DEC-10-MZZA-D

PDP-10 MONITOR COURSE MATERIALS

OBSOLETE
Aug 72

DIGITAL EQUIPMENT CORPORATION . MAYNARD, MASSACHUSETTS

1st Printing October 1970

The material in this document is for information purposes and is subject to change without notice.

The following are trademarks of Digital Equipment Corporation, Maynard, Massachusetts:

DEC	PDP
FLIP CHIP	FOCAL
DIGITAL	COMPUTER LAB

1. Introduction to the Monitor

Readings

PDP1Ø Timesharing Handbook

Introduction to Timesharing

12 pages

Good introduction if you are not familiar with timesharing.

Program Logic Manual

Memo #1, Executive Mode Use of the Priority Interrupt System

8 pages

Basic principles of priority interrupt programming.

Handout - "Introduction to Monitor"

Summary of classroom presentation

Flowcharts and Diagrams

Handout 46 - Functional Diagram of the Monitor

Handout 31 - The Monitor Cycle

PROGRAM LOGIC MANUAL

for

PDP 1Ø TIME-SHARING MANUAL

MEMO #1

EXECUTIVE MODE USE OF THE PRIORITY INTERRUPT SYSTEM

The PDP-10 incorporates a flexible seven channel priority interrupt system that is particularly useful in programming efficient multiple input/output operations. The purpose of this section is to acquaint the user with some of the programming techniques involved with using this system. Topics to be discussed include

1. The manner in which input/output (I/O) devices are connected to the interrupt channels;
2. Programmed control of the PI system;
3. The action taken by the system upon acknowledgment of an interrupt request from a device;
4. The action most appropriately taken by the programmer in anticipation of such an interrupt.

1. THE SIGNIFICANCE OF PRIORITY LEVELS

The seven priority interrupt channels are numbered according to their priority level, with channel 1 having the highest priority. When an interrupt request on a given channel is being serviced, no further interrupts can occur on that channel or on any channel of a lower priority; however, a channel of a higher priority can interrupt the routine servicing the original interrupt. The system is designed so that the original routine is resumed following the servicing of the higher priority interrupt. Further, requests occurring on lower priority channels are never lost, but are simply held until such time as they can be acknowledged and serviced. In general, an interrupt request from a device consists of a level present on one of the seven PI request channels (lines) which are part of the I/O bus. This level remains present until the device is serviced.

2. PRIORITY CHANNEL ASSIGNMENTS

Up to 126 input/output devices can be connected to the central processor via the I/O bus. Under control of the Monitor, one or more devices can be connected to any one of the seven priority interrupt channels. A particular device is connected to a channel with a Conditions Out (CONO) instruction directed to that device. The CONO instruction contains the channel number in its effective address portion, usually in bits 33 through 35.

```
Example     CONO PTR, 000005        ;ASSIGNS THE PAPER TAPE READER TO CHANNEL
                                    ;5, SO THAT WHENEVER THE READER'S "DONE
                                    ;FLAG IS SET, AN INTERRUPT WILL BE RE-
                                    ;QUESTED ON CHANNEL 5
```

Note that in this example, as in all others to follow, the coding is presented in a format acceptable to the Macro-10 Assembler and that all numbers are in octal. After this instruction has been executed, the paper tape reader will request an interrupt each time its "done flag" is set; whether or not the request is acknowledged depends on the state of the PI system, a condition entirely within control of Monitor (this is discussed below).

3. CONTROL OF THE PI SYSTEM

The PI system itself is considered to be an I/O device and is controlled by a Conditions Out (CONO) instruction with a device code of 004. As in the case of other I/O devices, the PI system may be thought to contain a control register whose bits are set according to the bits in the effective address of the CONO instruction. The significance of these control bits is summarized below. Note that in this summary the term "selected channels" refers to those channels corresponding to 1's in bits 29 through 35 of the control register (the effective address of the CONO instruction), where bit 29 corresponds to channel 1, bit 30 to channel 2, etc.

BIT	OCTAL	FUNCTION
23	10000	Clear the entire PI system.
24	4000	Activate an interrupt on the selected channels.
25	2000	Turn on the selected channels.
26	1000	Turn off the selected channels.
27	400	Turn on the PI system.
28	200	Turn off the PI system.

Bit 23, if a 1, cancels all previous requests, turns off all channels, and turns off the PI system. Bit 24 is used to request an interrupt on a different priority channel than the one which is active. Bits 25 and 26 allow the user to turn on or off (but not both in any single instruction) any desired channel or channels. A request level present on the I/O bus line connected to a channel which has been turned off will not be acknowledged, but the level remains present awaiting reactivation of the channel. Thus, the user can delay an interrupt or prevent it from occurring at an inopportune time by turning off the appropriate channel.

The entire PI system can be turned off with bit 28. From the viewpoint of

external devices, the system appears as if it were permanently servicing a request on a channel with a higher priority than channel 1. Any interrupt requests which occur will be acknowledged if their respective channels are on, but they will not be serviced until the system is turned back on with bit 27.

Some examples may serve to illustrate these concepts.

```
CONO PI, 10000          ;CLEAR THE PI SYSTEM.  THIS INSTRUCTION
                        ;MAY BE USED TO ADVANTAGE IN THE
                        ;INITIALIZATION SECTION OF A PROGRAM
                        ;USING THE PI SYSTEM.
CONO PI, 1007           ;TURNS OFF PI CHANNELS 5 THROUGH 7
CONO PI, 12577          ;CLEAR THE PI SYSTEM, TURN ON THE PI
                        ;SYSTEM, AND TURN ON ALL SEVEN CHANNELS
```

Note that conflicting requests (e.g., both bits 27 and 28 set to 1) will yield unpredictable results; the "clear PI system" operation (bit 23), however, does not conflict with any other operation and occurs first when the CONO instruction is executed. Also, the following two instructions are equivalent in effect, if channel 1 is the only channel being used.

```
CONO PI, 200            ;BIT 28 - TURN OFF THE PI SYSTEM
CONO PI, 1100           ;BITS 26 AND 29 - TURN OFF CHANNEL 1
```

4. MACHINE ACTION UPON INTERRUPT

When an interrupt level appears and the selected channel is free and no higher priority channel is in use, an interrupt is granted at the end of the instruction in progress. The mechanism is as follows.

Control is transferred to core memory location 40 + 2n, where n is the channel number. The program counter is not affected in any way by the interrupt unless the instruction in location 40 + 2n changes the program counter during execution (if, for example, this location contains a jump-type instruction, it is the programmer's responsibility to preserve the contents of the program counter if he has any intention of returning to the interrupted program sequence). The system is designed so that the instruction in location 40 + 2n should be one of the following:

```
JSR
BLKI
BLKO
```

Each of these will be considered in detail later. While other instructions are not illegal, their use is never necessary and should, in general, be avoided.

One further point should be understood: when an interrupt is serviced (when the program sequence beginning in location 40 + 2n is being executed), the PI system is disabled to the extent that further interrupts may not occur on the channel currently in use or on any lower priority channel. This condition prevails until the program dismisses the channel in use. This action of dismissing the channel must be taken by the program before control is returned to the interrupted sequence if any further use of the affected channels is expected. There are only two ways in which a channel can be dismissed: the first is through the execution of a JRST instruction with bit 9 equal to 1; the other is through the execution of a BLKI or BLKO instruction, both of which will automatically dismiss the current channel if the transfer of a data block is still incomplete. These concepts will be illustrated and clarified in the programming examples which follow.

5. INPUT/OUTPUT PROGRAMMING

NOTE

It is assumed that the reader is familiar with the operation of the JSR and JRST instructions as well as the eight I/O instructions given in the PDP-10 System Reference Manual.

Consider first the use of a JSR instruction in location 40 + 2n. As a specific example, consider the instruction

```
JSR 1000
```

in location 44, to which control is transferred when an interrupt request is serviced on channel 2. The state of the flags and the program counter (which is pointing to the instruction which was about to be executed when the interrupt occurred) is stored in location 1000; control is then transferred to location 1001 , with channels 2 through 7 disabled. Beginning at location 1001 should be a routine to service the device connected to channel 2. If several devices are connected to channel 2, the routine must contain appropriate CONSI or CONSO instructions to determine which "done flag" has been set. The last instruction in the routine should be a JRST 12, @1000. The specification of AC 12 causes bits 9 and 11 to be 1's; bit 11 specifies that the flags stored in location 1000 are to be restored to their former states and bit 9

causes the PI channel currently in use (channel 2) to be dismissed, thus freeing channels 2 through 7. Control is then transferred to the location specified in the address portion of location 1000 (the interrupted sequence). The execution of the routine beginning in location 1001 might have been interrupted by a request from a device assigned to channel 1. If, in location 42, there was a JSR to a similar service routine which ended with its own JRST 12, @nnnn, then control would automatically transfer back to the channel 2 routine and from there to the original interrupted sequence.

The above techniques may be extended to cover all seven channels and are sufficient for full utilization of the PI system. A partial schematic representation of the program structure appears on the next page.

When blocks of data must be transmitted into or out of core memory, especially when it is desired that the transfer take place at the maximum rate the I/O device allows, the BLKI and BLKO instructions may be used to advantage with the PI system. The technique is somewhat different from that of the JSR example above. As a specific example, consider the case of reading three words from paper tape into memory locations 6000 through 6002 while performing some computation elsewhere. The instruction

```
CONO PTR , 63
```

assigns the paper tape reader to channel 3 and causes one word (six frames) of tape to be read into the interface buffer. The instruction

```
CONO PI, 12420
```

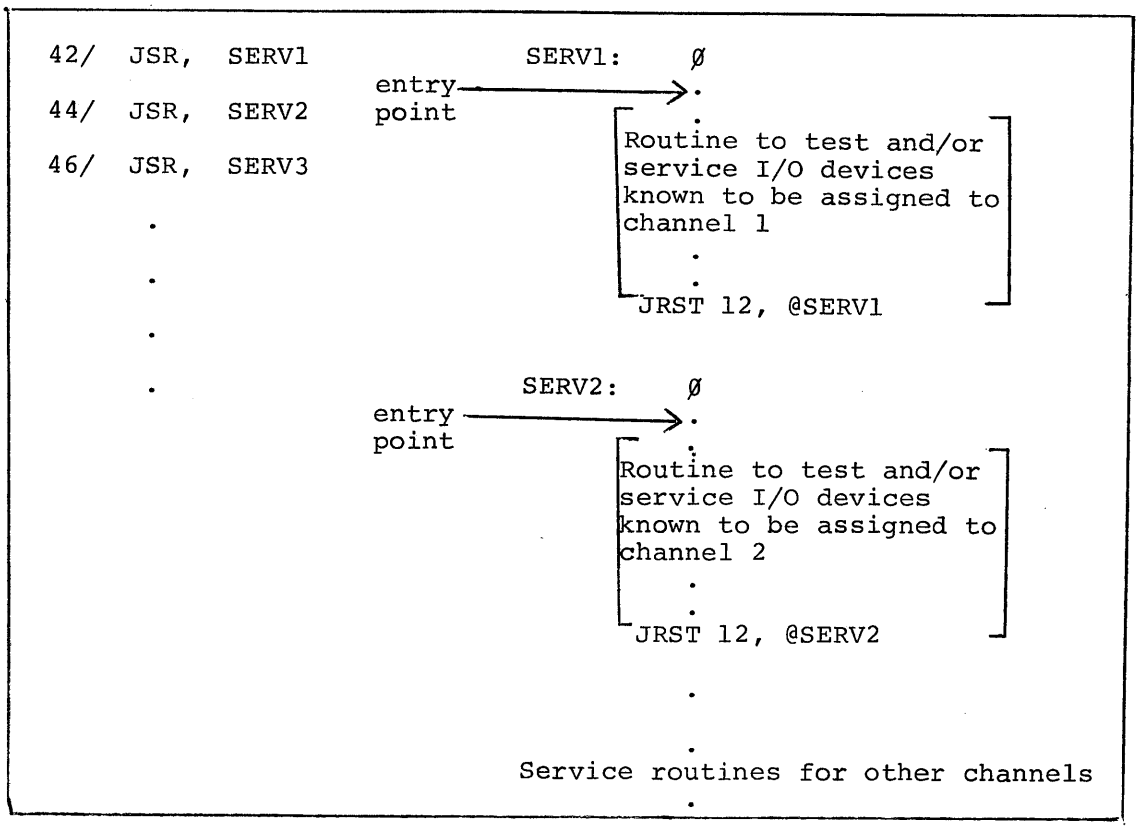
clears the PI system, turns it on, and turns on channel 3. When the reader "done flag" becomes a 1, an interrupt is requested on channel 3 and is serviced at the completion of the instruction in progress. Control is transferred to location 46, which should contain the instruction

```
BLKI  PTR, BPWD
```

where the block transfer pointer word is defined elsewhere using the IOWD pseudo-instruction

```
BPWD: IOWD  3, 6000
```

The execution of the BLKI instruction proceeds in the usual (non-PI) manner, except that when the single word transfer is complete and the pointer word



Partial Schematic Representation of JSR Example

has been tested for an end-of-block condition, one of two actions is taken by the hardware.

a. If the last data word of the block has not been read in, the interrupt channel currently in use is automatically dismissed and control is returned to the interrupted sequence (pointed to by the program counter).

b. If the last data word has been read in, the channel is not dismissed and control goes to the instruction following the BLKI in struction (in this case, location 47). The program counter is still pointing to the interrupted sequence, but it can be lost at this point through careless programming. The safest instructions to have in location 40 + 2n + 1 are JSR instructions to dismissal routines. In this example, the instruction

```
JSR  DISM
```

in location 47 might be used to complete the input operation by jumping to the brief routine

```
DISM:  Ø                ;BLANK REGISTER FOR PC
                        ;AND FLAGS

       JRST 12, @DISM
```

which would dismiss the channel and return to the interrupted sequence. Alternately, the routine beginning at DISM might turn off the PI system or take any other desired action before returning. There are slight differences between input operations and output operations. The reader should refer to the System Reference Manual for these distinctions.

Here is another example, this one of the output variety. The user desires to punch out a block of 100_8 locations, beginning at LIST, in binary format on paper tape while an independent program is running. Assume that the main program has executed the following three instructions to initiate the process.

```
MOVE  17,[IOWD 77, LIST +1]    ;IOWD XWD - 77, LIST

CONO PTP, 41

DTAO PTP, LIST
```

The first of these three instructions sets up a pointer and counter word in AC17. The next instruction sets the punch to binary mode and assigns it to PI channel 1. The last of these three instructions activates the punch and punches the first word. When the punch has finished punching the contents of LIST, its "done flag" is set and an interrupt occurs on channel 1. Consider the following two program sequences to service the interrupt. Assume that it is desired to turn off channel 1 to prevent further interrupts until after the last data word has been punched. The two sequences accomplish the same task. Note the manner in which each extracts the second data word correctly from LIST+1.

SEQUENCE 1

```
    42/ JSR OUTPUT

OUTPUT: Ø
        DATAO PTP,1(17)
        AOBJN 17, .+2
        CONO  PI, 1100
        JRST  12, @OUTPUT
```

SEQUENCE 2

```
    42/ BLKO PTP, 17
    43/ JSR  FINISH

FINISH: Ø
        CONO PI, 1100
        JRST 12, @FINISH
```

One final point: the use of the arithmetic processor as an I/O device. The processor can be assigned to any PI channel by a CONO instruction having a device code of Ø (mnemonic = APR). With the arithmetic processor so assigned, an interrupt is requested on the assigned channel whenever any one of the six flags listed below is set.

Memory Protection flag[1]
Nonexistent Memory flag[1]
Clock Count flag (if enabled)[1]
Floating Overflow flag (if enabled)
Overflow flag (if enabled)
Pushdown List Overflow flag (if enabled)[1]

A program designed to service an interrupt requested by the arithmetic processor would have to contain a series of Condition Skip instructions to determine which of the above flags caused the interrupt.

The diagram on the following page illustrates the priority interrupt levels, their functional relationships, and their relative processing intervals.

[1]The Time-Sharing Monitors always enable these flags for all users. The others, too, can be enabled privately by request of a user program.

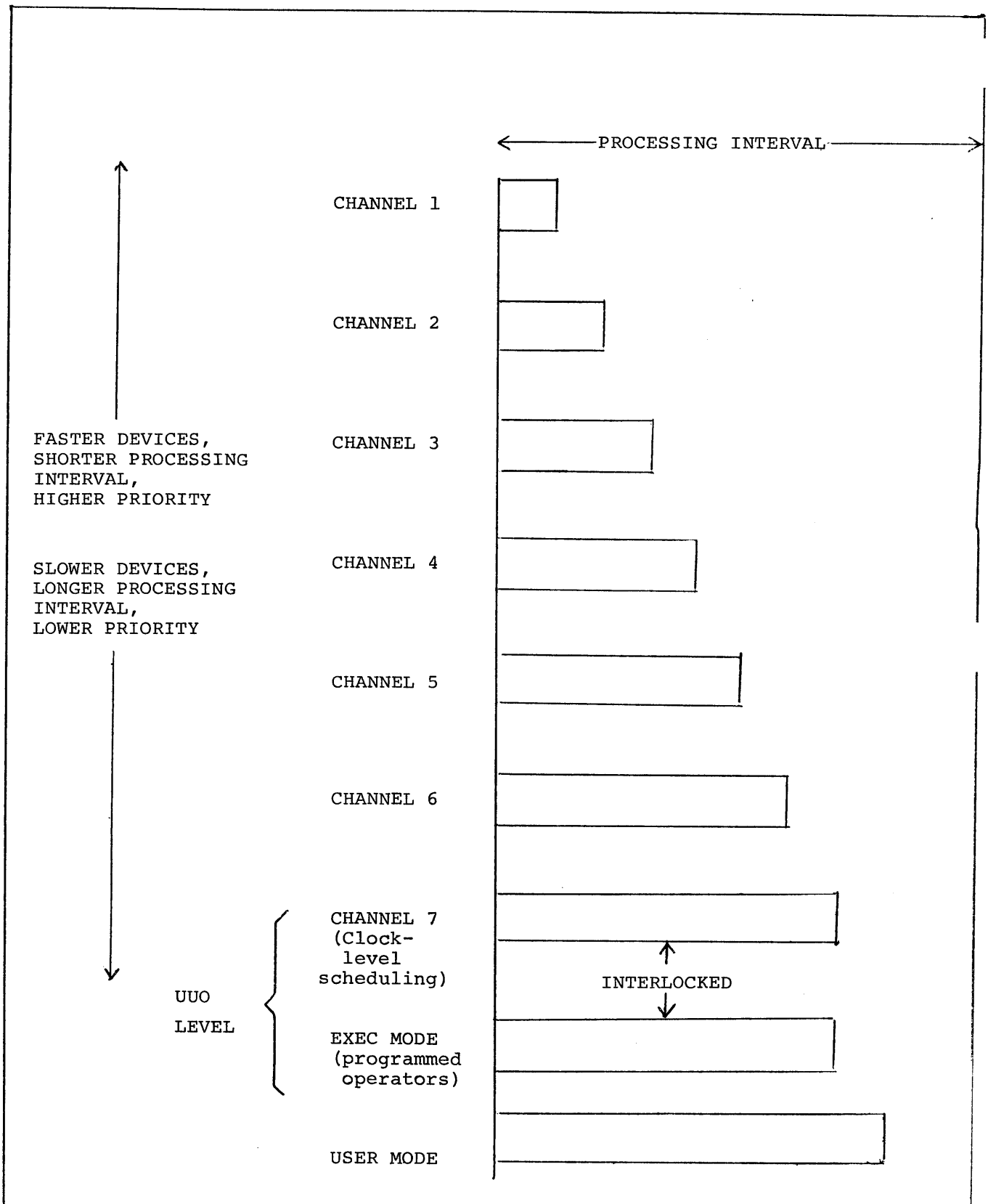


Diagram of Monitor Priority Interrupt Levels

In the PDP1Ø timesharing system, all user jobs operate under the control of an executive program known as the monitor. One of the major functions of the monitor is to allocate the resources of the system among the various users. Computer time is divided into slices of one sixtieth of a second by a hardware clock interrupt. Each time slice is allocated to a user job, which is allowed to run until the next clock interrupt. In addition to CPU time, all other resources of the system - I/O devices, controllers, etc., are allocated among the users by the monitor. Some resources, known as sharable resources, can be allocated to a job for a short time and then can be reclaimed and given to another job. Other resources are assigned to a job until that job chooses to release them.

In addition to controlling user jobs, the monitor provides a number of services to user jobs, and to the users themselves. In either case, the monitor will perform a specific operation in response to a user request. Thus, while user jobs are controlled by the monitor, some parts of the monitor are controlled by the users.

The monitor, in fact, consists of a number of separate and somewhat autonomous routines. The Command Processor interprets commands typed by users on their Teletypes, and the UUO Processor interprets UUO's executed by user programs and other monitor routines. The UUO processor provides access to the File Handler and the device service routines. The File Handler allows users to refer to files in terms of file names and logical block numbers, without being concerned about physical locations. Device service routines handle all device dependent functions required by the UUO Processor. Also, there is a Scheduler, which selects the user job to run during each time slice, and a Swapper, which rotates jobs between core memory and disk or drum memory.

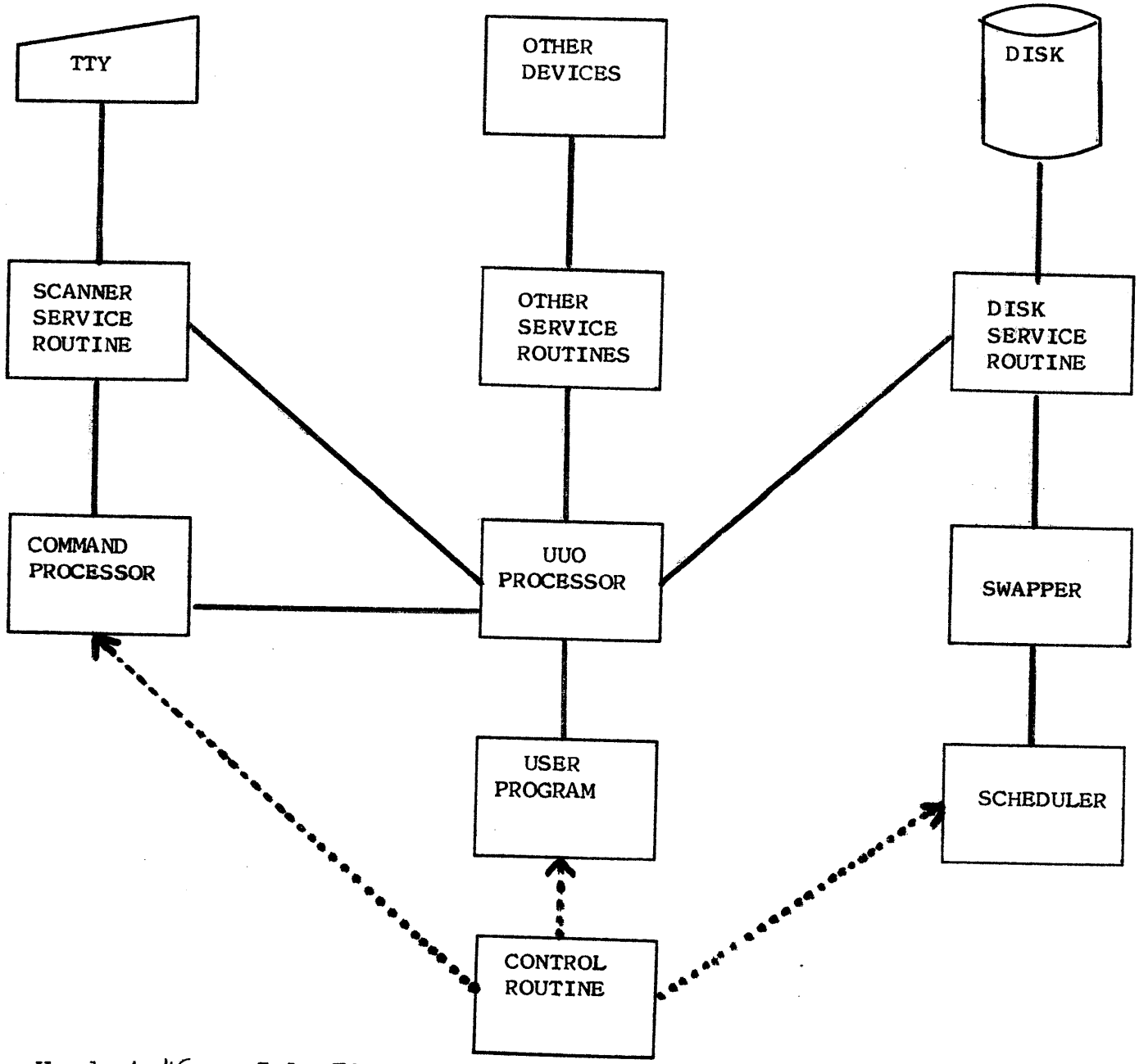
Some monitor routines make up a cycle, which is repeated on a fairly regular basis. Other routines are noncyclic - operate only when called for by a device interrupt or a UUO. The cycle comprises the Command Processor, the Scheduler, the Swapper, and the Context Switching Routine. Context switching consists of saving all conditions necessary to restart the program interrupted by the last clock interrupt, and restoring those conditions for the job selected to run next. When this series of functions has been finished, control is given to the restored user job, which can then run until the next clock interrupt, or

tick. Upon the next clock tick, the cycle is repeated.

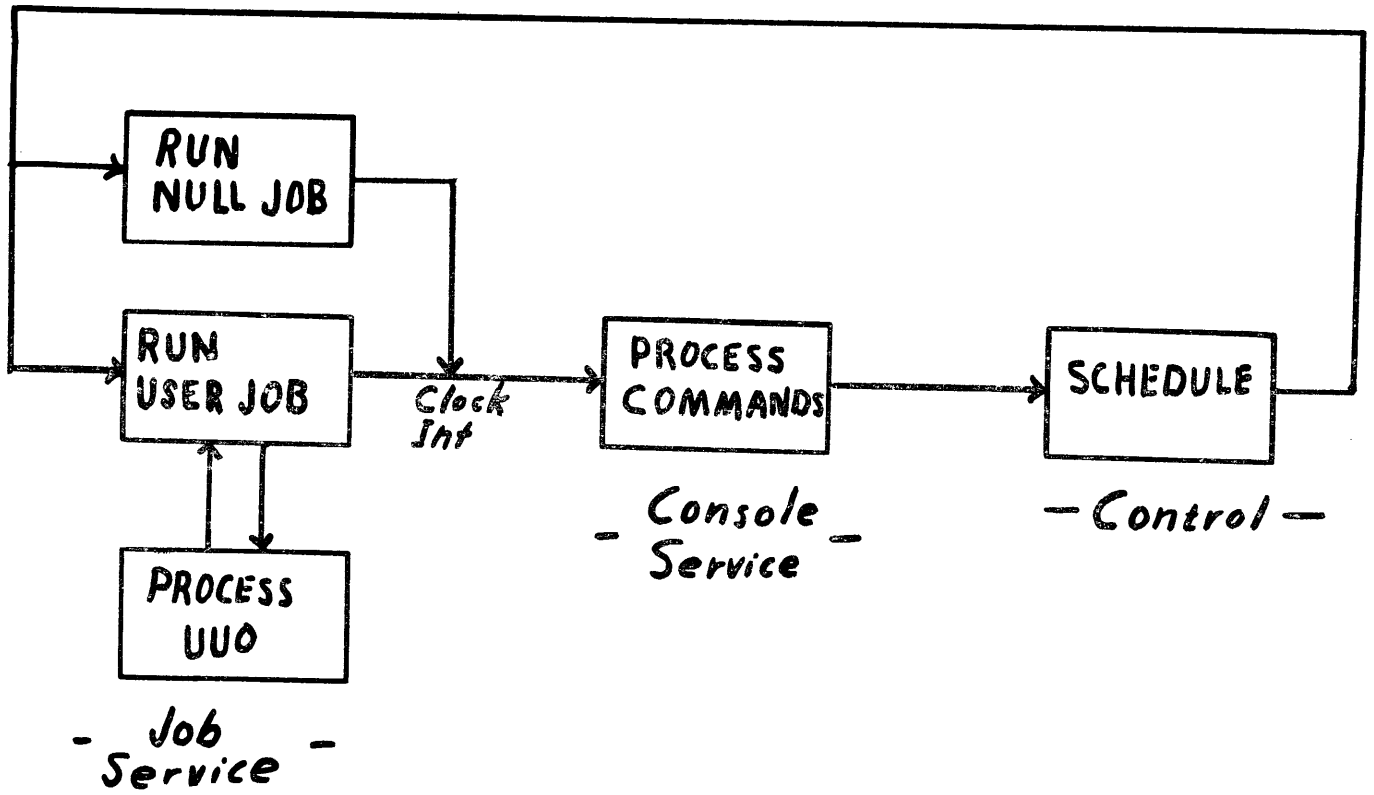
Noncyclic routines include the UUO processor and all the I/O device interrupt routines. UUOs are the sole means by which a user program can give control to the monitor in order to have some function accomplished. Each time a UUO is executed, control passes back to the monitor. When the UUO has been completed, control is returned to the user program. I/O devices are initially started by the execution of UUOs. Once started, an I/O device can cause an interrupt at any time -- during a user job, during execution of a UUO, or during any of the cyclic routines. When any I/O interrupt occurs, control passes to the routine to process interrupts from that specific device. The interrupt routine performs its function -- usually an I/O transfer -- and then returns control to the interrupted routine. The interrupt routine is responsible for restoring all conditions so that the interrupted routine is unaffected by the interrupt.

Overall, the monitor can be envisioned as an operator which performs specific functions in response to specific events which occur within the system. A regular, periodic event -- the clock interrupt -- drives the cyclic routines. The UUO Processor responds to UUOs being executed by a program, and the Command Processor responds to a user's typing a command on his Teletype. Each I/O device interrupt is an event which results in the operation of a corresponding interrupt routine. There is a well-defined function which the monitor performs in response to each system event, but a given event will not necessarily result in the same action every time it occurs. Rather, the specific action taken may depend on the state of the system. The system state is a many valued variable depending on the past history of the system. In general, it can be considered to be represented by the information stored in the many tables and data items in the monitor, and in the registers of the peripheral devices. Given the state of the system, the monitor will perform a specific predictable function in response to any specific event.

In summary, the monitor both controls user jobs and provides various services to them. Many control functions are performed on a regular basis in response to a clock interrupt. A user job is given control for one cycle, and then stopped in such a way that it can be restarted later. Since every interrupt routine eventually restores control to the interrupted program, user programs need never be concerned about interrupts -- either from the clock or from I/O devices. The only monitor action detectable to a user program is the processing of UUOs. These are processed upon request and have the appearance of single instructions to the user program.


TTY
OTHER DEVICES
DISK
SCANNER SERVICE ROUTINE
OTHER SERVICE ROUTINES
DISK SERVICE ROUTINE
COMMAND PROCESSOR
UUO PROCESSOR
SWAPPER
USER PROGRAM
SCHEDULER
CONTROL ROUTINE

MONITOR CYCLE



2. Building a Monitor

Readings

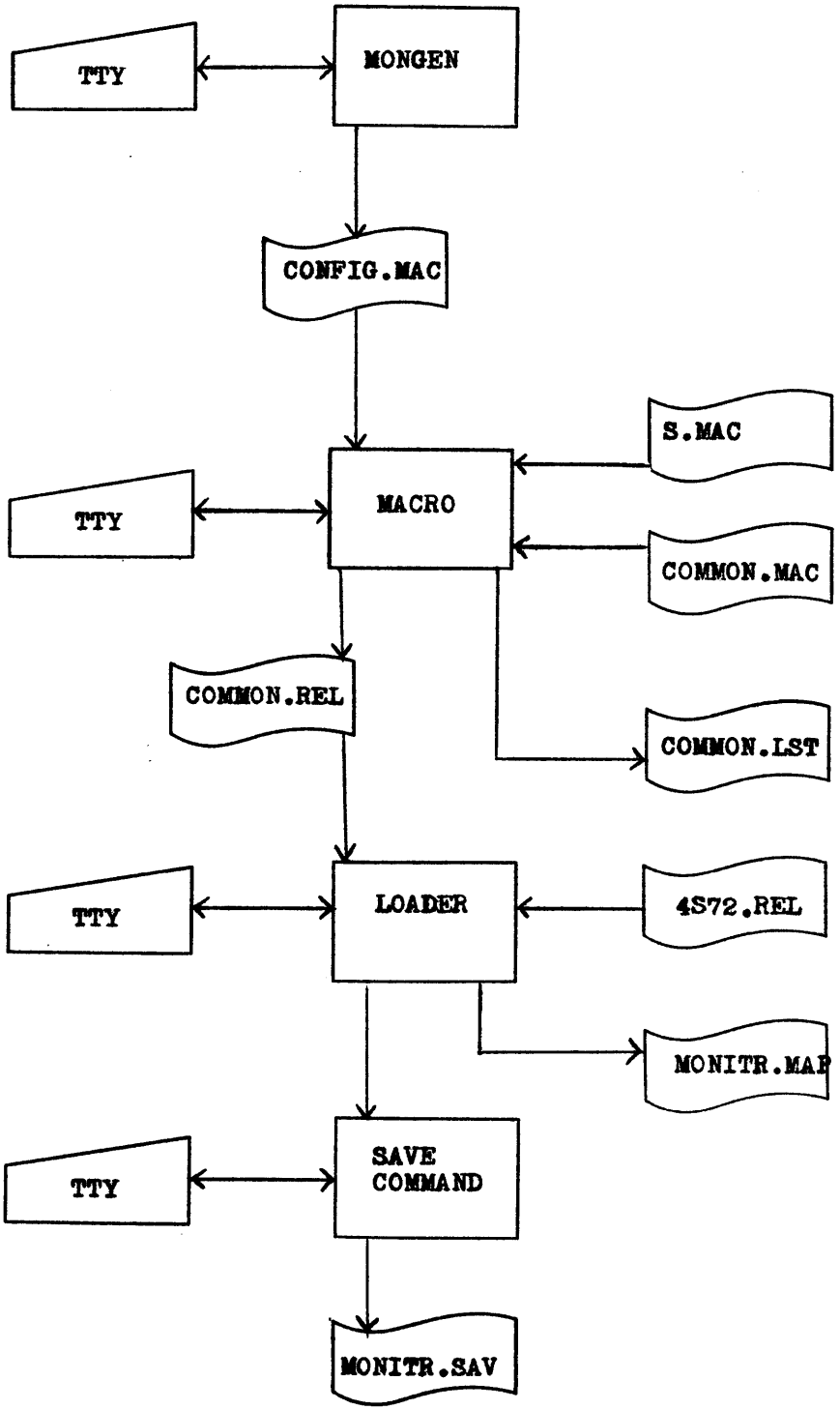
Memo "MONITR.OPR"
Sections 1 and 2
24 pages

Flowcharts and Diagrams

Handout 21 - Generation of a Monitor

Project 1 - Building a Monitor

MONITOR GENERATION



Monitor Handout 21 -- July 70

PROJECT - BUILD A MONITOR

1. Given a binary file, consisting of an appropriate monitor and a copy of TENDMP on either paper tape or DECtape, load and start the monitor.

   References:

   1. MONITR.OPR, Section 1.1.
   2. PDP-10 Reference Handbook, TENDMP, p 621.

   Procedures:

   1. Follow procedures on p 6.1 of MONITR.OPR to read in and start your monitor.
   2. Try several commands to assure yourself that it works properly.

2. Given a library file, 4S72.REL, and source files COMMON.MAC and S.MAC, build a new monitor for a specific configuration. The configuration is as follows:

   1. 10/50 Swapping System
   2. RD10 Disk only, for both swapping and storage
   3. 20 jobs attached
   4. Job size can be all of core
   5. PDP-10 Processor
   6. 2 Relocation register software
   7. No more high segments than jobs
   8. Load EXEC DDT, local symbols, user DDT
   9. Name of system, PROJECT 1 MONITOR
   10. Serial number of CPU: 16
   11. System device: DSK
   12. COMMON.MAC not edited for the TTY configuration
   13. DC10 Data line scanner only - no 68ØI
   14. Full Duplex Software
   15. 3 DC1ØB 8-line groups
   16. No DC1ØE dataset control groups
   17. No data set lines
   18. No lines with hardware tabs
   19. Remote lines: 16-21
   20. No half duplex lines

21. 1 PT reader
22. 1 PT punch
23. No plotter
24. 1 line printer
25. No card reader
26. No card punch
27. No display
28. 5 DECtapes, TD1Ø Control
29. No mag tape
30. No pseudo-TTY
31. No CCL commands in core
32. No special symbols or devices

References:

1. MONITR.OPR 2.0 - 2.5
2. PDP-10 Reference Handbook
   TENDMP, p 621
   MACRO, p 273
   LOADER, p 526

Procedures:

1. Load TENDMP (32K).
2. Use TENDMP to read in SPMON 32K.
3. Under control of SPMON, run program MONGEN.
4. Build CONFIG.MAC on a scratch DECtape.
5. Still under control of SPMON, run MACRO to assemble COMMON from the following source files:
   a. S.MAC
   b. CONFIG.MAC
   c. COMMON.MAC
6. Run LOADER, to load the COMMON.REL which you have just produced, with a library search on file 4S72.REL.
7. Save the monitor which you have just loaded on a scratch DECtape as P1MON.SAV.
8. Following the procedures of Part 1 of this project, load and start this monitor. (Do <u>not</u> refresh the disk!)
9. Try several commands.

3. Monitor Data Base

Readings

Handout - The Monitor Data Base
Handout - Queue Transfers

Table Descriptions

JBTQ - Job Queues Table
JBTADR - Job Address Table
JBTSTS - Job Status Table
JBTSWP - Job Swap Table
JOBDAT - Job Data Area

Other References

Handout 13 - Job Queues

Written Assignment

Question Set 1 - Introduction to the Monitor
Question Set 2 - Job Queues

THE MONITOR DATA BASE

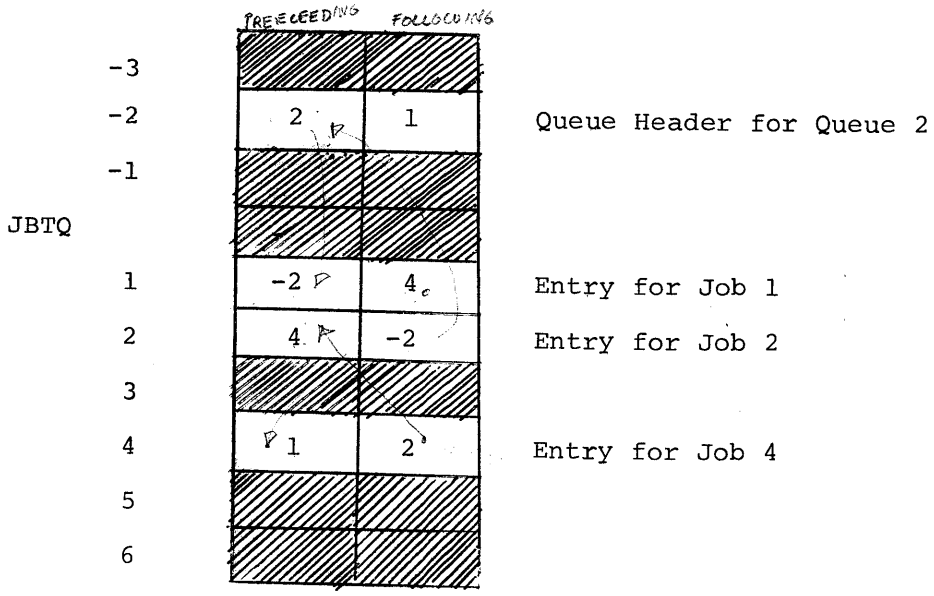
In order to control and operate user programs, the monitor must have available a considerable amount of information about the current user jobs. This information makes up a data base which is used throughout the monitor. The data base consists primarily of tables having an entry for each job number in the system, and in order of job number. Some tables also have entries for each high segment number. (High segment numbers start after the highest job number.) Several of the most significant tables are described below.

The monitor keeps all jobs in groupings called queues. A queue is simply an ordered grouping such as the waiting line at the coffee machine. There are a number of queues corresponding to the various things for which a job might be waiting. Jobs waiting for CPU time are in processor queues; jobs waiting for sharable resources are in various sharable resource wait queues; those waiting for completion of I/O are in I/O wait queues; etc. Every job number is in one and only one queue; there is a Null Queue for job numbers not currently assigned. In some cases a job's position in its queue is significant; in other cases it is not. For example, in a sharable resource wait queue, the first job has highest priority for the resource when it becomes available. But in the I/O Wait Queue a job is waiting for some device to fill or empty a buffer and must remain there until the I/O operation is completed. The position of a job in the I/O Wait Queue is of no significance.

All the job queues are maintained in a single table, JBTQ. JBTQ has one entry for each job number plus one entry for each queue. Queues are given negative numbers and have entries starting just before the base address, JBTQ, and extending in the negative direction. The table entry corresponding to each job number is at the position relative to JBTQ corresponding to that job number. A queue is represented by a chain of entries through the table. The chain begins at the entry corresponding to the queue number, called the queue header. From there it proceeds in order to the entries corresponding to each job in the queue. From the last job's entry the chain goes back to the queue header, completing a ring of entries. Each entry, queue header or job number entry, consists of pointers to the following and preceding entries. The LH contains the position of the preceding entry; the RH contains the position of the following entry. This position value is also the job number or queue number corresponding to the entry to which it points.

Example: Suppose queue 2 contains jobs 1, 4, and 2, in that

order. Queue 2 would be represented in JBTQ as follows:



The core location and length of each job and high segment in core are contained in the table JBTADR. These values are used to set up the hardware relocation and protection register when a job is set up to run. If a job is swapped out, its JBTADR entry is zero. The entry is updated whenever a job is swapped in or moved to a new location in core.

The Job Status Table, JBTSTS, contains status information about each job and high segment. The entry format differs between job number entries and high segment entries. For job numbers, the right half contains the quantum run time. This value is initialized when a job is put into a run queue, and is decremented each time the job is given a time slice. If the quantum run time expires, the job will be requeued to another run queue and the quantum run time reset. Positive quantum run time does not assure a job that it will get any particular time slice or that it will not be swapped out. Its only purpose is in circulating jobs around in the run queues.

The LH of a job number entry contains a number of bits with specific meanings. The RUN bit is set whenever the user wants a program to run, and no error has occurred. The RUN bit is cleared when an error is detected or the user indicates that he wants to stop the program. The Command Wait bit, CMWB, is set when the user has typed a command which cannot be executed until the job is in core, and the job is swapped out. This bit will cause the job to be put into the Command Wait Queue, where it will have very high priority for being swapped in. The Swap bit, SWP, is set when the job is swapped out. The JRQ bit indicates that

the job needs to be requeued, or moved to a new queue. When the JRQ bit is se the Wait State Code, bits 1Ø-14, indicates the queue which the job should be pu into. Normally, if JRQ is not set, the Wait State Code indicates which queue the job is in. However, there are three Run Queues, and all have the Wait State Code of zero. Also, jobs are put into the Stop Queue and Command Wait Queue according to the values of the RUN and CMWB bits. These jobs retain their previous Wait State Code, so that they may be returned to their previous queue.

The JBTSWP table contains information necessary for swapping jobs into and out of core. For a job which is in core, the LH contains its In-Core Protect Time. The In-Core Protect Time is set when a job is swapped in, and decremented on each clock tick until it reaches zero. As long as the job has positive In-Core Protect Time, it will not be swapped out. For swapped-out jobs, the LH of the JBTSWP entry tells where to find the job. If the job was written on the swapping device in a single group of contiguous blocks, this will be the disk address where it is written and bit Ø will be Ø. If the job had to be fragmented, or written in several separate disk areas, bit Ø is 1 and the bits 1-17 contain the address of a table in core. This table, called a Fragment Table, tells the location and length of each disk area. The Out-Core Image Size, in bits 19-26, is the size of the job as written on disk. The In-Core Image Size is the size of the core area which this job should be read into. Normally, these two size are equal. They will differ if the job is expanding in core size.

The first 140_8 locations of each user area contain the Job Data Area. This area is primarily a storage area for information about this job while it is inactive. There is an area where the hardware AC's are saved for the job while a UUO is being executed. There is another area where the AC's are saved while the job is not running. There is a considerable amount of additional information stored here which will be of value to various sections of the monitor.

The tables described above, and a number of others as well, represent the state of the system to the monitor. Their contents frequently affect the action taken by the monitor as a result of a particular event, and frequently the action taken includes making changes to the tables.

QUEUE TRANSFERS

Queue transfers are accomplished by routine QXFER in SCHED. A transfer table specifies the type of transfer to be made. AC J (alias ITEM) must contain the job number to be requeued. If the destination queue depends on the source queue, the source queue number will be in AC T2 (alias TAC).

The simplest queue transfer is the fixed destination transfer. The same code is also used as the final part of the other transfers.

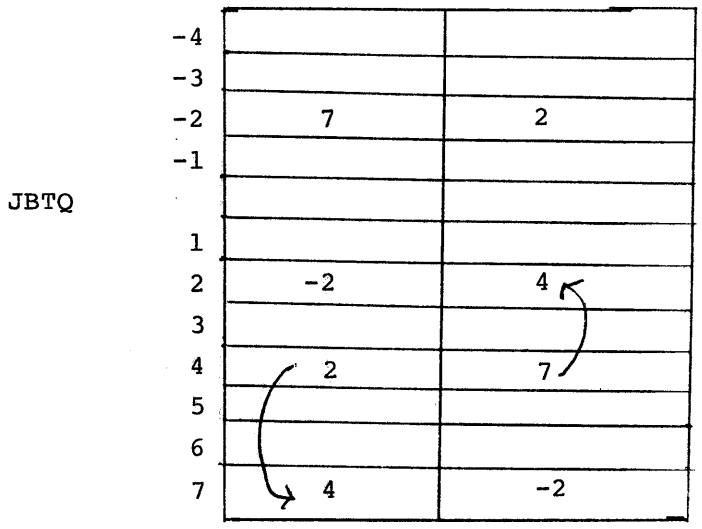
The transfer table is as follows:

Ø for xfer to beg of Q - for xfer to end of Q	Adr of QFIX
Quantum Run time if positive	Destination Queue #

(QXFER) AC Q is loaded from the second word of the Transfer Table. Then there is a jump to the routine whose address is in the RH of the first word

(QFIX) First the job is deleted from its present queue. This is done by giving its "following job" entry to the preceding job and its "preceding job" to the following job.

Example: Deletion of Job 4 from its queue



Next the job will be inserted into the chain of entries which make up the destination queue. The job will be inserted following either the first link (i.e., the queue header) or the last link - depending on the value of the "PLACE" entry in the Transfer Table.

AC J points to the entry which is to be inserted. AC Q must point to the entry which this entry will follow.

AC Q initially points to the queue header, which is correct if the insertion is to be made at the beginning of the queue; if the insertion is to be at the end of the queue, AC Q is backed up one entry, to point at the last entry. This is done with the instruction:

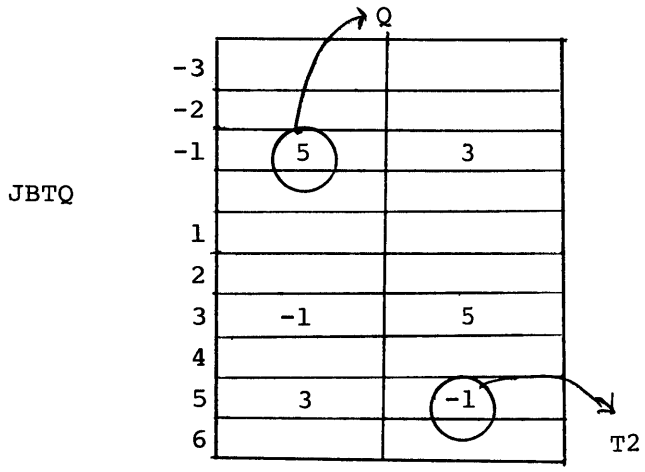
(QFIX+5) HLR Q, JBTQ (Q)

(QFIX+6) AC T2 is loaded with the index of the entry in front of which the insertion will be made. This is the value in the RH of the entry Q now points to.

Now the new linkages are set up with four easy instructions:

(QFIX+7)	HRRM	J, JBTQ (Q)	New "preceding" entry gets (J) as following entry
	HRLM	J, JBTQ (T2)	New "Following" entry gets (J) as preceding entry
	HRRM	T2, JBTQ (J)	This job's entry gets (T2) as following entry
	HRLM	Q, JBTA (J)	This job's entry gets (Q) as preceding entry

Example: Insert Job 1 at end of queue 1:



Q initially contains -1, the index of the queue into which the insertion is to be made.

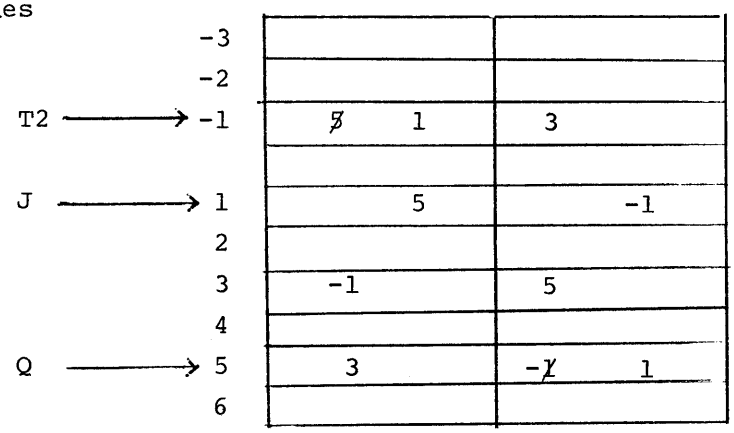
Since the insertion is to be at the end of the queue, AC Q is backed up one entry, by loading it from the LH of the entry it points to.

J points to the entry to be inserted
Q points to the entry after which it will be inserted
T2 points to the entry before which it will be inserted

(J) is placed into the RH of JBTQ (Q) and LH of JBTQ (T2)
(T2) is placed into the RH of JBTQ (J)
(Q) is placed into the LH of JBTQ (J)

Final result

AC values



(QFIX+11) if QUANT = Ø, there is a jump to the routine exit.

(QFIX+12) Otherwise, QUANT is inserted into the RH of the JBTSTS entry for Job (J) - i.e., Job J's quantum run time is reset. Also the Wait State Code in that word is set to Ø

(QX3) Routine returns to calling routine with a POPJ.

There are two additional types of queue transfers:

1. Destination queue depends on job size.
2. Destination queue depends on source queue.

Both types of transfers depend on tables to supply the destination queue. Th "Queue Progression Table" gives destination queues corresponding to various source queues. The " Job Size Queue Table" gives destination queues corresponding to various job sizes.

Either type of table may have associated with it a "Quantum Time Table" with an entry giving a quantum time value corresponding to each destination file.

For both these types of transfers, the Transfer Table is as follows:

Ø for xfer to beg of Q - for xfer to end of Q	Adr of QLINK or QJSIZ
Adr of Quantum Time Table if positive	Adr of Table giving Dest. Q

These routines scan through the appropriate table for the correct destination queue. Then the destination queue number and the quantum run time (if specified) are copied from their tables into AC Q. Now the QFIX routine is used to do the transfers.

JOB QUEUES

(Queue Number (Wait State Code)	Label	Meaning
0	RNQ	Run
1	WSQ	I/O Wait Satisfied
2	TSQ	TTY wait satisfied
3	STQ	System Tape Wait
4	AUQ	Alter UFD
5	MQQ	Monitor Disk Buffer
6	DAQ	Disk Storage Allocation
7	DTQ	DECtape Controller
10	DCQ	Data Controller-Magtape or DECtape
11	MTQ	Magtape Controller
12	IOWQ	I/O Wait
13	TIOWQ	TTY Wait
14	SLPQ	Sleep
15	NULQ	Null
16	STOPQ	Stop
17	PQ1	Processor 1
20	PQ2	Processor 2
21	PQ3	Processor 3
22	CMQ	Delayed Command

Notes:

1. RNQ, WSQ, and TSQ never actually hold jobs. The queues are defined only to define the corresponding Wait State Codes.
2. The values of PQ1, PQ2, PQ3, CMQ, and STOPQ are never used as wait state codes. Jobs in any of the PQ's have wait state codes of ØØØ. When jobs are put into CMQ or STOPQ they retain their previous codes, so that they can be returned to their previous queues.

QUESTIONS ON INTRODUCTION TO MONITOR

1. List five kinds of events to which the monitor responds.

2. What is meant by a queue?

3. List four kinds of queues into which the monitor might put a job.

4. List four events which could result in a job being requeued.

5. What are some of the functions of the monitor which are not under control of the user?

6. What are some functions which the monitor performs in response to a user or user program request?

7. In which source file is all configuration dependent code?

8. Which monitor "program" (Rel File) is the only one which varies with configuration for the same version of the monitor?

9. What is the purpose of a parameter file such as S.MAC?

10. What prevents unneeded routines from being included in the monitor when it is built?

11. Where can the address and length of each job be found?

12. What would this instruction accomplish:

```
JRST    CPOPJ
```

13. With what type of system would each of the following monitors be used?
    a. 4S72
    b. 4D72
    c. 4N72

14. What is the primary function of TENDMP?

15. What is the function of SPMON?

QUESTIONS ON THE JOB QUEUES

Questions 1 - 4 refer to a Job Queues Table as shown on the following page. Indicate your answers to 2, 3 by making appropriate changes to the given tables. No question uses the answers of a previous question.

1. List the job numbers found in each queue, in order first to last.

2. Show the changes which would be made to the table if QXFER were called with the following transfer table specified.

Ø	QFIX
-1	-3

AC J/ 1
AC T2/ 1

3. Show the changes that would be made to the table if QXFER were called with this transfer table:

4ØØØØØ	QLINK
-1	QPROG

AC J/ 5
AC T2/ 3

QPROG:

-1	-3
-3	-2
-2	-1

4. List, in order, all the job numbers which will be returned by successive scans of the above Job Queues (before changes) by QSCAN, according to the following scan table.

-3	- QFOR
-1	- QBAK1
-2	- QBAK
-1	- QFOR1

5. List the code that would be generated by the standard QUEUES macro in the following context:

```
DEFINE      X (A,B)
<   A'C:          XWD      A'Q,Ø  >
QUEUES
```

6. Why is the ring of jobs in a queue linked in <u>both</u> directions?

Answer #2

JBTQ		
-3	3	5
-2	-2	-2
-1	2	4
	Ø	Ø
1	4	6
2	6	-1
3	5	-3
4	-1	1
5	-3	3
6	1	2

Answer #3.

JBTQ		
-3	3	5
-2	-2	-2
-1	2	4
	Ø	Ø
1	4	6
2	6	-1
3	5	-3
4	-1	1
5	-3	3
6	1	2

4. I/O Processing

Readings

Handout "I/O Processing"
This is a brief introduction, considering only buffered I/O on I/O bus devices. Dump mode I/O and data channel devices are not considered.

Program Logic Manual, Memo 8, p 1-5
This memo is included in Section 8, Device Service Routines.

Table Descriptions

Buffer Ring
Buffer Ring Header
Device Data Block
JDA - Job Device Assignment Table

Diagrams

Handout 44 Buffer Pointers
Handout 7 I/O Linkages

Written Assignment

Questions on Input-Output

I/O PROCESSING

All Input and Output for user programs is done by the monitor, at the request of the user programs. The user programs execute UUO's to inform the monitor of their requirements, and the monitor then handles all actual communication with the device. Normally, buffers are used to allow the user program and the monitor I/O routines to operate asynchronously. While the user program uses one buffer, the monitor fills or empties another buffer as interrupts occur from the device.

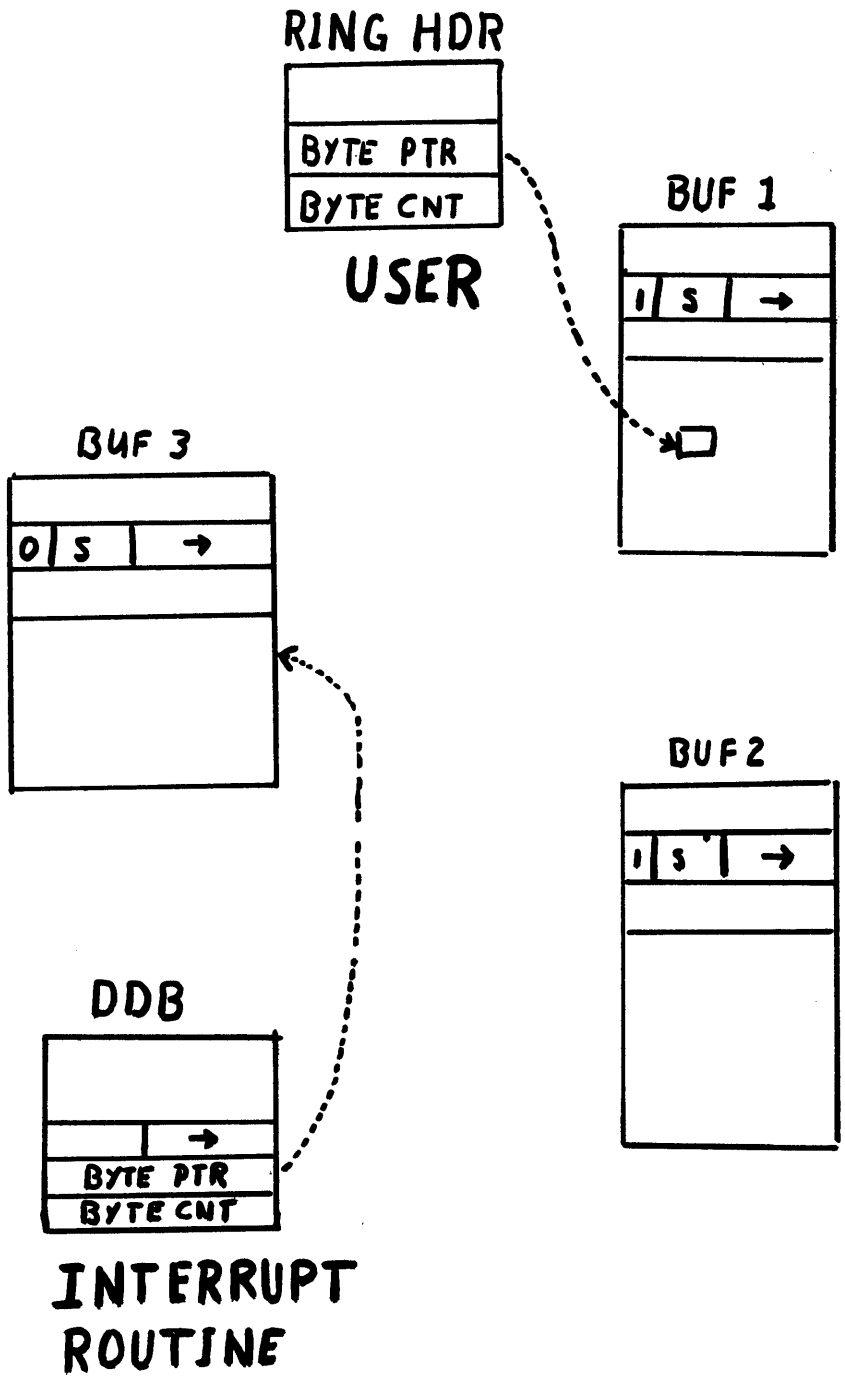
Buffers are set up, by the monitor, in the user's area. The buffers to be used for one device or file are linked together to form a ring structure. Each buffer begins with an area containing descriptive information about that buffer. This includes a pointer to the next buffer and a bit, which indicates whether or not the buffer is filled with data. Associated with each buffer ring there is a ring header containing the information which permits the user program to access its current buffer. There is a byte pointer and a counter which are initialized for a new buffer each time the program executes a UUO. The user program must keep these up to date as it works through the buffer, and execute another UUO when it has finished that buffer.

For each different device in the system there is a Device Data Block in the monitor. The Device Data Block, or DDB, contains a great deal of information relating to that device. Included in the DDB are the address of the buffer currently available to the interrupt routine, and a byte pointer for the interrupt routine to access the next byte in its buffer. There are also pointers to the ring header and to the device service routine for this device.

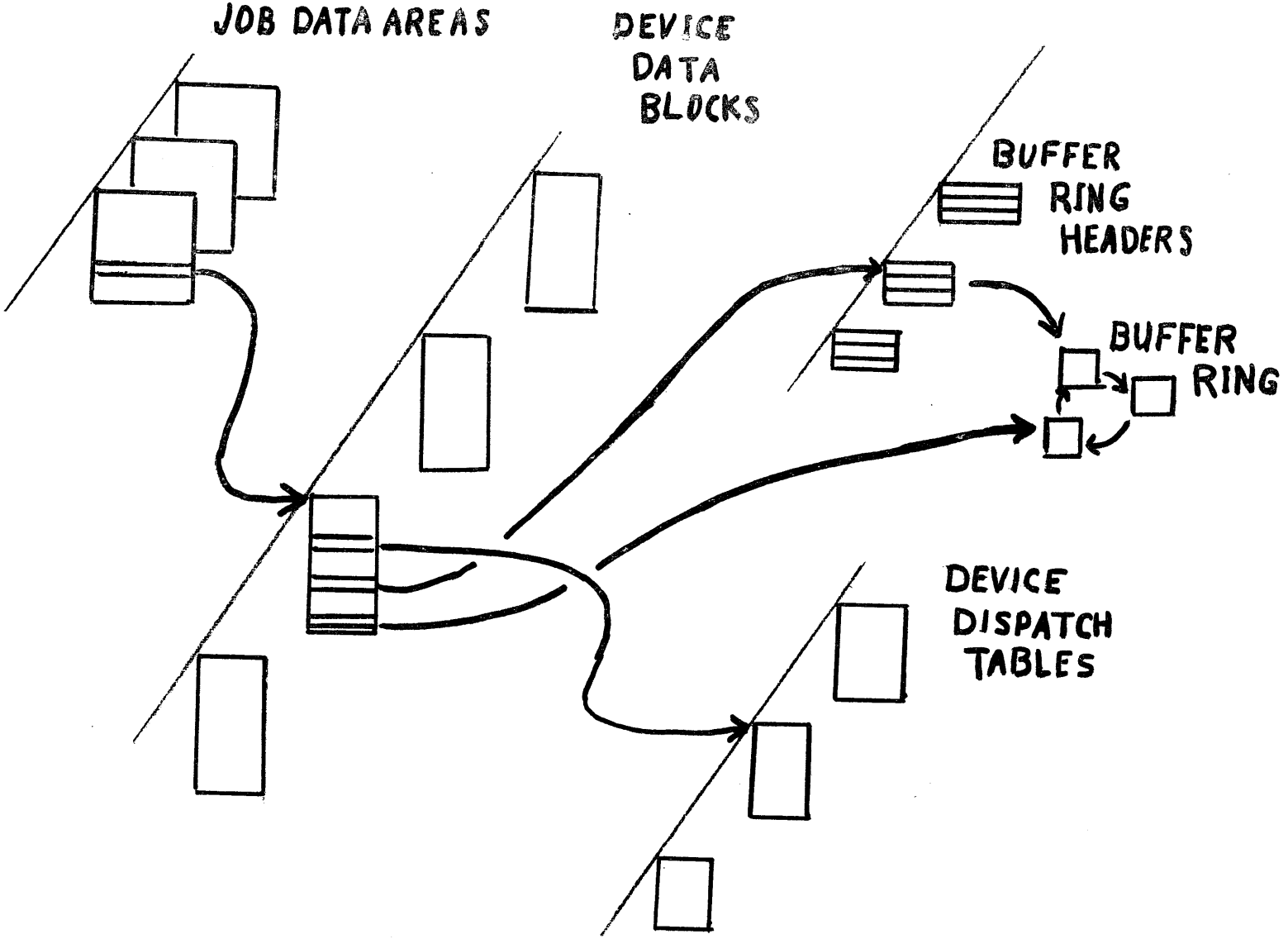
When the user program executes a UUO, it makes the buffer which it was using available to the monitor and requests another for its own use. If the monitor has finished filling or emptying the next buffer of the ring, the only action taken by the UUO routine is to advance the user's pointers to the next buffer. If the next buffer is still in use by the monitor, the program must be stopped and put into I/O Wait. When the interrupt routine finishes that buffer it will cause the program to be taken out of I/O Wait, and the interrupted UUO will be completed. The user program can assume that each UUO makes the next buffer available to it, because control never returns to the instruction following the UUO until that buffer is available.

Each time a device interrupt occurs, control passes to the routine to service that device. Another byte is read or written, upon each interrupt, until the interrupt routine finishes its current buffer. When the buffer is finished, the interrupt routine makes it available for the user program and advances its own pointer to the next buffer. If the user program has released that buffer, by requesting another, the interrupt routine proceeds to fill or empty it. If that buffer is still in use by the user program, the device must be stopped so that no more interrupts will occur until there is a buffer available. Output devices are restarted by the UUO routine as soon as the user executes the next UUO, making a bufferfull of data available to be written. Input devices are restarted by the UUO routine when the last full buffer is made available to the user program, assuring the maximum number of free buffers before restarting the device.

BUFFER POINTERS



JOB - DEVICE LINKAGES



QUESTIONS ON INPUT-OUTPUT

1. How does the monitor know which device a program refers to when it does a UUO such as the following:

```
IN      2,
```

2. Explain the process by which the monitor could scan thru all the Device Data Blocks.

3. Where are the Device Data Blocks located?

4. What is the purpose of the Device Data Blocks?

5. How can the monitor tell if a device has been assigned by a console? - inited by a program?

6. How can the monitor tell if an Input Close has been done on a given software channel?

7. What location gives the next address in an input buffer to be used by the monitor? - by the user?

8. How is the "software channel" specified in an assembled (i.e., machine language) I/O UUO?

9. Why would the monitor want the DDB's linked together?

10. How does the DDB address get into a Job Device Assignment Table entry?

5. The Monitor Cycle

Readings

Handout "The Monitor Cycle"

Table Descriptions

JOBDAT - Job Data Area

Flow Charts

Handout 10 - the APR Interrupt Routine
Handout 6 - The Clock Cycle
Handout 23 - Wait Routines

Timing Diagrams

Handout 36 - Clock Tick
Handout 37 - Scheduling
Handout 35 - UUO Interrupted
Handout 38 - Clock Tick During Interrupt

Monitor Listings

CLOCK1	Routine APRINT	lines 54 - 86
	Routine RSCHED	lines 315 - 479
	Routine CLKINT	lines 281 - 314
	Routine USCHED	lines 264 - 280
	Routine WSCHED	lines 247 - 263

Written Assignment

Questions on the Monitor Cycle

THE MONITOR CYCLE

The outermost loop of the Monitor is the control routine beginning at RSCHED in CLOCK1. All cyclic functions are performed either in this routine or in a subroutine called by it. The cycle is repeated according to events which occur in the system, normally at least sixty times per second. On each cycle the following functions may be performed:

1. Time accounting
2. Processing of timing requests
3. Processing of a console command
4. Scheduling and swapping
5. Context switching
6. Operation of a user program.

The first three functions are performed only when the cycle is started as a result of a "clock tick."

These functions should not be thought of as equals, either in time consumed or in importance. Hopefully, most of the time of each cycle is spent in user program operation. This is the system's reason for existence; the other functions can be thought of as system overhead. Of the first five functions, console command processing, and scheduling and swapping will use most of the time. The amount of time spent in each will, of course, depend on the system load and the nature of the jobs. Context switching, time accounting, and timing request processing are logically autonomous functions, performed each clock tick, but the amount of time they use in each cycle is normally insignificant.

Time accounting is the first function in the cycle. Several different time totals are updated, as appropriate:

1. Lost time - time spent running the null job while there were jobs in the processor queues.
2. Current job's incremental run time (time since the job last cleared the total).
3. Current job's total run time.
4. Current job's total (time) X (core size).

There are also checks for end of the day and end of the month. The monitor's date and time are corrected, if necessary.

Timing requests are processed next. A monitor routine can request that a specified subroutine be run after some amount of time has passed. To do so it adds an entry to the Timing Request Queue, CIPWT. (See table description for details.) At this time the delay time for each entry is decremented. If the delay time for any entry goes negative, the specified subroutine is run and the entry is removed from the queue.

The monitor performs some functions only once per second. A counter is decremented to indicate when another second has elapsed. If the count reaches zero, the once-per-second tasks are performed. One such function is checking for hung devices.

Next, if there are any console commands awaiting execution, there is a dispatch to the command processing routine, COMCON. This is a major section of the monitor, and is treated in detail in its own section. COMCON will interpret and execute one console command, and then return control.

The scheduler is called next. This also is a major section of the monitor. The scheduler requeues any jobs which have changed status during the last cycle and determines which user job will be allowed to run next. It also calls the swapping routine. Eventually, it returns control. Note that although the scheduler determines which job is to run next, it does not give control to that job.

Finally, arrangements are made to run the user job selected by the scheduler. This process is called context switching. If the job to run next is the same one which ran last, all that is necessary is to restore its accumulators, processor flags, and program counter -- the "hardware state." If a new job is to run, certain software information must be saved for the old job and restored for the new job. This information includes the job's I/O device assignments and the address at which control is to be returned to it. Then the hardware AC's of the new job are restored and control is given to it with a JEN instruction. The JEN restores the new job's PC and processor flags and dismisses the interrupt. The user job then has control and can run for the rest of this cycle.

A new cycle will be started as the result of one of three possible events:

1. A clock interrupt occurs while the CPU is in user mode.
2. The UUO processor finishes a function, and the clock interrupt occurred while the UUO was being processed.
3. The user job reaches a point at which it cannot immediately continue.

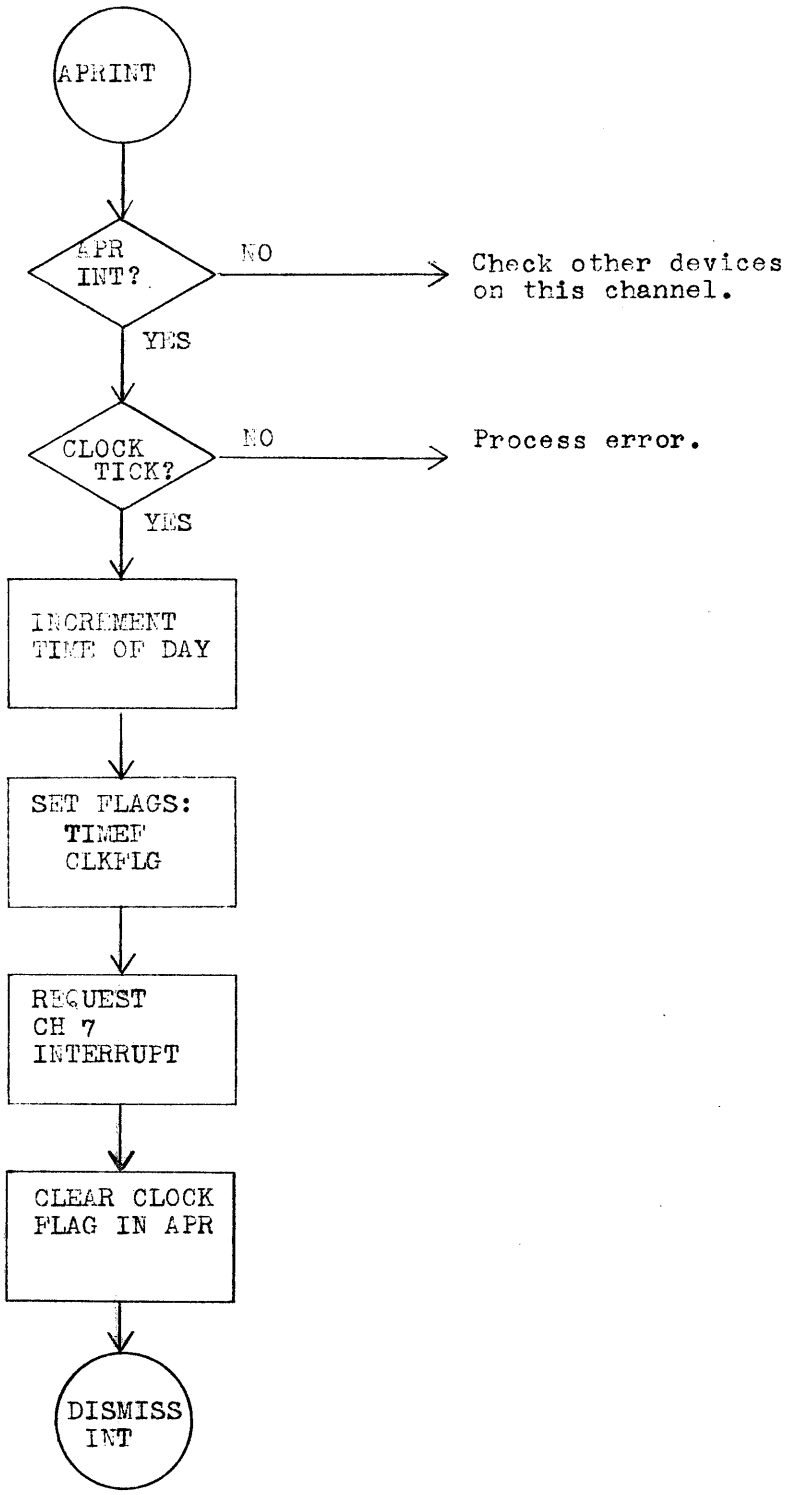
The clock interrupt is one of several interrupts caused by the arithmetic processor, device APR. The APR is always assigned to a high priority channel, so that error conditions which cause APR interrupts may be recognized immediately. However, there is no urgency for restarting the control cycle. Therefore the hardware APR interrupt is used to drive a lower priority "software" clock interrupt. The software clock interrupt is always assigned to Channel 7, so that all I/O device interrupts can take priority over it. The software clock interrupt is also requested by certain other monitor routines in order to start a new cycle before the hardware clock interrupt has occurred. The flag CLKFLG is set by any routine which requests the Channel 7 interrupt; the flag TIMEF is set only by the APR interrupt routine and indicates that an actual clock tick has occurred.

When the software clock interrupt occurs, control passes to the CLKINT routine in CLOCK1. If the interrupted program was in exec mode, the interrupt is immediately dismissed and the new cycle is delayed until the current monitor function is finished. Otherwise, the user's AC's and PC are saved and control passes to RSCHED, the beginning of the control cycle.

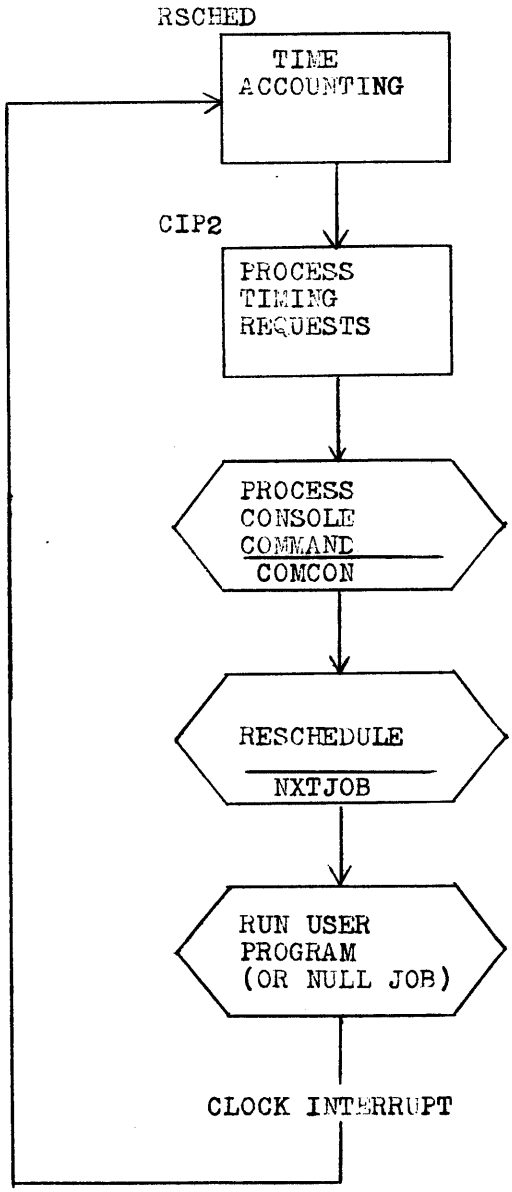
The UUO Processor checks if a clock tick occurred while it was running, before returning control to the user program. If it finds TIMEF set, it passes control to the USCHED routine, which performs essentially the same function as CLKINT. USCHED sets the user program return address to the next address in the UUO Processor. It then saves the necessary AC's and passes control on to RSCHED. Whenever the interrupted program is selected to run again, it will be restarted in the UUO Processor at the point where control is restored to the user program.

In some cases a user program may reach a point where it can not immediately continue. For example, it may execute an INPUT UUO at a time when the next buffer has not yet been filled. In such cases, the monitor routine can request a new cycle be started, so that another job may be selected to run. To do so, the monitor routine passes control to WSCHED. WSCHED, like USCHED, will set up the return address, save the AC's, and pass control to RSCHED. When the program is restarted it will be at the point where the monitor routine requested a new cycle.

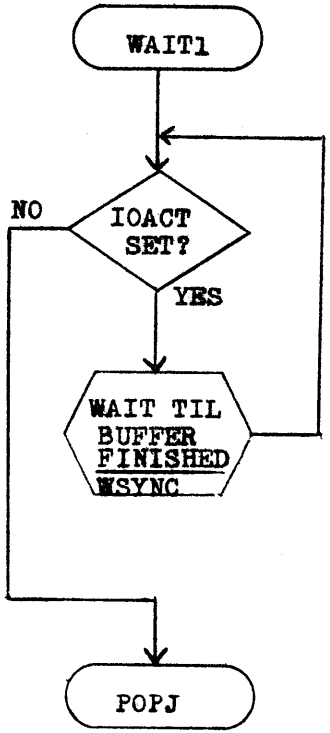
APR INTERRUPT ROUTINE

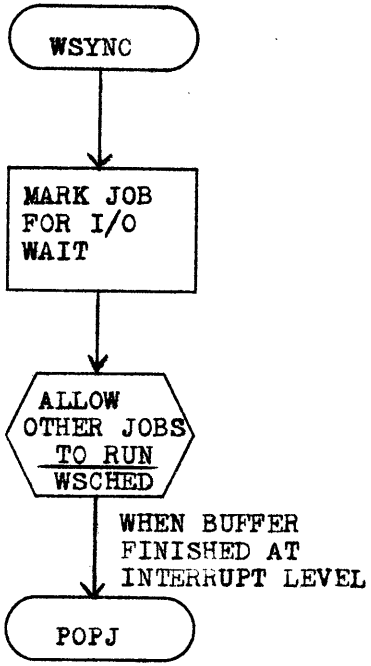


THE MONITOR CYCLE

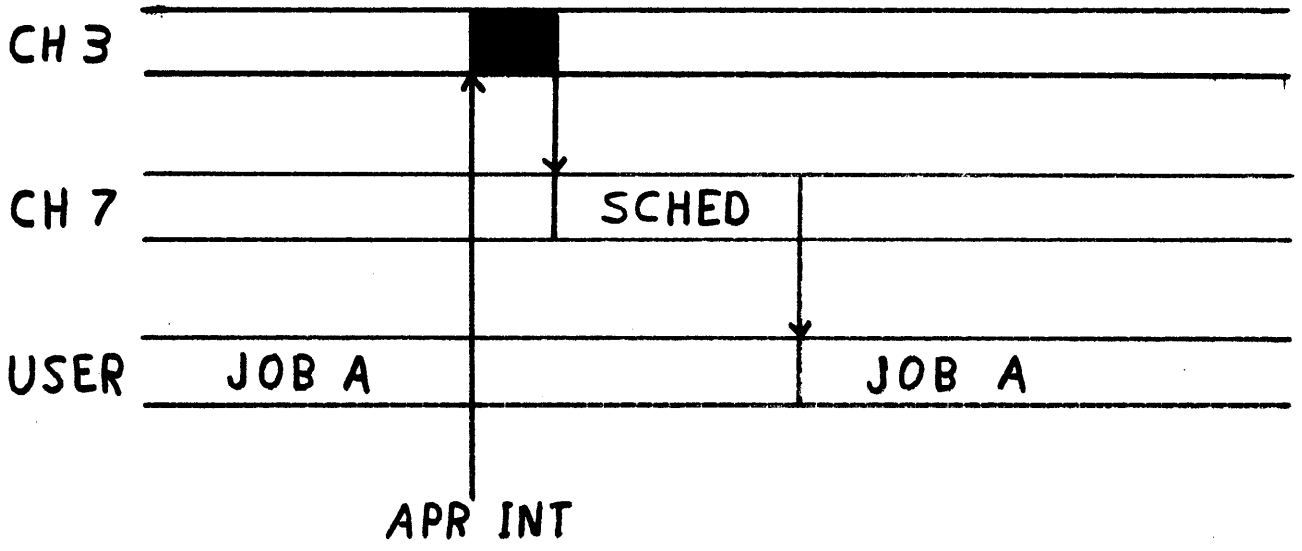


WAIT ROUTINES

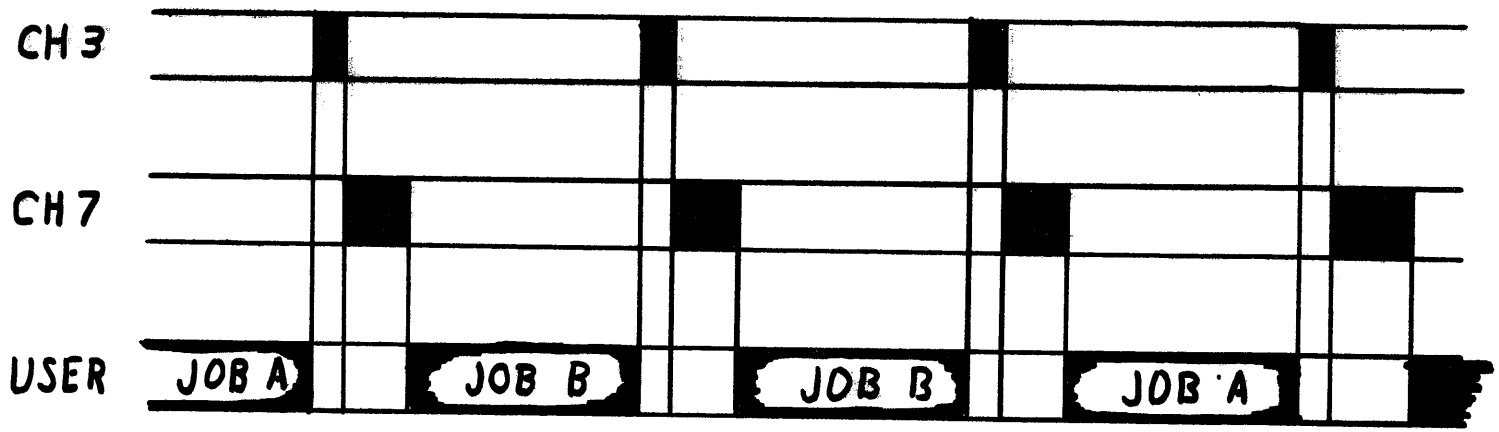




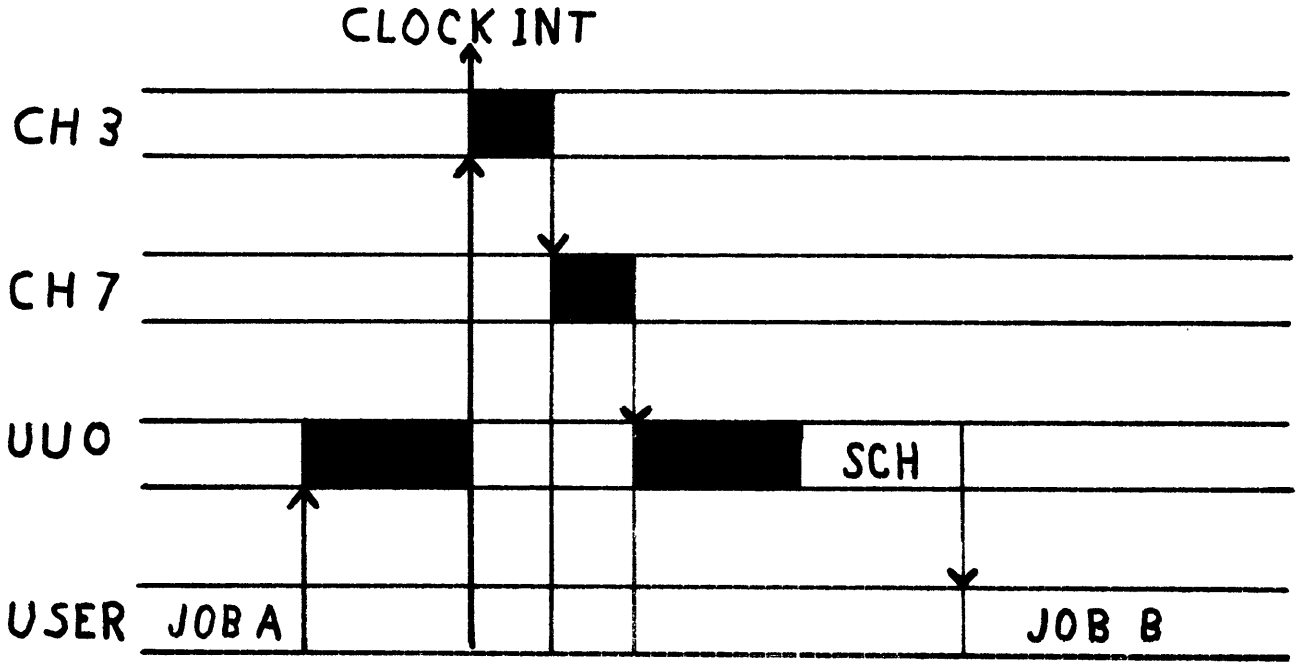
CLOCK TICK



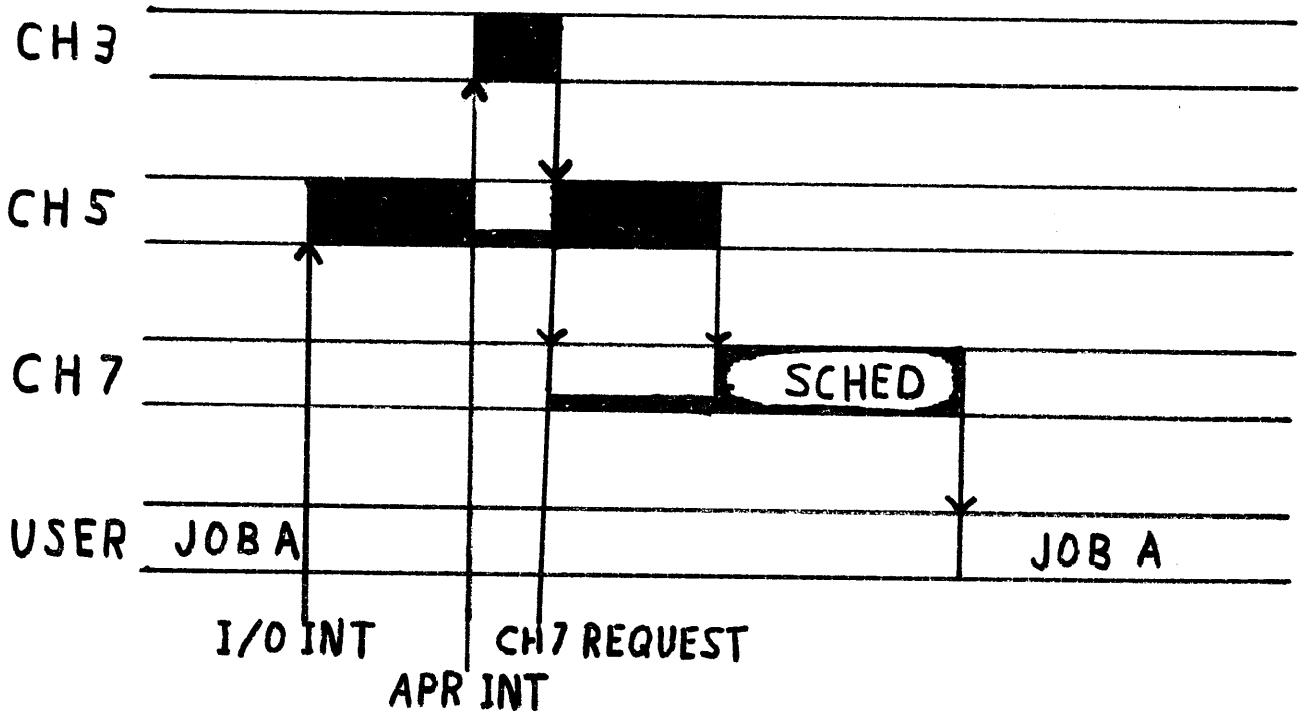
SCHEDULING



UUO INTERRUPTED



CLOCK TICK DURING INTERRUPT



QUESTIONS ON THE MONITOR CYCLE

Specify routines and line numbers for all questions that ask "where".

1. Where is control passed to RSCHED?

2. Where does the monitor give control to a user program ?

3. At what location (label) is the user program's PC stored when a channel 7 clock interrupt occurs?

4. When is a clock tick counted as "lost?"

5. Where is context switching performed?

6. In what place are a job's PC and processor flags saved while it is inactiv

7. How is the restart address set up for a program stopped at the end of UUO processing?

8. How do USCHED and WSCHED differ? How do you account for these differences?

9. What determines whether the CPU is in exec mode or user mode when control is restored to a user job?

10. When would a user job be restarted in exec mode?

6. The Command Processor

Readings

Handout "The Command Processor"

Table Descriptions

TTYTAB - TTY Table
COMTAB - Command Names Table
DISP - Command Dispatch Table

Flow Chart

Handout 19 COMCON

Monitor Listings

COMCON Lines 1 - 523

Written Assignment

Questions on Command Processing

THE COMMAND PROCESSOR

As each user types on his Teletype keyboard, the characters are stored in an input buffer in the monitor until requested as input. If a TTY is not being used for I/O by a user program, it is in "monitor command mode", and any line typed on it will be taken as a command to the monitor. A line is terminated by a break character, such as altmode or carriage return. When a break character is typed on a TTY in monitor command mode, the scanner service routine takes it as the indication that user has completed typing a command and updates the monitor data base accordingly.

There is a table, TTYTAB, with an entry for each TTY and a bit in the entry to indicate that a command is awaiting processing. The scanner service sets this bit and increments a counter, COMCNT, each time a command is completed. On the monitor cycle following each clock tick, if COMCNT is greater than zero, control is passed to the command processor to interpret and execute one command. The entries in TTYTAB are then scanned cyclically to determine which TTY has a command waiting. When a command completed bit is found, the first line in the corresponding input buffer will be interpreted as a command.

The first word is converted to sixbit format and looked up in a table of command names, COMTAB. The lookup is successful if any command exactly matches the command that was typed, or if one and only one command matches for as many characters as were typed. If the lookup is successful, another table, DISP, specifies a dispatch address and a number of legality bits for the command.

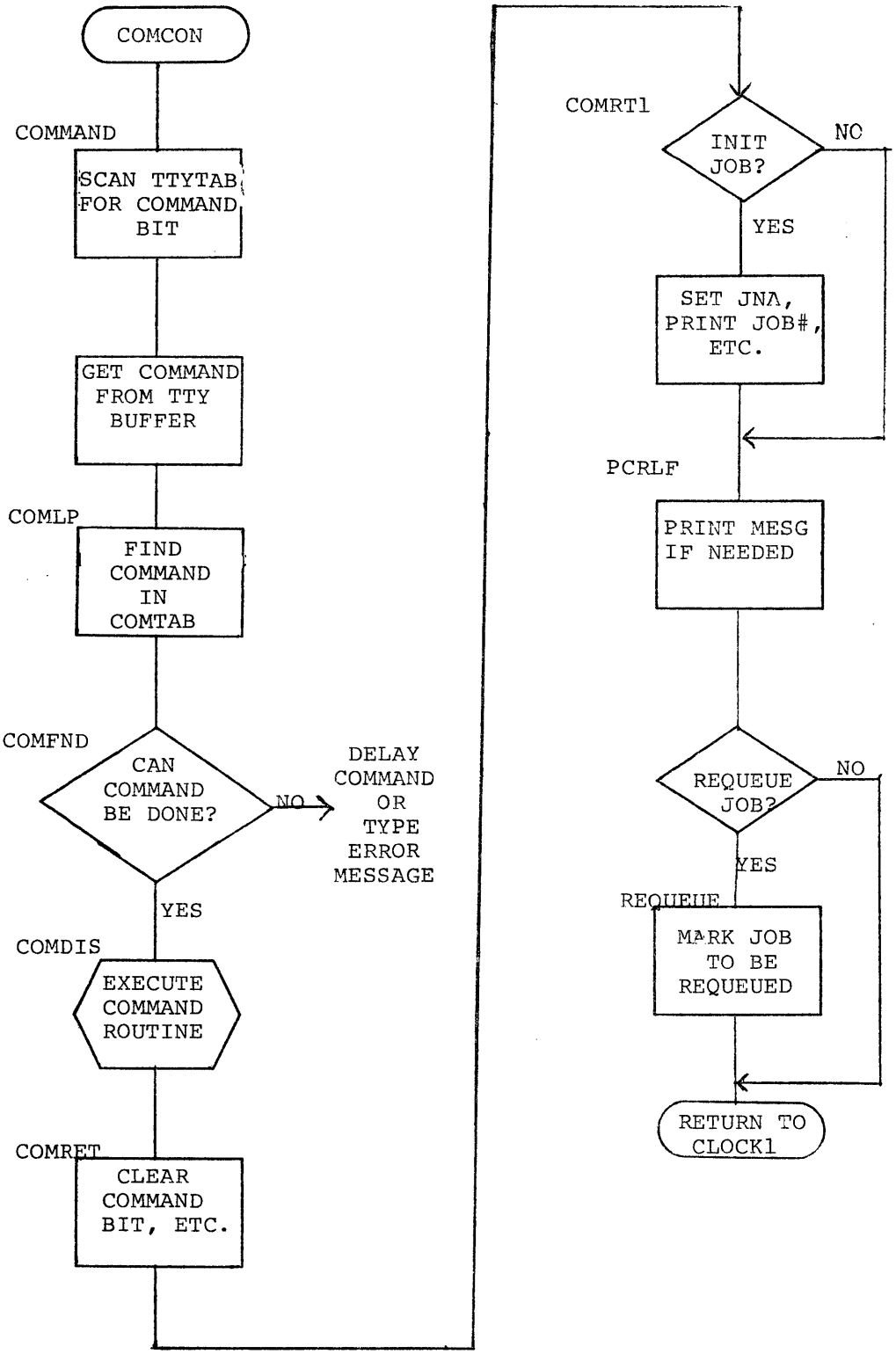
Conditions are checked according to the legality bits to determine if the command can be performed immediately. Accordingly, an error message may be typed or the command may be delayed. The command must be delayed if it is a legal command which merely cannot be performed at the present time. If a command is delayed, the monitor will try again to perform it, on a later cycle. If the command requires a job number and there is none for the user who typed the command, a job number is assigned at this time. If all conditions are met, control will be passed to the routine to handle this specific command.

The command routine must run to completion quickly and return control to the dispatch routine, because user programs are being delayed while the command is processed. Many commands will, therefore, simply set up a routine to run during the user's time. This will result in control being passed to a monitor routine when this job is made the current user job. The monitor routine may call in a CUSP to run as the user program, or may call in a user program

specified by the command.

When the command routine has finished its immediate function, it returns control to the dispatch routine. Here a number of functions may be performed, according to bits in the dispatch table entry. If this command resulted in initialization of the job, then the job number, system identification, and date are typed on the user's TTY. An error message is typed if needed, and a carriage return-line feed and period are typed if specified by the dispatch table. The TTY is left in user mode or monitor command mode, also according to bits in the dispatch table. If the job has been in the command wait queue and now needs to be requeued, it is marked as needing to be requeued. The actual requeuing will be done later, by the scheduler. Finally, control is returned to the monitor's outer loop, in CLOCK1.

COMMAND PROCESSOR FLOW



Monitor Handout 19 -- July 70

QUESTIONS ON COMMAND PROCESSING

1. When is the "Command Completed" bit set in TTYTAB?

2. Where (routine and line) is the "Command Completed" bit cleared?

3. For what reasons must a command be delayed?

4. For what reasons will a job be put into the Command Wait QUEUE?

5. When is a user assigned a job number by the monitor?

6. When is the job number, system identification, and date typed upon completion of a command?

7. What correlation do you see between the answers to questions 5 and 6?

8. Which commands result in some function being performed during the job's time?

9. What action does the monitor take in response to an invalid command (i.e., a command which is not in the table)?

10. After which commands does the monitor not type a period? Why?

7. UUO Processing

Readings

Program Logic Manual, Memo #5, 9 pages
Program Logic Manual, Memo #8, p 5-11
Memo #8 is included in Section 8.

Table Descriptions

UUOTAB
UCLTAB
UCLJMP
Buffer Ring
Buffer Ring Header
JDA - Job Device Assignment Table
Device Data Block

Flow Charts

Handout 45 - UUO Flow
Handout 17 - Input UUO
Handout 18 - Output UUO
Program Logic Manual, Memo #8, p 12-23
Handouts 17 and 18 are less detailed than the flowcharts in Memo #8.

Diagrams

Handout 43	IO UUO - Device Running
Handout 39	UUO Actions - INIT and INBUF
Handout 42	UUO Actions - INPUT
Handout 41	UUO Actions - OUTPUT
Handout 40	UUO Actions - CLOSE

Timing Relationships

Handout 32	IO Wait
Handout 33	IO UUO - Device Running
Handout 34	IO UUO - Device Not Running

Monitor Listings

ONCE lines 94-95; 140-163
How the UUO Trap locations are set up.
COMMON Routine UUOØ lines 2540-2578
UUOCON lines 1-433

Written Assignment

Questions on UUO Processing

PROGRAM LOGIC MANUAL

for

PDP-10 Time-Sharing Monitor

Memo #5

PDP-10 TIME-SHARING MONITORS

PROGRAMMED OPERATOR SERVICE (UUOCON)

I. DESCRIPTION

The function of UUOCON is to service in some manner those op codes which are trapped to absolute locations 40 and 41 by the processor hardware. These are op codes 000, 040, through 077, and (in user mode) 7xx (input/output, HALT (JRST 4,), and JEN (JRST 10,). In addition, the PDP-6 traps codes 001 through 037 as well.

The operations of UUOCON might, for the purpose of discussion, be divided into three sections.

1. Operator-independent preprocessing and dispatch;
2. Operator service (operator-dependent algorithms); and
3. Exit routines

Preprocessing includes saving of user accumulators if the machine was in user mode when the trap occurred (the Monitor may itself contain programmed operators), filtering out error codes, entering the user's UUO (User-Utilized Operations) handler if the machine is a PDP-6 and codes 001 through 037 are encountered, loading of accumulators with information to be used by the operator service routines, and dispatching to the proper service routine.

Operator service routines perform the algorithm designed for the particular UUO code, allowing the user to receive information about the system, to alter the operation of the system concerning his job, and to communicate with the input/output devices. A few specific examples are included in this chapter to demonstrate the information flow between the three sections of UUOCON and the user's job. Input/output UUO's are dealt with in the chapter on Input/Output Service.

The exit routines (normal or error) perform the setup necessary to return to the calling program or, in the case of errors, produce error messages and appropriately alter the status of the job. One important function of the normal exit routine is to check the status of the Scheduler before returning to the calling program. A software interlock between the Scheduler and UUOCON allows a UUO (which is, after all, one "instruction") to run to completion before the current job is stopped. The normal exit routine calls the Scheduler if the

interlock flag was set sometime during the UUO processing.

OPERATOR PREPROCESSING AND DISPATCH

SPECIAL REGISTERS

A rather important function of this section is to place information about this user's job (i.e., the job that issued the UUO) into certain accumulators and index registers before dispatching. Therefore, these registers and their contents are described briefly before going into the operations of this section.

PDP — A pushdown pointer to a 20-location list in the user's job data area. The first item placed in this list (JOBPDl) is the user's return, i.e., a copy of the PC word formed by the JSR in location 41.

PROG — Contains a copy of the contents of JOBADR: XWD highest relative address, relocation for this job. Used as an index register by the system to relocate references to the user's program area.

JDAT — Currently the same physical register as PROG but, strictly speaking, contains the protection and relocation for references to the user's job data (JOBDAT) area.

UUO — A copy of the programmed operator as trapped into location 40. The address PROG is set into the X field so that operator service can refer to (E) indirectly through UUO.

UCHN — A copy of the AC address field of the UUO. UCHN stands for User Channel, which it is in the case of input/output operators.

DEVDAT[1] — A copy of USRJDA (protected JOBJDA) for this software channel. This register contains Ø if this channel is unassigned. If the channel is in use, the left half of this word has status bits indicating what UUO's have been performed for the device so far; the right half contains the base address of the device data block (DDB).

IOS[1] — A copy of the DEVIOS status word for the device on this channel.

DSER[1] — A copy of the DEVSER word for the device on this channel. The left half of this word contains the address of the next DDB in a chain of all such blocks; the right half contains the base address of the dispatch table for this device's service routine.

[1] These registers are pertinent only to input/output programmed operators, but will be loaded, in any case, when an AC address (UCHN) happens to correspond to an assigned I/O channel.

FUNCTIONAL DESCRIPTION

The following is a narrative of the operator-independent preprocessing and dispatch section of UUOCON.

UUO1 — The user mode flag bit of the trapped PC word is used to detect whether the call is from the Monitor (as in a GET command) or from the user. If from the Monitor, certain AC's have been set up and a portion of the UUOCON coding can be skipped; control goes to UUOSY1. If the call is from the user and in the range 001 through 037 (PDP-6 only), then a software trap to the user's UUO handler is created, provided that the user has a nonzero address in his JOB41. If that location contains either Ø or an illegal address, an appropriate error message is typed on the user's Teletype and the job is stopped. If the call is from the user and is not in the 001 through 037 range, control goes to UUOSYS.

UUOSYS — The user's AC's are saved in the JOBAC part of his job data area and the contents of PROG, JDAT, and PDP are established.

UUOSY1 — This routine PUSH's the PC word (return address) as the first entry on the list and then tests the UUO for legality, now trying to exclude a 000 op code.

ILEGAL — If the routine is entered at this location, UUOERR is called, which types a message "ILLEGAL UUO . . ." and stops the job.

If the routine is entered with a skip, it sets the contents of UUO for indexing by PROG and then checks the op code for a value greater than 100 (illegal at this point). If the value is not illegal, accumulator UCHN is set up. If there is a device on this channel, DEVDAT, IOS, and DSER are set up. If no device has been assigned to this channel coincident with this UUO's AC address, the routine NOCHAN is entered. Otherwise, if this UUO is indeed an I/O operator of op code 72 or greater, then routine DISP1 is entered. ROUTINE DISPØ is entered directly for non-I/O UUO's or I/O UUO's between codes 55 and 71 if the channel is found to be assigned.

DISPØ, DISP2 — This coding obtains an address from a 2-address-per-word dispatch table, using the op code as an index. If this UUO was from user mode, the service routine is dispatched to by a PUSHJ which puts the address of the user exit routine on the list as it jumps. If it was from the Monitor, then the desired address is already on the list and is left undisturbed when dispatching to the service routine.

NOCHAN This routine calls DISPØ if the UUO was from the Monitor, or if it was from the user and is not an I/O operator. If the UUO is a CLOSE or RELEASE operator, the successful return exit is called. Otherwise, th routine IOIERR is entered to type the message "I/O TO UNASSIGNED CHANNEL. . ." and stop the job.

DISP1 This routine "fakes" a successful return to the user if the UUO was a "long dispatch" one and the device service routine does not have a long dispatch table (this is an important concept in making user programs "device independent"; e.g., it enables a LOOKUP to a physical paper tape reader to be "successful"). If the device service routine is capable of performing long UUO's, the dispatch routine DISPØ is called.

OPERATOR SERVICE

Before discussing a particular operator, let us first see how communication between the user's program and the operator service routine is made possible by setting up the AC's before dispatching. A most important point to note is that any Exec level software that refers to addresses in the user area must provide address checking equivalent to that performed by the hardware in user mode. A reference, especially one that stores information, must address a location equal to or greater than $(PROG)_{RH}$ and equal to or less than $(USRREL)_{RH}$. There are also some locations in the job data area which should be protected. Three address checking routines exist in Monitor and can be called from a UUO service routine.

UADCK1 This routine is called with a PUSHJ after loading AC1 with the address to be checked. It returns if this relative address is in the user's accumulator area or between JOBPFI (the top of the protected area of JOBDAT) and $(USRREL)_{RH}$.

UADRCK This routine is called in the same manner as UADCK1, but considers accumulator area references illegal. Both UADCK1 and this routine stop the job and print the message "ADDRESS CHECK . . ." message if a failure occurs.

IADRCK This routine, more forgiving than either of the above, is called (PUSHJ) with the address to be checked previously placed in TAC and PROG already set up. This routine considers an address acceptable if it lies between JOBPFI and the relative address in the left half of PROG. Failure is indicated by a no-skip return to the calling program, success by a skip return.

After careful address checking, access to user locations may be made in any of the following ways.

1. Fetch the contents of the effective address of the UUO.
   MOVE TAC, @UUO, where TAC is an accumulator available for use.

   NOTE

   Two things make "@UUO" work: (1) the hardware has computed the relative effective address at the time of the UUO trap, and (2) the UUO preprocessor routine has placed PROG in the index address field of AC UUO.

2. Store a result in the effective address location.
   MOVEM TAC, @UUO

3. Get an argument from the AC addressed by the UUO (recall that UCHN contains this AC address and that AC's are in the JOBAC area.

```
HRLI    UCHN, JDAT      ; relocate AC reference
MOVE    TAC, @UCHN      ; get contents
```

4. A routine STOTAC exists which stores the contents of accumulator TAC indirectly into the location addressed by UUO after checking the address (UADCK1 routine) and exits with a POPJ. To end a service routine by returning a result to the effective address of the UUO and immediately return to the user, the following instructions are executed.

```
MOVE TAC, result
JRST STOTAC
```

   If the call to STOTAC is made from the same level (with reference to the pushdown list) to which the preprocessor routine dispatched (via a PUSHJ), STOTAC's POPJ exit will return to the exit routine that followed the dispatch coding.

In returning to the user, one may wish to skip one or more arguments that followed the UUO, or to give a skip or no-skip return to signify success or failure of the operation. The UUOCON exit routine is designed to pass on to the user either a skip or no-skip return. If, when at the level equal to that following the dispatch, a POPJ PDP is used to exit, the user will receive a no-skip return. If the sequence

```
AOS (PDP)
POPJ PDP,
```

is used, a skip return occurs. This could be used to bypass one argument following the UUO (a system routine, CPOPJ1 performs this action if called by a JRST CPOPJ1). If it is necessary to bump up the user's return by more than one, the routine must take care of adding the correct quantity to the correct entry on the pushdown list (recall that, if the original UUO was issued by the Monitor, the preprocessor dispatch was <u>not</u> a PUSHJ). If, for example, two arguments are to be skipped in return to a user mode call, this sequence

could be used.

```
AOS  -1(PDP)
JRST CPOPJ1
```

To give the same return to a call from the Monitor,

```
AOS  (PDP)
JRST CPOPJ1
```

Example

Presently, all operators that do not deal with some phase of input/output appear as subfunctions of the CALL programmed operator. To keep this example reasonably simple, we will choose one of these:

```
CALL  AC, [SIXBIT/RUNTIM/]
```

The referenced AC is loaded with a job number before the CALL, and the CALL returns the total running time (in "jiffies") of that job in the same AC.

The preprocessor routine of UUOCON sets up the standard accumulators and, using the UUO op code (CALL = 040), dispatches to UCALL. UCALL picks up the contents of the UUO effective address, the literal value RUNTIM. This argument is used to effect another dispatch to the routine JOBTIM, which gets the appropriate run time and stores it in the user accumulator. Before this second dispatch, the UCALL routine places the contents of the user's accumulator into TAC and changes the right half of UUO to contain the address of this accumulator. The accumulator ITEM is loaded with the job number of the currently running job.

When entered, the JOBTIM routine checks the contents of TAC for a valid job number and then uses it as an index to fetch from the TTIME table (where running times for all jobs are kept) the desired time and place it into TAC. A JRST STOTAC causes this result to be stored in the user's accumulator, now addressed by UUO, and return to the UUOCON exit routine.

EXIT ROUTINES

ERROR EXITS

Error exits, which do not allow a return to the user, occur when a UUO op code is illegal or an address supplied by the user is illegal. A nonimplemented UUO in the range 40 through 77, or a UUO of Ø will stop the job with the error bit on (cannot continue) and print "ILLEGAL UUO at USER loc". An illegal op code

(e.g., a DATAI in user mode) causes the job to be stopped with the error bit set and the message "ILL. INST. AT ...." to be printed. The HALT instruction stops the job, types "HALT AT USER loc.", but does not set the error bit. Thus, the CONT(INUE) command does function after a HALT.

When an illegal address is detected by a non-I/O UUO, the UUOERR routine is called to print the message noted above ("ILLEGAL UUO AT USER loc") and puts the job into an error stop. When a UUO is associated with a particular device, ADRERR may be called. ADRERR prints "ADDRESS CHECK FOR DEVICE dev: EXEC CALLED FROM loc", and results in an error stop condition.

NORMAL EXITS

If the original UUO was issued by the Monitor, the preprocessor dispatch was by a JRST rather than by a PUSHJ. The service routine's last POPJ would bypass the user exit routine and go directly back to the Monitor coding following the call.

If the UUO was from the user, the service routine's terminating POPJ returns to location USRXT1-1 (no-skip return) or a JRST CPOPJ1 returns to USRXT1, which passes a skip return to the user by adding 1 to the address on the pushdown list.

USRXIT This routine checks to see if the user has typed a CTRLC (↑C), or if the clock has ticked (software interlock), or if the system wants to stop this job (to swap it, for instance). If none of these conditions exists, the user's accumulators are restored and control is returned to his program. Otherwise, the Scheduler is called (SCHED) to take appropriate action. If the user's job continues in the future, control will come back here to restore the user's accumulators and continue the job.

II. ADDING A PROGRAMMED OPERATOR

There are two ways to add a new UUO function to the Monitor. One is to use a previously unused op code (42 through 46 are open at the time of this writing - May, 1968). The other is to add a subfunction to the CALL operator. Before adding anything to any section of the Monitor, it is, of course, desirable to understand what is already there. Assuming that one already has this understanding and has written a tightly coded new routine that obeys the rules of address protection and uses as much existing coding as possible, we can

investigate the process of getting this routine included in a running Monitor.

ADDING A NEW OPERATOR

1. Edit the new coding into the source file for UUOCON. If it is desired to make this routine a conditional feature, it may be enclosed in conditional assembly brackets preceded by a symbol like the feature test switches presently in use.

2. Edit into the UUO dispatch table, UUOTAB, the address of this routine in the proper half of the XWD found there. For instance, if you are adding a routine, UDUMP, as op code 43, you would replace XWD UUO42, UUO43, with XWD UUO42, UDUMP. Conditional assembly could be used to set up the dispatch table entry if conditional assembly was used with the routine itself For example,

Routine Coding	Dispatch Table Entry
IFN FTDMPU, <UDUMP:	IFN FTDMPU,<
(coding)	XWD UUO42, UDUMP
	>
	IFE FTDMPU,<
	XWD UUO42, UUO43
>	>

In this example, the routine will be assembled and the address of UDUMP is added to the dispatch table if the feature switch FTDMPU is nonzero.

3. In preparation for assembling the new UUOCON, edit the correct feature test switch settings into the S (system parameter) source file, including any new ones you have established.
4. Assemble, naming as input first the S file, then the new UUOCON file.
5. Use FUDGE2 to Replace the old version of UUOCON with the new one in the library file to be used in building your system.
6. Build a new monitor, using this new library file, according to the procedures in MONITR.OPR.

ADDING A NEW CALL SUBFUNCTION

This method is an attractive alternative to adding an entire new operator when some job-number-dependent function is to be performed or when arguments to be passed are few. Recall that, before the CALL dispatches to a subfunction, it places the job number in accumulator ITEM, the contents of the UUO AC into TAC, and the address of the UUO AC into UUO, which has previously been set for relocation. Thus, arguments or argument addresses can easily be passed via this accumulator. The CALL operator dispatches to a subfunction by searching a table of SIXBIT names (UCLTAB) for a match with the contents of the UUO effective address and then selecting a corresponding jump address from a half word in a second table (UCLJMP). Alternately, the user may use the CALLI (CALL Immediate) operator and directly supply the index to the jump table. Because of the latter, any additions to the CALL dispatch tables must be appended to those entries already in existence.

Example

The PDP-10 hardware will display a word in the console data lights when the instruction

```
DATAO  PI,[display information]
```

is executed. Let us add a new CALL to allow any user program logged in under project number 2 to display information by loading the data into AC and issuing the command

```
CALL  AC, [SIXBIT/CONLIT/]
```

Let us further specify that, if the user is not logged in with the proper project number (2), the call is to be treated as a no-operation. Finally, let us write the code in such a way that, in a Monitor with no login feature (feature switch FTLOGIN = Ø), this operator always works.

```
LIGHTS:
IFN FTLOGIN,<
    HLRZ    TAC1, PRJPRG (ITEM)      ;get project number
    CAIN    TAC1, 2                  ;equal to 2?
    >
    DATAO   PI, TAC                  ;display contents of AC
    POPJ    PDP,
```

After editing this coding into an appropriate area of UUOCON, the dispatch tables must be updated. This is done by adding one entry to the list following the NAMES macro which is called to build the two tables. An entry has the general form

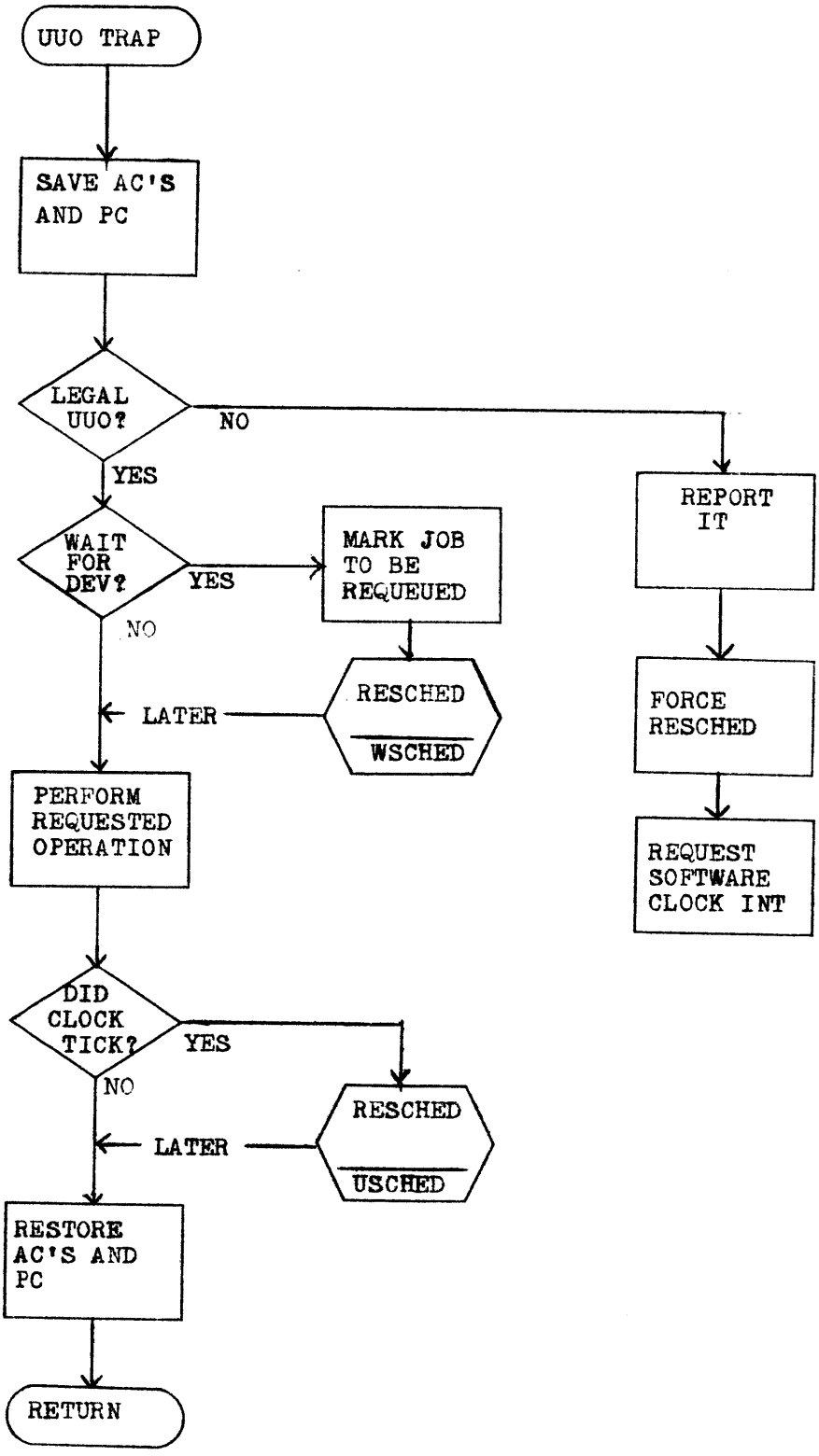
```
X  function-name, routine address;    comment
```

To add our new display function, insert after the last name and before the LIST statement

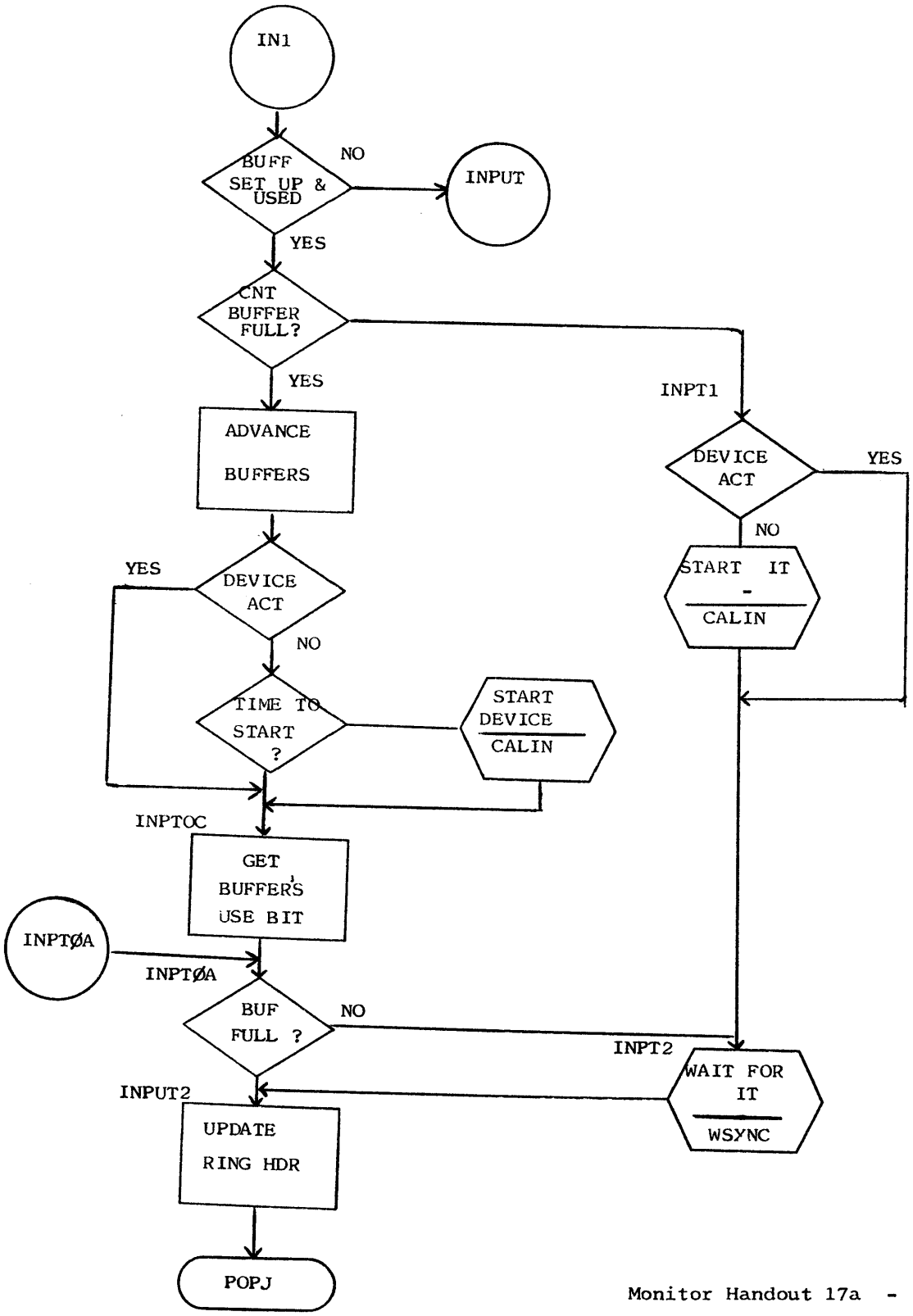
```
X CONLIT, LIGHTS; DISPLAY (AC) IN DATA LIGHTS
```

To create a working Monitor, follow steps 3 through 6 as outlined under "Adding a New Operator."

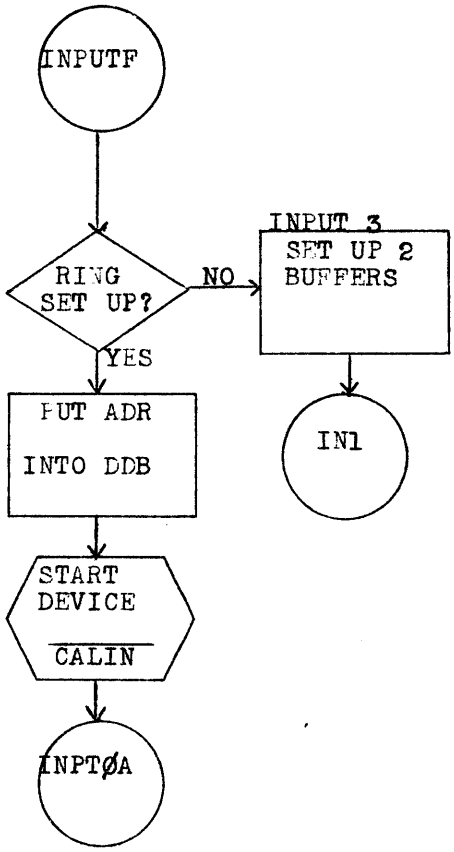
UUO FLOW



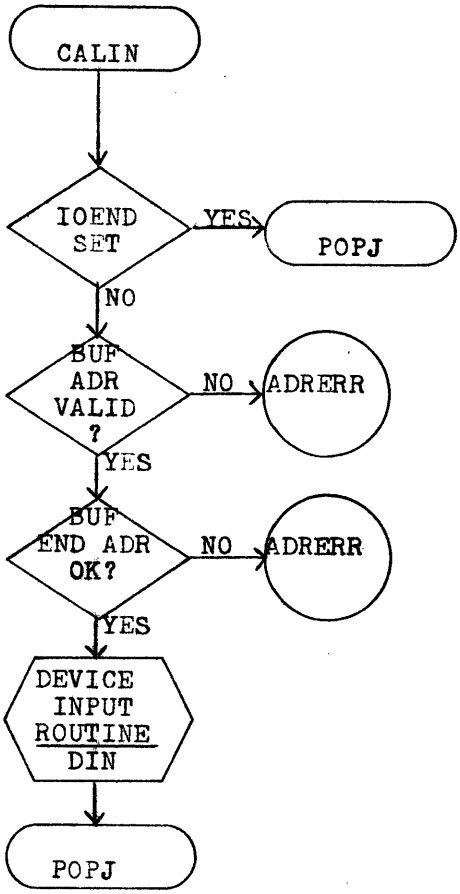
Monitor Handout 45 -- July 70


IN1
BUFF SET UP & USED
NO
INPUT
YES
CNT BUFFER FULL?
YES
ADVANCE BUFFERS
INPT1
DEVICE ACT
YES
NO
START IT
CALIN
YES
DEVICE ACT
NO
TIME TO START ?
START DEVICE
CALIN
INPTOC
GET BUFFER'S USE BIT
INPTØA
INPTØA
BUF FULL ?
NO
INPT2
WAIT FOR IT
WSYNC
INPUT2
UPDATE RING HDR
POPJ

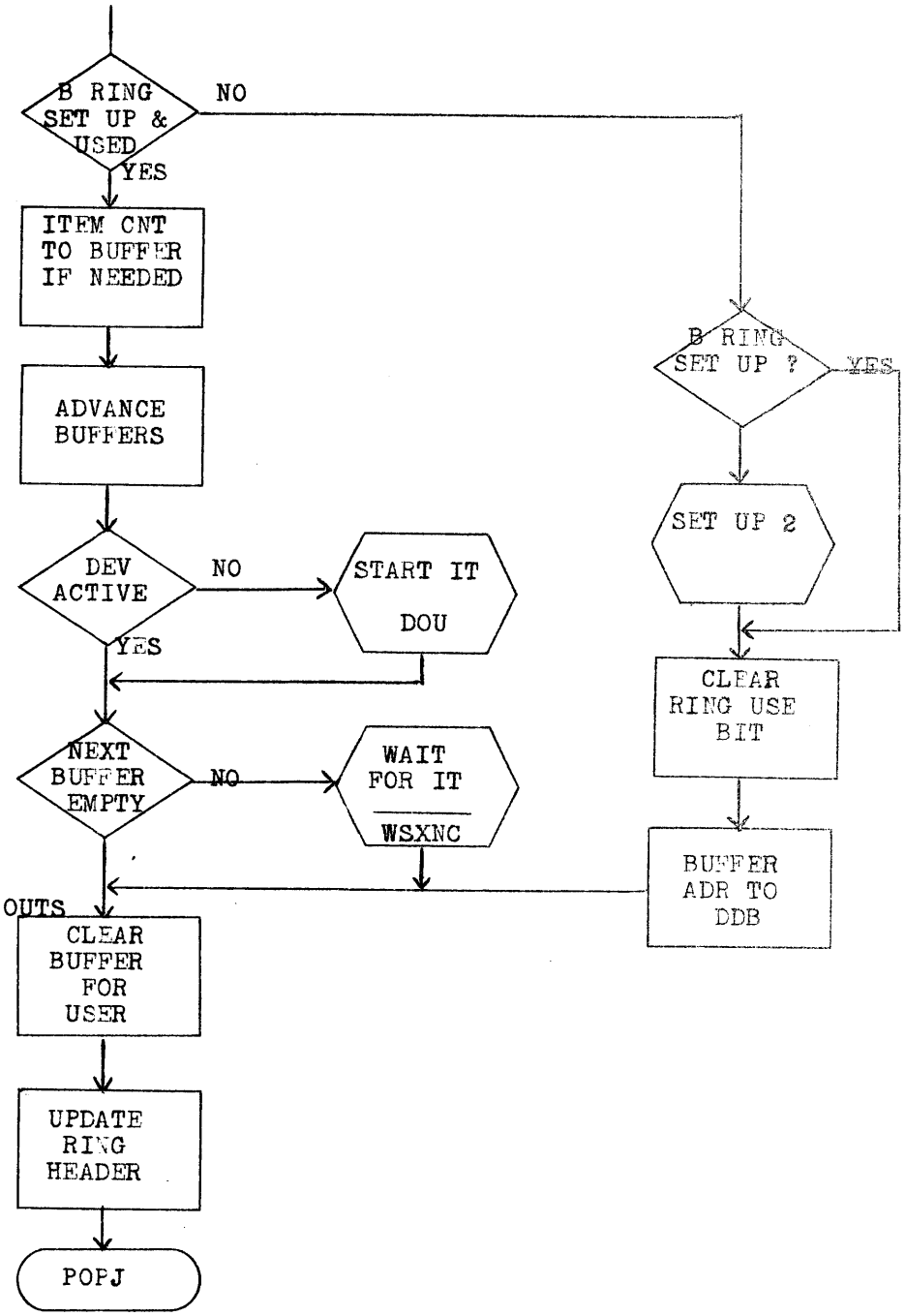
INPUT UUO



CALIN

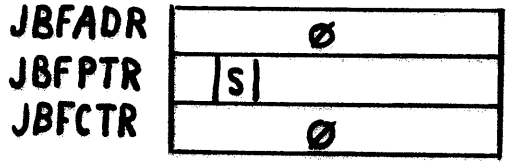


OUTPUT UUO

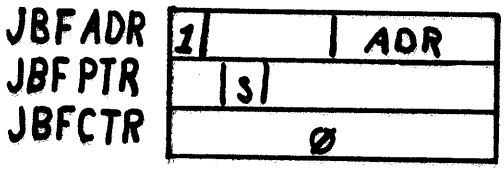

B RING SET UP & USED
NO
YES
ITEM CNT TO BUFFER IF NEEDED
ADVANCE BUFFERS
DEV ACTIVE
NO
START IT
DOU
YES
NEXT BUFFER EMPTY
NO
WAIT FOR IT
WSXNC
OUTS
CLEAR BUFFER FOR USER
UPDATE RING HEADER
POPJ
B RING SET UP ?
YES
SET UP 2
CLEAR RING USE BIT
BUFFER ADR TO DDB

RING HEADER CHANGES

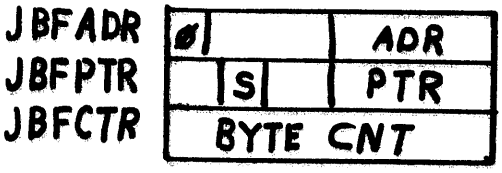
INIT



OUTBUF

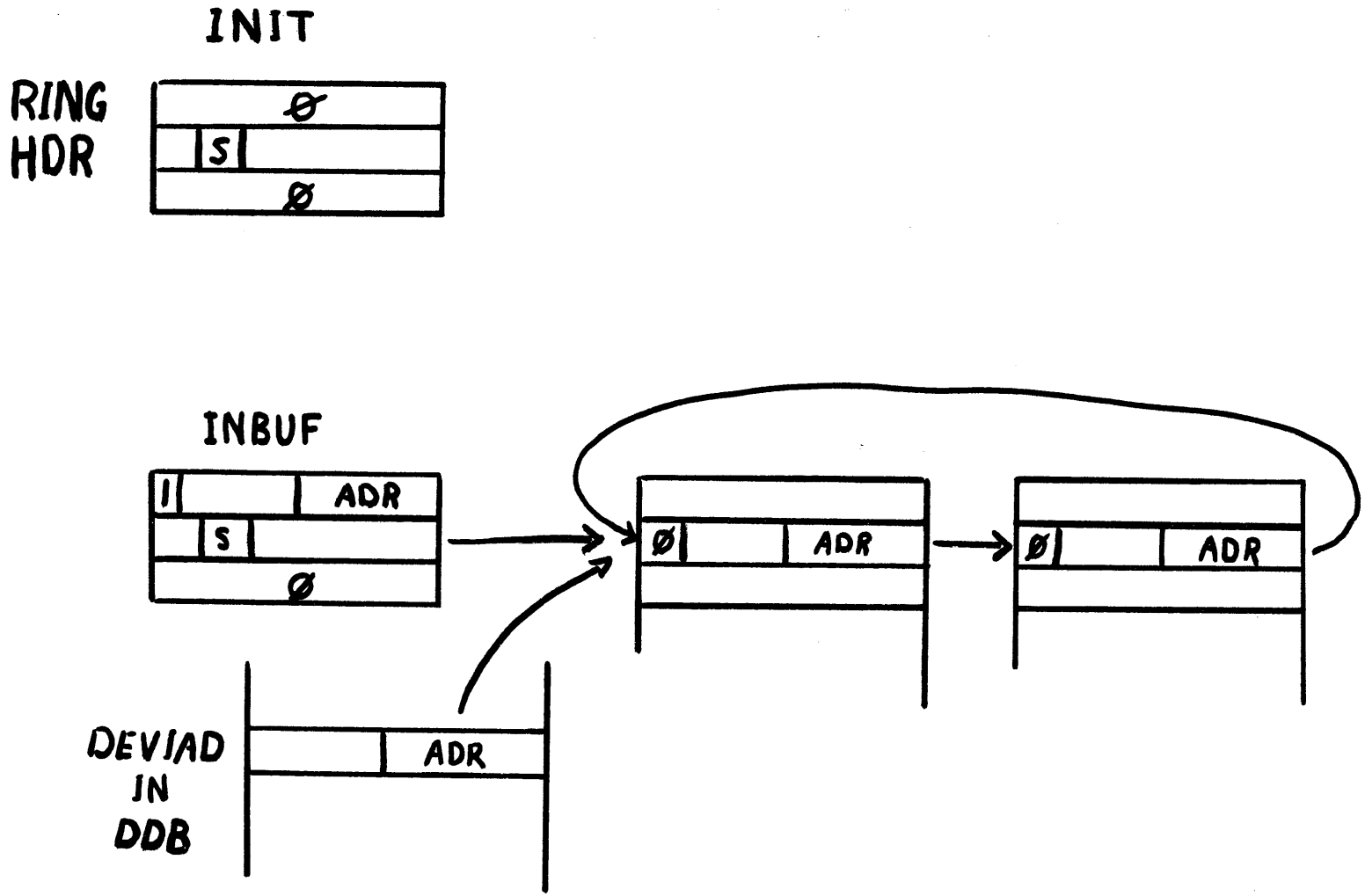


OUTPUT

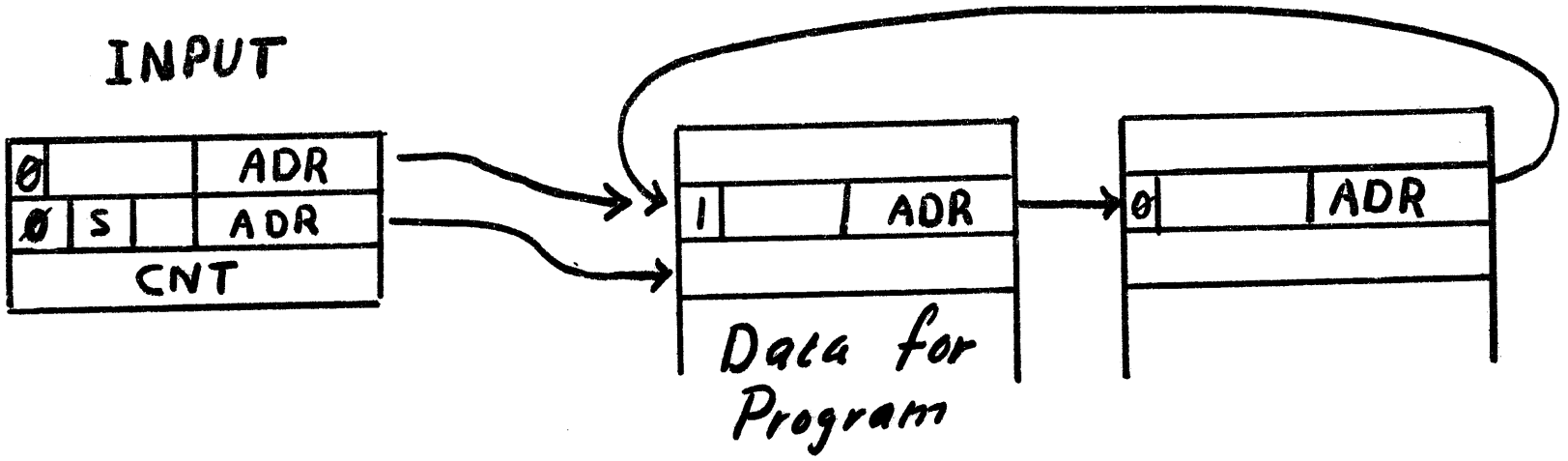


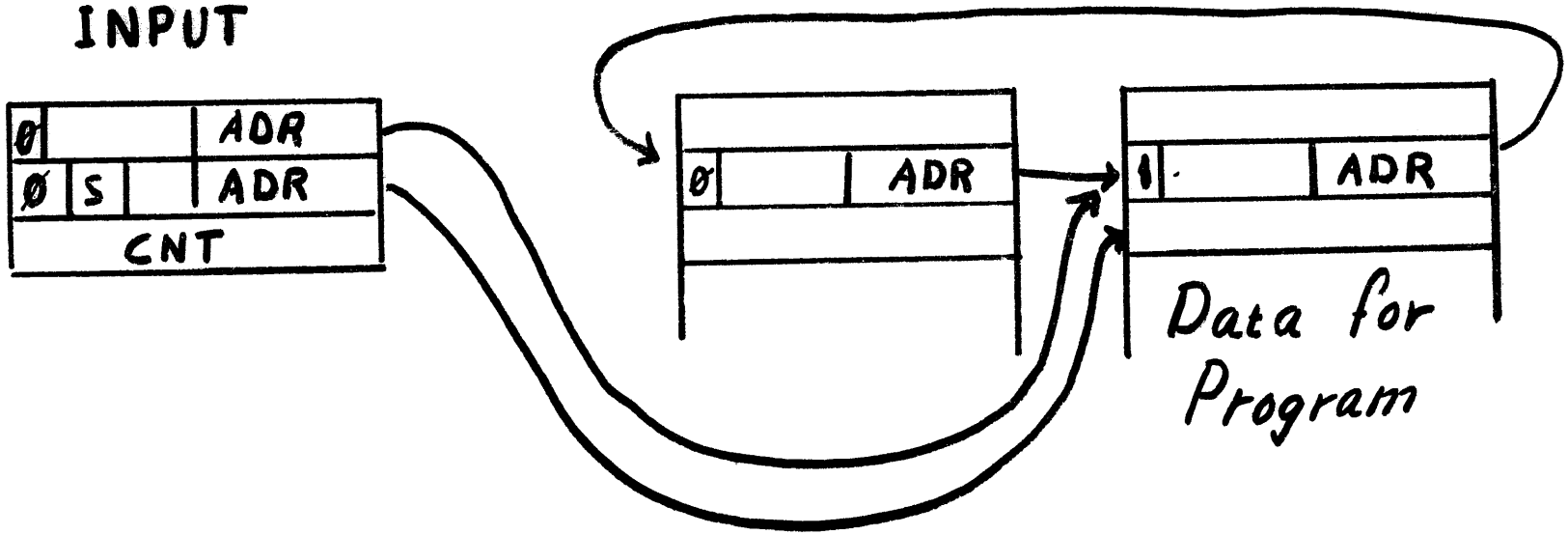
Monitor Handout 43 -- July 70

UUO ACTIONS

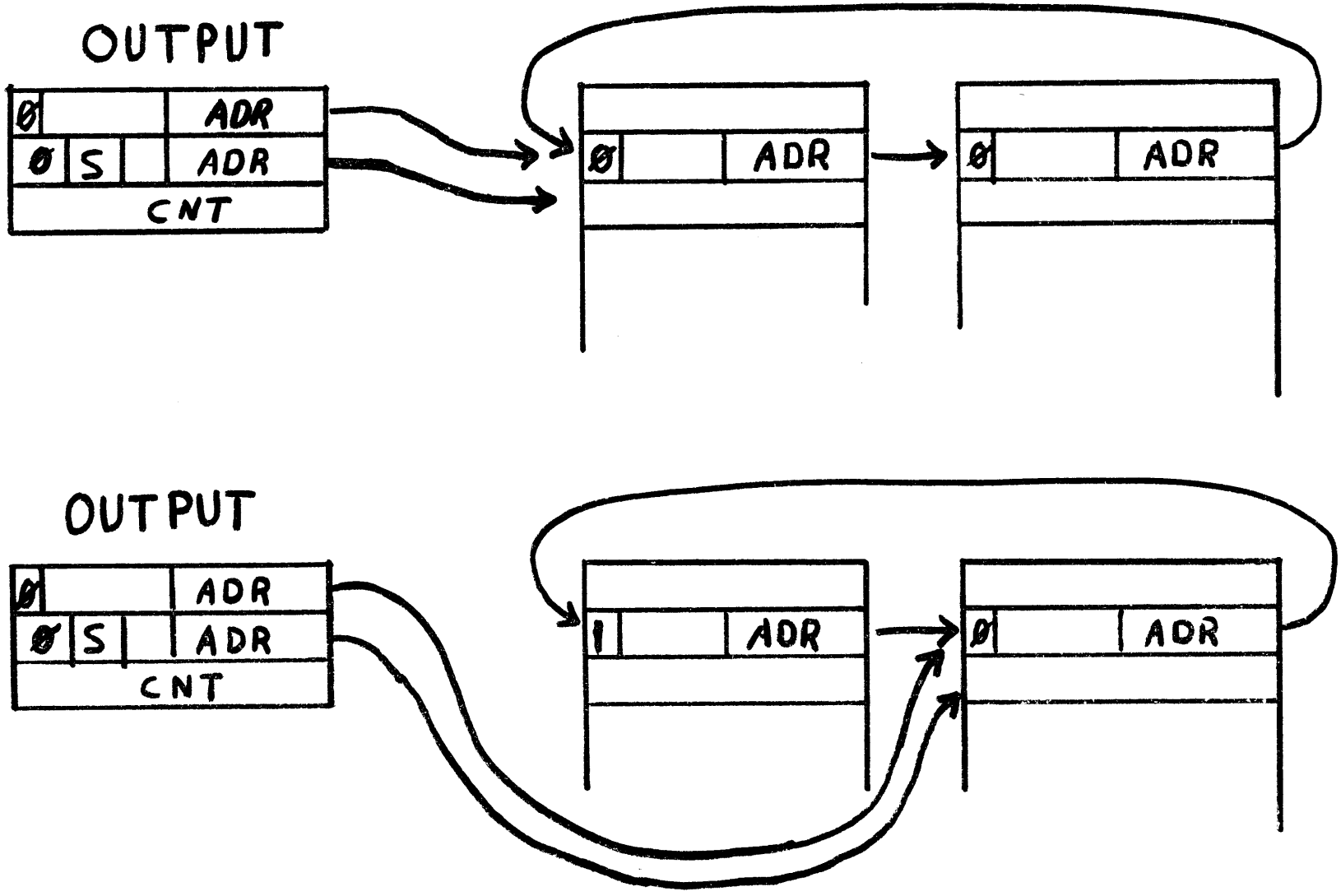


UUO ACTIONS

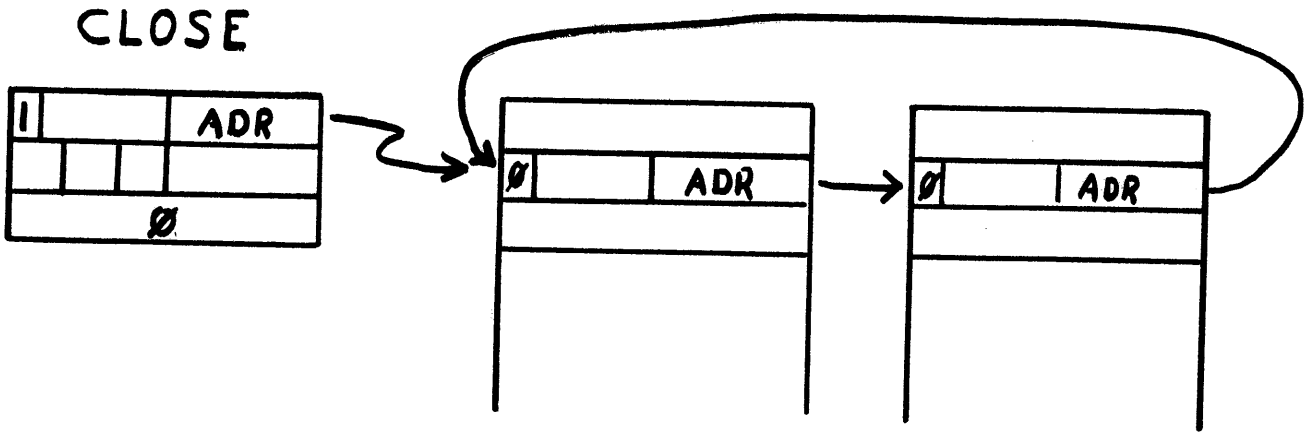

INPUT
0
ADR
0
S
ADR
CNT
1
ADR
Data for Program
0
ADR


INPUT
0
ADR
0
S
ADR
CNT
0
ADR
1
ADR
Data for Program

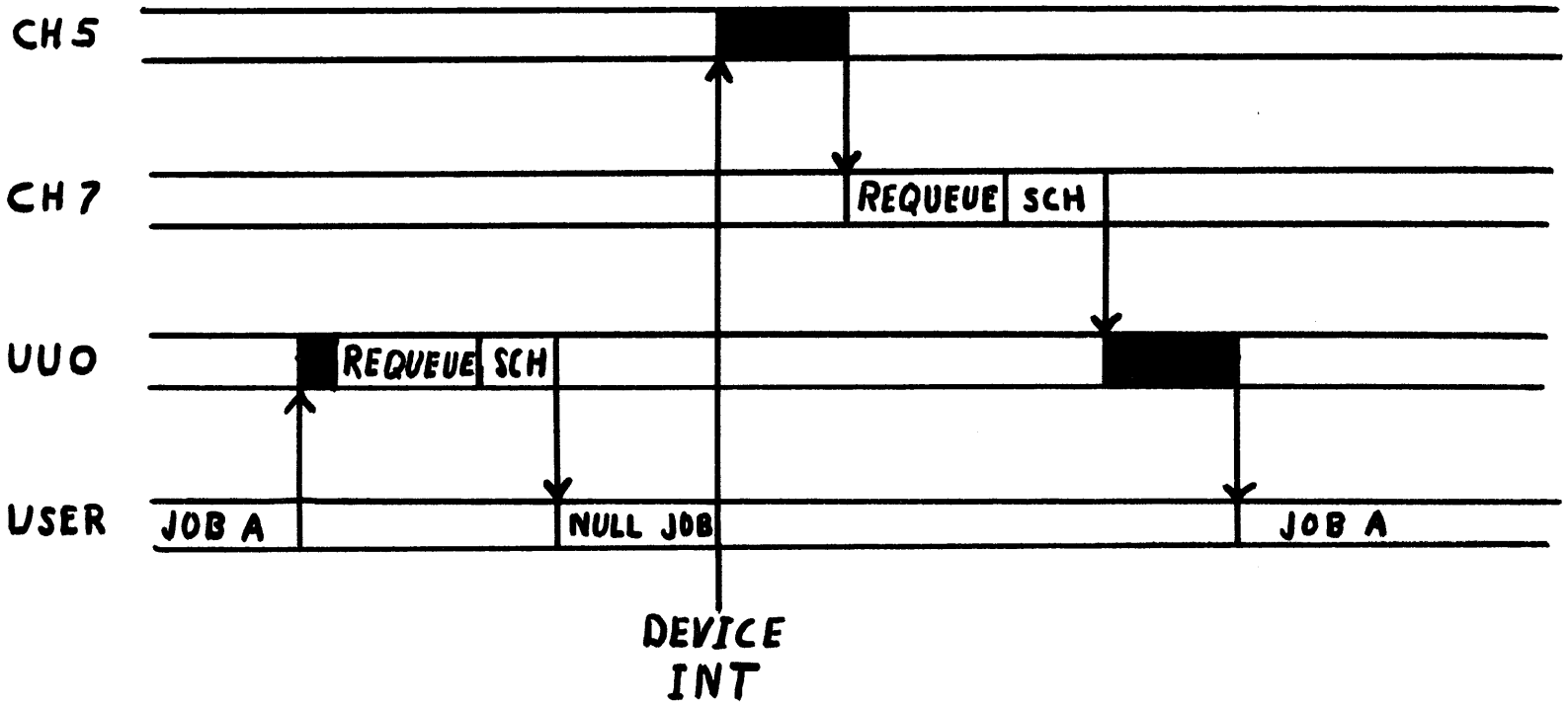
UUO ACTIONS


OUTPUT
0
ADR
0
S
ADR
CNT
0
ADR
0
ADR
OUTPUT
0
ADR
0
S
ADR
CNT
1
ADR
0
ADR

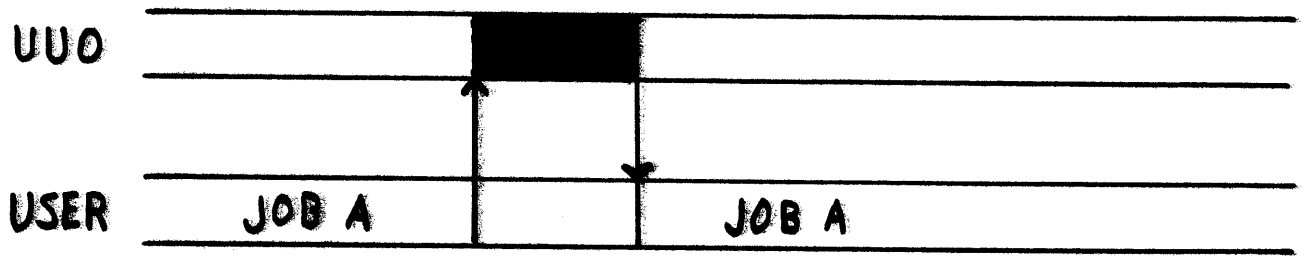
UUO ACTIONS



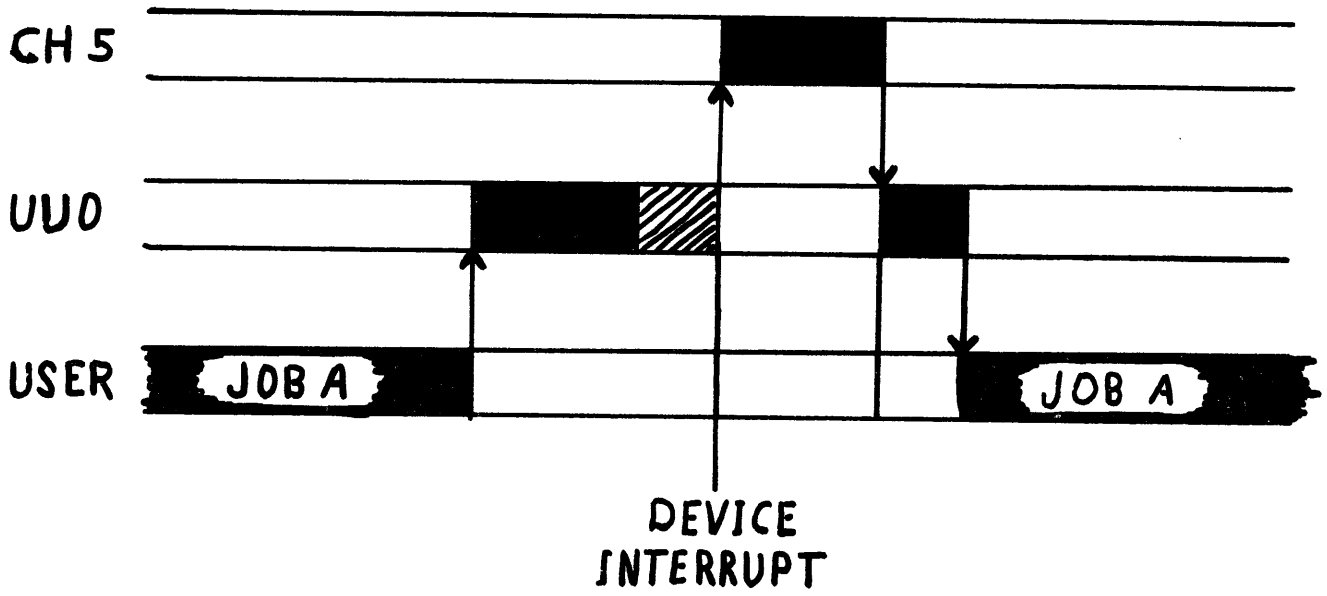
I/O WAIT


CH 5
CH 7
REQUEUE
SCH
UUO
REQUEUE
SCH
USER
JOB A
NULL JOB
JOB A
DEVICE
INT

I/O UUO -- DEVICE RUNNING



I/O UUO -- DEVICE NOT RUNNING



Questions on UUO Processing

1. When a UUO is executed, what will be the contents of location 40?

2. What is the range of op codes which are legal monitor UUO's?

3. How is the address of the routine for a specific UUO determined?

4. Where are the user's AC's saved while a UUO is being processed?

5. Suppose a UUO can not run to completion (IO problems, etc.). When context switching occurs, USRPC will contain the address for restarting the job - somehwere in UUOCON. How does UUOCON then know the return address in the user's program, considering that UUOCON may have been called by any number of other jobs in the meantime?

6. What action does UUOCON take if a UUO requiring a long dispatch table is executed, and the specified device service routine has a short dispatch table?

7. How does UUOCON respond to a UUO without a currently defined function - i.e., 000?

8. How does UUOCON respond to a CALL UUO with an undefined function - i.e., CALLI, 400?

9. Which AC's are loaded with what values by UUOCON, before it dispatches to the routine for a specific function?

10. Write the routine and specify all necessary monitor modifications to implement the following new CALL.

    CALL AC, [SIXBIT/CHAN/]

    The CHAN routine will put into the user's specified AC the number of the first unused software channel for his job.

8. Device Service Routines

Readings

Handout "Device Service Routines"
Program Logic Manual, Memo #8, p 32-33

Table Descriptions

INTTAB

Flow Charts

Handout 9 Device Interrupt Routine
Program Logic Manual, Memo #8, p 34-36

Other References

Handout 8 - Interrupt Routine Chain
Handout 28 - Channel Save Routine

Monitor Listings

ONCE lines 94-163
PTPSER lines 1-266

Written Assignment

Questions on Device Service Routines

All device dependent code required for I/O operations on a specific device is contained in the device service routine for that device. Each device service routine interfaces with the rest of the monitor in such a way that other routines are independent of the particular device being used for an I/O operation. Each routine may be optionally included in a monitor, or left out, according to whether or not the device which it services is included in the system.

Each service routine consists of two sections -- UUO level routines, and an interrupt routine. Control passes to UUO level routines directly from the UUO processor. Therefore they effectively act as subroutines callable by the user program. Control passes to the interrupt routine as a result of a hardware interrupt by the device which it services. All actual data transfers are performed by the interrupt routine. But before a device will cause any interrupts it must be conditioned, or "started" by a UUO level routine.

The linkage between the device independent UUO processor and the UUO level routines of a device service routine is the Device Dispatch Table. This is a table of JRST instructions to all the UUO level routines. The format of the dispatch table is identical for all service routines. Therefore, the UUO processor needs only the base address of the dispatch table to pass control to the routine for any UUO level function, regardless of the device. The UUO level routine may quite possibly call device independent subroutines back in the UUO processor during the course of performing its function. When finished, the UUO level routine returns control to the UUO processor, and from there control is returned to the user program.

Interrupt routines are written in such a way that they are independent of the specific priority interrupt channel on which they will operate. This allows PI Channel assignments to be changed without reassembling the service routines. All references to the PI Channel are made symbolically, and resolved when the monitor is loaded. The table INTTAB tells which channel each routine is assigned to. During system initialization, all routines on the same channel are linked together, according to the information in INTTAB. When an interrupt occurs, the PC is saved with a JSR, and then control is passed from one interrupt routine to the next, along the chain of routines assigned to that channel. Each routine checks if its device has an interrupt pending. If not, it passes control to the next routine; if so, it proceeds to process the interrupt. If control reaches the end of the chain, because no device is found to have an interrupt

pending, the interrupt is simply dismissed with a JEN which restores control to the interrupted program.

Most interrupt routines need a routine to save a number of accumulators and set up a pushdown list. Since this is a frequently needed function, one such routine for each channel needing it is set up in COMMON. One routine per channel is sufficient, because no interrupt routine will get control until any previous interrupts on its channel have been dismissed. Control is passed to the channel save routine with a JSR. (Note that JSP and PUSHJ cannot be used because we know nothing about the accumulators when the interrupt occurs.) A minimum of ten AC's is saved in an area within the routine, AC PDP is set up to point to another area in the routine reserved for a pushdown list, and control is returned to the interrupt routine. The first address on the pushdown list is preset to point to a routine which restores the AC's just saved. If the interrupt routine exits with a POPJ, control will return to the Restore Routine. The Restore Routine returns the original values to the AC's and dismisses the interrupt, returning control to the interrupted program.

PROGRAM LOGIC MANUAL

FOR

PDP-1Ø TIME-SHARING MONITOR

MEMO #8

INPUT/OUTPUT PROGRAMMED OPERATORS AND DEVICE SERVICE ROUTINES

I. SOFTWARE LINKS BETWEEN USER AND DEVICE

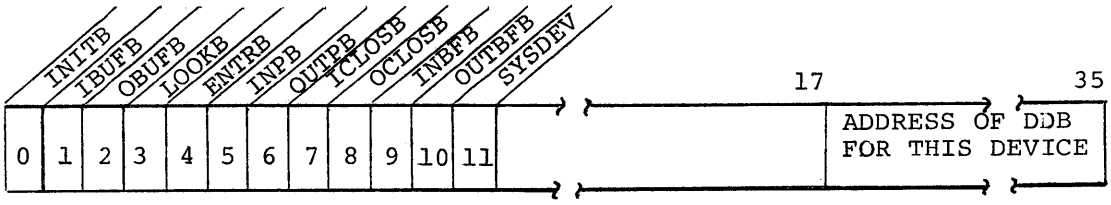
User input/output is made possible by the programmed operators and several tables existing in Monitor and the user's job data area. When desired, software linkage is made between a user program and a device (file) via these tables. For each physical device (or each active file on disk) there is a device data block in the Monitor describing the characteristics of the device: name, legal data modes, standard buffer size, and location of the service routine dispatch table. For a complete description of these tables, see Section III. When a device is assigned to the user and is being used by him, certain locations in the device data block (DDB) will contain certain information concerning the current activity: the job number using the device, status, data mode in which the file is to be read or written, and location of the user's buffers. The user, or some part of the Monitor, may look in the DDB to find out something about this device. The device service routine may obtain information about the user of this device by taking the job number from the DDB and referring to one of these Monitor tables indexed by job number (job status JBTSTS, job project-programmer PRJPRG, core assignment JBTADR, etc.).

The user's link to the DDB, and thus to the device, is one word in a 16-word table, JOBJDA, in his job data area. A location in this table is accessed by an index called the "software channel", supplied by the user. Figure 1 depicts one such location containing the address of the DDB and bits indicating the UUO operations done so far for this device. The user never directly accesses a JOBJDA location, but the Monitor does at UUO level using the software channel number specified in the AC field of the I/O operator. For protection, these locations are copied into the Monitor (starting at Monitor location USRJDA) when a job is running and Monitor works with these copies, restoring them to JOBJDA when the job is not running. A seventeenth location, JOBHCU (USRHCU), contains the channel number of the highest channel in use. These 17 locations, and others, are in an area of JOBDAT protected from input data transfers. The symbol JOBPFI ("protect-from-input") is a relative location in the job data area below which data must not be transferred, and user buffer addresses are checked against this value by those I/O UUO's concerned.

The last table to consider is the jump table or dispatch table in the device

service routine. This table (see Figure 11) links the device dependent coding in the routine with the device independent portion of UUOCON. The address of this table is in the right half of the DEVSER word in the DDB. UUOCON dispatches to this address plus or minus an appropriate index so that the service routine may perform whatever is necessary to service this UUO (start or stop the device, initialize hardware or software registers, etc.). Much of the work of UUO service is done by the device independent UUOCON routines and, even after dispatch to the device dependent routine, portions of the Monitor (IOCSS, in particular) are called as subroutines. The ultimate effect is that the user's program deals with all devices (or files) in a similar manner and the device service routine has only to interface some specific hardware device with the general coding of UUOCON. This latter topic, along with interrupt level operations, is dealt with in Section III, "Device Service Routines".

The next section describes the operations within UUOCON as it handles the communication between user and device.



UUO PROGRESS BITS

INITB	INIT or OPEN has been performed
IBUF	An input ring header was specified (by INIT)
OBUFB	An output ring header was specified (by INIT)
LOOKB	A LOOKUP has been performed
ENTRB	An ENTER has been performed
INPB	At least one INPUT has been performed
OUTPB	At least one OUTPUT has been performed
ICLOSB	A CLOSE input has been performed
OCLOSB	A CLOSE output has been performed
INBFB	An input buffer ring has been set up
OUTBFB	An output buffer ring has been set up
SYSDEV	This is the system tape device

NOTE: This word is completely cleared by RESET or RELEASE UUO's

Figure 1. JOBJDA or USRJDA Word Contents

II. I/O OPERATORS

REVIEW OF USER I/O

This section assumes previous familiarity with user I/O programming as described in the PDP-10 Reference Handbook.

Two methods are used to effect data transfers: unbuffered and buffered. In unbuffered modes, the user supplies to the device the address of a command list in his program area. This list consists essentially of block pointers to relative locations in the user area to or from which data is to be transferred. Upon initiating such a transfer, the user's job is scheduled into an I/O Wait where it remains until the device signals (to the Scheduler) the completion of the entire transfer. The device, at interrupt level, follows the command list in making the transfer until a termination word (null) is found and then notifies the Scheduler.

Buffered data transfers are made using a ring of buffers set up in the user area. A ring may contain one buffer or as many as will fit in the job area. A 3-word ring header in the user's program contains a byte pointer and item counter to be used by that program in accessing the "current" buffer (the one the user's program is working on). The device data block of the device involved in this data transfer contains like information concerning that buffer which is current to the interrupt level data transfers (see Figure 2). Monitor routines called by UUO's (INPUT or OUTPUT) work to supply a new buffer to the user, setting up the ring header appropriately. Monitor routines called at interrupt level likewise supply a new buffer for the device to work on, updating the pointer and item count in the DDB. To prevent the user and the device from using the same buffer at the same time, each buffer contains a use bit in the second word of the buffer header that is checked and altered by the Monitor's buffer handling routines. At UUO or interrupt level, a 1 means that the buffer is full and a 0 means that the buffer is not full. If the user "overtakes" the device and requires as his next buffer the one currently being used by that device, the user's job is scheduled into an I/O Wait. Upon completion of its use of that buffer, the device calls the Scheduler to reactivate the job. If the device "overtakes" the user, the device is stopped (always at the end of a buffer) and is restarted when the user finished with the buffer. (Input devices are not actually restarted until all but one of the buffers in the ring have been emptied by the user.)

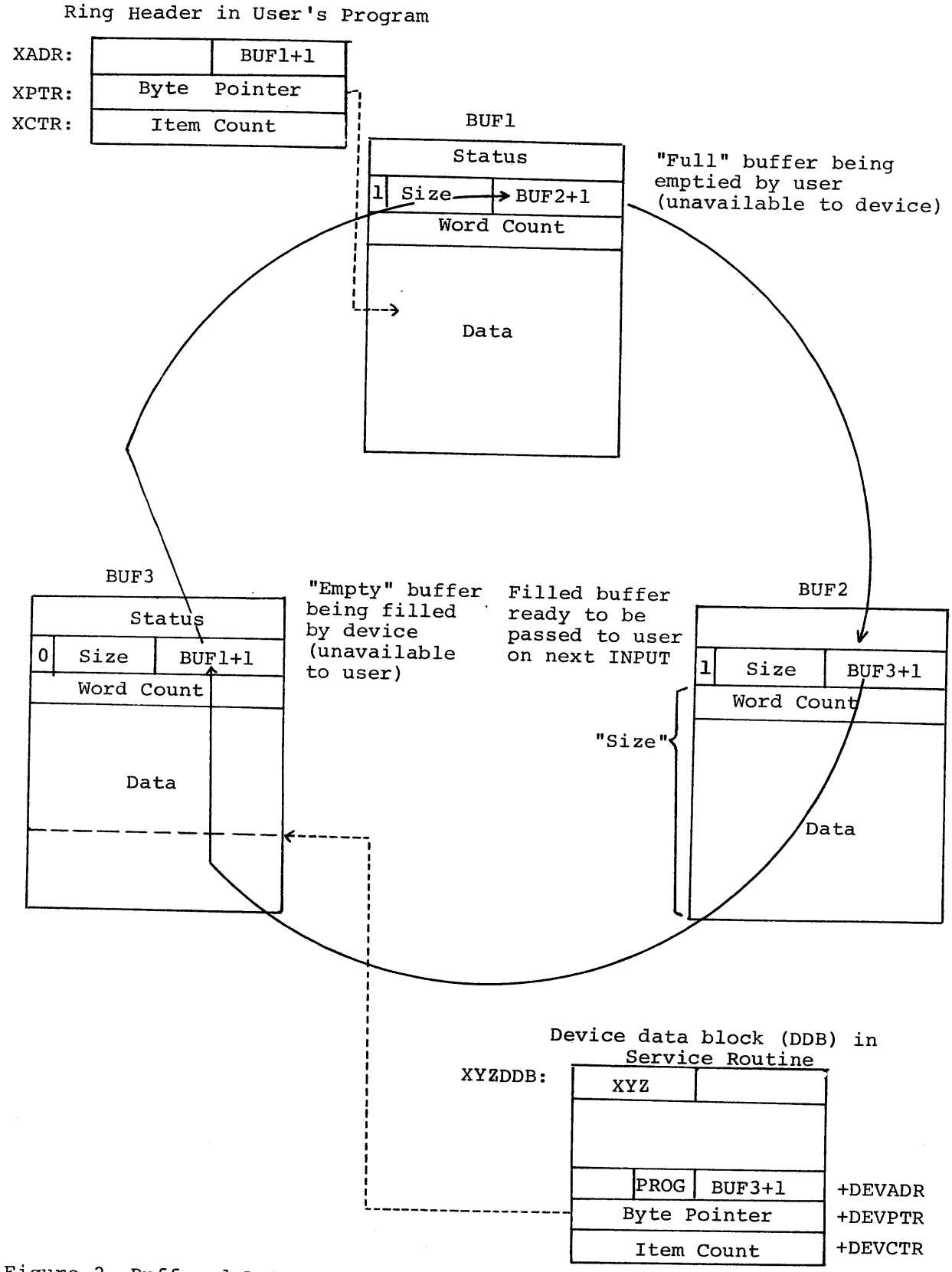


Figure 2. Buffered Data Transfer Between an Input Device and User Via a 3-Buffer Ring

INIT AND OPEN OPERATORS

These operators assign a device to a user's program, establishing the link between the software channel and the device data block. The initial status, including data mode, is placed in the DEVIOS word, and the DEVBUF word is given the relative addresses of the output ring header and input ring header, if specified. A byte pointer size field according to mode is placed in the second word of each ring header. An error return to the user occurs if the device is not found or is unavailable at this time. No dispatch to the device service routine is necessary for INIT or OPEN. See Figure 3.

INBUF AND OUTBUF OPERATORS

These operators create a buffer ring in free locations in the user area. The number of buffers is specified by the user as the effective address of the operator (one buffer is established if that value is equal to or less than 1). The size of each buffer data area is obtained from the righthand 12 bits of DEVCHR for the device assigned to the software channel. Two words are added to this amount for buffer head use.

As each buffer is appended to the ring, the last word of the buffer is address checked. A use bit of 0, the buffer size, and the link to the next buffer in the ring is inserted into the 2nd word of the new buffer. The 2nd word of the last buffer created is made to point to the first buffer, thereby closing the ring. JOBFF is then updated to point to the first location beyond the last buffer of the ring. Depending upon the operator, either DEVIAD or DEVOAD receives the relative address of the 2nd word of the first buffer. The first word of the user's ring header is then set with a "virgin" ring use bit and the address of the first buffer. No dispatch to the device service routine is necessary.

See Figure 4.

INPUT OPERATOR

Dump (Unbuffered) Mode

The device independent part of this operation is quite simple. The Monitor waits, if necessary, until the device is inactive, then dispatches to the device service dump mode input routine (device's dispatch table entry indexed

by "DDI"). The device service routine takes care of command list checking, initializing its interrupt level program, and starting the device. Upon return to UUOCON, the routine WAIT1 is called to place the job in an I/O Wait until the device becomes inactive. Dump mode input, therefore, goes on at interrupt level for this job while other user's jobs are running. When the device service routine recognized the end of input activity, it calls a routine (SETIOD - "set I/O done") that notifies the Scheduler to take this job out of I/O Wait. The job (at the appropriate time) then commences running, in this case at the UUOCON normal exit.

Buffered Modes

To somewhat simplify this description, let us take the first INPUT and subsequent INPUT's as two separate cases. Reference should be made to the flow chart (Figure 5).

Case 1 - First Issuance of INPUT operator

IN	If the device is doing output (IO=1), force output to stop at the end of next buffer and wait until it has done so. Zero the input close bit in the left half of DEVDAT (a copy of USRJDA for this channel).
IN1	If a buffer ring has not been established (by INBUF, for instance) or if this is the first INPUT ("virgin" ring), go to do first input, INPUTF.
INPUTF	Clear the header ring use bit. If a ring has not been set up, go to INPUT3. Otherwise, take the address of the first buffer from the first word of the ring header and place it in DEVIAD of the DDB. PUSHJ to CALIN to start the device. Device service then returns here and control falls into INPT0A.
INPT0A	Is the buffer's use bit a 1 yet? (Probably not, because we have just started the device.) If not a 1, call INPT2.
INPT2	Calls WSYNC to place the job in an I/O Wait state until the device calls SETIOD at the end of the buffer. If the buffer use bit is now a 1, go to INPUT2.
INPUT2	Gets the word count from the buffer and calls IOSETC which sets up the correct byte pointer and item counter in the user's ring header, and then exits to the user.
CALIN	If the device has previously sensed end of file (IOEND=1), return immediately; otherwise, address check the buffer

limits and dispatch to the device's input routine. The device routine initializes itself for interrupt-level data transfers, starts the device, and returns. CALIN then returns to the calling routine.

Case 2 - Subsequent Issuances of INPUT Operator

IN1 — If a buffer ring exists and has been used, test the use bit of the buffer now being returned by the user. If the use bit is 0 (as when a user's program is doing OUTPUT from this same ring), bypass header updating and go to INPT1. If the use bit is a 1 (probably more typical), clear it and advance the header first word to point to the next buffer in the ring. If the device is active, (IOACT=1), go to INPT0C; else, determine if it is time to make the device active. For all devices except the Teletype, this determination is made by looking at the use bit of the buffer beyond the one to be returned to the user. If it is 0, CALIN is called to start the device. In the case of Teletype, the new buffer's use bit is examined because a Teletype has a Monitor buffer in addition to the user's ring. Whether or not a call to CALIN is made, control now goes to INPT0C.

INPT0C. — The new buffer's use bit is fetched for examination, and control passes to INPT0A.

INPT0A — If the buffer's use bit is a 1, go to INPUT2, as described under Case 1; otherwise, go to INPT2.

INPT1 — If the device is active, go to INPT2; else, call CALIN to start the device and then return to INPT2.

INPT2 — Calls WSYNC to put the job in an I/O Wait if the device is active. Control returns if the device is not active or when the device calls SETIOD. If the buffer use bit is a 1 upon return, control goes to INPUT2; if not, control goes to INEOF.

INEOF — Did control get to here because of an error or end of file? If not, go to INEOFE. If so and the cause was end of file (IOEND=1), then set the user end file bit, IODEND. The use of these two end-file bits simplifies user programming by guaranteeing that when he detects end of file (with a STATX operation), there is no residue of information in a buffer. For instance, if a user's INPUT causes a buffer to partially fill and then an EOF to be detected by the device, the device returns the "full" buffer to the user while remembering the end condition (IOEND+1). IOEND is not detectable by the user, so he empties the buffer until the reduced header item count forces a call to INPUT. The IOEND bit prevents CALIN from starting the device which ultimately causes (at INPT2) an immediate return from WSYNC with a 0 buffer use bit. INEOF now finds IOEND=1 and turns on IOEND, then returns to the user.

INEOFE — Control should never get here. If it does get here and any of the job status STOPIO bits are on, control goes back to UUOCON dispatch and the input UUO is repeated to make the device do something (fill one buffer, or return with EOF or error bits). If no STOPIO bits are set, the error message routine UERROR is called to print

```
? ERROR IN MONITOR AT EXEC nnn
```

and stop the job.

OUTPUT OPERATOR

Dump (Unbuffered) Mode

UOUT — Set output UUO bit and clear output close bit in left half of DEVDAT.

OUT — If device is busy doing input, wait until the next bufferful. When stopped, if dump mode, go to OUTDMP.

OUTDMP — Call WSYNC to make sure device is inactive, then dispatch to the device's dump mode output routine (dispatch table "DDO" index). The device routine checks the command list, starts the device, and returns. WAIT1 is called to place the job in an I/O Wait state until the entire output is done. Control then returns to the calling routine.

Buffered Modes

OUT — If not dump mode, call OUTA to get new buffer address if user specified one. If a buffer ring has not been set up or if this is the first OUTPUT, go to OUTF. Otherwise, if the user is not computing his own word count, take the header item count, convert it to word count, and store it in the third word of the buffer. Don't compute if the user so indicates. Go to OUT2.

OUT2 — Turn on the buffer's use bit, then advance the header to the next buffer. If the device is not now active, dispatch to the service routine to start it going. If the new buffer is not empty, call WSYNC to put the job in I/O Wait until the buffer is empty (the device calls SETIOD at interrupt level to take the job out of Wait). Go to OUTS.

OUTS — Calls BUFCLR to clear the buffer, then calls IOSETC to set the ring header byte pointer and item counter for this device. Return to calling routine.

OUTF — If a buffer ring has not been established, call UOUTBF

(OUTBUF operator routine) to set up a 2-buffer ring. Go to OUTF1 in any case.

OUTF1 — Clear the ring use bit. Supply the address of the first buffer to DEVOAD of the DDB. Go to OUTS.

OUTA — If a new buffer address is not specified, return immediately. (NOTE: The mask used in making this test ignores bits 34 and 35. This is because OUT is called by the CLOSE routine in which case one of these bits may be a 1 to inhibit closing "half" the channel. We don't, in that case, want to believe that location 1 or 2 is being specified as the start of a new buffer!) If an address is specified, wait until the device is inactive, then put the new address into the user's ring header and DEVOAD. Mark the ring as being referenced (clear the use bit) and return.

N.B. Observe that if the first OUTPUT does specify a buffer address, the assumption is that the buffer contains data already, i.e., this will not be treated as a "dummy" output.

See Figure 6 for a flow chart of the OUTPUT operator.

CLOSE OPERATOR

CLOSE1 — Calls WAIT1 to be sure that the device is inactive before proceeding. If an input close is requested and the file has previously been closed, go to UCLS2. Otherwise, if the file was read in DECtape save mode (2), return. If not save mode and not dump mode, go to UCLSBI to close buffered input. If dump mode, dispatch to the device service routine input close function (dispatch index "DCLI") and return to UCLS2.

UCLSBI — If an input buffer ring was never established, go to UCLS2; else, if this is a long dispatch table device, dispatch to the device service routine (dispatch index "DCLI"). Then get the address of the first buffer from the ring header. If 0, go to UCLS1; otherwise, go to UCLS0.

UCLS0 — Address each buffer in the ring, clearing its use bit. Go to UCLS1.

UCLS1 — Set the ring use bit to 1 ("never referenced") and clear the item count word to 0. Clear both end-file bits in DEVIOS. Go to UCLS2.

UCLS2 — If an output close is not desired, or if already closed, go to UCLS3. If DECtape save mode (2) was used to write the file, go to UCLS3. If not dump mode, go to UCLSBO to

to close buffered output; otherwise, dispatch to the device service routine output close function (dispatch index "DCL"), then return to UCLS3.

UCLSBO — If an output buffer was not set up or never referenced, go to UCLS3; otherwise, fall into UCLS2A.

UCLS2A — If DEVOAD addresses an empty buffer, go to UCLS2B; otherwise, clear the device error bits and call OUT (the OUTPUT UUO routine) to output this buffer. WAIT1 is called to stall until the buffer is emptied and the device has advanced the buffers. If no device error occurred, return to UCLS2A (this loop will continue until all full buffers have been output). If a device error is detected, go to UCLS2B.

UCLS2B — Dispatch to the device service output close routine (dispatch index "DCL"). Then clear the ring use bit and item count in the ring header. WAIT1 is called to be sure the device is inactive, then UCLS3 is entered.

UCLS3 — Stores DEVDAT (in which the UUO bits have been modified) back into USRJDA for this channel, then returns to the calling routine.

See Figure 7 for a flow chart of the CLOSE operator.

RELEASE OPERATOR

RELEA0, RELEA1, RELEA2, RELEA3 — The routine CLOSE1 is called to close both the input and output sides of the channel. WAIT1 is then called to be sure activity has ceased. Go to RELEA5.

RELEA5 — Dispatches to the device service release routine (dispatch index "DRL"). Then clears the active bit in DEVIOS and the USRJDA entry for this channel. Fetches the number of the highest channel in use from USRHCU.

RELEA4, RELE4A — These sections of code perform two important functions while scanning USRJDA from the old highest channel down to 0. First, USRHCU is changed, if necessary, to point to the highest nonzero entry in the JDA table. Second, if it is discovered during the scan that the device just released in RELEA5 is also assigned on another channel, then no further release housekeeping is performed, and an immediate return occurs. (It is possible to INIT the input and output sides of a bidirectional device on two separate channels.) Otherwise, the DEVIAD and DEVOAD words in the DDB are cleared, and control goes to RELEA9.

RELEA9 — If the device is a disk or is not the system tape, go to RELEA7. If it is the system tape but has already been returned to the system, go to RELEA7; otherwise, clear out the system tape user word, decrement the request count, and set the available flag if someone is waiting. Go to RELEA7.

RELEA7 — Supplies the ASSPRG bit to RELEA6 (RELEA6 may also be called by the DEASSIGN command, in which case an ASSCON bit would be supplied).

RELEA6 — Clears the assignment bit supplied by the calling routine. If the device is still assigned by another means, an immediate return is made. If the device is now wholly deassigned (ASSCON and ASSPRG = 0), then the job number field of DEVCHR is cleared. If the device is DSK, the routine CLRDDB is called to return to free storage the space occupied by the DDB. Return to the calling routine.

See Figure 8 for a flow chart of the RELEASE operator.

LOOKUP AND ENTER OPERATORS

These operators are extremely device dependent, most of the work being performed by the device service routine. Before dispatching to the device service routine, however, LOOKUP performs an input close, and ENTER performs an output close, with appropriate alteration of the device status bits.

See Figure 9 for a flow chart of the LOOKUP and ENTER operators.

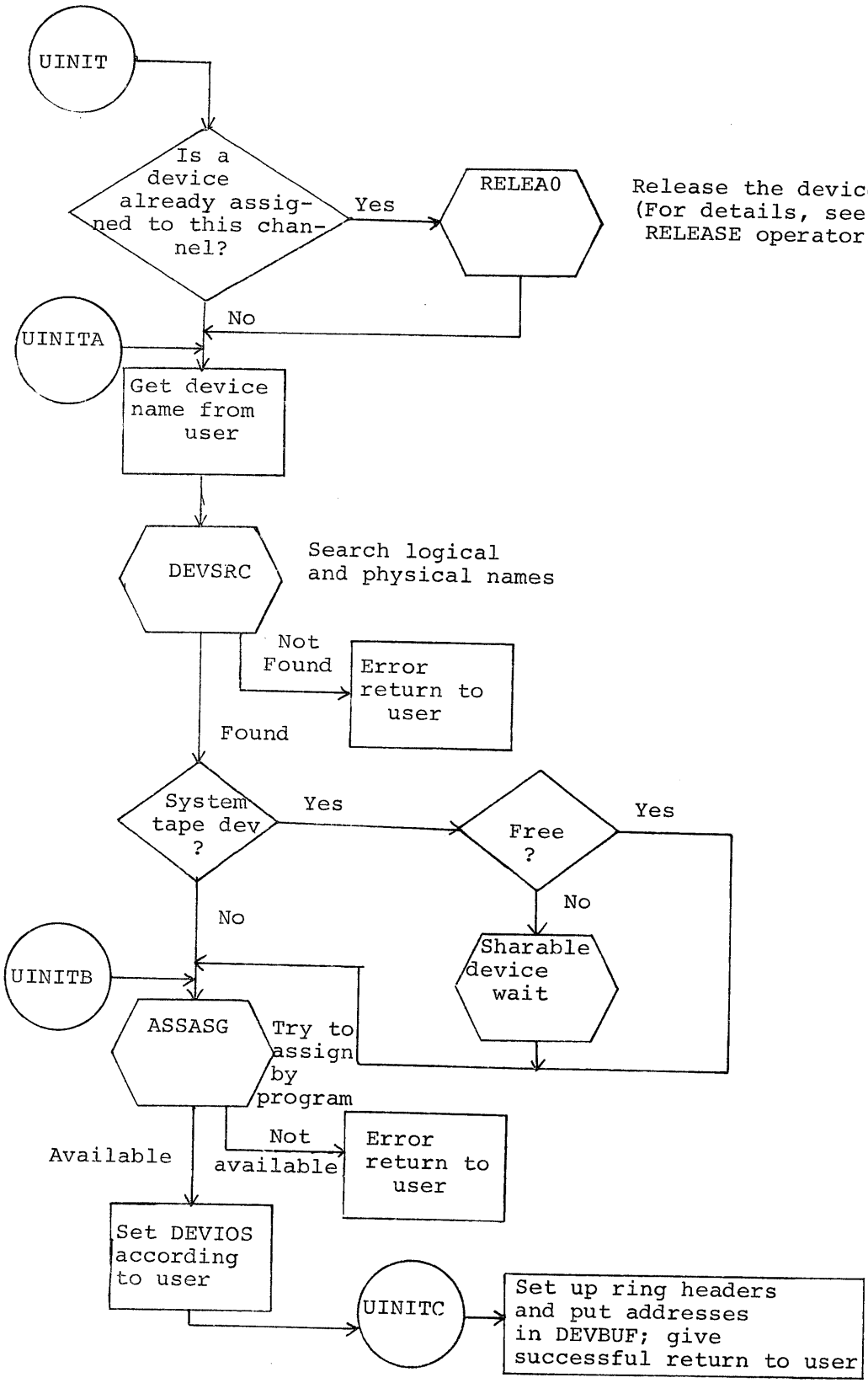


Figure 3. Flow Chart of INIT

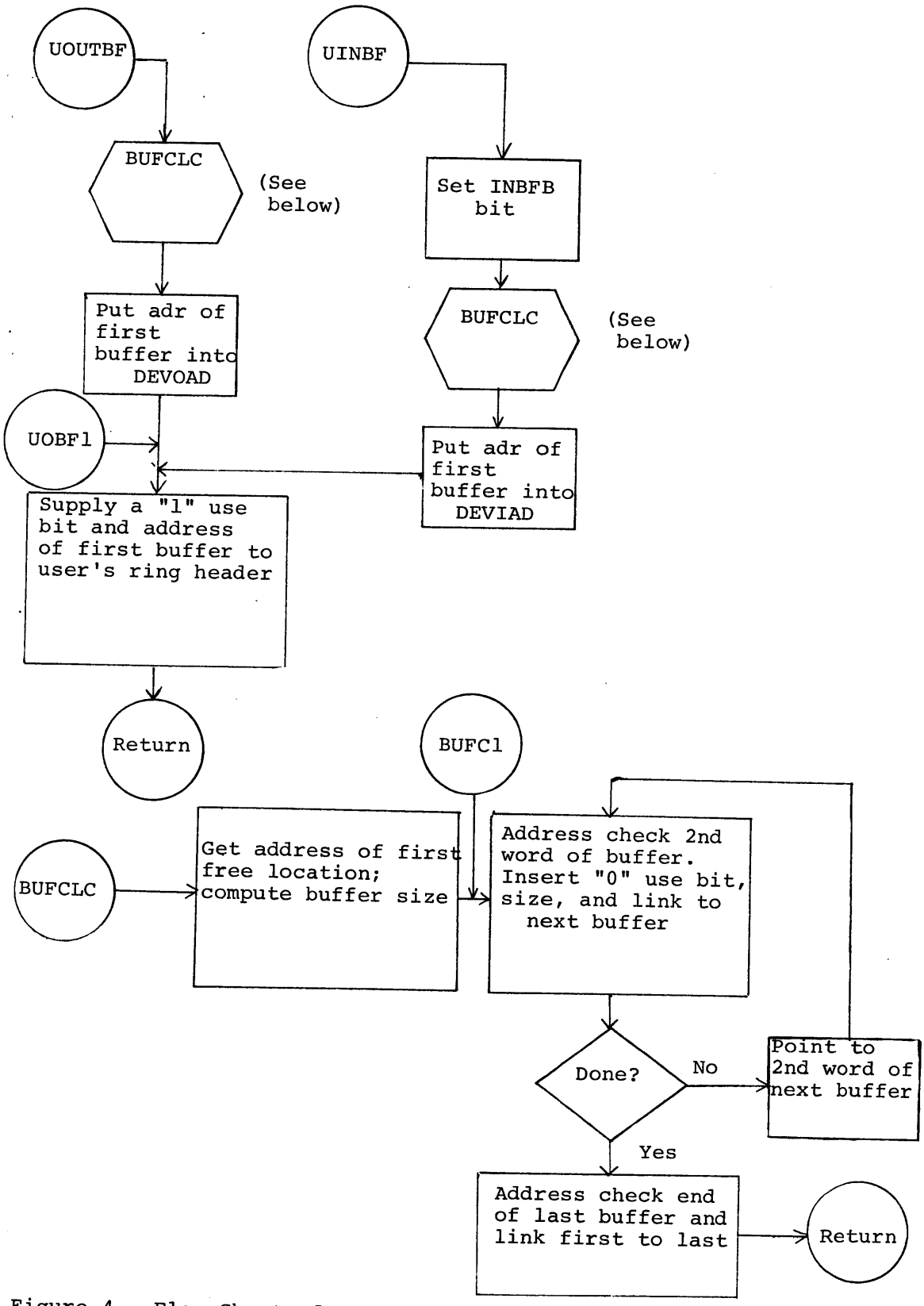


Figure 4. Flow Chart of INBUF, OUTBUF

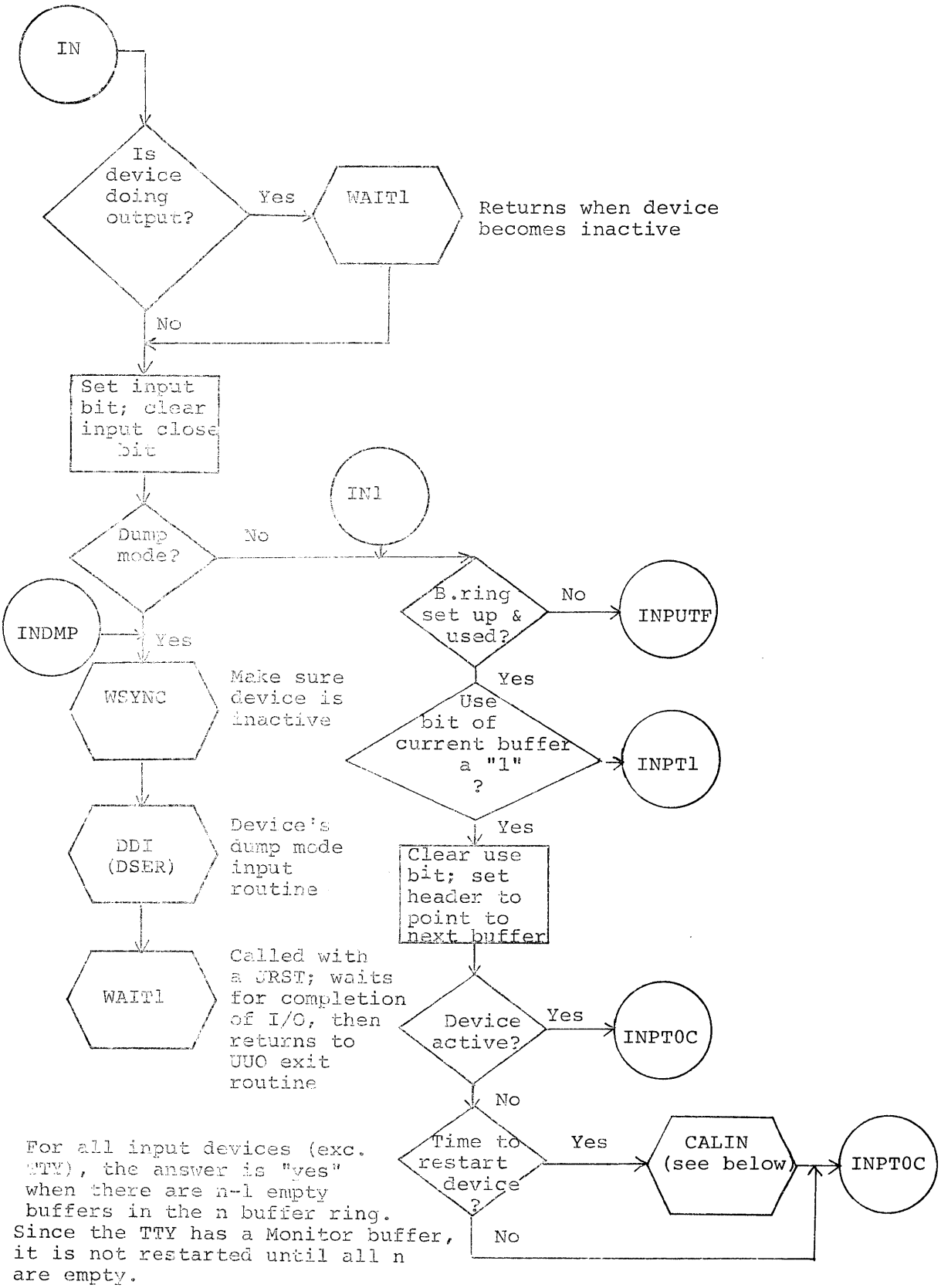


Figure 5. Flow Chart of INPUT Operator

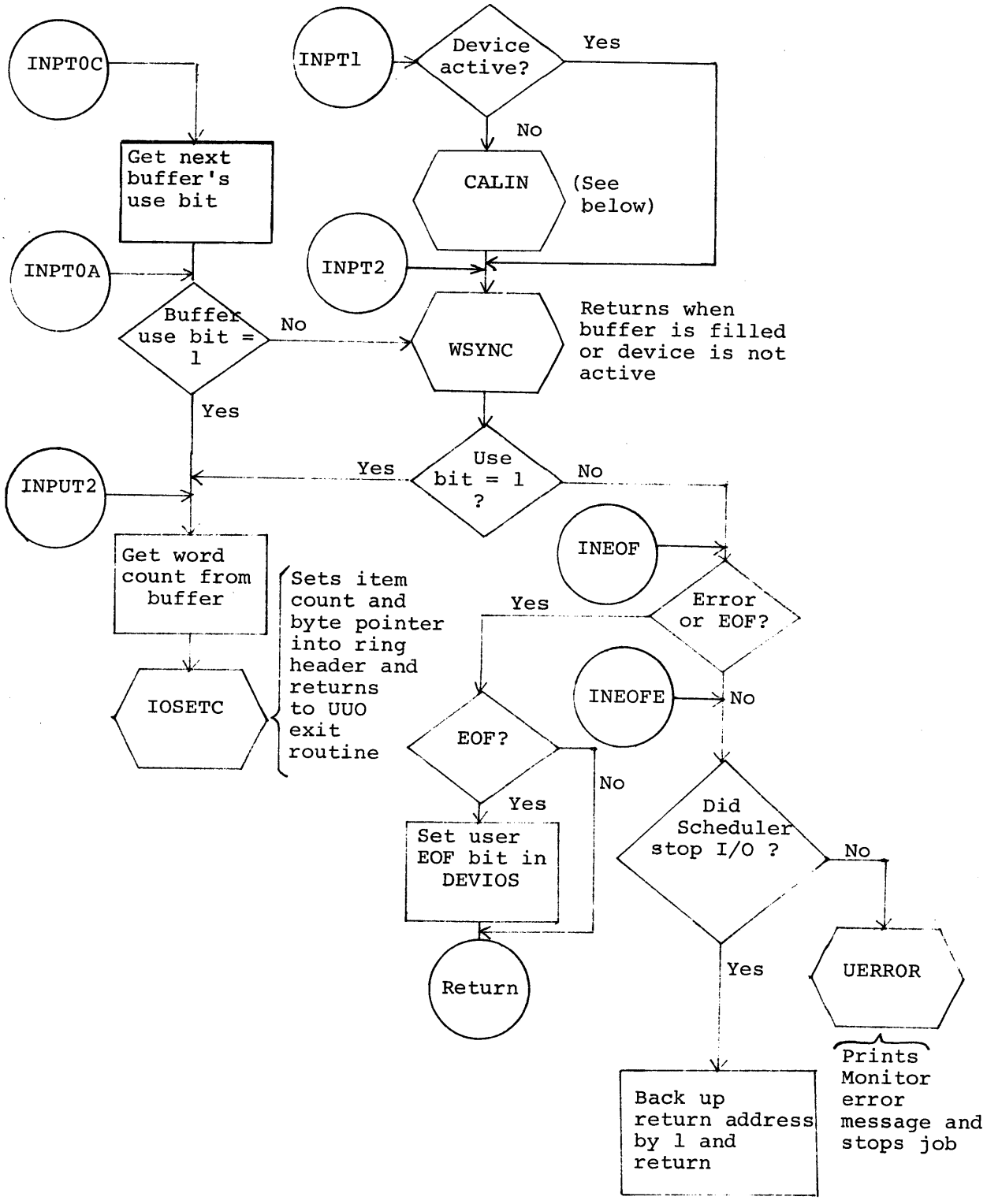
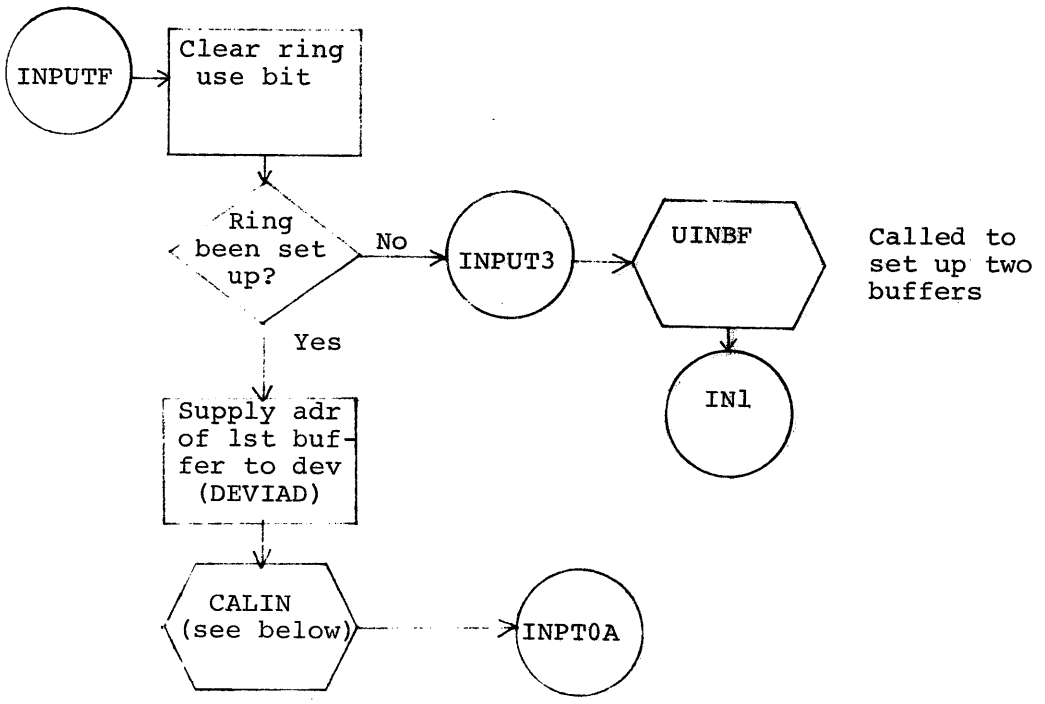


Figure 5 (Cont.) Flow Chart of INPUT Operator



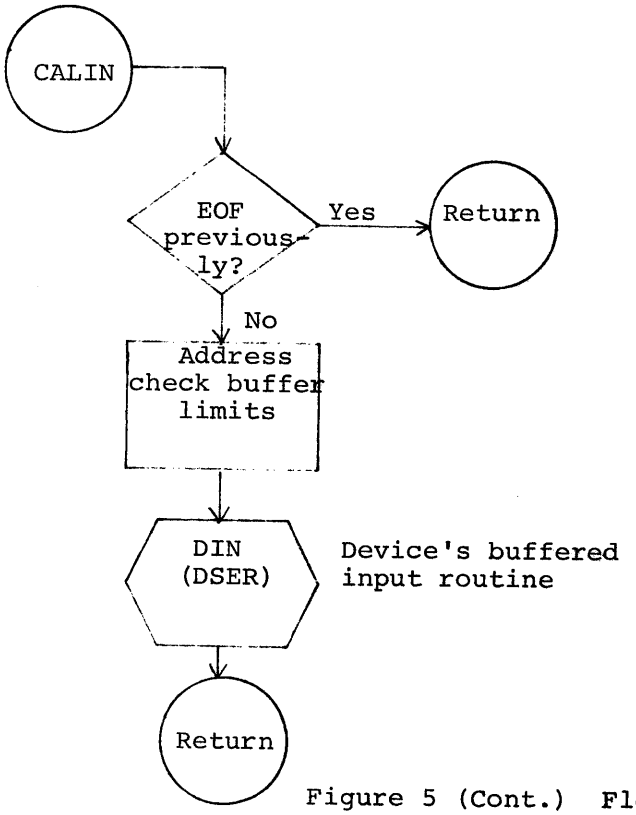


Figure 5 (Cont.) Flow Chart of INPUT Operator

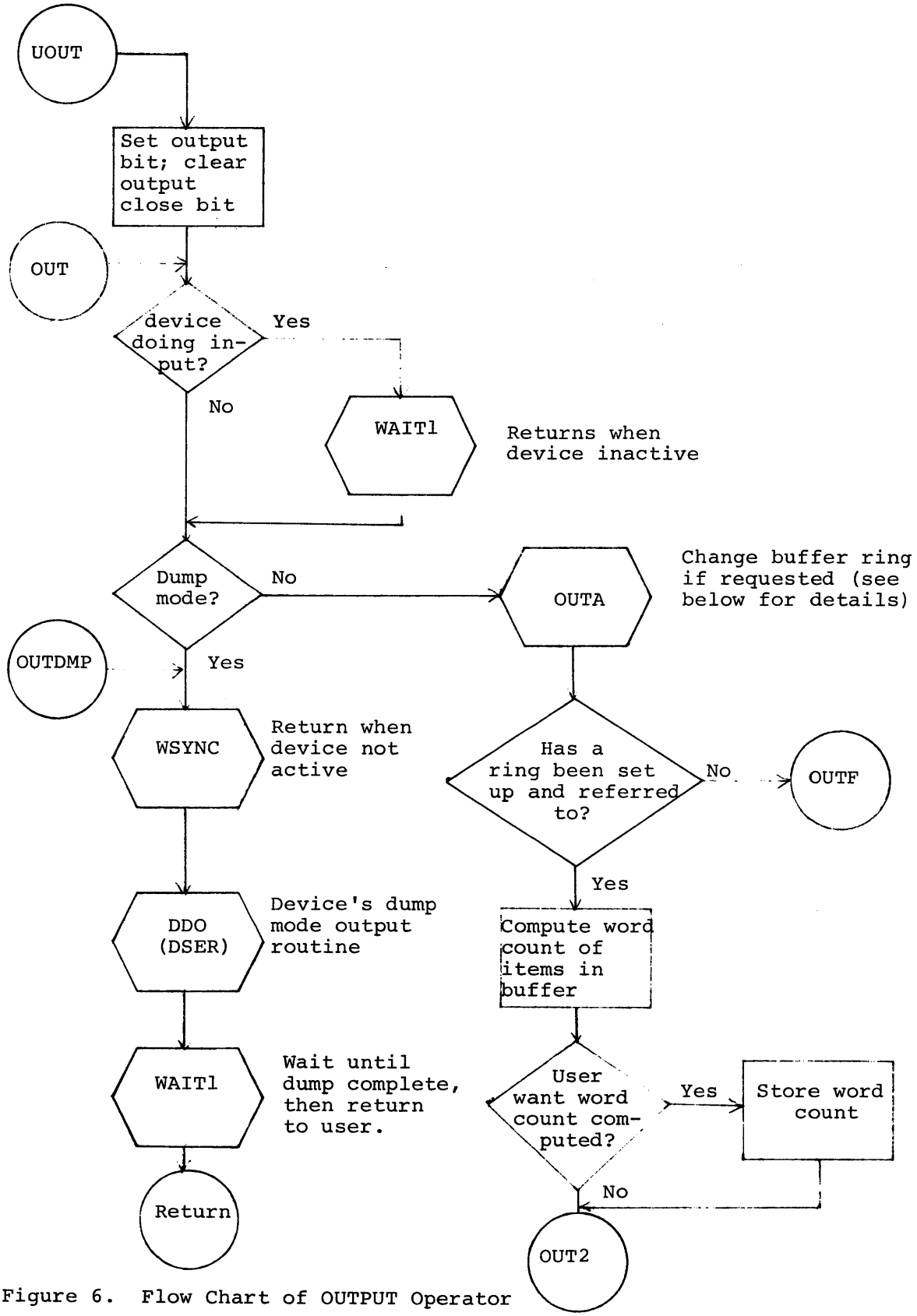


Figure 6. Flow Chart of OUTPUT Operator

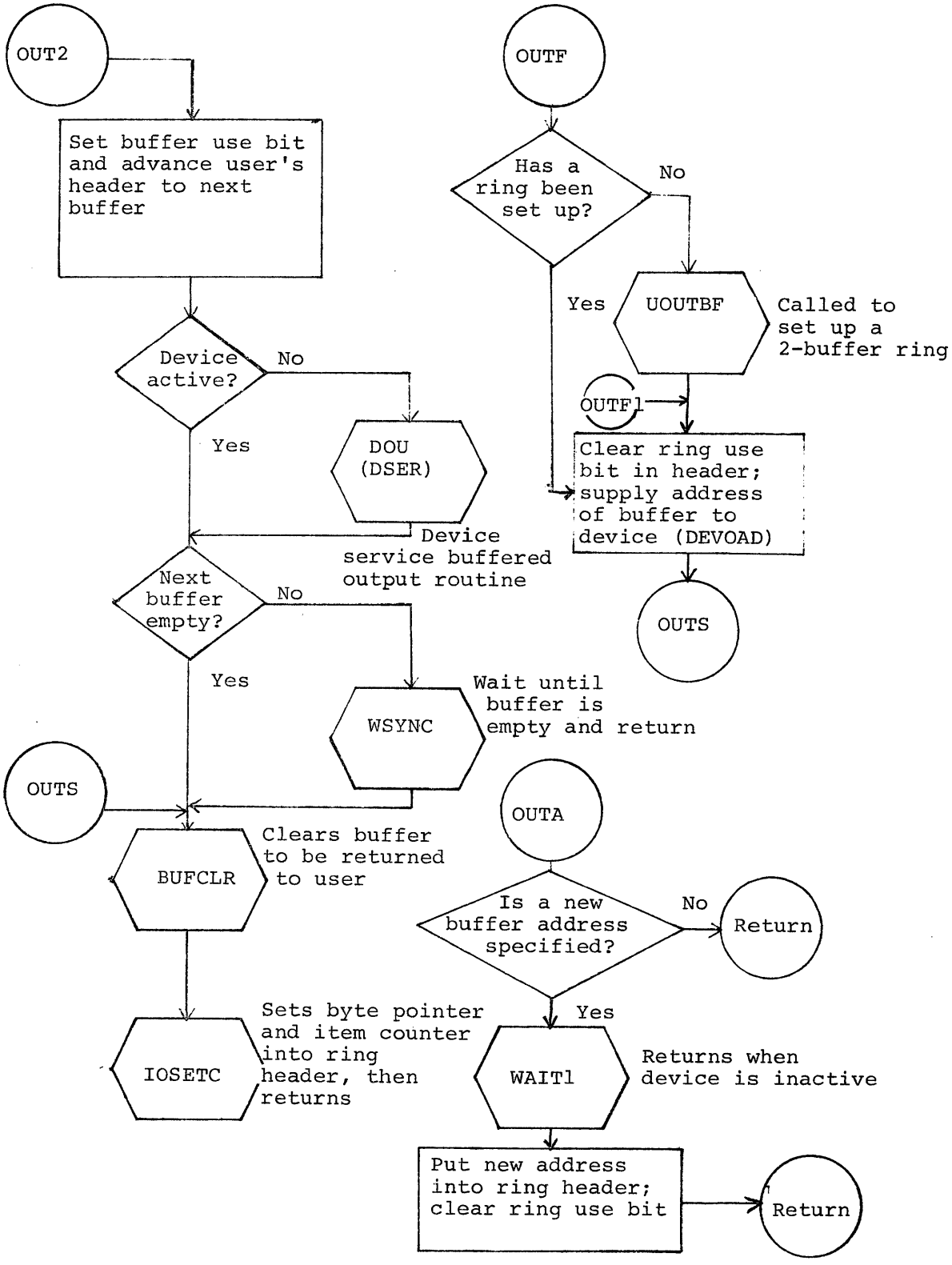


Figure 6 (Cont.) Flow Chart of OUTPUT Operator

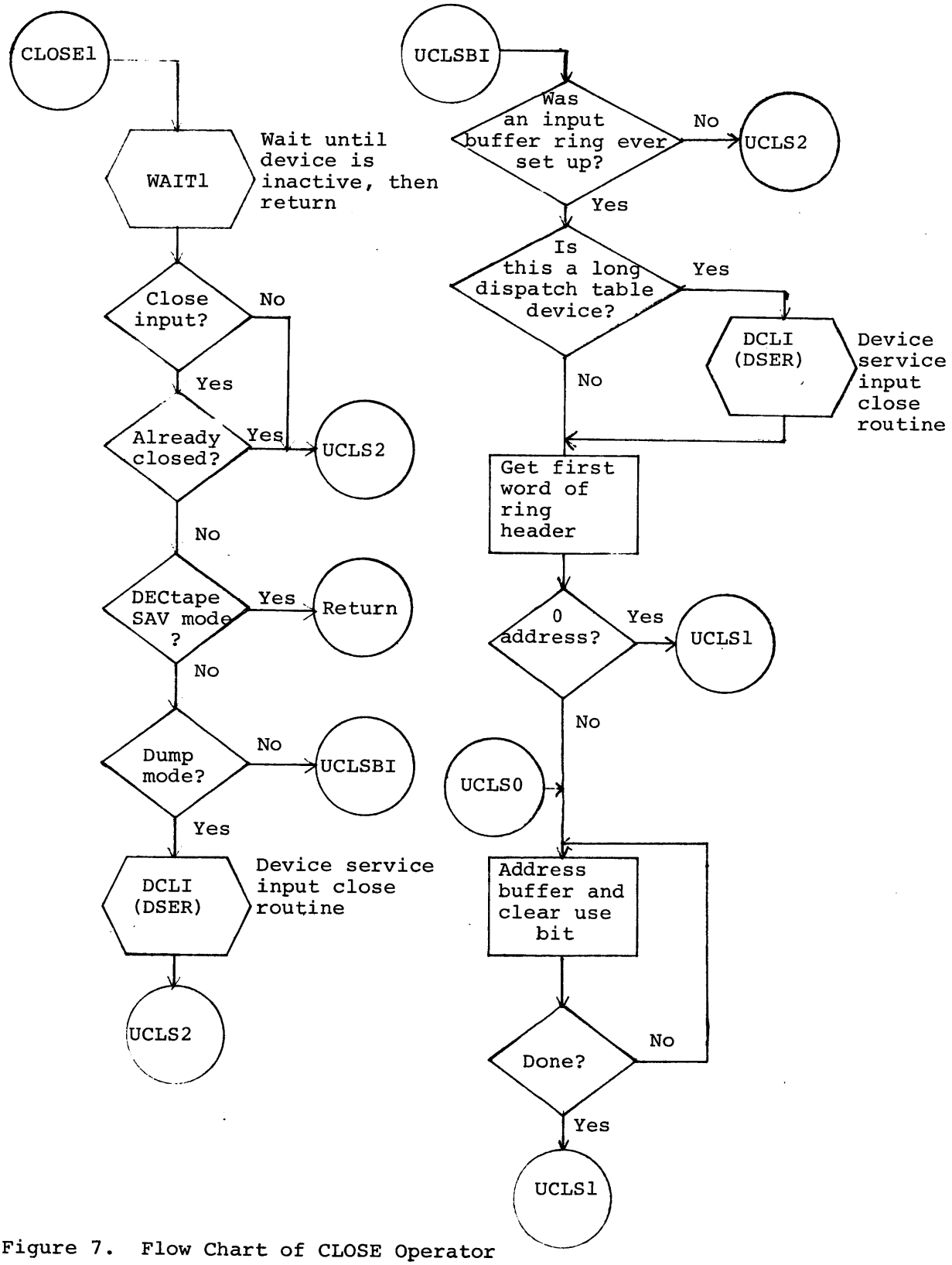


Figure 7. Flow Chart of CLOSE Operator

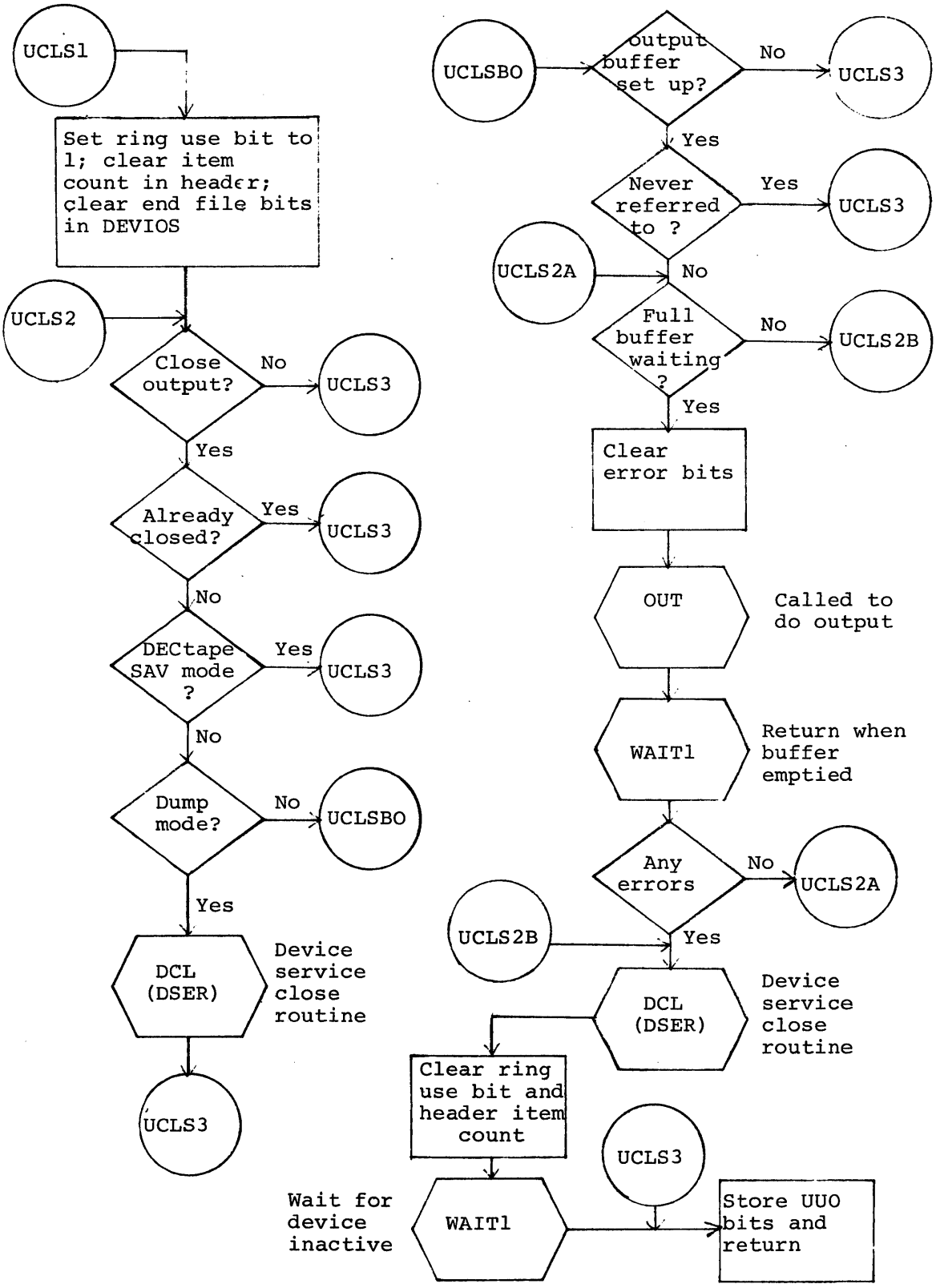


Figure 7 (Cont.) Flow Chart of CLOSE Operator

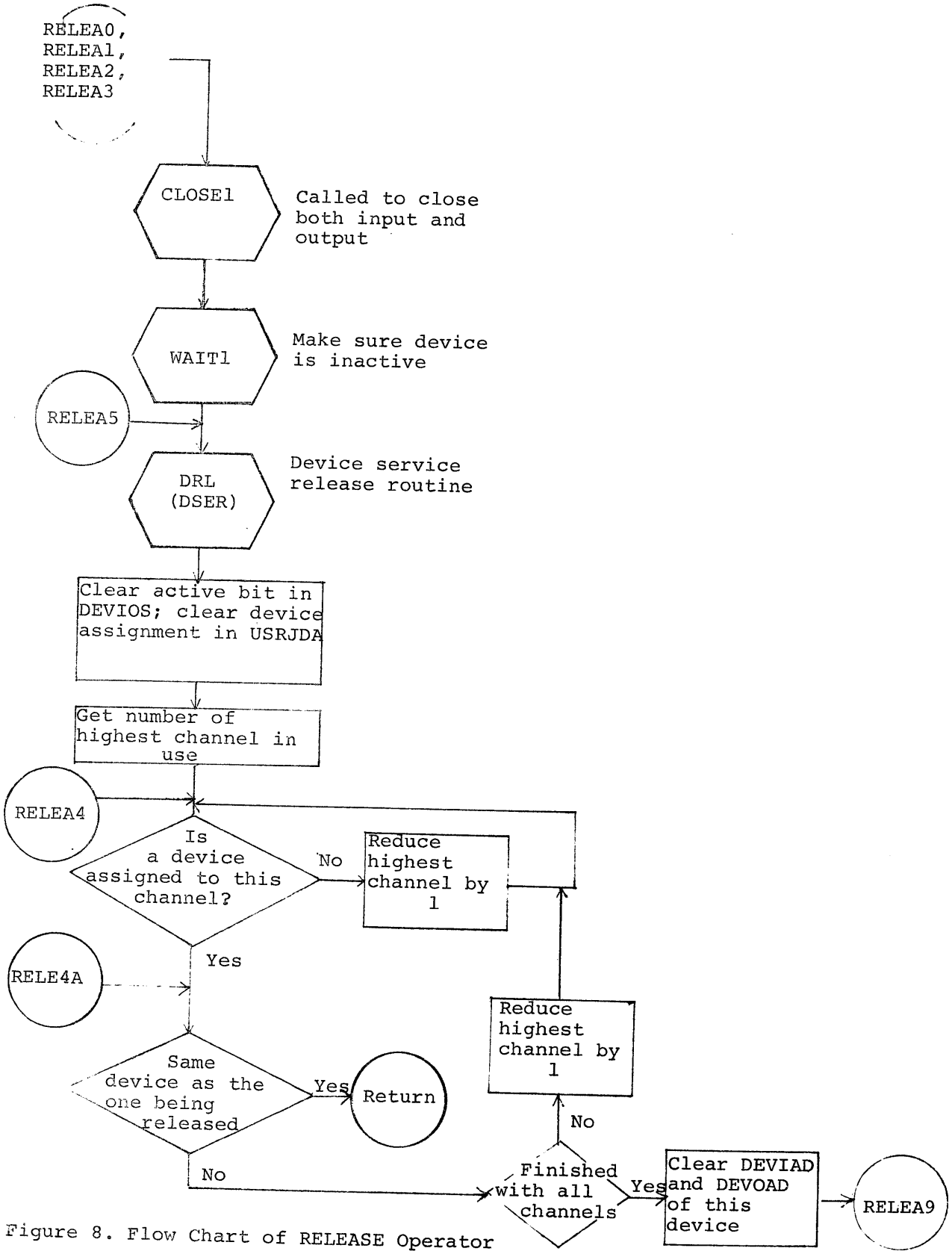


Figure 8. Flow Chart of RELEASE Operator

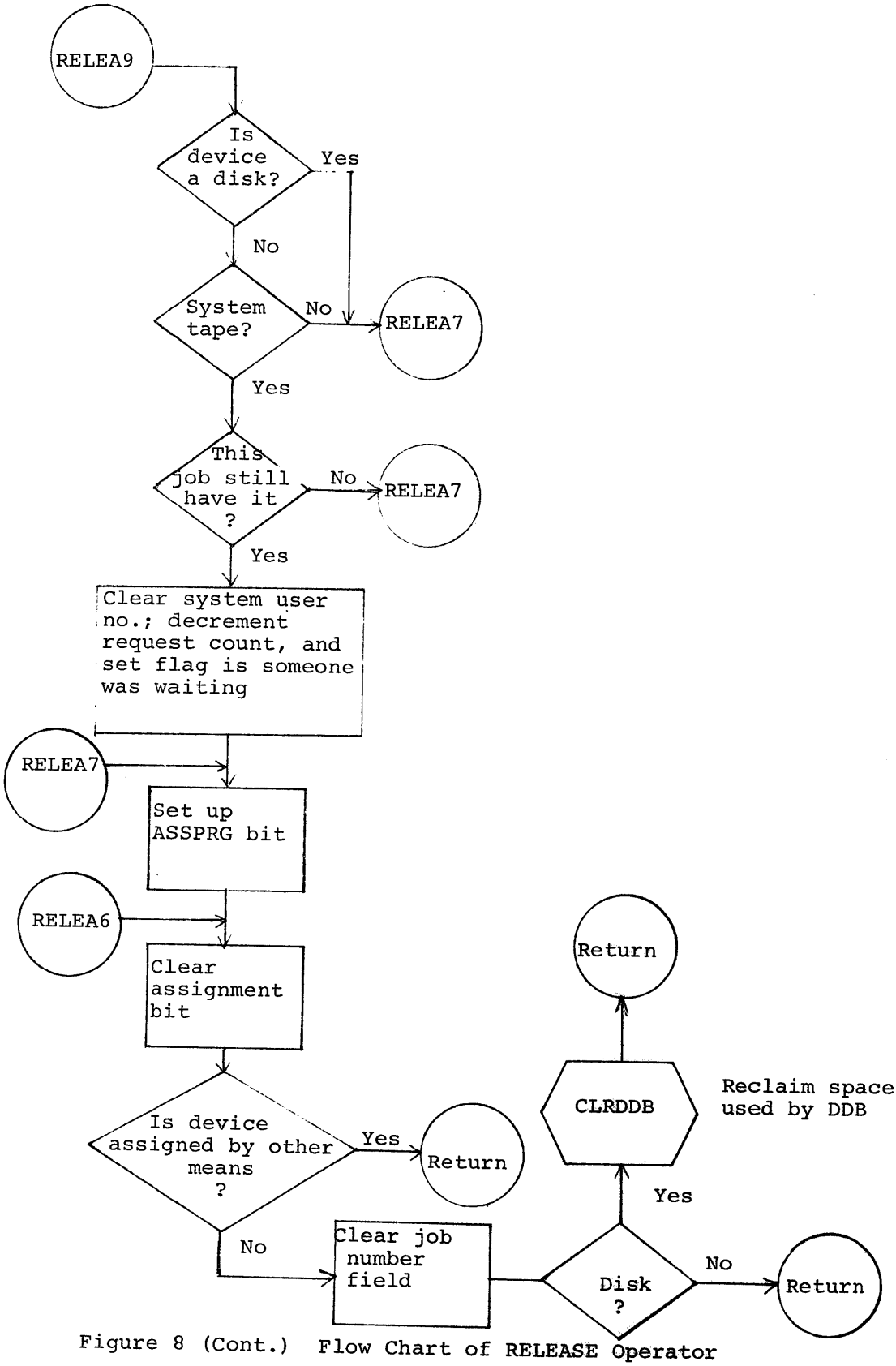


Figure 8 (Cont.) Flow Chart of RELEASE Operator

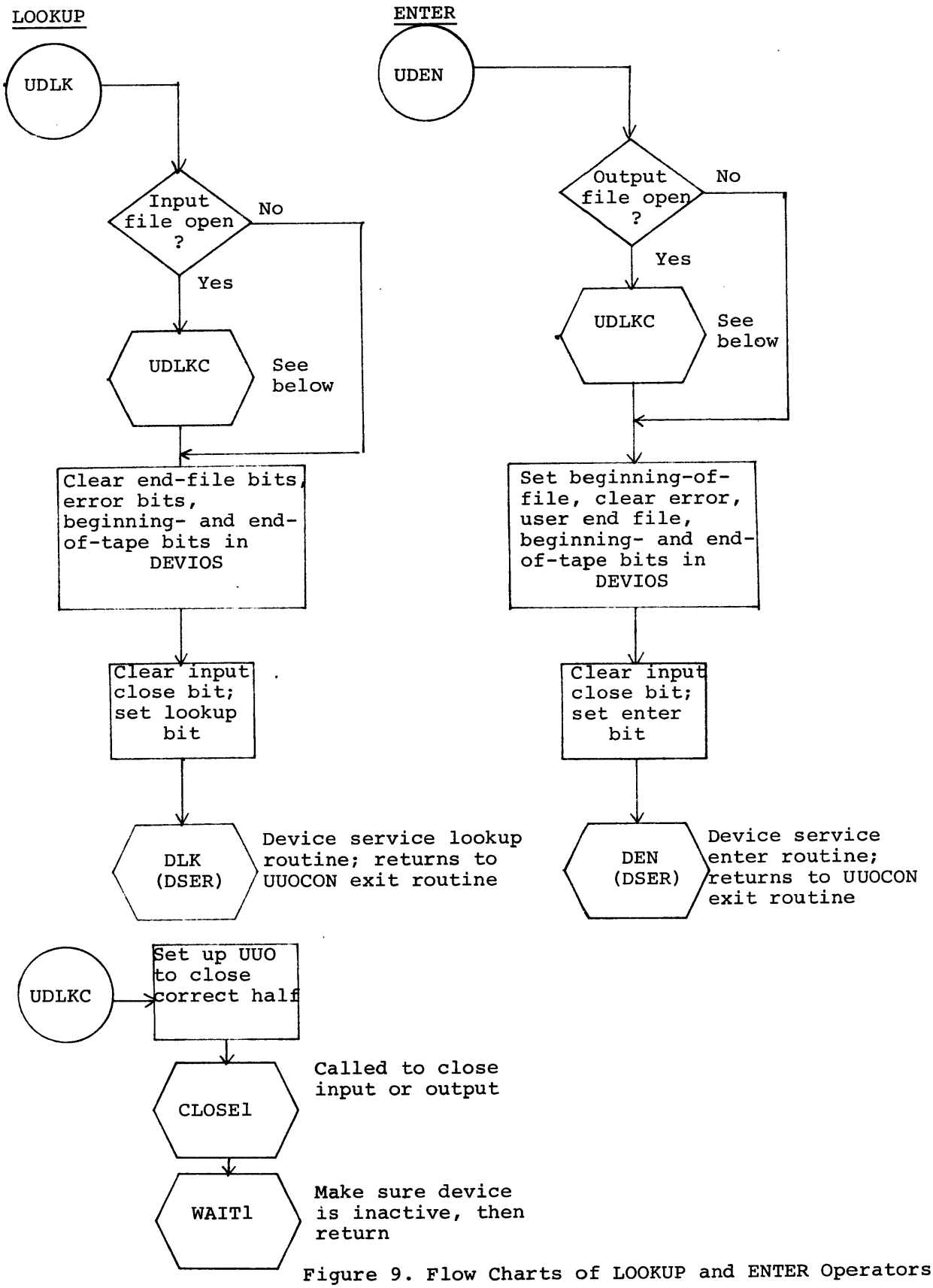


Figure 9. Flow Charts of LOOKUP and ENTER Operators

DEVICE DATA BLOCKS

(See Figure 10)

The device data block (DDB) structure is the key to I/O handling on the UUO level in the PDP-10 Monitors. Each physical device is represented by a block of words beginning at dev DDB, where dev is the 3-letter device mnemonic. The contents of the device data block completely describe a particular device at any given time; this description includes the physical characteristics of the device, the I/O status of the device, and the information required to link sections of the Monitor that communicate with one another while referring to the device described by the data block. While any routine is referring to a DDB, its address (devDDB) is kept in the accumulator DEVDAT, which is then used as an index register.

Each location in a DDB is known by a logical 6-letter mnemonic, which is defined in the System Parameter Tape to be a constant equal to the address of the location relative to devDDB (the address of the specific device data block). Thus, DEVxxx(DEVDAT) is the address of a specific word in a particular DDB, where DEVxxx represents the relative DDB location. Following is a description of the function of each location within the DDB, starting with the first word, DEVNAM (DEVNAM=0; others in ascending order.

DEVNAM — Contains the physical device name, left justified, in 6-bit ASCII (in the case of multiple devices, this causes the device number to fall left justified in the right half).

DEVCHR — Contains information giving the device assignment, hung device count, buffer size, binary device number (the bits set in each word are defined in the System Parameter Tape).

DEVIOS — Contains bits describing the current I/O status of the device. The left half is used only by the Monitor, while the right half becomes a user's device status register, which can be referred to by GETSTS, SETSTS, STATO, and STATZ UUO's (see Table 1 for DDB bit definitions).

DEVSER — Contains system linking information. The left half contains the address of the next DDB in a "chain" of all DDB's; the address of the first DDB in the chain is in the left half of DEVLST (a location in COMMON), while the last DDB in the chain has zeroes in the left half of DEVSER. The right half

contains the address of the Device Service Dispatch Table, which is referred to by UUOCON.

DEVMOD — The left half contains bits which, for the most part, describe the physical characteristics of the device; most of these are assembled as part of the DDB. These bits can be called by the user with a GETCHR or DEVCHR UUO (this is not to be confused with the DEVCHR DDB word - see above). The right half has bits indicating whether assignment of the device was by console command and/or by the INIT UUO, as well as bits reflecting which data modes are legal for the device (see Table 1).

DEVLOG — Contains the logical device name (left justified, in 6-bit ASCII) assigned by the user from the console Teletype. When executing the INIT UUO, which links the word in location USRJDA (UCHN) with a DDB, the Monitor scans the contents of DEVLOG through the DDB chain before trying to match the user's specified device with the contents of DEVNAM.

DEVBUF — Contains addresses of buffer headers associated with the device by INIT UUO; the left half contains the output header address, while the right half contains the input header address.

DEVIAD (or DEVADR) — Contains the address of the user's input buffer which is currently being filled (DEVIAD=DEVADR=7).

DEVOAD (or DEVPTR) — Contains the address of the user's output buffer currently being emptied.

Note: In the time-sharing Monitor, the accumulator used for relocation (PROG) is designated in the index register field of both DEVIAD and DEVOAD.

DEVCTR (or DEVFIL) — Contains item count for the buffer (same as the third word of user's buffer header). For directory devices which have long dispatch tables, this location is called DEVFIL and contains the 6-bit ASCII name of the file being referred to, while the next location (DEVEXT) contains the extension, if any, of the file.

There are devices that reserve more locations for the DDB's than those mentioned above, but these additional locations are required by the special characteristics of the particular device rather than by the system itself.

When a device service routine services a class of multiple devices (e.g., DTASER services DTA0, DTA1, ...etc.), only the DDB of the first device, DEV0 is assembled into the routine. The rest of the blocks are loaded outside the routine by ONCE, being modeled after the DTA0 DDB and being linked in the chain via DEVSER. ONCE determines the number of DDB's to create for a device service routine from responses received during the console dialogue.

System Index

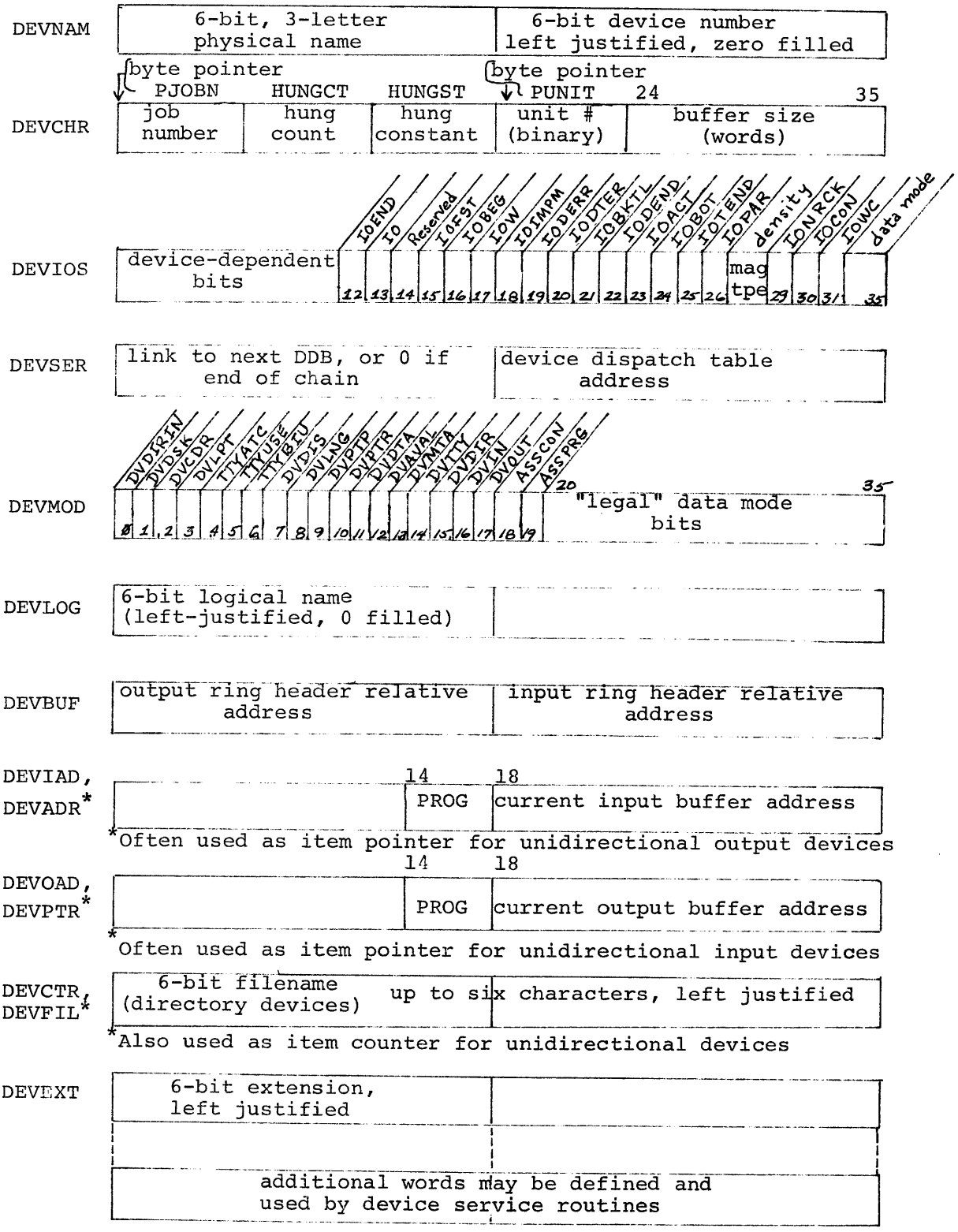


Figure 10. Device Data Block (DDB)

Table 1. Device Data Block (DDB) Bit Definitions

DEVIOS		I/O Status
	IOEND	Set at interrupt level by input device when end of file recognized
	IO	Direction of transfer: Out = 1, In = 0
	IOFST	Set by service routine to indicate that next interrupt is first item of a buffer.
	IOBEG	Set by INIT or ENTER operator to indicate a "new file"
	IOW	Set when a job is placed in an I/O Wait State
	IOIMPM	Improper mode detected by input service routine
	IODERR	General device error bit
	IODTER	Device data error bit
	IOBKTL	"Data block too large" error
	IODEND	End of file (to user)
	IOACT	Device active, expecting interrupts
	IOBOT	Beginning of magnetic tape
	IOTEND	End of magnetic tape
	IOPAR	Write even parity (mag tape command)
	IONRCK	Read with no reread
	IOCON	Discontinuous I/O if set to 1. Device stops after filling or emptying each buffer
	IOWC	Inhibit system computation of word count for output device
DEVMOD		Device characteristics and legal data modes
	DVDIRIN	Directory is in core
	DVDSK	Device is a disk
	DVCDR	Device is a card reader or card punch
	DVLPT	Device is a line printer
	TTYATC	This Teletype is attached to a job
	TTYUSE	This Teletype is in user mode
	TTYBIU	Teletype DDB in use
	DVDIS	Device is a display
	DVLNG	Device service routine has a long dispatch table
	DVPTP	Device is a paper tape punch

Table 1 (Cont.) Device Data Block (DDB) Bit Definitions

DVPTR	Device is a paper tape reader
DVDTA	Device is a DECtape
DVAVAL	Device is available (set by DEVCHR UUO)
DVMTA	Device is a magnetic tape
DVTTY	Device is a Teletype
DVDIR	Device has a file directory
DVIN	Device is capable of doing input
DVOUT	Device is capable of doing output
ASSCON	Device has been assigned by a console command
ASSPRG	Device has been assigned by a program (INIT or OPEN)

Bit (35 - n) is a 1 if mode n is legal for this device

NOTE

DVCDR is set for both CDR and CDP. DVIN and DVOUT distinguish which device it actually is.

Mode	n (decimal)
ASCII	0
ASCII line	1
DECtape SAV	2
Image	8
Image Binary	11
Binary	12
Image Dump	13
Dump Records	14
Dump	15

Dispatch Table

The Monitor dispatches to device dependent coding via a dispatch table located in that coding. The base address of this table exists in accumulator DSER during the processing of an I/O operator. The dispatch is usually performed by a PUSHJ PDP, INDEX (DSER), where INDEX is a constant used to select the appropriate entry of the table. See Figure 4 for an illustration of such indices. A "basic" dispatch table has six entries and is sufficient for service routines of "simple" physical devices such as card readers, tape punches, and line printers. Devices which require directory maneuvers or complex activities in file positioning use a so-called "long dispatch table" containing 17 entries (including the "basic" ones). Examples of these are DECtape, magnetic tape, and disk. Before attempting to dispatch on a "long-type" UUO, Monitor checks the DVLNG bit in the DEVMOD word (see routine DISP1 in UUOCON, for example).

Table 2. Device Service Dispatch Table Entries

	System Index			Purpose
"BASIC"	-2	DINI		Device and service routine initialization
	-1	DHNG		"Hung device" action
	0	DRL	DEVDSP:	Release (table base address)
	1	DCL, DCLO		Close, close output
	2	DOU		OUTPUT Operator
	3	DIN		INPUT Operator
"LONG"	4	DEN		ENTER Operator
	5	DLK		LOOKUP Operator
	6	DDO		Dump Mode output
	7	DDI		Dump Mode input
	10	DSO		USETO Operator
	11	DSI		USETI Operator
	12	DGF		UGETF Operator
	13	DRN		RENAME Operator
	14	DCLI		Close input (dump mode)
	15	DCLR		CALL X,[SIXBIT/UTPCLR/]
	16	DMT		MTAPE Operator

Basic Operations

This section attempts to describe, in summary fashion, the actions performed by the device service routine upon receiving one of the six "basic" dispatches.

1. Initialization (Index DINI)

Entered from SYSINI Monitor initialization when Monitor is first loaded or upon certain restarts. The service routine should set the hardware control unit to some known free state (usually a CONO DEV, 0). The routine may also have to preset its own software "flag" registers (the mask bits for interrupt level "CONSOing" are usually kept in a register, DEVCON, which should now be cleared). Return to the calling routine is via a POPJ PDP,

2. Hung Timeout (Index DHNG)

Entered from routine DEVCHK at clock interrupt level (refer to CIP5 in CLOCK). When a device is started by an INPUT or OUTPUT operator and each time an interrupt is serviced for this device, the HUNGCT field of DEVCHR is set to the value HUNGST. Every second, the HUNGCT field of all active devices is examined and, if nonzero, decremented. If decrementation causes HUNGCT to become zero, this dispatch is made (preloading of HUNGST with zero will prevent this from ever occurring). Upon return (via a POPJ), the Monitor types out an informative message on the user's console and places the job in an error stop state.

3. RELEASE Operator (Index DRL)

Entered from RELEA5 in UUOCON. If the service routine controls a single unit device (paper tape reader, card punch, etc.), the hardware is released by an action similar to that described in (1) above. If it is capable of controlling multiple units (e.g., magtape), the control unit should not be disturbed as it is likely servicing another job's I/O. The service routine for a directory device (DECtape, disk) should use this entry to write out a fresh copy of the directory if it has changed since it was first read into core. Thus, the releasing action may range from an immediate return (POPJ) to an actual output data transfer with consequent placing of the job in I/O Wait, then returning.

4. CLOSE Operator (Index DCL, DCLO

Entered from UCLS2 or UCLS2B in UUOCON for closing either dump or buffered output. "Basic" devices are never entered when an input close is performed; this occurs only for "long dispatch table" devices at index DCLI.

For buffered output modes, an attempt should be made to output a possible partially filled buffer with a PUSHJ PDP,OUT (this does no harm if there is no more output to be done. Also there is no possibility at this point of there being more than one buffer to flush because the device independent part of CLOSE has taken care of all full buffers). WSYNC may now be called to allow completion of activity if the service routine wants to perform some additional operations in closing the file. If not, a POPJ will return to the CLOSE coding in UUOCON, which does a wait before returning to the user.

Additional operations include end-of-file marking and formatting. Examples: Magnetic tape service writes two end-file marks and backspaces over one of them. Line printer service sends out a carriage return, form feed combination. Paper tape punch service punches about 13 in. of blank tape.

5. OUTPUT Operator (Index DOU)

Entered from OUT2 in UUOCON to start device doing buffered output. The activities of file positioning, formatting, and data transfer all take place at interrupt level. The job of the OUTPUT routine is to condition the interrupt level coding (by setting software switches, counters, etc.) to perform the desired activity and then to prime the hardware control unit so that an interrupt will occur. The OUTPUT routine must also set some indicators so that other sections of the Monitor will know that this device has been made active for OUTPUT.

If desired, the first dispatch (beginning of file) to the output routine may be detected by testing the bit IOBEG in accumulator IOS. This bit is set by an INIT operator and should be cleared by the service routine. For example, detection of this bit causes paper tape punch service to output a fanfold of blank tape before the data of a file. The first output call is also used to get the address of the first buffer from word 1 of the user's ring header and store it in DEVOAD of the device data block. In IOS, the IO bit should be set to 1 (output) and the IOFST bit set to 1 (first item of a buffer).

As part of initialization, the byte pointer used to get data from a user buffer is set up. IOS contains the data mode supplied as part of initial status. When called by PUSHJ PDP, SETBYT, this routine will return in TAC a partial byte pointer containing a size field according to the data mode and "PROG" in the index field. The left half of TAC may now be stored in the pointer location of the device data block. The right half is usually filled in at interrupt level each time a new buffer is begun (detection of IOFST = 1).

When all IOS bits have been set up, the routine SETACT may be called with a

PUSHJ. This coding sets the active bit, IOACT, stores IOS into the device data block and initializes the hung count (HUNGST→HUNGCT) before returning.

The next operation is to start the physical device with a CONO. Simple output devices are started by supplying an interrupt channel address and setting the "DONE" (ready for data transfer) flag. It may also be necessary to supply other conditions to the hardware, but the former are essentials. The CONSO instruction issued at interrupt level to test for expected flags may pick up a mask indirectly to allow the same instruction to test different conditions at different times. If desired (and this is typical), the mask bits should be placed in this location at the time the device is started. A macro STARTDV defined in the file "S" may be used as follows.

> Place the desired CONO bits in the right half of TAC and the CONSO mask bits in the left half. Then write STARTDV XYZ, where XYZ is the device mnemonic (first three letters of service routine title, XYZSER). This macro expands as follows.
>
> ```
> STARTDV  XYZ↑ EXTERNAL PIOFF, PION
> CONO PI, PIOFF
> CONO XYZ, (TAC)
> HRLM TAC, XYZCON
> CONO PI, PION
> ```
>
> Location XYZCON must, of course, be defined within the service routine.

Having started the device, return to UUOCON with a POPJ PDP,.

6. INPUT Operator (Index DIN)

Entered from CALIN in UUOCON to start the device doing buffered input. While the actual data transfers will take place at interrupt level, the job of the INPUT coding is to condition the interrupt coding to perform the desired actions and then to start the device so that an interrupt will occur. The first input call for a file (IOBEG = 1) is used to get the address of the first buffer in the ring from word 1 of the user's ring header and store it in DEVIAD. The desired bits of DEVIOS are manipulated in IOS, then stored with a PUSHJ PDP, SETACT which also turns on the IOACT bit and resets the hung timeout count. The device may then be started using the STARTDV macro as described under (5). When starting an input device, the CONO bits assign a PI channel number and turn on the BUSY flag. The latter sets the physical device in motion to gather the first word or character from the input medium, at completion of which the DONE flag sets, causing the interrupt.

Interrupt Channel Routines CHAN and NULL

The Monitor contains one of these routines for each of the seven priority interrupt levels. A CHAN routine exists for a given level if there is at least one service routine assigned to that level. A NULL routine exists for each unused level. At initialization time, the routine LINKSR in Once Only code places the instruction JSR CHn in each location 40+2n (42, 44, ... 56). The NULL routine defines CHn as a location to contain the PC word and the next instruction attempts to dismiss this spurious interrupt with a JEN @CHn.

A CHAN routine contains a like entry point, but the next location contains a JRST to the interrupt entry of the first service routine built on this PI level. A CHAN routine also contains a subroutine, SAVn (called by a JSR) to save accumulators 0 through 10 and set up a pushdown pointer, and a subroutine RETn to restore accumulators and dismiss the interrupt. The pushdown list and accumulator storage locations are in the body of the CHAN routine.

When a service routine is coded, it is not known what PI level will be assigned at Build time; therefore, there is a standard symbology used to refer to the CHAN entries. If XYZ is the device mnemonic, the following symbols (declared EXTERNAL in the device service routine) will be equated by Build.

```
XYZCHN = PI channel number, 1 through 7
XYZCHL = CHn, interrupt PC word
XYZSAV = SAVn, AC storage subroutine
XYZRET = RETn, AC restore and dismissal
```

There are three ways to exit from an interrupt routine. If the routine has saved and restored all accumulators within its own coding, the dismissal may simply be JEN @XYZCHL. If the initial part of the routine called XYZSAV to save accumulators and set up a pushdown pointer, a JRST XYZRET will cause restoration and dismissal. Alternately, an "extra" POPJ PDP, can be used because the pushdown list is assembled with the address RETn as its 0 entry.

Interrupt Service

The interrupt level coding of a device service routine handles data transfers and error conditions. The routine is responsible for transmission of one byte between a user's buffer and the file, and for advancing buffers when necessary. The routine must stop the hardware device when no buffer is available (device

has caught up with the user or, conversely, take the job out of a Wait if the latter condition is detected upon completion of a buffer (user caught up with device). The flow charts in this section (Figures 11 and 12) describe the general logic used for interrupt level processing. In practice, some alterations in flow and wide variations in coding technique will occur because of differences in device speeds and hardware buffering. We suggest that the reader study the paper tape reader service (PTRSER) and paper tape punch service (PTPSER) routines, which reveal the coding techniques that support the functions outlined in these flow charts.

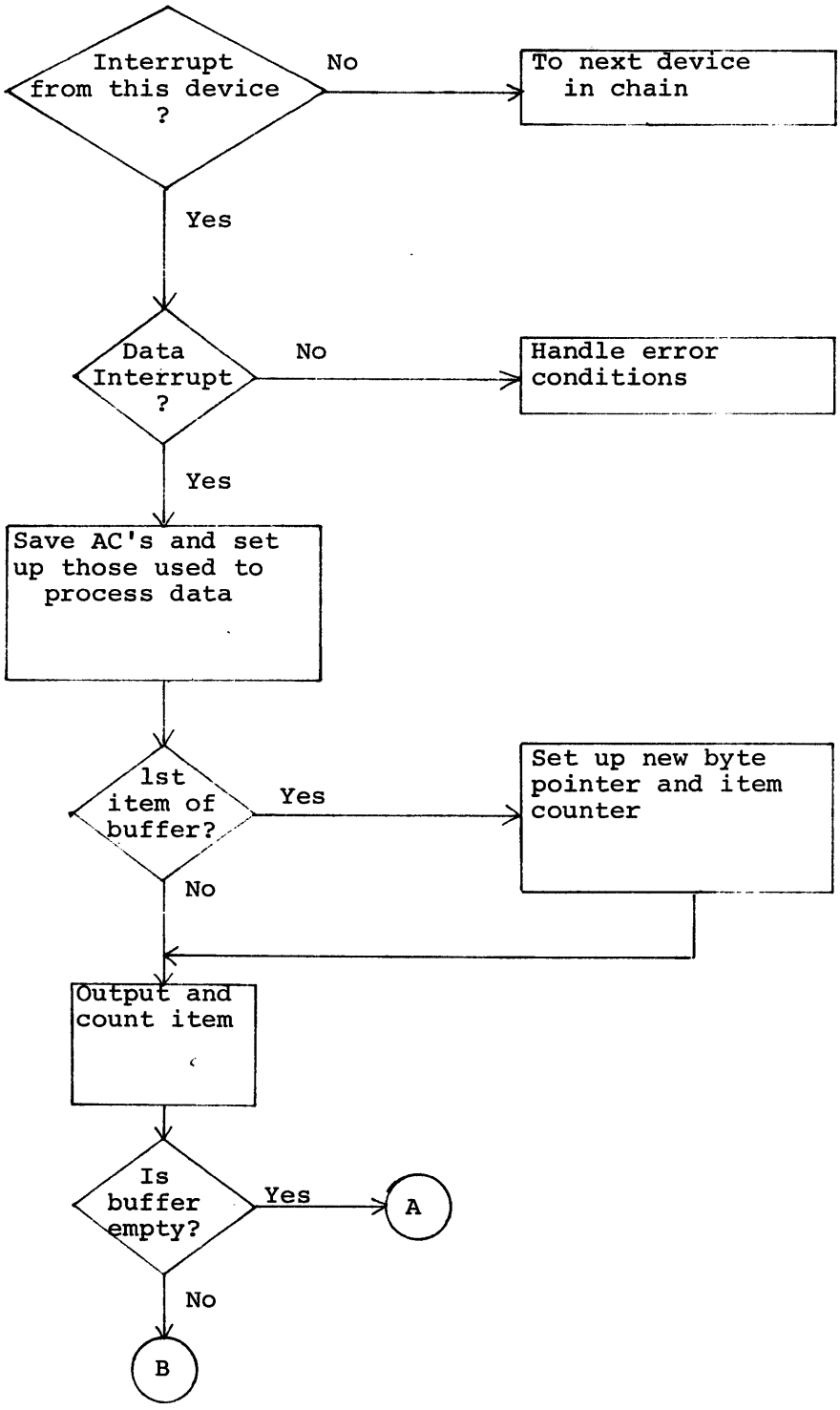


Figure 11. General Flow for Output Interrupt Routine

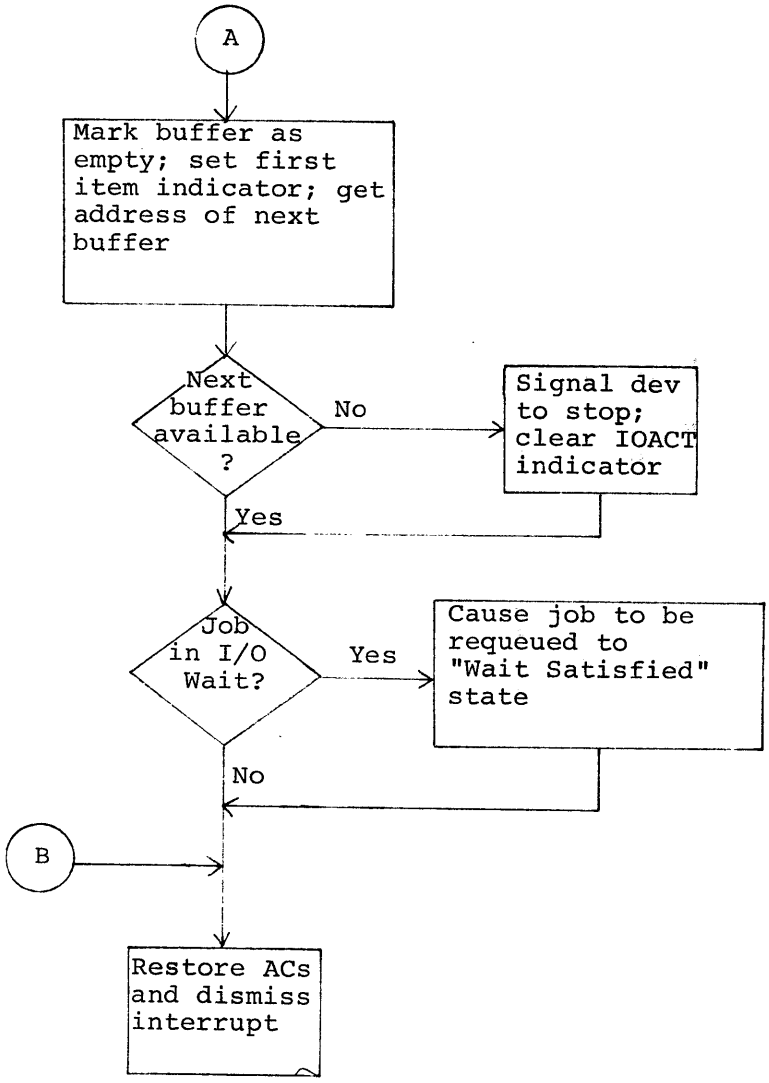


Figure 11 (Cont.) General Flow for Output Interrupt Routine

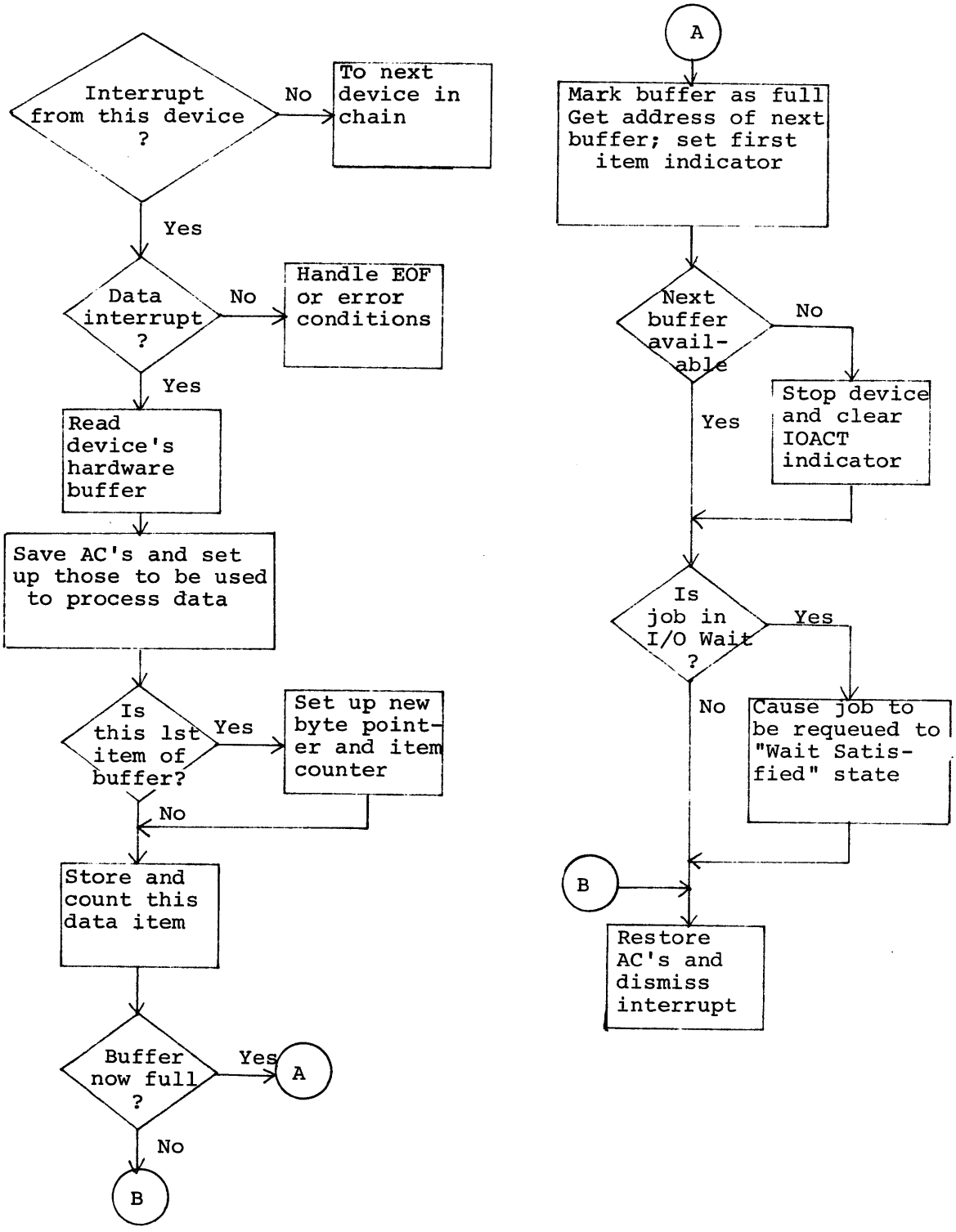


Figure 12. General Flow for Input Interrupt Routine

Table 3

MONITOR UUO'S

Octal	Mnemonic	Description
040	CALL	Extended operation code (see Table 4)
041	INIT (D)	Allocate device with parameter in following words; error return at 3, normal at 4
042 043 044 045 046	Reserved for installation use	
047	CALLI	Immediate mode extended operation code (see Table 4)
050	OPEN (D)	Allocate device; parameter block at E; skip if no error
051 052 053 054	Reserved for future DEC use	
055	RENAME (D)	Change file parameters to block at E; skip if no error
056	IN (D)	Input buffer; use buffer or command list at E ($\neq$0); skip if no error
057	OUT (D)	Output buffer; use buffer or command list at E ($\neq$0); skip if no error
060	SETSTS (D)	Wait for device inactive; load device status word with E
061	STATO (D)	Skip if any device status word bit masked by a 1 in E is a 1
062	GETSTS (D)	Store device status word in E
063	STATZ (D)	Skip if all device status word bits masked by a 1 in E are 0
064	INBUF (D)	Set up a ring of E standard size input buffers
065	OUTBUF (D)	Set up a ring of E standard size output buffers
066	INPUT (D)	Input buffer; use buffer or command list at E if $\neq$ 0

Table 3 (Cont.)

Octal	Mnemonic	Description
067	OUTPUT (D)	Output buffer; use buffer or command string at E if ≠ 0
070	CLOSE (D)	Finish I/O and close file; E = 0 - input and output, 1 - input, 2 - output, 3 - neither
071	RELEAS (D)	CLOSE input and output files and deallocate device
072	MTAPE (D)	Magnetic tape positioning (see below)
073	UGETF (D)	Store number of free DTA blocks in E
074	USETI (D)	Set DTA or DSK to input block E next
075	USETO (D)	Set DTA or DSK to output block E next
076	LOOKUP (D)	Select input file, parameter block at E; skip if no error
077	ENTER (D)	Select output file, parameter block at E; skip if no error

NOTES:

1. I/O is performed by associating a device, a file, and a buffer ring or command list with one of a user's I/O channels (D).

2. MTAPE Commands:

1	Rewind	11	Rewind and unload [1]
2	Write end of record	13	Write 3 in. of blank tape
6	Skip record	16	Skip file
7	Backspace record	17	Backspace file
10	Skip to logical end of tape		

[1] Documented but not implemented (hardware incompatible)

3. (D) - Channel number used (in AC field)

Table 4

CALL [SIXBIT/name/] and CALLI n

Name	n	Description
RESET	0	Terminate user's I/O, user I/O mode; deallocated unASSIGNed dev
DDTIN (AC)	1	Wait for character, load buffer (address in AC) with characters typed since last DDTIN
	2	Not presently used
DDTOUT (AC)	3	Wait until output complete; type characters in buffer (address in AC)
DEVCHR (AC)	4	Load AC with device characteristics of device whose SIXBIT name is in AC
	5 6 7	Not presently used
WAIT (D)	10	Delay running until device inactive
CORE[1] (AC)	11	Change core assigned to number of blocks in AC (0 = no change); skip if granted. AC contains highest address
EXIT	12	RELEASe all I/O devices, type "EXIT, ↑C" on console; console enters Monitor mode
UTPCLR (D)	13	Clear (DTA) directory
DATE (AC)	14	Load 12-bit date in AC (right justified)
LOGIN[2] (AC)	15	Read n words from system file, pointer in AC (-n, TABLE)
APRENB[1] (AC)	16	Enable processor traps to user; AC contains enable bits in CONO APR, format
LOGOUT[1]	17	RELEASe all I/O devices, return job number, core, and devices to Monitor pool; do bookkeeping[3]
SWITCH (AC)	20	Load AC with processor switch register
REASSIgn[1] (AC)	21	Assign device (SIXBIT name in AC+1) to job number in AC; skip if successful
TIMER[1] (AC)	22	Load AC with time of day in jiffies (clock ticks)
MSTIME[1] (AC)	23	Load AC with time of day in milliseconds

Table 4 (Cont.)

Name	n	Description
GETPPN[1] (AC)	24	Load AC_L with proj number, AC_R with prog number of job whose number is in AC
TRPSET[1] (AC)	25	Enter user I/O mode; if AC_L = 40 to 57, put $C(C(AC)_R)$ properly relocated into $C(AC)_R$; skip if no error
TRPJEN[1]	26	Dismiss exec mode interrupt and restore PC from address in .+1
RUNTIMe[1] (AC)	27	Load AC with accumulated run time (ms) of job whose number is in AC (0 = current job)
PJOB[1] (AC)	30	Load AC with job number of current job
SLEEP[1] (AC)	31	Delay running of job for C(AC) seconds.
	≫32	Not used

NOTES:

(AC) - AC used (D) - User's I/O channel number used (in AC field)

[1]Not available in 10/20 or 10/30 single-user Monitors

[2]10/50 Monitor only; available only during LOGIN procedure, not for user

[3]Feature under development; may vary

DATE STORAGE

12-bit field $31\{12(\text{year}-1964)+(\text{month}-1)\}+(\text{day}-1)$

1 Jan 1964 to 4 Jan 1975

FILE PROTECTION BITS

9-bit field

PROT CHG PROT	READ PROT	WRITE PROT	PROT CHG PROT	READ PROT	WRITE PROT	PROT CHG PROT	READ PROT	WRITE PROT
OWNER			PROJECT			OTHERS		

COMMAND LISTS

-n,location-1	Transfer n words starting at location
0,address	Take next command from address
-n,0	Skip n words of data (hardware channel only)
0,0	Stop

Table 5

Cross Reference Listing of I/O Programmed Operator Symbols[1]

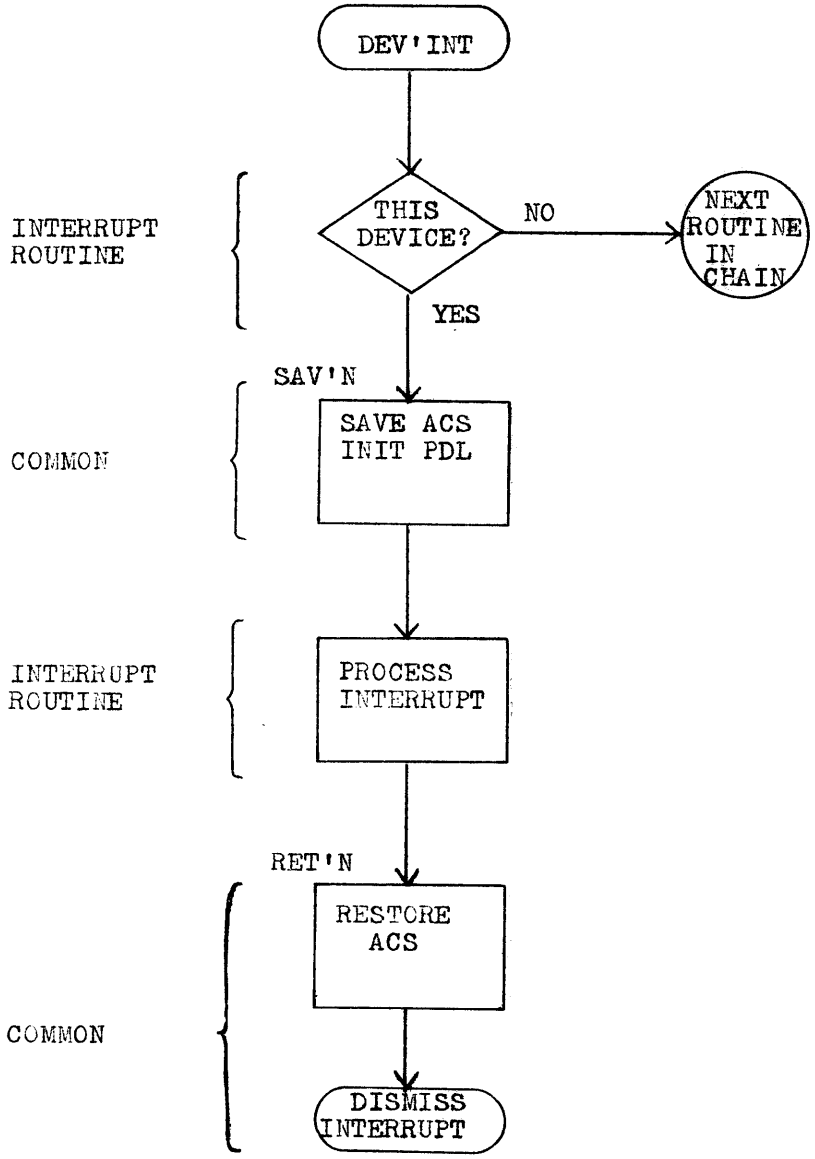
xxxCHL	D33	DIN	D29(t2),D30	IODEND	D27(t1)
xxCHN	D33	DINI	D29(t2)	IODERR	D27(t1)
xxxSAV	D33	DLK	F23,D29(t2)	IODTER	D27(t1)
xxxXIT	D33	DMT	D29(t2)	IOEND	D27(t1)
		DOU	F18,D29(t2),D30	IOFST	D27(t1)
ASSASG	F12	DRL	F21,D29(t2),D30	IOIMPM	D27(t1)
ASSCON	D28(t1)	DRN	D29(t2)	IONRCK	D27(t1)
ASSPRG	D28(t1)	DSI	D29(t2)	IOPAR	D27(t1)
		DSO	D29(t2)	IOSETC	F15,F18
BUFC1	F13	DVAVAL	D28(t1)	IOTEND	D27(t1)
BUFCLC	F13	DVCDR	D27(t1)	IOW	D27(t1)
BUFCLR	F18	DVDIR	D28(t1)	IOWC	D27(t1)
		DVDIRIN	D27(t1)		
CALIN	D6,F14,F15,F16	DVDIS	D28(t1)	JBTADR	D1
CHAN	D32	DVDSK	D27(t1)	JBTSTS	D1
CHn	D32	DVDTA	D28(t1)	JOBFF	D5
CLOSE1	D9,F19,F21,F23	DVIN	D28(t1)	JOBJDA	D1,D2(f1)
CLRDDB	F22	DVLNG	D28(t1)	JOBHCU	D1
		DVLPT	D27(t1)	JOBPFI	D1
DCL	F20,D29(t2),	DVMTA	D28(t1)		
	D30	DVOUT	D28(t1)	LOOKB	D2(f1)
DCLI	F19,D29(t2)	DVPTP	D28(t1)		
DCLO	D29(t2),D30	DVPTR	D28(t1)	NULL	D30
DCLR	D29(t2)	DVTTY	D28(t1)		
DDI	D5,F14,D29(t2)			OBUFB	D2(f1)
DDO	F17,D29(t2)	ENTRB	D2(f1)	OCLOSB	D2(f1)
DEN	F23,D29(t2)			OUT	D8,F17,F20
DEVADR	D4(f2),D25,	IBUFB	D2(f1)	OUT2	D8,F18
	D26(f10)	ICLOSB	D2(f1)	OUTA	D9,F17,F18
DEVBUF	D5,D25,D26(f10)	IN	D6,F14	OUTBFB	D2(f1)
DEVCHR	D5,D24,D26(f10)	IN1	D6,D7,F14	OUTDMP	D8,F17
DEVCTR	D4(f2),D25,D26	INBFB	D2(f1)	OUTF	D9,F18
	(f10)	INDMP	F14	OUTF1	D9,F18
DEVDAT	D6,D24	INEOF	D7,F15	OUTPB	D2(f1)
DEVEXT	D25,D26(f10)	INEOFE	D8,F15	OUTS	D9,F18
DEVFIL	D25,D26(f10)	INITB	D2(f1)		
DEVIAD	D5,D6,D25,	INPB	D2(f1)	PRJPRG	D1
	D26(f10)	INPT0A	D6,D7,F15		
DEVIOS	D5,D24,D26(f10),	INPT0C	D7,F15	RELEA0	D10,F12,F21
	D27(t1)	INPT1	D7,F15	RELEA1	D10,F21
DEVLOG	D25,D26(f10)	INPT2	D6,D7,F15	RELEA2	D10,F21
DEVMOD	D25,D26(f10)	INPUT2	D6,F15	RELEA3	D10,F21
DEVNAM	D24,D26(f10)	INPUT3	F16	RELEA4	D11,F21
DEVOAD	D5,D25,D26(f10)	INPUTF	D6,F16	RELE4A	D11,F21
DEVPTR	D4(f2),D25,D26	IO	D27(t1)	RELEA5	D11,F21
	(f10)	IOACT	D27(t1)	RELEA6	D11,F22
DEVSER	D1,D24,D26(f10)	IOBEG	D27(t1)	RELEA7	D11,F22
DEVSRC	F12	IOBKTL	D27(t1)	RELEA9	D11,F22
DGF	D29(t2)	IOBOT	D27(t1)		
DHNG	D29(t2),D30	IOCON	D27(t1)	SYSDEV	D2(f1)

[1] A D preceding a page number indicates that a description of the item is found on that page; an F indicates that the item appears in a flow chart on that page. (tn) = Table (fn) = Figure

Table 5 (Cont.)

TTYATC	D27(t1)	UCLSBI	D9,F19	UINITC	F12
TTYBIU	D27(t1)	UCLSBO	D10,F20	UOBFI	F13
TTYUSE	D27(t1)	UDEN	F23	UOUT	D8,F17
		UDLK	F23	UOUTBF	F13,F18
UCLS0	D10,F19	UDLKC	F23	USRJDA	D1,D2(f1)
UCLS1	D10,F20	UERROR	F15		
UCLS2	D10,F20	UINBF	F13,F16	WAIT1	F14,F17,F18,
UCLS2A	D10,F20	UINIT	F12		F19,F20,F21,
UCLS2B	D10,F20	UINITA	F12		F23
UCLS3	D10,F20	UINITB	F12	WSYNC	F14,F17,F18

DEVICE INTERRUPT ROUTINE



Monitor Handout 9-Apr 70

INTERRUPT ROUTINE CHAIN

```
40 + 2N:        JSR     CH'N
                JSP     DAT,ERROR

CH'N:           ø
                JRST    DEV1'INT

DEV1'INT:       CONSO DEV1, Conditions
                JRST  DEV2'INT

                Process DEV1 Interrupt

DEV2'INT:       CONSO DEV2,Conditions
                JRST DEV3'INT
                Process DEV2 Interrupt

DEV3'INT:       CONSO DEV3,Conditions
                JEN   @CH'N
                Process DEV3 Interrupt
```

CHANNEL SAVE ROUTINE

The symbol "x" represents any channel number, 1 - 7.

```
CH'x:   Ø               ;PC STORED HERE BY JSR AT 4Ø+2x
        JEN  @CH'x      ;END OF INTERRUPT CHAIN
                        ;THIS INST WILL BE REPLACED BY A JRST TO THE FIRST
                        ;INTERRUPT ROUTINE ASSIGNED TO THIS CHANNEL

SAV'x:  Ø               ;ROUTINE TO SAVE AC'S FOR INTERRUPT ROUTINE
        MOVEM HIGHAC,SAVAC'x+HIGHAC     ;"HIGHAC" IS DEFINED AS THE HIGHEST
                                        ;AC TO SAVE FOR THIS CHANNEL
        MOVEI HIGHAC,SAVAC'x
        BLT   HIGHAC,SAVAC'x+HIGHAC -1  ;SAVE AC'S IN AREA RESERVED BELOW
        MOVE  PDP,SAVAC'x+HIGHAC + 1    ;INITIALIZE PUSHDOWN POINTER FOR
                                        ;PDL BELOW
        JRST  @SAV'x    ;RETURN TO CALLING ROUTINE
                        ;CONTROL IS PASSED TO RET'x IF THE INTERRUPT EXITS
                        ;WITH A POPJ
RET'x:  MOVSI  HIGHAC,SAVAC'x       ;ROUTINE TO RESTORE AC'S SAVED BY
                                    ;THE ABOVE ROUTINE
        BLT    HIGHAC,HIGHAC
        JEN    @CH'x    ;DISMISS THIS INTERRUPT AND RETURN TO INTERRUPTED
                        ;ROUTINE

SAVAC'x: BLOCK  HIGHAC  1           ;SPACE TO SAVE AC'S Ø - HIGHAC
CH'x'PDP: XWD   -PDL+1, .+1         ;INITIAL PUSHDOWN POINTER
CH'x'PDP1: EXP  RET'x               ;FIRST ENTRY ON PDL -- RETURN ADR FOR
                                    ;LAST POPJ IN INTERRUPT ROUTINE

          BLOCK  PDL-1              ;SPACE FOR REMAINDER OF PDL
```

QUESTIONS ON DEVICE SERVICE ROUTINES

1. What is the basic function of the OUTPUT routine in a device service routine?

2. In general, where is an output device turned off if it has emptied the last buffer supplied by the user? Where is the specific instruction for the paper tape punch?

3. What is the INTTAB entry for the paper tape reader? -- the software clock interrupt?

4. Which instruction sets up the interrupt locations, 4Ø +2N?

5. If an interrupt occurs, but no device assigned to the channel admits causing it, how is the interrupt dismissed?

6. Which interrupt assignment macros generate (directly) INTTAB entries? -- device save routine definitions?

7. Which instruction results in CHKCHL being defined? How is it defined?

8. What would be the result of defining UNIQ3 = 1?

9. How could you ensure that a special device is assigned as the only device on a specific channel, without making any changes to COMMON.MAC?

10. List the actions taken by PTPINT if a buffer has been finished and the next buffer is not available to it.

9. Allocation of Core

Readings

Handout "Allocation of Core"

Table Descriptions

CORTAB
JBTADR

Flow Chart

Handout 24 Core Allocation Flow

Monitor Listings

COREl	Routine COREl	lines 334-587
COREl	Routine COREØ	lines 248-333
COREl	Routine CHKSHF	lines 129-186

Written Assignment

Questions on Allocation of Core

ALLOCATION OF CORE

In a swapping system, user jobs are allocated memory space of two types, virtual core -- or swapping space -- and physical core. Both virtual core and physical core are allocated in 1K blocks. Every job with a program to run must have virtual core, but physical core is assigned and deassigned as jobs are swapped in and swapped out. Core allocations are made initially by the RUN and GET Commands. A job's core allocation may be changed by the CORE Command and the CORE UUO. It is finally relinquished by the KJOB Command. Physical core assignments, but not virtual core allocations, are changed by the Swapper and Shuffler Routines.

Normally, core assignments are changed only for jobs that are in core. Obviously a job must be in core to execute a CORE UUO. And the Core Command will be delayed until the job is in core. Therefore, any change in core allocation will require both a change of virtual core allocation and a change of physical core assignment. However, the KJOB, RUN and GET commands may be executed with the job swapped out. These commands will initially request a change of core allocation to the minimum allocation, and therefore will never require an increase in size for a swapped out job. In these cases, there is no physical core assignment to be changed. The total of available virtual core is imcremented by the decrease in the job's size and the job's In-Core Image Size is reduced to the new value. The job's physical core assignment is made according to its In-Core Image Size when it is swapped in.

When a job requests a change in core size, two conditions must be met. The change must not cause the total virtual core assigned to all jobs in the system to exceed the amount of swapping space, and the size of this job must not exceed the maximum job size specified when the system was generated. If both these conditions are met, the job will be allocated the requested amount of virtual core. Virtual core is allocated to a job by subtracting the amount of increase from the total of available virtual core, and giving a normal return to the routine which made the request.

When a job is allocated virtual core, it may or may not be assigned physical core. Physical core is assigned to a job by setting bits in a core map table, subtracting that amount from the total of free physical core, and updating the address of the job at all places where it is known. If a job is to be given a new physical core assignment, any old assignment is first deassigned. In order to make a new assignment of physical core, the monitor must find a group of

contiguous unassigned blocks, or hole, large enough to hold it. Physical core is assigned starting at the beginning of the first hole large enough to hold the job. The job is moved to the new area, and all additional core assigned to the job is cleared.

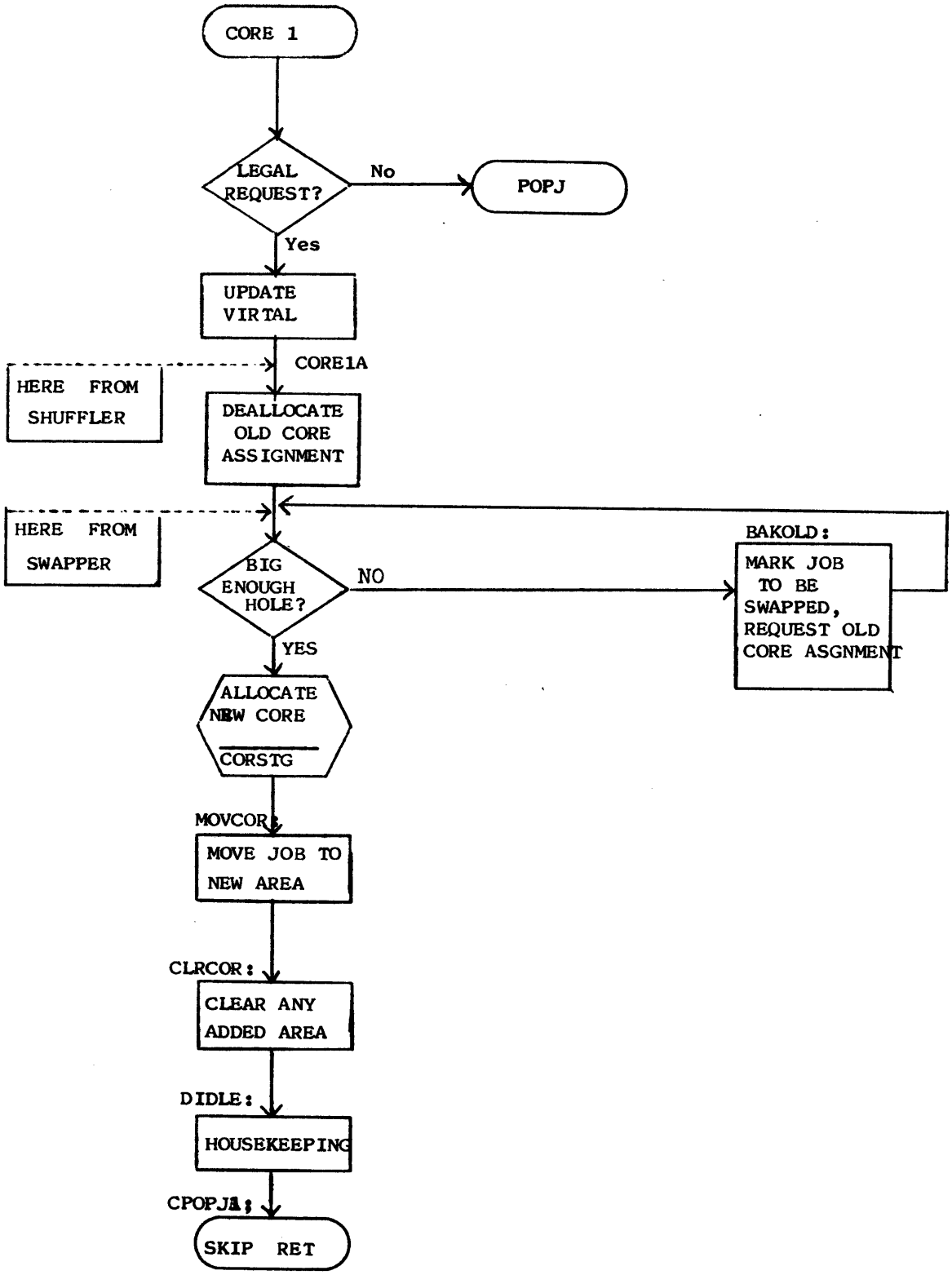
If there is no hole large enough to hold the job, it will have to be swapped out. A bit is set to inform the Swapper that this job must be swapped out, and its In-Core Image Size is set to its new size. It is assigned physical core of the same amount it originally had, to hold it until it can be swapped out. Once swapped out, it will be considered for being swapped in according to the normal swapping algorithm. When its turn comes to be swapped in, physical core will be assigned to it according to its In-Core Image Size.

The Swapper calls the core allocation routine to assign physical core for a job which is about to be swapped in. The Swapper assures that there is a big enough hole for the job before requesting the assignment. It skips over the allocation of virtual core, because a swapped out job already has virtual core allocated.

The Swapper also calls, under some conditions, a routine known as the Shuffler. The function of the Shuffler is to move the job following the first hole up into the hole. Successive calls to the Shuffler tend to pack the jobs toward the beginning of core, and to combine holes as they are moved toward the end of core. The Shuffler must assure that the job to be shuffled does not have active I/O. Then it simply requests a physical core assignment of the same size that the job already has. The core which was previously assigned to the job is deassigned, increasing the size of the first hole to be more than large enough for the request. Since the new core assignment is made at the beginning of the first sufficiently large hole, it will always be at the desired location.

The job is moved into its new area by the core allocation routine just as it would be for any change of physical core assignment. The Shuffler, like the Swapper, skips over the allocation of virtual core because the job's virtual core allocation is not changed.

CORE ALLOCATION ROUTINE



Monitor Handout 24 -- July 70

QUESTIONS ON ALLOCATION OF CORE

1. When a new area of core is allocated to a job, how do the CORTAB bits corresponding to that area get set?

2. For what reasons will CORE1 take the error (nonskip) return?

3. Why would the JXPN bit be set for a job? What instruction actually sets it, and how does control get to that point?

4. When a job's area is increased, which instruction clears the additional area given to it? Why is this necessary?

5. When is it necessary to reset the hardware relocation and protection registers upon changing a segment's core allocation?

6. Where is JBTADR updated after a segment has been moved?

7. How is it determined whether the segment just moved was stopped in User Mode or Exec Mode? What difference does it make?

8. What determines the number of words moved when a segment is moved to a new core area?

10. The Scheduler

Readings

Handout "The Scheduler"

Table Descriptions

JBTSTS	Job Status Table
JBTQ	Job Queues Table
JBTSWP	Job Swap Table
AVALTB	Available Resources Table
QBITS	Requeuing Table
AVLQTB	Requeuing Table

Scan Table

Transfer Table

Queue Progression Table

Flow Chart

Handout 3 Scheduler Macro Flow

This flow chart does not include the case of the current job being unrunnable.

Other References

Handout 1 Items Referenced by the Scheduler

Monitor Listings

SCHED Routine NXTJOB lines 49-208

Written Assignment

Questions on Scheduling

THE SCHEDULER

The basic function of the Scheduler is to select the job to run during the next time slice. It also keeps the job queues up to date and does some additional housekeeping. Control passes to the Scheduler from the Clock Cycle Routine, CLOCK1. When the Scheduler has finished its functions, it returns control to the Clock Cycle. Context switching is then performed. The job which the Scheduler selected to run is set up, and control is given to it for the remainder of this time slice.

Rescheduling is required for one of two reasons. Either the clock interrupt has occurred -- ending the time slice -- or the current job has reached a point at which it can not immediately continue. Whenever a monitor routine finds that it can not immediately complete a function requested by the current user job, it will return control to the Clock Cycle so that another job may be selected to run. The job which was stopped will be rescheduled at some later time when the function which it requested can be completed.

The functions performed by the Scheduler are as follows:

1. If the clock has ticked, decrement all positive In-Core Protect Times.
2. Check if the current job is still runnable. If not, requeue it according to its Job Status Table entry, and immediately select another job to run.
3. If the current job is still runnable, and if the clock has ticked, decrement the current job's quantum run time. If it has reached zero, requeue the current job into another Processor Queue.
4. Check if other jobs need requeuing. If so, requeue each one according to its Job Status Table entry.
5. Check if any sharable resources have become available. If so, requeue a job from the corresponding sharable resource wait queue.
6. Call the Swapper. The Swapper will return control to the next instruction.
7. Scan the Processor Queues for a job to run. If a runnable job is found, return control to the Clock Cycle with the job number in AC ITEM. If no runnable job is found, return control with zero in AC ITEM, for the Null Job.

All queue transfers are done by the Scheduler. When other routines determine that a job should be requeued, they set the appropriate bits in that job's entry in the Job Status Table.

The JRQ bit is set to indicate that the job should be requeued, and the Wait

State Code is set to indicate the queue to which the job should be moved. The JRQ bit is never set, however, for the current job. In that case, the non-zero wait state will show that the job is unrunnable, and must be requeued.

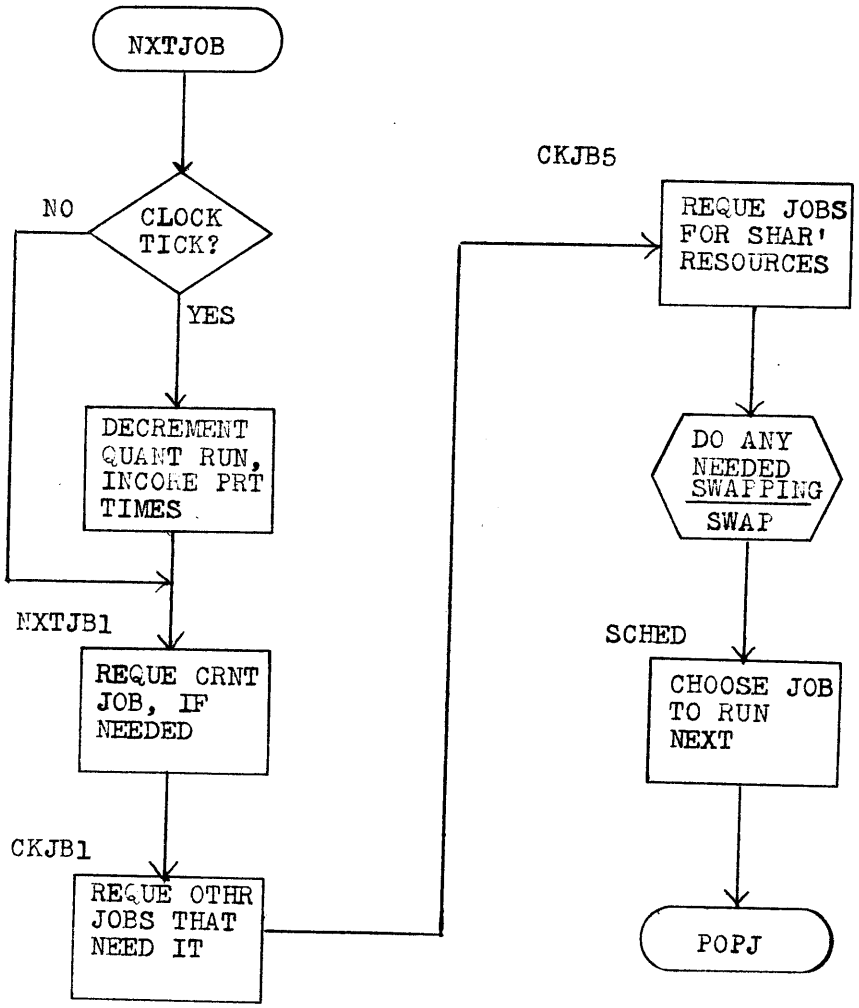
All jobs which are not in a wait state are kept in the processor queues -- PQ1, PQ2, and PQ3. The fact that a job is in a processor queue does not, however ensure that it is runnable. The job still might be swapped out, might need to be swapped out for expansion, or might be selected to be shuffled. A job which stays in a processor queue until its quantum run time expires is requeued to the end of another processor queue, according to the following plan:

PQ1 ⟶ PQ2
PQ2 ⟶ PQ3
PQ3 ⟶ PQ2

A job can be removed from a processor queue for a number of reasons. If it executes a UUO requesting a buffer which is not yet available to it, it will be put into the IO Wait Queue. When the interrupt routine makes the buffer available to the job, it will mark the job to be requeued. The job will then be requeued to the front of PQ1, according to the present entry in the QBITS table.

When a job executes a UUO which requires use of a sharable resource, and that resource is not available, the job must be requeued into the corresponding sharable resource wait queue. A job is taken out of a sharable resource wait queue according to the corresponding entry in AVLQTB. These entries all call for the job to be requeued to the beginning of PQ1.

SCHEDULER -- MACRO FLOW



Monitor Handout 3 - Apr 70

ITEMS REFERENCED BY THE SCHEDULER

Label	Defined in	Contents
JOB	COMMON	Current job number
JOBQUE	SCHED	Queue number of current job
QJOB	SCHED	Number of jobs needing requeuing
TIMEF	COMMON	Nonzero if clock has ticked
HIGHJB	COMMON	Highest job number currently assigned
RUNMSK	COMMON	Defines bits which do not affect runability of the job
RUNABLE	COMMON	Defines bit values required for job to be runnable
POTLST	COMMON	Flag that Null Job is to be run even though there are jobs in the processor queues

TABLES

AVALTB	SCHED	Sharable resources available
JBTPRV	COMMON	Job privilege table
JBTQ	COMMON	Job queues table
JBTSTS	COMMON	Job Status table
JBTSWP	COMMON	Job swap table
QBITS	SCHED	Table of Transfer Tables for wait states
QQSTAB	SCHED	Quantum run times
QSTAB	SCHED	New queue for job size
QSTOP	SCHED	Transfer Table - to stop queue
QTIME	SCHED	Transfer Table - time expired
QTTAB	SCHED	New queue for old queue
QXXS	SCHED	Transfer Table - out of sharable resource wait XX specifies which resource
QXXW	SCHED	Transfer Table - new queue depends on wait state code
SSCAN	SCHED	Scan table - job to run next

QUESTIONS ON SCHEDULING

1. Why might a job be unrunnable even though in one of the processor queues? List line numbers to support your answer.

2. Where does the monitor give control to the user program?

3. What is the function of the quantum run time?

4. What determines the order in which the scheduler considers jobs for the possibility of running next?

5. What is the meaning of a non zero entry in AVALTB?

6. When is a clock tick considered "lost" rather than simply idle?

7. If there are several jobs waiting for a sharable device which has become available, which job gets it?

8. Where is the Null Job terminated before a clock tick?

9. How does the Scheduler determine the transfer table to use to requeue a job whose JRQ bit is set?

10. Which transfer table would be used to requeue the job in each of the following situations?

Current Job	RUN Bit	JRQ Bit	CMWP Bit	Wait State Code	Transfer Table
yes	1	0	Ø	IOWQ-12	
yes	0	0	0	RNQ-Ø	
no	1	1	0	WSQ-1	
no	0	1	1	STOPQ-16	
no	0	1	0	RNQ-Ø	

11. The Swapper

Readings

Handout "The Swapper"

Table Descriptions

CORTAB
Scan Table
JBTADR
JBTSGN
JBTSTS
JBTSWP

Flow Charts

Handout 4 Swapper Logical Flow

This flow chart shows the flow of the Swapper as a continuous process, rather than according to the code.

Handout 5 Swapper-Simplified Macro Flow

This flow chart follows the code, but is "simplified" by not considering high segments. Several sections will be repeated if there is a high segment to swap

Other References

Handout 2 Items Referenced by the Swapper

Monitor Listings

SCHED Routine SWAP lines 824-1192

Written Assignment

Questions on the Swapper

THE SWAPPER

The basic job of the swapper is to keep in core the jobs which are most likely to be runnable. A job is swapped out only when necessary - because a more eligible job needs to be swapped in, or because it wants to expand its core size and there is not enough room.

The swapper checks periodically if there is a job which should be swapped in. If there is no job to be swapped in, it checks for expanding jobs. If no job is expanding, it has nothing more to do.

When a job requests more core and there is no hole large enough for its new size, it must be swapped out, then back in with the new core allocation. The swapper handles this just as it does its normal swap-out described below.

The swapper determines whether or not it needs to swap in a job by scanning the job queues in a prescribed manner. It uses the QSCAN routine for this and the scan table specified is ISCAN. The order in which it considers jobs for swapping in is as follows:

1. Command Wait Queue
2. Processor Queue 1
3. Processor Queue 2
4. Monitor Disk Queue - first entry only
5. Processor Queue 3
6. Other sharable resource queues - first entry in each

If a swapped out job is found, the swapper then attempts to fit the job into core. First it computes the total amount of core necessary to swap the job in. There are three possible cases:

1. Size of this job's low segment plus size of this job's high segment.
2. Size of low segment only, because either there is no high segment or it is already in core.
3. Size of high segment only, because low segment already swapped in.

If there is enough core available, the swapper will shuffle jobs toward the beginning of core until there is enough space in one place to bring in the segment which it needs to read in. If shuffling does not generate enough space in a single hole, unused high segments in core will be deleted, and shuffling will continue. This process must create a sufficiently large hole, or it

would have originally found that there was not enough core available.

If there are two segments, the low segment is swapped in first. The swapper ensures that there is enough space available for both segments before doing anything else. Then it shuffles core to create a hole for one segment at a time to be read in. When a large enough space is available for the segment which is to be swapped in next, the I/O request is set up and given to the disk service routine. If the swapper finds that there is not enough space available to swap in the job it has chosen, it will attempt to swap a job out. It scans the job queues for a job to swap out according to the scan table OSCAN. The order specified is:

1. Stop Queue - forward
2. Sleep Queue - forward
3. Sharable Resource Queues - backward except first entry
4. TTY I/O Wait - forward
5. Processor Queue 3 - backward
6. Sharable Resource Queues - first entries
7. Processor Queue 2 - backward
8. Processor Queue 1 - backward
9. Command Wait Queue - backward

The swapper continues the scan, accumulating the core space that could be gained by swapping out each swappable job until:

1. Enough core is found that job could be swapped in if those jobs were swapped out, or
2. The job number of the job to be swapped in is returned by the scan. This means the job can not be swapped in without swapping out jobs with higher priority for being in.

During this scan, a job is considered unswappable if any of the following conditions are met:

1. RUN or CMWB set, and job still has in core protect time
2. NSWP set - job cannot be swapped out
3. SWP set, job is already swapped out.

If enough core can be made available by swapping jobs out, the swapper picks the largest job found in the scan and proceeds to swap it out.

If the job has a high segment to be swapped out, this will be done first, so that I/O may continue in the low segment while the high segment is being

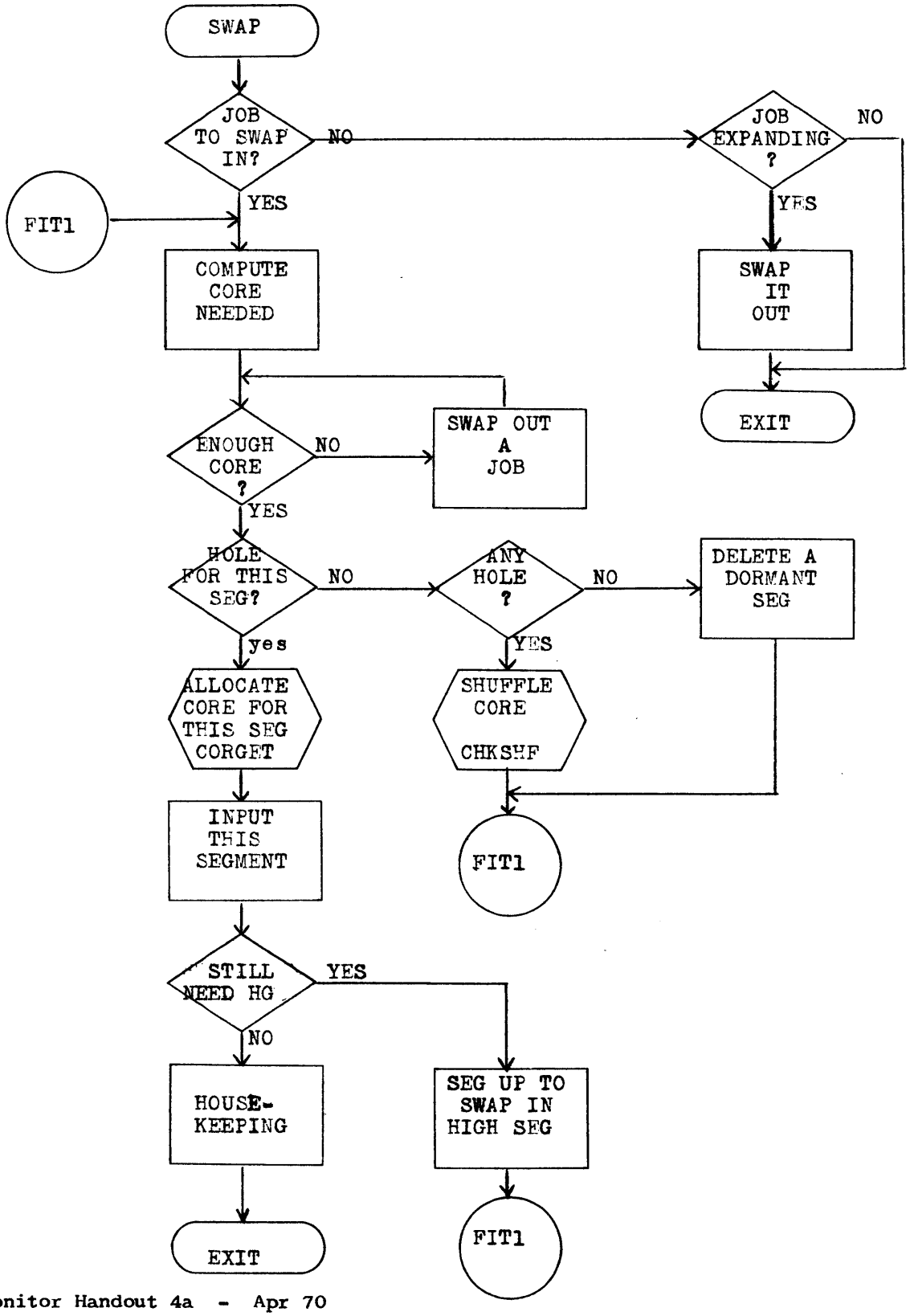
swapped. This will occur only if all the following conditions are met:

1. The job has a high segment.
2. High segment is in core.
3. This job is the only job using that high segment - i.e., in core count = 1
4. This high segment is not already on disk.
5. This high segment is not used by the job being fitted into core.

When swap out has been completed, the swapper starts from the beginning of the process of fitting the new job into core. The entire process is repeated until there is finally enough core available for the job to be swapped in. Then the swapper proceeds to swap in the job that it originally selected according to ISCAN. Note that the output scan is repeated each time another job must be swapped out. The priorities could very well change while the previous job is being swapped out. However, the input scan is not repeated. Assuming enough core can be made available for it, the job originally selected to be swapped in will be the next job swapped in.

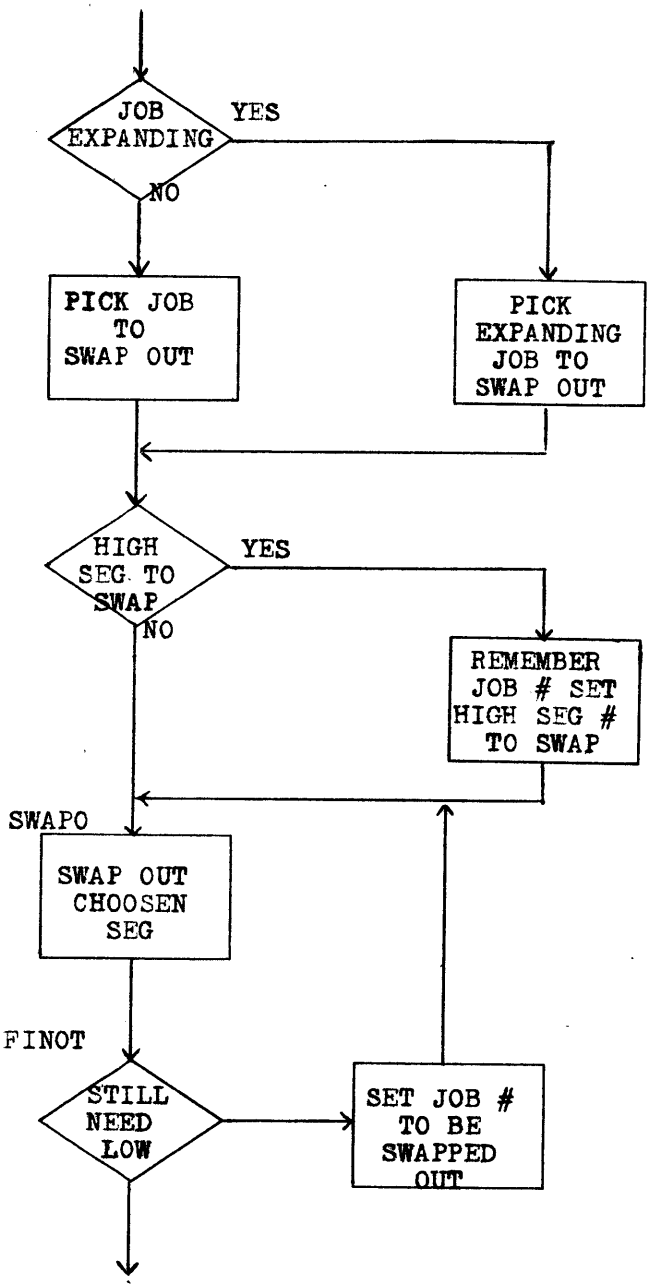
Although the actions above are described as one continuous process, and can logically be thought of as such, they may actually be spread out over several calls to the swapper on several different clock ticks. Any time the swapper reaches a point at which it cannot immediately continue, it exits for that clock tick. On the next clock tick it will try to continue the process that it left off on the last clock tick. It uses a number of memory flags to indicate what needs to be done next. These flags are tested at the beginning of the routine, and control is returned to the place to continue the action it was doing most recently.

SWAPPER LOGICAL FLOW

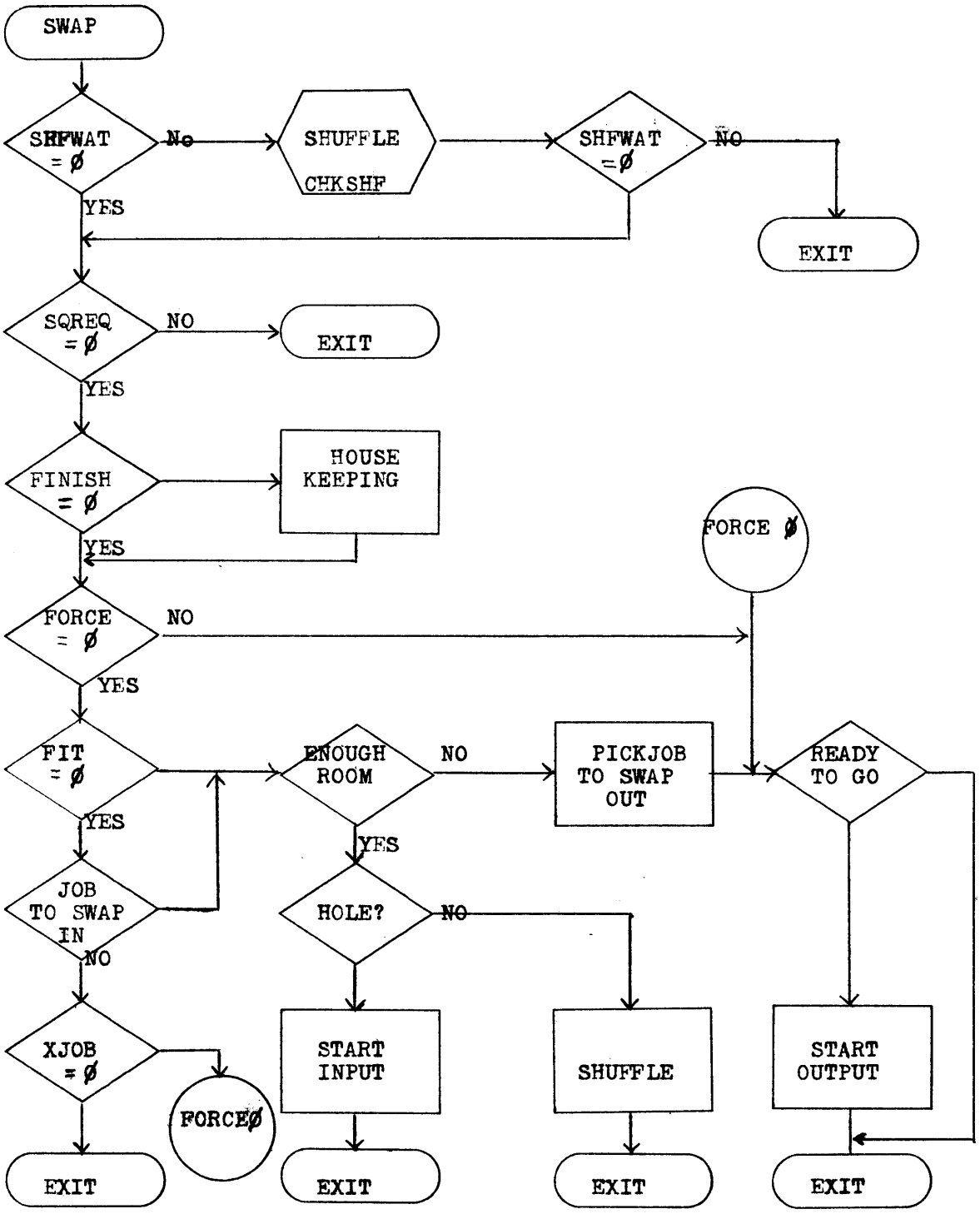


Monitor Handout 4a - Apr 70

SWAP OUT


JOB EXPANDING
YES
NO
PICK JOB TO SWAP OUT
PICK EXPANDING JOB TO SWAP OUT
HIGH SEG TO SWAP
YES
NO
REMEMBER JOB # SET HIGH SEG # TO SWAP
SWAPO
SWAP OUT CHOOSEN SEG
FINOT
STILL NEED LOW
SET JOB # TO BE SWAPPED OUT

SWAPPER
- SIMPLIFIED MACRO FLOW -



ITEMS REFERENCED BY THE SWAPPER

Label	Defined In	Contents
BIGHOL	COMMON	Size of largest hole in core
CORLST	COMMON	Pointer to last "existent" bit in CORTAB
CORTAL	COMMON	Total available core
FINISH	COMMON	Number of job with swapping transfer in progrsss
FIT	COMMON	Number of job selected to be swapped in
FORCE	COMMON	Number of job selected to be swapped out
HIGHJB	COMMON	Highest job number currently assigned
HOLEF	COMMON	Location of job following first hole in core
IMGIN	COMMON	Pointer to In-Core Image size in JBTSWP Table
IMGOUT	COMMON	Pointer to Out-Core Image size in JBTSWP Table
JOBMAX	COMMON	Highest job number times 2^9
SEGPTR	COMMON	XWD - (number high seg's), first high segment number
SHFWAT	COMMON	Number of job selected to be shuffled
SQREQ	SCHED	IO Word for swapping xfer; Ø if none in progress
SWPIN	COMMON	Low seg just swapped in (saved by SEGCON)
VIRTAL	COMMON	Amount of free swapping space
XJOB	SCHED	Number of jobs trying to expand

TABLES

CORTAB	COMMON	In-use core table
ISCAN	SCHED	Scan table for job to swap in
JBTADR	COMMON	Job address table
JBTSGN	COMMON	Job high segment table
JBTSTS	COMMON	Job status table
JBTSWP	COMMON	Job swap table
OSCAN	SCHED	Scan table for job to output

Questions on the Swapper

1. ıder what conditions will a job be swapped out?

2. In what order does the swapper consider jobs for possibly swapping in? What determines this order?

3. Under what conditions will the shuffler be called?

4. What conditions must be met before a high segment will be swapped out?

5. Why can the swapper be sure of a successful return from CORGET when it tries to assign core for a job to bring in?

6. If both segments of a two segment job are to be swapped out, which segment goes first? Why?

7. If two segments must be swapped in, which segment is swapped in first?

8. What determines the order in which jobs are considered for swapping out? What is the order in which they are considered?

9. What happens if the swapper can't find enough room to swap in the job it has chosen?

10. Why could CORTAL be greater than Ø, but no hole be indicated on CORTAB?

11. Suppose the swapper has selected a job to swap in, and has been making room for it by swapping out jobs over a number of clock ticks. If, when there is enough room a higher priority job has now become eligible to swap in, which job will actually be swapped in? What justification do you see for this?

12. When a job is shuffled, which instruction (Program and line) actually moves it to the new area?

13. Why would a job be rejected by the swapper when it is looking for jobs to swap out?

14. Under what conditions will a high segment be swapped out? in?

15. List as many ways as you can that the swapping algorithm can be changed, or "tuned", without changing any significant number of instructions.

12. Scanner Service

Readings

Handout "The Scanner Service"

Table Descriptions

TTY Device Data Block
TTYTAB
LINTAB
SPCTAB

Diagrams

Handout 14 - DC1Ø Data Line Scanner
Handout 20 - Scanner Instructions

Flow Charts

Handout 47, a - d Scanner Service Flow Charts

Monitor Listings

DLSINT	Routine SCNINT	lines 94 - 108
SCNSRF	Routine XMTINT	lines 1703 - 1777
	Routine RECINT	lines 1568 - 1641
	Routine TTEDIT	lines 1800 - 1830
	Routine ADJHP	lines 640 - 650
	Routine SPCHEK	lines 389 - 446
	Routine TTYIN	lines 831 - 903
	Routine TTYOUT	lines 904 - 973

Written Assignment

Questions on Scanner Service

THE SCANNER SERVICE

All device dependent functions for Teletype I/O are performed by the scanner service. The scanner service actually consists of two routines, a small routine which depends on the particular scanner being used, and a large scanner independent routine called, for full duplex service, SCNSRF.

The scanner dependent routine gives the immediate response to TTY interrupts, a DATAI to the scanner. This instruction allows the routine to determine the specific TTY which caused the interrupt, and whether it was an input interrupt or an output interrupt. On input interrupts it also supplies the character. Control is then passed to the routine in SCNSRF which handles that type of interrupt, XMTINT for output interrupts, or RECINT for input interrupts.

Input and output to a Teletype may be done at any time, because each active TTY has an input buffer and an output buffer in the monitor. The input interrupt routine places the character read in from the scanner into the input buffer, corresponding to the line which caused the interrupt. The output routine takes the next character from the line's output buffer and sends it to that TTY. Because the buffers are in the monitor, these operations can be performed even whi the corresponding job is swapped out.

On input, or receiver interrupts, control goes to RECINT. The typed character and the TTY line number are already in core as a result of the DATAI performed by the dependent routine.

If the TTY causing the interrupt was previously inactive, a DDB must be set up for it. If all the DDB's are taken, input cannot be accepted, and an "X" is echoed to inform the user.

If the TTY has a DDB, the edit routine, TTEDIT, is called to perform most of the manipulations for the received character. The edit routine calls ADJHP, which adjusts the horizontal position counter, and passes control on to SPCHEK. SPCHEK picks up the SPCTAB entry for the received character or else a Ø, and returns to TTEDIT. Here some housekeeping is done, and the special action routine for this character is called if SPCTAB specifies one. The character is put into the input buffer and, unless echo suppressed, into the output buffer. TTEDIT then returns control to RECINT.

There is next a check for the output buffer being almost full. If so, the XOFF bits are set to tell the transmit interrupt routine to send an XOFF character on

the next interrupt. If input is coming from a paper tape reader on the TTY, rather than the keyboard, this will stop the reader. If the character received is not a break character, processing is now finished. The Type Out In Progress bit, TOIP, is checked and, if it is not set, the transmit interrupt routine is called to start typeout for the echo character.

If it is a break character, some additional housekeeping must be done. If this is the first break character awaiting processing, COMSET is called. If the TTY is in monitor command mode, the command completed bit is set and COMCNT is incremented by COMSET.

If the job using this TTY is in TTY Input Wait, it will be requeued as a result of the break character being received. STTIOD is called to mark the job as needing to be requeued for TTY Wait Satisfied. Finally the same exit procedures are followed as were for a non-break character.

On each output - or transmit - interrupt, control passes to XMTINT. If the output buffer is empty, the TTY printer is "turned off" simply by not sending another character to it. The Type Out In Progress bit is cleared, indicating that this TTY is ready to accept a character at any time and will not cause an interrupt. Routines which place characters into the output buffer must check this bit, and when it is clear, send the first character out to the TTY without an interrupt.

If there are characters in the output buffer, the next one is picked up by the GETCHR routine. If the job using this TTY is in output wait, there is a check for the monitor output buffer being almost empty. If it is down to the last eight characters, the STTIOD routine is called to mark this job as needing to be requeued for TTY Wait Satisfied. Control is returned to the scanner dependent routine to set up and execute the DATAO which sends the character to the Teletype.

The UUO level routines for TTY transfer characters between the user area and the monitor buffers. The TTCALL routines move characters to or from a location specified by the instruction. The INPUT and OUTPUT UUO's refer to a buffer ring in the user area, just as they do for all other devices. Unlike other device routines, however, the TTY UUO level routines actually empty or fill the user's buffers.

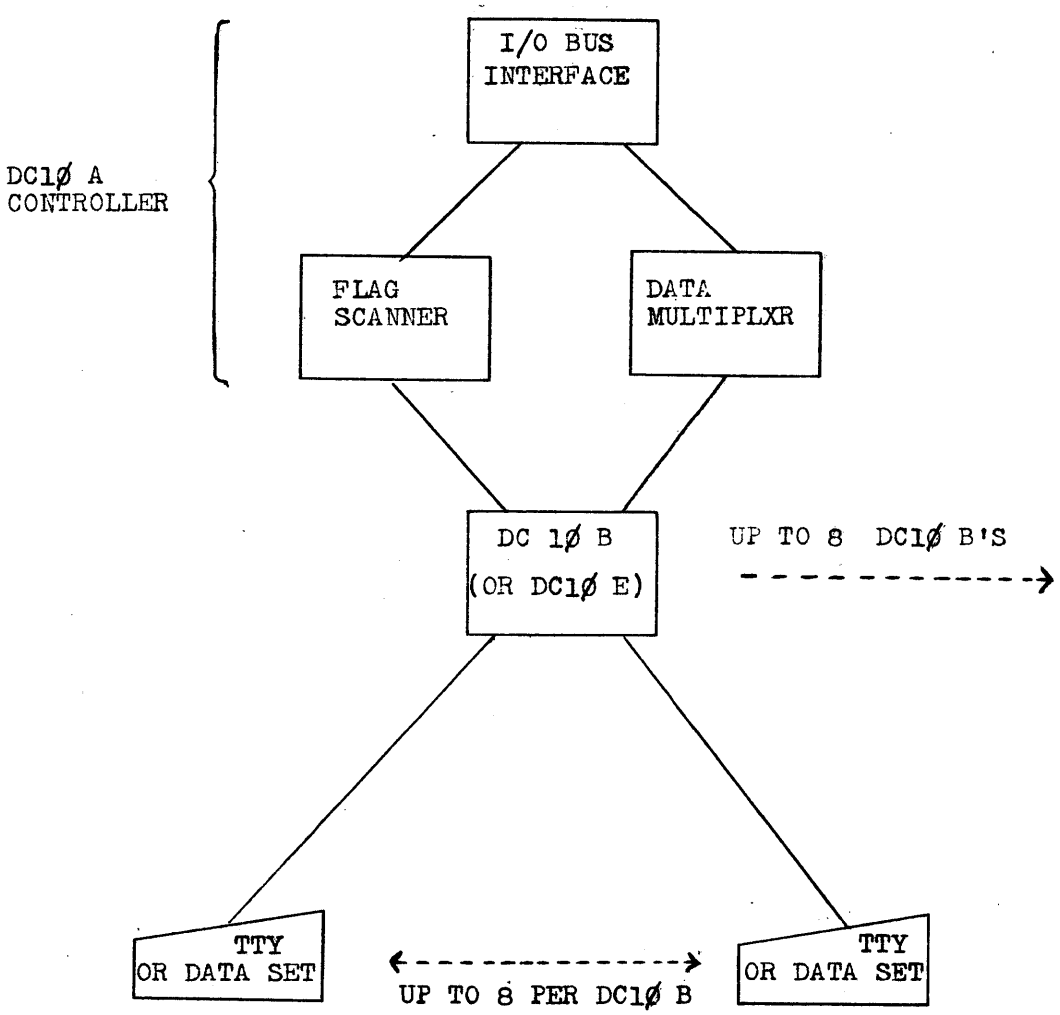
When an INPUT UUO for TTY is executed by a user program, UUOCON advances the user's pointers to the next buffer. If the next buffer is full, control is returned to the user. The Input routine in SCNSRF is called only when all the

user's buffers are empty. The Input routine then copies as much data as it can from the monitor buffer to the user's ring of buffers. It advances to the next user buffer when either a break character is reached, or the user buffer is filled. The Input routine exits when all the user buffers have been filled, or the last full line has been copied from the monitor buffer.

If the Input routine is called and there is not a full line in the monitor buffer, the job must be put into TTY IO Wait. Control is passed to TWSYNC and on to WSYNC. WSYNC marks the job to be requeued to TTY IO Wait and calls WSCHED to start a new monitor cycle.

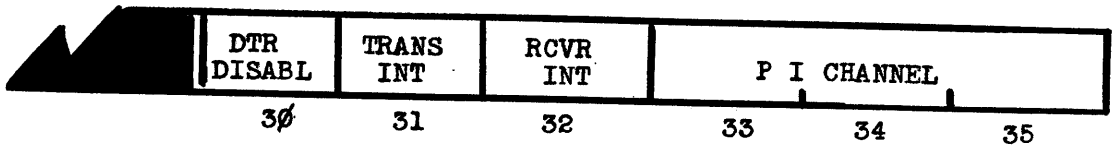
The TTY Output routine is called by UUOCON each time the user program executes an OUTPUT UUO for TTY. The TTY Output routine copies the user's buffer into the monitor output buffer and exits. If there is not enough room in the monitor output buffer, the job is put into TTY IO Wait while the buffer is emptied at interrupt leve. When the buffer is almost empty, the job will be requeued to run and will continue where it was interrupted. When the user's buffer has been copied into the monitor buffer, control is returned to UUOCON, and from there to the user program.

DC1Ø DATA LINE SCANNER

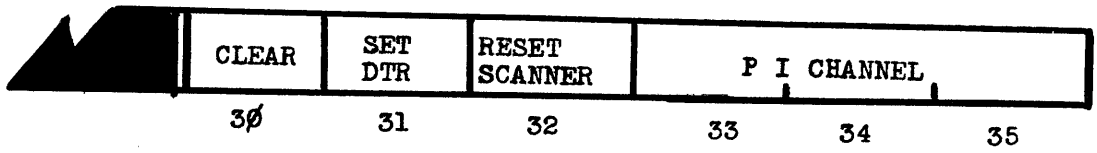


SCANNER INSTRUCTIONS

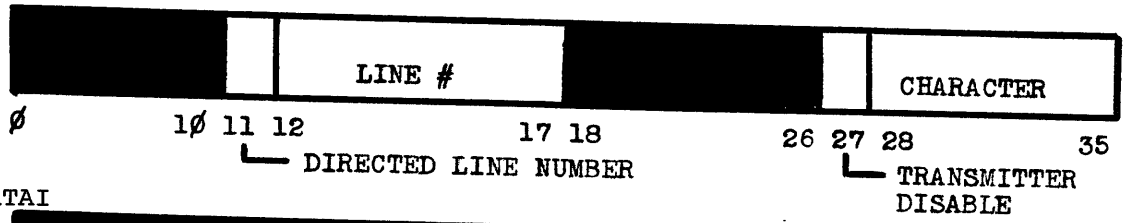
CONI



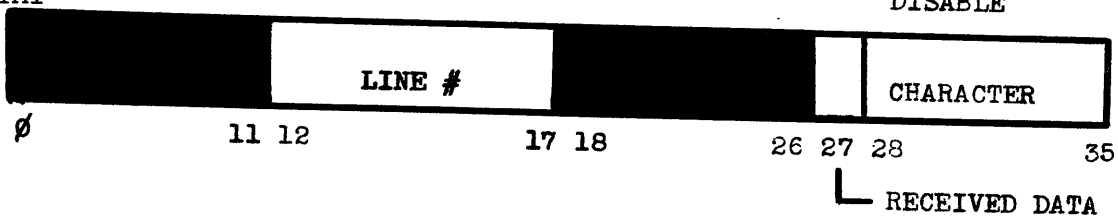
CONO



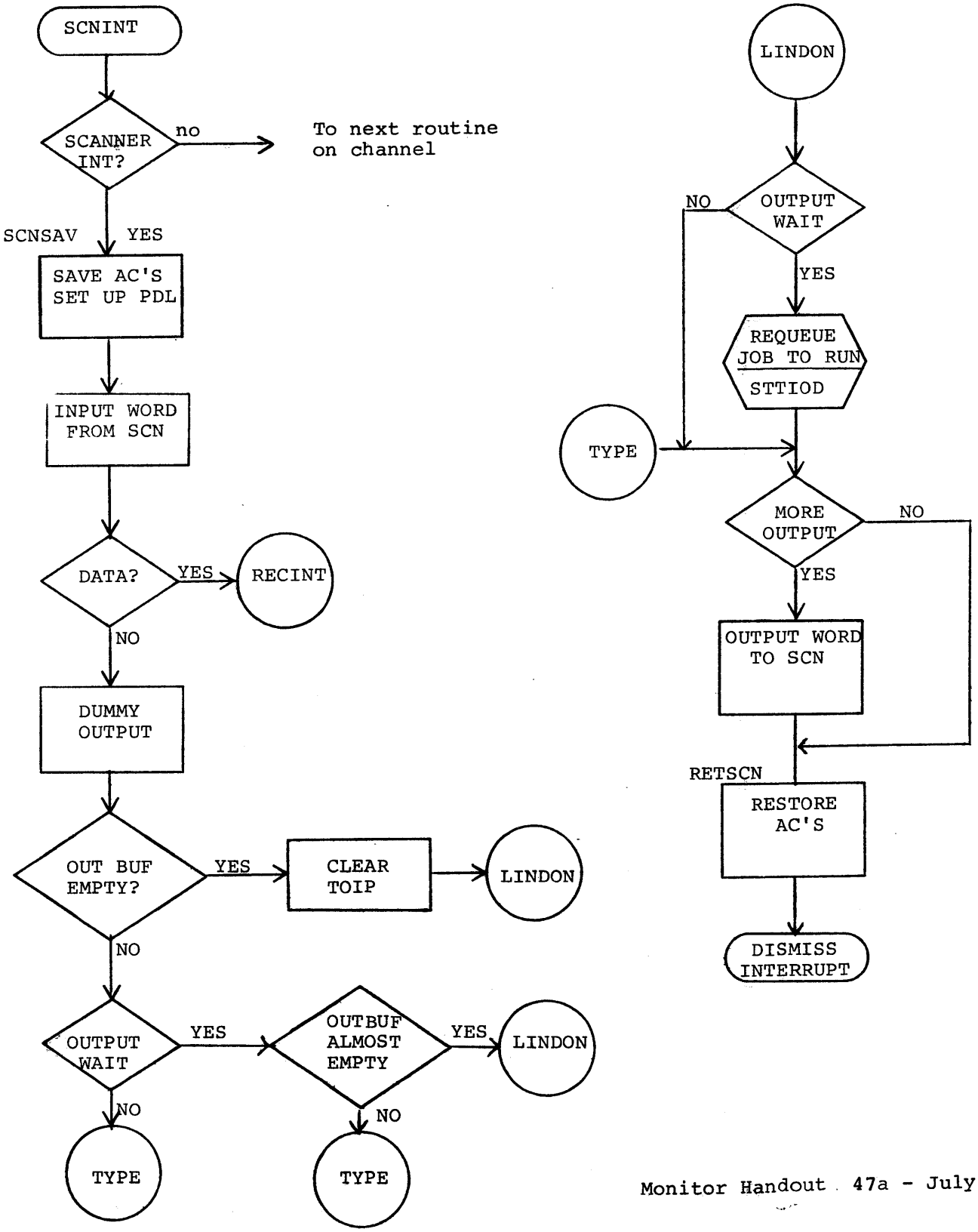
DATAO



DATAI

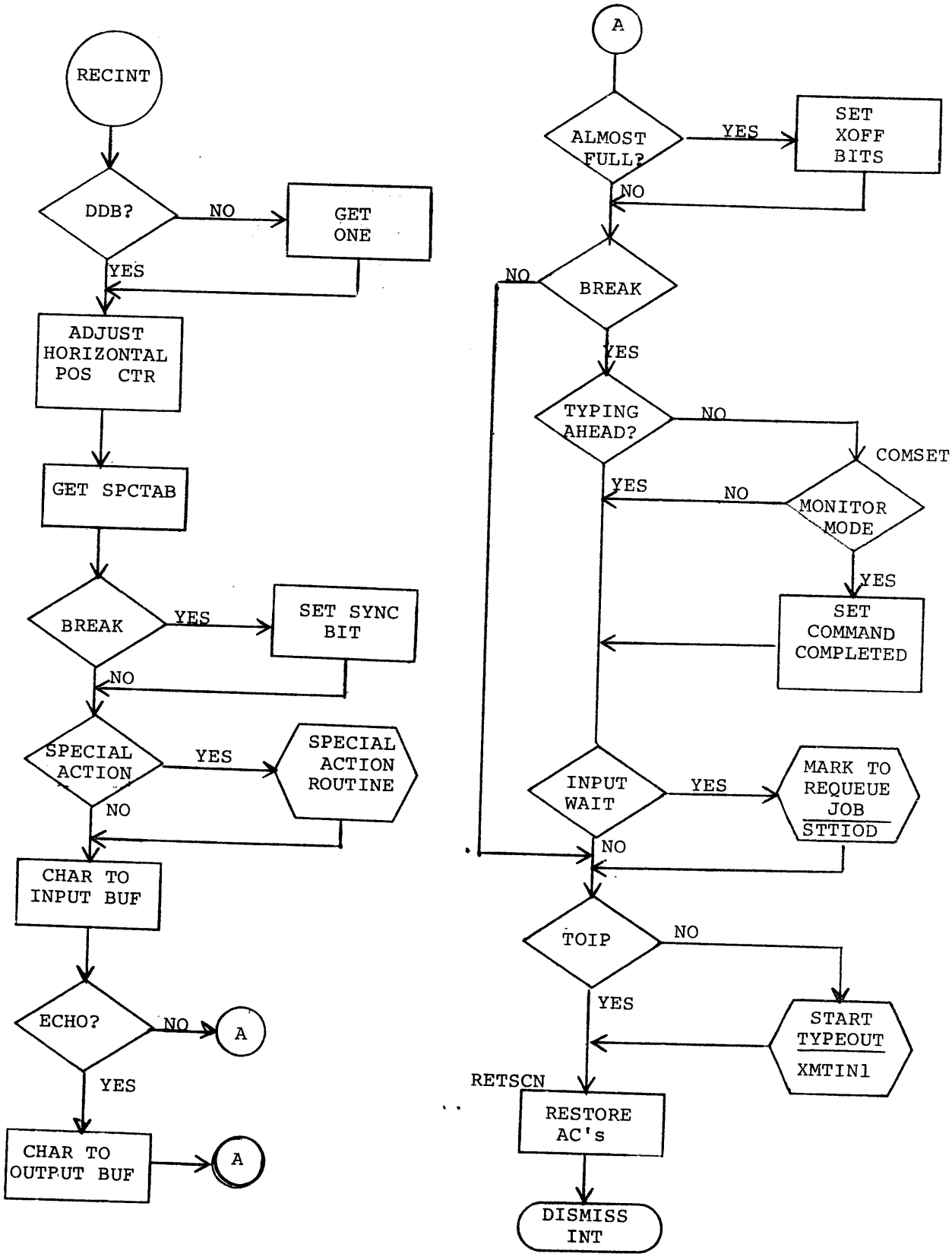


SCANNER INTERRUPT ROUTINE

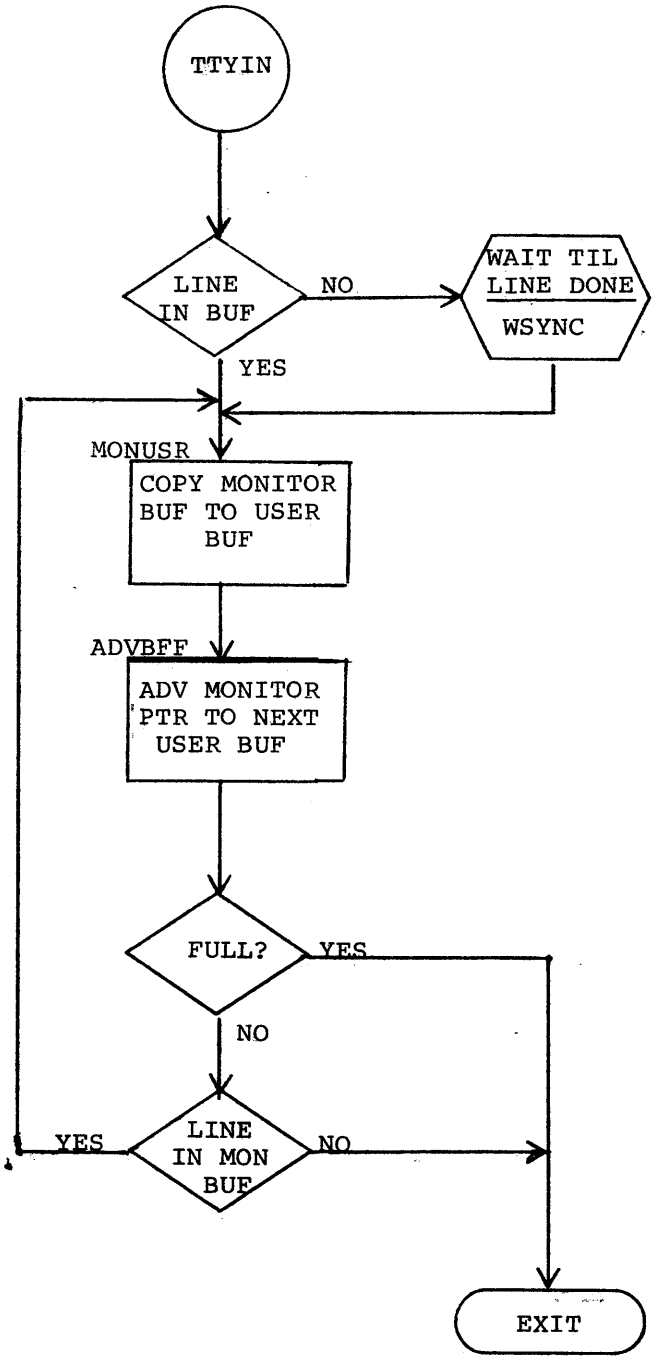


Monitor Handout 47a - July 70

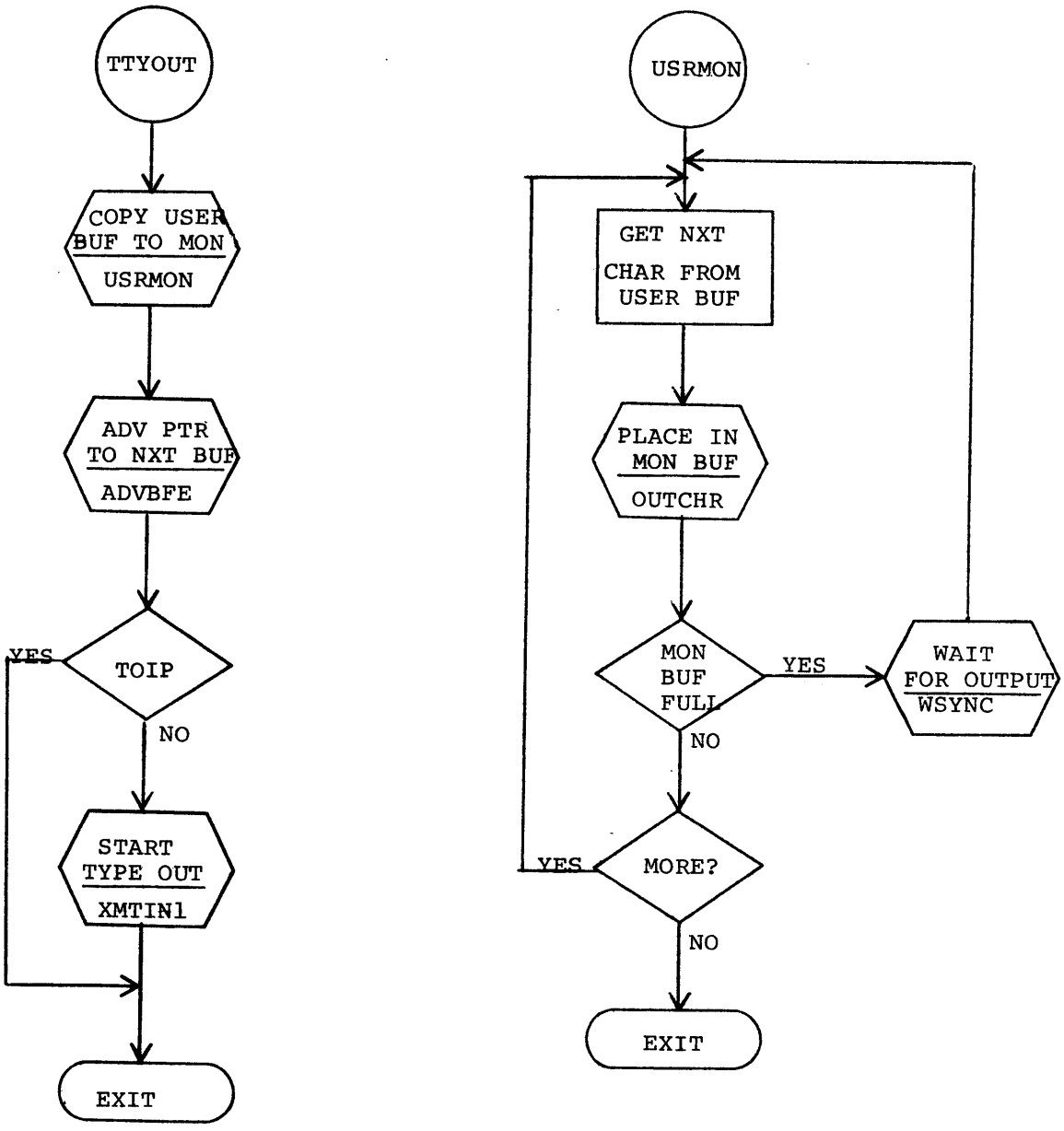
SCANNER INTERRUPT ROUTINE



Monitor Handout 47b - July 7Ø


TTYIN
LINE IN BUF
NO
WAIT TIL LINE DONE
WSYNC
YES
MONUSR
COPY MONITOR BUF TO USER BUF
ADVBFF
ADV MONITOR PTR TO NEXT USER BUF
FULL?
YES
NO
YES
LINE IN MON BUF
NO
EXIT

TTY OUTPUT UUO



QUESTIONS ON SCANNER SERVICE

On all questions which ask "where", specify program and line where the action is is taken, and describe the circumstances under which that line will be executed.

1. When will the device independent (UUOCON) routine call the SCNSRF routine for (a) Input? (b) Output?

2. Where is the decision made to put a program into TTY IO Wait for the INCHWL TTYCALL?

3. Where is a program put into IO Wait for TTY buffered input?

4. Where is a program put into IO Wait for TTY buffered output?

5. Where is the COMSET routine called to set the "Command Completed" bit?

6. Where is TOIP set? cleared?

7. Where is a TTY DDB set up for a particular physical line?

8. Where is the routine to respond to ↑O. How does control get to that point?

9. Where is the next character actually sent out to the scanner for a transmit interrupt? What determines the TTY on which it will be printed?

10. What does SCNSRF do if someone tries to type on a TTY, and all DDB's are in use? Where is this decision made? How could such a condition arise?