DAVID LEIGH
(08) 8332 2627

CONTROL DATA CORPORATION

Development Division - Applications

THE DEAD START PROCESS AND THE SYSTEM LOADER

Chippewa Operating System

9/25/65
REV. 1

THE DEAD START PROCESS AND THE SYSTEM LOADER

INTRODUCTION

The dead start process requires that a short program (up to 12 instructions) be set up on the matrix of toggle switches on the dead start panel. When the dead start switch is toggled, this dead start program is transmitted to peripheral processor zero's memory and executed. The dead start program in turn transmits a bootstrap program to another peripheral processor. This bootstrap program brings in the system loader from the library tape and transfers control to it. The system loader transfers a resident program to each peripheral processor, causes the Display and Monitor programs to be loaded, loads the central memory resident, library, and tables, and places the remaining library programs on the disk. It then inititates execution of the Display and Monitor programs.

THE IAM INSTRUCTION

A detailed understanding of the dead start loading process requires some familiarity with the functioning of the IAM instruction. The IAM instruction is a 24-bit instruction: the d portion of the instruction holds the channel number and the m portion of the instruction contains the address in peripheral processor memory where the first data word is to be stored. The A register is assumed to contain the number of words to be read. The functioning of the IAM instruction is shown in figure 1. Note the following points:

- During execution of the IAM instruction, the contents of the P register are stored in location 0, and the P register used to hold the memory address for the next word to be stored. At the time the contents of the P register are stored, P holds the address of the second word (m portion) of the IAM instruction. Before exiting the instruction, the contents of location 0 are read, incremented by one, and placed in the P register to provide the address of the next instruction.

- The IAM instruction tests the word count in the A register to see if it has been reduced to one: if so, (A) is reduced by one and the instruction exited. Therefore, if the IAM instruction is entered with the contents of the A register equal to zero, the word count is effectively 77777_8.
- The IAM instruction may be exited in one of two ways: (1) because the word count has been reduced to zero or (2) because the channel has become inactive. If the word count has not been reduced to zero and the channel is active, exit will not take place even though no data is being read: the processor will idle in trip 4, waiting for the channel to become full.

THE DEAD START SEQUENCE

When the dead start switch is toggled, the following sequence is initiated:

- The Master Clear signal is generated
- The A register of each peripheral processor is set to 10000_8: the P register of each peripheral processor is set to zero
- The K register of each peripheral processor is set to 712 (trip 4 of an IAM instruction)
- All channels are set to empty and active
- All peripheral processors are connected to their respective channels (i.e., PP0 to channel 0, PP1 to channel 1, etc.) by setting the appropriate channel number in each processor's Q register
- The first synchronizer on each channel is selected: the first unit on that synchronizer is selected
- The dead start synchronizer is selected on channel 0
- The program on the dead start panel is transferred to PP0 memory: first, a zero byte is transmitted (stored in location 0); next, the 12 bytes from the panel switches are transmitted (stored in location 1 - 14); finally, another zero byte is transmitted (stored in location 15)

THE IAM INSTRUCTION

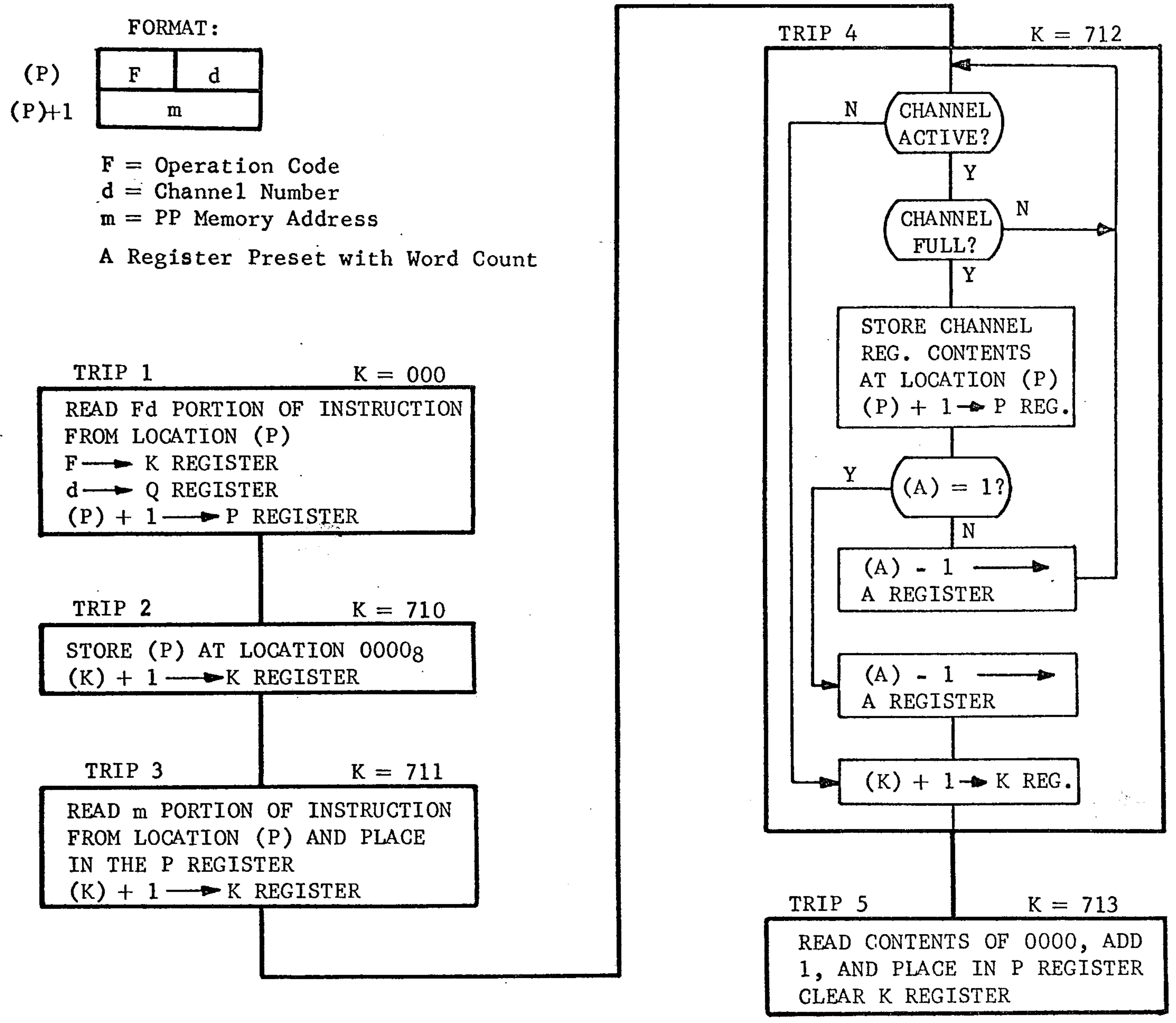

FORMAT:
(P)
(P)+1
F
d
m
F = Operation Code
d = Channel Number
m = PP Memory Address
A Register Preset with Word Count
TRIP 1
K = 000
READ Fd PORTION OF INSTRUCTION FROM LOCATION (P) F ⟶ K REGISTER d ⟶ Q REGISTER (P) + 1 ⟶ P REGISTER
TRIP 2
K = 710
STORE (P) AT LOCATION 0000₈ (K) + 1 ⟶ K REGISTER
TRIP 3
K = 711
READ m PORTION OF INSTRUCTION FROM LOCATION (P) AND PLACE IN THE P REGISTER (K) + 1 ⟶ K REGISTER
TRIP 4
K = 712
CHANNEL ACTIVE?
N
Y
CHANNEL FULL?
N
Y
STORE CHANNEL REG. CONTENTS AT LOCATION (P) (P) + 1 ⟶ P REG.
(A) = 1?
Y
N
(A) - 1 ⟶ A REGISTER
(A) - 1 ⟶ A REGISTER
(K) + 1 ⟶ K REG.
TRIP 5
K = 713
READ CONTENTS OF 0000, ADD 1, AND PLACE IN P REGISTER CLEAR K REGISTER

Figure 1

- The dead start synchronizer disconnects channel 0, initiating the execution of the dead start program

Peripheral processor zero treats the data sent by the dead start synchronizer as it would data arriving from any other controller. When the dead start synchronizer disconnects from channel zero, peripheral processor zero exits from the IAM instruction. In exiting, the contents of location 0 are incremented by 1 and used as the address of the next instruction. Since this location was cleared to 0 by the dead start process, the address of the next instruction is 0001: this location holds the first instruction of the program sent by the dead start synchronizer from the dead start panel.

THE DEAD START PROGRAM

The dead start program is shown in figure 2. The purpose of the dead start program is to transmit a bootstrap program to peripheral processor xx (PPxx), where xx is the channel number of the controller on which the system tape is mounted. The dead start program begins by transmitting a block of 8 words on channel xx. PPxx is connected to this channel and is idling in trip 4 of an IAM instruction: it will therefore read in these 8 words and store them in its memory beginning at location 0. PPxx will not, however, begin execution yet, since the channel is still active and the word count has not been reduced to zero.

The dead start program next disconnects channel xx: when channel xx becomes inactive, PPxx exits from the IAM instruction and begins execution of the bootstrap program. In exiting from the IAM instruction, the contents of location 0 are read, incremented by 1, and used as the address of the next instruction. Since the first word of the 8-word block sent by PP0 was equal to zero, this address is equal to 0001, and the instruction at this address is read and executed.

The dead start program then issues an input instruction for channel 13: since

THE DEAD START PROGRAM

xx = CHANNEL NUMBER FOR CONTROLLER ON WHICH SYSTEM TAPE IS MOUNTED

	PPO MEMORY LOCATION	INSTRUCTION	FUNCTION
THESE INSTRUCTIONS ARE EXECUTED IN PPO	0001	1410	SET THE A REGISTER TO 10_8
	0002	73xx	OUTPUT 8 WORDS BEGINNING AT LOCATION 0006 ON CHANNEL xx (PROCESSOR xx WILL STORE THESE INTO ITS MEMORY BEGINNING AT LOCATION 0)
	0003	0006	
	0004	75xx	DISCONNECT CHANNEL xx (THIS PERMITS PROCESSOR xx TO EXIT FROM THE IAM INSTRUCTION AND BEGIN EXECUTION OF THE PROGRAM IT HAS RECEIVED)
	0005	7113	SET PPO TO INPUT FROM CHANNEL 13 (CHANNEL 13 IS EMPTY AND ACTIVE: PPO WILL IDLE IN THIS INSTRUCTION UNTIL CHANNEL 13 BECOMES FULL)
	0006	0000	
THESE INSTRUCTIONS ARE TRANSFERRED TO AND EXECUTED IN PPxx	0007	77xx	
	0010	2060	
	0011	77xx	
	0012	2020	
	0013	74xx	
	0014	71xx	
	0015	0000	← CLEARED DURING DEAD START

Figure 2

channel 13 is empty and active, PPO will idle in trip 4 of this instruction waiting for channel 13 to become full.

THE LOADER BOOTSTRAP

The bootstrap program (figure 3) in PPxx issues the necessary function, activate, and input instructions to read the first record on the system tape into its memory beginning at location 0. When this record, which contains the loader program, has been read, PPxx will exit the IAM instruction when the controller disconnects the channel upon detecting the end-of-record gap. PPxx, in exiting the IAM instruction, reads the contents of location 0, adds 1 to it, and uses this as the address of the next instruction. Location 0 contains the first word of the record read from tape: thus, this word supplies the address of the first instruction of the loader program.

THE LOADER PROGRAM

The layout of the loader program in PPxx is shown in figure 4. As mentioned earlier, the first word of the loader program (location 0) contains the address - 1 of the first instruction of the loader. The loader program also contains a peripheral processor resident package in locations 0001 - 0777_8. This package is transmitted by the loader to each of the other peripheral processors. The resident program is contained in locations 0100 - 0777_8: locations 75, 76, and 77_8 contain the values 60, 61, and 62, respectively. These values are the central memory addresses of the Input Register, the Output Register, and the Message Buffer for PP1, and must be modified when the resident package is transmitted to processors other than PP1.

At the time the loader program begins execution, all channels except the channel corresponding to the processor containing the loader program (PPxx) are active and empty: their corresponding processors are idling in an IAM instruction, waiting for input. Channel xx, however, was disconnected by the tape controller

THE DEAD START PROGRAM

LOADER BOOTSTRAP

PPxx MEMORY LOCATION	INSTRUCTION	FUNCTION
0000	0000	NOT EXECUTED (ADDRESS - 1 OF FIRST INSTRUCTION)
0001 0002	77xx 2060	ISSUE FUNCTION CODE: SELECT REWIND
0003 0004	77xx 2020	ISSUE FUNCTION CODE: SELECT BINARY READ
0005	74xx	ACTIVATE CHANNEL xx (xx DISCONNECTED BY PPO PROGRAM)
0006 0007	71xx 0000	INPUT (A) WORDS FROM CHANNEL xx BEGINNING AT LOCATION 0000 (THE A REGISTER HAS NOT BEEN USED AND STILL CONTAINS 0)

- THE TAPE CONTROLLER WILL DISCONNECT THE CHANNEL WHEN THE END-OF-RECORD GAP IS DETECTED AND THUS CAUSE THE PROCESSOR TO EXIT FROM THE 71 INSTRUCTION.
- THE FIRST WORD FROM TAPE WILL BE READ INTO LOCATION 0: ON EXIT FROM THE IAM INSTRUCTION, THE CONTENTS OF LOCATION 0 ARE READ, INCREMENTED BY 1, AND USED AS THE ADDRESS OF THE NEXT INSTRUCTION.

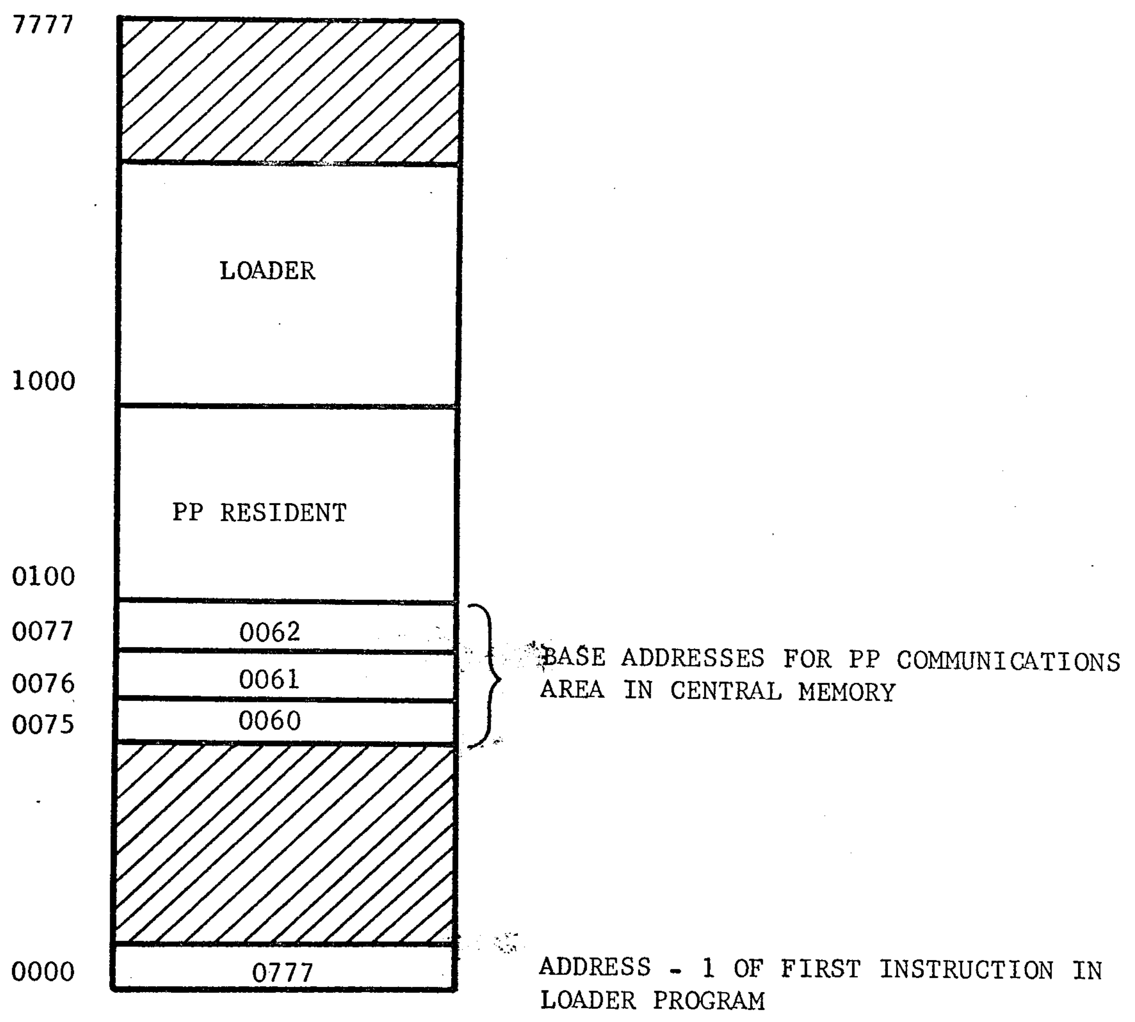
Figure 3

when the end-of-record gap following the loader program was detected, and is therefore inactive. The loader program searches for an inactive channel in order to determine which processor it resides in: it also inserts this channel number in the appropriate I/O instructions.

The loader program then proceeds to determine if the system tape is mounted on a 607-B unit or a 626-B unit, and modifies the function codes accordingly. Transfer of the resident package to each of the other processors then takes place. The loader first outputs a single word to the receiving processor, which stores it in its memory at location zero. Since the receiving processor is in trip 4 of an IAM instruction, it will, upon exiting this instruction, use the contents of location 0 as the address - 1 of the next instruction it is to execute. For processors 1 - 8, this address is 77_8: the address - 1 of the first instruction of the resident program. For processors 0 and 9, this address is 777_8: the address - 1 of the first instruction of the MTR and DSD programs, respectively. After transmitting this single word, the loader then transmits the resident package, which the receiving processor stores in its memory beginning at location 0001. The receiving processor does not exit the IAM instruction at this time, however, since the conditions for exiting (either word count reduced to zero or channel inactive) have not been met. As the transfer of each resident takes place, the loader program modifies the Input Register, Output Register, and Message Buffer pointers to the proper values for each processor.

When all processors have been loaded with the resident package, the loader program then proceeds to load the MTR and DSD programs from the system tape into processors 0 and 9, respectively.

The format of the system tape is illustrated in figure 5. The tape contains a single file of binary records: a full physical record contains 1000_8 CM words. A logical record, such as the MTR program or the CM resident, may be composed of more than one physical record: the last physical record for a specific program may be a short record of less than 1000_8 CM words. The end of a logical record is indicated when a short physical record is processed or when a zero length record


7777
LOADER
1000
PP RESIDENT
0100
0077
0062
0076
0061
0075
0060
0000
0777
BASE ADDRESSES FOR PP COMMUNICATIONS AREA IN CENTRAL MEMORY
ADDRESS - 1 OF FIRST INSTRUCTION IN LOADER PROGRAM

<u>LOADER PROGRAM LAYOUT IN PPxx</u>

(RECORD 1 ON SYSTEM TAPE)

Figure 4

(4 PP words) is detected, except for the disk library routines: the end of each disk library routine is indicated by a short record only. The end of a library is indicated by a zero length record.

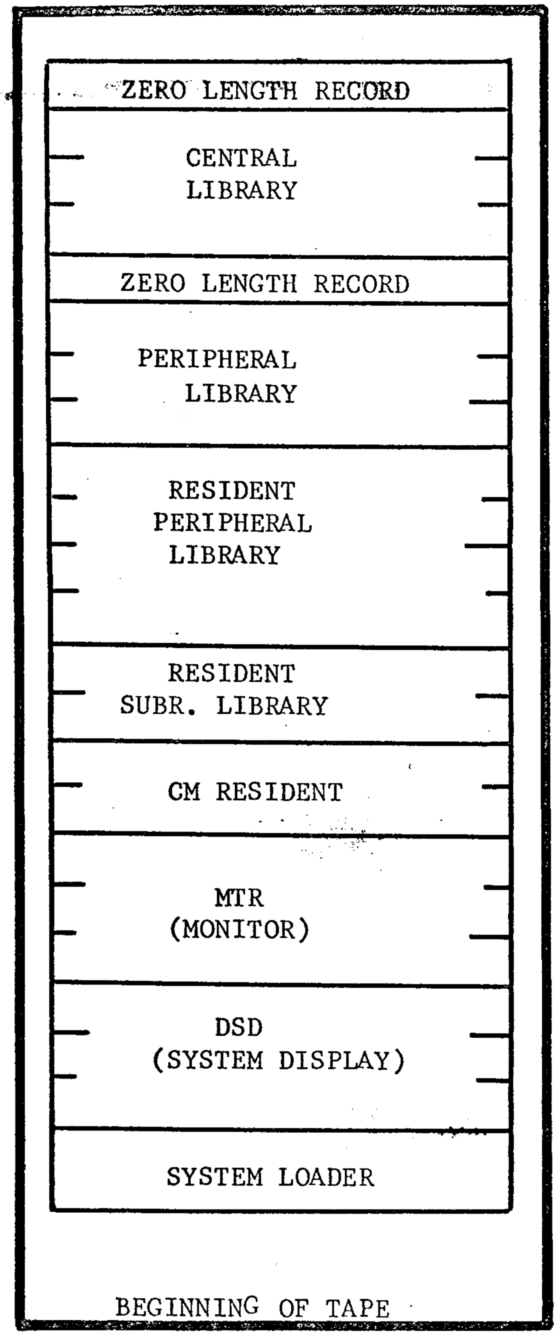
The loader reads the records comprising the DSD program, transferring each record as it is read to PP9: when a short record is processed or a zero length record is detected, loading of the DSD program is complete. This process is repeated for the MTR program records.

The CM Resident is loaded next. This resident contains table pointers and initial values for certain tables, such as the track reservation tables. The resident subroutine library is loaded using the RSL pointer from the CM Resident to provide the starting address: the resident peripheral library (RPL) is similiarly loaded. In all three cases (CM Resident, RSL, and RPL), records are read and transferred to central memory until either a zero length record is detected or a short record is processed.

The loader program then disconnects the channels for each of the other processors, permitting these processors to exit from the IAM instruction and begin execution of their programs. Now that MTR is executing, the loader program can utilize the assistance of MTR in loading the libraries on the disk.

The loader requests a track from MTR via its resident, and picks up the Peripheral Library Directory pointer from central memory in order to obtain the starting address of the directory. It then reads a record from the system tape, builds the PLD entry and writes it in the directory, and transfers the record to the disk. The next record is then read from tape and written to the disk: this process continues until a short record is processed, indicating that a complete program has been transferred. The next record is read from tape, the directory entry constructed and written in the directory, and the process of reading records from tape and transferring them to the disk repeated. The end of a library is indicated by the detection of a zero length record.

When the peripheral library has been transferred to the disk, the transfer of the central library to the disk is initiated and executed in the same manner.



- SYSTEM TAPE CONTAINS A SINGLE FILE OF BINARY RECORDS
- A FULL PHYSICAL RECORD IS A BLOCK OF 1000_8 CM WORDS
- A LOGICAL RECORD (i.e., MTR, RPL, etc.) MAY BE COMPOSED OF A NUMBER OF PHYSICAL RECORDS
- THE END OF A LOGICAL RECORD IS INDICATED WHEN A SHORT RECORD IS PROCESSED OR WHEN A ZERO LENGTH RECORD (4 BYTES) IS DETECTED
- THE END OF PERIPHERAL LIBRARY OR CENTRAL LIBRARY ROUTINES IS INDICATED BY A SHORT RECORD ONLY: THE END OF THE LIBRARY IS INDICATED BY A ZERO LENGTH RECORD

SYSTEM TAPE FORMAT

Figure 5

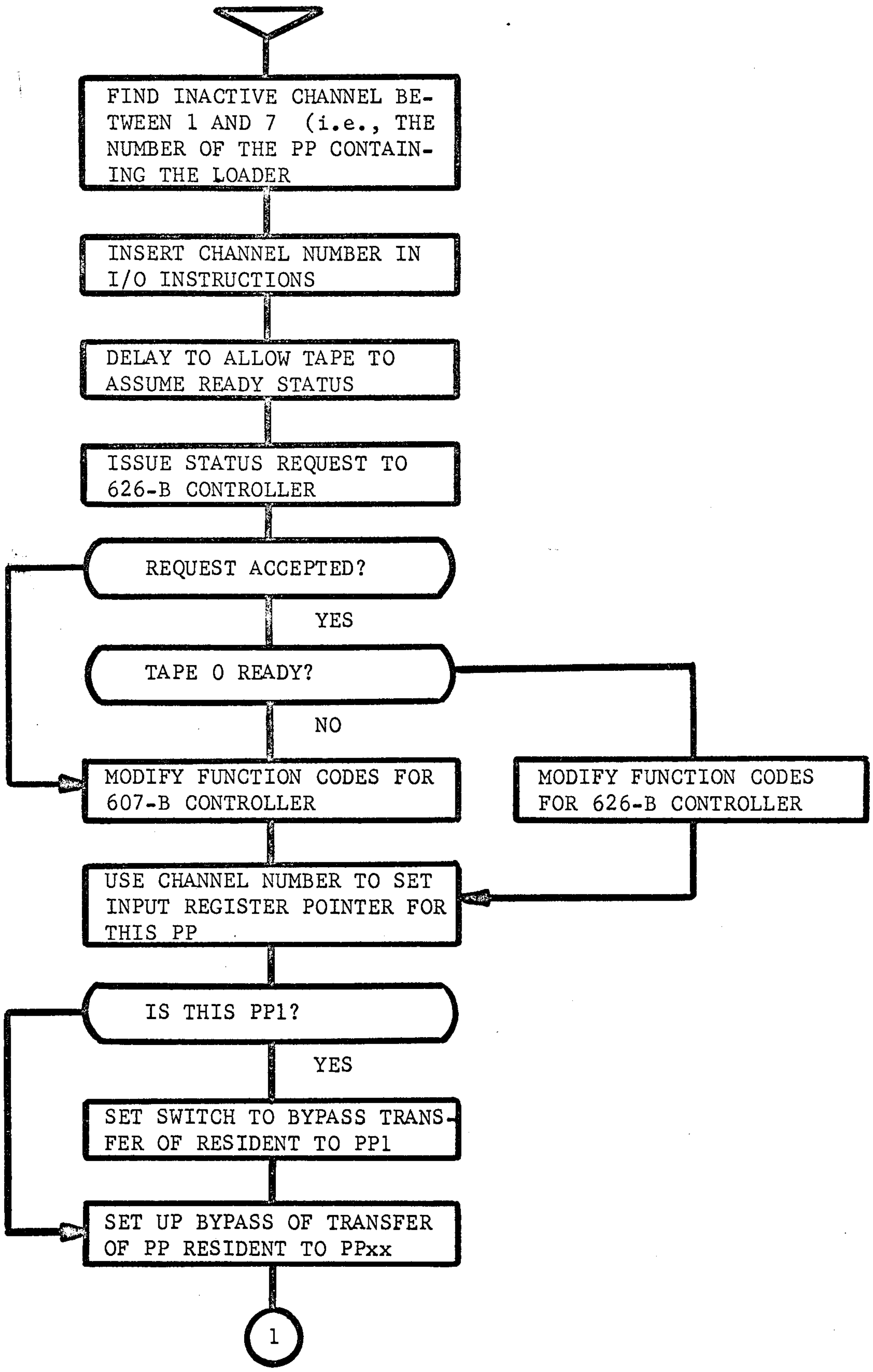
When a half track is filled during library transfers to the disk, the loader program requests a new half track from MTR via its (the loader's) resident program. The resident's POSITION DISK routine is used to position the disk to the new half track position.

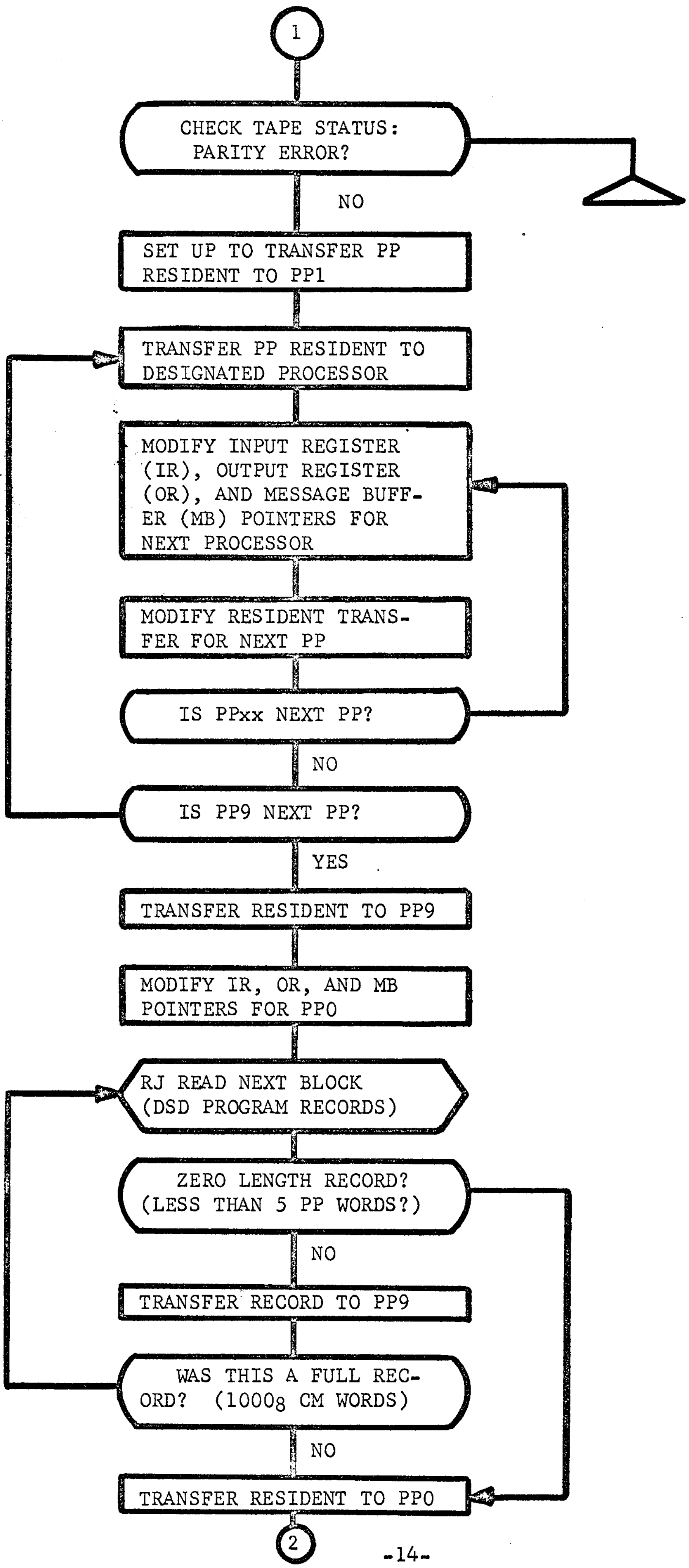
When the transfer of the central library to the disk is completed, the loader program exits to the idle loop of its resident.

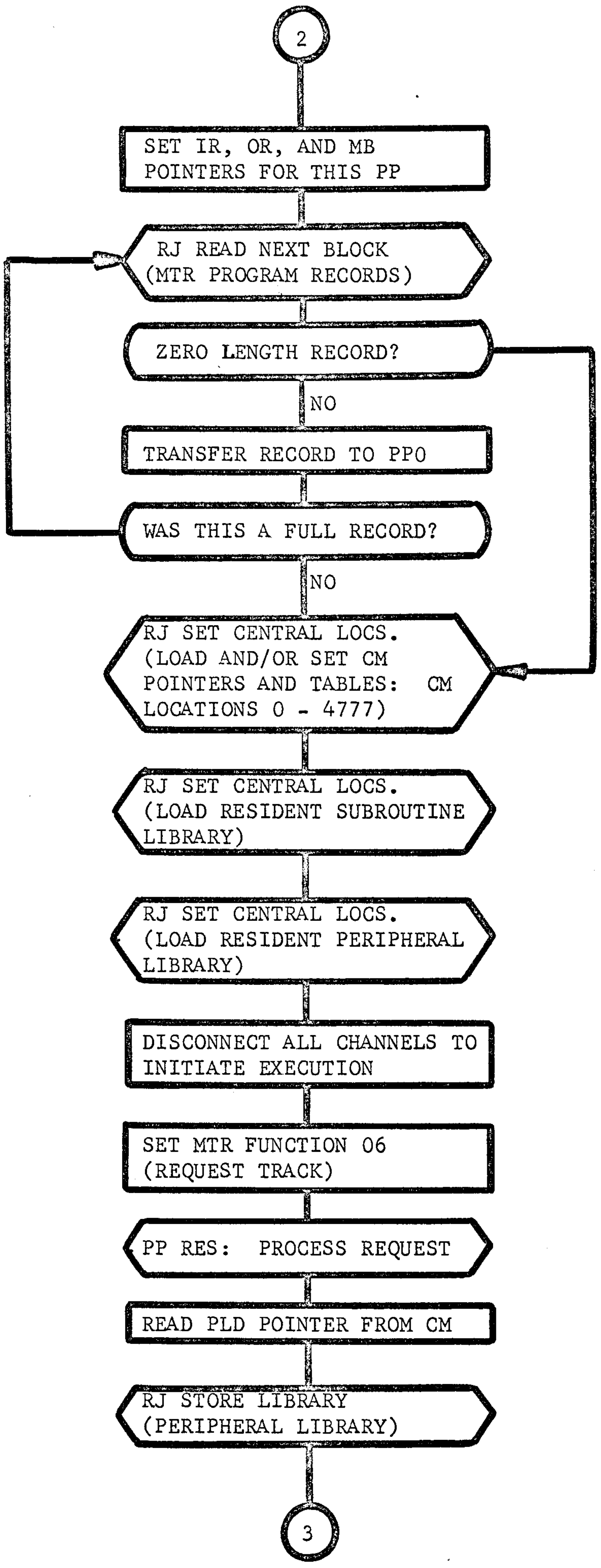
Notes:

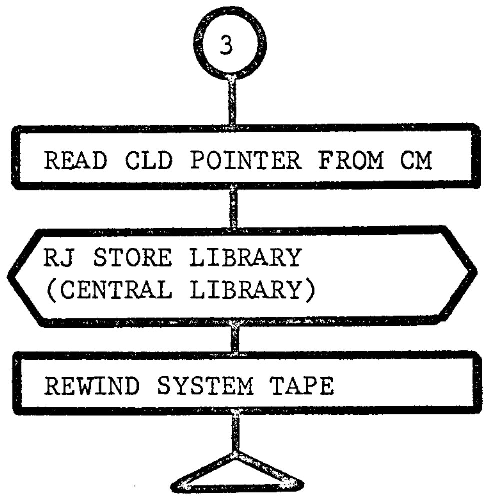
1. For the loading process described, the system tape should be mounted on unit zero of the first controller on channel xx. If channel xx has both 607-B and 626-B controllers, the unused controller's unit zero should be made not ready. The channel xx may be any channel from 1 to 9.
2. In addition to the bootstrap routine described, a variety of others are in use. Many of these use a one-card loader.

SYSTEM TAPE LOADER

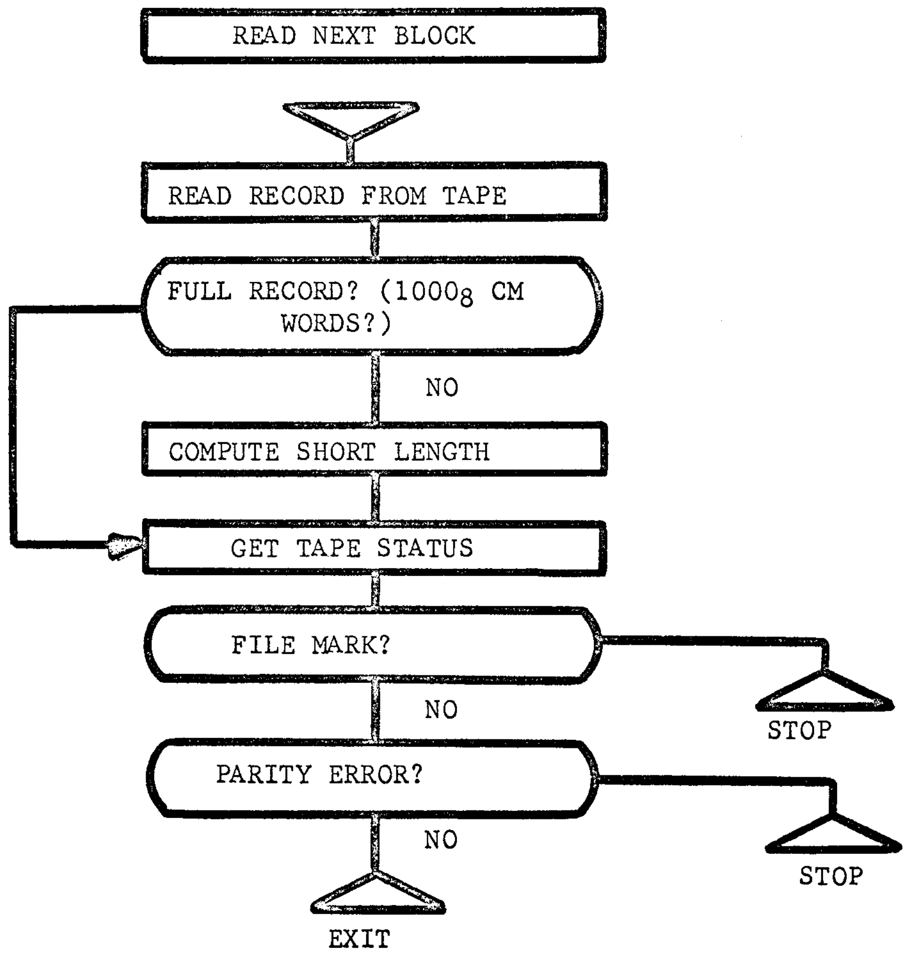

FIND INACTIVE CHANNEL BETWEEN 1 AND 7 (i.e., THE NUMBER OF THE PP CONTAINING THE LOADER
INSERT CHANNEL NUMBER IN I/O INSTRUCTIONS
DELAY TO ALLOW TAPE TO ASSUME READY STATUS
ISSUE STATUS REQUEST TO 626-B CONTROLLER
REQUEST ACCEPTED?
YES
TAPE 0 READY?
NO
MODIFY FUNCTION CODES FOR 607-B CONTROLLER
MODIFY FUNCTION CODES FOR 626-B CONTROLLER
USE CHANNEL NUMBER TO SET INPUT REGISTER POINTER FOR THIS PP
IS THIS PP1?
YES
SET SWITCH TO BYPASS TRANSFER OF RESIDENT TO PP1
SET UP BYPASS OF TRANSFER OF PP RESIDENT TO PPxx
1

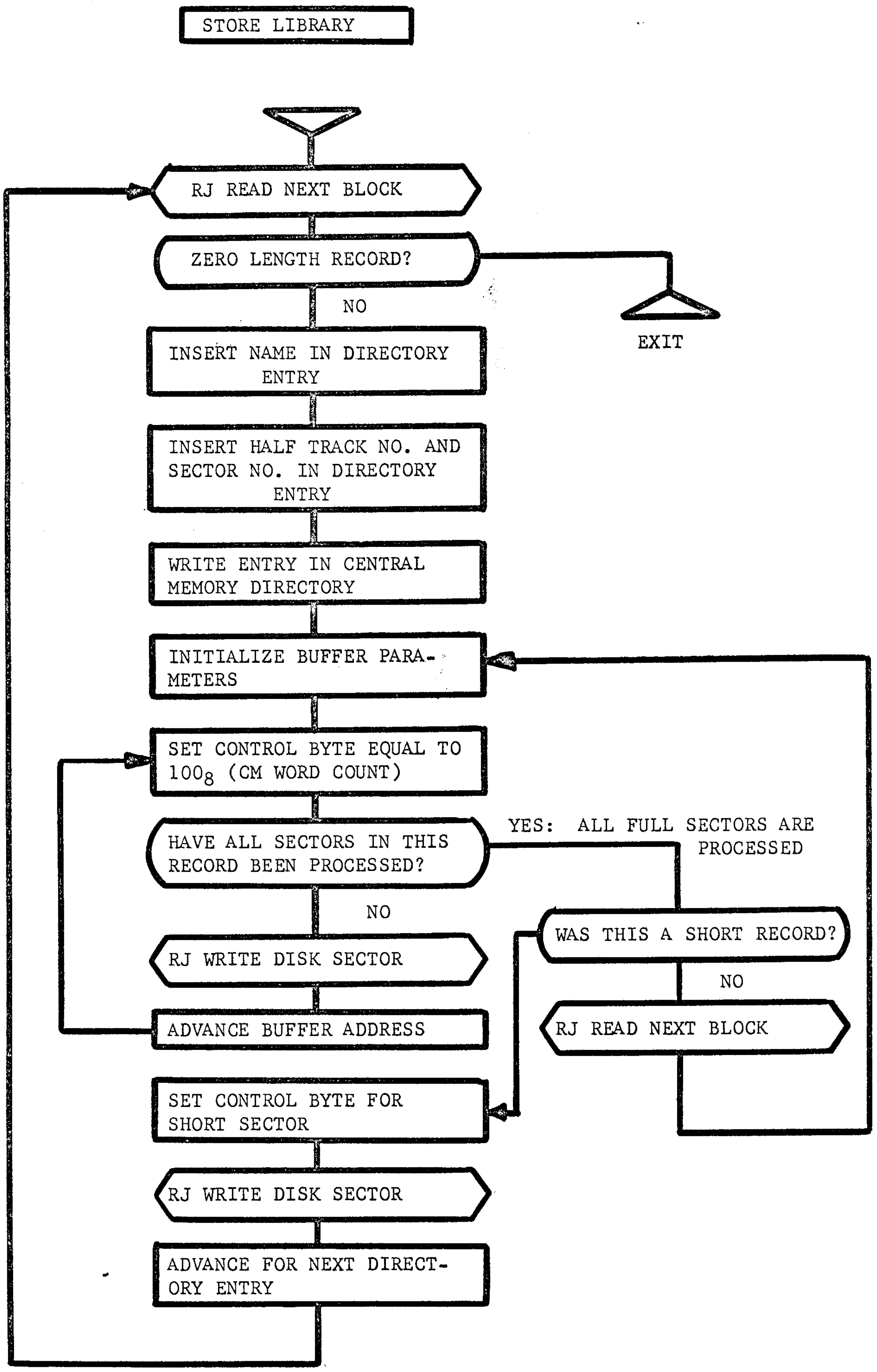

1
CHECK TAPE STATUS: PARITY ERROR?
NO
SET UP TO TRANSFER PP RESIDENT TO PP1
TRANSFER PP RESIDENT TO DESIGNATED PROCESSOR
MODIFY INPUT REGISTER (IR), OUTPUT REGISTER (OR), AND MESSAGE BUFFER (MB) POINTERS FOR NEXT PROCESSOR
MODIFY RESIDENT TRANSFER FOR NEXT PP
IS PPxx NEXT PP?
NO
IS PP9 NEXT PP?
YES
TRANSFER RESIDENT TO PP9
MODIFY IR, OR, AND MB POINTERS FOR PP0
RJ READ NEXT BLOCK (DSD PROGRAM RECORDS)
ZERO LENGTH RECORD? (LESS THAN 5 PP WORDS?)
NO
TRANSFER RECORD TO PP9
WAS THIS A FULL RECORD? (1000_8 CM WORDS)
NO
TRANSFER RESIDENT TO PP0
2


2
SET IR, OR, AND MB POINTERS FOR THIS PP
RJ READ NEXT BLOCK (MTR PROGRAM RECORDS)
ZERO LENGTH RECORD?
NO
TRANSFER RECORD TO PPO
WAS THIS A FULL RECORD?
NO
RJ SET CENTRAL LOCS. (LOAD AND/OR SET CM POINTERS AND TABLES: CM LOCATIONS 0 - 4777)
RJ SET CENTRAL LOCS. (LOAD RESIDENT SUBROUTINE LIBRARY)
RJ SET CENTRAL LOCS. (LOAD RESIDENT PERIPHERAL LIBRARY)
DISCONNECT ALL CHANNELS TO INITIATE EXECUTION
SET MTR FUNCTION 06 (REQUEST TRACK)
PP RES: PROCESS REQUEST
READ PLD POINTER FROM CM
RJ STORE LIBRARY (PERIPHERAL LIBRARY)
3

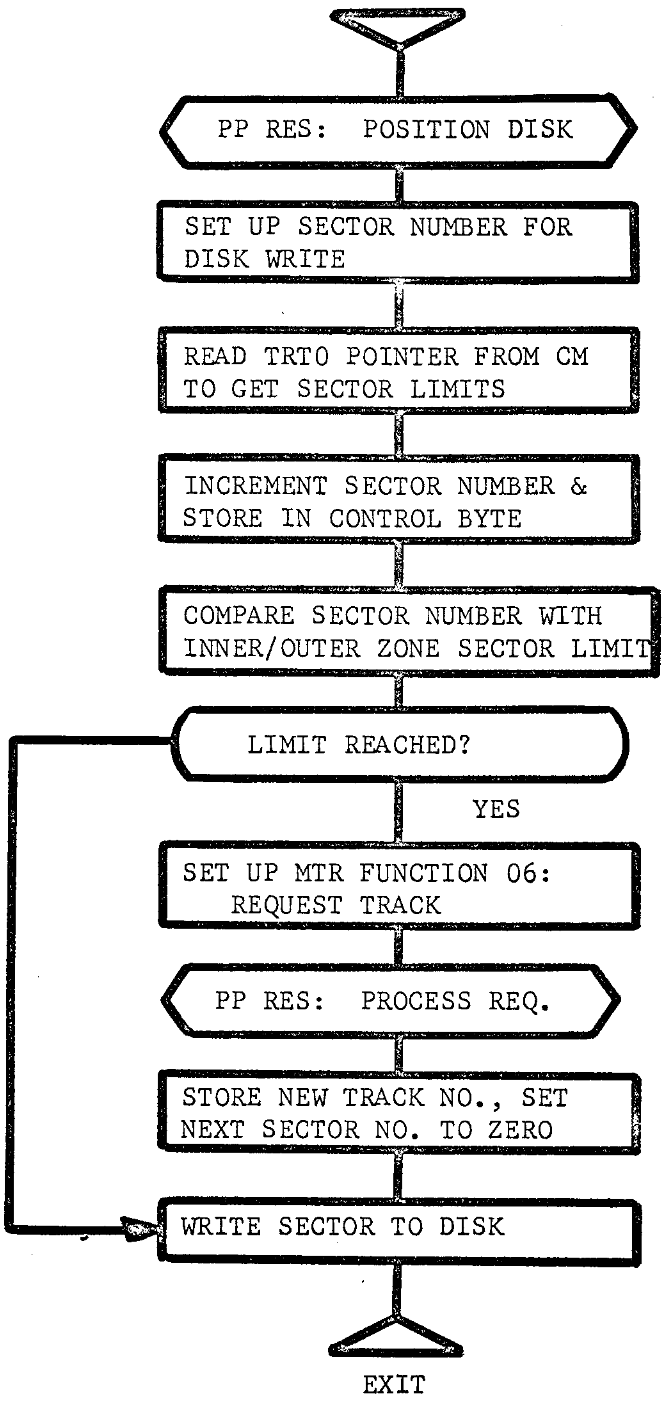


EXIT TO PPxx RESIDENT
IDLE LOOP

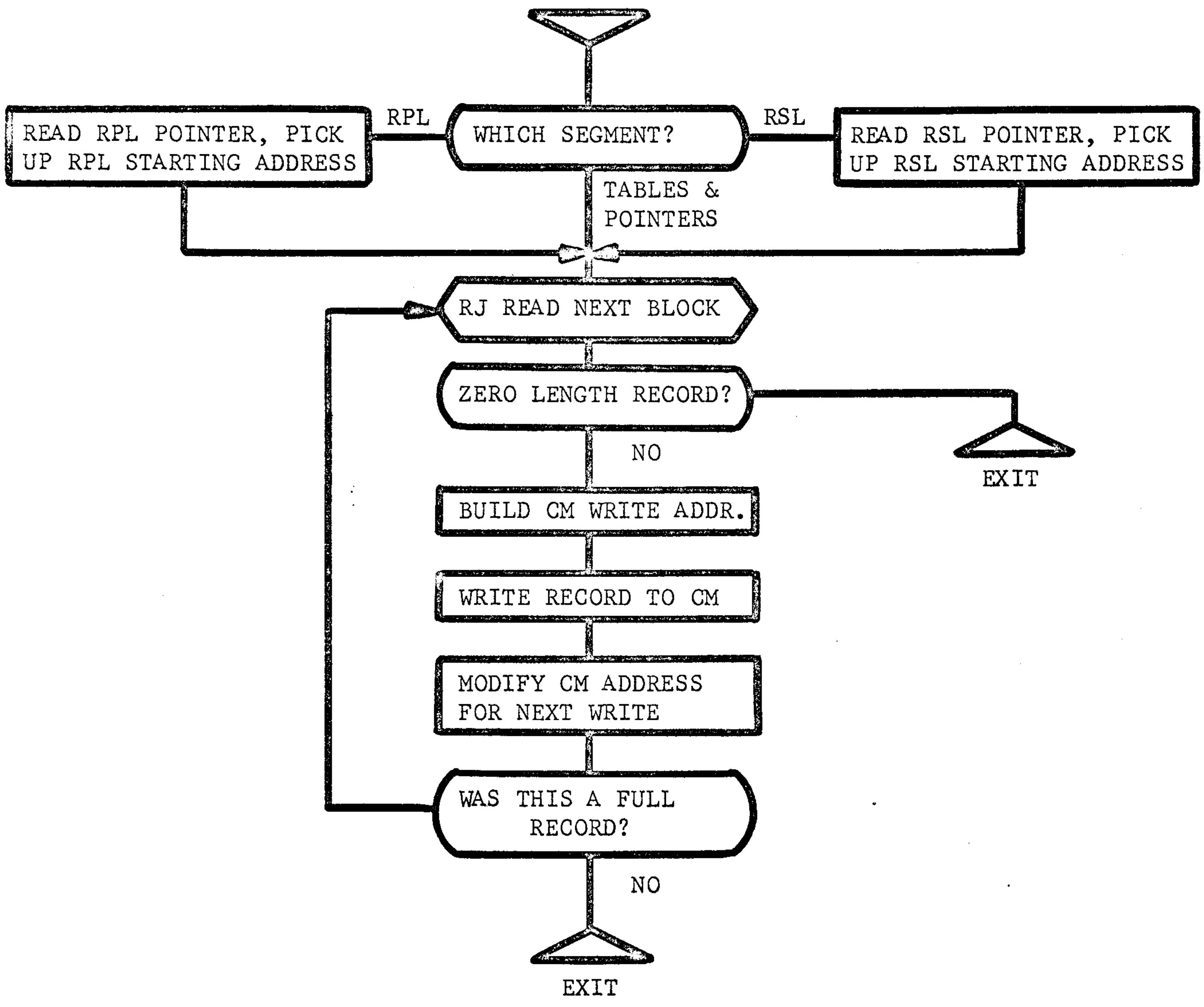



STORE LIBRARY
RJ READ NEXT BLOCK
ZERO LENGTH RECORD?
NO
EXIT
INSERT NAME IN DIRECTORY ENTRY
INSERT HALF TRACK NO. AND SECTOR NO. IN DIRECTORY ENTRY
WRITE ENTRY IN CENTRAL MEMORY DIRECTORY
INITIALIZE BUFFER PARAMETERS
SET CONTROL BYTE EQUAL TO 100_8 (CM WORD COUNT)
HAVE ALL SECTORS IN THIS RECORD BEEN PROCESSED?
YES: ALL FULL SECTORS ARE PROCESSED
NO
RJ WRITE DISK SECTOR
ADVANCE BUFFER ADDRESS
WAS THIS A SHORT RECORD?
NO
RJ READ NEXT BLOCK
SET CONTROL BYTE FOR SHORT SECTOR
RJ WRITE DISK SECTOR
ADVANCE FOR NEXT DIRECTORY ENTRY

WRITE DISK SECTOR


PP RES: POSITION DISK
SET UP SECTOR NUMBER FOR DISK WRITE
READ TRTO POINTER FROM CM TO GET SECTOR LIMITS
INCREMENT SECTOR NUMBER & STORE IN CONTROL BYTE
COMPARE SECTOR NUMBER WITH INNER/OUTER ZONE SECTOR LIMIT
LIMIT REACHED?
YES
SET UP MTR FUNCTION 06: REQUEST TRACK
PP RES: PROCESS REQ.
STORE NEW TRACK NO., SET NEXT SECTOR NO. TO ZERO
WRITE SECTOR TO DISK
EXIT

SET CENTRAL LOCATIONS



CONTROL DATA CORPORATION

Development Division - Applications

POOL PROCESSORS AND PERIPHERAL PROCESSOR RESIDENTS

Chippewa Operating System

10/15/65
REV. 1

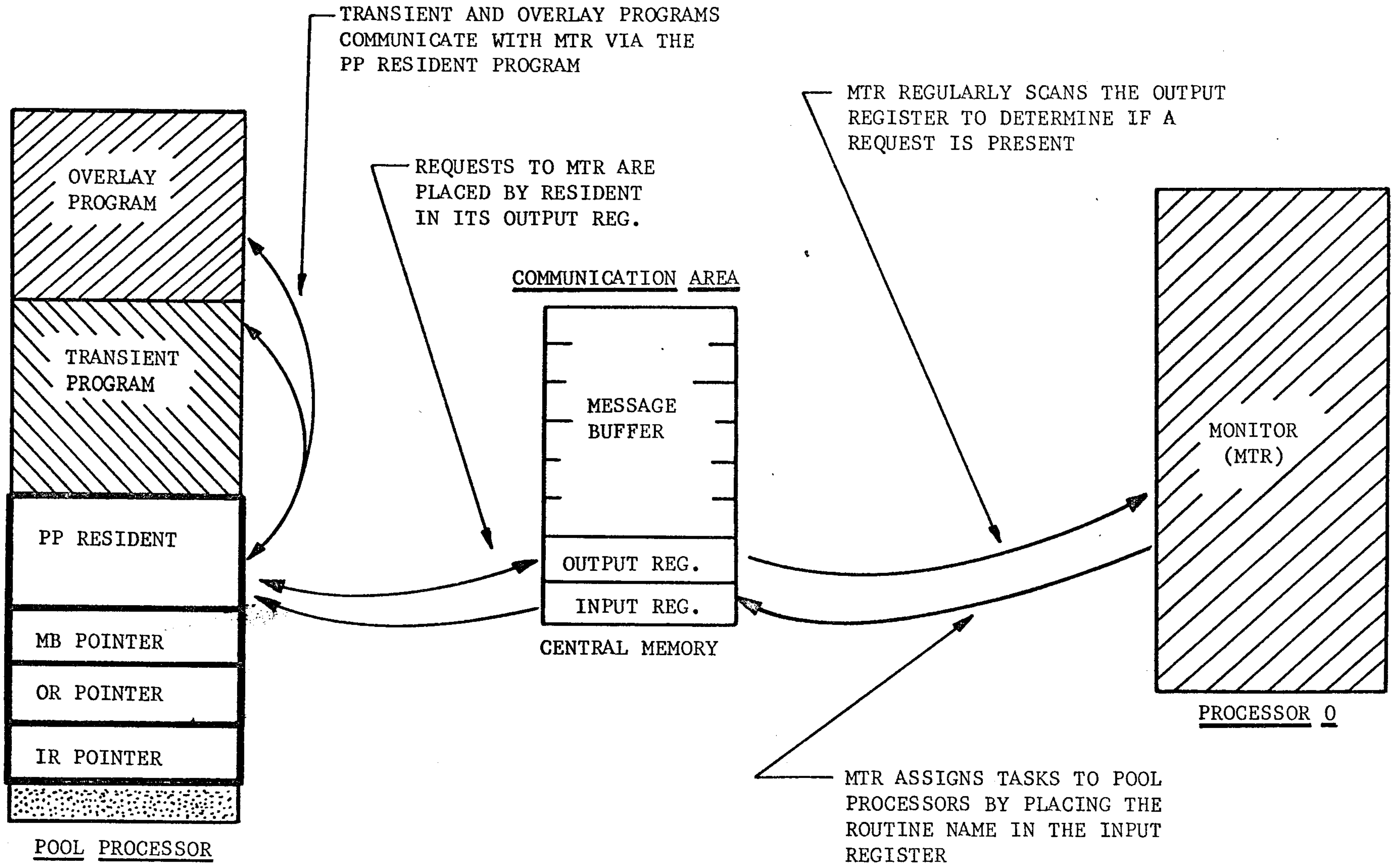
POOL PROCESSORS AND PERIPHERAL PROCESSOR RESIDENTS

INTRODUCTION

In the Chippewa Operating System, the System Display program (DSD) and the Monitor program (MTR) permanently reside in two of the ten peripheral processors. MTR and DSD reside in processors 0 and 9, respectively. The remaining processors, 1 - 8, form a pool of processors to which MTR may assign tasks as required. These pool processors have no fixed assignments: any processor may be assigned to the execution of any system routine, and it is possible that more than one processor may be executing the same routine at the same time. All ten processors contain a small resident program which handles the communications between pool processor programs and the Monitor, and initiates the execution of these programs as directed by MTR.

POOL PROCESSOR STRUCTURE

The structure of a pool processor is illustrated in figure 1. The resident program is contained in locations 0100 - 0772: locations 75, 76, and 77 contain pointers to the Input Register, the Output Register, and Message Buffer in central memory. When directed to do so by MTR, the resident loads a program into its memory and executes it: since that program remains in that processor only for the period of time required to perform its function, it is called a transient program. Transient programs occupy locations 0773 - 1772, although the first instruction is at location 1000. Transient programs generally load overlays to perform specific tasks. For example, CIO, which is a transient program, calls various overlays depending on the task (read, write, backspace) and the equipment (disk, tape, etc.) specified. Overlays are loaded into memory beginning


TRANSIENT AND OVERLAY PROGRAMS COMMUNICATE WITH MTR VIA THE PP RESIDENT PROGRAM
MTR REGULARLY SCANS THE OUTPUT REGISTER TO DETERMINE IF A REQUEST IS PRESENT
REQUESTS TO MTR ARE PLACED BY RESIDENT IN ITS OUTPUT REG.
OVERLAY PROGRAM
TRANSIENT PROGRAM
PP RESIDENT
MB POINTER
OR POINTER
IR POINTER
POOL PROCESSOR
COMMUNICATION AREA
MESSAGE BUFFER
OUTPUT REG.
INPUT REG.
CENTRAL MEMORY
MONITOR (MTR)
PROCESSOR 0
MTR ASSIGNS TASKS TO POOL PROCESSORS BY PLACING THE ROUTINE NAME IN THE INPUT REGISTER

POOL PROCESSORS & PP RESIDENT

Figure 1

at location 1773: the first instruction falls at location 2000. Overlays are generally entered via a return jump. Transient programs have names beginning with a letter (CIO, EXU) or the numeral 1 (1BJ, 1LT): overlays have names beginning with the numeral 2 (2WD, 2BP, etc.).

Both transient and overlay programs, as well as the resident program, make extensive use of the low core locations 01 - 74.

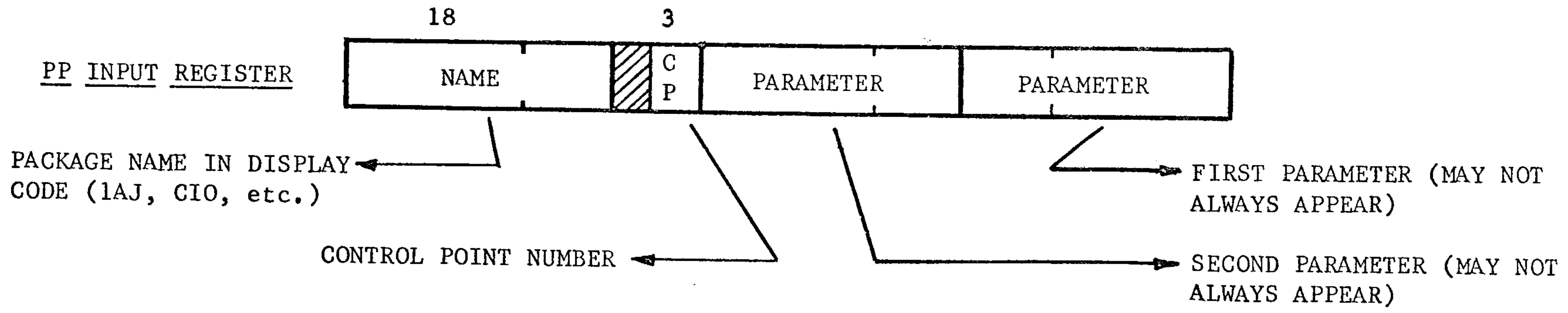
THE RESIDENT

The peripheral processor resident program has two main functions to perform:

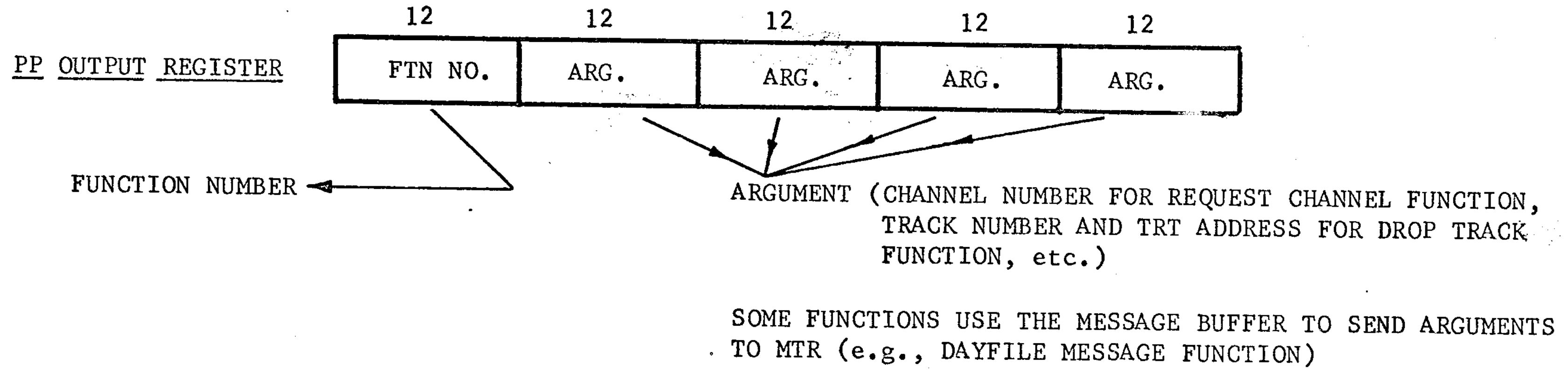
- all communication between MTR and the transient or overlay programs is handled by the resident;
- the resident, when directed by MTR, loads transient programs from either the RPL or the disk library and initiates the execution of these programs.

Communication between MTR and the resident programs is carried out through the use of ten communication areas in central memory, one for each processor. (Note: MTR on occasion communicates with itself by this means.) Each communication area consists of a one-word Input Register, a one-word Output Register, and a six-word Message Buffer. Pool processors address these areas by means of pointers in locations 75 - 77.

MTR assigns a task to a pool processor by placing the request in the processor's Input Register. The format of the request is shown in figure 2. The name of the program package which is to be loaded and executed appears in the high-order 18 bits of the Input Register. This name consists of three display code characters, such as 1AJ, CIO, etc. The number of the control point to which this package is assigned appears in the low-order three bits of byte 2 of the Input Register. Package parameters, such as the address of arguments required by the package, appear in the low-order 36 bits of the Input Register. The request remains in the Input Register until the task is completed. On completion of a task, the transient program requests MTR to release the processor: MTR then clears the processor's


18
3
PP INPUT REGISTER
NAME
C P
PARAMETER
PARAMETER
PACKAGE NAME IN DISPLAY CODE (1AJ, CIO, etc.)
CONTROL POINT NUMBER
FIRST PARAMETER (MAY NOT ALWAYS APPEAR)
SECOND PARAMETER (MAY NOT ALWAYS APPEAR)

MTR REQUEST TO RESIDENT


12
12
12
12
12
PP OUTPUT REGISTER
FTN NO.
ARG.
ARG.
ARG.
ARG.
FUNCTION NUMBER
ARGUMENT (CHANNEL NUMBER FOR REQUEST CHANNEL FUNCTION, TRACK NUMBER AND TRT ADDRESS FOR DROP TRACK FUNCTION, etc.)
SOME FUNCTIONS USE THE MESSAGE BUFFER TO SEND ARGUMENTS TO MTR (e.g., DAYFILE MESSAGE FUNCTION)

RESIDENT REQUEST TO MTR

Figure 2

Input Register. The Input Register of a pool processor is thus clear only when the processor is idle. When MTR needs a pool processor to assign to a task, it searches the communication areas for a cleared Input Register: when one is found, the corresponding processor is assigned to the task.

All communication between the Monitor and the transient and overlay programs is handled by the resident program. MTR performs a variety of functions, each of which is identified by a function code of one or two octal digits. Some of these functions are listed below:

Code	Function
1	Process Dayfile Message
2	Request Channel
7	Drop Track
12	Release PP
33	Assign Equipment

To transmit a request to MTR, the resident places the request in its Output Register. The format of this request is illustrated in figure 2. Byte 1 of the Output Register contains the function code in the low-order bit positions. Bytes 2 - 5 are used for arguments: the number of argument bytes depends on the particular function. Thus, for a Request Channel function (function number 2), the channel number is placed in byte 2. For a Drop Track function, byte 2 contains the address of the Track Reservation Table and byte 3 contains the half track number. For some functions, the function arguments are placed in the Message Buffer and only the function code appears in the Output Register.

MTR regularly scans the Output Register of each processor to determine if a request is present. When the request has been detected, analyzed, and processed, MTR clears the Output Register. The resident, after placing the request in the Output Register, waits for the Output Register to be cleared before proceeding.

Some functions require that information be returned by MTR to the requesting program: for example, the Request Track function (function number 6) returns a half track number to the requestor. MTR places any information to be sent to the requestor in the Message Buffer. The resident returns control to the requesting transient or overlay program when it detects that the Output Register has been cleared by MTR: the requesting program then reads the Message Buffer to obtain the required information.

The resident contains a routine called Process Request which handles the transmission of function requests to MTR. The Process Request routine uses locations 10 - 14 in peripheral processor memory as temporary storage for the request to be written in the Output Register. A peripheral processor program may utilize this routine by placing the arguments for the function in bytes 11 and 12, setting the A register with the function number, and executing a return jump to the Process Request routine at location 761. The Process Request routine will enter the function number in location 10 and write the contents of locations 10 - 14 in the Output Register. Control will be returned to the requesting program upon MTR's clearing the Output Register.

THE RESIDENT PROGRAM

When a pool processor program completes execution, it exits to location 100, which is the address of the resident idle loop. In this idle loop, the processor's Input Register is scanned at intervals of slightly greater than 125 microseconds until a request is found in the Input Register. The delay between successive scans avoids unnecessary memory and read pyramid conflicts. When a request is detected, the resident stores the routine name and the control point number. It then sends function code 17, Pause for Storage Relocation, to MTR and waits for MTR to clear the Output Register before continuing. Should MTR be in the process of relocating the storage assigned to this control point, the Output Register clear will be delayed until relocation is complete. The resident then searches the RPL for the requested routine: if found, the package is read from the resident library into the processor's memory beginning at location 773, and resident turns control over to this routine by jumping to location 1000. If the routine name was not found in the RPL, resident then initiates a search of the PLD. If the routine is in the disk library, the resident loads it from the disk into its memory at location 773, and jumps to it to begin execution. If the routine is not found in the PLD, the resident enters the message "XXX NOT IN PPLIB" in the dayfile, and requests MTR to abort the job which called the routine. The resident then returns to its idle loop.

In loading a program from the disk, resident begins by reserving channel 0 via the appropriate MTR function request. Next, resident compares the track number of the requested routine with the current position of the disk as contained in the TRT pointer word for disk 0. Repositioning and/or head group switching is done only if necessary. Once the disk has been properly positioned, the sectors composing the desired routine are read into peripheral processor memory. The end of the routine is indicated when a short record (less than 100_8 central memory words) is read. If a parity error is detected, the sector in which the error occurred is reread twice, each time at a different clipping level. Should these reads also fail, the resident enters the message "DISK 0 PARITY ERROR Gx Txxx Sxxx" in the dayfile and then stops (via a UJN 0 instruction). A dead start load is necessary to renew systems operation.

Several resident routines are used by transient and overlay programs. These routines are described below.

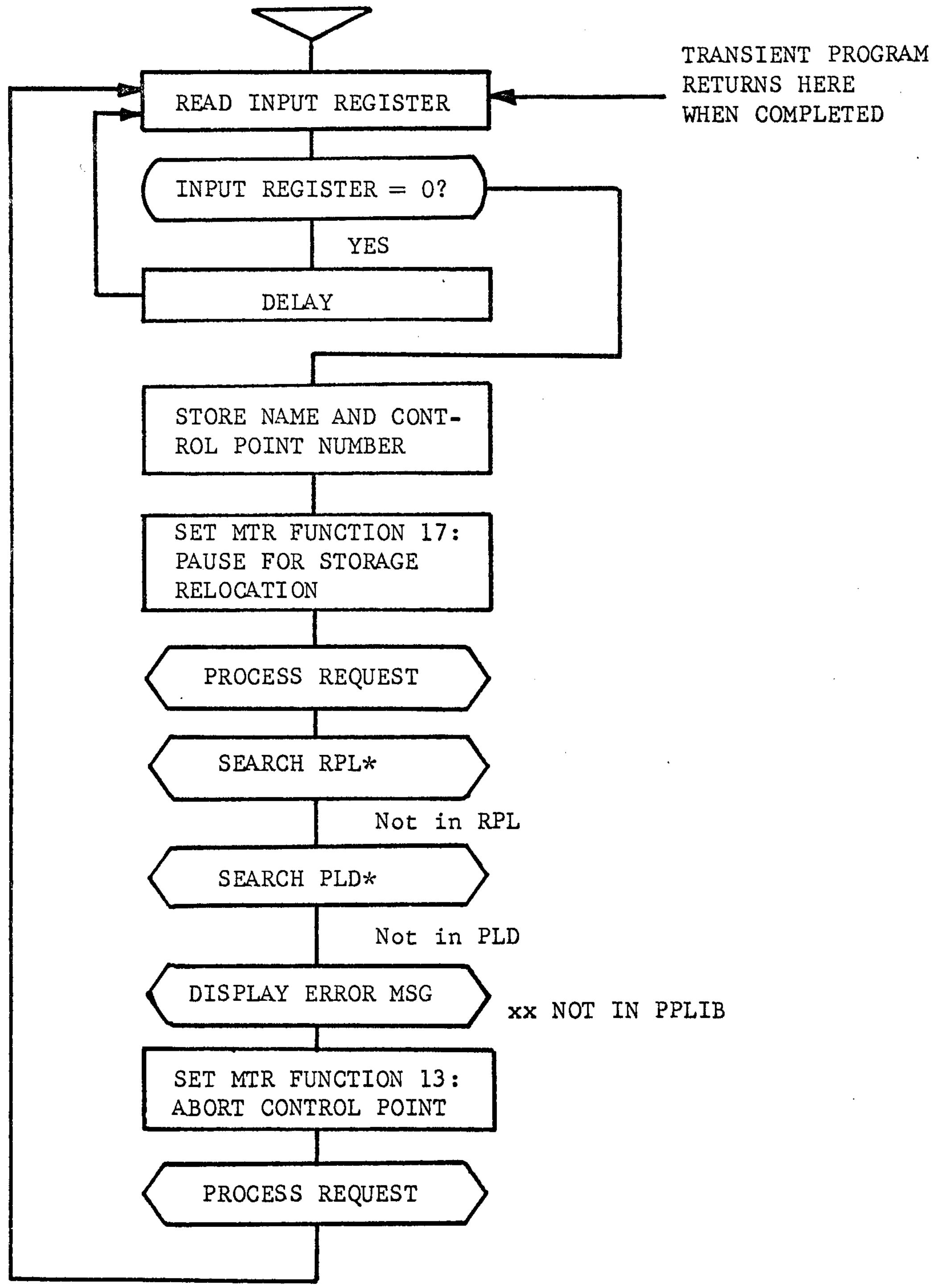
Address	Routine	Entry Conditions	Description
761	Process Request	Function number in A register	Enters function number in location 10, and writes locations 10 - 14 to the Output Register. Exits when the Output Register has been cleared

Address	Routine	Entry Conditions	Description
741	Request Channel	Channel number in A register	Stores channel number in location 11, sets function code 2 in A register, and jumps to Process Request
751	Drop Channel	Channel number in A register	Stores channel number in location 11, sets function code 3 in A register, and jumps to Process Request
531	Dayfile Message	Message address in A register	Write message (less than 6 CM words, terminated by a zero byte) in Message Buffer, sets function code 1 in the A register, and jumps to Process Request
701	Position Disk*	Half track number in A register	Repositions heads and/or switches head groups as necessary (for disk 0 only)
200	Disk Parity Error Exit*	Half track number in location 6, sector number in location 7	Enters error message in the dayfile and halts
401	Read Sector from Disk 0*	Read address in A register, half track number in location 6, sector number in location 7	Reads one sector from disk 0 into memory at the designated address. Jumps to Disk Parity Error Exit if an error occurs.

* Not a MTR function

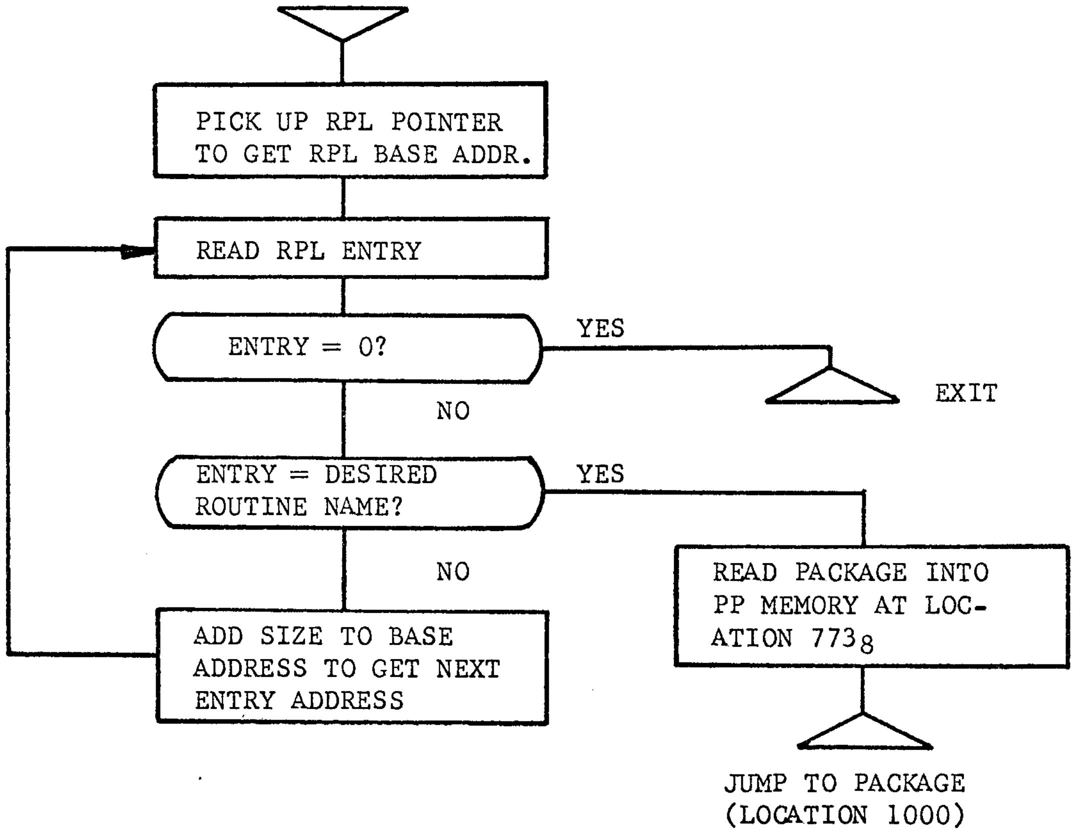
All of the foregoing routines are entered via a return jump instruction to the specified address except the Disk Parity Error Exit, which is entered via a long jump instruction.

PP RESIDENT: IDLE LOOP

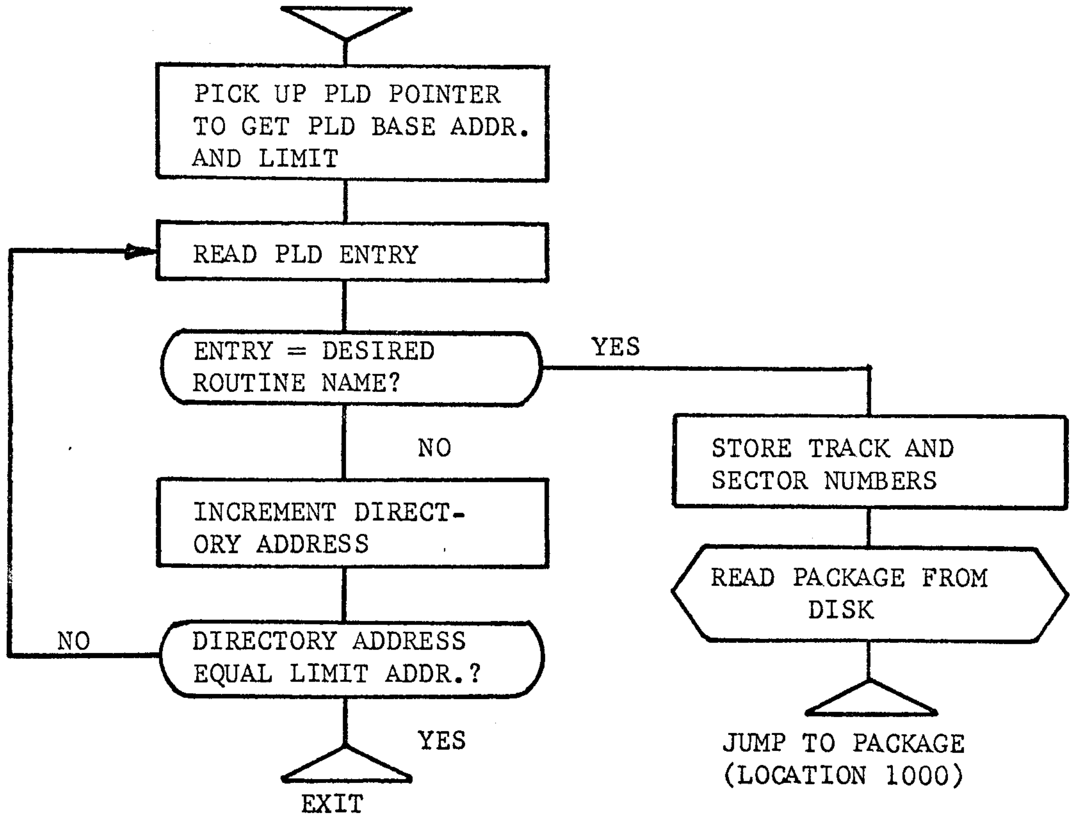

TRANSIENT PROGRAM RETURNS HERE WHEN COMPLETED
READ INPUT REGISTER
INPUT REGISTER = 0?
YES
DELAY
STORE NAME AND CONTROL POINT NUMBER
SET MTR FUNCTION 17: PAUSE FOR STORAGE RELOCATION
PROCESS REQUEST
SEARCH RPL*
Not in RPL
SEARCH PLD*
Not in PLD
DISPLAY ERROR MSG
xx NOT IN PPLIB
SET MTR FUNCTION 13: ABORT CONTROL POINT
PROCESS REQUEST

* IF FOUND, LOAD PROGRAM AND JUMP TO IT

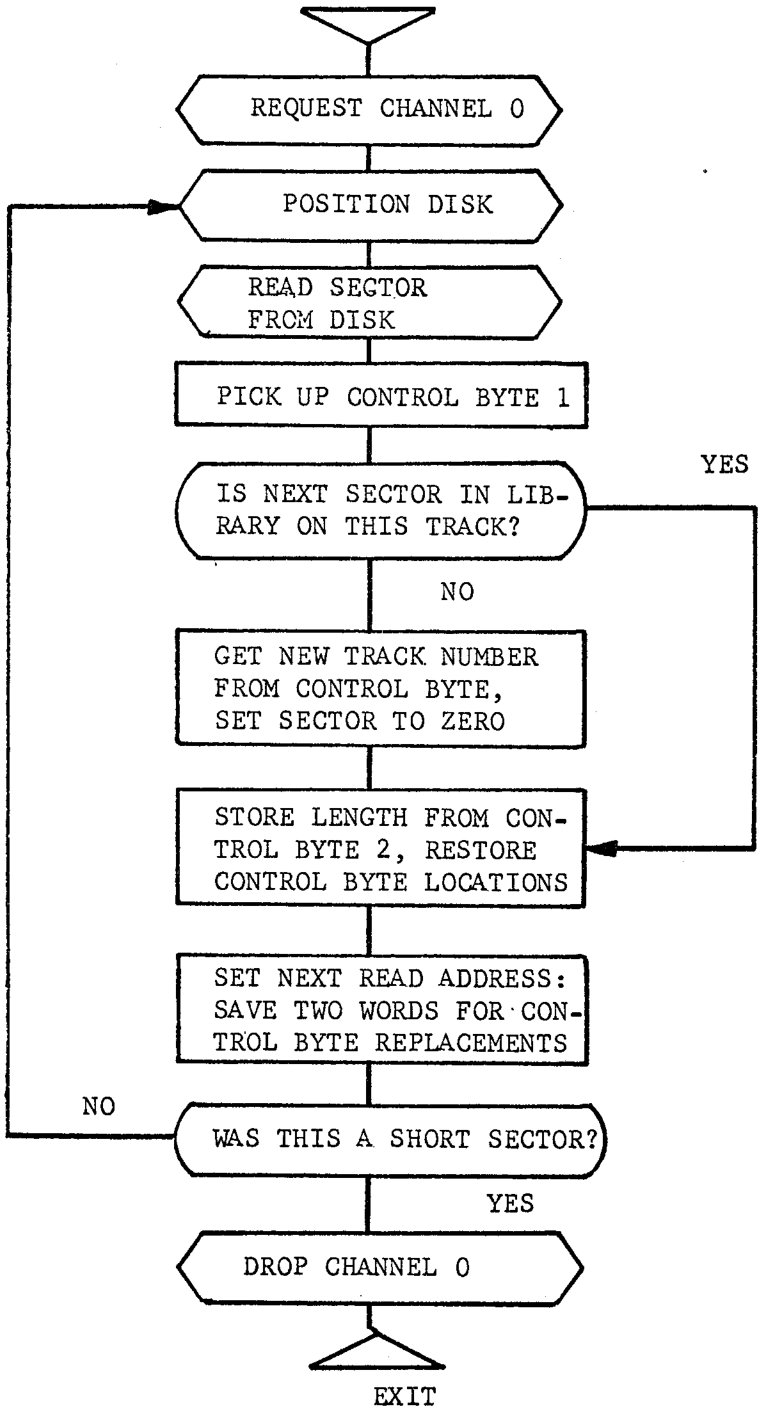
SEARCH RPL


PICK UP RPL POINTER TO GET RPL BASE ADDR.
READ RPL ENTRY
ENTRY = 0?
YES
EXIT
NO
ENTRY = DESIRED ROUTINE NAME?
YES
READ PACKAGE INTO PP MEMORY AT LOCATION 773₈
JUMP TO PACKAGE (LOCATION 1000)
NO
ADD SIZE TO BASE ADDRESS TO GET NEXT ENTRY ADDRESS

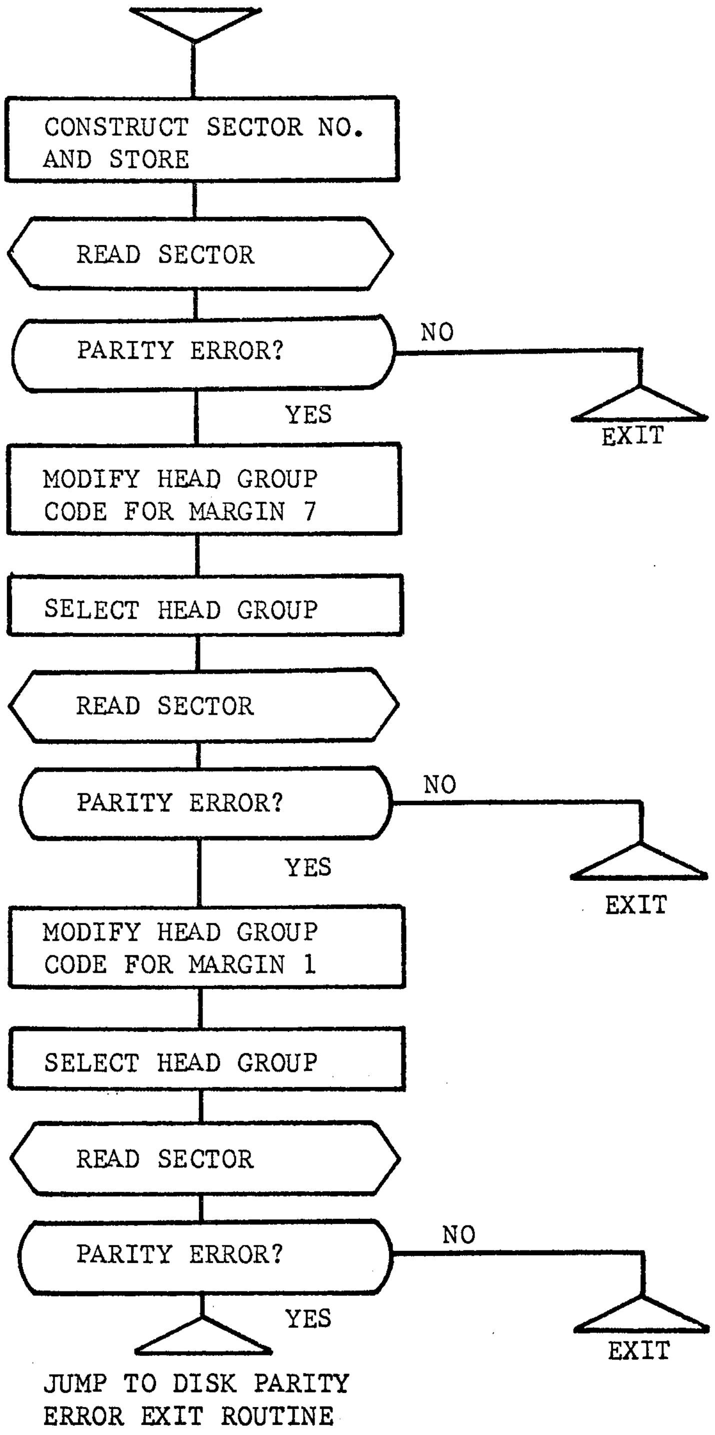
SEARCH PLD


PICK UP PLD POINTER TO GET PLD BASE ADDR. AND LIMIT
READ PLD ENTRY
ENTRY = DESIRED ROUTINE NAME?
YES
STORE TRACK AND SECTOR NUMBERS
READ PACKAGE FROM DISK
JUMP TO PACKAGE (LOCATION 1000)
NO
INCREMENT DIRECTORY ADDRESS
NO
DIRECTORY ADDRESS EQUAL LIMIT ADDR.?
YES
EXIT

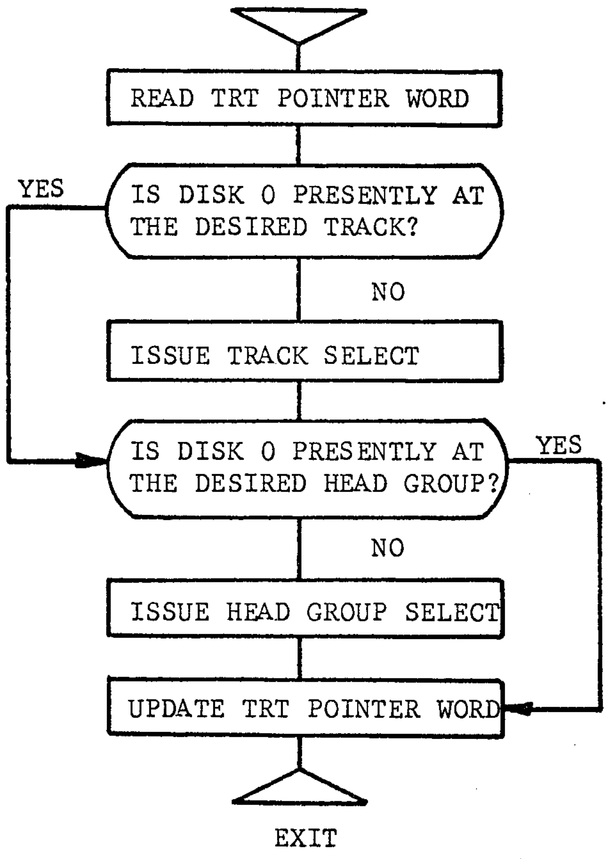
READ PACKAGE FROM DISK


REQUEST CHANNEL 0
POSITION DISK
READ SECTOR FROM DISK
PICK UP CONTROL BYTE 1
IS NEXT SECTOR IN LIBRARY ON THIS TRACK?
YES
NO
GET NEW TRACK NUMBER FROM CONTROL BYTE, SET SECTOR TO ZERO
STORE LENGTH FROM CONTROL BYTE 2, RESTORE CONTROL BYTE LOCATIONS
SET NEXT READ ADDRESS: SAVE TWO WORDS FOR CONTROL BYTE REPLACEMENTS
NO
WAS THIS A SHORT SECTOR?
YES
DROP CHANNEL 0
EXIT

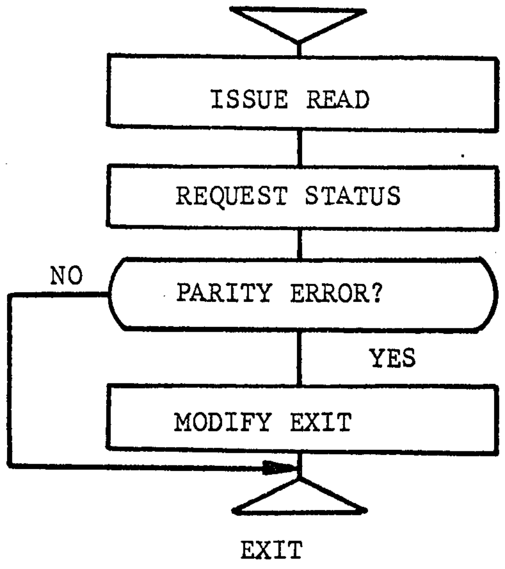
READ SECTOR FROM DISK


CONSTRUCT SECTOR NO. AND STORE
READ SECTOR
PARITY ERROR?
NO
EXIT
YES
MODIFY HEAD GROUP CODE FOR MARGIN 7
SELECT HEAD GROUP
READ SECTOR
PARITY ERROR?
NO
EXIT
YES
MODIFY HEAD GROUP CODE FOR MARGIN 1
SELECT HEAD GROUP
READ SECTOR
PARITY ERROR?
NO
EXIT
YES
JUMP TO DISK PARITY ERROR EXIT ROUTINE

POSITION DISK

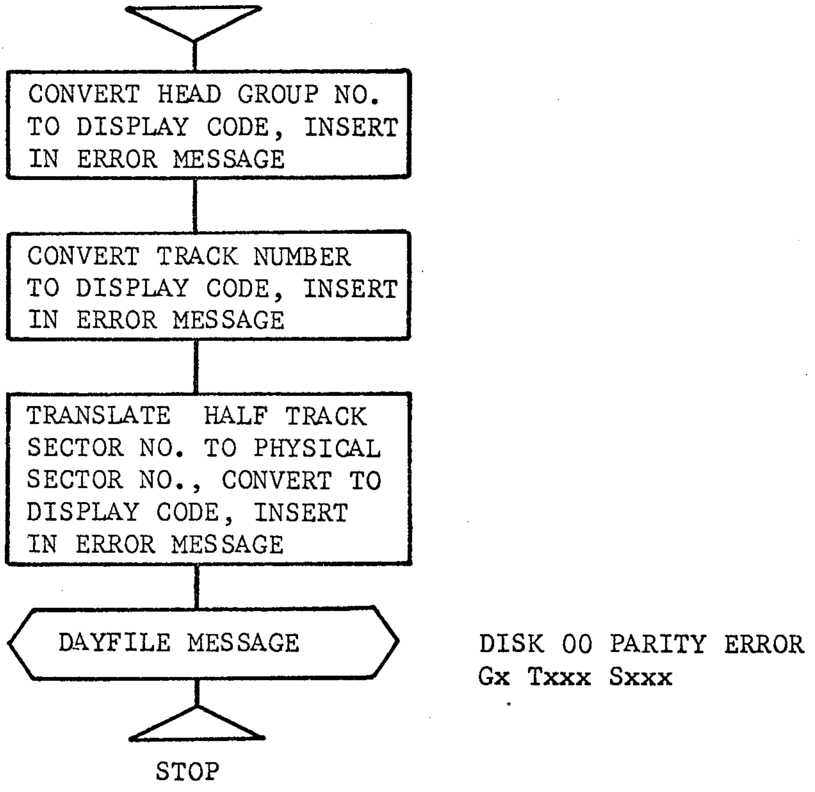


READ SECTOR

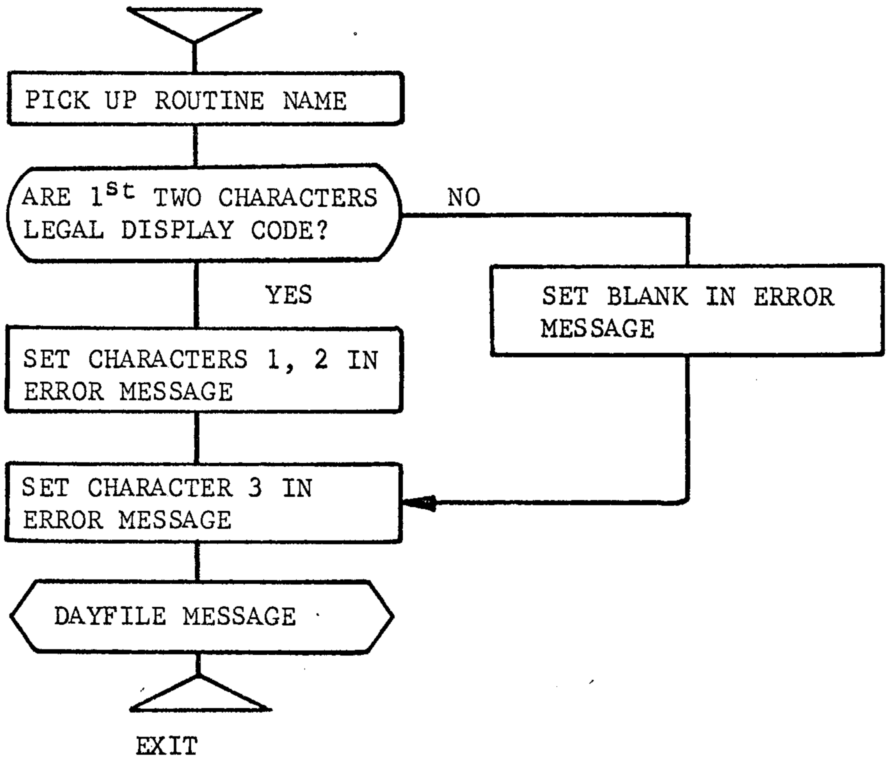


DISK PARITY ERROR EXIT

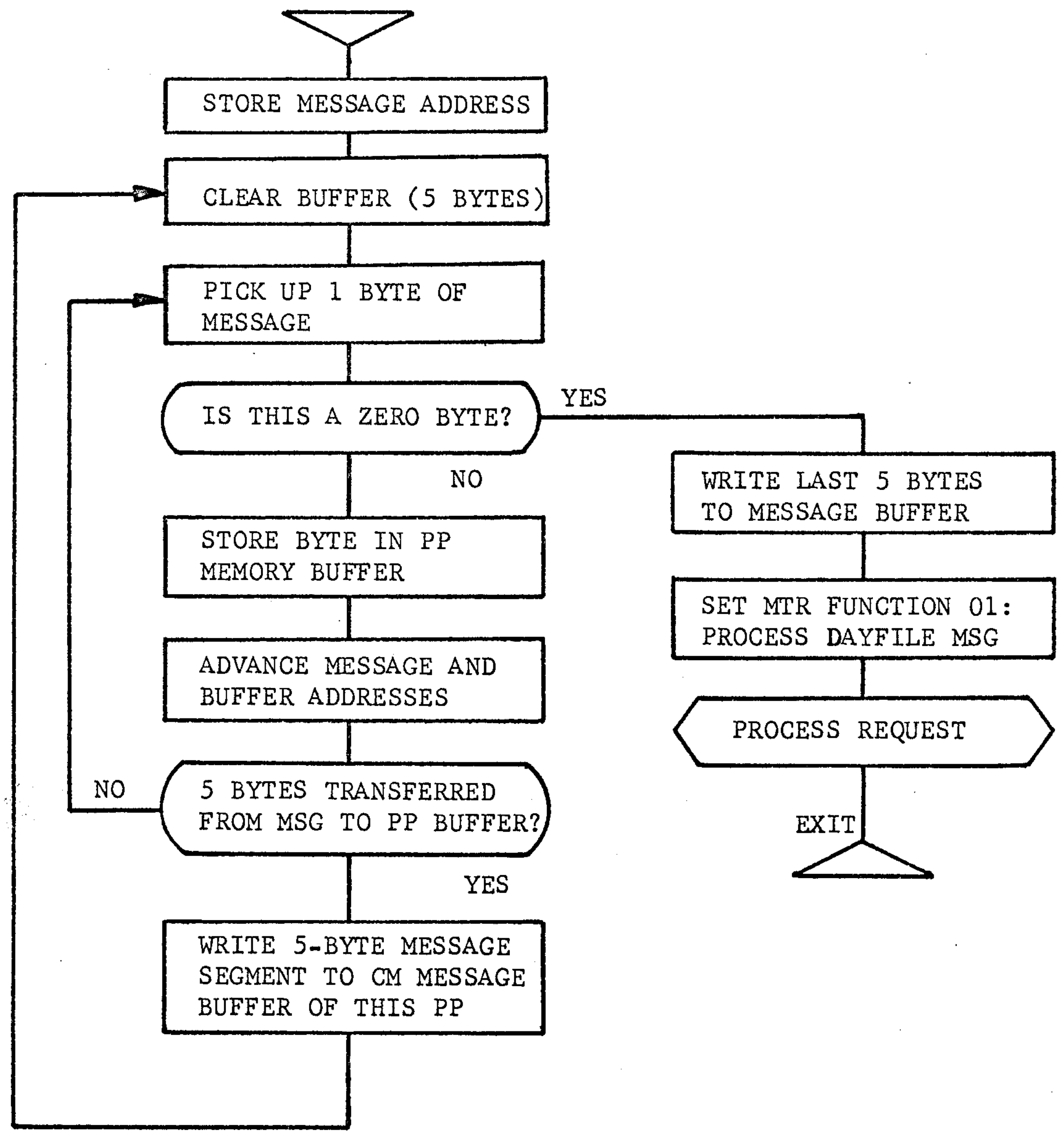
Used by transient programs and overlays as well as by PP Resident


CONVERT HEAD GROUP NO. TO DISPLAY CODE, INSERT IN ERROR MESSAGE
CONVERT TRACK NUMBER TO DISPLAY CODE, INSERT IN ERROR MESSAGE
TRANSLATE HALF TRACK SECTOR NO. TO PHYSICAL SECTOR NO., CONVERT TO DISPLAY CODE, INSERT IN ERROR MESSAGE
DAYFILE MESSAGE
DISK 00 PARITY ERROR Gx Txxx Sxxx
STOP

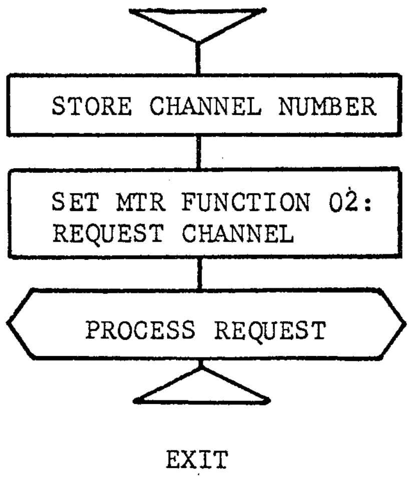
DISPLAY ERROR MESSAGE


PICK UP ROUTINE NAME
ARE 1st TWO CHARACTERS LEGAL DISPLAY CODE?
NO
YES
SET BLANK IN ERROR MESSAGE
SET CHARACTERS 1, 2 IN ERROR MESSAGE
SET CHARACTER 3 IN ERROR MESSAGE
DAYFILE MESSAGE
EXIT

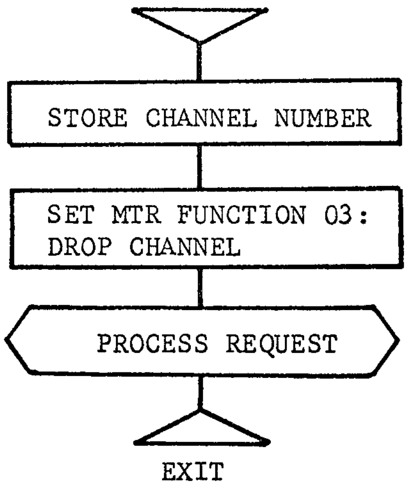
DAYFILE MESSAGE


STORE MESSAGE ADDRESS
CLEAR BUFFER (5 BYTES)
PICK UP 1 BYTE OF MESSAGE
IS THIS A ZERO BYTE?
YES
NO
STORE BYTE IN PP MEMORY BUFFER
ADVANCE MESSAGE AND BUFFER ADDRESSES
NO
5 BYTES TRANSFERRED FROM MSG TO PP BUFFER?
YES
WRITE 5-BYTE MESSAGE SEGMENT TO CM MESSAGE BUFFER OF THIS PP
WRITE LAST 5 BYTES TO MESSAGE BUFFER
SET MTR FUNCTION 01: PROCESS DAYFILE MSG
PROCESS REQUEST
EXIT

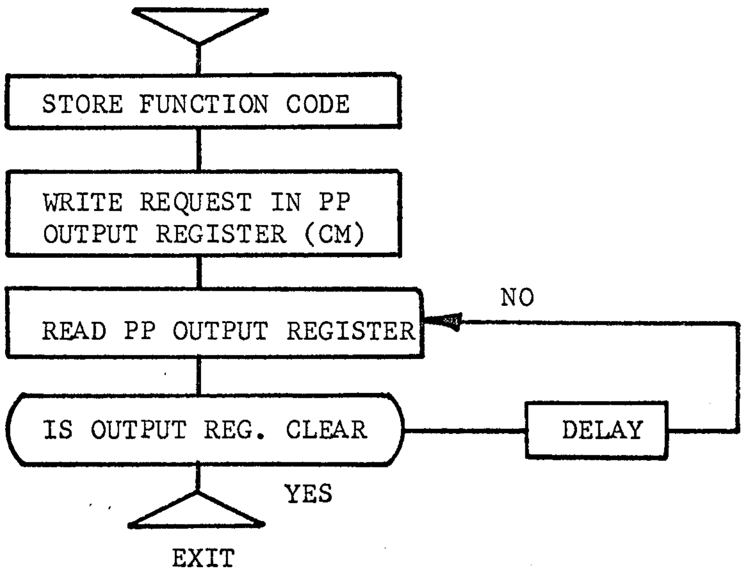
REQUEST CHANNEL



DROP CHANNEL



PROCESS REQUEST



CONTROL DATA CORPORATION

Development Division - Applications

THE SYSTEM MONITOR, MTR

Chippewa Operating System

10/19/65
Rev. 1

THE SYSTEM MONITOR, MTR

CONTENTS

THE SYSTEM MONITOR, MTR

INTRODUCTION

The monitor, or executive, of the Chippewa Operating System is the MTR program, which permanently resides in peripheral processor 0. Among the functions performed by MTR is the allocation of the physical components of the system to various users. The components controlled by MTR include:

- pool processors
- peripheral equipment - tapes, printers, card readers, etc.
- data channels
- disk tracks
- central memory

MTR directs the loading and initiates the execution of central processor programs, monitors central processor programs for I/O requests and assigns these requests to available peripheral processors, and monitors peripheral processor programs for function requests. MTR maintains the time accounting in the system and is responsible for the maintenance of the dayfile.

THE CONTROL POINT CONCEPT

In a multiprogrammed multiprocessor such as the 6600 system, central memory is shared by a number of users. In addition to the active and inactive central processor programs residing in central memory, many peripheral processor programs require central memory buffers. The allocation of central storage to these various users is a function which the operating system can handle in one of two ways:

1. Storage can be allocated to a number of users limited only by

the amount of memory available. This assures the maximum utilization of central memory, but requires an elaborate bookkeeping system. In particular, the manipulation of the variable length tables required, and the relocation of storage to avoid arriving at a "patchquilt" of unallocated memory locations as jobs complete, present interesting design problems.

2. Storage can be allocated to a fixed number of users. If the limit is properly selected, losses in memory utilization efficiency will be minimal. In this method, control of storage allocation and relocation is greatly simplified.

For many job mixes, system throughput is not materially affected by the use of one or the other of the above methods.

The Chippewa Operating System uses the second method described. In the Chippewa Operating System, central memory may be simultaneously shared by up to seven users. For each of the seven users sharing central memory, there is an area in the central memory resident called the control point area. As each user is assigned storage, pertinent information about the user is entered in the control point area: as execution proceeds, entries are made in the control point area to reflect the current status of the user.

The seven control point areas are each 200_8 central memory locations in length, and occupy a portion of the central memory resident between locations 0200 and 1777. The control point areas are numbered one through seven in accordance with their relative (to one another) locations in central memory resident: control point 1 refers to the control point area in locations 0200 - 0377, control point 2 refers to the control point area in locations 0400 - 0577, and so forth. If the information about a user is contained in a given control point area, the user is said to be assigned to that control point.

The user assigned to a control point may be a peripheral processor program, a central processor program, or both: the last case occurs when a central processor program employs a peripheral processor program to perform an input-output operation. Control point assignments are required not only for external users (i.e., jobs) but for many of the operating system programs as well. Thus, the system program which transfers jobs from the card reader to the disk (1LJ) must be assigned to a control point, since a central memory buffer is required.

In many instances, the system packages READ and PRINT will each be assigned to a control point (usually to control points one and two). The READ package loads a job from the card reader and places it on the disk, and the PRINT package prints the output of a job. Each of these packages requires central memory space: the total space required by both packages is $10{,}000_8$ locations. These two packages plus the central memory resident occupy $24{,}000_8$ locations or about $10{,}200_{10}$ locations. This leaves approximately $120{,}000_{10}$ locations to be shared by users assigned to the remaining five control points, which should provide ample storage for a wide variety of jobs. If necessary, the READ and PRINT packages can be dropped to rpovide more capacity.

The control point area is illustrated in figure 1. The first sixteen words of the control point area contain the exchange jump package. If the user assigned to the control point is a peripheral processor program, no use is made of this exchange jump package insofar as this user is concerned. If the user assigned to this control point is a central processor program, this package is set with the appropriate values of P, RA, FL and EM when the program is initiated: as central processor programs are interrupted and restarted, the exchange jump packages for other central processor programs appear here.

Regardless of whether the user assigned to this control point is a central processor program or a peripheral processor program, the storage allocated

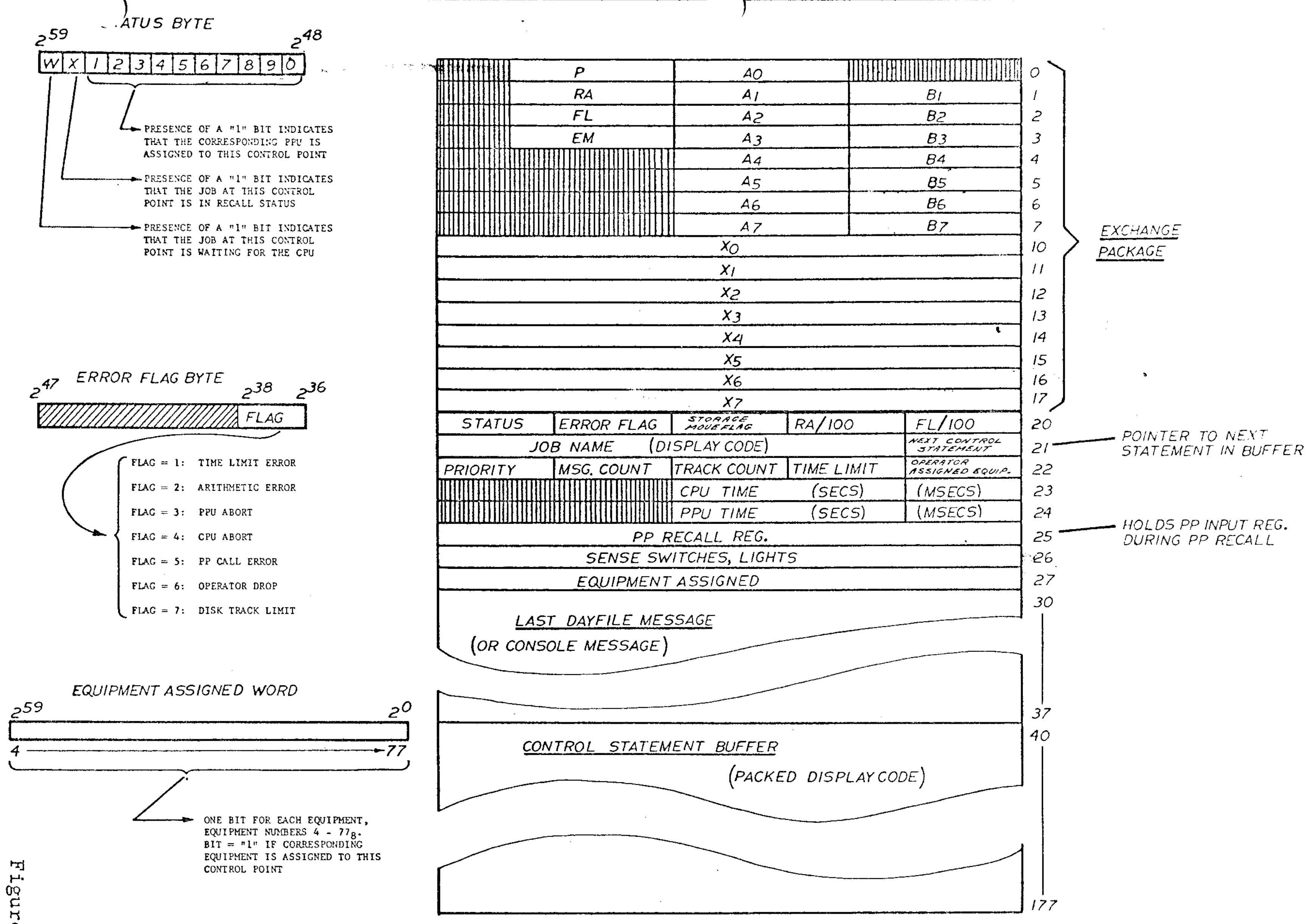

STATUS BYTE
2^{59} W X 1 2 3 4 5 6 7 8 9 0 2^{48}
PRESENCE OF A "1" BIT INDICATES THAT THE CORRESPONDING PPU IS ASSIGNED TO THIS CONTROL POINT
PRESENCE OF A "1" BIT INDICATES THAT THE JOB AT THIS CONTROL POINT IS IN RECALL STATUS
PRESENCE OF A "1" BIT INDICATES THAT THE JOB AT THIS CONTROL POINT IS WAITING FOR THE CPU
ERROR FLAG BYTE
2^{47} 2^{38} FLAG 2^{36}
FLAG = 1: TIME LIMIT ERROR
FLAG = 2: ARITHMETIC ERROR
FLAG = 3: PPU ABORT
FLAG = 4: CPU ABORT
FLAG = 5: PP CALL ERROR
FLAG = 6: OPERATOR DROP
FLAG = 7: DISK TRACK LIMIT
EQUIPMENT ASSIGNED WORD
2^{59} 2^{0} 4 77
ONE BIT FOR EACH EQUIPMENT, EQUIPMENT NUMBERS 4 - 77_8. BIT = "1" IF CORRESPONDING EQUIPMENT IS ASSIGNED TO THIS CONTROL POINT
P A0 0
RA A1 B1 1
FL A2 B2 2
EM A3 B3 3
A4 B4 4
A5 B5 5
A6 B6 6
A7 B7 7
X0 10
X1 11
X2 12
X3 13
X4 14
X5 15
X6 16
X7 17
EXCHANGE PACKAGE
STATUS ERROR FLAG STORAGE MOVE FLAG RA/100 FL/100 20
JOB NAME (DISPLAY CODE) NEXT CONTROL STATEMENT 21
POINTER TO NEXT STATEMENT IN BUFFER
PRIORITY MSG. COUNT TRACK COUNT TIME LIMIT OPERATOR ASSIGNED EQUIP. 22
CPU TIME (SECS) (MSECS) 23
PPU TIME (SECS) (MSECS) 24
PP RECALL REG. 25
HOLDS PP INPUT REG. DURING PP RECALL
SENSE SWITCHES, LIGHTS 26
EQUIPMENT ASSIGNED 27
LAST DAYFILE MESSAGE (OR CONSOLE MESSAGE) 30
37
CONTROL STATEMENT BUFFER (PACKED DISPLAY CODE) 40
177

Figure 1

CONTROL POINT AREAS

CHIPPEWA OPERATING SYSTEM

CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS TRAINING	MACH. TYPE 6000	PROJECT NO.	DOCUMENT ABSTRACT	REV.	APPROVED	DATE	REV.	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE CONTROL POINT AREA		PROJECT MGR.							
FLOWCHART ☐		PAGE 1 OF 1	PROJECT NAME							
DECISION TABLE ☐	NUMBER	ISSUE DATE 11-1-65	TASK NO.							
OTHER [✓ X]	DRAWN BY MM	DATE 11/65	TASK NAME							

CA 127-3

is always defined by the values of RA and FL in bytes 4 and 5, respectively, of location 20_8 within the control point area. Note that these values are in hundreds (upper 12 bits of an 18-bit address).

The control point number is often maintained in the low-order three bits of a byte. On many occasions, the system derives the control point area address by shifting the control point number left 7 places from its low-order bit positions. For example, a routine might pick up a byte containing the number of control point 2, which would appear as 0002: shifting this left 7 places, we obtain 0400, the beginning address of the control point 2 area.

MTR: DEAD START HOUSEKEEPING, THE CP IDLE PROGRAM, AND CONTROL POINT 0

During the loading of the system tape, the lower portion of the central memory resident is initialized by reading a series of records totalling 5000_8 CM words into central memory beginning at location 0. This initialization process sets the first entry in the FNT/FST with the file name DAYFILE and the file type COMMON.

When the loader releases peripheral processor zero to MTR, MTR obtains the next available track number from the Track Reservation Table for disk 0. This half track number is set in the Beginning Track (byte 2) and Current Track (byte 3) bytes of the FST entry for the dayfile. Byte 1, the Equipment Number, is set to zero as is byte 4, the Current Sector byte. The Buffer Status byte (byte 5) is set to 1, indicating that this file is not reserved.

Once the FNT/FST entry for the dayfile has been completed, MTR issues an exchange jump to the central processor idle program. This idle program executes a jump to relative location 2, which contains a stop instruction, and thus halts the central processor with $P \neq 0$. The function of the idle program is to keep $P \neq 0$ in all cases except in the case of an error exit from a central processor program.

The idle program is a central processor program, and as such must be assigned to a control point. A pseudo control point, called control point zero, is used for this purpose. Referring to the control point area illustration (figure 1), note that relative locations 21_8 and 20_8 contain, respectively, the job name and the job status. The control point area for control point zero is assumed to start at location 0 in central memory: central memory locations 21_8 and 20_8 (absolute) contain the job name and the job status for control point zero. These are the only locations in this portion of the resident which are actually a part of the control point zero area: the exchange jump package for the idle program begins at location 2040. Location 21_8 contains MONITOR as the job name. Byte 1 of location 20_8 contains the job status: the low order bits of this byte are used to indicate the assignment of peripheral processors to a control point. For control point zero, the status byte contains 0003, indicating that processor 0 (MTR) and processor 9 (DSD) are assigned to this control point.

The use of the pseudo control point zero is a mechanism simplifying the manner in which MTR controls the assignment of jobs to the central processor. The reason for using location 0 as the start of the control point area for control point zero is evident when we remember that the address of a control point may be obtained from its control point number by shifting the control point number left seven places.

After initiating the central processor idle program, MTR enters its master loop.

MTR: USE OF LOW CORE LOCATIONS

MTR uses low core locations 26 - 77 to maintain various flags, pointers, and special-purpose buffers: these are illustrated in figure 2. Locations 75 - 77 contain the Input Register, Output Register, and Message Buffer pointers for the peripheral processor zero communication area. A five-byte area

MTR: USE OF LOW CORE

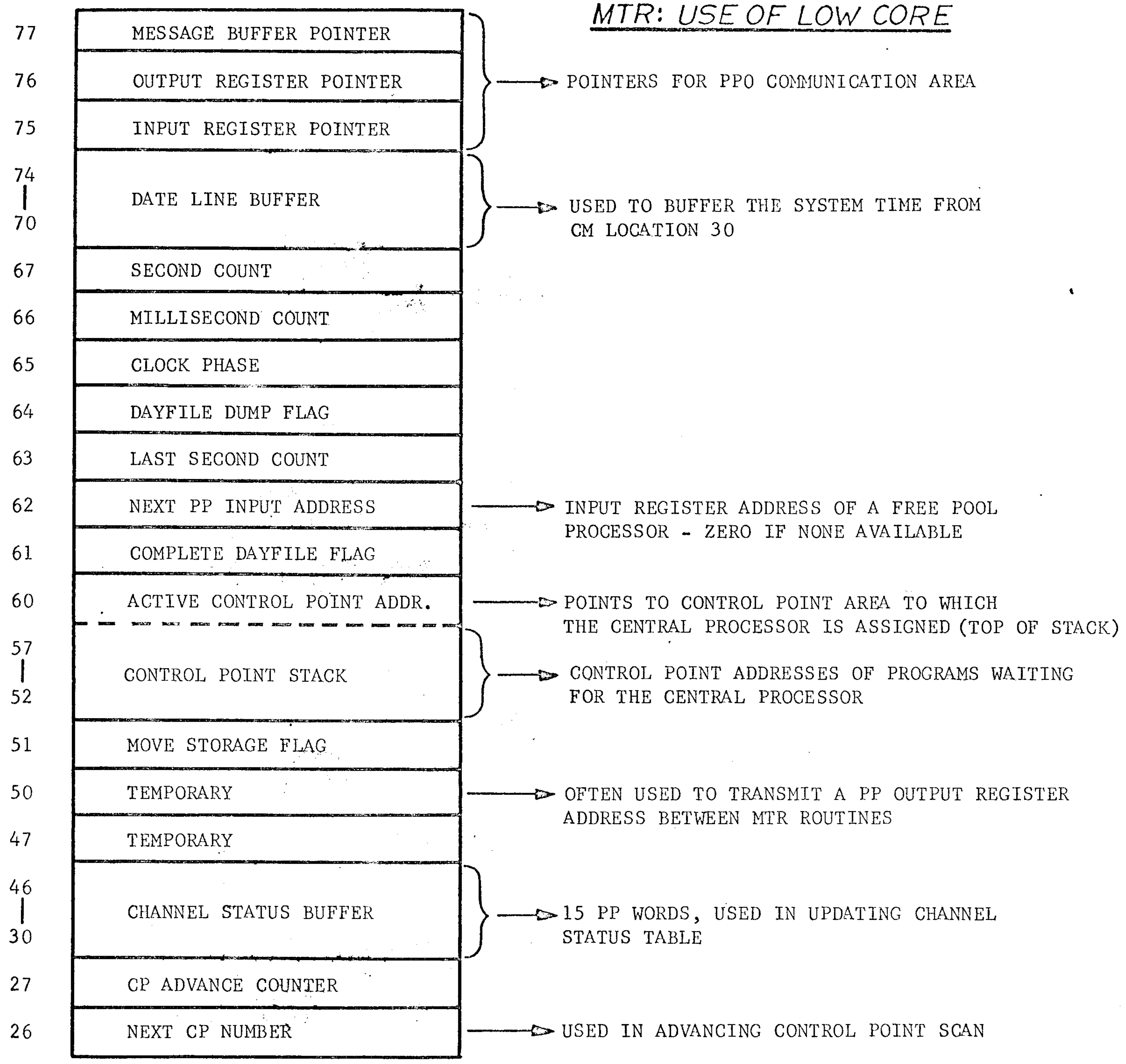

77 MESSAGE BUFFER POINTER
76 OUTPUT REGISTER POINTER
75 INPUT REGISTER POINTER
POINTERS FOR PPO COMMUNICATION AREA
74 – 70 DATE LINE BUFFER
USED TO BUFFER THE SYSTEM TIME FROM CM LOCATION 30
67 SECOND COUNT
66 MILLISECOND COUNT
65 CLOCK PHASE
64 DAYFILE DUMP FLAG
63 LAST SECOND COUNT
62 NEXT PP INPUT ADDRESS
INPUT REGISTER ADDRESS OF A FREE POOL PROCESSOR - ZERO IF NONE AVAILABLE
61 COMPLETE DAYFILE FLAG
60 ACTIVE CONTROL POINT ADDR.
POINTS TO CONTROL POINT AREA TO WHICH THE CENTRAL PROCESSOR IS ASSIGNED (TOP OF STACK)
57 – 52 CONTROL POINT STACK
CONTROL POINT ADDRESSES OF PROGRAMS WAITING FOR THE CENTRAL PROCESSOR
51 MOVE STORAGE FLAG
50 TEMPORARY
OFTEN USED TO TRANSMIT A PP OUTPUT REGISTER ADDRESS BETWEEN MTR ROUTINES
47 TEMPORARY
46 – 30 CHANNEL STATUS BUFFER
15 PP WORDS, USED IN UPDATING CHANNEL STATUS TABLE
27 CP ADVANCE COUNTER
26 NEXT CP NUMBER
USED IN ADVANCING CONTROL POINT SCAN

Figure 2

consisting of locations 70 through 74 is used to buffer the time line from the time-date area in central memory locations 30 - 37. This central memory area contains the time and, optionally, the date: it is initialized via the DSD keyboard entry "TIME". The first word of this central resident area contains the time: this word is read into locations 70 - 74 whenever the time is to be advanced or entered in a dayfile message. Locations 63, 65, 66, and 67 contain counts used in advancing the clock and in computing time charges to a control point.

Location 64 contains the Dayfile Dump Flag, which, when set, indicates that the data in the dayfile buffer is being dumped to the disk, and also indicates which phase of the dumping process is to be executed next. Location 61 contains another dayfile related flag, the Complete Dayfile Flag, which is used in insuring that dayfile messages for a specific job are dumped to the disk at the end of a job.

Location 62 contains the address of the Input Register of a free pool processor: this processor will be assigned by MTR to the next peripheral processor task. If all pool processors are busy, this location contains zero.

Locations 60 and 52 - 57 hold the control point stack. Location 60 represents the top of the stack and contains the address of the control point area for the program currently being executed by the central processor: if this location contains zero, the central processor is unassigned (i.e., is assigned to pseudo control point zero, the control point for the idle program). Control points representing programs waiting for the central processor are stacked in locations 52 - 57.

Location 51 contains a Move Storage Flag, used when storage is being reallocated to control points. Location 50 is a temporary storage area: it is often used to transmit the Output Register address of a peripheral processor between MTR routines.

Locations 30 - 46 provide a buffer used by MTR in updating the Channel Status Table. Locations 26 and 27 are used by MTR in advancing the control point scan: location 27 contains a count used in determining the time interval between successive scans, and location 26 contains the number of the control point to be processed on the next scan.

The remaining low core locations, 01 - 25, are used for a variety of temporary storage needs. For example, locations 10 - 14 are used at various times to hold a peripheral processor's Output Register, the status word from a control point area, a TRT pointer, and a variety of other quantities.

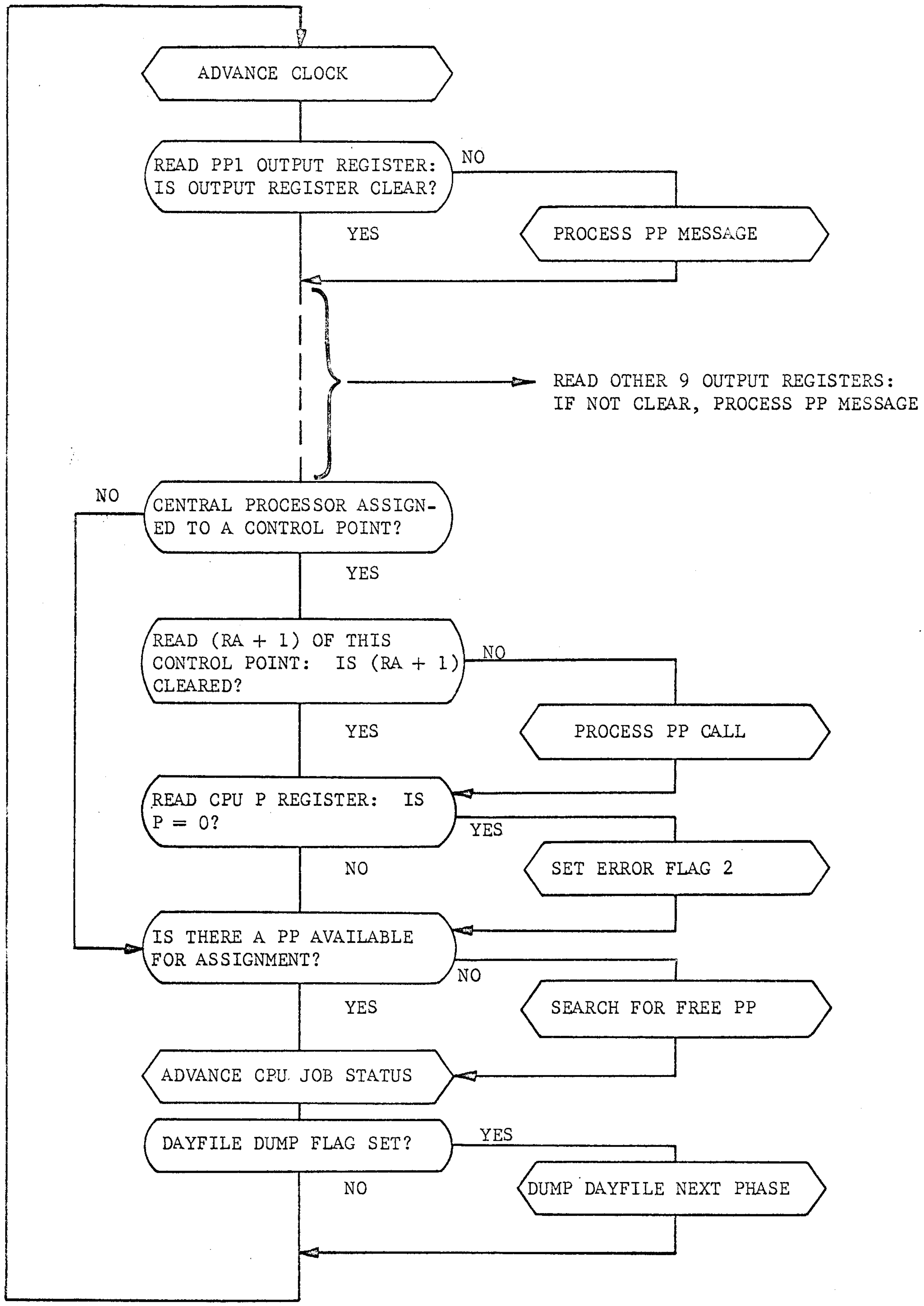
MTR: MASTER LOOP

The MTR Master Loop is illustrated in figure 3. This loop, from which all MTR routines are entered (either directly or indirectly), performs the following four major functions:

- Advances the system clock
- Monitors peripheral processors for function requests
- Monitors the central processor program currently being executed for I/O requests and normal or abnormal exit conditions
- Examines one of the seven control points for PP or CP recall status and may initiate another central processor program: if the control point is inactive, the 1AJ routine is called to bring a job from the disk to this control point.

The time between successive scans is primarily a function of the number and type of requests serviced during a scan. In any case, the fourth function mentioned above (Advance CPU Job Status) is performed at intervals of no less than 64 milliseconds.

The Advance Clock routine updates the system clock, which is stored in location 30 in central memory resident. This word generally has the format


ADVANCE CLOCK
READ PP1 OUTPUT REGISTER: IS OUTPUT REGISTER CLEAR?
NO
PROCESS PP MESSAGE
YES
READ OTHER 9 OUTPUT REGISTERS: IF NOT CLEAR, PROCESS PP MESSAGE
NO
CENTRAL PROCESSOR ASSIGNED TO A CONTROL POINT?
YES
READ (RA + 1) OF THIS CONTROL POINT: IS (RA + 1) CLEARED?
NO
YES
PROCESS PP CALL
READ CPU P REGISTER: IS P = 0?
YES
NO
SET ERROR FLAG 2
IS THERE A PP AVAILABLE FOR ASSIGNMENT?
NO
YES
SEARCH FOR FREE PP
ADVANCE CPU JOB STATUS
DAYFILE DUMP FLAG SET?
YES
NO
DUMP DAYFILE NEXT PHASE

MTR MASTER LOOP

Figure 3

"sp HR . MN . SC .", where HR, MN, and SC are each two display code digits representing, respectively, hours, minutes, and seconds.

On each pass through the loop, MTR reads the Output Register of each peripheral processor, including its own. All requests to MTR from peripheral processor programs are transmitted in the form of function codes placed in the requesting processor's Output Register. When MTR finds a request in an Output Register (i.e., Output Register not cleared), it performs a table look-up for the routine corresponding to the function number, and jumps to that routine. If the request can be executed, the routine clears the Output Register before exiting back to the master loop: if the request cannot be executed, the routine exits to the master loop without clearing the Output Register. In the latter case, MTR will pick up the request again on its next trip through the master loop, and attempt to execute the request once again.

The functions performed by MTR for peripheral processor programs are listed below. The flow chart page numbers refer to the attached flow charts: memory addresses refer to the version of MTR dated 10/15/64.

Function Number	Starting Address	Flow Chart Page No.	Function
1	1500	A-3	Process Dayfile Message
2	2000	A-4	Request Channel
3	2040	A-4	Drop Channel
4	2440	A-4	Assign PP Time
5	1560	A-5	Monitor Step Control
6	2200	A-5	Request Disk Track
7	2300	A-5	Drop Disk Track
10	4300	A-6	Request Storage
11	1300	A-7	Complete Dayfile

Function Number	Starting Address	Flow Chart Page No.	Function
12	3730	A-7	Release PP
13	4040	A-7	Abort Control Point
14	3600	A-8	Enter New Time Limit
15	2600	A-8	Request Central Processor
16	3760	A-8	Release Central Processor
17	5200	A-9	Pause for Storage Relocation
20	4640	A-9	Request Peripheral Processor
21	2750	A-9	Recall Central Processor
22	5600	A-10	Request Equipment
23	5240	A-10	Drop Equipment
24	3240	A-10	Request Priority
25	3630	A-11	Request Exit Mode
26	3030	-	Reserved for Future Use
27	3100	A-11	Toggle Simulator
30	2160	A-12	Operator Drop
31	4200	A-12	Ready Tape
32	4240	A-12	Drop Tape
33	6100	A-12	Assign Equipment
34 - 37	3030	-	Reserved for Future Use

After servicing any peripheral processor requests which may have been present, MTR proceeds to determine if any action is required by the central processor. To determine if the central processor is executing a program, MTR looks at the top of the control point stack (location 60 in processor 0's memory). If this location contains zero, the central processor is idle: if the contents of this location are non-zero, then the central processor is currently executing a program. The entries in the stack are control point addresses: thus, location 60 contains the address of the control point area for the program currently being executed by the central processor. MTR adds 20_8 to this address to form the address of the Status word in the control point area (see figure 1), reads the Status word and extracts byte 4, which contains the reference address in hundreds. MTR then reads the contents of RA + 1 to determine if the central processor program has issued a request.

If the contents of RA + 1 are not zero, MTR jumps to a routine to process the request. If RA + 1 contains END or RCL, another central processor program is initiated in place of the current one: if RA + 1 contains ABT or if the request in RA + 1 does not begin with a letter, the appropriate error flag is set in byte 2, location 20_8, of the control point area. If RA + 1 contains a legitimate PP call, MTR places the call and the control point number of the requestor in the Input Register of an available pool processor and assigns the processor to this control point by setting the appropriate bit in byte one of the Status word. After processing the call, (RA + 1) is cleared to inform the central processor that the request has been processed, and control is then returned to the master loop. If the call was END, ABT, or illegal, or if the request could not be processed at this time (no free pool processor), the routine exits to the master loop without clearing RA + 1. The subroutine which processes central processor requests for peripheral programs is entitled "Process PP Call". Its starting address is 2700, and it appears on page A-14 of the attached flow charts.

After processing the central processor program request, MTR reads the central processor P register. If P contains zero, it is assumed that an error exit has occurred, and MTR sets the appropriate error flag in byte 2 of the Status word in location 20_8 of the control point area.

MTR then looks at location 62 in its memory to see if it has a pool processor available for assignment. If this location contains zero (no processor available), MTR scans the Input Registers of processors 1 - 8 and writes the address of the first cleared Input Register in location 62.

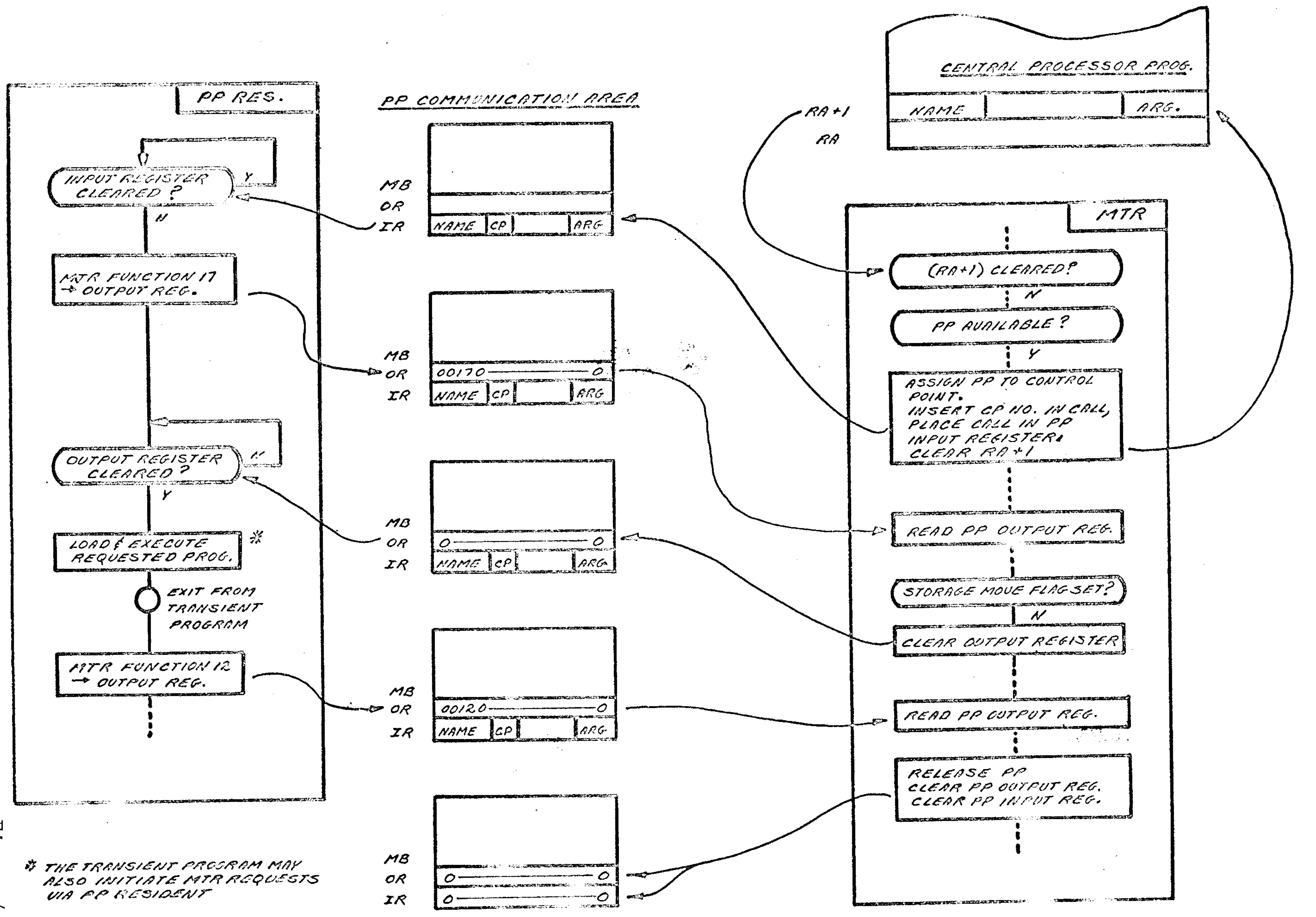
MTR next examines one of the seven control points and determines if the control point is in recall status. If it is, then this program may be re-initiated by MTR, depending upon its priority. If the control point is inactive, MTR directs the loading of another job at this control point. MTR scans only one control point on each pass through the master loop: the number of the control point most recently scanned is maintained in location 26 of MTR's memory. The MTR subroutine which performs this processing is entitled "Advance CPU Job Status" and is shown on page A-13 of the attached flow charts. This subroutine will be discussed at greater length during the description of the control point stack.

If dayfile dumping is not in process, MTR returns to the beginning of the loop and begins its scan once more.

As a review of CPU - MTR - PP communication, the sequence which takes place when a central processor program requests that a task be performed by a peripheral processor program is described below. (Refer to figure 4.)

1. The central processor program requests a peripheral processor by writing the routine name (three display code characters), left-justified, in location 1 of its program. The address of any parameters required are written in the low-order bits of this location.

2. MTR examines the contents of RA + 1 during its master loop: if (RA + 1) is non-zero, MTR jumps to a subroutine to process the call. This subroutine inserts the control point number of the requesting job in the low-order three bits of byte 2 of the word read from RA + 1 and then writes this word in the Input Register of a free pool processor. The routine also sets the bit corresponding to the processor in byte 1 of word 20_8 in the control point area.


PP RES.
INPUT REGISTER CLEARED ?
Y
N
MTR FUNCTION 17 → OUTPUT REG.
OUTPUT REGISTER CLEARED ?
N
Y
LOAD & EXECUTE REQUESTED PROG. *
EXIT FROM TRANSIENT PROGRAM
MTR FUNCTION 12 → OUTPUT REG.
PP COMMUNICATION AREA
MB
OR
IR
NAME
CP
ARG
00170
0
00120
CENTRAL PROCESSOR PROG.
NAME
ARG.
RA+1
RA
MTR
(RA+1) CLEARED?
N
PP AVAILABLE ?
Y
ASSIGN PP TO CONTROL POINT. INSERT CP NO. IN CALL, PLACE CALL IN PP INPUT REGISTER. CLEAR RA+1
READ PP OUTPUT REG.
STORAGE MOVE FLAG SET?
N
CLEAR OUTPUT REGISTER
READ PP OUTPUT REG.
RELEASE PP CLEAR PP OUTPUT REG. CLEAR PP INPUT REG.

* THE TRANSIENT PROGRAM MAY ALSO INITIATE MTR REQUESTS VIA PP RESIDENT

Figure 4

CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS TRAINING	MACH TYPE 6600	PROJECT NO.
SAMPLE CODE ☐	DOCUMENT TITLE CP-MTR-PP COMM.		PROJECT MGR
FLOWCHART ☐		PAGE OF	PROJECT NAME
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.
OTHER ☒	DRAWN BY HH	DATE 9-15	TASK NAME

DOCUMENT ABSTRACT

CP PROGRAM - MTR - PP RES. COMMUNICATION
CHIPPEWA OPERATING SYSTEM

REV	APPROVED	DATE	REV	APPROVED	DATE

3. When the peripheral processor resident finds the routine name in its Input Register, it asks MTR if the storage assigned to this control point is to be relocated by issuing the appropriate function request (function code 17).
4. If the storage assigned to this control point is to be relocated, MTR will delay the execution of the requested transient program by not clearing the Output Register of the peripheral processor until relocation is completed. If no storage relocation is to be done, MTR clears the Output Register immediately upon recognizing the function request.
5. When the Output Register has been cleared by MTR, the resident proceeds to load the requested program either from the resident library or the disk library, and then transfers control to it.
6. The transient program and any overlays it may use may also communicate with MTR by using the resident subroutines to transmit function requests to MTR. These programs may also communicate directly with the central program by adding the parameter address (held in the Input Register) to the value of RA from the control point area in order to obtain the absolute address of information within the central processor program.
7. When the transient program completes execution, it sends (via peripheral resident) a Release PP function request (function code 12) to MTR. MTR clears the processor's Input Register and Output Register, and clears the bit in byte 1 of the Status word (control point area, location 20_8) corresponding to this processor.

MTR: JOB INITIATION

Once the system loader has released control of the peripheral processors to their respective programs, the pool processors begin spinning in their idle loops while MTR and DSD, after performing some initial housekeeping, enter their master loops. To initiate job loading and execution in the system, the operator may use the DSD keyboard entry "AUTO.". This assigns routines to control points as follows:

Control Point	Routine
1	1LJ (READ)
2	1DJ (PRINT)
3	1BJ (NEXT)
4	1BJ (NEXT)
5	1BJ (NEXT)
6	1BJ (NEXT)

DSD accomplishes this assignment by placing the routine name and control point number in the first two bytes of its Message Buffer and (via peripheral resident) issuing a Request PP function (function code 20) to MTR. MTR assigns a processor to the control point by writing the routine name and control point number in the Input Register of a free pool processor, and then setting the appropriate bit in byte one of the control point area status word.

The READ package (1LJ and its overlays) brings jobs in from the card reader and places them on the disk. It enters the job name as the file name in the FNT/FST table, inserts the priority from the job card in the FNT entry, and sets the file type to INPUT.

The NEXT package (1BJ and its overlays) loads a job from the disk to the control point to which it (NEXT) is assigned. 1BJ searches the FNT for the highest priority unassigned file of type INPUT. The file name (job name from the job card) is entered as the job name in the control point area. The file name is changed to INPUT, the file type changed to LOCAL, and the file assigned to this control point. The priority is placed in the control point area by

1BJ via an MTR function request (function code 24). 1BJ then calls overlays to read the first record from the file into the control statement buffer in the control point area. This record, which may be up to 95 words in length, contains the control cards for this job. The Next Control Statement pointer in the control point area is then initialized.

An overlay is called to complete job card translation. The time and the field length from the job card are inserted in the control point area by 1BJ via MTR function requests 10 and 14, respectively. (When 1BJ was initiated, it requested storage from MTR, who set the values of RA and FL in the control point area. Central storage is used by 1BJ for a buffer area: when the job is brought from the disk to the control point, the storage required by the job is requested by 1BJ. This storage is considered by MTR to be a replacement for, and not an addition to, the storage originally assigned to the control point.) In processing the storage request, MTR may have to relocate storage assigned to other (higher) control points. After this relocation is performed, MTR sets the value of FL in the control point area, both in byte 4 of location 20 and in the exchange jump package.

When 1BJ has completed its function, it requests MTR to release the peripheral processor. MTR then clears the processor's Input and Output Registers, and clears the appropriate bit in byte 1 of the status word in the control point area. This byte then contains zero. The processing performed by 1BJ has resulted in the control point area being set as follows:

- the job name, time limit, priority, and field length have been set.
- the pointer to the next control statement has been set to address whatever control card followed the job card.
- byte 1 of the Status word (location 20_8 of the control point area is zero.

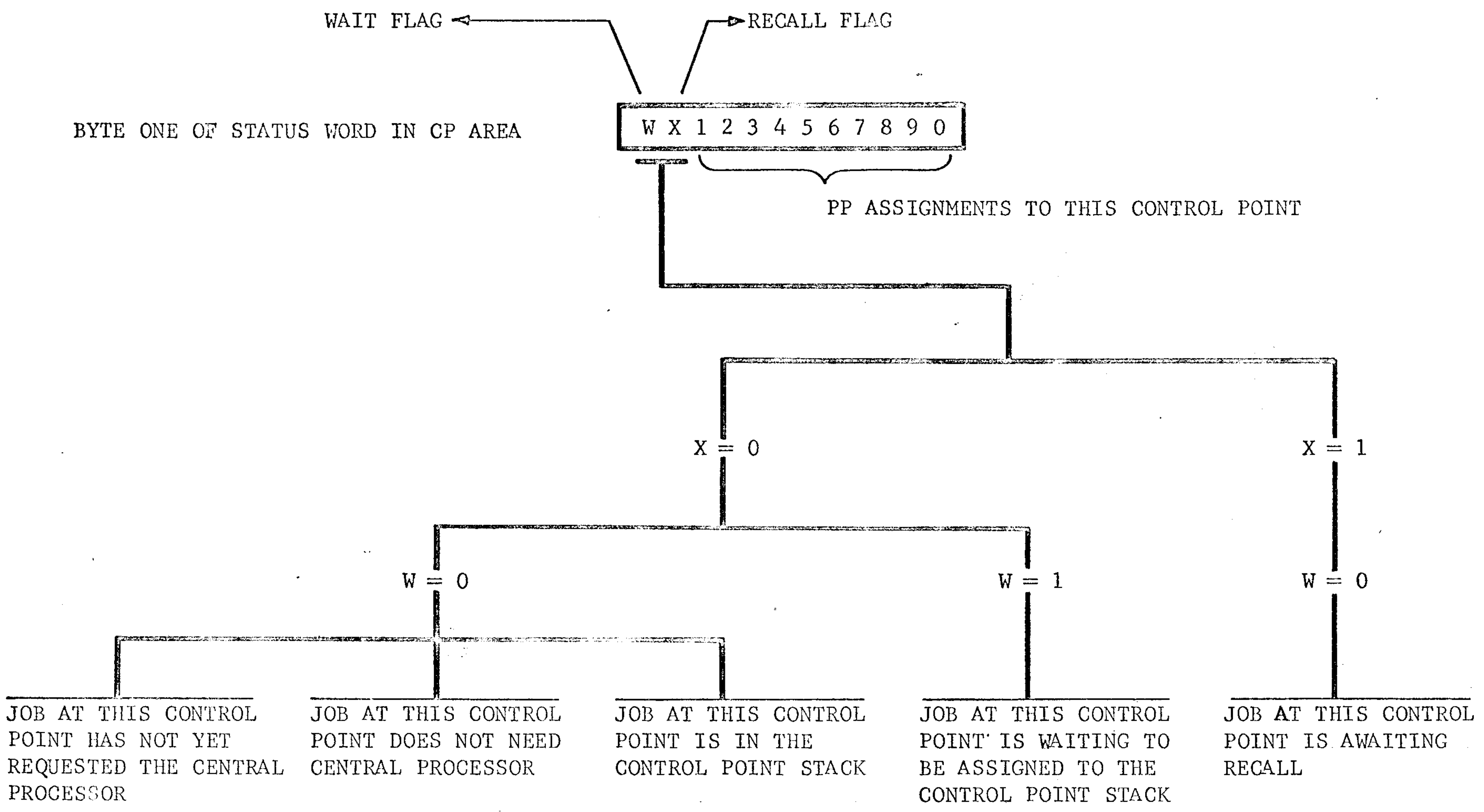
(Note; the control point areas are cleared during loading of the system and are also cleared each time a job is dropped from a control point.)

Once 1BJ has brought a job to a control point, further action concerning that job is initiated by MTR. This action will be described shortly: first we shall discuss the status of a job relative to the central processor.

MTR: JOB STATUS AND THE CONTROL POINT STACK

The status of a job is defined by the setting of bits 2^{11} and 2^{10} in byte one of the control point status word, and by the presence or absence in the control point stack of the control point address for the job. The 2^{10} bit is the X, or recall, flag. This flag is set when MTR detects RCL in RA + 1 of the program being executed by the central processor. The 2^{11} bit is the W, or wait, flag. This flag is set by various MTR routines to indicate that the job at the associated control point is waiting for the central processor. We may have two queues of jobs waiting for the central processor: one queue consists of jobs in the control point stack, and the other consists of jobs in W status. The top of the control point stack (location 60 in PP0 memory) represents the job currently being executed by the central processor. The remaining entries in the stack represent jobs interrupted because of the entry of a higher priority job into the system.

Whenever MTR sets the W flag for a job at a control point, a subroutine called Search for CP Priority is called. The flow chart for this subroutine appears on page A-16 of the attached flow charts. This subroutine checks the status of control points beginning with control point one. If the W flag at a control point is set, MTR compares the priority of this job with the priority of the job currently being executed by the central processor. If the job at the control point with the W flag set (i.e., the first control point found in wait status) has a higher priority than the job currently being executed, the routine pushes down the stack and inserts the control point address of the new job at the top of the stack, clears the W flag in the control point area, and


WAIT FLAG
RECALL FLAG
BYTE ONE OF STATUS WORD IN CP AREA
W X 1 2 3 4 5 6 7 8 9 0
PP ASSIGNMENTS TO THIS CONTROL POINT
X = 0
X = 1
W = 0
W = 1
W = 0
JOB AT THIS CONTROL POINT HAS NOT YET REQUESTED THE CENTRAL PROCESSOR
JOB AT THIS CONTROL POINT DOES NOT NEED CENTRAL PROCESSOR
JOB AT THIS CONTROL POINT IS IN THE CONTROL POINT STACK
JOB AT THIS CONTROL POINT IS WAITING TO BE ASSIGNED TO THE CONTROL POINT STACK
JOB AT THIS CONTROL POINT IS AWAITING RECALL

CP STATUS FLAGS

Figure 5

issues an exchange jump to interrupt the current program and initiate the new job. If the priority of the job currently being executed is higher than the priorities of any job which is in wait status (W flag set), then the routine leaves the flag set. This priority search is repeated on an aperiodic basis.

When a central processor program issues a recall request (by placing RCL in RA + 1), MTR processes this by interrupting this program and initiating the next program in the stack, and then setting the X flag in the control point area of the interrupted program. At an interval of time after the X flag is set, MTR will switch the control point from X status to W status and call the Search for CP Priority routine to re-initiate the job.

If the X flag is set, then, the job at the associated control point is awaiting recall. If the W flag is set, the job is waiting to enter the stack. The stack is always entered at the top: a job always enters the stack by taking control of the central processor. Note that the W flag and the X flag are never both set at the same time. If the W and X flags are both cleared, there are three possibilities:

- the job at the associated control point is in the stack
- the job at the associated control point does not require the central processor
- the job at the associated control point is inactive or has not yet requested the central processor

The interpretation of the X and W flag settings is charted in figure 5.

The status of each control point is examined by a MTR subroutine called Advance CPU Job Status. This routine is called each time MTR makes a pass through its master loop: however, unless 64 milliseconds or longer has passed since the routine was last entered, control is immediately returned to the master loop. This routine examines only one control point on each entry:

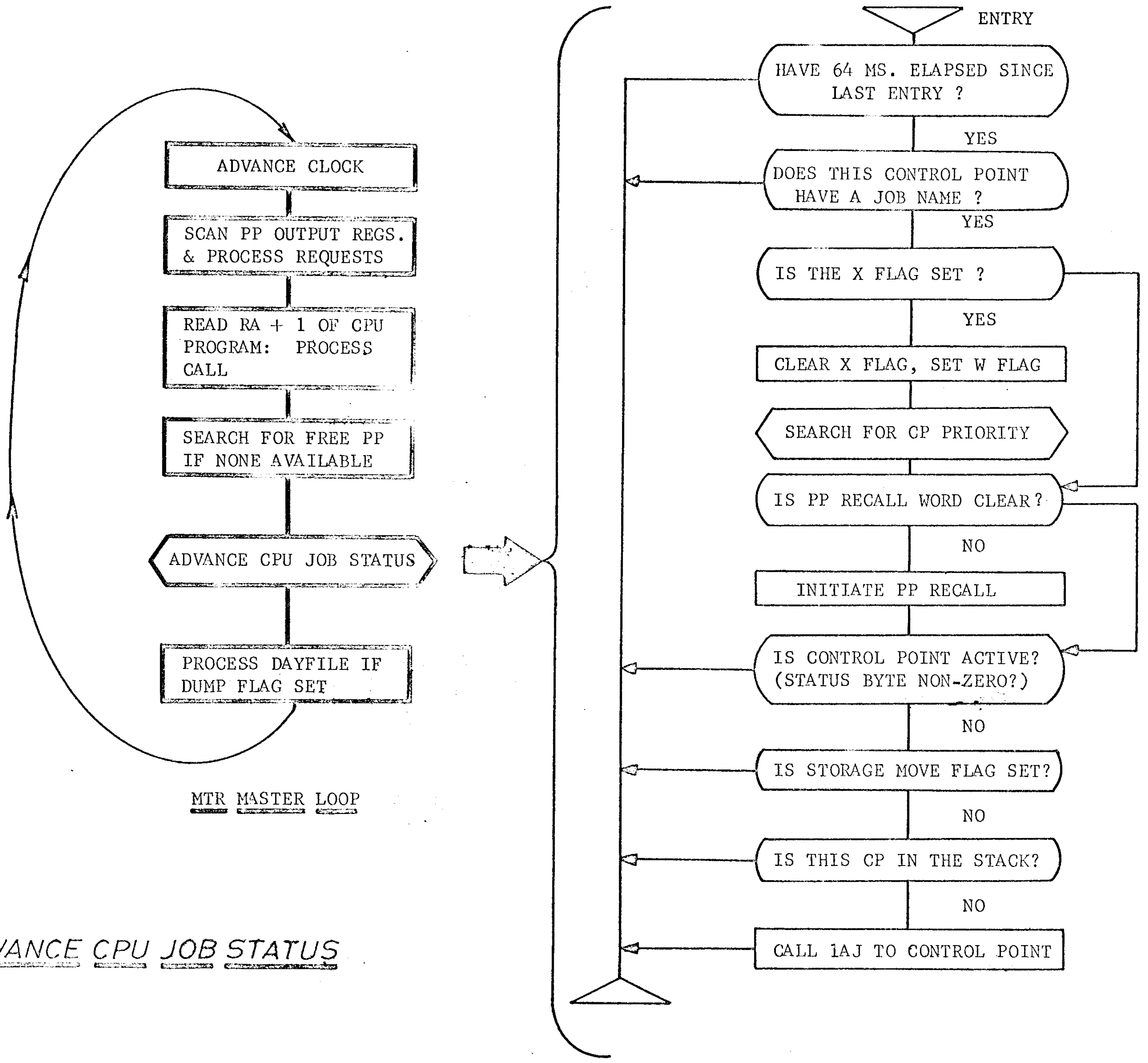
thus, a minimum interval of 7 x 64 = 448 milliseconds elapses between successive scans of the same control point. The Advance CPU Job Status subroutine and its relationship to the MTR master loop are illustrated in figure 6. (See page A-13 of the attached flow charts for a more detailed flow chart of this routine.)

Upon entering the routine (if the 64 ms. interval has elapsed) the pointer for the control point to be scanned is advanced. This pointer is maintained in location 26 of peripheral processor zero's memory. The X flag for the control point is then examined: if this flag is set, the W flag is set and the X flag cleared. The subroutine Search for CP Priority is then called to re-initiate the program. If this program's priority is higher than the priority of the program currently being executed, the running program will be interrupted and its control point address pushed down in the stack: the higher priority program will be initiated, its control point address placed at the top of the stack, and its W flag cleared.

After processing the central processor recall flag, the Advance CPU Job Status routine examines the PP recall word in location 25_8 of the control point area. If this word is non-zero, a peripheral processor is assigned to complete execution of the recalled task.

The routine next examines byte one of location 20_8 in the control point area: if this byte is non-zero, then the W flag or the X flag at this control point is set, and/or a peripheral processor is performing a task for the job at this control point. If this byte is non-zero, then, the job at this control point is active (although perhaps not in execution at the moment): the routine therefore exits back to the master loop.

If the storage move flag in byte three of location 20_8 in the control point area is set, the storage assigned to this control point is to be or is being relocated. The routine exits back to the master loop, thus delaying further action until this relocation is completed.


ADVANCE CLOCK
SCAN PP OUTPUT REGS. & PROCESS REQUESTS
READ RA + 1 OF CPU PROGRAM: PROCESS CALL
SEARCH FOR FREE PP IF NONE AVAILABLE
ADVANCE CPU JOB STATUS
PROCESS DAYFILE IF DUMP FLAG SET
MTR MASTER LOOP
ENTRY
HAVE 64 MS. ELAPSED SINCE LAST ENTRY ?
YES
DOES THIS CONTROL POINT HAVE A JOB NAME ?
YES
IS THE X FLAG SET ?
YES
CLEAR X FLAG, SET W FLAG
SEARCH FOR CP PRIORITY
IS PP RECALL WORD CLEAR?
NO
INITIATE PP RECALL
IS CONTROL POINT ACTIVE? (STATUS BYTE NON-ZERO?)
NO
IS STORAGE MOVE FLAG SET?
NO
IS THIS CP IN THE STACK?
NO
CALL 1AJ TO CONTROL POINT

ADVANCE CPU JOB STATUS

Figure 6

It is possible for byte one of the status word (control point area, location 20_8) to be zero and for the program at that control point nevertheless to be active. For example, the non-executing programs in the stack will have zero status (i.e., byte one of the status word = 0), and the running program will also have zero status if it is not using a peripheral processor. Therefore, in addition to determining that the program has zero status, the routine must also determine if the job is in the control point stack before it can be ascertained that all activity associated with the control point has stopped. If it is found that this is the case, the routine assigns a peripheral processor to the control point and calls the 1AJ routine to that processor. The 1AJ routine promptly calls the statement translator, 2TS, to interpret the next control statement. (Note: although not shown in the simplified flow chart of figure 6, the Advance CPU Job Status routine also checks to see if the running program has exceeded its time limit; if so, the appropriate error flag is set.)

Now let us return for a moment to an earlier point in our discussion. After 1BJ has brought a job to its control point, the job name, time limit, priority, and field length have been set in the control point area. Also, byte one of the status word is zero. Further action involving the control point is initiated by the MTR subroutine Advance CPU Job Status. When this routine scans the control point to which the job was brought by 1BJ, it finds that the job has zero status (byte one of location $20_8 = 0$) and that the job's control point address is not listed in the stack. Thus, there is no activity at this control point. The Advance CPU Job Status routine therefore assigns a pool processor to this control point and calls 1AJ to that processor (by setting the name in the processor's Input Register). 1AJ in turn calls an overlay, 2TS, to process the next control statement from the control statement buffer. If this is a statement such as ASSIGN, RELEASE, COMMON, etc., the statement translator, 2TS, processes the control statement, moves the next control statement pointer (byte 5 of location 21_8 in the control point area) to point to the next control statement in the buffer, and releases the processor. This ends the activity at the control point temporarily: the Advance CPU Job Status routine will recognize this inactivity, and call 1AJ to advance the job at this control point again. This process repeats until a statement not recognized as a control point statement - a program card - is processed by 2TS. When 2TS processes a program card, it searches the FNT, the CLD, and the PLD, in that order, for the program. If the program is found in the FNT, 2TS proceeds to read the program from disk 0 into central memory beginning at location RA. Upon reaching the end of record (or upon detecting the end of the storage assigned to the job), 2TS sets the proper value of P in the exchange area, sets the field length in A_0, transfers the arguments from the program card to the program area beginning at RA + 2, and clears RA and RA + 1. The value of P is obtained by adding 3 to the number of arguments: the latter quantity is supplied by the low-order six bits of the second word in the program record. The field length is set in A_0 so that the program can determine the upper limit of its memory area. The remainder of the exchange area is cleared.

When 2TS has completed setting up the control point area and the program area, it requests the central processor for the job by sending function code 15 (Request Central Processor) to MTR. MTR sets the W flag in the control point area, and calls the Search for CP Priority subroutine to initiate the job. This subroutine will compare the priority of the new job with the priority of the running job, initiating the execution of the new job if it is higher in priority.

MTR: EXCHANGE PACKAGE SWITCHING

As jobs are brought into the system or are recalled from X status, the running program may be interrupted to permit a higher priority program to take the central processor: when this occurs, an exchange jump is issued which results in the exchange package of the interrupted job being stored in the control point area of the newly initiated job. Also, the stack is pushed down and the control point address of the newly initiated job is placed at the top of the stack. Each control point in the stack contains the exchange package for the control point immediately below it in the stack. An example will help to illustrate how this comes about. Assume that 1BJ routines at control points 3, 4, and 5 bring jobs C, B, and A, respectively, to their control points. Job C has a priority of 1, job B has a priority of 2, and job A has a priority of 3. At the time the jobs are loaded, the central processor is executing the idle program. Figure 7 illustrates a possible sequence of events involving these jobs:

1. 1BJ has brought jobs C, B, and A to control points 3, 4, and 5. The central processor is executing the idle program and thus the top of the stack contains zero - the address of the control point area for pseudo control point zero.
2. The Advance CPU Job Status routine has recognized the presence of the job at control point 3 and called 1AJ to advance the job. 1AJ's overlay, 2TS, has loaded the program into memory and, via a MTR request, requested that the job be executed. The MTR subroutine which processed this request called the Search for CP Priority routine. Since job C has a higher priority than the running program, the latter routine issued an exchange jump to job C which resulted in the idle program's exchange package being stored in control point three's exchange area, and then, after pushing down the stack, placed the address of control point three at the top of the stack.
3. When the process described above was performed for control point 4, the Search for CP Priority routine recognized that job B had a higher priority than job C. It therefore issued an exchange jump to start job B, thus storing C's exchange package

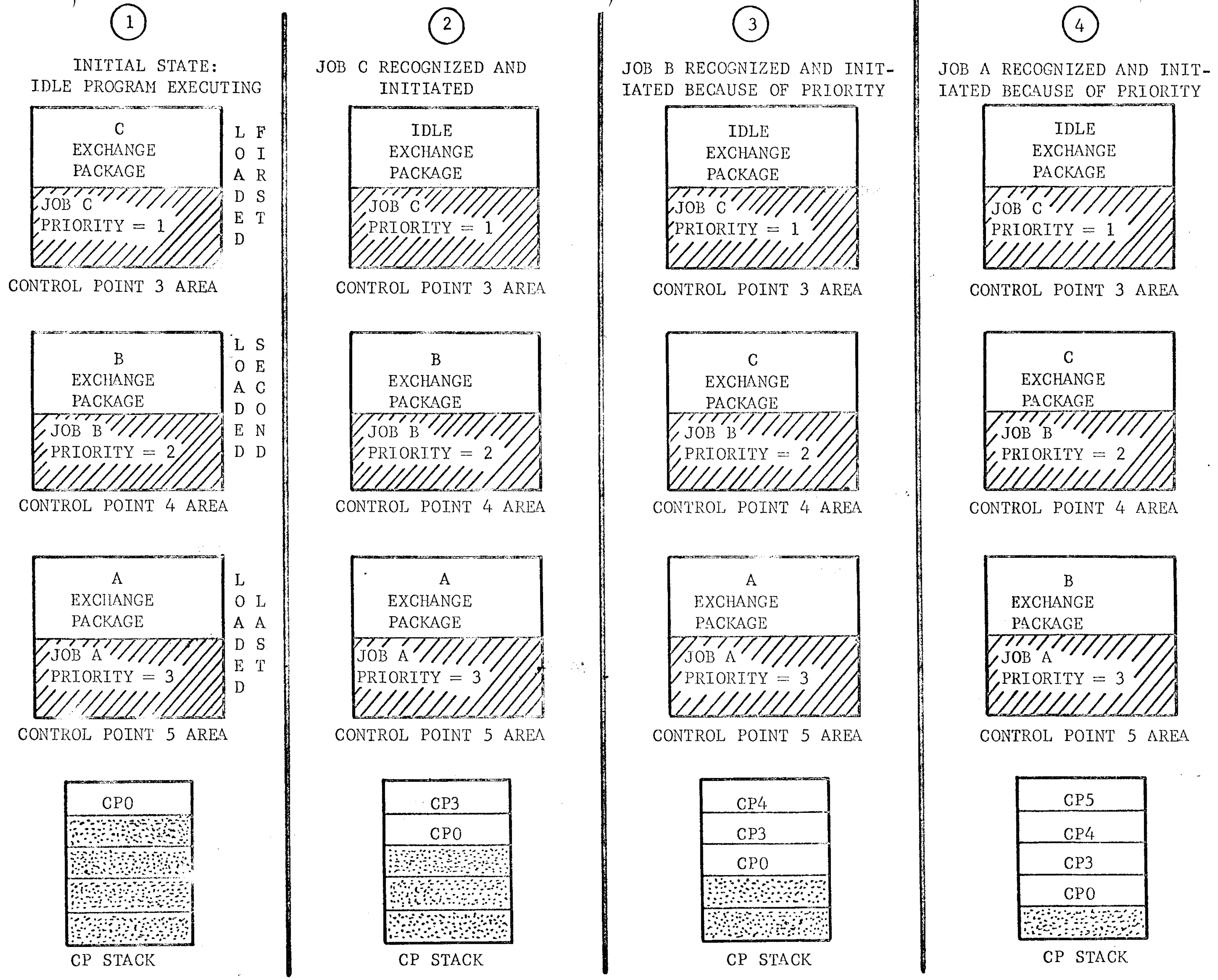

1
INITIAL STATE: IDLE PROGRAM EXECUTING
C EXCHANGE PACKAGE
JOB C PRIORITY = 1
CONTROL POINT 3 AREA
LOADED FIRST
B EXCHANGE PACKAGE
JOB B PRIORITY = 2
CONTROL POINT 4 AREA
LOADED SECOND
A EXCHANGE PACKAGE
JOB A PRIORITY = 3
CONTROL POINT 5 AREA
LOADED LAST
CP0
CP STACK
2
JOB C RECOGNIZED AND INITIATED
IDLE EXCHANGE PACKAGE
JOB C PRIORITY = 1
CONTROL POINT 3 AREA
B EXCHANGE PACKAGE
JOB B PRIORITY = 2
CONTROL POINT 4 AREA
A EXCHANGE PACKAGE
JOB A PRIORITY = 3
CONTROL POINT 5 AREA
CP3
CP0
CP STACK
3
JOB B RECOGNIZED AND INITIATED BECAUSE OF PRIORITY
IDLE EXCHANGE PACKAGE
JOB C PRIORITY = 1
CONTROL POINT 3 AREA
C EXCHANGE PACKAGE
JOB B PRIORITY = 2
CONTROL POINT 4 AREA
A EXCHANGE PACKAGE
JOB A PRIORITY = 3
CONTROL POINT 5 AREA
CP4
CP3
CP0
CP STACK
4
JOB A RECOGNIZED AND INITIATED BECAUSE OF PRIORITY
IDLE EXCHANGE PACKAGE
JOB C PRIORITY = 1
CONTROL POINT 3 AREA
C EXCHANGE PACKAGE
JOB B PRIORITY = 2
CONTROL POINT 4 AREA
B EXCHANGE PACKAGE
JOB A PRIORITY = 3
CONTROL POINT 5 AREA
CP5
CP4
CP3
CP0
CP STACK

Figure 7

in B's exchange area in control point four. The stack was pushed down and the address of control point 4 placed at the top of the stack.

(4) The process is repeated again for control point 5: job A takes over the central processor, and the address of control point 5 is placed at the top of the stack after the stack has been pushed down.

Should job A complete execution or enter recall status, an exchange jump is issued in which the address specified for the exchange package is the address at the top of the stack. The stack is then pushed up. This would cause the exchange package for job A to be stored in control point five's exchange area: the stack and control point areas would then appear as shown in (3).

The use of the central processor by a job may be suspended by means of a "DCP" keyboard entry to the DIS package assigned to the job's control point. When DIS encounters this entry, it transmits the control point number and function code 16 (Release Central Processor) to MTR. The MTR subroutine which processes this request determines whether or not the control point is in the stack. If the control point is not in the stack, the W and X flag bits are set to zero, and the routine exits. Although the job's control point is not in the stack and neither the W flag nor the X flag is now set, the Advance CPU Job Status routine will not consider this job inactive, since byte one of the status word is non-zero by virtue of the fact that a bit is set corresponding to the number of the peripheral processor containing the DIS package.

If the MTR subroutine which processes this request finds that the control point address of the job to be suspended is contained in the stack, it must push the control point address of the job up out of the stack and reorder the exchange packages so that the control point area of the suspended job contains its own exchange package. To accomplish this, MTR exchanges the running program with the program immediately below it in the stack, and pushes up the stack. If the program exchanged (i.e., the former running program) was not that of the job to be suspended, MTR sets the W flag for this job and repeats the above process. When the job to be suspended has been exchanged, MTR calls the Search for CP Priority routine to reconstruct the stack. The W flag for the suspended job is not set.

Many of the DIS entries which modify program parameters utilize this function to halt the running program so that parameters can be changed.

To re-initiate execution of the suspended job, the DIS keyboard entry "RCP." is used. On detecting this entry, DIS sends function code 15 (Request Central Processor) to MTR. MTR then sets the W flag for this job and calls the Search for CP Priority routine to re-initiate execution.

Several MTR subroutines are required to push up the stack to extract a control point address: these routines call the Search for CP Priority routine to reconstruct the stack. The latter routine is the only routine which pushes down the stack and adds new entries to it.

Whenever a routine pushes the stack up or down, a copy of the stack is written in locations 56 and 57 of central memory resident for use by DSD. DSD uses an alphabetic code to indicate the position of a control point in the stack. The control point at the top of the stack (i.e., the running program) is displayed as having program status "A", the next control point in the stack is displayed as having program status "B", and so forth. The W and X flags are also displayed for control points not in the stack.

MTR: PP RECALL PROCESSING

When certain transient programs find they cannot immediately continue to perform their functions, they enter a process called PP Recall. Some of the instances where this takes place are:

- 1BJ - while waiting for storage to be assigned
- 1DJ - while waiting for an output file
- 1LJ - while waiting for a card reader to become ready

To enter PP Recall, a routine simply copies the contents of its Input Register in the PP Recall register for the control point (location 25_8 of the control point area), requests MTR to release the processor, and exits to the resident idle loop.

The PP Recall register in the control point area is examined by the Advance CPU Job Status routine. When this routine finds that the contents of the PP Recall register are non-zero, it recalls the task by copying the contents of the PP Recall register into the Input Register of an available pool processor, clearing the PP Recall register, and assigning the processor to the control point (by setting the appropriate bit in byte one of the status word). The design of the transient programs is such that no internal modifications or special flags are required for recall: a recall entry is treated just like an initial entry.

The recall process is also utilized in the loading of peripheral processor programs. When the statement translator, 2TS, processes a program card, it first assumes that the program requested is a central processor program, and searches the FNT and the CLD for the program. If the program is not found in either the FNT or the CLD, the statement translator then assumes that the request is for a peripheral processor program, and so searches the PLD. If the routine is found in the PLD, the statement translator places the routine name and control point number in the PP Recall register for the control point.

MTR's Advance CPU Job Status routine treats this as if it were a recall entry. It assigns a processor to the control point and copies the PP Recall register into the processor's Input Register. The processor's resident program then proceeds to load the program from the disk library and execute it.

MTR: NORMAL AND ABNORMAL JOB TERMINATION

In normal termination of a central processor job, the central processor program initiates this termination by writing "END" in RA + 1. When MTR detects this request during its master loop, it exchanges this program with the program at the control point below it in the stack, and pushes up the stack. If all peripheral processor activity associated with this control point has ceased, then byte one of the status word in the control point area will be zero. When the Advance CPU Job Status routine detects that this control point is inactive, it will call the 1AJ routine to the control point. If all control statements in the control statement buffer have been processed, 1AJ will wrap up the job.

When a job at a control point involves only peripheral processors and does not use the central processor, normal termination involves a process similiar to that described above. When a peripheral processor program completes execution, it requests MTR to release the processor (function code 12). MTR does this by clearing the processor's Input and Output Registers and clearing the appropriate bit in byte one of the control point's status word. When all processors associated with this control point have been released, the job will have zero status. This will be detected by MTR's Advance CPU Job Status routine, which will call 1AJ to the control point.

Abnormal termination of a job may be initiated by a central processor program, a peripheral processor program, or by MTR. Regardless of who initiates abnormal termination, the general procedure followed by MTR is to set an error flag in byte two of the status word at the control point and cause the job to assume zero status by clearing the W or X flag, releasing peripheral processor assignments, and/or pushing the job's control point out of the stack. When the Advance CPU Job Status routine detects that the job at this control point has zero status, it will call 1AJ to advance the job. 1AJ senses that the error flag is set and calls an overlay to process the remaining control statements in the control statement buffer. This overlay, 2EF, searches the control statement buffer for an EXIT statement: if none is found, control is returned to 1AJ to wrap up the job. If an EXIT statement is found, the control statement immediately following it is picked up for translation and processing.

For certain of the error flag conditions, the 2EF overlay inserts (via a MTR request) an error message in the dayfile. Error messages for other flags are placed in the dayfile by the initiating routine.

The setting and processing of the various error flags is described below.

Error Flag 1: Time Limit. Byte four of location 22 in the control point area contains the central processor time limit for the job. Bytes three and four of location 23 in the control point area contain the central processor running time in seconds for the job. Each time the Advance CPU Job Status routine is entered, the running time of the active central processor program is incremented. The running time (bytes 3 and 4 of location 23 in the control point area) is then compared with the time limit (byte 4 of location 22). If the time limit has been exceeded, a subroutine called Set Error Flag (page A-15 of attached flow charts) is called. This subroutine drops the job from the central processor either by clearing the W/X flag or by exchanging the program with the next one in the stack and pushing up the stack. It then sets the error flag bit in byte two of the control point's status word. In the case of a time limit error, the error message is later inserted by 1AJ's overlay, 2EF.

Error Flag 2: Arithmetic Error. On each pass through its master loop, MTR reads the central processor P register. If (P) is zero, it is assumed that an error exit due to an infinite/indefinite operand or bounds error has occurred. MTR then calls the Set Error Flag subroutine to drop the job from the central processor and set the flag in byte two of the control point's status word. The error message is later inserted by 2EF.

Error Flag 3. PP Abort. There are several instances in which a peripheral processor program finds it necessary to abandon a task. Some of these are:

- a peripheral resident is unable to locate a package in the resident or peripheral libraries
- CIO's overlay, 2BP, finds an error in the buffer parameters specified in a call
- a parity error is encountered when backspacing (after three attempts)

In these instances, the peripheral processor program sends an error message to the dayfile and then requests MTR to abort the control point (function code 13). MTR releases the processor (thus clearing the corresponding bit in byte one of the status word) and then calls the Set Error Flag subroutine to set the flag in the status word and to drop the job from the central processor.

Error Flag 4: CPU Abort. When a central processor program finds it necessary to abort execution, it writes "ABT" in RA + 1. When MTR detects this during its master loop, it calls the Set Error Flag Subroutine to set the flag in byte 2 of the status word and to drop the central processor.

The central processor program may abort because of some computational condition, or because of a problem in the execution of a peripheral processor program associated with the job. For example, if a tape read operation encounters a parity error, after the third unsuccessful read it sets the 2^0 bit in byte four of RA and pauses (function code 17: Pause for Storage Relocation). The central processor program presumably monitors this location: when it detects that this bit is set, it must decide whether to abort execution or to ignore the error. To ignore the error, the central processor program clears this bit: the peripheral processor program will sense when the bit is cleared and proceed with its execution. To abort execution, the central processor program places "ABT" in RA + 1, which results in an error flag being set. The peripheral processor program senses this error flag and, when it finds that the error flag is set, it requests MTR to release the processor. In either case, the peripheral processor program will place a message in the dayfile.

Error Flag 5: PP Call Error. When MTR senses, during its master loop, that the contents of RA + 1 are non-zero, it calls a subroutine to process the request. If the contents of RA + 1 are not END, RCL, or ABT, then it is assumed that a peripheral program is being called. The subroutine checks the first character of the call to see if it is a letter. If it is, the request is issued to a free pool processor. If it is not, the Set Error Flag subroutine is called to set the flag in byte two of the control point's status word and to drop the central processor. The 2EF overlay later inserts an error message in the dayfile.

Error Flag 6: Operator Drop. When DSD detects the keyboard entry "n.DROP." (n = control point number), it transmits the control point number and function code 30 (Operator Drop) to MTR. MTR calls the Set Error Flag subroutine to set the error flag in byte two of the control point's status word and to drop the central processor.

Error Flag 7: Track Limit. The number of half tracks on disk 0 requested by peripheral processor programs assigned to a control point is maintained in byte 3 of location 22 in the control point area. This quantity is incremented as tracks are requested and decremented as tracks are dropped. Each time a track is requested, MTR checks to see if more than 777_8 half tracks have been assigned to this control point: if so, the Set Error Flag subroutine is called to set the flag in byte 2 of the control point's status word and to drop the job from the central processor. The 2EF overlay later inserts the error message in the dayfile.

If a routine requests a half track assignment from MTR, and MTR, in searching the Track Reservation Table, reaches the end of the table before an available half track is found, a zero byte is returned to the requestor in byte one of the first word in the Message Buffer. The requestor then aborts the control point via an MTR request, resulting in the setting of error flag 3.

The error message "TRACK LIMIT" indicates that a control point has requested the assignment of more than 777_8 half tracks on disk 0. The error message "DISK X TRACK LIMIT" indicates that disk X has overflowed.

MTR: STORAGE ALLOCATION AND RELOCATION

The blocks of central memory storage assigned to the various control points always occupy positions in central memory relative to the number of the control point to which they are assigned. Thus, the storage assigned to control point 2 appears immediately above the storage assigned to control point 1, the storage assigned to control point 3 appears immediately above that assigned to control point 2, and so forth. As the jobs at control points request and release storage, the storage assigned to higher control points is relocated up or down so that no gaps of unassigned storage appear between the storage blocks of consecutive control points. All unassigned storage appears at the high end of memory.

Peripheral processor programs request storage from MTR via a Request Storage function (function code 10). Whenever 1BJ brings a job to a control point, it requests MTR to assign the storage specified on the job card. In addition, many peripheral processor programs request storage for their own use, primarily for buffers. Thus, 1BJ requests 300_8 words of storage to use as a buffer in reading the record containing the control statements, 1LJ requests 4000_8 words of storage to use in buffering jobs from the card reader to the disk, and so forth.

The MTR Request Storage subroutine is shown on page A-6 of the attached flow charts. Upon entry, a storage move flag is set in location 51 of peripheral processor zero's memory. This flag is the Output Register address of the requesting processor. The difference between the amount of storage requested and the amount currently assigned to the control point is then computed. For example, when 1BJ requests that storage be assigned for the job to be loaded, MTR computes the difference between the requested storage (from the job card) and the storage already assigned to the control point (the 300_8 locations used as a buffer). If the requested storage represents an increase, MTR ascertains if there is enough unassigned memory available to provide room for the increase. It does this by subtracting (RA + FL) for control point 7 from 400000_8 to determine the amount of unassigned storage. The storage increase requested is then compared with the amount of unassigned storage. If there is insufficient unassigned storage available to meet the request, MTR clears the requesting processor's Output Register, clears the storage move flag in location 50, and exits. The requesting routine senses if the storage requested has been assigned by reading the value of FL in byte 5 of location 20 in its control point area and comparing this value with the amount of storage requested.

If there is room for the storage increase, or if the request represents a decrease, MTR sets a storage move flag in each control point above the requesting control point. This flag is the 2^0 bit in byte 3 of location 20 in the control point area. Thus, if the requesting processor is assigned to control point 4, storage move flags would be set in control points 5, 6, and 7.

After setting the storage move flags, MTR determines if there is any peripheral processor activity at the flagged control points. It does this by first examining byte one of the control point status word. If bits 2 - 9 of this byte are cleared, then there is no peripheral processor activity at this control point. If one of these bits is set, then MTR reads the Output Register of the corresponding processor. If this Output Register contains anything but a 17 function code, MTR exits from the Request Storage routine. Only when all control points whose storage is to be relocated either have no peripheral processor assignments or have paused for storage relocation by issuing a 17 function code does MTR proceed to relocate storage.

To relocate storage, MTR sets up the exchange package for the storage move program with the parameters required to effect the relocation. This exchange package begins at location 2000_8 of the central memory resident. MTR then proceeds to push up the stack, exchanging each program in turn and setting the W flag for the control point, until control point 0 is at the top of the stack. The storage move program is then exchanged for the idle program. When the storage move program completes execution, it stops with P = 0. The Request Storage subroutine monitors P and, when it becomes zero, exchanges the idle program for the storage move program. The RA value in each of the flagged control point areas (both in byte five of location 20 and in the exchange package) is then updated by adding the increase to the original value, and the storage move flags cleared. The FL value is then set in the exchange package and status word of the control point area for the requesting processor. Finally, the Search for CP Priority routine is called to reconstruct the stack, the storage move flag in location 50 is cleared, and the Output Register of the requesting processor is cleared.

Many peripheral processor programs, such as 1BJ, 1DJ, and 1LJ, enter PP Recall if a storage request cannot be immediately satisfied because of lack of space. In this case (insufficient unassigned storage), the Request Storage routine clears the requesting processor's Output Register and the storage move flag in location 50 prior to exiting. Even though sufficient unassigned storage is available, storage relocation can be initiated only when all peripheral processor activity has ceased for the control points whose storage is to be relocated. Should one or more control points have active peripheral processors, the Request Storage routine exits without clearing the requesting processor's Output Register. This does two things; it inhibits the requesting processor's resident from exiting the Process Request subroutine, and it causes MTR to re-enter the Request Storage subroutine on every pass through its master loop, since the request remains in the processor's Output Register. Requests for storage from other processors are ignored while this request is in process. Effectively, then, MTR checks, on every pass through its master loop, the peripheral processor activity at the flagged control points to see if relocation can be initiated.

Cessation of peripheral processor activity may come about because a processor has completed its assigned task and requested MTR to release it (in which case it may immediately be assigned to another task), or because the processor has paused by sending function code 17, Pause for Storage Relocation, to MTR. When MTR detects this function request, it examines the storage move flag in the associated control point area: if set, MTR returns to its master loop without clearing the processor's Output Register, thus effectively stopping the processor. When all processors assigned to the flagged control points have paused, storage relocation can begin.

The peripheral processor residents all pause for storage relocation immediately upon recognizing a request in their Input Registers. In addition, many peripheral processor programs pause for storage relocation when delayed in their execution. For example, tape drivers pause for storage relocation when the tape unit is not ready or when a tape error occurs.

The READ package (1LJ and its overlays) pauses for storage relocation only when the card reader is not ready. Similiarly, the PRINT package (1DJ and its overlays) pauses for storage relocation only when the line printer is not ready. It is therefore important that these packages should be assigned to control points one and two. Should they be assigned to higher control points, they could hold up the allocation of storage for jobs at lower control points for considerable periods of time.

The coding for the storage move program is shown on page B-1, together with the 6600 central processor timing for the loop used in moving storage up. Storage relocation requires approximately 7.2 microseconds for an increase of $10,000_8$ words.

MTR: TIME ACCOUNTING

Location 30 of the central memory resident contains the system time in hours, minutes, and seconds. This location may be initialized to real time via the DSD keyboard entry "TIME". If not initialized, this location reflects the elapsed time since the system was loaded. This time is updated by the Advance Clock subroutine. This routine is shown on page A-2 of the attached flow charts.

In addition to maintaining the system time in location 30 of central memory resident, the subroutine also maintains a current second count and a current millisecond count in locations 67 and 66, respectively, of PPO's memory. Upon entering the Advance Clock subroutine, MTR reads the real time clock on channel 14 and extracts the high-order two bits. These two bits are interpreted as follows:

high-order bits = 00:	real time clock has advanced 0 milliseconds
high-order bits = 01:	real time clock has advanced 1 millisecond
high-order bits = 10:	real time clock has advanced 2 milliseconds
high-order bits = 11:	real time clock has advanced 3 milliseconds
high-order bits = 00:	real time clock has advanced 4 milliseconds
. . . .	. . . .

These two bits are compared with the clock phase in location 65 of PPO's memory. This clock phase is the value of these two bits on the last entry to the subroutine: if these two bits are unequal to the clock phase, then a millisecond has elapsed since the last entry, and so the millisecond count in location 66 is incremented.

As the real time clock runs through a full period, the millisecond count is advanced by 4. Actually, a full period represents 4.096 milliseconds: therefore, rather than waiting for the millisecond count to reach 1000 before advancing the second count, MTR advances the second count when the millisecond count reaches 976. This represents 244 full periods of the real time clock, or an actual elapsed time of 999+ milliseconds.

If the second count was not advanced, the real time clock is read again before exiting to determine if a millisecond advance has taken place while the computations described above were taking place. If no advance has occurred, the subroutine is exited. If the second count was advanced, MTR reads the system time from location 30 of central memory resident, updates it, and writes it back in central memory. The real time clock is then read again to determine if an advance has occurred: if no advance has occurred, the subroutine is exited.

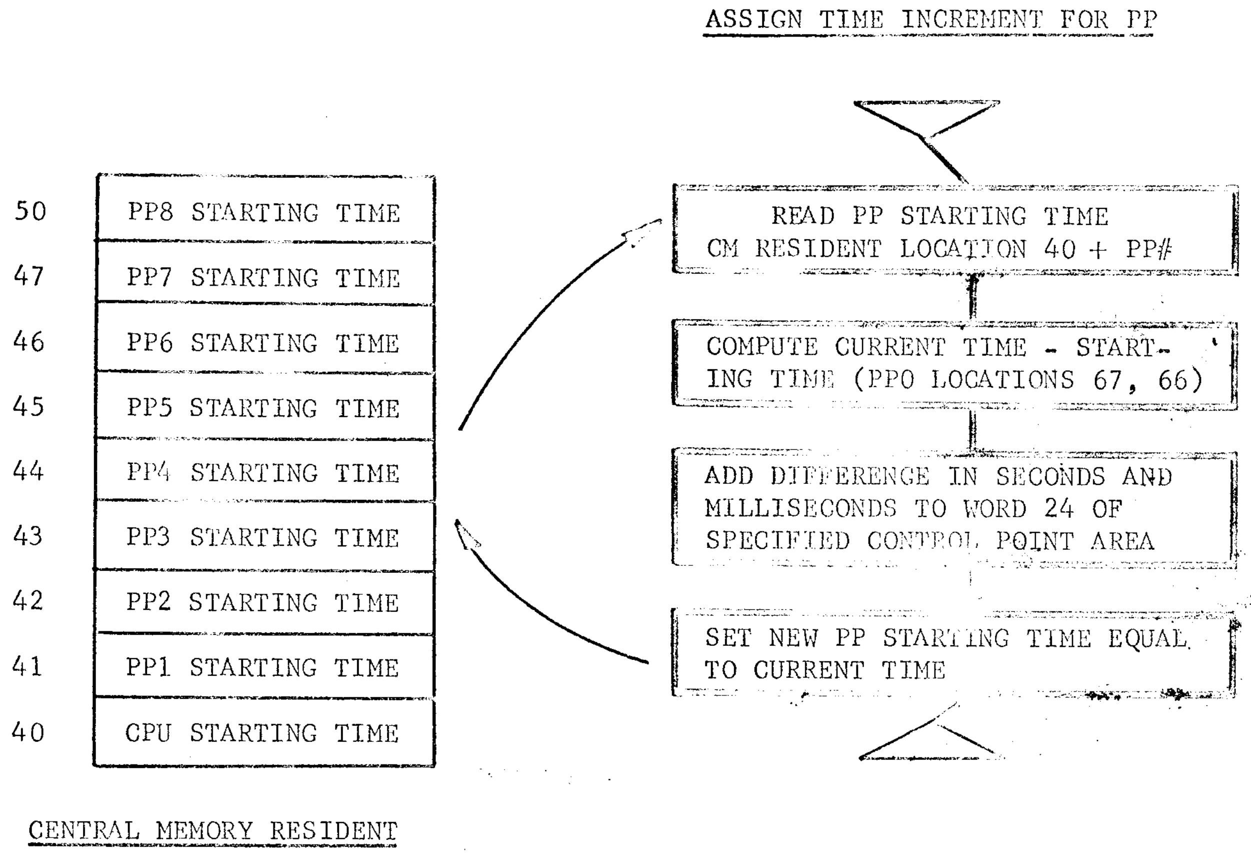
In order for the system time to be properly maintained, MTR must, on the average, enter this subroutine every millisecond. Therefore, entry to this subroutine is made from several points in MTR. The MTR routines which call the Advance Clock subroutine are:

- MTR master loop
- Process PP Message Routine
- Request Storage (function 10)
- Release Central Processor (function 16)
- Request Exit Mode (function 25)
- Set Error Flag Routine

The system time in location 30 of central memory resident is inserted by MTR in each dayfile message and is displayed by DSD. This time is not, however, used in computing time charges to control points.

Location 23_8 in the control point area holds the central processor time charged to the job at the control point, while location 24_8 contains the peripheral processor time charged to the job. These times are maintained in seconds and milliseconds, and are entered in the dayfile by 1AJ upon completion of the job. Peripheral processor time charges are accumulated by the Assign Time Increment for PP subroutine. This subroutine maintains a starting time for each pool processor in central memory locations 41 - 50_8. This starting time represents the time at which the peripheral processor most recently became idle or active and is maintained in seconds and milliseconds. The Assign Time Increment for PP subroutine is illustrated in figure 8. This subroutine is entered with the number of the pool processor and with the control point address. On entry, the starting time for this processor is read from central memory (location 408 + PP number) and subtracted from the current time in seconds and milliseconds maintained in locations 67 and 66 of PP0's memory. The difference is added to the contents of word 24_8 in the specified control point area. The starting time for the processor is then reset to the current time in seconds and milliseconds.

When MTR assigns a pool processor to a task, it enters this subroutine with the number of the processor and with the address of control point zero. The difference between the starting time and the current time is the length of time which the processor has been idle. The new starting time represents the time at which the processor began execution of the task assigned to it by MTR. On completion of the task, MTR agains enters the subroutine - this time with the number of the processor and the address of the control point to which the processor was assigned. The difference between the current time and the starting


ASSIGN TIME INCREMENT FOR PP
READ PP STARTING TIME CM RESIDENT LOCATION 40 + PP#
COMPUTE CURRENT TIME - STARTING TIME (PPO LOCATIONS 67, 66)
ADD DIFFERENCE IN SECONDS AND MILLISECONDS TO WORD 24 OF SPECIFIED CONTROL POINT AREA
SET NEW PP STARTING TIME EQUAL TO CURRENT TIME
50 PP8 STARTING TIME
47 PP7 STARTING TIME
46 PP6 STARTING TIME
45 PP5 STARTING TIME
44 PP4 STARTING TIME
43 PP3 STARTING TIME
42 PP2 STARTING TIME
41 PP1 STARTING TIME
40 CPU STARTING TIME
CENTRAL MEMORY RESIDENT

- ON ASSIGNING A POOL PROCESSOR TO A CONTROL POINT, THE ROUTINE IS ENTERED WITH THE PP NUMBER AND THE ADDRESS OF CONTROL POINT 0
- ON RELEASE OF A POOL PROCESSOR, THE ROUTINE IS ENTERED WITH THE PP NUMBER AND ADDRESS OF THE CONTROL POINT TO WHICH THE PP WAS ASSIGNED

Figure 8

PP TIME ACCOUNTING

time is the length of time the processor was assigned to the control point: the new starting time is the time at which the processor becomes idle. This subroutine accumulates all PP usage times for a job in word 24 of the control point area. All the idle time for the pool processors is accumulated is control point zero's area - location 24 of central memory resident.

The Assign Time Increment for PP subroutine is called by the following MTR routines:

- Assign PP Time to CP (function 4)
- Release PPU (function 12)
- Abort Control Point (function 13)
- Request PPU (function 20)
- Process PP Call
- Advance CPU Job Status

Routines processing jobs not associated with the control point, such as the READ and PRINT packages, must handle their own time charges. When these routines begin processing a file, they send function request 4 to MTR. MTR assigns the idle time to control point zero and sets a new starting time for the processor in which the routine resides. The routines then sets location 24 in the control point area to zero. When processing of the file is completed, these routines again send function request 4 to MTR, and MTR computes the processing time and stores it in location 24 of the control point area. The routines then read this time from the control point area, convert it to decimal, and write it in the dayfile via an MTR request. MTR inserts the job name in the dayfile message: it is to provide this job name that these routines change the job name in the control point area from READ or PRINT to the file name when processing of the file is initiated.

Time charges for the central processor are accumulated in a similiar manner. There is, however, one exception: since central processor programs have a time limit, the central processor time charges to a control point are advanced every second. Location 40 in central memory resident contains the central processor starting time. At intervals of one second or less, this time is read and subtracted from the current time in seconds and milliseconds maintained by MTR in locations 67 and 66 of PP0's memory. The difference is added to the contents of word 23 in the specified control point area, and a new starting time is set in location 40 in central memory. (Note: the control point area is cleared at dead start time and whenever 1AJ drops a job from a control point.) Central processor time charges are updated by the Advance CPU Job Status routine, and whenever the control point stack is pushed up or down (i.e., whenever the running program is exchanged).

The Advance CPU Job Status routine maintains a Last Second count in location 63 of PP0's memory. Each time the Advance CPU Job Status routine is entered, the Last Second count is compared with the current second count in location 67 of PP0's memory. If these two quantities are not equal, the Last Second count is updated, the central processor time charges are accumulated for the job currently using the central processor, and a test is made to determine if the time limit has been exceeded.

<u>MTR: THE DAYFILE</u>

The dayfile is a combination of a time accounting medium and a job log. The contents of the dayfile include:

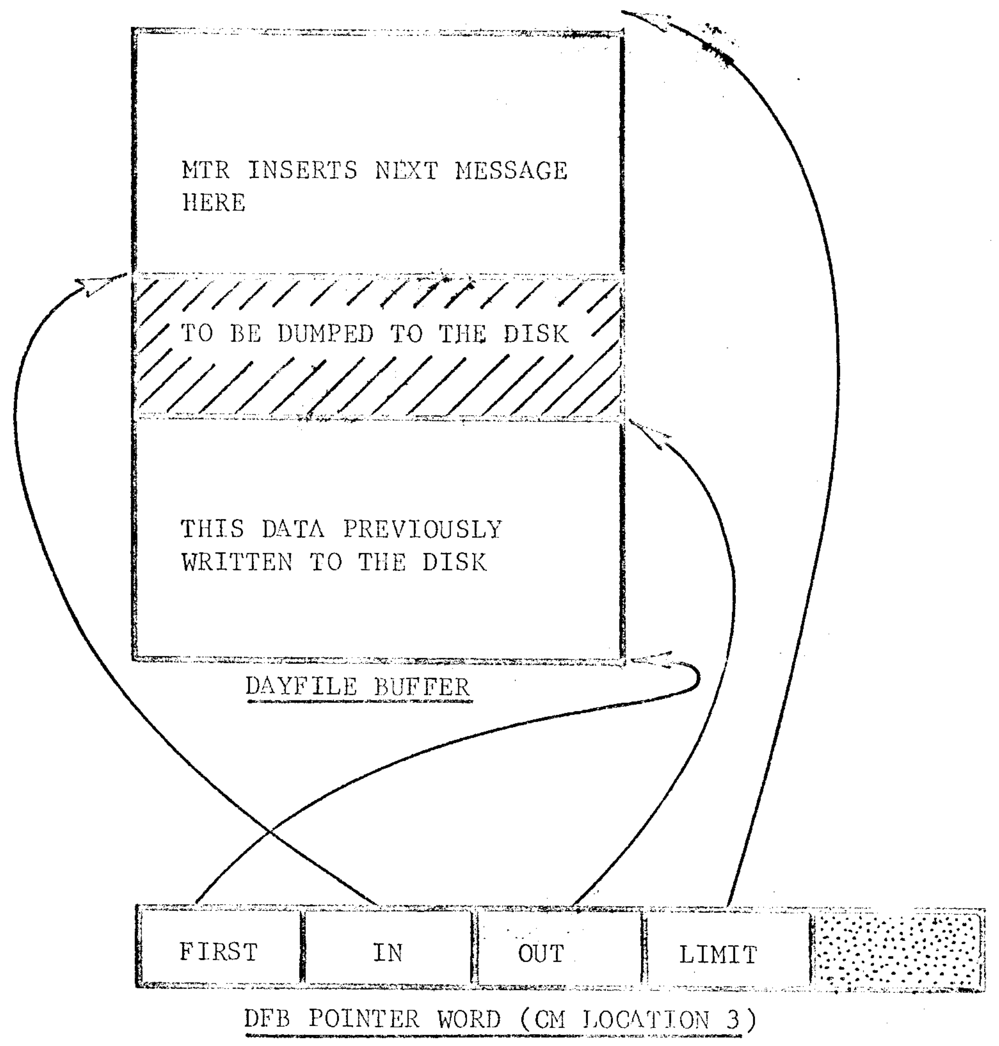
- all control cards
- all diagnostic messages
- job loading times, job execution times (both for the central processor and the peripheral processors), and job printing times
- messages to the operator

The dayfile is maintained as a COMMON file on the disk. In addition, a number of the most recent dayfile entries are displayed on the console by DSD. At the end of a job, all dayfile entries for that job are printed as part of that job's output.

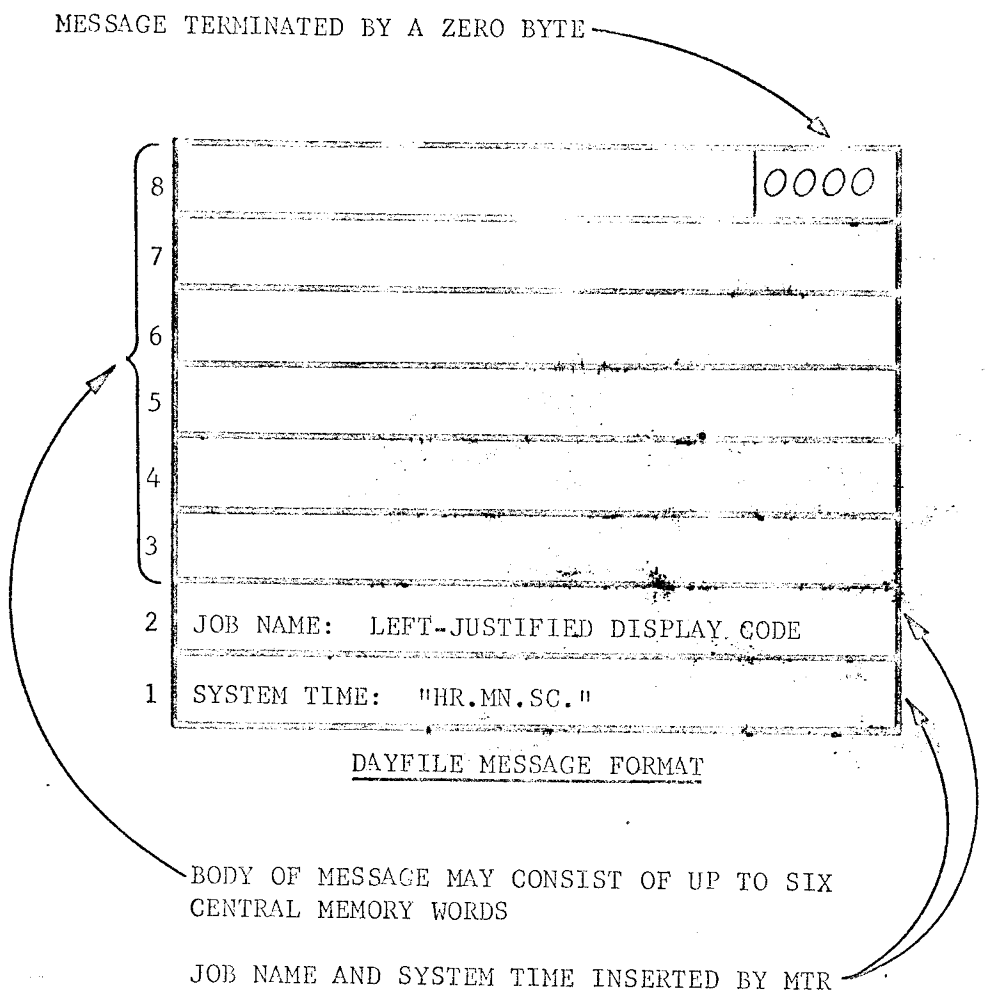
Messages are entered in the dayfile by peripheral processors via a request to MTR. A central processor program may enter a message in the dayfile by calling the MSG peripheral package. A peripheral processor program initiates the entry of a message in the dayfile by placing the message in its Message Buffer and then placing function code 1 (Process Dayfile Message) in byte one of its Output Register. The message may be up to six central memory words in length and is terminated by a zero byte in byte five of the last word of the message.

When MTR processes this request, it first checks the dayfile dump flag in location 64 of PP0's memory. If this flag is set, then dumping of the dayfile to the disk is in process, and so MTR returns to its master loop, delaying the processing of this message to later. If the dayfile dump flag is not set (i.e., location 64 contains zero), MTR proceeds with the processing of the message. The contents of the Message Buffer are copied into words 30 - 35 of the control point area for the control point to which the requesting processor is assigned. These locations (words 30 - 35), together with the control point status, the next control statement, and the exchange area, are displayed on the console by the DIS "B" display.

The dayfile message, together with the system time and the job name from the control point to which the requesting processor is assigned, is placed in the dayfile buffer. The dayfile buffer (see figure 9) is an area of central memory resident used to buffer dayfile messages to the disk. Its starting (i.e., FIRST) address and LIMIT address (last entry address plus one) are specified by bytes one and four, respectively, of the dayfile buffer (DFB) pointer word in central memory location 3. Bytes two and three of the DFB pointer contain the IN and OUT addresses for the buffer. In inserting the



MTR INSERTS NEXT MESSAGE HERE
TO BE DUMPED TO THE DISK
THIS DATA PREVIOUSLY WRITTEN TO THE DISK
DAYFILE BUFFER
FIRST
IN
OUT
LIMIT
DFB POINTER WORD (CM LOCATION 3)



MESSAGE TERMINATED BY A ZERO BYTE
8
7
6
5
4
3
2
1
0000
JOB NAME: LEFT-JUSTIFIED DISPLAY CODE
SYSTEM TIME: "HR.MN.SC."
DAYFILE MESSAGE FORMAT
BODY OF MESSAGE MAY CONSIST OF UP TO SIX CENTRAL MEMORY WORDS
JOB NAME AND SYSTEM TIME INSERTED BY MTR

THE DAYFILE

Figure 9

message in the dayfile buffer, MTR first copies the system time from location 30 of central memory resident into the buffer. Next, MTR reads the job name from word 21 of the requestor's control point area and copies it into the dayfile buffer. In doing so, MTR changes spaces in the job name to blanks, and inserts a period at the end of the job name. Next, MTR copies the body of the message from the requestor's Message Buffer into the dayfile buffer, copying word after word until a word ending with a zero byte (byte five) is copied.

Within the dayfile buffer, a message comprises three to eight words: one word contains the system time, one word contains the job name, and one to six words contain the body of the message. When the entire message has been copied into the dayfile buffer, MTR increments the dayfile message count in byte two of word 22 in the requestor's control point area. Although this count is incremented each time a message is entered in the dayfile, it is tested against a limit only by the peripheral package MSG.

As MTR enters each word in the dayfile buffer, it advances the IN address and compares it with the LIMIT address. When IN = LIMIT, MTR resets IN to the value of FIRST. After the message has been entered in the dayfile, MTR compares the IN and OUT addresses to determine if the dayfile buffer contains a full sector of data: if it does, the dayfile dump flag is set to initiate the dumping of this data to the disk. MTR dumps the dayfile to the disk in a series of phases: after each phase has been executed, MTR returns to its master loop to process requests from the central processor or from peripheral processors. In this manner, MTR avoids being tied up in a disk operation for a prolonged period of time. The dayfile dump flag, when set, contains the address of the subroutine to be called to perform the next phase of dumping.

Although the nominal size of the dayfile buffer is 1000_8 words, dumping is initiated whenever messages totalling 100_8 words have accumulated. From the standpoint of buffering messages to the disk, the dayfile buffer need be be no longer than 107_8 words (since it is possible that entry of the last message increased the total to over 100 words), since no entries can be made while dumping is in process. By increasing the buffer size to several times its minimum requirements, however, the size of the dayfile display on the console is increased.

The six subroutines corresponding to the dayfile dump phases are shown on pages A-16, A-17, and A-18 of the attached flow charts. These subroutines are described below.

<u>Phase 1</u>. In phase one, MTR requests channel 0 and sets the dump flag to the address of the phase 2 subroutine. It is interesting to note that in this case MTR transmits a request to itself: the channel number and the appropriate function number are placed in PPO's Output Register to be processed by MTR when it returns to its master loop.

<u>Phase 2</u>. One entering phase two, MTR reads its Output Register to determine if its reservation request has been accepted. If the channel has been reserved for MTR, then positioning is initiated. All other disk users maintain the current half track address for a file in the FST entry for that file. Although MTR sets the Beginning Half Track byte in the FST entry, it does not update the Current Half Track byte as sections of the dayfile are written to the disk. Instead, the current half track address is maintained by modifying the appropriate instructions within the dumping subroutines. To position the disk, MTR uses the Position Disk subroutine in peripheral processor resident. After initiating repositioning, the dump flag is set to the address of the phase 3 subroutine.

<u>Phase 3</u>. In phase three, MTR writes the full sector in the dayfile buffer and a record mark to the disk. Since this write is directed

to a specified sector, it is conceivable that up to 66 milliseconds could elapse between the time at which this subroutine was entered and the time at which this sector came under the heads. In order to avoid this delay, MTR issues a status request to obtain the number of the sector currently passing under the heads and, unless the disk is positioned two sectors before the desired sector, MTR returns to its master loop. A sector may pass under the heads in as little as 490 microseconds. The minimum time required for MTR to make a pass through its master loop is approximately 150 microseconds (assuming an active CPU program but no request processing required) and may be several times longer. It is not impossible, then, that a revolution or more may be required before the desired coincidence is found.

Once coincidence has been obtained, MTR writes the full sector from the dayfile buffer to the disk, and advances the buffer's OUT address accordingly. If this sector was the last sector on this half track, the subroutine coding is modified for the spare half track. (MTR maintains a spare half track for the dayfile: this is picked up in the phase four subroutine whenever required.)

It is probable that the message which completed a full sector in the dayfile buffer resulted in the buffer's containing something more than 100_8 words of data. If so, MTR will include these extra words, which are part of the last message entered in the dayfile buffer, in the short sector written as an end-of-record after each full sector is written. (The dayfile is a single logical record on the disk and is not terminated by a file mark sector.)

The phase two subroutine is called again to position the disk, and the short sector is then written by the phase three subroutine.

Although this sector may include a few words of data from the buffer, the OUT pointer is not advanced to reflect the transfer of these words: also, the coding is not modified to reflect the writing of this sector. The next full sector written to the disk will also include these few words and will be written over this end-of-record sector.

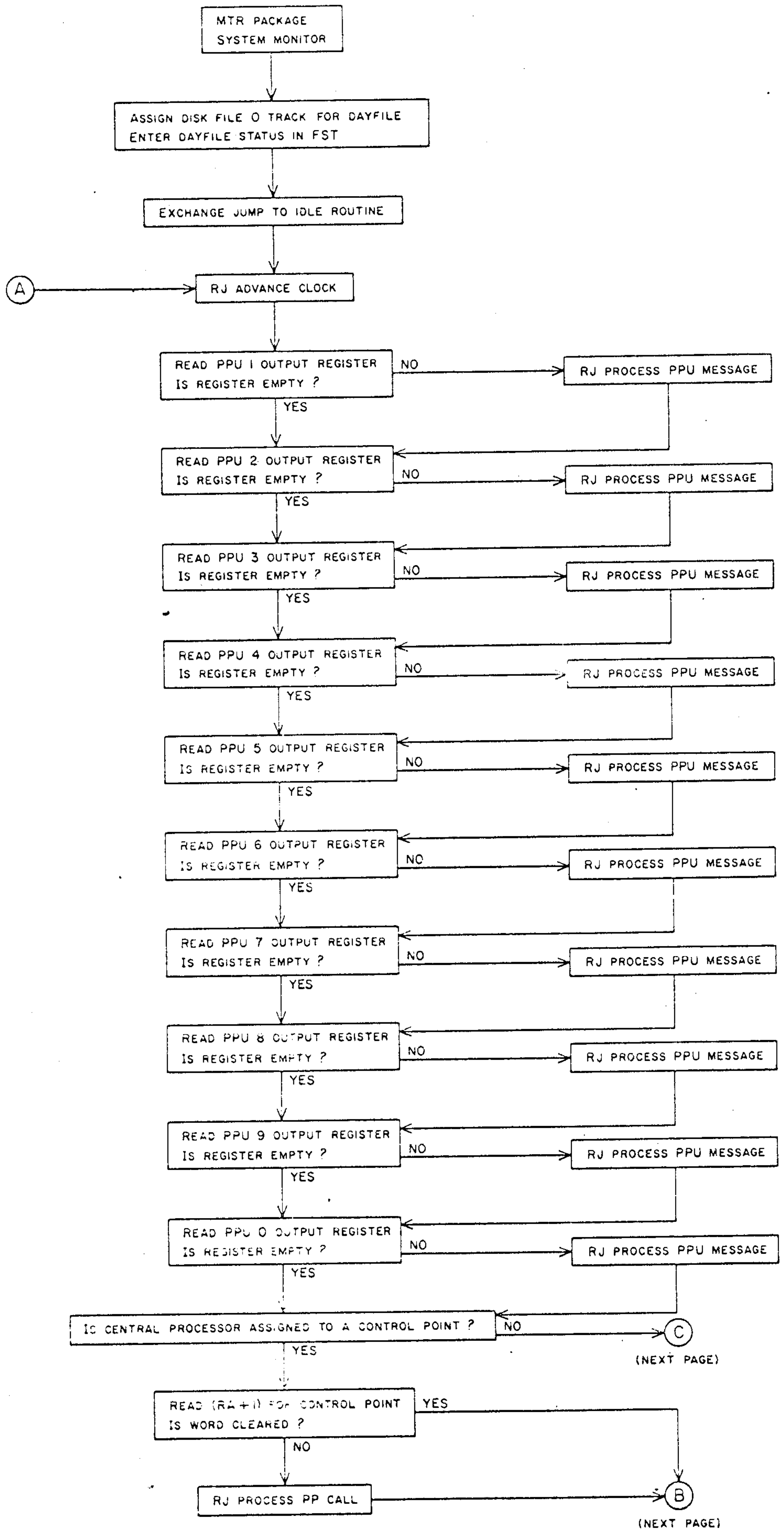
After the end-of-record sector has been written, MTR constructs a release channel reservation request by placing the channel number and the appropriate function code in PPO's Output Register. It then sets the dump flag to either the address of the phase four subroutine (if another spare half track is required) or the phase six subroutine (if the spare half track was not used during this dump operation).

Phases 4, 5, 6. The phase four subroutine requests a spare half track from MTR and sets the dump flag to the address of the phase five subroutine. The phase five subroutine stores the spare half track number and clears the dump flag.

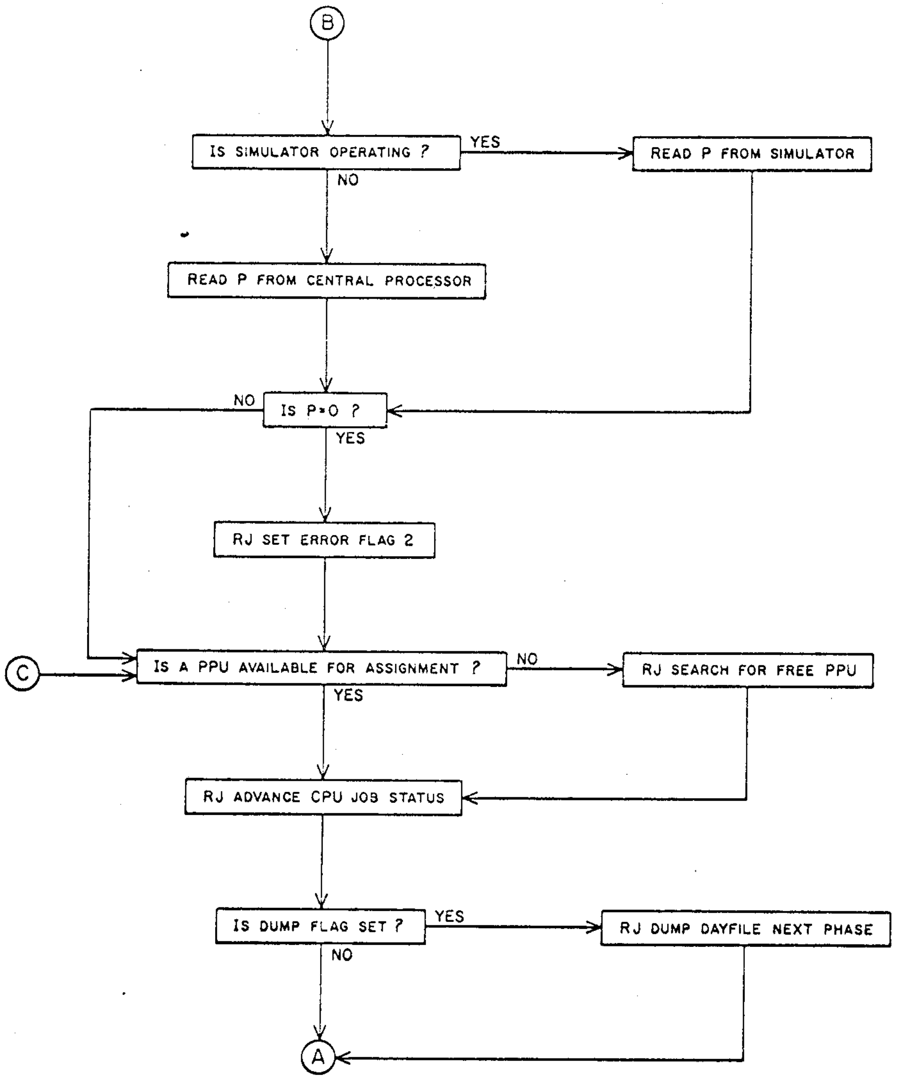
If it was not necessary to pick up the spare half track, the phase three subroutine sets the dump flag for phase six. The phase six subroutine clears the dump flag.

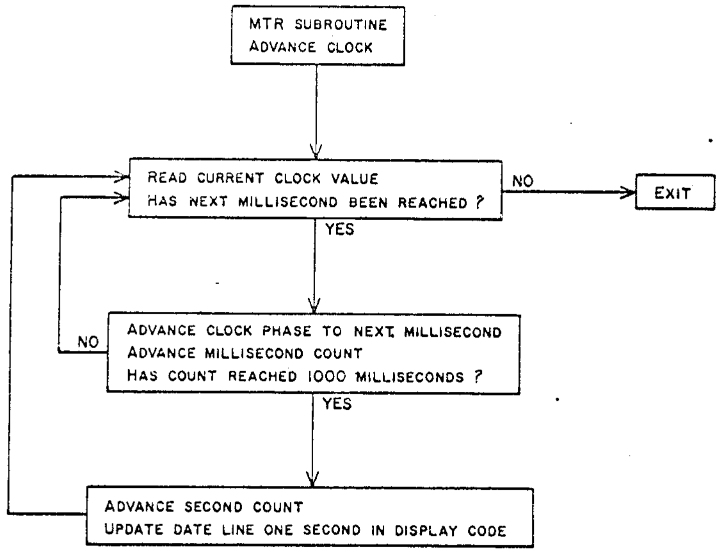
The dayfile buffer is dumped whenever messages totalling a full sector have accumulated. It is also dumped, even though a full sector has not been accumulated, at the end of each job. As part of a job's output, all dayfile messages for that job are printed. In order to simplify searching of the dayfile for the job's messages, the dayfile is dumped to the disk so that only the disk has to be searched. The PRINT package (1DJ and its overlays) and the DUMP package (1TD and its overlays) initiate this dumping by sending function request 11, Complete Dayfile, to MTR: MTR in turn sets the dayfile dump flag to phase one.

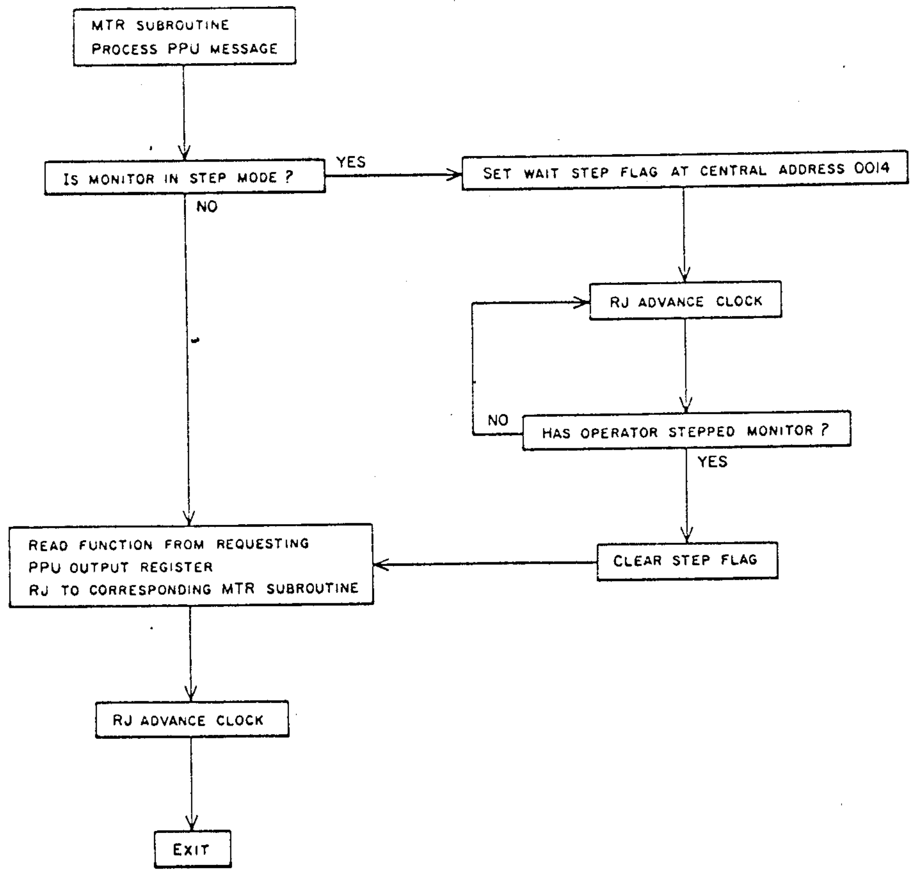
The PRINT (or DUMP) package calls 2RD to load a central memory buffer with data from the dayfile, and then calls the Search Dayfile overlay (2SD) to extract the messages for the specified job. 2SD searches each word in the buffer for the specified job name. When a word is read which does not contain the job name, it is copied into a peripheral processor memory area, but the address in this area is not advanced. Thus, when a word containing the job name is found, the peripheral processor memory area already contains the system time associated with the message. The remainder of the message is then transferred to the peripheral processor memory area: word after word is transferred until a word ending in a zero byte is found. The sectors comprising the dayfile are searched until a short sector has been processed. When 2SD recognizes that the end of the dayfile has been reached, it requests MTR to compute the time required to process this job's output. MTR stores this time in word 24 of the control point area: 2SD reads it, converts it to decimal seconds, and writes it in a central memory buffer. The PRINT (or DUMP) package then copies the contents of the peripheral processor memory area which contains the dayfile messages for the job to this central memory buffer. The PRINT package then calls the 2LP overlay to print the contents of this buffer as the last page of a job's output.

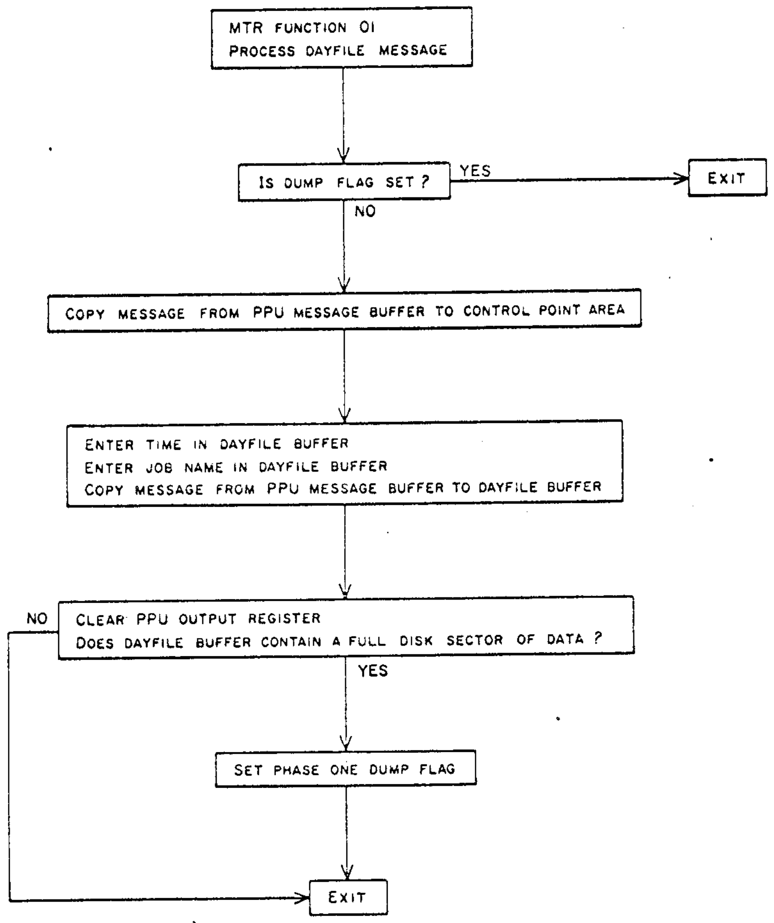

MTR PACKAGE
SYSTEM MONITOR
ASSIGN DISK FILE 0 TRACK FOR DAYFILE
ENTER DAYFILE STATUS IN FST
EXCHANGE JUMP TO IDLE ROUTINE
A
RJ ADVANCE CLOCK
READ PPU 1 OUTPUT REGISTER
IS REGISTER EMPTY ?
NO
RJ PROCESS PPU MESSAGE
YES
READ PPU 2 OUTPUT REGISTER
IS REGISTER EMPTY ?
NO
RJ PROCESS PPU MESSAGE
YES
READ PPU 3 OUTPUT REGISTER
IS REGISTER EMPTY ?
NO
RJ PROCESS PPU MESSAGE
YES
READ PPU 4 OUTPUT REGISTER
IS REGISTER EMPTY ?
NO
RJ PROCESS PPU MESSAGE
YES
READ PPU 5 OUTPUT REGISTER
IS REGISTER EMPTY ?
NO
RJ PROCESS PPU MESSAGE
YES
READ PPU 6 OUTPUT REGISTER
IS REGISTER EMPTY ?
NO
RJ PROCESS PPU MESSAGE
YES
READ PPU 7 OUTPUT REGISTER
IS REGISTER EMPTY ?
NO
RJ PROCESS PPU MESSAGE
YES
READ PPU 8 OUTPUT REGISTER
IS REGISTER EMPTY ?
NO
RJ PROCESS PPU MESSAGE
YES
READ PPU 9 OUTPUT REGISTER
IS REGISTER EMPTY ?
NO
RJ PROCESS PPU MESSAGE
YES
READ PPU 0 OUTPUT REGISTER
IS REGISTER EMPTY ?
NO
RJ PROCESS PPU MESSAGE
YES
IS CENTRAL PROCESSOR ASSIGNED TO A CONTROL POINT ?
NO
C
(NEXT PAGE)
YES
READ (RA+1) FOR CONTROL POINT
IS WORD CLEARED ?
YES
NO
RJ PROCESS PP CALL
B
(NEXT PAGE)

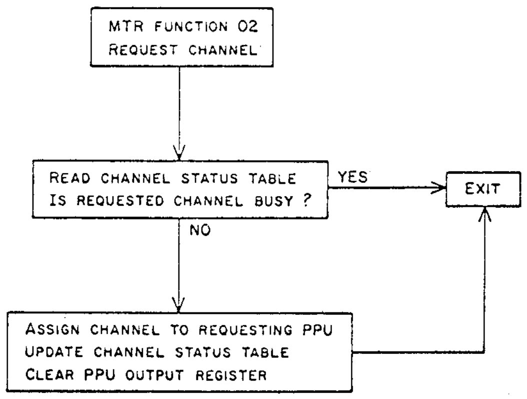
(MTR PACKAGE CONTINUED)

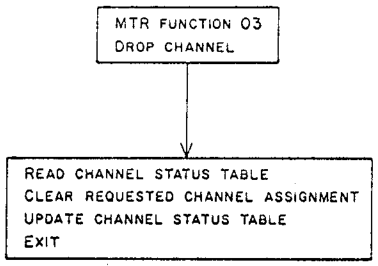


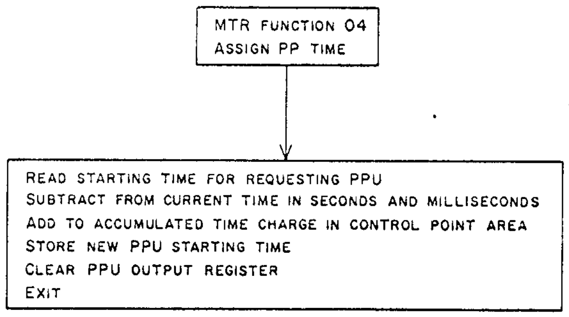

MTR SUBROUTINE
ADVANCE CLOCK
READ CURRENT CLOCK VALUE
HAS NEXT MILLISECOND BEEN REACHED ?
NO
EXIT
YES
NO
ADVANCE CLOCK PHASE TO NEXT MILLISECOND
ADVANCE MILLISECOND COUNT
HAS COUNT REACHED 1000 MILLISECONDS ?
YES
ADVANCE SECOND COUNT
UPDATE DATE LINE ONE SECOND IN DISPLAY CODE

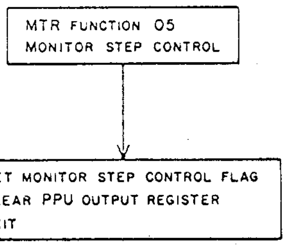

MTR SUBROUTINE
PROCESS PPU MESSAGE
IS MONITOR IN STEP MODE ?
YES
NO
SET WAIT STEP FLAG AT CENTRAL ADDRESS 0014
RJ ADVANCE CLOCK
NO
HAS OPERATOR STEPPED MONITOR ?
YES
CLEAR STEP FLAG
READ FUNCTION FROM REQUESTING PPU OUTPUT REGISTER RJ TO CORRESPONDING MTR SUBROUTINE
RJ ADVANCE CLOCK
EXIT

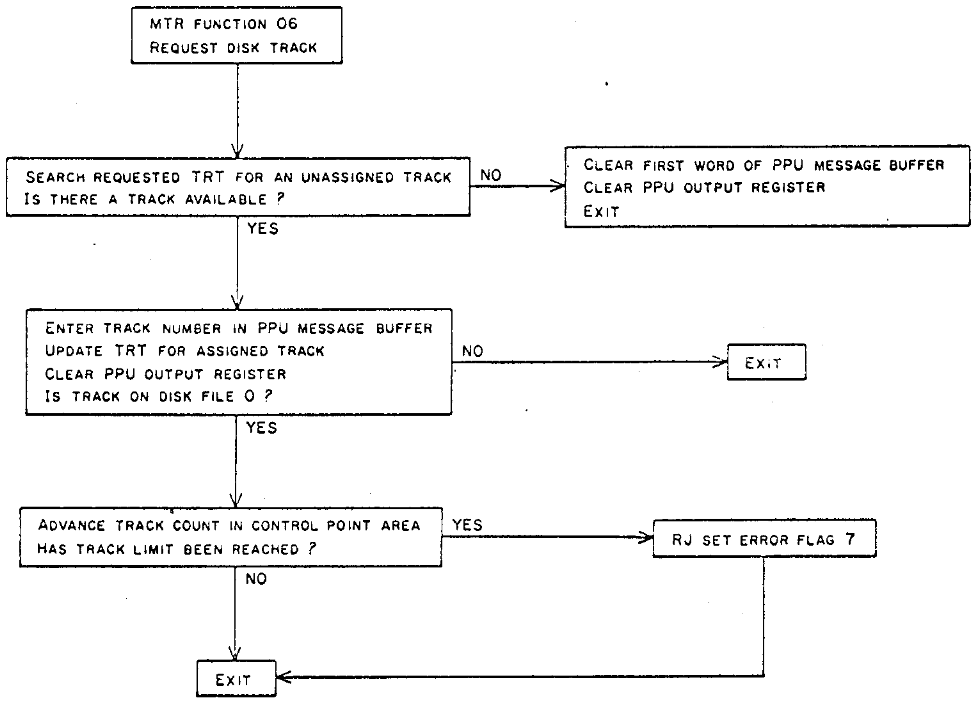

MTR FUNCTION 01
PROCESS DAYFILE MESSAGE
IS DUMP FLAG SET ?
YES
EXIT
NO
COPY MESSAGE FROM PPU MESSAGE BUFFER TO CONTROL POINT AREA
ENTER TIME IN DAYFILE BUFFER
ENTER JOB NAME IN DAYFILE BUFFER
COPY MESSAGE FROM PPU MESSAGE BUFFER TO DAYFILE BUFFER
NO
CLEAR PPU OUTPUT REGISTER
DOES DAYFILE BUFFER CONTAIN A FULL DISK SECTOR OF DATA ?
YES
SET PHASE ONE DUMP FLAG
EXIT

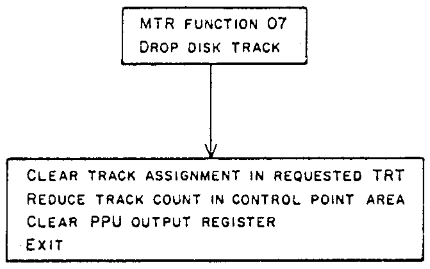

MTR FUNCTION 02
REQUEST CHANNEL
READ CHANNEL STATUS TABLE
IS REQUESTED CHANNEL BUSY ?
YES
EXIT
NO
ASSIGN CHANNEL TO REQUESTING PPU
UPDATE CHANNEL STATUS TABLE
CLEAR PPU OUTPUT REGISTER

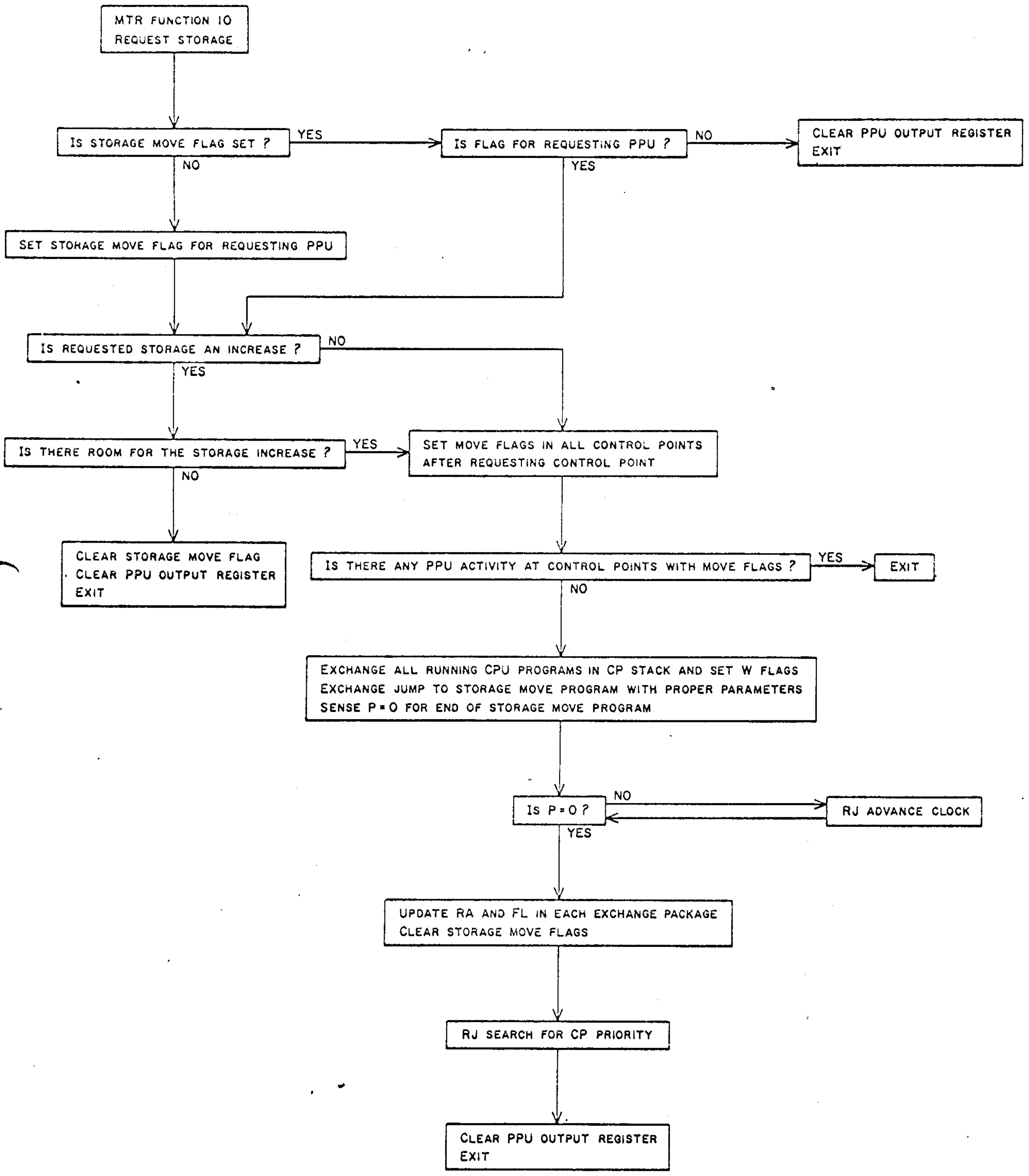

MTR FUNCTION 03
DROP CHANNEL
READ CHANNEL STATUS TABLE
CLEAR REQUESTED CHANNEL ASSIGNMENT
UPDATE CHANNEL STATUS TABLE
EXIT

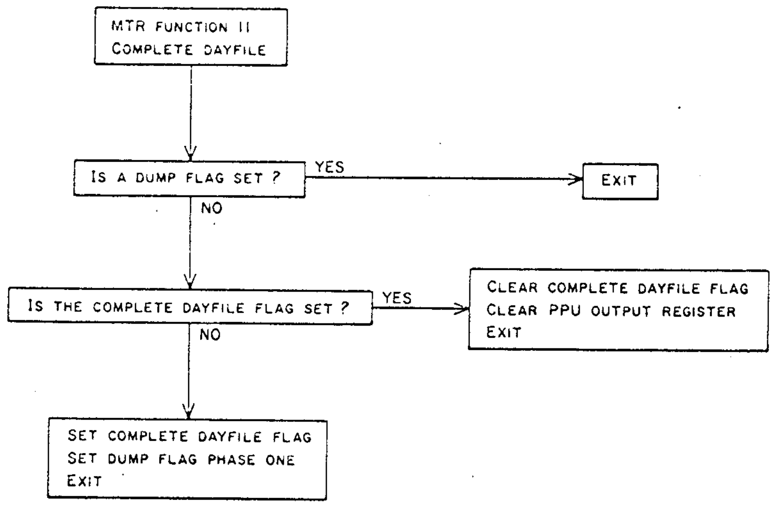

MTR FUNCTION 04
ASSIGN PP TIME
READ STARTING TIME FOR REQUESTING PPU
SUBTRACT FROM CURRENT TIME IN SECONDS AND MILLISECONDS
ADD TO ACCUMULATED TIME CHARGE IN CONTROL POINT AREA
STORE NEW PPU STARTING TIME
CLEAR PPU OUTPUT REGISTER
EXIT

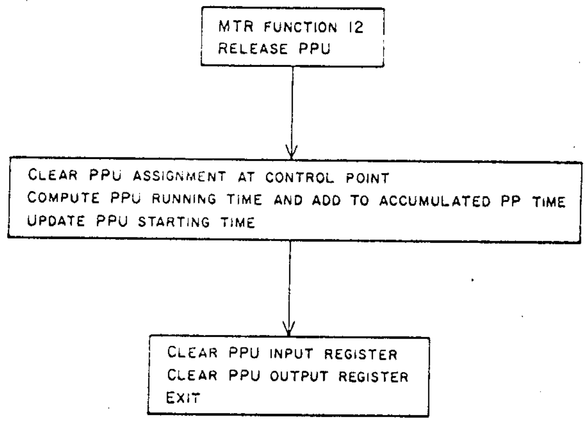

MTR FUNCTION 05
MONITOR STEP CONTROL
SET MONITOR STEP CONTROL FLAG
CLEAR PPU OUTPUT REGISTER
EXIT

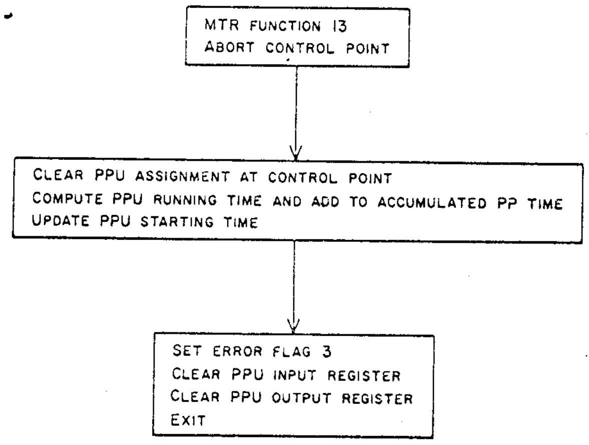

MTR FUNCTION 06
REQUEST DISK TRACK
SEARCH REQUESTED TRT FOR AN UNASSIGNED TRACK
IS THERE A TRACK AVAILABLE ?
NO
CLEAR FIRST WORD OF PPU MESSAGE BUFFER
CLEAR PPU OUTPUT REGISTER
EXIT
YES
ENTER TRACK NUMBER IN PPU MESSAGE BUFFER
UPDATE TRT FOR ASSIGNED TRACK
CLEAR PPU OUTPUT REGISTER
IS TRACK ON DISK FILE 0 ?
NO
EXIT
YES
ADVANCE TRACK COUNT IN CONTROL POINT AREA
HAS TRACK LIMIT BEEN REACHED ?
YES
RJ SET ERROR FLAG 7
NO
EXIT

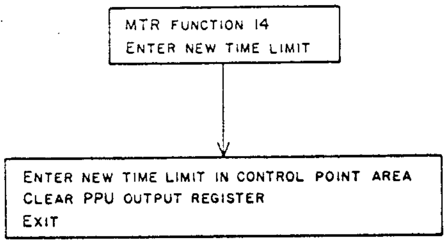

MTR FUNCTION 07
DROP DISK TRACK
CLEAR TRACK ASSIGNMENT IN REQUESTED TRT
REDUCE TRACK COUNT IN CONTROL POINT AREA
CLEAR PPU OUTPUT REGISTER
EXIT

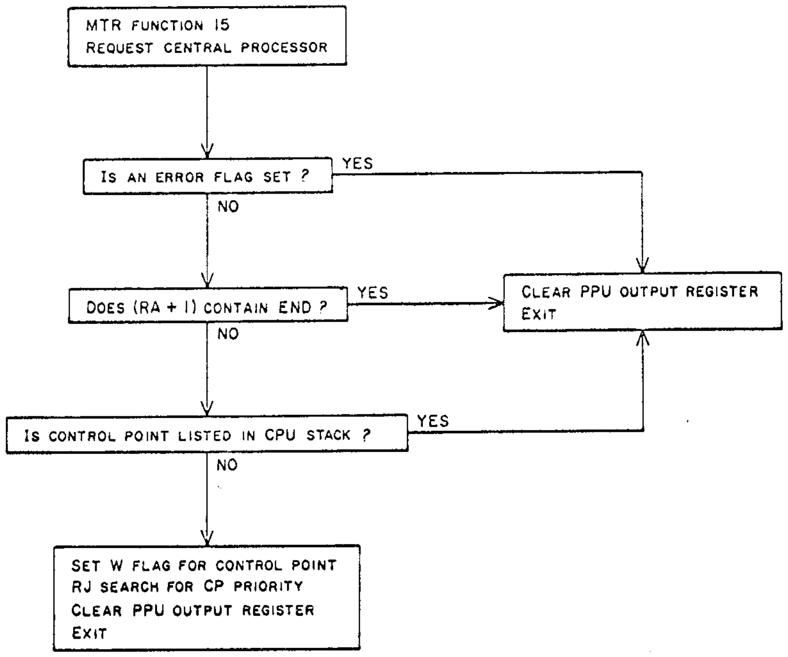

MTR FUNCTION 10
REQUEST STORAGE
IS STORAGE MOVE FLAG SET ?
YES
NO
IS FLAG FOR REQUESTING PPU ?
NO
YES
CLEAR PPU OUTPUT REGISTER
EXIT
SET STORAGE MOVE FLAG FOR REQUESTING PPU
IS REQUESTED STORAGE AN INCREASE ?
NO
YES
IS THERE ROOM FOR THE STORAGE INCREASE ?
YES
NO
SET MOVE FLAGS IN ALL CONTROL POINTS
AFTER REQUESTING CONTROL POINT
CLEAR STORAGE MOVE FLAG
CLEAR PPU OUTPUT REGISTER
EXIT
IS THERE ANY PPU ACTIVITY AT CONTROL POINTS WITH MOVE FLAGS ?
YES
EXIT
NO
EXCHANGE ALL RUNNING CPU PROGRAMS IN CP STACK AND SET W FLAGS
EXCHANGE JUMP TO STORAGE MOVE PROGRAM WITH PROPER PARAMETERS
SENSE P = 0 FOR END OF STORAGE MOVE PROGRAM
IS P = 0 ?
NO
RJ ADVANCE CLOCK
YES
UPDATE RA AND FL IN EACH EXCHANGE PACKAGE
CLEAR STORAGE MOVE FLAGS
RJ SEARCH FOR CP PRIORITY
CLEAR PPU OUTPUT REGISTER
EXIT

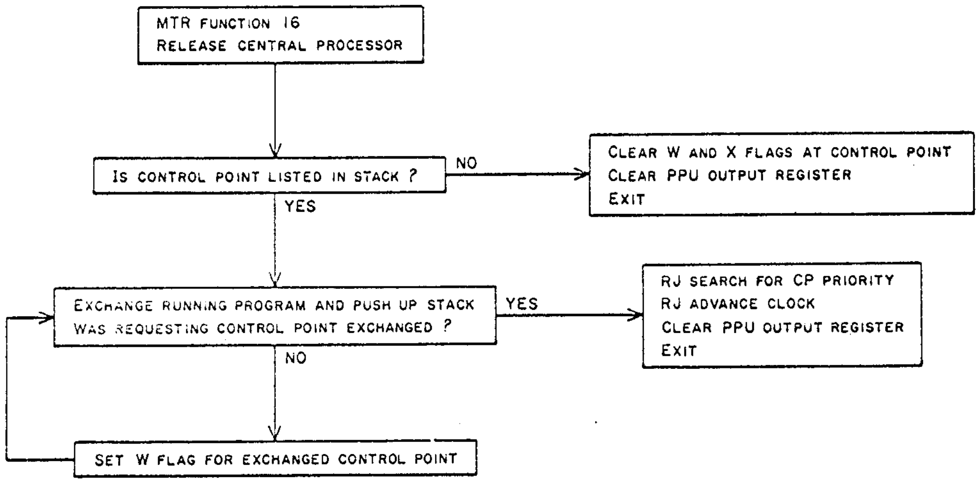

MTR FUNCTION 11
COMPLETE DAYFILE
IS A DUMP FLAG SET ?
YES
EXIT
NO
IS THE COMPLETE DAYFILE FLAG SET ?
YES
CLEAR COMPLETE DAYFILE FLAG
CLEAR PPU OUTPUT REGISTER
EXIT
NO
SET COMPLETE DAYFILE FLAG
SET DUMP FLAG PHASE ONE
EXIT

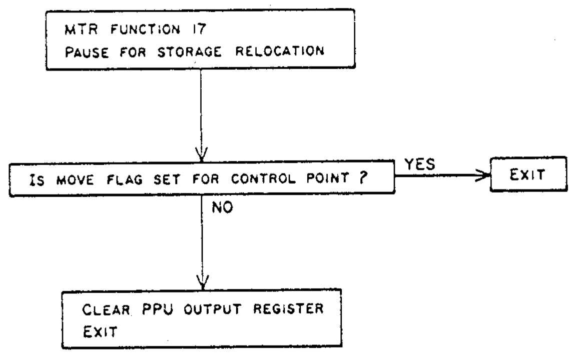

MTR FUNCTION 12
RELEASE PPU
CLEAR PPU ASSIGNMENT AT CONTROL POINT
COMPUTE PPU RUNNING TIME AND ADD TO ACCUMULATED PP TIME
UPDATE PPU STARTING TIME
CLEAR PPU INPUT REGISTER
CLEAR PPU OUTPUT REGISTER
EXIT

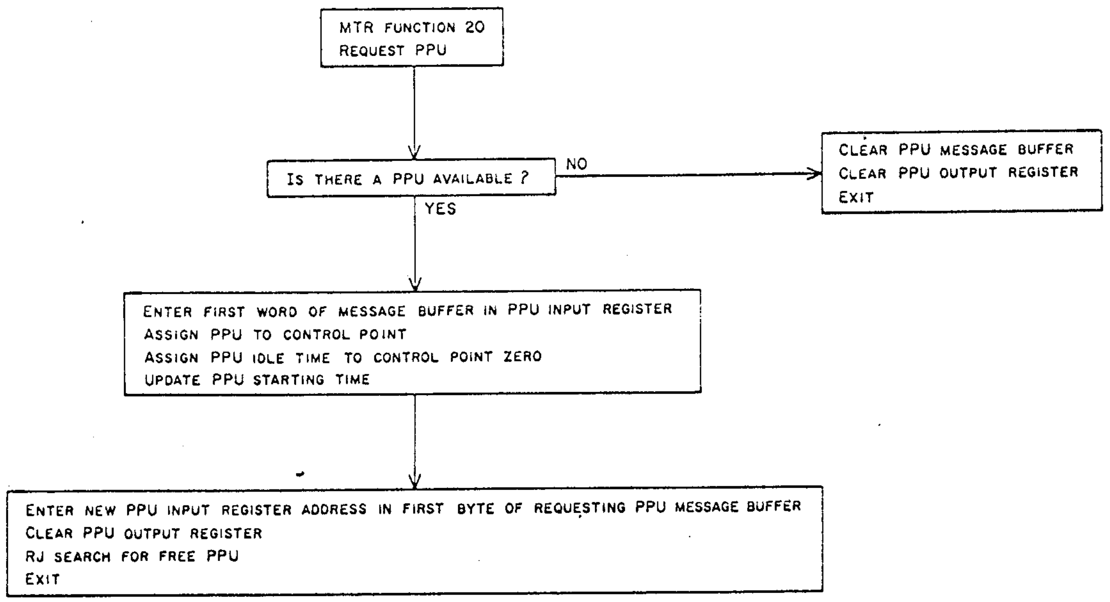

MTR FUNCTION 13
ABORT CONTROL POINT
CLEAR PPU ASSIGNMENT AT CONTROL POINT
COMPUTE PPU RUNNING TIME AND ADD TO ACCUMULATED PP TIME
UPDATE PPU STARTING TIME
SET ERROR FLAG 3
CLEAR PPU INPUT REGISTER
CLEAR PPU OUTPUT REGISTER
EXIT

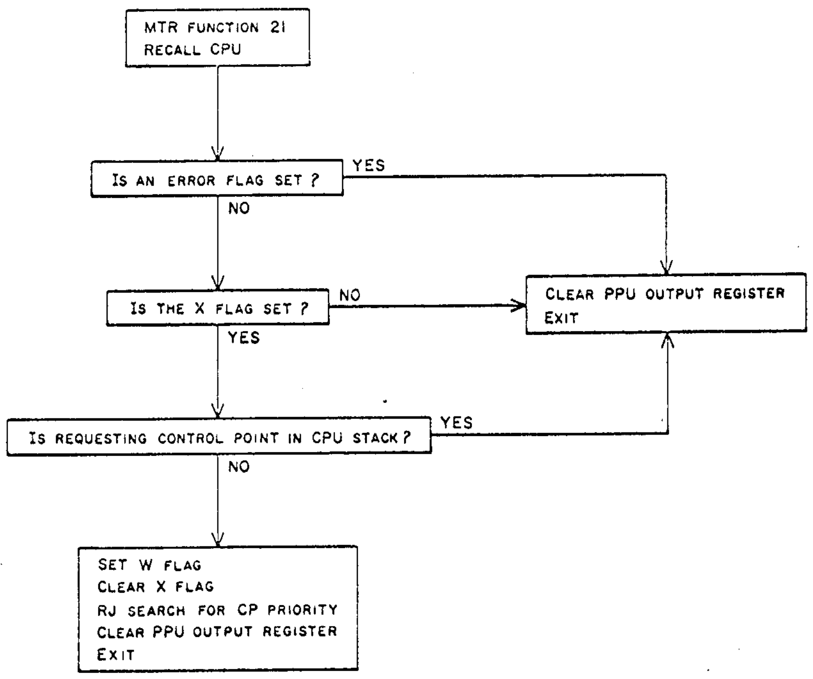

MTR FUNCTION 14
ENTER NEW TIME LIMIT
ENTER NEW TIME LIMIT IN CONTROL POINT AREA
CLEAR PPU OUTPUT REGISTER
EXIT

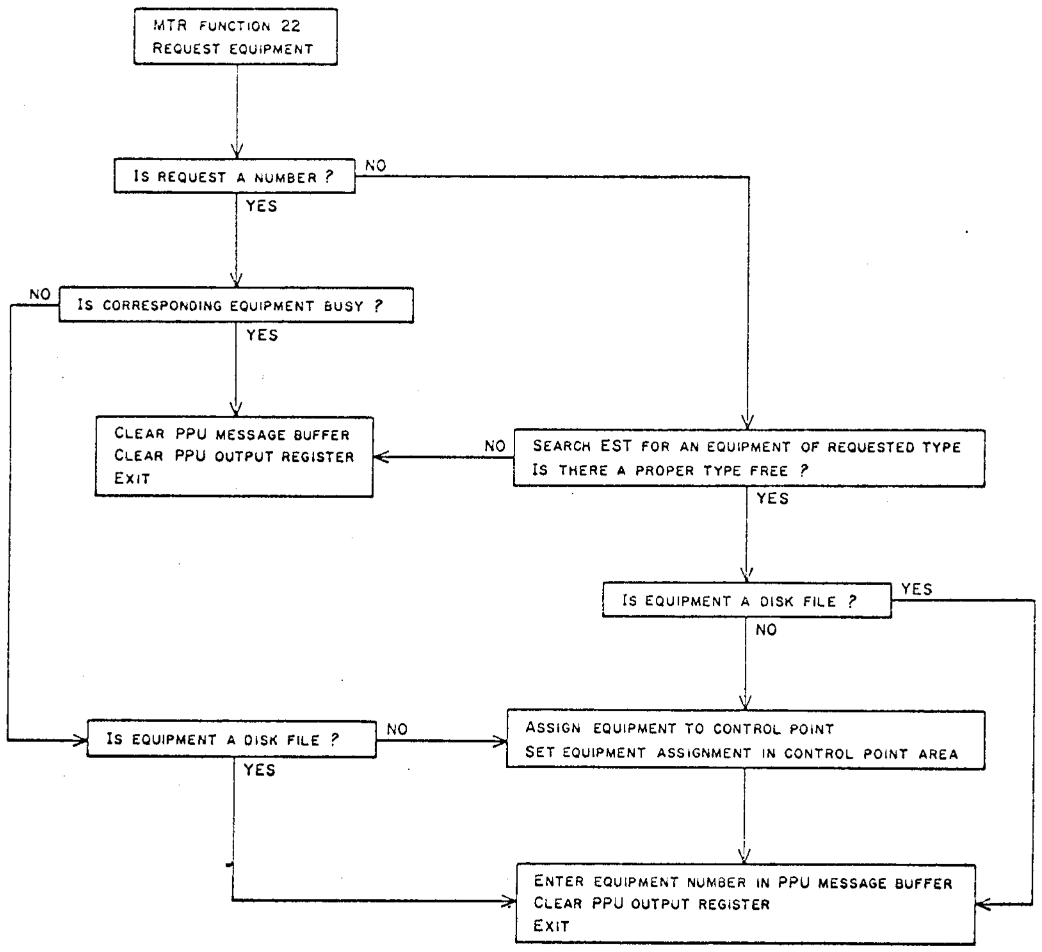

MTR FUNCTION 15
REQUEST CENTRAL PROCESSOR
IS AN ERROR FLAG SET ?
YES
NO
DOES (RA + 1) CONTAIN END ?
YES
NO
CLEAR PPU OUTPUT REGISTER
EXIT
IS CONTROL POINT LISTED IN CPU STACK ?
YES
NO
SET W FLAG FOR CONTROL POINT
RJ SEARCH FOR CP PRIORITY
CLEAR PPU OUTPUT REGISTER
EXIT

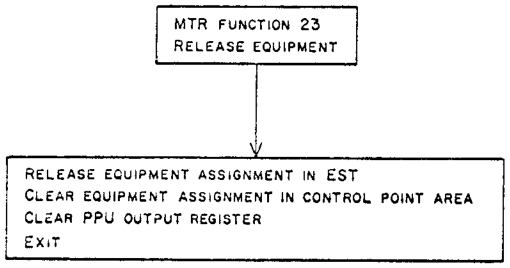

MTR FUNCTION 16
RELEASE CENTRAL PROCESSOR
IS CONTROL POINT LISTED IN STACK ?
NO
CLEAR W AND X FLAGS AT CONTROL POINT
CLEAR PPU OUTPUT REGISTER
EXIT
YES
EXCHANGE RUNNING PROGRAM AND PUSH UP STACK
WAS REQUESTING CONTROL POINT EXCHANGED ?
YES
RJ SEARCH FOR CP PRIORITY
RJ ADVANCE CLOCK
CLEAR PPU OUTPUT REGISTER
EXIT
NO
SET W FLAG FOR EXCHANGED CONTROL POINT

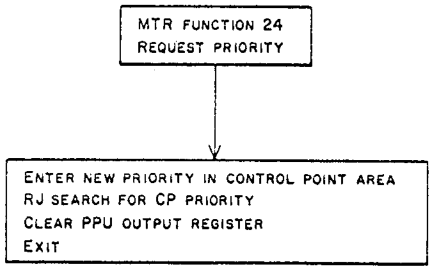

MTR FUNCTION 17
PAUSE FOR STORAGE RELOCATION
IS MOVE FLAG SET FOR CONTROL POINT ?
YES
EXIT
NO
CLEAR PPU OUTPUT REGISTER
EXIT

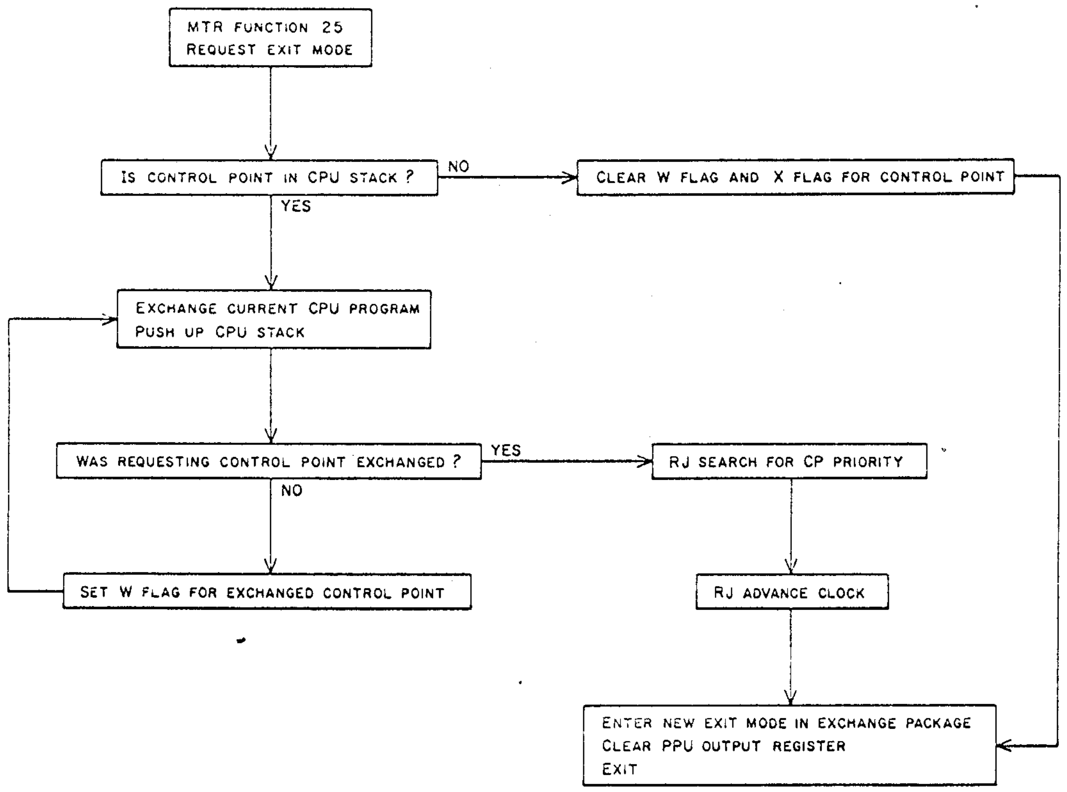

MTR FUNCTION 20
REQUEST PPU
IS THERE A PPU AVAILABLE ?
NO
CLEAR PPU MESSAGE BUFFER
CLEAR PPU OUTPUT REGISTER
EXIT
YES
ENTER FIRST WORD OF MESSAGE BUFFER IN PPU INPUT REGISTER
ASSIGN PPU TO CONTROL POINT
ASSIGN PPU IDLE TIME TO CONTROL POINT ZERO
UPDATE PPU STARTING TIME
ENTER NEW PPU INPUT REGISTER ADDRESS IN FIRST BYTE OF REQUESTING PPU MESSAGE BUFFER
CLEAR PPU OUTPUT REGISTER
RJ SEARCH FOR FREE PPU
EXIT

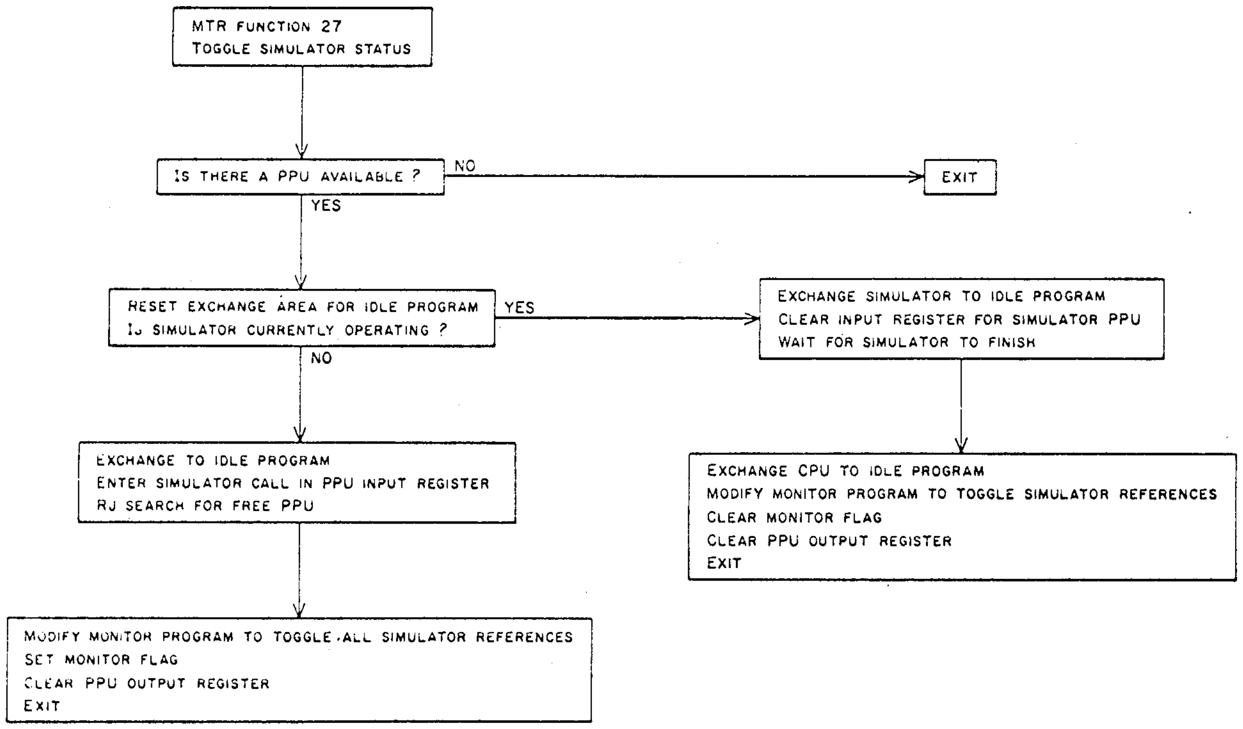

MTR FUNCTION 21
RECALL CPU
IS AN ERROR FLAG SET ?
YES
NO
IS THE X FLAG SET ?
NO
CLEAR PPU OUTPUT REGISTER
EXIT
YES
IS REQUESTING CONTROL POINT IN CPU STACK ?
YES
NO
SET W FLAG
CLEAR X FLAG
RJ SEARCH FOR CP PRIORITY
CLEAR PPU OUTPUT REGISTER
EXIT

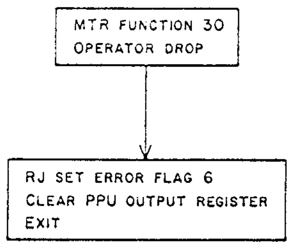

MTR FUNCTION 22
REQUEST EQUIPMENT
IS REQUEST A NUMBER ?
NO
YES
NO
IS CORRESPONDING EQUIPMENT BUSY ?
YES
CLEAR PPU MESSAGE BUFFER
CLEAR PPU OUTPUT REGISTER
EXIT
NO
SEARCH EST FOR AN EQUIPMENT OF REQUESTED TYPE
IS THERE A PROPER TYPE FREE ?
YES
IS EQUIPMENT A DISK FILE ?
YES
NO
IS EQUIPMENT A DISK FILE ?
NO
YES
ASSIGN EQUIPMENT TO CONTROL POINT
SET EQUIPMENT ASSIGNMENT IN CONTROL POINT AREA
ENTER EQUIPMENT NUMBER IN PPU MESSAGE BUFFER
CLEAR PPU OUTPUT REGISTER
EXIT

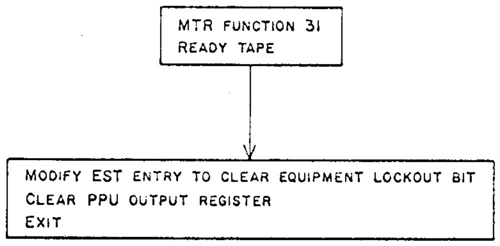

MTR FUNCTION 23
RELEASE EQUIPMENT
RELEASE EQUIPMENT ASSIGNMENT IN EST
CLEAR EQUIPMENT ASSIGNMENT IN CONTROL POINT AREA
CLEAR PPU OUTPUT REGISTER
EXIT

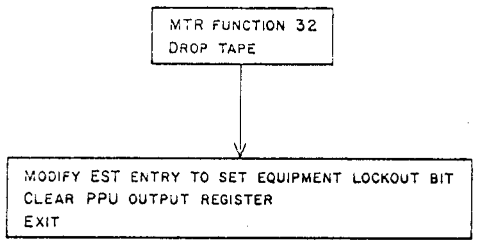

MTR FUNCTION 24
REQUEST PRIORITY
ENTER NEW PRIORITY IN CONTROL POINT AREA
RJ SEARCH FOR CP PRIORITY
CLEAR PPU OUTPUT REGISTER
EXIT

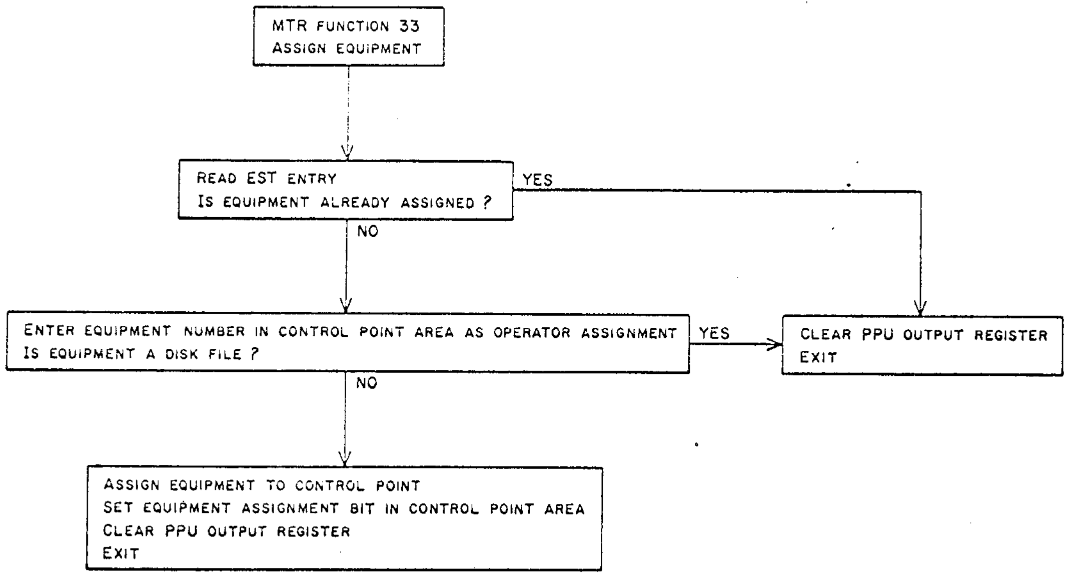

MTR FUNCTION 25
REQUEST EXIT MODE
IS CONTROL POINT IN CPU STACK ?
NO
CLEAR W FLAG AND X FLAG FOR CONTROL POINT
YES
EXCHANGE CURRENT CPU PROGRAM
PUSH UP CPU STACK
WAS REQUESTING CONTROL POINT EXCHANGED ?
YES
RJ SEARCH FOR CP PRIORITY
NO
SET W FLAG FOR EXCHANGED CONTROL POINT
RJ ADVANCE CLOCK
ENTER NEW EXIT MODE IN EXCHANGE PACKAGE
CLEAR PPU OUTPUT REGISTER
EXIT

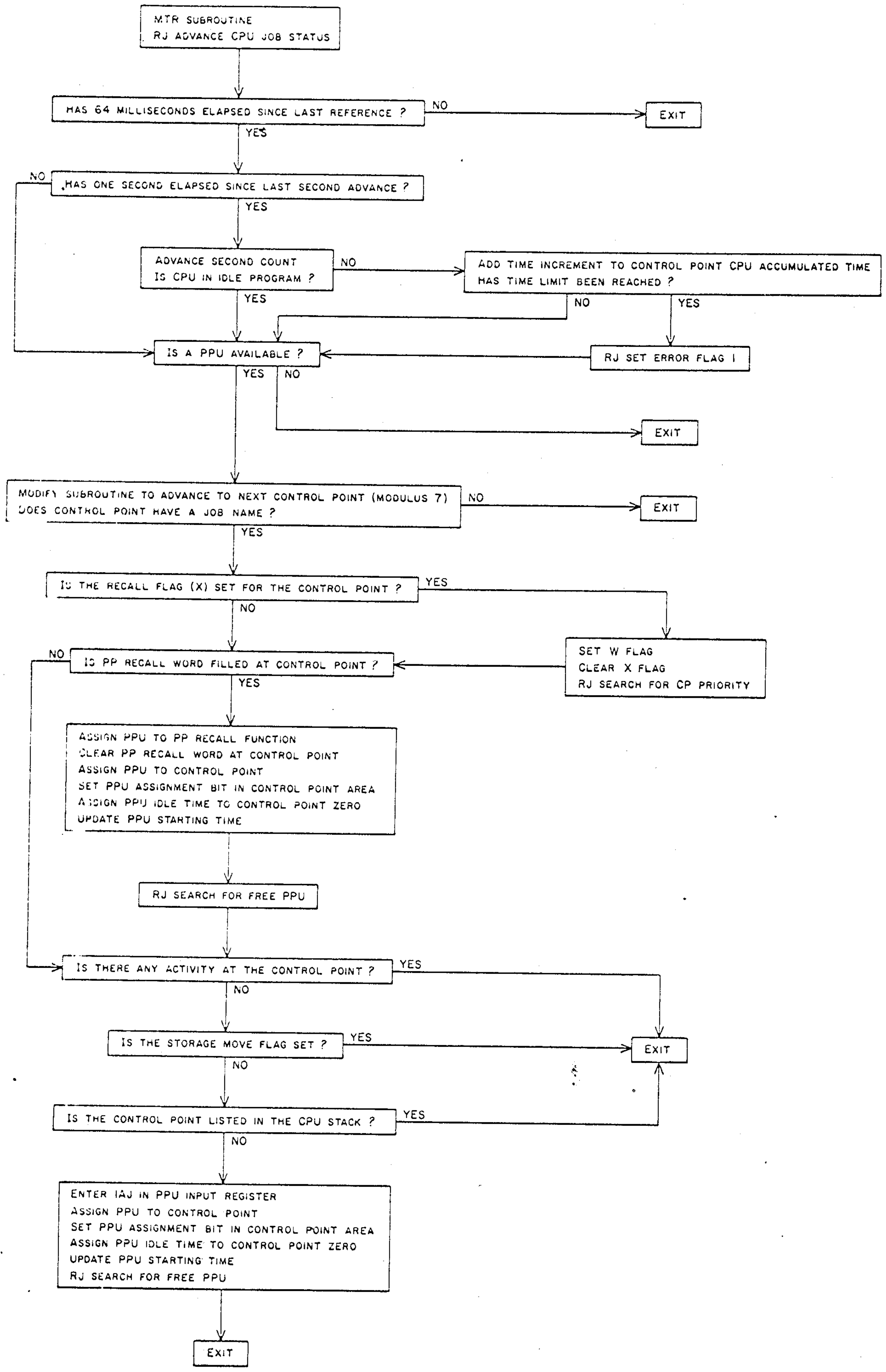

MTR FUNCTION 27
TOGGLE SIMULATOR STATUS
IS THERE A PPU AVAILABLE ?
NO
EXIT
YES
RESET EXCHANGE AREA FOR IDLE PROGRAM
IS SIMULATOR CURRENTLY OPERATING ?
YES
EXCHANGE SIMULATOR TO IDLE PROGRAM
CLEAR INPUT REGISTER FOR SIMULATOR PPU
WAIT FOR SIMULATOR TO FINISH
NO
EXCHANGE TO IDLE PROGRAM
ENTER SIMULATOR CALL IN PPU INPUT REGISTER
RJ SEARCH FOR FREE PPU
EXCHANGE CPU TO IDLE PROGRAM
MODIFY MONITOR PROGRAM TO TOGGLE SIMULATOR REFERENCES
CLEAR MONITOR FLAG
CLEAR PPU OUTPUT REGISTER
EXIT
MODIFY MONITOR PROGRAM TO TOGGLE ALL SIMULATOR REFERENCES
SET MONITOR FLAG
CLEAR PPU OUTPUT REGISTER
EXIT

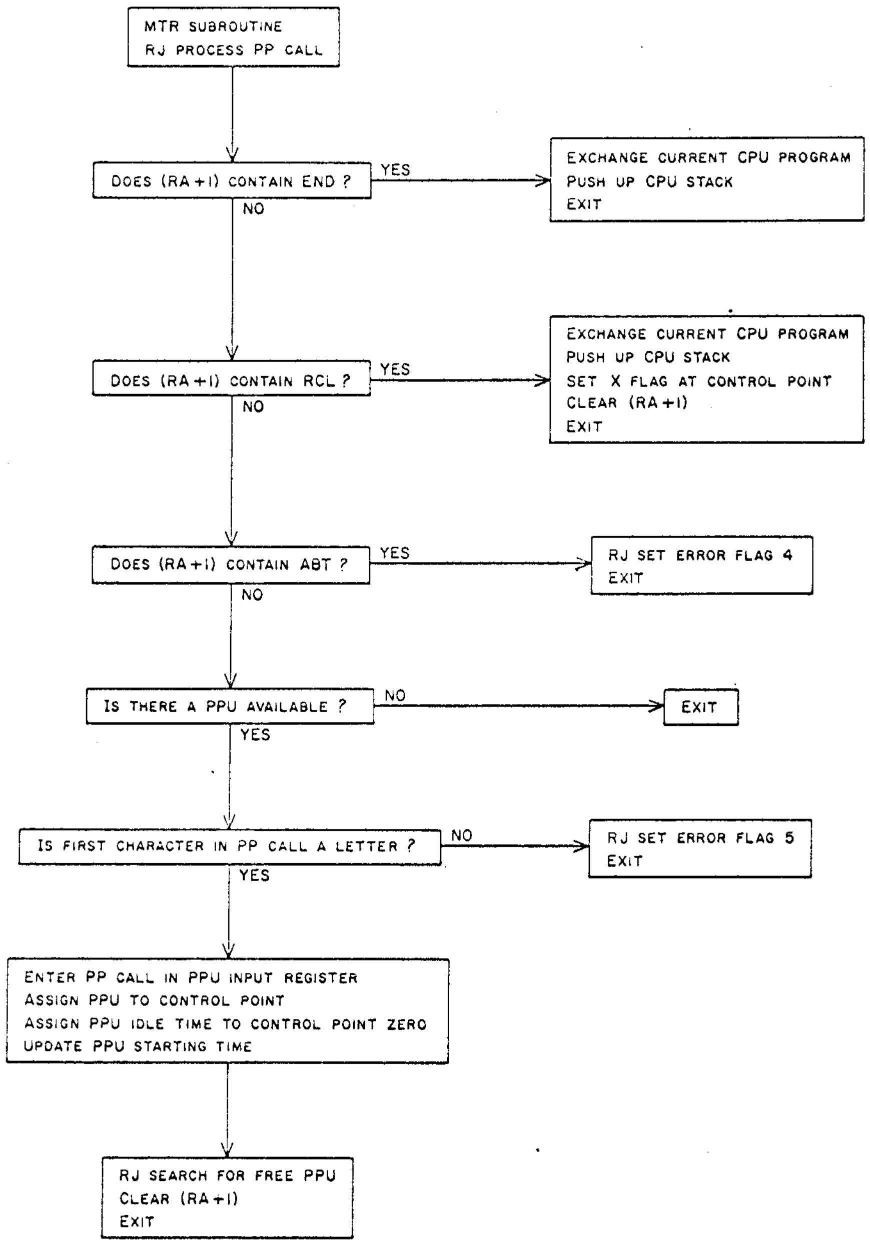

MTR FUNCTION 30
OPERATOR DROP
RJ SET ERROR FLAG 6
CLEAR PPU OUTPUT REGISTER
EXIT

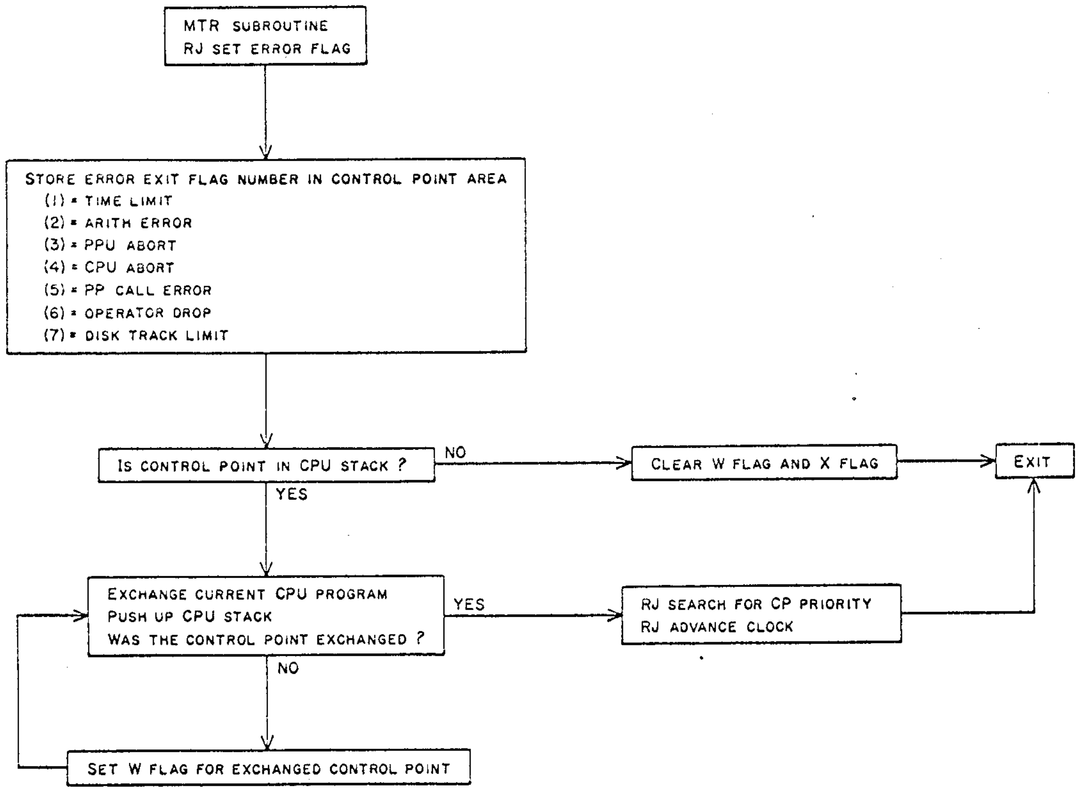

MTR FUNCTION 31
READY TAPE
MODIFY EST ENTRY TO CLEAR EQUIPMENT LOCKOUT BIT
CLEAR PPU OUTPUT REGISTER
EXIT


MTR FUNCTION 32
DROP TAPE
MODIFY EST ENTRY TO SET EQUIPMENT LOCKOUT BIT
CLEAR PPU OUTPUT REGISTER
EXIT

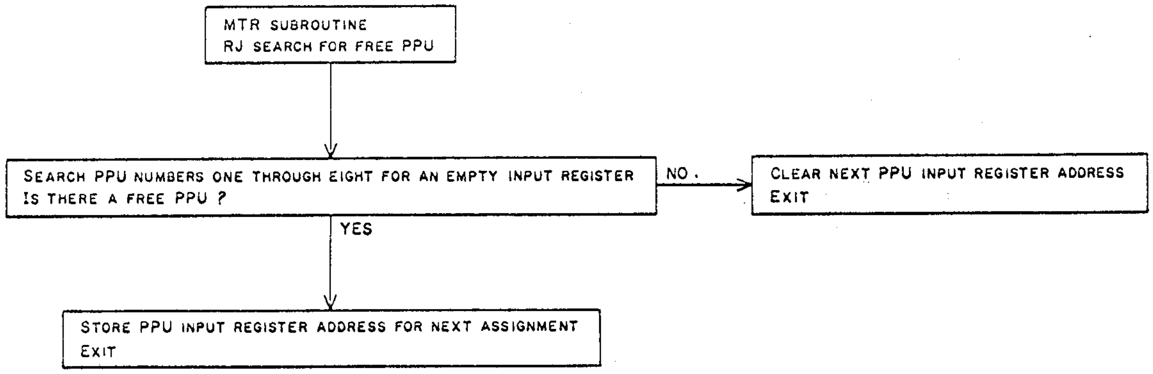

MTR FUNCTION 33
ASSIGN EQUIPMENT
READ EST ENTRY
IS EQUIPMENT ALREADY ASSIGNED ?
YES
NO
ENTER EQUIPMENT NUMBER IN CONTROL POINT AREA AS OPERATOR ASSIGNMENT
IS EQUIPMENT A DISK FILE ?
YES
NO
CLEAR PPU OUTPUT REGISTER
EXIT
ASSIGN EQUIPMENT TO CONTROL POINT
SET EQUIPMENT ASSIGNMENT BIT IN CONTROL POINT AREA
CLEAR PPU OUTPUT REGISTER
EXIT


MTR SUBROUTINE
RJ ADVANCE CPU JOB STATUS
HAS 64 MILLISECONDS ELAPSED SINCE LAST REFERENCE ?
NO
EXIT
YES
NO
HAS ONE SECOND ELAPSED SINCE LAST SECOND ADVANCE ?
YES
ADVANCE SECOND COUNT
IS CPU IN IDLE PROGRAM ?
NO
ADD TIME INCREMENT TO CONTROL POINT CPU ACCUMULATED TIME
HAS TIME LIMIT BEEN REACHED ?
YES
NO
YES
IS A PPU AVAILABLE ?
RJ SET ERROR FLAG I
YES
NO
EXIT
MODIFY SUBROUTINE TO ADVANCE TO NEXT CONTROL POINT (MODULUS 7)
DOES CONTROL POINT HAVE A JOB NAME ?
NO
EXIT
YES
IS THE RECALL FLAG (X) SET FOR THE CONTROL POINT ?
YES
NO
SET W FLAG
CLEAR X FLAG
RJ SEARCH FOR CP PRIORITY
NO
IS PP RECALL WORD FILLED AT CONTROL POINT ?
YES
ASSIGN PPU TO PP RECALL FUNCTION
CLEAR PP RECALL WORD AT CONTROL POINT
ASSIGN PPU TO CONTROL POINT
SET PPU ASSIGNMENT BIT IN CONTROL POINT AREA
ASSIGN PPU IDLE TIME TO CONTROL POINT ZERO
UPDATE PPU STARTING TIME
RJ SEARCH FOR FREE PPU
IS THERE ANY ACTIVITY AT THE CONTROL POINT ?
YES
NO
IS THE STORAGE MOVE FLAG SET ?
YES
EXIT
NO
IS THE CONTROL POINT LISTED IN THE CPU STACK ?
YES
NO
ENTER IAJ IN PPU INPUT REGISTER
ASSIGN PPU TO CONTROL POINT
SET PPU ASSIGNMENT BIT IN CONTROL POINT AREA
ASSIGN PPU IDLE TIME TO CONTROL POINT ZERO
UPDATE PPU STARTING TIME
RJ SEARCH FOR FREE PPU
EXIT

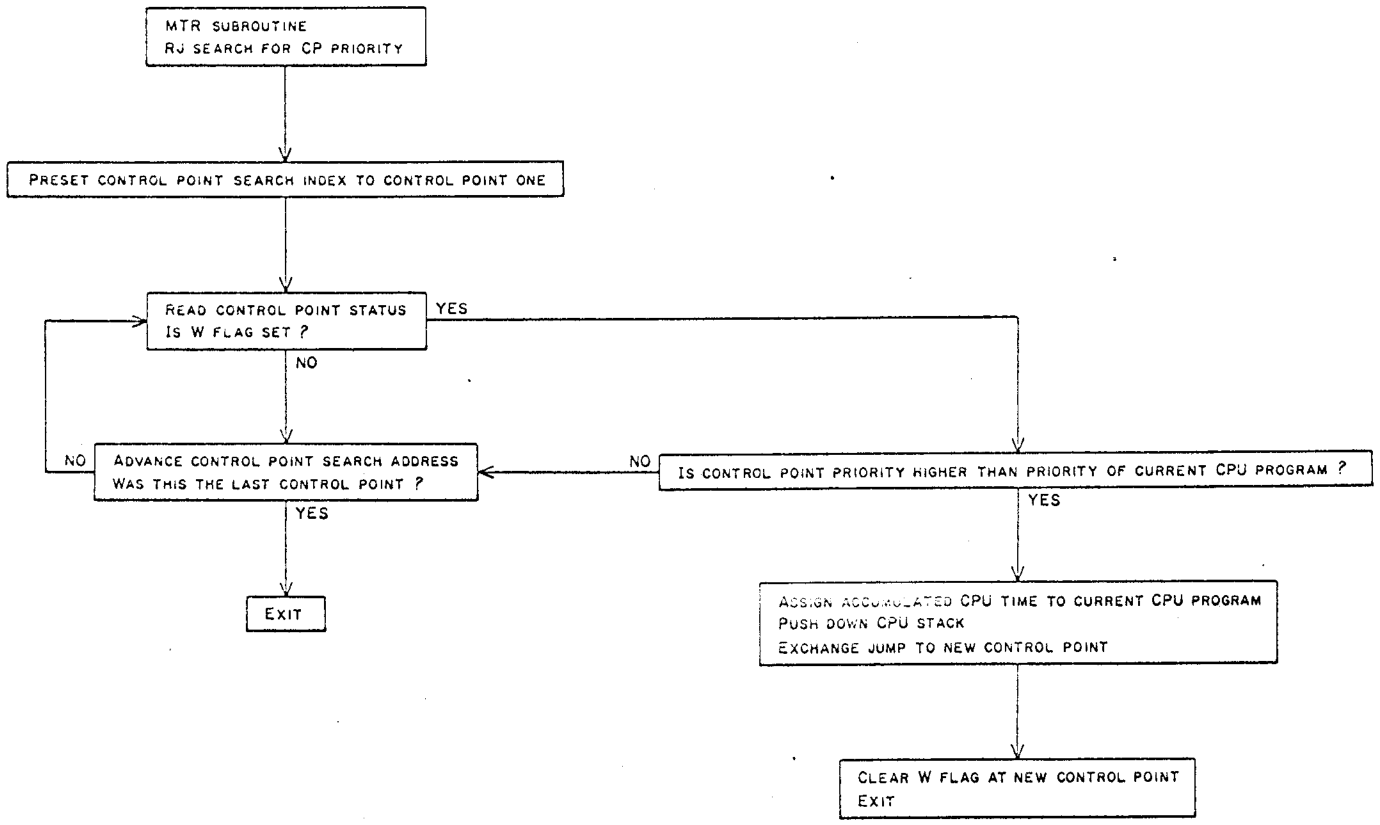

MTR SUBROUTINE
RJ PROCESS PP CALL
DOES (RA+1) CONTAIN END ?
YES
NO
EXCHANGE CURRENT CPU PROGRAM
PUSH UP CPU STACK
EXIT
DOES (RA+1) CONTAIN RCL ?
YES
NO
EXCHANGE CURRENT CPU PROGRAM
PUSH UP CPU STACK
SET X FLAG AT CONTROL POINT
CLEAR (RA+1)
EXIT
DOES (RA+1) CONTAIN ABT ?
YES
NO
RJ SET ERROR FLAG 4
EXIT
IS THERE A PPU AVAILABLE ?
NO
YES
EXIT
IS FIRST CHARACTER IN PP CALL A LETTER ?
NO
YES
RJ SET ERROR FLAG 5
EXIT
ENTER PP CALL IN PPU INPUT REGISTER
ASSIGN PPU TO CONTROL POINT
ASSIGN PPU IDLE TIME TO CONTROL POINT ZERO
UPDATE PPU STARTING TIME
RJ SEARCH FOR FREE PPU
CLEAR (RA+1)
EXIT

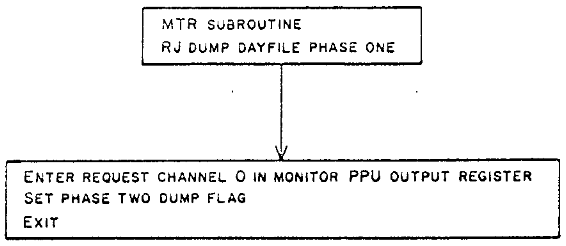


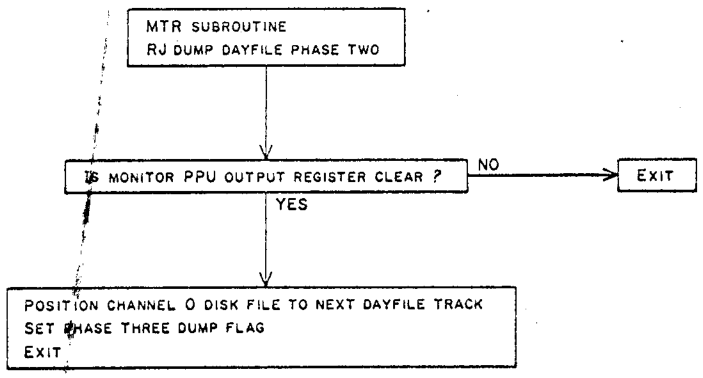
MTR SUBROUTINE
RJ SET ERROR FLAG
STORE ERROR EXIT FLAG NUMBER IN CONTROL POINT AREA
(1) = TIME LIMIT
(2) = ARITH ERROR
(3) = PPU ABORT
(4) = CPU ABORT
(5) = PP CALL ERROR
(6) = OPERATOR DROP
(7) = DISK TRACK LIMIT
IS CONTROL POINT IN CPU STACK ?
NO
CLEAR W FLAG AND X FLAG
EXIT
YES
EXCHANGE CURRENT CPU PROGRAM
PUSH UP CPU STACK
WAS THE CONTROL POINT EXCHANGED ?
YES
RJ SEARCH FOR CP PRIORITY
RJ ADVANCE CLOCK
NO
SET W FLAG FOR EXCHANGED CONTROL POINT

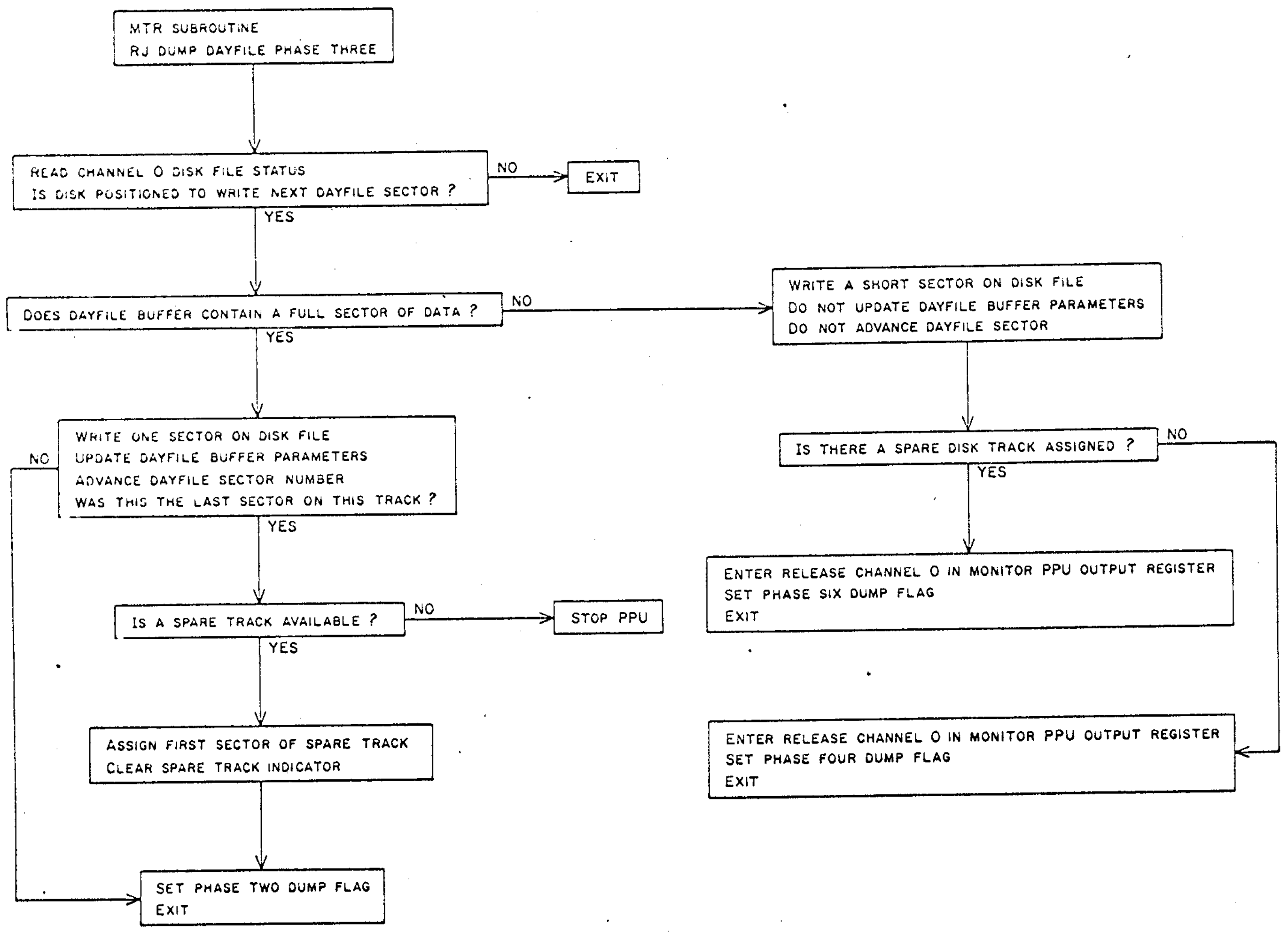


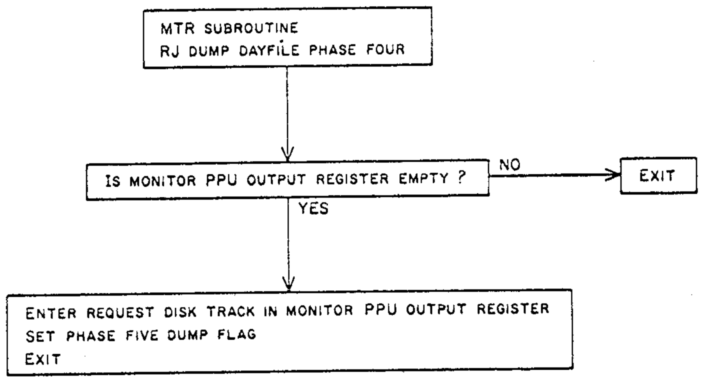
MTR SUBROUTINE
RJ SEARCH FOR FREE PPU
SEARCH PPU NUMBERS ONE THROUGH EIGHT FOR AN EMPTY INPUT REGISTER
IS THERE A FREE PPU ?
NO
CLEAR NEXT PPU INPUT REGISTER ADDRESS
EXIT
YES
STORE PPU INPUT REGISTER ADDRESS FOR NEXT ASSIGNMENT
EXIT

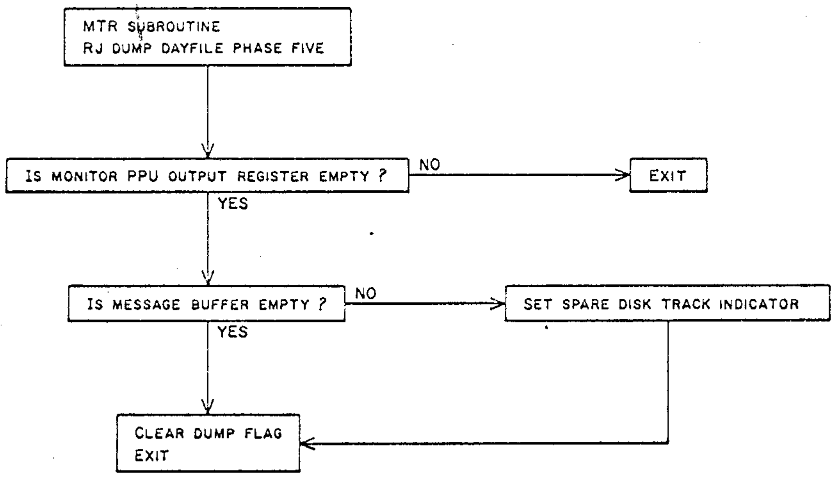

MTR SUBROUTINE
RJ SEARCH FOR CP PRIORITY
PRESET CONTROL POINT SEARCH INDEX TO CONTROL POINT ONE
READ CONTROL POINT STATUS
IS W FLAG SET ?
YES
NO
NO
ADVANCE CONTROL POINT SEARCH ADDRESS
WAS THIS THE LAST CONTROL POINT ?
YES
EXIT
NO
IS CONTROL POINT PRIORITY HIGHER THAN PRIORITY OF CURRENT CPU PROGRAM ?
YES
ASSIGN ACCUMULATED CPU TIME TO CURRENT CPU PROGRAM
PUSH DOWN CPU STACK
EXCHANGE JUMP TO NEW CONTROL POINT
CLEAR W FLAG AT NEW CONTROL POINT
EXIT

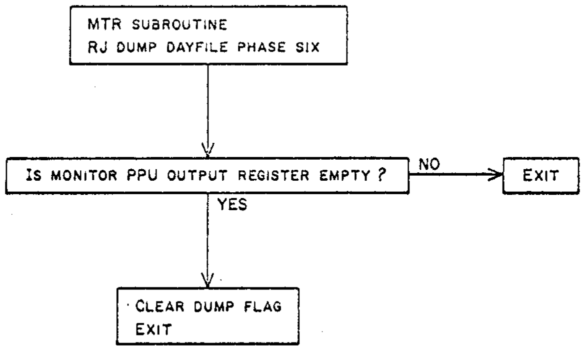

MTR SUBROUTINE
RJ DUMP DAYFILE PHASE ONE
ENTER REQUEST CHANNEL 0 IN MONITOR PPU OUTPUT REGISTER
SET PHASE TWO DUMP FLAG
EXIT


MTR SUBROUTINE
RJ DUMP DAYFILE PHASE TWO
IS MONITOR PPU OUTPUT REGISTER CLEAR ?
NO
EXIT
YES
POSITION CHANNEL 0 DISK FILE TO NEXT DAYFILE TRACK
SET PHASE THREE DUMP FLAG
EXIT


MTR SUBROUTINE
RJ DUMP DAYFILE PHASE THREE
READ CHANNEL 0 DISK FILE STATUS
IS DISK POSITIONED TO WRITE NEXT DAYFILE SECTOR ?
NO
EXIT
YES
DOES DAYFILE BUFFER CONTAIN A FULL SECTOR OF DATA ?
NO
WRITE A SHORT SECTOR ON DISK FILE
DO NOT UPDATE DAYFILE BUFFER PARAMETERS
DO NOT ADVANCE DAYFILE SECTOR
YES
WRITE ONE SECTOR ON DISK FILE
UPDATE DAYFILE BUFFER PARAMETERS
ADVANCE DAYFILE SECTOR NUMBER
WAS THIS THE LAST SECTOR ON THIS TRACK ?
NO
YES
IS A SPARE TRACK AVAILABLE ?
NO
STOP PPU
YES
ASSIGN FIRST SECTOR OF SPARE TRACK
CLEAR SPARE TRACK INDICATOR
SET PHASE TWO DUMP FLAG
EXIT
IS THERE A SPARE DISK TRACK ASSIGNED ?
NO
YES
ENTER RELEASE CHANNEL 0 IN MONITOR PPU OUTPUT REGISTER
SET PHASE SIX DUMP FLAG
EXIT
ENTER RELEASE CHANNEL 0 IN MONITOR PPU OUTPUT REGISTER
SET PHASE FOUR DUMP FLAG
EXIT


MTR SUBROUTINE
RJ DUMP DAYFILE PHASE FOUR
IS MONITOR PPU OUTPUT REGISTER EMPTY ?
NO
EXIT
YES
ENTER REQUEST DISK TRACK IN MONITOR PPU OUTPUT REGISTER
SET PHASE FIVE DUMP FLAG
EXIT


MTR SUBROUTINE
RJ DUMP DAYFILE PHASE FIVE
IS MONITOR PPU OUTPUT REGISTER EMPTY ?
NO
EXIT
YES
IS MESSAGE BUFFER EMPTY ?
NO
SET SPARE DISK TRACK INDICATOR
YES
CLEAR DUMP FLAG
EXIT


MTR SUBROUTINE
RJ DUMP DAYFILE PHASE SIX
IS MONITOR PPU OUTPUT REGISTER EMPTY ?
NO
EXIT
YES
CLEAR DUMP FLAG
EXIT

STORAGE MOVE PROGRAM

1. The exchange package for the storage move program is set by MTR as follows:

```
P = 2022
RA = 0
FL = 400000
B1 = RA + FL for requesting control point
B2 = RA + FL for control point 7
B3 = Increase or decrease
```

2. The coding for the storage move program is shown below.

```
Location    Instruction         Remarks

  2020      CØN 0
  2021      CØN 0
  2022      SB7 = 1
            EQ  B1, B2, 0       Exit if control point 7 is the requestor
  2023      LT  B3,B0,2027      Jump if decrease (B3 negative)
            NØ                  Pass
            NØ                  Pass
  2024      SA1 = B2 - B7       "Shuttle up" loop
            SA2 = A1 - B7
            BX6 = X1
            BX7 = X2
  2025      SA6 = A1 + B3
            SA7 = A2 + B3
            SB2 = B2 - 2
  2026      NE  B2,B1,2024
            JP  0               Go to stop with P = 0
  2027      SA1 = B1            "Shuttle down" loop
            SA2 = B1 + B7
            BX6 = X1
            BX7 = X2
  2030      SA6 = A1 + B3
            SA7 = A2 + B3
            SB1 = B1 + 2
  2031      NE  B2,B1,2027
            JP  0               Go to stop with P = 0
```

3. Timing for the "Shuttle up" loop of the storage move program is as follows:

Location	Instruction	Issue	Begin Execution	Result Avail.	Unit Avail.
2024	SA1 = B2 - B7	0	0	3(A) 8(X)	4
	SA2 = A1 - B7	2	3	6(A) 11(X)	7
	BX6 = X1	3	8	11	12
	BX7 = X2	12	12	15	16
2025	SA6 = A1 + B3	13	13	16	17
	SA7 = A2 + B3	15	15	18	19
	SB2 = B2 - 2	17	17	20	21
2026	NE B2,B1, 2024	19	20	28	-

Note: timing is in minor cycles - loop time approximately 2.8 microseconds for transfer of two words.

CONTROL DATA CORPORATION

Development Division - Applications

CENTRAL MEMORY RESIDENT

Chippewa Operating System

10/1/65
REV. 1

CENTRAL MEMORY RESIDENT

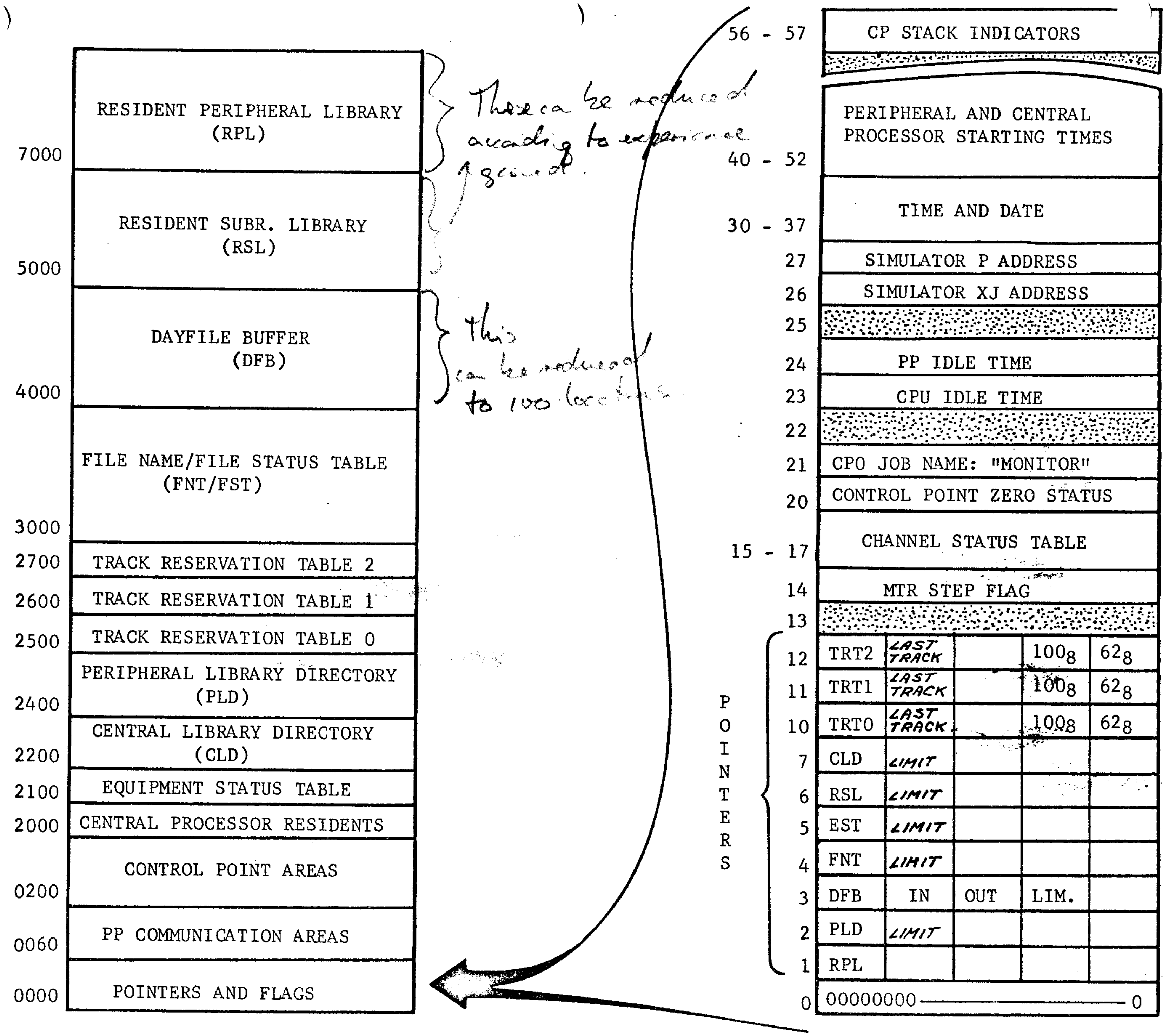
INTRODUCTION

The Chippewa Operating System uses a portion of central memory to store various types of libraries, tables, and flags: storing these tables and libraries in central memory allows them to be readily accessed by any peripheral processor. The central memory resident is illustrated in figure 1. Generally, the resident occupies central memory locations 0 - 13777_8. Resident elements between locations 0 and 2077_8 are directly addressed: resident elements above location 2077_8 are addressed via pointers contained in central memory locations 1 - 12_8. Since the major portion of the resident is relatively addressed, the size of the resident can readily be reduced or expanded to met installation requirements.

CM RESIDENT: LOCATIONS 0 - 57

Central memory location 0 always contains a full word of zeroes: a peripheral processor may read this location in order to clear a 5-byte area in its memory. Central memory locations 1 - 12_8 contain pointers to various libraries, tables, and pointers.

The Channel Status Table, illustrated in figure 2, occupies locations 15, 16, and 17. Each of the 12 data channels is represented by a byte in this table. If a data channel is not in use, the corresponding byte is cleared: if a data channel is being used by a peripheral processor program, the processor number (in display code) is entered in the byte for that channel. The channel number for a particular equipment is obtained by a peripheral processor program from the Equipment Status


RESIDENT PERIPHERAL LIBRARY (RPL)
7000
RESIDENT SUBR. LIBRARY (RSL)
5000
DAYFILE BUFFER (DFB)
4000
FILE NAME/FILE STATUS TABLE (FNT/FST)
3000
TRACK RESERVATION TABLE 2
2700
TRACK RESERVATION TABLE 1
2600
TRACK RESERVATION TABLE 0
2500
PERIPHERAL LIBRARY DIRECTORY (PLD)
2400
CENTRAL LIBRARY DIRECTORY (CLD)
2200
EQUIPMENT STATUS TABLE
2100
CENTRAL PROCESSOR RESIDENTS
2000
CONTROL POINT AREAS
0200
PP COMMUNICATION AREAS
0060
POINTERS AND FLAGS
0000
These can be reduced according to experience gained.
this can be reduced to 100 locations
56 - 57 CP STACK INDICATORS
40 - 52 PERIPHERAL AND CENTRAL PROCESSOR STARTING TIMES
30 - 37 TIME AND DATE
27 SIMULATOR P ADDRESS
26 SIMULATOR XJ ADDRESS
25
24 PP IDLE TIME
23 CPU IDLE TIME
22
21 CP0 JOB NAME: "MONITOR"
20 CONTROL POINT ZERO STATUS
15 - 17 CHANNEL STATUS TABLE
14 MTR STEP FLAG
13
12 TRT2 LAST TRACK 100₈ 62₈
11 TRT1 LAST TRACK 100₈ 62₈
10 TRT0 LAST TRACK 100₈ 62₈
7 CLD LIMIT
6 RSL LIMIT
5 EST LIMIT
4 FNT LIMIT
3 DFB IN OUT LIM.
2 PLD LIMIT
1 RPL
0 00000000 — 0
POINTERS

CENTRAL MEMORY RESIDENT (TYPICAL)

Figure 1

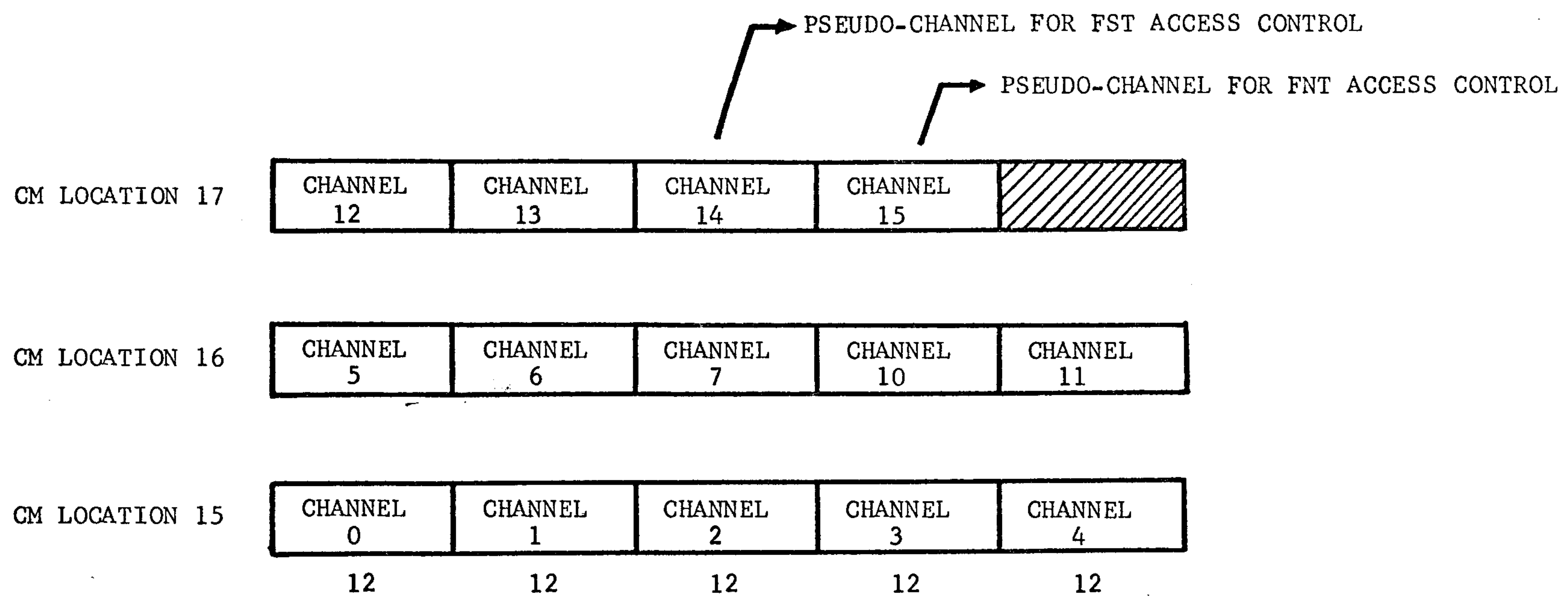
Table (EST) entry for that equipment. The program transmits its request for that channel to MTR via its resident: if the table byte corresponding to that channel is cleared, MTR enters the number of the requesting processor in that byte and notifies the requestor that the channel has been assigned. When the requesting processor completes its operation on that channel, it requests MTR to drop the channel assignment, and MTR clears the corresponding byte in the table.

The first 12 bytes in the Channel Status Table correspond to the 12 data channels: the next two bytes refer to pseudo-channels 14 and 15. These two pseudo-channels serve as an interlock to the File Name Table/File Status Table. Pseudo-channel 15 controls access to File Name Table (FNT) entries: pseudo-channel 14 controls access to File Status Table (FST) entries. Peripheral processor programs request these pseudo-channel assignments in the same manner as data channel assignments are requested. Not all accesses to the FNT/FST entries require channel reservation: the function of the interlock scheme is to prevent two (or more) processors from attempting to modify the same entry at the same time. Pseudo-channel reservations are required in the following cases:

- whenever an entry is added to the FNT/FST
- whenever a file is assigned to a control point (FNT entry modified)
- whenever the buffer status byte is initialized at the beginning of an operation (FST entry modified)

Once the appropriate pseudo-channel reservation has been acknowledged by MTR, the requesting program may proceed to perform the desired modification: upon completion, the pseudo-channel reservation should be dropped by issuing the appropriate request to MTR.

The remaining locations in this portion of the central memory resident are used by MTR for flags, indicators, and temporary storage.



- CHANNEL NOT IN USE: CORRESPONDING TABLE BYTE CONTAINS ZERO
- CHANNEL IN USE: CORRESPONDING BYTE CONTAINS USER PP NUMBER

CHANNEL STATUS TABLE

Figure 2

CM RESIDENT: LOCATIONS 60 - 177; PP COMMUNICATIONS AREAS

These central memory locations contain ten peripheral processor communication areas, one for each processor. The communication areas are illustrated in figure 3. There are eight words in each communication area:

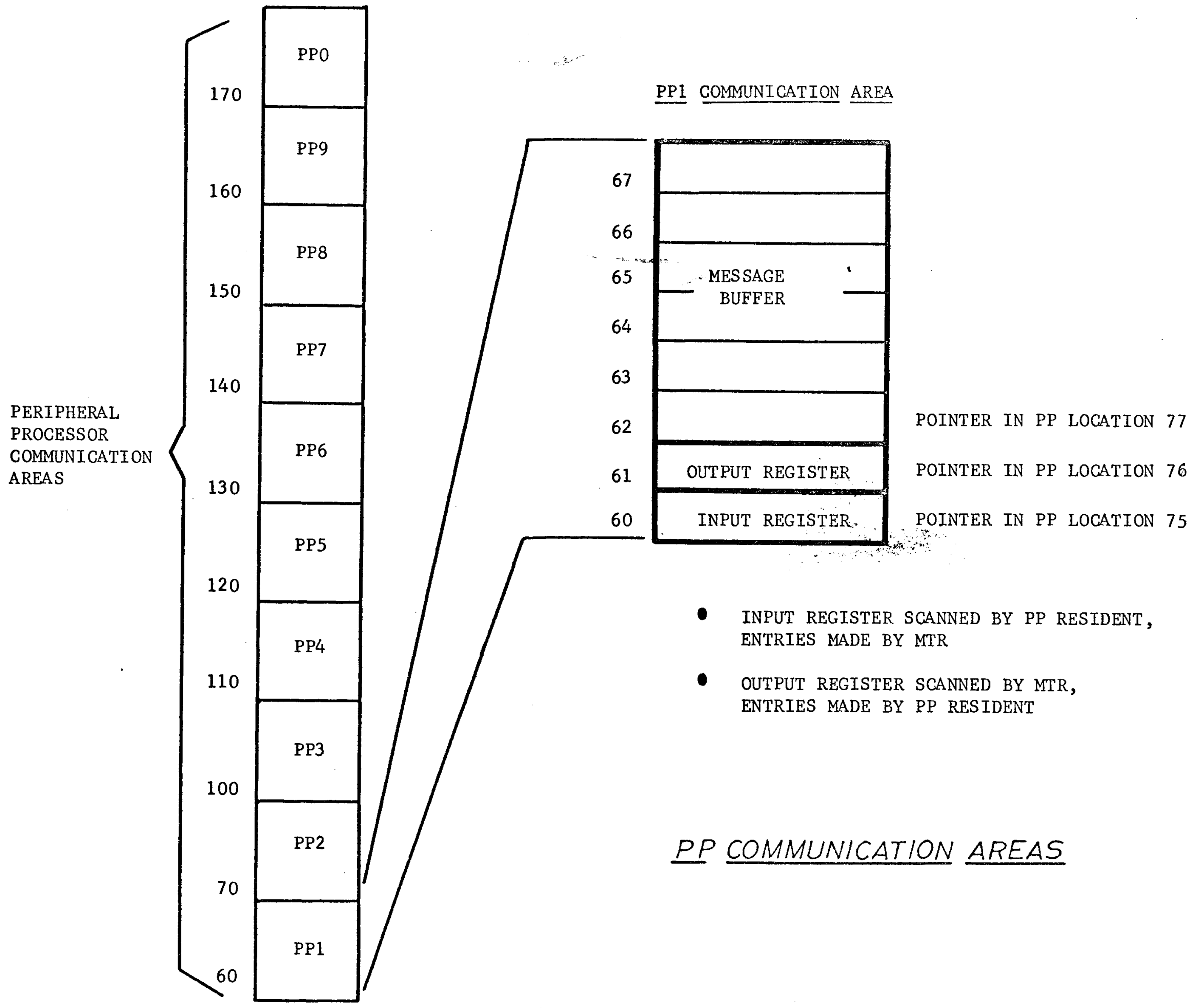
word 1	Input Register (IR)
word 2	Output Register (OR)
words 3 - 8	Message Buffer (MB)

Each peripheral processor contains pointers to its Input Register, Output Register, and Message Buffer in peripheral processor memory locations 75, 76, and 77, respectively. The communication areas are used to provide a means of communication between MTR and peripheral processor programs. When a peripheral processor is idle, its resident program continuously scans its Input Register. When MTR has a task for that processor, it sets the name of the appropriate routine in the Input Register of the idle processor, which, when it recognizes the request, loads the routine and executes it. MTR regularly scans the Output Register of each peripheral processor. When a peripheral processor program requires MTR assistance (such as, for example, reserving a data channel), it places a code in its Output Register. MTR detects the request during its scan of the output registers and processes it. When the request has been processed, MTR clears the requesting processor's Output Register: this informs the requesting processor that the request has been processed.

The six-word Message Buffer is used to pass parameters and messages between MTR and the peripheral processor resident programs.

CM RESIDENT LOCATIONS 200 - 1777; CONTROL POINT AREAS

Central memory locations 200 - 1777_8 contain seven control point areas, one for each control point. Each control point area occupies 200_8 locations. The first 20_8 words of a control point area contain the exchange jump package for the central processor program which may be associated with this control point. The next 10_8 words contain various flags, status indicators, counters,



- INPUT REGISTER SCANNED BY PP RESIDENT, ENTRIES MADE BY MTR
- OUTPUT REGISTER SCANNED BY MTR, ENTRIES MADE BY PP RESIDENT

PP COMMUNICATION AREAS

Figure 3

etc., which pertain to this particular job. Another 10_8 words are used to store the most recent console or dayfile message. The remaining 140_8 locations are used to hold the control statements for the job assigned to this control point.

CM RESIDENT: LOCATIONS 2000 - 2077; CP RESIDENT

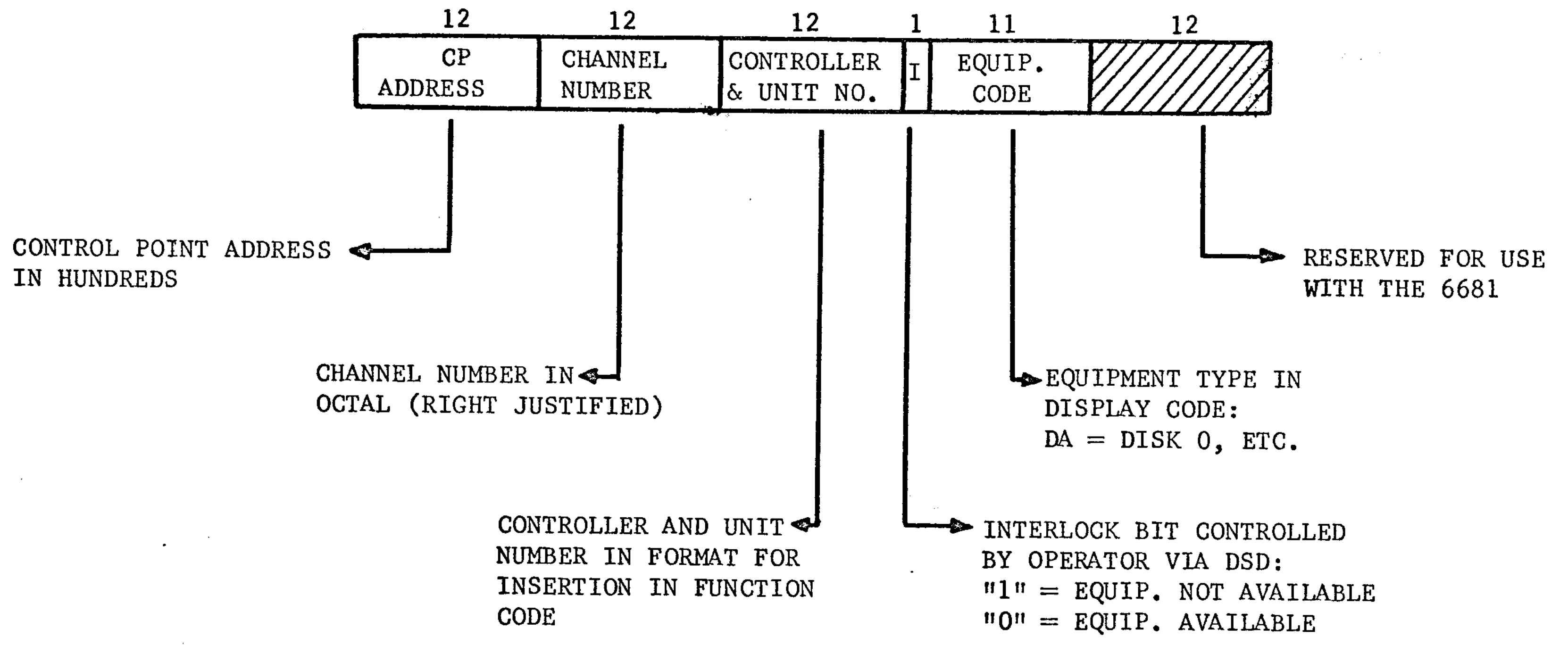
There are two resident central processor programs: a storage move program of some 24 instructions, and a two-instruction idle program. These two programs, together with their exchange jump packages, occupy locations 2000 - 2077_8 of the central memory resident.

CM RESIDENT: THE EQUIPMENT STATUS TABLE

The Equipment Status Table (EST) contains a one-word entry for each peripheral device. The table occupies 64_{10} locations: its base address is provided to the system by the EST pointer in central memory location 5. The format of the EST entry is shown in figure 4. The first (leftmost) byte contains zero if the equipment is not assigned: if the equipment is assigned to a job at a given control point, this byte contains the control point address (in hundreds). The second byte contains the channel number for this equipment, while the third byte contains the controller and unit number in the form required by the function codes for this equipment. Byte 4 contains the equipment type in display code: each type of equipment is assigned a two-letter code, as shown below.

DA	Disk 0	CR	Card Reader
DB	Disk 1	CP	Card Punch
DC	Disk 2	MT	Magnetic Tape (607)
DS	Display	WT	Magnetic Tape (626)
LP	Printer		

The 2^{11} bit in byte 4 of the EST entry is used as an operator controlled interlock for equipment availability. If this bit is zero, the equipment defined by


12
12
12
1
11
12
CP ADDRESS
CHANNEL NUMBER
CONTROLLER & UNIT NO.
I
EQUIP. CODE
CONTROL POINT ADDRESS IN HUNDREDS
CHANNEL NUMBER IN OCTAL (RIGHT JUSTIFIED)
CONTROLLER AND UNIT NUMBER IN FORMAT FOR INSERTION IN FUNCTION CODE
INTERLOCK BIT CONTROLLED BY OPERATOR VIA DSD: "1" = EQUIP. NOT AVAILABLE "0" = EQUIP. AVAILABLE
EQUIPMENT TYPE IN DISPLAY CODE: DA = DISK 0, ETC.
RESERVED FOR USE WITH THE 6681

- ONE EST ENTRY FOR EACH PERIPHERAL DEVICE
- EQUIPMENT NUMBER DEFINES LOCATION OF ENTRY IN TABLE

EST ENTRY

Figure 4

by this entry is available for assignment. If this bit is one, the equipment is not available. This bit is set or cleared by use of the DSD keyboard entries OFF and ON.* Byte 5 of the EST entry is reserved for use with equipments connected via the 6681 data channel converter.

As an example of an EST entry, suppose 607-B unit 3 on the first controller on channel six is available and not assigned to a control point: the EST entry would appear as follows:

0000	0006	2003	5524*	0000

*Display Code for MT

Within the system, equipments are identified by an equipment number. The equipment number for a given device is the relative address in the Equipment Status Table of the entry for that device.

To illustrate the use of the Equipment Status Table, consider the processing of the control statement

ASSIGN MT, INFILE

When the statement translator (2TS overlay) processes this statement, it requests MTR to assign an equipment of this type. MTR searches the EST until an entry with the equipment type (byte 4) equal to MT is found. If this equipment is not assigned (byte 1 = 0), MTR enters the control point address in byte 1 and returns the relative location of this entry in the Equipment Status Table to the statement translator. This relative location is the equipment number: the statement translator inserts this number in byte 1 of the FST entry for this file. The routines called to process this file at some later time will use this equipment number to obtain the EST entry from the table, and from the EST entry will obtain in turn the channel number and the controller and unit number.

* The interlock bit is set to "1" at load time for equipment types MT and WT.

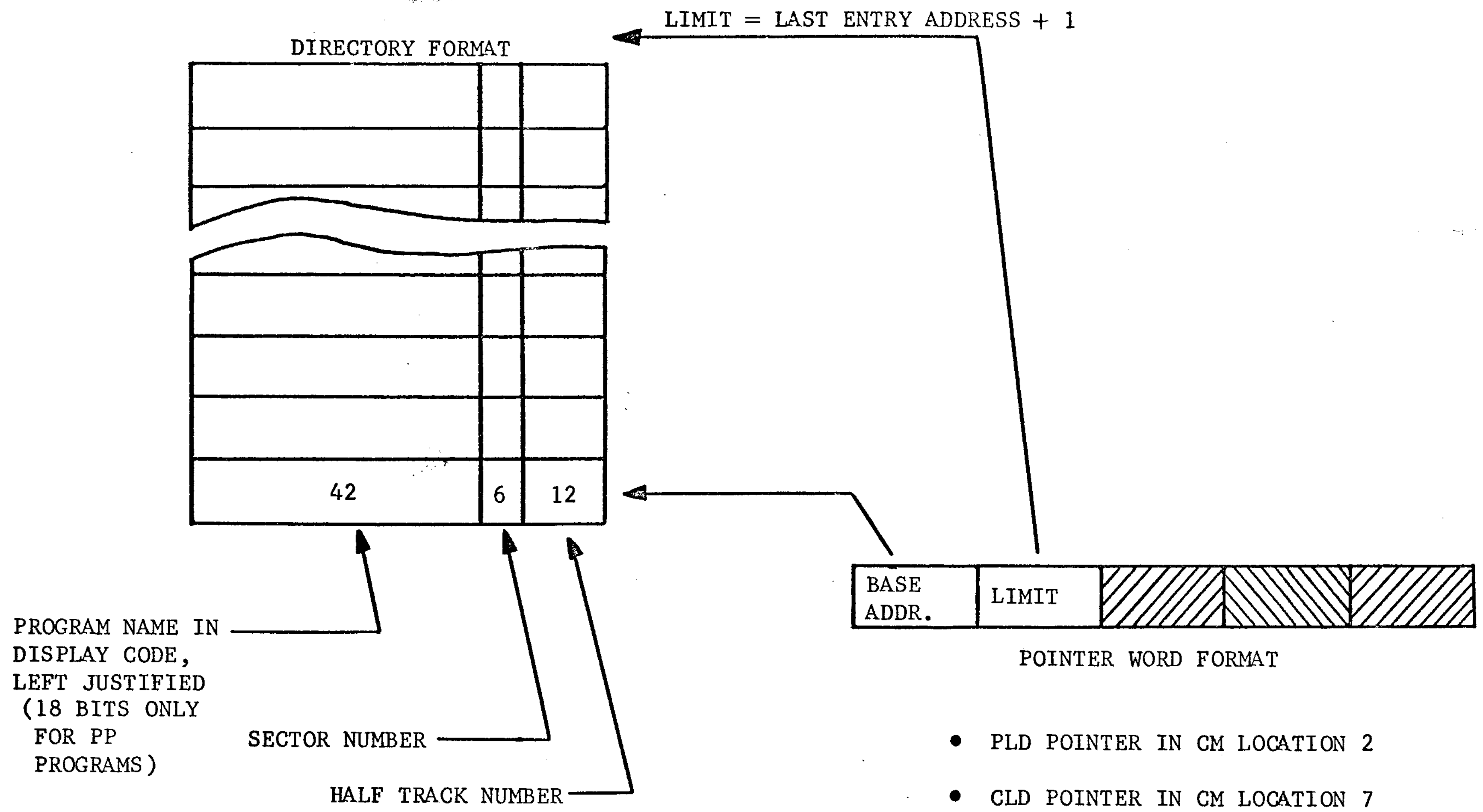
CM RESIDENT: DISK LIBRARY DIRECTORIES

The central memory resident contains two disk library directories: the Peripheral Library Directory (PLD) for the library of peripheral processor programs on the disk, and the Central Library Directory (CLD) for the library of central processor programs on the disk. The location and size of the Peripheral Library Directory is defined by the PLD pointer word in central memory location 2, and the location and size of the Central Library Directory is defined by the CLD pointer word in central memory location 7. The nominal size of the CLD is 200_8 locations, while that of the PLD is 100_8 locations.

The directory format is the same for both the PLD and the CLD, and is illustrated in figure 5. The high-order 42 bits of the directory entry contain the program name in display code, left-justified. The next six bits contain the sector number of the first disk sector for this program, while the low-order 12 bits give the half track number for this program. The program may occupy one or more sectors on the disk: the end of the program is indicated by a short ector (a sector of less than 100_8 central memory words).

Byte one (the leftmost byte) of the pointer word supplies the base address of the directory: byte two supplies the directory limit address, which is the address + 1 of the last directory entry. When the directory is being searched, exit from the search occurs (in the non-hit case) when the limit is reached or, in some cases, when an entry with byte one equal to zero is detected. If it is desired to delete an entry temporarily for some reason, then, the entry should be set to something other than zero.

The PLD is searched by the peripheral processor resident programs and by certain of the transient programs and overlays. When a peripheral processor resident is directed by MTR to load and execute a peripheral processor program, it first searches the central memory Resident Peripheral Library (RPL) for that program: if the program is not found in the resident library, the peripheral


LIMIT = LAST ENTRY ADDRESS + 1
DIRECTORY FORMAT
42
6
12
PROGRAM NAME IN DISPLAY CODE, LEFT JUSTIFIED (18 BITS ONLY FOR PP PROGRAMS)
SECTOR NUMBER
HALF TRACK NUMBER
BASE ADDR.
LIMIT
POINTER WORD FORMAT

- PLD POINTER IN CM LOCATION 2
- CLD POINTER IN CM LOCATION 7

<u>DISK</u> <u>LIBRARY</u> <u>DIRECTORIES</u> - <u>PLD</u> & <u>CLD</u>

Figure 5

processor resident proceeds to search PLD. It is thus possible to reduce the size of the resident library by placing some of the peripheral processor programs on the disk. Some peripheral processor transient programs follow the same procedure in loading their overlays: others, however, search only the resident library. It is therefore not possible to move all peripheral processor programs from the resident library to the disk library.

The CLD is searched by two programs: the Central Library Loader (CLL) and the control statement translator (2TS). The function of the Central Library Loader is to load overlays into central memory when called by a central processor program. CLL first searches the central memory Resident Subroutine Library (RSL) for the requested overlay: if not found, the CLD is then searched. If the overlay is not found in either the resident library or the disk library, CLL then searches the File Name Table (FNT) for a file with this name. The control statement translator, 2TS, searches CLD when processing program cards. When the statement translator finds a program card, it first searches the File Name Table for a file with that name: if not found, CLD is searched next. If the program is not found in either the FNT or the CLD, a search is made of the Peripheral Library Directory.

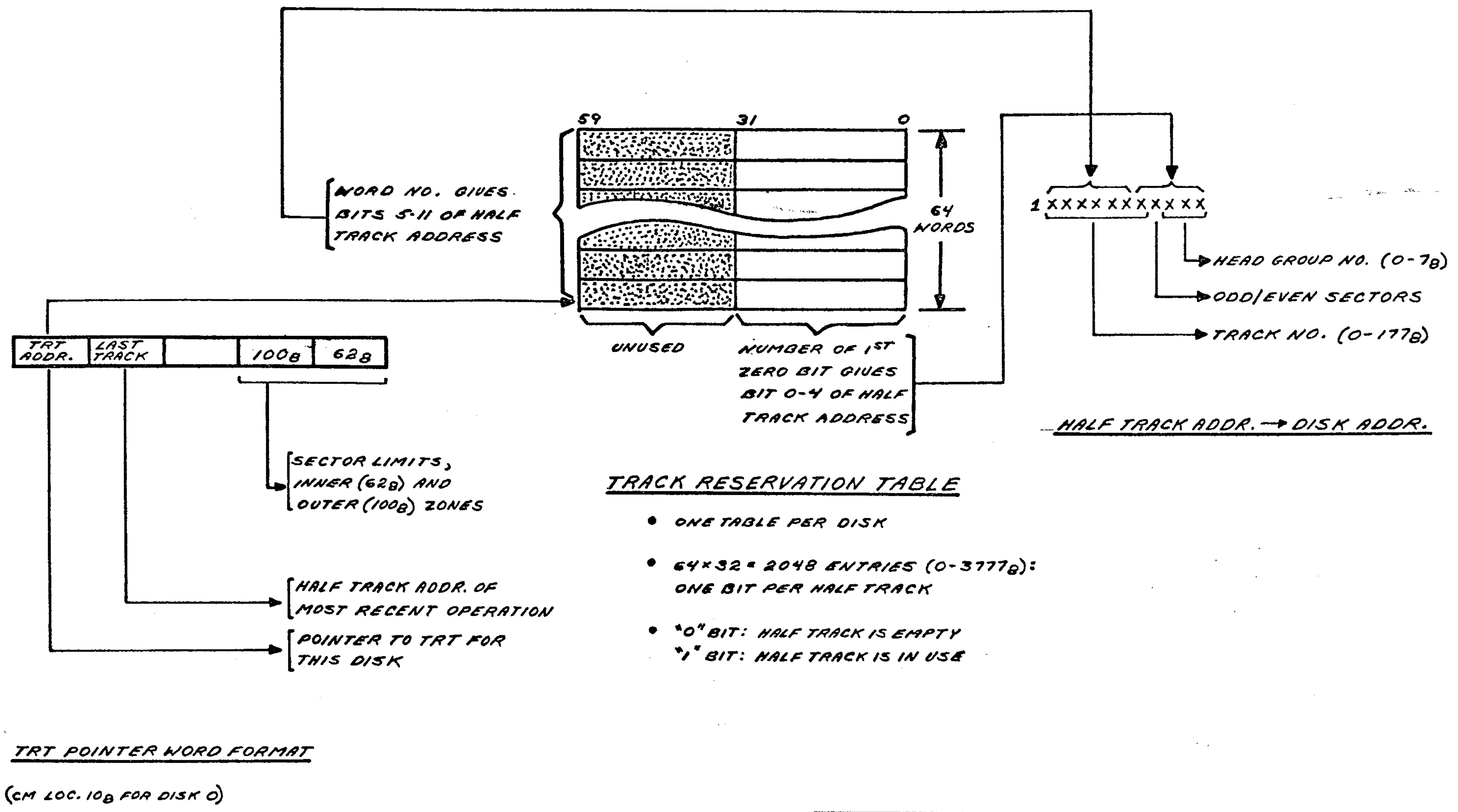
CM RESIDENT: THE TRACK RESERVATION TABLES

The Chippewa Operating System is designed to permit the use of up to three 6603 diskfiles with the system: these diskfiles are identified as Disk 0, Disk 1, and Disk 2. Both the system and the user may store data on Disk 0, while Disk 1 and Disk 2 are reserved soley for the user. The utilization of space on a given diskfile is recorded in a table called the Track Reservation Table (TRT). There is a Track Reservation Table for Disk 0, for Disk 1, and for Disk 2: the locations of these tables are given by the TRT pointer words in central memory locations 10, 11, and 12_8, respectively. The tables are identical and are identically manipulated.

Since a single peripheral processor cannot maintain a continuous flow of data between a diskfile and central memory, the Chippewa Operating System employs an interlacing scheme in which data is recorded on only the odd-numbered sectors or only the even-numbered sectors in a track during a revolution over that track. From a hardware standpoint, a track contains either 128 or 100 sectors, depending upon whether the track lies in the two outer zones or the two inner zones. The Chippewa Operating System considers a physical track to be composed of two half tracks; one consisting of the odd-numbered sectors on the physical track, the other consisting of the even-numbered sectors on that track. A half track, then, contains either 64_{10} or 50_{10} sectors, depending upon its location. Since a diskfile contains 128 tracks at each of 8 head group selections (1024 tracks), a diskfile contains 2048 (3777_8) half tracks. A given half track is never used for records of more than a single file: should a file consist of only a single sector, an entire half track would be reserved for that file.

The Track Reservation Table is illustrated in figure 6. The table is made up of 64 words: only the rightmost 32 bits in a word are used. The table thus contains 2048 bits, one bit for each half track on a diskfile. If a bit is zero, the corresponding half track is not in use. If a bit is one, the corresponding half track has been assigned. Should a section of the diskfile become defective, the corresponding bit or bits in the TRT may be permanently set to one (by modifying the library tape) in order to avoid accessing the defective areas.

Half track assignments are handled by MTR. When MTR receives a track request from a peripheral processor program, it searches the TRT until the first zero bit is found. The coordinates of this bit are then assembled to form a half track number: the bit position in the word ($0 - 37_8$) comprises the low


59
31
0
64 WORDS
WORD NO. GIVES BITS 5-11 OF HALF TRACK ADDRESS
UNUSED
NUMBER OF 1ST ZERO BIT GIVES BIT 0-4 OF HALF TRACK ADDRESS
1XXXXXXXXXXX
HEAD GROUP NO. (0-7₈)
ODD/EVEN SECTORS
TRACK NO. (0-177₈)
HALF TRACK ADDR. → DISK ADDR.
TRT ADDR.
LAST TRACK
100₈
62₈
SECTOR LIMITS, INNER (62₈) AND OUTER (100₈) ZONES
HALF TRACK ADDR. OF MOST RECENT OPERATION
POINTER TO TRT FOR THIS DISK
TRT POINTER WORD FORMAT
(CM LOC. 10₈ FOR DISK 0)

TRACK RESERVATION TABLE

- ONE TABLE PER DISK
- $64 \times 32 = 2048$ ENTRIES (0-3777₈): ONE BIT PER HALF TRACK
- "0" BIT: HALF TRACK IS EMPTY
  "1" BIT: HALF TRACK IS IN USE

TRACK RESERVATION & ADDRESSING
CHIPPEWA OPERATING SYSTEM

Figure 6

CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS TRAINING	MACH TYPE 6000	PROJECT NO	DOCUMENT ABSTRACT	REV / APPROVED / DATE / REV / APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE TRACK RES. & ADDR.		PROJECT MGR		
FLOWCHART ☐		PAGE OF	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO		
OTHER ☒	DRAWN BY MHM	DATE 9/13	TASK NAME		

order five bits, while the word position in the table (0 - 77_8) provides the next six bits. MTR returns this half track number to the requesting processor and sets the bit in the table to one. Dropping of track assignments takes place in a reverse fashion. To drop a half track assignment, the requesting processor sends MTR the number of the half track to be dropped. MTR disassembles the half track number into table coordinates and clears the bit in the table.

The low-order three bits of the half track number specify the head group; the next bit (2^3) specifies whether this half track uses the odd-numbered sectors ($2^3 = 1$) or the even-numbered sectors ($2^3 = 0$); the next seven bits specify the track number. Since the lower portion of the half track number comes from the bit position in a table word, the order of selection is such that the even-numbered half tracks at head groups 0 - 7 are selected first, and the odd-numbered half tracks at head groups 0 - 7. are selected next. Only when all the half tracks at a given physical position of the heads have been assigned is a half track number selected which requires repositioning. Thus, the layout of the table eliminates unnecessary repositioning.

Byte 1 of the TRT pointer word contains the base address of the table. Byte 2 contains the last half track used by this diskfile, and thus reflects the current physical position of the heads and the currently selected head group. Whenever a disk operation is initiated, the half track number for the operation is compared with the contents of byte 2, and repositioning or head group selection performed only if necessary.. This byte is updated at the end of each disk operation.

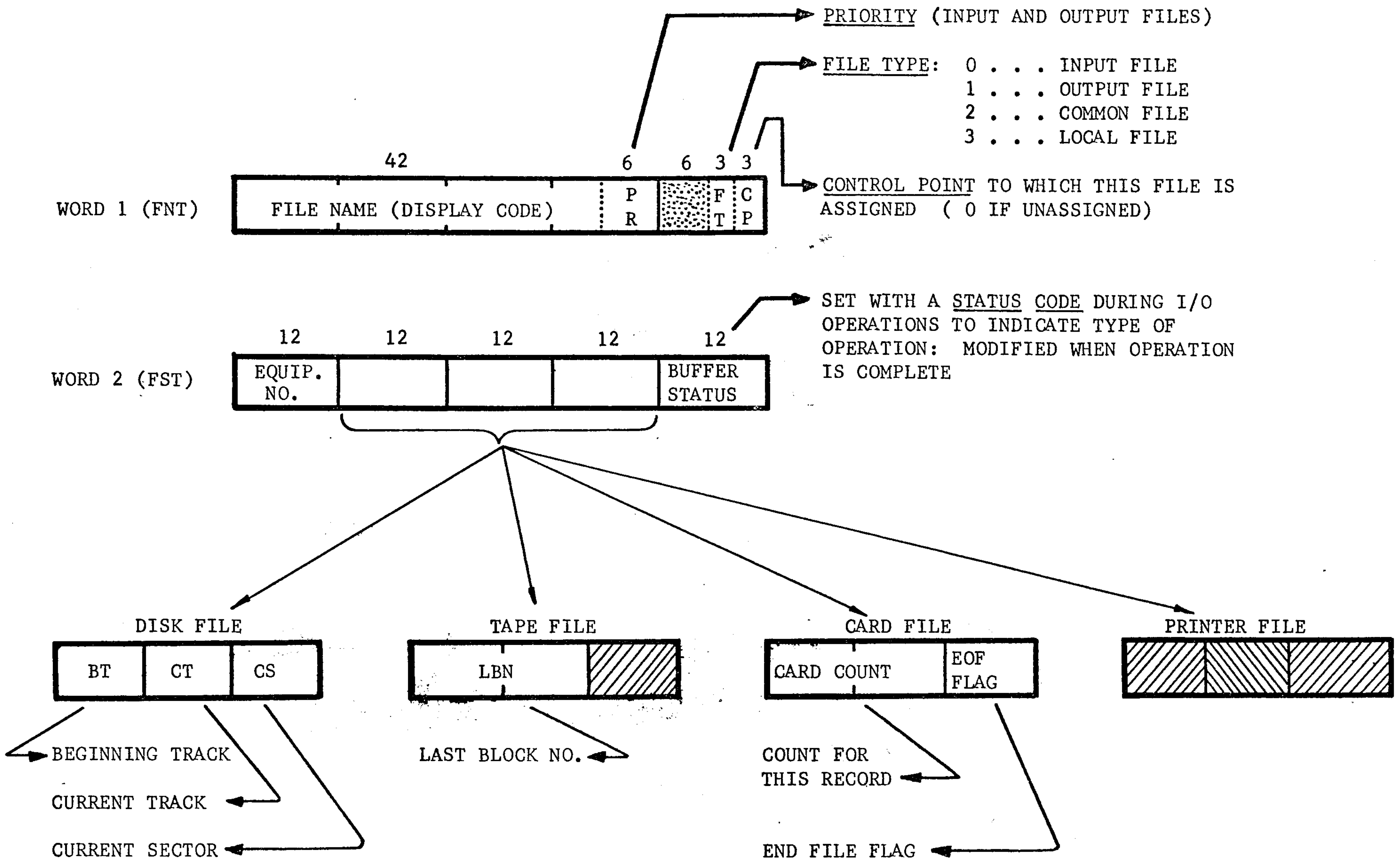
Bytes 4 and 5 of the pointer word always contain the constants 100_8 and 64_8, respectively. These constants are the sector limits for tracks in the outer zones (hig-order bit of the head group number = 0) and the inner zones (high-order bit of the head group number = 1). The disk routines compare the sector number for the current operation with the appropriate one of these two constants in order to determine when the end of a half track has been reached.

CM RESIDENT: FILE NAME TABLE/FILE STATUS TABLE

The various types of files currently being controlled by the system are defined by entries in the File Name Table/File Status Table. These two tables are interleaved such that the File Name Table (FNT) entry and the File Status Table (FST) entry for a specific file occupy successive central memory locations. The base address and limit address (last entry address + 1) of the FNT/FST are contained in bytes 1 and 2, respectively, of the FNT/FST pointer word in central memory location 4. The nominal size of the FNT/FST is 1000_8 central memory words, permitting up to 256_{10} files to be defined at any one time.

The format of the FNT/FST entry is shown in figure 7. The FNT entry contains the file name in display code in the leftmost 42 bits of the word. The next six bits contain the priority, if any, associated with this file. The low-order six bits of the entry contain a File Type indicator (3 bits) and the Control Point Number (3 bits) to which this file is assigned: if unassigned, the Control Point Number is zero. The File Type indicator may take on the values 0, 1, 2, or 3, indicating that this file is, respectively, an INPUT file, an OUTPUT file, a COMMON file, or a LOCAL file.

When a job enters the system (either from a card reader or from a tape unit), the File Type indicator is set to 0 (INPUT file), the file name is set to the job name as given on the job card, and the priority is entered from the job card. Unassigned (Control Point Number = 0) files on the disk of type INPUT, then, constitute a job stack, and the FNT serves as a job table. When the system is ready to bring in the next job from the disk, it searches the FNT table for the highest priority unassigned INPUT file. When this file is assigned to a control point, the number of this control point is set in the low-order three bits of the FNT entry, and the File Type indicator is set to LOCAL. The job name (file name of an INPUT file) and priority are placed in the control point



FNT/FST ENTRY

Figure 7

area. The file name is then set to INPUT, and the priority field cleared. Files may be initiated by a job: if a CIO call specifies a file name which is not contained in the FNT, a new entry is added to the FNT which contains the specified file name, has the file type LOCAL, and is assigned to the disk.

Data to be printed at the end of a job is written by the job to a LOCAL file on the disk with the file name OUTPUT. At the end of the job, all LOCAL files except the LOCAL file named OUTPUT are dropped. The system routine which closes out a job (1AJ) changes the name of this file from OUTPUT to the job name, changes the type from LOCAL to OUTPUT, and enters the priority from the control point area. Effectively, then, files on the disk of type OUTPUT constitute a job stack for the print package. The print package selects the next file to be printed by searching the FNT for the file of type OUTPUT with the highest priority.

If it is desired to retain a file at the end of a job for use with some subsequent job, the file must be declared type COMMON by means of a COMMON control card. At the end of a job, a file of type COMMON will not be dropped: the control point assignment will simply be cleared. COMMON type files may be dropped when desired by use of a RELEASE control card.

The format of the FST entry varies, depending upon the type of equipment assigned for the file. Files are assigned to disk 0 unless another equipment is specified by means of an ASSIGN control card. Byte 1 of the FST entry always contains the equipment number, which gives the relative location in the Equipment Status Table of the equipment type for this file. This byte is either set by the statement translator (2TS) when an ASSIGN control card is processed or, if no ASSIGN card appears for this file, is set to correspond to disk 0 when the first reference to this file is made (by the 2BP overlay). Byte 5 of the FST entry contains the buffer status: this is obtained from the CIO call and inserted in the FST entry (by 2BP) for use by the various I/O routines: this status indicates the type of operation to be performed (read, write, rewind, etc.). If the 2^0 bit of this byte is zero, an operation involving this file is in process: if the 2^0 bit is one, this file is not reserved.

Bytes 2, 3, and 4 of the FST entry vary according to the equipment type. In the case of the printer, these bytes are not used. In the case of the card reader, bytes 2 and 3 are used to maintain a count of the number of cards processed in a record, and byte 4 is set when an end-of-file card (6-7-8-9 card) is processed. For tape files, bytes 2 and 3 are used to maintain a count of the number of blocks recorded for this file.

For disk files, byte 2 holds the beginning half track number for the file, byte 3 holds the current half track number (i.e., the half track on which the most recent operation involving this file took place) for this file, and byte 4 holds the current sector number. The next read or write to this file will be to the sector supplied by byte 4 on the half track supplied by byte 3. When housekeeping for this read or write is performed, the current half track number in byte 3 will be compared with the last half track byte in the TRT pointer word to determine if repositioning and/or head group selection is necessary.

When a file assigned to the disk is rewound, the current half track byte is set equal to the beginning half track byte and the current sector number is set to zero. Disk files which are COMMON type files are not rewound at the end of the job.

CM RESIDENT: DAYFILE BUFFER

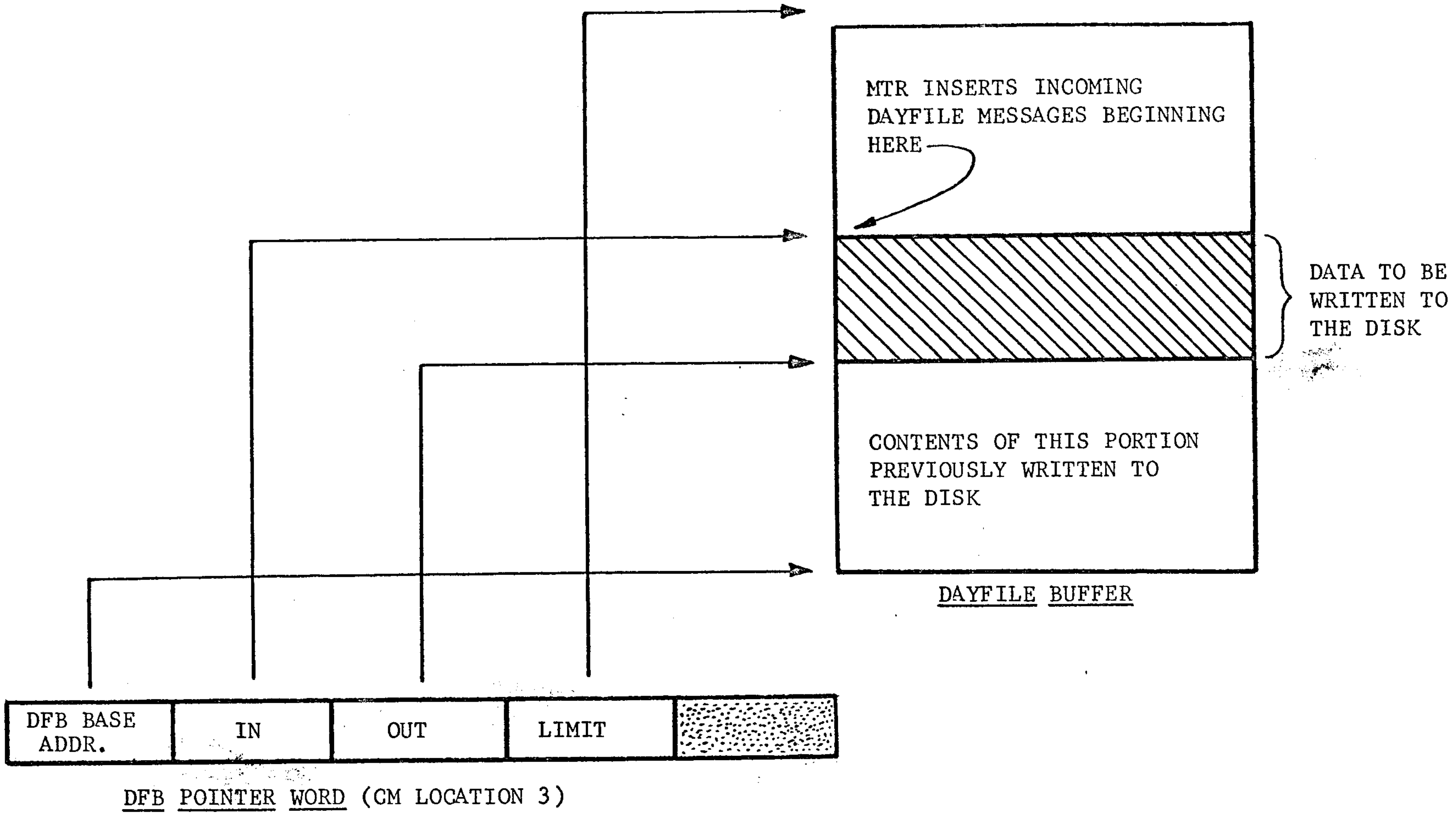
The dayfile contains a variety of information concerning the status and progress of jobs in the system, such as start and finish times, peripheral and central processor usage, diagnostics, etc. Dayfile messages may be issued by any of the system peripheral processor programs and may also be issued by a user's central processor program via the MSG routine.

The dayfile is maintained on the disk: dayfile entries are buffered through a portion of the central memory resident area called the Dayfile Buffer. The base address and limit address (last word address + 1) of this buffer are supplied by the DFB pointer word in central memory location 3. The nominal size of the Dayfile Buffer is 1000_8 locations.

The Dayfile Buffer and its pointer word are illustrated in figure 8. The first four bytes in the pointer word contain the base address, IN pointer, OUT pointer, and the limit address: these quantities are analogous to the FIRST, IN, OUT, and LIMIT pointers used in CIO, and the Dayfile buffer is handled in much the same manner as a CIO processed buffer.

When a peripheral processor program wishes to insert a message in the dayfile, it places the message in its Message Buffer and issues the appropriate request to MTR. MTR copies the message from the Message Buffer into the Last Dayfile Message area in the control point area of the job to which the requesting processor is assigned. MTR then enters the message, together with the job name and the time, in the Dayfile Buffer beginning at the location specified by the IN pointer byte of the DFB pointer word.

Whenever MTR enters a message in the Dayfile Buffer, a test is made to determine if the buffer contains a full sector of data. (It is possible that the message just entered resulted in the buffer's containing slightly more than a full sector.) If it does, a flag is set which causes the full sector and the partial sector, if any, to be dumped to the disk. Dumping is done by MTR in six phases in order to avoid tying up MTR for an extended period of time. After each phase has been executed, MTR returns to its master loop to perform any functions required by other peripheral processors or the central processor. As data is transferred from the buffer to the disk, the OUT pointer is adjusted accordingly. Insofar as maintaining the dayfile on the disk is concerned, only slightly more than 100_8 words are required. The nominal size of the Dayfile Buffer is set at 1000_8 words to permit DSD to display as much dayfile activity as possible. Thus, if the buffer size is reduced to about 110_8 words, the sole effect is to reduce the size of the dayfile console display.


MTR INSERTS INCOMING DAYFILE MESSAGES BEGINNING HERE
DATA TO BE WRITTEN TO THE DISK
CONTENTS OF THIS PORTION PREVIOUSLY WRITTEN TO THE DISK
DAYFILE BUFFER
DFB BASE ADDR.
IN
OUT
LIMIT
DFB POINTER WORD (CM LOCATION 3)

DAYFILE BUFFER

Figure 8

CM RESIDENT: RESIDENT LIBRARIES

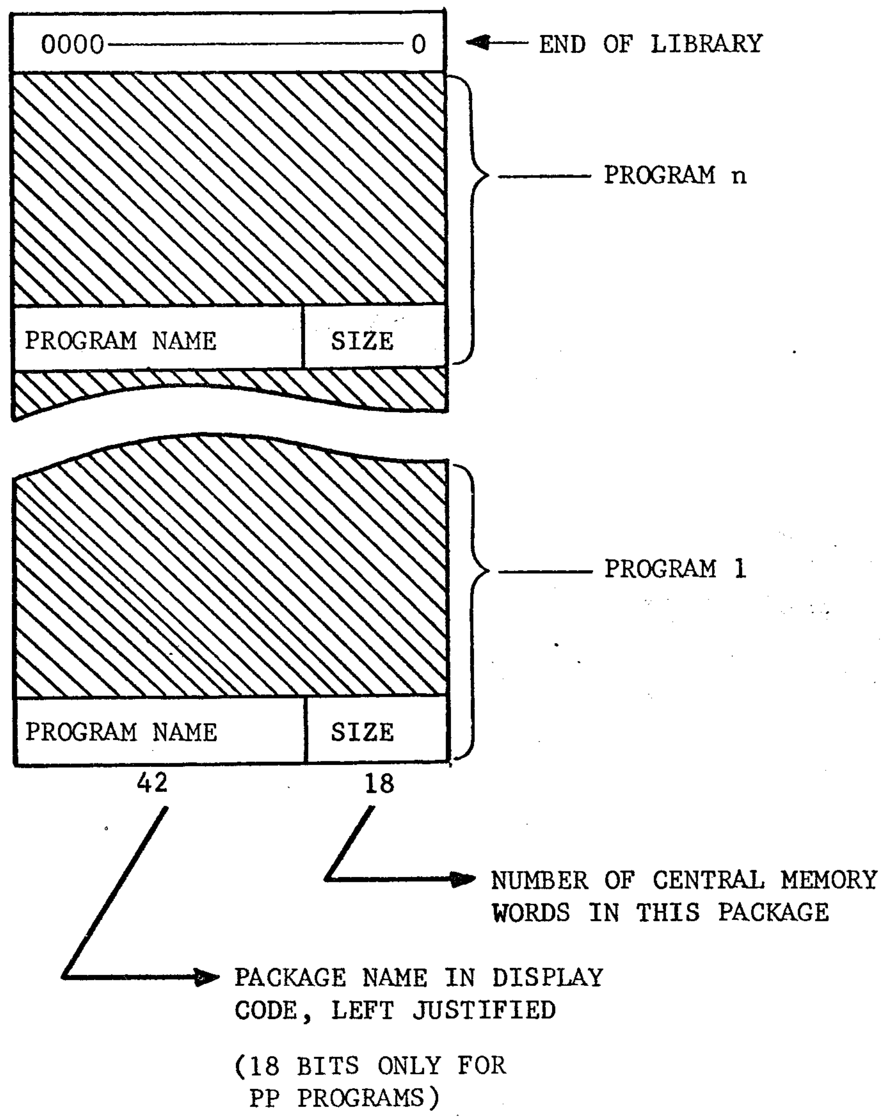
The central memory resident contains two libraries: the Resident Peripheral Library (RPL), which contains peripheral processor programs such as 1AJ, 2RD, and CIO, and the Resident Subroutine Library, which contains central programs such as ACOS and TIME. The starting addresses of the Resident Peripheral Library and the Resident Subroutine Library are defined by the RPL and RSL pointer words in central memory locations 1 and 6, respectively.

The library format is the same for both RSL and RPL, and is illustrated in figure 9. The first word of a library program contains the name in display code in the leftmost 42 bits. The low-order 18 bits of the word contain the package size in central memory words. In searching the library, the searching routine reads the program name of the first package and tests to see if this is the desired routine: if it is not, the size of the routine is added to the base address to form the address of the first word of the next program. It is important then, that the size value be correct. The end of each library is indicated by a word of zeroes.

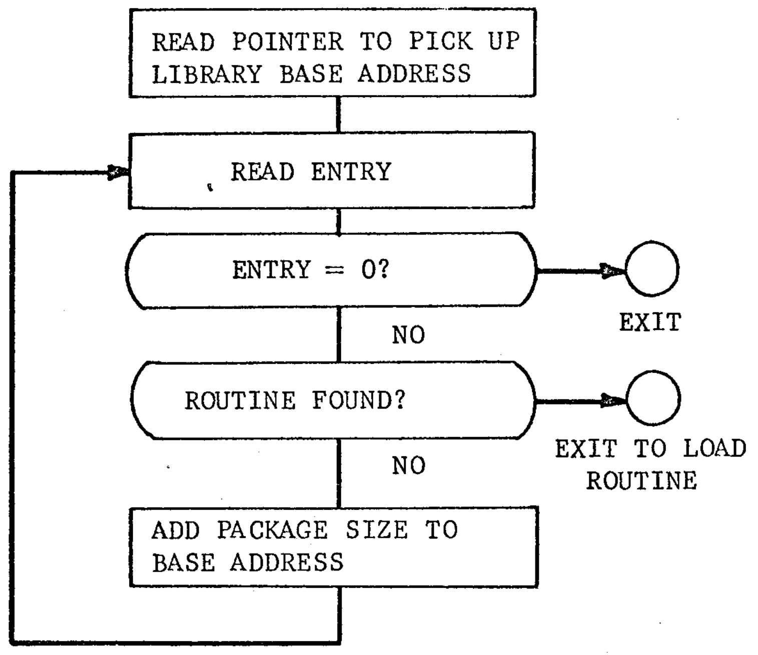
The RPL is searched by the peripheral processor resident programs and by most of the transient programs. If the peripheral processor resident does not find a routine in the RPL, it proceeds to search the PLD. Transient programs such as 1AJ, 1BJ, CIO, etc., are loaded into peripheral processor memory beginning at location 773_8: the first executable instruction, which is in the first byte of the second central memory word in the package, is thus at location 1000_8. Overlay programs, such as 2RD, 2BP, etc., are loaded into peripheral processor memory beginning at location 1773_8. Since these programs are entered via a return jump (to location 2001_8), the first executable instruction is at location 2002_8, with location 2000_8 containing the LJM order code for the exit point.

The RSL is searched by the CLL (Central Library Loader) routine. Programs in the RSL are assembled to execute beginning at location 0, and so must be

LIBRARY FORMAT


0000
0
END OF LIBRARY
PROGRAM n
PROGRAM NAME
SIZE
PROGRAM 1
PROGRAM NAME
SIZE
42
18
NUMBER OF CENTRAL MEMORY WORDS IN THIS PACKAGE
PACKAGE NAME IN DISPLAY CODE, LEFT JUSTIFIED
(18 BITS ONLY FOR PP PROGRAMS)

RPL/RSL SEARCH


READ POINTER TO PICK UP LIBRARY BASE ADDRESS
READ ENTRY
ENTRY = 0?
EXIT
NO
ROUTINE FOUND?
EXIT TO LOAD ROUTINE
NO
ADD PACKAGE SIZE TO BASE ADDRESS

RESIDENT LIBRARIES

Figure 9

relocated by the user to the desired location.

The size of the RSL and RPL can be reduced by transferring programs to the disk libraries. Certain programs, however, may not be transferred, since not all peripheral processor transient programs search the PLD if a routine is not found in the RPL. For example, system programs such as 1AJ, 1BJ, and 1DJ search only the RPL for their overlays (the transient programs themselves, however, could be transferred to the disk library). Other transient programs, such as CIO, search both RPL and PLD for overlays.