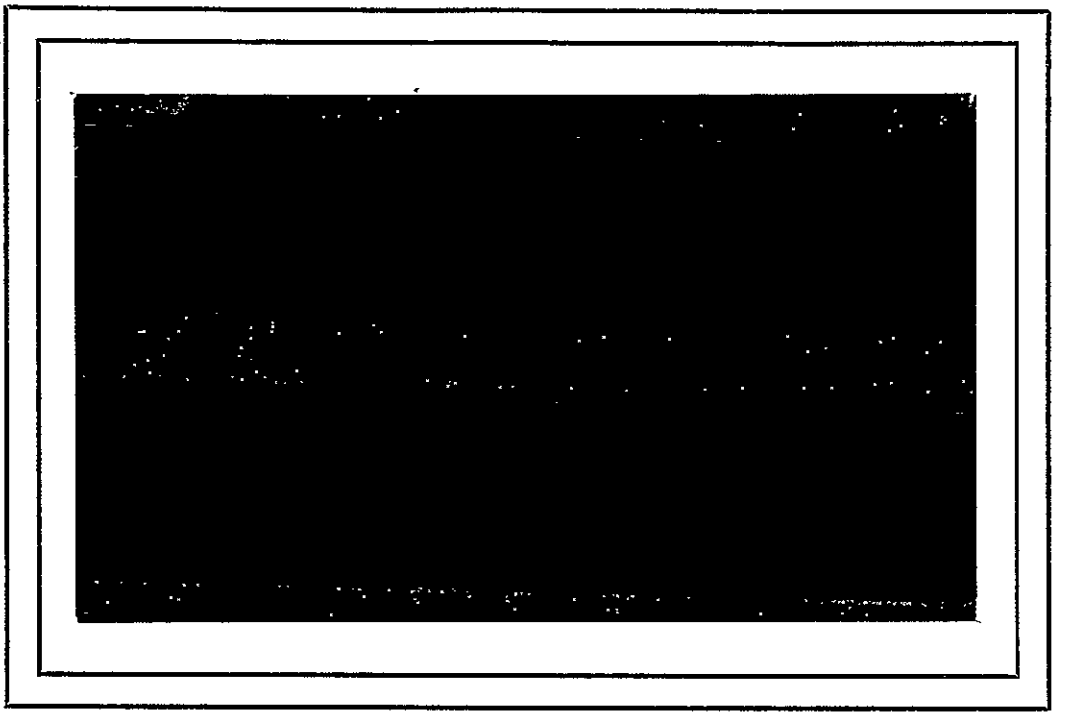


UNIVERSITY OF MARYLAND
COMPUTER SCIENCE CENTER

COLLEGE PARK, MARYLAND

Technical Report TR-155 May 1971
NGR-21-002-197

Simulation of a Paging Drum Channel

by

Gee-yin Kwok

This research was supported by Grant NGR-21-002-197 from the National Aeronautics and Space Administration to the Computer Science Center of the University of Maryland.

Table of Contents

Abstract

This report describes the simulation of a paging drum channel system by Algol and Simula programs on the UNIVAC 1108 in order to ascertain the paging algorithms. The paging drum channel system consists of five system units: paging drum channel, central processing unit, paging drum memory, main memory, and page table memory. They are described by block diagrams and in CDL (computer design language) statements. The algorithms for the paging sequences and the sequential operation of the system units are presented in flow charts and described in detail.

Simulation of a Paging Drum Channel

1. Organization

A paging drum channel may be regarded as a processor which pages-in and pages-out main memory pages. Paging-in transfers a page of words from the drum through the paging drum channel to the main memory, while paging-out transfers a page of words from the main memory through the paging drum channel to the drum.

Figure 1 is a block diagram which shows a part of a computer system. There are five system units: paging drum channel PDC, central processing unit CPU, paging drum memory PDRUM, main memory MM, and page table memory PTM. Except the CPU, these system units constitute a part of the virtual memory that has been described elsewhere (8). This part functions briefly as follows. When the CPU needs a page not in the MM, it initiates the PDC to fetch the page. The PDC generates an interrupt when the page transfer is completed or when there is an error. System units PDRUM, PTM and MM are now described.

1.1 Paging drum memory (PDRUM)

The words, tracks, pages, sectors and fields on the drum surface are shown in Fig. 2. The drum surface is divided along the circumference into 16 sectors. It is also divided along the axis into 64 fields. The intersection of a sector and a field is a drum page. There are 64 fields, 16 sectors, and 1024 pages on the drum. There are 36 tracks in each field with one read/write head for each track. The 36 bits parallel with the axis form a word. There are 1024 words in each page. The data is trans-

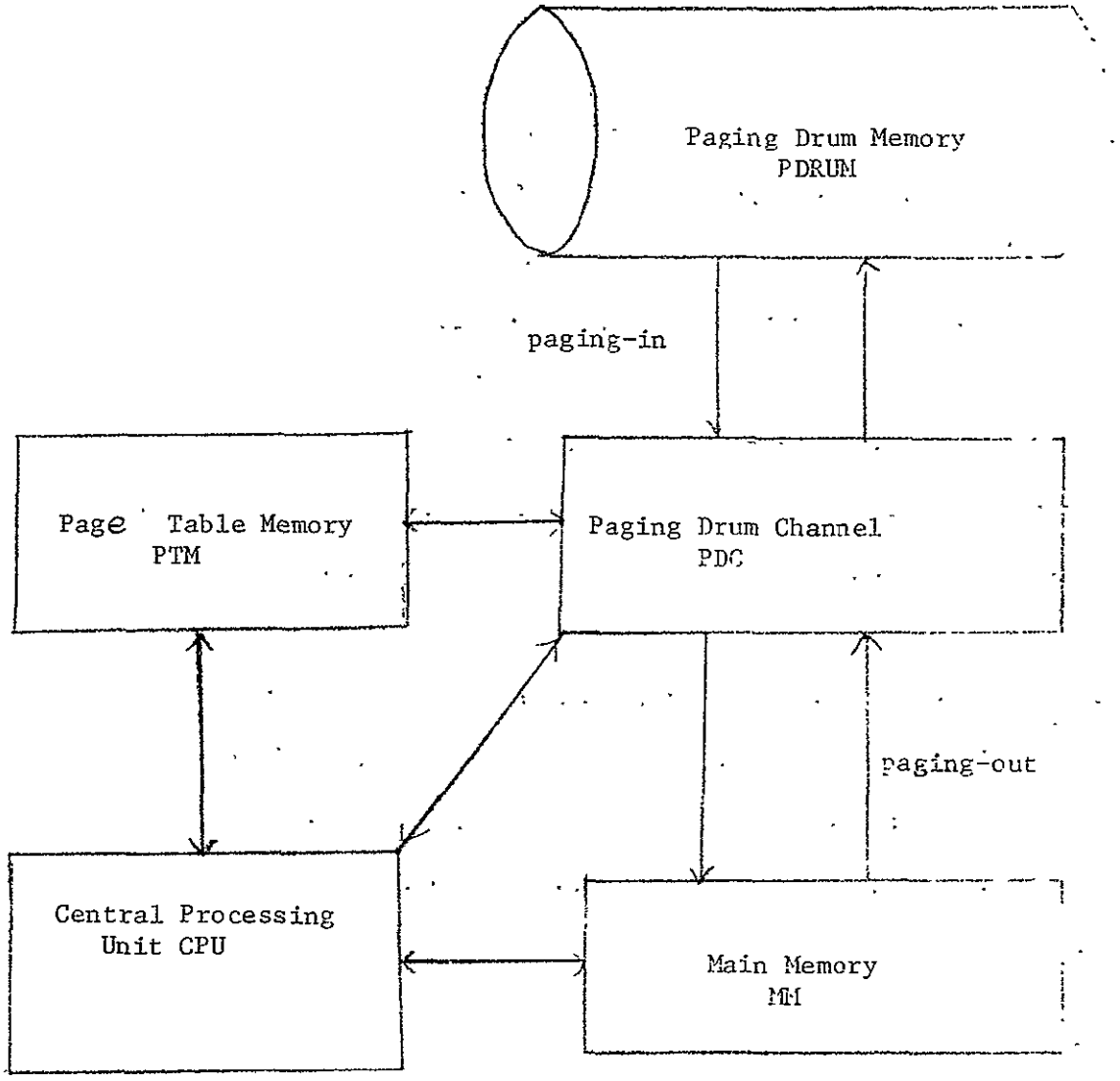
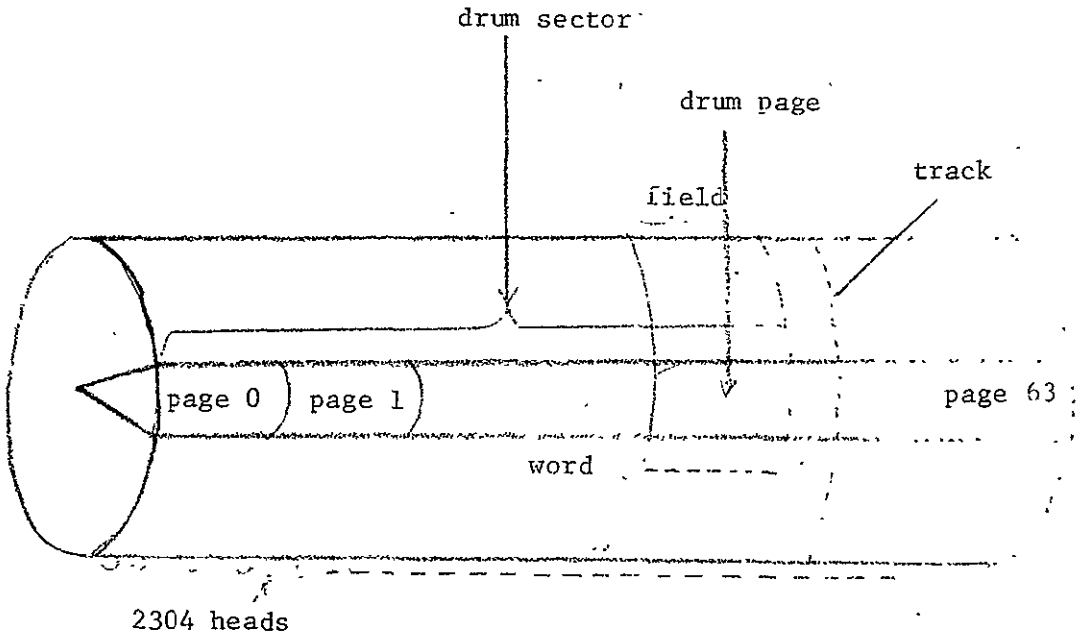


Fig. 1 System Units of a Computer System

ferred one word at a time.

The configuration of the PDRUM is shown in Fig. 3. The PDRUM is addressed by both registers FIELD and CWORD. Register CWORD contains two subregisters, CWORD(SECT) and CWORD(COUNT). CWORD(SECT) is the sector address and CWORD(COUNT) the word address of the current drum page. There are four single-bit registers, namely, DACTV, RW, BS, and PAGEI in addition to drum buffer register DBR. Register DACTV, when 1, indicates a request for a page transfer. Register BS, when 1, indicates that DBR is available. Register PAGEI is set to 1 when the drum is at the beginning of a drum sector. The configuration of the PDRUM is described by the CDL declaration statements as follows:

```
Comment, paging drum configuration                                    (1)

Memory,       PDRUM(FIELD,CWORD)=PDRUM(0-63,0-16383,1-36)

Register,     FIELD(1-6),     $drum field register

              CWORD(1-14),    $drum field word address

              DBR(1-36),      $drum buffer register

              DACTV,          $drum request when 1

              RW,             $drum write when 1; else, drum read

              BS,             $drum buffer is available when 1

              PAGEI,          $drum at the beginning of a sector when 1

Subregister,  CWORD(SECT)-CWORD(1-4),

              CWORD(COUNT)=CWORD(5-14)
```



36 bits/word
64 pages/sector
16 pages/field
1024 pages on the drum

Fig. 2 Pages, Sectors, and Fields of a Paging Drum

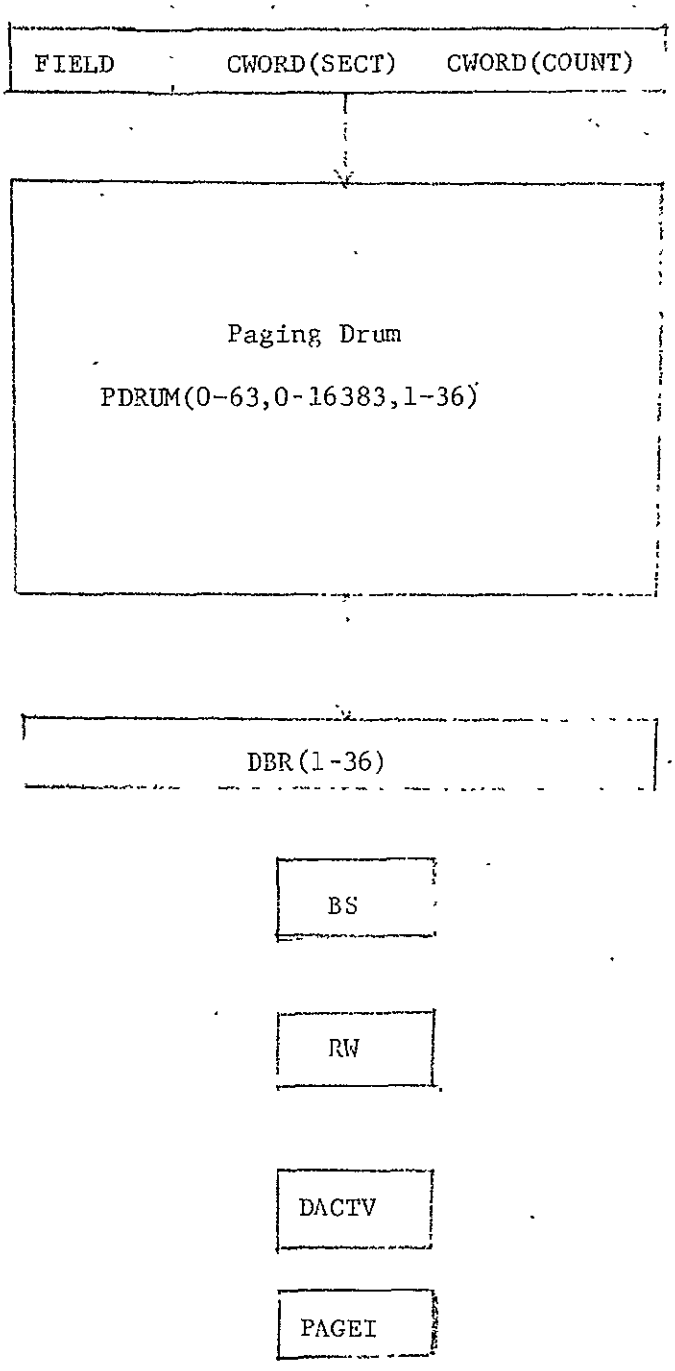


Fig. 3 Configuration of the Paging Drum Memory

The operation of the paging drum is shown in the sequence chart of Fig. 4. The PDRUM, after started, tests register DACTV. When DACTV is 1, there is a page transfer request and the read/write control register RW is examined. If RW is 1, a data word is transferred from the drum memory buffer register DBR into the PDRUM which is addressed by both FIELD and CWORD; this is called the output sequence. Register BS, when 1, indicates that register DBR is empty. If register RW is 0, a data word is transferred from the PDRUM to register DBR; this is called the input sequence. Register BS, when 1, indicates that DBR is full. Next, the word count in register CWORD is incremented. If a whole page has been read or written, CWORD(COUNT) contains 0, and register PAGEI is set to 1 to indicate the transfer being complete, and register DACTV is set to 0 to indicate that the paging drum is ready for next page transfer. If a page has not yet been read or written, CWORD(COUNT) is non-zero, so the read or write operation is repeated, depending on the contents of register RW.

1.2 Page table memory (PTM)

The configuration of the page table memory PTM is shown in Fig. 5. The PTM consists of a memory PAGETABLE, address register PADR, buffer register PTR2, read/write control register RW3, and access control register PTSEM. When PTSEM(1) is 1, the PTM is accessable by the CPU. When PTSEM(2) is 1, the PTM is accessable by the PDC. The configuration of the PTM is described by the CDL statements below:

```
Comment, configuration of the page table memory                    (2)

Memory,      PAGETABLE(PADR)=PAGETABLE(0-63,1-64)   $page table memory
```

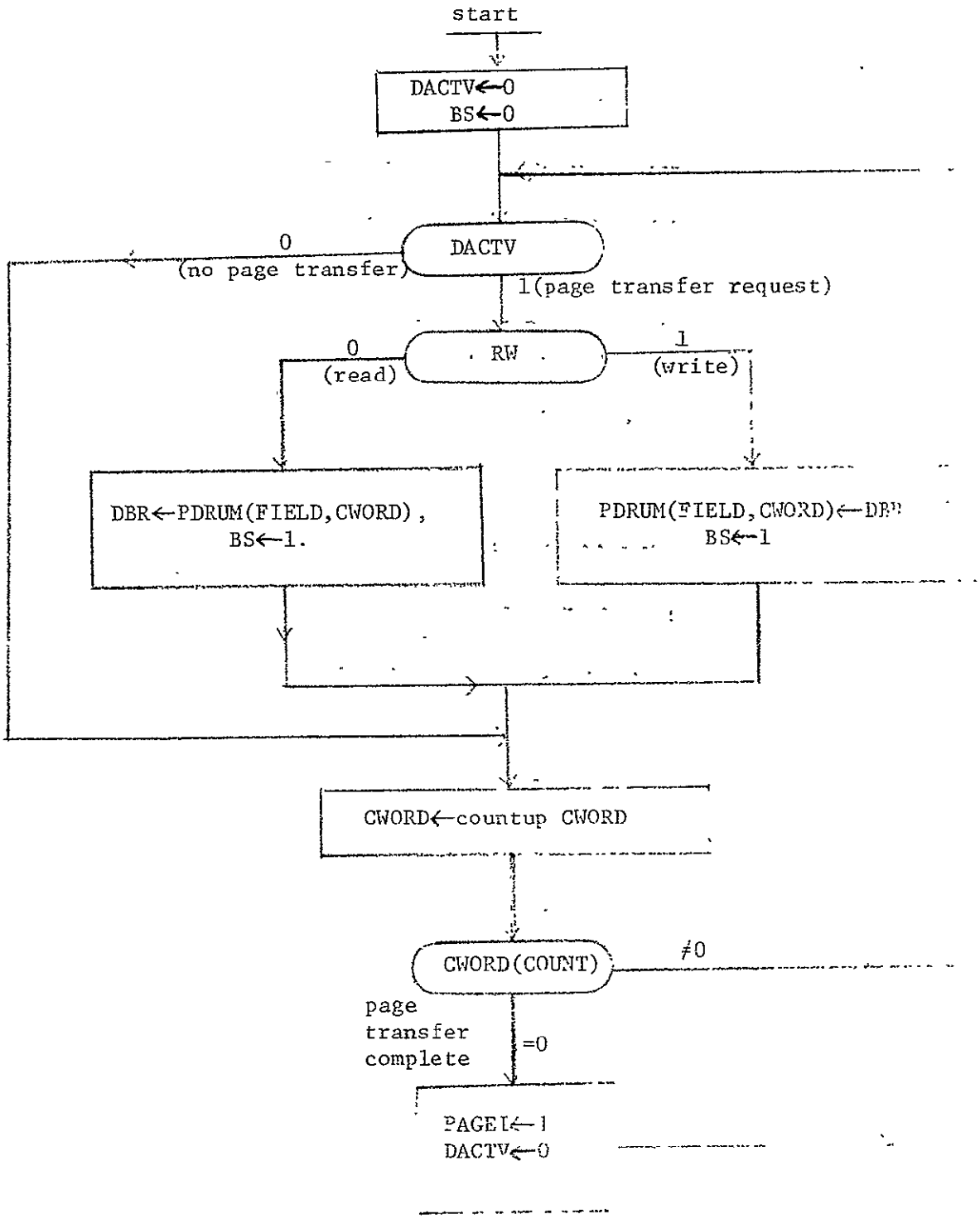


Fig. 4 Sequence Chart for Paging Drum Memory

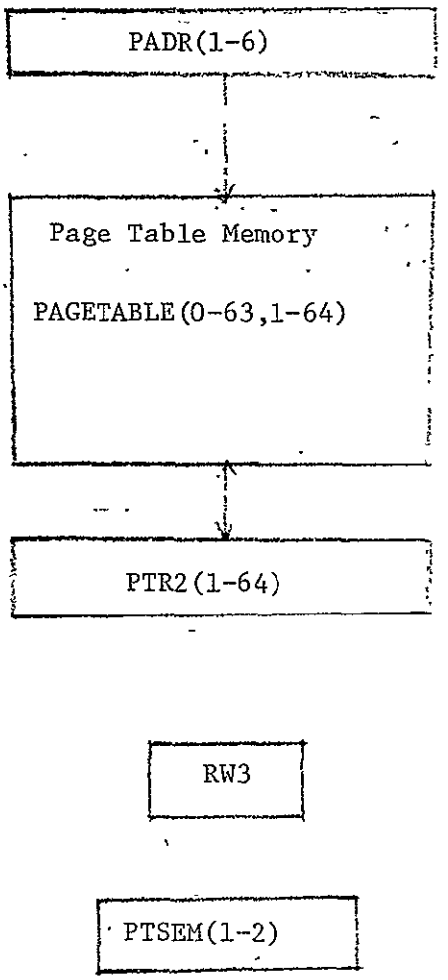


Fig. 5 Configuration of the page table memory

PADR(1-6), $PTM address register

PTR2(1-64), $PTM buffer register

PTSEM(1-2), $PTM access control register

RW3, . . $PTM read/write control

Subregister, PTR2(CH,SEC,ROW)=PTR2(56-61,60-63,64),

PTR2(LB,LF)=PTR2(3-8,9-14),

The page table memory PAGETABLE contains 64 page descriptors which link the 64 MM pages. The information in a page descriptor is listed below:

(a) the current status of the page,

(b) the task which the page is or was attached to,

(c) protection bits,

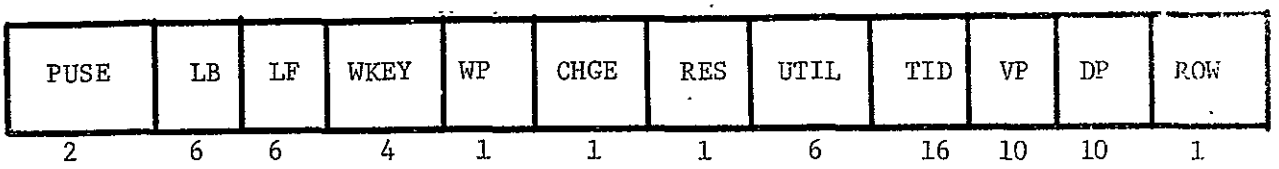
(d) utilization information,

(e) the corresponding virtual address of the page,

(f) the drum address of the page, and

(g) list linkage information.

The page descriptor format is shown in Fig. 6. There are 12 fields in a page descriptor. However, only fields LB, LF, DP, and ROW are needed for this simulation. LB is the backward link pointing to the previous page descriptor of the list of page descriptors. LF is the forward link pointing to the next page descriptor of the list. DP is a 10-bit drum page address; the first 6 bits of DP specifies the field address of the drum page while the other 4 bits of DP specifies the sector address. ROW is a read/write indicator. If ROW is 0, it is the read operation; else, write operation.



LB: backward link

LF: forward link

DP: drum address of this page

ROW: read/write indicator

Note. The other fields are not used in the simulation. See reference (7)

Fig. 6 Page descriptor format

The operation of the PTM as shown in Fig. 7 starts by checking PTSEM(2) for 1. If PTSEM(2) is 0, PTM stays in a wait loop. If PTSEM(2) is 1, PTSEM(1) is set to 0 so that the CPU cannot access the PAGETABLE memory. The read/write control register RW3 is next examined. If RW3 is 1, it is the write sequence. A page descriptor is transferred from the buffer register PDR2 in the PDC to the buffer register PTR2 in the PTM, and the address of the PAGETABLE memory is transferred from the address register PADR2 in the PDC to the PAGETALBE memory address register PADR in the PTM. The page descriptor is then stored into the PAGETABLE memory. If RW3 is 0, it is the read sequence. The page descriptor address is transferred from PADR2 in the PDC to PADR in the PTM. The corresponding page descriptor is fetched from the PAGETABLE memory into the buffer register PTR2. Next, the page descriptor is transferred to PDR2 in the PDC. At the end of a read or a write sequence, PTSEM(2) is set to 0 to indicate a complete operation and PTSEM(1) is set to 1 so that the CPU may access the PAGETABLE memory. The PTM then waits until PTSEM(2) is set to 1 again.

1.3 Main memory (MM)

The main memory MM subsystem as shown in Fig. 8 consists of a core memory MEM, a memory address register MAR, and a memory buffer register MBR. In addition, there are control registers MA and RW2. To access MM, the PDC sets RW2 to 1 for a read or to 0 for a write; a memory access request is made by setting MA(2) to 1. The configuration of the MM is now described by the following CDL statements:

```
Comment, configuration of the main memory                    (3)
```

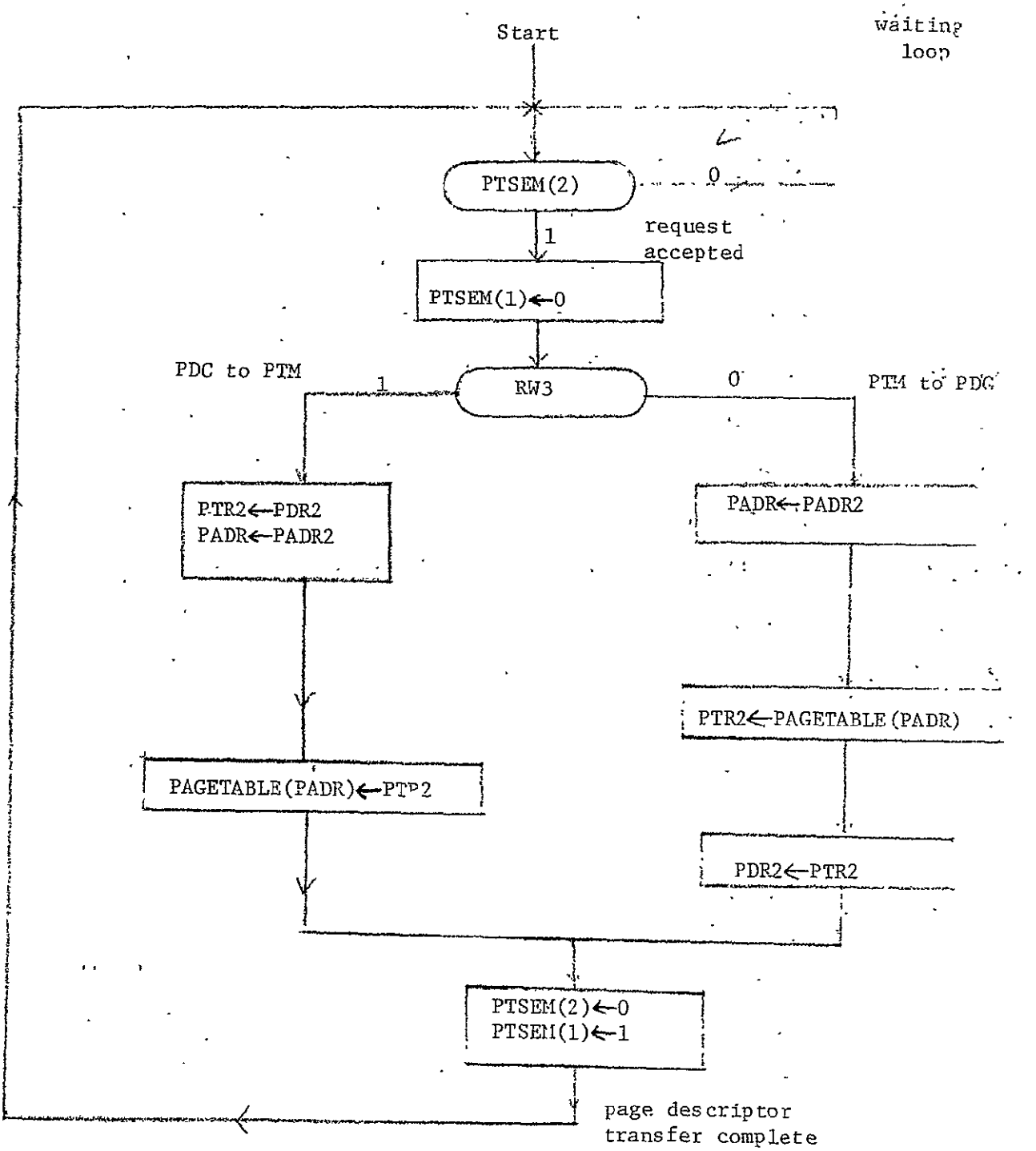


Fig. 7 Sequence Chart for the Page Table Memory

```
Memory,       MEM(MAR)=MEM(0-65535,1-36)

Register,     MAR(1-16),     $MM address register

              MA(1-3),       $MM access register

              READ;          $read control

              WRITE,         $write control

              RW2,           $MM read/write control register

Subregister,  MAR(BLOCK,WRD)=MAR(1-6,7-16)
```

The operation of the MM as shown in Fig. 9 begins by examining register MA. Whenever MA(2) is set to 1, the read/write control register RW2 is next tested. If RW2 is 0, it is the input sequence. The MM buffer register MBR receives data from the auxiliary buffer register SBR2 in the PDC while memory address register MAR obtains the MM address from the auxiliary register MADR2 in the PDC. The WRITE control is set to 1 to initiate a write operation into the main memory proper. If RW2 is 1, it is the output sequence. MM address is transferred from MADR2 in the PDC to MAR in the MM and the READ control is set to 1 to initiate a read operation into the buffer register MBR. Subsequently, the out-going data is transferred into SBR2 in the PDC, and MA(2) is clear to notify the PDC that data has been transferred. The MM unit stays in a wait loop until MA(2) is set to 1 again.

The characteristics of the MM and the PDRUM are summarized in Table 1. So far we have described all the system units except the PDC which will indeed tie up all the units and perform paging.

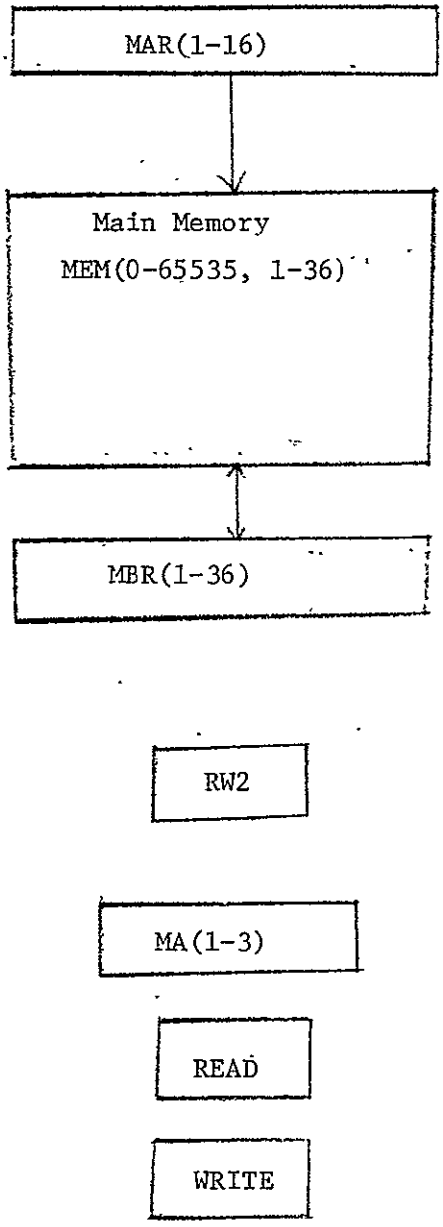


Fig. 8 Configuration of the Main Memory

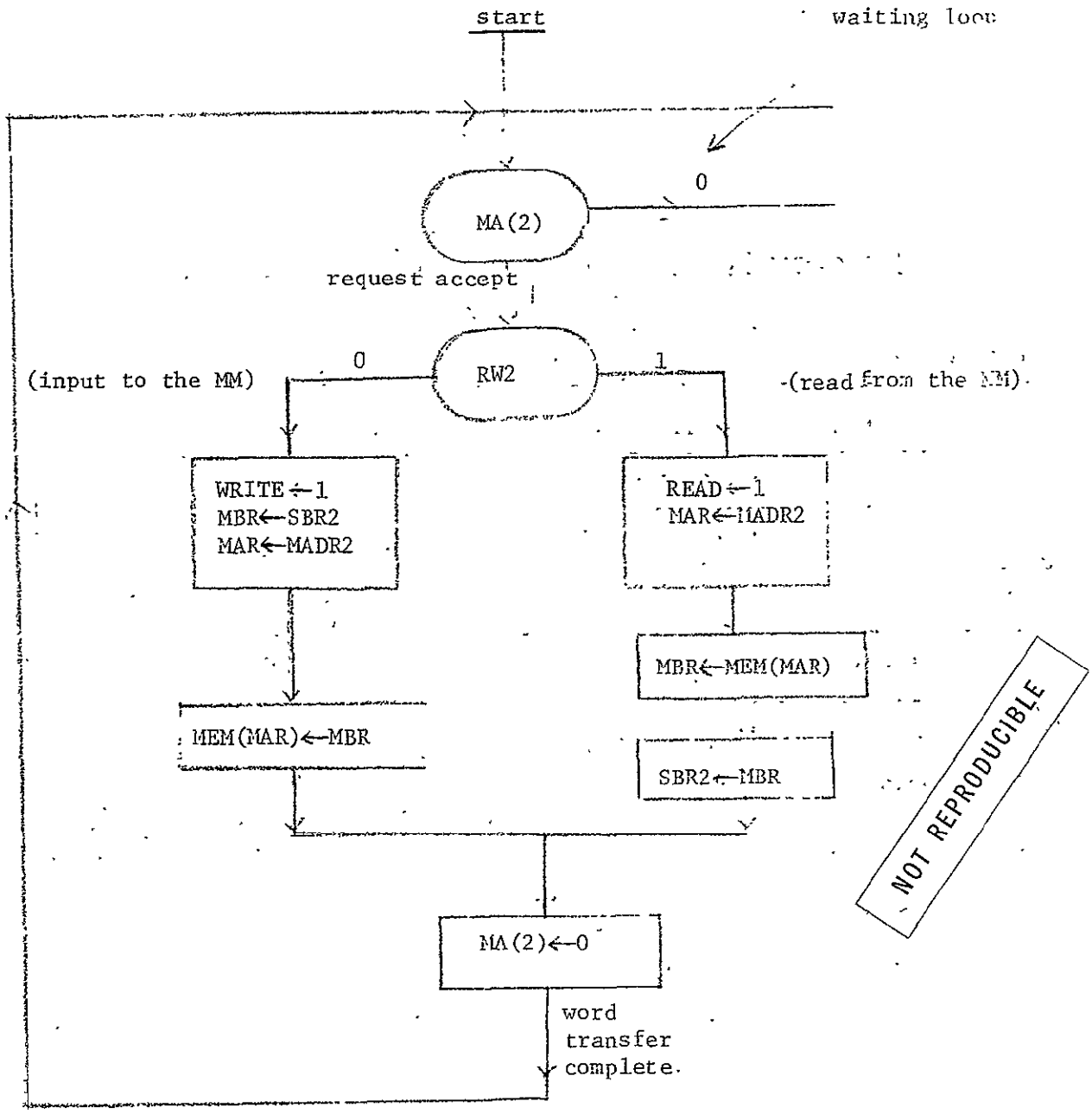


Fig. 9 Sequence Chart for Main Memory Operation

Table 1 Characteristics of the Main Memory and the Drum Memory

Characteristics	Main Memory	Drum Memory
memory cycle time	1 microsecond	0.001 microsecond
data transfer width	36 bits or 1 word	36 bits or 1 word*
data units	(a) 36 bits per word (b) 1K** words per page	(a) 36 bits per word (b) 16 pages per field (c) 16 K bits per track (d) 64 bands
memory capacity	(a) 64 K words (b) 64 pages	(a) 1024 K words (b) 1024 pages
address register	word address: 16 bits	(a) field address: 6 bits (b) sector address: 4 bits (c) page address: 22 bits (d) word address: 10 bits

* there are 36 read/write heads per field

** K represents a multiple of 1024

2. Paging-drum Channel (PDC)

The paging drum memory PDRUM which provides the backing storage of a virtual memory system is controlled by the PDC. A page of words is transferred from the PDRUM through the PDC to the main memory MM when the CPU needs a missing page, or a page of word is transferred from the MM through the PDC to the PDRUM when the CPU needs the space of a page for a new page. This section describes the configuration of the PDC, and the formats of the channel command word and the listheads for 16 sector queues in the PTM. It also describes the handling of the lists and queues in the page table memory both by the CPU and by the PDC, and the paging sequence which consists of a main sequence, a drum read/write subsequence, an updating subsequence, and a request-accept subsequence.

2.1 Configuration

The configuration of the PDC is shown in the block diagram of Fig. 10. There are two memories. Memory COM with address register SEC and buffer register COMMAND has a capacity of 16 52-bit words. Memory LISTS with address register SECTORS and buffer register PTL has a capacity of 16 12-bit words. Register PAGINT contains the MM page which causes the interrupt. Register INTERRUPT is a 10-bit interrupt register regularly tested by the CPU. If the fourth bit of register INTERRUPT is set, interrupt is a result of a successful page transfer; if the 10th bit is set, interrupt is a result of an unsuccessful page transfer. Register FIELD contains the field address of a drum page. Register POST is an indicator which is set to 1 when the CPU requests a MM page. Register PC contains the current MM page address. Whenever the drum heads are at the beginning of a drum sector, register PAGEI is set to 1. Register PTRAN may contain 0.

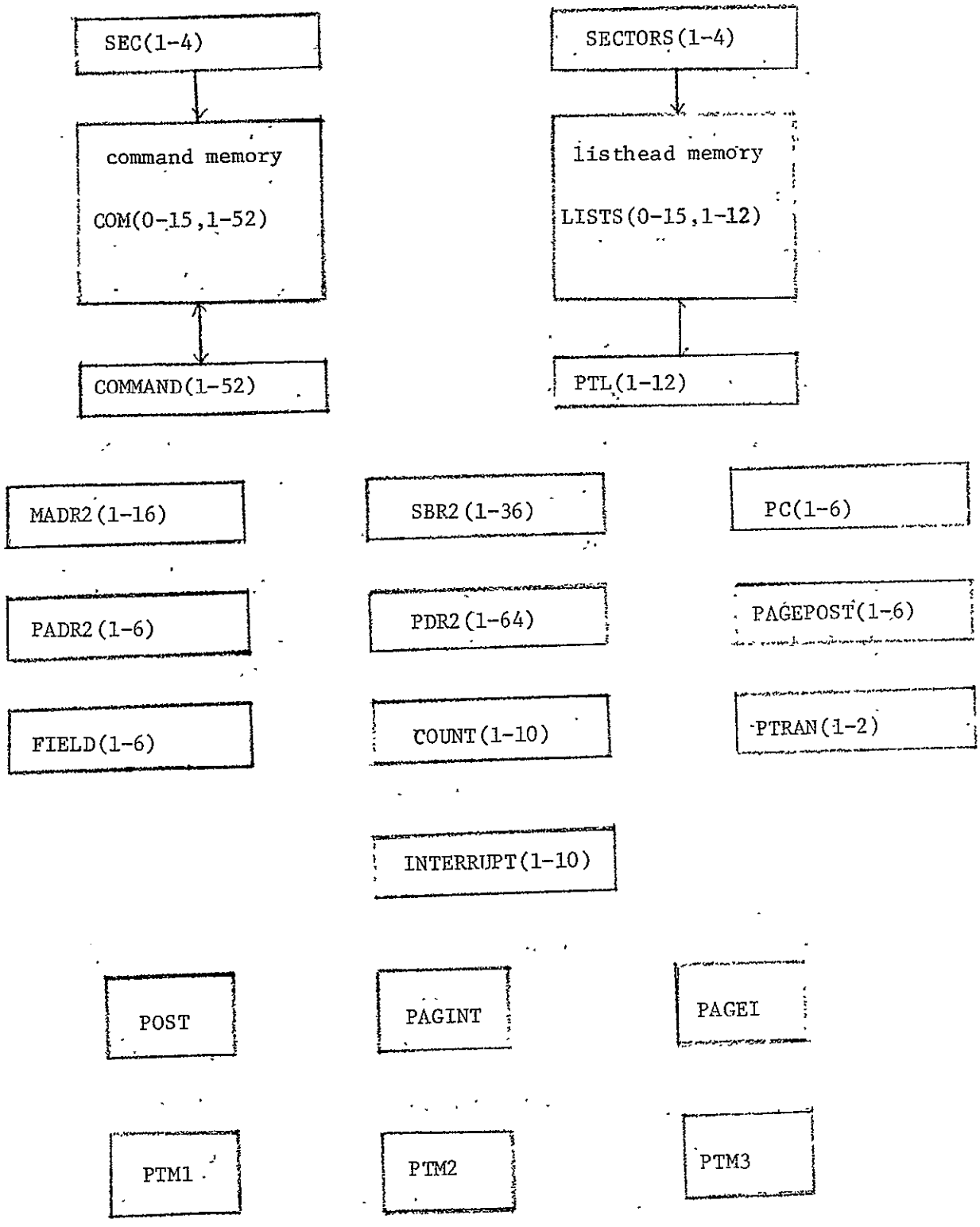


Fig. 10 Configuration of the Paging Drum Channel

1, 2, or 3. If it contains 0, there is no page transfer. If it contains 1, it is the read operation. If it contains 2, it is the write operation. If it contains 3, an error occurs in the transfer. Register COUNT stores the word address of the page being transferred. In order that the PDC and the MM can operate asynchronously, registers SBR2 and MADR2 are needed for transferring the MM address from the PDC to the MM. SBR2 corresponds to the MM buffer register and MADR2 corresponds to the MM address register. In order that the PDC and the PTM can operate asynchronously, registers PDR2 and PADR2 are needed for transfering data between the PDC and the PTM. PDR2 corresponds to the PTM buffer register and PADR2 corresponds to the PTM address register.

The PDC configuration is now described in the following CDL statements.

```
Comment, configuration of the paging drum channel

Memory,       COM(SEC)=COM(0-15,1-52),     $command memory

Register,     SEC(1-4),          $command memory address register

              COMMAND(1-52),     $command memory buffer register

Subregister,  COMMAND(C,RWC,CHAN) = COMMAND(1,2;3-10),

              COMMAND(PGE,FIRSTWORD) = COMMAND(11-16,17-52),

Memory,       LISTS(SECTORS)=LISTS(0-15,1-12),   $listhead memory

Register,     SECTORS(1-4),      $listhead memory address register

              PTL(1-12),         $listhead memory buffer register

Subregister,  PTL(FP,LP)=PTL(1-6,7-12),  $listheads for the sector list in PTM
```

```
Comment, registers for a page transfer operation

Register,   FIELD(1-6),        $field address register

            POST,              $page transfer request when 1 by the CPU

            PAGEPOST(1-6),     $MM page address for a page-transfer re-
                                quest by the CPU

            PC(1-6),           $current MM page-address register

            PAGEI,             $indicates when 1 that drum heads are at
                                the beginning of a sector

            PTRAN(1-2),        $page-transfer status register

            COUNT(1-10),       $word address of the current page

            SBR2(1-36),        $MM buffer register

            MADR2(1-16),       $MM address register

            PDR2(1-16),        $PTM buffer register

            PADR2(1-6),        $PTM address register

            PTM1,              $initiates when 1 procedure 1

            PTM2,              $initiates when 1 procedure 2

            PTM3,              $initiates when 1 procedure 3
```

In the above description, there are three single-bit register PTM1, PTM2, PTM3 for initiating three page-table procedures to maintain the sector queues. When PTM 1 is 1, the page table procedure 1 is activated. This procedure as shown in Fig. 11 detaches the first page descriptor from the list pointed to by the listheads in PTL, places the MM page address in

register PC, and leaves a copy of the page descriptor in the buffer register PDR2. When PTM2 is 1, the page table procedure 2 is activated. This procedure as shown in Fig. 12 loads PDR2 from the location specivicd by register PAGEPOST in the page table memory PAGETABLE. When PTM3 is 1, the page table procedure 3 is activated. This procedure as shown in Fig. 13 adds the page descriptor addressed by PAGEPOST to the list of page descriptors addressed by the listheads in PTL while PDR2 is used as the buffer register.

2.2 Formats

The channel command word CCW format is shown in Fig. 14. The 52 bits of a CCW are partitioned as follows: 1 bit for the C field, 1 bit for the RWC field, 8 bits for the CHAN field, 6 bits for the PGE field and 36 bits for the FIRSTWORD field. When C is 0, there is no page transfer between the PDRUM and the MM. When RWC is 0, a drum page is to be transferred from the PDRUM to the MM through the PDC. When RWC is 1, a MM page is to be transferred from the MM to the PDRUM through the PDC. Only 6 bits of the 8-bit CHAN field are used because there are 64 fields per drum sector. The 6-bit PGE field contains the MM page address. However, PGE is non-zero since we assume that MM page 0 is not available. The 36-bit FIRSTWORD field contains the first actual word of MM page just in case the transfer is from the MM to the PDRUM.

The format of an entry in the listhead memory is shown in Fig. 15. There are two 6-bit fields. The first 6 bits specify the location of the fron node of the doubly linked sector queue in the PAGETABLE memory. The last 6 bits specify the location of the rear node of the doubly linked sector queue.

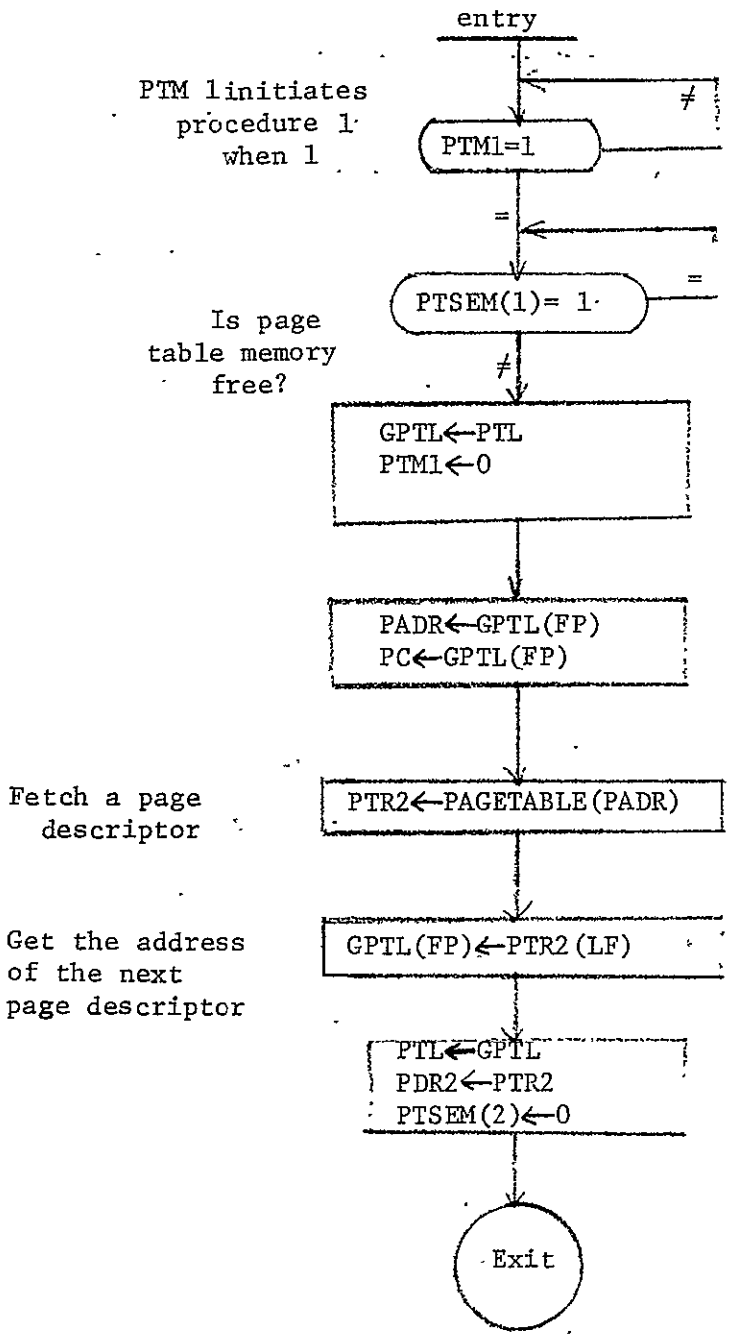


Fig. 11 Flow Chart of Procedure 1 which Fetches the Page Descriptor from the list addressed by PTL

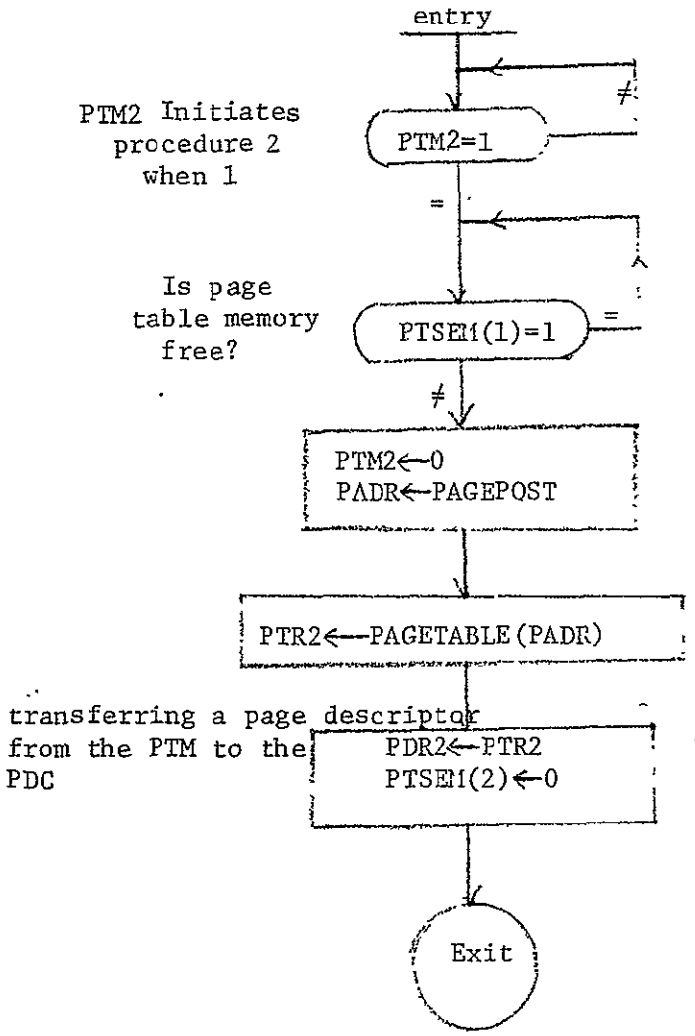


Fig. 12 Flow Chart of Procedure 2 which transfers a page descriptor to the PDC

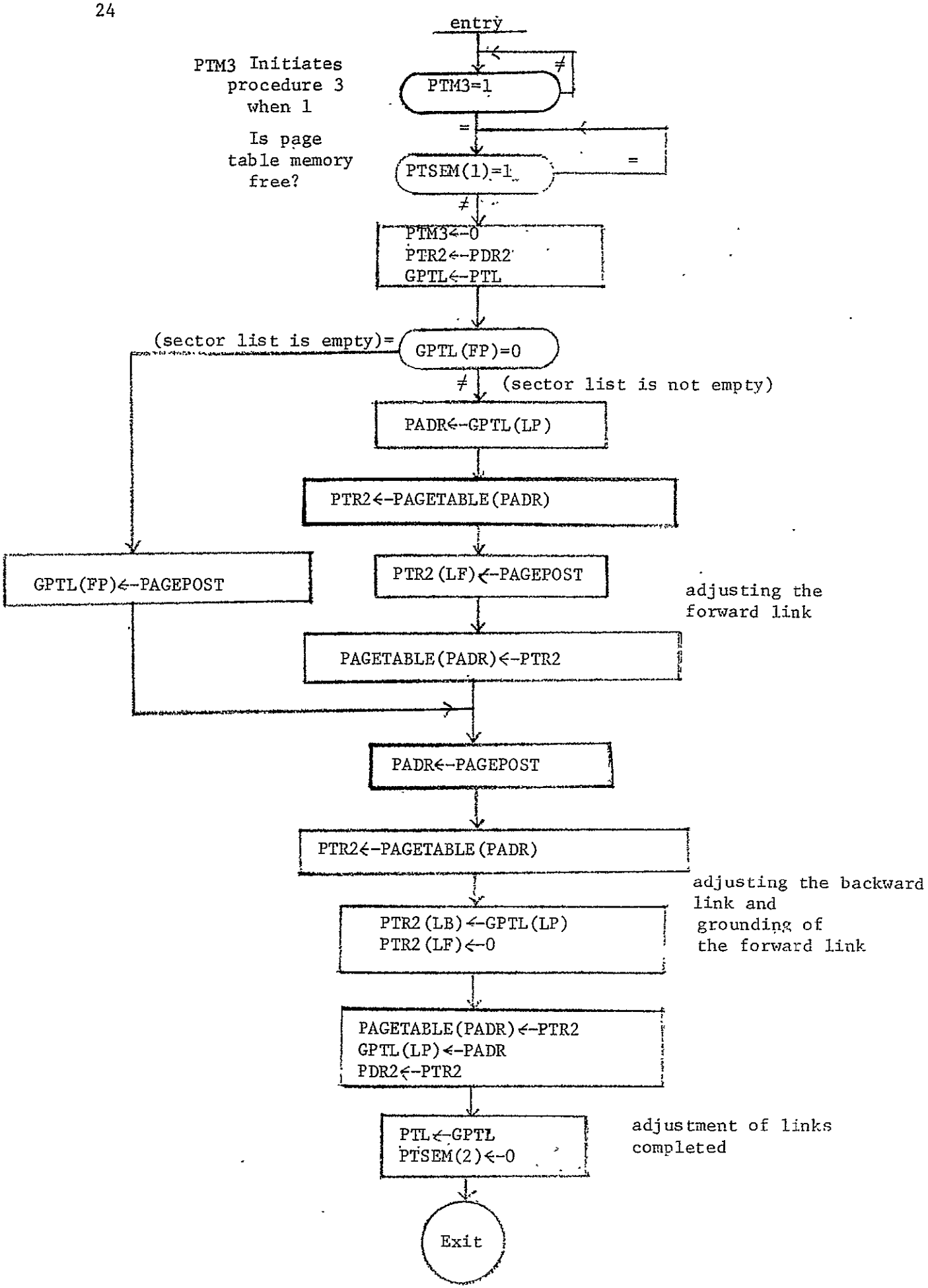
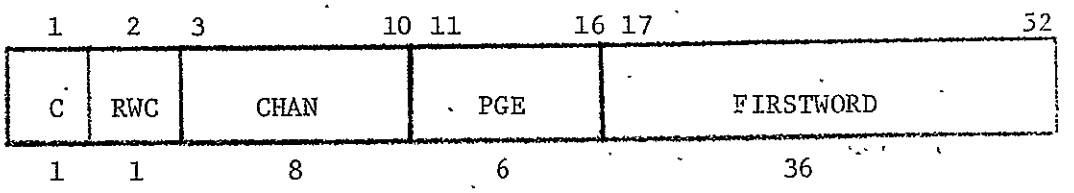


Fig. 13 Flow chart of procedure 3 which stores and updates a page descriptor lists



C: no page transfer when C=0; else, there is a page transfer

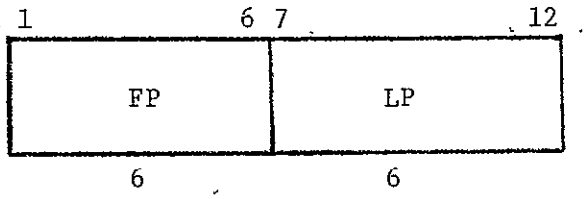
RWC: page to be read when 0; else, page to be written

CHAN: drum field address.

PGE: MM page address

FIRSTWORD: first word of the transferring page

Fig. 14 Channel Command Word Format



FP: the listhead for the front of the doubly linked list in the PAGETABLE memory when an element of the list is detached.

LP: the listhead for the rear of the doubly linked list when an element of the list is inserted

Fig.15 Format of an Entry in the Listhead Memory

2.3 The handling of queues

This section outlines the manner in which the lists and queues in the page table memory are manipulated. Sixteen sector queues are handled by the PDC while the other lists are maintained by the CPU. In the following, the lists in the Page table memory are first described in detail; the contents of the CCW's in the command memory are explained; and the interaction between the CPU and the PDC during a paging operation is described.

2.3.1 The Lists in the PAGETABLE Memory

There are 64 pages in the MM. Each page is pointed by a page descriptor stored in the PAGETABLE memory. Each PAGETABLE memory address of the page descriptor corresponds to one MM page address. Thus, 64 PAGETABLE memory words store 64 page descriptors for 64 MM pages. The format of the page descriptor has been shown in Fig. 6. As shown, there are two 6-bit fields LB and LF(backward link and forward link) in the page descriptor. By means of these backward and forward links, the 64 MM pages can be linked into one or more lists in the PAGETABLE memory. Each node of the list is doubly linked with one link pointing in the forward direction by the forward links LF's and the other link pointing in the backward direction by the backward links LB's.

There are many lists of the MM pages linked by fields LB and LF of the page descriptors as follows:

(a) one available-page list which links those pages of the MM that are available to the CPU;

(b) one swappable-page list which links those pages in the MM that are released by the CPU to be transferred from the MM to the drum;

(c) one or more users' lists, each of which links those pages of the MM that belong to a particular user;

(d) sixteen sector lists, each of which links those pages of the MM that are waiting (i.e. already in the queue) to be transferred either from the MM to the drum or vice versa. Since these 16 lists are used as queues, they are also called sector queues.

To enable a quick access of the first entry and the last entry in each sector list, two 6-bit listheads are provided for each sector list. The 16 listheads for the 16 sector lists are stored in the 16 words of the LISTS memory. The listheads for the available-page list and the swappable-page list are stored in register LAVP(1-12) and LSP(1-12), respectively. These registers are not included in this simulation, since these lists are handled by the CPU. The listheads for the current user is stored in register PTLIST(1-12) or register GPTL(1-12); the listheads for the users' lists are stored in the system table permanently resident in the MM described elsewhere (8).

2.3.2 The Channel Command Words in the COM Memory

The words in the COM memory are called Channel Command Words, CCW's. Each CCW stores the following pertinent information for initiating a drum-memory transfer:

(a) MM page address,

(b) drum field address,

(c) read/write operation,

(d) transfer request, and

(e) the first actual words of the MM page in case the transfer is from the MM to the PDRUM.

There are 16 CCW's each of which is for one sector of the drum. The PDC constructs the CCW's using the information from the listheads and page descriptors in the sector lists.

There is an important exception to the way that the first page descriptor of every sector list is linked: this first page descriptor is deleted from its sector list, after its pertinent information for initiating the data transfer is used to construct the CCW for that sector. The page descriptor of this page can be located in the PAGETABLE memory by the CCW whose field PGE holds the MM page address of this page. The reason for not linking the first MM page to its sector list is to enable the immediate accessability of the COM memory in the PDC, since the COM memory is exclusively accessable by the PDC while the PAGETABLE memory is accessable by the PDC and the CPU. As a result, the PDC can rapidly respond to the drum each time when a new drum sector begins to be scanned. In other words, as the drum heads reach the beginning of each sector, the CCW of this sector is accessed from the COM memory and the data transfer, if called for, is initiated right away. After the initiation, the current CCW is of no further use; this COM memory location can now be refilled with the pertinent information for initiating the next page transfer for the same sector which occurs when the drum completes another revolution and again begins to scan this sector. This refilling is accomplished as follows. While a page is being transferred to or from a drum sector, the next page descriptor is detached from the sector list in the PAGETABLE memory and the pertinent information of the page obtained from the page descriptor is used to construct a CCW for the current drum sector. This CCW will be used after one drum revolution.

There are 16 CCW's in the COM memory. Thus, there can be as many as 16 pages in the MM that are not linked at all by the page descriptors

in the PAGETABLE memory, though they are pointed by (and thus indirectly linked by) the CCW's as mentioned above. In fact, the 16 pairs of list-heads in the LISTS memory are pointers which point to the second MM pages of these 16 sector lists in the PAGETABLE memory.

2.3.3 The Page-transfer Requests

As mentioned previously, the swappable-page list links those pages in the main memory that are released by the CPU to be transferred to the drum, while each of the sector lists links those MM pages queued to be paged-in or paged-out. The reason why two kinds of lists are required is that the computer operating system has to schedule alternately the read and write page transfers in the sector lists.

Fig. 16 is a block diagram showing the flow of page transfer requests. The 16 sector lists are maintained by the PDC; the available-page list and the swappable-page list are maintained by the CPU. When a page is requested by the CPU and found missing in the main memory, a page-fault interrupt is generated; this interrupt signifies that a new page is to be paged-in. The CPU allocates a page descriptor from the available-page list and posts a read page-transfer request to the PDC. The PDC responds by placing the request in the appropriate sector list for the drum sector where the page is stored. The CPU next posts a write page-transfer request to the PDC for swapping out a page from the swapping-page list. In this manner, the read and write page transfers are scheduled alternately, as also indicated in Fig. 16.

Fig. 17 is a block diagram showing handling of the lists. As mentioned before, the CPU posts a read page-transfer request owing to a page fault or posts a write page-transfer request owing to a swapping. The

PDC queues these requests in the appropriate sector lists. Whenever a page transfer is completed, the PDC notifies the CPU by means of the signals in registers PTRAN and PAGINT. The CPU then makes a note in the users' lists and resumes the execution of a previous program, or it attaches the page descriptor now not needed to the available list. The CPU posts page-transfers one after another, while the PDC initiates and completes the transfers in an order which optimizes the drum transfer operation. Note that the CPU handles all the lists including the users' lists except the sector lists which is handled by the PDC. Manipulations of the available-page list, the swappable-page list, and the users' list are not included in this simulation. Only the manipulation of the 16 sector lists by the PDC is included in the simulation.

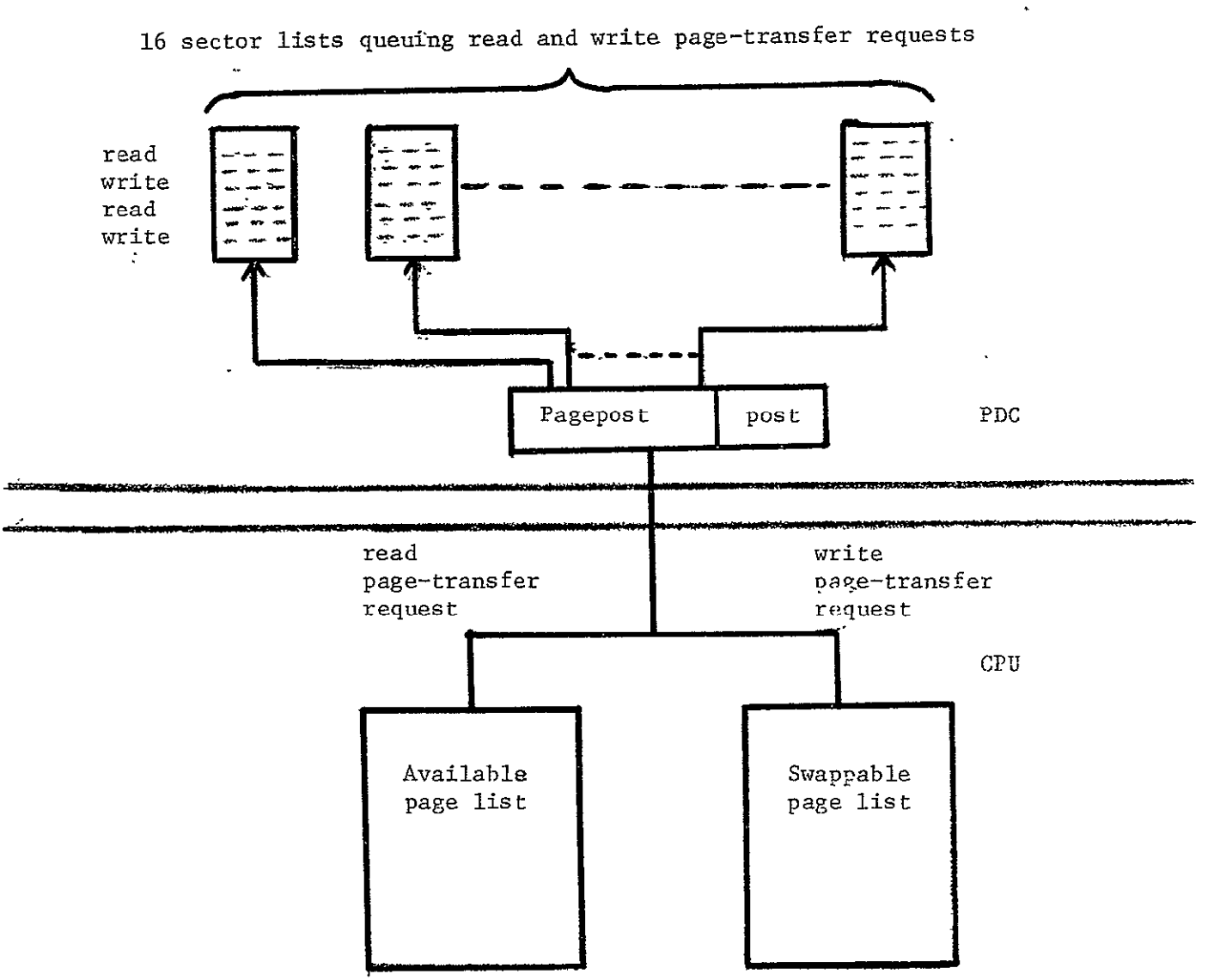


Fig. 16 Block Diagram Showing the Read and Write Page-Transfer Requests

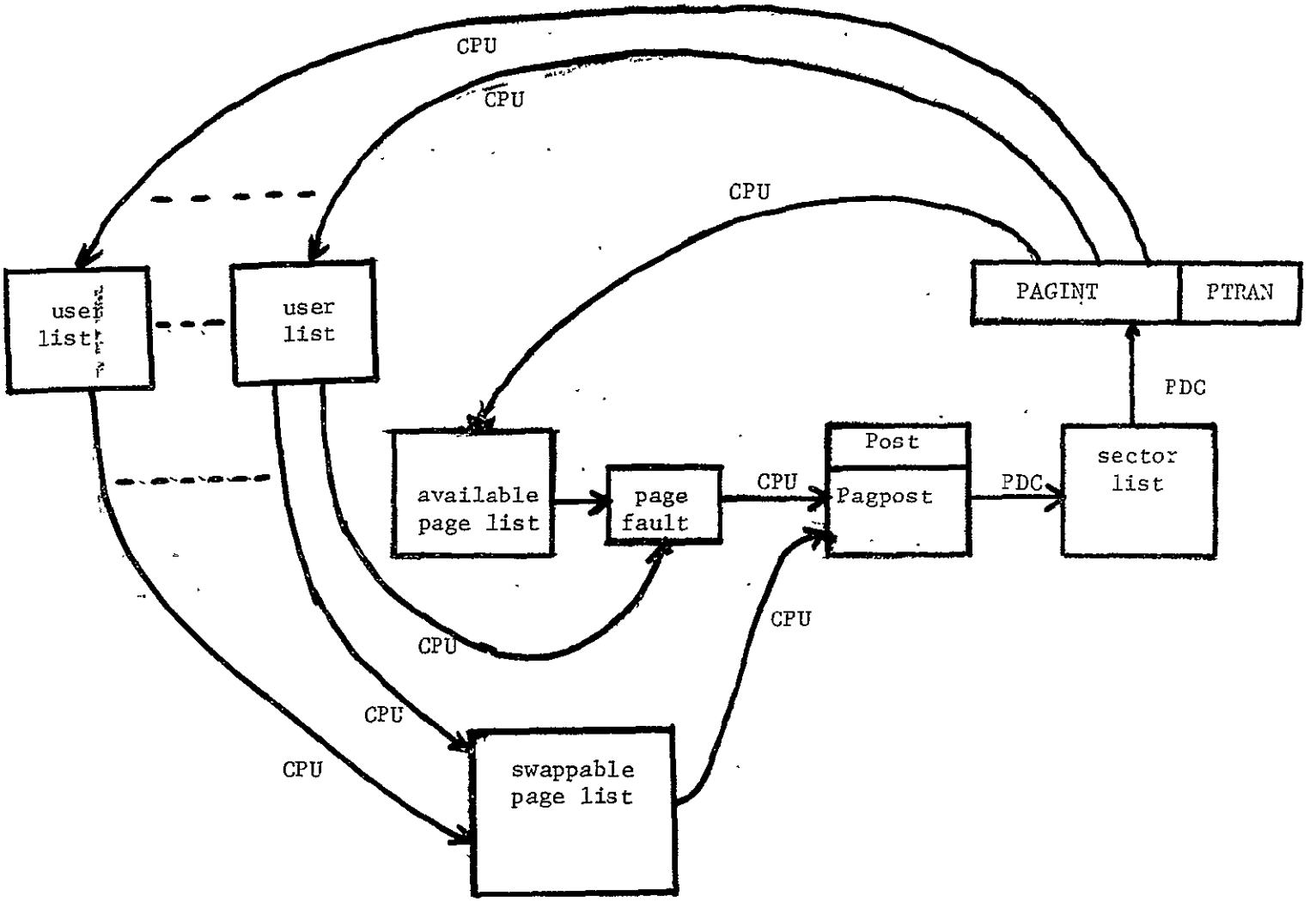


Fig. 17 Block Diagram showing handling of the Lists

2.4 The paging sequence

The paging sequence of the PDC is first described by a flow chart in Fig. 18 and then by sequence charts in Figs. 19 through 22. As shown in Fig. 18, when the PDC is initiated, there is one single process. Later at FORK, this single process splits into two parallel processes. The left-hand process transfers data between the PDRUM and the MM, and signals the CPU by setting appropriate bits in the CPU interrupt register INTERRUPT when a page transfer is completed. The right-hand process updates the channel command words CCW's in the channel command memory, the listheads of the sector queues in the listheads memory, and the page descriptors in the PTM. The right-hand process when it is completed merges with the left-hand process at JOINT, if the drum heads are at the beginning of a drum sector. But, if the drum heads are inside of a sector, the PDC is free to test if the requested page has been found and then updates the page descriptors and listheads. If not found, the PDC remains in a wait loop.

2.4.1 Main sequence

The paging sequence may be considered as a main sequence together with the drum read/write subsequence, the updating subsequence, and the request accept subsequence. The sequence chart of the main sequence with the parallel processes is shown in Fig. 19. When PDC is initiated, the page interrupt register PAGEI is tested for 1. The PDC waits until PAGEI is set to 1 which indicates that the drum is at the beginning of a page. The PDC acknowledges it by setting PAGEI to 0. The drum sector address in the subregister CWORD(SECT) is put into the command memory address register SEC to fetch the corresponding CCW. When the next drum page is ready to be transferred, the word count in register COUNT is set to 0 and the CCW is

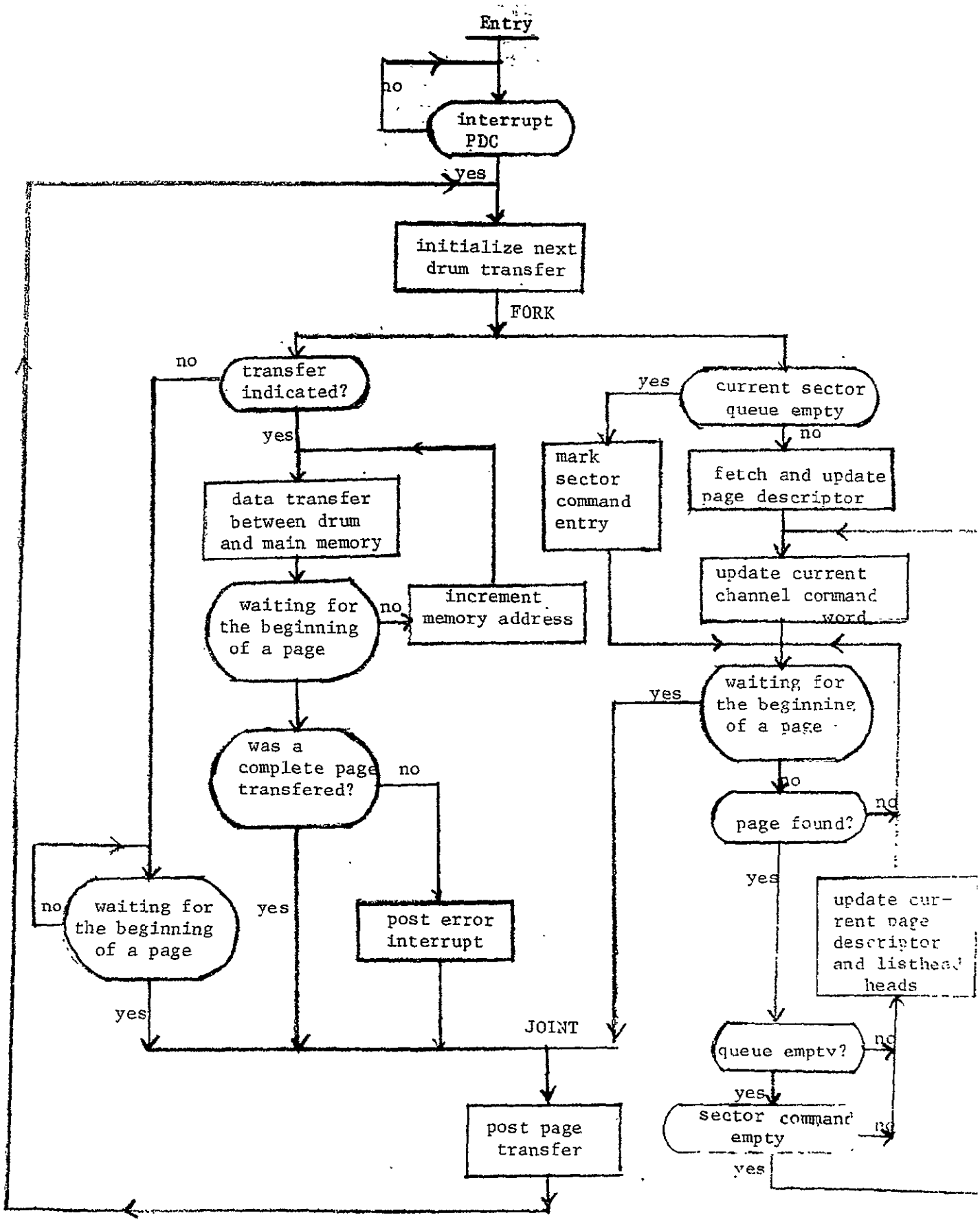


Fig. 18 Flow Chart of the Paging Sequence of the PDC

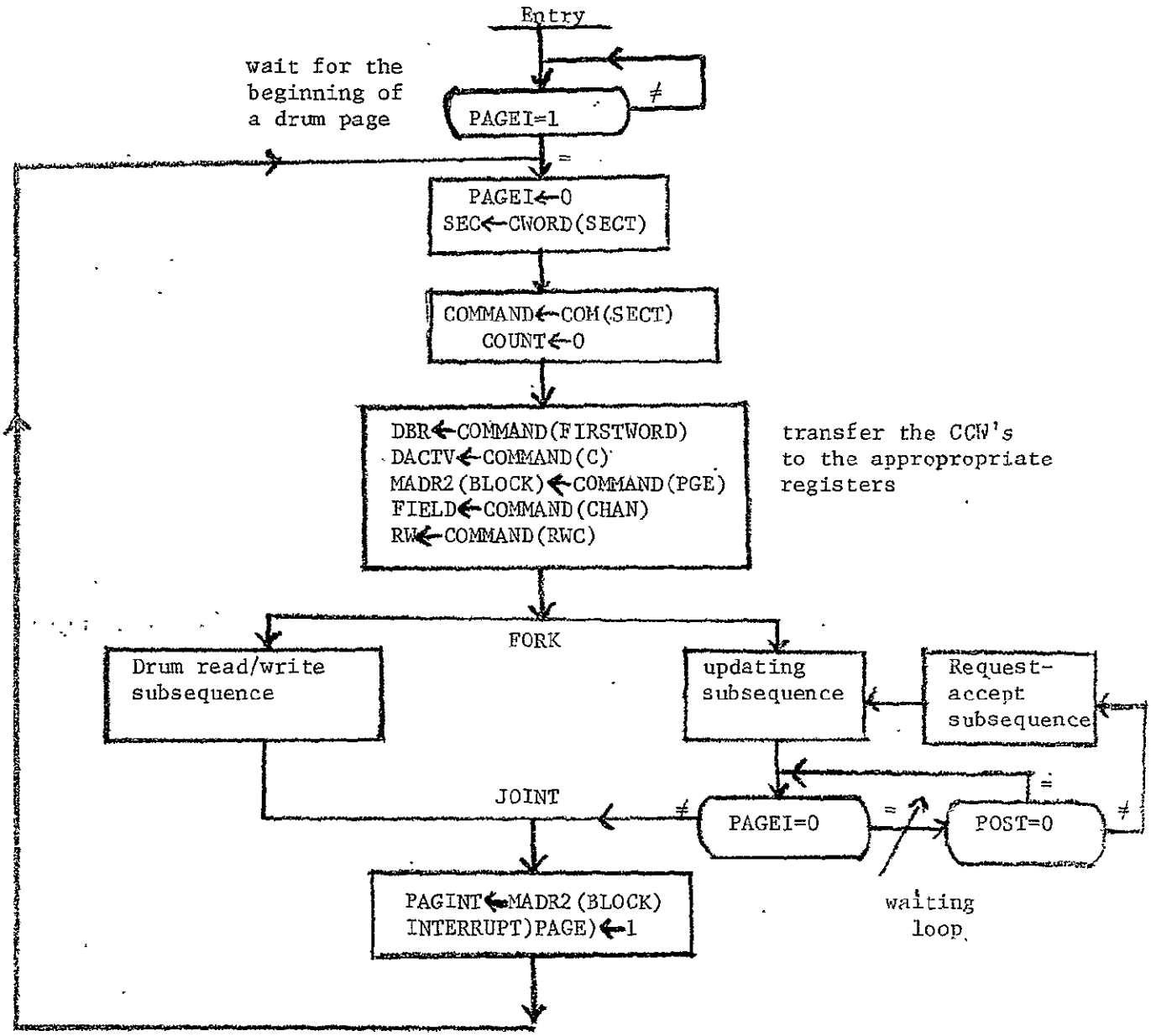


Fig. 19 Main Sequence With Parallel Processes

transferred into the command memory buffer register COMMAND. The PDC then decodes the control information in the CCW and transfers them to appropriate registers as follows. FIRSTWORD enters the drum buffer register DBR (writing 1 word ahead); C enters DACTV; PGE enters subregister MADR2(BLOCK); CHAN enters register FIELD; and RWC enters RW. Then the main process splits at FORK into the drum read/write subsequence and the updating subsequence. At the end of the updating subsequence, register PAGEI is tested. If PAGEI is 1, then the two subsequences joints into a single process. If PAGEI is 0, register POST is next examined. If POST is 0, the PDC loops back to check register PAGEI again. If POST is 1, the PDC enters the request-accept subsequence and then enters the updating subsequence again. Finally, the MM page address is put into register PAGINT and the 4th bit of the register INTERRUPT is set to 1. Now the PDC is ready for a new request.

2.4.2 Drum read/write subsequence

The drum read/write subsequence is shown in Fig. 20. When this subsequence is initiated, either there is a page transfer or there is no page transfer between the MM and the PDRUM through the PDC. The sequential operation starts by testing DACTV. If DACTV is 0, the PDC tests PAGEI for 1, and it waits until PAGEI is set to 1 by the paging drum. If PAGEI is 1, then PTRAN is set to 0 since no page has been transferred. If DACTV is 1, the read/write control register RW is next examined.

When RW is 1, it is the output sequence; a page is paged-out. The PDC waits until MA(2) is set to 0 by the MM. Output indicator enters RW2; word count enters MADR2(WRD); and MA(2) is set to 1 to make a transfer request to the MM. The PDC is held waiting until a word is transferred from the MM to the auxiliary MM buffer register SBR2 in the PDC and MA(2)

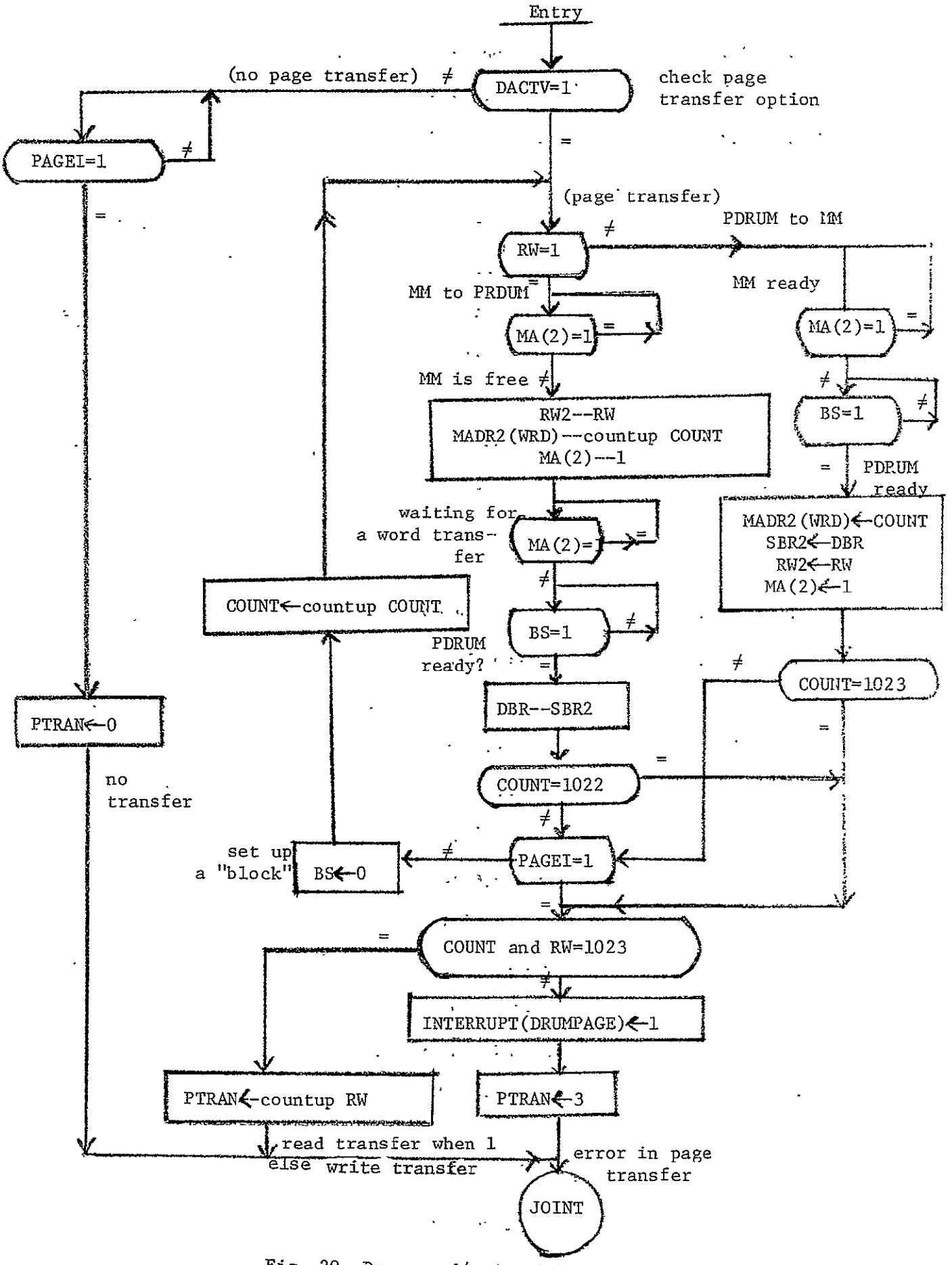


Fig. 20 Drum read/write Subsequence

is cleared by the MM. Register BS is next tested for 1 and if BS is 1, the drum buffer register DBR is ready and a data word is transferred from SBR2 into DBR. Next, the word count COUNT is examined to see if it is 1022; if so, a whole page has been transferred. If COUNT is not 1022, PAGEI is checked. If PAGEI is 1 and the whole page has not been paged-out, the PDC sets INTERRUPT(10) to 1 and indicates error condition by setting PTRAN to 3. If PAGEI is 0, and the whole page has not been paged out, BS is set to 0 and the word count is stepped up. If COUNT is 1022, then a page has been paged-out. Next, the sum of COUNT and ROW is compared with 1023 to make sure that there is no error in paging. If the sum is 1023, PTRAN is set to 0. The PDC goes to JOINT.

When RW is 0, it is the input sequence. A page is to be paged-in. The PDC waits until MA(2) is set to 0 by the MM. If MA(2) is 0 and BS is 1, a data word is transferred from DBR to SBR2. Next, input command enters RW2; word count enters subregister MADR2(WRD) to form the main memory address; and MA(2) to 1 to make a transfer request to the MM. The PDC waits until the word is transferred into the main memory. Next, the word COUNT is examined to see if it is 1023; if it is 1023, a whole page has been paged-in. If COUNT is not 1023, PAGEI is next tested, If PAGEI is 1 and the page has not been paged-in, the PDC generates an interrupt as before to indicate an error condition. If PAGEI is 0 and page transfer is not yet complete, BS is set to 0 for a word transfer and the word count is incremented by 1 so that next word of the page may be transferred. If COUNT is 1023, the sum of COUNT and ROW is compared with 1023. If the sum is 1023, PTRAN is set to 1. The PDC goes to JOINT.

2.4.3 Updating subsequence

The updating subsequence as shown in Fig. 21 updates the channel

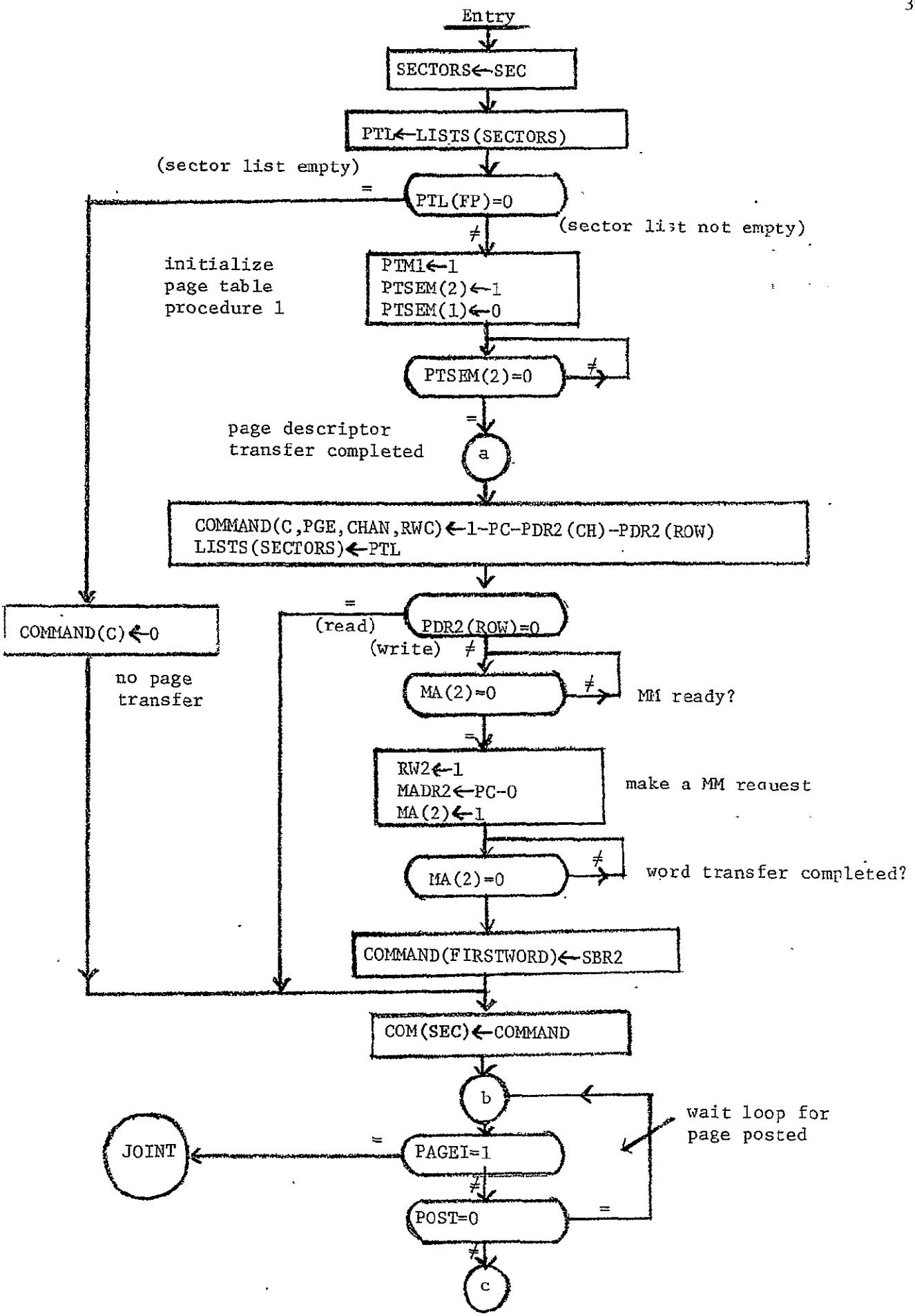


Fig. 21 Updating Subsequence for 3 Memories

command memory. The listheads memory, and the page table memory according to the current channel command. The updating subsequence for the 3 memories starts by entering the drum sector address into the listheads memory address register SECTORS to fetch listheads for the sector queue. If a sector queue in the PTM is empty, the sector command is marked "empty" and the marked CCW is returned to the channel command memory. If the sector queue in the PTM is not empty, then procedure 1 is called in by setting PTM 1 to 1 to make the front pointer of the sector queue point to the current page descriptor. The modified listheads are returned to the listhead memory. The subfields of a CCW are modified as follows: the C field is set to 1; the PGE field is set to PC; the CHAN field is set to CH field of the page descriptor; the RWC field is set to the ROW field of the page descriptor.

Next, the ROW field of the page descriptor in register PDR2 is examined. If PDR2(ROW) is 0, it is the read operation and the FIRSTWORD field of the CCW is not modified. When PDR2(ROW) is 1, it is the write operation, the firstword of the page transfers from the MM buffer register into the FIRSTWORD field of the CCW. For either read or write, the CCW is put back into the channel command memory. If PAGEI is 1 at this point, control goes to JOINT: else, if PAGEI is 0, register POST is tested. If POST is 0, then the requested page is not yet posted, and it is necessary to wait by looping back to the previous micro-operation that checks PAGEI.

2.4.4 Request-accept subsequence

The request-accept subsequence as shown in Fig. 22 starts at point (C). When POST is set to 1, procedure 2 is called in by setting PTM2 to 1 to fetch the page descriptor of the posted page into PDR2. The sector queue number as indicated by the subregister PDR2(SEC) is

transferred to register SECTORS to fetch the next set of listheads.

If PTL(FP) is 0, the sector queue in the PTM is empty, and the CCW for this sector queue is unchanged. If the C field of the command is 0, then there is no page transfer. In this case, the PDC sets PC to PAGEPOST, sets POST to 0 and loops back to take care of the CCW for the next drum sector. If PTL(FP) is not 0, the sector queue in the PTM corresponding to the current drum sector is not empty. Procedure 3 is activated by setting PTM3 to 1 to add the page descriptor addressed by PAGEPOST to the list of page descriptors addressed by PTL. The updated listheads PTL(FP) and PTL(LP) are then stored into the listhead memory and register POST is clear. The PDC waits until PAGEI is set to 1 and then goes to JOINT.

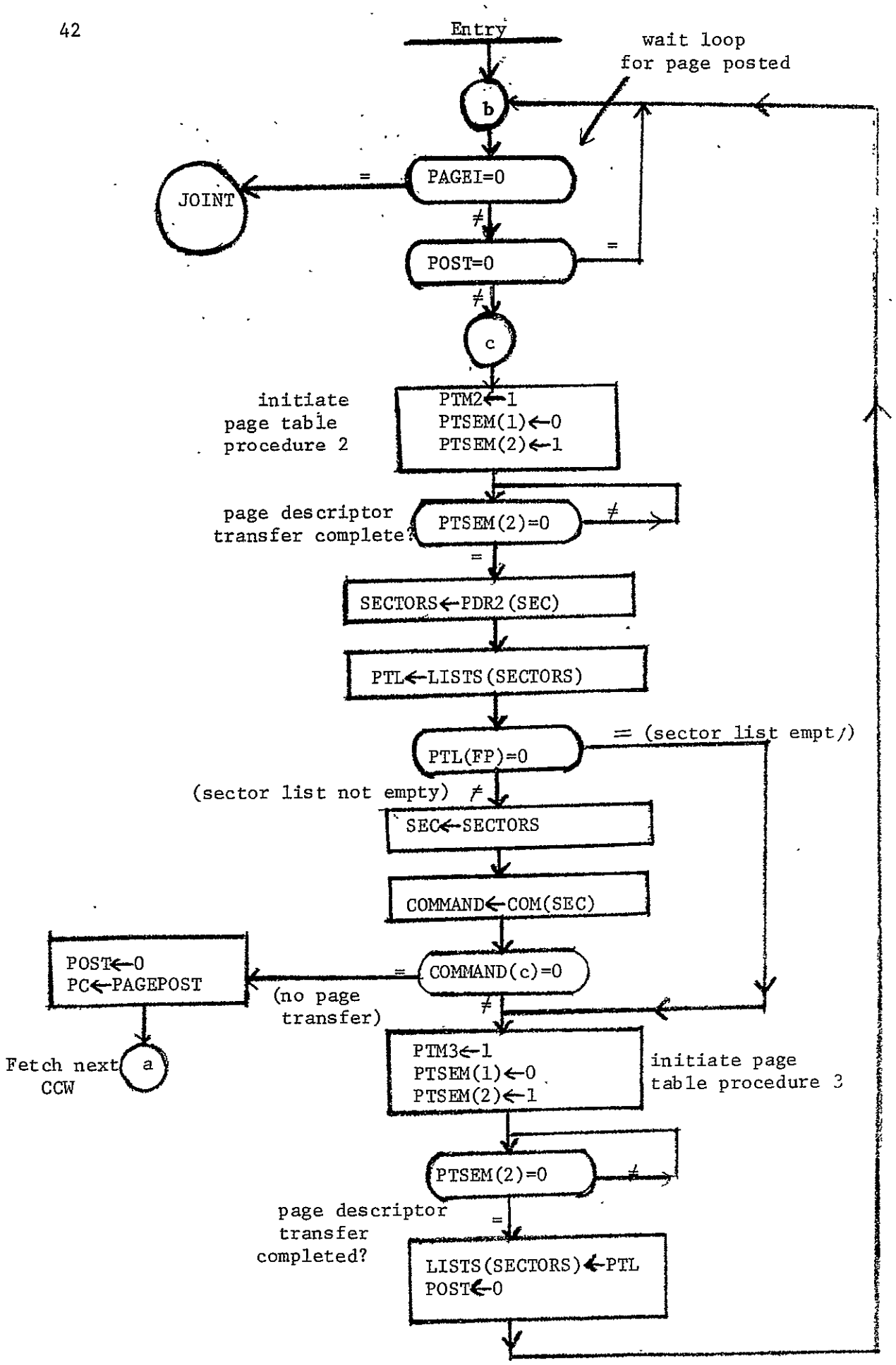


Fig. 22 Request-accept subsequence

3. An Example of Sector-Queue Manipulation

For each drum sector, there is a sector queue in the page-table memory. These sector queues are doubly linked lists of page-descriptors whose MM pages are to be transferred by the PDC. An example is now presented to show how the 16 sector queues are manipulated as the paging drum rotates. Four drum revolutions are traced through one drum sector at a time, and 48 MM pages are transferred by the PDC during these revolutions.

The MM page addresses lie between 1 through 63. Consecutive numbers are chosen for MM page addresses in order to make the example easier to understand; however, in actual operation the MM page address in a sector queue are not consecutive. It is assumed that, initially, the drum read/write head starts at drum sector 0, and the channel command memory is empty.

3.1 Initial linkage of the sector-queues in the PTM

The initial linkage of the 16 sector queues are shown by the arrowheads in Fig. 23. Let each sector queue have three nodes; each node represents a page descriptor and contains a number. This number is the MM page address which this page descriptor represents. There are 16 pairs of listheads for the 16 sector queues. The first page FP field contains the MM page address of the first page and the last page LP field contains the MM page address of the last page.

Corresponding to drum sector 0, the sector queue has node 1, node 2, node 3 while the listheads are 1 and 3. Corresponding to drum sector 1, the sector queue has node 4, node 5, node 6 while the listheads are 4 and 6. Similarly, corresponding to drum sector 15, the sector

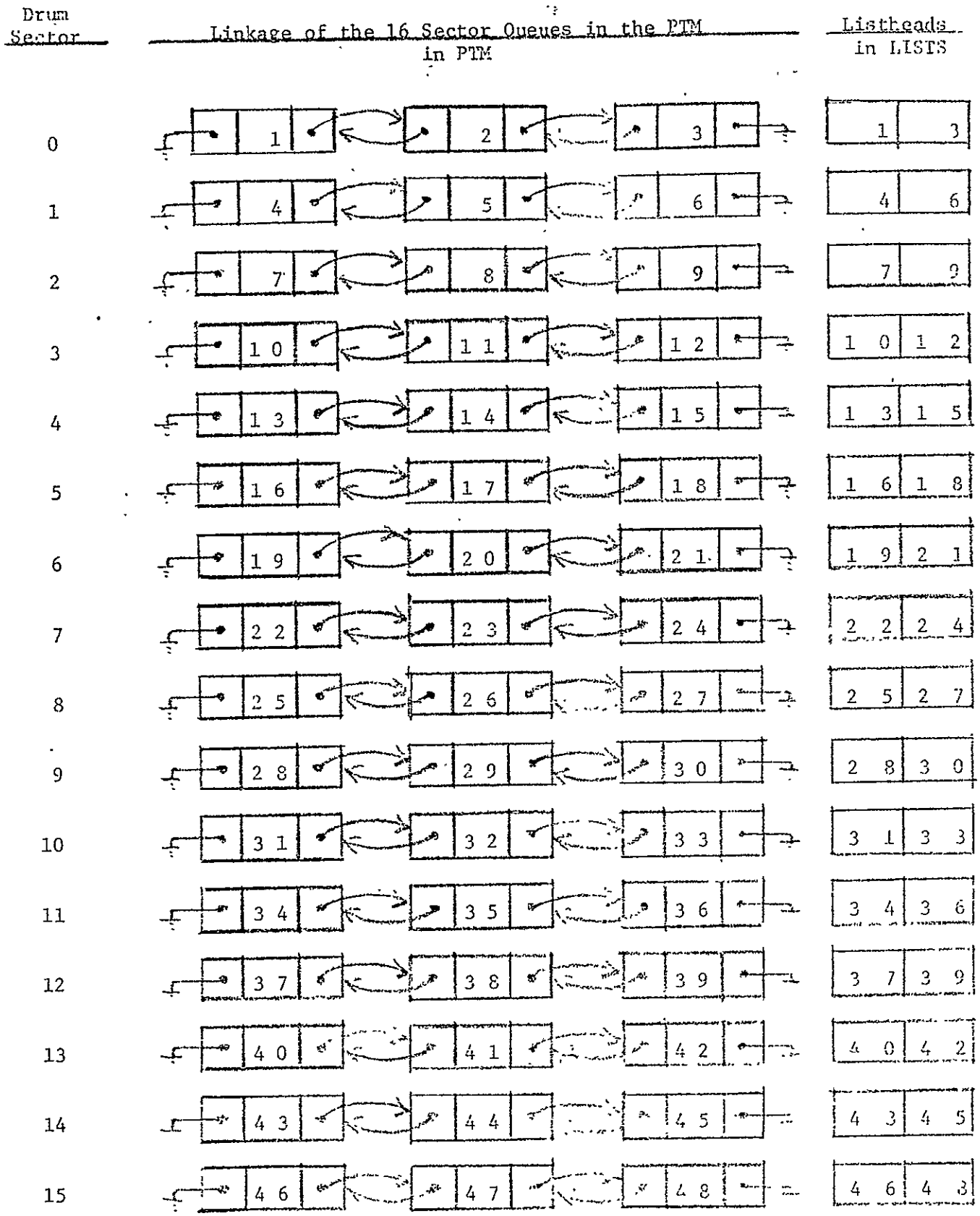


Fig. 23 Linkage of page descriptors which links 16 sector queues in the PTM

queue has node 46, node 47, node 48 while the listheads are 46 and 48.

The forward link LF and backward link LB of each sector queue are shown in Fig. 24. Corresponding to drum sector 0, the LB of node 1 is 0 (grounded) and the LF of node 1 is 2 (pointing to node 2); the LB of node 2 is 1 (pointing to node 1), and the LF of node 2 is 3 (pointing to node 3); the LB of node 3 is 2 (pointing to node 2), and the LF of node 3 is 0 (grounded). Corresponding to drum sector 1, the LB of node 4 is 0, and the LF of node 4 is 5 (pointing to node 5); the LB of node 5 is 4 (pointing to node 4), and the LF of node 5 is 6 (pointing to node 6); the LB of node 6 is 5 (pointing to node 5) and the LF of node 6 is 0. Similarly, corresponding to drum sector 15, the LB of node 46 is 0, and the LF of node 46 is 47; the LB of node 47 is 46 and the LF of node 47 is 48; the LB of node 48 is 47 and the LF of node 48 is 0.

3.2 Manipulation of Sector-Queues during the First Drum Revolution

The head of each sector queue will be deleted by the PDC to construct a channel command word for the coresponding drum sector. Thus, no page-transfer can occur during the first drum revolution. The manipulation of sector queues is shown in Fig. 25.

After 1/16 of a drum revolution, node 1 is removed and the channel command word CCW for drum sector 0 is constructed using the information in the page-descriptor. The head of the sector queue for drum sector 0 is deleted. Therefore, the FP of the listheads point to node 2 while the LP of the listheads is unchanged. After 2/16 of a drum revolution, the CCW for drum sector 1 is constructed according to the information in the page-descriptor which is node 4. The head of the sector queue for drum sector 1 is deleted. Therefore, the FP of the listheads points to

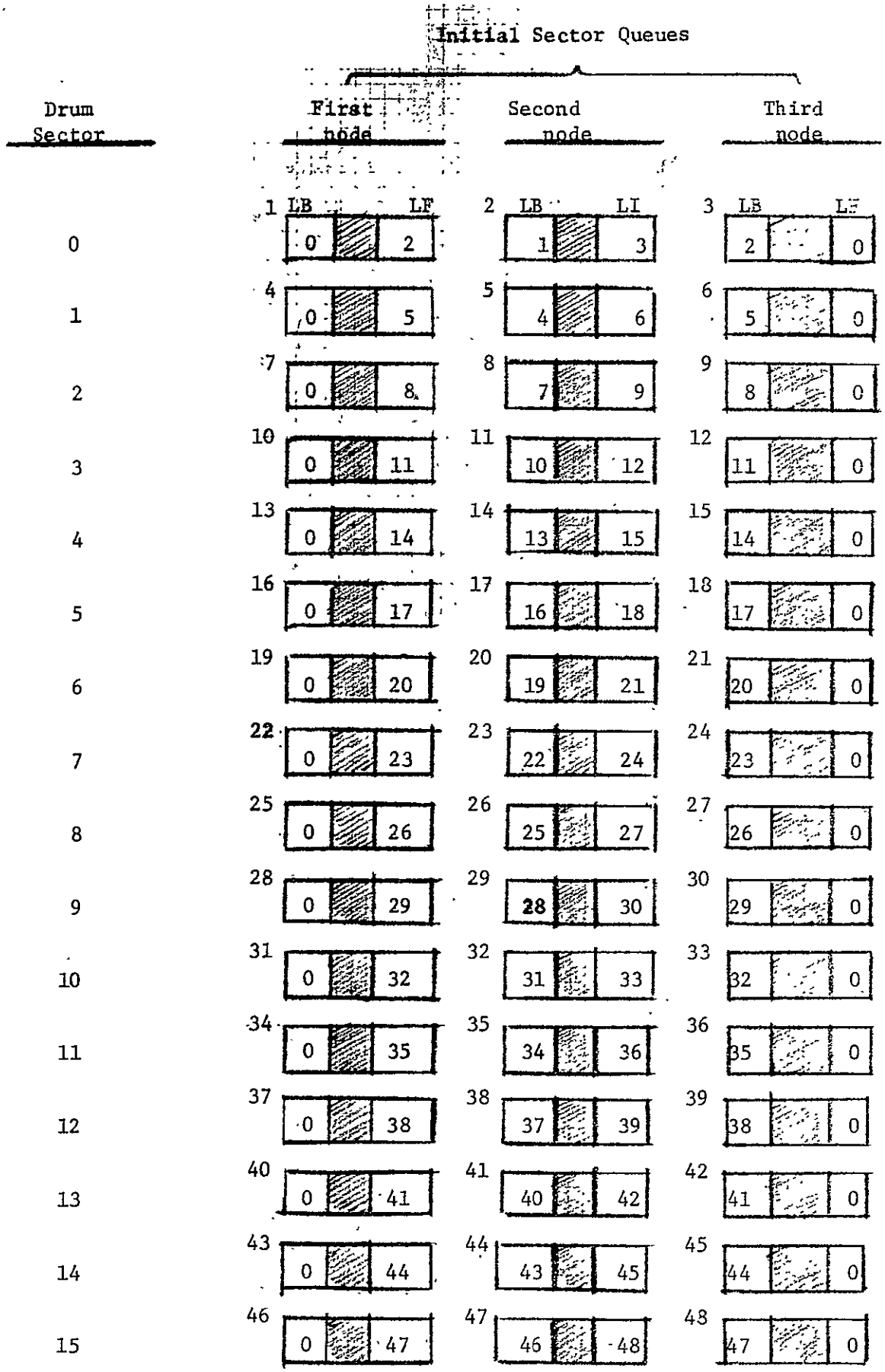
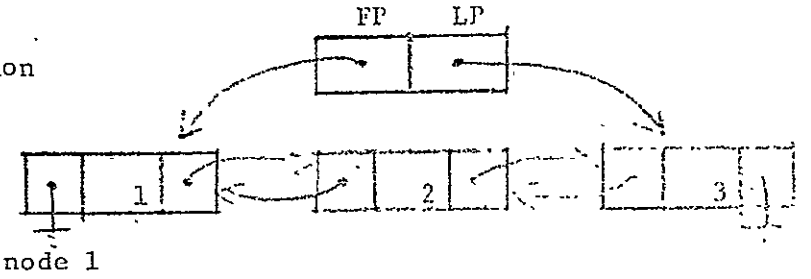


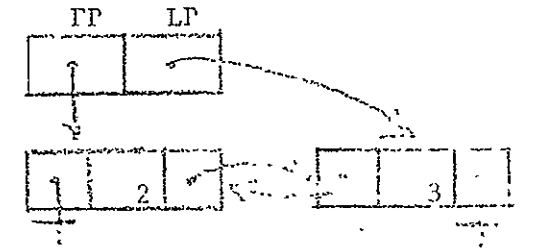
Fig. 24 Initial Forward and Backward Links in the Page Descriptors Which Form the 16 Sector Queues

Sector Queue 0

initial condition

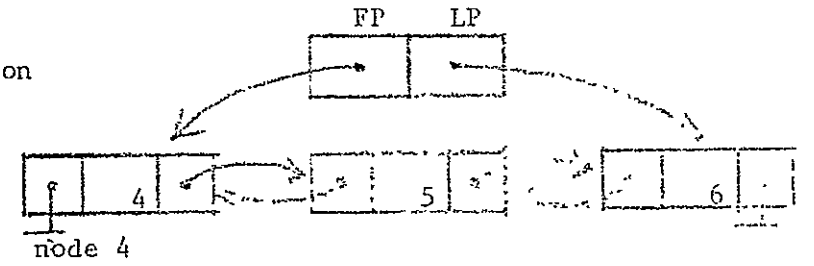


after 1/16 drum revolution

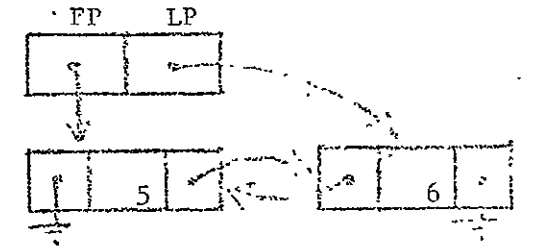


Sector Queue 1

initial condition

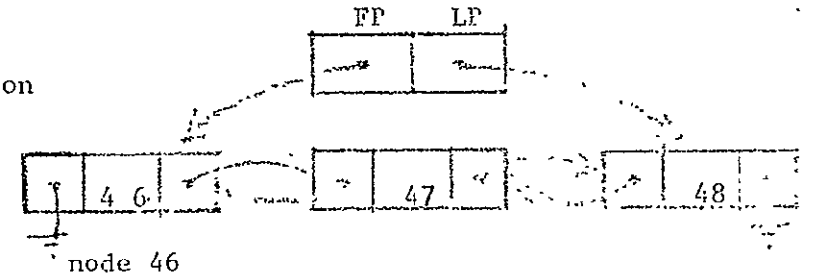


after 2/16 drum revolution



Sector Queue 15

initial condition



after one drum revolution

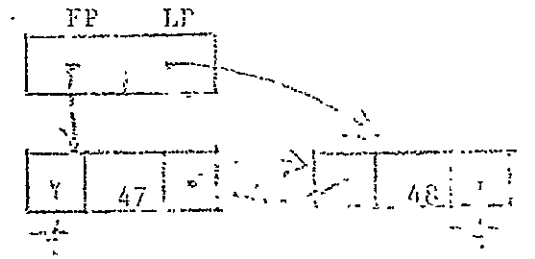


Fig. 25 Manipulation of the sector queue during the first drum revolution (only the first, the second and the last are shown)

node 5 while the LP of the listheads is unchanged. After 1 full drum revolution the CCW for drum sector 15 is constructed according to the information in the page descriptor which is node 46. The head of the sector queue for drum sector 15 is deleted. Therefore, the FP of the listheads points to node 47 while the LP of the listheads is unchanged.

Up to this point, the channel command memory is completely loaded with CCW's. The 16 sector queues after the first drum revolution are summarized in Fig. 26. The forward link LF and backward link LB of each sector queue are summarized in Fig. 27.

3.3 Manipulation of Sector-Queues during the Second Drum Revolution

Since the channel command memory is loaded with CCW's and the drum read/write heads are at drum sector 0, page-transfer can now begin. The manipulation of sector queues in the second drum revolution is presented in Fig. 28.

After 1-1/16 drum revolution, the MM page 1 has been transferred by the PDC, and a new CCW for drum sector 0 is constructed according to the page-descriptor which is node 2. Consequently, node 2 is deleted from the sector queue and the FP of the listheads points to node 3. After 1-2/16 drum revolution, the MM page 4 has been transferred by the PDC, and a new CCW for drum sector 1 is constructed according to the page-descriptor which is node 5. Consequently, node 5 is deleted from the sector queue and the FP of the listheads points to node 6. After 2 full drum revolutions, the MM page 46 has been transferred by the PDC and a new CCW for drum sector 15 is constructed according to the page-descriptor which is node 47. Consequently, node 47 is deleted from the

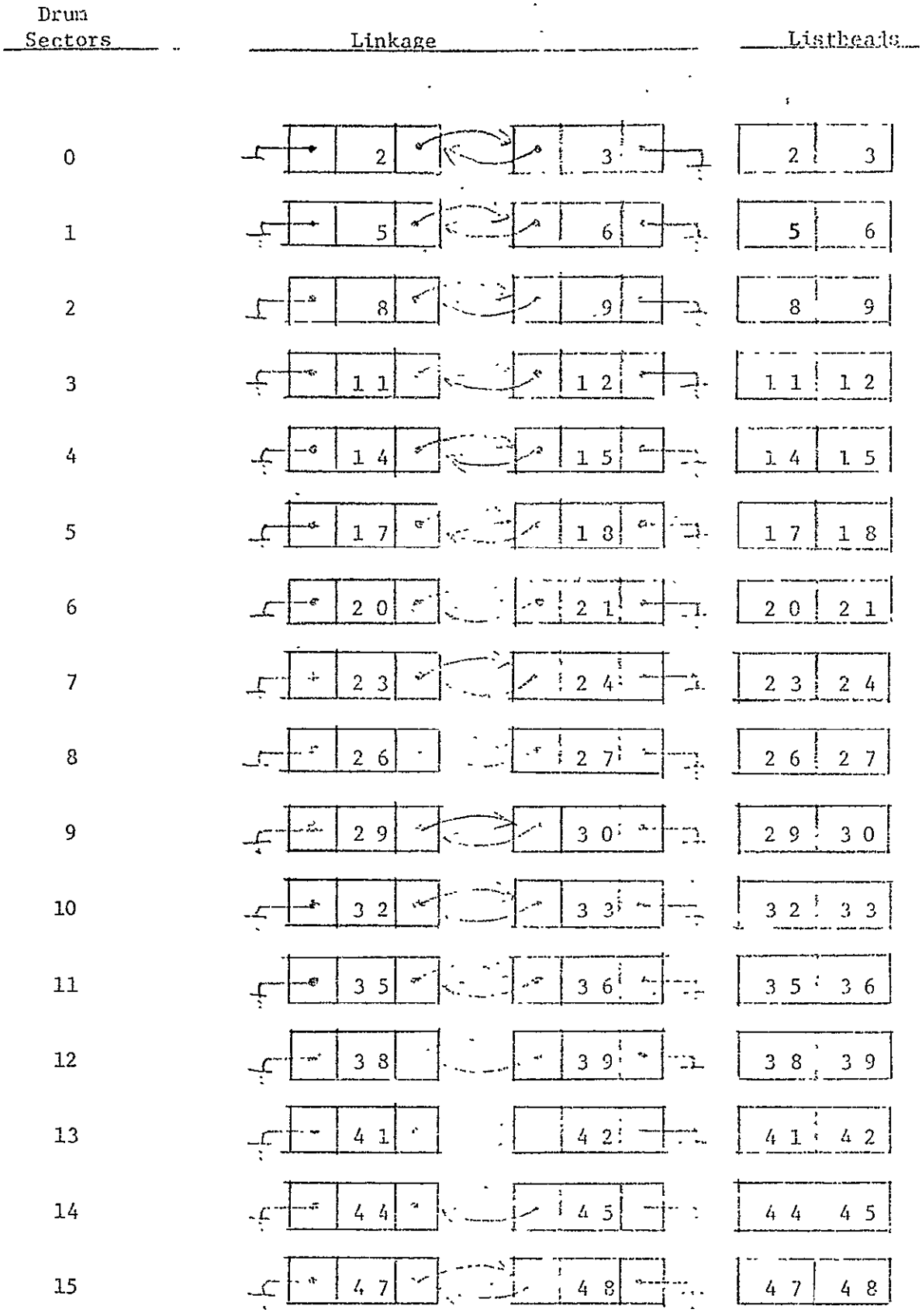


Fig. 26 Linkage of page descriptors which link the 16 Sector queues after one drum revolution

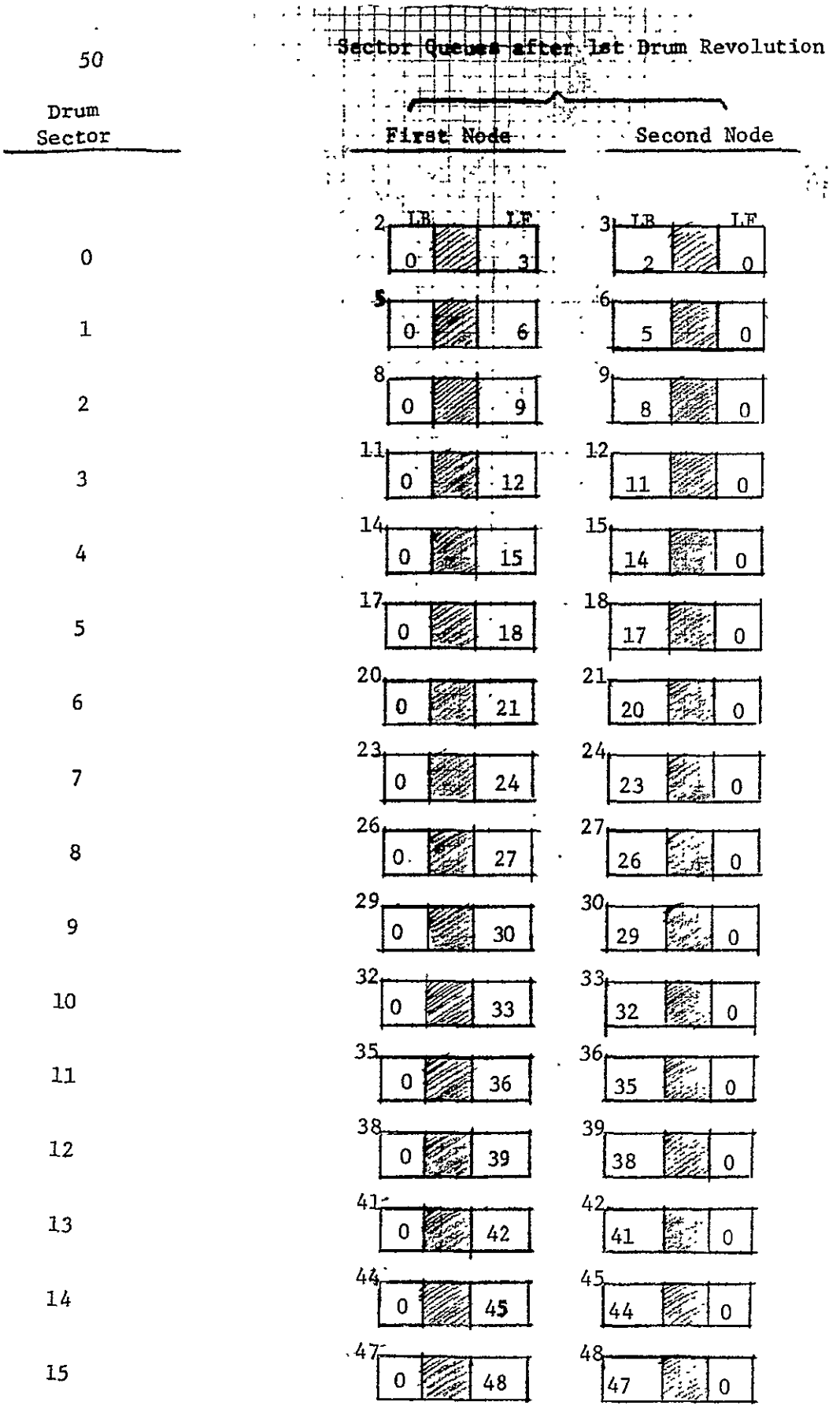
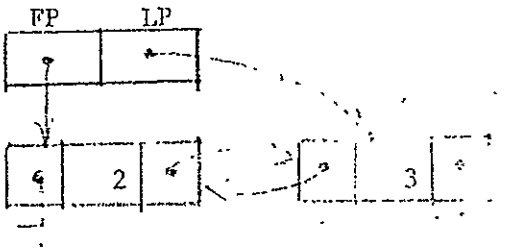


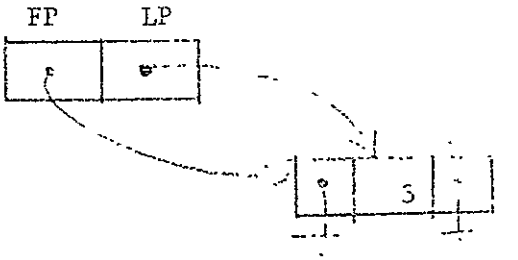
Fig. 27 Forward and backward links in the page descriptors which form the 16 sector queues after one drum revolution

Sector Queue 0

after 1/16 drum revolution

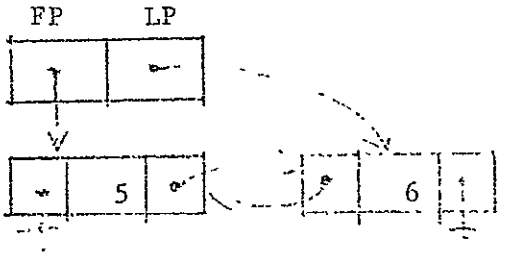


after 1-1/16 drum revolutions

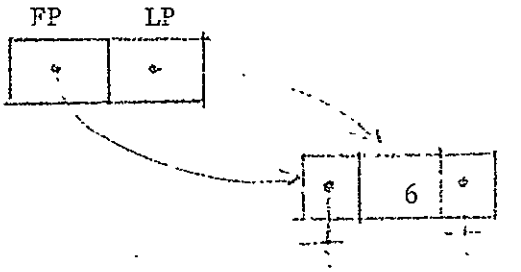


Sector Queue 1

after 2/16 drum revolution

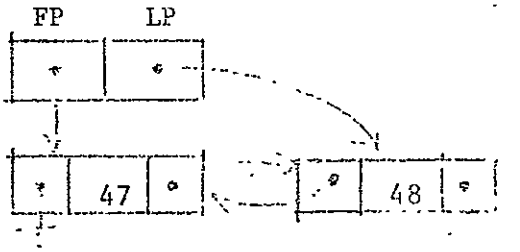


after 1-2/16 revolutions



Sector Queue 15

after 1 drum revolution



after 2 drum revolutions

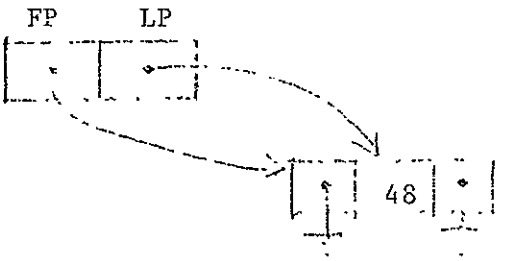


Fig. 28 Manipulation of Sector Queues 0, 1, and 15

sector queue and the FP of the listheads points to node 48.

Up to this point, MM pages 1, 4, 7, 10, 13, 16, 19, 22, 25, 28, 31, 34, 37, 40, 43, 46 have been transferred by the PDC and the channel command memory is again fully loaded with CCW's for transferring MM pages 2, 5, 8, 11, 14, 17, 20, 23, 26, 29, 32, 35, 38, 41, 44, 47. The 16 sector queues after the second drum revolution are summarized in Fig. 29. The forward link LF and backward LB of each sector queue are summarized in Fig. 30.

3.4 Manipulation of Sector-Queues during the 3rd Drum Revolution

During the 3rd drum revolution, MM pages 2, 5, 8, 11, 14, 17, 20, 23, 26, 29, 32, 35, 38, 41, 44, 47 are transferred by PDC, the channel command memory is again fully loaded with CCW's for transferring MM pages 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48. The 16 listheads are all grounded at the end of the 3rd drum revolution.

At this point, all the sector queues in the page table memory are empty as shown in Fig. 30. Therefore, there is no forward link or backward link for the empty queues. This fact is expressed by the word EMPTY in Fig. 31.

3.5 Page transfer during the 4th drum revolution

During the fourth drum revolution, MM page 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48 are transferred by the PDC. However, the channel command memory is completely empty at the end of the 4th drum revolution. There is no sector queue in the page table memory, and all the listheads are grounded. Hence, during the next drum revolution, no page-transfer occurs.

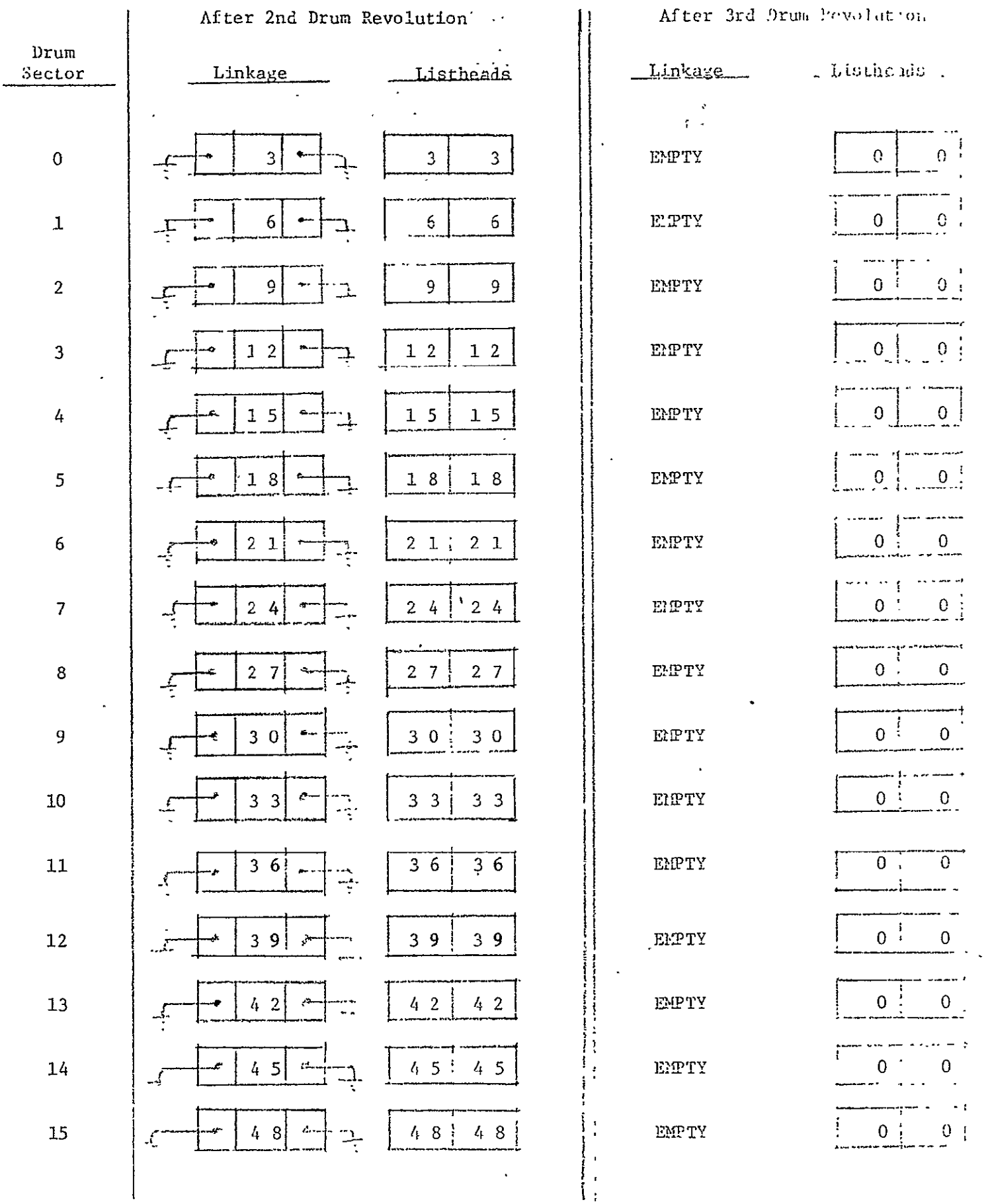


Fig. 29 Linkage of page descriptors which links 16 sector queues after the two and three drum revolutions

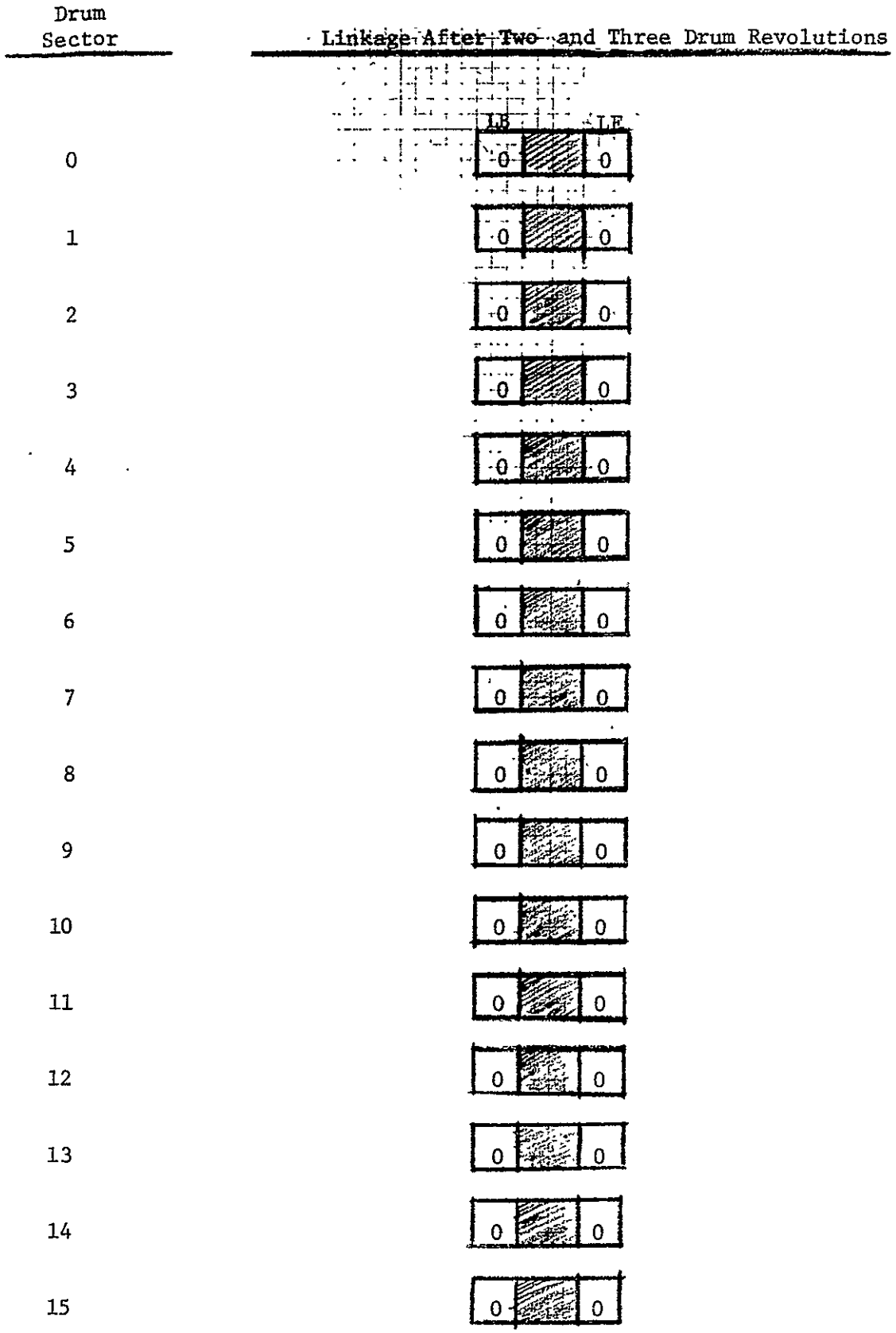


Fig. 30 Forward and backward links in the page descriptors which form the sector queues after two and three drum revolutions

Sector Queue	Status of Sector Queues	Number of Drum Revolutions at which the queue is e
0	EMPTY	2-1/16
1	EMPTY	2-2/16
2	EMPTY	2-3/16
3	EMPTY	2-4/16
4	EMPTY	2-5/16
5	EMPTY	2-6/16
6	EMPTY	2-7/16
7	EMPTY	2-8/16
8	EMPTY	2-9/16
9	EMPTY	2-10/16
10	EMPTY	2-11/16
11	EMPTY	2-12/16
12	EMPTY	2-13/16
13	EMPTY	2-14/16
14	EMPTY	2-15/16
15	EMPTY	3

Fig. 31 Emptying the sector queues during the third drum revolution

4. Simulation by Algol

This simulation is to test the correct operations of the PDC. The algorithm for the paging sequence presented in section 2.4 are now described by an Algol program; this program is shown in Appendix A.

In order to restrict the program size for simulation on the UNIVAC 1108, it is assumed that there are 8 words per main memory page. Initially, the main memory pages are loaded with data to be transfered to the empty paging drum memory.

There are two sets of test data: the first set is for paging-out 8 MM pages, and the second set is for paging-in 8 MM pages. They are shown in Tables 2-7. The output from the simulation is shown in Table 8 and 9. The Algol program is summarized in the flow chart of Fig. 34.

4.1 Simulation inputs

Two sets of test data are chosen for simulation. Each set contains three elements: 8 page descriptors, 8 listheads, and 8 CCW's. These test data deal with 8 MM pages, 32 through 39. The initial contents of these MM pages are shown in Table 2. In Table 2, page 32 contains 8 words with each word set to 32, page 33 consists of 8 words with each word set to 33, etc.

With test data 1, we shall page-out 8 MM pages to Field 1 of the drum as shown in Fig. 32. In this figure, page 32 goes through the PDC and becomes the first drum page of sector 0; page 33 goes through the PDC and becomes the first drum page of sector 1; and page 39 goes through the PDC and becomes the first drum page of sector 7. The contents of the 8 page descriptors are shown in Table 3. The ROW fields in the page descriptors are set to 1 to indicate paging-out. Since only 8 pages are going into 8 drum sectors, the backward link and the forward link of every page descriptor are set to 0 to indicate

Table 2 Test data 1: 8 Main Memory Pages

MM page number	MM word number	Contents of words	MM page number	MM word number	Contents of words
32	0	32	36	0	36
	1	32		1	36
	2	32		2	36
	3	32		3	36
	4	32		4	36
	5	32		5	36
	6	32		6	36
	7	32		7	36
33	0	33	37	0	37
	1	33		1	37
	2	33		2	37
	3	33		3	37
	4	33		4	37
	5	33		5	37
	6	33		6	37
	7	33		7	37
34	0	34	38	0	38
	1	34		1	38
	2	34		2	38
	3	34		3	38
	4	34		4	38
	5	34		5	38
	6	34		6	38
	7	34		7	38
35	0	35	39	0	39
	1	35		1	39
	2	35		2	39
	3	35		3	39
	4	35		4	39
	5	35		5	39
	6	35		6	39
	7	35		7	39

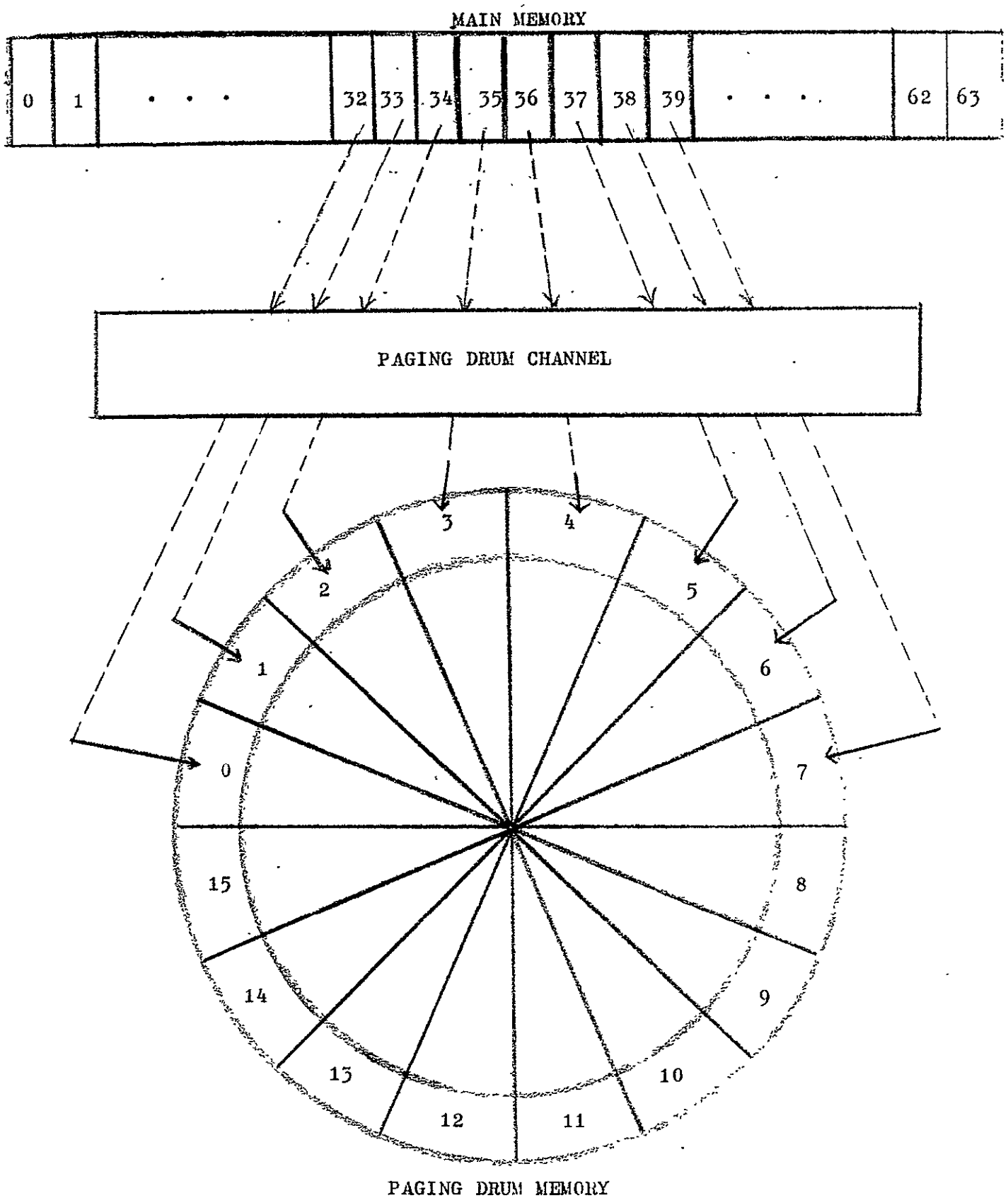


Figure 32 Paging-out 8 pages from main memory into 8 drum sectors (Test data 1)

Table 3 Page Descriptors for Test Data 1:
(for paging-out 8 MM pages to 8 drum sectors)

PTM address*	Field (in binary)			
	LB	LF	DP	ROW
32	0	0	0000010000	1
33	0	0	0000010001	1
34	0	0	0000010010	1
35	0	0	0000010011	1
36	0	0	0000010100	1
37	0	0	0000010101	1
38	0	0	0000010110	1
39	0	0	0000010111	1

* Decimal

Note: Other fields of the page descriptor are not needed and thus not shown.

Table 4. Channel Command Words* for Text Data 1:
(for paging-out 8 MM pages to 8 drum sectors)

Command memory address	Field				
	C	RWC	CHAN	PGE	FIRSTWORD
0	1	1	1	32	32
1	1	1	1	33	33
2	1	1	1	34	34
3	1	1	1	35	35
4	1	1	1	36	36
5	1	1	1	37	37
6	1	1	1	38	38
7	1	1	1	39	39
8 . . . 15	not used				

* Decimal Number

Table 5 Test Data 2: 8 Drum Sectors

(after 8 MM Pages are Paged to 8 Drum Sectors)

Drum Sector Number	Field Number	Word address	Contents of Word	Drum Sector Number	Field Number	Word address	Contents of Word
0	1	0	32	4	1	0	36
0	1	1	32	4	1	1	36
0	1	2	32	4	1	2	36
0	1	3	32	4	1	3	36
0	1	4	32	4	1	4	36
0	1	5	32	4	1	5	36
0	1	6	32	4	1	6	36
0	1	7	32	4	1	7	36
1	1	0	33	5	1	0	37
1	1	1	33	5	1	1	37
1	1	2	33	5	1	2	37
1	1	3	33	5	1	3	37
1	1	4	33	5	1	4	37
1	1	5	33	5	1	5	37
1	1	6	33	5	1	6	37
1	1	7	33	5	1	7	37
2	1	0	34	6	1	0	38
2	1	1	34	6	1	1	38
2	1	2	34	6	1	2	38
2	1	3	34	6	1	3	38
2	1	4	34	6	1	4	38
2	1	5	34	6	1	5	38
2	1	6	34	6	1	6	38
2	1	7	34	6	1	7	38
3	1	0	35	7	1	0	39
3	1	1	35	7	1	1	39
3	1	2	35	7	1	2	39
3	1	3	35	7	1	3	39
3	1	4	35	7	1	4	39
3	1	5	35	7	1	5	39
3	1	6	35	7	1	6	39
3	1	7	35	7	1	7	39

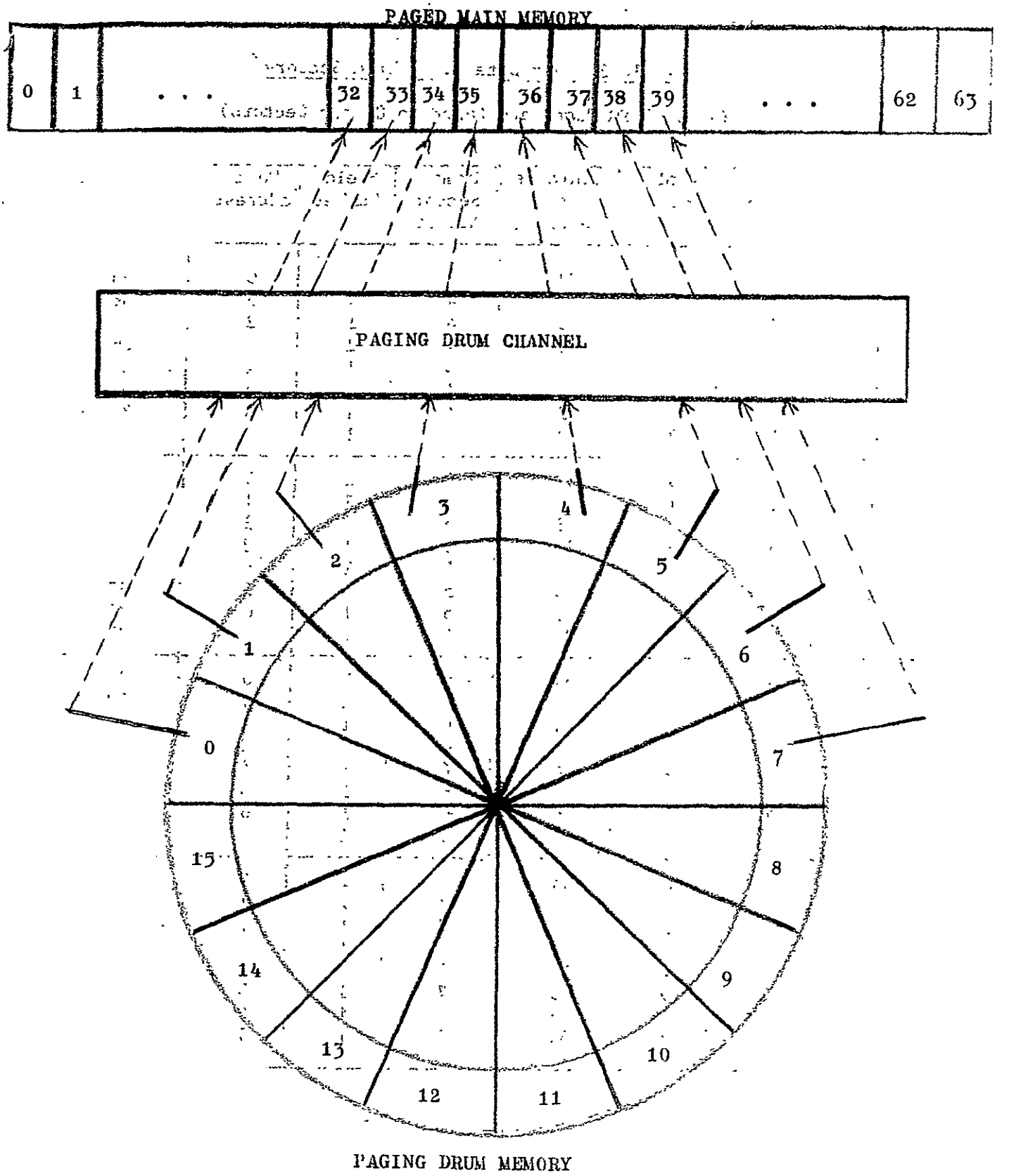


Fig.33 Paging-in 8 Pages into Main Memory from 8 sectors (test data 2)

that the 8 page descriptors are not linked.

The 10-bit DP field in the page descriptor specifies the drum page address. The DP field of the page descriptor for MM page 32 is 0000010000 in binary which represents field 1 of drum sector 0. The DP field of the page descriptor for MM page 39 is 0000010111 in binary which represents field 1 of drum sector 7. Thus MM page 32 goes to sector 0, MM page 33 goes to sector 1, and MM page 39 goes to sector 7.

The contents of the listheads of the sector queues in PTM are all set to 0 since the simulation model deals only with the heads of 8 sector queues assuming that the 8 CCW's for the corresponding 8 MM pages have already been constructed by the PDC.

The contents of the CCW's for paging-out 8 pages to 8 drum sectors are shown in Table 4. The C fields are all set to 1 to indicate that the 8 MM pages are to be transferred. The RWC fields are all set to 1 to indicate a paging-out operation. The CHAN fields are all set to 1 since only Field 1 is used for paging. The first MM page to be paged-out is page 32 whose first word is also 32. The second MM page to be paged-out is 33 whose first word is also 32. The second MM page to be paged-out is 33 whose first word is 33, etc. Thus, the PGE field and the FIRSTWORD field of each CCW contain the same number.

After the paging-out operations, the contents of drum sector 0 through 7 are tabulated in Table 5. As shown, in drum sector 1, there are eight 32's; in drum sector 2, there are eight 33's, etc. These eight drum pages are then paged-in by test data 2 to be discussed immediately.

With test data 2, we shall page-in 8 drum pages as shown in Fig. 33. In the figure, the first drum page of sector 0 goes through the PDC and becomes the 32nd MM page; the first drum page of sector 1 goes through the PDC and becomes the 33rd MM page; and the first drum page sector 7 goes through

the PDC and becomes the 39th MM page. The contents of the 8 page descriptors are shown in Table 6. The ROW fields are all set to 0 to indicate paging-in. The remaining bit setting of the page descriptors are the same as those of the paging-out case described in Table 4.

The contents of the listhead memory are again set to 0. The contents of the 8 CCW's for paging-in 8 MM pages are shown in Table 7 which is to be compared with Table 4. In Table 7, the RWC field of each CCW is set to 0 instead of 1. The rest of the contents of the CCW's in Table 7 remain the same as those in Table 4.

Table 6 Page Descriptors for Test Data 2:

(for paging-in 8 pages from 8 drum sectors)

PTM address*	Field (in binary)			
	LB	LF	DP	ROW
32	0	0	0000010000	0
33	0	0	0000010001	0
34	0	0	0000010010	0
35	0	0	0000010011	0
36	0	0	0000010100	0
37	0	0	0000010101	0
38	0	0	0000010110	0
39	0	0	0000010111	0

* Decimal

Table 7 Channel Command Words* for Test Data 2:

(for paging-in 8 pages from 8 drum sectors)

Command memory address	Field				
	C	RWC	CHAN	PAG	FIRSTWORD
0	1	0	1	32	32
1	1	0	1	33	33
2	1	0	1	34	34
3	1	0	1	35	35
4	1	0	1	36	36
5	1	0	1	37	37
6	1	0	1	38	38
7	1	0	1	39	39
8 . . . 15	not used				

4.2 Simulation Program

The Algol simulation program is described in the flow chart shown in Fig. 34. At the beginning, counter I is set to 1 and the initial contents of MM pages 32 through 39 are set as shown in Table 2. 8 page descriptors, 8 listheads, and 8 CCW's are initialized for paging-out the 8MM pages. These page descriptors, listheads, and CCW's are printed in octal before any page transfer. After the above initialization, the paging algorithm starts. The paging algorithm consists of the main sequence, the drum read/write subsequence, the updating subsequence, and the request-accept subsequence (see Figs. 19-22). As data words are transferred, the contents of important registers are printed. The paging algorithm is repeated 8 times for paging-out 8 MM pages.

After 8 MM pages have been transferred to the drum memory, their corresponding 8 page descriptors, 8 pairs of listheads, and 8 CCW's in octal are printed. Next, counter I is tested. Since I is 1, I is incremented by 1, and 8 page descriptors as well as 8 CCW's for paging-in 8 drum pages are initialized. The listheads are set to 0. Then the 8 page descriptors, the 8 listheads, and the 8 CCW's for paging-in 8 drum pages are printed in octal. The paging algorithm is employed 8 times for paging-in 8 drum pages. After 8 drum pages are paged-in, the 8 page descriptors, 8 pairs of listheads, and the 8 CCW's are printed in octal. Counter I is tested. Since now I is 2, the program terminates.

This simulation program is so designed that logic can flow through all parts of the program. Thus, some flipflops are set and reset whenever it is necessary to enter a loop or skip a loop.

An assembly language function DECODE is incorporated for extract-

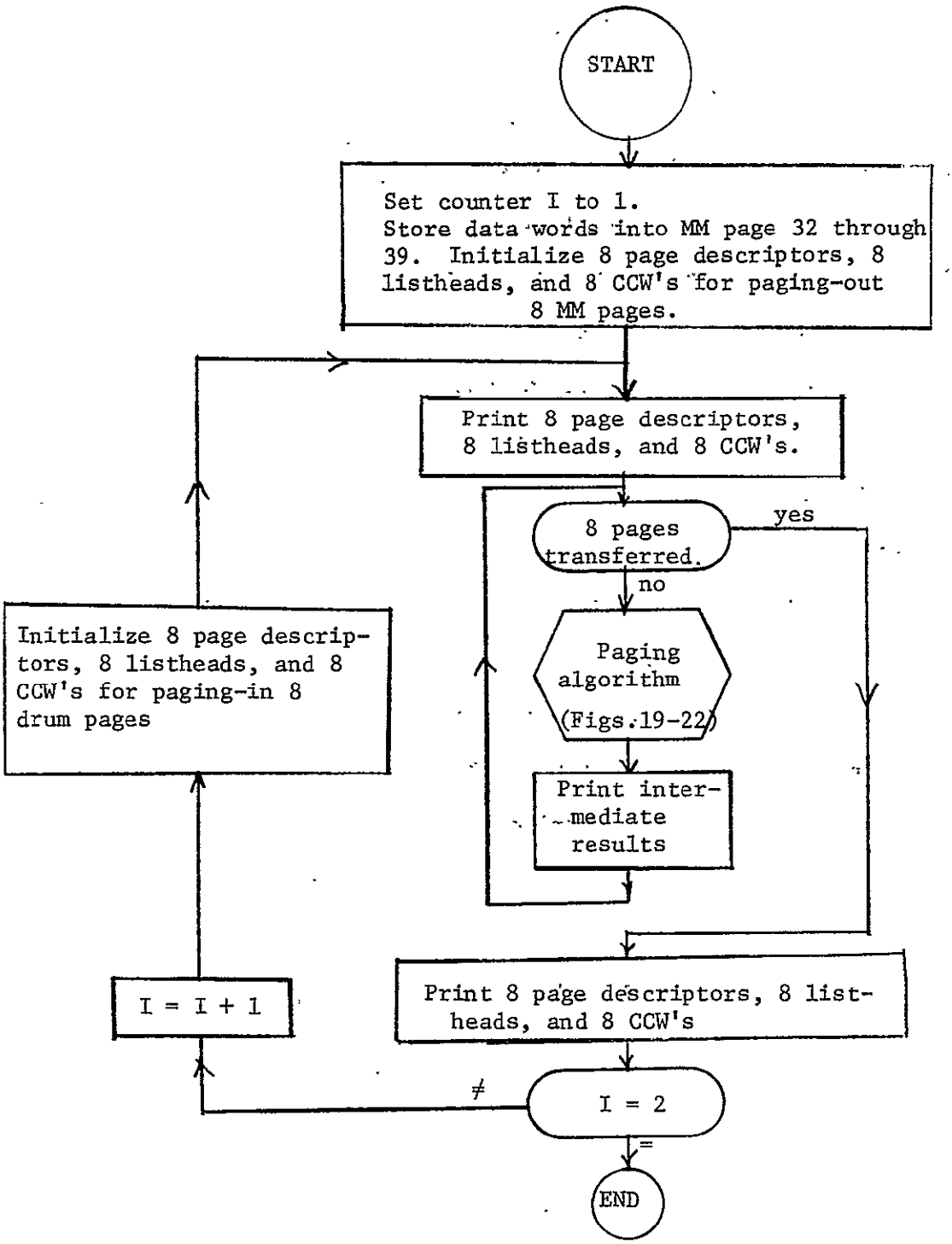


Fig. 34 Flow chart for the Algol Simulation Program

ing control information from the page descriptors, CCW's, and listheads. The calling sequence for DECODE is as follows:

FLDA(I,J,K),

where FLDA is the function name of the function, I is the starting position of the control information, J is the total number of bits to be extracted, and K is the 36-bit data word to be operated upon.

The following steps are taken by the function when it is called:

(a) Fetch the word K,

(b) Fetch the bit position I,

(c) Get rid of the upper bits by first shift left I bit positions and then shift right I bit positions,

(d) Compute: (36-I-J) which is the number of lower bits to be shifted,

(e) Get rid of the lower bits by shifting right (36-I-J) bit positions,

(f) Return to the Algol simulation program.

4.3 Simulation outputs

The output for the simulation is classified into 3 types as follows:

(a) After a word is transferred, print out contents of main memory buffer register, drum buffer register, word count in the page, and page transfer direction.

(b) After a page is transferred, print out the modified channel command word, listheads, drum field address, and drum sector address of the page.

(c) After 8 pages are transferred, print out channel command words, page descriptors, and listheads to check the number of pages in a drum sector there are, how the page descriptors and the CCW's are set.

The outputs by test data 1 for paging-out 8 MM pages to 8 drum sectors are summarized in Table 8-(a) through 8-(e). The output, after the first page (MM page 40_8) is transferred, is shown in Table 8-(a). The variables in the Algol program correspond to the registers of the PDC described in the sections 1 and 2. In Table 8-(a), PAGEPOST is 40_8. When a word of the first page is transferred, SBR2 is 40_8 and DBR is 40_8. COUNT keeps the word counter of the first page and it varies from 0 to 7. RW is 1 indicating a paging-out operation. SECTOR is 0 and FIELD is 1 so that the first page is transferred to drum sector 0 and field 1 of the sector.

After the first page is paged-out, the print out of the corresponding CCW, listheads for sector queue 0, the page transfer interrupt signal, and the page transfer status are summarized in Table 8-(b). Since a page descriptor requires two 36-bit computer word to represent, the page table memory PAGETABLE is represented by a two dimensional array. For the same reason, the channel command memory is also represented by a two-dimensional array. In Table 8-(b), COM(0,1) is $\underline{1401}40_8$ where the underlined portion $\underline{1401}_8$ is 1100000001 in binary which is interpreted as follows: (a) the C field of the CCW is 1 implying that there is a page transfer, (b) the RW field is 1 implying that there is a paging-out operation, (c) the drum field address is 1; and the remaining 40_8 not underlined is the MM page address of the current page COM(0,2) contains 40_8 which is the first actual word of the first page. The listheads of sector queue 0 are 40_8 and 40_8 and thus LISTS(0) is 4040_8. INTERRUPT(PAGE) is 1 indicating that page transfer is successfully completed. PTRAN is 2 indicating a paging-out operation.

The output, after a word of the last page (MM page 47_8), is shown in Table 8-(c). In this Table, PAGEPOST is 47_8. Each time when there is a word

Table 8 Test Data 1: Output in Octal
(for paging-out 8 main memory pages)

8-(a) Print out when the first page is being transferred

PAGEPOST	RW	SECTOR	FIELD	SBR2	DBR	COUNT
40	1	0	1	40	40	0
40	1	0	1	40	40	1
40	1	0	1	40	40	2
40	1	0	1	40	40	3
40	1	0	1	40	40	4
40	1	0	1	40	40	5
40	1	0	1	40	40	6
40	1	0	1	40	40	7

8-(b) Print out after the first page is transferred

COM(0,1)	COM(0,2)	LISTS(0)	INTERRUPT(PAGE)	PTRAN
140140	40	4040	1	2

8-(c) Print-out when the last page is being transferred

PAGEPOST	RW	SECTOR	FIELD	SBR2	DBR	COUNT
47	1	7	1	47	47	0
47	1	7	1	47	47	1
47	1	7	1	47	47	2
47	1	7	1	47	47	3
47	1	7	1	47	47	4
47	1	7	1	47	47	5
47	1	7	1	47	47	6
47	1	7	1	47	47	7

8-(d) Print-out after the last page is transferred

COM(7,1)	COM(7,2)	LISTS(7)	INTERRUPT(PAGE)	PTRAN
140147	47	4747	1	2

8-(e) Print-out after 8 pages are transferred

PAGETABLE(40,1)	0
PAGETABLE(41,1)	0
PAGETABLE(42,1)	0
PAGETABLE(43,1)	0
PAGETABLE(44,1)	0
PAGETABLE(45,1)	0
PAGETABLE(46,1)	0
PAGETABLE(47,1)	0

PAGETABLE(40,2)	4100
PAGETABLE(41,2)	4300
PAGETABLE(42,2)	4500
PAGETABLE(43,2)	4700
PAGETABLE(44,2)	5100
PAGETABLE(45,2)	5300
PAGETABLE(46,2)	5500
PAGETABLE(47,2)	5700

COM(0,1)	140140
COM(1,1)	140141
COM(2,1)	140142
COM(3,1)	140143
COM(4,1)	140144
COM(5,1)	140145
COM(6,1)	140146
COM(7,1)	140147

COM(0,2)	40
COM(1,2)	41
COM(2,2)	42
COM(3,2)	43
COM(4,2)	44
COM(5,2)	45
COM(6,2)	46
COM(7,2)	47

LISTS(0)	4040
LISTS(1)	4141
LISTS(2)	4242
LISTS(3)	4343
LISTS(4)	4444
LISTS(5)	4545
LISTS(6)	4646
LISTS(7)	4747

transfer, SBR2 is 47_8 and DBR is also 47_8. COUNT contains 0 through 7. RW is 1 indicating a paging-out operation. SECTOR is 7 and FIELD is 1 so that the last page is transferred to the drum sector 7 and field 1 of this sector.

After paging-out the last page, the print out of the corresponding page descriptor, CCW, listheads for sector queue 7, the page transfer interrupt signal, and the page transfer status are summarized in Table 8-(d). COM(7,1) is $\underline{140147}_8$ where the $\underline{1401}_8$ underlined portion has the same meaning as that for the first page, and the remaining 47_8 not underlined is the MM page address of the current page. The listheads of sector queue 7 are 47_8 and 47_8; therefore, LISTS(7) is 4747_8. The settings for INTERRUPT(PAGE) and PTRAN are the same as those for the first page.

After 8 MM pages have been paged-out for the 8 MM pages, their corresponding page descriptors, channel command words, and listheads for the sector queues are shown in Table 8-(e). In the table PAGETABLE (40_8,1) is 0 and PAGETABLE(47_8,1) is 0. ; PAGETABLE(40_8,2) is 4100_8 and PAGETABLE(47_8,2) is 5700_8; COM(0,1) is 140140_8 and COM(7,1) is 140147_8; COM(0,2) is 40_8 and COM(7,2) is 47_8. For MM page 40_8, LIST(0) is 4040_8, and for MM page 47_8, LISTS(7) is 4747_8.

The outputs by test data 2 for paging-in the 8 MM pages mentioned in the previous section are summarized in Table 9-(a) to 9-(e). Table 9-(a) and 9-(b) differ only by the value for RW. In Table 9-(a), RW is 0 indicating a paging-in operation. Table 9-(b) and 8-(b) differ only by the values for COM(0,1), and PTRAN. In Table 9-(b), COM(0,1) is $\underline{100140}_8$ where the $\underline{1001}_8$ is 1000000001 in binary which is interpreted as follows: (a) the C field of the CCW is 1 and there is a page transfer; (b) it is paging-in; and (c) the drum field address is 1; and the remaining 40_8 not underlined is the MM page address of the current page. In Table 9-(b), PTRAN is set to 1 for paging-in.

Table 9 Test Data 2: Output in Octal

(for paging=in 8 drum pages from 8 drum sectors)

9-(a) Print-out when the first page is being transferred

PAGEPOST	RW	SECTOR	FIELD	SBR2	DBR	COUNT
40	0	0	1	40	40	0
40	0	0	1	40	40	1
40	0	0	1	40	40	2
40	0	0	1	40	40	3
40	0	0	1	40	40	4
40	0	0	1	40	40	5
40	0	0	1	40	40	6
40	0	0	1	40	40	7

9-(b) Print-out after the first page is transferred

Variables				
COM(0,1)	COM(0,2)	LISTS(0)	INTERRUPT(PAGE)	PTRAN
100140	40	4040	1	1

9-(c) Print-out when the last page is being transferred

PAGEPOST	RW	SECTOR	FIELD	SBR2	DBR	COUNT
47	0	7	1	47	47	0
47	0	7	1	47	47	1
47	0	7	1	47	47	2
47	0	7	1	47	47	3
47	0	7	1	47	47	4
47	0	7	1	47	47	5
47	0	7	1	47	47	6
47	0	7	1	47	47	7

9-(d) Print-out after the last page is transferred

COM(7,1)	COM(7,2)	LISTS(7)	INTERRUPT(PAGE)	PTRAN
100147	74	4747	1	1

9-(e) Print-out after 8 pages are transferred

PAGETABLE(40,1)	0
PAGETABLE(41,1)	0
PAGETABLE(42,1)	0
PAGETABLE(43,1)	0
PAGETABLE(44,1)	0
PAGETABLE(45,1)	0
PAGETABLE(46,1)	0
PAGETABLE(47,1)	0

PAGETABLE(40,2)	4000
PAGETABLE(41,2)	4200
PAGETABLE(42,2)	4400
PAGETABLE(43,2)	4600
PAGETABLE(44,2)	5000
PAGETABLE(45,2)	5200
PAGETABLE(46,2)	5400
PAGETABLE(47,2)	5600

COM(0,1)	100140
COM(1,1)	100141
COM(2,1)	100142
COM(3,1)	100143
COM(4,1)	100144
COM(5,1)	100145
COM(6,1)	100146
COM(7,1)	100147

COM(0,2)	40
COM(1,2)	41
COM(2,2)	42
COM(3,2)	43
COM(4,2)	44
COM(5,2)	45
COM(6,2)	46
COM(7,2)	47

LISTS(0)	4040
LISTS(1)	4141
LISTS(2)	4242
LISTS(3)	4343
LISTS(4)	4444
LISTS(5)	4545
LISTS(6)	4646
LISTS(7)	4747

Table 9-(c) and Table 8-(c) differ only by the value for RW. In Table 9-(c), RW is 0 indicating a paging-in operation.

Table 9-(d) and Table 8-(d) differ by the values for COM(7,1), and PTRAN. In Table 9-(d), COM(7,1) is $\underline{100}147_8$ where the $\underline{1001}_8$ underlined has the same meaning as that for the first page. In Table 9-(d), PTRAN is set to 1 for paging-in.

After 8 MM pages have been paged-in, for the 8 MM pages, their corresponding page descriptors, CCW's, and listheads for the sector queues are shown in Table 9-(e). In the table PAGETABLE(40_8,1) is 0 and PAGETABLE(47_8,2) is 5600_8; COM(0,1) is 100140_8 and COM(7,1) is 100147_8; COM(0,2) is 40_8 and COM(7,2) is 47_8; for MM page 40_8, LISTS(0) is 4040_8, and for MM page 47_8, LISTS(7) is 4747_8.

4.4 Discussions

As a result of this simulation study of the paging drum channel system, we have found that the PDC system design can be improved as follows:

(a) Instead of having a channel command word and a listheads word for each drum sector, these two words are preferrable to be combined into one 64-bit word. Thus, only one memory is needed to store both 16 CCW's and 16 listheads words. These are several advantages of having one memory instead of two memories. First, it is more economical to have one memory. Second, only one memory access is needed to obtain all the control information for a page transfer at a particular drum sector. Lastly, the number of micro-operations and registers can be reduced.

(b) The page table memory should become a part of the PDC; this reduces the number of registers, the time for fetching a page descriptor, and the time for updating the sector queues in the page table memory.

5. Simulation by Simula

The Simula language can be considered as a super set of the Algol-60 since all the attributes of Algol-60 are available in a Simula program. Therefore, the conversion of a given Algol program into a Simula program is relatively easy. Simulation by Simula provides the user with the capability of simulating parallel processes in the sense that these processes all start at the same simulated time. Furthermore, a Simula program provides execution holding time for various processes in addition to the output gathered from the original Algol program (7).

A Simula user may view a large system as a number of smaller subsystems called processes. These processes are logically separable although they can interact with one another. Besides, these processes may change their states in parallel or asynchronously (7). A process described in code is called an activity which is syntactically equivalent to an Algol procedure. There is a timing chain in Simula called Sequencing set which is a list of ordered pairs: (event identification, event time). The event time of any activity can be changed by means of HOLD Statements (9).

The conversion of the Algol simulation program mentioned in the previous section to a Simula simulation program is done in the following three steps:

(a) isolate the drum input-output subsequence and make it an activity by inserting execution holding time and Simula keywords.

(b) isolate the updating subsequence and the request-accept subsequence and make it an activity as in step (a),

(c) convert the remaining Algol program into a Simula program by rearranging the program statements and by inserting keywords into appropriate places.

Some HOLD statements are added to the Simula program at the end of each process to simulate execution holding time of each process. In order to add the HOLD statements, the following assumptions are made as shown in Fig. 35: initialization takes 10 units of time from the beginning to the point of the main program with label LO; decoding of a CCW takes 40 units of time from LO to the point of the main program with label Fork; the drum read/write process takes 320 units of time; and the updating process takes 50 units of time. These assumptions are based upon the number of micro-operations to be executed in each process. For example, the drum read/write process takes more time because it contains a loop for read and a loop for write; for one page transfer by our simulation model, it is necessary to stay in one of these loops 8 (number of words per page) times for write or 7 times for read.

5.1 Simulation Inputs

The two sets of test data designed for the Algol simulation are again used as the input data for the Simula simulation for comparison of the outputs. However, in Simula simulation here, the relative time among different processes can be shown and compared.

5.2 Simulation Program

The Simula simulation program is described in the flow chart shown in Fig. 36. At the beginning of the program, the drum read/write subsequence is declared as activity LEFT, while the updating subsequence together with the request-accept subsequence are declared as activity RIGHT. Counter I is first set to 1 so that when I is 1, it is paging-out, and when I is 2, it is paging-in operation. Next, data words are stored into MM pages 32 through

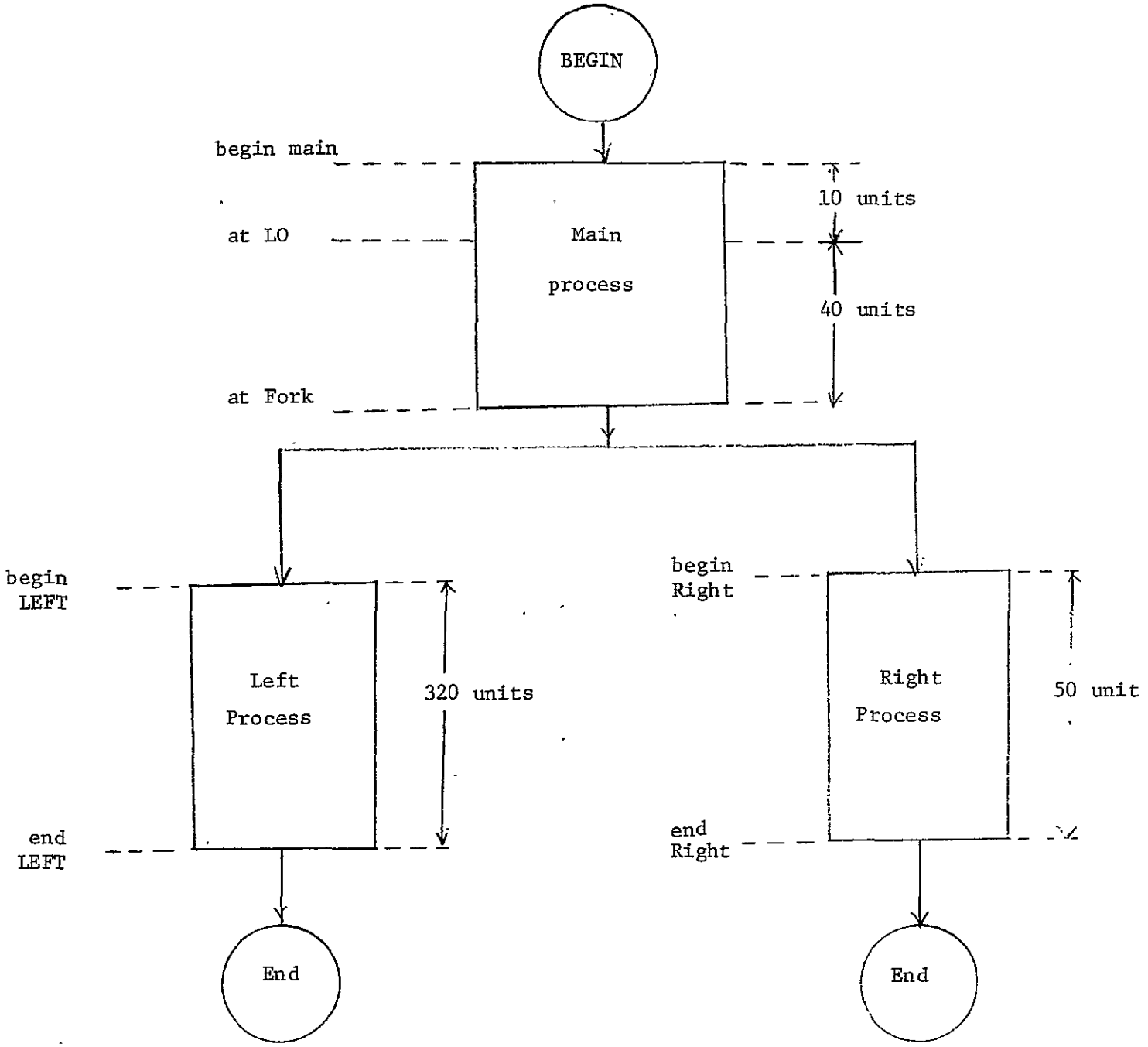


Fig. 35 Simulated execution time-units for the 3 processes in paging one page

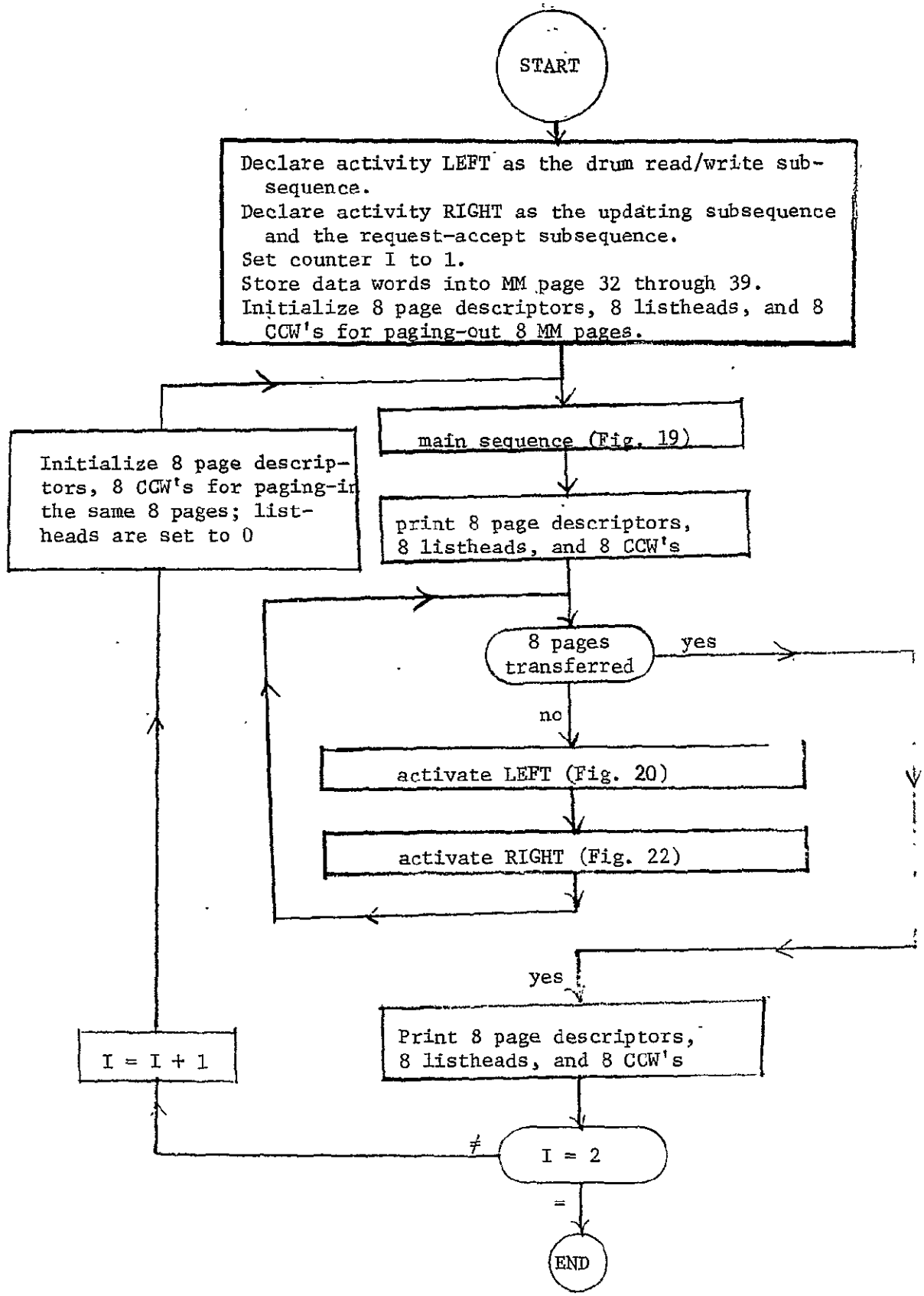


Fig. 36 Flow Chart for the Simula Simulation Program

39; 8 page descriptors, 8 listheads, and 8 CCW's are initialized to page-out the 8 MM pages. After the initialization, the main sequence is initiated and the 8 page descriptors, 8 listheads, and 8 CCW's are printed. To page-out an MM page, activity LEFT and activity RIGHT are activated at the same simulation time but are terminated at different simulation time. Nevertheless, whenever an MM page is to be paged-out, these two activities are reactivated at the same simulation time. After the 8 pages have been transferred, the octal representation of 8 descriptors, 8 listheads, and 8 CCW's are printed. Counter I is next tested. Since I is not 2, I is incremented by 1 and initialization for paging-in the 8 MM pages is done. 8 page descriptors and 8 CCW's are reset for paging-in the same 8 pages. The listheads are set to zero. The main sequence is again reactivated and the 8 page descriptors, 8 listheads, and 8 CCW's are printed. To page-in an MM page, activity LEFT and activity RIGHT are activated at the same simulation time but are terminated at different simulation time. However, whenever a drum page is to be paged-in, these two activities are reactivated simultaneously. After 8 drum pages have been transferred to the MM, the 8 page descriptors, 8 listheads, and 8 CCW's are printed. Counter I is next tested. Since I is 2, the program ends.

A listing of the Simula simulation program is included in Appendix B.

5.3 Simulation Results

In addition to the output of the Algol simulation program, we also get the simulation time for the three processes during paging. The timing of activities at the SQS have been obtained from the Simula simulation program and summarized in Table 10. There are three events in this simulation, namely,

main process, LEFT process, and RIGHT process. LEFT process and RIGHT process are parallel processes in the sense that they start at the same event time.

In Table 10, MM page 32 is the first page to be paged-out. Therefore, the main process starts at time 0. At LO, the time is incremented to 10, and at Fork, the time is incremented to 50. Now the main process is held 1000 time units to allow other events to occur. Thus the MM page 33 is paged-out at time 1050 which is incremented to 1060 at LO and to 1100 at Fork. Next, the main process is held 1000 units and the old MM page 33 is paged-in at time 9450. At LO, the time is incremented to 9460 and at Fork, the time is incremented to 9500. The time for paging other pages are obtained similarly.

Next, we consider the LEFT process in Table 10. MM page 32 starts transferring to the drum at time 50 and ends transferring at time 370 since we allow 320 time units for paging-out an MM page. MM page 32 starts transferring to the drum at time 1100 and stops transferring at time 1420. After MM page 39 ends transferring to the drum, paging-in operation starts. Old page 32 starts transferring from the drum to the MM at time 8450 and ends transferring at time 8770 since we allow 320 time units for paging-in a drum page. Old page 33 starts transferring to the MM at time 9500 and ends transferring at time 9820. The timing in the LEFT process for other pages are obtained similarly.

Finally, we consider the RIGHT process in Table 10. RIGHT process for MM page 32 starts at 50 which is the same as the starting time of the LEFT process for MM page 32. Since we allow 50 time units to update the page-table memory, the channel command memory, and the listheads memory , the RIGHT process terminates at time 100. The RIGHT process for MM page 32 starts at time 1100 and ends at time 1150. After the paging-out of

Table 10 Begin and end time units of the activities at the SQS
(Paging-out 8 Pages and Paging-in 8 Pages)

Page Number	Main process			LEFT process		RIGHT process	
	begin	at LO	at Fork	begin	end	begin	end
32	0	10	50	50	370	50	100
33	1050	1060	1100	1100	1420	1100	1150
34	2100	2110	2150	2150	2470	2150	2200
35	3150	3160	3200	3200	3520	3200	3250
36	4200	4210	4250	4250	4570	4250	4300
37	5250	5260	5300	5300	5620	5300	5350
38	6300	6310	6350	6350	6670	6350	6400
39	7350	7360	7400	7400	7720	7400	7450
32	8400	8410	8450	8450	8770	8450	8500
33	9450	9460	9500	9500	9820	9500	9550
34	10500	10510	10550	10550	10870	10550	10600
35	11550	11560	11600	11600	11920	11600	11650
36	12600	12610	12650	12650	12970	12650	12700
37	13650	13660	13700	13700	14020	13700	13750
38	14700	14710	14750	14750	15070	14750	14800
39	15750	15760	15800	15800	16120	15800	15850

8 MM pages, the paging-in operation starts. The RIGHT process for old MM page 32 starts at time 8450 and ends at time 8500. The RIGHT process for old MM page 39 starts at time 15800 and ends at time 15850. The timing in the RIGHT process for other pages are obtained similarly.

However, the sequencing of activities in the SQS could be very complicated if many processes of various execution time are activated randomly. In this case, we must know the first event of the sequencing set at all time, which process will be activated next, and the reactivation point of the next process.

6. Acknowledgement

The author wishes to express his thanks to Professor Yaohan Chu for his inspiring advice, helpful suggestions, and guidance as well as for his writing of the section on handling of queues; to his wife Marion Kwok for her patience and encouragement; to Mr. R. Pardo and Mr. J. Yeh for their helpful discussions and suggestions; and to Miss N. Nowell for her typing of the entire manuscript.

7. References

1. Chu, Y., "Introduction to Computer Organization", Prentice-Hall, Inc., 1970.

2. Chu, Y., "Notes on Channel Organization", Computer Science Center, University of Maryland, November, 1970.

3. Coffman, E. G., Jr., "Analysis of a Drum Input/Output Queue Under Scheduled Operation in a Paged Computer System", Journal of the ACM, Vol. 16, No. 1, January 1969, pp. 73-90.

4. Denning, Peter J., "The Working Set Model for Program Behavior", CACM, Vol. 11, No. 5, May 1968, pp. 323-333.

5. Gordon, Geoffrey, "System Simulation", Prentice-Hall, Inc., Englewood Cliffs, N.J., 1969.

6. Knuth, D.E., "Fundamental Algorithms" (The Art of Programming), Vol. 1, Addison-Wesley, 1969.

7. McCredic, J. W., Jr., SIMULA--AN ALGOL BASED SIMULATION LANGUAGE, Computer Science, Carnegie-Mellon University, pp. 315-322.

8. Pardo, O. R., "A Virtual Memory System Design", Technical Report 71-144, Computer Science Center, University of Maryland, January 1971.

9. Simula Manual, UNIVAC 1108 programmers reference, UP-7556 Rev. 1, 1970.

APPENDIX A. LISTING OF THE ALGOL SIMULATION PROGRAM

```
'RUN AL,001-11-768,KWOK,1,100
'ALG,IS          PDCHAN,PDCHAN
  EXTERNAL NON-RECURSIVE INTEGER PROCEDURE FLDA $
  BEGIN
  COMMENT ***A PAGING DRUM CHANNEL ***$
COMMENT FOR CONVENIENT SIMULATION 48 BIT DATA WORD IS CUT TO 36 BITS$
COMMENT FOR TESTING AND SIMULATION ON THE 1108 MEMORY,
        PAGE SIZE IS REDUCED TO 8 WORDS PER PAGE
        64 PAGES IN THE MAIN MEMORY
        1024 PAGES IN THE PAGING DRUM WHICH HAS 16 SECTOR QUEUES
         64 PAGES IN A QUEUE (OR LIST WITH A FRONT AND REAR POINTER)$
  COMMENT WE ASSUME MAIN MEMORY PAGE 0 DOES NOT EXIST $
COMMENT I IS THE LOOP CONTROL VARIABLE$
  INTEGER I$
COMMENT MAIN MEMORY AND RELATED REGISTERS$
COMMENT MAIN MEMORY$
  INTEGER ARRAY MEM(0..63,0..7)$
COMMENT MAIN MEMORY ADDRESS REGISTER$
  INTEGER MAR$
COMMENT MAIN MEMORY BUFFER REGISTER$
  INTEGER MBR$
COMMENT MAIN MEMORY READ/WRITE CONTROL REGISTER$
  INTEGER RW2$
COMMENT MAIN MEMORY PAGE ADDRESS$
    INTEGER MADR2BLOCK$
COMMENT MAIN MEMORY BUFFER REGISTER$
  INTEGER SBR2$
COMMENT MA(2) IS MAIN MEMORY ACCESS REGISTER$
  INTEGER MA2$
COMMENT MADR2(WRD) IS MAIN MEMORY WORD ADDRESS$
  INTEGER MADR2WRD$
COMMENT MAIN MEMORY PAGE ADDRESS OF THE POSTED PAGES$
 INTEGER  PAGEPOST$
COMMENT PAGE-TABLE MEMORY$
  INTEGER ARRAY PAGETABLE(0..63,1..2)$
COMMENT PAGE-TABLE MEMORY ADDRESS REGISTER$
  INTEGER PADR$
COMMENT PAGE-TABLE MEMORY BUFFER REGISTER$
  INTEGER ARRAY PTR2(1..2)$
  INTEGER PTR2CH$
  INTEGER PTR2SEC$
  INTEGER PTR2ROW$
  INTEGER PTR2LR$
 INTEGER PTR2LF$
COMMENT LISTHEAD MEMORY$
  INTEGER ARRAY LISTS(0..15)$
COMMENT LISTHEAD MEMORY ADDRESS REGISTER$
  INTEGER SECTOR$$
COMMENT LISTHEAD MEMORY BUFFER REGISTER$
  INTEGER PTL$
COMMENT POINTER THE FIRST PAGE OF A SECTOR QUEUE IN THE DRUM$
  INTEGER PTLFP$
COMMENT POINTER THE  LAST PAGE OF A SECTOR QUEUE IN THE DRUM$
```

```
  INTEGER PTLLP$
COMMENT GPTL IS AN AUXILIARY REGISTER$
  INTEGER GPTL$
  INTEGER GPTLFP$
  INTEGER GPTLLP$
COMMENT PAGE TABLE SEMAPHOR$
  INTEGER PTSEM1$
  INTEGER PTSEM2$
COMMENT PAGING DRUM AND RELATED REGISTER$
COMMENT PAGING DRUM MEMORY$
  INTEGER ARRAY PDRUM(0..15,0.. 31,0..7)$
COMMENT PAGING DRUM SECTOR ADDRESS$
  INTEGER CWORDSECT$
COMMENT PAGING DRUM CHANNEL ADDRESS$
  INTEGER CHANNEL$
COMMENT PAGING DRUM CHANNEL WORD COUNT$
  INTEGER CWORDCOUNT$
COMMENT DRUM READ/WRITE CONTROL REGISTER$
    INTEGER RW$
COMMENT DRUM BUFFER REGISTER$
    INTEGER DBR$
COMMENT DRUM ACTIVE INDICATOR$
    INTEGER DACTV$
COMMENT COMMAND MEMORY AND RELATED REGISTERS$
    INTEGER ARRAY COM(0..15,1..2)$
COMMENT COMMAND MEMORY ADDRESS REGISTER$
    INTEGER SEC$
COMMENT COMMAND MEMORY BUFFER REGISTER$
    INTEGER ARRAY COMMAND(1..2)$
COMMENT SUBREGISTERS OF THE COMMAND WORD$
    INTEGER COMC$
    INTEGER COMRWC$
    INTEGER COMCHAN$
    INTEGER COMPGE$
    INTEGER COMFIRSTWORD$
COMMENT DRUM BUFFER STATUS REGISTER$
  INTEGER BS$
COMMENT INTERRUPT(DRUMPAGE)$
  INTEGER INTERRUPTDP$
COMMENT MAIN MEMORY PAGE WHICH INTERRUPT OCCURED$
   INTEGER PAGINT$
COMMENT INTERRUPT(PAGE)$
  INTEGER INTERRUPTPGE$
COMMENT CURRENT PAGE ADDRESS$
  INTEGER PC$
COMMENT WORD COUNT OF THE PAGE$
  INTEGER COUNT$
COMMENT PAGE TRANSFER DIRECTION,0 WHEN NO TRANSFER,1 WHEN DRUM TO MEMORY,
            2 WHEN MEMORY TO DRUM,3 WHEN ERROR OCCURS$
  INTEGER PTRAN$
COMMENT PAGE TRANSFER COMPLETE WHEN 1$
  INTEGER PAGET$
COMMENT PAGE POSTING INDICATOR$
  INTEGER POST$
COMMENT COUNTER FOR THE NUMBER OF PARALLEL PROCESSES$
  INTEGER NPPROCESS$
  FORMAT F1 ( '  **  TRANSFER A PAGE  **********',A3,3)$
  FORMAT F2(X8 ,'PAGE-DESCRIPTORS IN OCTAL', A1,3)$
  FORMAT F3( X5,I2,8,X2,I10,8, X2, I10,8, A1,1)$
  FORMAT F4 (' COMMAND IN OCTAL =', I10,8,A1,1)$
  FORMAT F20( X8 ,' LISTHEADS IN OCTAL', A1,3)$
```

```
  FORMAT F21 (X5,I2,X2,I10.8, A1.1)$
  FORMAT F22(X8 ,' CHANNEL COMMAND WORDS IN OCTAL',A1.3)$
  FORMAT F11 ( F,'* * *  STMULATION  INPUT * * * ',A4.4)$
  FORMAT F12 ( F,'* * *  STMULATION OUTPUT * * * ',A4.4)$
    LOCAL LABEL LAST$
    LOCAL LABEL JOINT$
    LOCAL LABEL FINISH$
COMMENT INITIALIZE COUNTER $
  I= 1 $
COMMENT INITIALIZE THE MAIN MEMORY$
  FOR MAR=0 STEP 1 UNTIL 63 DO BEGIN
COMMENT THE I-TH PAGE CONTAINS ALL I'S$
  FOR COUNT=0 STEP 1 UNTIL 7 DO
  MEM(MAR,COUNT)= MAR$
  END$
COMMENT INITIALIZE THE COMMAND MEMORY FOR PAGING OUT $
COMMENT COMMANDS ARE TO WRITE 8 PAGES,PAGE 32,33,34,35,36,37,38,39  $
    FOR SEC=0 STEP 1 UNTIL 7 DO
    COM(SEC,1)= 8**5+4*8**4+8**2 + 4*8 + SEC $
  COMMENT SET UP FIRST WORD IN THE COM MEMORY$
  FOR SEC=0 STEP 1 UNTIL  7 DO COM(SEC,2)= 32  + SEC$
  COMMENT SET UP LINKAGES FOR THE PAGE-DESCRIPTORS IN THE PAGE -TABLES$
  PAGETABLE (32,1)= 0 $
  PAGETABLE (33,1)= 0 $
  PAGETABLE (34,1)= 0 $
  PAGETABLE (35,1)= 0 $
  PAGETABLE (36,1)= 0 $
  PAGETABLE (37,1)= 0 $
  PAGETABLE (38,1)= 0 $
  PAGETABLE (39,1)= 0 $
  FOR PADR = 32 STEP 1 UNTIL 39 DO
COMMENT ROW = 1 WHEN WRITE $
  PAGETABLE(PADR,2) = 8**3*4 + (PADR-32)*(8**2)*2 + 8**2$
  COMMENT INITIALIZATION OF LISTSHEAD MEMORY TO ENTER GETPAGE $
COMMENT PUT 8 PAGES IN 8 SECTOR QUEUE $
  FOR SECTORS = 0 STEP 1 UNTIL  7 DO
  LISTS(SECTORS) = 0 $
COMMENT ************************************************************$
START .. WRITE( F11)$
COMMENT PRINT TITLE$
  WRITE( F22)$
  FOR SEC =0 STEP 1 UNTIL  7 DO
  WRITE( SEC, COM(SEC,1), COM(SEC,2),F3)$
COMMENT PRINT TITLE$
  WRITE( F11)$
COMMENT PRINT TITLE$
    WRITE( F2)$
  FOR PADR = 32 STEP 1 UNTIL 39 DO
  WRITE(PADR,PAGETABLE(PADR,1)/((8**7)*2),PAGETABLE(PADR,2),F3)$
COMMENT PRINT TITLE$
    WRITE(F11)$
  WRITE(F20)$
  FOR SECTORS = 0 STEP 1 UNTIL  7 DO
  WRITE( SECTORS, LISTS(SECTORS), F21)$
  CWORDSECT=-1 $
COMMENT PAGING DRUM CHANNEL OPERATIONS START HERE$
PDC.. PAGEI=1$
  IF PAGEI EQL 0 THEN GOTO PDC$
L0..    PAGEI=0$
COMMENT  SET PAGE TRANSFER INDICATORS$
       NPPROCESS =2$
```

```
  CWORDSECT= CWORDSECT+1$
COMMENT CHECK IF 8 MM PAGES HAVE BEEN TRANSFERRED BY THE PDC $
  IF CWORDSECT EQL  8 THEN GOTO LAST$
COMMENT PRINT TITLE$
  WRITE( F12)$
COMMENT  OBTAIN A CHANNEL COMMAND WORDS$
  SEC=CWORDSECT$
COMMENT INPUT FROM CARD IS A COMMAND WORD$
  COUNT=0$
  CWORDCOUNT=0$
  WRITE(F1)$
  COMMAND(1)=COM(SEC,1)$
  WRITE( COMMAND(1), F4)$
  COMMAND(2)=COM(SEC,2)$
COMMENT
  WRITE( ' FIRSTWORD = ', COMMAND(2))$
COMMENT DECODING A COMMAND WORD AND PUT THE CONTROL INFORMATION
        INTO THE APPROPRIATE REGISTERS$
  COMC=FLDA(20,1,COMMAND(1))$
COMMENT  INDICATE A PAGE HAS BEEN SWAPPED WHEN 1$
  DACTV= COMC$
  COMRWC=FLDA(21,1,COMMAND(1))$
  RW= COMRWC$
  COMCHAN=FLDA(22,8,COMMAND(1))$
  CHANNEL = COMCHAN$
  COMPGE=FLDA(30,6,COMMAND(1))$
  MADR2BLOCK=COMPGE$
  PAGEPOST = COMPGE$
  WRITE(' PAGEPOST ',PAGEPOST)$
COMMENT DATA TRANSFER$
  COMFIRSTWORD = COMMAND(2)$
COMMENT  TRANSFER THE FIRST WORD OF A PAGE TO THE DRUM BUFFER REGISTER$
     DBR= COMFIRSTWORD$
  WRITE(' CHANNEL = ',CHANNEL)$
  IF RW EQL 1 THEN BEGIN
  WRITE( 'WRITE OPERATION,RW= ',RW)$
  COMMENT OUTPUT FIRST WORD TO DRUM$
  WRITE( '**************************************************MEMORY TO DRUM')$
COMMENT FOR WRITE OPERATION THE FIRST WORD IS ALREADY IN THE BUFFER
        BEFORE ENTERING THE WRITE LOOP $
  WRITE('WORD COUNT =', COUNT)$
  WRITE(' DBR= ',DBR)$
  PDRUM(CWORDSECT,CHANNEL,0)= DBR$
  END ELSE
                    WRITE( ' READ OPERATION, RW= ',RW)$
COMMENT  PARALLEL PROCESS STARTS HERE$
  WRITE(' *** DRUM READ/WRITE SUBSEQUENCE ***')$
FORK .. NPPROCESS =NPPROCESS -1$
COMMENT
  WRITE('NPPROCESS=',NPPROCESS)$
COMMENT  BRANCH TO ANOTHER PROCESS$
  IF NPPROCESS NEQ 1 THEN GO TO L10$
COMMENT  CHECK PAGE SWAP INDICATOR$
  IF DACTV EQL 1 THEN GOTO L1$
  PAGEI=1$
  PTRAN=0$
  WRITE('PTRAN=',PTRAN)$
  GOTO JOINT$
L3 .. PAGINT=MADR2BLOCK$
     INTERRUPTPGE=1$
  WRITE( ' INTERRUPT(PAGE) = ', INTERRUPTPGE)$
```

```
  GOTO L0$
COMMENT  PAGE HAS NOT BEEN SWAPPED COMES HERE$
L1.. BS=0$
COMMENT  BRANCH OUT TO WRITE.$
  IF RW EQL 1 THEN GO TO L5$
  MA2=0$
COMMENT
  WRITE('BS= ',BS,' RW= ',RW)$
COMMENT  DRUM TO MAIN MEMORY TRANSFER( READ BRANCH)$
L4.. IF MA2 EQL 1 THEN GO TO L4$
  BS=1$
COMMENT
  WRITE('MA(2)=',MA2,' BS= ',BS)$
L6.. IF BS EQL 0 THEN GOTO L6$
  MADR2WRD= COUNT$
  WRITE( '  INPUT WORD COUNT = ',MADR2WRD)$
  DBR = PDRUM(CWORDSECT,CHANNEL,CWORDCOUNT)$
  WRITE( ' DBR =', DBR)$
COMMENT INPUT FROM THE PAGING DRUM ONE WORD$
  SBR2=DBR$
  RW2=RW$
  MA2=1$
  WRITE('**************************DRUM TO MEMORY')$
  MBR= SBR2$
  WRITE( ' MBR ', MBR)$
  MEM(MADR2BLOCK,MADR2WRD)= MBR$
  IF COUNT EQL 7    THEN GOTO L7 ELSE GOTO RETURN$
L7.. PAGEI=1$
      DACTV=0$
RETURN.. IF PAGEI EQL 0 THEN BEGIN
      COUNT= COUNT+1$
  CWORDCOUNT= CWORDCOUNT +1$
      GOTO L1$
  END$
  COUNT=COUNT+RW$
  IF COUNT EQL 7    THEN BEGIN
  PTRAN=RW+1$
  WRITE('PTRAN = ',PTRAN)$
  GOTO JOINT$
  END
ELSE BEGIN
COMMENT  SET ERROR INDICATOR$
  INTERRUPTDP=1$
  PTRAN=3$
  WRITE('PTRAN =', PTRAN)$
  GOTO JOINT$
  END$
COMMENT  MAIN MEMORY TO DRUM TRANSFER ( WRITE BRANCH)$
L5.. MA2=0$
  IF MA2 EQL 1 THEN GOTO L5$
  RW2= RW$
  MADR2WRD= COUNT+1$
  WRITE( ' WORD COUNT = ', MADR2WRD)$
  MA2=1$
L8.. MA2=0$
  IF MA2 EQL 1 THEN GOTO L8$
L9.. BS =1$
COMMENT DATA TRANSFERS FROM MEMORY BUFFER TO DRUM BUFFERS$
  SBR2= MEM(MADR2BLOCK,MADR2WRD)$
  WRITE('SBR2 =',SBR2)$
  WRITE( '**************************************MEMORY TO DRUM')$
```

```
  DBR=SBR2$
  WRITE('DBR= ',DBR)$
  PDRUM(CWORDSECT,CHANNEL,CWORDCOUNT)= DBR$
COMMENT IN THE WRITE OPERATION THE WORD COUNT DOES NOT INCLUDE THE
        FIRST WORD OF THE PAGE. IF EXACTLY 7 MORE WORDS (0-6)
        WERE WRITTEN, AN ENTIRE PAGE WILL BE COMPLETELY TRANSFERRED$
  IF COUNT EQL 6    THEN GOTO L7 ELSE GOTO RETURN$
COMMENT MEMORIES UPDATING SUBSEQUENCE STARTS HERE$
L10 .. SECTORS = SEC$
       WRITE(' *** MEMORIES UPDATING SUBSEQUENCE ***')$
  PTL= LISTS(SECTORS)$
  PTLFP=FLDA(24,6,PTL)$
  PTLLP= FLDA(30,6,PTL)$
COMMENT FOR EMPTY QUEUE, SET PAGE SWAPPING INDICATOR$
  IF PTLFP EQL 0 THEN BEGIN
  COMC=0$
  GOTO L13$
  END$
COMMENT GETTING A PAGE$
K1.. PTSEM1=0$
  IF PTSEM1 EQL 1 THEN GOTO K1$
  PTSEM2=1$
COMMENT  PUT THE LIST HEAD INTO REGISTER GPTL$
  GPTL=PTL$
  GPTLFP=FLDA(24,6,GPTL)$
  GPTLLP= FLDA(30,6,GPTL)$
  WRITE('GPTL(FP)=',GPTLFP,' GPTL(LP)=', GPTLLP)$
  PADR=GPTLFP$
  PC= GPTLFP$
COMMENT  GET A PAGE DESCRIPTOR FROM THE PAGE TABLE MEMORY$
  PTR2(1)= PAGETABLE(PADR,1)$
  PTR2(2)= PAGETABLE(PADR,2)$
  PTR2CH = FLDA(17,8,PTR2(2))$
  WRITE( ' CHANNEL = ', PTR2CH)$
  PTR2SEC= FLDA(25,4,PTR2(2))$
  WRITE( ' SECTOR = ', PTR2SEC)$
  PTR2ROW= FLDA(29,1,PTR2(2))$
  PTR2LB= FLDA(2,6,PTR2(1))$
  PTR2LF= FLDA(8,6,PTR2(1))$
  GPTLFP= PTR2LF$
COMMENT  TRANSFER THE UPDATED LIST HEAD TO PTL$
  PTL = GPTL$
  PTSEM2= 0$
COMMENT  STORE THE UPDATED LISTHEAD TO THE LISTS MEMORY$
L15 .. IF SECTORS GTR 16 THEN GOTO LAST$
       LISTS(SECTORS)= PTL$
COMMENT SET UP A CHANNEL COMMAND WORDS$
    COMC=1$
    COMPGE=PC$
    COMCHAN=PTR2CH$
    COMRWC= PTR2ROW$
  COMMAND(1)= COMC*8**5+COMRWC*4*8**4+COMCHAN*8**2+ COMPGE$
  IF PTR2ROW EQL 0 THEN GOTO L13$
COMMENT  WRITE ONTO DRUM$
L11.. MA2= 0$
 IF MA2 EQL 1 THEN GOTO L11$
  RW2=1$
COMMENT  SET UP MAIN MEMORY ADDRESS REGISTERS$
  MADR2BLOCK= PC$
  MADR2WRD= 0$
  MA2=1$
```

```
L12 .. MA2=0$
  IF MA2 EQL 1 THEN GOTO L12$
COMMENT SET UP THE FIRST DATA WORD IN THE COMMAND REGISTERS$
  MBR= MEM(MADR2BLOCK,0)$
  SBR2= MBR$
  COMMAND(2)= SBR2$
COMMENT  PUT THE NEW CHANNEL COMMAND WORD INTO THE COMMAND MEMORY$
L13.. COM(SEC,1)= COMMAND(1)$
      COM(SEC,2)= COMMAND(2)$
  WRITE( COMMAND(1), F4)$
COMMENT THIS IS TO TEST THE PUTPT AND LOADPAGEDESCRIPTOR ROUTINE$
  PAGEI = 0$
  POST = 1$
L14.. IF PAGEI EQL 1 THEN GOTO JOINT$
  IF POST EQL 0 THEN GOTO L14$
COMMENT  LOAD PAGE DESCRIPTOR SEQUENCE STARTS HERE$
K2 .. PTSEM1 = 0$
  IF PTSEM1 EQL 1 THEN GOTO K2$
  PTSEM2=1$
COMMENT  GET A PAGE DESCRIPTOR$
  PADR= PAGEPOST$
  PTR2(1)= PAGETABLE(PADR,1)$
  PTR2(2)= PAGETABLE(PADR,2)$
COMMENT  FREE THE PAGE-TABLE MEMORY$
  PTSEM2=0$
  PTR2SEC= FLDA(25,4,PTR2(2))$
  SECTORS= PTR2SEC$
  PTL= LISTS(SECTORS)$
  PTLEP= FLDA(24,6,PTL)$
COMMENT  BRANCH TO PUT HTE PAGE ON DRUM IF QUEUE IS EMPTY$
  IF PTLEP NEQ 0 THEN GOTO K3$
COMMENT  GET THE NEXT CHANNEL COMMAND WORDS$
  SEC= SECTORS$
  COMMAND(1)= COM(SEC,1)$
  COMMAND(2)= COM(SEC,2)$
  IF COMC EQL 0 THEN BEGIN
  PC= PAGEPOST$
  WRITE( 'PC', PC)$
  PAGEI = 1$
  GOTO L15$
  END$
COMMENT  PUT A PAGE BACK TO THE SECTOR QUEUE ON THE PTM $
K3.. PTSEM1=0$
  IF PTSEM1 EQL 1 THEN GOTO K3$
  PTSEM2=1$
  GPTL= PTL$
  GPTLEP= FLDA(24,6,GPTL)$
COMMENT  IF THE SECTOR QUEUE IS EMPTY THE CURRENT PAGE BECOMES THE FIRST
         PAGE (FRONT OF THE QUEUE)$
  IF GPTLEP EQL 0 THEN BEGIN
  GPTLEP = PAGEPOST$
  GPTL = GPTLEP*8**2 + FLDA(30,6,GPTL)$
  GOTO K4$
  END$
COMMENT INSERT THE NEW PAGE AT THE REAR OF AN NON-EMPTY
  LIST OF PAGE DESCRIPTORS$
  GPTLLP= FLDA(30,6,GPTL)$
  PADR= GPTLLP$
  PTR2(1)= PAGETABLE(PADR,1)$
  PTR2(2)= PAGETABLE(PADR,2)$
COMMENT UPDATE A PAGE DESCRIPTOR$
```

```
   PTR2LF= MOD( PTR2(1),2**28)$
   PTR2LB = PTR2(1) - PTR2LF* 2**28 $
   PTR2(1)= PTR2LB + PAGEPOST*2**22$
   PAGETABLE(PADR,1)= PTR2(1)$
   PAGETABLE(PADR,2)= PTR2(2)$
COMMENT  GET THE PAGE DESCRIPTOR OF THE POSTED PAGES$
K4.. PADR = PAGEPOST$
   PTR2(1)= PAGETABLE(PADR,1)$
   PTR2(2)= PAGETABLE(PADR,2)$
COMMENT  LET THE BACKWORD POINTER POINT TO THE REAR OF
   THE LIST OF PAGE DESCRIPTORS$
   PTR2LB= GPTLLP$
COMMENT GROUNDING A LIST $
   PTR2LF =0$
   PTR2(1)= GPTLLP*2**28$
COMMENT  RETURN THE NEW PAGE DESCRIPTOR TO THE PAGE-TABLE MEMORY$
   PAGETABLE(PADR,1)= PTR2(1)$
   PAGETABLE(PADR,2)= PTR2(2)$
COMMENT UPDATE LISTHEADS$
   GPTLLP= MOD( GPTL,8**2)$
   GPTLFP= GPTL - GPTLLP$
   GPTL = GPTLFP + PADR$
   PTL= GPTL$
   PTSEM2=0$
COMMENT  STORE LISTHEAD$
   LISTS(SECTORS)= PTL$
   WRITE( ' SECTORS   LISTHEADS')$
   WRITE( SECTORS, LISTS(SECTORS),F21)$
   POST=0$
   PAGEI = 1$
   GOTO L14$
JOINT.. IF NPPROCESS EQL 0 THEN GOTO L3 ELSE GOTO FORK$
COMMENT OUTPUT THE UPDATED LISTS$
LAST..  WRITE (F12)$
   WRITE(F22)$
   FOR SEC = 0 STEP 1 UNTIL  7 DO
   WRITE( SEC, COM(SEC,1), COM(SEC,2),F3)$
   WRITE(F12)$
        WRITE( F2)$
   FOR PADR = 32 STEP 1 UNTIL 39 DO
   WRITE(PADR,PAGETABLE(PADR,1)/((8**7)*2),PAGETABLE(PADR,2),F3)$
   WRITE(F12)$
   WRITE(F20)$
   FOR SECTORS = 0 STEP 1 UNTIL  7 DO
   WRITE(SECTORS,LISTS(SECTORS),F21)$
   IF I EQL 2 THEN GOTO FINISH$
COMMENT SET UP CHANNEL COMMAND WORDS TO READ 8 PAGES$
   I= I+1$
COMMENT INITIALIZE THE COMMAND MEMORY FOR PAGING IN  $
   FOR SEC = 0 STEP 1 UNTIL 7 DO
     COM(SEC,1)= 8**5+ 8**2 + 4*8 + SEC $.
COMMENT INITIALIZE PAGE-DESCRIPTORS FOR PAGING-IN$
COMMENT ROW = 0 WHEN READ $
   FOR PADR = 32 STEP 1 UNTIL 39 DO
   PAGETABLE(PADR,2) = 8**3*4 + (PADR-32)*(8**2)*2$
   FOR SECTORS = 0 STEP 1 UNTIL  7 DO
   LISTS(SECTORS) = 0 $
   GOTO START$
FINISH .. END$
'ASM,IS         DECODE,DECODE
$(1)     AXR$.
```

```
.         ROUTINE FOR DECODING PAGE DESCRIPTORS,CHANNEL COMMAND WORDS
.         AND LISTHEADS FOR THE PAGING DRUM SECTOR QUEUES.
.         INPUT FORMAT  ***  FLDA(I,J,K)
FLDA*   LA      A2,*3,X11     . FETCH THE WORD K
        LA      A0,*1,X11     . FETCH THE BIT POSITION I
        SA,H2   A0,L1         . GETTING RID OF UPPER BITS
        SA,H2   A0,L2         .
L1      LSSL    A2,0          . SHIFT LEFT I BIT POSITIONS
L2      SSL     A2,0          . SHIFT RIGHT I BIT POSITIONS
        LA,U    A1,36         . COMPUTE 36-I-J
        AN      A1,*2,X11     .
        AN      A1,A0         .
        SA,H2   A1,L3         . GETTING RID OF THE LOWER BITS
L3      SSL     A2,0          . SHIFT RIGHT 36-I-J BIT POSITIONS
        J       4,X11         . RETURN TO THE ALGOL SIMULATION PROGRAM
        END
*MAP
*XQT
```

APPENDIX B. LISTING OF THE SIMULA SIMULATION PROGRAM

```
'RUN AC,001-11-768,KWOK,1,100
'ALG,CIS        PDCSIM,PDCSIM
  EXTERNAL NON-RECURSIVE INTEGER PROCEDURE FLDA $
  SIMULA BEGIN
  COMMENT ***A PAGING DRUM CHANNEL ***$
COMMENT FOR CONVENIENT SIMULATION 48 BIT DATA WORD IS CUT TO 36 BITS$
COMMENT FOR TESTING AND SIMULATION ON THE 1108 MEMORY,
        PAGE SIZE IS REDUCED TO 8 WORDS PER PAGE
        64 PAGES IN THE MAIN MEMORY
        1024 PAGES IN THE PAGING DRUM WHICH HAS 16 SECTOR QUEUES
         64 PAGES IN A QUEUE (OR LIST WITH A FRONT AND REAR POINTER)$
COMMENT I IS THE LOOP CONTROL VARIABLE$
  INTEGER I$
COMMENT MAIN MEMORY AND RELATED REGISTERS$
COMMENT MAIN MEMORY$
  INTEGER ARRAY MEM(0..63,0..7)$
COMMENT MAIN MEMORY ADDRESS REGISTER$
  INTEGER MAR$
COMMENT MAIN MEMORY BUFFER REGISTER$
  INTEGER MBR$
COMMENT MAIN MEMORY READ/WRITE CONTROL REGISTER$
  INTEGER RW2$
COMMENT MAIN MEMORY PAGE ADDRESS$
    INTEGER MADR2BLOCK$
COMMENT MAIN MEMORY BUFFER REGISTER$
  INTEGER SBR2$
COMMENT MA(2) IS MAIN MEMORY ACCESS REGISTER$
  INTEGER MA2$
COMMENT MADR2(WRD) IS MAIN MEMORY WORD ADDRESS$
  INTEGER MADR2WRD$
COMMENT MAIN MEMORY PAGE ADDRESS OF THE POSTED PAGE$
 INTEGER  PAGEPOST$
COMMENT PAGE-TABLE MEMORY$
  INTEGER ARRAY PAGETABLE(0..63,1..2)$
COMMENT PAGE-TABLE MEMORY ADDRESS REGISTER$
  INTEGER PADR$
COMMENT PAGE-TABLE MEMORY BUFFER REGISTER$
  INTEGER ARRAY PTR2(1..2)$
  INTEGER PTR2CH$
  INTEGER PTR2SEC$
  INTEGER PTR2ROW$
  INTEGER PTR2LR$
 INTEGER PTR2LF$
COMMENT LISTHEAD MEMORY$
  INTEGER ARRAY LISTS(0..15)$
COMMENT LISTHEAD MEMORY ADDRESS REGISTER$
  INTEGER SECTOR$$
COMMENT LISTHEAD MEMORY BUFFER REGISTER$
  INTEGER PTL$
COMMENT POINTER THE FIRST PAGE OF A SECTOR QUEUE IN THE DRUM$
  INTEGER PTLF$
COMMENT POINTER THE  LAST PAGE OF A SECTOR QUEUE IN THE DRUM$
  INTEGER PTLL$
```

```
COMMENT GPTL IS AN AUXILIARY REGISTERS$
  INTEGER GPTL$
  INTEGER GPTLFP$
  INTEGER GPTLLP$
COMMENT PAGE TABLE SEMAPHORS$
  INTEGER PTSEM1$
  INTEGER PTSEM2$
COMMENT PAGING DRUM AND RELATED REGISTERS$
COMMENT PAGING DRUM MEMORY$
  INTEGER ARRAY PDRUM(0..15,0.. 31,0..7)$
COMMENT PAGING DRUM SECTOR ADDRESS$
  INTEGER CWORDSECT$
COMMENT PAGING DRUM CHANNEL ADDRESS$
  INTEGER CHANNEL$
COMMENT PAGING DRUM CHANNEL WORD COUNT$
  INTEGER CWORDCOUNT$
COMMENT DRUM READ/WRITE CONTROL REGISTERS$
    INTEGER RW$
COMMENT DRUM BUFFER REGISTERS$
    INTEGER DBR$
COMMENT DRUM ACTIVE INDICATORS$
    INTEGER DACTV$
COMMENT COMMAND MEMORY AND RELATED REGISTERS$
    INTEGER ARRAY COM(0..15,1..2)$
COMMENT COMMAND MEMORY ADDRESS REGISTER$
    INTEGER SEC$
COMMENT COMMAND MEMORY BUFFER REGISTER$
    INTEGER ARRAY COMMAND(1..2)$
COMMENT SUBREGISTERS OF THE COMMAND WORD$
    INTEGER COMC$
    INTEGER COMRWC$
    INTEGER COMCHAN$
    INTEGER COMPGE$
    INTEGER COMFIRSTWORD$
COMMENT DRUM BUFFER STATUS REGISTER$
  INTEGER BS$
COMMENT INTERRUPT(DRUMPAGE)$
  INTEGER INTERRUPTDP$
COMMENT MAIN MEMORY PAGE WHICH INTERRUPT OCCURED$
   INTEGER PAGINT$
COMMENT INTERRUPT(PAGE)$
  INTEGER INTERRUPTPGE$
COMMENT CURRENT PAGE ADDRESS$
  INTEGER PC$
COMMENT WORD COUNT OF THE PAGE$
  INTEGER COUNT$
COMMENT PAGE TRANSFER DIRECTION,0 WHEN NO TRANSFER,1 WHEN DRUM TO MEMORY,
          2 WHEN MEMORY TO DRUM,3 WHEN ERROR OCCURS$
  INTEGER PTRAN$
COMMENT PAGE TRANSFER COMPLETE WHEN 1$
  INTEGER PAGET$
COMMENT PAGE POSTING INDICATORS$
  INTEGER POST$
  FORMAT F1 ( '  **  TRANSFER A PAGE  **********************',A3,3)$
  FORMAT F2(X8 ,'PAGE-DESCRIPTORS IN OCTAL', A1,3)$
  FORMAT F3( X5,I2.8,X2,I10.8, X2, I10.8, A1,1)$
  FORMAT F4 (' COMMAND IN OCTAL =', I10.8,A1,1)$
  FORMAT F20( X8 ,' LISTHEADS IN OCTAL', A1,3)$
  FORMAT F21 (X5,I2,X2,I10.8, A1,1)$
  FORMAT F22(X8 ,' CHANNEL COMMAND WORDS IN OCTAL',A1,3)$
  FORMAT F11 ( F,'* * *  SIMULATION  INPUT * * * ',A4,4)$
```

```
  FORMAT F12 ( E,'* * *  SIMULATION OUTPUT * * * ',A4,4)$
    LOCAL LABEL LAST$
    LOCAL LABEL FINISH$
  LOCAL LABEL L3$
COMMENT ************************************************************$
  ACTIVITY LEFT$
  BEGIN
  WRITE(' *** FIRST PROCESS OF THE PARALLEL PROCESS *** ')$
  WRITE( ' IN LEFT TIME IS * * * ',TIME)$
COMMENT  CHECK PAGE SWAP INDICATOR$
  IF DACTV EQL 1 THEN GOTO L1$
  PAGEI=1$
COMMENT
  WRITE('DACTV = ',DACTV,'PAGEI= ',PAGEI)$
  PTRAN=0$
  WRITE('PTRAN=',PTRAN)$
  HOLD(5.0)$
  WRITE( ' IN LEFT TIME IS * * * ',TIME)$
  TERMINATE( CURRENT)$
COMMENT  PAGE HAS NOT BEEN SWAPPED COMES HERE$
L1.. BS=0$
COMMENT  BRANCH OUT TO WRITE.$
  IF RW EQL 1 THEN GO TO L5$
  MA2=0$
COMMENT
  WRITE('BS= ',BS,' RW= ',RW)$
COMMENT  DRUM TO MAIN MEMORY TRANSFER( READ BRANCH)$
L4.. IF MA2 EQL 1 THEN GO TO L4$
  BS=1$
L6.. IF BS EQL 0 THEN GOTO L6$
  MADR2WRD= COUNT$
  WRITE( '  INPUT WORD COUNT = ',MADR2WRD)$
  DBR = PDRUM(CWORDSECT,CHANNEL,CWORDCOUNT)$
  WRITE( ' DBR =', DBR)$
COMMENT INPUT FROM THE PAGING DRUM ONE WORD$
  SBR2=DBR$
  RW2=RW$
  MA2=1$
  WRITE('*************************DRUM TO MEMORY')$
  MBR= SBR2$
  WRITE( ' MBR ', MBR)$
  MEM(MADR2BLOCK,MADR2WRD)= MBR$
  IF COUNT EQL 7     THEN BEGIN
    PAGEI = 1$
      DACTV=0$
  ENDS$
L7 ..     IF PAGEI EQL 0 THEN BEGIN
      COUNT= COUNT+1$
  CWORDCOUNT= CWORDCOUNT +1$
      GOTO L1$
  ENDS$
  COUNT=COUNT+RW$
  IF COUNT EQL 7     THEN BEGIN
  PTRAN=RW+1$
  WRITE('PTRAN = ',PTRAN)$
  HOLD(320.0)$
  WRITE( ' IN LEFT TIME IS * * * ',TIME)$
  TERMINATE( CURRENT)$
  GOTO  L3$
  END
ELSE BEGIN
```

```
COMMENT  SET ERROR INDICATORS$
  INTERRUPTDP=1$
  PTRAN=3$
  WRITE('PTRAN =', PTRAN)$
  HOLD(320.0)$
  WRITE( ' IN LEFT TIME IS * * * ',TIME)$
  TERMINATE( CURRENT)$
  GOTO  L3$
  ENDS$
COMMENT  MAIN MEMORY TO DRUM TRANSFER ( WRITE BRANCH)$
L5.. MA2=0$
  IF MA2 EQL 1 THEN GOTO L5$
  RW2= RW$
  MADR2WRD= COUNT+1$
  WRITE( ' WORD COUNT = ', MADR2WRD)$
  MA2=1$
L8.. MA2=0$
  IF MA2 EQL 1 THEN GOTO L8$
L9.. BS =1$
COMMENT DATA TRANSFERS FROM MEMORY BUFFER TO DRUM BUFFERS$
  SBR2= MEM(MADR2BLOCK,MADR2WRD)$
  WRITE('SBR2 =',SBR2)$
  WRITE( '**********************************************MEMORY TO DRUM')$
  DBR=SBR2$
  WRITE('DBR= ',DBR)$
  PDRUM(CWORDSECT,CHANNEL,CWORDCOUNT)= DBR$
COMMENT IN THE WRITE OPERATION THE WORD COUNT DOES NOT INCLUDE THE
        FIRST WORD OF THE PAGE. IF EXACTLY 7 MORE WORDS (0-6)
        WERE WRITTEN, AN ENTIRE PAGE WILL BE COMPLETELY TRANSFERRED$
  IF COUNT EQL 6    THEN BEGIN
  PAGEI = 1$
  DACTV= 0$
  ENDS$
  GOTO L7 $
COMMENT THE SECOND PARALLEL PROCESS STARTS HERE$
COMMENT  CHECK IF THE READ/WRITE LOOP NEEDED.$
  ENDS$
COMMENT ************************************************************$
ACTIVITY RIGHT $
  BEGIN
  WRITE( ' IN RIGHT TIME IS ',TIME)$
COMMENT MEMORIES UPDATING SUBSEQUENCE STARTS HERE$
L10 .. SECTORS = SEC$
       WRITE(' *** MEMORIES UPDATING SUBSEQUENCE ***')$
  PTL= LISTS(SECTORS)$
  PTLFP=FLDA(24,6,PTL)$
  PTLLP= FLDA(30,6,PTL)$
COMMENT  FOR EMPTY QUEUE, SET PAGE SWAPPING INDICATORS$
  IF PTLFP EQL 0 THEN BEGIN
  COMC=0$
  GOTO L13$
  ENDS$
COMMENT GETTING A PAGES$
K1.. PTSEM1=0$
  IF PTSEM1 EQL 1 THEN GOTO K1$
  PTSEM2=1$
COMMENT  PUT THE LIST HEAD INTO REGISTER GPTL$
  GPTL=PTL$
  GPTLFP=FLDA(24,6,GPTL)$
  GPTLLP= FLDA(30,6,GPTL)$
  WRITE('GPTL(FP=)',GPTLFP,' GPTL(LP)=', GPTLLP)$
```

```
   PADR=GPTLFP$
   PC= GPTLFP$
 COMMENT  GET A PAGE DESCRIPTOR FROM THE PAGE TABLE MEMORY$
   PTR2(1)= PAGETABLE(PADR,1)$
   PTR2(2)= PAGETABLE(PADR,2)$
   PTR2CH = FLDA(17,8,PTR2(2))$
   WRITE( ' CHANNEL = ', PTR2CH)$
   PTR2SEC= FLDA(25,4,PTR2(2))$
   WRITE( ' SECTOR = ', PTR2SEC)$
   PTR2ROW= FLDA(29,1,PTR2(2))$
   PTR2LB= FLDA(2,6,PTR2(1))$
   PTR2LF= FLDA(8,6,PTR2(1))$
   GPTLFP= PTR2LF$
 COMMENT  TRANSFER THE UPDATED LIST HEAD TO PTL$
   PTL = GPTL$
   PTSEM2= 0$
 COMMENT  STORE THE UPDATED LISTHEAD TO THE LISTS MEMORY$
 L15.. LISTS(SECTORS)= PTL$
 COMMENT  SET UP A CHANNEL COMMAND WORDS
      COMC=1$
      COMPGE=PC$
      COMCHAN=PTR2CH$
      COMRWC= PTR2ROW$
   COMMAND(1)= COMC*8**5+COMRWC*4*3**4+COMCHAN*8**2+ COMPGE$
   IF PTR2ROW EQL 0 THEN GOTO L13$
 COMMENT  WRITE ONTO DRUM$
 L11.. MA2= 0$
  IF MA2 EQL 1 THEN GOTO L11$
   RW2=1$
 COMMENT  SET UP MAIN MEMORY ADDRESS REGISTERS$
   MADR2BLOCK= PC$
   MADR2WRD= 0$
   MA2=1$
 L12 .. MA2=0$
   IF MA2 EQL 1 THEN GOTO L12$
 COMMENT SET UP THE FIRST DATA WORD IN THE COMMAND REGISTERS$
   MBR= MEM(MADR2BLOCK,0)$
   SBR2= MBR$
   COMMAND(2)= SBR2$
 COMMENT  PUT THE NEW CHANNEL COMMAND WORD INTO THE COMMAND MEMORY$
 L13.. COM(SEC,1)= COMMAND(1)$
       COM(SEC,2)= COMMAND(2)$
   WRITE( COMMAND(1), F4)$
 COMMENT THIS IS TO TEST THE PUTPT AND LOADPAGEDESCRIPTOR ROUTINES$
   PAGEI = 0$
   POST = 1$
 L14.. IF PAGEI EQL 1 THEN BEGIN
   HOLD(50)$
   WRITE( ' IN RIGHT TIME IS ', TIME)$
   TERMINATE (CURRENT)$
   END$
   POST= 1$
   IF POST EQL 0 THEN GOTO L14$
 COMMENT  LOAD PAGE DESCRIPTOR SEQUENCE STARTS HERE$
 K2 .. PTSEM1 = 0$
   IF PTSEM1 EQL 1 THEN GOTO K2$
   PTSEM2=1$
 COMMENT  GET A PAGE DESCRIPTOR$
   PADR= PAGEPOST$
   PTR2(1)= PAGETABLE(PADR,1)$
   PTR2(2)= PAGETABLE(PADR,2)$
```

```
COMMENT  FREE THE PAGE-TABLE MEMORY$
  PTSEM2=0$
  PTR2SEC= FLDA(25,4,PTR2(2))$
  SECTORS= PTR2SEC$
  PTL= LISTS(SECTORS)$
  PTLFP= FLDA(24,6,PTL)$
COMMENT  BRANCH TO PUT HTE PAGE ON DRUM IF QUEUE IS EMPTY$
  IF PTLFP EQL 0 THEN GOTO K3$
COMMENT  GET THE NEXT CHANNEL COMMAND WORDS$
  SEC= SECTORS$
  COMMAND(1)= COM(SEC,1)$
  COMMAND(2)= COM(SEC,2)$
  WRITE( COMMAND(1),F4)$
  IF COMC EQL 0 THEN BEGIN
  PC= PAGEPOST$
  WRITE( 'PC', PC)$
  PAGET = 1$
  GOTO L15$
  END$
COMMENT  PUT A PAGE BACK TO THE SECTOR QUEUE $
K3.. PTSEM1=0$
  IF PTSEM1 EQL 1 THEN GOTO K3$
  PTSEM2=1$
  GPTL= PTL$
  GPTLFP= FLDA(24,6,GPTL)$
COMMENT  IF THE SECTOR QUEUE IS EMPTY THE CURRENT PAGE BECOMES THE FIRST
         PAGE (FRONT OF THE QUEUE)$
  IF GPTLFP EQL 0 THEN BEGIN
  GPTLFP = PAGEPOST$
  GPTL = GPTLFP*8**2 + FLDA(30,6,GPTL)$
  GOTO K4$
  END$
COMMENT INSERT THE NEW PAGE AT THE REAR OF AN NON-EMPTY QUEUES$
  GPTLLP= FLDA(30,6,GPTL)$
  PADR= GPTLLP$
  PTR2(1)= PAGETABLE(PADR,1)$
  PTR2(2)= PAGETABLE(PADR,2)$
COMMENT UPDATE A PAGE DESCRIPTOR$
  PTR2LF= MOD( PTR2(1),2**28)$
  PTR2LR = PTR2(1) - PTR2LF* 2**28 $
  PTR2(1)= PTR2LR + PAGEPOST*2**28$
  PAGETABLE(PADR,1)= PTR2(1)$
  PAGETABLE(PADR,2)= PTR2(2)$
COMMENT  GET THE PAGE DESCRIPTOR OF THE POSTED PAGE$
K4.. PADR = PAGEPOST$
  PTR2(1)= PAGETABLE(PADR,1)$
  PTR2(2)= PAGETABLE(PADR,2)$
COMMENT  LET THE BACKWORD POINTER POINT TO THE REAR OF THE QUEUE$
  PTR2LR= GPTLLP$
  PTR2LF =0$
  PTR2(1)= GPTLLP*2**28$
COMMENT  RETURN THE NEW PAGE DESCRIPTOR TO THE PAGE-TABLE MEMORY$
  PAGETABLE(PADR,1)= PTR2(1)$
  PAGETABLE(PADR,2)= PTR2(2)$
COMMENT UPDATE LISTHEADS$
  GPTLLP= MOD( GPTL,8**2)$
  GPTLFP= GPTL - GPTLLP$
  GPTL = GPTLFP + PADR$
  PTL= GPTL$
  PTSEM2=0$
COMMENT  STORE LISTHEAD$
```

```
  LISTS(SECTORS)= PTL$
  WRITE( ' SECTORS   LISTHEADS')$
  WRITE( SECTORS, LISTS(SECTORS),F21)$
  POST=0$
  PAGEI = 1$
  GOTO L14$
  END$
COMMENT ************************************************************$
  COMMENT MAIN ACTIVITY STARTS HERE $
COMMENT INITIALIZE COUNTER $
  I= 1 $
  COMMENT SET UP LINKAGES FOR THE PAGE-DESCRIPTORS IN THE PAGE -TABLE$
COMMENT INITIALIZE THE COMMAND MEMORY FOR PAGING OUT $
COMMENT COMMANDS ARE TO WRITE 8 PAGES,PAGE 32,33,34,35,36,37,38,39  $
    FOR SEC=0 STEP 1 UNTIL 7 DO
    COM(SEC,1)= 8**5+4*8**4+8**2 + 4*8 + SEC $
COMMENT ONLY CHANNEL 1 IS ACTIVATE IN THE RUN$
  PAGETABLE (32,1)=0 $
  PAGETABLE (33,1)= 0 $
  PAGETABLE (34,1)= 0 $
  PAGETABLE (35,1)= 0 $
  PAGETABLE (36,1)= 0 $
  PAGETABLE (37,1)= 0 $
  PAGETABLE (38,1)= 0 $
  PAGETABLE (39,1)= 0 $
  FOR PADR = 32 STEP 1 UNTIL 39 DO
COMMENT ROW = 1 WHEN WRITE $
  PAGETABLE(PADR,2) = 8**3*4 + (PADR-32)*(8**2)*2 + 8**2$
  COMMENT INITIALIZATION OF LISTSHEAD MEMORY TO ENTER GETPAGE $
  COMMENT SET UP FIRST WORD IN THE COM MEMORY$
  FOR SEC=0 STEP 1 UNTIL  7 DO COM(SEC,2)= 32  + SEC$
  FOR SECTORS = 0 STEP 1 UNTIL  7 DO
  LISTS(SECTORS) = 0 $
COMMENT PUT 8 PAGES IN 8 SECTOR QUEUE $
COMMENT INITIALIZE THE MAIN MEMORY$
  FOR MAR=0 STEP 1 UNTIL 63 DO BEGIN
COMMENT THE I-TH PAGE CONTAINS ALL I'S$
  FOR COUNT=0 STEP 1 UNTIL 7 DO
  MEM(MAR,COUNT)= MAR$
  END$
COMMENT ************************************************************$
COMMENT PRINT TITLE$
START .. WRITE( F11)$
COMMENT PRINT TITLE$
  WRITE( F22)$
  FOR SEC =0 STEP 1 UNTIL  7 DO
  WRITE( SEC, COM(SEC,1), COM(SEC,2),F3)$
COMMENT PRINT TITLE$
    WRITE( F2)$
  FOR PADR = 32 STEP 1 UNTIL 39 DO
  WRITE(PADR,PAGETABLE(PADR,1)/((8**7)*2),PAGETABLE(PADR,2),F3)$
COMMENT PRINT TITLE$
    WRITE(F11)$
  WRITE(F20)$
  FOR SECTORS = 0 STEP 1 UNTIL  7 DO
  WRITE( SECTORS, LISTS(SECTORS), F21)$
  CWORDSECT=-1 $
  COMMENT TIMING STARTS HERE$
PDC.. PAGEI=1$
  IF PAGEI EQL 0 THEN BEGIN
  HOLD(1,0)$
```

```
  GOTO PDC$
  END ELSE
L3 .. PAGINT=MADR2BLOCK$
      INTERRUPTPGE=1$
L0..    PAGFT=0$
  CWORDSECT= CWORDSECT+1$
COMMENT CHECK IF 8 MM PAGES HAVE BEEN TRANSFERRED BY THE PDC $
  IF CWORDSECT EQL 8 THEN GOTO LAST$
  HOLD (10,0)$
  WRITE( '  IN MAIN TIME IS * * * ', TIME)$
COMMENT PRINT TITLE$
  WRITE( F12)$
COMMENT  OBTAIN A CHANNEL COMMAND WORD$
  SEC=CWORDSECT$
COMMENT INPUT FROM CARD IS A COMMAND WORD$
  COUNT=0$
  CWORDCOUNT=0$
  WRITE(F1)$
  COMMAND(1)=COM(SEC,1)$
  WRITE( COMMAND(1), F4)$
  COMMAND(2)=COM(SEC,2)$
COMMENT
  WRITE( ' FIRSTWORD = ', COMMAND(2))$
COMMENT DECODING A COMMAND WORD AND PUT THE CONTROL INFORMATION
        INTO THE APPROPRIATE REGISTERS$
  COMC=FLDA(20,1,COMMAND(1))$
COMMENT  INDICATE A PAGE HAS BEEN SWAPPED WHEN 1$
  DACTV= COMC$
  COMRWC=FLDA(21,1,COMMAND(1))$
  RW= COMRWC$
  COMCHAN=FLDA(22,8,COMMAND(1))$
  CHANNEL = COMCHAN$
  COMPGE=FLDA(30,6,COMMAND(1))$
  MADR2BLOCK=COMPGE$
  PAGEPOST = COMPGE$
  WRITE(' PAGEPOST ',PAGEPOST)$
COMMENT DATA TRANSFER$
  COMFIRSTWORD = COMMAND(2)$
COMMENT  TRANSFER THE FIRST WORD OF A PAGE TO THE DRUM BUFFER REGISTER$
     DBR= COMFIRSTWORD$
  WRITE(' CHANNEL = ',CHANNEL)$
  IF RW EQL 1 THEN BEGIN
  WRITE( 'WRITE OPERATION,RW= ',RW)$
  COMMENT OUTPUT FIRST WORD TO DRUM$
  WRITE( '**************************************************MEMORY TO DRUM')$
COMMENT FOR WRITE OPERATION THE FIRST WORD IS ALREADY IN THE BUFFER
        BEFORE ENTERING THE WRITE LOOP $
  WRITE( ' WORD COUNT = ', COUNT)$
  WRITE(' DBR= ',DBR)$
  PDRUM(CWORDSECT,CHANNEL,0)= DBR$
  END ELSE
                    WRITE( ' READ OPERATION, RW= ',RW)$
COMMENT  PARALLEL PROCESS STARTS HERE$
COMMENT ************************************************************$
  HOLD (40,0)$
  WRITE( ' IN MAIN AT FORK TIME IS * * * ',TIME)$
  ACTIVATE NEW LEFT$
  ACTIVATE NEW RIGHT$
  HOLD( 1000)$
  WRITE( ' IN MAIN TIME IS ',TIME)$
  GO TO PDC $
```

```
LAST..  WRITF (F12)$
  WRITF(F22)$
  FOR SEC = 0 STEP 1 UNTIL  7 DO
  WRITE( SEC, COM(SEC,1), COM(SEC,2),F3)$
  WRITE(F12)$
       WRITE( F2)$
  FOR PADR = 32 STEP 1 UNTIL 39 DO
  WRITE(PADR,PAGETABLE(PADR,1)/((8**7)*2),PAGETABLE(PADR,2),F3)$
  WRITF(F12)$
  WRITF(F20)$
  FOR SECTORS = 0 STEP 1 UNTIL  7 DO
  WRITF(SECTORS,LISTS(SECTORS),F21)$
  IF I EQL 2 THEN GOTO FINISH$
COMMENT SET UP CHANNEL COMMAND WORDS TO READ 8 PAGES$
  I= I+1$
COMMENT INITIALIZE THE COMMAND MEMORY FOR PAGING IN  $
  FOR SEC = 0 STEP 1 UNTIL 7 DO
    COM(SEC,1)= 8**5+ 8**2 + 4*8 + SEC $
COMMENT INITIALIZE PAGE-DESCRIPTORS FOR PAGING-INS.
COMMENT ROW = 0 WHEN READ $
  FOR PADR = 32 STEP 1 UNTIL 39 DO
  PAGETABLE(PADR,2) = 8**3*4 + (PADR-32)*(8**2)*2$
  FOR SECTORS = 0 STEP 1 UNTIL  7 DO
  LISTS(SECTORS) = 0 $
  GOTO START$
FINISH .. END$
'ASM,IS          DECODE,DECODE
$(1)     AXR$.
.          ROUTINE FOR DECODING PAGE DESCRIPTORS,CHANNEL COMMAND WORDS
.          AND LISTHEADS FOR THE PAGING DRUM SECTOR QUEUES.
.          INPUT FORMAT  ***  FLDA(I,J,K)
FLDA*  LA        A2,*3,X11      . FETCH THE WORD K
       LA        A0,*1,X11      . FETCH THE BIT POSITION I
       SA,H2     A0,L1          . GETTING RID OF UPPER BITS
       SA,H2     A0,L2          .
L1     LSSL      A2,0           . SHIFT LEFT I BIT POSITIONS
L2     SSL       A2,0           . SHIFT RIGHT I BIT POSITIONS
       LA,U      A1,36          . COMPUTE 36-I-J
       AN        A1,*2,X11      .
       AN        A1,A0          .
       SA,H2     A1,L3          . GETTING RID OF THE LOWER BITS
L3     SSL       A2,0           . SHIFT RIGHT 36-I-J BIT POSITIONS
       J         4,X11          . RETURN TO THE ALGOL SIMULATION PROGRAM
       END
'MAP
'XQT
```