# SYSTEMS ANALYSIS GUIDE FOR OS910 VERSION 2, A REAL-TIME OPERATING SYSTEM FOR THE SDS-910 COMPUTER



Leonard C. Moon
and Anthony P. Lucido

AMES LABORATORY, USERDA
IOWA STATE UNIVERSITY
AMES, IOWA

Date Transmitted: March 1976

PREPARED FOR THE U. S. ENERGY RESEARCH AND DEVELOPMENT ADMINISTRATION
UNDER CONTRACT W-7405-eng-82

## DISCLAIMER

IS-3860

# SYSTEMS ANALYSIS GUIDE FOR OS910 VERSION 2, A REAL-TIME OPERATING SYSTEM FOR THE SDS-910 COMPUTER

By

Leonard C. Moon and Anthony P. Lucido



Ames Laboratory, ERDA
Iowa State University
Ames, Iowa 50011

Date Transmitted: March 1976

PREPARED FOR THE U.S. ENERGY RESEARCH AND DEVELOPMENT ADMINISTRATION UNDER CONTRACT NO. W-7405-eng-82

Available from: National Technical Information Service
U. S. Department of Commerce
P.O. Box 1553
Springfield, VA 22161

Price: Microfiche $2.25
Paper Copy $4.50

TABLE OF CONTENTS

## ABSTRACT

OS910 Version 2 operating system is a real-time experiment control system for the SDS-910. The SDS-910 is coupled to a DEC PDP-15 where the PDP-15 is the computation element and the SDS-910 is used as an intelligent interrupt handler. The SDS-910 operating system along with the communication scheme between the SDS-910 and the PDP-15 are discussed in detail.

Distribution List

| | |
|---|---|
| Chicago Patent Office | 1 |
| USERDA-TIC | 27 |
| Ames Laboratory Library | 5 |
| Leonard C. Moon | 20 |
| Anthony P. Lucido | 3 |
| Total | 56 |

# I. Introduction:

OS910, the operating system for experiment control using the SDS-910 computer, has been designed to do task management and experiment control in an interrupt-driven environment. The Ames Laboratory Research Reactor has a multitude of experiments on it that need real-time data collection and control facilities.

The computer system that is designed to perform the requisite task management and experiment control consists of an SDS-910, used as an "intelligent interrupt handler", and DEC PDP-15 used as the computational element, with the two computers linked by an interface designed at the Ames Laboratory. Both computers have Ames Laboratory designed operating systems.

This report will detail the operating system for the SDS-910 computer, OS910. Facilities provided by OS910 include:

a) SDS-910/PDP-15 communications,

b) task scheduling,

c) scheduling of tasks on clock queues,

d) fielding of real-time error interrupts.

Also, great flexibility is allowed the user in managing experiment communication and experiment derived interrupts.

All SDS-910--PDP-15 communication is handled by OS910 routines, as well as the four clocks (12.8 msec. fast clock, 102.4 msec. medium clock, 1.6384 sec. slow clock, and 1.6384 sec. real-time clock), and

real-time error interrupts. A real-time error may be generated by either the user or the operating system, and is, for example, symptomatic of an invalid experiment address. When a RT error is encountered the user is terminated.

Under OS910, tasks may not be transient during execution. The code for a task must reside completely in storage, and may not be moved around. However, for control purposes, a set task/program format is imposed. A user's program has the form of:

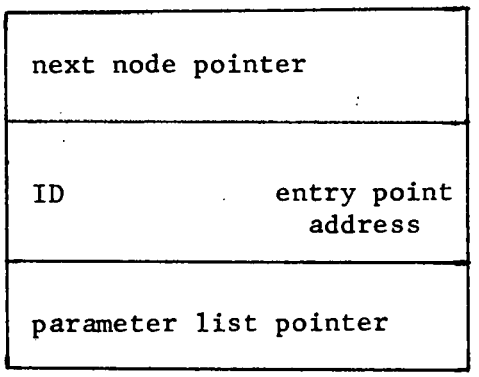
a) ID/KEY table entry for the user (to be explained later),

b) the sequence of tasks for the user.

As a 'KEY' is used with READ(s)/WRITE(s) in the PDP-15, and (it is assumed that) the user program is set up to accept one data length for a specific 'KEY', a table of the 'KEYS' and their related buffer areas may be maintained. Because of this, when the PDP-15 transmits data to the SDS-910, a predefined buffer area already exists in the SDS-910 to accept it. Dynamic buffering must exist for SDS-910 to PDP-15 transmissions, as the user will not know when the data are finally sent, nor, when the buffer is empty. OS910 supplies the user with POPs that take care of the buffering problem.

To provide this dynamic buffering, a transient buffer storage area is system maintained. The transient buffer storage area contains nodes of seven words in length. The format of a buffer will be shown later.

Also included are POPs to perform queuing of tasks. A task may be placed on the task queue, or one of the three clocks (fast, medium, or slow). Also, a parameter list may be referenced in this call. The format of a node on one of these queues is:




Here, the next node pointer is a simple forward reference, to create a linked list of nodes. The ID value (in bits 0-8 of the second word) is the user ID of the task to be invoked with the entry point address 'ANDED' into bits 10-23. If there aren't any parameters, the third word is set to minus one, else it contains the address of the parameter list.

A transient storage area is maintained for queue nodes. Currently, there are 30 nodes of three words each for queue nodes. If necessary, this area can easily be expanded or contracted to accommodate any run-time needs.

For further instruction as to the use of OS910 the reader is asked to refer to 'USERS MANUAL FOR OS910, THE SDS-910 OPERATING SYSTEM', IS-3773.

II. Communications Vector Table (CVT):

Included (in the CVT) are (system and user) constants, an interrupt idler routine, POPs, interrupt vectors, user interrupt handlers, and the table of entry points. We will now explain the different CVT entries.

The interrupt idler routine entry point, DUMMY, is at location 2, octal. When a regular interrupt entry uses a BRM DUMMY, then the interrupt level is cleared, and a return to the interrupted program results.

The real-time clock counter, TIME, is at location 4, octal. This counter may be directly referenced by a user. The value at TIME will not be allowed to exceed 400000, octal. Thus, an 18 bit word will always be able to hold the counter value, in its entirety.

The next four locations contain entry point address to the allocation/deallocation routines. Each routine is invoked via an indirect branch and mark (BRM*). To allocate a queue node, GETNDE is used. To get a buffer node, GETBUF is used. To free-up either a queue node or buffer node, use FRENDE or FREBUF, as required. The servicing routines, themselves, will be explained later.

At locations 11 and 12, octal, are the free storage list addresses. QNDEFQ references the free storage list for queue nodes, and BNDEFQ references the free storage list for buffer nodes.

Next come six numeric constants that may be used by the user, and are heavily relied upon by the system. These constants are in locations 13 to 20, octal. In order, the constants are -1, 0, 1, 2, 3, and 4. These may be symbolically referenced by MINUS1, ZERO, ONE, TWO, THREE, and FOUR, respectively. Octal location 21, is ADRMSK, which is an address mask. When used as the operand of an extract (ETR), ADRMSK will set all but bits 10-23 to zero.

Locations 22 through 24 octal contain pointers to the COLD START routine (COLD), the dispatcher (TASKER), and RAID (RAID), respectively. The general user will use these pointers.

Octal locations 25 through 27 contain C510, C210, and CCBP, respectively. C510 is the system ID (identifier). Each task in the PDP-15 is assigned a unique user ID. The system assumes control of ID 510. To signify which user currently has control, the variable CCBP is kept updated by the system. C210 is subtracted from the user's ID number to obtain an offset into the User Entry Point table which starts at $300_8$. CCBP contains the ID of the task that is currently in control of the computer.

Octal locations 30 through 33 are reserved for the W and Y buffer interrupts. These are not used in the present implementation but the hardware is attached to these locations.

Octal locations 34 through 36 are system mask variables, BIT9, IDNUM1, and SKDID, respectively. BIT9 is used to test/set bit location 9 of an SDS-910 word which tells the system if the particular

user is active (0) or inactive (1). IDNUM1 and SKDID are both used to extract and merge in the user ID number into different parts of the SDS-910 word.

Locations 40 through 42 octal are the system temporaries QA, QB, and QC, respectively. These temporaries are saved when an interrupt of higher priority occurs and restored upon completion of the interrupt servicing routine.

Location 100 octal begins the POP invocation area. Included are all system and user oriented POPs. This table will change as user/system requirements change.

Location 200 octal begins the interrupt vector area. The current interrupt locations should be obtained from the listing of the operating system code.

Locations 300 to 327 octal comprise the Entry Point table, which contains the entry point address value for each user/system task. Each location in the table is either set to the user's ID/KEY table address if the user is in core, or to -1 ($77777777_8$). Unless C210 is suitably modified, moving this table will provide catastrophic results.

In order to maintain system integrity, all user interrupts are first handled by the system so the "old" environment can be saved. When a user does an OPEN in the PDP-15 the user's interrupt(s) in the SDS-910 are brought from the idler state to and active state. This action is also done when a user does a PON of his/her interrupt.

III. Transient Storage Areas:

Octal location 341 begins the transient storage area for the system at the reactor. The free queue node uses three words per entry, and extends from locations 341 through 474, which is 30 three word nodes. The buffer node free storage area extends from location 475 through 1020, and gives 30 seven word nodes. The interrupt buffer queue is 20 ten word nodes extending from location 1021 through 1333. Each list is forward linked, with the last node's link field set to minus one.

It should be noted that changing the amount of free storage area is very easy, but an appropriate change to the COLD START routine (COLD) must be made to allow for the change in number of entries.

IV. COLD START:

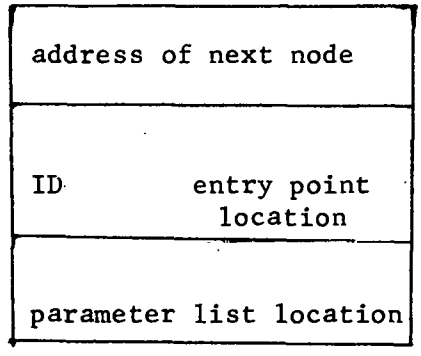
The COLD START routine will initialize the free storage areas, set the user interrupts to the idler state (BRM DUMMY for regular interrupts, the special interrupts are not touched), set system-oriented interrupts to the required values, ground (set to $77777777_8$) all queue pointers, and then enable the interrupts and branch to the dispatcher (TASKER).

NOTE: If the transient storage area locations or sizes are modified, the appropriate COLD START routine locations must also be modified.

V. Dispatcher:

We now come to the heart of OS910, the dispatcher (TASKER). Three queues are managed by the dispatcher, the interrupt and regular write queues, and the task queue. The interrupt write queue has the highest priority, with the regular send (write) queue next in priority, leaving the task queue lowest in priority. All three of these queues use the same three word node buffer. Queuing of tasks will be covered later.

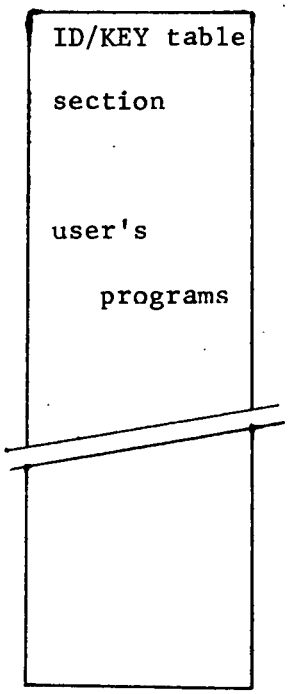
As noted earlier, the format of a queue node is:




(if this is the last node on the queue, the "address of next node" field is set to -1)

For the write (send) queues, a buffer node or buffer node list is expected to be referenced by the parameter list field. The entry point field is assumed to be the routine to perform the I/O, either SEND72 (for a 72 bit send), or DMASND (for a DMA transfer). The ID field is expected to be that of the user issuing the transfer.

With all tasks, if the user has been set inactive, then the task is not dispatched. To determine activity, a bit is kept in a preamble to each user's code. The format of a user's task is:



The ID/KEY table, which is supplied by the user, must have the form:

| BIT | 0——8 | 9 | 10————23 |
|---|---|---|---|
| WORD 1 | ID # | a/i | # KEY ENTRIES |
| 2 | ADDRESS OF INTERRUPT$_1$ | | |
| 3 | ADDRESS OF INTERRUPT$_1$ HANDLER | | |
| 4 | ADDRESS OF INTERRUPT$_2$ | | |
| 5 | ADDRESS OF INTERRUPT$_2$ HANDLER | | |
| 6 | OPEN ENTRY POINT | | |
| 7 | CLOSE ENTRY POINT | | |
| 8 | KEY 1 BUFFER LOCATION | | |
| 9 | KEY 1 ENTRY POINT | | |
| . | . | | |
| . | . | | |
| . | . | | |
| 2N+6 | KEY N BUFFER LOCATION | | |
| 2N+7 | KEY N ENTRY POINT | | |

Taking each section in order, we have:

1. WORD 1:

   bits 0-8: the ID number (in the range 511-537) assigned by the system staff.

   bit 9: the a/i (active/inactive) flag for the user. If this bit is set to zero, then the user is assumed to be in the active (running) state.

bits 10-23: maximum key value used.

2. WORDS 2 and 3:

   Addresses of the user's first interrupt location and the user interrupt handler for it. If the user does not have an interrupt for his or her experiment, then put:

```
WORD2    OCT    77777777
WORD3    BRM    2.
```

3. WORDS 4 and 5:

   Addresses of the user's second interrupt location and the user interrupt handler for it. If the user does not have an interrupt or does not have two interrupts for his or her experiment, then put:

```
WORD4    OCT    77777777
WORD5    BRM    2.
```

4. WORD 6:

   User's OPEN routine entry point.

5. WORD 7:

   User's CLOSE routine entry point.

6. THE ENTRIES:

   These are pairs of data buffer location and entry point associated with each key used in the PDP-15 ALECS program. To conserve space in the SDS-910 the user is asked to assign the keys in numerical order.

This table is used in the following manner.

1. When the task that resides in the PDP-15 performs an OPEN for the EXPERIMENT file, the a/i bit is set to zero for that user. The user's interrupt(s) is(are) connected through the system and is(are) set active. Finally, the user's open routine is scheduled for execution.

2. When a task is to be dispatched, the a/i bit is checked. If the bit is set to one (inactive) the task is flushed from the system.

3. When information is sent to the SDS-910 for the user, the key, say i, associated with the data, is used to find the address of that area of the user's storage which will contain the data (the storage area must be large enough to contain both control information i.e., two words of storage for a read return, and the data). Associated with the buffer address is a service routine entry point address. Using the address of the (user supplied) buffer, the entry point address for the service routine, and the user's ID value, a task queue node is created, thus scheduling the execution of the user's service routine.

4. When the user incurs a real-time (RT) error (by using an invalid interface (EOM) address for his experiment) the RT error servicing routine resets the a/i bit, sends information to the PDP-15 that the user has incurred an RT error and then allows the user to finish execution of the task.

5. When the user's code in the PDP-15 performs a CLOSE of the EXPERIMENT file, a record is transmitted to the SDS-910 to indicate this. The SDS-910 system then resets the a/i bit to inactive, turns off the interrupts associated with this user, and schedules the user's CLOSE routine.

The ID/KEY table is user generated. An example follows for a ID number of 524, and a total of two keys.

```
OCT     52440002
OCT     242         ADDRESS of INTERRUPT1
BRM     CHAR        ADDRESS of INTERRUPT1 HANDLER
OCT     243         ADDRESS of INTERRUPT2
BRM     CHAR        ADDRESS of INTERRUPT2 HANDLER
HLT     OPEN        OPEN ENTRY POINT
HLT     CLOSE       CLOSE ENTRY POINT
HLT     BLOCK       KEY ONE BUFFER AREA
HLT     READ        KEY ONE ENTRY POINT
HLT     DATA        KEY TWO BUFFER AREA
HLT     SEND        KEY TWO ENTRY POINT
```

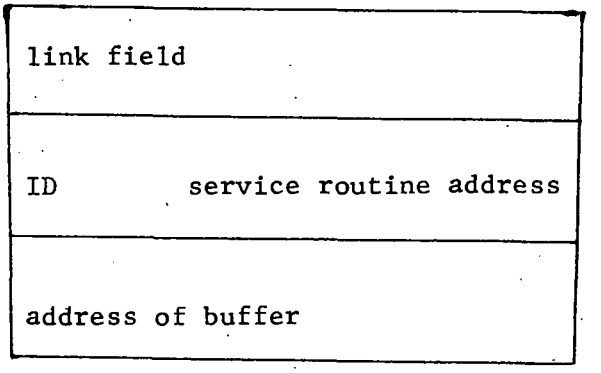
The first word, declared OCT 52440002, has the field values: bits 0-8 set to $524_8$ i.e., the user ID number, bit 9 set to 1 (inactive)

and bits 10-23 set to 00002, the total number of keys. The routine names have been arbitrarily chosen. The declarations using

```
HLT     name
```

create a word containing the storage address for 'name'. The buffer area will have to be large enough to contain the maximum amount of data sent, plus whatever control information must be kept (see the I/O routines).

When information is sent from the SDS-910 to the PDP-15, it is placed in a dynamically acquired buffer area, and a queue entry is made. The queue entry is of the form:




If this is a 72 bit transfer, then the service routine address is to the SEND72 routine. Otherwise, it is to the DMASND routine. The format

of a dynamic buffer is:

a) for a 72 bit transfer:

| | |
|---|---|
| word 1 | -1 (grounded) |
| 2 | |
| 3 | 72 bits of control information and data |
| 4 | |
| 5 | 72 bits of unspecified data |
| 6 | |
| 7 | |

b) for a DMA transfer:

1. First buffer node:

| | 0 | 1 | 8 | 9 | 10 | 23 |
|---|---|---|---|---|---|---|
| word 1 | e | # transfers | | 0 | link | |
| 2 | 72 bits of data to set up DMA transfer | | | | | |
| 3 | | | | | | |
| 4 | | | | | | |
| 5 | First 72 bits of DMA data | | | | | |
| 6 | | | | | | |
| 7 | | | | | | |

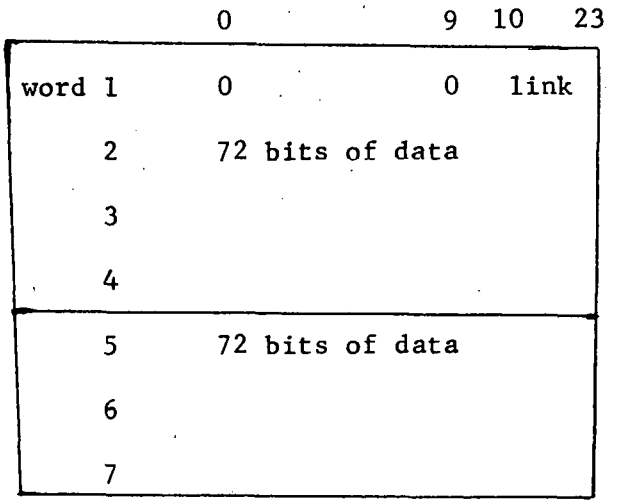
where:

e - extent bit. If this is the last node on the list, then e=1, else e=0.

# transfers - this is the number of 72 bit transfers to be done, excluding the 72 bits of set-up data.

link - this is the address of the next buffer node, or undefined if this is the last node on the list.

2. Intermediate nodes (full):




where: link is the address of the next buffer node.

3. Final node:

| | 0 1 | 9 10 | 23 |
|---|---|---|---|
| word 1 | 1 | 0 | undefined |
| 2 | 72 bits of data | | |
| 3 | | | |
| 4 | | | |
| 5 | either 72 bits of data, or undefined | | |
| 6 | | | |
| 7 | | | |

Note that bit 0 (the extent bit) is set to one (1).

Each queue has the same form, and each is treated the same. A two word queue pointer set exists for each queue, consisting of a head and tail pointer. If the queue is not empty (head, tail ≠ -1), then the first node on the queue is tested for being from an active task. If the task is active, then the task is dispatched. Otherwise, the task is cleared off. This latter operation amounts to releasing all buffer space, for an I/O queue entry, and freeing the queue node.

The activation of a task is as follows:

a) the entry point address is extracted from the second word of the queue node;

b) the A register is loaded with the parameter list address field contents;

c) a BRM is done relative to the entry point pointer.

Task deactivation is very simple. For an I/O queue node, the buffer list is assumed to be already freed. Otherwise, the A register is set to the address of the node. Then, the instruction:

```
BRM*  FRENDE
```

is executed. This routine will chain the queue node back onto the queue node free storage list, and return.

The routine ACTIVE tests the a/i bit of the indicated task. On entrance, the X register must point to the node to be scheduled. The ID value is then extracted. The associated ID/KEY tables a/i bit is checked. If set to 1, then the return address is incremented before returning. If a/i = 0, then the normal return is performed. The system ID value of $510_8$ which maps into location $300_8$ points at the zero location (ZERO) for its ID/KEY table. This makes the system always active.

VI. Clocks:

The clocks are used for time-dependent task scheduling. Tasks may request scheduling on the queue for one (or more) of the clocks. The task chain looks and is used in the same manner as the dispatcher chains. Also, tasks are subject to the same tests and conditions. There are three clock signals which are supplied by the interface. The three clocks have periods of 12.8 ms. (fast clock), 102.4 ms. (medium clock), and 1.6384 second (slow clock), and are normally

employed to provide a set time delay between operations in the SDS-910. Shaft movement, character transmission, etc., are some areas where one might employ the clocks. The slow clock tick is also used to increment the TIME variable at location 4. Therefore, 'gross' period timing can be done by the user.

A user may schedule a task onto one of the three clock queues by using one of three POPs (SKDFST, SKDMED, SKDSLO). When a task is on a clock queue, it is executed the next time that clock "ticks". The action taken when a clock "ticks" is:

1. If the queue for that clock is empty, then the interrupt level is cleared and execution resumes where it was interrupted.

2. If there are nodes present, then the contents of the head pointer for the respective clock queue are copied to the area referenced by either FCLKQ, MCLKQ, or SCLKQ (fast, medium, or slow clocks), and the queues' head/tail pointer are both set to -1. Then,

3. Each task on the queue now referenced by FCLKQ, MCLKQ, or SCLKQ is, if active, invoked in turn, then deleted from the queue. Meanwhile, new tasks may be scheduled on this or any other queue.

4. Finally, the interrupt level is cleared and execution resumes when it was interrupted.

This task dispatching operates with the interrupt system enabled, thus allowing higher priority interrupts to hit at any time, but stopping the same level interrupt, or any lower one. When all the nodes on the queue (referenced by FCLKQ, MCLKQ, or SCLKQ) have been treated, the interrupt level is cleared and execution resumes where the clock "tick" interrupted.

NOTE: The clocks are considered the only 'real-time' experiment and therefore are high in priority. Only the RT error interrupt is higher.

VII. Task Scheduling:

Using OS910, the facility to cause the scheduling of tasks exists. A user may schedule a normal task (managed by the dispatcher), or a clock task (managed by the respective clock queue manager).

There are four POPs used for scheduling, SKDFST, SKDMED, SKDSLO, and SKDTSK. The parameters to each of these POPs is the same:

1. the A register contains the entry point argument,
2. the B register contains the parameter list pointer.

Using these parameters, the routine MAKNDE constructs a queue node and sets the X register to the address of the newly constructed node.

The MAKNDE routine employs the A and B register, and CCBP, to fill in the node fields. An attempt is first made to allocate a queue node from the free storage list. If the list was empty at the time of the request, the X register will have been set to -1 ($77777777_8$ octal). and the MAKNDE routine will halt. Otherwise, the link field (first word) of the newly allocated node is set to -1 (to "ground" it); word two is set to the pair (user ID, entry point address) arranged as:

ID - bits 0-8
Entry Point address - 10-23
Bit 9 is set to zero

The third word is set to the contents of the B register (user parameter list pointer). When MAKNDE returns to the calling routine, the node may be added to the required queue list. Normally, the addition will be to the tail of the queue.

VIII. I/O Scheduling:

There are a set of four POPs to queue up data transmission to the PDP-15. Three POPs, SNDFIX, SNDFLT, and SENDMA, are used to send data back on a read return (see communication modes Section XI). The fourth, SNDINT, effects the transmission of an ONCODE resulting from an experiment interrupt (see the ALECS Language Reference Manual, IS-3339).

Each POP will construct a queue node, and as many buffer nodes as required for the amount of data. For all but SENDMA, this will be one buffer node. In any case, if a node (buffer on queue) is required and none are available, the POP will execute a PAUSE instruction and wait for a node to free-up. This condition, similarly raised in the task scheduling POPs, is indicative of a condition of too few nodes on that free storage list. Thus, the system programmer response must be to increase the transient storage area for that type node.

The SNDFIX POP requires an operand to be present. The data referenced by the operand must be the three words:

Word 1: Length of transfer
Word 2: "Key Loc. in CAL" value
Word 3: Data value

The "Key Loc. in CAL" value is passed by the PDP-15 on the initial set-up. This value will be put into the user's desired location by the system. The above data, along with the current CCBP value will be used to construct a buffer node. Then, a queue node will be constructed

using the address of the buffer node (whose first word has been set to -1) as the parameter list field value, and the entry point to the I/O routine SEND72 and the CCBP value as the entry point/ID field values. The link word of the node will be set to -1, and the node chained onto the tail of the send queue.

SNDFLT is similar to SNDFIX except that the area referenced by the POPs' operand is the four words of:

Word 1: Length of transfer
Word 2: "Key Loc. in CAL" value
Word 3: Data value (low order Mantissa and exponent)
Word 4: Data value (high order Mantissa)

The same send queue node format is used, and the buffer is constructed in the same fashion.

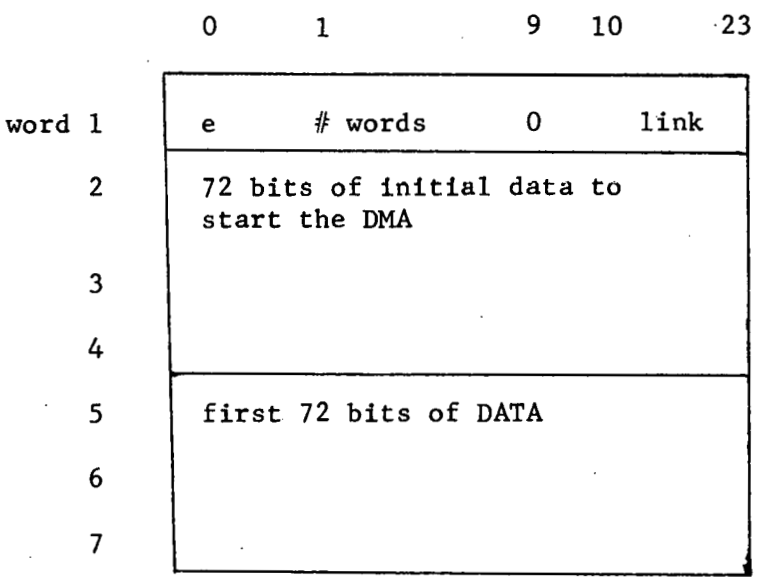
SENDMA will construct a (possible) list of buffer nodes. The length of a buffer node is set at seven (7) 24 bit words. This allows for a link word (word 1), and two 72 bit data sections (words 2-4, and 5-7). This not only makes filling the buffer area easier, but also makes the transmission of the data easier, as boundary problems won't exist (the SDS-910 must, because of the interface structure, transmit data in groups of 72 bits).

The operand of the SENDMA POP references an area arranged as:

```
# data entries
"Key Loc. in CAL" value
data value 1
      .
      .
      .
data value N
```

The # data entries field gives the number of data entries to be transmitted back to the PDP-15. The buffer nodes are set up as:

1. First buffer node:




Note: a) if this is the only buffer node on the list, e=1; otherwise, e=0.

b) the # words field gives the number of 24 bit words to be transmitted in the DMA transfer, not counting the first 72 bit setup transmission that prepares the PDP-15 for the transfer. Zeroes are added onto the right of the data as required, to pad out to 72 bits in a transmission. Thus, if a DMA transmission were to be for only 54 bits of data, 18 bits, all zero, would be appended onto the right of the actual data to get to a 72 bit transmission.

Other than the possibility of having a list of buffers, and that the service routine will be DMASND rather than SEND72, the DMA node creation is the same as the SNDFIX and SNDFLT POPs.

NOTE: It takes four (4) words from the SDS-910 (24 bits each), to compress down to the 72 bits of data (4 PDP-15 words).

The fourth POP in this series will construct an interrupt send queue (INTQUE) entry, and an associated buffer entry. This POP, SNDINT, assumes that the A register contains the ONCODE to be returned to the PDP-15. A dummy operand value must be present on the POP, even though it isn't used. The queue node entry point field is set to SEND72, the ID field is set to the current CCBP value, and the parameter list field points to a buffer node. The buffer node contains -1 as the first word entry. Then, the 72 bit field includes the ONCODE value and the system communication information.

IX. Real-Time Error Management:

A real-time (RT) error is caused by an I/O operation that incurs an addressing error using the interface to either the PDP-15 or an experiment. When an RT error occurs:

1. The interface "E" register is read and stored.
2. An interrupt queue entry with the ONCODE set to zero is setup to be sent to the PDP-15.
3. The user is allowed to continue execution at the statement following the RT-error-causing instruction.

The PDP-15 upon receipt of the RT error indication will shutdown the user. In doing so, the PDP-15 system sends a CLOSE for the user to the SDS-910. Upon receipt of this CLOSE the user interrupts are

idled. The user is then scheduled for execution to do whatever shutdown is required. Then a task is scheduled to turn-off the user's active/inactive bit.

It should be noted that:

1. The queuing priority is (from highest to lowest): interrupt queue, send queue, and task queue;

2. To guard against scheduling problems, it is better to queue up the shutdown action than to do the work directly;

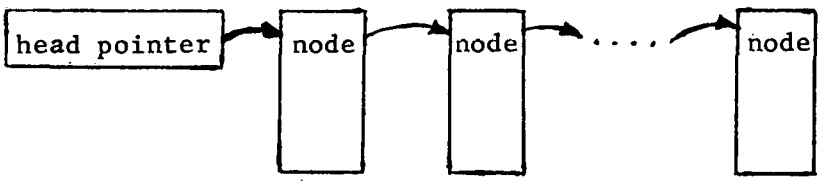
3. We must send the RT error generated ONCODE before the routine is shutdown; otherwise, the a/i bit = 1, and the send will be ignored.

Along with the above queuing of information, the RT error routine clears the interrupt level and returns to the interrupted program. As the interrupted program, which incurred the RT error, will have no global side effects, it appears to be safe to let it complete on its own, rather than trying to shut the user down immediately and return to the task previous to the user causing the RT error. This return is effected by executing the instruction: BRU* RTERR, where RTERR is the entry point name of the RT error interrupt servicing routine.

If a user generates a second RT error the system will again go to the RT error handling routine and proceed through the above procedure. Eventually, the PDP-15 will send a shutdown for the user which will cause the SDS-910 system to shutdown this user.

X. Transient Storage Area Management:

There are two transient storage areas in OS910. These provide nodes for queuing and buffering. Queue nodes are three words long, buffer nodes are seven. Each transient storage area is arranged as a simple forward linked list, as:




Note that the value -1 (octal 77777777) is used to denote a "grounded" node; i.e., the last node on a list.

There are two routines that directly affect the free storage list (transient storage area). The routine ALLOC will allocate a node from the free storage list referenced by the X register. On return, the X register will either be set to -1 (if the list was empty) or reference a node from the free storage list.

The FREE routine will return the node referenced by the A register to the free storage list referenced by the X register. No modifications are made to the registers.

Neither ALLOC nor FREE is used directly. To allocate a queue or buffer node, the user employs either GETNDE or GETBUF, respectively. On returning, these routines have the X register set to -1, if the required

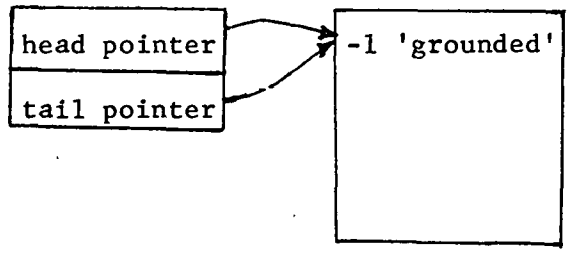
free storage list is empty, or to the address of a node, if there is at least one node left.

To free a node (return it to the required free storage list), the user sets the A register to the address of the surplus node, and executes either DALNDE or DALBUF to deallocate a queue or buffer node, respectively.

XI. Interrupt Buffer Queue Storage Area Management:

This area is used by the SAVE and RESTOR routines and the PAUSE POP. It is used to retain the working environment of the interrupted user. Initially, the queue is set up by the COLD START routine (COLD) as a simple forward linked list of ten word nodes.

Associated with the buffer queue is a set of pointers (BUFQUE) which are 'head' and 'tail' pointers for the nodes. When the buffer queue is in use because of an interrupt or a user or the system using the PAUSE POP, the queue looks like:




The PAUSE POP and SAVE routine add nodes to the queue. The RESTOR routine takes nodes off the queue. The queue is set up as follows:

| WORD | CONTENTS |
|---|---|
| 1 | link pointer |
| 2 | "A" register |
| 3 | "B" register |
| 4 | "X" register |
| 5 | location 0 |
| 6 | location 1 |
| 7 | system temporary QA |
| 8 | system temporary QB |
| 9 | system temporary QC |
| 10 | user ID CCBP. |

The SAVE and RESTOR routines are called by the READ routine, RTERR routine, and all system user-interrupt-handlers. The PAUSE POP is used extensively by the system to wait for nodes to free-up when the queues are full. The PAUSE POP can also be invoked by users.

As in the transient storage area management, the ALLOC and FREE routines are used to allocate and free the nodes in the buffer space.

XII. Interface Communication Protocol:

The SDS-910/PDP-15 interface protocol developed by D. Anderson, J. Campbell, B. Carter, W. Thomas, B. Helland, and M. Conley was to establish the most feasible communications mode possible between the two computers. The SDS-910 uses a 24 bit word and the DEC PDP-15 uses an 18 bit word. Thus, to minimize waste and for other hardware consideration, a 72 bit transfer was decided upon. The following

description of the types of transfers were detailed by B. Helland. They are in terms of the 18 bit word format, i.e., using four words to create a block of data.

Each transfer, whether it is a DMA transfer or not, starts with some status and identifying information. Bits 0-8 of the first word constitute the status portion. The explanation of this part is:

1. On a DEC PDP-15 to SDS-910 transfer:

   bit: 0 = 1 : shutdown indication
   1 = 1 : read operation
   2 = 1 : write operation
   3 = 1 : DMA transfer-only in case of write
   4 = 1 : 2 word write (bit 2 = 1)
   5 = 1 : 1 word write (bit 2 = 1)
   6 = 1 : system task send

2. On a SDS-910 to DEC PDP-15 transfer:

   bit: 0 = 1 : interrupt
   1 = 1 : four word read from SDS-910
   2 = 1 : DMA read from SDS-910
   3 = 1 : RT error, only if bit 0 = 1
   4 = 1 : if bit 3 = 1, then this indicates software problem
   5 = 1 : system task response
   6 : unused

Also, bits 7,8 of the first word are currently unused.

Bits 9-17 give the ID of the user/system doing the I/O operation. The ID value, in the range of 510-537 (octal) is a unique code assigned to a user by system personnel.

The buffer forms for the different I/O operations are:

1. On a transmission from the PDP-15 to the SDS-910:

   a) Write

i. One word

word 1: bits: 0-5: 001001
6-8: unused
9-17: ID value

word 2: Key value

word 3: Data value

word 4: Unspecified

ii. Two words:

word 1: bits: 0-5: 001010
6-8: unused
9-17: ID value

word 2: Key value

word 3: Data value

word 4: Data value

iii. DMA transfer:

word 1: bits: 0-5: 001100
6-8: unused
9-17: ID value

word 2: Key value

word 3: Length of the DMA transfer

word 4: Unused

Then, the data will follow

iv. SHUTDOWN:

word 1: bits: 0-5: 100000
6-8: unused
9-17: ID value

words 2-4: Unspecified

b) Read:

A read is generated by a read of an experiment file from the PDP-15. It generates a read return transmission that asks the SDS-910 to transmit data back to the PDP-15. The transfer form is:

word 1: bits: 0-5: 010000
6-8: unused
9-17: ID value

word 2: Key value

word 3: Length value (in terms of 18 bit words)

word 4: "key location in CAL" value. This field is used for a check on returning to the PDP-15.

This read request is for a one or two word transfer, or a DMA transfer.

2. On a transmission from the SDS-910 to the PDP-15:

a) Read Return:

i. One word of data:

word 1: bits: 0-5: 010000
6-8: unused
9-17: ID value

word 2: Address of key location in CAL

word 3: Data value

word 4: Unspecified

ii. Two words of data:

word 1: bits: 0-5: 010000
6-8: unused
9-17: ID value

word 2: Address of key location in CAL

word 3: Data value

word 4: Data value

iii. DMA transfer:

word 1: bits: 0-5: 001000
6-8: unused
9-17: ID value

word 2: Address of key location in CAL

words 3, 4: Unspecified

b) Interrupt transmission:

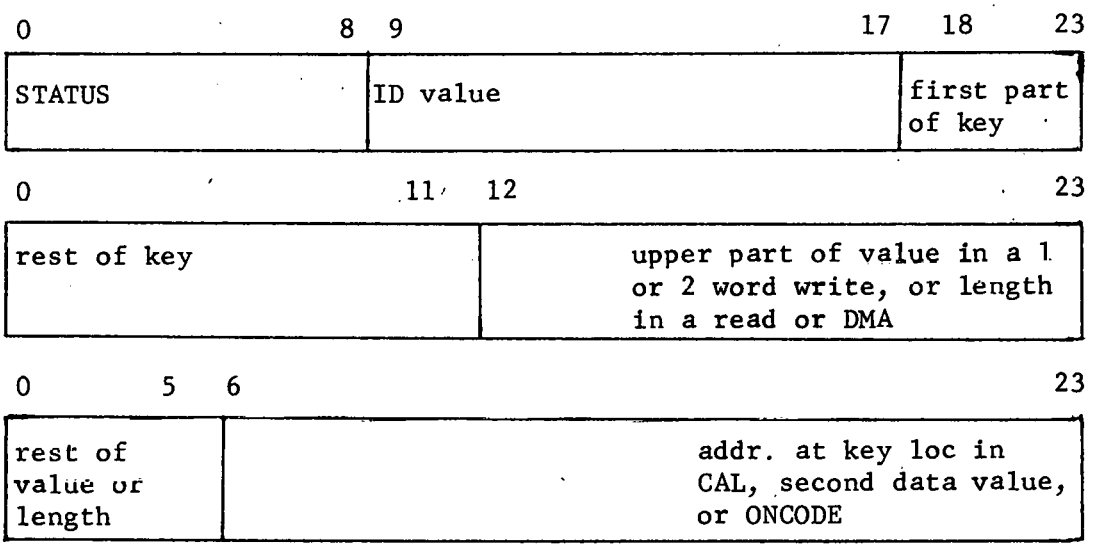
This results from an interrupt generating an interrupt signal containing data required by the PDP-15, or from incurring a real-time error (in which case the ONCODE is forced to be zero).

word 1: bits: 0-5: 100000
6-8: unused
9-17: ID value

words 2, 3: Unspecified

word 4: ONCODE value

The layout of these first transfers is, in detail, and in terms of the SDS-910 word format:



XIII. SDS-910/PDP-15 Interface Protocol:

This section will discuss the machine interface from the point of view of the SDS-910. The PDP-15 side is discussed in another manual. The task of the interface is to allow the exchange of data between the two computers. It does so by allowing 72 bit at a time transfers. This was determined to be the most efficient method of transfer for many reasons. For a more complete discussion see B. Carter's M.S. thesis.

Both computers have access to certain registers and indicators in the interface. To communicate with the interface, and allow data transmission, a SDS-910 system employs the three instruction set combinations of:

1. To read from a register, buffer, etc.:

```
EOM     30(A_XY B_Z)
PIN     loc.
```

2. To write to a register, buffer, etc.:

```
EOM     32(A_XY B_Z)
POT     loc.
```

3. To test an indicator:

```
EOM     36(A_XY B_Z)
SKS     300(mask)
```

The $A_{XY}B_Z$ value is an experiment/machine interface address. The mask is a six bit value. It is set to test the control register, buffer full, etc. Table 3 gives the indicator addresses, functions, and what operations are allowed.

At present, the only times the computer to computer interface will cause an interrupt of the SDS-910 is when an RT error is incurred, and when the PDP-15 wants the SDS-910 to read some data. At all other times, interrupts are not generated for the SDS-910, by said interface.

XIV. SDS-910 To PDP-15 Transmission Routines:

There are two programs in OS910 to handle data transmission from the SDS-910 to the PDP-15. They are SEND72 and DMASND. Both are used as tasks in the system, and are invoked because of having been placed onto either the interrupt or regular send queues (INTQUE or SNDQUE).

SEND72 will only transmit 72 bits of information across the interface. It is employed by all interrupt transmissions, and by one and two word read returns.

DMASND will be executed only when a regular send queue entry must perform a DMA-type read return. Any amount of data desired may be sent. The system also may employ DMASND for other purposes, for example, as a debugging tool.

Both SEND72 and DMASND will have their data presented to them as a (possible) list of buffer nodes. After all the information from a buffer node has been sent, the node must be freed. Each I/O routine encountering buffer nodes does this management.

XV. READ Routine:

This routine handles all transfer from the PDP-15 to the SDS-910. These transfers are one of the following:

1. System send.
2. Read request
   a) one word
   b) two words
   c) DMA.
3. Write of information to the SDS-910
   a) one word
   b) two words
   c) DMA.

   Note: A one word transfer (either read request or write) is one FIXED or BIT data. A two word transfer is one FLOAT data. A DMA transfer is an array of FIXED, BIT, FLOAT, or CHARACTER data or a character string or a structure of data items.

4. OPEN of experiment file
   (i.e., awaken user's programs in the SDS-910).
5. CLOSE
   (i.e., shutdown all user's programs in the SDS-910).
6. Handle the loading of tasks into the SDS-910 by connecting the SDS-910 operating system to the SDS-910 loader.

On a system send, the status word is changed to show it is a SDS-910 to PDP-15 transfer and the key and "Key in Cal Location" are placed in the appropriate positions for the transfer. The information is sent back immediately so the PDP-15 can see if the interface is still awake and reliable.

With a read request the "Key in Cal Location", which the PDP-15 system needs back so it can match up the transfer with the right user, and the length of transfer desired are put into the user's data area as specified in the ID/Key tables. The first word put into the user data area is the length of transfer desired. The second is the "Key in Cal Location". The read routine will then schedule the user's task for execution.

A write of information moves the data from the PDP-15 into the user specified data area and then schedules that user's service routine associated with that particular KEY. The three SDS-910 interface data words @ PDP-15 words) are translated into four SDS-910 words. If the fifth bit of the new SDS-910 word (the sign bit in the PDP-15 representation) is set, the system will propagate this to the left so the number will be the same in the SDS-910. If the user is not transferring fixed numbers he must make the necessary adjustment to his data for the above modification.

On an OPEN the user's interrupt vector(s) is(are) routed through the system user interrupt handlers. The user active/inactive bit is set active. The user is then scheduled for execution to do whatever initialization is necessary for his/her experiment.

On a CLOSE the user's interrupt vector(s) is(are) idled. The user is then scheduled for execution to do whatever shutdown is required. Then a task is scheduled to turn-off the user's active/inactive bit.

When a user wants to load a program into the SDS-910 from the PDP-15 the READ routine in OS910 passes the length of the transfer to the SDS-910 loader and then passes control to the loader to load the program.

## XVI. FLOATING POINT POPS

All of the floating point routines, except the conversion routines, have been taken from the SDS-910 POPs Manual and modified to work with OS910. The ones which are available are addition (FSA), subtraction (FSS), multiplication (FSM), division (FSD), and negation (FSN). Two other POPs fixed to float conversion (FFL) and float to fixed conversion (FFI) (for SDS-910 numbers only) are available. All of the above POPs are for single precision floating point numbers.

The conversion routines assume the value to be converted is in the A register (for fixed values) or the A and B registers (for float values) and return with the value converted in the A or A and B registers.

The A and B registers should be loaded with the value the user wants negated (in the case of FSN) or with the first operand, the A register should contain the mantissa and the B register should contain the exponent. The address field specified in the POP instruction should reference the second operand field (EXPONENT, MANTISSA). The result of the operation will be in the A and B registers.

EXAMPLE A/B:

```
        LDB     A
        LDA     A+1
        FSD     B
         .
         .
         .
         .
         .
A       OCT             /EXPONENT for A
        OCT             /MANTISSA for A
B       OCT             /EXPONENT for B
        OCT             /MANTISSA for B.
```

The result of the division will be in the A and B registers when control is given back to the user's program.

XVII. FIXED POINT POPS

All of the fixed point routines, have been taken from the SDS-910 POPS manual and modified to work with OS910. Multiplication (MUL), Division (DIV) and a branching routine, skip if equal (SKE) are available to the user.

The first operand should be in the A register and the second operand in the address field of the POP.

EXAMPLE A/B:

```
        LDA     A
        DIV     B
         .
         .
         .
         .
A       OCT             /FIXED VALUE for A
B       OCT             /FIXED VALUE for B.
```

The result is in the A register when control is given back to the user.

XVIII. PAUSE POP

This routine is designed to allow a user to cause a delay in execution. The address field will allow the user to return to some other point in his or her code when control is restored to the program.

EXAMPLE:

```
        SKN     VALUE
        BRU     *+2
        PAUSE   *-2
         .
         .
         .
```

This example will cause a delay in execution until some other code (presumably in an interrupt routine) sets VALUE negative. PAUSE is a variable delay depending on the number of users in the system at the time.

The user is asked to use the PAUSE POP instead of an infinite loop so other users of the system can be executing codes while the user is waiting for the desired event to happen. NOTE: The PAUSE POP cannot be used in an interrupt routine.

XIX. DMA24 POP

This POP can be used to transfer SDS-910 words to the PDP-15 directly from the user's buffer area. This POP should be used when a large amount of data is to be transferred to the PDP-15. Proper invocation is

```
DMA24     DMABLK
```

where DMABLK is a three word area set-up as

1. number of data words to be transferred,
2. "Key Location in Cal" value,
3. address of the start of the data.

TABLE 1

OS910 POPS

| LOCATION | POP | PURPOSE |
|---|---|---|
| 100 | SKDFST | Schedule a task on the fast clock. |
| 101 | SKDMED | Schedule a task on the medium clock. |
| 102 | SKDSLO | Schedule a task on the slow clock. |
| 103 | SKDTSK | Schedule a task for execution. |
| 104 | SNDFIX | Send PDP-15 a fixed value on a read return. |
| 105 | SNDFLT | Send PDP-15 a float value on a read return. |
| 106 | SENDMA | Send PDP-15 a DMA transfer on a read return. |
| 107 | SNDINT | Send PDP-15 an interrupt. |
| 110 | FFL | Fixed to float conversion. |
| 111 | DIV | Fixed divide. |
| 112 | MUL | Fixed multiply. |
| 113 | FSN | Float negate. |
| 114 | FSD | Float divide. |
| 115 | FSM | Float multiply. |
| 116 | FSS | Float subtract. |
| 117 | FSA | Float add. |
| 120 | FFI | Float to fixed conversion. |
| 121 | PAUSE | Create pause in execution. |
| 122 | SKE | Skip on equal, |
| 123 | PON | Turn-on user interrupt. |
| 124 | POFF | Turn-off user interrupt. |
| 125 | DMA24 | Sends 24-bit information to PDP-15 directly from user's buffer area. |

TABLE 2

INTERRUPT VECTOR ASSIGNMENT AT AMES LABORATORY RESEARCH REACTOR

| OCTAL ADDRESS | DEFINITION OF INTERRUPT LEVEL |
|---|---|
| 200 | SPECIAL 1.6 SECOND CLOCK |
| 201 | AVAILABLE SPECIAL INTERRUPT LOCATION |
| 202 | HILGER WATTS THETA DECREMENT |
| 203 | HILGER WATTS THETA INCREMENT |
| 204 | HILGER WATTS OMEGA DECREMENT |
| 205 | HILGER WATTS OMEGA INCREMENT |
| 206 | HILGER WATTS PHI DECREMENT |
| 207 | HILGER WATTS PHI INCREMENT |
| 210 | HILGER WATTS CHI DECREMENT |
| 211 | HILGER WATTS CHI INCREMENT |
| 212 | DATEX THETA INCREMENT |
| 213 | DATEX THETA DECREMENT |
| 214 | DATEX OMEGA INCREMENT |
| 215 | DATEX OMEGA DECREMENT |
| 216 | DATEX PHI INCREMENT |
| 217 | DATEX PHI DECREMENT |
| 220 | DATEX CHI INCREMENT |
| 221 | DATEX CHI DECREMENT |
| 222 | AVAILABLE SPECIAL INTERRUPT LOCATION |
| 223 | RT ERROR |

TABLE 2 (Continued)

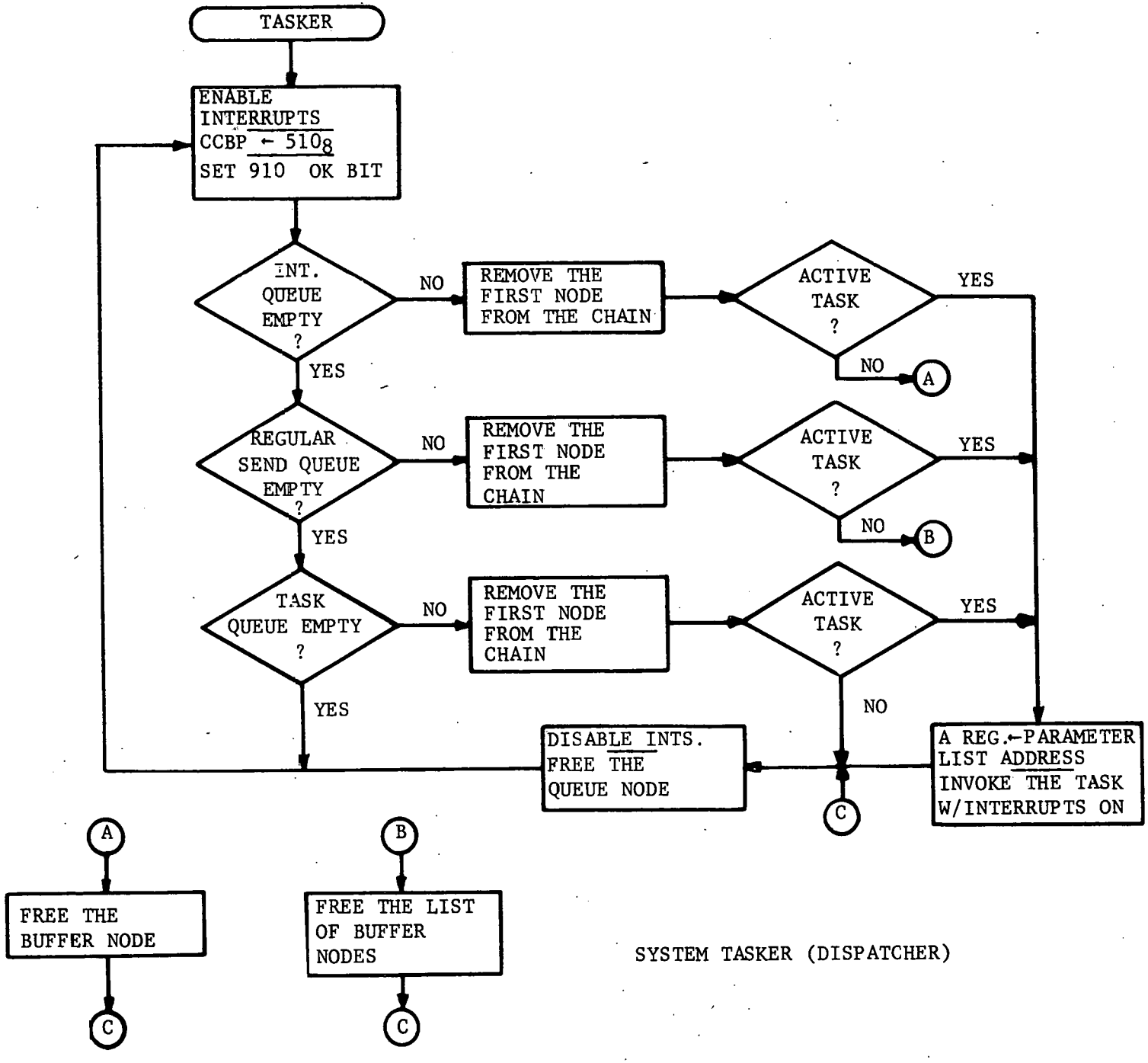
| OCTAL ADDRESS | DEFINITION OF INTERRUPT LEVEL |
|---|---|
| 224 | FAST CLOCK (12.8 MS.) |
| 225 | INTERMEDIATE CLOCK (102 MS.) |
| 226 | SLOW CLOCK (1.6 SECONDS) |
| 227 | PDP-15 TO SDS-910 INTERFACE |
| 230 | HILGER WATTS LIMIT |
| 231 | HILGER WATTS COUNT |
| 232 | DATEX LIMIT |
| 233 | DATEX COUNT |
| 234 | ISOTOPE SEPARATOR |
| 235 | ISOTOPE SEPARATOR |
| 236 | THUMB CHARACTER |
| 237 | THUMB BLOCK |
| 240 | FINGER CHARACTER |
| 241 | FINGER BLOCK |
| 242 | TRIPLE AXIS |
| 243 | PLOTTER |
| 244-277 | AVAILABLE REGULAR INTERRUPTS |

TABLE 3

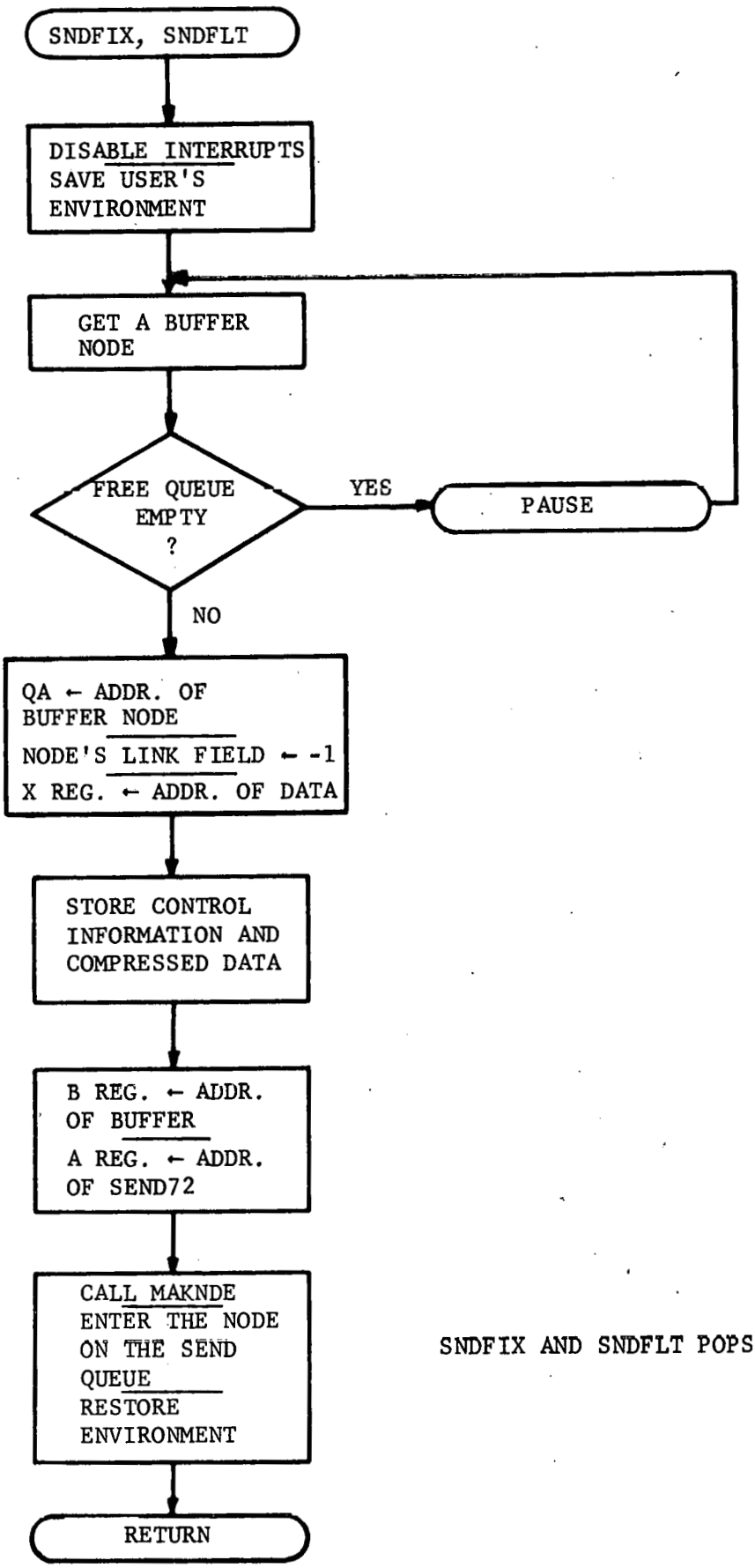
SDS-910 ADDRESS ASSIGNMENTS

| FLIP FLOP(S) | Axy | Bz | INSTRUCTIONS THAT APPLY | | |
|---|---|---|---|---|---|
| CLEAR 910 INTERRUPT | 30 | 0 | | POT | |
| BUFFER FULL | 30 | 0 | | | SKS |
| DATA WORD 1 | 30 | 1 | PIN | POT | |
| DATA WORD 2 | 30 | 2 | PIN | POT | |
| DATA WORD 3 | 30 | 3 | PIN | POT | |
| REQUEST | 30 | 4 | | POT | |
| BUSY | 30 | 5 | | POT | SKS |
| 15 INTERRUPT | 30 | 5 | PIN | | |
| 910 OK | 30 | 6 | | POT | |
| CONTROL WORD | 30 | 7 | PIN | POT | |

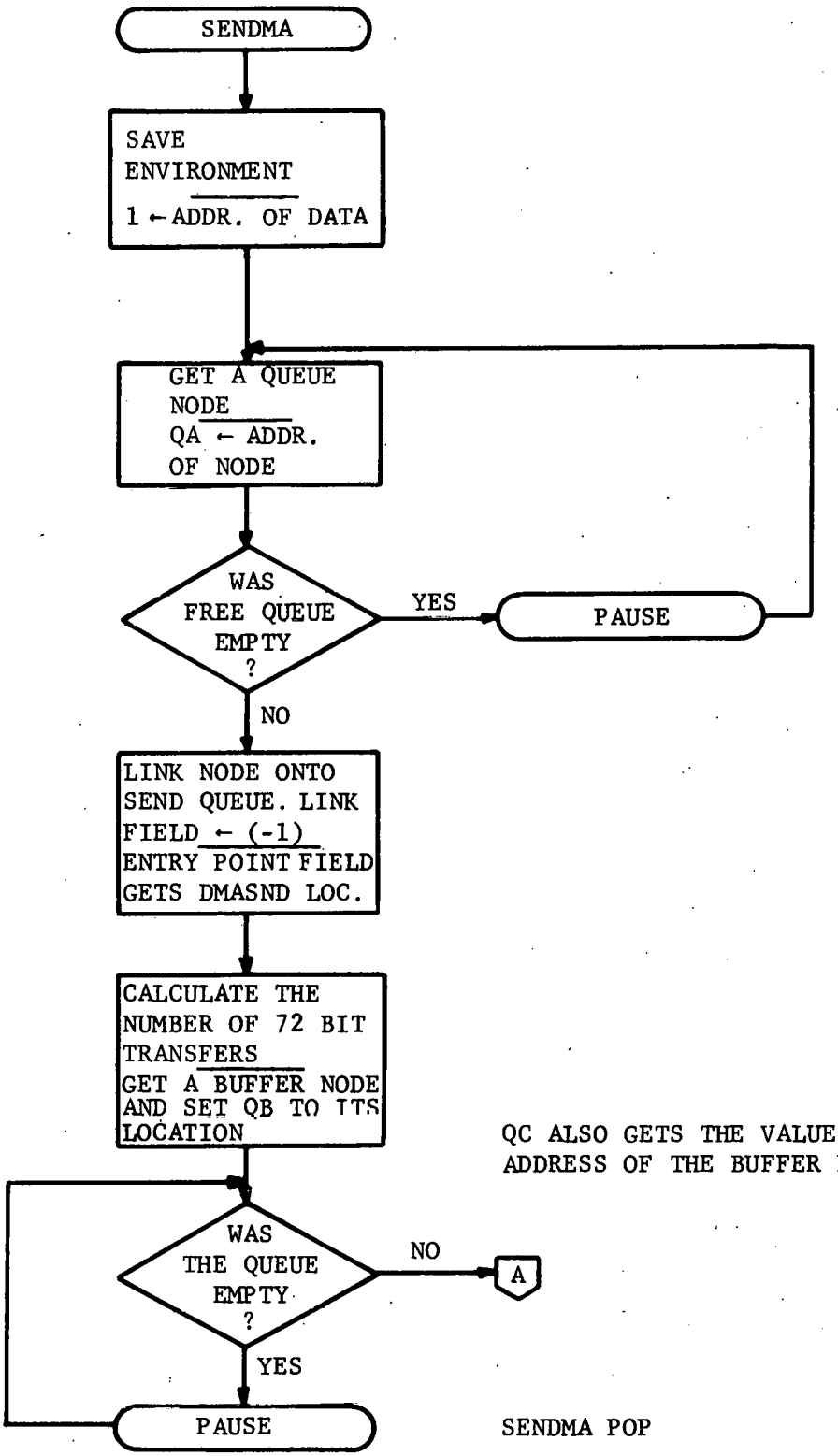
# APPENDIX A

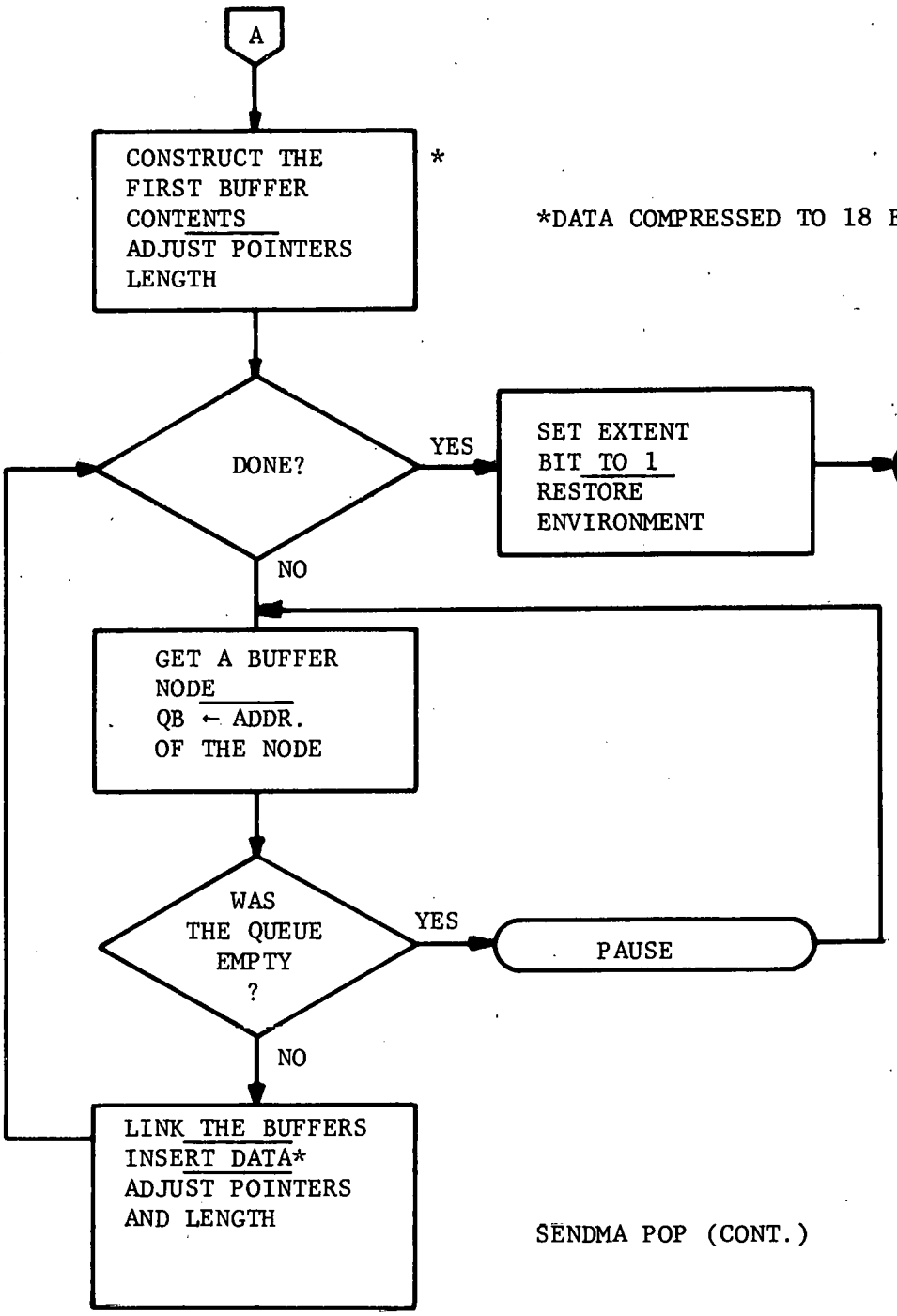



SYSTEM TASKER (DISPATCHER)



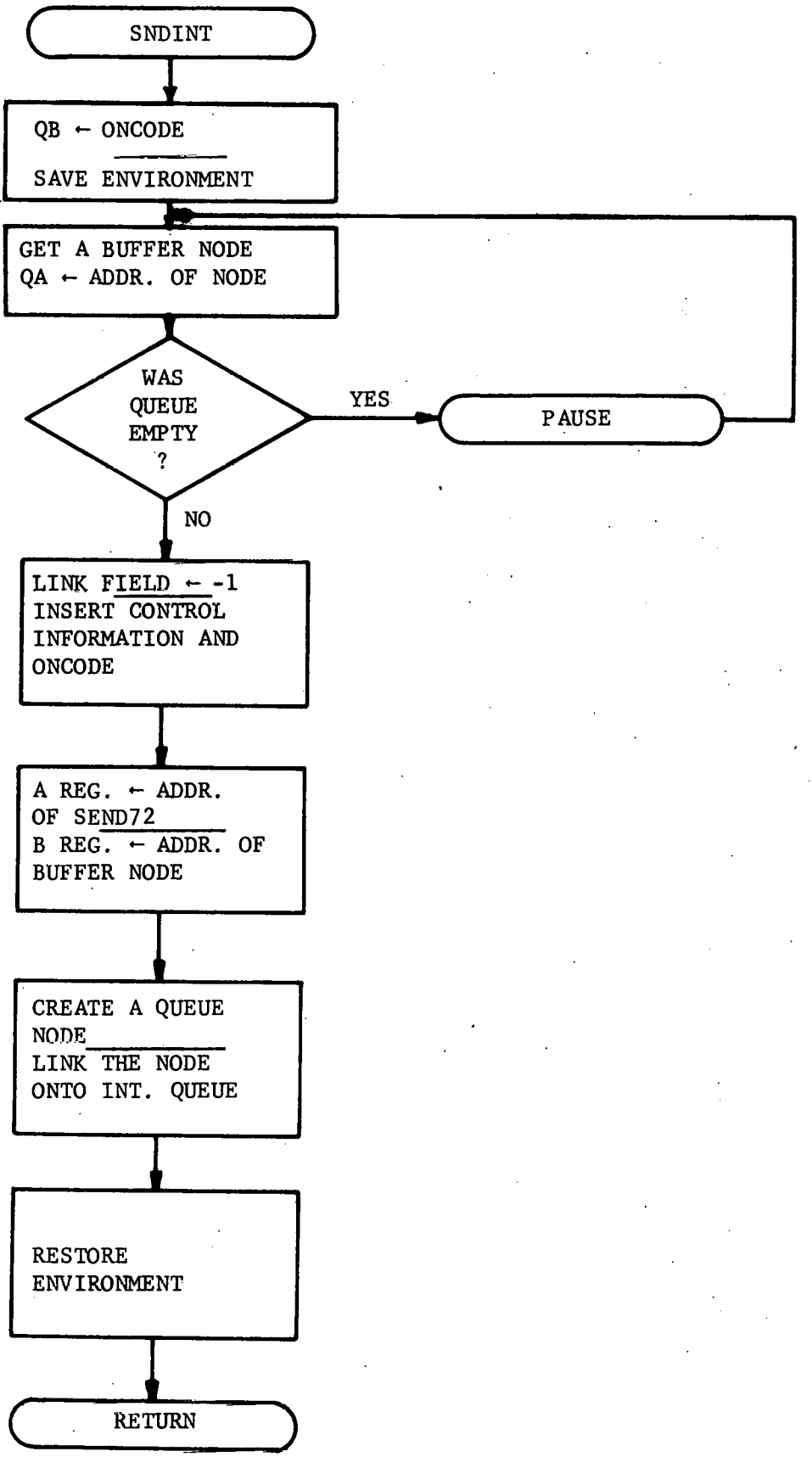

SNDFIX AND SNDFLT POPS


SENDMA
SAVE ENVIRONMENT
1 ← ADDR. OF DATA
GET A QUEUE NODE
QA ← ADDR. OF NODE
WAS FREE QUEUE EMPTY ?
YES
PAUSE
NO
LINK NODE ONTO SEND QUEUE. LINK FIELD ← (-1)
ENTRY POINT FIELD GETS DMASND LOC.
CALCULATE THE NUMBER OF 72 BIT TRANSFERS
GET A BUFFER NODE AND SET QB TO ITS LOCATION
QC ALSO GETS THE VALUE OF THE ADDRESS OF THE BUFFER FIELD
WAS THE QUEUE EMPTY ?
NO
A
YES
PAUSE
SENDMA POP

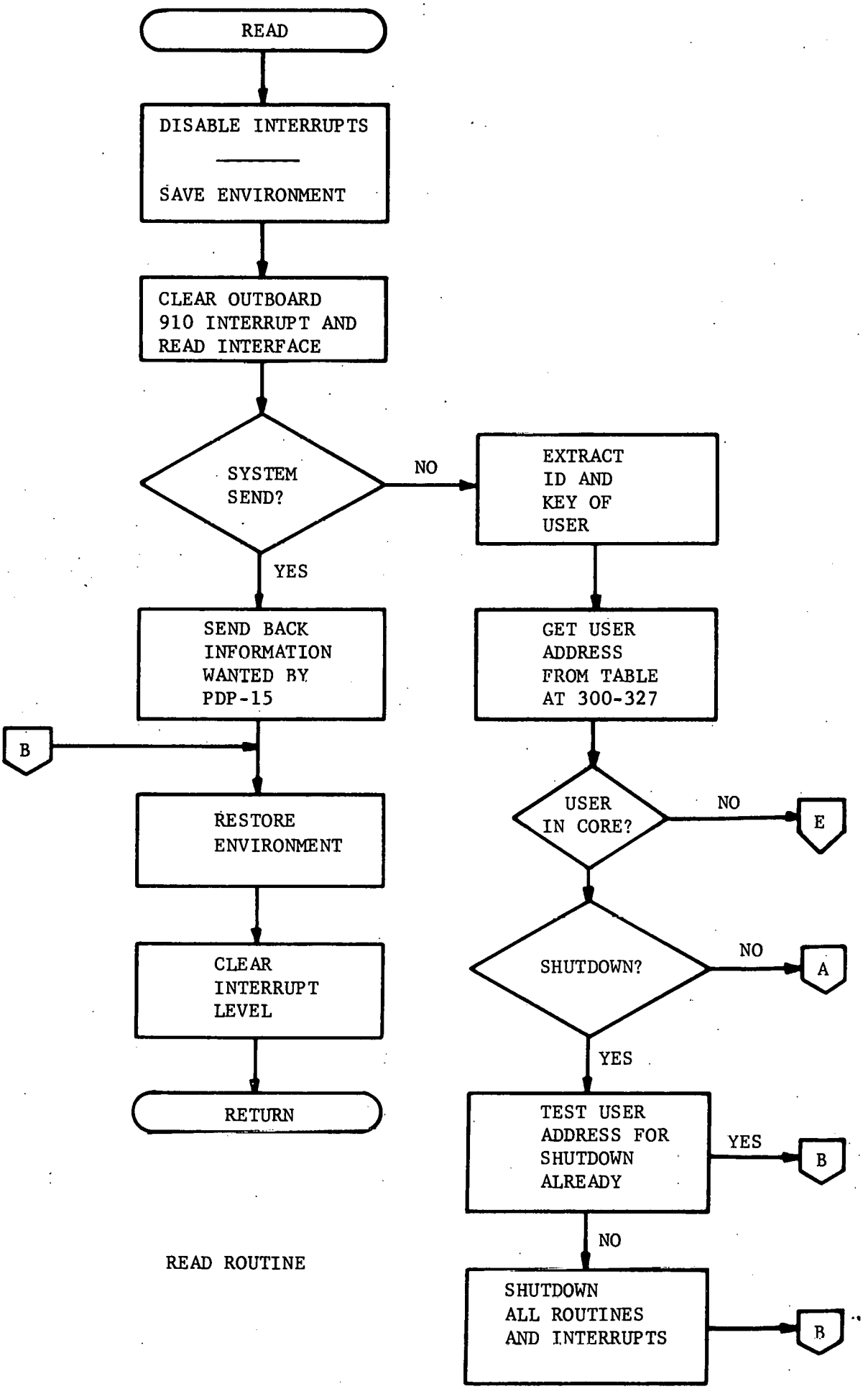



*DATA COMPRESSED TO 18 BIT FORMAT

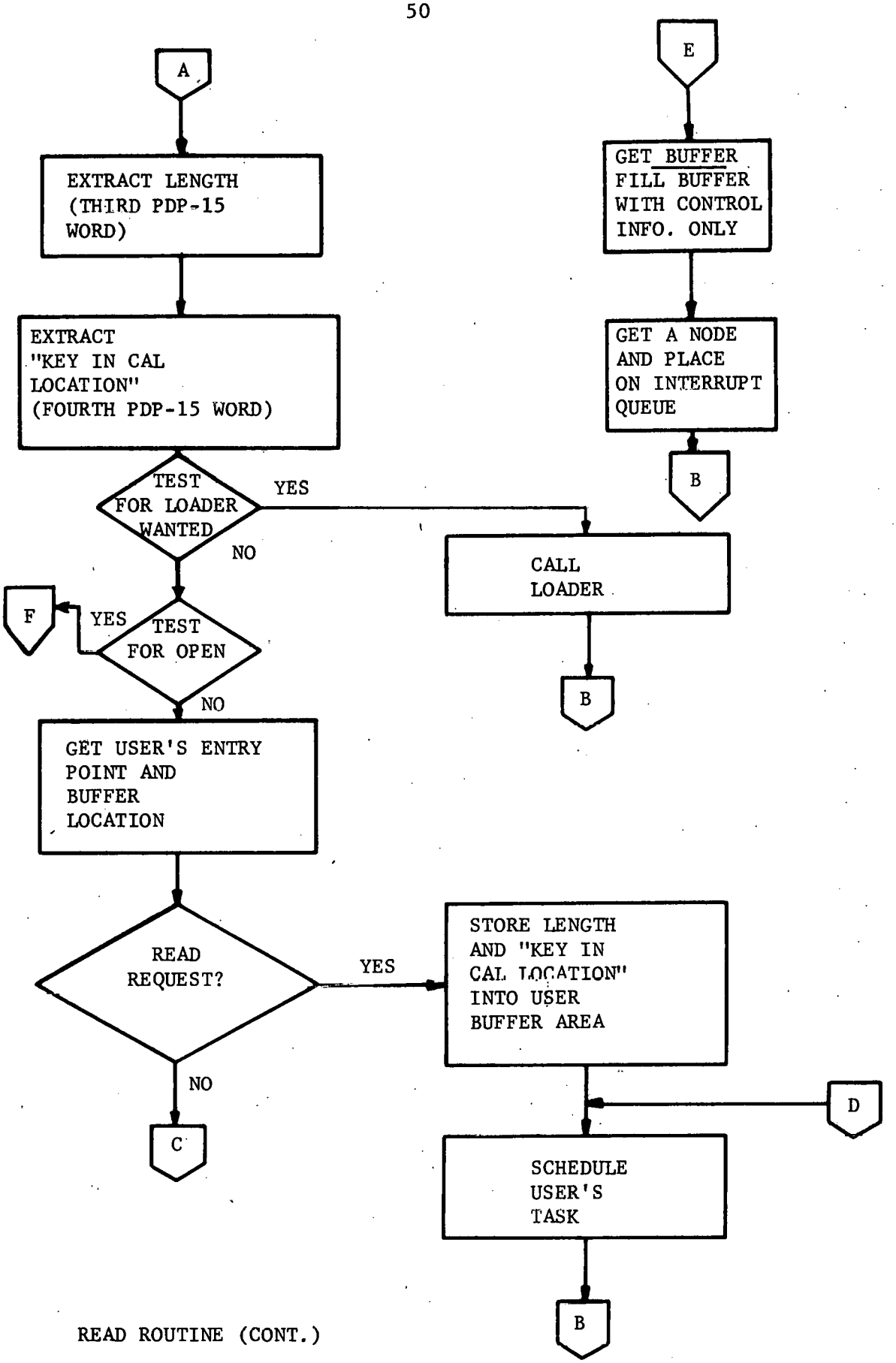
SENDMA POP (CONT.)


SNDINT
QB ← ONCODE
SAVE ENVIRONMENT
GET A BUFFER NODE
QA ← ADDR. OF NODE
WAS QUEUE EMPTY ?
YES
PAUSE
NO
LINK FIELD ← -1
INSERT CONTROL INFORMATION AND ONCODE
A REG. ← ADDR. OF SEND72
B REG. ← ADDR. OF BUFFER NODE
CREATE A QUEUE NODE
LINK THE NODE ONTO INT. QUEUE
RESTORE ENVIRONMENT
RETURN

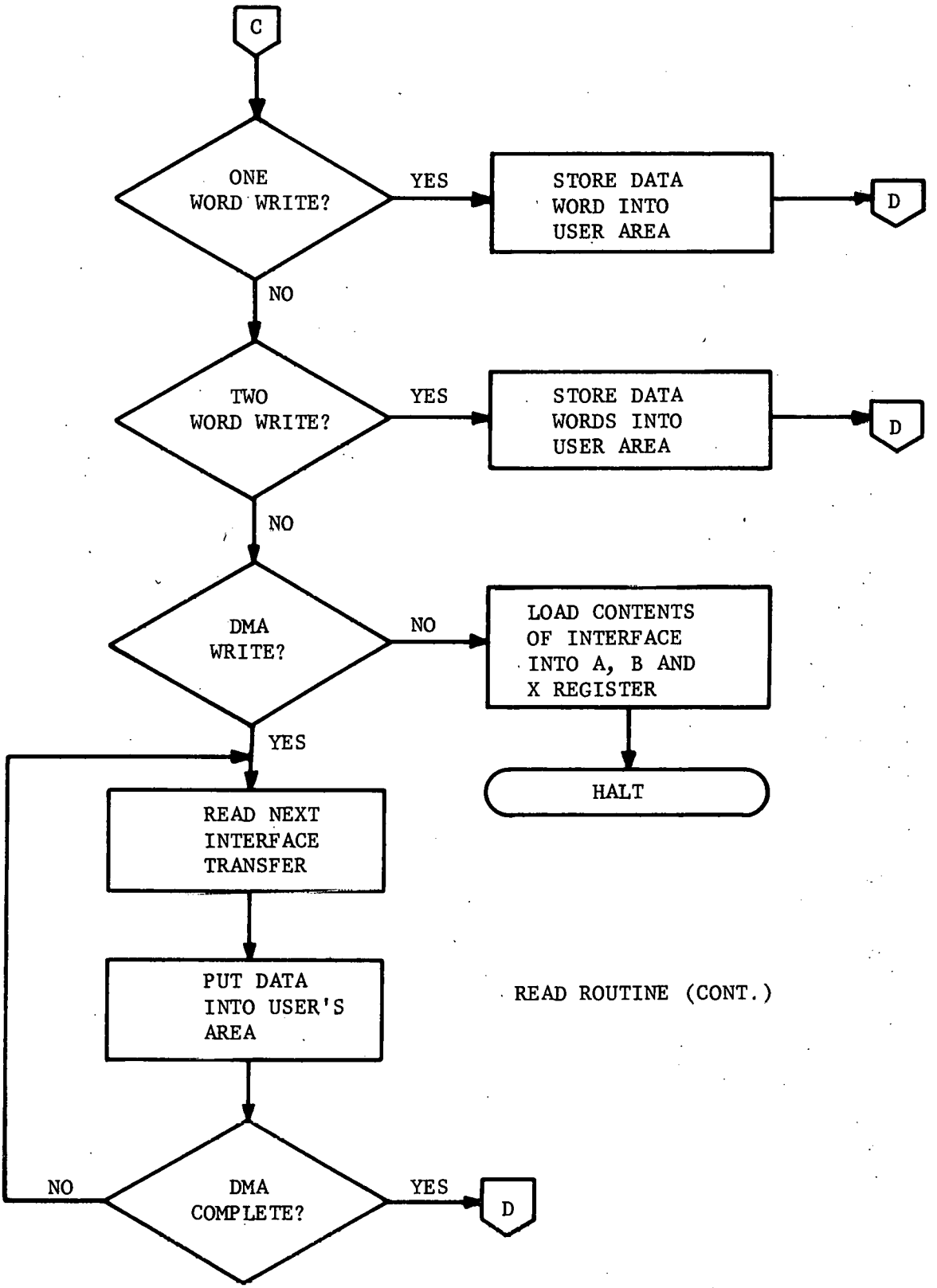
SNDINT POP



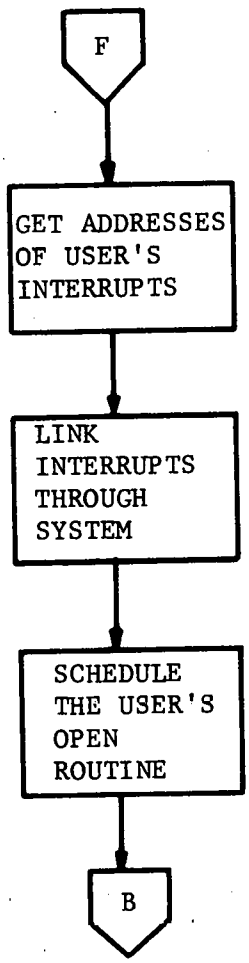

READ ROUTINE



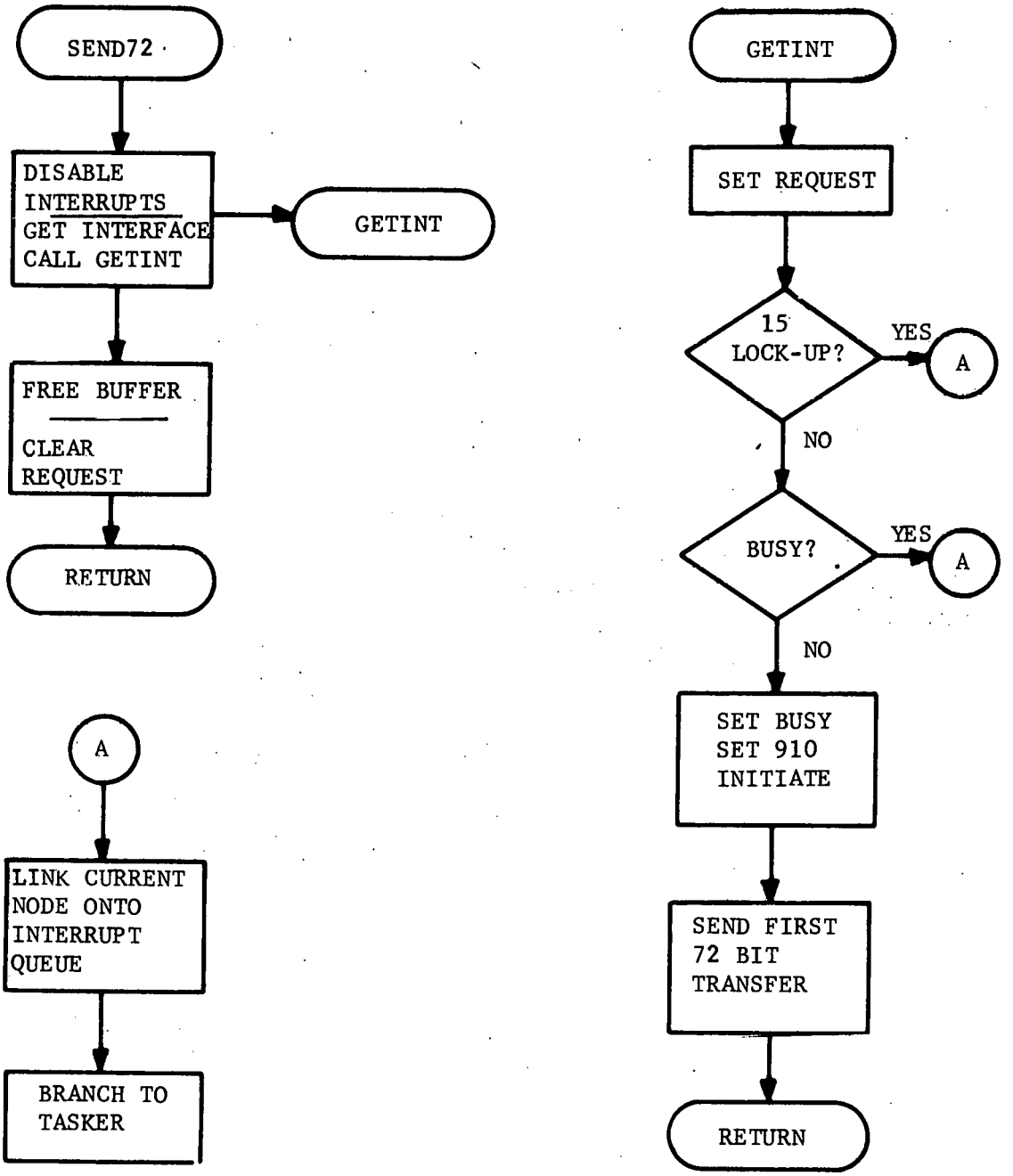

READ ROUTINE (CONT.)



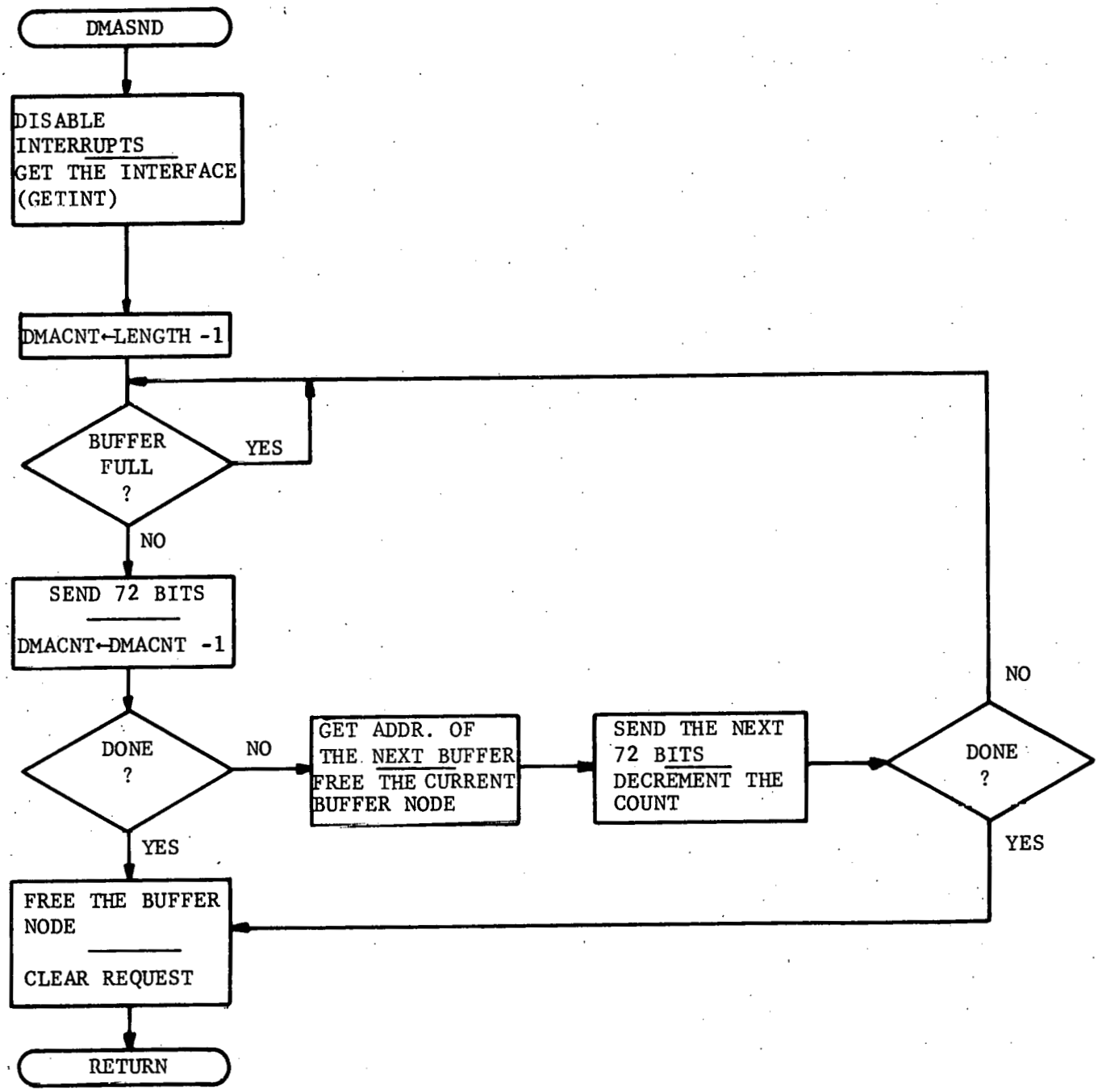

READ ROUTINE (CONT.)



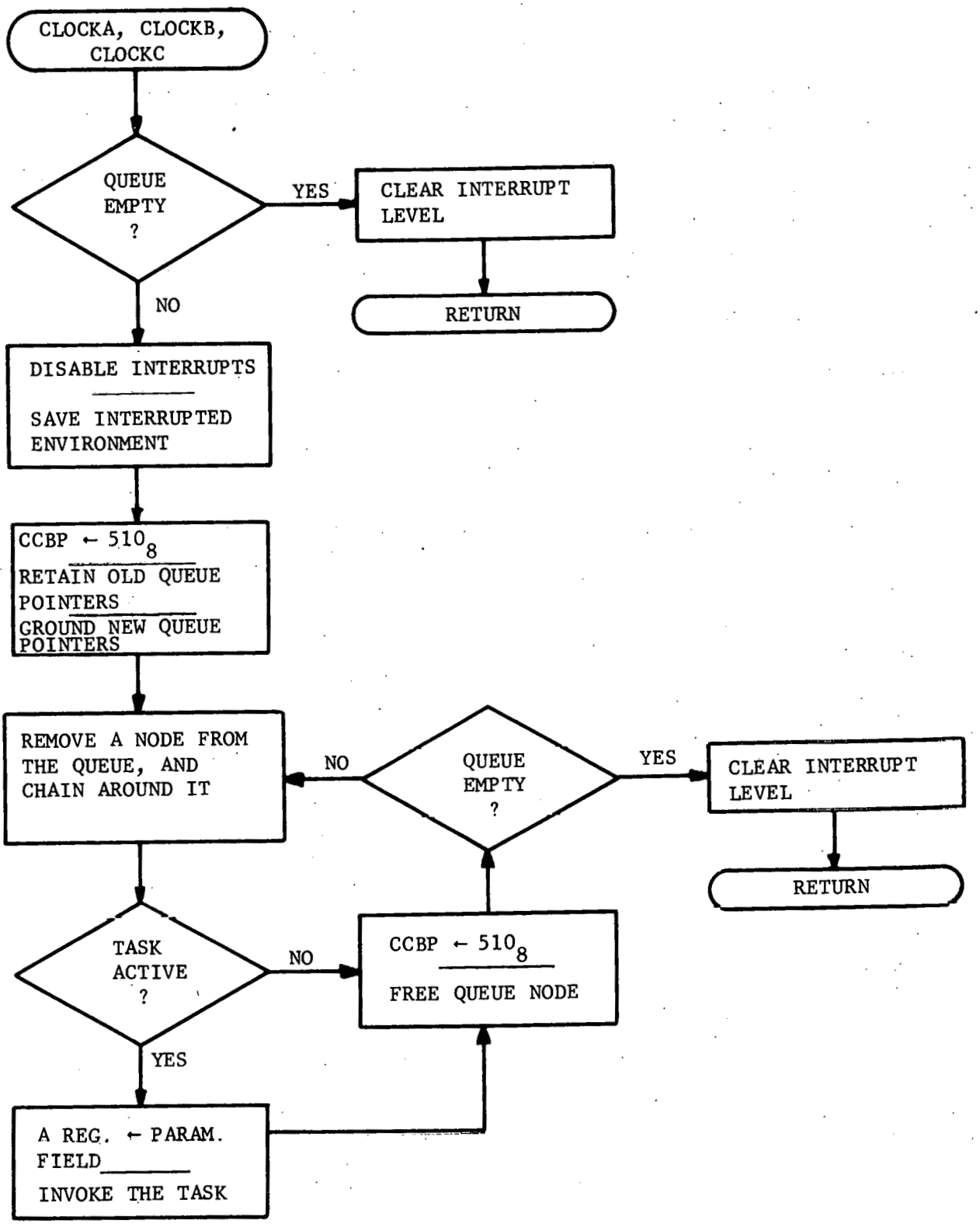

READ ROUTINE (CONT.)



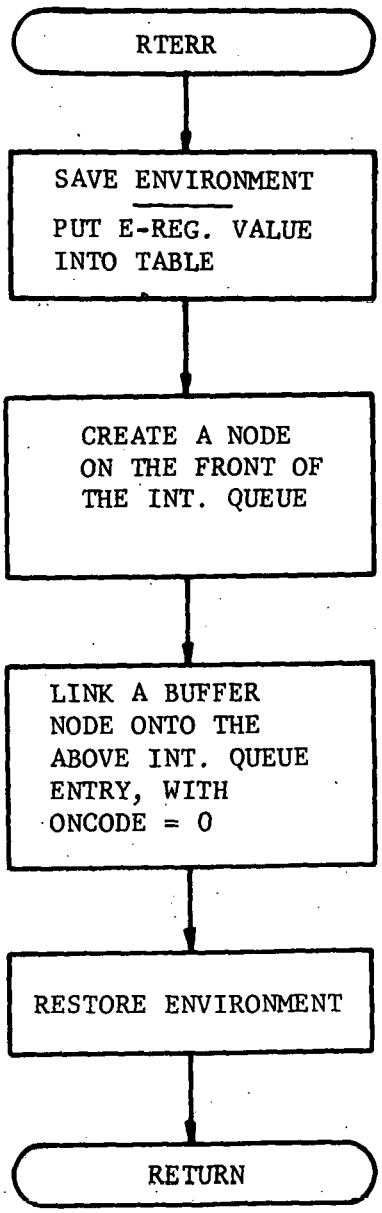

SEND72 AND GETINT ROUTINES



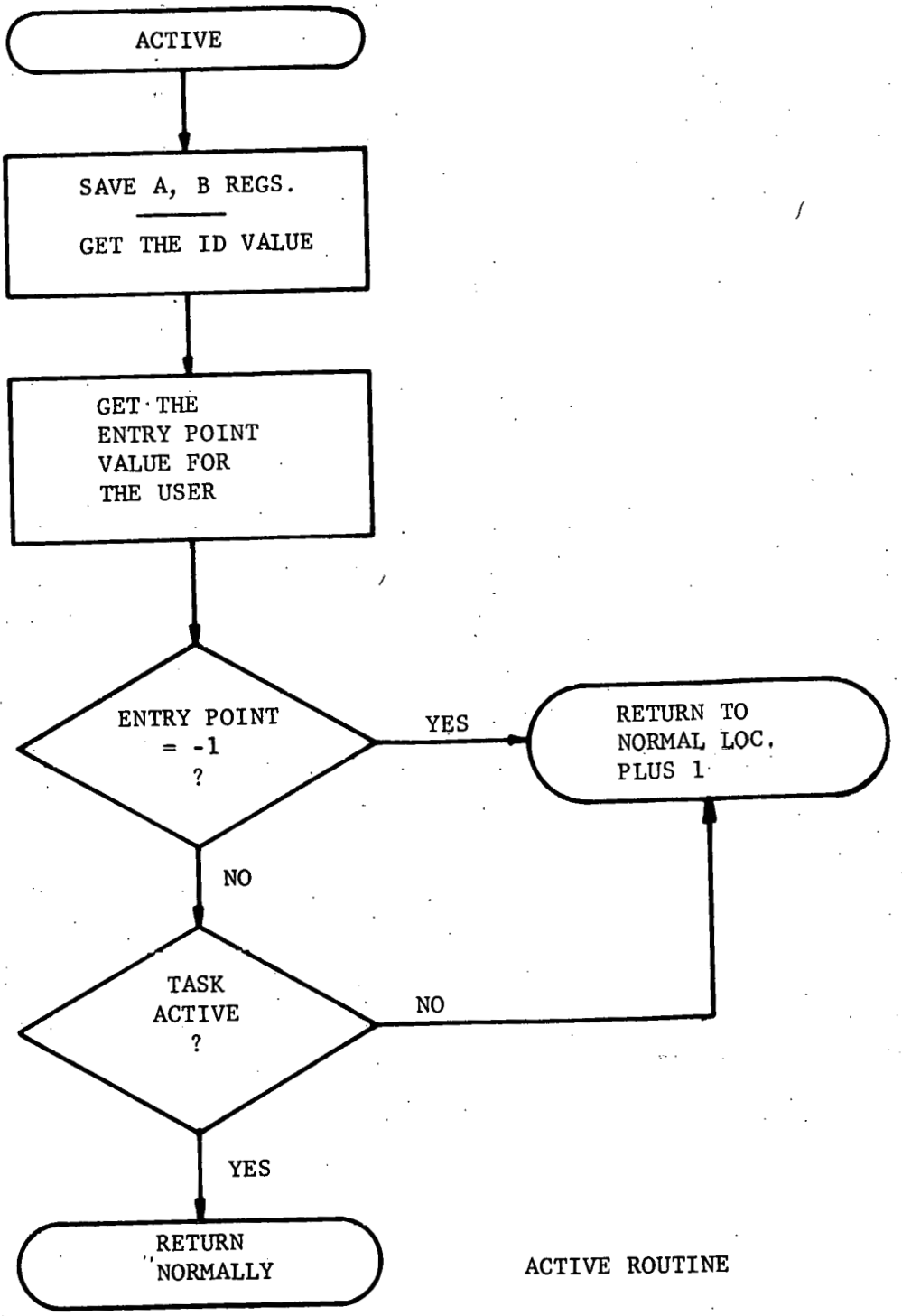

DMASND ROUTINE



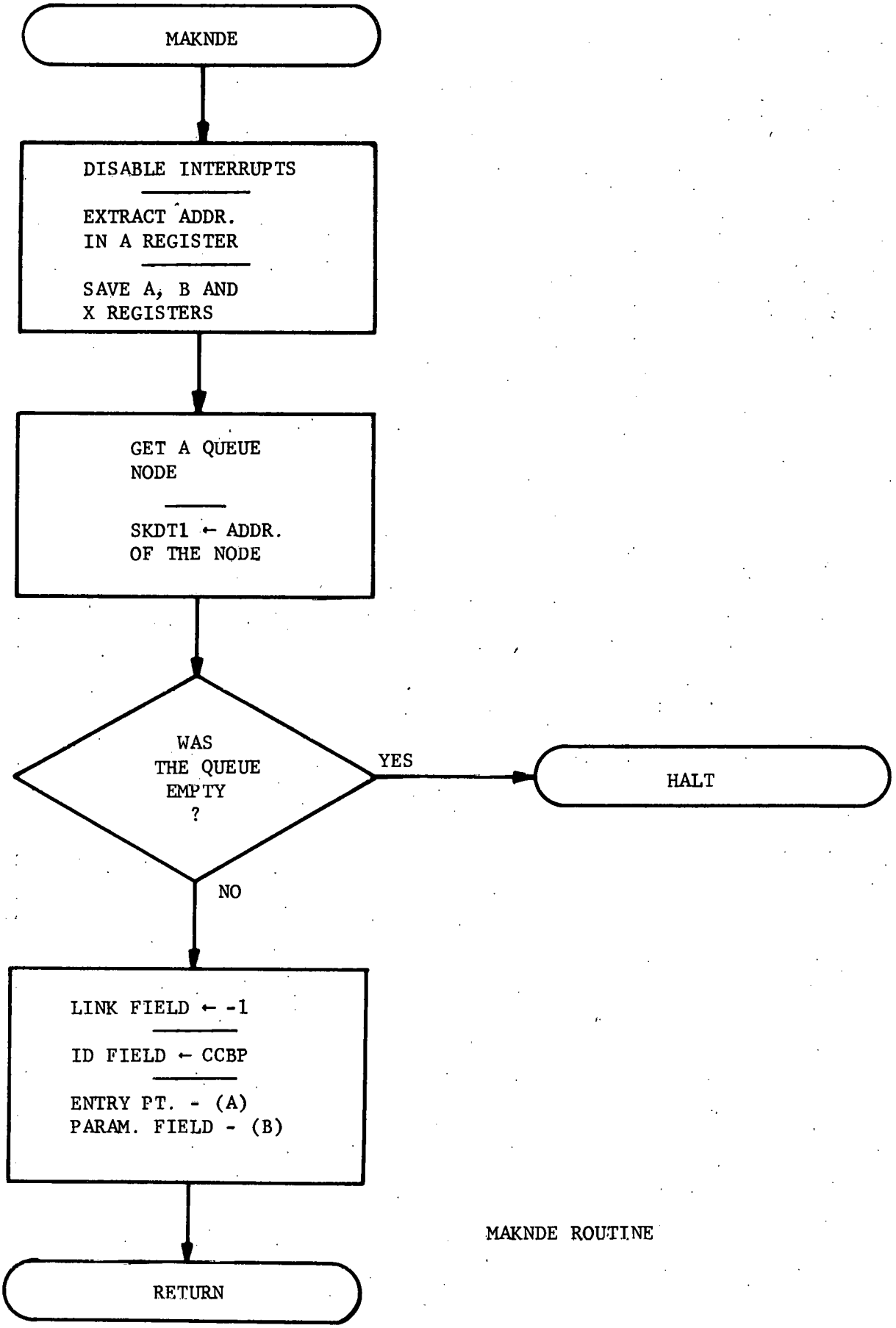

FAST, MEDIUM, AND SLOW CLOCK HANDLERS



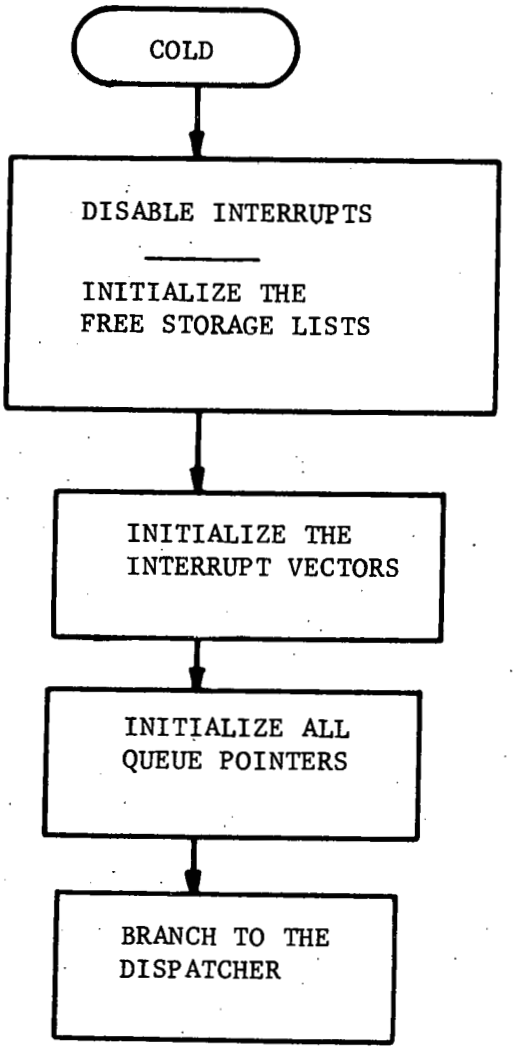

RT ERROR ROUTINE



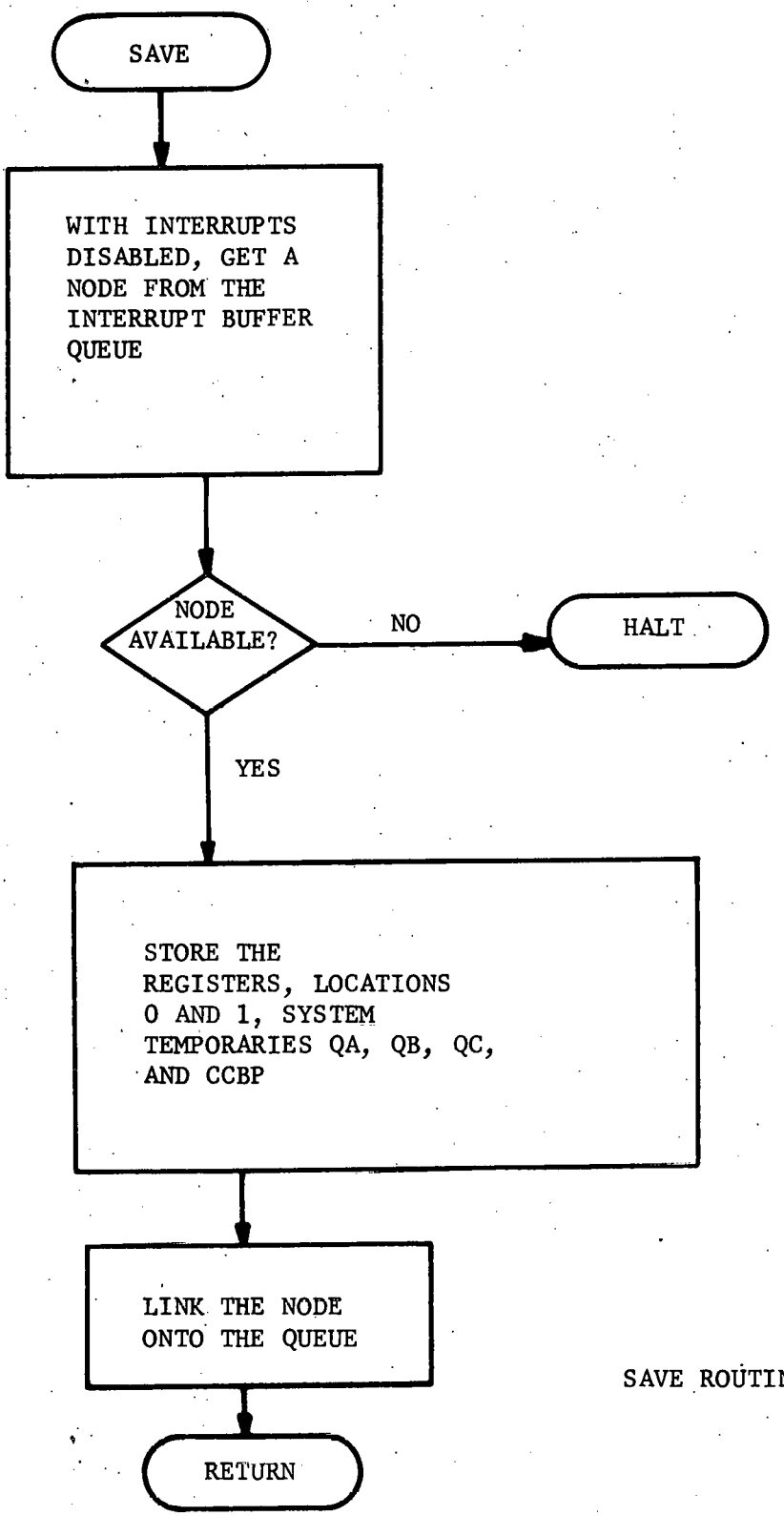

ACTIVE ROUTINE



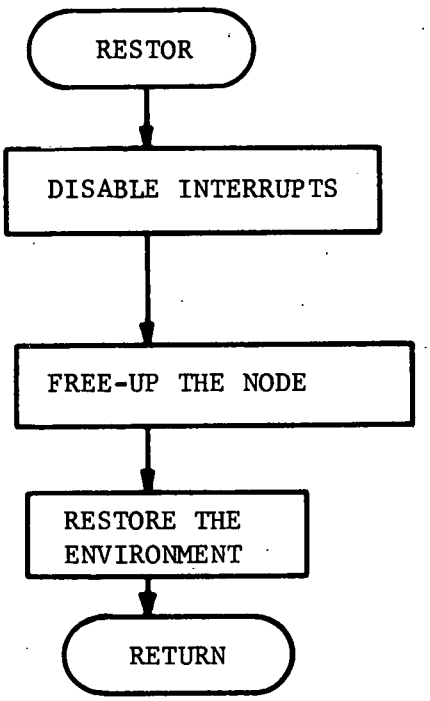

MAKNDE ROUTINE




COLD START ROUTINE




SAVE ROUTINE




RESTORE ROUTINE

# APPENDIX B

## Priority Scheduling*:

OS910 employs a priority scheduling and queuing technique that, although simple in nature, insures a useful priority ranking. Two places exist where task dispatching may take place. First is in the dispatcher routine, TASKER, where transmissions (SENDS) from the SDS-910 to the DEC PDP-15 are arranged and regular user task scheduling is handled. Secondly, the clock may have tasks scheduled on their queues for execution at the time of a clock "tick".

In light of their volatile nature, the transmission of interrupt derived information must be handled in a high priority fashion. Thus, interrupt queue (INTQUE) transmissions are managed first. Regular data transmissions are next in importance for the dispatcher, and are handled after all queued interrupt transmissions have taken place. Last in line for dispatcher handling are the user tasks that have been scheduled for execution. The clock queue management will be covered later.

The above method ensures that transmissions are handled with the greatest speed. Although using a scheme where there must be a specified (threshold) number of nodes present in a set of queues before any node scheduling is done seems desirable, the pitfalls presented by this method are many. Included in the pitfalls are the following:

1. Interrupt transmissions will be unnecessarily held up;

---

*The reader should consult the appropriate flowcharts in Appendix A.

2. Transmissions may never be scheduled if the threshold value is too high;

3. No discernable difference between having the threshold and not having it can occur if the threshold value is too low;

4. Since the DEC PDP-15 will normally be waiting for data from the SDS-910, it is necessary that the data are sent without interruption.

Thus, the queues available to the dispatcher are handled sequentially from the interrupt send queue (INTQUE), to the regular send queue (SNDQUE), to the task queue (TSKQUE). The first non-empty queue, in order, is handled in the following fashion (the interrupts are first disabled):

1. The front node on the queue (queues are arranged as a front linked list, and with a front/back pointer pair available that references the list) is unchained from the list, and the list is rechained to exclude this node;

2. If the ID present in the node is for a currently inactive task, then:

   a) if this is not a TSKQUE entry, all the acquired buffer storage is freed,

   b) the queue node free storage is released,

   c) start over on checking the queues;

3. If the task is active:

   a) set the A register contents to the value of the parameters pointer field, and

   b) execute the task.

4. When the task returns:

   a) disable the interrupts,

b) free the queue node entry,

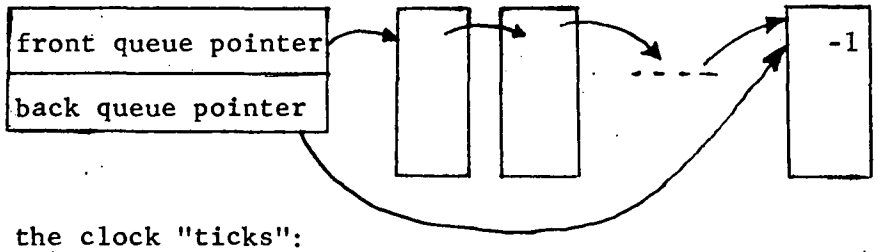
c) enable the interrupts,

d) and start over again on scanning the queues;

Thus, a rudimentary priority ranking is maintained.

The clock queues are managed somewhat differently, as each clock has only a single queue associated with it. A clock queue's entries are collected until that clock "ticks".

The form of a clock queue is:




When the clock "ticks":

1. the queue pointer contents are copied to a working area, allowing later reference to the queue;

2. the queue pointer value is set to "empty" (-1);

3. the tasks on the queue are sequentially dispatched (if active);

4. when the queue is empty, the clock manager returns to the interrupted environment by executing an indirect branch, thus clearing the interrupt level.

Obviously, the algorithm used prevents the same clock level from "hitting" during execution of the tasks on that clock's queue. Also, all lower priority interrupts are excluded from being recognized until the clock queue is emptied, and all higher interrupt levels are allowed to hit, as the interrupt system is enabled.

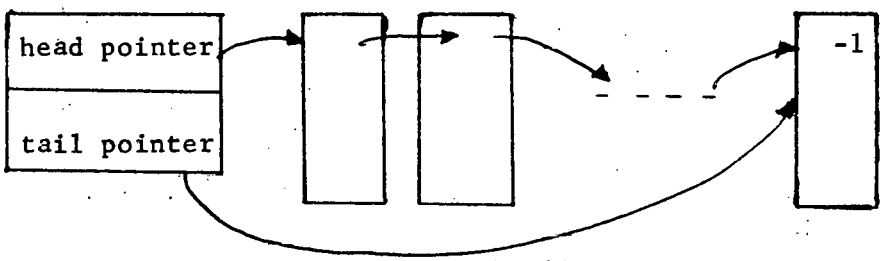
It is anticipated that all tasks queued onto a clock queue are kept small (no more than 50-100 instructions, each), and perform only absolutely essential time-dependent processing. Any operation that usurps a great amount of time will seriously degrade the performance of the complete system.

# APPENDIX C

Round-Robin Scheduling:

The task scheduling technique employed in OS910 is the most rudimentary of all (multiprogramming) methods available. Called the round-robin approach it is a scheduling technique based on the "trust thy neighbor" concept.

First, we will consider all tasks to be on a simple (forward linked) list, as:




Beginning with the first task on the list, we:

1. chain around the first node on the list,
2. execute the code for the task that was just unchained from the list,
3. upon completion of execution of that code (indicated by a return from the code), free up the list node storage, and
4. start over on the above process.

As can be seen, the amount of time allocated to each task is that amount of time the task needs to completely execute. If a task were to get into an "infinite loop", then no subsequent tasks would be scheduled for execution.

At this point, the reader may wonder why anyone would even employ such a naive scheduling technique as the round-robin. As OS910 assumes relatively short tasks will be scheduled (300-500 instructions, taking, at most, about 5 msec.), the use of even the fast clock (12.8 ms.) is too slow to cause a resolution.

Another problem avoided by using a round-robin technique is that of rather extensive queuing and data structuring problems. Interrupts would be more difficult for users to directly service under a time-sliced method (using a clock to interrupt the execution of one task and go on to the next), and would force the operating system to do more of the processing than is really required by the problem. Because of the queuing difficulties, a more extensive set of data structures would be needed.