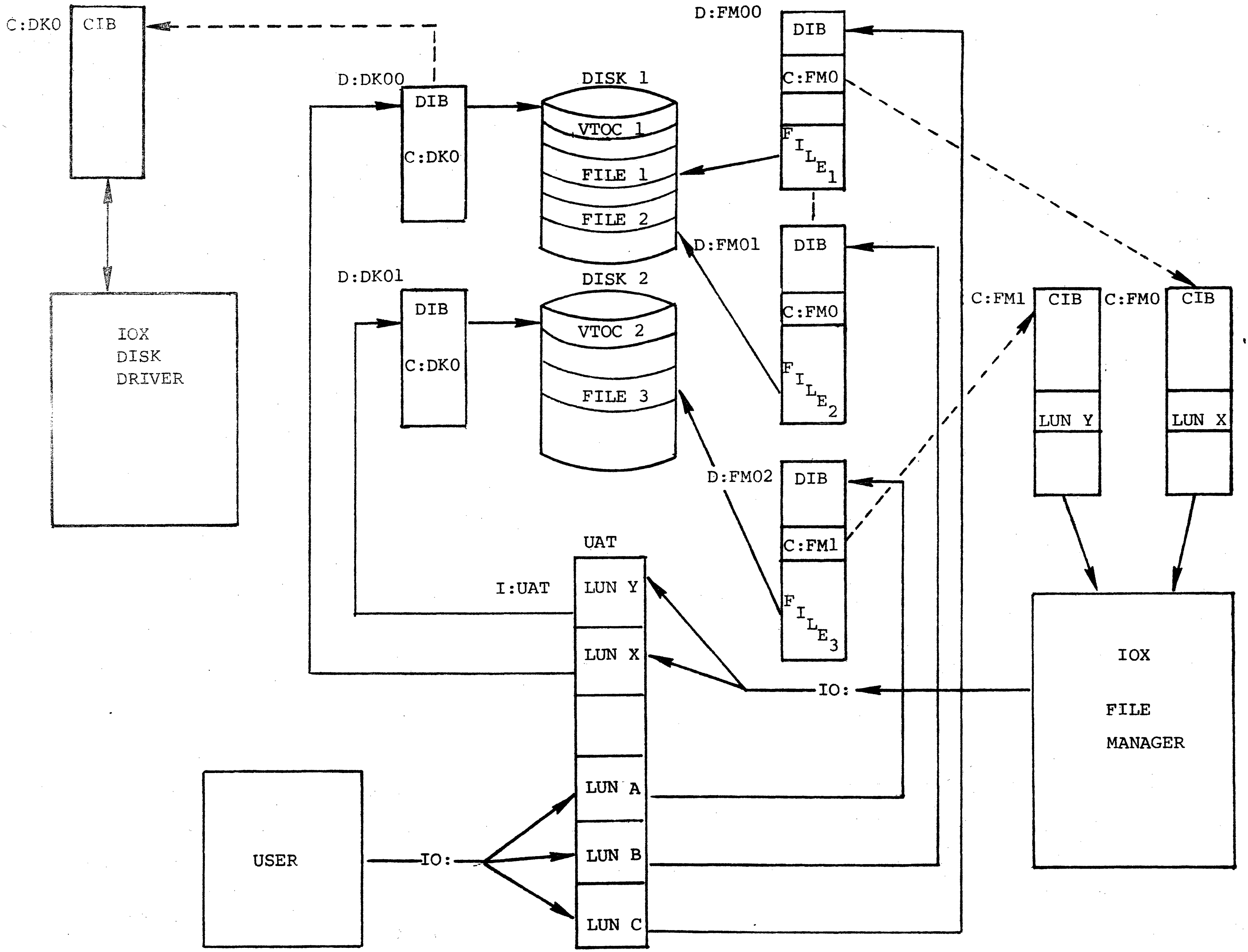


Figure 5-5. Table Organization

0	STANDARD FOR IOX	
9		
10	PHYSICAL I/O ERROR STATUS	DHST
11	FILE STATUS WORD	DFST
12	FILE NAME	DFNAM
13		
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 sequentail access was made on a ramdom 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).

5.2.2 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		
	IOB		
10			
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 compatable 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 othewise mal functions 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    EXAMPLE FILE MANAGER APPLICATION
MACRO2  (A2) SI= EX:S     BO=          ** 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       PIO    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    $
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   PIO      PHYSICAL I/O LOGICAL UNIT
0021  0005 0000              DATA   D:DK01   PHYSICAL I/O LOGICAL UNIT
0022  0006 FFF8       I:UAT  DATA   UATTOP-$-2
0023                         END

0000  ERRORS
0000  WARNING
```

```
PAGE  0001  03/11/77  15:36:32     EXAMPLE FILE MANAGER APPLICATION
MACRO2 (A2) SI= EX:S     BO=             ** RTX MAINLINE CODE **

0026                         *
0027                         *       RTX MAINLINE CODE
0028                         *
0029  0000                           NAM     MAIN
0030                                 EXTR    TASK
0031                                 EXTR    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:       INITTIALIZE RTX
0037  0001 000A                      DATA    N
0038  0002 000B                      DATA    WKAREA
0039  0003 0800                      HLT                ERROR
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
      0008 0000
      0009 0000
      000A 0000
0045  000B 0000              WKAREA RES     5*N,0      RTX WORK AREA
0046         0000                    END     MAIN

0000  ERRORS
0000  WARNING
```

```
PAGE  0001  03/11/77  15:36:32    EXAMPLE FILE MANAGER APPLICATION
MACRO2 (A2) SI= EX:S    BO=             ** APPLICATION PROGRAM **

0049                         *
0050                         *       APPLICATION PROGRAM. THIS COPIES EVERY THIRD RECORD FORM
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      OPCODE 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 001D                 JMP     CHECK      ABNORMAL RETURN, TEST FOR END OF FILE
0070                         *
0071  000A C601                      LAP     NEW        LOGICAL UNIT FOF FILE "NEW"
0072  000B 9A37 0043                 STA     IOB+4      SET UP IOB
0073  000C C605                      LAP     :0005      OPCODE FOR SEQUENTIAL WRITE
0074  000D 9A36 0044                 STA     IOB+5
0075  000E 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      OPCODE FOR POSITION RELATIVE RECORDS
0083  0015 9A2E 0044                 STA     IOB+5
0084  0016 C602                      LAP     2          FORWARD RECORD COUNT TO SKIP
```

```
PAGE  0002  03/11/77  15:36:32     EXAMPLE FILE MANAGER APPLICATION
MACRO2 (A2) SI= EX:S     BO=             ** APPLICATION PROGRAM **

0085   0017 9A20 0045              STA    IOB+6
0086   0018 F820 0039              JST    IO:        SKIP TWO RECORDS ON FILE "OLD"
0087   0019 003F                   DATA   IOB
0088   001A F81F 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     CHECK    EQU    $
0093   001D B226 0044              LDA    IOB+5      GET IOB COMPLETION STATUS
0094   001E 821C 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     ERROR    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                   LAM    1          COUNT = -1
0104   0023 9A21 0045              STA    IOB+6      SET UP IOB
0105   0024 C609                   LAP    :0009      OPCODE FOR POSITION ABSOLUTE FILES
0106   0025 9A1E 0044              STA    IOB+5
0107   0026 C602                   LAP    OLD        LOGICAL UNIT OF FILE "OLD"
0108   0027 9A1B 0043              STA    IOB+4
0109   0028 F810 0039              JST    IO:        CLOSE READ FILE "OLD"
0110   0029 003F                   DATA   IOB
0111   002A F80F 003A              JST    END:
0112   002B F60B 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 F80A 0039              JST    IO:        CLOSE WRITE FILE "NEW"
0117   002F 003F                   DATA   IOB
0118   0030 F809 003A              JST    END:
0119   0031 F611 0020              JMP    ERROR      ABNORMAL RETURN
0120                      *
```

```
PAGE  0003  03/11/77  15:36:32     EXAMPLE FILE MANAGER APPLICATION
MACRO2 (A2) ST= EX:S    BO=             ** APPLICATION PROGRAM **

0121   0032 F607 003A            JST     END:      ALL DONE
0122   0033 CDCD         DATE:   TEXT    'MMDDYY'
       0034 C4C4
       0035 D9D9
0123   0036 C8C8         TIME:   TEXT    'HHMMSS'
       0037 CDCD
       0038 D3D3
0124        0006                 LP00I
       0039 0000
       003A 0000
       003B 0200
       003C
       003D
       003E
0125                     *
0126                     *       INPUT OUTPUT BLOCK (IOB)
0127                     *
0128   003F 0000         IOB     DATA    0,0,0,0,$-$,$-$,$-$,BUFFER,0,0
       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
```

```
PAGE  0001  03/11/77  15:36:32  EXAMPLE FILE MANAGER APPLICATION
MACRO2 (A2) ST= EX:S     BO=             ** FILE MANAGER CTB **

0135  0000                        NAM    C:FM0
0136                              EXTR   FM:REA
0137                              EXTR   FM:WRT
0138                              EXTR   FM:POS
0139                              EXTR   FM:FUN
0140                      *
0141        0000          C:FM0   EQU    $         FILE MANAGER CTB
0142  0000 0000                   RES    5,0
0143  0005 0003                   DATA   PIO       PHYSICAL I/O LOGICAL UNIT
0144  0006 0000                   RES    8,0
0145  000E 002B                   DATA   OCBUF     OPEN/CLOSE BUFFER ADDRESS
0146  000F 0000                   DATA   FM:REA    READ ENTRY POINT
0147  0010 0000                   DATA   FM:WRT    WRITE ENTRY POINT
0148  0011 0000                   DATA   FM:POS    POSITION ENTRY POINT
0149  0012 0000                   DATA   FM:FUN    FUNCTION ENTRY POINT
0150  0013 0000                   RES    24,0
0151                      *
0152  002B 0000           OCBUF   RES    256,0     OPEN/CLOSE BUFFER
0153                              END

0000  ERRORS
0000  WARNING
```

```
PAGE  0001  03/11/77  15:36:52     EXAMPLE FILE MANAGER APPLICAITON
MACRO2 (A2) SI= EX:S    BO=               ** FILE DTB **

0156                       *
0157                       *      OLD FILE DTB. DEVICE STATUS WORD IS SET TO ALLOW FILE READ
0158                       *      OR POSITION OPERATIONS ONLY.
0159                       *
0160  0000                        NAM    D:FM0
0161                              EXTR   C:FM0
0162                       *
0163  0000                 D:FM0  CHAN   X::
0164  0001 0000                   DATA   C:FM0     FILE MANAGER CTB ADDRESS
0165  0002 0000                   RES    2,0
0166  0004 0F0F                   DATA   :0F0F     DEVICE STATUS WORD
0167  0005 C6CD                   TEXT   'FM00'     DEVICE NAME
      0006 B0B0
0168  0007 0000                   RES    4,0
0169  000B 0000                   DATA   :0000     FILE STATUS WORD
0170  000C CFCC                   TEXT   'OLD    '  FILE NAME
      000D C4A0
      000E A0A0
0171  000F 0000                   RES    9,0
0172  0018 001C                   DATA   BBUF      BLOCKING BUFFER ADDRESS
0173  0019 0000                   RES    3,0
0174                       *
0175  001C 0000            BBUF   RES    510+2/2,0 BLOCKING BUFFER, SIZE = BLOCK SIZE+2 BYTES
0176                              END

0000  ERRORS
0000  WARNING
```

```
PAGE  0001  03/11/77  15:36:52     EXAMPLE FILE MANAGER APPLICAITON
MACRO2 (A2) ST= EX:S    BO=             ** FILE DTB **

0179                          *
0180                          *      NEW FILE DTB. 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   A::
0188  0001 0000                      DATA   C:FM0     FILE MANAGER CTB ADDRESS
0189  0002 0000                      RES    2,0
0190  0004 FFFF                      DATA   :FFFF     DEVICE STATUS WORD
0191  0005 C6CD                      TEXT   'FM01'     DEVICE NAME
      0006 B0B1
0192  0007 0000                      RES    4,0
0193  000B 0000                      DATA   :0000     FILE STATUS WORD
0194  000C CEC5                      TEXT   'NEW    '  FILE NAME
      000D D7A0
      000E A0A0
0195  000F 0000                      RES    2,0
0196  0011 0050                      DATA   80        RECORD SIZE
0197  0012 0000                      DATA   0         BLOCK SIZE (UNBLOCKED)
0198  0013 0000                      RES    5,0
0199  0018 0000                      DATA   0         BLOCKING BUFFER ADDRESS (NONE REQUIRED)
0200  0019 0000                      RES    5,0
0201                          *
0202                                 END

0000  ERRORS
0000  WARNING
```

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

6.1.4 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

6.1.5 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

6.2.3 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 te 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 <u>past</u> 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 INTELLIGNET 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. Backsapce. Character backspace is implemented using the back arrow (←) character. One character is erased for each back arrow character input. Since it is impossible to physically backsapce 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 proprely 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.

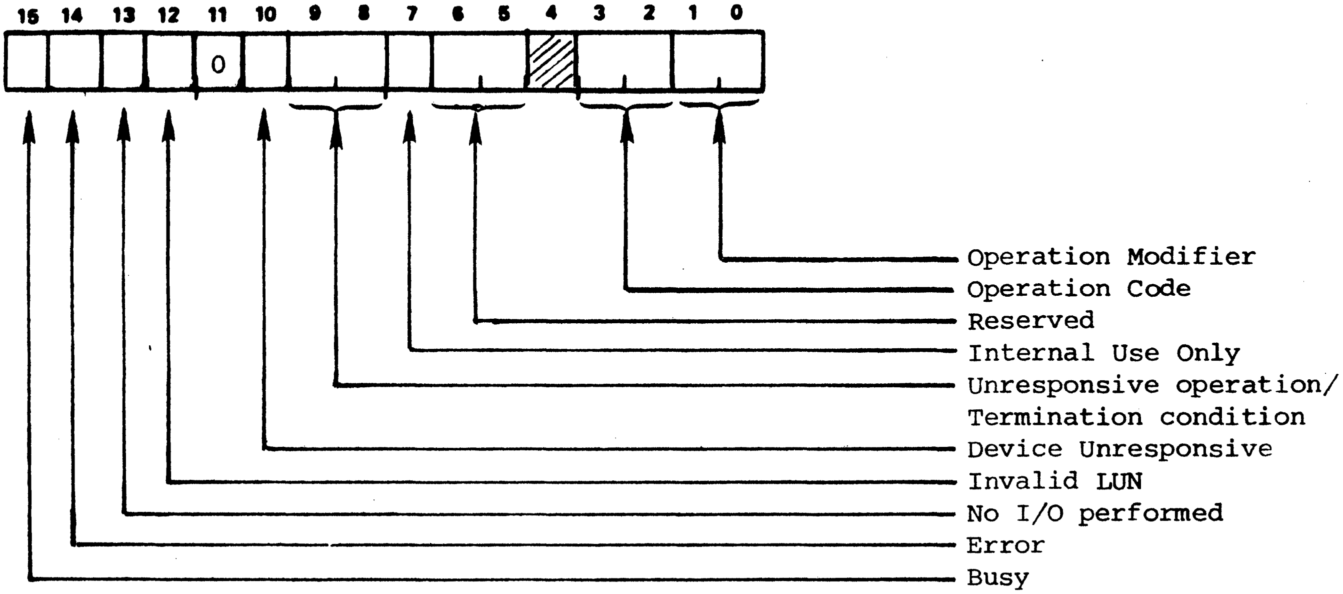
INPUT/OUTPUT BLOCK
FOR THE IEEE INTELLIGENT CABLE HANDLER

Standard Name	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	word
																	0
IDT	STANDARD IOB																1
ICUN or ITCB	CONFIGURATION																2
CN																	3
ILUN																	4
ISTA,IOP	BUSY	ERROR	NO I/O	BAD LUN	0	DE-VICE	OP OR TERM COND.		IN-TERN	RES.		▨	OP CODE		OP MOD.		5
IRCNT	USED FOR ALL REQUESTED FUNCTION CODES																6
IBUFF	USED FOR ALL REQUESTED FUNCTION CODES																7
IACNT	USED FOR ALL REQUESTED FUNCTION CODES																8
IRCNTU	USED IF OP CODE IS 00 OR 01																9
IBUFFU	USED IF OP CODE IS 00 OR 01																10
ITIME	USED IF OP CODE IS 00 OR 01																11

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.

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

Format 1

Op Code (bits 3 and 2)	Op Modifier (bits 1 and 0)	Function
00	00	Write control and read to END or stop character. (CIB location CSTPCH is the stop character)
00	10	Same as 0000 with parity standardization.
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.

Operations 0000 and 0010 require a DMA IOD or a standard IOD with a programmable stop character.

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 (Bits 3 and 2)	Op Modifier (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

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0	word
NOT USED	6
NOT USED	7
BYTE REQUEST	8

Words 6 and 7 are not used but must be provided.

Word 8 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.

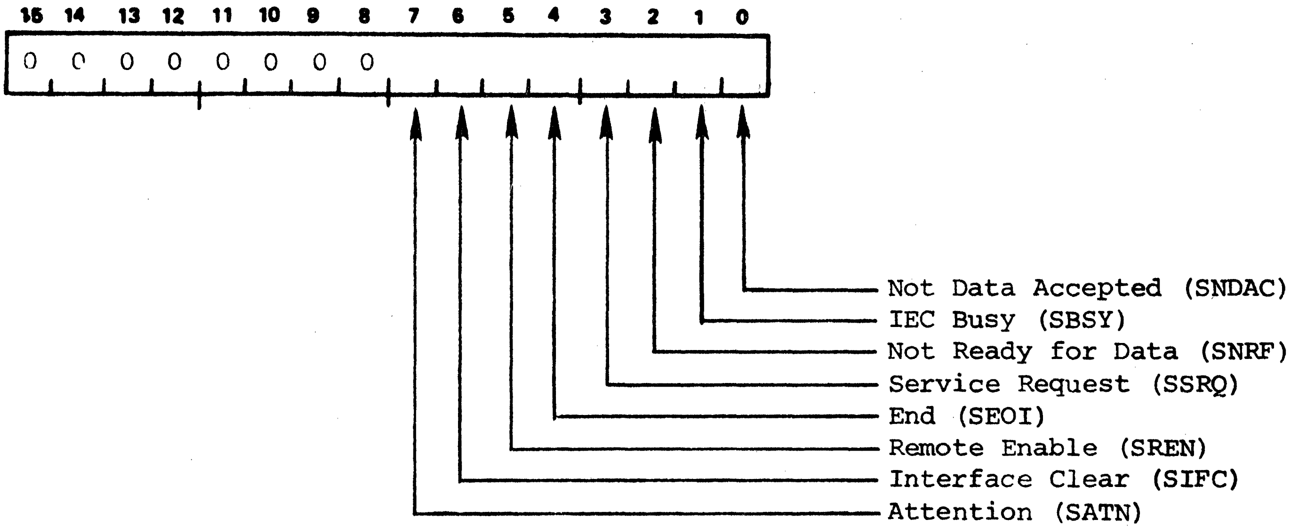


Figure 7-2. IEC Status Byte Configuration

Format 4

Op Code (bits 3 and 2)	Op Modifier (bits 1 and 0)	
11	01	Set IEC control lines

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0	word
NEW CONTROL LINE VALUES	6
NOT USED	7
NOT USED	8

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.

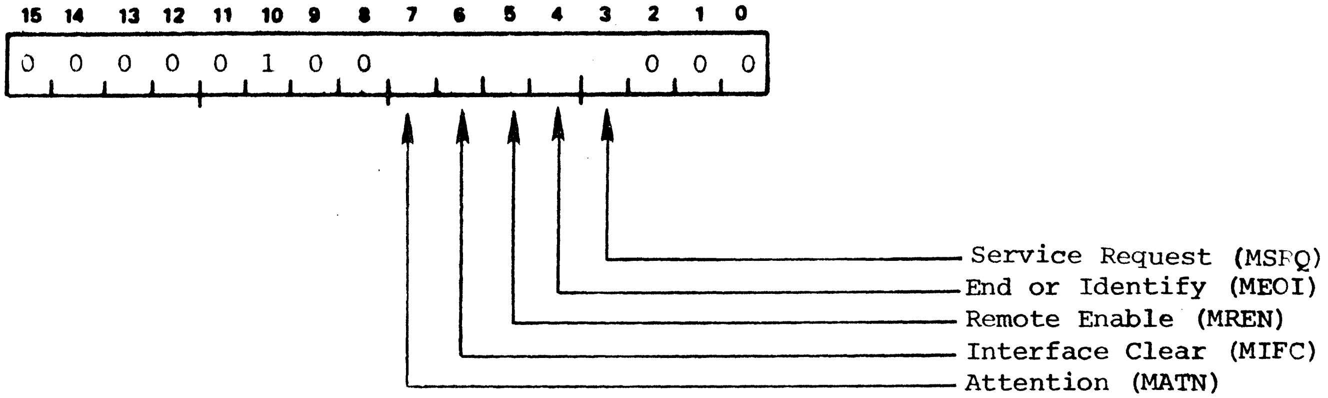


Figure 7-3. IEC Set Mode Command Word Format

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

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

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
CEBTSK	DATA INTP: TASK ADDRESS	7
	DATA 8180 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
CJTBL	DATA IECR:	15
	DATA IECW:	16
	DATA 0	17
	DATA IECF:	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)	26
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 EOF:--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

7.2 A/D, D/A HANDLER PACKAGE (ADAHP)

The Analog-Digital-Analog Handler Package (ADAHP) consists of six functional subroutines which perform typical analog-digital (A/D) and digital-analog (D/A) conversions. The input subroutines allow the following capabilities: single input utilizing internal and external trigger modes; repetitive inputs from the same channel with external and internal clock triggers; and multiple inputs from sequential channels with external and internal clock triggers. The output subroutines include single channel output, and dual channel outputs with Z-pulse (intensification) on odd numbered channels.

The ADAHP is atypical in that each subroutine is called directly by a task rather than called by IO:. Each subroutine locks the other out until it has finished. The general policy of adding 'pauses' between subroutine calls is needed to relinquish control to other tasks by the scheduler.

The supplied tape contains two separate files. The first file contains the six main A/D/A subroutines plus several service routines. The second file contains dummy RTX routines (SUBX:,SUBR:,DELAY:). This second file must not be linked when creating an RTX system. The externally referenced RTX subroutines will be loaded from the RTX library.

Refer to the Analog-Digital-Analog Handler Package User's Manual, 93325-00A0, for a detailed description.

PART III

COMMUNICATIONS EXECUTIVE (COMX)

SECTION 1

INTRODUCTION TO COMX

COMX is a communications executive that is a subsystem of IOX that operates under RTX/IOX control. The purpose of COMX is to facilitate the input and output of communications information be providing the user with device and protocol independence when he writes his programs. (A thorough understanding of RTX and IOX is assumed.)

Data communications requires far more flexibility than ordinary input/output. No universal driver is possible because of the variety of available terminals, protocols and interface devices and their possible combinations. COMX's two-level driver structure provides the needed flexibility. The Protocol Driver maintains the prescribed line protocol and links the computer or terminal with the Device Driver. In turn, the Device Driver controls the hardware interface with the communications channel. COMX can maintain multiple protocol and device drivers and can link them as required, several protocol drivers to a device driver or vice versa, not necessarily one to one.

COMX provides a standard Protocol Driver that implements the IBM Binary Synchronous Communications protocol (BiSync) with a standard Device Driver that drives the Synchronous Modem Controller (SMC). However, COMX also allows the user to write his own drivers for non-standard protocols and devices. The BiSync protocol interfaces with the user's program in a manner compatible with the IOX device-independent specifications.

Although numerous character code sets are used in various protocols, COMX provides translation to allow the application program to deal solely with ASCII. The user may even provide his own character code set and a table to be used in translation. The optional transparent mode permits transmission of data in any bit configuration.

1.1 HARDWARE

The hardware required for COMX implementation includes:

LSI-2 or LSI-3/05 Series computer with a minimum of 8K words of memory to run COMX, 16K for development

The standard interface devices that are supported are:

Bell 201C-compatible modem
Synchronous Modem Controller 53513 or 53617

Other interface devices are supported on an optional basis, providing that the user writes his own drivers to operate under COMX control.

1.2 SOFTWARE

The required development software includes:

LSI-2 or LSI-3/05 OS, DOS, FDOS
MACRO2 or MACRO3 assembler
RTX/IOX/COMX object library
    Tape No. 93300-30 and -31 for LSI-2
    Tape No. 93301-30 and -31 for LSI-3/05
COMX table(s) (provided by user)

1.3 PROTOCOL DRIVERS

When a programmer needs to write his own Protocol Driver for a non-standard protocol, COMX provides the facilities for :

- Causing a line connection to be made - or waiting for an incoming call.
- Ignoring/reading input data until a prescribed character, or one of a group of characters, is received.
- Transmitting or receiving data and control information until a special character, or one of a group of characters, is recognized.
- Requesting translation of all transmitted or received data from EBCDIC to ASCII or the reverse, according to a COMX-supplied table, or to another character code set, according to a user-supplied table.
- Analyzing, reporting and handling a line error status. (When an error occurs, the Protocol Driver is notified which, in turn, notifies the user's program.)
- Disconnecting the communications line.
- Block check calculation and verification with any given polynomial.

1.4 CIRCUIT CONFIGURATIONS

COMX allows for control of virtually any line configuration that conforms to a current-loop or an RS232-standard convention. Device drivers are capable of driving half-duplex, full-duplex or simplex modems. Full-duplex operation on the Distributed I/O System is supported by connecting two Intelligent Cables in simplex mode, one for input and one for output.

1.5 SYSTEM CONFIGURATIONS

Protocol Drivers and Device Drivers are designed to accommodate more than one line of the same type. Hence, the BiSync standard Protocol Driver can be used to drive as many BiSync lines as a given user needs. At System Generation time, the user must specify the connections of drivers to lines and devices. Appropriate tables are constructed during System Generation to produce these connections. Figure 1-1 gives the general flow of putting together a program using COMX.

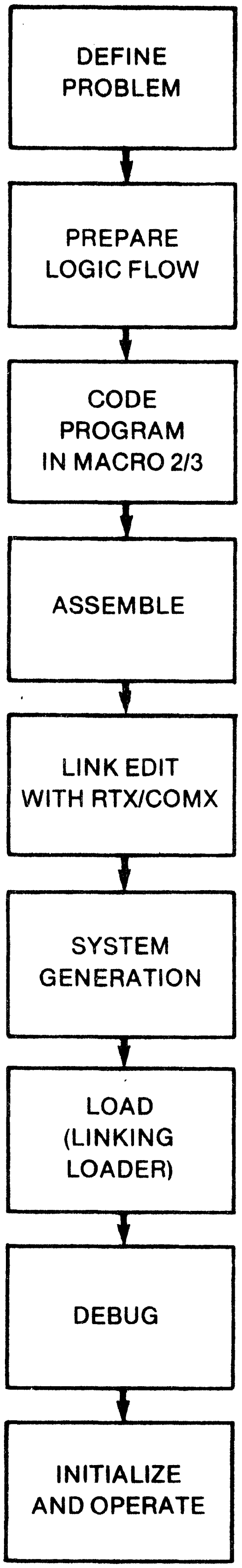


Figure 1-1. Flow of Preparing a Program Using COMX

1.6 RELATED DOCUMENTS

The following documents provide related information:

LSI-2 Series Minicomputer Handbook	Pub. No. 91-20400-00
LSI-3/05 Software Manual	Pub. No. 90-20010-00
Synchronous Modem Controller Interface Description	Pub. No. 91-20363-00
General Information - Binary Synchronous Communications	IBM Pub. No. GA27-3004
OS User's Manual	Pub. No. 96530-00
OS Assembler Language (MACRO2/3) Reference Manual	Pub. No. 96552-00

SECTION 2

CONCEPTS OF DATA COMMUNICATIONS

The purpose of any data communications system is to transfer information from point to point. Data communications permits data processing functions to be extended to remote locations so that data processing facilities can be used with greater efficiency. Also, data communications can sometimes substitute for remote data processing equipment.

A data communications system ordinarily consists of at least a sender, a receiver, a transmission medium and optional interface equipment (see Figure 2-1). The data is sometimes processed, refined, recalculated or recorded, either in transit or entering the network.

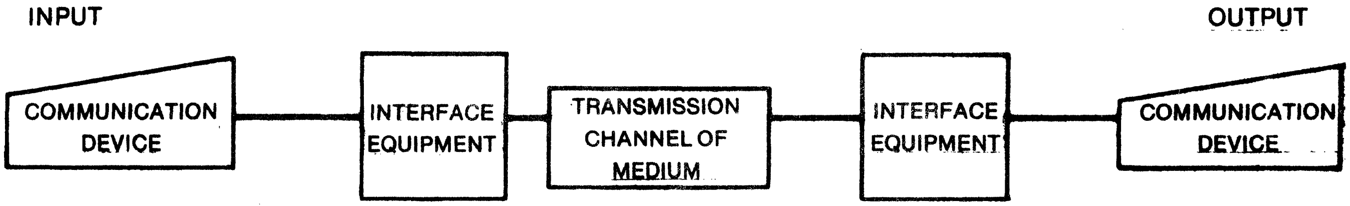


Figure 2-1. Diagram of Data Communications

Input/output devices typically are teletype keyboard, CRT, paper tape, magnetic tape, card, etc. COMX, as a subsystem of RTX/IOX, supports all of these.

Interface equipment is most commonly a modem. The term "modem" is a contraction of "modulator-demodulator." The modem controls the interface between a telephone line, or any other form of communications line (analog), and a terminal or computer (digital) and does the necessary analog/digital conversion.

Messages are transmitted via a communications channel which most frequently is the public message network (telephone). The channel can also be a private line, a cable, radio, microwave, or even just a pair of wires.

2.1 DATA TRANSMISSION

The transmission of data can be synchronous or asynchronous. In the former, resynchronization of data on the line is done in intervals of blocks or frames. No start or stop bits are transmitted to sychronize individual characters. Incoming data bits are counted to form characters; and characters must be consecutive or an idle pattern must be transmitted to fill the gaps.

In asynchronous transmission, resynchronization is accomplished for each character by using the character start and stop bits. There may be blank periods between the transmission of individual characters in a message.

Communications devices and interfaces are either asynchronous or synchronous. Usually, devices that operate at slower speeds are asynchronous. Typical applications are message switching and interactive time sharing systems. High speed transmission (1200 baud or higher) usually is synchronous. High performance communications use synchronous transmission to help cut overhead to a minimum.

2.2 ERROR CHECKING

Most protocols perform periodic error checking of the data received to ensure communications integrity. When the ASCII code set is used, an odd-parity check is performed on each character. Another form of checking is by a block check character (BCC), a form of longitudinal or "column" parity checking. The total bit count is accumulated at both the sending and receiving stations during transmission of a block. The transmitted BCC is compared with the accumulated BCC at the receiving station for an equal condition. In another form of block checking, the BCC is derived from the numeric binary value of the message divided by a constant. The quotient is discarded; and the remainder is used as the BCC. The receiving station checks its calculated remainder with the transmitted remainder for an equal condition. A common BCC is the 16-bit polynomial Cyclic Redundancy Check (CRC).

2.3 CIRCUIT CONFIGURATIONS

The three basic circuit configurations are simplex, half duplex and full duplex. Simplex is the simplest. The circuit is arranged to transmit in one direction only. In a half-duplex configuration, circuits are arranged to transmit in either direction, one way at a time. A full-duplex circuit transmits in both directions simultaneously. Both duplex configurations are conversational.

A channel links two or more communications stations and may encompass multiple circuits. However, in writing the application program, the user deals with the channel rather than the individual circuits.

2.4 SYSTEM CONFIGURATIONS

A variety of system configurations are possible. The two key factors in defining a system are:

If the system is multipoint or point-to-point
If the system is switched or nonswitched.

A multipoint system has a central station and multiple tributary stations. Normally, this type of system has nonswitched, permanent data links. (A data link is everything required to connect two facilities, including interface equipment and transmission channel.) The central station selects and polls the tributaries.

A point-to-point system consists of a communications facility between only two stations at a time. With a switched system, there may be multiple stations; but only two can be linked at any one time. With a nonswitched system, only two stations are involved.

In a switched system, the data link may be freshly established for each transmission and may be broken between transmissions. Normally, the data link in a switched system is established by manual or computer-controlled dialing procedures using the telephone as the communications device.

Nonswitched systems ahve leased or permanently connected communications lines that provide a permanent connection between stations.

2.5 PROTOCOLS

Communications protocols are analogous to the protocol used in telephoning. On a phone, the idea you wish to communicate ususally is preceded by a "start-of-message" phrase such as, "Hello, is Jane there? This is Ed calling." The person you are talking with may respond, "Hi. No, Jane is not in. Do you want to leave a message?" or, Hi. This is Jane." A phone conversation is ended with, "Goodbye, Jane," and, "Goodbye, Ed."

In a similar manner, a communications protocol may preface data being transmitted with a start-of-text (STX) character and terminate the data with an end-of-text (ETX) character. The protocol may acknowledge the receipt of data with an acknowledge (ACK) character. Also, a protocol can provide header information for addressee/sender indentification, message numbering and additional control characters.

2.6 TRANSMISSION CODES

A transmission code set consists of graphic characters (numeric, alphabetic and special), functional characters and data link control characters. Graphic characters are those that can be printed or displayed. Each code provides different capacities for total graphics and functional assignments. The two most commonly used codes are:[1]

EBCDEC	(Extended Binary Coded Decimal Interchange Code) - 256 assignment positions
ASCII	(American Standard Code for Information Interchange) - 128 assignment positions

When either of these code sets is used with the transparent transmission mode, the flexibility of the communications system is further increased because all possible bit configurations are treated as "data only" within transparent text. Character parity checking also is available when transmitting ASCII-coded data.

---

[1] Code set formats are given in Appendix B.

SECTION 3

CONCEPTS OF COMX

The basics of COMX are simple. The user writes his program to input and output data using calls to the protocol driver via the standard RTX input/output block. COMX manages the communication with the Protocol Driver, which formats the data for transmission and provides acknowledgement of received data. The Protocol Driver then calls the Device Driver, via RTX/IOX, to transmit the data to the line device (the modem controller) in a form acceptable to the transmission channel, translating the transmission code first, if necessary. Figure 3-1 diagrams the interrelationships between RTX/IOX, COMX and the Protocol and Device Drivers.

It is quite common for an installation to use a single protocol and Protocol Driver for all devices. However, COMX is set up with the flexibility to link Protocol Drivers and Device Drivers in any configuration required - one to one or several to one.

3.1 PROTOCOLS

A commonly used protocol is IBM's Binary Synchronous Communications protocol. This is the standard protocol supported by COMX and is referred to as "BiSync" in this document. Other protocols may be used with COMX: however, the user will have to provide the Protocol Drivers for them.

3.2 TRANSMISSION CODES

With COMX, the programmer usually handles data in ASCII format. When necessary, the Device Driver then translates the data to EBCDIC for transmission. EBCDIC is the code most commonly used with IBM and IBM-compatible communications devices.

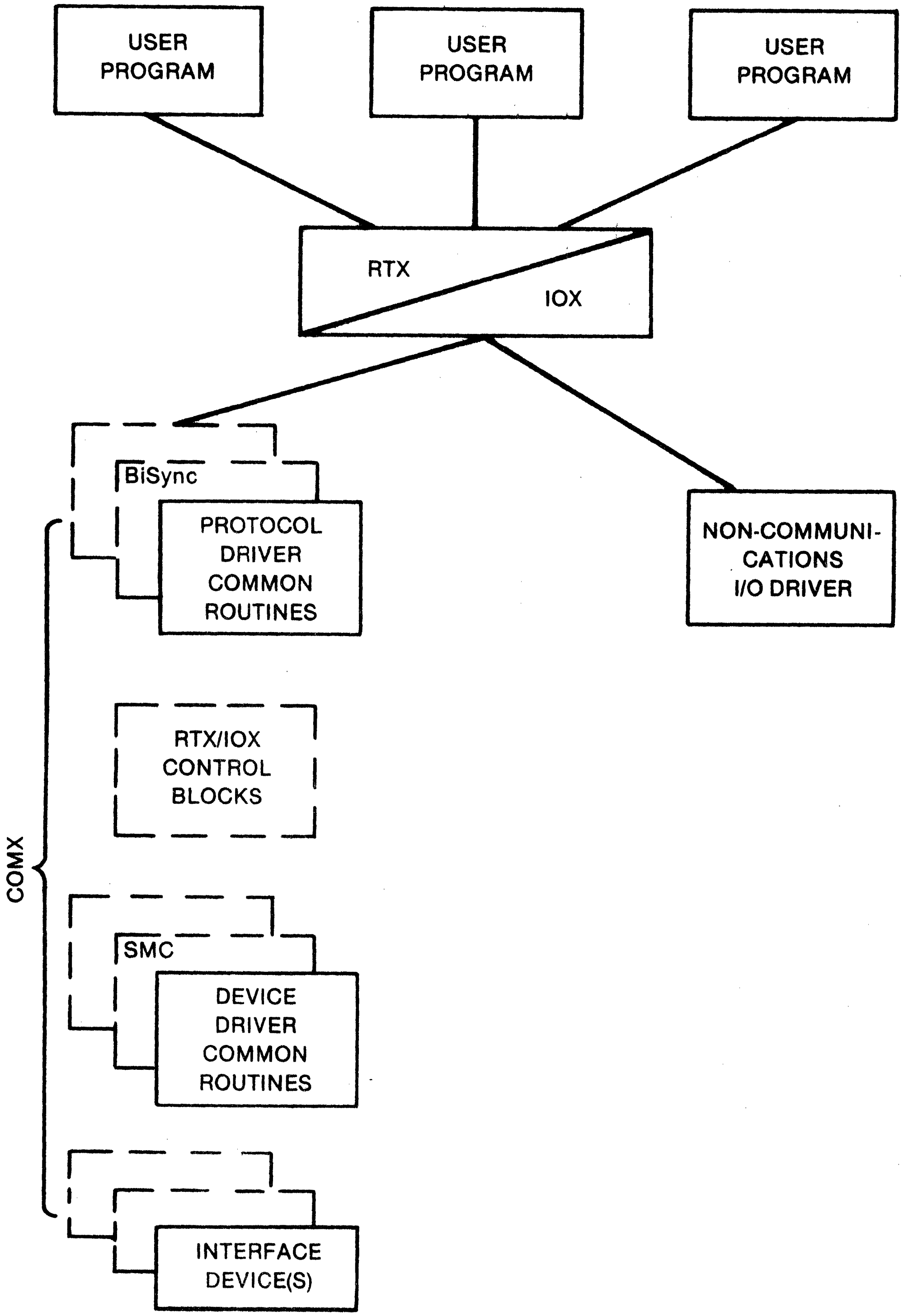


Figure 3-1. Diagram of COMX

SECTION 4

PROTOCOL DRIVERS

A COMX Protocol Driver is a special kind of IOX driver that is designed to control the transmission of data along communications lines. It uses IOX for its control so that it may be entered by a standard user call to IO:. It uses some of the normal IOX mechanisms and services, but also relies on many of its own. The Protocol Driver controls the line by issuing requests to the associated Device Driver. The Device Driver performs most of the work, as directed by the Protocol Driver's requests.

The associations between application program, Protocol Driver, Device Driver and device address are made through the use of Logical Units (LUNs) and Device Information Blocks (DIBs). The Unit Assignment Table (UAT) makes the associations. Each Protocol Driver is capable of supporting many lines by the use of Controller Information Blocks (CIBs) and (DIBs). An application program can be written to work on any Protocol Driver; and a Protocol Driver can work on any Device Driver.

4.1 CLASSES

I/O function requests modify the control performed by the Protocol Driver. Options can be turned on or off and line control can be modified. The exact method in which a Protocol Driver handles requests depends on the line control class: demand or continuous. A Demand Control Class driver acts only on demand and needs no attention between requests. However, a Continuous Control Class driver is in operation continuously once it has been started, even when it has no specific request to be handled.

Demand Control Class

In the Demand Control class, reading and writing are done only when the application program directs. Only simple line protocols can be implemented within this class without placing part of the burden on the application program. A Protocol Driver in this class words like a standard IOX driver except for the method of issuing requests for I/O operations. The CIB-DIB (Controller Information Block - Device Information Block) interlock works like that of the standard IOX. When a request is issued and the CIB is available, the driver is entered with the address of the CIB. The Protocol Driver then issues its I/O requests by calling a Protocol Driver Service Subroutine that issues I/O requests to the Device Drivers. The Protocol Driver receives control back from the subroutine when the I/O operation is complete. After completing all of its processing, the Protocol Driver can reissue the I/O request to process the next block of data by calling the appropriate service routine, or it can exit to EOR: (IOX End-of-Request routine) to complete the user's request and release the CIB-DIB interlock. The service routines are described later in this section.

Continuous Control Class

In the Continuous Control class, transmissions over the line take place independent of application program requests. Once communication has begun, the Protocol Driver is continuously performing I/O in order to retain control of the line. This class is used whenever the line protocol requires that each station stay synchronized with the control and/or data flow across the line.

A Protocol Driver can be written to use multiple DIBs per CIB (e.g., an IBM 2780 terminal with two stations-a CRT and a printer) or only one DIB per CIB (e.g., point-to-point circuit). Figure 4-1 diagrams the former case.

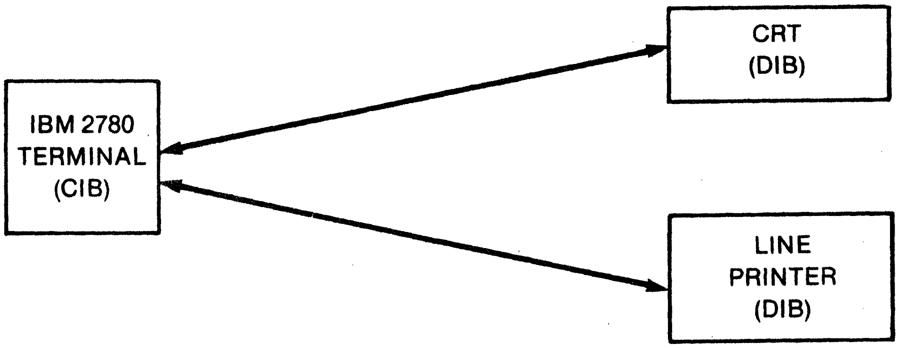


Figure 4-1. Example of Multiple DIB's per CIB

In the case of multiple DIBs per CIB, the Protocol Driver can poll by following the chain of DIBs, polling all that have a READ request pending at the top of the queue. Then the Protocol Driver can issue a selection sequence rather than a poll. Subroutine CPLCRQ (Protocol Driver Services[1]) can be used to test the queue for these requests. If an acknowledgement to selection or positive response to poll is received, then the Protocol Driver may allow IOX to schedule the pending READ or WRITE request via the normal IOX mechanisms to initiate this action. This subroutine (CPLCRQ) is the main interface with the IOX request scheduling mechanism. The CIB-DIB interlock must be kept busy during normal control operations, such as polling, so that the Protocol Driver, rather than IOX, can select the DIB/request to process next.

After determining that a particular pending request is to be processed, the Protocol Driver can use the service subroutine CP:EXR to set up the interlocks in the proper order to ensure operation. When an operation is complete, the CP:EOR routine can be called to complete the request.

In the case of a single DIB per CIB, the complexity is reduced, but the rules stay the same. In other words, the Protocol Driver maintains control of the line until time to transmit or receive data. The queue then is checked for the appropriate request at top of queue. If found, IOX is allowed to schedule the request as above. The request termination procedure also is the same.

Some protocols (e.g., BiSync) require constant attention on the line looking for input, but are not really polling. If a WRITE request is issued by an application during this period of time, the Protocol Driver can begin immediately to perform the request, assuming that data is not currently being received. To allow the immediate processing of the WRITE request from IOX, the CIB-DIB interlock should be left unlocked either until the line begins to receive data or until a user request is received.

---

[1] Subsection 4.3

4.2 BISYNC PROTOCOL DRIVER

The BiSync protocol[1] was designed by IBM to provide reliable half-duplex communications with high-speed, synchronous modems. The BiSync Protocol Driver implements the following features of the protocol:

- Insertion and deletion of start-of-text (STX), end-of-text(ETX), end-of-block (ETB) and/or end-of-intermediate-block (ITB) characters around the message.
- Line sequencing through enquiry(ENQ), acknowledge (ACK) and end-of-transmission (EDT) characters.
- Acknowledgement of receipt of messages.
- Cyclic redundancy error checking on text data.
- Timeouts on line failures.
- Error recovery via retransmission (retries).
- A choice of character sets (ASCII or EBCDIC).
- An optional transparent mode of operation in which every possible bit configuration is valid data.
- User line control.

With the BiSync driver, the station may lock the communications line while transmitting. In Normal Line Control mode (default), the Protocol Driver will lock the line into WRITE mode before transmitting user data during a WRITE request. The line is then unlocked to other requests after transmission. (This should not be confused with the CIB-DIB interlock.) In User Line Control mode, the line is not unlocked after a WRITE operation. A sequence of WRITE requests can be issued, the first one locking the line into WRITE mode and the last one (identified by the user) unlocking the line. A WRITE request issued when the line is already locked by the other station is an error and is flagged accordingly.

No general restrictions are imposed by the BiSync protocol on the length of data transmissions. However, a maximum length of 400 bytes is frequently specified by many commercially available terminals. As the byte count increases beyond this value, block checking is increasingly likely to miss errors, particularly when the EBCDIC code set is used.

Requests issued to the BiSync Protocol Driver are initiated one at a time in priority order (by task priority). No simultaneous processing of requests is performed.

The BiSync protocol is called and its options are selected by the BSC macro which is defined in a later section.[2]

---

[1] For detailed information on this protocol, refer to General Information - Binary
[2] Synchronous Communications, IBM Pub. No. GA27-3004.
Section 6.2.

4.3 PROTOCOL DRIVER SERVICES

Several service routines exist to aid in the writing of Protocol Drivers. These are available for all Protocol Drivers.

The service routines have a great deal in common; concepts, calling and return sequences are quite similar. The following conventions are common to all services:

1. All services are called with the appropriate CIB address in the X register, unless otherwise noted. At return, the X register contains the CIB address.

2. All I/O-oriented requests have an error return and a normal return. The error return is in the next location beyond the call. The error status (Protocol Driver IOB word 5) is returned in the A register. The normal return is in the next location past the error return.

3. For the several requests that allow for the checking of special characters by table comparisons:

   a. A sequence is defined as a single character or two characters (ordered) denoting a special meaning in the protocol.

   b. Sequence tables are used for checking data. A sequence table consists of contiguous one-word entries. Each word contains a one- or two-character sequence (left-justified if one character).

   c. If the table address is zero, no checking is performed.

Requests are in the form of subroutine calls or macros. Usually, the macros merely create the specific subroutine calls.

CP:SIO (Start I/O)

The CP:SIO subroutine performs an operation using the IOB in words 1-10 of the CIB.[1] The logical unit for the I/O operation is obtained from DIB word DLUN, using the DIB whose address is in CIB word CPDIB. CP:SIO will return to the IOX scheduler SCH:, if necessary (CBOR = 0).

Calling sequence:

```
EXTR      CP:SIO
  .
  .
LDX       CIB Address
JST       CP:SIO
. . .             ERROR RETURN
. . .             NORMAL RETURN
```

Returns with:

INTERRUPTS - ENABLED
STATUS - UNCHANGED
A REGISTER - STATUS, IF ERROR
X REGISTER - UNCHANGED

---

[1] Subsection 4.5

CP:XIO (Execute I/O)

The CP:XIO subroutine performs an I/O operation using the IOB in words 1-10 of the CIB. The IOB is set up as requested in the calling sequence.

In Calling Sequence 1, a control sequence is written from a protocol table. The OPCODE must specify a WRITE operation. The TABLE NUMBER parameter specifies which table in the Protocol Master Table list is used; and OFFSET specifies the word in that table that contains the sequence to be transmitted.

In Calling Sequence 2, any I/O operation can be performed. The OFFSET parameter specifies the table to be used for stop-character checking; a zero requests no checking. OFFSET is the table position in the Protocol Master Table list. BUFFER ADDRESS and BYTE COUNT can be specified or may be zero, in which case this information is taken from the caller's IOB (found in CBUFF and CRCNT). The BUFFER ADDRESS is a byte address. Symbolic opcode definitions can be found in the RTX/IOX/COMX MACROS file.[1]

```
     Calling sequence 1:                      or Calling Sequence 2:

          EXTR       CP:XIO                        EXTR       CP:X10
            .                                        .
            .                                        .
          LDX        CIB Address                   LDX        CIB Address
          LAP        OFFSET                        LAP        OFFSET
          JST        CP:XIO                        JST        CP:XIO
          DATA       IOCODE                        DATA       OPCODE
          DATA       TABLE NUMBER                  DATA       0
                                                   BAC        BUFFER ADDRESS
                                                   DATA       BYTE COUNT

     followed by:
          . . .      ERROR RETURN
          . . .      NORMAL RETURN

     Returns with:

          INTERRUPTS - ENABLED
          A REGISTER - TABLE POSITION, IF NORMAL; STATUS, IF ERROR
          X REGISTER - UNCHANGED
```

CPREAD (Protocol Driver READ)

This macro produces a call to CP:XIO to perform a READ operation. The OFFSET parameter specifies a stop-sequence table by position within the Protocol Master Table list; a zero denotes no stop-character checking; a -1 denotes that the program already has set up the A register to call CP:XIO.

The various options specify the opcode, buffer address and byte count symbolically. The options may be specified in any meaningful combination. Input to the caller's buffer is allowed; but no other buffer can be provided via this macro in which to store incoming data. Figure 4-2 shows examples.

---

[1] Section 6.2 and, in Part II of this manual, Section 3.3

Even though XLAT, ACHK or CCHK may be specified in the macro call, the protocol line DIB may not contain specifications for their processing. If the DIB contains a null value (zero) for one of these, no processing will be done, despite any request in the calling macro. No error will be indicated in such cases.

If the Protocol Driver specifies a character timer in the IOB (in clock ticks), data must be received within that time interval. An error will be indicated if data does not appear or continue to appear within that time.

Calling sequence:

```
CPREAD    OFFSET[,options[, . .]]
```

where options are:

DELT	Delete all incoming data until first unspecified character.
TEXT	Data is text; caller's buffer (from CBUFF and CRCNT) will be used to store data.
ACHK	Accumulate checksum (BCC) on incoming data.
XLAT	Translate incoming data prior to storing in buffer.
CCHK	Treat incoming data as checksum (BCC) and check it for validity.
INIT	READ initialization (i.e., start of new transmission).
CHK	The next data character or sequence received must be a stop character. Otherwise, error status is indicated and the received character(s) is returned in IOB word CBUFF.

This macro returns in the same way as CP:XIO.

```
CPREAD    STXTBL,INIT
```

will check all received characters until a character or sequence specified in the STXTBL table is found. Return will occur when a proper character or sequence is found or when an error occurs. At normal completion, the table position of the received character is returned in the A register.

```
CPREAD    ETXTBL,TEXT,ACHK,XLAT
```

will READ, translate and accumulate checksum on each character received and store each in the caller's buffer (CIB word CBUFF). The request will terminate normally upon receipt of a character or sequence contained in ETXTBL. Abnormal termination occurs upon expiration of the caller's byte count (CIB word CRCNT) or an error.

Figure 4-2. Examples of CPREAD

CPWRIT (Protocol Driver WRITE)

This macro produces a call to CP:XIO to perform a WRITE operation. Data to be written can originate from the caller's buffer (CIB words CBUFF and CRCNT) or from a protocol sequence table.

The OFFSET parameter serves two functions in this macro call. It specifies the table number in the Protocol Master Table list to be used for stop-sequence checking while writing from the caller's buffer. Also, it can specify the sequence to be transmitted when the data comes from a protocol sequence table. When in the latter mode, protocol sequence table 1 is used arbitrarily (i.e., the table addressed at one word into the Protocol Master Table list is the sequence table from which control sequences may be transmitted).

When the OFFSET parameter is used to specify a stop-sequence-checking table, the parameter may be:

0 = no stop checking
-1 = the A register already has been set up by the program to call CP:XIO
+n = a positive value giving the stop-sequence-checking table to be used

The various options specify opcode, buffer address and byte count for the CP:XIO call. If an option is specified in the macro call that is not requested in a protocol line DIB, processing will not occur and no error will be flagged. Examples of CPWRIT are shown in Figure 4-3.

Calling sequence:

```
CPWRIT    OFFSET[,option[, . . .]]
```

where options are:

TEXT	Data is text; caller's buffer (from CBUFF and CRCNT) contains the data to be transmitted.
XLAT	Translate data prior to transmitting.
ACHK	Accumulate checksum (BCC) on outgoing data. When used in a request with ENDS or INTS, this option transmits the accumulated checksum after the requested data.
WCS	Write control sequence or control character specified as OFFSET into table 1 of the Protocol Master Table list.
INTS	Intermediate stop specifies that a stop sequence is in process without a line turnaround. If ACHK is specified, the accumulated checksum is transmitted after completion of the requested WRITE operation.
HDR	Write the header, if one is contained in the DIB, ahead of the given data.
ENDS	Perform an ending stop sequence with a line turnaround. After completion of the WRITE operation, the appropriate pad characters are transmitted. A line turnaround occurs if required by the line being used. If ACHK is specified, the accumulated checksum is transmitted prior to pad character transmission.

This macro returns in the same way as CP:XIO.

```
CPWRIT    STXSEQ,WCS,HDR
```

causes the control sequence addressed at STXSEQ in the first table in the Protocol Master Table list to be written out with a header preceding it.

```
CPWRIT    SPEC,TEXT,ACHK,XLAT
```

causes the caller's buffer (CIB word CBUFF) to be translated and written to the line. Checksum is accumulated dynamically. If any characters or sequences in the data correspond to values in the table of special characters SPEC, the request is completed. This usually means that an error in the user's data has occurred.

```
CPWRIT    ETX,WCS,ACHK,ENDS
```

causes the control sequence addressed as ETX to be written over the line, accumulated into the checksum and followed by the proper checksum and pad character sequence. If the protocol or line is half-duplex, a line turnaround takes place after transmission is complete.

Figure 4-3. Examples of CPWRIT

CPFUNC (Protocol Driver FUNCTION)

This macro produces a call to CP:XIO to perform a FUNCTION request. The parameters available for each function are described later in this section.[1]

Calling sequence:

```
CPFUNC    FUNCTION[,parameter[, . . .]]
```

where the functions are:

LNP	Set Line Parameters
SPC	Set Special Character
SYN	Set Sync Character
INT	Initialize Line
ENL	Enable Line
DSL	Disable Line
RST	Read Status
RON	Raise and/or Wait for Data Set Lines On
LOF	Lower and /or Wait for Data Set Lines Off
RVC	Reverse Channel
BRK	Issue Break
NSY	Issue New Sync
RSY	Reestablish Synchronization
SDW	Set Device-Dependent Word 1
RDW	Reset Device-Dependent Word 1

This macro returns in the same way as CP:XIO.

---

[1] Subsection 4.5, FUNCTION Requests

CP:INT (Protocol Request Initialization)

This subroutine initializes variables upon entry into the driver as necessary to perform a Protocol Driver I/O request. It also raises the task priority to CPDPRI (:8178), the standard Protocol Driver priority.

Calling sequence:

```
                EXTR      CP:INT
                  .
                  .
     label      JST       CP:INT     entry from I/O Scheduler with I/O request
                                     to process
```

Returns with:

```
     INTERRUPTS - ENABLED
     A REGISTER - UNDEFINED
     X REGISTER - CIB ADDRESS
```

CP:EXR (Execute Request)

This subroutine causes the appropriate IOX scheduler mechanisms to initiate the I/O request at the top of the queue linked from the specified DIB. Return is not immediate. Upon return, the IOB is in process and the CIB-DIB interlock is set as requested.

Calling sequence:

```
     EXTR      CP:EXR
       .
       .
     LDX       CIB Address    X REGISTER CONTAINS POINTER TO CIB
     LDA       DIB Address    A REGISTER CONTAINS POINTER TO DIB
     JST       CP:EXR
```

Returns with:

```
     INTERRUPTS - ENABLED
     STATUS - UNCHANGED
     A REGISTER - UNDEFINED
     X REGISTER - UNCHANGED
     CIB-DIB INTERLOCK SET AS REQUESTED
```

CP:CRQ (Check Request Queue)

This subroutine checks the IOB queue of pending requests in the DIB. If the top entry has the opcode specified, the FOUND return is taken; otherwise, the NOT-FOUND return is taken. Upon return, the opcode and modifier fields from the IOB are left in the A register in the same format as in the IOB.

Calling sequence:

```
EXTR      CP:CRQ
  .
  .
LDX       DIB Address     X REGISTER CONTAINS POINTER TO DIB
LAP       OPCODE
JST       CP:CRQ
. . .                     NOT-FOUND RETURN
. . .                     FOUND RETURN
```

Returns with:

```
INTERRUPTS - ENABLED
STATUS - UNCHANGED
A REGISTER - IOB OPCODE, MODIFIER FIELDS
X REGISTER - UNCHANGED
```

CP:EOR (End of Request)

This subroutine terminates a user's I/O request by BEGINning the IOX EOR: routine. Control then is returned with the CIB-DIB interlock left as it was upon entry.

Calling sequence:

```
EXTR      CP:EOR
  .
  .
LDX       CIB Address     X REGISTER CONTAINS POINTER TO CIB
JST       CP:EOR
```

Returns with:

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

CP:ERI (End of Request, Return to Scheduler)

This subroutine terminates a user's request by BEGINning the IOX EOR: routine, then returns to the I/O Scheduler through IORTN:.

Calling sequence:

```
EXTR      CP:ERI
  .
  .
LDX       CIB Address     X REGISTER CONTAINS POINTER TO CIB
JST       CP:ERI
```

Does not return.

4.4 INPUT/OUTPUT REQUESTS

The user's requests for I/O operations are processed by the Protocol Driver. The Protocol Driver in turn uses IO: and its own IOB to issue requests to the Device Driver. The three types of I/O requests are READ, WRITE and FUNCTION. The macros in the previous subsection (CPREAD, CPWRIT and CPFUNC) can be used to initialize some of the words described below.

Operation Code and Status Word

Word 5 of the IOB defines the operation requested and, upon return, the status of that request. The format of the word is:

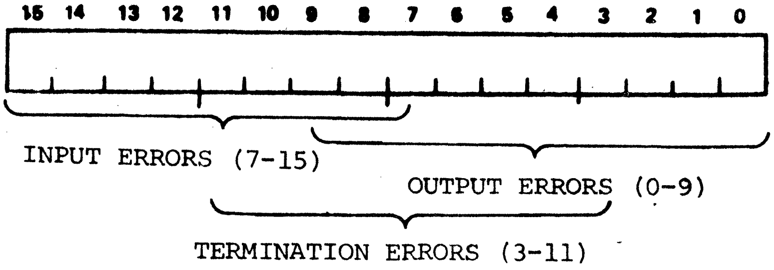
Operation Word

Bits	Definition
0-1	Operation code modifier READ mod 00 = read characters READ mod 01 = delete characters READ mod 10 = checksum verify WRITE mod 01 = write characters
2-3	Operation code 00 = READ 01 = WRITE 10 = FUNCTION
5	Character control 0 = store characters 1 = discard characters
6	Message type 0 = control message 1 = text message
8-9	Operation code transfer modifier 00 = no modification READ mod 01 = initial transfer WRITE mod 01 = intermediate transfer WRITE mod 11 = ending transfer
10	Translate
11	Accumulate checksum
12	Header requested
13	Next character must be a stop character

Status Word

Bits	Definition
14	1 = error
10-13	Error code if bit 14 is set 0 = line error (see word CLSTAT in CIB for detailed status) 1 = device unresponsive 2 = invalid checksum 3 = halt I/O occurred 4 = invalid LUN 5 = request requested 6 = input buffer full 7 = data rejected

CLSTAT - Line Status Word



Bits	Definition
0	Transmit data overrun
3	Transmit framing error
5	Transmit timeout
6	Loss of CTS
7	Loss of DSR
8	Loss of CD
9	Special timeout
10	Receive timeout
12	Receive framing error
14	Parity error
15	Receive data overrun

READ Requests

A READ request is made with an IOB designating:

Requested Function Code - READ (0)
Requested Count - Maximum Byte Count for this READ
Buffer Address - Caller's Buffer Address
Word 9 - Termination Table Address

Operation Code Modifier = 00
The Device Driver processes the READ request by inputting and optionally storing characters in the user's buffer, until the requested byte count is satisfied or a character or sequence in the termination table is detected. In the latter case, the termination character may be stored in the user's buffer; its displacement in the termination table is returned in the A register. The actual transfer count contains the number of characters stored in the user's buffer. Characters input but not stored are not counted.

Operation Code Modifier = 01
The Device Driver processes the READ request by deleting characters from the Device Driver input buffer until a character or sequence in the termination table is not detected. This could be used to strip leading blanks or trailing pad characters. The actual transfer count contains the number of characters deleted.

Operation Code Modifier = 10
The Device Driver processes the READ request by reading the block check character(s) and verifying the checksum.

An example of the IOB for a READ is shown in Figure 4-4.

IOB when READ request is made

WORD		DEFINITION
4	400E	Logical Unit Number of device (:4000 + 14)
5	0C40	READ characters and store into given buffer, text message, translate to ASCII, accumulate checksum
6	0100	Store up to 256 characters
7	2000	Byte address of user's buffer; buffer starts at :1000
8	0800	Byte address of protocol stop-sequence table
9	0000	No protocol modification routine
10	012C	Protocol input character timer of 3 secs (300 ticks)

IOB upon completion of READ request

WORD		DEFINITION
4	400E	Unchanged
5	0040	No errors (bits 10-14)
6	00D3	Last character read was 'S'
7	2000	Unchanged
8	002A	Actual count read was 42 characters (bytes)
9	0000	Unchanged
10	012C	Unchanged

Figure 4-4. Example of I/O READ Request from Protocol Driver to Device Driver

WRITE Requests

A WRITE request is made with an IOB designating:

Requested Function Code - WRITE (0)
Requested Count - Maximum Number of Bytes of Data in Buffer
Buffer Address - Caller's Buffer Address
Word 9 - Termination Table Address

The Device Driver processes the WRITE request by transmitting characters from the caller's buffer or control character table until the byte count is satisfied or a character or sequence in the termination table is detected. In the latter case, the termination character(s) is transmitted or not, depending on the state of a flag bit in the IOB; and the displacement of the termination character in the termination table is returned in the A register.

FUNCTION Requests

A FUNCTION request is made with an IOB designating:

Requested function code - FUNCTION (10)
Operation modifier - 10 (COMX function)
Requested count - COMX function code (Table 4-1)

Information required by the various COMX function codes is stored in the IOB (word 7) as required for each function. The two types of function codes are queued and immediate. An immediate function code is processed when it is detected, despite any other concurrent operation. For example, the function code Halt I/O will be processed while a WRITE operation is in progress. Queued functions are processed as they come to the top of the queue. The function codes are given in Table 4-1.

Parameter Word Definitions

The following functions have parameter words associated with them:

Code 1 - Set Line Parameters

Bits	Definition	
0	Parity enable	1 = enabled
1	Parity select	1 = even
2-3	Word length select (modulo 5)	00 = 5 bits, 11 = 8 bits

Code 2 - Set Special Character

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

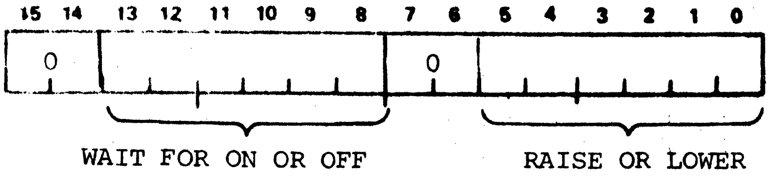
Code 3 - Set Sync Character

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

Table 4-1. Function Codes

CODE	FUNCTION
0	Invalid
1	Set line parameters
2	Set special character
3	Set synch character
4	Initialize line
5	Enable line
6	Disable line
7	Read status
8	Raise and/or wait for data set lines on
9	Lower and/or wait for data set lines off
10	Issue reverse channel
11	Issue break
12	Issue new sync
13	Reestablish line synchronization
14	Set device-dependent word 1
15	Reset device-dependent word 1

Codes 8 and 9 - Raise/On or Lower/Off Data Set Lines



Bits	Definition	Bits	Definition
0	Transmit mode	8	Data set ready
1	Receive mode	9	Clear to send
2	Echo mode	10	Carrier detect
3	Loop-back mode	11	Ring indicator
4	Data terminal ready	12	Signal quality
5	Request to send	13	Secondary channel receive

Code 14 - Set Device-Dependent Word 1

Bits	Definition
15	Transmit modification
14	Transmit transparent mode
13	Receive transparent mode
12-0	Device dependent

4.5 PROTOCOL DRIVER TABLES

The Protocol Driver requires tables that relate to the protocol driving function and tables that relate to the RTX/IOX environment in which it runs. Each line must have a Protocol Driver CIB and a Protocol Driver DIB compatible with the equivalent IOX tables. Each Protocol Driver must have a set of protocol tables. In the master station of a multi-drop line, each terminal should have its own DIB. Details of the tables are given in an appendix.[1]

Controller Information Block (CIB)

The Controller Information Block for the Protocol Driver contains 37 words, most of which are standard for IOX. The words that have been changed or added for the COMX Protocol Driver are:

WORD	CONTENTS
20	Protocol Driver Flags
21	Protocol Master Table Address
31	DIB Word 8- Protocol Flags
35	Temporary Storage
36	Temporary Storage
37	Length of Protocol CIB - CPST2+1

---

[1] Appendix C

The flags in Word 20 are defined as:

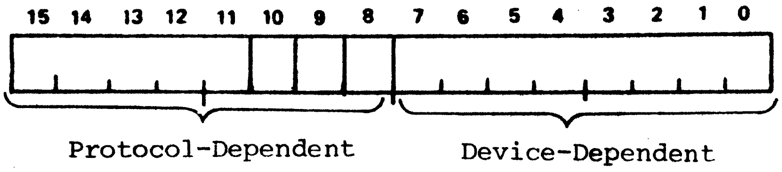
Bits	Definition
13-15	NAK Counter
10-12	Invalid ACK Counter
8-9	Timeout Counter
7	Inter-Block Flag
6	Transparent Mode Flag
5	Acknowledgement Flag
4	WRITE Request Outstanding
3	READ Request Outstanding
0-2	Current Protocol State

Device Information Block (DIB)

The Device Information Block for the Protocol Driver contains 29 words, many of which are standard for IOX. The words that have been changed or added for the COMX Protocol Driver are:

WORD	CONTENTS
7	Input Char Timer Value
8	Protocol Flags
10	Device/Protocol-Dependent Flags
11	Maximum Block Size
12	Line Parameters
13	Input Translate Routine Address
14	Output Translate Routine Address
15	Device LUN
16	Line DIB Thread
17	Subroutine Return Address
18	Temp Cell 1
19	Temp Cell 2
20	Checksum Routine Address
21	Checksum Flags
22	Checksum Polynomial
23	Checksum Accumulation
24	Header Byte Count
25-29	Header Bytes 1-10

The device-dependent and protocol-dependent flags in Word 10 are defined as:



Bits	Definition
11	Disconnect on Write File Mark
10	User Line Control
9	ALL WRITES in Transparent Mode
8	Unformatted WRITES in Transparent Mode
3	Auto-Call this Line
0	Master Station

Protocol Master Table

The Protocol Master Table contains an entry for every protocol table, except DIBs and CIBs, that are required for a given Protocol Driver.

Control Message Table

The Control Message Table is segmented. Each segment consists of a word giving the message length (in words), followed by the message itself. There can be as many segments as the Protocol Driver requires.

Sequence Table

Sequence Tables hold sequences of control characters that are used by some of the Protocol Driver service routines (e.g., CPREAD, CPWRIT). Word 0 of the Sequence Table contains flags and the number of contiguous one-word entries following it in the table. Each entry word contains a left-justified, one-character sequence or a two-character sequence. In the example shown in Figure 4-5, the first two entries have only one character; the remaining one has two.

WORD	BITS 15-8	BITS 7-0	CONTENTS
0	FLAGS	3	Flags and Number of Entries
1	E1C1	0	Entry 1, Character 1
2	E2C1	0	Entry 2, Character 1
3	E3C1	E3C2	Entry 3, Characters 1 & 2

Figure 4-5. Example of a Sequence Table

The flags are defined as:

Bits	Definition
15	Reserved for use by the Device Driver.
14	Error - the first character of a two-character sequence is found, but the second does not immediately follow.
13	Error - the next character is not a character sequence in the table.
12	Error - a character sequence in the table is found.
11	Transfer (store) character sequence found.
10	Accumulate checksum on character sequence found.
9,8	Undefined.

SECTION 5

DEVICE DRIVERS

A COMX Device Driver is a set of routines running as a driver under IOX, designed to transmit and receive data over a communications line. A COMX Device Driver has more capability than an IOX driver. As directed by the Protocol Driver, the Device Driver actually controls most activity on the line.

Certain limitations (device-dependent considerations) exist due to the characteristics of the lines, protocols, controllers, etc. For example, the BiSync protocol cannot be run on an asynchronous device; and an asynchronous modem cannot be run on an SMC (Computer Automation's Synchronous Modem Controller).

5.1 SYNCHRONOUS MODEM CONTROLLER DEVICE DRIVER AND ROUTINES

Computer Automation's Synchronous Modem Controller (SMC) interfaces only with synchronous modems. The SMC device handler, which is the standard device handler in COMX, controls most of the activity on the communications channel, as directed by the Protocol Driver. Like any user-written Device handler, the SMC device handler consists of several device-dependent routines called by the COMX Device Driver. The functions of these routines are described below.

Set Line Parameter

When applicable to the device, this routine presents the device with the given line parameter word.

Called by:

CD:LNP	COMX Device Driver Line Parameter Subroutine

Calling sequence:

```
LDA     line-parameter-word (defined under CD:FLP function¹)
JST     DEVDEP    CALL DEV-DEPENDENT ROUTINE
. . .             ERROR RETURN
. . .             NORMAL RETURN
```

---

[1] Subsection 5.3

Set Sync Character

When applicable to the device, this routine presents the device with the synchronization character from the Line Information Block (LIB).

Called by:

CD:SYN	COMX Device Driver Sync Character Subroutine

Calling sequence:

```
JST       DEVDEP    CALL DEV-DEPENDENT ROUTINE
. . .               ERROR RETURN
. . .               NORMAL RETURN
```

Set Special Character

When applicable to the device, this routine presents the device with the special character from the LIB.

Called by:

CD:SPC	COMX Device Driver Special Character Subroutine

Calling sequence:

```
JST       DEVDEP    CALL DEV-DEPENDENT ROUTINE
. . .               ERROR RETURN
. . .               NORMAL RETURN
```

Initialize Line

This routine initializes and/or resets the device and line, as necessary.

Called by:

CD:INT	COMX Device Driver Initialization Subroutine

Calling sequence:

```
JST       DEVDEP    CALL DEV-DEPENDENT ROUTINE
. . .               ERROR RETURN
. . .               NORMAL RETURN
```

Enable Line

This routine does whatever is necessary to enable the communications line through the device (e.g., raise DTR).

Called by:

CD:ENL   COMX Device Driver Enable Line Subroutine

Calling sequence:

```
JST       DEVDEP    CALL DEV-DEPENDENT ROUTINE
. . .               ERROR RETURN
. . .               NORMAL RETURN
```

Disable Line

This routine does whatever is necessary to disable/disconnect the communications line (e.g., drop DTR).

Called by:

CD:DSL   COMX Device Driver Disable Line Subroutine

Calling sequence:

```
JST       DEVDEP    CALL DEV-DEPENDENT ROUTINE
. . .               ERROR RETURN
. . .               NORMAL RETURN
```

Start Input

This routine does whatever is necessary for the device to begin inputting data (e.g., raise Receive Mode).

Called by:

CD:STI   COMX Device Driver Start Input Subroutine

Calling sequence:

```
    JST       BEGIN:    START A TASK
    DATA      DEVDEP    START INPUT ROUTINE ADDRESS
    DATA      STIPRI    START INPUT PRIORITY
     .
     .
(Error Exit)
    JST       BEGIN:    START A TASK
    DATA      CD:IER    INPUT ERRPR ROUTINE
    DATA      CRPPRI+1  ERROR PRIORITY (from PRIORITY = C2IPRI)
     .
     .
(Normal Exit after EOB Occurs)
    JST       CD:SRP    SCHEDULE RECEIVE PROCEDURE
     .
     .
```

Stop Input

This routine does whatever is necessary for the device to stop inputting data.

Called by:

```
CD:SPI    COMX Device Driver Stop Input Subroutine
```

Calling sequence:

```
JST        DEVDEP    CALL DEV-DEPENDENT ROUTINE
. . .                ERROR RETURN
. . .                NORMAL RETURN
```

Start Output

This routine does whatever is necessary for the device to start outputting data (e.g., raise Transmit Mode and RTS).

Called by:

```
CD:STO    COMX Device Driver Start Output Subroutine
```

Calling sequence:

```
    JST        BEGIN:    START A TASK
    DATA       DEVDEP    START OUTPUT ROUTINE ADDRESS
    DATA       STOPRI    START OUTPUT PRIORITY
      .
      .
(Error Exit)
    JST        BEGIN:    START A TASK
    DATA       CD:OER    OUTPUT ERROR ROUTINE
    DATA       CTLPRI    ERROR PRIORITY
      .
      .
(Normal Exit after EOB Occurs)
    JST        BEGIN:    START A TASK (usually done by INTQ:)
    DATA       CD:DEX    OUTPUT EOB EXIT ROUTINE
    DATA       CINPRI    INTERRUPT HANDLER PRIORITY
      .
      .
```

Stop Output

This routine does whatever is necessary for the device to stop outputting data.

Called by:

```
CD:SPO    COMX Device Driver Stop Output Subroutine
```

Calling sequence:

```
JST      DEVDEP   CALL DEV-DEPENDENT ROUTINE
. . .             ERROR RETURN
. . .             NORMAL RETURN
```

Input Character Processor 1

This routine is called for each character input (READ, Modifier = 00) after the protocol modification routine has been executed and before the stop-character check, checksumming or translation is done.

Called by:

CD:RVP   COMX Device Driver Receive Processor

Calling sequence:

```
LDA      CHAR     (loads into the low-order bits 7-0)
JST      DEVDEP   CALL DEV-DEPENDENT ROUTINE
. . .             ERROR RETURN
. . .             IGNORE RETURN, GET NEXT RETURN
. . .             NORMAL RETURN (current character is updated
                  with character returned in A register)
```

Input Character Processor 2

This routine is called for each character that is input (READ, Modifier = 01) after the protocol modification routine has been executed and before the stop-character check or checksumming has been done.

Called by:

CD:RVP   COMX Device Driver Receive Processor

Calling sequence:

```
LDA      CHAR
JST      DEVDEP   CALL DEV-DEPENDENT ROUTINE
. . .             ERROR RETURN
. . .             IGNORE RETURN, GET NEXT CHAR
. . .             NORMAL RETURN (current character is updated
                  with character returned in the A register)
```

Output Character Processor 1

This routine is called for each character output (WRITE) after the translation and the protocol modification routines have been done, and before stop-character check and checksumming.

Called by:

CD:DTP   COMX Device Driver Data Transmit Procedure

Calling sequence:

```
LDA       CHAR
JST       DEVDEP    CALL DEV-DEPENDENT ROUTINE
. . .               ERROR RETURN
. . .               IGNORE RETURN, GET NEXT CHAR
. . .               NORMAL RETURN (current character updated
                    with character returned to A register)
```

Raises and/or Wait for Data Set Signals On

This routine does whatever is necessary for the device to raise the given signal(s) and/or wait for data set signal(s) to come on, then return.

Called by:

CD:RON COMX Device Driver Raise and/or Wait for Data Set Signal on Subroutine

Calling sequence:

```
LDA       parameter-word (described under the CD:FRN function¹)
JST       DEVDEP    CALL DEV-DEPENDENT ROUTINE
. . .               ERROR RETURN
. . .               NORMAL RETURN
```

Lower and/or Wait for Data Set Signals Off

This routine does whatever is necessary for the device to lower the given signal(s) and/or wait for the data set signal(s) to go off, then return.

Called by:

CD:LOF COMX Device Driver Lower and/or Wait for Data Set Signals Off Subroutine

Calling sequence:

```
LDA       parameter-word (described under CD:LOF function¹)
JST       DEVDEP    CALL DEV-DEPENDENT ROUTINE
. . .               ERROR RETURN
. . .               NORMAL RETURN
```

Resynchronize Line

When applicable, this routine does whatever is necessary for the device to reestablish synchronization.

Called by:

CD:RSY COMX Device Driver Resync Line Subroutine

---

[1] Subsection 5.3

Calling sequence:

```
JST       DEVDEP    CALL DEV-DEPENDENT ROUTINE
. . .               ERROR RETURN
. . .               NORMAL RETURN
```

Issue New Synchronization

When applicable to the device, this routine does whatever is necessary for the device to send a new sync pulse (also called quick sync) over the line.

Called by:

CD:NSY   COMX Device Driver New Sync Subroutine

Calling sequence:

```
JST       DEVDEP    CALL DEV-DEPENDENT ROUTINE
. . .               ERROR RETURN
. . .               NORMAL RETURN
```

Issue Break

When applicable to the device, this routine does whatever is necessary for the device to send a break over the line.

Called by:

Calling routine non-existent; ASYNCH only

Calling sequence:

```
JST       DEVDEP    CALL DEV-DEPENDENT ROUTINE
. . .               ERROR RETURN
. . .               NORMAL RETURN
```

Reverse Channel

When applicable to the device, this routine does whatever is necessary for the device to be in a reverse-channel mode.

Called by:

Calling routine non-existent; ASYNCH only

Calling sequence:

```
JST       DEVDEP    CALL DEV-DEPENDENT ROUTINE
. . .               ERROR RETURN
. . .               NORMAL RETURN
```

Line Transmit Procedure

This routine does whatever is necessary (e.g., start output, turn line around) to transmit the data prepared by the COMX Data Transmit Procedure over the line.

Called by:

CD:DTP	COMX Device Driver Data Transmit Procedure

Calling sequence:

```
JST       BEGIN:    START A TASK
DATA      DEVDEP    CALL DEV-DEPENDENT ROUTINE
DATA      CLTPRI    LINE TRANSMIT PRIORITY
  .
  .
```

On error or normal return, return to CD:WRX (COMX Device Driver WRITE Request Exit) with the request status (STATUS in CIB) set accordingly.

5.2 DEVICE DRIVER SERVICES

Several service routines are provided to aid in the writing of device-dependent routines. The services are described below.

CD:IER (Input Error)

This common input error service combines the given status with the existing line status (CLSTST) and checks for termination errors. If termination errors have not occurred, CD:IER will end and allow the receive procedure to continue. If termination errors have occurred, CD:IER stops input on the line, then checks for loss of DSR (data set ready). If DSR is lost, CD:IER also disables the line. The receive procedure is scheduled to end the request.

Calling sequence:

```
LDX       input-CIB-address
LDA       error-status
```

```
followed by:                      or, if operating at CRPPRI+1
     JST       BEGIN:                  JMP       CD:IER
     DATA      CD:IER,CRPPRI+1
```

Returns with:

Does not return

CD:OER (Output Error)

This common output error service combines the given status with the existing line status (CLSTST) and checks for termination errors. If no termination errors have occurred, CD:OER ends and allows the transmission to complete. If transmission errors have occurred, CD:OER stops output on the line, then checks for loss of DSR (data set ready). If DSR is lost, CD:OER also disables the line. The A register is loaded with the line status, and control is returned to the output initiator (the CD:STO caller).

Calling sequence:

```
          LDX       output-CIB-address
          LDA       error-status

     followed by:                        or, if operating at CLTPRI:
          JST       BEGIN:                         JMP       CD:OER
          DATA      CD:OER,CLTPRI
```

Returns with:

```
          Does not return
```

CD:OEX (Output EOB Interrupt Exit)

This common output end-of-block service cancels the deadman timer, clears output active in the line software status, and returns to the output initiator (the CD:STO caller).

Calling sequence:

```
          LDX       output-EOB-interrupt-routine-address

     followed by:                        or, if operating at CLTPRI:
          JST       BEGIN:                         JMP       CD:OEX
          DATA      CD:OEX,CLTPRI
```

Returns with:

```
          A REGISTER - ZERO
          X REGISTER - OUTPUT CIB ADDRESS
```

CD:IFN (Issue Function)

This service gets the device address from the Line Information Block (LIB), combines it with the given instruction, then executes it with the given A register. An example is given in Figure 5-1.

Calling sequence:

```
        LDX       LIB address
        LDA       desired A register when function is issued

        SIN       3
        JST       CD:IFN
        . . .     O,FC       INSTRUCTION AND FUNCTION CODE
        . . .                RETURN, SIN IN EFFECT FOR 1 INSTRUCTION
```

Returns with:

```
        A REGISTER - NEW A REGISTER, IF THE A REGISTER HAS CHANGED.
```

```
                  LDX       LIB-address
                  SIN       3
                  JST       CO:IFN
                  INA       0,1        FUNCTION TO INPUT STATUS TO A REG
                                       RETURN, A REG = STATUS
```

Figure 5-1. Example of CD:IFN

CD:NOP (NOP)

This service simply returns. It can be used in the Device-Dependent-Routine Table when no function needs to be performed.

CD:NCP (NOP Character Processor)

This service simply takes the normal return. It can be used in the Device-Dependent-Routine Table when no function needs to be performed in a character processor.

• • • • • • • • • •

Unless otherwise stated, the following services have the following calling sequences:

```
        LDX       LIB-address
        JST       CD:xxx
        . . .                RETURN
```

All of these services call the appropriate device-dependent service to perform the particular function for the device. Additional operations are noted.

CD:LNP (Set Line Parameters)

Line parameters are defined in the DIB. They include word length parity selection and enable.

CD:SYN (Set Sync Character)

CD:SPC (Set Special Character)

CD:INT (Initialize Line)

This service sets the line initialized in the line software status.

CD:ENL (Enable Line)

This service sets line enabled in the line software status.

CD:DSL (Disable Line)

This service resets line enabled in the line software status.

CD:STI (Start Input)

This service sets input active in the line software status, sets Auto-I/O byte count to -1, primes the current character address for the Receive Procedure (CD:RVP), and stores the input buffer address in the Auto-I/O instruction. It returns immediately to the caller. The error return is to CD:IER (Interrupt Error Routine); the completion return is to CD:SRP (Schedule Receive Procedure).

CD:SPI (Stop Input)

This service cancels the input timer, clears the Auto-I/O byte count and buffer address, and resets input active in the software line status.

CD:STO (Start Output)

This service saves the return address in the CIB, sets output active in the software line status, sets byte count and buffer address from the LIB into the Auto-I/O instruction, and starts a deadman timer. If completion is normal, the A register contains a zero; if an error, the A register contains the line error status.

CD:SPO (Stop Output)

This service cancels the output timer and resets output active in the software line status.

CD:NSY (Issue New Sync)

CD:RSY (Resynchronize Line)

5.3 I/O REQUEST HANDLING

Three types of requests control the functions of the Device Driver: READ, WRITE and FUNCTION requests. These requests are made by the Protocol Driver to the Device Driver, as detailed in Section 4.4. The Device Driver processing is detailed below.

READ Request Processing

The function of a READ request is to input data to the protocol for passage to the user. This is independent of the actual reading of data from the communications line. The line will be actively reading data either all the time or only when transmission is active. Data will be buffered by the Device Driver, if necessary. The driver, of course, will wait for data as necessary when processing a READ request.

The READ request received by the Device Driver from the Protocol Driver commands the usage of received data. Based on previously set parameters[1], the Device Driver will locate control information, transmit data to the destination buffer, translate the data, establish timeouts and detect errors. The parameters controlling these operations are provided in the protocol tables[2], in previously issued functions and in the READ IOB.

WRITE Request Processing

The function of the WRITE request is to transmit data supplied by the Protocol Driver over the communications line. Each WRITE request contains part or all of a message. In accordance with previously set parameters[1], the Device Driver will locate control information, transmit data characters over the communications line, translate the data, recognize the end of a message and turn the line around. The parameters controlling these operations are provided in the protocol tables, in the previously issued functions and in the WRITE IOB.

Due to line timing constraints, it is sometimes necessary for the device handler to assemble an entire message prior to transmission of any part of it. The Device Driver will complete each WRITE request by performing any specified processing (e.g., translation) and moving the data to the device buffer. When the final WRITE request (end of message) is issued, the device handler will transmit over the communications line.

FUNCTION Request Processing

FUNCTION requests control the state of various parameters of the communications line. They are used to affect hardware control of the line and to command the device handler (or the hardware) to perform certain protocol-related functions. The two kinds of functions are queued and immediate. Queued functions are performed as they rise to the top of the queue. Immediate functions are performed as soon as they are issued.

5.4 DEVICE DRIVER TABLES

The Device Driver requires a table that lists the addresses of all of the device-dependent routines, an input CIB for each device, an output CIB for devices attached to full-duplex lines, an output EOB routine for devices attached to half-duplex lines, and a Line Information Block (LIB) for each line. Details of these tables are given in an appendix.[3]

---

[1] Subsection 6.1

[2] Subsection 4.6

[3] Appendix C

Device-Dependent-Routine Table

Each device handler must have a Device-Dependent-Routine Table. This table contains the addresses of the routines that handle device-unique procedures. The user must prepare these routines himself when writing a device handler. The table is formatted as follows:

WORD	ROUTINE ADDRESS
0	Reserved
1	Set Line Parameter
2	Set Sync Character
3	Set Special Character
4	Initialize Line
5	Enable Line
6	Disable Line
7	Start Input
8	Stop Input
9	Start Output
10	Stop Output
11	Input Character Processor 1
12	Input Character Processor 2
13	Output Character Processor 1
14	Output Character Processor 2
15	Raise and/or Wait for Data Set Signals On
16	Lower and/or Wait for Data Set Signals Off
17	Resync Line
18	Issue New Sync
19	Issue Break
20	Reverse Channel
21	Line Transmit Procedure

These routines are described in an earlier part of this section.[1]

Controller Information Block (CIB)

For a half-duplex line, the Device Driver needs only one CIB. The same CIB is used for both input and output processing because input and output are not concurrent. However, a separate output end-of-block interrupt routine is needed. This consists of words 1-14 of an output CIB.

For a full-duplex line, both an input and an output CIB are required.

---

[1] Subsection 5.2

Many of the CIB entries are standard for IOX. The entries added for COMX are:

WORD	CONTENTS
0	Beginning of Request Flag/Input or Output End-of-Block Interrupt Address
1-12	Input or Output End-of-Block Interrupt Routine
13	Timer/Interrupt State Flag
14	Line Information Block Address
20	Input Interrupt Routine Address
21	Device Driver Subroutine Return Address
23	Line Error Status
33	Return Address for Device Driver Start Output Routine
34	Device CIB Chain
35	Current Character Address
36	Current Character(s)
37-38	Temporary Cells
39	Stop Table Flags
40	Translate Routine Address from Caller's DIB
41	A Register Returned to Caller at EOR
42	Current Interrupt Buffer Address
43	Additional Interrupt Byte Count
44	Character Bit Mask
45	Length of Input/Output CIB

The Line Error Status in word 23 is defined as follows:

Bits	Definition
0	Transmit data overrun
3	Transmit framing error/break detect
5	Transmit timeout
6	Loss of clear to send
7	Loss of data set ready
8	Loss of carrier detect
9	Special timeout
10	Receive timeout
12	Receive framing error/invalid stop character
14	Parity error
15	Receive data overrun

The Stop Sequence Table flags in word 39 are defined as follows:

Bits	Definition
0-7	Table count
10	Accumulate checksum on stop character(s)
11	Transfer stop char(s) (READ-store, WRITE-transmit)
12	Error if stop character not found
13	Error if current char is not stop char
14	Error if first found and pair is not a stop condition
15	First stop character found

Line Information Block (LIB)

The Device Driver requires a Line Information Block to define each communications line attached to the device. The LIB is not standard for IOX; it serves only COMX. The contents of the LIB are:

WORD	CONTENTS
0	Standard Line Status Word
1	Software Line Status
2	Device Address
3	Device Driver Flags
4	Device Transfer Vector
5	Input CIB Address
6	Output CIB Address
7	Output EOB Interrupt Rtne Address
8	Input Word Interrupt Location
9	Output Word Interrupt Location
10	Start Input Instruction
11	Input Interrupt Buffer Start Addr.
12	Input Interrupt Buffer End Addr.
13	Output Device Buffer Address
14	Output Interrupt Buffer Addr.
15	Output Interrupt Byte Count
16	Function Proc. Current CIB
17	Function Proc. Next CIB
18	Last Function Issued
19	Last Hardware Status Word
20	Current Line Parameter Word
21	Line (Modem) Signal Delay
22	Device-Dependent Word 1
23	Device-Dependent Word 2
24	Device-Dependent Word 3
25	Pad Character Information
26	SMC Sync/Special Character
27	Exception Interrupt Point of Interrupt

The Standard Line Status in word 0 is defined as follows:

Bits	Definition
0	Data set ready
1	Clear to send
2	Carrier detect
3	Ring indicator
4	Signal quality
5	Secondary channel receive
6	Transmit framing error
7	Receive framing error
8	Parity error
9	Transmit data overrun
10	Receive data overrun
11	Secondary channel receive
12	Data ready
13	Transmitter empty

The Software Line Status in word 1 is defined as follows:

Bits	Definition
0	Transmit mode/output active
1	Receive mode/input active
2	Echo mode
3	Loop back mode
4	Protocol duplex
5	Line duplex
6	Modem/controller type
7	Answer mode
8	Carrier type

Bits	Definition
9	Carrier detect must be lowered before request to send is raised
12-13	Line code set
14	Line enabled
15	Line initialized

The Pad Character Information in word 25 is defined as follows:

Bits	Definition
0-7	Number of pad chars to be sent
8-15	The character to be sent

SECTION 6

SYSTEM GENERATION

In communications systems, there are an infinite variety of possible combinations of protocols, code sets, devices and lines. Even when the standard protocol and device are used, the options are numerous. Therefore, the tables must be supplied by the user. COMX tables also reference the object modules required to support the desired configuration. During loading (or Link Edit) these references call the modules to be included in the load.

The tables that must be supplied by the user include those for the protocol and any non-standard translation as well as the device, controller and line information blocks (DIBs, CIBs and LIBs). In addition, the user has the normal IOX control over the Unit Assignment Table (UAT).

COMX System Generation is in two phases: 1) the assembly of the tables, and 2) the load phase which is identical to that performed under RTX/IOX. The tables can be generated from the macros described in this section and assembled with the MACRO2 or MACRO3 assembler. Alternately, the user can build his own tables according to the required format and content presented in another section.[1]

6.1 TABLE REQUIREMENTS

The tables required for COMX System Generation are dependent on the number of communication lines and the protocol and line configurations. The table requirements are:

- Protocol Driver Tables
    - 1 set of protocol tables per driver
    - translation tables for nonstandard code sets (other than ASCII or EBCDIC)
  - for half-duplex protocol
    - 1 CIB per line
    - 1 DIB per line
  - for full-duplex protocol
    - 2 CIBs per line
    - 2 DIBs per line
- Device Driver Tables
  - for half-duplex line
    - 1 full input CIB per line
    - 1 partial output CIB per line
    - 1 DIB per line
    - 1 LIB per line
  - for full-duplex line
    - 1 full input CIB per line
    - 1 full output CIB per line
    - 1 input DIB per line
    - 1 output DIB per line
    - 1 LIB per line

---

[1] Subsection 4.6 and 5.4

6.2 TABLE-GENERATING MACROS

The macros described below work in combination to generate all of the tables required for a communications line. Figure 6-1 shows an example of the use of the table-generating macros. The macro source file on paper tape is supplied with RTX. The file listing is shown in Part II, Subsection 3.3, of this manual.

To process a line, the user first must specify the protocol, device and line configurations and then generate the tables. The DEVICE macro is used to set up all device-related and line-related parameters. A protocol driver designating macro (BSC for the standard BiSync) is used to set up all protocol-related parameters. Once specified, device, header, line and protocol parameters can be left unchanged for the generation of several sets of line tables. A CLINE macro is used to generate the set of tables based on these parameters. The CLINE macro has the ability to override certain device and line parmeters that were previously supplied. The NAMDEF macro performs initialization and must precede any other macro in the assembly.

The NAMDEF Macro

This macro generates NAM directives at the beginning of the module and initially defines all necessary symbols for use of subsequent macro processing. The form is:

```
NAMDEF    [dib-name1[,dib-name2[,...]]]
```

where: dib-name1 A four-character DIB name that the user will use in a subsequent CLINE macro.

```
NAMDEF    PD00,DD00
BSC       400,EBCDIC,XLATE,DISCON
HEADER    6,:32,:32,:32,:32,:32,:32
DEVICE    513
CLINE     PD00,12,DD00,:D,:A0,8,2400
```

This set of macro calls will generate the Protocol DIB (D:PD00) and CIB and the Device Driver DIB (D:DD00), CIB and LIB. The BiSync Protocol Handler will be loaded. The maximum transmission size is 400 characters. The line code is EBCDIC; and translation will be done so that the user will deal with ASCII. Whenever a Write File Mark is issued, the Protocol Handler will disconnect the line. The header consists of six sync characters (:32 in EBCDIC). The device being used is the Synchronous Modem Controller 53513. The device address is :0; and the interrupt address starts at :A0. The line is a synchronous, half-duplex, switched line (line type 8) with a speed of 2400 BPS.

Figure 6-1. Example of Using Table-Generating Macros

Every Device Information Block (DIB) name used in the module being assembled must be specified in a NAMDEF macro. As many names as can be typed prior to column 72 can be placed on each NAMDEF; and as many NAMDEFs can be used as are required; but all must appear prior to the first statement that generates code. The macro will concatenate a prefix of "D:" to each DIB name specified. The user can write NAM statements for the DIB names and prefix each with a "D:", or he can use NAM statements to declare other entry names; but at least one NAMDEF macro must be present to define the symbols necessary for macro processing even if the NAMDEF has no operands.

The DEVICE Macro

This macro describes the kind of device being used, its address and any options on it. The form is:

```
     DEVICE    device-type[,line-id[,speed[,device-address,interrupt-address[,
               options[,. . .]]]]]
```

where:	device-type	The last three digits of the CA controller part number; for SMC, use 513 or 617.
	line-identifier	The id of the communications line and modem configuration; Table 6-1 lists all possible line identifiers; it will be 4 or 8 for the standard COMX.
	speed	The line speed expressed in Baud rate.
	device-address	The device's (SMC's) address.
	interrupt-address	The device's (SMC's) first or main interrupt address (same as the address used to strap the controller).
	options are:	
	CONSTC	CONSTant Carrier. On a line so specified, the carrier will be held on during I/O. On a synchronous full-duplex line (line-id 3 or 7) that is being used in a half-duplex manner, setting CONSTC will cause COMX to output continuous sync characters during input operations. This also assumes that the terminal does the same (i.e., the line is always in sync in both directions).
	WAITCO	WAIT for Carrier Off. This causes COMX to complete each input message by waiting for carrier detect to go off prior to transmitting the message. This is useful on half-duplex modems that present clear-to-send too early (e.g., ITT Synchronous Modems).

A DEVICE macro must be issued prior to any CLINE macros defining lines connected to a particular device. Each CLINE issued after a given DEVICE macro will generate Device CIB and DIB tables for that device. This specification can be overridden by a subsequent DEVICE macro.

Table 6-1. Line Identification

Line Identifier	Duplex	Modem Type	Line Type	RS-232 Compatible	Compatible BELL System Modem (examples only)
0	Full	None-Current Loop	Dedicated	No	None
1	Full	Async	Dedicated	Yes	103/113 Series
2	Half	Async	Dedicated	Yes	202D/202T
3	Full	Synchronous	Dedicated	Yes	201B/208A/201C
4	Half	Synchronous	Dedicated	Yes	201B/208A/201C
5	Full	Async	Switched	Yes	102/113 Series
6	Half	Async	Switched	Yes	202C/202S
7	Full	Synchronous	Switched	Yes	CODEX LSI 48D
8	Half	Synchronous	Switched	Yes	201A/208B/201C

The line-identifier, device-address, and interrupt-address parameters are optional since they can also be specified on a CLINE macro. The line-identifier parameter must be specified on one of the two macros. The device-address and interrupt-address parameters will default, if not specified, to the COMX standard addresses for that device-type. Any parameters specified must be in the proper operand position on the line; i.e., the proper number of commas, if any, must precede a specified parameter if some parameters are omitted.

Appropriate EXTR statements are produced to cause loading of the Device Driver denoted by the type parameter.

The CLUN Macro

This macro generates a 16-bit Logical Unit Name/Number according to the standard naming convention used in COMX for the interface between the Protocol Driver and the Device Driver. The CLUN macro adds 16384 (:4000) to the lun parameter, creating a COMX standard LUN. This sum is used as the operand of a DATA statement. This macro is used by the CLINE macro.

The macro format is:

```
     CLUN      lun
```

where:	lun	The LUN number (<16384).

The BSC Macro

This macro designates the line as being attached to a Binary Synchronous Communications Protocol Driver. It is issued prior to the CLINE macros defining BSC lines. Each CLINE after a BSC macro generates a BSC Protocol Driver CIB and DIB. A subsequent Protocol-Driver-designating macro will override this.

Issuance of this macro causes the BSC Protocol Driver modules, the BSC Protocol Tables and a translate table (if requested) to be declared in EXTR statements and, thus, be included in the final load.

The macro format is:

```
     BSC       max-msq-size,char-set[,options[, . . .]]
```

where:	max-msq-size	The size of the largest message transmitted or received over the line (in bytes).
	char-set	Either ASCII or EBCDIC; also selects the block check character (BCC) sequence for error checking.

Options:

XLATE	Selects auto-translation when character set is EBCDIC, allowing all user data to be in ASCII.
XPRNT	Selects Transparent Mode for all WRITEs; a FUNCTION request can change this.
UXPRNT	Selects Transparent Mode only for Unformatted WRITE requests; a FUNCTION request can change this option.
ULC	User line control to transmit a sequence of WRITEs at one time.
DISCON	Disconnect on File Mark.
SLAVE	Slave Mode; default is Master Mode.

The HEADER Macro

This macro defines a one-to-ten character header to be used on transmitted messages. When using the SMC (either 513 or 617) and transmitting in ASCII, the bit positions 8-0 in the character must be reversed (0-8). For example, :16 should appear as :68, and :C4 should be :23.

The macro format is:

```
     HEADER    count,char-1[,char-2 . . .[,char-10]]
```

where:

count	The number of characters (1-10) in the header.
char-1 (-10)	The nth character in the heading.

The HEADER macro must be used in conjunction with the BSC macro to define the leading synchronization characters on a message. The suggested form is:

```
     HEADER    4,:32,:32,:32,:32,:32,:32      for EBCDIC
     HEADER    4,:68,:68,:68,:68,:68,:68      for ASCII
```

The CLINE Macro

This macro provides the basic definition of a communications line. It generates the Line Information Block (LIB) and the basic CIBs and DIBs for a line.

On a half-duplex line, the tables generated are:

1 Device Driver input CIB
1 partial Device Driver output CIB
1 Device Driver DIB

On a full-duplex line, the tables generated are:

1 Device Driver input CIB
1 Device Driver output CIB
1 Device Driver input DIB
1 Device Driver output DIB

For a half-duplex protocol, the tables generated are:

1 Protocol Driver CIB
1 Protocol Driver DIB

For a full-duplex protocol, the tables generated are:

2 Protocol Driver CIBs
2 Protocol Driver DIBs

The Device Driver tables are chosen according to the specification in the preceding DEVICE macro. The Protocol Driver tables are generated according to the specification in the preceding BSC macro. If the device address and interrupt address parameters are not supplied in the CLINE macro, they are obtained from the preceding DEVICE macro or are defaulted.

The macro format is:

```
CLINE     dib-name1,device-LUN,dib-name2[,dev-address,int-address[,line-id
          [,speed[,options[, . . .]]]]]
```

where:

dib-name1	The name given to the Protocol Driver DIB for the line, specified as four characters.
dev-LUN	The Logical Unit Name/Number (LUN) to which the Protocol Driver CIB will be connected. In the UAT, the user must assign this LUN to dib-name2 (see below).
dib-name2	The name given to the Device Driver DIB for the line, specified as four characters.
dev-address	The device's (controller's) address. This parameter can be supplied on the CLINE or DEVICE macro or can be defaulted.
int-address	The device's (controller's) first or main interrupt address. This parameter can be supplied on the CLINE or DEVICE macro or can be defaulted.
line-id	The identifier describing the communications line and modem configuration. Table 6-1 gives the values for this parameter. This parameter must be supplied on the CLINE or DEVICE macro.
speed	The line speed expressed in Baud rate. This parameter must be supplied on the CLINE or DEVICE macro.

DIB names are chosen by the user. These names must be assigned to Logical Units (LUNs) by the user in the Unit Assignment Table (UAT). The application program will perform I/O to a given LUN, which should be assigned as the DIB name (dib-name1) for the appropriate Protocol Driver line. The Protocol Driver will perform I/O to the LUN specified by the user in the CLINE macro (device-lun). This user-supplied LUN should be assigned in the UAT to the appropriate Device Driver DIB for that line (dib-name2).

Standard DIB names recommended are:

D:CSnn	for synchronous Device Driver Lines
D:BSnn	for BiSync Protocol Driver lines

where nn are two digits unique to each line. The CLINE macro will prefix all user names with "D:".

The standard convention for naming the LUN connecting the Protocol Driver to Device Driver DIB is to use an unsigned integer (<16384) plus 16384 (:4000). A macro (CLUN) parameter is available to generate such a LUN. The CLINE macro uses the CLUN macro. The device-LUN parameter on the CLINE macro, therefore, should be an integer <16384. The user may use the CLUN macro in his UAT definition to generate the LUN values.

APPENDIX A

GLOSSARY

asynchronous transmission	Transmission of data in which each character consists of data bits and start and stop bits; may be blank periods between characters.
channel	User view of the circuit configuration.
circuit	The medium carrying the message; may be a line, cable, radio, microwave, or even just a pair of wires.
data link	A group of lines between two facilities; everything required to link two facilities including interface equipment and the transmission channel (line).
full-duplex circuit	A circuit permitting simultaneous two-way communication.
half-duplex circuit	A circuit permitting two-way communication, one way at a time.
modem	A device that provides the interface between a telephone or any other form of communications line (analog) and a terminal or computer (digital) and does the necessary analog/digital conversion.
multipoint system	A communications system having a central station with multiple tributary stations, normally with nonswitched permanent data links; the central station selects and polls the tributaries.
network	The aggregate of all data links used by an organization.
nonswitched system	A communications system in which dedicated lines provide permanent connections between stations.
point-to-point system	A communications system in which only two stations are linked at a time; with a switched system, there may be multiple stations although only two can be linked at any one time.

polling	A centrally controlled method of calling a number of transmitting stations (terminals) to permit them to transmit information; an invitation to send.
protocols	Rules of procedure; the formats for packaging data for transmission.
selection	A request made by the central controller to receive a message from a transmitting station.
simplex circuit	A circuit permitting the transmission of data in only one direction.
switched system	A communications system in which the data link is disconnected between transmissions and reestablished for each new transmission by standard dialing procedures (telephone link).
synchronous transmission	Transmission of data bits only; no start or stop bits are present to synchronize each character; an entire block is synchronized and the incoming bits are counted to form a character; blank periods between characters in a block are limited.
transparent mode	A convention in which all possible bit combinations can be used as data; none are regarded as control characters (unless distinguished by a leading escape character, such as DLE in BiSync).

APPENDIX B.1

ASCII CHARACTER CODES

CHARACTER	ASCII CODE	CHARACTER	ASCII CODE
Null	00	> greater	3E
Bell	07	?	3F
CR	0D	@	40
LF	0A	A	41
blank	20	B	42
! exclamation	21	C	43
" quote	22	D	44
#	23	E	45
$	24	F	46
%	25	G	47
&	26	H	48
' apostrophe	27	I	49
(open paren.	28	J	4A
) close paren.	29	K	4B
* asterisk	2A	L	4C
+	2B	M	4D
, comma	2C	N	4E
- minus	2D	O	4F
. period	2E	P	50
/ slash	2F	Q	51
0 zero	30	R	52
1	31	S	53
2	32	T	54
3	33	U	55
4	34	V	56
5	35	W	57
6	36	X	58
7	37	Y	59
8	38	Z	5A
9	39	[open bracket	5B
: colon	3A	\ back slash	5C
; semicolon	3B	] close bracket	5D
< less	3C	↑ up arrow	5E
= equal	3D	← back arrow	5F
		Rubout	7F

APPENDIX B.2

EBCDEC CHARACTER CODES

CHARACTER	EBCDIC CODE	CHARACTER	EBCDEC CODE	EBCDIC lower case
Null	00	>greater	6E	
Bell	2F	?	6F	
CR	0D	@	7C	
LF	25	A a	C1	61
blank	40	B b	C2	62
! exclamation	5A	C c	C3	63
" quote	7F	D d	C4	64
#	7B	E e	C5	65
$	5B	F f	C6	66
%	6C	G g	C7	67
&	50	H h	C8	68
' apostrophe	7D	I i	C9	69
(open paren.	4D	J j	D1	6A
) close paren.	5D	K k	D2	6B
* asterisk	5C	L l	D3	6C
+	4E	M m	D4	6D
, comma	6B	N n	D5	6E
- minus	6D	O o	D6	6F
. period	4B	P p	D7	70
/ slash	61	Q q	D8	71
0 zero	F0	R r	D9	72
1	F1	S s	E2	73
2	F2	T t	E3	74
3	F3	U u	E4	75
4	F4	V v	E5	76
5	F5	W w	E6	77
6	F6	X x	E7	78
7	F7	Y y	E8	79
8	F8	Z z	E9	7A
9	F9	[open bracket	C0	
: colon	7A	\ back slash	E0	
; semicolon	4F	] close bracket	D0	
< less	4C	↑ up arrow	4F	
= equal	7E	← back arrow	5F	
		Rubout (DEL)	07	

APPENDIX C.1 PROTOCOL DRIVER DEVICE INFORMATION BLOCK (DIB)

WORD		CONTENTS
0	CHAIN	*DIB Chain Pointer
1	DCIB	*CIB Address
2	QUEQE	*IOB Request Queue
3	CN	*Coordination Number
4	DSW	*Device Specification Word
5	DT	*Device Type
6	DCUN	*Controller No./Unit No.
7	DCTMR	Input Char. Timer Value
8	DFLGS	*Function Codes
9	DULS	*Max. Bytes - ASCII/Binary
10	DPFLGS	Protocol- and Device-Dependent Flags
11	DBLKSZ	Block Size
12	DLPARM	Line Parameters
13	DIXLTR	Input Translate Routine
14	DOXLTR	Output Translate Routine
15	DLUN	Device LUN
16	DDIB	Line DIB Chain
17	DSRTN	Subroutine Return Address
18	DTMP1	Temp. Cell 1
19	DTMP2	Temp. Cell 2
20	DCKSR	Checksum Routine Address
21	DCKSF	Checksum Flags
22	DCKSP	Checksum Polynomial
23	DCKSA	Checksum Accumulation
24	DHDRCT	Header Byte Count
25-29		Header Bytes 1-10

Words 1-6 - Standard for IOX

Word 7 - Input Character Timer Value

Word 8 - Standard for IOX

Word 9 - Standard for IOX

---

*Standard for IOX

Word 10- Device-Dependent Flags

The device-dependent flags are defined as follows:

Bits	Equate	Definition
0-7	DDFLGS	Device-dependent flags
0	DDFMAS	Master station
3	DDFACU	Auto-call this line
8-15	DPFLGS	Protocol-dependent flags
8	DPFUWX	Unformatted WRITEs transparent
9	DPFAWX	All WRITEs transparent
10	DPFUSR	User line control
11	DPFDIS	Disconnect on Write File Mark

Word 11- Maximum Block Size

Word 12- Line Parameters
The line parameters are:

Bits	Equate	Definition
0	DLPPEN	Parity enable
1	DLPPSL	Parity select, 1=even
2-3	DLPWSL	Word length select, 00=5 bits per character, 11=8 bits per character

Word 13- Input Translate Routine Address

Word 14- Output Translate Routine Address

Word 15- Device Logical Unit Name/Number

Word 16- Line DIB Chain

Word 17- Device Driver Subroutine Return Address

Word 18- Device Driver Temporary Cell 1

Word 19- Device Driver Temporary Cell 2

Word 20- Checksum Routine Address

Word 21- Checksum Flags

The checksum flags are defined as follows:

Bits	Equate	Definition
0-7	DCKSM	Checksum character mask
15	DCKSL	Checksum length, 1=2 characters

Word 22- Checksum Polynomial

Word 23- Checksum Accumulation

Word 24- Header Byte Count

Words 25-29- Header Bytes 1-10

APPENDIX C.2 PROTOCOL DRIVER CONTROLLER INFORMATION BLOCK (CIB)

WORD				CONTENTS
0	CBOR			*Beginning of Request Flag
1	CHAIN	0		Temp. Cell
2	IDT,PRI	1		Device Type
3	ICUN,ITCB	2		Device No.
4	CN	3	I	Coordination No.
5	ILUN	4		Device LUN
6	ISTA,IOP	5	O	Status Opcode
7	IRCNT	6		Requested Count
8	IBUFF	7	B	Buffer Address
9	IACNT,ISTPTB	8		Actual Count /Stop Seq. Table Address
10	IPCMR	9		Protocol Char Mod Routine Address
11	ICTMR	10		Character Timer
12				
13				Unused by Protocol Driver
14				
15	CJTBL			*READ Entry
16				*WRITE Entry
17				*POSITION Entry
18				*FUNCTION Entry
19				*Special Operation Entry
20	CPFLGS			Protocol Driver Flags
21	CPTBL			Protocol Driver Master Table
22	EXCESS			*Caller IOB Word 8 - Prompt Chars
23	CLSTAT			Line Status
24	CIOB			*Caller IOB Address
25	COP			*Caller IOB Word 5 - Opcode, Status
26	CRCNT			*Caller IOB Word 6 - Req. Count
27	CBUFF			*Caller IOB Word 7 - Buffer Address
28	CTCNT			*Actual Count
29	CDAA			*Caller IOB Word 9
30	CDIB			*Busy Flag (Current DIB Address)
31	CFLGS			*DIB Word 8

(Cont'd)

---

*Standard for IOX

WORD		CONTENTS
32	STATUS	*Status for EOR
33	CRTN	*SIO (CP:XIO) Return Address
34	CPDIB	Current DIB Address for COMX
35	CURCNT,CPST1	Temp Cell 1
36	CUBUFF,CPST2	Temp Cell 2
37	LPCIB	Length of Protocol CIB - CPST2 -1

Word 0 - Standard for IOX

Words 1-11 - Carried as the Input/Output Block (IOB) for the Protocol Driver.

Words 12-19 - Standard for IOX

Word 20- Protocol Driver Flags

The Protocol Driver flags are defined as follows:

Bits	Equate	Definition
0-2	CSTATE	Current protocol state
3	CRDRQ	READ request outstanding
4	CWRRQ	WRITE request outstanding
5	CACKFG	Acknowledgement (ACK) flag
6	CXPRNT	Transparent mode
7	CITBFG	Inter-block flag
8-9	CTOCNT	Timeout counter
10-12	CAKCNT	Invalid ACK counter
13-15	CNKCNT	Negative acknowledgment (NAK) counter

Word 21- Protocol Master Table Address

Word 22- Standard for IOX

Word 23-Line Status

The line status is defined as follows:

Bits	Equate	Definition
0	CLSTDO	Transmit data overrun
3	CLSTFE	Transmit framing error
5	CLSTTO	Transmit timeout
6	CLSCTS	Loss of clear to send (CTS)
7	CLSDSR	Loss of data set ready (DSR)
8	CLSCD	Loss of carrier detect (CD)
9	CLSSTO	Special timeout

Bits	Equate	Definition
10	CLSRTO	Receive timeout
12	CLSRFE	Receive framing error
14	CLSPE	Parity error
15	CLSRDO	Receive data overrun

Words 23-33 - Standard for IOX

Word 34- Current DIB Address for COMX

Word 35- Temporary
Used as storage for user's requested count and subroutines.

Word 36- Temporary
Used as storage for user's buffer address and subroutines.

Word 37- Length of Protocol CIB - CEST2+1

APPENDIX C.3 DEVICE DRIVER INPUT CIB

WORD		CONTENTS
0	CBOR	Beg. of Request Flag/Input EOB Interrupt Address
1		JST INTQS:
2		DATA $
3		DATA 0
4		DATA 0
5		DATA 0
6	CINTSK	DATA Input EOB Exit
7	COTTSK	DATA PRIORITY
8		DATA A REG
9		DATA X REG (CIB Address)
10		DATA P-LOC (CIB Address)
11		LDX $+2
12		JMP Input EOB Interrupt Routine
13	CTIMER	Timer/Interrupt State Flag
14	CLIB	LIB Address
15	CJTBL	*READ Entry
16		*WRITE Entry (if Full Duplex, Unused - 0)
17		*Unused - 0
18		*FUNCTION Entry
19		*Unused - 0
20	CINRTN	Input Interrupt Routine Address
21	CDSRTN	Subroutine Return Address
22	CSTPTB	*IOB Word 8 - Stop Seq. Table Address
23	CLSTAT	Line Error Status
24	CIOB	*Caller IOB Address
25	COP	*Caller IOB Word 5 - Opcode, Status
26	CRCNT	*Caller IOB Word 6 - Req. Count
27	CBUFF,CFUMP	*Buffer Address/Function Parameter
28	CTCNT	*Actual Count
29	CCMR	*IOB Word 9 - Char. Proc. Routine
	(Cont'd)	

*Standard for IOX

WORD		CONTENTS
30	CDIB	*Busy Flag (Current DIB Address)
31	CFLGS	*DIB Word 8
32	STATUS	*Status for EOR
33	CRTN	Return Address for CD:STO
34	CCHAIN	Device CIB Chain (DC::)
35	CCCADR	Current Character Address
36	CCCHAR	Current Character(s)
37	CTMP21	Temp Cell 1
38	CTMP22	Temp Cell 2
39	CSTPFG	Stop Sequence Table Flags
40	CXLTR	Translate Routine Address
41	CAEOR	A Reg Returned to Caller at EOR
42	CINTBF	Current Interrupt Buffer Address
43	CINTCT	Additional Interrupt Byte Count
44	CCHMSK	Character Bit Mask
45	LDICIB,LDOCIB	Length of Input/Output CIB - CCHMSK+1

Word 0 - Input End-of-Block Interrupt Address

Words 1-12 - End-of-Block Interrupt Routine

Word 13- Timer/Interrupt State Flag

Word 14- Line Information Block Address

Words 15-19 - Standard for IOX

Word 20- Input Interrupt Routine Address

Word 21- Device Driver Subroutine Return Address

Word 22- Standard for IOX

---

*Standard for IOX

Word 23- Line Error Status
The line error status is defined as follows:

Bits	Equate	Definition
0	CLSTDO	Transmit data overrun
3	CLSTFE	Transmit framing error/break detect
5	CLSTTO	Transmit timeout
6	CLSCTS	Loss of clear to send (CTS)
7	CLSDSR	Loss of data set ready (DSR)
8	CLSCD	Loss of carrier detect (CD)
9	CLSSTO	Special timeout
10	CLSRTO	Receive timeout
12	CLSRFE	Receive framing error/invalid stop character
14	CLSPE	Parity error
15	CLSRDO	Receive data overrun

Words 24-32- Standard for IOX

Word 33- Return Address for Device Driver Start Output Routine

Word 34- Device CIB Chain

Word 35- Current Character Address

Word 36- Current Character(s)

Words 37-38 - Temporary Cells

Word 39- Stop Sequence Table Flags
The stop table flags are defined as follows:

Bits	Equate	Definition
0-7	STPCNT	Table count
10	STPCKS	Accumulate checksum on stop character(s)
11	STPTSC	Transfer stop char(s) (READ-store, WRITE-transmit)
12	STPESF	Error if stop character not found
13	STPECN	Error if current char is not stop char
14	STPEPN	Error if first found and pair is not a stop condition
15	STP1ST	First stop character found

Word 40- Translate Routine Address from Caller's DIB

Word 41- A Register Returned to Caller at EOR

Word 42- Current Interrupt Buffer Address

Word 43- Additional Interrupt Byte Count
Header and device-dependent characters added to message by Device Driver

Word 44- Character Bit Mask
Masks out invalid bits as determined by the line parameter word in the DIB.

Word 45- Length of Input/Output CIB

APPENDIX C.4 DEVICE DRIVER OUTPUT CIB

WORD		CONTENTS
0	CBOR	Beg. of Request Flag/Output EOB Interrupt Address
1		JST INTQS:
2		DATA $
3		DATA 0
4		DATA 0
5		DATA 0
6		DATA Output EOB Exit
7		DATA PRIORITY
8		DATA A REG
9		DATA X REG (CIB Address)
10		DATA P-LOC (CIB Address)
11		LDX $+2
12		JMP Output EOB Interrupt Routine
13		
.		Same as Input CIB
.		
45		

Word 0 - Output End-of-Block Interrupt Address

Words 1-12 - Output End-of-Block Interrupt Routine

Words 13-45 - Same as for Device Driver Input CIB

APPENDIX C.5 LINE INFORMATION BLOCK (LIB)

WORD		CONTENTS
0	LSTNS	Standard Line Status Word
1	LSOFS	Software Line Status
2	LDADR	Device Address
3	LDFLG	Device-Dependent Flags
4	LDRTBL	Device Transfer Vector
5	LICIB	Input CIB Address
6	LOCIB	Output CIB Address
7	LOBIR	Output EOB Interrupt Rtne Address
8	LIINT	Input Word Interrupt Location
9	LOINT	Output Word Interrupt Location
10	LISIN	Start Input Instruction
11	LIBFS	Input Interrupt Buffer Start Addr.
12	LIBFE	Input Interrupt Buffer End Addr.
13	LOBFS	Output Device Buffer Address
14	LOBUF	Output Interrupt Buffer Addr.
15	LOBCT	Output Interrupt Byte Count
16	LFCIB	Function Proc. Current CIB
17	LFWCB	Function Proc. Next CIB
18	LLFUN	Last Function Issued
19	LHDWS	Last Hardware Status Word
20	LLPARM	Current Line Parameter Word
21	LLSDLY	Line (Modem) Signal Delay
22	LDDWD1	Device-Dependent Word 1
23	LDDWD2	Device-Dependent Word 2
24	LDDWD3	Device-Dependent Word 3
25	LPAD	Pad Character Information
26	LSPCH	SMC Sync/Special Character
27	LEXCPT	Exception Interrupt Point of Interrupt

Word 0 - Standard Line Status Word
Line status is defined as follows:

Bits	Equate	Definition
0-5	LSTLSG	Line signals
0	LSTDSR	Data set ready
1	LSTCTS	Clear to send
2	LSTCD	Carrier detect
3	LSTRI	Ring indicator
4	LSTQ	Signal quality
5	LSTSCR	Secondary channel receive
6	LSTTFE	Transmit framing error
7	LSTRFE	Receive framing error
8	LSTPE	Parity error
9	LSTTDO	Transmit data overrun
10	LSTRDO	Receive data overrun
11	LSTSCD	Secondary channel receive
12	LSTDRD	Data ready
13	LSTTMT	Transmitter empty

Word 1 - Software Line Status
Software line status is defined as follows:

Bits	Equate	Definition
0	LSFTM	Transmit mode/output active
1	LSFRM	Receive mode/input active
2	LSFEM	Echo mode
3	LSFLB	Loop back mode
4	LSFPDX	Protocol duplex (0 = full, 1 = half)
5	LSFLDX	Line duplex (0 = full, 1 = half)
6	LSFMTY	Modem/controller type (0 = async, 1 = sync)
7	LSFANM	Answer mode (0 = dedicated, 1 = switched)
8	LSFCTY	Carrier type (0 = constant, 1 = controlled)
9	LSFNCD	Carrier detect must be lowered before request to send is raised
12-13	LSFCOD	Line code set (00 = internal ASCII, 01 = USASCII, 10 = EBCDIC)
14	LSFENL	Line enabled (1 = enabled)
15	LSFINT	Line initialized (1 = initialized)

Word 2 - Device Address

Word 3 - Device-Dependent Flags

Word 4 - Device Transfer Vector
Address of Device-Dependent-Routines Table.

Word 5 - Input CIB Address

Word 6 - Output CIB Address
If half-duplex line, same as word 5 (i.e., only one CIB for both input and output).

Word 7 - Output End-of-Block Interrupt Routine Address
If full-display line, same as word 6 (output CIB address).

Word 8 - Input Word Interrupt Location

Word 9 - Output Word Interrupt Location

Word 10- Start Input Instruction
Complete with device address (e.g., AIN DA,1)

Word 11- Input Interrupt Buffer Start Byte Address

Word 12- Input Interrupt Buffer End Byte Address

Word 13- Output Device Buffer Byte Address

Word 14- Output Interrupt Buffer Byte Address

Word 15- Output Interrupt Byte Count

Word 16- FUNCTION Request Processing in Current CIB

Word 17- FUNCTION Request Waiting for Processing in Next CIB

Word 18- Last Function/Command Word to the Device

Word 19- Last Hardware Status Word Input
Status returned by device.

Word 20- Current Line Parameter Word
Same format as Line Parameter Word in Protocol Driver DIB.

Word 21- Line (Modem) Signal Delay (in ticks)

Word 22- Device-Dependent Word 1
Device-Dependent factors are defined as follows:

Bits	Equate	Definition
0	LDWDLE	DLE seen
13	LDWRXP	Receive in transparent mode
14	LDWTXP	Transmit in transparent mode
15	LDWTMD	Transmit modification

Word 23- Device-Dependent Word 2

Word 24- Device-Dependent Word 3

Word 25- Pad Character Information
Pad character information is defined as follows:

Bits	Equate	Definition
0-7	LPDCNT	No. pad chars. to be sent
8-15	LPDCHR	The char. to be sent

Word 26- SMC Sync/Special Character

Word 27- Exception Interrupt Point of Interrupt
Word 27 plus an additional 24 words are used only if the device has exception interrupts (e.g., the SMC).