GY20-0590-1

Type III Class A Program

IBM

Control Program-67/Cambridge Monitor System (CP-67/CMS) Version 3.1
Program Number 360D-05.2.005
CP-67 Program Logic Manual

This publication describes the internal logic of the CP-67 (Control Program-67) system. The system consists of a Control Program that creates a multiprogramming, time-sharing environment by providing virtual machines for users to run their own operating systems concurrently with other users. This manual is directed to personnel who will be responsible for the maintenance and modification of CP-67.

PREFACE

The following documents are referenced in the CP-67 Program Logic Manual:

Functional Characteristics and Principles of Operation

IBM System/360 Model 67: Functional Characteristics, A27-2719

IBM System/360 Principles of Operation, A22-6521

Assembler

IBM OS/360: Assembler Language, C28-6514

IBM OS/360: Assembler (F) Programmer's Guide, C26-3756

The following documents provide further information on CP-67:

CP-67/CMS User's Guide, GH20-0859

CP-67 Operator's Guide, GH20-0856

CP-67/CMS Installation Guide, GH20-0857

CP-67/CMS System Description Manual, GH20-0802

CP-67 Program Logic Manual, GY20-0590

CMS Program Logic Manual, GY20-0591

CMS SCRIPT User's Manual, GH20-0860

CP-67/CMS Hardware Maintainability Guide, GH20-0858

CP-67: Operating Systems in a Virtual Machine, GH20-1029

Second Edition (October 1971)

This edition is a major revision of, and obsoletes, GH20-0590-0.

This Type III Program performs functions that may be fundamental to the operation and maintenance of a system.

It has not been subjected to formal test by IBM.

Until the program is reclassified, IBM will provide for it: (a) Central Programming Service, including design error correction and automatic distribution of corrections; and (b) FE Programming Service, including design error verification, APAR documentation and submission, and application of Program Temporary Fixes or development of an emergency bypass when required. IBM does not guarantee service results or represent or warrant that all errors will be corrected.

You are expected to make the final evaluation as to the usefulness of this program in your own environment.

THE FOREGOING IS IN LIEU OF ALL WARRANTIES EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

This edition applies to Version 3, Modification Level 1, of Control Program-67/Cambridge Monitor System (360D-05.2.005) and to all subsequent versions and modifications until otherwise indicated in new editions or Technical Newsletters.

Changes are continually made to the information herein. Therefore, before using this publication, consult the latest System/360 SRL Newsletter (GN20-0360) for the editions that are applicable and current.

Copies of this and other IBM publications can be obtained through IBM branch offices.

A form has been provided at the back of this publication for readers' comments. If this form has been removed, address comments to: IBM Corporation, Technical Publications Department, 1133 Westchester Avenue, White Plains, New York 10604.

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

Figure

Tables

SECTION 1: INTRODUCTION TO CP-67

CP-67 is a Control Program designed for execution on an IBM System/360 Model 67. Its objective is to create an environment in which many users can simultaneously perform work and in which each user can perform his own work under the supervision of the programming system of his choice. It achieves its objective by generating a "virtual computer" for each user and by sharing the resources of the real computer (CPU time, main storage, etc.) among the virtual computers for all users that are concurrently logged into the system.

When a user identifies himself from a terminal, the Control Program "creates" for his personal use a virtual computer from a predefined configuration. (Before the system becomes available to users, the systems administrator defines the configuration of each user's virtual machine. He may define different configurations for different users.) To the user, his virtual computer appears real and he uses it as if it were. The Control Program also provides, as part of the virtual computer, commands that parallel the functions of the buttons and switches on an operator's console. The user issues these commands from his terminal, and, thus, the terminal becomes a pseudo-console for his virtual machine.

After the Control Program has created the virtual computer, the user equips it with the programming system that gives him the desired functional capabilities. He does this by issuing a command from his terminal. CP-67 is designed so that the user can run the programming system (for example, Operating System/360) of his choice on his virtual computer. The user who desires a terminal-oriented, conversational programming system that allows him to directly monitor his work will choose CMS.

MACHINE CONFIGURATION

Devices Supported by CP-67

CP-67 is structured to run on an IBM System/360 Model 67. The minimum machine configuration for CP-67 is:

2067-1 or 2067-2 Processing Unit
Recommended feature:
#4434 Floating Storage Addressing (Model 1 only)

2365	Processor Storage
1052	Printer-Keyboard Model 7
1403	Printer

2540 Card Read Punch
3 2311 Disk Storage Drives or 2314 Direct Access Storage Facility (2 drives minimum)
2400 Nine-Track Magnetic Tape Unit, 800 or 1600 bpi
2702 or 2703 Transmission Control or
2701 Data Adapter Unit

Terminals Supported by CP-67 as Machine Operator's Console

1051/1052 Model 1 or Model 2 Data Communication System
Features and Specifications:
Data Set Attachment (#9114)
IBM Line Adapter (#4647)
Receive Interrupt (#6100 or RPQ E27428) required
Transmit Interrupt (#7900 or RPQ E26903) required
Text Time-out Suppression (#9698) required

1056 Card Reader Model 3

2741-1,-2 Communication Terminals
Features and Specifications:
Data Set Attachment (#9114)
Data Set Attachment (#9115)
IBM Line Adapter (#4635, #4647)
Dial-Up (#3255) required
Receive Interrupt (#4708) required
Transmit Interrupt (#7900 or RPQ E40681) required
Print Inhibit (#5501) desirable

Line control for teletypewriter terminals (*) compatible with the IBM Telegraph Terminal Control Type II Adapter (8-level ASCII code at 110 bps).

Transmission Control Units Supported by CP-67

2701 Data Adapter Unit

Terminals	2701 Adapter
8-level ASCII, 110 bps*	7885

2702 Transmission Control

Terminals	Terminal Control Base	Terminal Control	Line Adapter
2741s, 1050	9696 or 7935	4615, 9684, 8200**	3233
8-level ASCII, 110 bps*	9697 or 7935	7912	3233

2703 Transmission Control

Terminals	Line Speed Option	Line Set	Terminal Control	Line Bases
2741s,1050	4878	3205/6	4619,4696,8200****	7505
8-level ASCII, 110 bps*	4877	3205/6	7905, 7912	7505

* The customer is responsible for terminal compatibility with this program. IBM assumes no responsibility for the impact that any changes to the IBM-supplied products or programs may have on terminals provided by others.

** Feature 8200 on the 2702 is equivalent to the 2741 Break feature #8055 and the Type I Break RPQ E46765 on the 2702.

**** Feature 8200 on the 2703 is equivalent to the 2741 Break feature #8055 and the Type I Break RPQ E53715 on the 2703.

Other Devices Supported by CP-67

Additional devices used by CP-67 are:

2301 Drum Storage
2303 Drum Storage

2870 Multiplexer Channel
#6990, 6991, 6992 1, 2, 3 Selector Subchannels

Devices Used Only by an Operating System in a Virtual Machine and not by CP-67

2321 Data Cell Drive

2400 Magnetic Tape Units

2250 Display Unit
2260 Display Station

2860 Selector Channel
#1850 Channel-to-Channel Adapter

2780 Data Transmission Terminal
1130 Computing System

VIRTUAL COMPUTERS

A virtual computing system is a time-sharing system that provides greater flexibility of application to the user. A time-sharing system provides a set of software facilities through which users share machine facilities; the extent of the software facilities available to a user depends on how the system is defined. A virtual computing system simulates hardware facilities that allow the user to load a software system (Operating System/360, for example) that provides the particular facilities he requires; the user - not the system - determines the facilities available to him.

For each user, CP-67 creates a virtual computer which is an exact replica of a System 360; a programmer at a remote installation can use the computing system as if it were exclusively his. CP-67 accomplishes this by:

Scheduling and allocating main storage space, CPU time, and I/O devices to the virtual computers

Handling all interruptions

Protecting system files, user programs, and user data during execution

Keeping statistics on the use and performance of the "real" system

CP-67 can simulate a Model 65 or Model 67 (simplex, 24 bit addressing) computing system, capable of executing any instruction except Diagnose.

For direct access storage devices, CP-67 will support more than one "user" or virtual machine on a pack. This concept is called "mini-disks". Essentially, a virtual machine is allocated a number of contiguous cylinders from the disk pack, and these cylinders can be located starting at any "real" cylinder address. A "relocation" factor and "boundary" number define the start and extent of a user's "mini-disk".

TIME SHARING

The Control Program shares execution time in the central processing unit (CPU) among the virtual computers on a demand basis and on a scheduled basis. The Control Program schedules and allots units of CPU time to the virtual computers. When a particular virtual computer has used up its unit of time, the Control Program locates the next "runnable" virtual computer and passes control to it for a corresponding interval of time. If the virtual computer currently in control must wait for some event, the Control Program gives control to another virtual computer which has demanded the CPU.

PROGRAM STATES

When instructions in the Control Program (CP-67) are being executed, the real computer is in the supervisor state; at all other times, when running virtual machines, it is in the problem state. Therefore, privileged instructions can be executed only by the Control Program. Programs running on a virtual computer can issue privileged instructions; such an instruction causes an interruption that is handled by the Control Program. Under certain conditions, the Control Program simulates the virtual privileged instructions.

PAGING

Paging is the technique used by the Control Program to share main storage among concurrent users. The objective of this technique is to keep in main storage only those portions of each user's program that are required at a given point in time. This eliminates the need for the programmer to externally segment each program into manageable units. The units automatically used by CP-67 are 4096-byte blocks called "pages". By breaking programs into pages, main storage can be allocated in page increments, and pages can be loaded dynamically for execution. Thus, at execution time, main storage holds only the active part of each user's program.

When a user starts his session, the Control Program, as a result of an IPL operation (see the description of IPL under "Console Function Subroutines" in Section 2) places the user's programming system IPL program into main storage. The page is loaded into an available block of main storage that starts on a page boundary. The page is not necessarily loaded at the same relative main storage position as it would occupy were the programming system running on a real computer. This is possible because of the dynamic address relocation abilities of the Model 67. (Refer to IBM System/360 Model 67: Functional Characteristics, A27-2719.)

As the user's program is executing, the hardware dynamically converts references to relative addresses into actual main storage addresses. When the program refers to an address in a page that is not in main storage, an interruption occurs and the Control Program loads the required page into main storage. Then execution continues with the referenced addresses being dynamically relocated.

Because of the dynamic address relocation feature, the pages of a user program need not occupy contiguous locations and may be scattered throughout main storage (see Figure 1). Also, because of the high demand for main storage in a multiple-user environment, the Control Program shares main storage among the active pages of the programming systems of competing users.

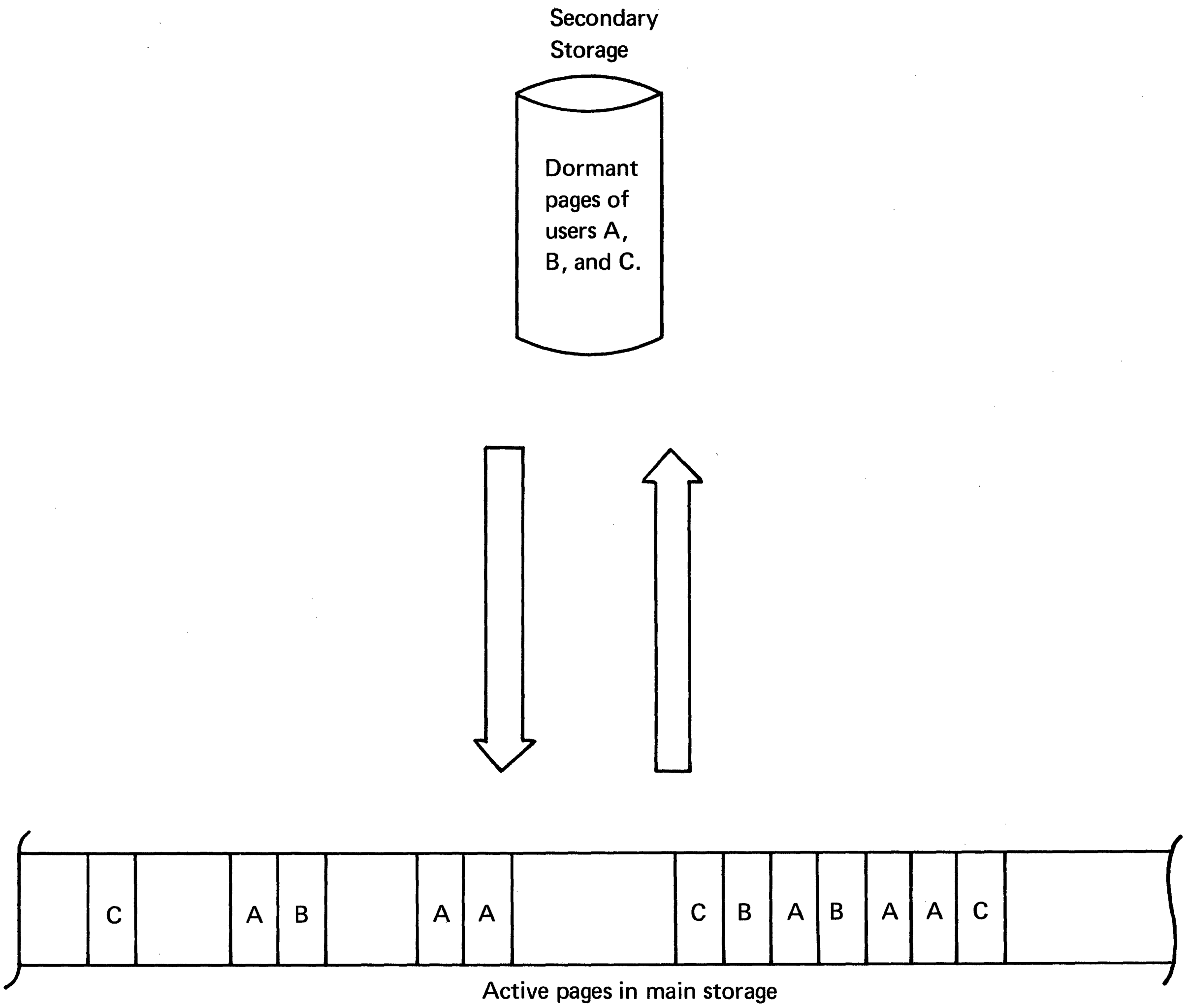

Secondary Storage
Dormant pages of users A, B, and C.
C
A
B
A
A
C
B
A
B
A
A
C
Active pages in main storage

FIGURE 1. Sharing Storage Among Concurrent Users

Finally, when main storage is completely filled and it becomes necessary to bring in another page, page swapping occurs. An appropriate page of one user's program in main storage is written onto secondary storage and the required page is brought into main storage in its place. (If the page to be replaced has previously been swapped, and has not been modified since it was last swapped, it is not necessary to write it onto secondary storage because a copy already exists there.) When the particular page that was replaced is again required, it is obtained from secondary storage and swapped with one that is in main storage (see figure 2).

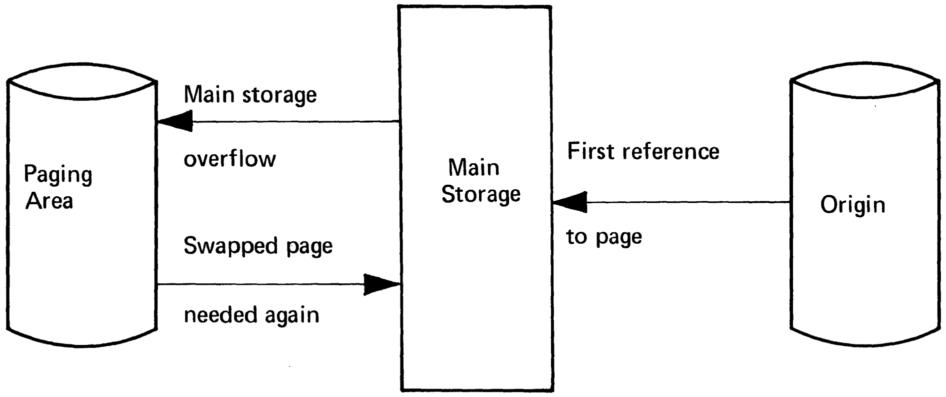


FIGURE 2. Page Swapping

The following list contains some statistics on the drums and disks used for paging.

Paging Devices

2301	4096 bytes/record	9 records/2 tracks
2303	4096 bytes/record	1 record/track
2314	829 bytes/record, 5 records/page	15 records/2 tracks
2311	829 bytes/record, 5 records/page	4 records/track

The following are guidelines for the number of cylinders required for paging virtual memory. Note that CP-67 does not allocate pages for virtual memory until each page has been referenced. When the first page is referenced, the address of the swapping area is put in the swap table. These guidelines represent the total number of cylinders required if all the pages of 256K virtual memory are referenced.

Virt Memory Size	Device Type	Number of Cylinders Required for Paging
256K	2311	8
256K	2314	3

Figure 3 gives an overview of the paging operation.

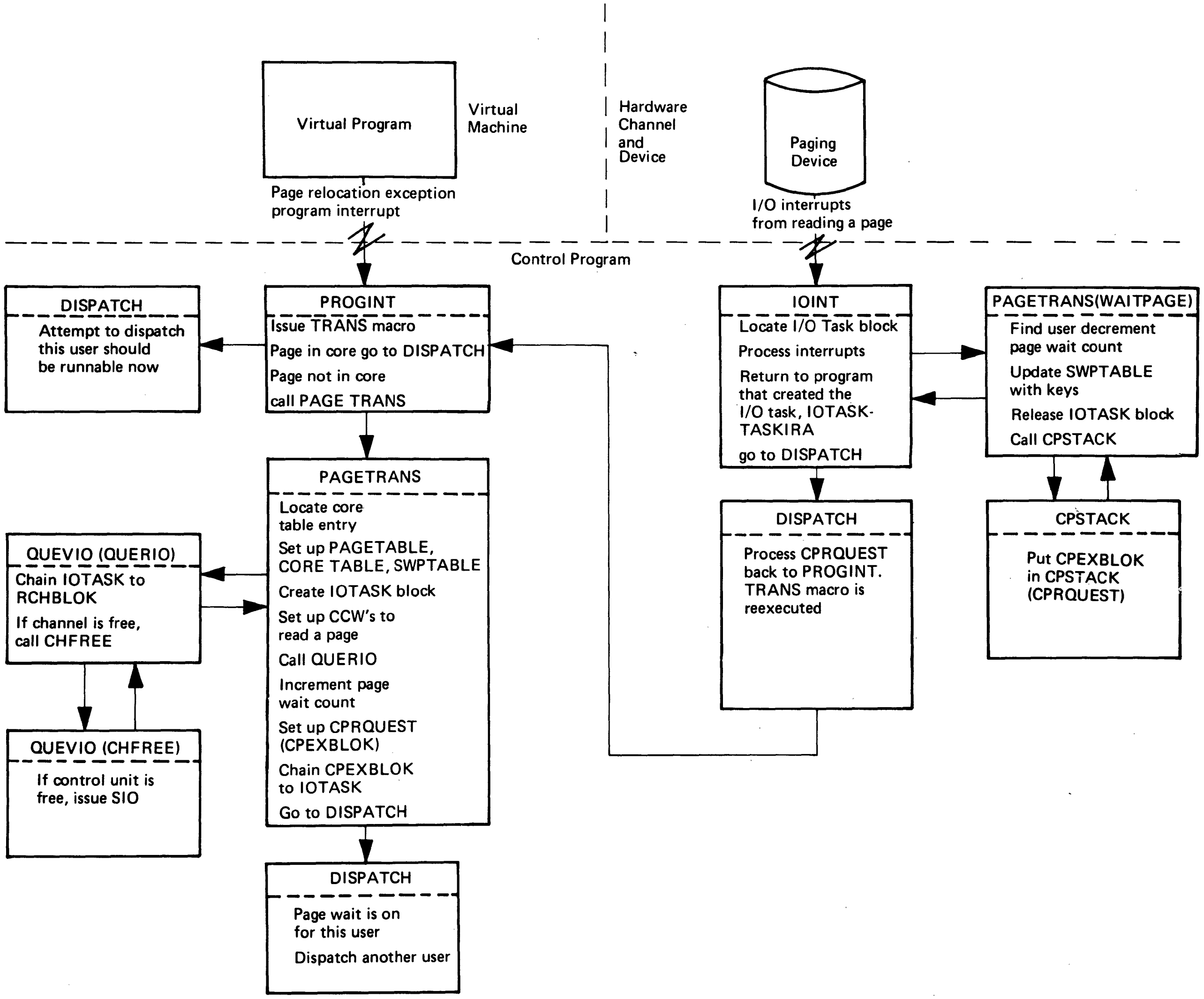

Virtual Program
Virtual Machine
Hardware Channel and Device
Paging Device
Page relocation exception program interrupt
I/O interrupts from reading a page
Control Program
DISPATCH
Attempt to dispatch this user should be runnable now
PROGINT
Issue TRANS macro
Page in core go to DISPATCH
Page not in core
call PAGE TRANS
IOINT
Locate I/O Task block
Process interrupts
Return to program that created the I/O task, IOTASK-TASKIRA
go to DISPATCH
PAGETRANS(WAITPAGE)
Find user decrement page wait count
Update SWPTABLE with keys
Release IOTASK block
Call CPSTACK
PAGETRANS
Locate core table entry
Set up PAGETABLE, CORE TABLE, SWPTABLE
Create IOTASK block
Set up CCW's to read a page
Call QUERIO
Increment page wait count
Set up CPRQUEST (CPEXBLOK)
Chain CPEXBLOK to IOTASK
Go to DISPATCH
QUEVIO (QUERIO)
Chain IOTASK to RCHBLOK
If channel is free, call CHFREE
QUEVIO (CHFREE)
If control unit is free, issue SIO
DISPATCH
Process CPRQUEST back to PROGINT. TRANS macro is reexecuted
CPSTACK
Put CPEXBLOK in CPSTACK (CPRQUEST)
DISPATCH
Page wait is on for this user
Dispatch another user

FIGURE 3. Paging Operation

READER/PRINTER/PUNCH INPUT-OUTPUT

The Control Program simulates card reader, punch, and printer operations requested for programs running on virtual computers by using a spooling operation to simulate multiple virtual unit record devices. If a program running on a virtual machine is to process a card file, that file must first be submitted to the machine-room operator, headed by a card identifying the user for whom it is intended, and entered by the operator into the system. When the operator enters the file (through the real card reader) the Control Program converts it to a disk file which is associated with the corresponding virtual computer. Then, when a program running on that virtual machine issues a start input-output (SIO) instruction to the virtual card reader, the Control Program intercepts it, takes the appropriate card image from the disk file, and makes it available to the program in the same manner as the real card reader would. This process is repeated for each subsequent operation directed to the virtual card reader. This process works in reverse for punch and printer operation. When a program on a virtual machine wishes to create printer or punch output, it issues successive SIO operations to its virtual printer or punch. The Control Program intercepts these attempted input-output operations, obtains the print line or punched card images, and creates a disk file from them. The disk file is then printed or punched on the real devices at a later time when the device is available for use.

OTHER INPUT-OUTPUT

Other input-output operations issued by programs running on a user's virtual machine are converted to real input-output operations by the Control Program. Translation consists of four major steps: (1) device address translation, (2) command sequence translation with appropriate paging operations, (3) scheduling the input-output operation on the real hardware, and (4) receiving and properly reflecting the interrupts returning from the input-output operation after being started.

During device address translation, the Control Program converts the virtual device address associated with the SIO operation to its real equivalent. This conversion is required because each virtual device has been mapped to an extent or area on an equivalent device on the real computer during system set-up operations. To illustrate how this conversion works, assume that the user has a virtual disk at address 190 and that this has been mapped to an extent starting at cylinder 10 on a real disk whose label is DISK01. Assume further that at system start-up time it has been ascertained that DISK01 is currently mounted on real disk drive 235. If a user program issues a write to cylinder 00 track 0 record 1 of the virtual disk 190, the Control Program will intercept it and convert it to a write to cylinder 10 track 0 record 1 of the real disk at 235.

Conversion of reads from virtual disks are handled similarly.

During command sequence translation, the Control Program (via CCWTRANS) converts the channel command sequence provided by the virtual machine into an equivalent real channel command word list. This is required because virtual channel command words can refer to contiguous virtual memory space overlapping a page boundary. In the real machine, these virtual pages would not necessarily be in contiguous real pages, and the channel command word involved must be split (via the chain data feature) into two or more channel command words which refer to the real core addresses and which perform the same function. Thus the entire virtual CCW sequence is translated into an equivalent sequence held in free storage. The channel is then run off of the real sequence. Note that this is the source of a major restriction in CP-67--channel command sequences may not be modified while the input-output operation is in progress. The modifications will not be reflected in real memory, on which the real channel is running.

If the ISAM option has been chosen during the generation of CP, and a virtual machine has been assigned the ISAM option in the directory, certain self-modifying I/O sequences will be supported (specifically OS-ISAM). The channel program is scanned to determine whether any of the channel command words modify other channel command words within this I/O sequence. The channel program is retranslated and reexecuted for each channel command word that modifies another channel command word within the channel program. (See "CCW Translator - CCWTRANS" for details.)

The scheduling of the input-output operation is handled by QUEVIO and CHFREE, which are discussed elsewhere. They return to the virtual input-output executive (VIOEXEC) when the operation is finished.

The interruption processing is provided by VIOEXEC after initial processing by IOINT. The interrupts are unstacked to the user in the same order as they would appear in the real machine. UNTRANS is called to convert the addresses returned in the channel status word (which refer to the input-output string in real memory) to the virtual addresses required by the user.

SECTION 2: METHOD OF OPERATION

This section segments CP-67 into its functional units and discusses each as an entity.

SYSTEM SETUP OPERATIONS

Before initializing the Control Program, the DIRECT stand-alone utility routine must be used to allocate cylinders between permanent file space and temporary spooling and paging space. It is assumed that the disk packs involved have been formatted and labeled (via the FORMAT utility) into the CP-67 format.

Input to DIRECT may be of two types: (1) control statements specifying allocation of DASD cylinders (ALLOCATE) and (2) control cards defining a user's virtual system (DIRECTORY). Figure 4 illustrates the relationships of tables and files created by DIRECT.

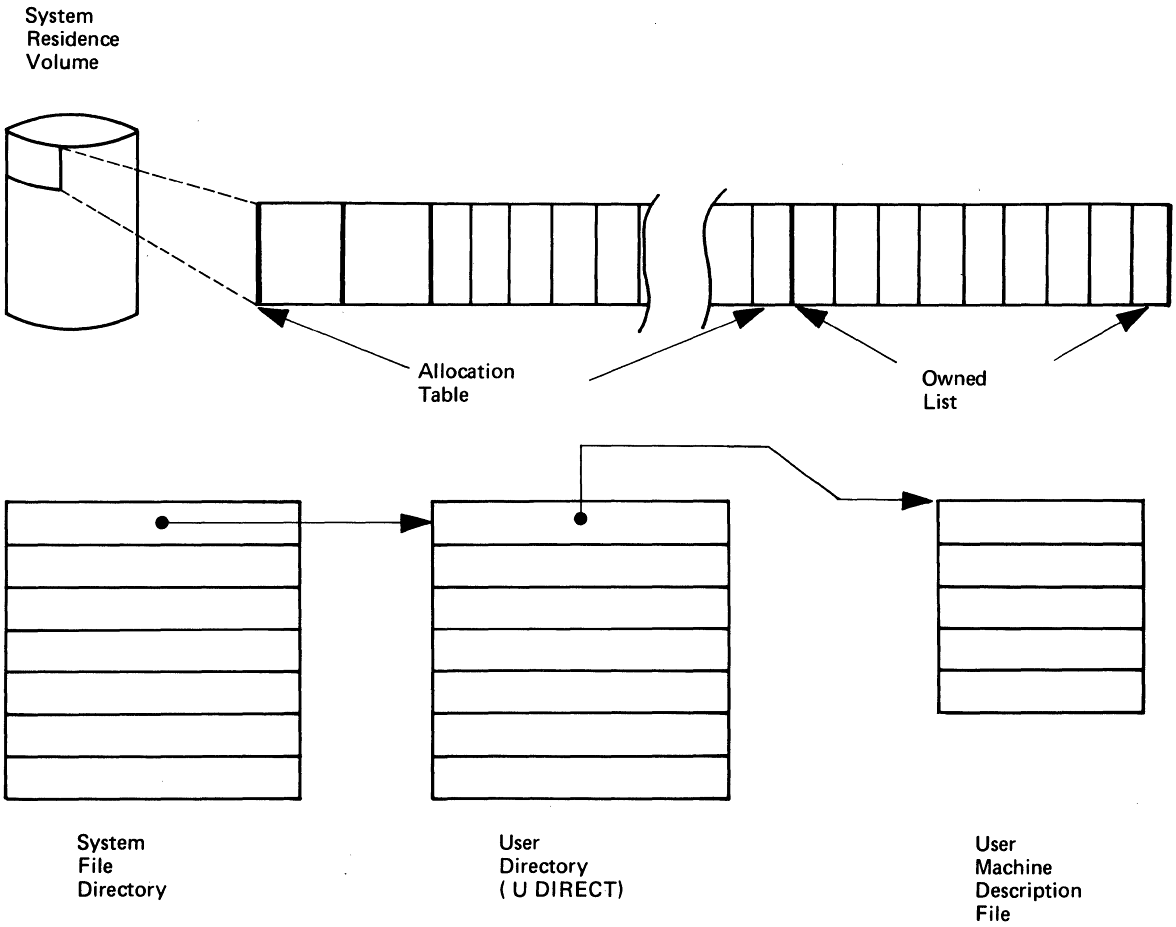


FIGURE 4. Tables and Files Created by DIRECT

Cylinder Allocation

DIRECT reads the allocation table from the volume specified in the ALLOCATE statement and determines whether temporary or permanent allocation is requested.

Temporary cylinder allocation (making the cylinders available for temporary usage, such as paging and spooling) is indicated by placing an x'00' in the corresponding allocation table entry. Permanent cylinder allocation (making cylinders available for permanent file residence) is indicated by placing an x'01' in the entry. Cylinders to be used as T (temporary) disk space are designated by an x'02' while cylinders containing user directories are marked x'04'.

At the end of an allocation run for a particular volume (indicated by an *EOA* statement), cylinder 0 is permanently allocated (for the allocation table itself and the label) and an x'0F' is placed in the last allocation table entry.

Establishing User Directories

When a DIRECTORY control statement is read by DIRECT, a system residence volume will be created on the unit specified in the control statement. The allocation table is read from the system residence volume, and the "owned" list is initialized to contain the system residence volume. The owned list, beginning with the first byte after the allocation table, contains the VOLIDS of all volumes to be considered owned by the Control Program and available for possible temporary allocation. The system residence volume VOLID becomes the first entry in the owned list.

The "system file directory" is created; the system file directory contains information (such as file name, volume label, and device position of first record) for all files used internally by the Control Program. An entry for the "user directory file" (U.DIRECT) is initially placed in the system file directory.

Additional Control Statements

After the owned list and the system file directory have been initialized, additional control statements which identify users and configure their virtual machines are read. The following paragraphs describe the processing performed for each record type.

USER Statement Processing

USER statements supply identification and accounting information for users of CP-67. Before a user directory file entry is created for the USER statement, the user machine description file must be opened, and the first four bytes of a new machine description entry are reserved for the virtual machine core size. Entries are created for USER statements and written onto disk as records in the user directory file (U.DIRECT). User directory entries contain the following information for each user:

User's external identification

User's password

Accounting information

User's machine description file name

User's privilege class

User's priority

User's options

CORE Statement Processing

CORE statements define the size of core storage in the virtual machine being defined for the user identified in the preceding USER statement. The core size desired must be a multiple of 8K (=8192) bytes and may be specified as either "nnnK" or "nnnM". The size is entered into the first four bytes of the user's machine description record.

UNIT Statement Processing

UNIT statements define virtual devices in the virtual machine being defined for the preceding USER card. The following type of information is placed in the user machine description file entry (MDENT) for each specified device:

Virtual device address

Device type

Device relocation factor for DASD devices

Device bound for DASD devices

Passwords and status information for device access

See the description of control block MDENT in Section 4 for details.

OWN Statement Processing

OWN statements specify the VOLIDs of volumes to be considered "owned" by the Control Program. Each specified VOLID is added to the "owned" list, which is retained after the allocation table on cylinder 0 head 0 record 3 of the system residence volume. An "owned" volume is any disk on which an allocation table has been written; it contains user files and/or temporary spooling and paging areas used by the Control Program.

EOU and *EOD* Statement Processing

An *EOU* statement indicates the end of a machine description for a particular user. A unique name is generated for the user machine description file (actually a floating point number starting at 1.0 and incrementing by 1.0 for each new file), and is placed in the corresponding user directory entry. The user machine description file is then written onto disk.

An *EOD* statement indicates the end of input for the user directory creation process. The user directory (U.DIRECT), the system file directory, and the system residence volume allocation table are written onto the disk to complete DIRECT processing.

Complete specifications for creating the user directory are contained in the CP-67 Operator's Guide under "Directory Allocation and Creation".

System Backup Operation

The CMS Tape Dump command is designed for user virtual machine back-up functions. The CMS program, CPDMPRST, is available for both users and the operations department, to back-up 2311 or 2314 disk packs--either minidisks or full volumes containing one or more minidisks of varying formats. During dumps, if a bad track is encountered for which an alternate track was assigned by the MINIDASD program, data from the alternate location will be written to the dump volume. The restore function, however, cannot make such use of alternate track assignments; during a restore, a bad track will cause a fatal I/O error.

The CPDMPRST program is modeled after the stand-alone dump/restore utility program of OS/360.

CONTROL PROGRAM INITIALIZATION

CHKPT Program

The IPL sequence reads the CHKPT program from the IPL'ed disk into low core at location X'800'. The CHKPT program performs the following functions:

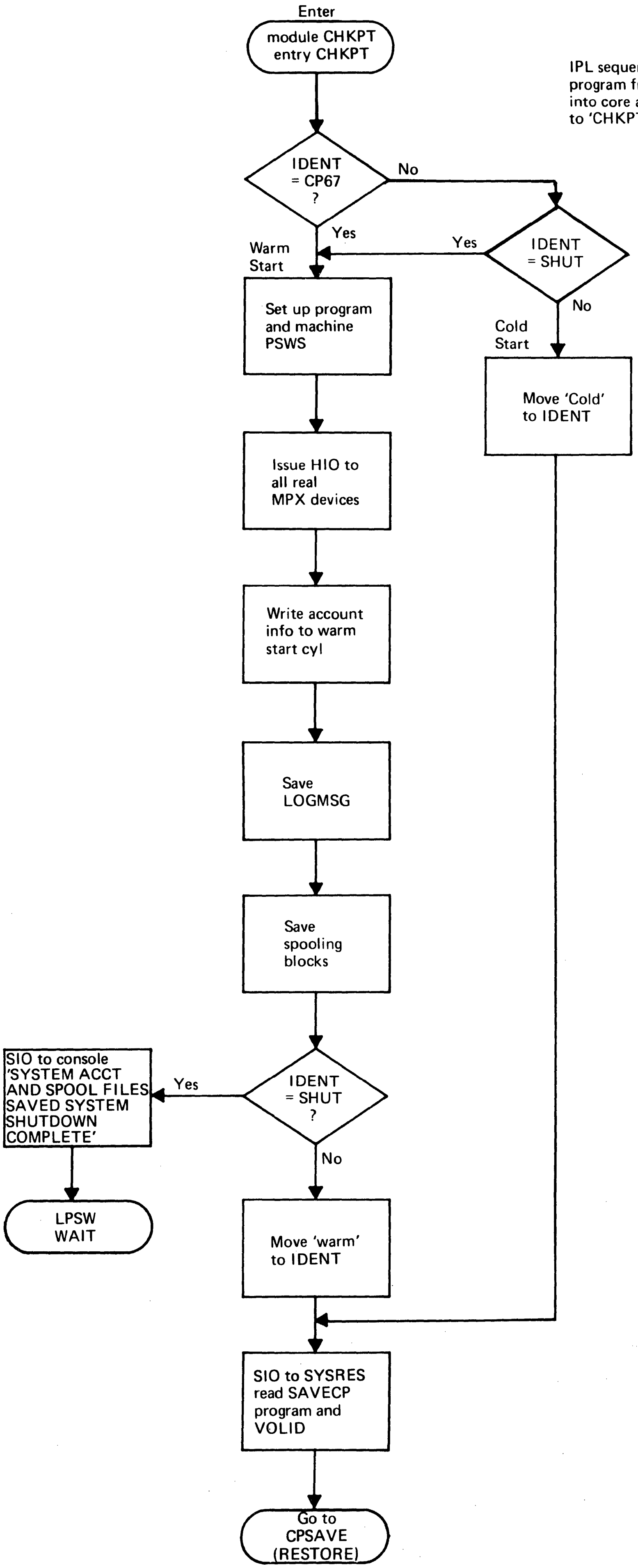
> Examines the CPID word at X'1FC'. If the word contains "CP67" or "SHUT", the IPL is to a "warm" machine (that is, CP-67 has been running, and accounting information and spool file data is available in core); if the CPID word contains anything else, a "cold" machine is assumed and the CHKPT program proceeds to the second phase of initialization described below.
>
> For a "warm" machine, the CHKPT program retrieves user accounting data from the UTABLES and unpunched accounting cards; gets accounting for dedicated devices; saves the system LOGMSG; saves spool file control blocks for active printers and punches and all "closed" user spool files. The data is written on the IPL'ed disk at the SYSWRM cylinder.

If the CPID word contains "CP67", the CHKPT program proceeds to the second phase below. If the CPID word contains "SHUT", shutdown messages are printed, and processing is completed.

The second phase of initialization involves reading the SAVECP program and VOLID from the IPL'ed disk (records 2 and 3) into high core (X'25000') and transfering control to the RESTORE function of SAVECP.

SAVECP (RESTORE function) reads the CP-67 nucleus from disk (SYSDNC cylinder) into core from X'33D' to X'25000'; control is transfered to the CPINIT program now loaded at X'23000'.

| See Figure 5 for a diagram of the CHKPT program operation.



IPL sequence reads 'CHKPT' program from the IPL'ed disk into core and XFERS control to 'CHKPT'

FIGURE 5. CP-67 CHKPT

CPINIT Program

The CPINIT program performs the following functions: (See Figures 6, 7, and 8 for flowcharts of Main Storage, CPSAVE, and CPINIT operation.)

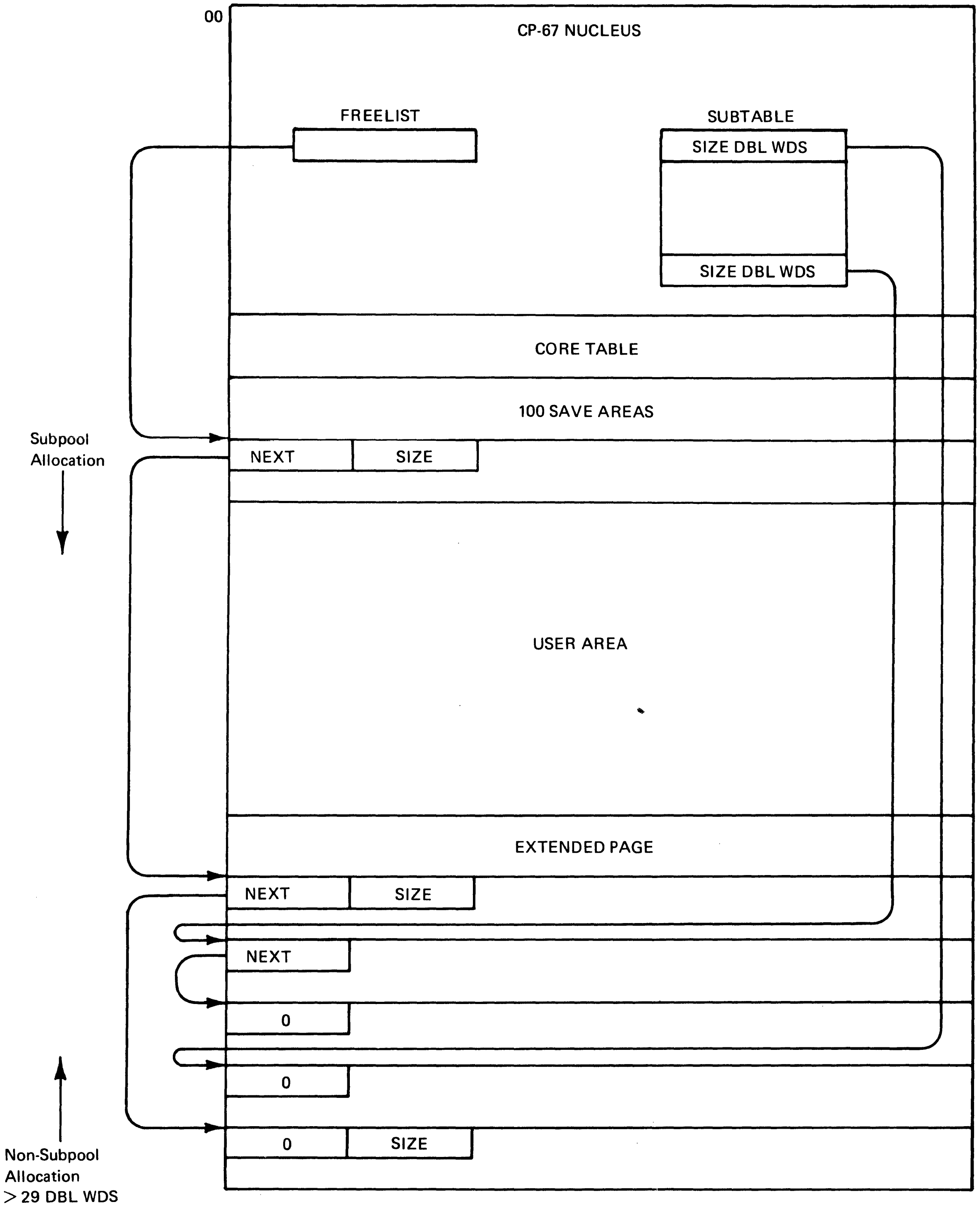


Figure 6. Flowchart of Main Storage Operation

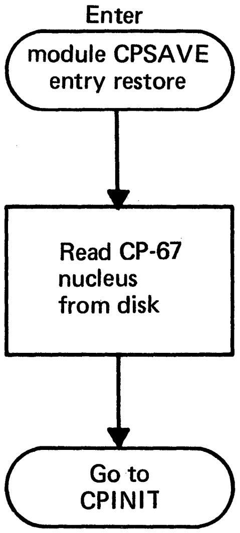


Figure 7. Flowchart of CPSAVE Operation

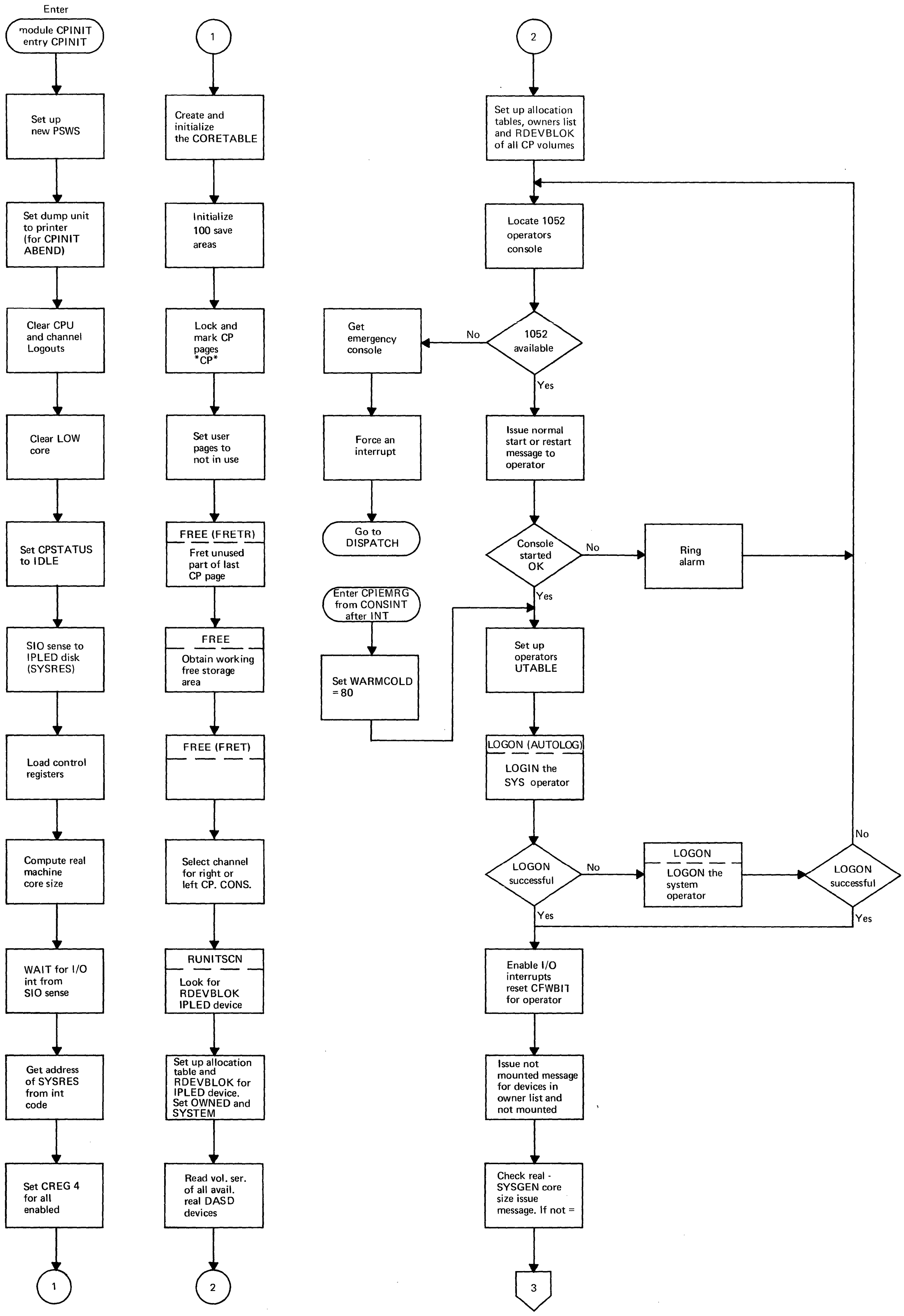

Enter
module CPINIT entry CPINIT
Set up new PSWS
Set dump unit to printer (for CPINIT ABEND)
Clear CPU and channel Logouts
Clear LOW core
Set CPSTATUS to IDLE
SIO sense to IPLED disk (SYSRES)
Load control registers
Compute real machine core size
WAIT for I/O int from SIO sense
Get address of SYSRES from int code
Set CREG 4 for all enabled
1
1
Create and initialize the CORETABLE
Initialize 100 save areas
Lock and mark CP pages *CP*
Set user pages to not in use
FREE (FRETR)
Fret unused part of last CP page
FREE
Obtain working free storage area
FREE (FRET)
Select channel for right or left CP. CONS.
RUNITSCN
Look for RDEVBLOK IPLED device
Set up allocation table and RDEVBLOK for IPLED device. Set OWNED and SYSTEM
Read vol. ser. of all avail. real DASD devices
2
Get emergency console
Force an interrupt
Go to DISPATCH
Enter CPIEMRG from CONSINT after INT
Set WARMCOLD = 80
2
Set up allocation tables, owners list and RDEVBLOK of all CP volumes
Locate 1052 operators console
1052 available
No
Yes
Issue normal start or restart message to operator
Console started OK
No
Yes
Ring alarm
Set up operators UTABLE
LOGON (AUTOLOG)
LOGIN the SYS operator
LOGON successful
No
Yes
LOGON
LOGON the system operator
LOGON successful
No
Yes
Enable I/O interrupts reset CFWBIT for operator
Issue not mounted message for devices in owner list and not mounted
Check real - SYSGEN core size issue message. If not =
3

Figure 8. Flowchart of CPINIT Operation (1 of 2)

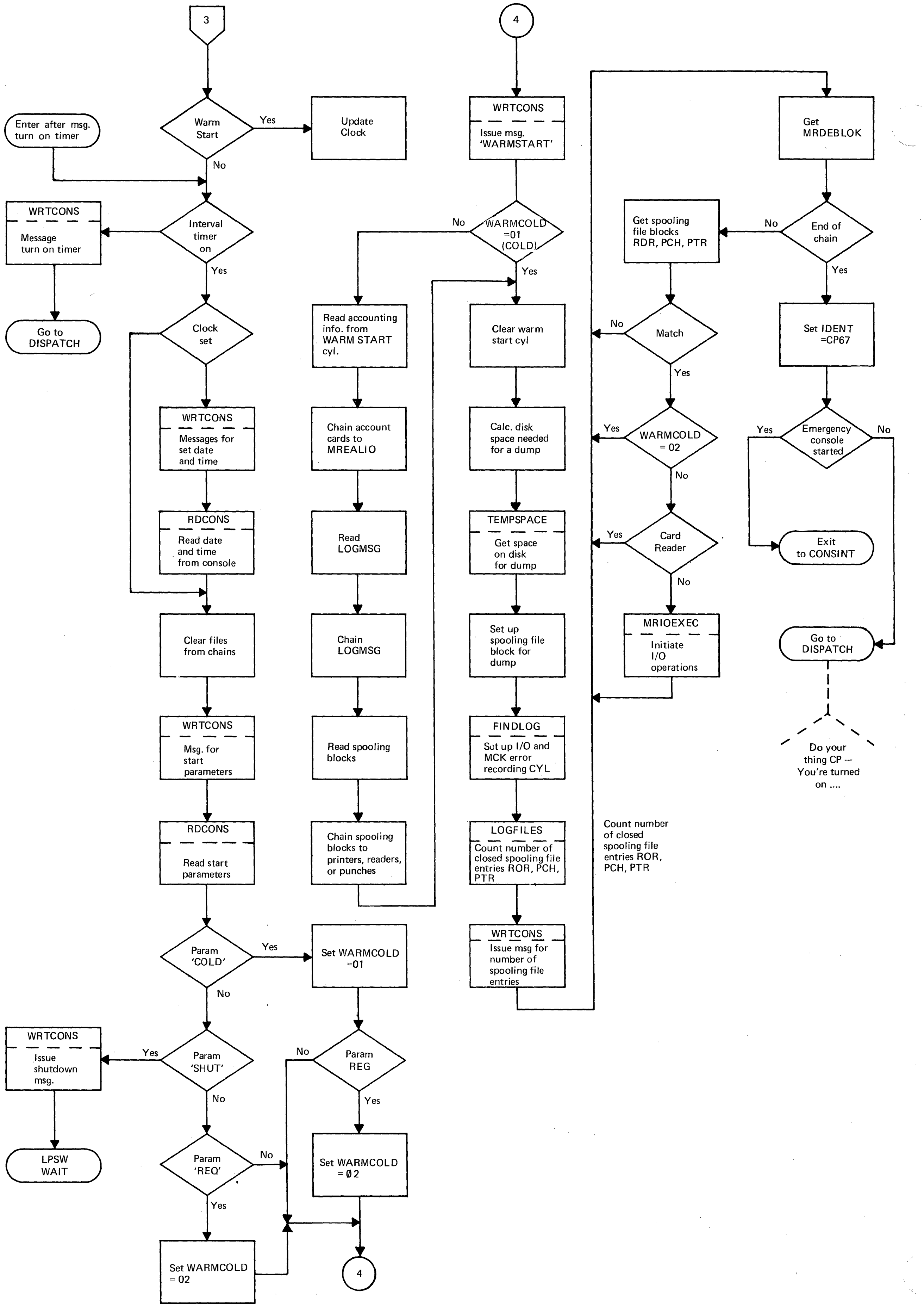

3
Enter after msg. turn on timer
Warm Start
Yes
Update Clock
No
WRTCONS
Message turn on timer
Interval timer on
Yes
Go to DISPATCH
Clock set
WRTCONS
Messages for set date and time
RDCONS
Read date and time from console
Clear files from chains
WRTCONS
Msg. for start parameters
RDCONS
Read start parameters
Param 'COLD'
Yes
Set WARMCOLD =01
No
WRTCONS
Issue shutdown msg.
Yes
Param 'SHUT'
No
LPSW WAIT
Param 'REQ'
No
Yes
Set WARMCOLD = 02
No
Param REG
Yes
Set WARMCOLD = Ø2
4
4
WRTCONS
Issue msg. 'WARMSTART'
No
WARMCOLD =01 (COLD)
Yes
Read accounting info. from WARM START cyl.
Chain account cards to MREALIO
Read LOGMSG
Chain LOGMSG
Read spooling blocks
Chain spooling blocks to printers, readers, or punches
Clear warm start cyl
Calc. disk space needed for a dump
TEMPSPACE
Get space on disk for dump
Set up spooling file block for dump
FINDLOG
Set up I/O and MCK error recording CYL
LOGFILES
Count number of closed spooling file entries ROR, PCH, PTR
WRTCONS
Issue msg for number of spooling file entries
Count number of closed spooling file entries ROR, PCH, PTR
Get spooling file blocks RDR, PCH, PTR
No
Match
Yes
Yes
WARMCOLD = 02
No
Yes
Card Reader
No
MRIOEXEC
Initiate I/O operations
Get MRDEBLOK
No
End of chain
Yes
Set IDENT =CP67
Yes
Emergency console started
No
Exit to CONSINT
Go to DISPATCH
Do your thing CP -- You're turned on ....

Figure 8. Flowchart of CPINIT Operation (2 of 2)

Determines, by examining the CPID word, whether initializing is on a warm machine after a disk ABEND dump

Loads the 360/67 control registers

Sets the new PSW's

Computes the real machine core size

Creates and initializes the CORTABLE at the end of the resident nucleus (size is determined by "real" machine size)

Initializes 35 save areas for CP-67 linkage at the end of the CORTABLE

Determines whether IPL'ed on left or right half of a possible duplex configuration

Calls FREE and FRET to obtain working free storage area based upon "real" machine size

Creates control block for IPL'ed disk allocation table and OWNED list

Determines availability of all DASD devices defined in the real I/O (RIO) configuration; reads VOLID of all available DASD devices; chains allocation tables of all available OWNed volumes

Locates 1052 system console and writes initialization message; if message fails, rings alarm, locates emergency console, and initializes for emergency startup

Calls LOGIN to log in the system operator

Checks the OWNED list for volumes not mounted and gives messages

Checks core size for SYSCORE size; gives message if not equal

Checks for timer in operation

Prompts operator to set date and time and to specify startup parameters

For a WARM start, reads the data from the SYSWRM cylinder and restructures the LOGMSG and spool file control blocks; chains the accounting information for punching

Invalidates the SYSWRM data to avoid future erroneous startup

Gets spooling space and control blocks for a disk dump

Calls FINDLOG to initialize the error recording

Commences spooling output if any

Sets the CPID word to "CP67"

Runs the system

Core Table Initialization

The core table consists of a 16-btye entry for each page (4096 bytes) of real core. Each core table entry will point to a corresponding entry in the swap table, which is used by core management routines in paging. The physical location of a page in real core is determined by the relative location of its corresponding entry in the core table; for example, the first core table entry corresponds to the first page of real core. The core table entries for the pages which contain the Control Program are locked with an identifier of "*CP*" to make them unavailable for paging operations. The remainder of the core table entries are initialized to X'00FFFFFF'.

For a real machine with a 256K main storage, the unused portion of the last Control Program page and six additional pages are reserved as a Control Program work area. For each additional core box, six more pages are reserved for the larger expected number of users. The pages for free storage are also locked and identified with "FREE".

Allocation Table Chaining

The address of the system residence VOLID and of the allocation table for the system residence volume is passed to CPINIT by the routine SAVECP. The VOLID and allocation table address are entered into the real device control block (RDEVBLOK) for the system residence device.

Each additional real device control block is examined to determine whether the corresponding device is mounted. VOLIDs are read from all mounted devices and compared against the entries in the OWNED list (obtained from the system residence volume). Allocation tables from all owned volumes are read and chained according to device type.

Figure 9 illustrates the chaining of allocation tables and their relationship to real device control blocks.

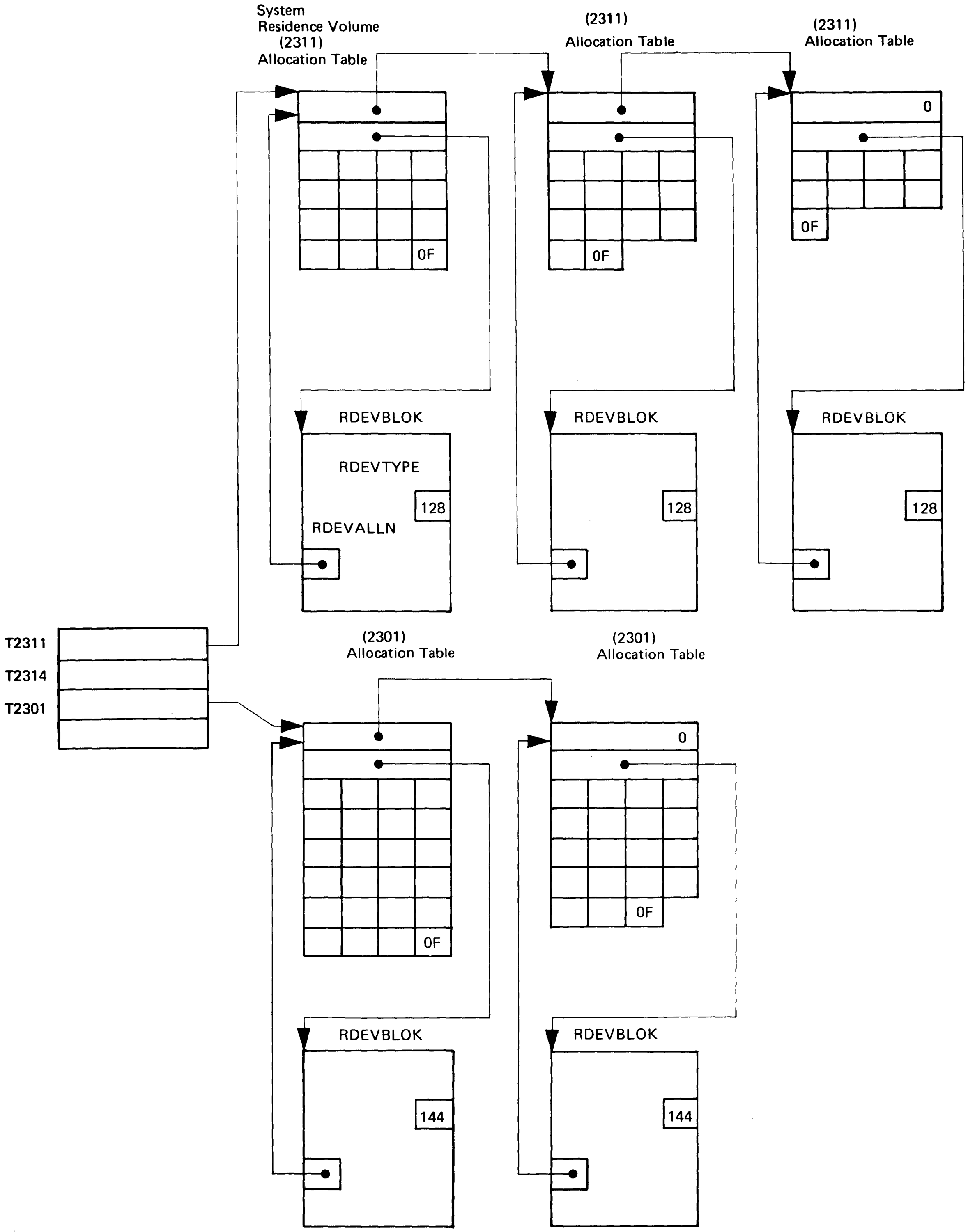


FIGURE 9. Chaining of Allocation Tables and Real Device Blocks.

ATTACHING A USER TO THE SYSTEM

(See Figure 10 for an overview diagram.)

IDENTIFY Routine

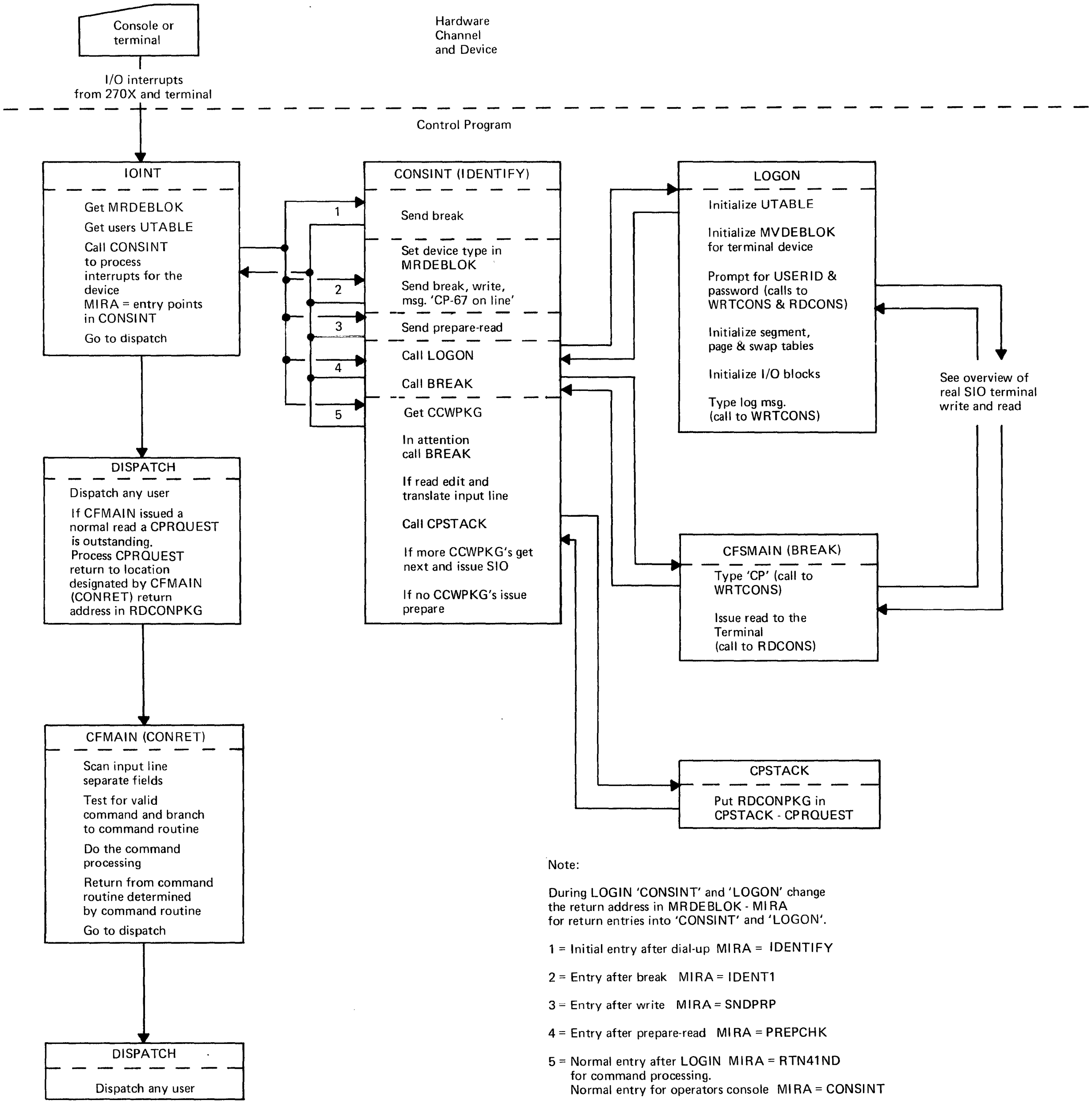
When the Control Program receives the initial interrupt from a terminal (normally initiated by dialing in on a data-phone) the IDENTIFY routine is entered. IDENTIFY performs the following operations:

Determines the terminal device type (1050 or 2741) and enters the type into the multiplexer real device block (MRDEBLOK).

Writes to the terminal the message "CP-67 Online".

Places the address of the BREAK routine in the multiplexer interrupt return address (MIRA).

Puts the terminal line in a state to receive an attention.


Console or terminal
Hardware Channel and Device
I/O interrupts from 270X and terminal
Control Program
IOINT
Get MRDEBLOK
Get users UTABLE
Call CONSINT to process interrupts for the device
MIRA = entry points in CONSINT
Go to dispatch
CONSINT (IDENTIFY)
1
Send break
Set device type in MRDEBLOK
2
Send break, write, msg. 'CP-67 on line'
3
Send prepare-read
4
Call LOGON
Call BREAK
5
Get CCWPKG
In attention call BREAK
If read edit and translate input line
Call CPSTACK
If more CCWPKG's get next and issue SIO
If no CCWPKG's issue prepare
LOGON
Initialize UTABLE
Initialize MVDEBLOK for terminal device
Prompt for USERID & password (calls to WRTCONS & RDCONS)
Initialize segment, page & swap tables
Initialize I/O blocks
Type log msg. (call to WRTCONS)
See overview of real SIO terminal write and read
DISPATCH
Dispatch any user
If CFMAIN issued a normal read a CPRQUEST is outstanding.
Process CPRQUEST return to location designated by CFMAIN (CONRET) return address in RDCONPKG
CFSMAIN (BREAK)
Type 'CP' (call to WRTCONS)
Issue read to the Terminal (call to RDCONS)
CPSTACK
Put RDCONPKG in CPSTACK - CPRQUEST
CFMAIN (CONRET)
Scan input line separate fields
Test for valid command and branch to command routine
Do the command processing
Return from command routine determined by command routine
Go to dispatch
DISPATCH
Dispatch any user

Note:

During LOGIN 'CONSINT' and 'LOGON' change the return address in MRDEBLOK - MIRA for return entries into 'CONSINT' and 'LOGON'.

1 = Initial entry after dial-up MIRA = IDENTIFY

2 = Entry after break MIRA = IDENT1

3 = Entry after write MIRA = SNDPRP

4 = Entry after prepare-read MIRA = PREPCHK

5 = Normal entry after LOGIN MIRA = RTN41ND
for command processing.
Normal entry for operators console MIRA = CONSINT

Figure 10. CP-67 Overview of Attaching a User to the System

CONSINT Routine

When the next terminal interrupt occurs, the CONSINT routine receives control (via MIRA). CONSINT is also entered whenever the input-output interrupt handler (IOINT) determines that a terminal interrupt has occurred from the request or attention button on the terminal. CINSINT determines whether a user is logged on at the terminal; if not, the LOGON routine is called to attach the new user to the system.

LOGON Routine

Operations performed by LOGIN are:

Allocating and initializing the primary user control table (UTABLE).

Checking the user's external identification (USERID) and password against entries in the user directory.

Allocating and initializing the segment table, page table, and swap table for the user's machine.

Allocating the UTABLE extension (EXTUTAB) if the virtual machine is a Model 67.

Creating virtual I/O blocks to describe the user's virtual machine.

Mapping virtual devices to real devices by chaining virtual device blocks to real device blocks.

Figure 11 indicates the relationships of tables created by the LOGIN routine. When LOGIN functions are completed, the user is placed in console function mode with a read on his terminal by CONSINT calling BREAK.

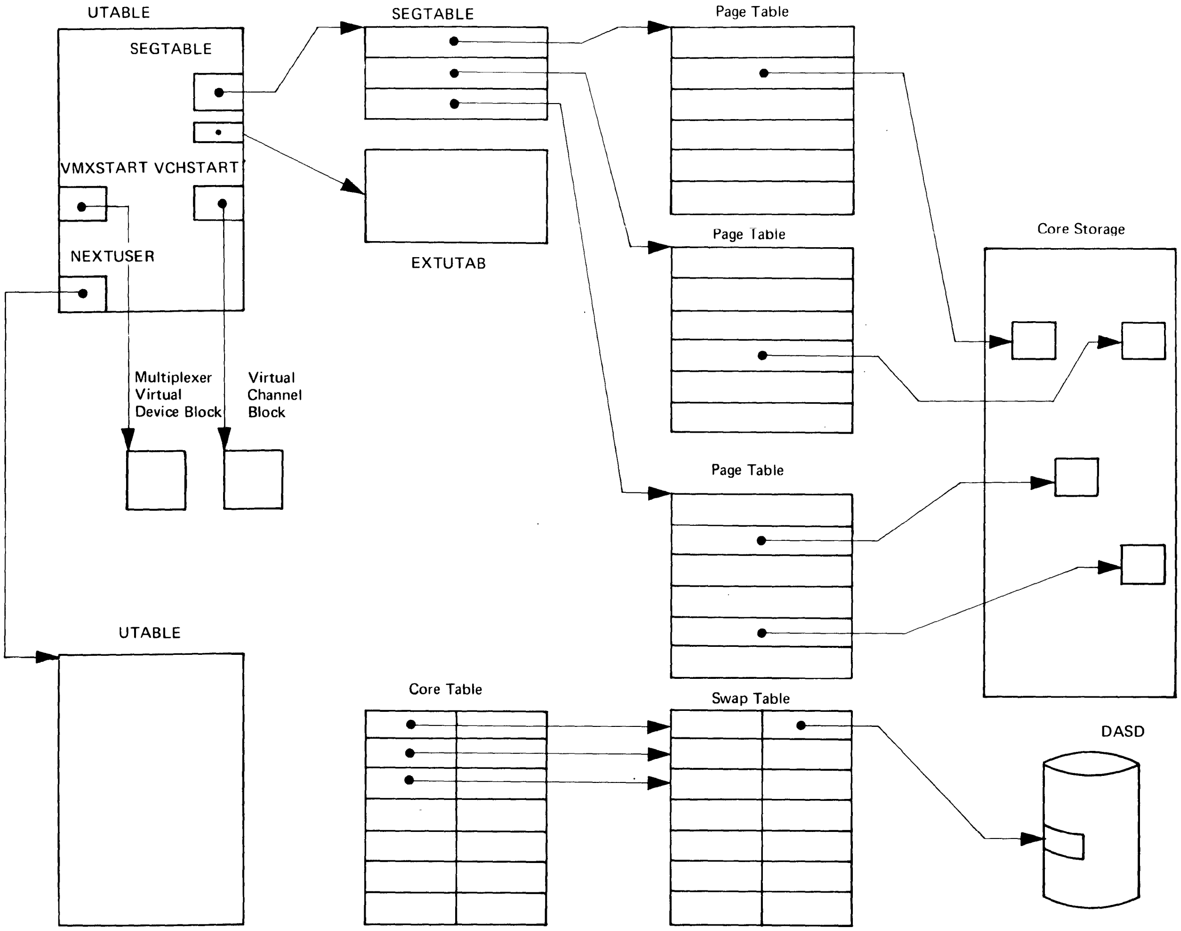


FIGURE 11. LOGON Tables

UTABLE Initialization

The primary user control table (UTABLE) contains a description of the user's virtual machine and information on the status of the machine. When a new user is logged on, space is obtained for his UTABLE from free storage, and the following information is entered:

The start of the virtual multiplexer device block list (the address of the virtual multiplexer block MVDEBLOK created for the user's terminal device).

USERID after it has been verified by comparing it against the entries in the user directory.

Virtual machine core size (obtained from the user's machine description file).

Address of the segment table.

Address of the first virtual channel block in the virtual channel list.

Address of the UTABLE extension, if the virtual machine has the ability to run in extended mode (virtual 67).

Segment Table Creation

LOGIN creates a four-byte segment table entry for each page table generated. The segment table entry contains the length and address of its corresponding page table. The address of the segment table (aligned on a 64-byte boundary) is placed in the UTABLE.

The relationship of the virtual storage addresses to the segment table and page tables is illustrated in Figure 12. The twelve low-order bits of the address provide addressability for 4K bytes of storage (one page); this number is used as a displacement from the beginning of the page, as defined by the page table entry. The next eight bits of the address provide addressability for 1024K bytes of storage (one segment); this number is used to find the appropriate page by providing a displacement from the beginning of the page table (the beginning of a segment is the address of the first page in the segment). The four high-order bits of the address provide addressability for 4096K bytes of storage; this number is used to find the appropriate segment by providing a displacement from the beginning of the segment table.

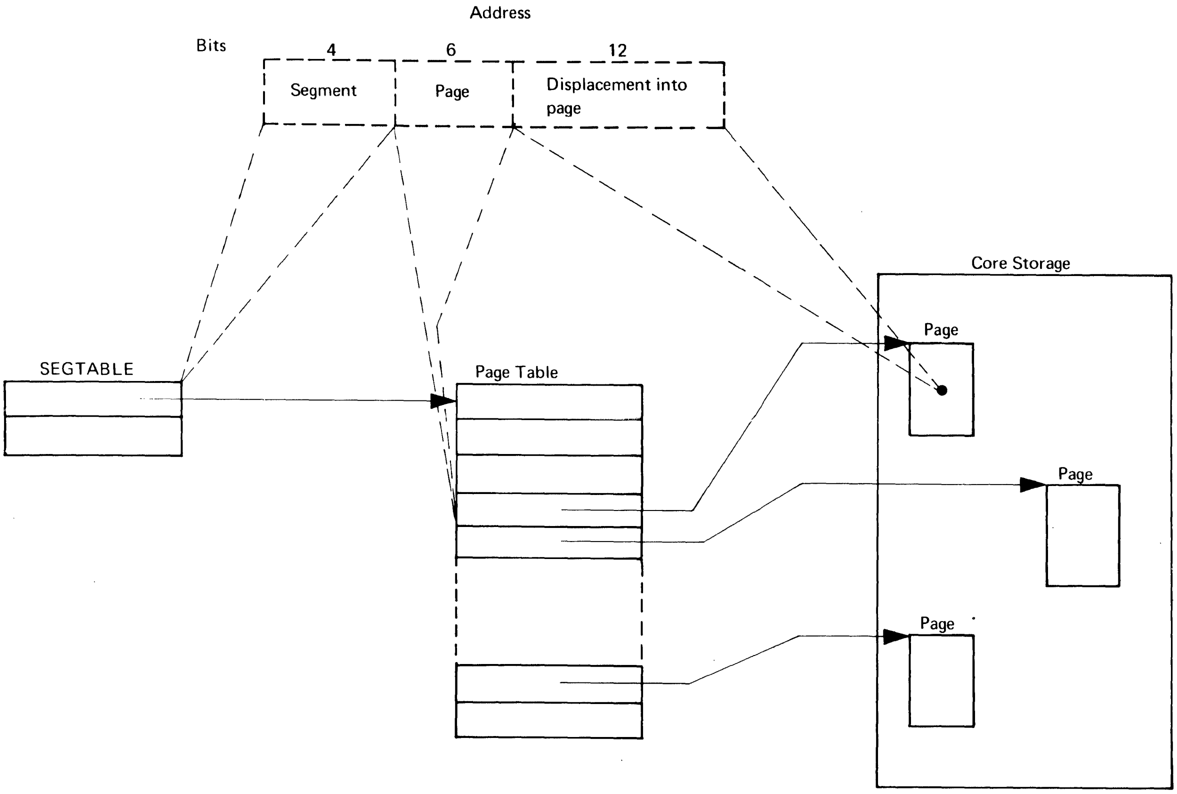


FIGURE 12. Virtual Addressing

Swap Table Creation

For each page table entry, LOGIN creates a corresponding eight-byte entry in a swap table (SWPTABLE). Whereas a page table entry contains the address of a page when it is core resident, a swap table entry contains the DASD address of a page when it is not core resident. The DASD address is contained in bytes 4-7 of the swap table entry; bytes 0-3 contain control information.

Virtual I/O Block Creation

When page and swap table creation is completed, LOGIN reads entries for I/O devices from the user's machine description file. After determining the channel type (selector or multiplexer), LOGIN creates the required virtual I/O blocks. Figure 8 illustrates the relationship of virtual and real I/O blocks.

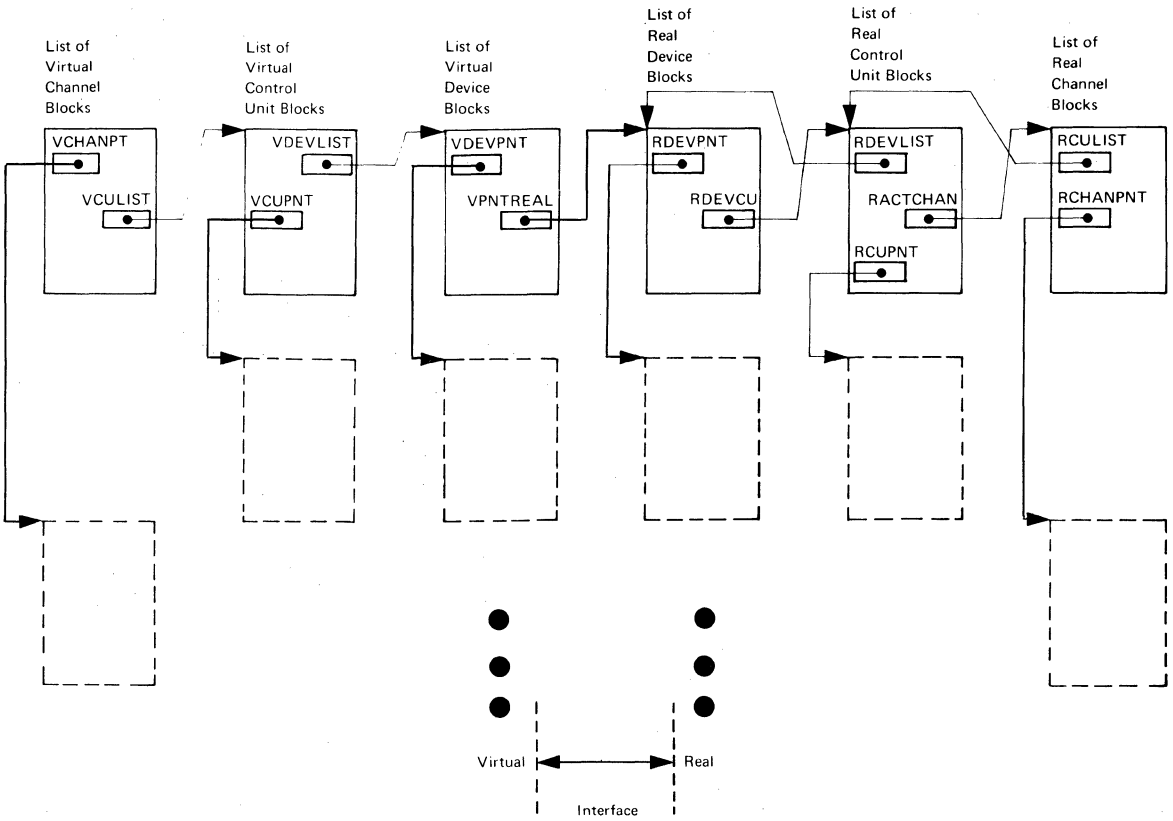

List of Virtual Channel Blocks
List of Virtual Control Unit Blocks
List of Virtual Device Blocks
List of Real Device Blocks
List of Real Control Unit Blocks
List of Real Channel Blocks
VCHANPT
VCULIST
VDEVLIST
VCUPNT
VDEVPNT
VPNTREAL
RDEVPNT
RDEVCU
RDEVLIST
RACTCHAN
RCUPNT
RCULIST
RCHANPNT
Virtual
Real
Interface

FIGURE 13. Virtual-Real I/O Blocks

For multiplexer devices, a new virtual multiplexer device block (MVDEBLOK) is created and chained to the last created MVDEBLOK. The address of the first MVDEBLOK in the chain (the MVDEBLOK for the user's terminal) is entered into the UTABLE.

For devices attached to selector channels, a virtual device block is created, and, if necessary, control unit and channel blocks.

A pointer to each virtual I/O block that is created is entered in the previous block, resulting in a chain (list) of virtual I/O blocks. Virtual device blocks are also chained to corresponding real device blocks (see Figure 13).

LOGON determines the right of access to a virtual DASD device based on information contained in the machine description entry of the user directory.

These rights of access are summarized in Table 1. The normal mode of access to a DASD device is read/write. In general, unless overridden by the presence of WRMULT, only one user can access a DASD device with write privileges. Any number of users can have simultaneous read-only access. The WRMULT parameter results in existing links being ignored. The use of WRMULT requires that the virtual machine operating system contain the proper data set protection mechanisms; in addition, CMS does not have interlocks. Therefore, WRMULT should be used with caution.

See the CP-67 Operator's Guide under "Directory Creation and Allocation".

Table 1. Summary of Access Allowed to DASD Devices by LOGIN

Directory Specification		Existing Links to Other Virtual Machines	Access Mode Allowed	Messages (see below)
RDONLY	WRMULT			
No	No	None	Read/Write	
		Read-only	Read-only	1
		Read/Write	None	2
Yes	No	None	Read-only	
		Read-only	Read-only	
		Read/Write	None	2
No	Yes	None	Read/Write	
		Read-only	Read/Write	3
		Read/Write	Read/Write	3
Yes	Yes	None	Read-only	
		Read-only	Read-only	
		Read/Write	Read-only	

1. DEV XXX IN USE BY userid; SET TO R/O
2. DEV XXX IN USE BY userid; NOT ATTACHED
3. DEV XXX IN USE BY userid

User Accounting Statistics

In the UTABLE for each virtual machine, three fields are used for time accounting.

TIMEON is a six-byte field that contains the date and time in packed decimal of user login. This is used with logout time and is punched in the user accounting card to give connect time.

TIMEUSED is a fullword binary value that represents _all_ CPU time charged to this virtual machine. The time is in extended precision (high resolution) time units and includes both user execution time and CP supervisor time executed for this user.

VTOTTIME is the same as TIMEUSED except that it includes _only_ user CPU execution time.

In addition there are statistics for user I/O activity. These are:

VMSSIO - number of selector channel SIO
VMPNCH - number of virtual "cards" punched
VMLINS - number of virtual "lines" printed
VMCRDS - number of virtual "cards" read
VMPGRD - number of pages read

Also, there are four words reserved for user data gathering that may be used by the installation. These are:

VMUSER1, VMUSER2, VMUSER3, and VMUSER4

PROCESSING CONTROL PROGRAM I/O REQUESTS

Control Program requested input-output operations can be divided into two general categories: (1) those initiated by a user (virtual) I/O request, and (2) those initiated by the Control Program itself (for example, paging or spooling requests). The following text describes the routines called by the Control Program to perform specific I/O operations. Processing required to analyze virtual I/O requests and to translate them to specific real operations is discussed later in this section under "Processing User Selector Channel I/O Requests" and "Processing User Multiplexer Channel I/O Requests". See Figure 14 for a flowchart of I/O Interrupt Handler operation.

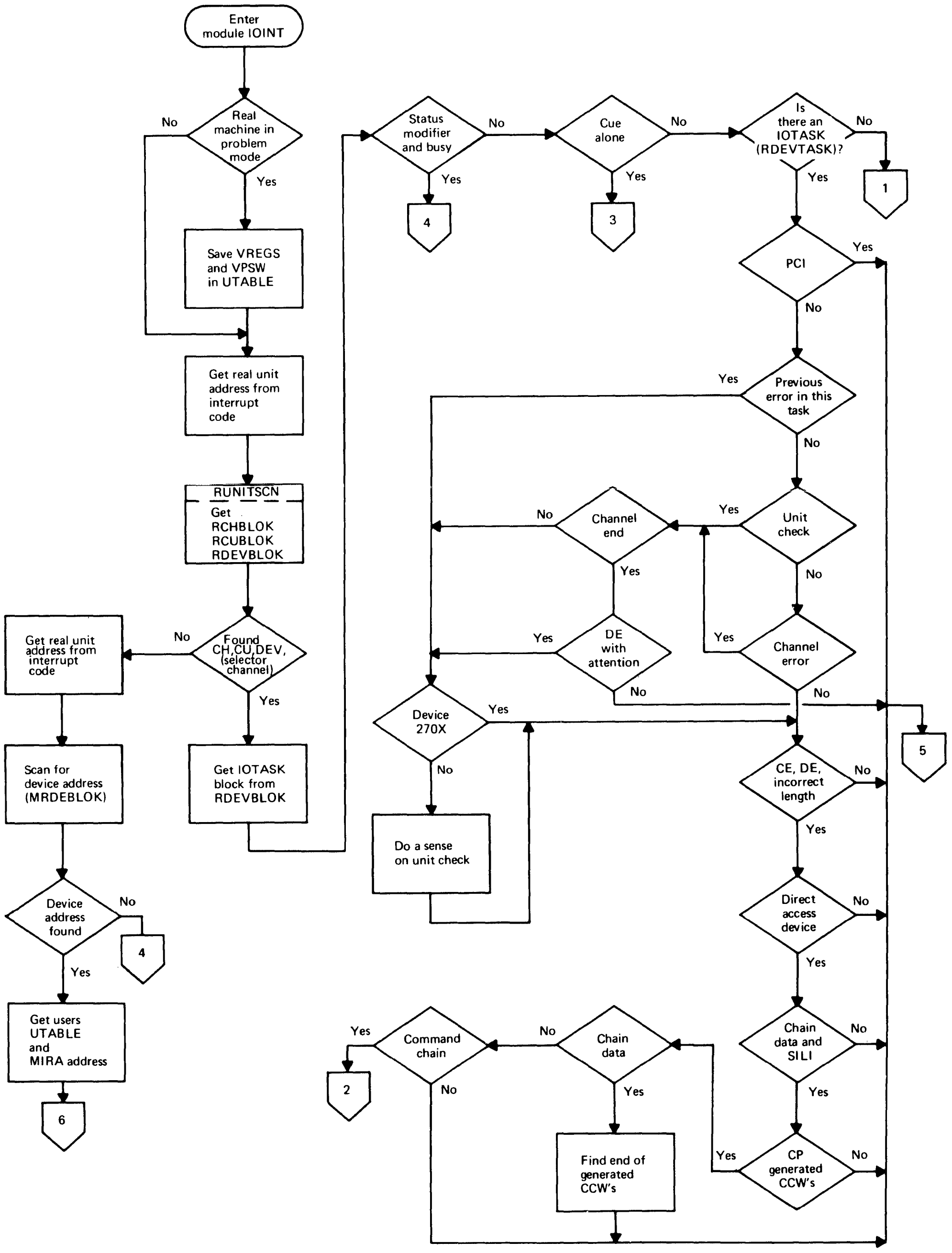

Enter module IOINT
Real machine in problem mode
No
Yes
Save VREGS and VPSW in UTABLE
Get real unit address from interrupt code
RUNITSCN
Get RCHBLOK RCUBLOK RDEVBLOK
Found CH,CU,DEV, (selector channel)
Get real unit address from interrupt code
Scan for device address (MRDEBLOK)
Device address found
4
Get users UTABLE and MIRA address
6
Get IOTASK block from RDEVBLOK
Status modifier and busy
4
Cue alone
3
Is there an IOTASK (RDEVTASK)?
1
PCI
Previous error in this task
Unit check
Channel end
DE with attention
Channel error
Device 270X
Do a sense on unit check
5
CE, DE, incorrect length
Direct access device
Chain data and SILI
CP generated CCW's
Chain data
Find end of generated CCW's
Command chain
2

Figure 14. Flowchart of I/O Interrupt Handler Operation (1 of 2)

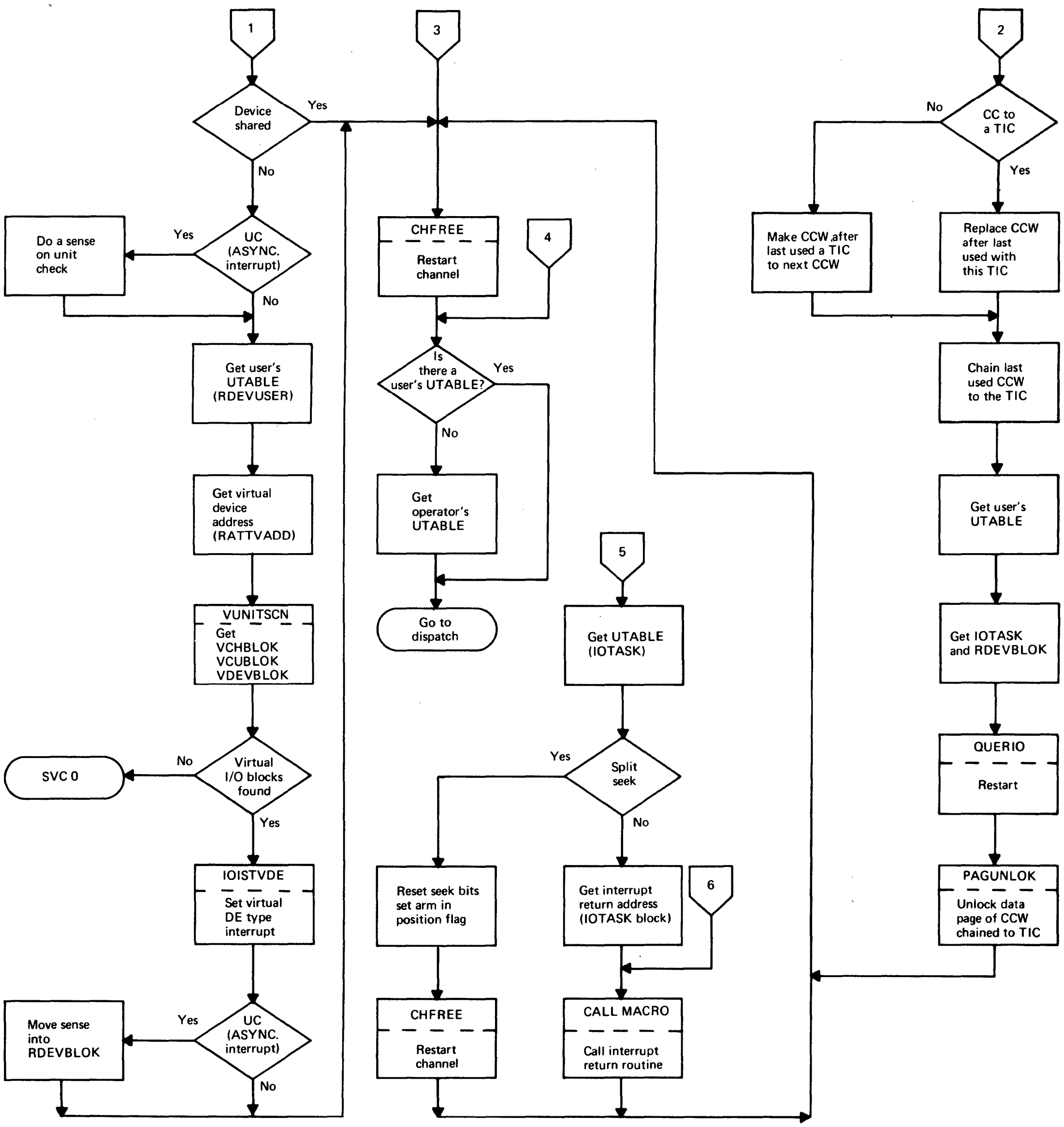

1
Device shared
Yes
No
Do a sense on unit check
Yes
UC (ASYNC. interrupt)
No
Get user's UTABLE (RDEVUSER)
Get virtual device address (RATTVADD)
VUNITSCN
Get VCHBLOK VCUBLOK VDEVBLOK
SVC 0
No
Virtual I/O blocks found
Yes
IOISTVDE
Set virtual DE type interrupt
Move sense into RDEVBLOK
Yes
UC (ASYNC. interrupt)
No
3
CHFREE
Restart channel
4
Is there a user's UTABLE?
Yes
No
Get operator's UTABLE
Go to dispatch
5
Get UTABLE (IOTASK)
Yes
Split seek
No
Reset seek bits set arm in position flag
CHFREE
Restart channel
Get interrupt return address (IOTASK block)
6
CALL MACRO
Call interrupt return routine
2
No
CC to a TIC
Yes
Make CCW after last used a TIC to next CCW
Replace CCW after last used with this TIC
Chain last used CCW to the TIC
Get user's UTABLE
Get IOTASK and RDEVBLOK
QUERIO
Restart
PAGUNLOK
Unlock data page of CCW chained to TIC

Figure 14. Flowchart of I/O Interrupt Handler Operation (2 of 2)

Real Multiplexer Channel I/O Operations

The multiplexer real I/O executive (MRIOEXEC) is entered whenever an interruption occurs on a unit record device (printer, card reader, or card punch) attached to a multiplexer channel. It is also called by the multiplexer virtual I/O executive routine (MVIOEXEC) to perform printer or punch input-output operations. MRIOEXEC determines the interrupting device type and performs appropriate processing. See Figure 15 for processing in the MRIOEXEC module.

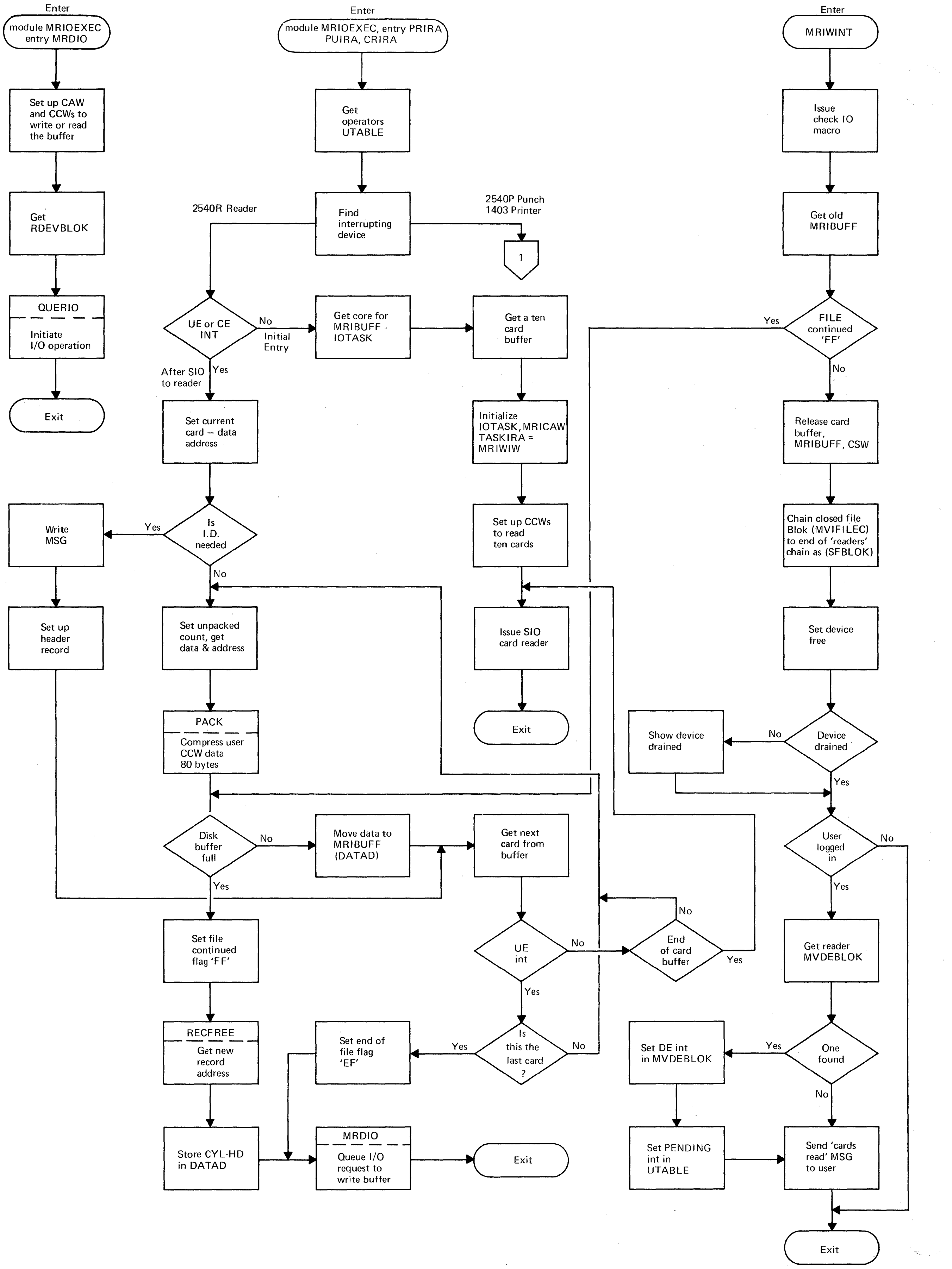


Figure 15. Processing in the MRIOEXEC Module (1 of 2)

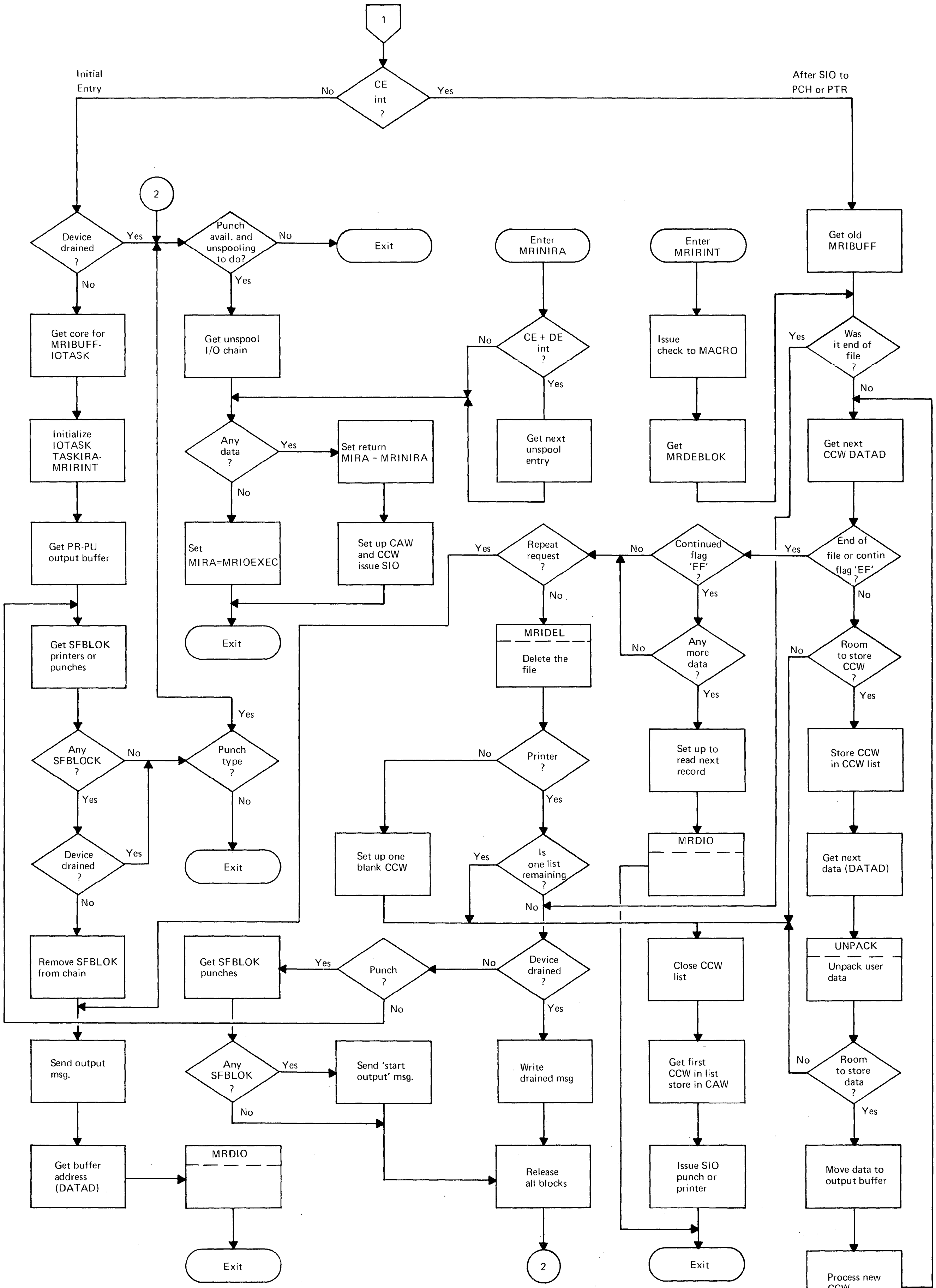


Figure 15. Processing in the MRIOEXEC Module (2 of 2)

Card Reader Interruption

To perform I/O operations on a card reader, MRIOEXEC reads card data into a buffer (ten cards at a time), compresses the data (by means of the PACK routine), and writes the packed records into a "spooling" file on a direct access device. The records will later be read from the spooling file by MVIOEXEC.

If MRIOEXEC is entered as the result of an interruption caused by the unit being made ready (that is, initial entry into the routine), the routine obtains an input buffer and a spooling buffer, constructs a CCW list to read from the card reader, and issues an SIO instruction.

If the interruption results from a channel end or a unit exception, MRIOEXEC calls PACK to compress the input data, and moves the packed data to the spooling buffer. When the buffer is full, or at end-of-file, it creates an I/O task block and a CCW list to write the buffer to a spooling file on a direct access device. The routine QUERIO is called to attach the task block to the appropriate channel block and schedule it for service.

When the buffer has been written to the spooling file, a test is made for an end-of-file indication (set when a unit exception interruption occurred, indicating that all cards have been read). If the end-of-file flag is on, buffers are returned to free storage, and the file is added to the chain of closed files. Reader files are chained off the READERS word in MRIOEXEC.

Printer or Punch Interruption

To perform I/O operations on a printer or card punch, MRIOEXEC reads records from a spooling file on a direct access device, unpacks the data (by means of the UNPACK routine), and prints or punches the records on the specified device.

If MRIOEXEC is entered as the result of an interruption caused by the unit being made ready (that is, initial entry into the routine), the routine obtains an I/O task block for reading records from a spooling file on a direct access device and a buffer area into which these records may be read. Printer and punch processing check the PRINTERS and PUNCHES chain respectively to locate a closed file entry (spool file control block). PRINTERS and PUNCHES are words in MRIOEXEC.

If a closed file is available, a message indicating the output device is written to the system operator's console by calling the routine WRTCONS. A CCW list for reading records from the file is created, the I/O task block is initialized, and the routine QUERIO is called to attach and schedule the task block to the appropriate channel queue.

When records have been read from the spooling file, the routine UNPACK is called to unpack the spooled records, the unpacked records are moved to an output buffer, and the next group of spooled records is read. When the output buffer is filled, or when the spooling file has been completely read (logical end-of-file encountered), an SIO instruction is issued for the appropriate device (printer or punch).

When a file has been completely written out, or if no closed spooling file was available, MRIOEXEC processes requests for unspooled punch output. Unspooled punch output requests are initiated by the Control Program (typically for accounting information cards) and are added to a MREALIO queue by RPUNCH, a subroutine within MRIOEXEC.

Real Terminal I/O Operations

The routines used by the Control Program to communicate with either the real operator's console or a remote terminal are RDCONS for read operations and WRTCONS for write operations. RDCONS and WRTCONS prepare CCW lists and I/O task blocks for their respective I/O operations, and call STCONSIO to stack and initiate the I/O requests. The console interruption handler (CONSINT) receives control when the I/O operation is completed.

Read From a Terminal - RDCONS

See Figure 16 for processing in RDCONS module.

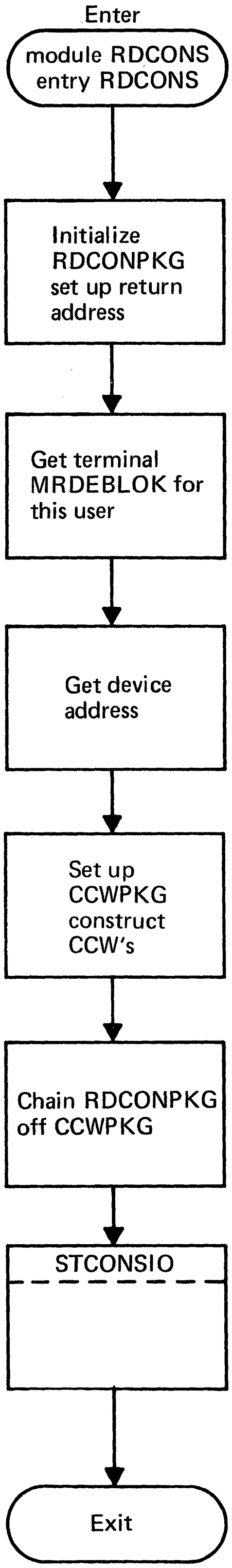

Enter
module RDCONS entry RDCONS
Initialize RDCONPKG set up return address
Get terminal MRDEBLOK for this user
Get device address
Set up CCWPKG construct CCW's
Chain RDCONPKG off CCWPKG
STCONSIO
Exit

Figure 16. Processing in RDCONS Module

When a read operation from a terminal is required, the Control Program calls RDCONS, passing in register 1 the address of a 132 byte input buffer, and, if required, in register 2 the parameters for the EDIT and/or UCASE options. EDIT and UCASE options, if requested, are processed by the console interruption handler, CONSINT.

RDCONS obtains storage for and initializes a control list for the read operation. The appropriate I/O device block (MRDEBLOK) is initialized. If the data is to be read from the real operator's console, the current operator's MRDEBLOK is used; otherwise, the address of the MRDEBLOK is obtained from the indicated user's virtual console MVDEBLOK.

An appropriate CCW list is constructed for the type of terminal device, and the address of the CCW list is placed in register 6. The EDIT and/or UCASE parameters, if present, and the device type are placed in the control list, and the routine STCONSIO is called. When control is eventually returned to RDCONS upon completion of the read function, an exit is taken to the calling routine.

Write to a Terminal - WRTCONS

See Figure 17 for WRTCONS module processing.

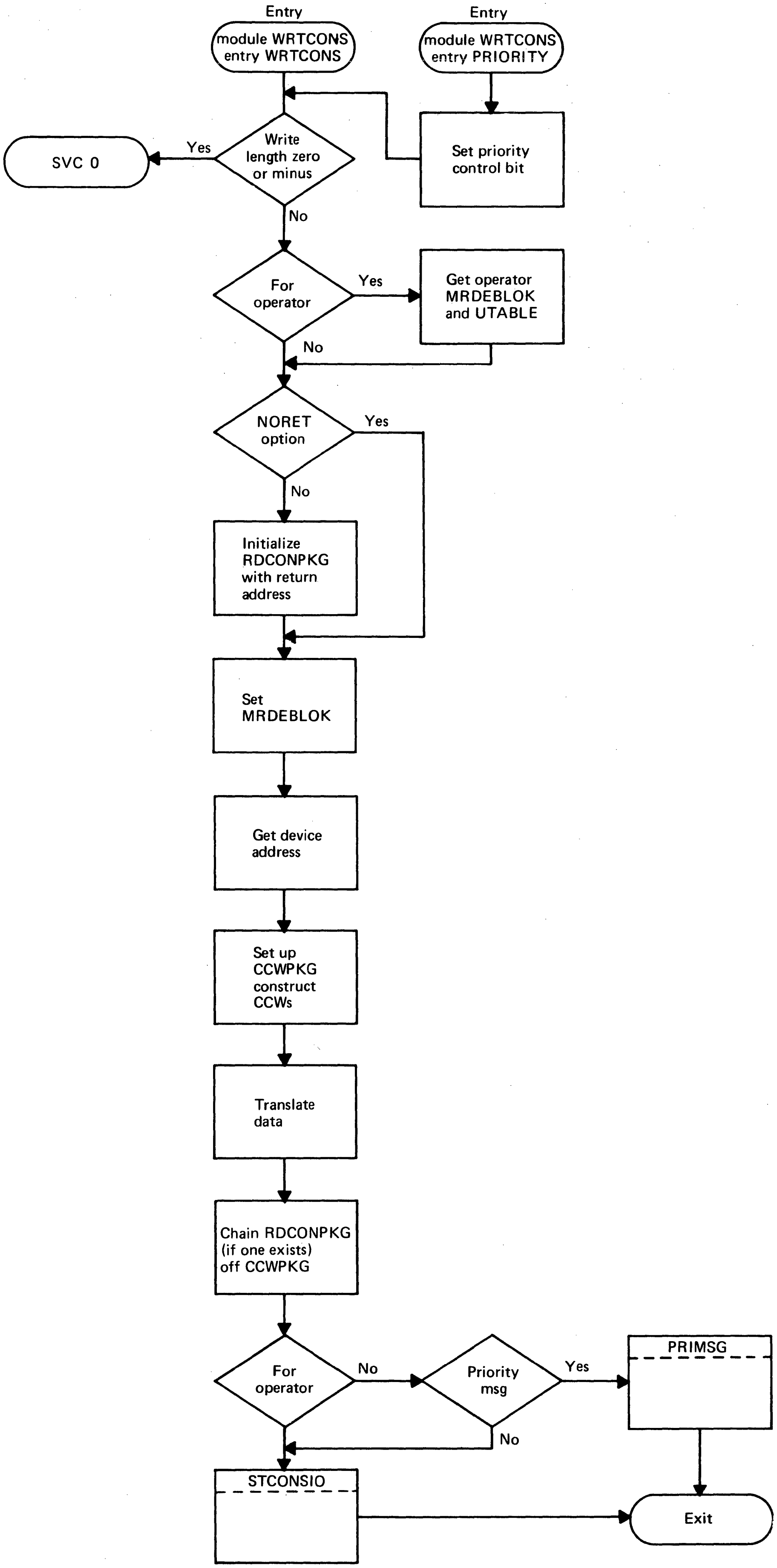

Entry
module WRTCONS entry WRTCONS
Entry
module WRTCONS entry PRIORITY
Set priority control bit
Write length zero or minus
Yes
SVC 0
No
For operator
Yes
Get operator MRDEBLOK and UTABLE
No
NORET option
Yes
No
Initialize RDCONPKG with return address
Set MRDEBLOK
Get device address
Set up CCWPKG construct CCWs
Translate data
Chain RDCONPKG (if one exists) off CCWPKG
For operator
No
Priority msg
Yes
PRIMSG
No
STCONSIO
Exit

Figure 17. WRTCONS Module Processing

When a write operation to a terminal is required, the Control Program calls WRTCONS, passing the following information in the indicated registers:

GPR 0 - the number of bytes in the output message;

GPR 1 - the location of the first byte of the output message;

GPR 2 - the parameters for the NORET, DFRET, OPERATOR, NOAUTO, and ALARM options;

GPR 11 - the appropriate user's UTABLE address.

Unless the NORET option was specified, WRTCONS obtains storage for and initializes a control list in which will be saved the return address and register contents. The appropriate I/O device block (MRDEBLOK) is initialized. If the message is to be written to the real operator's console, the current operator's MRDEBLOK is used; otherwise, the address of the MRDEBLOK is obtained from the user's UTABLE entry.

An appropriate CCW list is constructed for the type of terminal device being used and for the option. Option parameters, passed to WRTCONS in register 2, are stored in a control list preceding the CCW list.

The address of the CCW package (CCW list and control list) is placed in register 6, the device type and parameters for the DFRET option, if present, are stored in the control list, and the routine STCONSIO is called. When control is returned to WRTCONS, an exit is taken to the calling routine.

Two alternate entry points, PRIORITY and CLRCONS, are provided for the WRTCONS routine. If the routine is entered at PRIORITY, write requests will be created as usual, except that the STCONSIO routine will be entered at PRIMSG, causing the write request to be stacked on a priority basis. If the routine is entered at CLRCONS, all outstanding terminal I/O requests to that user will be deleted.

Stack or Start Terminal I/O Requests - STCONSIO

| See Figure 18 for STCONSIO module processing.

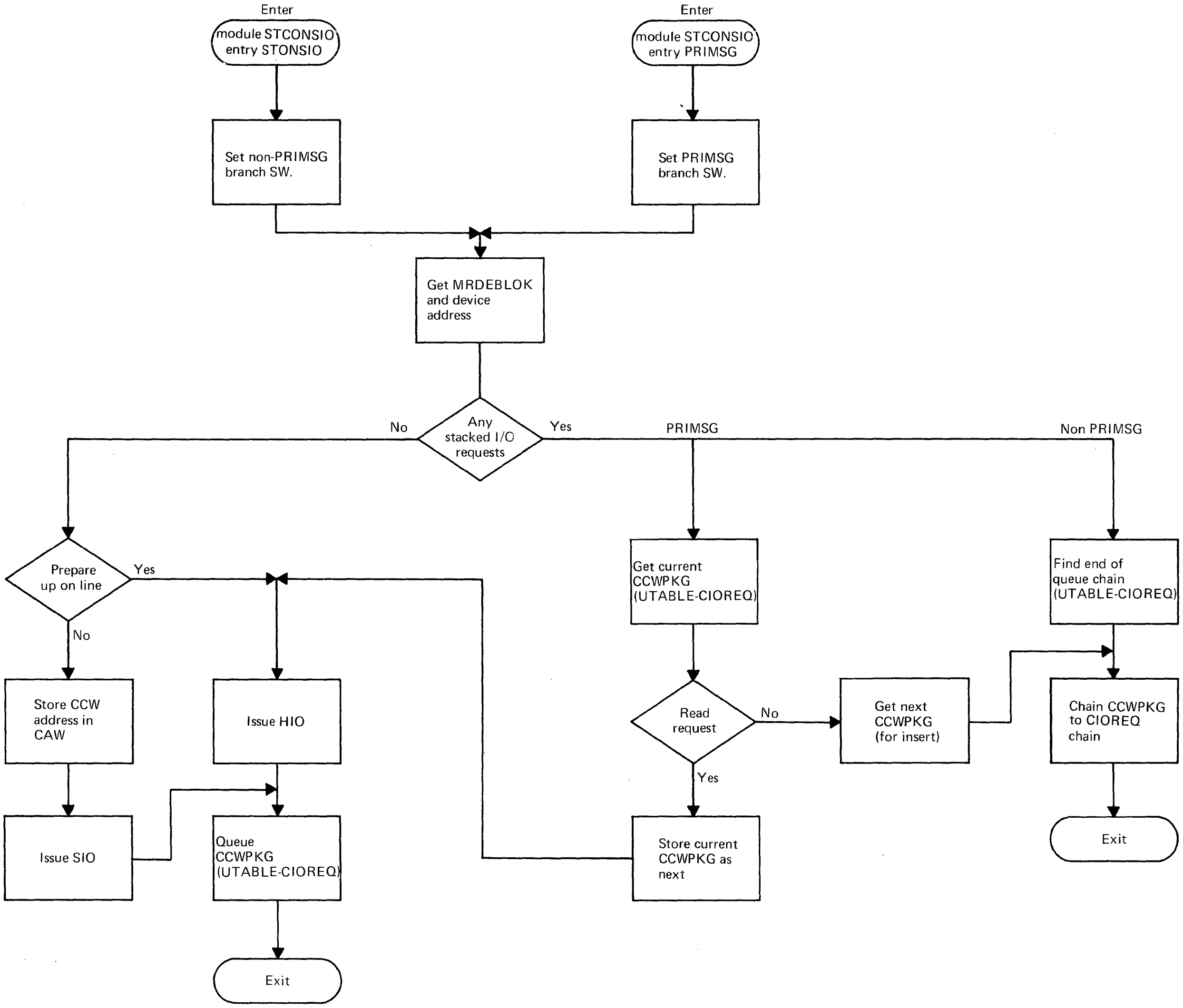

Enter
module STCONSIO entry STONSIO
Enter
module STCONSIO entry PRIMSG
Set non-PRIMSG branch SW.
Set PRIMSG branch SW.
Get MRDEBLOK and device address
Any stacked I/O requests
No
Yes
PRIMSG
Non PRIMSG
Prepare up on line
Yes
No
Store CCW address in CAW
Issue HIO
Issue SIO
Queue CCWPKG (UTABLE-CIOREQ)
Exit
Get current CCWPKG (UTABLE-CIOREQ)
Read request
No
Yes
Get next CCWPKG (for insert)
Store current CCWPKG as next
Find end of queue chain (UTABLE-CIOREQ)
Chain CCWPKG to CIOREQ chain
Exit

Figure 18. STCONSIO Module Processing

When a CCW package has been created for a terminal I/O operation, STCONSIO is called to add the I/O request to the chain of pending requests, or to start the operation if no other requests are pending. At entry to STCONSIO, register 6 contains the address of the CCW package, register 8 contains the device type, and register 11 contains the address of the appropriate user's UTABLE.

If no other I/O requests are pending, the address of the CCW package is placed in the channel address word and an SIO instruction is issued. When the I/O operation has been initiated, the current I/O request pointer is updated to point to the CCW package of the active operation, the count of pending I/O requests (NCIOREQ) is incremented by 1, and an exit is taken to the calling routine.

If other I/O requests are pending, the CCW package is added to the chain of pending requests, the count of pending requests is incremented by 1, and the exit is taken to the calling routine.

If the routine STCONSIO was entered at the entry point PRIMSG, a priority operation has been requested. If other I/O requests are pending, the current CCW package is examined to determine the type of operation in progress. If the current operation is a read, an HIO instruction is issued, the priority CCW package becomes the current package (added at the top of the chain), and the CCW package of the halted operation becomes the "next" package (second on the chain). If the current operation is a write, no HIO is issued; the priority CCW package becomes the next package (inserted after the current package in the chain). In either case, the count of pending requests (NCIOREQ) is incremented, and an exit is taken to the calling routine.

Processing Terminal I/O Interruptions - CONSINT

When an I/O interruption occurs on a terminal, the I/O interruption handler, IOINT, receives control and determines the type of interrupting device, obtains the multiplexer interruption return address (MIRA) from the MRDEBLOK, and gives control to the terminal I/O interruption handler (CONSINT) at the entry point specified by MIRA.

For an interruption following an output operation, CONSINT performs the following processing:

> If the NORET option is not specified, the routine CPSTACK is called to add an entry for the current user to the stack of Control Program execution requests. This entry notifies the caller of WRTCONS of the completion of the operation.

If other terminal requests are pending for this device an SIO instruction is issued for the next CCW package, and pointers to the "current" and "next" CCW packages are updated.

Control is returned to the main control routine (DISPATCH).

For an interruption following an input operation, CONSINT performs the following processing:

Unless the terminal is a 1052, the message is translated into EBCDIC from line code.

If the EDIT option is specified, the input message is scanned, and deletions are made as required.

If the UCASE option is specified, the input message is translated to uppercase letters.

The routine CPSTACK is called to add an entry for the current user to his stack of Control Program execution requests. This entry notifies the calling Control Program routine of completion of the input operation.

If other terminal requests are pending for this device, an SIO instruction is issued for the next CCW package, and pointers to the "current" and "next" CCW packages are updated.

Control is returned to DISPATCH.

Real Selector Channel Operations

The routine QUERIO is called by the Control Program whenever a selector channel I/O operation is to be performed. The address of a completed I/O task block is passed to QUERIO in register 1. QUERIO indicates that the operation is being requested by the Control Program, attaches the task block to the appropriate channel, and tests to see whether the channel is free.

Initiating Selector Channel I/O

If QUERIO determines that the channel is free, the routine CHFREE is called, with the address of the appropriate channel block (RCHBLOK) passed in register 1. CHFREE issues an SIO instruction to the indicated channel. The resulting condition code is checked and appropriate action taken:

For a condition code of 0, the task block is attached to the real device block (RDEVBLOK), the task count is decremented, and control is returned, through QUERIO, to the routine which requested the I/O operation.

For a condition code of 1, CSW information is obtained, the condition code is placed in register 0, and control is passed to the routine specified in the task interruption address (TASKIRA).

For a condition code of 2, a retry of the SIO instruction is issued.

For a condition code of 3, the task block is unchained from the channel, the task count is decremented, the condition code is placed in register 0, and control is passed to the routine indicated in TASKIRA.

Figure 19 shows the processing of I/O tasks on the selector channel and device blocks.

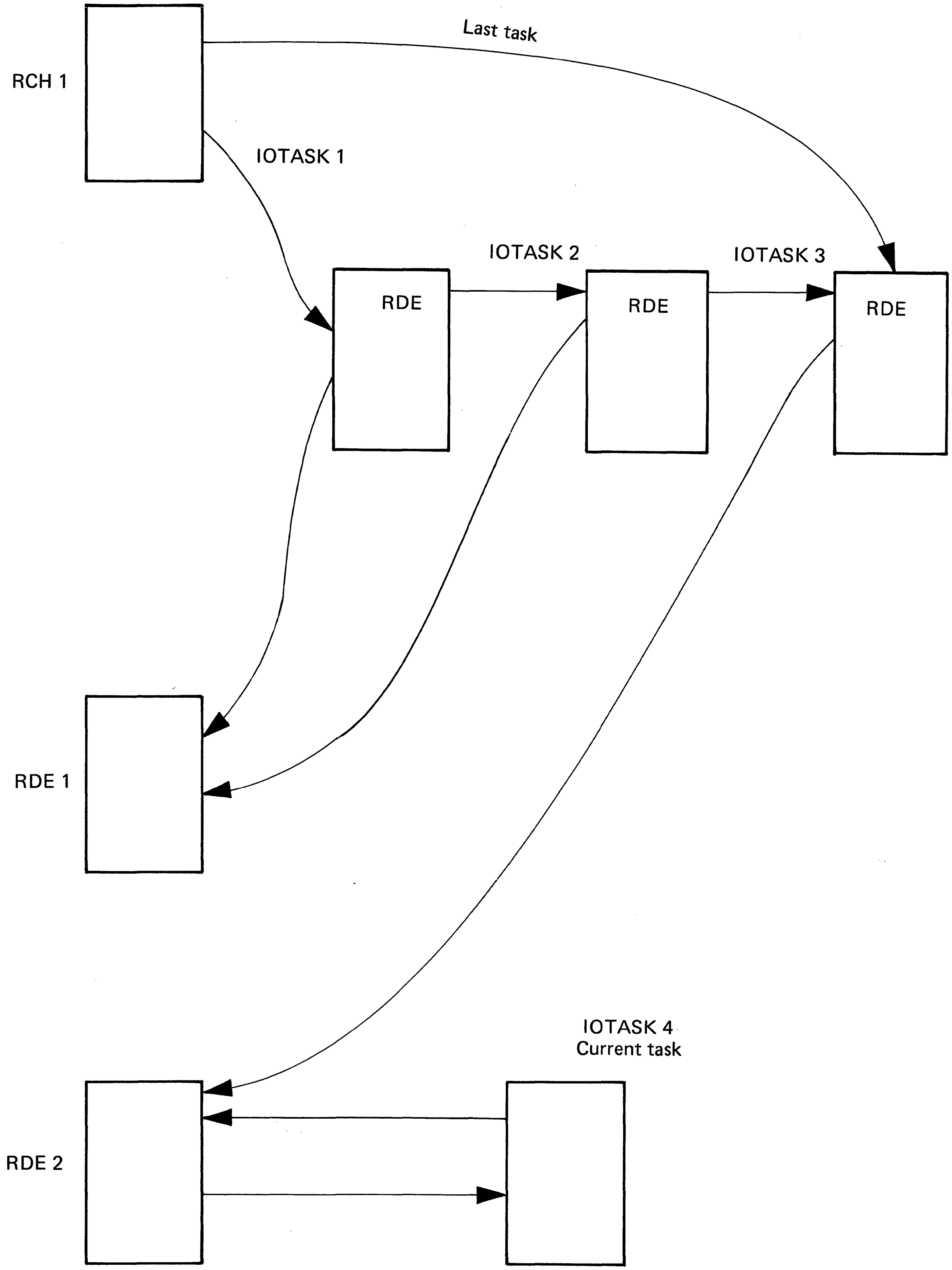


FIGURE 19. Processing Real Selector Channel I/O Tasks

Processing Selector Channel I/O Interruptions

When an I/O interruption occurs for a selector channel device, the I/O interruption handler, IOINT, receives control. Register 0 is cleared to indicate that an interruption has occurred, and control is given to the routine indicated in TASKIRA. When IOINT again receives control, control is passed to DISPATCH via a GOTO macro.

Processing of I/O Errors - IOERROR

When IOINT passes control to the routine whose address is indicated in TASKIRA, that routine issues a CHECKIO macro to check for successful completion of the I/O. If only the channel end and device end bits are set in the channel status word, the routine concludes that the I/O was successful and continues processing. In all other cases, IOERROR is called. When IOERROR receives control, a call is made to the subroutine RECERROR, which analyzes and, in some cases, records the error. (For details, see the subroutine description of RECERROR below.)

If the sense information indicates that intervention is required, a message is sent to the operator indicating the device address and asking "REPLY 'GO' WHEN AVAILABLE OR 'FAIL' IF NOT AVAILABLE". If the operator replies GO, the I/O operation is retried, whereas if the operator replies FAIL, a permanent error is assumed.

For CP-generated I/O (paging, spooling, and reading the directory), the I/O is retried up to 64 times if errors occur. This is accomplished by setting up a special retry I/O task consisting of a recalibrate CCW followed by a TIC to the original IOTASK block. TASKIRA is set up so that return is to the REPRTN entry point in IOERROR. If the I/O completes successfully, control returns to the program which originally generated the I/O request. If, on the other hand, the I/O is retried unsuccessfully 64 times, a major error message with error count, sense, and status information is printed at the operator's terminal and the system will ABEND.

Note that the error retry and recording procedure apply only to selector channel devices represented by RDEVBLOKS and not to shared unit record equipment or nondedicated terminals.

PROCESSING USER SELECTOR CHANNEL I/O REQUESTS

When a pseudo-supervisor (that is, a supervisor operating in a user's virtual machine) requests an I/O operation, a program interruption occurs, and the Control Program must determine the type of operation requested and the processing required to honor the request.

The following text describes the major routines involved in honoring user selector channel input-output requests. Only the I/O-related operations of the routines will be discussed in this section. See Figure 14, CP I/O Interrupt Handler.

Program Interruption Handler - PROGINT

Entrance: PROGINT receives control when a program interruption occurs.

Operation: PROGINT determines the mode of the user's virtual machine (problem or supervisor) and the cause of the program interruption (paging request, invalid operation, or privileged operation).

Routines Called: If the program interrupt is caused by a privileged operation that is in virtual supervisor mode, PROGINT transfers PRIVLGED to simulate it.

Privileged Instruction Simulator - PRIVLGED

Entrance: PRIVLGED receives control via a GOTO from PROGINT.

Operation: For other than I/O instructions, simulation is performed within PRIVLGED. PAGTRANS is called to bring in pages not in core that are necessary for the privileged instruction simulation. When simulation is finished, exit is taken via GOTO to DISPATCH.

If the privileged operation is an input-output request, PRIVLGED calls the virtual machine I/O executive program (VIOEXEC), passing the addresses of the first and second halves of the privileged operation in registers 4 and 5 respectively. When control is returned from VIOEXEC, an exit is taken to the main dispatcher and control routine (DISPATCH), via a GOTO macro instruction.

Virtual Machine I/O Executive Program - VIOEXEC

(See Figure 20 for VIOEXEC module processing.)

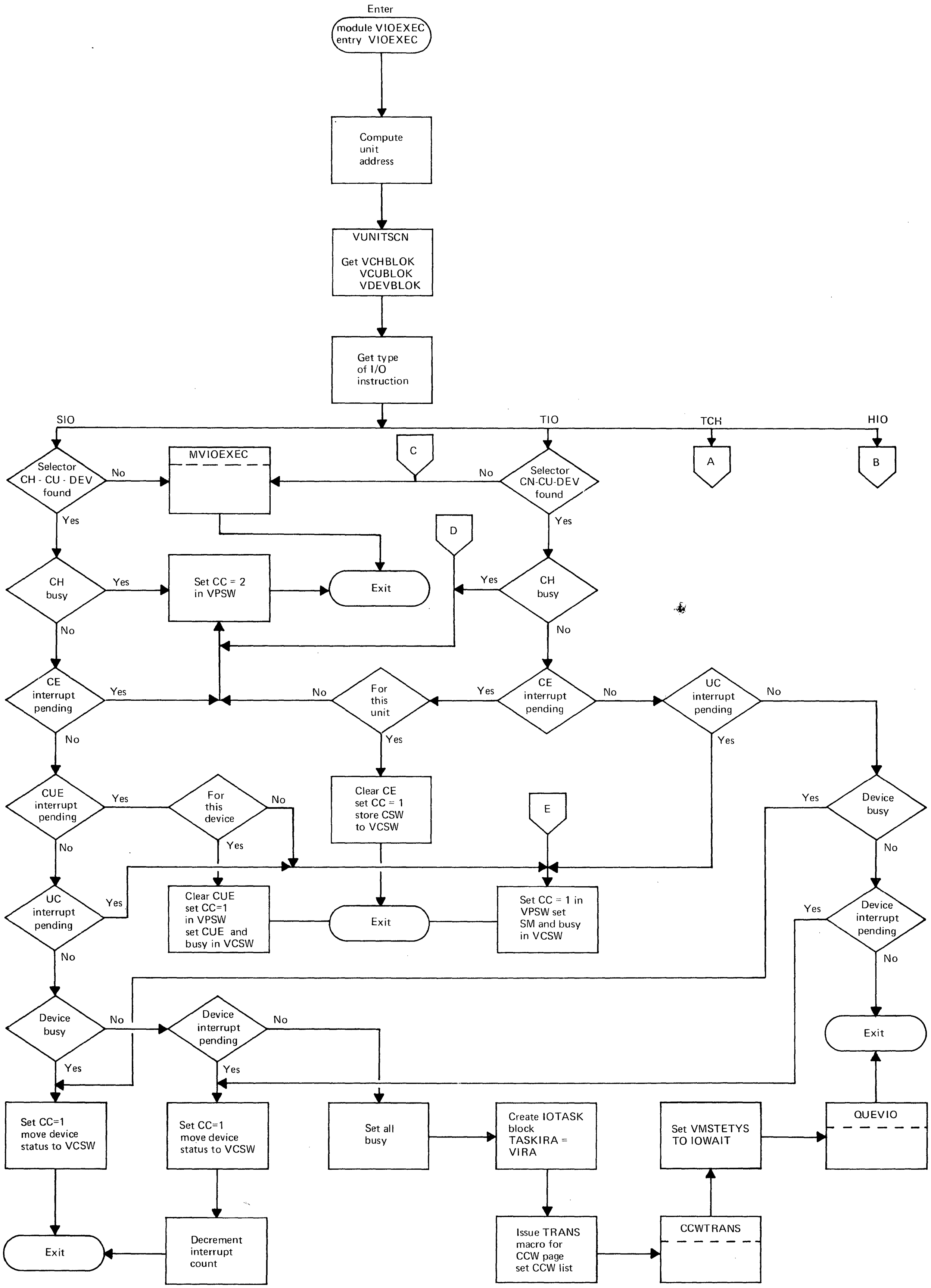


Figure 20. VIOEXEC Module Processing (1 of 4)

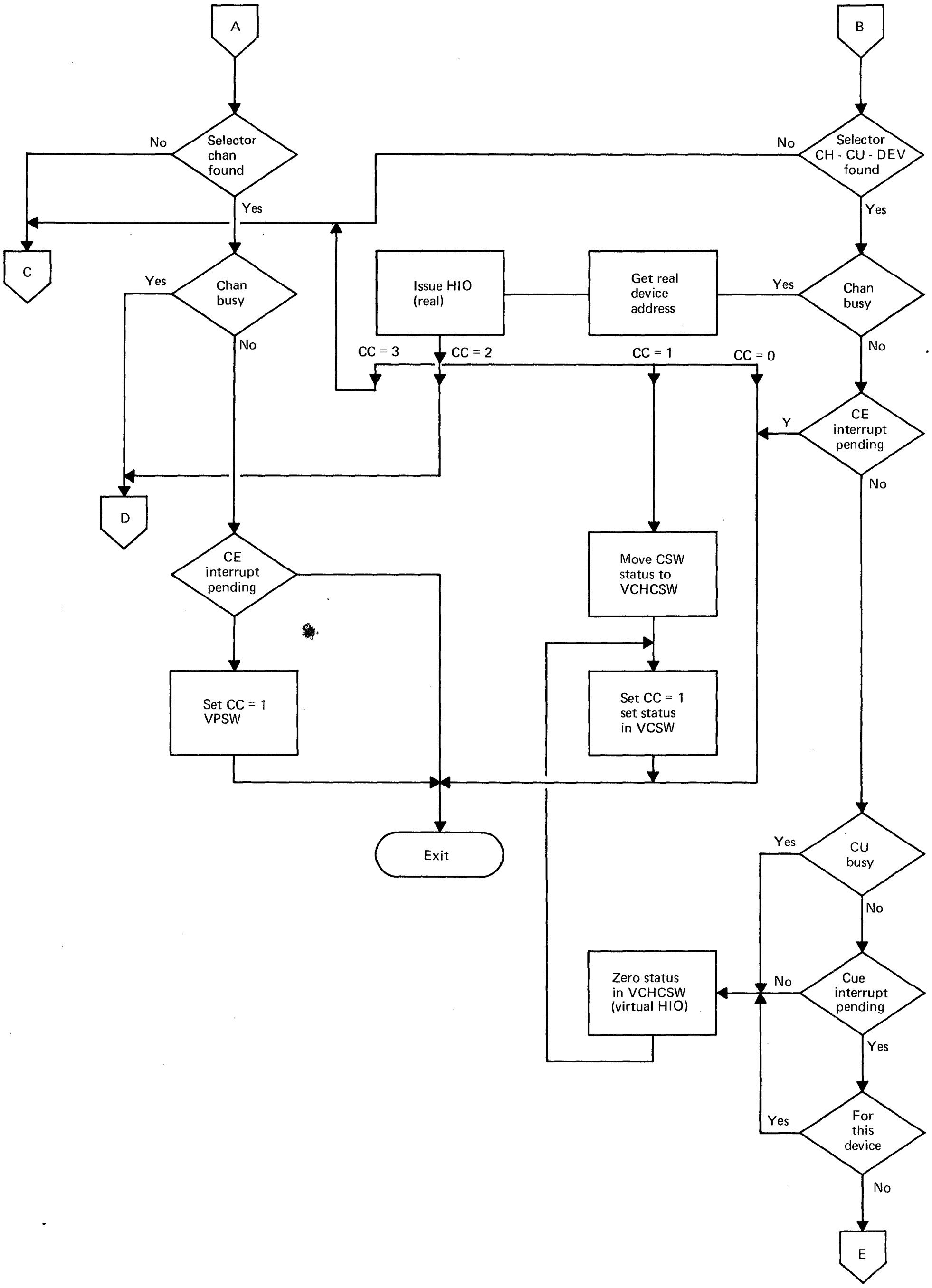


Figure 20. VIOEXEC Module Processing (2 of 4)

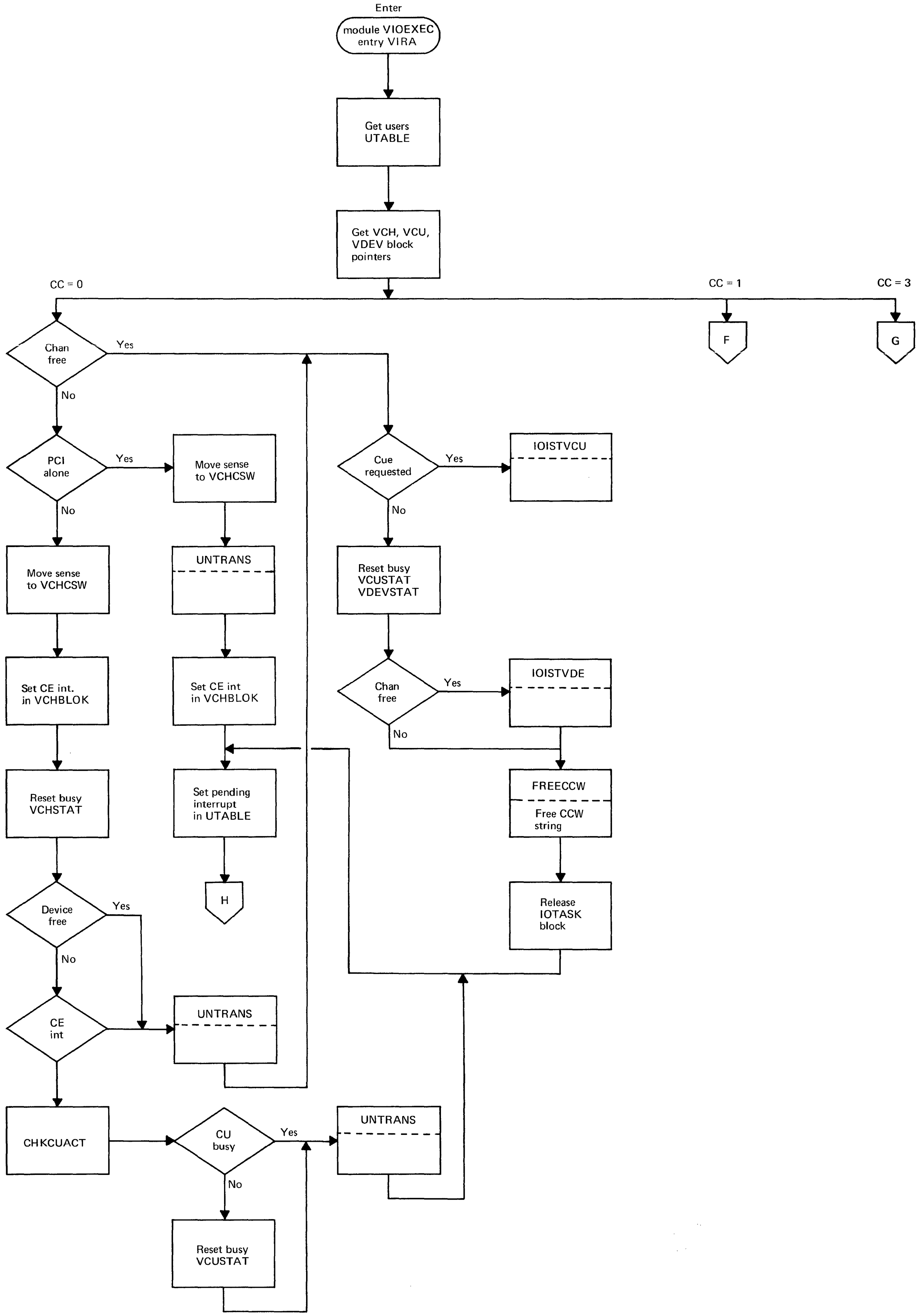


Figure 20. VIOEXEC Module Processing (3 of 4)

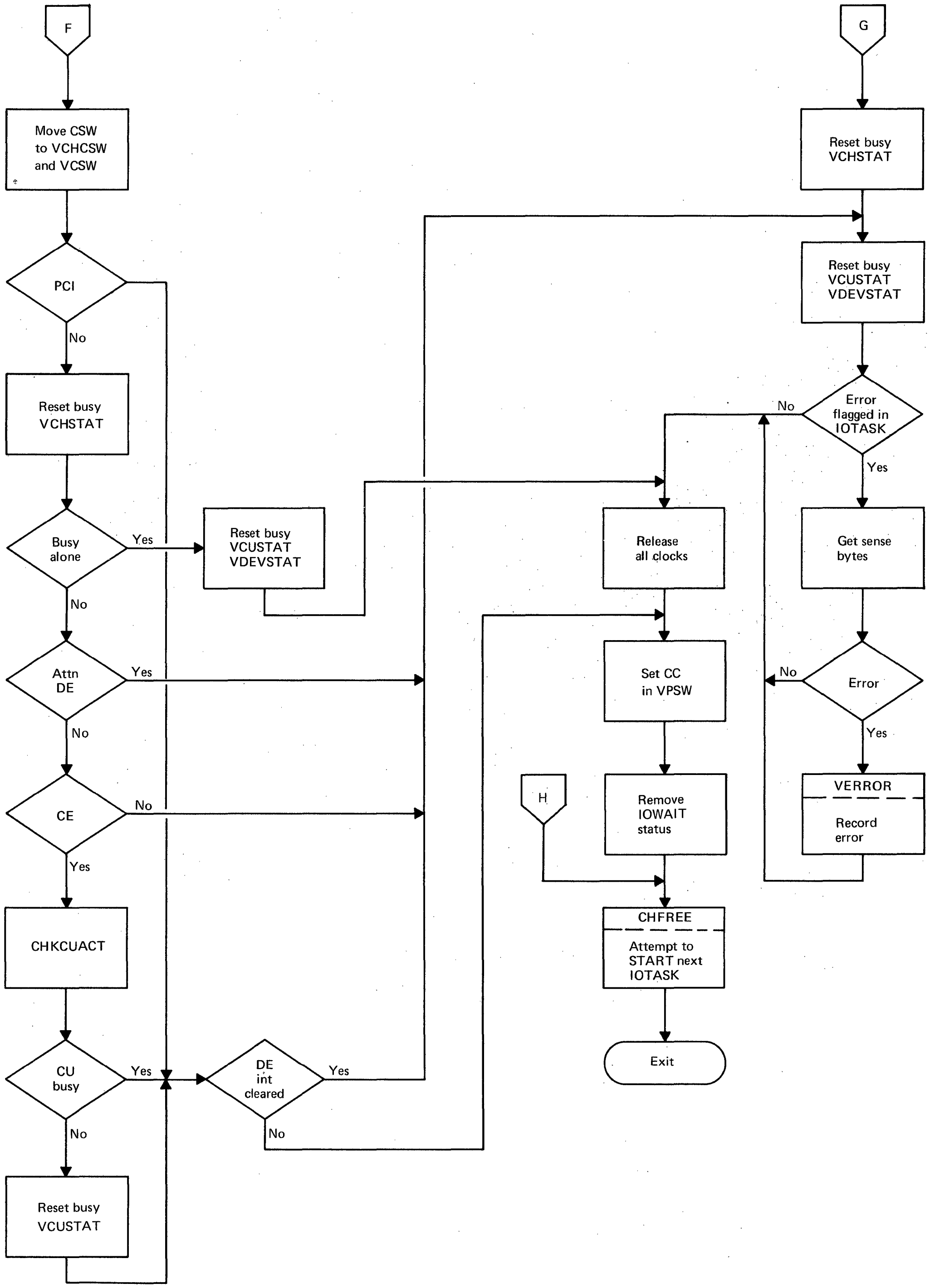


Figure 20. VIOEXEC Module Processing (4 of 4)

Entrance: VIOEXEC receives control from the privileged operation simulator (PRIVLGED) when a user-requested I/O operation has caused a program interruption.

Operation: VIOEXEC determines the type of I/O operation to be executed (SIO,TIO,HIO,TCH) and performs appropriate processing for each type.

For an SIO operation on a selector channel, VIOEXEC:

Obtains the channel, control unit, and device addresses, and tests for busy or status pending conditions on the addressed path. If the addressed channel is busy, sets condition code 2 in the virtual PSW and exits. If status is pending or the virtual control unit or device is busy, stores the relevant CSW status, sets condition code 1 and exits.

If the path to the device is free, creates an I/O task block, translating the virtual channel address word (CAW) into a real CAW

Calls the CCW translator (CCWTRANS) to translate virtual CCW's to real CCW's, returning the address of the start of the chain (TASKCAW)

Sets the I/O wait indicator in the user's VMSTATUS in UTABLE

Calls the virtual I/O request queueing routine, QUEVIO, to queue the I/O task block on the appropriate channel

Transfers to DSPTCHB (DISPATCH).

When the I/O operation is started, QUEVIO reflects the condition code to the user, and resets the I/O wait indicator to zero

For an SIO operation on a multiplexer channel, VIOEXEC: (See Figure 21 for MVIOEXEC module processing.)

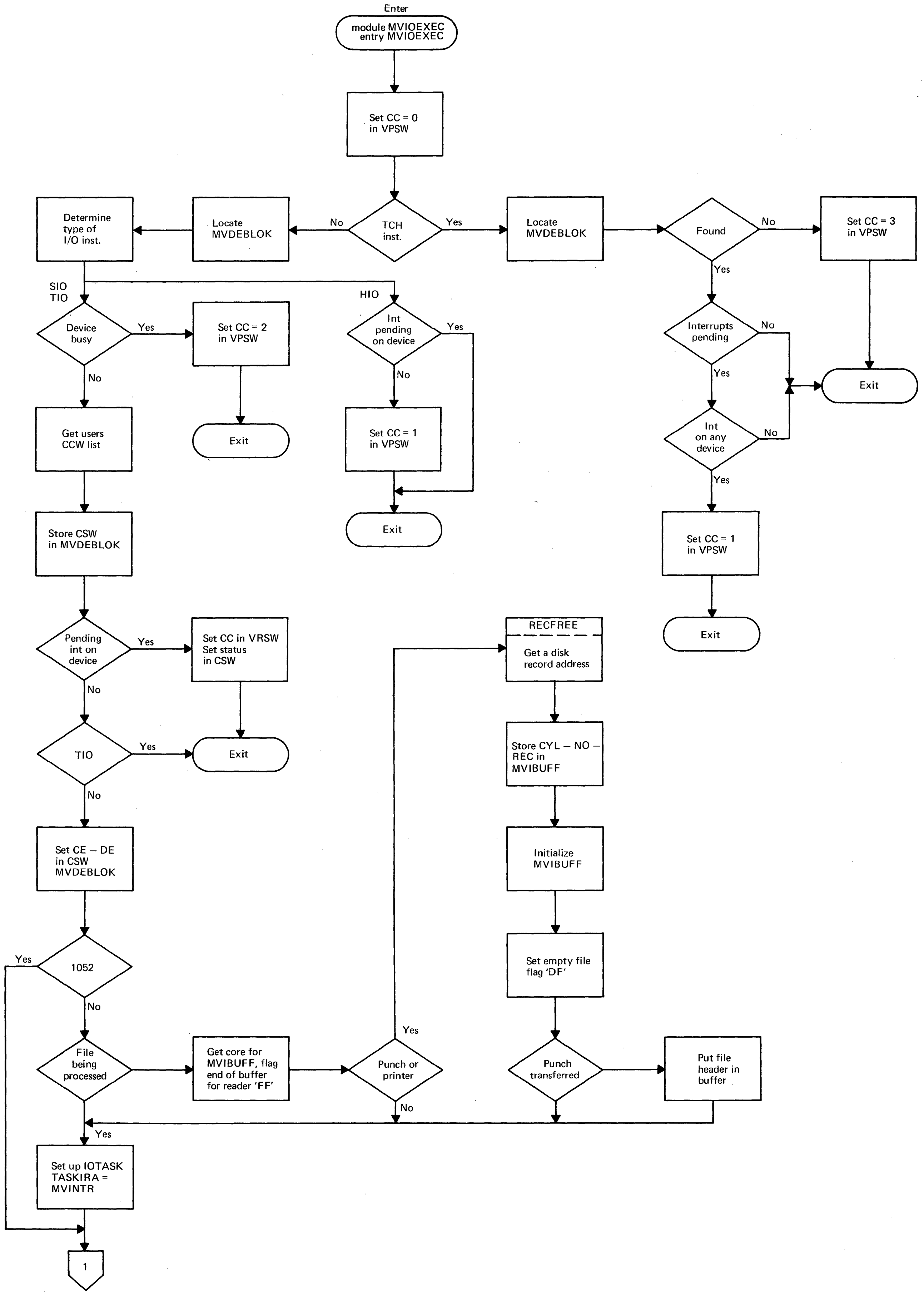


Figure 21. MVIOEXEC Module Processing (1 of 4)

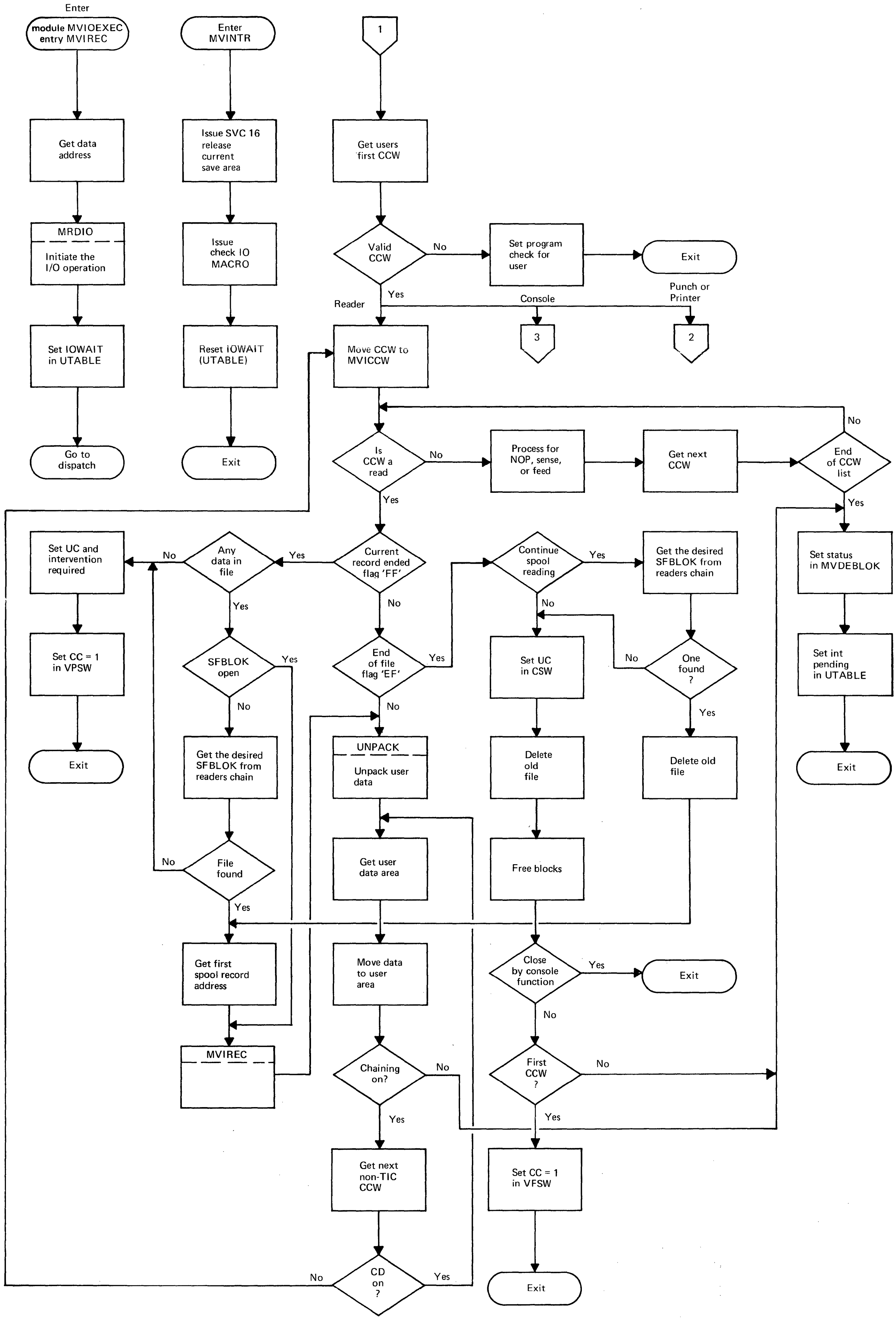


Figure 21. MVIOEXEC Module Processing (2 of 4)

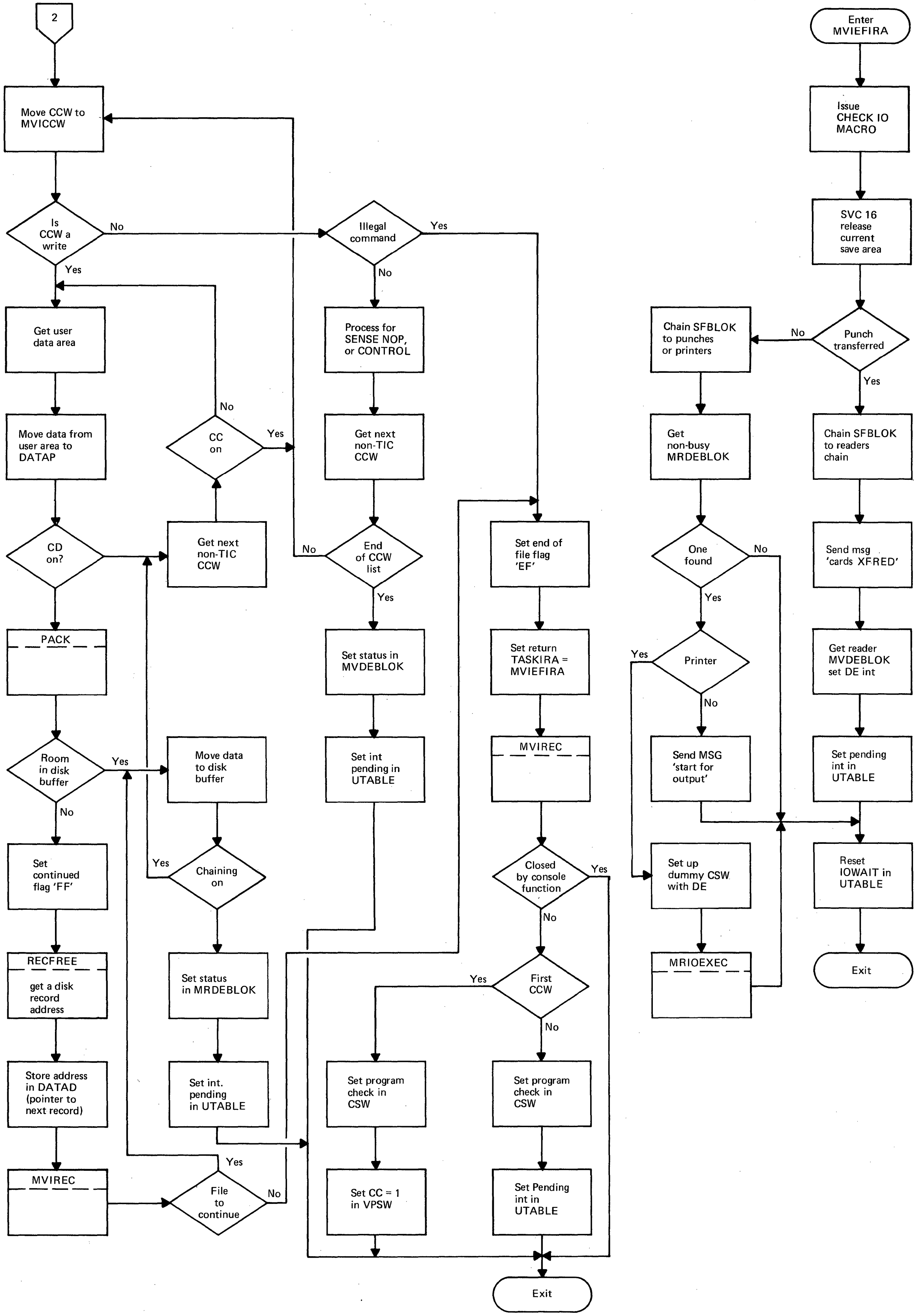


Figure 21. MVIOEXEC Module Processing (3 of 4)

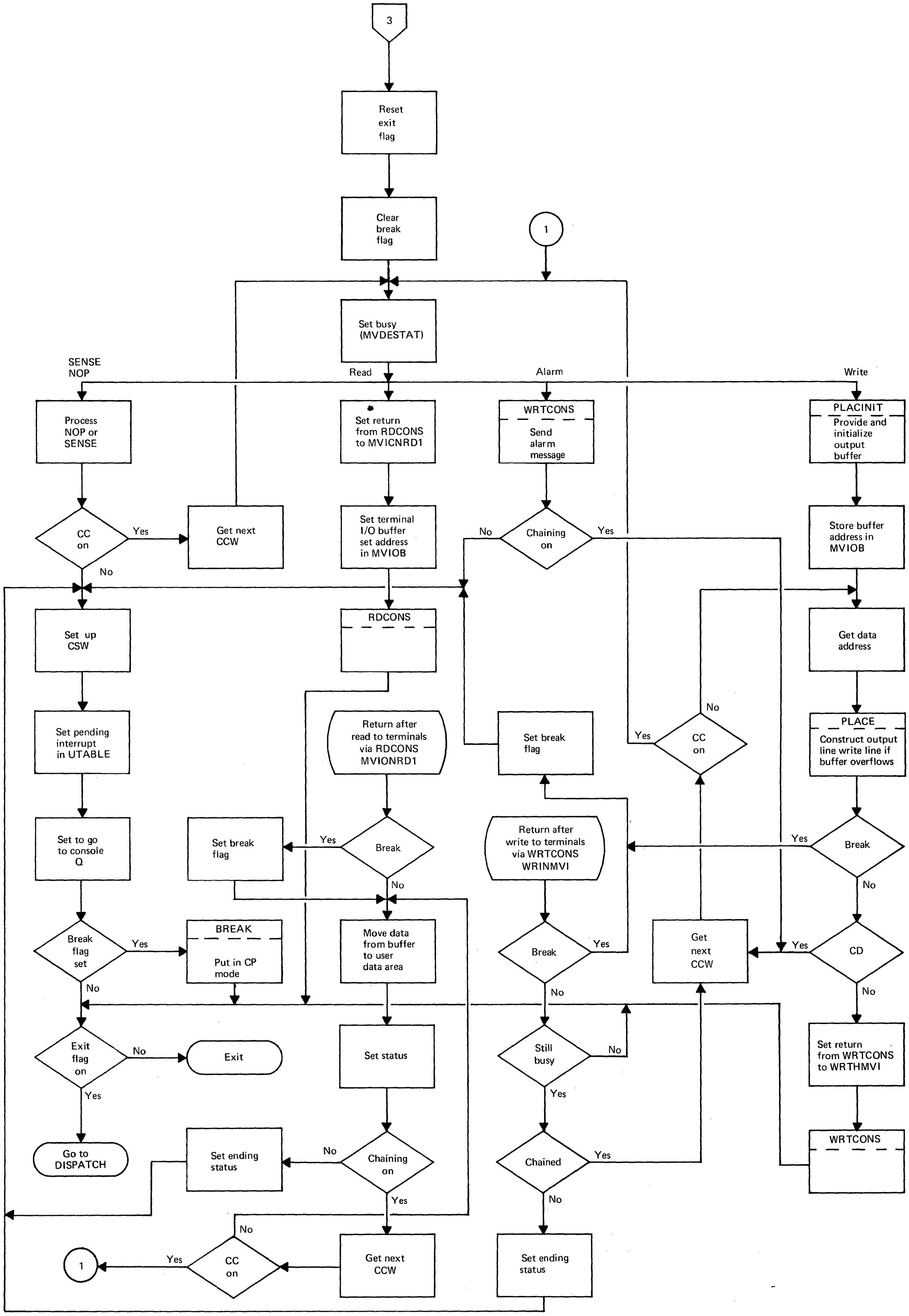


Figure 21. MVIOEXEC Module Processing (4 of 4)

Calls the multiplexer virtual I/O executive program (MVIOEXEC)

Transfers to DSPTCHA (DISPATCH).

For a TIO operation, VIOEXEC:

Tests the virtual channel for a pending channel end; if found, tests for channel end for addressed device. If channel end is found for the device, the channel end is cleared, a condition code of 1 is set, the CSW is updated, and transfers to DSPTCHA (DISPATCH). If a channel end is found, but not for the current device, a condition code of 2 is set. If this is the second time this has happened recently, DISDRQ is called to drop a user from a queue and then transfers to DSPTCHB (DISPATCH).

If a pending channel end is not found, the virtual control unit is tested for pending interruptions. If found, a condition code of 1 is set, the CSW is updated, and control is returned to PROGINT.

If a pending control unit interruption is not found, the virtual device is tested for pending interruptions. If found, the pending interruptions are cleared, the device status and the count of pending interruptions are updated, a condition code of 1 is set, the CSW is updated, and transfers to DSPTCHA (DISPATCH).

If a pending device interruption is not found, a condition code of zero is set, and transfers to DSPTCHA (DISPATCH).

For a TCH operation, VIOEXEC:

Finds the virtual unit address and the virtual channel block

Tests the virtual channel for a pending channel end. If a pending channel end is found, a condition code of 1 is set. If the channel is busy, a condition code of 2 is set; if not, a condition code of zero is set.

Transfers to DSPTCHA (DISPATCH).

For an HIO operation, VIOEXEC:

If I/O is not in progress on the device and interrupts are not pending, sets a condition code indicating that the device is available.

If I/O is in progress, issues an HIO to the device and reflects the condition code to the virtual machine. When the I/O is finished, VIOEXEC sets a condition code indicating interrupt pending.

CCW Translator - CCWTRANS

Entrance: CCWTRANS is called by the virtual machine I/O executive program (VIOEXEC) when an I/O task block has been created and a list of virtual CCW's associated with a user's SIO request must be translated into real CCW's. (See Figure 22 for CCWTRANS module processing.)

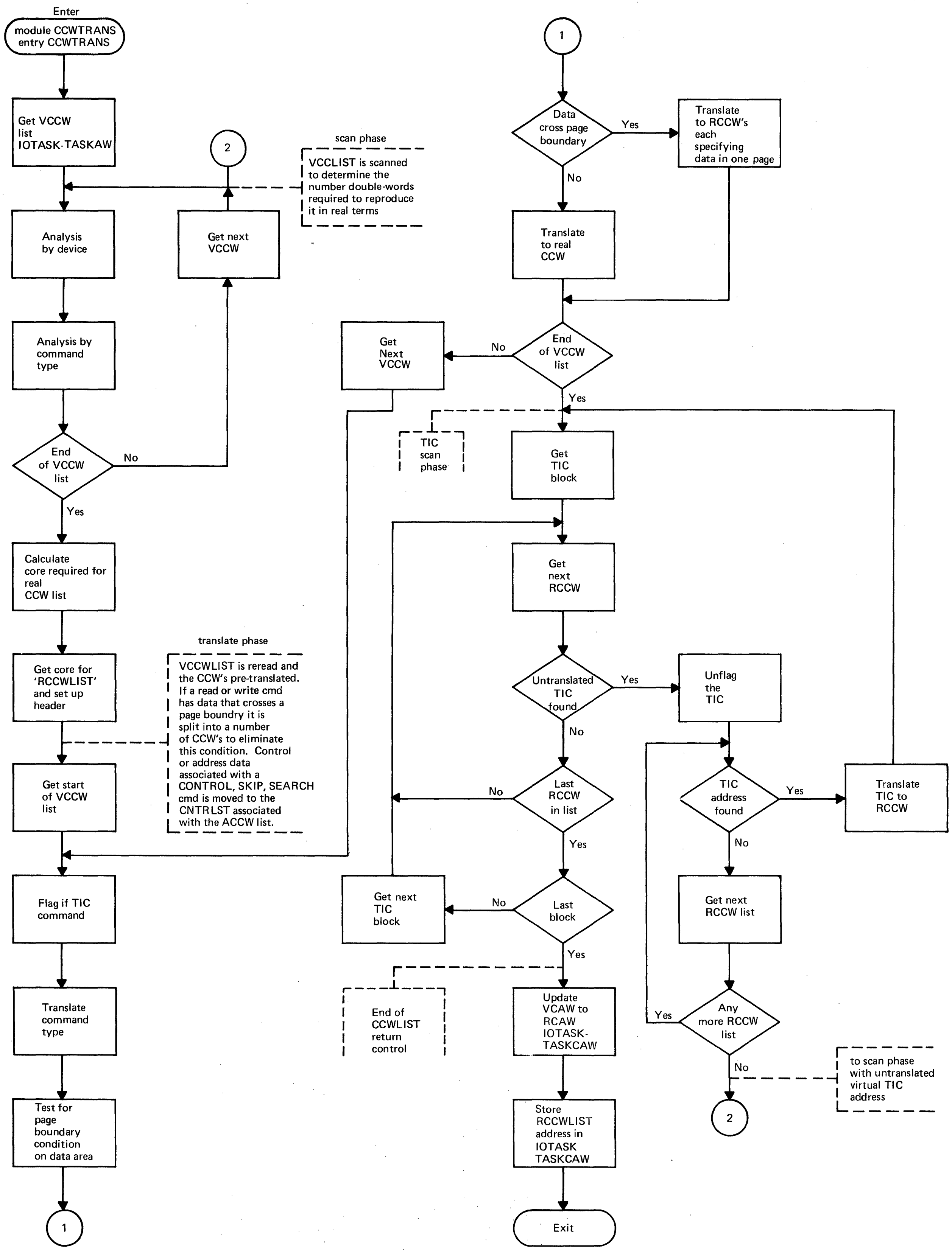

Enter
module CCWTRANS entry CCWTRANS
Get VCCW list IOTASK-TASKAW
2
scan phase
VCCLIST is scanned to determine the number double-words required to reproduce it in real terms
Analysis by device
Get next VCCW
Analysis by command type
End of VCCW list
No
Yes
Calculate core required for real CCW list
translate phase
Get core for 'RCCWLIST' and set up header
VCCWLIST is reread and the CCW's pre-translated. If a read or write cmd has data that crosses a page boundry it is split into a number of CCW's to eliminate this condition. Control or address data associated with a CONTROL, SKIP, SEARCH cmd is moved to the CNTRLST associated with the ACCW list.
Get start of VCCW list
Flag if TIC command
Translate command type
Test for page boundary condition on data area
1
1
Data cross page boundary
Yes
Translate to RCCW's each specifying data in one page
No
Translate to real CCW
Get Next VCCW
No
End of VCCW list
Yes
TIC scan phase
Get TIC block
Get next RCCW
Untranslated TIC found
Yes
Unflag the TIC
No
Last RCCW in list
No
TIC address found
Yes
Translate TIC to RCCW
Yes
No
Get next TIC block
No
Last block
Get next RCCW list
Yes
End of CCWLIST return control
Update VCAW to RCAW IOTASK-TASKCAW
Yes
Any more RCCW list
No
to scan phase with untranslated virtual TIC address
2
Store RCCWLIST address in IOTASK TASKCAW
Exit

Figure 22. CCWTRANS Module Processing

CCWTRANS is called by IOINT when the I/O operation is completed from a self-modifying channel program. The self-modifying channel program checking portion of CCWTRANS calls CCWTRANS when retranslation of CCW's is required.

Operation: CCWTRANS operates in four phases: a scan phase, a translate phase, a TIC-scan phase, and a self-modifying channel program checking scan phase if the ISAM option was chosen.

The scan phase analyzes the virtual CCW list to determine the total core storage requirement of the real CCW list. Additional real CCW's are required if the data area specified by the virtual CCW list crosses page boundaries. Some channel commands require additional doublewords for control information (for example, seek addresses).

The translation phase reexamines the virtual CCW list and translates it into a real CCW list. TIC commands that cannot be immediately translated are flagged for later processing by the TIC-scan phase. A read or write command that specifies data crossing page boundaries is translated into several CCW's, each specifying data in only one page.

The TIC-scan phase scans the real CCW list for flagged (untranslated) TIC commands and creates a new virtual CCW list for the untranslated commands. Scan phase processing is then repeated. When all virtual CCW's are translated, the virtual CAW in the IOTASK block is replaced by the real CAW (that is, a pointer to the real CCW list created by CCWTRANS), and CCWTRANS returns control to VIOEXEC. The user protection key is preserved.

Routines called: CCWTRANS calls the page handling routine (PAGTRANS), via a TRANS macro instruction, to translate virtual addresses to real addresses, and to lock in core storage pages required by I/O operations.

The self-modifying channel program checking portion of CCWTRANS calls CCWTRANS to retranslate the channel program and QUEVIO to start the I/O operation.

OS ISAM Handling - CCWTRAN

Because many of the OS ISAM channel programs are self-modifying, special handling is required in CP to allow virtual machines to use this access method. The particular CCW's that require special handling have the following general format:

```
0        2        4        6        8
+--------+--------+--------+--------+
   A  |        READDATA  C+7  10 BYTES         |
      +--------+--------+--------+--------+
   B  |               TIC TO E                 |
      +--------+--------+--------+--------+
   C  |                               |        |
      +--------+--------+--------+--------+
   D  |                                        |
      +--------+--------+--------+--------+
   E  |    |     SEEK: SEEK HEAD ON D          |
      +--------+--------+--------+--------+
   F  |             SEARCH ON D+2              |
      +--------+--------+--------+--------+
```

The CCW at A reads 10 bytes of data, the last byte of which forms the command code of the CCW at E. In addition, the data read in forms the seek and search arguments for the CCW's at E and F. The normal CP translated CCW string has the following format:

```
      0        2        4        6        8
      +--------+--------+--------+--------+
   1  |        READDATA  C+7  10 BYTES         |
      +--------+--------+--------+--------+
   2  |               TIC TO 3                 |
      +--------+--------+--------+--------+

      +--------+--------+--------+--------+
   2A |   VIRTUAL ADDRESS OF SEEK AT E         |
      +--------+--------+--------+--------+
   3  |        SEEK: SEEK HEAD ON 6            |
      +--------+--------+--------+--------+
   4  |             SEARCH ON D+2              |
      +--------+--------+--------+--------+
   5  |                 ETC.                   |
      +--------+--------+--------+--------+
   6  |         RELOCATED SEEK ARG.            |
      +--------+--------+--------+--------+
```

In order to accomplish an efficient and non-timing dependent translated operation for OS ISAM, the virtual CCW string is modified in the following manner.

The ISAM scan phase of CCWTRAN is entered if, during normal translation, a CCW of the type at A is encountered. The scan phase locates the TIC at 2 by searching the translated CCW strings. The TIC at 2 locates the seek at 3.

The virtual address of the virtual seek CCW at E is located at 2A. The 4 bytes at E and the four bytes at F are saved in the eight byte area at 6. The TIC at 2 is altered to TIC to the virtual CCW at E. The CCW address field at E is translated to reference D. The 4 bytes at F are modified to a TIC to the CCW's starting at 4. The completed CCW string has the following format:

```
      0        2        4        6        8
      +--------+--------+--------+--------+
1   |      READDATA  C+7  10 BYTES      |
    +--------+--------+--------+--------+
2   |            TIC TO E               |
    +--------+--------+--------+--------+

    +--------+--------+--------+--------+
2A  |   VIRTUAL ADDRESS OF SEEK AT E    |
    +--------+--------+--------+--------+
3   |              NOT USED             |
    +--------+--------+--------+--------+
4   |           SEARCH ON D+2           |
    +--------+--------+--------+--------+
5   |               ETC.                |
    +--------+--------+--------+--------+
6   |     SAVED E     |     SAVED F     |
    +--------+--------+--------+--------+
```

TRANSLATED CCW's

```
    0        2        4        6        8
    +--------+--------+--------+--------+
A   |      READDATA  C+7  10 BYTES      |
    +--------+--------+--------+--------+
B   |             TIC TO E              |
    +--------+--------+--------+--------+
C   |                              |    |
    +--------+--------+--------+--------+
D   |                                   |
    +--------+--------+--------+--------+
E   |    |     SEEK: SEEK HEAD ON D     |
    +--------+--------+--------+--------+
F   |             TIC TO 4              |
    +--------+--------+--------+--------+
```

VIRTUAL CCW's

It can be seen that the virtual area C, D, E, and F must reside in one page for the routine to function.

Once the I/O operation has completed, an untranslation scan phase restores the data at E and F and sets the correct CSW address if the channel program ended at E.

CCW Untranslator - UNTRANS

Entrance: UNTRANS is called by VIOINT when a channel end type of interrupt occurs for a user's virtual input-output operation. Its function is to convert the real CSW information into corresponding virtual CSW information.

Operation: The real CCW that caused the interrupt is located from the virtual channel CSW (VCHCSW), where the real CSW is temporarily stored. Taking into account the fact that some of the CCW's may be system-generated and artificially data-chained, a virtual CSW is created to represent the CSW that would be expected from the user's virtual CCW list(s).

CCW Return to Free Storage - FREECCW

Entrance: FREECCW is called when VIOINT determines that the channel has terminated operation on a user's virtual list. It returns the real CCW equivalent to the virtual list to free storage and clears the TASKCAW entry in the IOTASK block.

Operation: The real CAW is picked up from TASKCAW, which is an entry in IOTASK. From this, the real CCW list with its "header" information is located. The list is scanned. All I/O commands with data references have their referenced pages unlocked, and the received data for Read Home Address commands for shared disks is unrelocated. When the scanning is complete, the CCW list is returned to free storage.

Routines called: PAGUNLOK is called to unlock the page containing the I/O data area.

Virtual I/O Request Queueing Routine - QUEVIO

Entrance: QUEVIO is called by the virtual machine I/O executive program (VIOEXEC) when an I/O task block has been created and a virtual CCW list has been translated into a real CCW list. (See Figure 23 for QUEVIO module processing.)

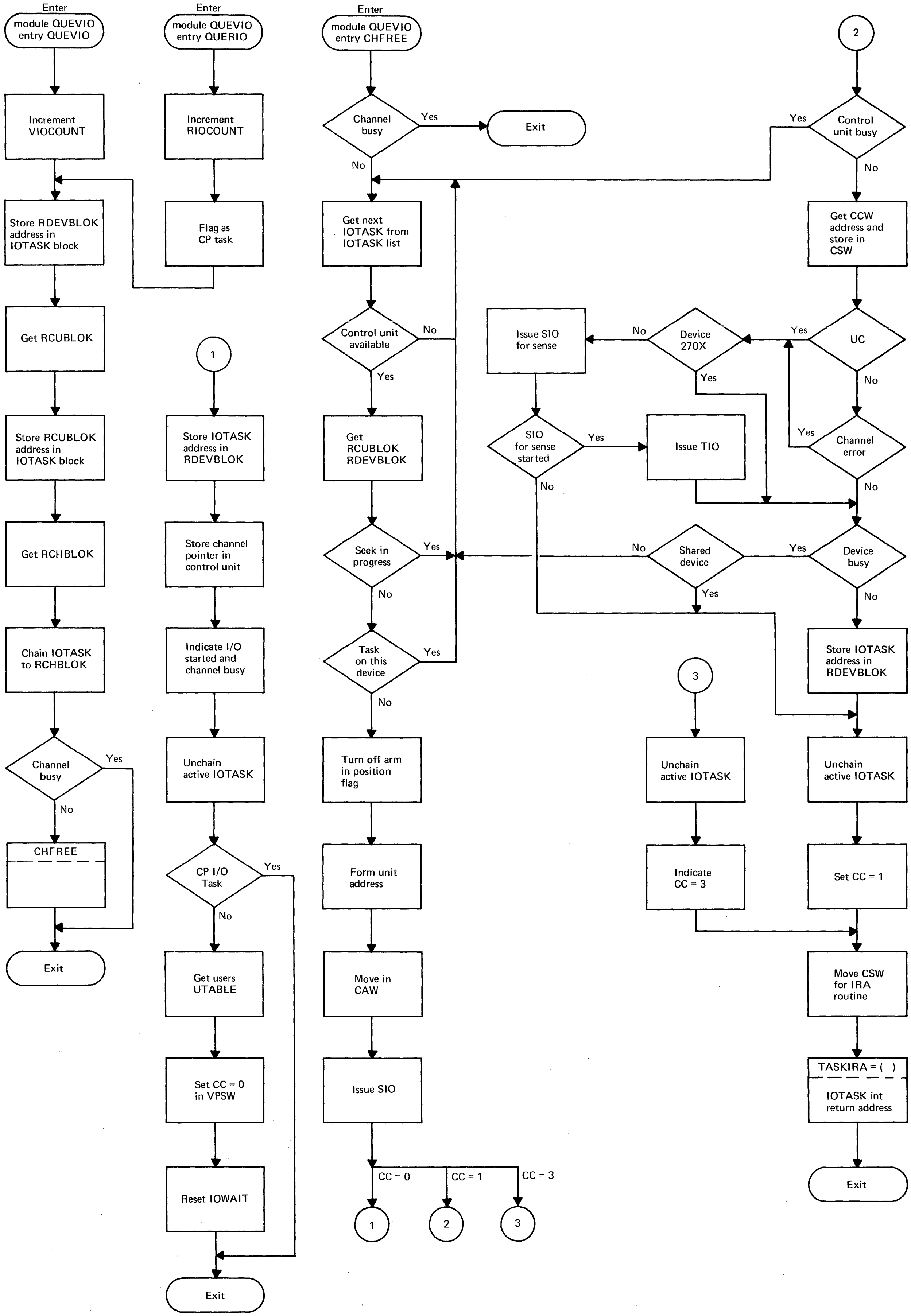


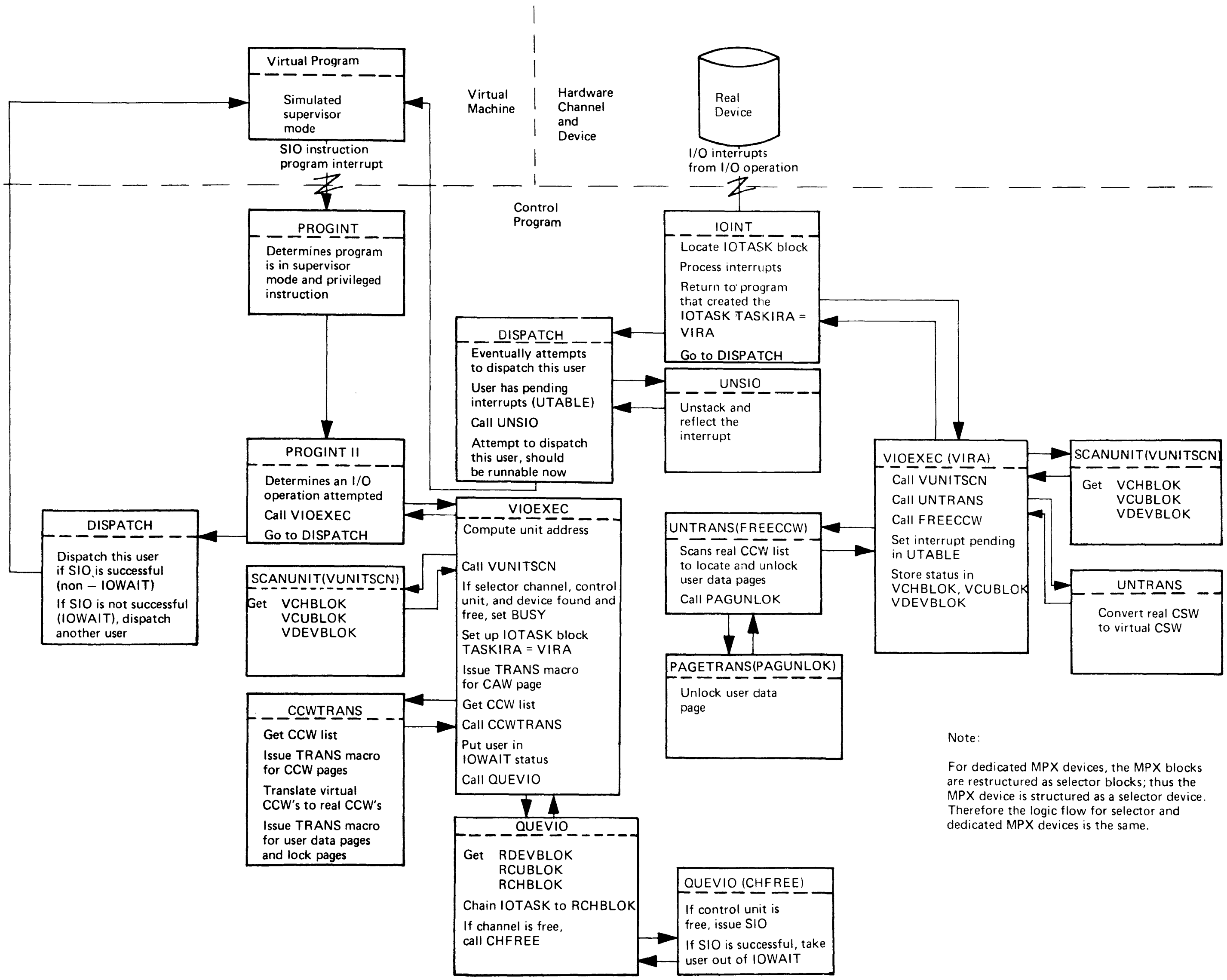
Figure 23. QUEVIO Module Processing

Operation: When QUEVIO is entered, register 1 contains the address of an I/O task block to be queued on a real channel, and register 2 contains the address of the appropriate virtual device block. QUEVIO attaches the I/O task block to the appropriate channel block, increments the task count, and tests the real channel.

Routines called: If QUEVIO determines that the channel to which the I/O task block has been attached is free, CHFREE is called to start the I/O operation. If the I/O operation is successfully started, the I/O task block is unchained from the channel blcck and chained to the real device block. If the I/O operation is not successfully started, the I/O task block is unchained from the channel block, and the task count is decremented.

When CHFREE processing is completed, QUEVIO returns control to its caller - VIOEXEC, after reflecting the SIO condition code to the virtual PSW, and taking the user out of IOWAIT.

Figure 24 illustrates the relationships of routines which process user selector channel I/O requests.


Virtual Program
Simulated supervisor mode
SIO instruction program interrupt
Virtual Machine
Hardware Channel and Device
Real Device
I/O interrupts from I/O operation
Control Program
PROGINT
Determines program is in supervisor mode and privileged instruction
IOINT
Locate IOTASK block
Process interrupts
Return to program that created the IOTASK TASKIRA = VIRA
Go to DISPATCH
DISPATCH
Eventually attempts to dispatch this user
User has pending interrupts (UTABLE)
Call UNSIO
Attempt to dispatch this user, should be runnable now
UNSIO
Unstack and reflect the interrupt
PROGINT II
Determines an I/O operation attempted
Call VIOEXEC
Go to DISPATCH
VIOEXEC (VIRA)
Call VUNITSCN
Call UNTRANS
Call FREECCW
Set interrupt pending in UTABLE
Store status in VCHBLOK, VCUBLOK VDEVBLOK
SCANUNIT(VUNITSCN)
Get VCHBLOK VCUBLOK VDEVBLOK
DISPATCH
Dispatch this user if SIO is successful (non – IOWAIT)
If SIO is not successful (IOWAIT), dispatch another user
VIOEXEC
Compute unit address
Call VUNITSCN
If selector channel, control unit, and device found and free, set BUSY
Set up IOTASK block TASKIRA = VIRA
Issue TRANS macro for CAW page
Get CCW list
Call CCWTRANS
Put user in IOWAIT status
Call QUEVIO
UNTRANS(FREECCW)
Scans real CCW list to locate and unlock user data pages
Call PAGUNLOK
UNTRANS
Convert real CSW to virtual CSW
SCANUNIT(VUNITSCN)
Get VCHBLOK VCUBLOK VDEVBLOK
PAGETRANS(PAGUNLOK)
Unlock user data page
CCWTRANS
Get CCW list
Issue TRANS macro for CCW pages
Translate virtual CCW's to real CCW's
Issue TRANS macro for user data pages and lock pages
QUEVIO
Get RDEVBLOK RCUBLOK RCHBLOK
Chain IOTASK to RCHBLOK
If channel is free, call CHFREE
QUEVIO (CHFREE)
If control unit is free, issue SIO
If SIO is successful, take user out of IOWAIT

Note:

For dedicated MPX devices, the MPX blocks are restructured as selector blocks; thus the MPX device is structured as a selector device. Therefore the logic flow for selector and dedicated MPX devices is the same.

FIGURE 24. Virtual SIO Selector Channel

Virtual Channel Interruption Handler - VIRA

Entrance: When a user-requested I/O operation is started on a selector channel, the interruption return address (TASKIRA) in the I/O task block points to the virtual channel interruption handler (VIRA). When the I/O operation is completed and an interruption occurs, VIRA receives control from IOINT, the real input-output interruption handler.

Operation: VIRA indicates in the user's control table (UTABLE) that an interruption is pending, and stores status information in the virtual channel block, virtual control unit block, and the virtual device block when appropriate. The I/O task block is unchained from the real channel block and returned to free storage if the operation is complete (that is, channel end and device end or their equivalents occurred). If an I/O error has occurred, control is passed to IOERROR. See "Processing of I/O Errors - IOERROR".

Routines called: VIOINT calls the routines IOISTVCU and IOISTVDE (subroutines within the real I/O interruption handler) to indicate a control unit end interruption and a device end interruption respectively. When VIOINT processing is completed, an exit is taken to the main dispatcher and control routine (DISPATCH).

Routine to Analyze and Record Errors - RECERROR

Entrance: If an I/O error occurs for a user-requested I/O operation on a selector channel, VIOINT calls RECERROR to analyze and record the error.

Operation: RECERROR analyzes the I/O error from information contained in sense byte zero. The following types of I/O errors are recorded.

Type of Error	Counter Number	Bit Position Within Sense Byte 0
Bus Out Parity	1	2
Equipment Check	2	3
Data Check	3	4
Seek Check	4	7

Counters for each of these types of errors are kept in the RDEVBLOK for each device. Note that errors are recorded for dedicated devices operating on a virtual multiplexer channel (unit record equipment, virtual 2702s). If the error is the first encountered of a given type for a given device, the error is recorded. If the error causes the counter to overflow (that is, upon the eighth error of this type for the device), a counter overflow error record is written. This error may represent the failure of a completely different channel program than the first error of this type which was recorded. If the error is neither the first encountered nor a cause of a counter overflow condition, control returns to VIOINT, and the error information is reflected back to the user's virtual machine.

The I/O error record has the following 112-byte format:

```
         ORG    LOGDATA   DEFINE I/O ERROR RECORD
LOGSNSE  DS     CL6       SENSE INFORMATION
LOGCODE  DS     CL1       DEVICE TYPE
LOGTYPE  DS     CL1       FIRST ENCOUNTERED OR COUNTER
                          OVERFLOW - TYPE OF ERROR
LOGVOLID DS     CL6       VOLID OF DEVICE (IF AVAILABLE)
LOGADDR  DS     CL2       PHYSICAL ADDRESS OF DEVICE
LOGDATE  DS     CL6       DATE AND TIME STAMP OF ERROR
LOGCSW   DS     CL8       CHANNEL STATUS WORD
         DS     CL2       UNUSED
LOGCCWS  DS     9D        FAILING CCW STRING (UP TO NINE
                          DOUBLEWORDS)
LOGSKLOC DS     1D        LAST SEEK ADDRESS (DASD ONLY)
```

The CCW in the string which failed is flagged with an asterisk in the unused fifth byte.

After the error record is written, the pointer to the next available slot on the CE cylinder is updated. Seven logical records are contained within one 829-byte physical record. Since 15 records may be written on two tracks of a 2314, up to 1050 error records may be written on one cylinder. If the attempt to write the error record fails, it is retried eight times. Upon continued failure, an error message "** IOERROR RECORDING FAILURE ON DEV___" is sent to the operator. If there is no more room on the CE cylinder for error records, the message "**CECYL FULL; I/O ERRORS NOT RECORDED **" is sent to the operator. Errors are not recorded for users with privilege class C in order to prevent the recording of intentional errors produced by CE diagnostics. Recording will be reinitiated after the CE executes the CLEARIO function.

Main Dispatcher and Control Routine - DISPATCH

Entrance: DISPATCH is entered from routines which have completed their processing for a user or cannot continue processing until some other process has been completed. (See Figure 10.1 for DISPATCH module processing.)

Operation: DISPATCH checks for pending interruptions and determines which user is to receive control next.

Routines called: When DISPATCH determines that an I/O interruption is pending, the I/O interruption unstacking routine (UNSTIO) is called. UNSTIO updates the virtual CSW, restores virtual PSW's, and indicates the address of the interrupting device. When UNSTIO processing is completed, DISPATCH attempts to restart the current user, if runnable and if his quantum is not exhausted.

DISPATCH may be entered at 4 locations: DISPATCH, DSPTCHA, DSPTCHB, and DSPTCHC. DISPATCH is the normal entry point used by all routines that are not sure of a user's status. DSPTCHA is entered from routines which have gained control after a program interrupt for a user and have changed the user's PSW. DSPTCHB is similar to DSPTCHA except the PSW is at most changed in its condition code field. DSPTCHC is used by routines which have done some processing for a user but in no way changed his status.

Figures '25-28' illustrate the relationships of routines which process an I/O interrupt returned from a selector channel device.

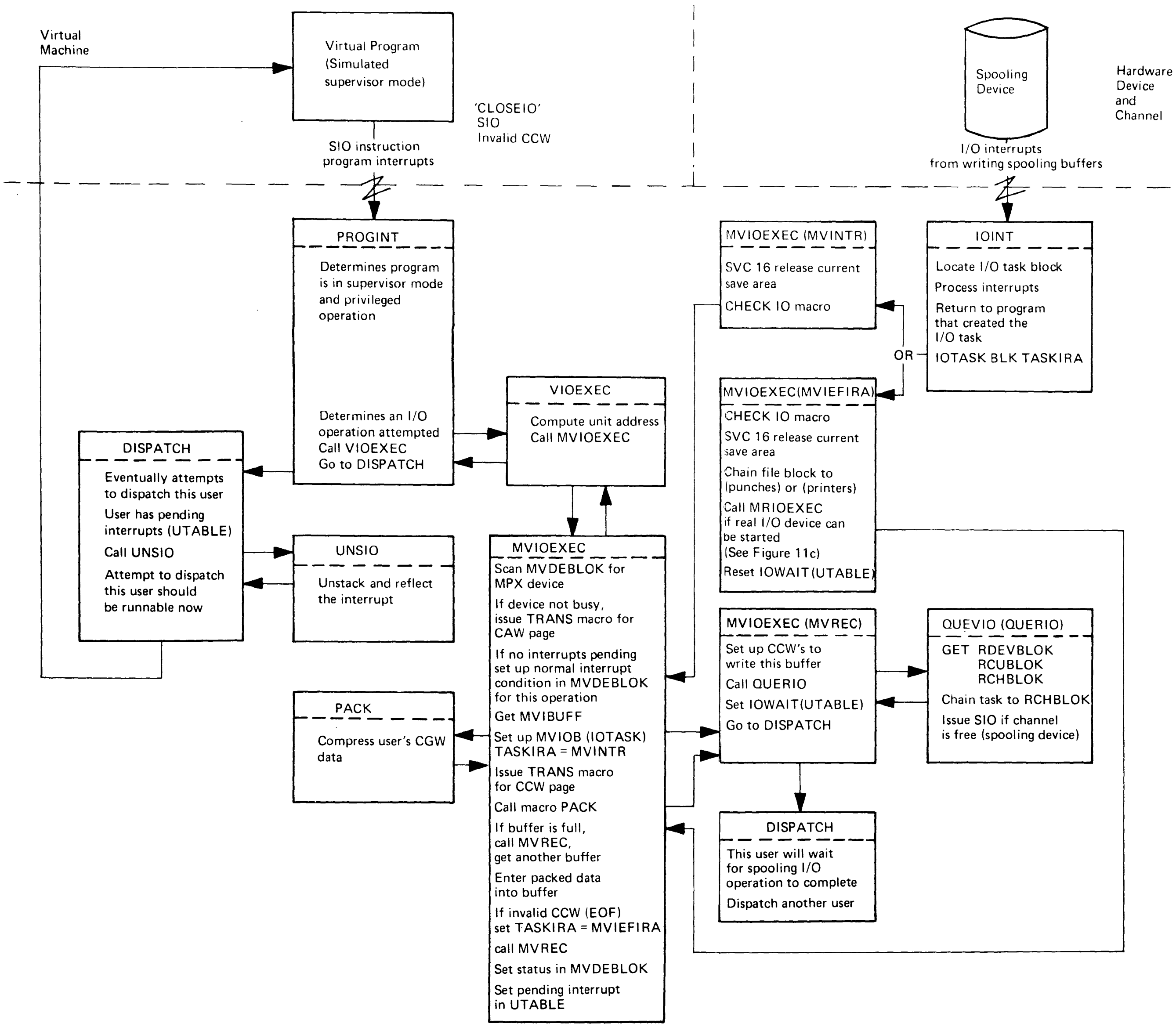

Virtual Machine
Virtual Program (Simulated supervisor mode)
SIO instruction program interrupts
'CLOSEIO' SIO Invalid CCW
Spooling Device
Hardware Device and Channel
I/O interrupts from writing spooling buffers
PROGINT
Determines program is in supervisor mode and privileged operation
Determines an I/O operation attempted
Call VIOEXEC
Go to DISPATCH
DISPATCH
Eventually attempts to dispatch this user
User has pending interrupts (UTABLE)
Call UNSIO
Attempt to dispatch this user should be runnable now
UNSIO
Unstack and reflect the interrupt
VIOEXEC
Compute unit address
Call MVIOEXEC
PACK
Compress user's CGW data
MVIOEXEC
Scan MVDEBLOK for MPX device
If device not busy, issue TRANS macro for CAW page
If no interrupts pending set up normal interrupt condition in MVDEBLOK for this operation
Get MVIBUFF
Set up MVIOB (IOTASK) TASKIRA = MVINTR
Issue TRANS macro for CCW page
Call macro PACK
If buffer is full, call MVREC, get another buffer
Enter packed data into buffer
If invalid CCW (EOF) set TASKIRA = MVIEFIRA
call MVREC
Set status in MVDEBLOK
Set pending interrupt in UTABLE
MVIOEXEC (MVINTR)
SVC 16 release current save area
CHECK IO macro
OR
MVIOEXEC(MVIEFIRA)
CHECK IO macro
SVC 16 release current save area
Chain file block to (punches) or (printers)
Call MRIOEXEC if real I/O device can be started (See Figure 11c)
Reset IOWAIT(UTABLE)
IOINT
Locate I/O task block
Process interrupts
Return to program that created the I/O task
IOTASK BLK TASKIRA
MVIOEXEC (MVREC)
Set up CCW's to write this buffer
Call QUERIO
Set IOWAIT(UTABLE)
Go to DISPATCH
QUEVIO (QUERIO)
GET RDEVBLOK RCUBLOK RCHBLOK
Chain task to RCHBLOK
Issue SIO if channel is free (spooling device)
DISPATCH
This user will wait for spooling I/O operation to complete
Dispatch another user

FIGURE 25. Virtual SIO MPX Channel (Nondedicated Punch or Printer)

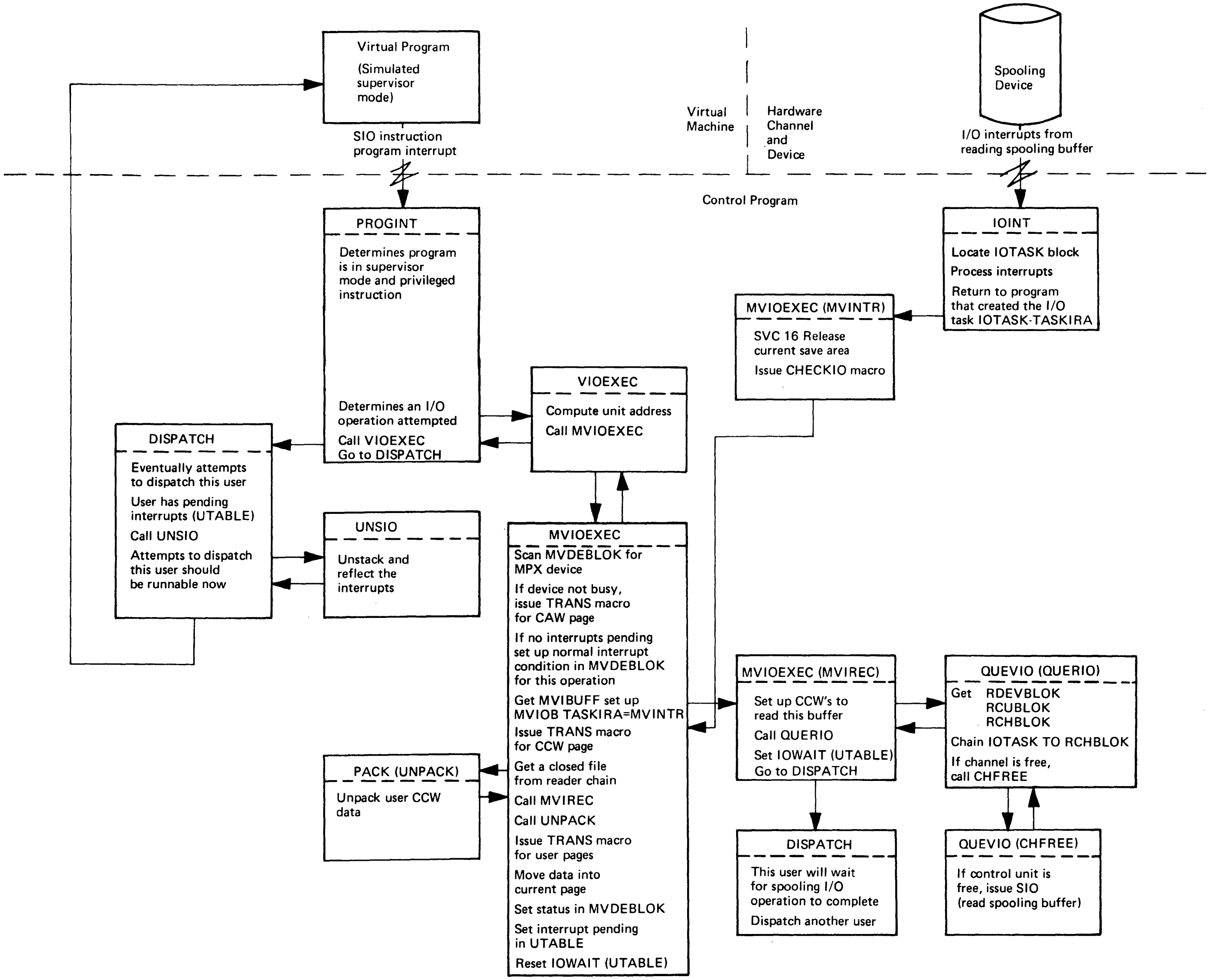
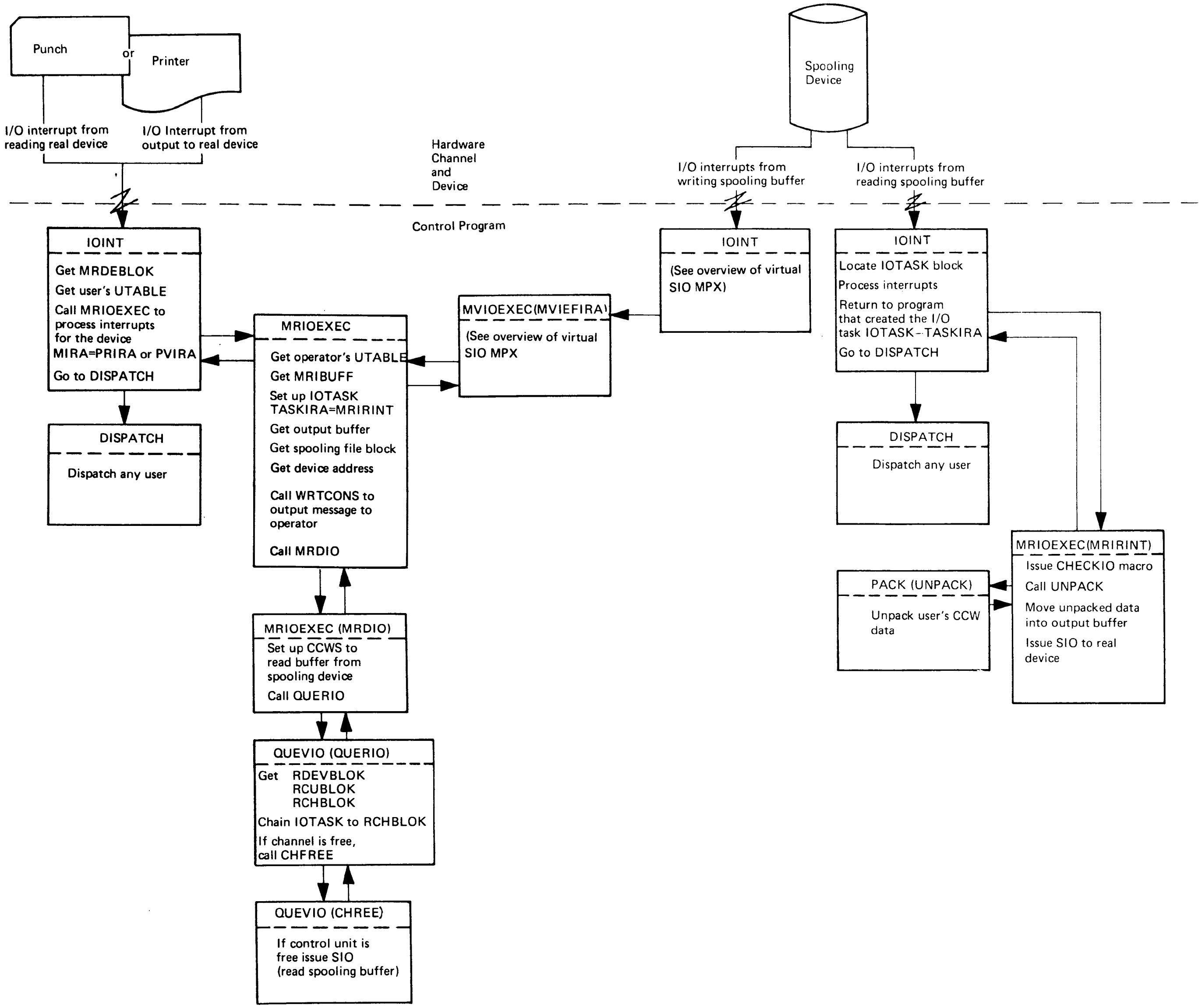

Virtual Program
(Simulated supervisor mode)
SIO instruction program interrupt
Virtual Machine
Hardware Channel and Device
Spooling Device
I/O interrupts from reading spooling buffer
Control Program
PROGINT
Determines program is in supervisor mode and privileged instruction
Determines an I/O operation attempted
Call VIOEXEC
Go to DISPATCH
IOINT
Locate IOTASK block
Process interrupts
Return to program that created the I/O task IOTASK-TASKIRA
MVIOEXEC (MVINTR)
SVC 16 Release current save area
Issue CHECKIO macro
VIOEXEC
Compute unit address
Call MVIOEXEC
DISPATCH
Eventually attempts to dispatch this user
User has pending interrupts (UTABLE)
Call UNSIO
Attempts to dispatch this user should be runnable now
UNSIO
Unstack and reflect the interrupts
MVIOEXEC
Scan MVDEBLOK for MPX device
If device not busy, issue TRANS macro for CAW page
If no interrupts pending set up normal interrupt condition in MVDEBLOK for this operation
Get MVIBUFF set up MVIOB TASKIRA=MVINTR
Issue TRANS macro for CCW page
Get a closed file from reader chain
Call MVIREC
Call UNPACK
Issue TRANS macro for user pages
Move data into current page
Set status in MVDEBLOK
Set interrupt pending in UTABLE
Reset IOWAIT (UTABLE)
MVIOEXEC (MVIREC)
Set up CCW's to read this buffer
Call QUERIO
Set IOWAIT (UTABLE)
Go to DISPATCH
QUEVIO (QUERIO)
Get RDEVBLOK
RCUBLOK
RCHBLOK
Chain IOTASK TO RCHBLOK
If channel is free, call CHFREE
PACK (UNPACK)
Unpack user CCW data
DISPATCH
This user will wait for spooling I/O operation to complete
Dispatch another user
QUEVIO (CHFREE)
If control unit is free, issue SIO (read spooling buffer)

FIGURE 26. Virtual SIO MPX Channel (Nondedicated Reader)



Punch
or
Printer
I/O interrupt from reading real device
I/O Interrupt from output to real device
Hardware Channel and Device
Control Program
Spooling Device
I/O interrupts from writing spooling buffer
I/O interrupts from reading spooling buffer
IOINT
Get MRDEBLOK
Get user's UTABLE
Call MRIOEXEC to process interrupts for the device
MIRA=PRIRA or PVIRA
Go to DISPATCH
DISPATCH
Dispatch any user
MRIOEXEC
Get operator's UTABLE
Get MRIBUFF
Set up IOTASK TASKIRA=MRIRINT
Get output buffer
Get spooling file block
Get device address
Call WRTCONS to output message to operator
Call MRDIO
MVIOEXEC(MVIEFIRA)
(See overview of virtual SIO MPX
IOINT
(See overview of virtual SIO MPX)
IOINT
Locate IOTASK block
Process interrupts
Return to program that created the I/O task IOTASK-TASKIRA
Go to DISPATCH
DISPATCH
Dispatch any user
MRIOEXEC (MRDIO)
Set up CCWS to read buffer from spooling device
Call QUERIO
QUEVIO (QUERIO)
Get RDEVBLOK
RCUBLOK
RCHBLOK
Chain IOTASK to RCHBLOK
If channel is free, call CHFREE
QUEVIO (CHREE)
If control unit is free issue SIO (read spooling buffer)
MRIOEXEC(MRIRINT)
Issue CHECKIO macro
Call UNPACK
Move unpacked data into output buffer
Issue SIO to real device
PACK (UNPACK)
Unpack user's CCW data

FIGURE 27. Real SIO MPX Channel
(Punch or Printer)

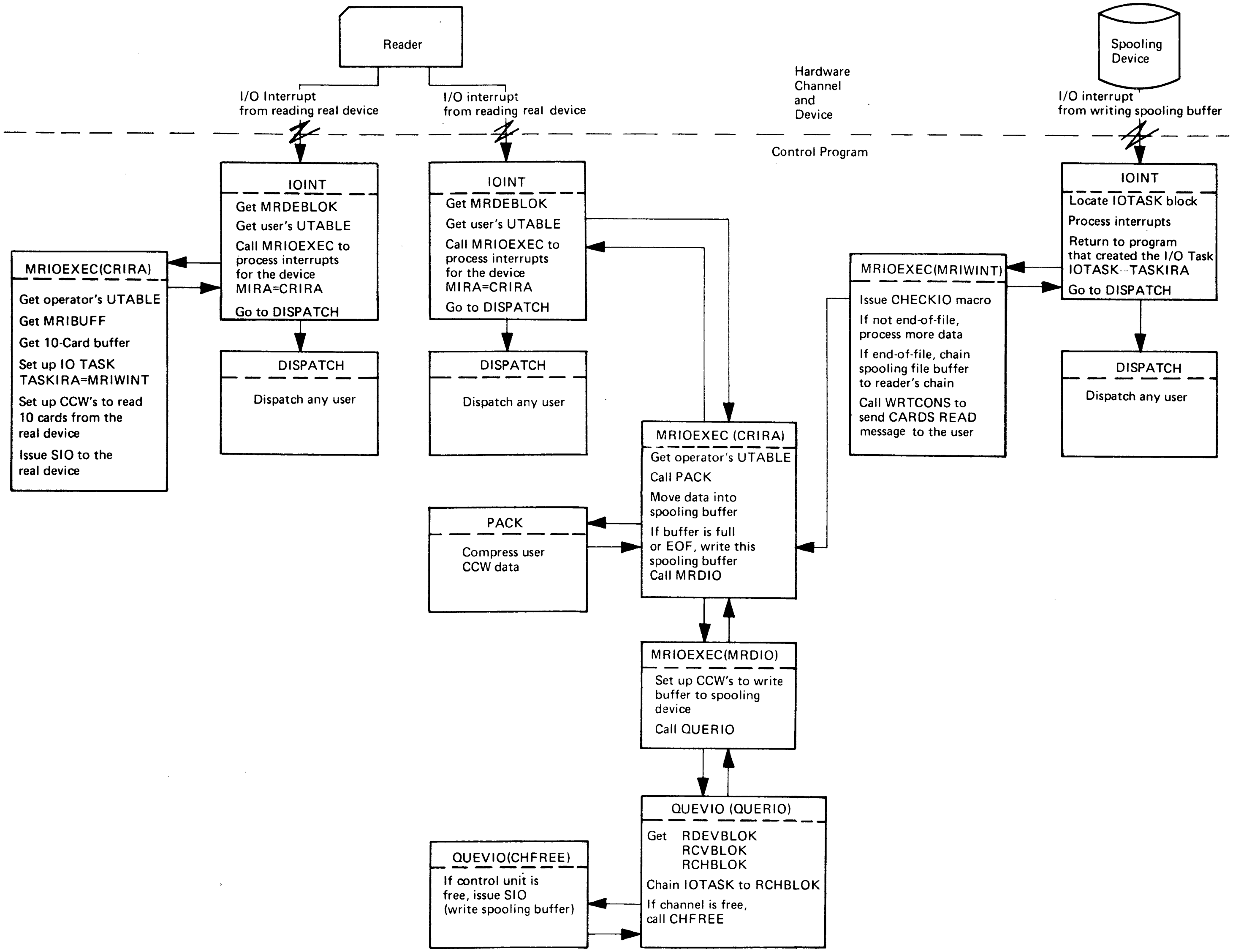

Reader
I/O Interrupt from reading real device
I/O interrupt from reading real device
Hardware Channel and Device
Spooling Device
I/O interrupt from writing spooling buffer
Control Program
IOINT
Get MRDEBLOK
Get user's UTABLE
Call MRIOEXEC to process interrupts for the device MIRA=CRIRA
Go to DISPATCH
IOINT
Get MRDEBLOK
Get user's UTABLE
Call MRIOEXEC to process interrupts for the device MIRA=CRIRA
Go to DISPATCH
IOINT
Locate IOTASK block
Process interrupts
Return to program that created the I/O Task IOTASK--TASKIRA
Go to DISPATCH
MRIOEXEC(CRIRA)
Get operator's UTABLE
Get MRIBUFF
Get 10-Card buffer
Set up IO TASK TASKIRA=MRIWINT
Set up CCW's to read 10 cards from the real device
Issue SIO to the real device
DISPATCH
Dispatch any user
DISPATCH
Dispatch any user
MRIOEXEC(MRIWINT)
Issue CHECKIO macro
If not end-of-file, process more data
If end-of-file, chain spooling file buffer to reader's chain
Call WRTCONS to send CARDS READ message to the user
DISPATCH
Dispatch any user
MRIOEXEC (CRIRA)
Get operator's UTABLE
Call PACK
Move data into spooling buffer
If buffer is full or EOF, write this spooling buffer
Call MRDIO
PACK
Compress user CCW data
MRIOEXEC(MRDIO)
Set up CCW's to write buffer to spooling device
Call QUERIO
QUEVIO (QUERIO)
Get RDEVBLOK RCVBLOK RCHBLOK
Chain IOTASK to RCHBLOK
If channel is free, call CHFREE
QUEVIO(CHFREE)
If control unit is free, issue SIO (write spooling buffer)

FIGURE 28. Real SIO MPX Channel (Reader)

PROCESSING USER MULTIPLEXER CHANNEL I/O REQUESTS

When a pseudo-supervisor (that is, a supervisor operating in a user's virtual machine) requests an I/O operation for a device attached to the multiplexer channel, the program interrupt handler (PROGINT), and the virtual machine I/O receive control. (See preceding section headed "Processing User Selector Channel I/O Requests".) When VIOEXEC determines that an I/O operation has been requested for a device attached to the multiplexer channel, the multiplexer virtual I/O executive program (MVIOEXEC) is called. Figures '29-32' illustrate the relationships of routines which process user multiplexer channel I/O requests.

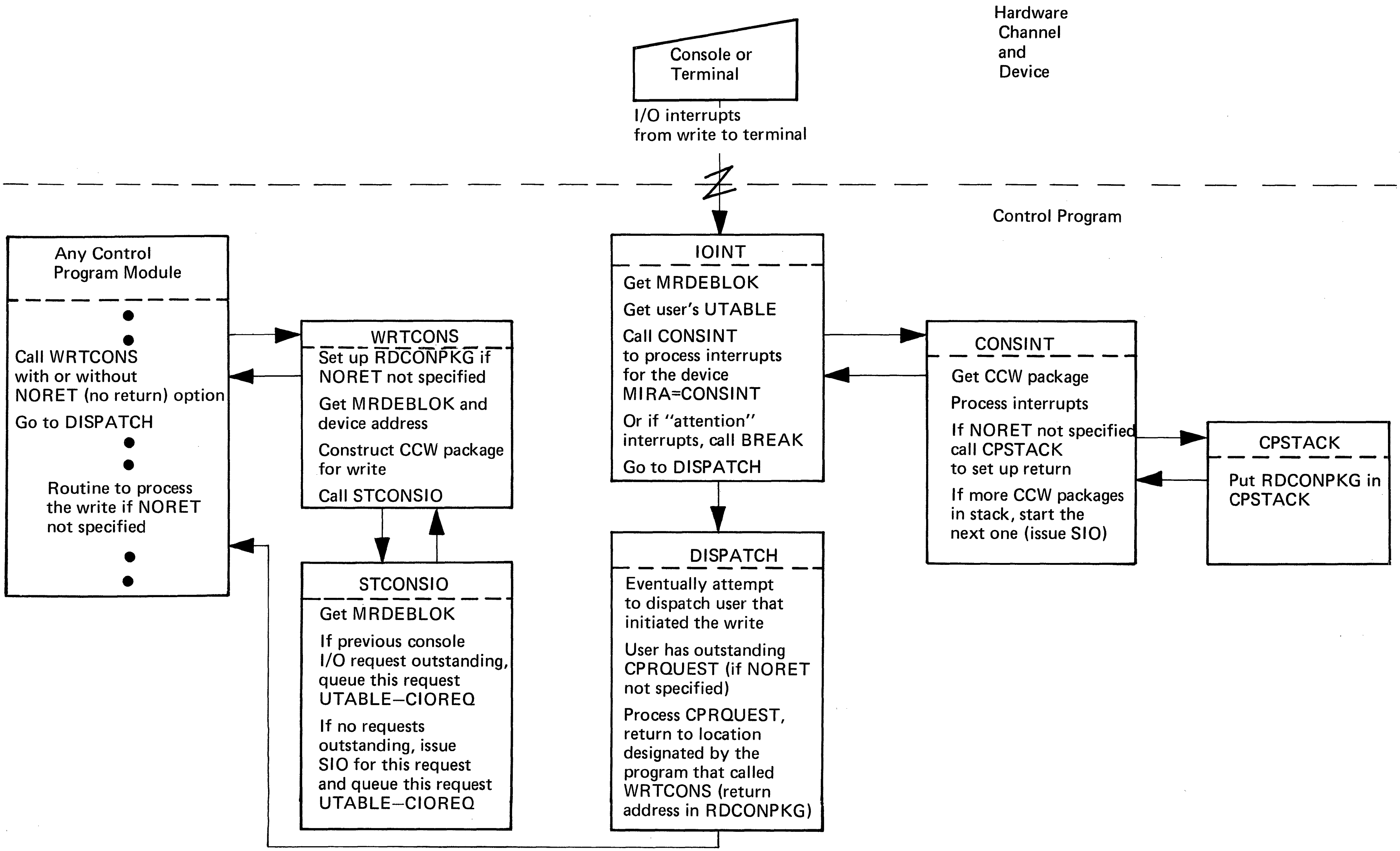

Hardware Channel and Device
Console or Terminal
I/O interrupts from write to terminal
Control Program
Any Control Program Module
Call WRTCONS with or without NORET (no return) option
Go to DISPATCH
Routine to process the write if NORET not specified
WRTCONS
Set up RDCONPKG if NORET not specified
Get MRDEBLOK and device address
Construct CCW package for write
Call STCONSIO
STCONSIO
Get MRDEBLOK
If previous console I/O request outstanding, queue this request UTABLE–CIOREQ
If no requests outstanding, issue SIO for this request and queue this request UTABLE–CIOREQ
IOINT
Get MRDEBLOK
Get user's UTABLE
Call CONSINT to process interrupts for the device MIRA=CONSINT
Or if "attention" interrupts, call BREAK
Go to DISPATCH
DISPATCH
Eventually attempt to dispatch user that initiated the write
User has outstanding CPRQUEST (if NORET not specified)
Process CPRQUEST, return to location designated by the program that called WRTCONS (return address in RDCONPKG)
CONSINT
Get CCW package
Process interrupts
If NORET not specified call CPSTACK to set up return
If more CCW packages in stack, start the next one (issue SIO)
CPSTACK
Put RDCONPKG in CPSTACK

FIGURE 29. Real Terminal SIO (Write)

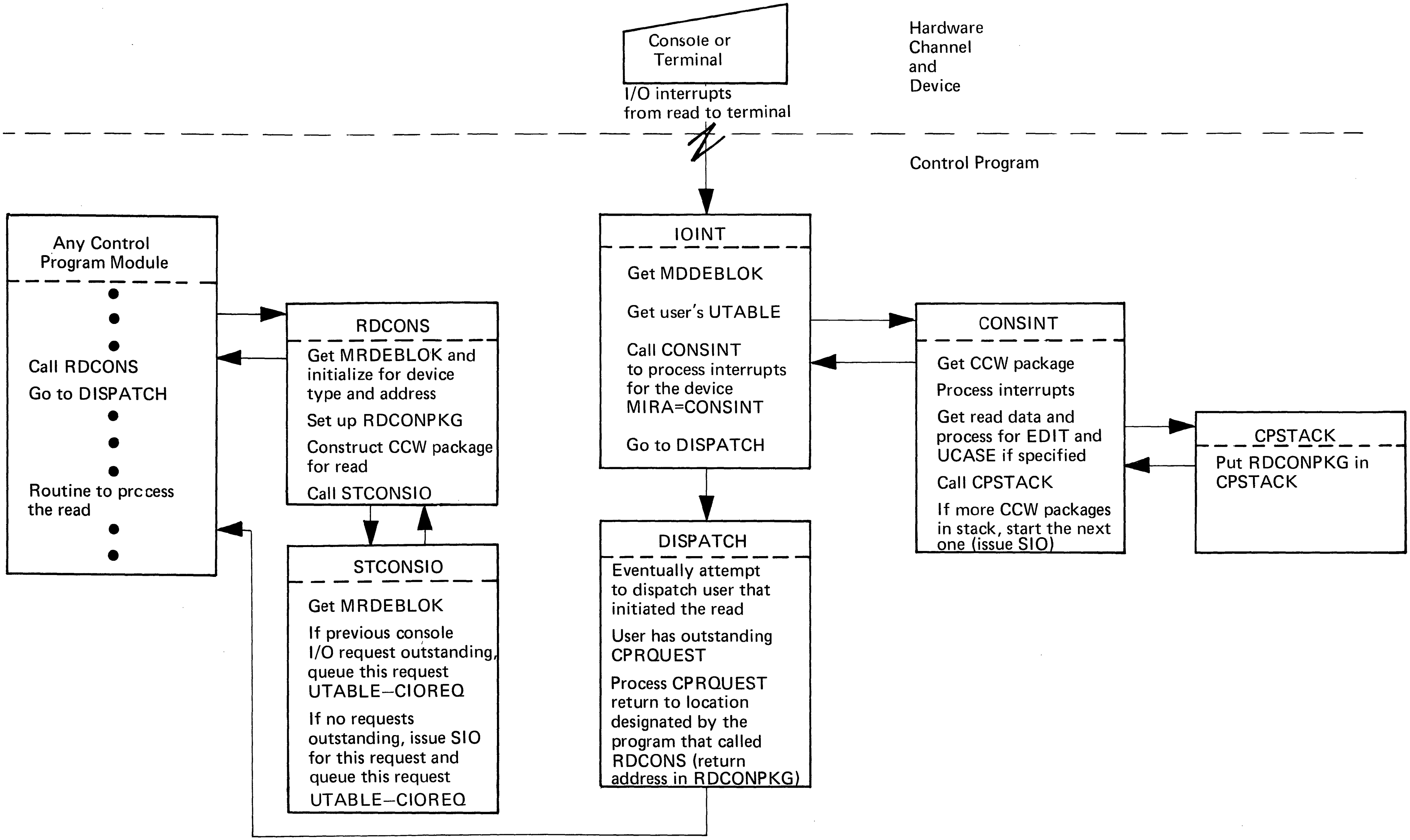

Console or Terminal
I/O interrupts from read to terminal
Hardware Channel and Device
Control Program
Any Control Program Module
Call RDCONS
Go to DISPATCH
Routine to prccess the read
RDCONS
Get MRDEBLOK and initialize for device type and address
Set up RDCONPKG
Construct CCW package for read
Call STCONSIO
STCONSIO
Get MRDEBLOK
If previous console I/O request outstanding, queue this request UTABLE–CIOREQ
If no requests outstanding, issue SIO for this request and queue this request UTABLE–CIOREQ
IOINT
Get MDDEBLOK
Get user's UTABLE
Call CONSINT to process interrupts for the device MIRA=CONSINT
Go to DISPATCH
DISPATCH
Eventually attempt to dispatch user that initiated the read
User has outstanding CPRQUEST
Process CPRQUEST return to location designated by the program that called RDCONS (return address in RDCONPKG)
CONSINT
Get CCW package
Process interrupts
Get read data and process for EDIT and UCASE if specified
Call CPSTACK
If more CCW packages in stack, start the next one (issue SIO)
CPSTACK
Put RDCONPKG in CPSTACK

FIGURE 30. Real Terminal SIO (Read)

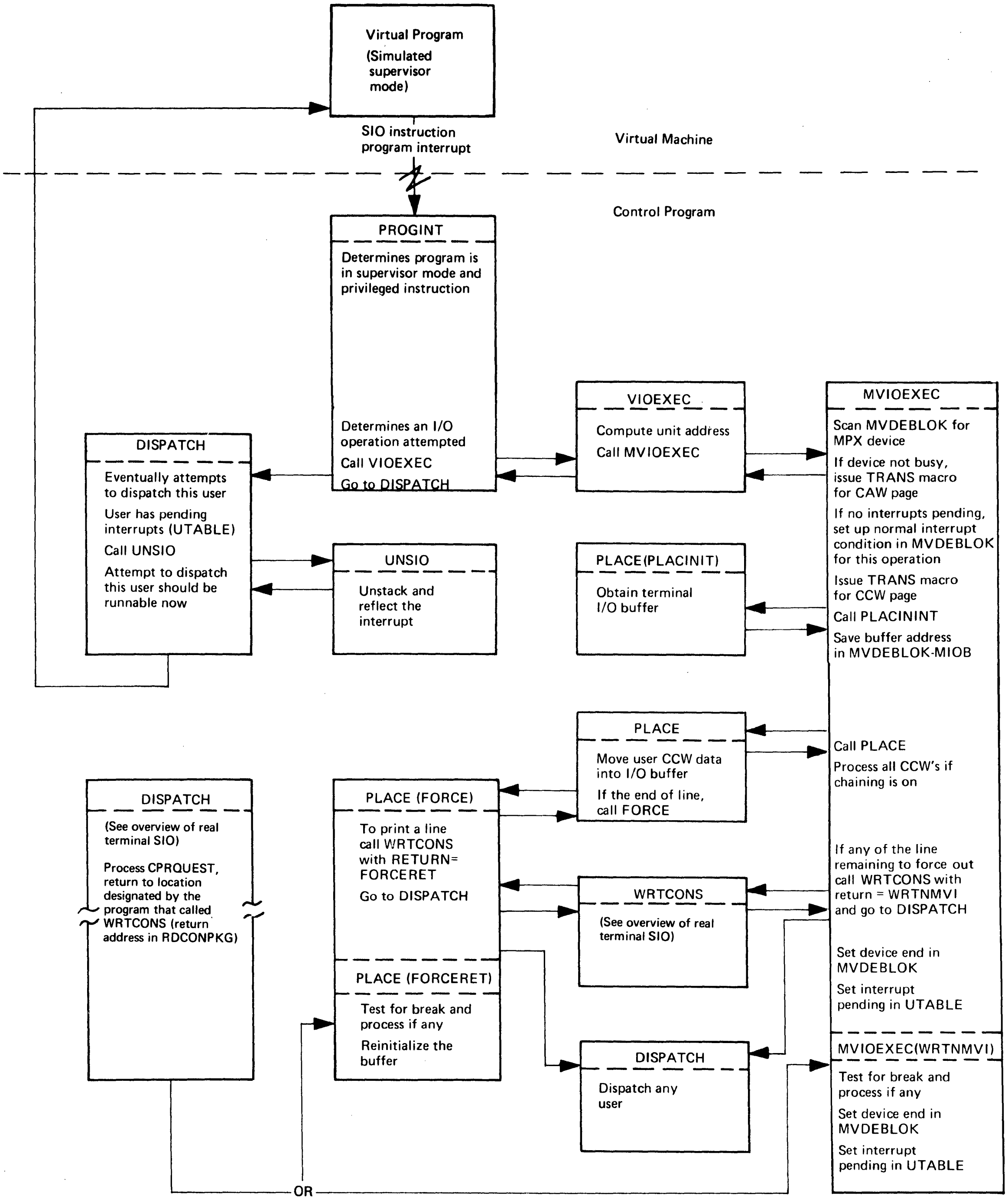

Virtual Program
(Simulated supervisor mode)
SIO instruction program interrupt
Virtual Machine
Control Program
PROGINT
Determines program is in supervisor mode and privileged instruction
Determines an I/O operation attempted
Call VIOEXEC
Go to DISPATCH
VIOEXEC
Compute unit address
Call MVIOEXEC
MVIOEXEC
Scan MVDEBLOK for MPX device
If device not busy, issue TRANS macro for CAW page
If no interrupts pending, set up normal interrupt condition in MVDEBLOK for this operation
Issue TRANS macro for CCW page
Call PLACININT
Save buffer address in MVDEBLOK-MIOB
Call PLACE
Process all CCW's if chaining is on
If any of the line remaining to force out call WRTCONS with return = WRTNMVI and go to DISPATCH
Set device end in MVDEBLOK
Set interrupt pending in UTABLE
MVIOEXEC(WRTNMVI)
Test for break and process if any
Set device end in MVDEBLOK
Set interrupt pending in UTABLE
DISPATCH
Eventually attempts to dispatch this user
User has pending interrupts (UTABLE)
Call UNSIO
Attempt to dispatch this user should be runnable now
UNSIO
Unstack and reflect the interrupt
PLACE(PLACINIT)
Obtain terminal I/O buffer
PLACE
Move user CCW data into I/O buffer
If the end of line, call FORCE
DISPATCH
(See overview of real terminal SIO)
Process CPRQUEST, return to location designated by the program that called WRTCONS (return address in RDCONPKG)
PLACE (FORCE)
To print a line call WRTCONS with RETURN= FORCERET
Go to DISPATCH
PLACE (FORCERET)
Test for break and process if any
Reinitialize the buffer
WRTCONS
(See overview of real terminal SIO)
DISPATCH
Dispatch any user
OR

FIGURE 31. Virtual Terminal SIO (Write)

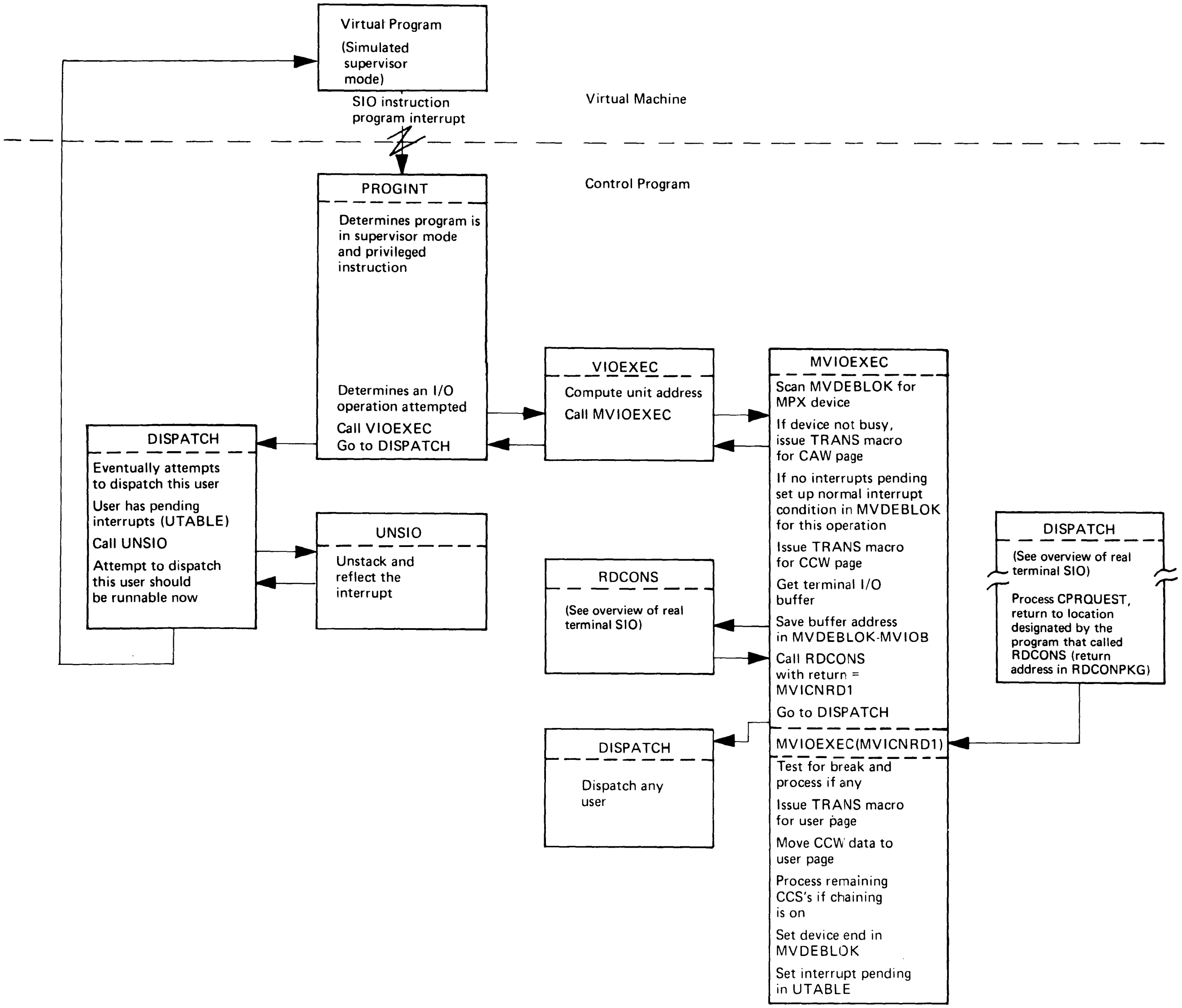

Virtual Program
(Simulated supervisor mode)
SIO instruction program interrupt
Virtual Machine
Control Program
PROGINT
Determines program is in supervisor mode and privileged instruction
Determines an I/O operation attempted
Call VIOEXEC
Go to DISPATCH
VIOEXEC
Compute unit address
Call MVIOEXEC
MVIOEXEC
Scan MVDEBLOK for MPX device
If device not busy, issue TRANS macro for CAW page
If no interrupts pending set up normal interrupt condition in MVDEBLOK for this operation
Issue TRANS macro for CCW page
Get terminal I/O buffer
Save buffer address in MVDEBLOK-MVIOB
Call RDCONS with return = MVICNRD1
Go to DISPATCH
DISPATCH
Eventually attempts to dispatch this user
User has pending interrupts (UTABLE)
Call UNSIO
Attempt to dispatch this user should be runnable now
UNSIO
Unstack and reflect the interrupt
RDCONS
(See overview of real terminal SIO)
DISPATCH
(See overview of real terminal SIO)
Process CPRQUEST, return to location designated by the program that called RDCONS (return address in RDCONPKG)
DISPATCH
Dispatch any user
MVIOEXEC(MVICNRD1)
Test for break and process if any
Issue TRANS macro for user page
Move CCW data to user page
Process remaining CCS's if chaining is on
Set device end in MVDEBLOK
Set interrupt pending in UTABLE

FIGURE 32. Virtual Terminal SIO (Read)

SIO on a Virtual Multiplexer Channel

When MVIOEXEC determines that an SIO operation has been executed, the page handling routine (PAGTRANS) is called, via the TRANS macro, to obtain the user's virtual CCW list starting address (from the virtual CAW), and an I/O task block and buffer area are created. If an interruption (device end or channel end) is pending on the virtual device, an indicator is set in the multiplexer virtual device block (MVDEBLOK), and an exit is taken to VIOEXEC.

If no interruptions are pending, MVIOEXEC determines the type of device for which the SIO operation is requested. If the device is a printer or card punch, the user's CCW data must be packed (via the PACK routine) and placed into a spooling buffer (829 bytes), preparatory to being written into a spooling area on a direct access device. If the device is a card reader, data will be read from a direct access spooling area into a buffer; it must then be unpacked (by means of the UNPACK routine) to be made available to the user.

If the device is a user's terminal, the virtual CCW is saved, and the type of command (SENSE,NOP,ALARM,READ, or WRITE) must be determined; special processing is required for each command.

Following is a summary of the processing required for SIO operations for devices attached to the multiplexer channel:

SIO - Printer or Punch: For an SIO operation to a printer or card punch, MVIOEXEC does the following:

Initializes MVIBUFF, which contains a buffer for user's packed CCW data, CCW's to write the buffer onto a direct access device, and control information.

Calls PAGTRANS to bring into core the pages which contain the user's CCW data.

Calls PACK to compress the user's CCW data.

Enters the packed data into the buffer; when the buffer is filled, it is written into a spooling file on a direct access device by calling QUERIO.

Calls the multiplexer real I/O executive program (MRIOEXEC) to perform the input-output operation when the spooling file is closed. (The file may be closed by the user including an illegal CCW or issuing a CLOSE command from console functions.) If the real printers and punches on the system are busy, the closed spooled file is placed in chains starting from PRINTERS or PUNCHES.

Note: If CP console function XFER had been previously initiated, no real deck is punched. Instead, the spooled card deck is set up as an input deck in the virtual card reader for the userid specified in the XFER command.

SIO - Card reader: For an SIO operation on a card reader, MVIOEXEC does the following:

Initializes MVIBUFF, which contains an area into which the user's packed data will be read, CCW's to read the data from a direct access device spooling area, and control information. The READER chain on the system is scanned to find a spooled file for the user. If none are found, the SIO is indicated to have terminated by an intervention-required condition.

Calls QUERIO to read packed data (80-byte card image records packed into 829-byte physical records) from the direct access spooling file associated with the user's ID.

Calls PAGTRANS to bring the required user's pages into core storage.

Moves data into the specified area in the user's page(s).

SIO - User terminal:

Sense Command - User terminal: For a SENSE command on a user terminal, MVIOEXEC does the following:

Calls PAGTRANS to determine the address of the area into which the sense information will be placed.

Moves sense information from the multiplexer virtual device block into the provided area.

NOP Command - User terminal: For a NOP command on a user terminal, MVIOEXEC does the following:

Scans virtual CCW flags. If the CC or CD flag is on, the next CCW in the chain is examined.

Indicates a pending multiplexer interruption in the user's UTABLE if neither the CC nor the CD flag is on.

WRITE Command - User terminal: For a WRITE command to a user terminal, MVIOEXEC does the following:

Calls PAGTRANS to obtain the user's pages associated with the I/O transfer.

Moves the user's data into the output buffer.

Processes each successive CCW in the chain if the chained data flag is on. All chained data is moved into the output buffer.

Calls WRTCONS to write the data contained in the output buffer on the user's terminal. (Control is given to DISPATCH until the real WRITE operation is completed.)

READ Command - User terminal: For a READ command for a user terminal, MVIOEXEC does the following:

Calls FREE to obtain an input buffer.

Calls RDCONS to read data into the input buffer from the user terminal. (Control is given to DISPATCH until the real READ operation is completed.)

Calls PAGTRANS to obtain the address of the user's pages into which data will be placed.

Moves data from the input buffer to the specified areas in the user's pages.

Processes virtual CCW flags.

Processes each successive CCW in the chain if the chained data or chained command flag is on.

ALARM Command - User terminal: For an ALARM command for a user terminal, MVIOEXEC does the following:

Calls WRTCONS to write an "alarm" message on the user terminal (control is given to DISPATCH until the ALARM is completed).

Processes each successive CCW in the chain if the chained data or chained command flag is on.

When special processing for each type of command is completed, MVIOEXEC performs the following:

Checks for command chaining and processes the next command if on.

Calls PAGTRANS to determine the address of the virtual CSW, stores the virtual CSW, and removes the I/O wait indication from the user's UTABLE.

Calls BREAK if the attention key was activated during a read or write operation.

Returns control to the virtual machine I/O executive program (VIOEXEC).

TIO on a Virtual Multiplexer Channel

When MVIOEXEC determines that a TIO operation has been requested, the multiplexer virtual device block (MVDEBLOK) is examined to determine whether an interruption (channel end or device end) is pending for the virtual device.

If a channel end interruption is pending, the channel end indication is removed from the MVDEBLOK. If a device end interruption is pending, the device end indication is removed, and device end is indicated in the virtual CSW. For either type of interruption, a condition code of 1 is set in the virtual PSW. If no interruptions are pending, the condition code remains zero.

When the condition code has been set, the normal MVIOEXEC exit is taken:

The virtual CSW is stored, and the I/O wait indication is removed from the user's UTABLE.

The BRKWR or BRKRD routines are called if required or if an "attention" is seen.

Control is returned to the virtual machine I/O executive program (VIOEXEC).

TCH on a Virtual Multiplexer Channel

When MVIOEXEC determines that a TCH operation has been requested, a SCAN macro is issued to obtain the channel status. If the channel is not operational (that is, no channel by that number is defined in the virtual machine), a condition code of 3 is set. If any interruptions are pending for the channel, a condition code of 1 is set. If the channel is busy, a condition code of 2 is set. If no interruptions are pending for the channel and it is not busy, a condition code of 1 is set.

When TCH processing is completed, control is returned to the virtual machine I/O executive program (VIOEXEC).

HIO on a Virtual Multiplexer Channel

When MVIOEXEC determines that an HIO operation has been requested, it sets the user's condition code to zero if there is an interruption pending, and to 1 if there is no interruption pending.

Pseudo Timer Device - TIMR

When MVIOEXEC detects an SIO to a virtual multiplexer device type TYPTIMR, it fills in the specified read buffer with the time of day (hh/mm/ss), date (mm/dd/yy), total virtual CPU time (VTOTTIME), and total CPU time (TIMEUSED) used since logging in. No actual I/O operation is performed, and no real device is associated with this operation. There is no interrupt from this device after the data is transferred. The SIO ends with a condition code of zero for a successful operation, or 3 if the pseudo timer does not exist in the user's virtual machine configuration.

PROCESSING DEDICATED MULTIPLEXER DEVICES

If multiplexer devices are dedicated to a particular user, they are structured and handled by CP-67 as though they were selector type devices. Thus a virtual SIO to a dedicated printer, for instance, would go through the selector I/O processing logic and not through the multiplexer spooling logic. Any CP-67 multiplexer device can be dedicated to a user at the time he logs in to CP-67 or through the ATTACH capability.

When a multiplexer device is attached to a user on a nonshared (dedicated) basis, a restructuring of the real and virtual control blocks is required. As an example, suppose the operator is attaching the real printer to a user as a dedicated device. The real printer is "030" and the virtual address is "00E". The user cannot already have a device of address 00E in either his virtual selector devices or multiplexer devices. The real multiplexer device block (MRDEBLOK) for the printer 030 is located. If the printer is not busy or already attached, the MRDEBLOK is marked as "dedicated". A routine called DEDICATE then creates a real selector channel, control unit, and device block for the printer, and chains these blocks with the other real blocks (RCHBLOK, RCUBLOK, and RDEVBLOK). Then virtual selector channel, control unit, and device blocks are created and are linked to the newly created real blocks by VPNTREAL in the

VDEVBLOK. Since the device is now structured as a selector device, I/O simulation and interrupt handling will be as outlined in "Processing User Selector Channel I/O Requests". This structure will be maintained until the user detaches the dedicated device or logs out. In either case, the logout routine (USEROFF) will detect a dedicated device that was structured using DEDICATE and will call RELEASE to free the real channel, control unit, and device blocks and to free (undedicate) the device on the multiplexer (MRDEBLOK) chains. The device (printer, for example) is now available for CP-67 spooled output.

PROCESSING VIRTUAL 2702 LINES

Virtual 2702 lines in a user's machine require special consideration because of the nature of the teleprocessing applications that these virtual machines may run.

For a virtual machine with nondedicated virtual 2702 lines defined in the CP-67 directory, the virtual I/O blocks are built as selector I/O blocks. Every virtual 2702 line has its own virtual selector channel, control unit, and device block (VCHBLOK, VCUBLOK, and VDEVBLOK). The blocks are structured this way so that a dedicated 2702 line can be linked to them when linkage is initiated by DIAL (see the next section for DIAL processing). In order to properly process a DIAL request, the virtual 2702 block must be initialized. This is under control of the virtual machine. When the virtual machine issues an "enable" sequence to a virtual 2702 line, CP-67 performs all the normal handling for a user selector I/O request with one major exception. Since there is no real device on which to perform the I/O operation when the "enable" is issued, the IOTASK created by VIOEXEC is held waiting for a DIAL request. The user is given the condition that the I/O is started, but it will not complete, of course, until a DIAL is handled, simulating a call completion. The "enable" CCW is changed to a "write circle C" to effect line behavior as though a call had been completed. Any SAD commands are made NOP since the real line has already been set by CP-67 and the SAD number could be different for virtual machines. The module CCWTRAN detects I/O to virtual 2702 lines and changes the "enable" and SAD commands. CCWTRAN also retains the IOTASK (pointed to by VPNTREAL in the VDEVBLOK for the virtual 2702 line) and indicates to VIOEXEC (which called CCWTRAN) not to call QUEVIO since no real device yet exists.

Figure 33 illustrates the processing of virtual 2702 lines before and after a DIAL console function is issued.

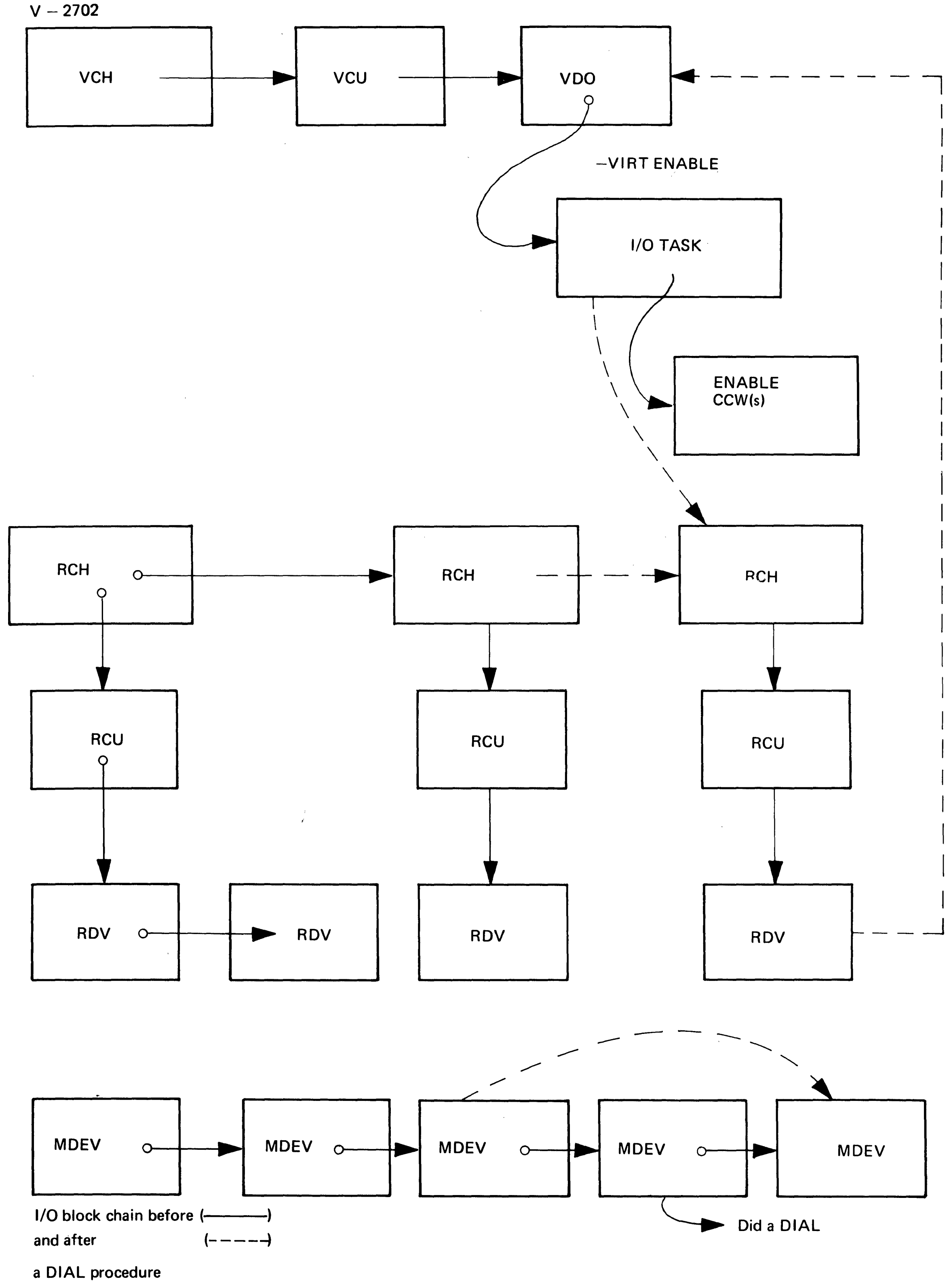


FIGURE 33. Processing a Virtual 2702 Line

PROCESSING A DIAL REQUEST

The DIAL method of attaching to a virtual machine is an alternative to LOGIN with a unique userid. After making contact with the computer and receiving the message "CP-67 online", a user can enter "dial xxxx", where xxxx is the userid of a virtual machine with virtual 2702 lines. The DIAL request can be considered as a self-initiated request to "attach" the terminal to the desired virtual machine on a dedicated basis. The module DIAL will search for an "enabled" (virtually) 2702 line that is not in use on the requested virtual machine. When one is found, DIAL will call DEDICATE to attach the terminal that entered "DIAL" to the virtual machine. DEDICATE will mark the terminal from the MRDEBLOK chain as dedicated and create real selector channel, control unit, and device blocks. These will be linked to the already existing virtual selector channel, control unit, and device blocks. The IOTASK that was being held (from the "enable" sequence) is now allowed to proceed by DIAL calling QUEVIO. I/O interrupts and subsequent I/O requests to that virtual 2702 line will now be handled in exactly the same fashion as dedicated multiplexer devices. However, in order to expedite the efficient handling of dedicated (DIALed) 2702 lines, a further step is taken. In CCWTRAN, detection is made when a virtual machine issues a disable to a DIALed 2702 line. When a virtual "disable" is detected, CCWTRAN creates a dummy CSW with normal completion and calls VIRA to process what appears to be the completion of the "disable"; however, no I/O operation is performed. CCWTRAN then calls RELEASE to free the real selector blocks for the dedicated 2702 line. RELEASE will set processing in effect to go to OFFHANG in CONSINT. OFFHANG writes a message to the terminal indicating that the terminal is now under CP-67 control. OFFHANG will then either disable the line and then reenable (as done in LOGOUT) or proceed to IDENT2, which will start with "CP-67 online" and then wait for a LOGIN or a DIAL (as done in LOGOUT HOLD). The alternative is an installation option defined in CONSINT.

Special handling is also required if the virtual machine with virtual 2702 lines either detaches 2702 lines, does an HIO to "dialed" or "enabled" lines, or logs out of CP-67. Since "enabled" lines have IOTASK blocks pending, these must be released if the virtual line is to be considered no longer active. The VIOEXEC module has special code to handle an HIO to an "enabled" 2702 line. VIOEXEC will call VIRA to indicate that the "enable" has been halted. HIO to a "dialed" 2702 line is allowed to proceed in the normal fashion. The RELEASE module (in USEROFF) also has code to call VIRA, release the IOTASK block, and return the device to the MRDEBLOK chain.

Five special functions are provided by CP-67 to virtual machines; these functions either are not available on a real System/360 or normally would require operator intervention.

RPQ Timer - This is a special device type (TIMR) defined in the directory to provide time information to a virtual machine. The device can have any address, but for CMS it is defined as OFF. The virtual machine issues an SIO to the device with a "read" CCW using a 24-byte data area, which must not cross a page boundary. The following information is placed in the 24-byte data area.

Location	Data
0-7	date as MM/DD/YY
8-15	time as HH.MM.SS
16-19	value from TIMEUSED
20-23	value from VTOTTIME

There is no interrupt from the device after the data transfer.

Readable punch - This function is provided by the XFER console command. It routes the output from a user's spooled card punch to his or another user's spooled card file input. This function operates simply by SFBLOK routing. When a user issues CLOSE to a spooled punch, the SFBLOK is chained on the spool READER chain for the XFERed userid to read instead of being chained on the PUNCH chain for real punch output. The XFER for a printer works in a corresponding fashion.

Rereadable reader - This function is provided by the SET CARDSAVE ON console command. This function is accomplished by exception handling when a spool reader is CLOSEd by a user. Instead of scheduling the file for deletion from the spooling space, the SFBLOK is maintained on the READER chain so that the file can be reread from the beginning.

Wide card reader - There are two types of special spool card readers. The first type is a "wide" 2540 reader that allows the user to read more than 80 bytes from one "card". For instance, this capability is used by CMS when reading a spool reader, since that reader may contain 80-byte "card" files or 132-byte "card" files as a result of XFERed printer files. The second type is used to retrieve spool data in special format. This type (called RPRT or RPUN) is used to read the CP-67 system disk dump, for instance. It is a spooling reader that transfers to the user data areas (CCW addresses) up to 825 bytes of packed spool data. No attempt is made by CP-67 to analyze op-codes, lengths, or data. Thus, core dumps on disk can be read by a virtual machine having this type of card reader. RPRT is for reading files normally scheduled for printer output, and RPUN is for punch output.

DIAGNOSE - This privileged instruction cannot be simulated or allowed to execute. Accordingly, this op-code is used as a means of communication at the programming level between a virtual machine and various CP-67 functions. See "The Diagnose Instruction" for a description of each code allowed.

INTERRUPTION HANDLING

Five major types of interruptions must be handled by the Control Program: SVC interruptions, external interruptions, program interruptions, machine check interruptions, and I/O interruptions. Handling of I/O interruptions is discussed under the earlier heading "Processing Control Program I/O Requests". This section describes how the other four types are handled.

SVC Interruptions

When an SVC interruption occurs, the SVC interruption routine (SVCINT) is entered. If the machine is in problem mode, the type of interruption is placed into register 14, and the REFLECT routine is called to reflect the interruption back to the pseudo-supervisor (that is, the supervisor operating in the user's virtual machine). If the machine is in supervisor mode, the SVC interruption code is determined, and a branch is taken to the appropriate SVC interruption handler. See Figure 34 for a flowchart of the SVC Interrupt Handler.

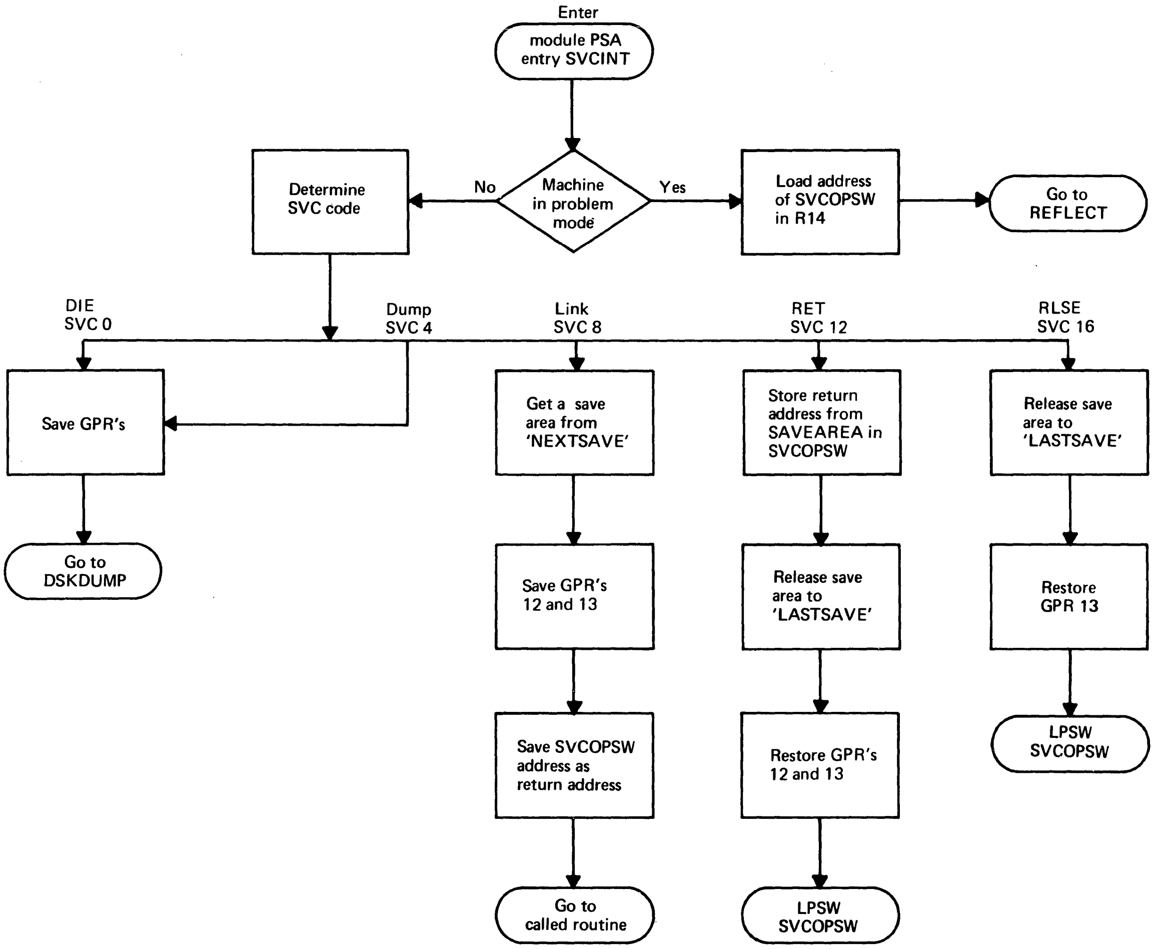


Figure 34. Flowchart of the SVC Interrupt Handler

SVC 0 - Impossible condition or fatal error: If the SVC interruption code is 0, the SVCDIE routine initiates an ABEND by going to the DSKDUMP routine.

SVC 4 - Reserved for future use.

SVC 8 - Link request (transfer control from calling routine to called routine specified by register 15): If the SVC interruption code is 8, the SVCLINK routine saves registers, sets up a new save area, inserts the contents of register 15 (the address of the routine for which the link is requested) into the SVCOPSW (and register 12), saves the old addressability in the save area, saves the old save area address in the new save area, and issues an LPSW instruction for the SVCOPSW to restart the Control Program at the linked address.

SVC 12 - Return request (transfer control from called routine to calling routine): If the SVC interruption code is 12, the SVCRET routine is entered to restore registers 12 and 13 (addressability and save area address saved by SVCLINK), places the user's return address (also saved in the area) back into the SVCOPSW, and returns control to the calling routine by loading the SVCOPSW.

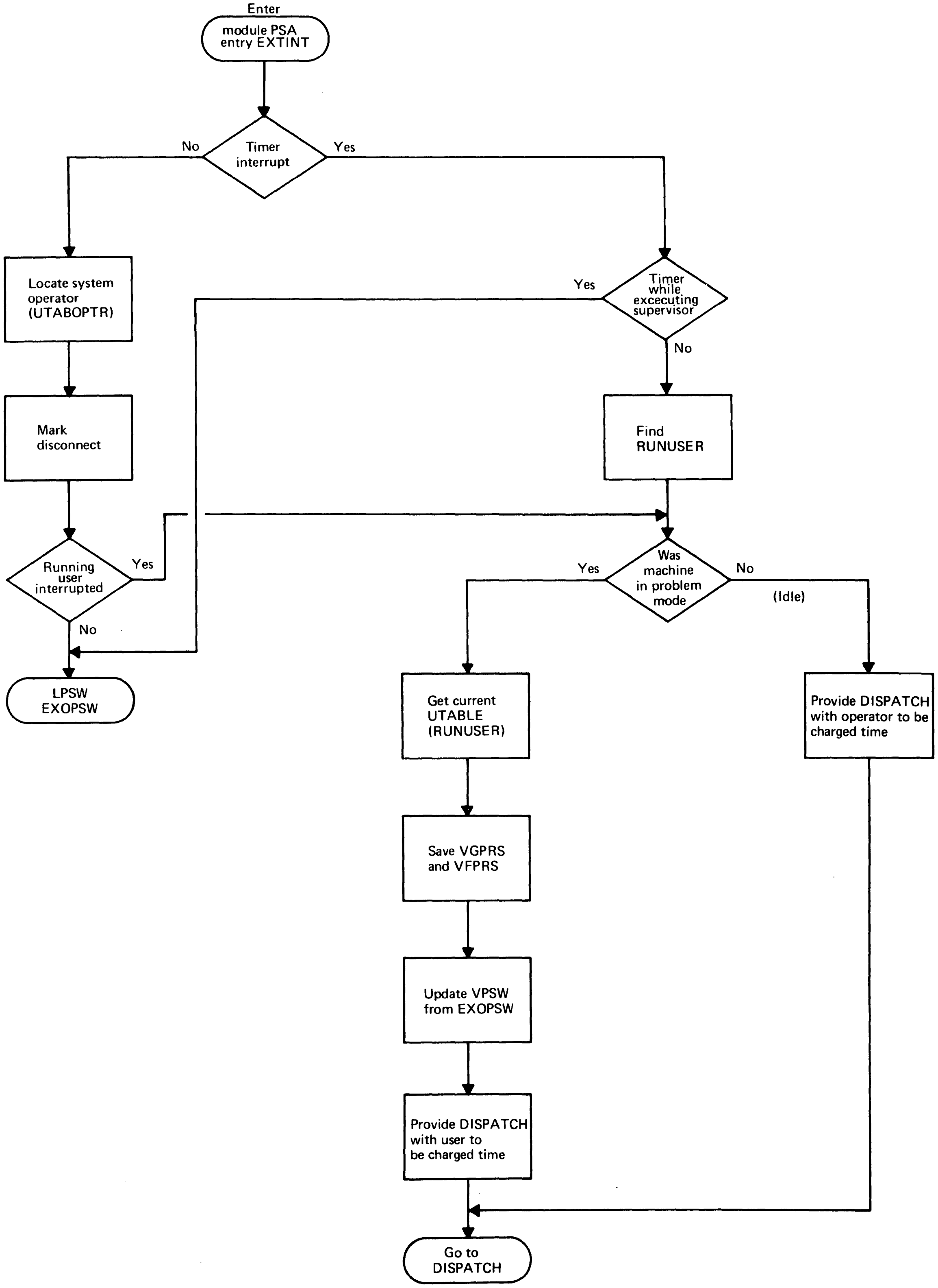
SVC 16 - Release current save area from the active chain (and thereby also remove linkage pointers to the calling routine): If the SVC interruption code is 16, the SVCRLSE routine releases the current save area by placing the address of the next higher save area in register 13, and returns control to the current routine by loading the SVCOPSW. This SVC is used by second level interrupt handlers to bypass returning to the first level handler under specific circumstances.

SVC 20 - Obtain a new save area: if the SVC interruption code is 20 the SVCGET routine places the address of the next available save area in register 13 and the address of the previous save area in the save area pointer field of the current save area.

There are 35 save-areas initially set up by CPINIT for use by the SVC linkage handlers. In addition, if the supply of available save areas drops to 0, the linkage handlers will call FREE to obtain one.

External Interruptions

When an external interruption occurs, the external interruption handler (EXTINT) is entered. See Figure 35 for an overview of the EXternal Interruption Handler.



Enter
module PSA entry EXTINT
Timer interrupt
No
Yes
Locate system operator (UTABOPTR)
Timer while excecuting supervisor
Yes
No
Mark disconnect
Find RUNUSER
Running user interrupted
Yes
No
Was machine in problem mode
Yes
No
(Idle)
LPSW EXOPSW
Get current UTABLE (RUNUSER)
Provide DISPATCH with operator to be charged time
Save VGPRS and VFPRS
Update VPSW from EXOPSW
Provide DISPATCH with user to be charged time
Go to DISPATCH

Figure 35. Overview of External Interruption Handler

If EXTINT is entered because of a timer interruption, the machine mode must be determined. If the machine was in supervisor mode, control is transferred to the main dispatcher and control routine (DISPATCH), which will become idle until another interruption occurs. If the machine is in problem mode, the address of the current user's UTABLE is obtained from RUNUSER. The user's current PSW (VPSW) is updated from the external interruption old PSW (EXOPSW), the address of the current UTABLE is placed in register 11, and control is transferred to DISPATCH.

If EXTINT is entered because of the operation of the console interrupt button (EXTERNAL), the following steps are taken: (1) the current system operator is located (via REALOPTR), and (2) his virtual machine is disconnected. He may now log in from another terminal. The operation of the console interrupt button is used to implement an alternate operator's console.

Program Interruptions

When a program interruption occurs, the program interruption handler (PROGINT) is entered. (See Figure 36 for an overview of PROGINT.) Program interruptions may result from (1) paging requests, (2) privileged operations (I/O), and (3) privileged operations (non-I/O). PROGINT determines the cause of the interruption by examining the interruption code. If (3) has occurred, PROGINT transfers control to PRIVLGED.

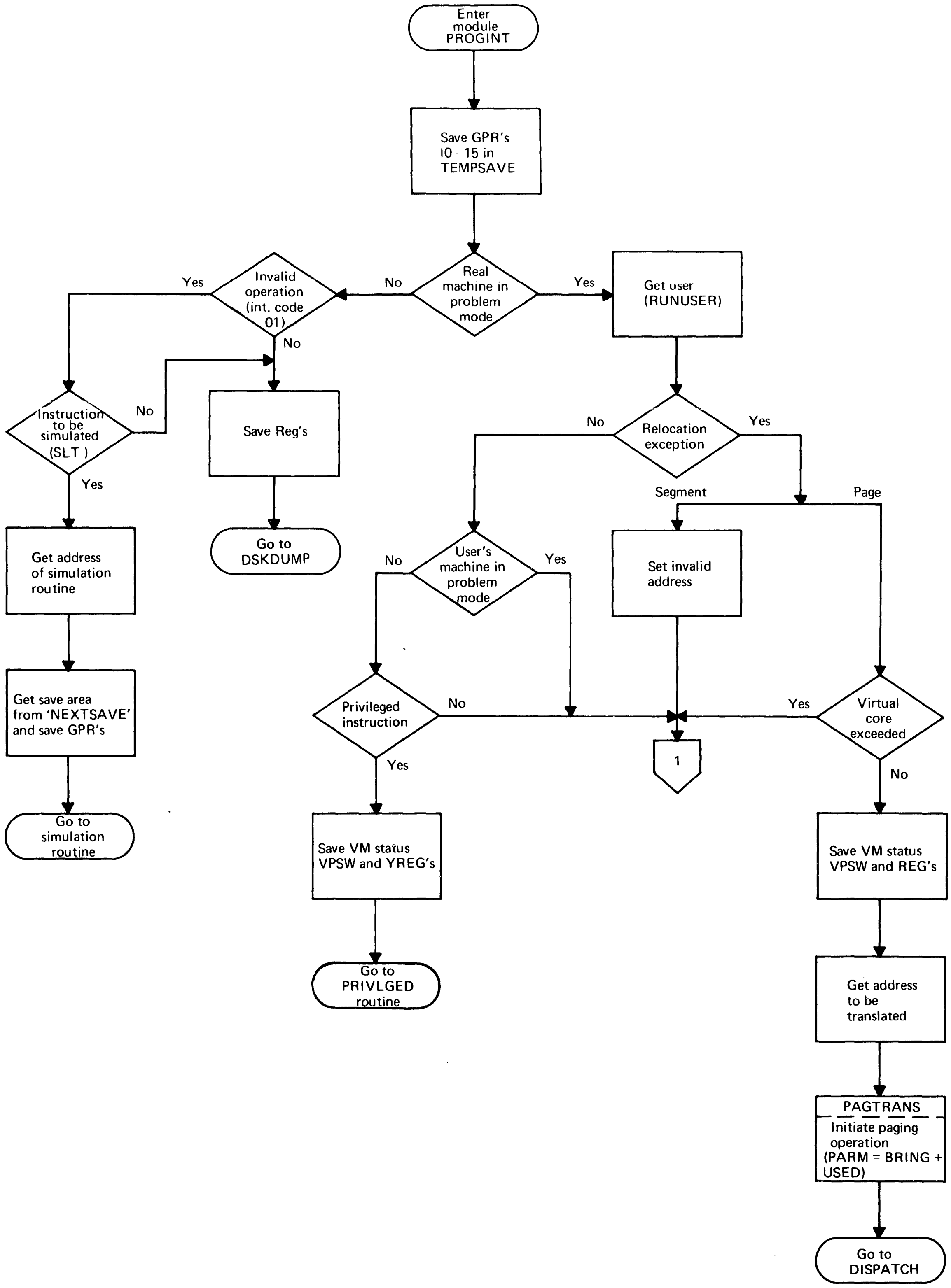

Enter module PROGINT
Save GPR's I0 - 15 in TEMPSAVE
Real machine in problem mode
No
Yes
Get user (RUNUSER)
Invalid operation (int. code 01)
Yes
No
Instruction to be simulated (SLT)
No
Yes
Save Reg's
Go to DSKDUMP
Get address of simulation routine
Get save area from 'NEXTSAVE' and save GPR's
Go to simulation routine
Relocation exception
No
Yes
Segment
Page
User's machine in problem mode
No
Yes
Set invalid address
Privileged instruction
No
Yes
1
Virtual core exceeded
Yes
No
Save VM status VPSW and YREG's
Go to PRIVLGED routine
Save VM status VPSW and REG's
Get address to be translated
PAGTRANS
Initiate paging operation (PARM = BRING + USED)
Go to DISPATCH

Figure 36. CP-67 Program and PRIVLGED Interrupt Handler (1 of 4)

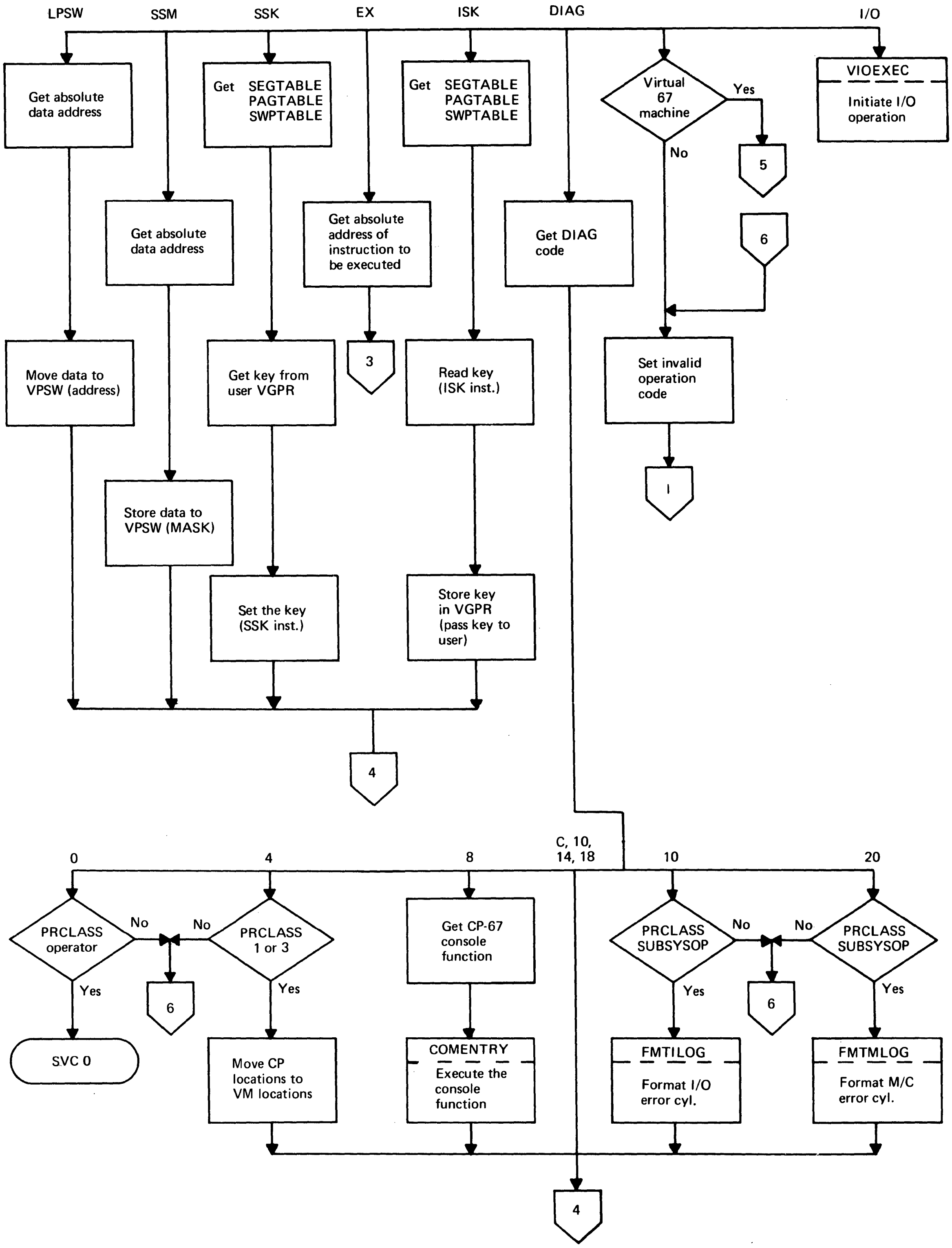

LPSW
SSM
SSK
EX
ISK
DIAG
I/O
Get absolute data address
Get SEGTABLE PAGTABLE SWPTABLE
Get SEGTABLE PAGTABLE SWPTABLE
Virtual 67 machine
Yes
No
VIOEXEC
Initiate I/O operation
5
Get absolute data address
Get absolute address of instruction to be executed
Get DIAG code
6
Move data to VPSW (address)
Get key from user VGPR
3
Read key (ISK inst.)
Set invalid operation code
1
Store data to VPSW (MASK)
Set the key (SSK inst.)
Store key in VGPR (pass key to user)
4
0
4
8
C, 10, 14, 18
10
20
PRCLASS operator
No
No
PRCLASS 1 or 3
Get CP-67 console function
PRCLASS SUBSYSOP
No
No
PRCLASS SUBSYSOP
Yes
6
Yes
Yes
6
Yes
SVC 0
Move CP locations to VM locations
COMENTRY
Execute the console function
FMTILOG
Format I/O error cyl.
FMTMLOG
Format M/C error cyl.
4

Figure 36. CP-67 Program and PRIVLGED Interrupt Handler (2 of 4)

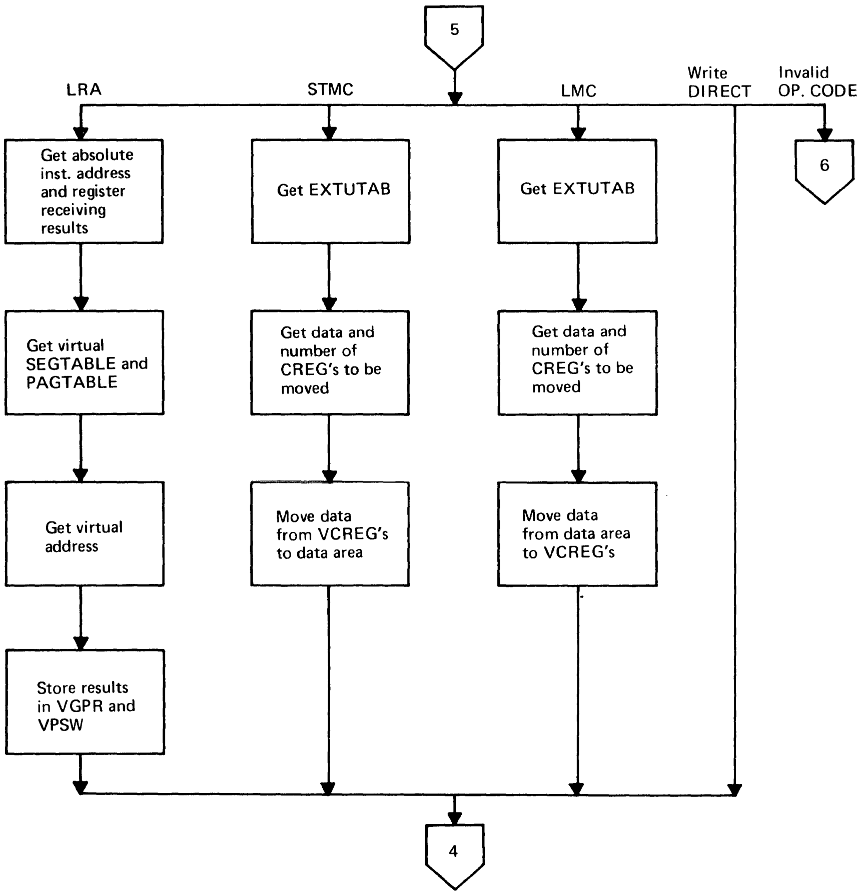
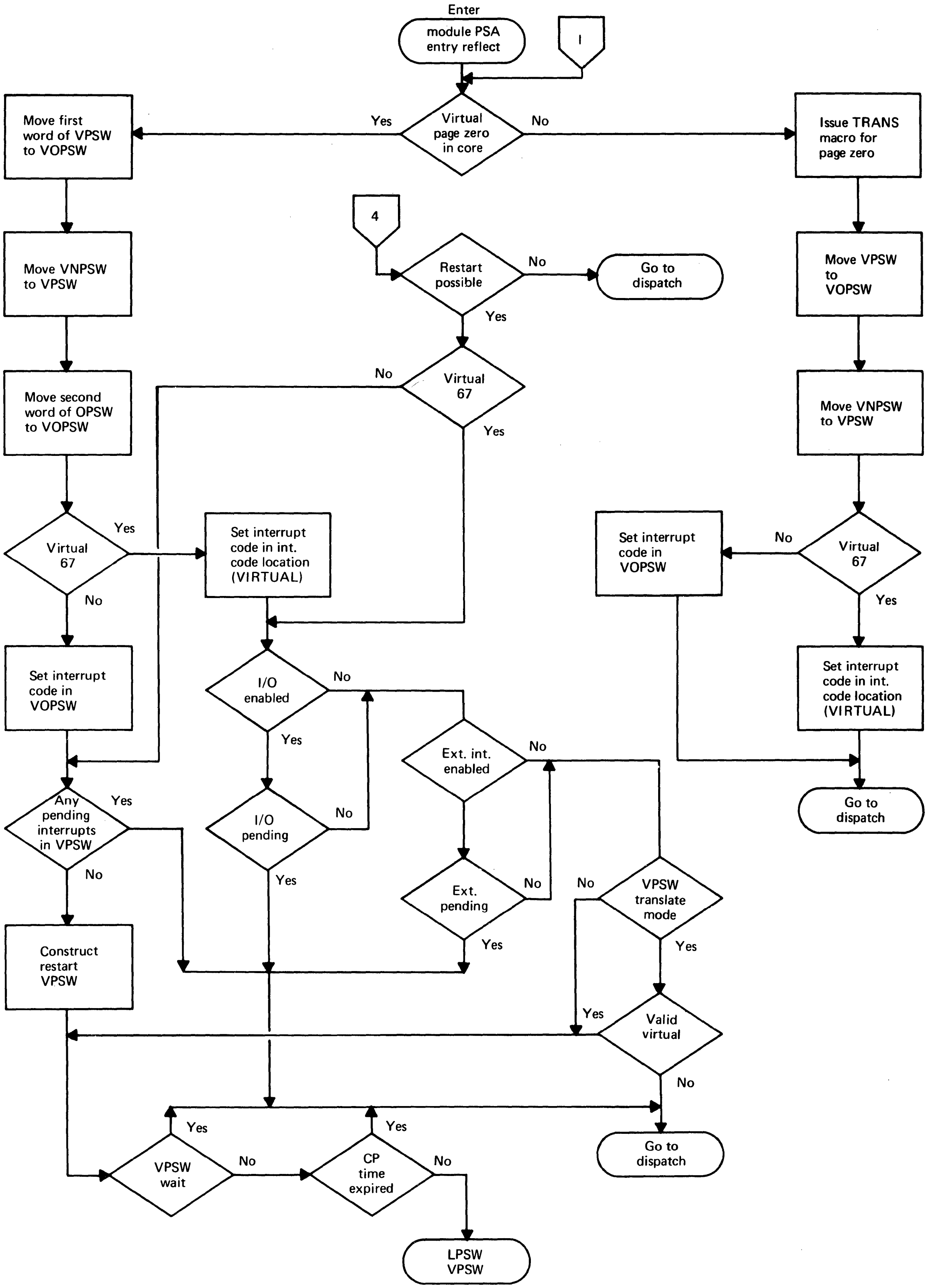

5
LRA
STMC
LMC
Write DIRECT
Invalid OP. CODE
Get absolute inst. address and register receiving results
Get EXTUTAB
Get EXTUTAB
6
Get virtual SEGTABLE and PAGTABLE
Get data and number of CREG's to be moved
Get data and number of CREG's to be moved
Get virtual address
Move data from VCREG's to data area
Move data from data area to VCREG's
Store results in VGPR and VPSW
4

Figure 36. CP-67 Program and PRIVLGED Interrupt Handler (3 of 4)

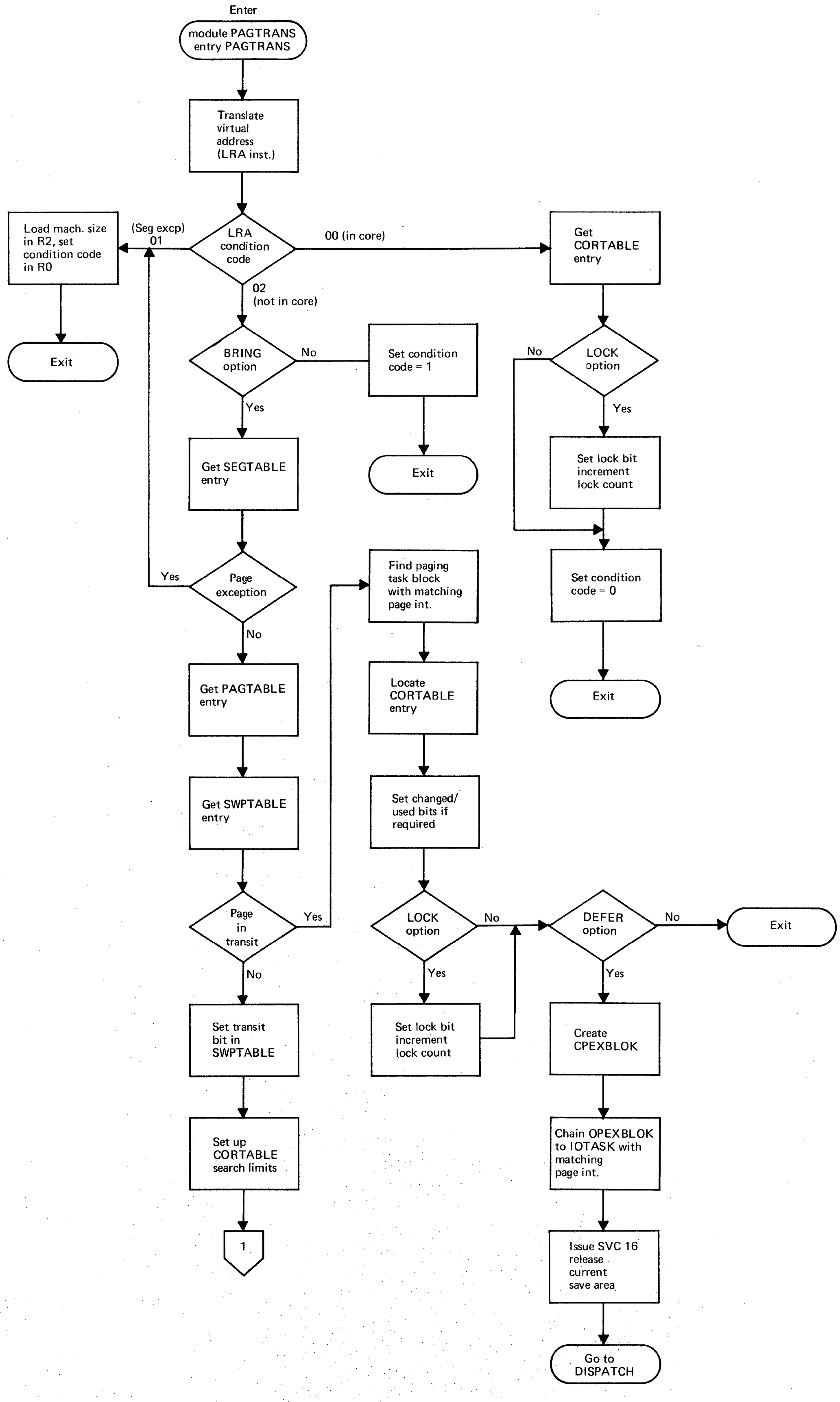


Fiqure 36. CP-67 Program and PRIVLGED Interrupt Handler (4 of 4)

Paging Interruptions

If the program interruption is caused by a paging request, and if the interruption occurs when a virtual 360/67 is running in extended mode with translation on, a special processing takes place. See "Running a Virtual 67" in the CP-67 Operator's Guide. Otherwise, PROGINT determines whether a segmentation error (a segment of the program missing) has occurred. If the interruption code resulted from a segmentation error, an invalid address interruption code is set, and the interruption is reflected to the user's virtual machine supervisor.

If a segmentation error has not occurred, the user's current PSW is updated from the program old PSW (PROPSW), the address of the current UTABLE is placed in register 11, and PAGTRANS is called to obtain the required page. When the paging operation is completed, control is returned to DISPATCH. (See Figure 37 for an overview of PAGTRANS.)



Enter
module PAGTRANS entry PAGTRANS
Translate virtual address (LRA inst.)
LRA condition code
(Seg excp) 01
Load mach. size in R2, set condition code in R0
Exit
00 (in core)
Get CORTABLE entry
02 (not in core)
BRING option
No
Set condition code = 1
Exit
Yes
Get SEGTABLE entry
Page exception
Yes
No
Get PAGTABLE entry
Get SWPTABLE entry
Page in transit
Yes
No
Set transit bit in SWPTABLE
Set up CORTABLE search limits
1
Find paging task block with matching page int.
Locate CORTABLE entry
Set changed/ used bits if required
LOCK option
No
Yes
Set lock bit increment lock count
DEFER option
No
Exit
Yes
Create CPEXBLOK
Chain OPEXBLOK to IOTASK with matching page int.
Issue SVC 16 release current save area
Go to DISPATCH
LOCK option
No
Yes
Set lock bit increment lock count
Set condition code = 0
Exit

Figure 37. Overview of PAGTRANS (1 of 3)

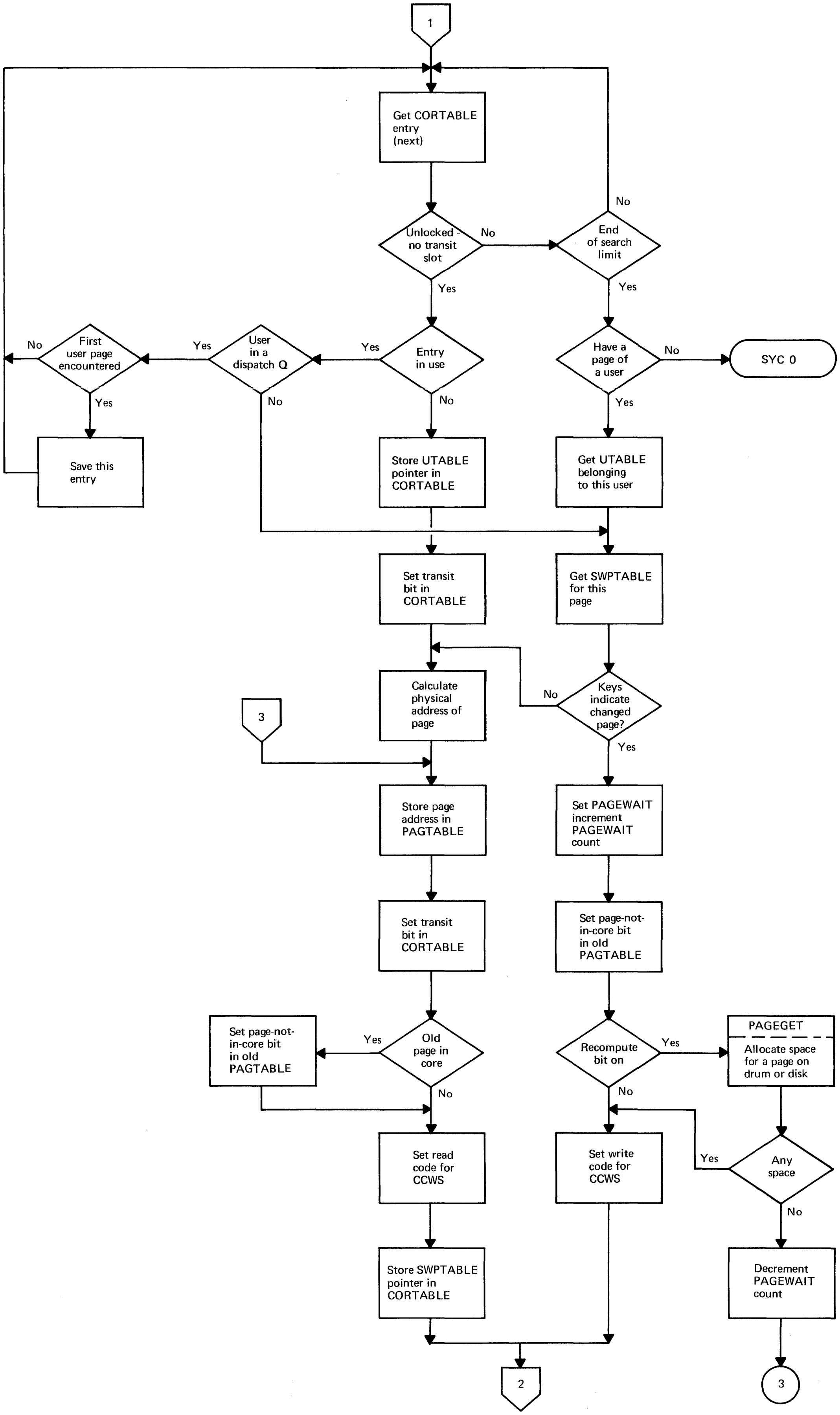

1
Get CORTABLE entry (next)
Unlocked - no transit slot
No
Yes
End of search limit
No
Yes
SYC 0
First user page encountered
No
Yes
User in a dispatch Q
Yes
No
Entry in use
Yes
No
Have a page of a user
No
Yes
Save this entry
Store UTABLE pointer in CORTABLE
Get UTABLE belonging to this user
Set transit bit in CORTABLE
Get SWPTABLE for this page
Calculate physical address of page
3
Keys indicate changed page?
No
Yes
Store page address in PAGTABLE
Set PAGEWAIT increment PAGEWAIT count
Set transit bit in CORTABLE
Set page-not-in-core bit in old PAGTABLE
Set page-not-in-core bit in old PAGTABLE
Old page in core
Yes
No
Recompute bit on
Yes
No
PAGEGET
Allocate space for a page on drum or disk
Any space
Yes
No
Set read code for CCWS
Set write code for CCWS
Store SWPTABLE pointer in CORTABLE
Decrement PAGEWAIT count
2
3

Figure 37. Overview of PAGTRANS (2 of 3)

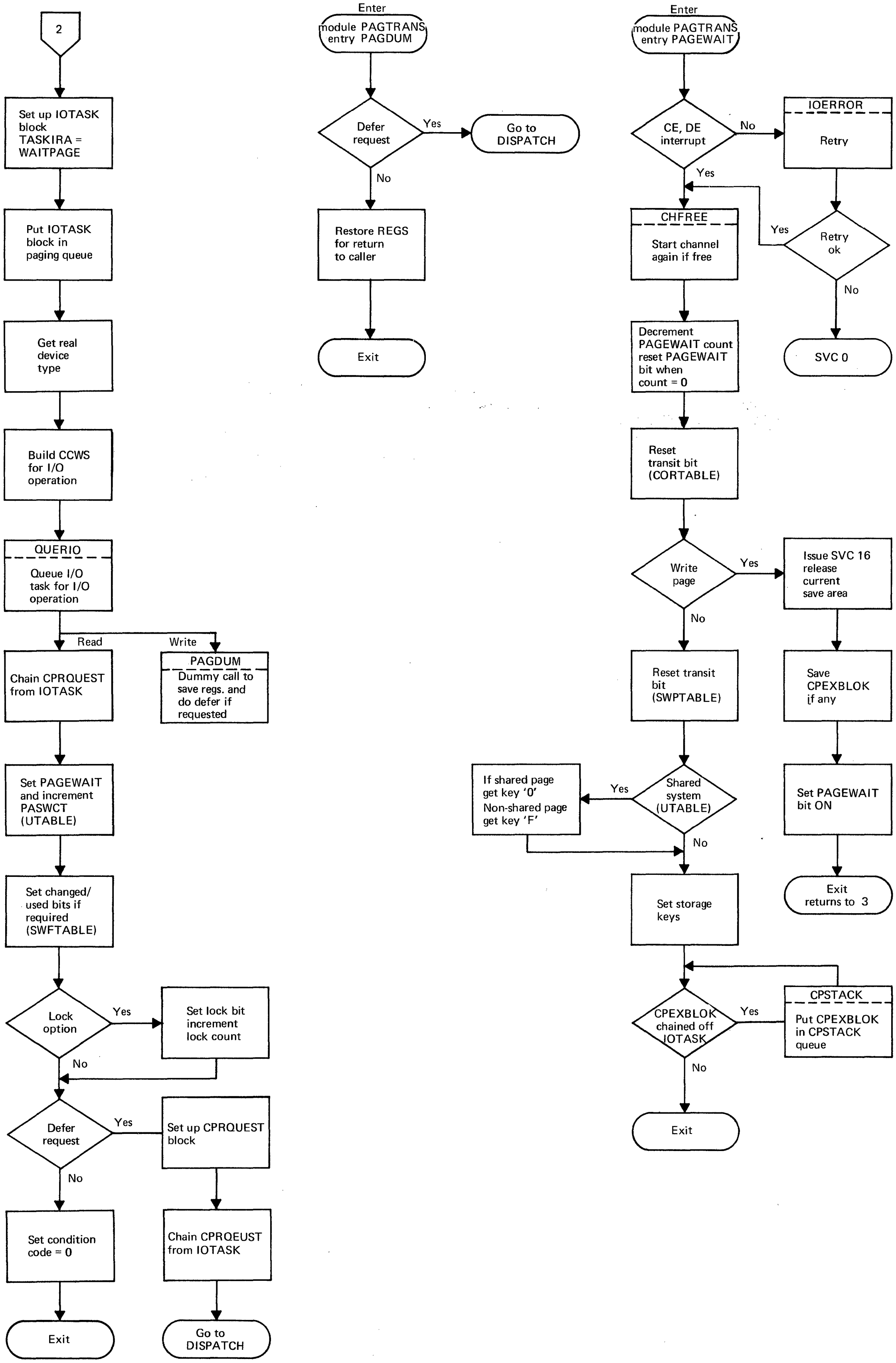

2
Set up IOTASK block TASKIRA = WAITPAGE
Put IOTASK block in paging queue
Get real device type
Build CCWS for I/O operation
QUERIO
Queue I/O task for I/O operation
Read
Write
Chain CPRQUEST from IOTASK
PAGDUM
Dummy call to save regs. and do defer if requested
Set PAGEWAIT and increment PASWCT (UTABLE)
Set changed/ used bits if required (SWFTABLE)
Lock option
Yes
Set lock bit increment lock count
No
Defer request
Yes
Set up CPRQUEST block
No
Set condition code = 0
Chain CPRQEUST from IOTASK
Exit
Go to DISPATCH
Enter
module PAGTRANS entry PAGDUM
Defer request
Yes
Go to DISPATCH
No
Restore REGS for return to caller
Exit
Enter
module PAGTRANS entry PAGEWAIT
CE, DE interrupt
No
IOERROR
Retry
Yes
CHFREE
Start channel again if free
Yes
Retry ok
No
SVC 0
Decrement PAGEWAIT count reset PAGEWAIT bit when count = 0
Reset transit bit (CORTABLE)
Write page
Yes
Issue SVC 16 release current save area
No
Reset transit bit (SWPTABLE)
Save CPEXBLOK if any
If shared page get key '0' Non-shared page get key 'F'
Yes
Shared system (UTABLE)
No
Set PAGEWAIT bit ON
Set storage keys
Exit returns to 3
CPEXBLOK chained off IOTASK
Yes
CPSTACK
Put CPEXBLOK in CPSTACK queue
No
Exit

Figure 37. Overview of PAGTRANS (3 of 3)

Privileged Operation Interruptions

If the program interruption is caused by the pseudo-supervisor issuing a privileged instruction, PRIVLGED obtains the address of the privileged instruction and determines the type of operation requested.

For I/O instructions, PRIVLGED calls the virtual I/O executive program (VIOEXEC). PRIVLGED simulates valid non-I/O privileged instructions and returns control to DISPATCH. For invalid privileged instructions, the routine sets an invalid interruption code and reflects the interruption to the pseudo-supervisor.

The non-I/O privileged instructions that are simulated are LPSW, SSM, SSK, ISK, and DIAG. For the "Virtual 67" option, the privileged instructions LRA, STMC, and LMC are also simulated.

The Diagnose Instruction

The diagnose instruction (DIAG) has special handling under CP-67. The diagnose command is used for communication between a virtual machine and the Control Program, CP-67. The machine-coded format for the diagnose command is:

```
-------------------------------
|  83  |  R1  |  R2  |  CODE  |
-------------------------------
```

The "CODE" is a base value that is used to select a particular specialized CP function. The codes currently assigned and their associated functions are:

CODE	FUNCTION
0	Dump CP core
4	Fetch CP location
8	Virtual console function
C	Pseudo timer
10	Release pages
14	Reserved for future IBM use
18	Disk I/O
1C	Clear I/O error recording
20	Clear M/C error recording
24-FC	Reserved for future IBM use

Note: User defined DIAG Codes:

X'00' through X'FC'	Reserved for IBM use
X'100' through X'1FC'	Reserved for users

Diag code should always be a multiple of 4.

See the module PRIVLGED for analysis and/or implementation of these functions.

The execution of diagnose code 0, dump system, causes a system abend by issuing SVC 0 (dump). This can only be executed by a privilege class A user. The format of the command is:

```
 ____________
|            |
|  83000000  |
|____________|
```

The execution of diagnose code 4, fetch CP locations, can only be issued by users with privilege class A or B. The format of the command is:

```
 ________________
|                |
|  83 R1 R2 0004 |
|________________|
```

R1 contains the virtual address of a list of CP (real) addresses.

R1+1 contains a count of entries in the list.

R2 contains the virtual address of the result field that will hold the values retrieved from the CP (real) locations.

The execution of function 8, virtual console function, allows a virtual machine to perform CP-67 console functions. The format of the diagnose command is:

```
 ________________
|                |
|  83 R1 R2 0008 |
|________________|
```

where R1 is a register that contains the address (virtual) of the CP console function command and parameters, and R2 is a register that contains the length of the associated console function input, up to 132 characters.

The following example will illustrate the virtual console function:

```
                  LA      R6,CPFUNC
                  LA      R10,CPFUNCL
                  DC      X'83',X'6A',XL2'0008'
                  .
                  .
         CPFUNC   DC      C'QUERY FILES'
         CPFUNCL  EQU     *-CPFUNC
```

The output of the console function is to the user's terminal, and then execution continues. Any valid and authorized console function can be executed in this manner.

A completion code is returned to the user as a value in the register specified in R2. Code 0 is normal, 4 is invalid command, and 8 is bad argument. Other condition codes may be used by processing routines in CP-67. LINK, for example, returns several codes to indicate device status (see LINK module).

Diagnose code C - pseudo timer. The format of the command is:

```
 ____________________
|                    |
|  83 R1 00 00 0C    |
|____________________|
```

R1 contains the virtual address that will receive 24 bytes of data in a format identical to the SIO to the pseudo-timer device (for example, 'OFF' in CMS). This data is provided by 'diagnose' as a faster method than SIO.

Diagnose code 10 - release pages. The format of the command is:

```
 __________________
|                  |
|  83 R1 R2 0010   |
|__________________|
```

R1 contains the virtual address of the first page to be released and R2 contains the virtual address of the last page to be released. Any of the virtual pages in real core or auxilliary storage are released.

Diagnose code 14 - reserved.

Diagnose code 18 - Disk I/O. The format of the command is:

```
 __________________
|                  |
|  83 R4 R8 0018   |
|__________________|
```

R4 contains the device address of the disk.

R8 points to a standard CCW chain to Read or Write the disk record of up to 4096 bytes.

Standard CCW string:

```
        SEEK,A,CC,6
        SRCH,A+2,CC,5
        TIC,*-8,0,0
        RD or WRT,DATA,cc,<4096
        NOP,0,SILI,1
     A  SEEK and SRCH arguments
```

The execution of diagnose code 1C, clear I/O recording, can only be issued by a privilege class C user. This code calls the FMTILOG routine to clear the I/O error recording data on disk. The format of the command is:

```
 ______________
|              |
|   8300001C   |
|______________|
```

The execution of diagnose code 20, clear MC recording, can only be issued by privilege class C user. This code calls the FMTMLOG routine to clear the machine check error recording data on disk. The format of the command is:

```
 ______________
|              |
|   83000020   |
|______________|
```

Machine Check Interruptions

When a machine check occurs in supervisor mode (CP-67 nucleus), a message is printed to the operator, the alarm is rung, and the system will ABEND with a dump.

When a machine check occurs in problem (user) mode, a message is typed on the operator's console, and a message is sent to the affected user. The user's machine is placed in console function mode. If the user enters "BEGIN", his machine will take a "machine check" by CP loading his machine check new PSW. CP-67 and other users are not affected.

Machine Check Error Recording Routine - MCKERR

See Figure 38 for an overview of the Machine Check Interruption Handler.

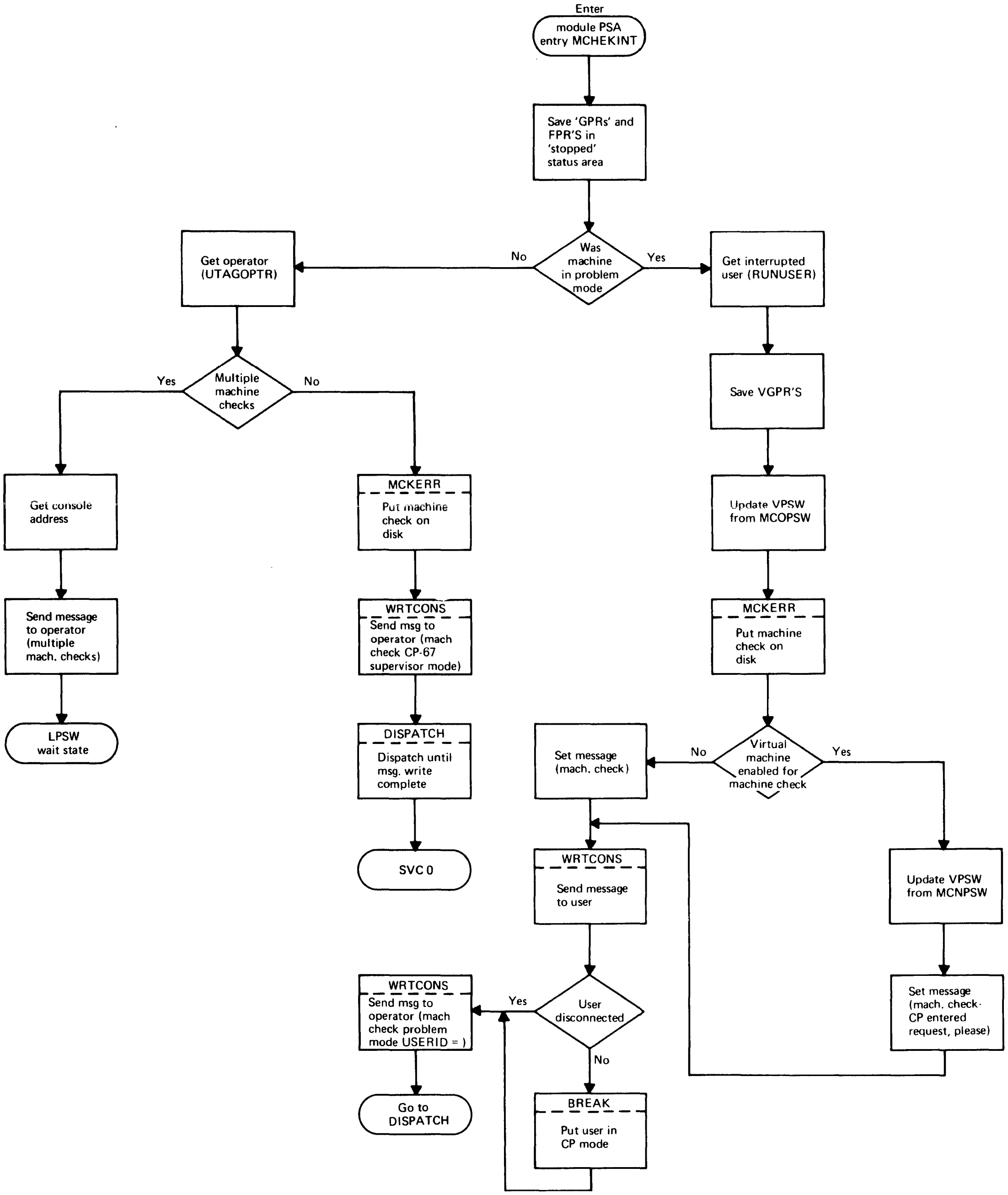

Enter
module PSA entry MCHEKINT
Save 'GPRs' and FPR'S in 'stopped' status area
Was machine in problem mode
No
Yes
Get operator (UTAGOPTR)
Get interrupted user (RUNUSER)
Multiple machine checks
Yes
No
Save VGPR'S
Get console address
MCKERR
Put machine check on disk
Update VPSW from MCOPSW
Send message to operator (multiple mach. checks)
WRTCONS
Send msg to operator (mach check CP-67 supervisor mode)
MCKERR
Put machine check on disk
LPSW wait state
DISPATCH
Dispatch until msg. write complete
Set message (mach. check)
Virtual machine enabled for machine check
No
Yes
SVC 0
WRTCONS
Send message to user
Update VPSW from MCNPSW
WRTCONS
Send msg to operator (mach check problem mode USERID =)
Yes
User disconnected
No
Set message (mach. check-CP entered request, please)
Go to DISPATCH
BREAK
Put user in CP mode

Figure 38. Overview of Machine Check Interruption Handler

All machine checks, whether supervisor or problem state, are recorded by CP-67. The first two tracks of the CE cylinder are reserved for machine checks. The format of the machine check error record is as follows:

```
           ORG       LOGDATA          M/C ERROR RECORD
LOGMDATE   DS        CL6              DATE AND TIME
LOGMCODE   DS        CL2              MACHINE CHECK CODE
LOGMCPU    DS        22D              CPU LOGOUT DATA
LOGMPSW    DS        5D               OLD PSW's
LOGMGRS    DS        16F              GENERAL REGISTERS
LOGMCRS    DS        16F              CONTROL REGISTERS
LOGMFPRS   DS        4D               FP REGISTERS
```

Two machine check error records are contained within one physical record. Thus a maximum of 30 records may be contained within two tracks of a 2314 SYSRES. When the machine check log is full, the message "** CECYL FULL; M/C ERRORS NOT RECORDED **" is printed at the operator's terminal, and subsequent machine checks are not recorded until CLEARMC is run by the customer engineer. Pointers are kept to the next available slot in the log so that machine check errors are recorded sequentially. If an I/O error occurs when attempting to write a machine check error record, it is retried eight times. Upon continued failure, an error message "** IOERROR RECORDING FAILURE ON DEV___ **" is sent to the operator.

INTERRUPTION REFLECTION

When an SVC interruption or a program interruption occurs and the user's virtual machine is operating in problem mode, the interruption is reflected back to the user's supervisor (pseudo-supervisor) for handling.

The program interruption handler (PROGINT), upon determining that the interrupted user is operating in problem mode, saves the virtual registers and their old PSW (PROPSW).

The current PSW is moved into the old PSW, and the interruption code is set. If necessary, PAGTRANS is again called to obtain the address of the new PSW, and the new PSW is moved into the current PSW. When adjustment of PSW's is complete, control is returned to DISPATCH, which will eventually allow the user to resume processing.

Figure 39 illustrates the processing and reflection of interrupts.

	Real Machine State		
Interrupts	Real Supervisor State	Real Problem State	
	CP	Virtual Supervisor State OS or CMS	Virtual Problem State Problem Program
External	Masked off	Start another user; end of 50 ms time slice for this user. External Timer Virtual interrupts simulated	
SVC	For subroutine linkage	Reflect interrupt to virtual machine	
Program	ABEND	Reflect interrupt to virtual machine	
Privileged	Not possible	Simulate instruction Do I/O for SIO	Reflect
Machine check	ABEND	ABEND	ABEND
I/O	Masked off	Restart channel. Record the device status in virtual machine description if virtual I/O.	

Reflect on interrupt:
Current PSW ---→ Old PSW
New PSW---→ Current PSW
Set interrupt code; decrement timer;
timer interrupt if required.

FIGURE 39. Processing and Reflecting of Interrupts

The PAGTRANS routine is responsible for satisfying the paging demands placed on the system by user programs. It satisfies requests for page access via the TRANS macro from various parts of the Control Program, including the program interrupt handler (PROGINT) for paging faults, the input-output string handler (CCWTRANS) for user-initiated input-output operations, etc. PAGTRANS has the responsibility for freeing up main memory space when required, performing the input-output operations necessary to free the space, and protecting the system against "paging overload" conditions that may arise during periods of peak demand for the memory resource.

All calls to PAGTRANS are made through the use of the macro instruction TRANS. If LOCK is not specified in the TRANS macro and the virtual page is already resident in memory, there is no need to call PAGTRANS, and the call is bypassed by the macro generation.

Required Page in Core

When PAGTRANS translates the virtual address (via the LRA instruction) and finds that the page containing the address is currently core resident, a test must be made to see whether the LOCK option has been specified. (Normally, this will be the case, for the TRANS macro would not have generated the call to PAGTRANS for an in-core page if the LOCK option was omitted.) If lock is requested for the page, the lock count for that page is incremented, and the lock flag is set in the core table entry for that virtual page. When the lock flag is set, the page is not available for "swapping" (that is, it will be retained in storage until the lock count is reduced to zero and the lock flag is cleared). The lock count cannot be greater than 65,535.

When lock processing is completed (or if LOCK was not requested), a condition code of zero is set, the translated address is stored in the calling routine's save area, and control is returned to the calling routine. A condition code of zero indicates that the address translation was successful and that the specified virtual page is in core. (Note that the TRANS macro will automatically perform an LRA instruction after the return from PAGTRANS. In some instances, it would be possible for the paging routines to return a page as in core and have it chosen for swapping, and therefore nonresident, before the actual return to the caller. This is true only in DEFER cases.)

Required Page Not in Core

When PAGTRANS translates the virtual address and finds that the page is not core resident, the entry for that page in the user's SWPTABLE is found. The SWPTABLE entry contains the direct access storage address of the required virtual page. A test is made to determine whether the BRING option was specified when PAGTRANS was called. If BRING was not specified, a condition code of 1 is set, and control is returned to the calling routine. A condition code of 1 indicates that the required page is not in storage.

Required Page in Transit

If the required page is not in core and the BRING option is specified, the transit flags in the SWPTABLE entry are examined to determine whether the virtual page is in transit (that is, a previous request to read in the page or a request to write the page out has not yet been completed.) If the page is in transit, a Control Program execution request block (CPEXBLOK) is created and chained to the input-output task block (IOTASK) for the pending read or write operation, and PAGEWAIT is indicated in the VMSTATUS entry of the user's UTABLE. When the page I/O operation has completed, the CPEXBLOK is added to the CPRQUEST queue, and control is returned to DISPATCH. If the operation was a read, the PAGEWAIT condition is removed and the CPEXBLOK indicates a return to the initial caller of PAGTRANS. If the operation was a write, the CPEXBLOK indicates a re-enter to PAGTRANS to retest the transit flags.

Obtaining Core for a Paging Operation

If the required virtual page is neither in core nor in transit, and the BRING option has been specified, PAGTRANS must prepare to read the page into storage. An available page of core into which the required virtual page may be read must be found.

The table used for managing the real machine core allocation is called the CORTABLE. There is one 16-byte entry in CORTABLE for each 4096-byte page of real core. See the description of the CORTABLE control block for the bit usage.

Each entry of the CORTABLE is examined in a round-robin manner to determine whether the associated page is available for a paging operation. The search begins at the first entry after the last selected page.

The Lock MASK byte must be zero in order to have that page eligible for paging.

On the first pass each entry is examined, and if either of the following two conditions is satisfied, the corresponding page is selected:

1. An entry with bytes 5-7 equal to X'FFFFFF' (pages not in use by any user).
2. Neither of the keys for the page has the reference bit set on.

If the first pass fails to find an eligible page, then on the second pass any entry with a Lock MASK of zero is selected, since all such pages are equal candidates for selection. Both passes are initiated and terminated at the next entry after the last one used. All non-locked pages that are examined and not selected have their reference bits turned off.

If the selected page has a changed bit on, the page must be written to its DASD location (that is, swapped) before the new virtual page is read in. The DASD address is obtained from the corresponding swap table entry, an input-output task block is created, the page table entry for the page is marked "not-in-core", and the IOTASK block is queued for execution.

The address of the page selected for the paging operation is stored in the page table, and the not-in-core flag is set in the page table entry.

Reading a Required Page into Core

When an available page of real core has been found, the page address is stored in the page table entry and the not-in-core flag is set. The transit flag is set in the corresponding swap table entry, and the transit bit is set in the core table entry.

The DASD address of the required virtual page is obtained from the SWPTABLE, and an IOTASK block and a channel command word (CCW) list for reading the page in are created; the routine QUERIO is then called to queue the task to the input-output task list.

The "recompute" flag is used when a new swapping DASD address is to be used when the page is changed. At login time (and at a re-IPL for a virtual machine) the swap table entries are all set to the DASD address of a "zeros" page on the CP-67 system residence volume.

The recompute bit is set in each entry by LOGIN so that the page will be assigned an appropriate secondary storage location when it is referenced. This process, called dynamic page allocation, ensures that only those pages in a user's virtual machine which change and must be rewritten are assigned paging space on drum or disk. When a page is to be written out for the first time (that is, the recompute bit is set), a routine called PAGEGET is called. This routine finds an available location on drum or disk (in that sequence) and saves the address of that DASD location in the SWPTABLE entry for that page. This DASD address will be used on all subsequent reads or writes of that page for the duration of the user's session. If the user logs out or re-IPL's a system, a routine called PAGEREL is called. This routine returns all of the user's paging DASD locations to the available pool and resets each SWPTABLE to zeros. Only those user pages which have actually been written out to secondary storage (that is, for which the recompute bit is off) are reclaimed at PAGEREL time.

Returning Control

When all other PAGTRANS operations are completed, the used and changed flags are set in the SWPTABLE entry for the page being read. If the LOCK option was specified when PAGTRANS was called, the lock count is incremented, and the lock flag in the core table entry is set.

If the DEFER option was not specified when PAGTRANS was called, control is returned to the calling routine. If the DEFER option was specified, PAGEWAIT is indicated in the current user's UTABLE, a Control Program execution request block is created, and a pointer to the request block is placed in the IOTASK block which was created to read in the required page. Control is then returned to DISPATCH.

When the page has been read in, the PAGEWAIT bit is reset in the UTABLE, and the Control Program execution request block is added to the CPRQUEST queue. The next time DISPATCH is entered, the Control Program execution request block will be honored, and since the required page is now resident in storage, the completion of the paging operation will be indicated.

Shared Pages

When more than one user is using a given operating system such as CMS, which has reentrant pages, it becomes possible to share those pages among those users. In order to allow CP to share these pages, the operating system must be IPL'd by name (for example, IPL CMS).

When the first user of a shared system issues the IPL command, all the shared pages are brought into core and locked to prevent their being swapped out. When a subsequent user IPL's the same system, no paging is required, but the PAGTABLE of such a user is set to point to the shared pages.

For store protection of the shared pages, the users are run with protection key = F. All shared pages' storage keys are set to zero and all other pages belonging to these users have storage keys = F.

Note: The module SYSTEM has to be assembled to indicate which of the pages of a given system are shareable. If none are so indicated, no pages will be shared.

FREE STORAGE MANAGEMENT

Note: &TRACE(4) option must be chosen at sysgen time in order to gather statistics in FREE/FRET.

The FREE routine is responsible for the efficient management of free storage, as heavily used within CP-67 for I/O tasks, CCW strings, various I/O buffers, and the like. It is used, in fact, for practically all such applications except real channel, control-unit, and device-blocks, and the CORTABLE.

Block sizes of 29 double words or less, constituting about 99 % of all calls for free storage, are grouped into ten subpool sizes, and are handled by very fast LIFO (push down stack) logic.

Blocks of greater than 29 double words are strung off a chained list in the classic manner.

Subpool blocks are generally obtained, when none are available, from the first larger sized block at the low sized end of available free storage. Large blocks, on the other hand, are obtained from the high-numbered end of the last larger block. This procedure tends to keep the volatile small subpool blocks separated from the large blocks, some of which stay in core for much longer periods of time, thus undue fragmenting of available core is avoided.

The various cases of calls to FREE for obtaining free storage, or to FRET for returning it, for subpool sizes and large sizes, are handled as follows:

Call to FREE for a Subpool Size:

Subpool Available:

If a call for a subpool size is made and a block of the suitable size is available, the block found is detached from the chain, the chain patched to the next subpool block of the same size (if any), and the given block returned to the caller.

Subpool Not Available:

If there is no suitable block when a call to FREE is made for a subpool size, then the chained list of free storage is searched for a block of equal or larger size. The first block of larger or equal storage is used to satisfy the call (an equal-size block taking priority), except that blocks within pages previously obtained from EXTEND are avoided if at all possible. If no equal or larger block is found, all the subpool blocks currently not in use are returned to the main free storage chain, and then the free storage chain is again searched for a big enough block to satisfy the call. If there is still not a big enough block, then EXTEND is called to obtain another page of storage, and the process is repeated to obtain the needed block.

Call to FREE for a Large Block:

If a call to FREE is made for a block larger than 29 double words, then the chained list of free storage is searched for a block of equal or larger size. If an equal size block is found it is detached from the chain and given to the caller. If at least one larger block is found, the desired block size is split off the high numbered end of the last larger block found, and given to the caller. If no equal or larger block is found, EXTEND is called to obtain another page of storage, and the above process is repeated (as necessary) to obtain the needed block.

Call to FRET for a Subpool Size:

If a subpool size block is given back via a call to FRET, the block is attached to the appropriate subpool chain on a LIFO (push down stack) basis, and return is made to the caller. If, however, the block was in a page previously obtained from EXTEND, the block is returned to the regular free storage chain instead.

Call to FRET for a Large Block:

If a block larger than 29 double words is returned via FRET, it is merged appropriately into the regular free storage chain. Then, unless exactly one page was given back (i.e. by EXTEND), a check is made to see if the area given back (after all merging has been done) is a page previously obtained from EXTEND. If so, it is returned via PAGFRET for use by the remaining programs in CP for their use.

The FREE/FRET logic as described above allows the number of pages allotted for main storage to "breathe" as necessary, expanding via calls to EXTEND when extra pages are needed, and contracting via PAGFRET when such pages have all been FRET'd and are no longer needed.

Initialization

The number of pages allocated to free storage depends upon the number of core boxes upon which CP is running, and is initialized by CPINIT. A special entry FRETR in the FREE/FRET routine is used by CPINIT and EXTEND to return blocks to the regular free storage chain regardless of their size.

EXECUTION CONTROL

When all interruption handling routines complete their processing, they transfer control (via a GOTO macro) to the main dispatcher and control routine (DISPATCH). DISPATCH charges time used within the Control Program to the appropriate user and determines which user is to receive control next.

Each time DISPATCH is entered, the time used by the current (interrupted) user within the Control Program is computed and added to the TIMEUSED entry in the user's UTABLE. If the current user has not exhausted his allotted time for this quantum, he will be restarted. In this case, his pending interrupts are reflected, and then if runnable he is restarted. If no time remains for the interrupted user, any CPRQUEST's are honored. Then another user is chosen for running.

The following checks are made by DISPATCH upon each entry to it and prior to the running of a new user:

The queue of Control Program execution requests (CPRQUEST) is examined for any pending work. If any requests are found, the appropriate execution request block (CPEXBLOK) is used to load the registers and dispatch control to a specified section of the Control Program. This section will return control to DISPATCH via a GOTO macro.

If the current user is not runnable and the CPRQUEST stack is empty, a new user is selected to run.

In order to prevent paging overload, the system allows only a subset of the users to run at any given time. Interactive users are in Q1, and the users who put a heavy load on the system in terms of CPU cycles required or amount of nonterminal I/O done are in Q2. There is a maximum limit on Q1. A table in the module EXTEND is used to set the maximum for Q1, depending upon the real core size. The limit of users in Q2 is dynamic and is dependent on current paging activity.

A user will be in one of the following five modes at any given time:

In Q1
Waiting to get into Q1
In Q2
Waiting to get into Q2
Dormant, not requiring system resources

Moreover, a user may or may not be runnable, regardless of whether he is in the queues. A user is not runnable if he is waiting for:

A page to be brought in
An I/O operation to be started
A CP console function
A VM interrupt (VM in wait state)

The next user to be run is selected according to the following priorities:

1. Current user if runnable
2. First member of Q1 encountered
3. Oldest runnable candidate for Q1 if Q1 is not full
4. Oldest runnable candidate for Q2 if Q2 is not full
5. Oldest member of Q2 not CPU-limited
6. Oldest member of Q2 if CPU-limited

To start (or restart) a user, DISPATCH loads the appropriate control registers from the contents of the chosen user's UTABLE entries, loads the interval timer with the user's quantum (or the unused portion of it), and gives control to the user by entering the problem mode.

Queue Management

Definitions:

Dispatching of users is described in terms of their movement from one state to another. The four states are described as follows, as well as the definition of an interactive and non-interactive user.

State 1
- runnable in Q:
- virtual machine not in wait state
  not in page wait
  not in I/O wait

State 2
- not runnable in Q:
- virtual machine in wait state, but enabled for an I/O interrupt on a busy channel
  in page wait
  in I/O wait

State 3
- runnable not in Q:
- virtual machine not in wait state
  not in page wait
  not in I/O wait
  but CPU time exceeded (.4 or 5 seconds) and number of interactive users at maximum, or paging activity index when added with in Q users will exceed system paging index.

State 4
- not runnable not in Q:
- virtual machine in wait state and disabled or enabled with no busy channels; stopped, CP console function mode - 'ATTN' on terminal.

Interactive user:
- interrupt from terminal
- use less than .4 seconds of CPU time
- have a priority between 0 and 15

Non-Interactive user:
- no terminal activity
- use more than .4 seconds of CPU time
- have a priority between 16 and 215

A user goes from runnable (in Q) to eligible (runnable, not in Q) when his CPU time (.4 for Q1, or 5 seconds for Q2) is exceeded; in order to be runnable (in Q), he must not be in I/O, console function or page wait, or virtual machine wait state. When any of these conditions pertains, he will be dropped to a non-runnable (whether in Q or not in Q) status. A user is advanced from eligible (runnable, not in Q) to runnable (in Q) on the basis of interrupt status or virtual machine priority.

Movement from state to state is illustrated in detail in the following chart:

From State	To State	Causing Condition
1	2	Pagewait; IOwait; VMwait-IOactive
1	3	CPU time exceeded (.4 or 5 seconds)
1	4	VMwait-no IO active; VM stopped-CFwait
3	1	Scheduled by Interrupts or priority
3	2	Not possible
3	4	VM stopped-CPwait
2	1	PageIO; IOstarted; IOinterrupt
2	3	Not possible
2	4	VM stopped-CFwait
4	1	Not possible
4	2	Not possible
4	3	AsynIOint; VM not stopped-begin

Note: 3 = 'eligible'

Users within states 1, 2, and 3 are ordered by priority. Priority is determined by a combination of paging activity index, user directory priority, and system priority number.

Number of in Q users:

- interactive users limited by a specific maximum based upon real machine size;
  i.e. 512K machine = 6
  768K machine = 9
  1024K machine = 12

- non-interactive users limited by paging activity index so that a system paging index is not exceeded. System paging index is a function of real machine size;
  i.e. 512K machine = 40
  768K machine = 70

Figure 40 is a state diagram illustrating the flow of users from one state to another.

Virtual timers are maintained in one of two ways. The default method is to increment a user's virtual timer (virtual location hex 50) by only the amount of virtual CPU time the user uses. A 'real timer' option is also available which will also update a user's virtual timer by the amount of time the user spends in virtual wait.

Note: this does not supply the user with a timer that runs the same as 'wall clock'. For instance, the virtual timer is not incremented by the amount of time a user spends waiting for his chance to run while appearing runnable to the system.

'Real Timers'

The real timer option attempts to provide a clock for systems maintaining time-slice environments. For this purpose the 'timer' (virtual location 50) is updated by the time spent in virtual execution (all virtual timers are updated by this value) and the time the virtual machine spends in virtual wait (or voluntary wait). The clock is not updated for elapsed time while the virtual machine is in IOWAIT or PAGEWAIT or while the virtual machine is runnable but cannot run because CP-67 has given control to some other user with a higher priority. This enables time-slicing systems to give reasonably constant time-slices independent of the activity going on in the overall CP-67 system.

The critical facility that 'real timers' supply that ordinary timers in CP-67 do not is the ability to have the timer cause an external interrupt while the virtual machine is in virtual wait. To provide this facility, CP-67 maintains on 'elapsed binary timer'. When a virtual machine enters waitstate, it is 'time stamped' with a value equal to the sum of its virtual timer plus the value of the binary timer, if the virtual timer value is positive. This represents the time when the virtual machine will expect an external interrupt. The lowest time stamp value is always kept by the real timer routine and every time the binary timer is updated it is compared against the lowest time stamp value. If the binary timer value exceeds the lowest time stamp value, the real timer routine is entered to update the virtual machines with real timers; otherwise, normal processing continues.

Figure 41 is an overview of the Dispatcher Scheduling Algorithm.

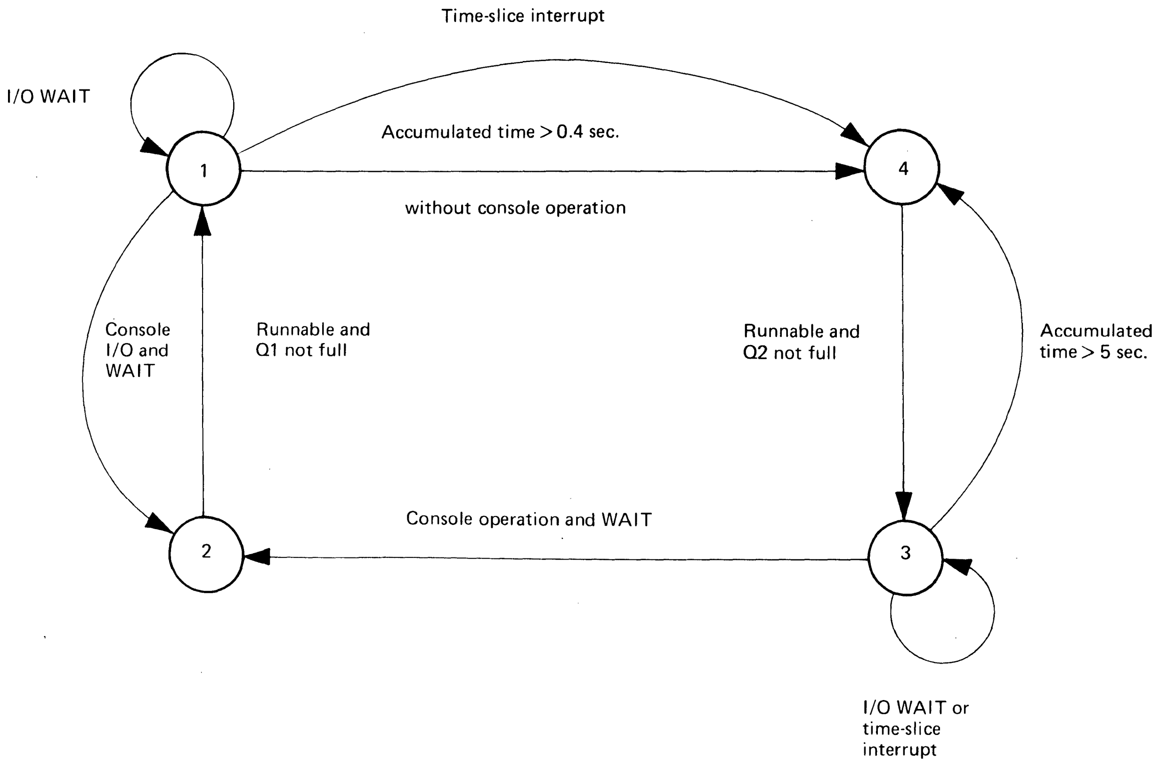

Time-slice interrupt
I/O WAIT
Accumulated time > 0.4 sec.
1
4
without console operation
Console I/O and WAIT
Runnable and Q1 not full
Runnable and Q2 not full
Accumulated time > 5 sec.
Console operation and WAIT
2
3
I/O WAIT or time-slice interrupt

FIGURE 40. State Representation of Scheduling Algorithm

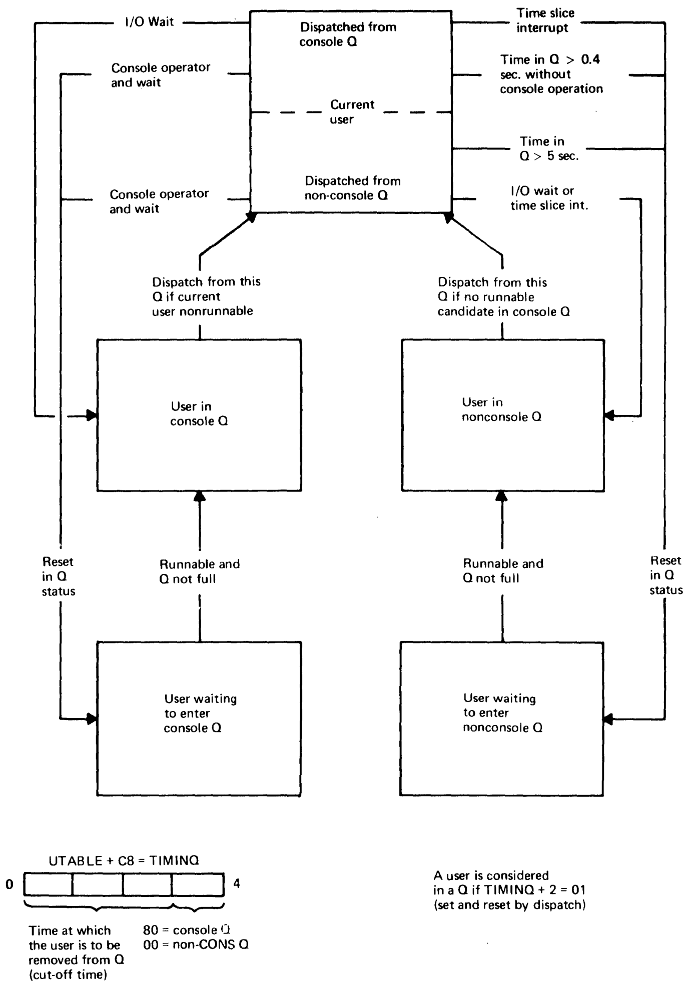

I/O Wait
Dispatched from console Q
Time slice interrupt
Console operator and wait
Time in Q > 0.4 sec. without console operation
Current user
Time in Q > 5 sec.
Console operator and wait
Dispatched from non-console Q
I/O wait or time slice int.
Dispatch from this Q if current user nonrunnable
Dispatch from this Q if no runnable candidate in console Q
User in console Q
User in nonconsole Q
Reset in Q status
Runnable and Q not full
Runnable and Q not full
Reset in Q status
User waiting to enter console Q
User waiting to enter nonconsole Q
UTABLE + C8 = TIMINQ
0
4
Time at which the user is to be removed from Q (cut-off time)
80 = console Q
00 = non-CONS Q
A user is considered in a Q if TIMINQ + 2 = 01 (set and reset by dispatch)

FIGURE 41. Overview of the Dispatcher Scheduling Algorithm

Handling of a Virtual 67

Six areas are discussed in this section:

1. Control blocks
2. Different format of the PSW
3. Special processing of the reset function
4. New instructions
5. Handling of the virtual dynamic address translation
6. Restrictions

Control Blocks

EXTUTAB is created at LOGIN time.

Each time a virtual 67 enters extended PSW, by loading control register 6 with bit 8 set to 1 (by means of the LMC instruction or STORE X6 console function), space is reserved for the shadow segment table and one shadow page table belonging to segment 0.

If the virtual 67 uses segments 1 to 15, a "copy segment table", an "image segment table" and the necessary number of additional shadow page tables will be allocated.

All those tables, if any, except EXTUTAB, will be returned to free storage each time the virtual 67 leaves extended PSW mode by loading control register 6 with bit 8 set to 0, or by the reset function.

Different Format of the PSW

The format of the PSW in a 360/67 running in extended mode (that is, bit 8 of control register 6 set to 1) differs from that of a standard System/360. Contents of certain reserved lower core locations are different after an interrupt has occurred. (See IBM System/360 Model 67 Functional Characteristics, A27-2719). The following modules have been modified to take into account that difference:

CFSMAIN
DISPATCH
IOINT
MVIOEXEC
PROGINT
PSA
QUEVIO
UNSTIO
VIOEXEC

Reset Function

When a reset function is executed for a virtual 360/67, control register 6 is reset to C00000FF, and all the control blocks specified for a 67, except EXTUTAB, are returned to free storage. The module affected is RESINT.

New Instructions

Among the five new instructions, two are nonprivileged and are executed normally (BAS,BASR), and three are privileged and thus simulated (LRA,LMC,STMC).

LRA modifies the condition code and the contents of the first operand register, according to the contents of the segment and page tables, which are located in the virtual machine core and pointed to by (virtual) control register 0.

For LMC and STMC, only control registers 0,2,4, and 6 are retained in EXTUTAB; the others always contain zeros and cannot be modified by LMC.

When loading control register 0, a possible data exception is reflected.

When loading control register 6, bit 8 is examined and the mode (normal or extended) is set according to its contents.

The module affected is PRIVLGED.

Handling Virtual Dynamic Address Translation

In this description the following terminology is used:

First level memory. The memory of the real 360/67.

Second level memory. The memory of a virtual 360/67.

Third level memory. The memory of a virtual machine running under the virtual 360/67.

Shadow segment and page tables. Segment and page tables used by the real machine. When CP gives control to a virtual 67 running in extended mode with translation on, these tables (in first level memory) will describe the third level memory and will be used to control the real address translation hardware.

Copy segment table. A copy, in first level memory, of the segment table, in second level memory, used by the virtual 67 when running in extended mode with translation on.

Image segment table. A copy, in first level memory, of the shadow segment table, with 00 in the first byte of each entry, and bit 31 set to 1 (unavailable bit) in each entry.

Monosegment machine. A virtual 67 in which segments 1 through 15 are not used.

Multisegment machine. A virtual 67 which has already used at least one segment other than segment 0.

For example, a virtual 360/67 running CP-67 and generating any number of virtual machines will be a monosegment machine so long as all these virtual machines use a core size less than or equal to one megabyte. That machine will become (dynamically) a multisegment machine as soon as it runs a virtual machine using more than one megabyte. Monosegment virtual machines are handled with much less overhead than multisegment virtual machines.

Each time CP-67 gives control (by means of DISPATCH) to a virtual 67 running in extended mode and with the translation control bit on, it checks the validity of the shadow tables: if those tables have been invalidated by a previous loading of control register 0 or by a previous paging interrupt, the following steps are taken:

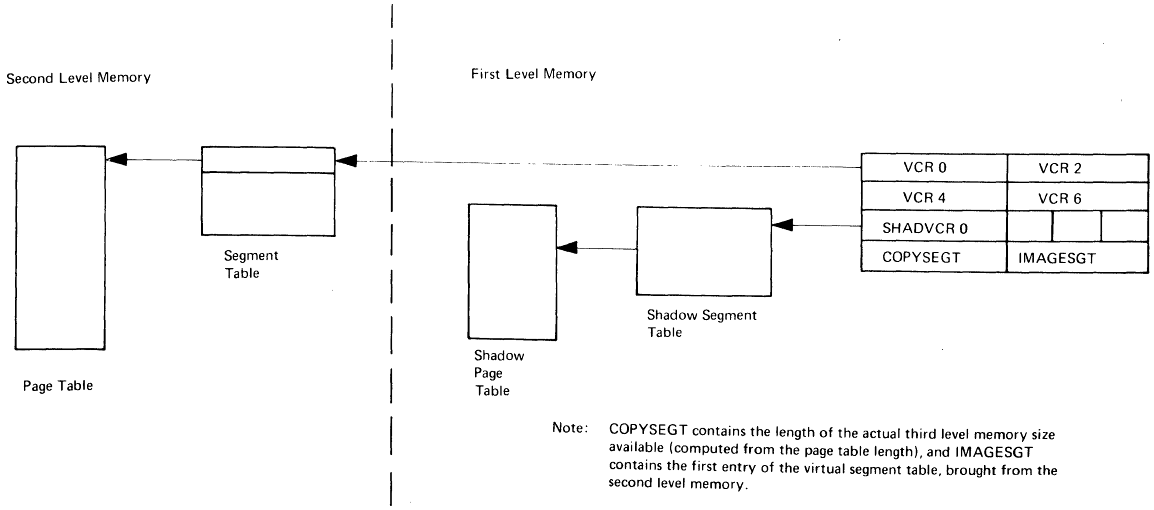
1. For a multisegment machine, a copy of the actual segment table is brought from second level memory into the copy segtable.

For a monosegment machine, the first entry of the actual segment table is brought from second level memory into IMAGESGT, and the size of the actual third level memory is updated into COPYSEGT.

2. For a multisegment machine, the image segment table is copied into the shadow segment table in order to reset it quickly with all the entries flagged with the unavailable bit on.

For a monosegment machine the single shadow page table is reset with the first n entries flagged with the unavailable bit on, n being the page table length.

If the shadow tables have been invalidated because a page of the virtual 67 has been removed from first level memory, only step 2 is taken. (See Figures 42 and 43)



Note: COPYSEGT contains the length of the actual third level memory size available (computed from the page table length), and IMAGESGT contains the first entry of the virtual segment table, brought from the second level memory.

FIGURE 42. Virtual 67--Monosegment Machine

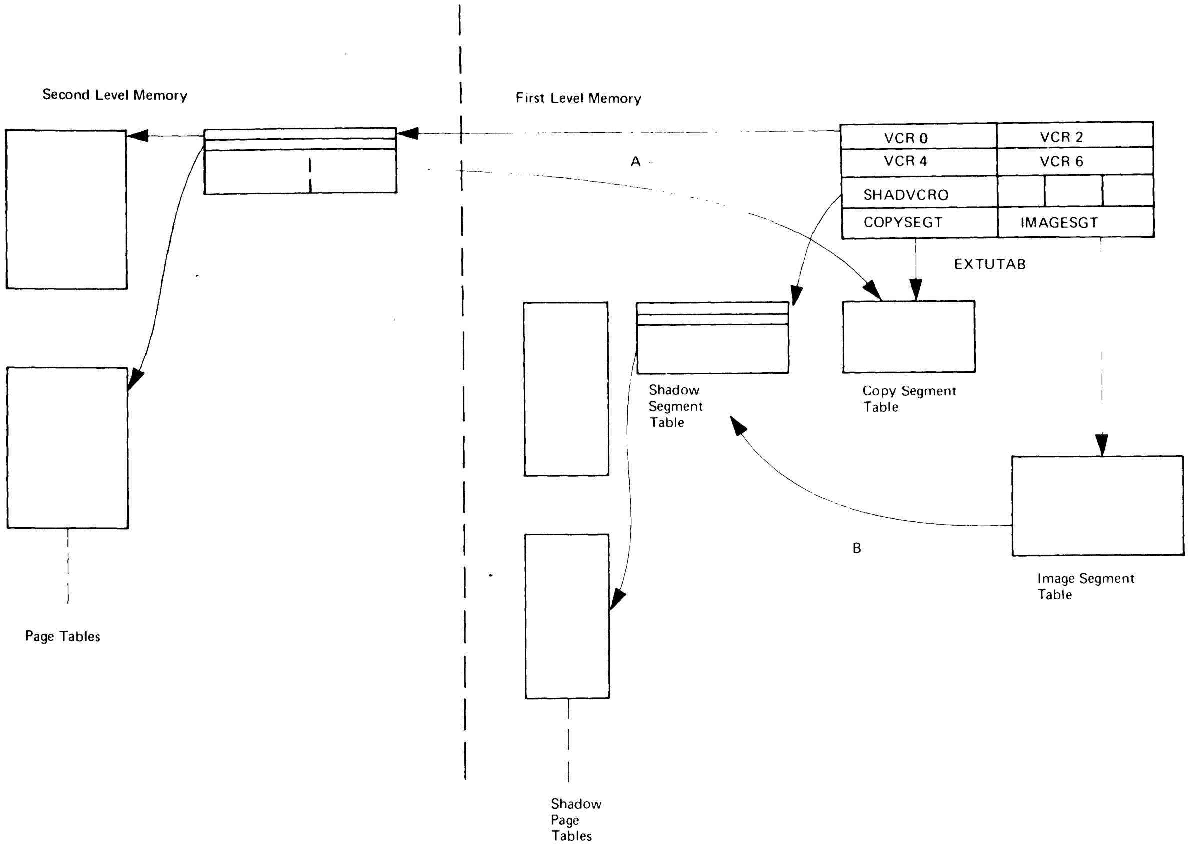


FIGURE 43. Virtual 67--Multisegment Machine

When a paging interrupt takes place, if the virtual machine interrupted is a 360/67 using the virtual dynamic address translation, the processing is the following:

If the interrupt is a page exception (interrupt code 11) a check is made to see whether the interrupt should be reflected; if it should not, a request is issued for the missing page, if necessary; otherwise (if the page is already in first level storage), the proper entry in the right shadow page table is loaded, and the virtual machine restarted.

If the interrupt is a segment exception (interrupt code 10) a check is made to see whether the interrupt should be reflected. If it should not, and if a shadow page table has already been allocated to the segment originating the interrupt, the unavailable bit is removed from that entry of the shadow segment table, the page table length is loaded, the corresponding shadow page table, according to that length, is reset with the unavailable bit in each entry, and the processing continues as for a paging interrupt for a multisegment machine.

If a shadow page table has not yet been allocated, one such table is allocated and, furthermore, if the virtual 67 is switching from monosegment to multisegment machine, the copy and image segment tables are allocated and initialized; then control is given to the dispatcher.

The modules modified to handle this algorithm are mainly DISPATCH and PROGINT and also CFSDBG and PAGTRANS.

Virtual 67 Restriction

A virtual machine may be a 360/67 provided it has a simplex CPU, with 24-bit addressing.

CONSOLE FUNCTIONS

When a console interruption occurs because the attention key has been activated at a user's terminal, the I/O interruption handler (IOINT) calls the CONSINT routine. CONSINT then calls BREAK in CFSMAIN if the terminal has a logged-on user.

BREAK determines whether the user was executing or was waiting for completion of a console function when the "attention" occurred. If the user was waiting for a console function, the "attention" is reflected to the user's machine as an online console attention button interrupt. If the user was executing, the routine RDCONS is called to read the console function request, and control is returned to the interrupted routine. If the user was receiving output from a console function request when the attention button was depressed, that output function is terminated, and the keyboard is unlocked waiting for another console function request.

When the console function request has been read, the console function processor CFSMAIN is entered to analyze the request. CFSMAIN determines the type of function requested and gives control to the appropriate subroutine. When all console functions have been processed, control is returned to the calling routine.

The console functions can also be executed from the virtual machine level by the diagnose instruction (code 8) and the required buffers. (See "Program Interruptions" earlier in Section 2.)

The following console function descriptions cover the four privilege classes of users:

A - operator
B - administrator
C - customer engineer
D - a normal user

Also included is the system operator class, which belongs to the first user to log in with privilege class A. Normally he is the operator of the Model 67.

The following console functions are described:

ACNT - punch and reset accounting information for active users
ATTACH - attach a device to a user or to the system
BEGIN - initiate execution of a virtual machine
CLOSE - give logical EOF on unit record equipment
DCP - display contents of real memory and registers
DMCP - dump contents of real memory and registers
DETACH - remove a unit from a virtual machine or from the system
DISABLE - inhibit 2702 line access to the system
DIRECT - allow and inhibit system DIRECTORY access

DISCONN - disconnect a terminal from a running virtual machine
DISPLAY - display contents of memory and registers
DRAIN - quiesce a unit record input or output
DUMP - dump contents of memory and registers
D_U_M_P - cause a system ABEND dump
ENABLE - enable 2702 lines for access to the system
EXTERNAL - give virtual external trap
IPL - perform an initial-program-load sequence; reset virtual memory to binary zeros
IPLSAVE - perform an IPL without resetting virtual memory to ZERO
KILL - log a user off the system
LINK - attach a DASD device using a directory unit description
LOCK - lock selected user pages in core
LOGIN - log into the system
LOGOUT - log out of the system
MSG - send a message to the user(s) or operator
PURGE - delete a user's spooled input or output files
QUERY - query the status of the system
READY - ready a virtual device
REPEAT - repeat the output of a currently active file on the real unit record devices
RESET - reset the interrupt status of a virtual machine
SET - establish system parameters or machine status
SHUTDOWN - bring the system to orderly shutdown
SLEEP - place a terminal in dormant state to receive messages
SPACE - force printed output for a file to single space
SPOOL - direct and control spool input and output
START - commence unit record output after a drain or when requested
STCP - store into real memory locations
STORE - store into memory or registers
TERM - terminate current unit record operation
UNLOCK - release previously LOCKed pages
WNG - issue a warning message to user(s)
XFER - transfer spooled punch output to a user's spooled reader input

Console Function Subroutines

The following brief descriptions cover some of the important subroutines in console function processing.

CONSTART - this routine is entered after the console function has been read by RDCONS. It analyzes the data and goes to COMANL to scan the command list for the desired function.

SCANFLD - this routine will return to the caller (via BAL) the starting location and the length of the next field in the command input, or an indication that no more data exists.

BEGIN - this routine releases the read buffer and large save area, resets the user's CFWAIT status, and exits.

BREAK - this routine is the entry point called when the user actuates the attention key. It will get a 17-doubleword buffer used by RDCONS to read the console function and a 17-doubleword large save area, which is used on subsequent call by CONSOL to other routines and as general working storage for various functions.

SIMATTN - this routine is entered if the user actuates the attention key while in console function mode, thus giving an "attention" to his virtual machine.

FINDUSER - this routine will search the chain of UTABLES for a specified "userid". A message is given if the user is not found, or his UTABLE address is returned in register 10.

The module CFSMAIN contains all these subroutines. CFSMAIN remains addressable through register 12 for all command processing. Individual commands are placed together in several other modules, each module addressable by register 9.

Console Function Descriptions

The following conventions are used throughout these descriptions: (1) variable information is indicated in lowercase letters, and system keywords are indicated in uppercase letters, whereas either case may be used when communicating with the system; (2) "<" and ">" are used to bracket choices when applicable in the description (for example, "MSG <userid,ALL>" would be used to indicate that "MSG userid" or "MSG ALL" could be used), whereas the brackets are not typed when communicating with the system.

ACNT (ACNT) - class A and B

```
ACNT
```

The following steps are taken:

- for each UTABLE in the system call ACNTIME to give accounting to each user

- call ACNTOFF for each user to punch an accounting card and reset the accounting data

Note: ACNT does not punch an accounting card or reset the accounting data for dedicated devices.

ATTACH (A) - class A and B

```
ATTACH ccu TO userid AS xxx
ATTACH ccu TO SYSTEM AS volid
ATTACH  RDR | PRT | PUN  TO userid AS xxx
```

The following steps are taken when attaching a device to a user or to the system.

- scan the selector device chain for the device "ccu"
- check that the device is not "owned" or already attached
- issue a sense command for DASD types to determine that the device is "ready"
- check that the "userid" is currently logged in to the system
- check that the "userid" does not already have a device of address "xxx"
- create the virtual device blocks for the user and link them to any existing blocks
- call DEDICATE if the device being attached is in the real multiplexer chain. DEDICATE will create and chain a set of real selector device blocks.
- link the virtual and real device blocks on an attached (nonshared) basis
- send a message to the "userid" that the device has been attached
- if the device is being attached to the system, CP will read and verify the "volid" and check that the volume is not already mounted
- ATTACH will check the "owned list" (in the CPDSK1 allocation table) to see whether the attached volume has a CP allocation table
- if the attached volume is "owned", the allocation table is linked to the real device block and to the allocation table chain
- if a "spooling" device (RDR PRT PUN) is being attached to a user, a virtual multiplexer block is created and chained to the user's virtual device chain
- various diagnostics are issued for a variety of error conditions that can occur

BEGIN (B) - any user (class A,B,C,D)

```
     BEGIN  <hexadd>
```

This command transfers control from CP console function mode to running the virtual machine.

The following steps are taken:

- set the user's virtual PSW to the address specified, if any

- free the console functions read buffer

- free the console functions large save area

- take the virtual machine out of "console function" wait

- exit to run the user

CLOSE (C) - any user (class A,B,C,D)

```
     CLOSE  ccu
```

The CLOSE command completes a user's spooled operation for the current file and schedules it for output, or clears the buffers for input.

The following steps are taken:

- locate the specified virtual device in the user's multiplexer chain
- call MVICLCR or MVICLPR or MVICLPN to close a reader or printer or punch, respectively
- output files will be scheduled for printing or punching or the punch file may be chained to a reader input if it was XFERed
- readers are cleared to accept the next spool file input. Remaining input is flushed.

DCP (DCP) - class A and B

```
DCP  arg1 arg2...argN
```

where the arguments (arg1...argN) are real memory location(s). The output goes to the terminal.

The following steps are taken:

1. The steps are the same as those for DISPLAY, except that the data is taken from real memory instead of virtual memory.

DMCP (DMCP) - class A and B

```
    DMCP  arg1 arg2...argN
```

where the arguments are the same as those for the DCP Console Function. The output goes to the first virtual printer defined in the user's virtual machine.

The following steps are taken:

1. The flag in the output buffer is set to indicate that the output is to go to the printer.

2. The remainder of the steps are the same as those for DISPLAY, except that the data is taken from real memory instead of virtual memory.

DETACH (DET) - any user (class A,B,C,D), except for certain functions

```
DETACH   ccu
```

The DETACH command allows any user to delete any virtual device from his current configuration.

The following steps are taken:

- the virtual device block(s) are located in the user's chain of devices
- a check is made to ensure that no tasks are queued for this device
- the virtual device blocks (for either selector or multiplexer devices) are removed from the chain and returned to free storage
- call RELEASE; if it is a nonshared device, RELEASE will make the real device available for use by other users. If the real device blocks were created by DEDICATE, the blocks are released to free storage, and the real multiplexer device is marked available (undedicated).
- a message is sent to the user indicating that the device is detached
- a message is sent to the operator if the DETACH has freed a previously dedicated device
- an operator (class A) can detach a device from a user by entering DETACH Rccu, where ccu is the real device address. The device must not be in use to do this.

DISABLE (DISA) - system operator only

```
    DISABLE  ccu ccu ccu
             ALL
```

This command allows the operator to selectively or generally inhibit access to the system from communication lines.

The following steps are taken:

- scan the MRDEBLOK chain for the selected (or ALL) terminal lines
- set the DISABLE bit in the MRDESTAT field of the block
- if the line is in use, return
- if the line is not in use, issue an SIO and HIO of a sense to kill any enables and force an interrupt
- CONSINT will handle the interrupt, detect the DISABLE bit and "disable" the line

DIRECT (DIR) - class A and B

```
    DIRECT  <lock,unlock>
```

This command inhibits or allows access to the system DIRECTORY. The following steps are taken:

- locate the directory lock byte and open file count
- if the directory is in use, exit with a diagnostic
- if the directory is not in use, set or reset the lock byte
- issue diagnostics if the byte was not already locked (lock) or unlocked (unlock)

DISCONNECT (DISC) - any user (class A,B,C,D)

```
    DISCONN  <xxx>
```

This command is used to release the user's terminal from his virtual machine but allow the virtual machine to continue running. The terminal is then free to log in as another virtual machine or to reconnect at a later time.

The following steps are taken:

- write a "disconnect" message to the user's terminal; if the optional field is present, do not disable the phone connection
- set the DISCNBIT in the user's UTABLE (TIMERMOD field)
- release the "console functions" read buffer and large save area
- write a "disconnect" message to the operator
- exit to run the virtual machine

DISPLAY (D) - any user (Class A,B,C,D)

DISPLAY (D) arg1 arg2 arg3 arg4 ... argN

where the arguments (arg1...argN) specify virtual memory location(s), general-purpose register(s), floating-point register(s), control register(s), storage key(s), and/or PSW. The output goes to the user's terminal.

The following steps are taken:

1. An 18 double word output buffer acquired from Free Storage.

2. The maximum number of characters to be displayed is set based upon the user's terminal type--16 bytes for teletype, 32 bytes for all other terminals.

   A BAL to subroutine DISWRITE to output any partially full buffer and reinitialize the buffer.

3. The location and length of the next argument is obtained by doing a BAL to SCANFLD. If there are no more arguments the output buffer is returned to Free Storage and return is made to READI in CFSMAIN.

4. The first character of the argument is inspected for a type code (P, G, Y, L, T, K, or X). If none is found an L is inserted in front of the argument. The code is used to select the routine to branch to branch to, to perform the unique processing for each type of display.

5. Each routine sets the default ending address and address increment and does a BAL on register 7 to subroutine DISINIT to determine the beginning and ending addresses of the data to be displayed.

6. Each routine loads the next four bytes of data to be displayed into register 3 and branches to DISCOMM.

   DISCOMM does a BAL to subroutine DISHEAD to build the header for the line if the buffer is empty.

7. The data is then converted to hexadecimal and stored in the next location in the output buffer.

8. If the buffer is full, a BAL is done to subroutine DISWRITE to output the buffer.

9. The next address to be displayed is computed by adding the increment address to the current address. If this address is greater than the ending address, the next argument is fetched (step 3). If this address is not greater than the ending address, the next four bytes are displayed by returning on register 7 (to step 6).

Subroutines:

DISINIT scans the argument for a hyphen or a blank. A hyphen indicates that a range of addresses is to be displayed. If the ending address is larger than the default ending address, the default address is used to end the display. If the beginning address is larger than the ending address, a "BAD ARGUMENT XX" message is sent to the user and the display terminated. If either the beginning address or ending address is omitted the default for that address is used in the display.

DISHEAD builds the header and trailer sections of each output line. For a register display, the register character identification is moved to the first three bytes of the output buffer and the register number is stored in bytes 5 and 6. For a display of core storage the next line to be displayed is compared to the last line. If both lines are the same, the "SUPPRESSED LINES" message is built in the buffer. If the lines are not the same, the last line is outputed and the current line is saved. For a display of core, the line is also translated to EBCDIC and moved to the trailer portion of the buffer. Before returning to the calling routine, the buffer pointer is set to the 8th byte of the buffer and the buffer count set to 7.

DISWRITE outputs the buffer either to the user's terminal or virtual printer based upon a flag in the buffer. After the I/O completes, the pointer is set back to the start of the buffer, the byte count is set to zero, and the buffer is cleared with blanks.

DRAIN (DR) - system operator only

```
    DRAIN <xxx . . . nnn>
```

This command will cause the specified unit record devices to stop processing at the completion of the currently active spool file.

The following steps are taken:

- find the specified real multiplexer device block, or locate each device in the chain if doing all devices

- set the MRIDRAIN bit in the MRIFLAG field of the MRDEBLOK

- if the device is not busy, print a message to the operator indicating the device is drained

- loop for all devices (readers, punches, printers) if draining all

DUMP (DU) - any user (class A,B,C,D)

```
DUMP arg1 arg2 ... argN
```

where the arguments are the same as those for the DISPLAY Console Function. The output goes to the first virtual printer defined in the user's virtual machine.

The following steps are taken:

1. The flag is set in the output buffer to indicate that the output is to go to the printer.

2. The remainder of the steps are the same as those for DISPLAY.

D_U_M_P (D_U_M_P) - system operator only

```
    D_U_M_P
```

This command will issue SVC 0 to cause a system ABEND dump.

The following steps are taken:

- verify complete command typed (no abbreviation)
- issue SVC 0

ENABLE (EN) - system operator only

```
    ENABLE  ccu ccu ccu . . . .
            ALL
```

This command allows the operator to selectively or generally enable 2702 lines for communication with CP-67.

The following steps are taken:

- scan the MRDEBLOK (multiplexer real device block) chain for the selected, or for every, 2702 line type

- determine whether the device is already enabled or otherwise in use; bypass if it is

- reset the DISABLE bit in the MRDESTAT field

- issue an SIO and HIO of a sense to force an interrupt

- CONSINT will receive the interrupt, issue the required (if any) SAD and ENABLE commands, and set the ENABLED bit in MRDESTAT

- set the return address (MIRA) to IDENTIFY for the termination of the ENABLE

EXTERNAL (EX) - any user (class A,B,C,D)

```
    EXTERNAL
```

This command simulates the operation of the CPU interrupt button to the virtual machine.

The following steps are taken:

- set a pending external interrupt status in the user's UTABLE (PENDING flags)
- exit to BEGIN2 to run the user's virtual machine

IPL (I) - any user (class A,B,C,D)

```
    IPL  xxx
```

This command will cause the loading and execution of a Control Program of the user's choice, where xxx is a virtual device containing an IPLable Control Program or is a presaved system name of a potentially shareable Control Program or "operating system".

The following steps are taken:

- call RESINT to reset the virtual machine status; that is, no interrupts pending

- call PAGOUT to clear the user's page table and the necessary system CORETABLE entries

- set the swap table entry for the user's virtual page number hex 20 (or the page at half virtual memory size, whichever is the smaller) to the location on the SYSRES volume of the IPL simulator page

- find the user's virtual device block if not IPLing by system name

- set the virtual address of the IPL simulator in the user's VPSW and exit to run the virtual machine (IPL simulator)

- if IPLing by system name, bring in the SYSTEM module which contains the table of system names and is actually the module, SYSTEM

- search for the desired system name

- move into the user's swap table entries the DASD locations of the saved system

- set a pointer to the shared page table, if any

- set the user's VPSW to the saved system execution address, and exit to run the virtual machine

IPLSAVE (IPLS) - any user (class A,B,C,D)

```
    IPLS   xxx
```

This command will initiate execution of the CP-67 IPL simulator in the user's virtual memory space for a device specification in xxx. xxx may be the name of a presaved system.

The following steps are taken:

- call RESINT to reset the virtual machine status
- bypass the call to PAGOUT so user's pages remain nonzeroed
- proceed as in IPL (after call to PAGOUT)

KILL (K) - system operator only

```
    KILL  userid
```

This command is used to force the logout of a particular user.

The following steps are taken:

- locate the desired user by linking to FINDUSER
- call the module ADSET (in USEROFF) to force the logout of the user
- the user receives a message indicating a forced logout by the operator

LINK (LI) - any user (class A,B,C,D)

```
    LINK  userid  xxx  yyy  <W R> <(NOPASS) | PASS=  password>
```

The link command will attach to the user a virtual DASD block of the specified address (yyy) from information contained in the system directory for user "userid" and his device, xxx. The user may request read or write status and may be prompted for a password. The user may also link to himself without a password. If LINKing to himself, the user may specify * for "userid". LINK can also be used as a virtual console function with a special (PASS= password) form to provide the password with the command.

The following steps are taken:

- retrieve all the parameters from the input command
- issue a "protected" read for the password if not linking to himself
- set up the parameters for, and then call LINK module
- on return, index on an error code to give a message
- the LINK module will grant the desired access and set up the necessary device blocks

The access modes permitted by the LINK command are summarized in 2. Note that when linking to one's own userid, access allowed is the same as at LOGIN. WRMULT is examined only when linking to oneself. The table assumes that the password supplied is correct and the device is shareable for the requested access mode.

Table 2. Summary of Access Allowed by LINK

Directory Specification		Access Requested	Existing Links	Access Mode Established	
				Link to Oneself	Link to Another Userid
RDONLY	WRMULT				
No	No	Read	None	Read	Read
			Read	Read	Read
			Write	None	None
No	No	Write	None	Write	Write
			Read	Read	Read
			Write	None	None
Yes	No	Read or Write	None	Read	Read
			Read	Read	Read
			Write	None	None
No	Yes	Read	Any	Read	
No	Yes	Write	Any	Write	
Yes	Yes	Read or write	Any	Read	

LOCK (LOC) - system operator only

```
    LOCK  userid  xxx  nnn
```

This command is used to lock specified pages of a user's virtual machine in core so that they will not be paged.

The following steps are taken:

- locate the desired user, who must be logged in to CP-67
- starting with the specified page (xxx), and looping for a number of contiguous pages (up to and including page nnn), call PAGTRAN if the page is not in core to BRING. Calculate the CORTABLE entry of the specified page and set the LOCKM bit in its CORTABLE entry.

LOGIN (L) - any user (class A,B,C,D)

```
    LOGIN  userid
```

The LOGIN command is used to initiate a terminal session. Although included here with console functions, the LOGIN command is processed by the LOGON module, and technically is not a CONSOL command.

When the connection between the terminal and the system is established, a recognition message will be sent from the system ("CP-67 Online") indicating that the system is ready to receive users. If the attention key (2741) or the break key (1050) is depressed, the system will respond with a carriage return, and it will unlock the keyboard waiting for an attempted logon process.

The format is LOGIN userid where "userid" is the eight-character or less external identification code assigned to the user by the systems administrator. If the userid is not found in the directory (U.DIRECT), a message is sent to the terminal and the terminal is reinitialized. If the external identification is found in the directory, a request is made for the user to enter his password:

```
    ENTER PASSWORD:
```

and the printer is disabled in preparation for the receipt of his password. In the case of a TTY terminal, each space of the eight-character input area is preprinted with an H, *, and S to hide the password. For the the 1050, and for the 2741 not equipped with the print inhibit feature, the same password protect characters can be obtained by either of the following methods:

```
       issue an x after your userid:
            login 'userid' x
or
       hit RETURN after the message ENTER PASSWORD is
       printed.
```

The password is checked against the directory, and if a match is made, the user is informed of the message of the day (LOGMSG), if any, and of any failures in allocating his virtual machine. Finally, the time and day of login are indicated at the terminal. If the password does not match the one in the directory, an appropriate message is issued. During the LOGIN procedure, CP-67 uses the 2702 read-with-time-out function to prevent unnecessary line tie-up.

LOGOUT (LOG) - any user (class A,B,C,D)

```
LOGOUT  <xxx>
```

where <xxx> is any nonblank character. xxx will prevent disconnect of the line.

This command will cause the user's virtual machine to be deleted from the CP-67 system.

The following steps are taken:

- free the CONSOL functions read buffer and large save area
- call ADSET to process the machine logout

MSG (M)

```
    MSG   userid   text-of-message   -  any user (class A,B,C,D)

    MSG   ALL      text-of-message   -  class A and B
                                        (class A or B)
```

This command is used to communicate with other users currently logged in to CP-67. The users with operator privileges (privilege class A or B) can send a message to all users by specifying ALL for a userid.

The following steps are taken:

- find the desired user; if the ID is "CP", find the system operator

- format the message to identify the sending user

- call WRTCONS to send the text to the user's terminal

- if ALL is specified, repeat the WRTCONS for all users

- send a message to the issuing user if the desired user is not currently accepting messages

PURGE (P) - any user (class A,B,C,D)

```
    PURGE  RDR | PRT | PUN
```

This command will delete all the user's particular spool files that are still awaiting processing.

The following steps are taken:

- starting from the PRINTERS, PUNCHES, or READERS chain, find the spool file blocks for the user
- call MRIDEL to delete the spool file block and to release all the records used by this spool file
- call WRTCONS to give the user a confirmation message

QUERY (Q) - any user (class A,B,C,D), except for certain functions

QUERY parameter

where parameter is either USERS, NAMES, PORT, userid, LOGMSG, MAX, Q2, DEVICE, FILES, or TIME. The parameters can be abbreviated to a unique value, for example, Q F will give the file status.

The following steps are taken for processing each parameter:

USERS - prints the number of logged on and "dialed" users

NAMES - prints the "userid" and 2702 line address of all currently active terminals with virtual machines. Terminals in the process of LOGIN and virtual machines that are DISCONNECTED are shown. The names are displayed four to a line.

"userid" - if the user is logged in, a message of the "NAMES" format is given; if not, "USER NOT ON SYSTEM" is given

LOGMSG - the current LOGMSG is printed

MAX - the current setting for maximum users is given (class A and B)

Q2 - the current size of the "nonconsole" queue (see "DISPATCH" in Section 5) is given (class A and B)

DEVICE - the address and status of the particular device or of all DASD and TAPE devices is given (operator and subsystem operator class A and B)

PORTS - the address and status (userid associated with line, or **FREE**) of the particular line or of all lines, or of all "free" lines is given as requested (class A and B)

FILES - the number of reader, printer, and punch spool files awaiting processing for the user. For the system operator, the status given is for all users, that is, the total number of files.

TIME - gives the connect, virtual and total time used so far by the user

DUMP - prints address of the ABEND dump unit (class A and B)

VIRTUAL - interrogates virtual machine configuration: 'all' as an option (or null option) elicits entire configuration; 'core' for core size, only; 'ccu' for specified device.

READY (R) - any user (class A,B,C,D)

    READY xxx

where xxx is a virtual device address.

The READY command will set a "device-end" interrupt status in the virtual device block.

The following steps are taken:

- locate the user's virtual device block
- set a "device-end" status in the block (VDEVSTAT)
- set a "pending" interrupt status in the user's UTABLE

REPEAT (REP) - class A

REPEAT ccu <nn>

This command will cause the currently active output of the specified device to be repeated nn times (1 is default).

The following steps are taken:

- find the specified device block, MRDEBLOK
- if the device is not active, print a message to the operator
- in the current spool file block, set a bit and a count to indicate that, upon reaching the end, the output should be restarted

RESET (RES) - any user (class A,B,C,D)

```
    RESET
```

This command performs a "system reset" of all the user's virtual devices. All interrupts are cleared.

The following steps are taken:

- call the module RESINT to perform a reset on all virtual devices

SET (SET) - any user (class A,B,C,D), except for certain functions

SET parameter

where parameter is either WNG ON, WNG OFF, MSG ON, MSG OFF, RUN ON, RUN OFF, CARDSAVE ON, CARDSAVE OFF, MAX=nn, LOGMSG, or Q2=nn. The parameters cannot be abbreviated.

The following steps are taken to process each parameter:

WNG ON - reset the WNGBIT in the user's TIMERMOD field of the UTABLE to allow receipt of "warnings", that is, priority messages

WNG OFF - set the WNGBIT in the user's UTABLE field (TIMERMOD) to inhibit receiving "warnings"

MSG ON - reset the MSGBIT to receive messages

MSG OFF - set the MSGBIT to inhibit receiving messages

RUN ON - set the RUNON bit to allow the virtual machine to keep running in "CONSOL function" mode (after "ATTN" interrupt)

RUN OFF - reset the RUNON bit for "normal" virtual machine operation, that is, to stop running on "ATTN"

CARDSAVE ON - set the MVIFSAV bit in the MVIFLAG field of all the users' virtual card readers

CARDSAVE OFF - reset the MVIFSAV bit in all the users' virtual card readers

TRACE ON - initiate tracing functions as specified by the included parameters

TRACE OFF - terminate tracing functions

ADSTOP xxxxxx - stop execution at virtual instruction address xxxxxx

ADSTOP OFF - terminate an address stop function.

The following functions are for class A and B only:

MAX=nn - for the system operator only; to set a value for the maximum number of users allowed to log on (0=no limit)

Q2=nn - for the system operator only; to set a value for the "non-CONSOL" dispatching queue (see "DISPATCH" in Section 5)

LOGMSG - to set or add to the system LOGMSG

LOGMSG NULL - to delete the entire existing log message

LOGMSG n - to set or delete LOGMSG line n

DUMP xxx - to change dump unit and core area dumped

SHUTDOWN (SH) - system operator only

```
    SHUTDOWN
```

This command will immediately terminate system operation with no messages.

The following steps are taken:

- set the CPID word to SHUT to indicate shutdown
- go to the DSKDUMP routine at RESTART to force an IPL of the system so that CHKP can save the machine status

SLEEP (SL) - any user (class A,B,C,D)

```
    SLEEP
```

This command places the terminal in a "prepared" status so that it may receive messages.

The following steps are taken:

- GOTO the DISPATCHER leaving the user in CFWAIT mode (nonrunnable)
- an ATTN will awaken the user

SPACE (SPA) - class A

```
    SPACE   ccu
```

This command will cause the current output on the printer (spool file) to be forced to single spacing. This will avoid excessive forms skipping.

The following steps are taken:

- find the specified printer real multiplexer device block
- set the MRISPACE bit in the MRIFLAG field of MRDEBLOK

SPOOL (SPO) - any user (class A,B,C,D)

```
    SPOOL  ccu  <ON xxx, OFF>
    SPOOL  ccu  <CONT, OFF>
```

This command is used to direct the output of a user's virtual printer or punch to a specific real printer or punch. The command can also specify "continuous" input for virtual card readers.

The following steps are taken:

- find the user's virtual device block (ccu)
- find the system real device block (xxx)
- set the MVIFRMT bit in the MVIFLAG of the user's MVDEBLOK
- store the address of the desired MRDEBLOK in the MVPNTREL field of the MVDEBLOK
- reset these bits if no real device is specified
- for a virtual card reader, set the MVICONT bit in the MVIFLAG field of the MVDEBLOK; or reset the bit for no "CONT" specified

START (STA) - system operator only

```
    START <xxx . . . yyy>
```

This command is used to start a previously drained unit record device.

The following steps are taken:

- the same logic of DRAIN is followed to locate the desired device or devices
- the MRIDRAIN list is reset in the device block
- a dummy "device end" CSW is created and a call is made to MRIOEXEC; this will cause any closed spool file blocks to commence output on the device

STCP (STCP) - class A and B

```
    STCP  arg1 arg2 ... argN
```

where the arguments (arg1...argN) are a real memory location and the data to be stored.

The following steps are taken:

1. The steps are the same as those for STORE except the data is stored in real memory instead of virtual memory.

STORE (ST) - any user (class A,B,C,D)

```
    STORE arg1 arg2 ... argN
```

where the arguments (arg1...argN) specify a virtual memory location, a general-purpose register, a floating-point register, a control register, and/or PSW and the data to be stored.

The following steps are taken:

1. Fetch the next argument and branch to the routine to handle that particular store function by doing a BAL to subroutine STOSCAN.

2. Each store routine sets the increment address and does a BAL to subroutine STOADDR to convert the beginning address to binary.

3. A BAL to STOSCAN is done to obtain the next argument.

4. If the current address is greater than the maximum allowable for the type, a "BAD ARGUMENT XX" message is sent to the user and the store function terminated.

5. The argument is converted to binary and stored at the current address.

6. The increment address is added to the current address to obtain the next address, and the store continues by fetching the next argument (step 3).

Subroutines:

STOSCAN does a BAL to SCANFLD to obtain the location and length of the next argument. The first character of the argument is inspected for a type code (P, G, Y, L, or X). The code is used to select the routine to branch to, to perform the unique processing for each type of store. If no valid code is found, the argument is assumed to be data and return is made to the calling routine to continue the store.

STOADDR converts the beginning address to binary, saves it, and returns to the caller.

TERMINATE (TERM) - system operator only

```
    TERM  xxx
```

where xxx is the real address of a unit record device whose output it is desired to terminate.

The following steps are taken:

- find the MRDEBLOK for the specified device
- set the TERMINAT bit in the MRIFLAG field of MRDEBLOK

UNLOCK (UN) - system operator only

```
    UNLOCK  userid  xxx  nnn
```

This command will unlock a previously LOCKed page.

The following steps are taken:

- the same logical steps as in LOCK, but turn off the LOCKM bit in the core table entry if the specified page is in core

WNG (W) - class A and B

```
    WNG  userid  text-of-message
         ALL
```

The "warning" function operates the same as MSG except that a priority call is made.

The following steps are taken:

- find the specified "userid", or if "ALL" is specified, do the following for all logged-on users:
  - format the message to identify the originator
  - call PRIORITY to send the message to the user immediately
  - send a message to the originator if a user is not receiving warnings

XFER (X) - any user (class A,B,C,D)

```
    XFER  ccu  TO  userid
    XFER  ccu  OFF
```

This command is used to transfer a punch spooled file to the reader input spool files of the specified user.

The following steps are taken:

- find the desired punch device (ccu)

- call USERLKUP to search the CP-67 directory to determine that the "userid" is valid

- move "userid" to the MVIXUSER field in the MVDEBLOK

- set the MVIXFER bit in the MVIFLAG of the MVDEBLOK

- for the OFF option of the XFER command, reset the MVIXFER bit and blank the MVIXUSER field

SECTION 3: PROGRAMMING CONVENTIONS

To allow for the orderly maintenance and growth of the CP-67 operating system, the programming conventions described should be followed by anyone working with CP-67 programs.

MAINTENANCE

The CP-67 system is maintained using the Cambridge Monitor System. A set of catalogued procedures (EXEC files) are distributed with the system (see the CP-67 Installation Guide for their descriptions).

ASSEMBLY DECK FORMATS

All decks contain a TITLE card as the physically first card with a unique label field and a suitable title in the operand field.

The primary entry point of a routine is indicated with a START card, which is the second card of the assembly deck in the absence of macro definitions or comments (required by the loader).

Unless required otherwise, all COPY statements are located at the end of the deck.

The END card must not have any operands. The loader will accept only one of such type, and this must be the one in SAVECP.

Information used by more than one routine will be contained in the file CPMACS MACLIB. This file will contain the macro definitions, equivalence packages, and control block definitions (DSECTs). All parameters and flag bits should be assigned symbolic names and defined in the appropriate equivalence package.

EQUIVALENCE PACKAGES AND CONTROL BLOCK DEFINITIONS

These packages will be included in an assembly by means of the COPY pseudo-operation.

CPFDEF	defines the CPFILE control blocks.
DEVTYPES	defines the CP-67 device type codes. A printout of DEVTYPES follows this list.
EQU67	defines references to physical lower core,

channel command words, CALL parameters, CPEXBLOK definition, etc. A printout of EQU67 follows this list.

IOBLOCKS defines the input-output control blocks and IOTASK block.

OPTIONS contains assembly option switches and macro definitions.

LOCAL contains assembly option with settings for the particular installation.

UDIRECT defines the directory blocks MDENT and UFDENT.

UTABLE defines the UTABLE and EXTUTAB blocks and included flag bits.

Obtain a listing of the appropriate ASP360 or COPY file from the CP-67 distributed system for a detailed and accurate description of the contents of each file.

CP-67 DEVICE CODES

```
********************************************************
*                                                      *
*        CP-67 DEVICE TYPE CODES                       *
*                                                      *
********************************************************
*
TYP1052  EQU   0
TYP1050  EQU   4
TYP2250T EQU   8
TYP2260T EQU   12
TYP2741T EQU   16              MPX/2702  2741
TYP 1052T EQU  20                        1052
TYP2703T EQU   24
TYP2702T EQU   24
TYP2701T EQU   24
TYPTT35T EQU   28              MDL 35 TELETYPE
TYPTTY35 EQU   TYPTT35T
TYPTIMER EQU   44             SIMULATED CHRONOLOG
TYP1403  EQU   48
TYP2540P EQU   52
TYP2540R EQU   60
TYP2671  EQU   64
TYPRMPRT EQU   X'44'          REMOTE PRINTER READER
TYPRMPUN EQU   X'48'          REMOTE PUNCH READER
TYPM20   EQU   96
TYP1800  EQU   100
TYP2311  EQU   128
TYP2314  EQU   132
TYP2302  EQU   136
TYP2321  EQU   140
TYP2301  EQU   144
TYP2303  EQU   148
TYP2250  EQU   180
TYP2260  EQU   184
TYP2400  EQU   192             GENERAL MAG TAPE
TYP2404  EQU   192
TYP2402  EQU   192
TYP2403  EQU   192
TYP7340  EQU   204
TYP2701  EQU   208
TYP2701L EQU   208            L IS A DEDICATED LINE
TYP2702L EQU   208
TYP2703L EQU   208
TYP2700L EQU   208
TYP2702D EQU   212            D IS A DIAL CONNECTED LINE
*
********************************************************
*
```

```
************************************************************************
*                                                                      *
*        CP-67 EQUIVALENCE AND MACHINE DEFINITION PACKAGE              *
*                                                                      *
************************************************************************
*
*              BITS IN STANDARD PROGRAM STATUS WORD
*
PROBMODE EQU   X'01'          PROBLEM MODE BIT.
WAIT     EQU   X'02'          WAIT BIT.
MCHEK    EQU   X'04'          MACHINE CHECK.
ASCII    EQU   X'08'          ASCII BIT.
*
*              BIT ASSIGNMENTS IN EXTENDED PROGRAM STATUS WORD
*
MODE32   EQU   X'08'          24/32 ADDRESSING MODE BIT.
TRANMODE EQU   X'04'          DYNAMIC TRANSLATION MODE BIT.
IOMASK   EQU   X'02'          OVERALL I/O MASK BIT.
EXTMASK  EQU   X'01'          OVERALL EXTERNAL INTERRUPTION MASK BIT.
*
*              DEFINED BITS IN CHANNEL STATUS WORD
*
ATTN     EQU   X'80'          ATTENTION BIT.
SM       EQU   X'40'          STATUS MODIFIER BIT.
CUE      EQU   X'20'          CONTROL UNIT END BIT.
BUSY     EQU   X'10'          BUSY BIT.
CE       EQU   X'08'          CHANNEL END BIT.
DE       EQU   X'04'          DEVICE END BIT.
UC       EQU   X'02'          UNIT CHECK BIT.
UE       EQU   X'01'          UNIT EXCEPTION BIT.
*
PCI      EQU   X'80'          PROGRAM-CONTROLLED INTERRUPT BIT.
WLR      EQU   X'40'          WRONG-LENGTH-RECORD BIT.
PRGC     EQU   X'20'          CHANNEL PROGRAM CHECK
PRTC     EQU   X'10'          CHANNEL PROTECTION CHECK
*
*              FLAGS DEFINED IN CHANNEL COMMAND WORDS
*
CD       EQU   X'80'          CHAIN DATA FLAG.
CC       EQU   X'40'          CHAIN COMMAND FLAG.
SILI     EQU   X'20'          SUPPRESS INCORRECT LENGTH INDICATOR FLAG.
SKIP     EQU   X'10'          SUPPRESS TRANSFER OF INFORMATION.
PCIF     EQU   X'08'          PROGRAM-CONTROILED-INTERRUPT FLAG.
*
*        FLAGS DEFINED IN FIFTH BYTE OF CCW TO AID CCW TRANSLATION
*
RCXIS    EQU   X'80'          CHECK ISAM INDICATOR
RCSUDO   EQU   X'40'               PSEUDO 2311 INDICATOR
RCUTIC   EQU   X'20'          UNTRANSLATED TIC
RCIO     EQU   X'10'          I/O CCW
RCGEN    EQU   X'08'          CP GENERATED CCW
RCDATA   EQU   X'04'          CP GENERATED CHAIN DATA
RC02     EQU   X'02'               RESERVED FOR FUTURE USE
```

```
  RC01      EQU   X'01'                RESERVED FOR FUTURE USE
  *
  *         DEFINED LOCATIONS IN MACHINE (EXTENDED AND STANDARD)
  *
  IPLPSW    EQU   0               INITIAL PROGRAM LOAD PSW.
  IPLCCW    EQU   8               INITIAL PROGRAM LOAD CCWS.
  INTCODES  EQU   14              INTERRUPTION CODES (EXTENDED)
  EXOPSW    EQU   24              EXTERNAL INTERRUPT OLD PSW.
  SVCOPSW   EQU   32              SUPERVISOR CALL INTERRUPT OLD PSW.
  PROPSW    EQU   40              PROGRAM INTERRUPT OLD PSW.
  MCOPSW    EQU   48              MACHINE CHECK INTERRUPT OLD PSW.
  IOOPSW    EQU   56              INPUT-OUTPUT INTERRUPT OLD PSW.
  CSW       EQU   64              CHANNEL STATUS WORD.
  CAW       EQU   72              CHANNEL ADDRESS WORD.
  TIMER     EQU   80              MACHINE INTERVAL TIMER.
  EXNPSW    EQU   88              EXTERNAL INTERRUPT NEW PSW.
  SVCNPSW   EQU   96              SUPERVISOR CALL INTERRUPT NEW PSW.
  PRNPSW    EQU   104             PROGRAM INTERRUPT NEW PSW.
  MCNPSW    EQU   112             MACHINE CHECK INTERRUPT NEW PSW.
  IONPSW    EQU   120             INPUT-OUTPUT INTERRUPT NEW PSW.
  SCANOUT   EQU   128             DIAGNOSTIC SCAN-OUT SECTION.
  CHANLOG   EQU   304             CHANNEL LOGOUT AREA (2860,2870)
  *
  *               STORAGE LOCATIONS USED BY THE CONTROL PROGRAM
  *
| RUNUSER   EQU   X'160'
| CPSTATUS  EQU   RUNUSER+4
| *                      BITS DEFINED IN CPSTATUS
| *
| CPIDLE    EQU   X'80'
| VMDONE    EQU   X'40'
| *IOMASK   EQU   X'02'           ON..FOR I-O ENABLED PROCESSOR
| *
| *
| MONTHS    EQU   CPSTATUS+1
| DAYS      EQU   MONTHS+1
| YEARS     EQU   DAYS+1
| HOURS     EQU   YEARS+1
| MINUTES   EQU   HOURS+1
| SECONDS   EQU   MINUTES+1
| *
| STARTIM   EQU   HOURS+4
| BINTIME   EQU   STARTIM+8
| DISPSW    EQU   BINTIME+4       NOTE: MUST BE DOUBLE WORD BOUNDARY
| *
| *         CP POINTERS FOR CPINIT, CHKPT AND BUZZARD
| *
| ASYSWRM   EQU   DISPSW+8        WARM START CYL ADDRESS
| ASYSINF   EQU   ASYSWRM+4       LOGMSG START
| ASYSCNSL  EQU   ASYSINF+4       1052 CONSOL ADDRESS LOC
| CPID      EQU   ASYSCNSL+4      CP-67 IDENTIFIER
| ARMXST    EQU   CPID+4          REAL MPX CHAIN START
| ARECBUF   EQU   ARMXST+4        SPOOL BUFFER START
| AZVOL     EQU   ARECBUF+4       ZERO VOLUME DEVICE
| APRINT    EQU   AZVOL+4         PRINTER FILE CHAIN
| APUNCH    EQU   APRINT+4        PUNCH FILE CHAIN
| AREADERS  EQU   APUNCH+4        READER FILE CHAIN
| AMREAL    EQU   AREADERS+4      ACCOUNTING CARD CHAIN
```

```
ARCHSTRT EQU   AMREAL+4           REAL SEL CHAN START
*
CPUTAB   EQU   ARCHSTRT+4         TABLE OF CPU'S AND PREFIXED PAGE 0
CPUOTH   EQU   CPUTAB+23          CPU IDS OF OTHERS
CPUID    EQU   CPUTAB+27          CPU ID WITHOUT EXTRANEOUS BITS
CPUSCR   EQU   CPUTAB+31          SCRATCH BYTE FOR CPUL/F
*
TEMPSAVE EQU   CPUTAB+48          TEMPORARY SAVE FOR INTERRUPT HANDLERS
*
BALRSAVE EQU   TEMPSAVE+64        FAST LINKAGE SAVE .. 80 BYTES
*
DISPATWK EQU   BALRSAVE+80        WORK AREA FOR DISPATCH (8 WORDS)
*
RUNINTIM EQU   DISPATWK+32        1 SECOND INTERVAL BINARY TIMER
DSCRO    EQU   RUNINTIM+4         CURRENT SEGMENT TABLE ORIGIN
KALG     EQU   DSCRO+4            PAGING ACTIVITY CONTROL
LOCKOUNT EQU   KALG+4             COUNT OF CURRENTLY LOCKED PAGES
MAXLOCK  EQU   LOCKOUNT+2         MAX. VALUE OBTAINED BY LOCKOUNT
*
*
*****    PSA ASSEMBLED DATA STARTS AT X'340'.
*
*****    CURRENT DEFINITION OF STAT COUNTERS STARTS AT X'350'.
*
*
*        TIMING MEASUREMENTS:
CPTIME   EQU   X'350'             CPU TIME IN SUPERVISOR STATE
PROBTIME EQU   CPTIME+4           CPU TIME IN PROBLEM    STATE
WAITTIME EQU   PROBTIME+4         CPU TIME IN WAIT       STATE
*
OVERHEAD EQU   WAITTIME+4         SUPVR TIME NOT CHARGED TO USERS
WAITIDLE EQU   OVERHEAD+4         WAIT TIME FROM PERIODS GTE 1/4 SEC.
WTPAGE   EQU   WAITIDLE+4         TIME SPENT WAITING FOR A PAGE
WTUSR    EQU   WTPAGE+4           TIME SPENT WAITING WITH N-IN-Q RUNNABLE USER
WTUSRA   EQU   WTUSR+4            WTUSR * NUMBER OF NON-IN-Q RUNNABLE USERS
*
*
*        CPU EVENT COUNTERS:
*
KPGEX    EQU   WTUSRA+4           COUNT OF PAGING EXCEPTIONS
PGREAD   EQU   KPGEX+4            PAGES READ IN
PGSWAP   EQU   PGREAD+4           PAGE SWAPS
QCOUNT   EQU   PGSWAP+4           COUNTER: USER IN Q LOST PAGE
*
*
*        INSTALLATION USER (4 WORDS):
*
INSTWRD1 EQU   QCOUNT+4
INSTWRD2 EQU   INSTWRD1+4
INSTWRD3 EQU   INSTWRD2+4
INSTWRD4 EQU   INSTWRD3+4
*
*
*        USER EVENT COUNTERS:
*
STATUSER EQU   INSTWRD4+4         COUNTERS FOR USER INSTR. STREAM EVENTS.
***********************************************************************
*                                                                     *
*     DEFINITION OF STATISTICS COUNTERS IN CP CORE --                 *
*      COUNTERS OF USER EVENTS.                                       *
*                                                                     *
***********************************************************************
STATINST EQU   STATUSER
*  COUNT OF INTERRUPTS
STATUEXT EQU   STATINST       COUNT OF USER EXT INTERRUPTS REFLECTED
STATUSVC EQU   STATUEXT+4     COUNT OF USER SVC INTERRUPTS REFLECTED
STATUPGM EQU   STATUSVC+4     COUNT OF USER PGM INTERRUPTS REFLECTED
STATUIOI EQU   STATUPGM+4     COUNT OF USER I/O INTERRUPTS REFLECTED
*
*  COUNT OF PRIVILEGED INSTRUCTIONS
STATSSK  EQU   STATUIOI+4     COUNT OF USER 'SSK' INSTRUCTIONS
STATISK  EQU   STATSSK+4      COUNT OF USER 'ISK' INSTRUCTIONS
STATSSM  EQU   STATISK+4      COUNT OF USER 'SSM' INSTRUCTIONS
STATLPSW EQU   STATSSM+4      COUNT OF USER 'LPSW' INSTRUCTIONS
STATDIAG EQU   STATLPSW+4     COUNT OF USER 'DIAGNOSE' INSTRUCTIONS
STATDDSK EQU   STATDIAG+4     COUNT OF DIAGNOSE DISK IO INSTRUCTIONS
STATSIO  EQU   STATDDSK+4     COUNT OF USER 'SIO' INSTRUCTIONS
STATTIO  EQU   STATSIO+4      COUNT OF USER 'TIO' INSTRUCTIONS
STATHIO  EQU   STATTIO+4      COUNT OF USER 'HIO' INSTRUCTIONS
STATTCH  EQU   STATHIO+4      COUNT OF USER 'TCH' INSTRUCTIONS
*
*  PRIVILEGED INSTRUCTIONS FOR VIRTUAL 67
STATWRD  EQU   STATTCH+4      COUNT OF 67 USER 'WRD' INSTRUCTIONS
STATSTMC EQU   STATWRD+4      COUNT OF 67 USER 'STMC' INSTRUCTIONS
STATLRA  EQU   STATSTMC+4     COUNT OF 67 USER 'LRA' INSTRUCTIONS
STATLMC  EQU   STATLRA+4      COUNT OF 67 USER 'LMC' INSTRUCTIONS
*
*  MODULE COUNTERS
STATDSP  EQU   STATLMC+4      COUNT OF CALLS TO CKUSR IN DISPATCH
*
*              BITS DEFINED FOR CORE MANAGEMENT ROUTINES
*
BRING    EQU   X'01'          BRING REQUESTED PAGE IN.
CHANGED  EQU   X'02'          STORAGE KEY  ,     PAGE CHANGED
USED     EQU   X'04'          STORAGE KEY,     PAGE REFERENCED
DEFER    EQU   X'08'          RETURN CONTROL ONLY AFTER PAGE IS IN CORE
LOCK     EQU   X'10'          SET LOCK BIT ON REQUESTED PAGE.
*        BITS SET IN SWPTABLE ENTRIES
SHARED   EQU   X'10'     PAGE IS SHARABLE, SET IN SWPTABLE
TRANSIT  EQU   X'80'          TRANSIT BIT FOR CORE HANDLER ROUTINES
RECOMP   EQU   X'40'          RECOMPUTE DASD ADDRESS IN SWPTABLE
*        BITS SET CORTABLE ENTRIES
*TRANSIT EQU   X'80'          SAME AS SWAPTABLE
LOCKON   EQU   X'40'          NON-ZERO LOCK COUNT FOR THIS PAGE
LOCKCM   EQU   X'20'          LOCK COMMAND SET FOR THIS PAGE
*
*              PARAMETER VALUES PROVIDED TO 'RDCONS' OR 'WRTCONS'
*
EDIT     EQU   1              PERFORM LINE EDITING FUNCTION.
UCASE    EQU   2              TRANSLATE LOWER TO UPPER CASE.
NORET    EQU   4              DON'T RETURN WHEN THROUGH.
DFRET    EQU   8              PERFORM 'FRET' OF SPECIFIED AREA.
NOAUTO   EQU   16             NO AUTOMATIC-CARRIAGE-RETURN WANTED
OPERATOR EQU   32             MESSAGE TO/FROM OPERATOR
ALARM    EQU   64             SEND ALARM TO USER TERMINAL
*
```

```
*              REGISTER EQUIVALENCES
*
R0       EQU   0              ..
R1       EQU   1              ..
R2       EQU   2              ..
R3       EQU   3              ..
R4       EQU   4              ..
R5       EQU   5              ..
R6       EQU   6              ..
R7       EQU   7              ..
R8       EQU   8              ..
R9       EQU   9              ..
R10      EQU   10             ..
R11      EQU   11             ..
R12      EQU   12             ..
R13      EQU   13             ..
R14      EQU   14             ..
R15      EQU   15             ..
*
CPEXBLOK DSECT                CONTROL PROGRAM EXECUTION REQUEST BLOCK
CPEXNEXT DS    1F             POINTER TO NEXT REQUEST.
CPEXADD  DS    1F             ADDRESS TO RECEIVE CONTROL.
CPEXRLGS DS    16F            REGISTERS TO RESTORE (EX. 15)
CPEXMISC DS    2F             UNASSIGNED.
*
CPEXSIZ  EQU   (*-CPEXBLOK)/8 ..
*                                                                     *
***********************************************************************
*
```

SUBROUTINE CONVENTIONS AND REGISTER USAGE

Except for certain isolated instances, the following conventions relative to subroutine calling sequences and addressability apply throughout CP-67.

Addressability is via register 12. Subroutines may assume that register 12 is properly loaded at the time the subroutine is entered.

The first instruction of a normally called subroutine should be the ENTER macro described in the section on macro usage. The return point of the subroutine should use the EXIT macro.

Register 13 points to a valid save area usable by the routine being called. It is 24 words in length. The first three words are reserved and used by the call linkage handler to save return information. Word 1 is the return address, word 2 is the caller's R12 (return base), and word 3 is the caller's R13 (return save area). The remainder of the space is used as the called routine sees fit. The ENTER and EXIT macros will store the saved registers into the area beginning at the fourth word of the save area. The called subroutine may not change the contents of register 12 or 13. Any registers that are changed must be restored, with the exception of registers 14 and 15, which may be considered destructible.

Subroutines expecting to return to the calling program should be called with the CALL macro. Subroutines which are called with the CALL macro and which will not return via the EXIT macro should perform an SVC 16 to return the currently assigned save area back to usable storage. This type of code should be avoided, if possible. It is used by second level interrupt handlers to bypass returning to the first level handler under specific circumstances.

Unconditional transfers to routines which expect no return should be made via the GOTO macro. The routine thus called has access to the same save area which the calling routine used.

Parameter transfers to subroutines will generally be via general purpose registers to enhance the ease of coding in a reentrant fashion. The specific calling sequences depend upon the subroutine being called, with the exception that if the PARM= parameter of the CALL macro is used, register 2 will be modified within the CALL macro.

Register 11 contains the UTABLE address for the user being serviced.

The following macros are defined and their usage explained:

CALL - establishes subroutine linkages via SVC interrupt
CPUF, CPUL - CPU lock protect for multiprocessing (not now functioning)
ENTER, EXIT - save and restore registers at entry and exit of system routines
GOTO - same parameters as CALL, but no return from called routine

TRANS - facilitates translation of virtual to physical memory address, with necessary paging

BAS, BASR, LMC, STMC, and LRA

Macros BAS, BASR, LMC, STMC, and LRA are merely defined to be equivalent to the machine instructions to be assembled. These macros are provided in the absence of the corresponding mnemonics of the F-level OS/360 Assembler so as to include them in its operation dictionary.

CALL

Subroutine linkages in the Control Program (with the exception of the call to SVCINIT in CPINIT) are made via the CALL macro, which generates the appropriate call (via SVC interrupt) to the supervisor, enabling automatic generation and stacking of save areas, etc. The format is:

```
|label|  CALL  <subr,gpr>,|EXTERNAL|,|PARM=(arg1+arg2+...)|
```

where "label" refers to the first generated machine instruction in the expansion; "subr" refers to a subroutine name (either defined internally or externally), or "gpr" refers to a general purpose register number (self-defining, not "R1"). "EXTERNAL" as an optional argument indicates that a V-type address constant is to be generated. The optional "PARM" argument, if included, provides for the loading of GPR 2 with the parameters indicated (specified normally as EQU values). If all parameters are to be turned off, PARM=0 must be specified; otherwise, GPR2 will not be set in the macro expansion.

ENTER and EXIT

The ENTER and EXIT macros are placed at the entry and exit points of system routines within the Control Program. They perform the function of saving and restoring registers and exiting to the calling program. Their format is:

```
|label|  ENTER  |<reg1 |,reg2|>|
         EXIT
```

With the standard calling sequence under the CALL macro description above, provision is made for the standard supply of save areas in an efficient manner. The ENTER and EXIT macros enable easy use of this facility.

If no arguments are provided, no saving of registers takes place at entry to the routine. If a single register is stated, it alone is saved in the provided save area at 12(13). If a range is provided, these registers are saved beginning at 12(13). The first three words of the save area are never to be modified except by the SVCINT routine. Sufficient space is provided for the saving of all registers. Care must be taken that the registers are restored (via the EXIT macro) in the same manner as they were stored in the ENTER instruction. The parameters for matching ENTER and EXIT pairs should be identical.

GOTO

The format of the GOTO macro is:

```
|label|  GOTO  <subr,gpr>|,PARM=(arg1+arg2+arg3+...)|
```

The parameters are identical to those of the CALL macro (see "CALL"). The difference is that the routine doing the GOTO will not expect a return from the called routine. Therefore, no provision is made for the generation of a save area address. The called subprogram may make use of the same save area as the calling program.

TRANS

The TRANS macro is used whenever a virtual address is to be translated to a physical memory address, and the page, if not core resident, may be required to be paged in. Its format is as follows:

```
|label|  TRANS  rgpr,vgpr|,OPT=(a(1),a(2),...)|
```

where "rgpr" is the register to receive the translated address; "vgpr" is the register containing the virtual address. OPT is an optional parameter which has as subparameters those options provided to the PAGTRANS routine via the CALL macro. These options will be passed in the event a call to PAGTRANS is required. They are discussed below.

Note: "rgpr" and "vgpr" cannot be the same register.

If LOCK is specified, PAGTRANS is called as it would be normally. If BRING is specified, the LRA instruction is used to determine whether the page is currently resident. If it is not, PAGTRANS is called as it would be normally; otherwise the call is bypassed. If neither is specified, the LRA alone is used and the condition code set. Note that if a call to PAGTRANS is required, registers 1,2, and 15 will not be preserved over the macro. If DEFER is specified, control will not be returned until the page is in core. If USED is specified, the used bit will be set for the specified page. If CHANGE is specified, the changed bit will be set for the specified page.

The following conditional branch macros are defined:

B(N)PE(R)	Branch on (no) page exception (RX, RR);
B(N)RE(R)	Branch on (no) reloc. exception (RX, RR);
B(N)SE(R)	Branch on (no) segment exception (RX, RR).

This section contains illustrations representing the formats of blocks and tables used by the control program. A brief description of the contents and use of the tables is also given. Further details may be found in the preceding and following sections. (In the list below, "1/" means one per user, device, etc.)

The following control blocks are described:

ALLOCTBL - index to DASD space available to CP for paging and spooling

CCWPKG - one for each request for a CP-67 terminal read or write

CORTABLE - eight-byte entry/page of real memory, indicating resident virtual page, user, and real page lock condition

CPEXBLOK - request for some CP-67 program execution that has been previously deferred pending an event

CPFDENT - CPFILE system dictionary entry containing file name and location of first record

CPFFDBLK - a file descriptor block/open file in CP File System routines (CPFILE), describing read/write status, etc.

CPFRECRD - record format of CPFILE records (user directory files,machine descriptor files, system directory) on systems tracks

EXTUTAB - one for each virtual 360/67. It is an extension of the UTABLE containing the information peculiar to a virtual 67.

IOTASK - 1/active user selector channel task and each CP-initiated I/O operation

LOGCDATA - describes the format of the error records saved by CP-67 for channel checks

LOGIDATA - describes the format of the error records saved by CP-67 for I/O errors

LOGMDATA - describes the format of the error records saved by CP-67 for machine checks

MDENT - machine description entry created by DIRECT to describe a device in a user's virtual machine

MRDEBLOK - 1/real multiplexer device defined in the system

MRIBUFF - buffer for spooled packed data when handled for real equipment; chained from MTASK in MRDEBLOK

MVDEBLOK - 1/virtual multiplexer device attached to a user's UTABLE

MVIBUFF - buffer for spooled packed data; chained from MVIOB in MVDEBLOK

PAGTABLE - describes status and main storage address of a virtual memory page

RHEADR, RCCWLIST - 1/user CCW list describing location and number, etc., of CCW's in user list

RCHBLOK - 1/real channel, describing pending tasks, channel address and status, attached control units, etc.

RCUBLOK - 1/real control unit, describing channel and devices attached as well as control unit status, address, etc.

RDCONPKG - one for each request for a CP-67 terminal read; contains return status information

RDEVBLOK - 1/any real device, describing address, device type, control unit, task block, etc.

RECBUF - 1/cylinder, describing records available/in use, cylinder number, etc.

SAVEAREA - format of the active and inactive save areas used in subroutine linkage

SEGTABLE - 1/user, describing user page table entries

SFBLOK - one for each "closed" file for spooled input and output

SWPTABLE - 1 entry/PAGTABLE entry, describing page swap area addresses

TREXT - built as a UTABLE extension when user invokes tracing functions

UFDENT - user file directory user information (ID,password,etc.) and user system access information (privilege class, priority code)

UTABLE - 1/user; primary control block onto which other user blocks are strung, reflecting complete virtual machine status

VCHBLOK - 1/virtual channel for each user, describing channel status, address, attached control units, etc.

VCUBLOK - 1/virtual control unit describing control unit status, address, attached devices, etc.

VDEVBLOK - 1/virtual device for each user, describing device address, status, corresponding real device control block, etc.

There is an allocation block for each volume which is "owned" by the system for uses such as paging and spooling. The module TMPSPACE scans down a list depending on device type (T2311 for 2311 disks, T2301 for 2301 drums, etc.). The format of the allocation table block in free storage is as follows:

```
     0        2        4
     +--------+--------+
   0 | Pointer to next |
     +--------+--------+
   4 |Pntr. to RDEVBLOK|
     +--------+--------+
   8 |                 |  256 Bytes - 2301
     |    Allocation   |  200 Bytes - 2303
     |      Data       |  202 Bytes - 2311,2314
     |                 |
     +----+---+--------+
     | 0F |
     +----+
```

where:

The first word is a pointer to the next block in the allocation queue for this type of device.

The second word is a pointer to the real device block on which this volume is mounted.

The allocation data for the drum consists of one bit for every page on the drum indicating whether the page is available for system paging (bit contains a 0) or in use (bit contains 1). The allocation tables are preformatted so that only those pages on a given drum track which are available for paging are initialized to zero. For the 2301 drum, one byte represents one drum track. Since five pages may fit on even track address and four on odd track address, the allocation table is initialized to X'270F070F...070FFF so that unavailable pages or illegal addresses are not selected as swapping space.

The halfword at location 212 contains the count of allocated records on this device and the 9 words located at bytes 220-256 contain pointers to the first IO task which references the corresponding records 1 through 9 on the drum.

For the 2303 drum, the mask is set to X'2F0F0F0F....0F0FFF'. The value FF indicates the end of the allocation table.

The allocation data for the 2311 and 2314 disks consists of one byte per cylinder indicating whether the cylinder is available for temporary use. If the cylinder is available, the byte contains 00; if not, it contains 08. The 0F indicates the end of the allocation table for this device.

For 2311 and 2314, any cylinder on an "owned" volume can be allocated for "temp" use (paging or spooling); "perm" (not available); "tdsk" (for T-disk allocation); or "drct" (for directory use). Only those cylinders marked "temp" (X"00") are available for spooling or paging.

CCWPKG

There is one CCWPKG for each terminal I/O request (read or write) generated by CP-67 or the virtual machine (virtual 1052 I/O). The CCWPKG's are chained from each user's UTABLE at CIOREQ.

```
      0        2        4        6        8
     0+--------+--------+--------+--------+
      |   NEXTCCWP      |JSPARE  |NUMWDCCW|
     8+--------+--------+--------+--------+
      |   PNTRDCON      |   JDEVICE       |
    16+--------+--------+--------+--------+
      |             CCWLIST               |
     ___                                 ___

     ___                                 ___
      |                                   |
      |                                   |
      +--------+--------+--------+--------+
```

where:

NEXTCCWP is a pointer to the next CCWPKG or zero if it is the last.

JSPARE are flag bytes for processing; the second byte contains the parameters (bits in R2 24-31) of the call to RDCONS or WRTCONS for the I/O, for example, NORET=X'04', OPERATOR=X'20'.

NUMWDCCW is the size of this package in doublewords.

PNTRDCON is a pointer (zero if none) to a RDCONPKG which becomes a CPEXBLOK for CPSTACK upon completion of this I/O operation.

JDEVICE is the terminal address.

CCWLIST is one or more (depending upon terminal type and operation) CCW's to perform the I/O.

CORTABLE

The CORTABLE contains a 16-byte entry for each 4096 bytes of real memory. It is created by CPINIT at system initialization time, depending on the size of real memory. The relative position of the entry indicates the core address of the page described. Its format is as follows:

```
    0                 2                 4
    +--------+--------+--------+--------+
  0 |      Pointer to SWPTABLE Entry    |
    +--------+--------+--------+--------+
  4 |Lock MSK|     UTABLE Pointer       |
    +--------+--------+--------+--------+
  8 |             Unused                |
    +--------+--------+--------+--------+
  C | Unused          | Lock CNT        |
    +--------+--------+--------+--------+
```

where:

The first four bytes contain a pointer to the corresponding SWPTABLE entry for the virtual page which currently occupies this real page (or zero if not in use).

The Lock MSK is a one-byte availability indicator. The bit X'80' indicates that the page is in transit. The bit X'40' indicates a nonzero Lock CNT. The bit X'20' indicates that the lock command has been issued for this page.

The UTABLE Pointer points to the user whose page is in that core space. A value of X'00FFFFFF' indicates that the page is available. If the UTABLE Pointer contains *CP*, that core space contains the CP nucleus; if FREE, it is for CP's free storage.

The Lock CNT is an integer indicating the number of outstanding locks on this real page for input-output purposes. The maximum lock count is 65,535.

CPEXBLOK

A CPEXBLOK represents a request for some CP-67 program execution that has been previously deferred pending an event. The CPEXBLOKs are chained to the desired user's UTABLE, and have the following format:

```
    0        4        8
    +--------+--------+
  0 |CPEXNEXT|CPEXADD |
    +--------+--------+
  8 |CPEXREGS         |
    |                 |
    |                 |
    +--------+--------+
 48 |CPEXMISC         |
    +-----------------+
```

where:

CPEXNEXT is a pointer to the next CP request block if any.

CPEXADD is the instruction address to resume CP execution.

CPEXREGS are the 16 general registers saved when the deferred execution request was set up.

CPEXMISC is for miscellaneous use by the routine that created the block.

CPFDENT

The CPFDENT block is the description of an entry in the system file directory which resides on the system residence volume. It is contained in a data record which is described in CPFRECRD. Its format is as follows:

```
        0        2        4        6        8
        +--------+--------+--------+--------+
      0 |             CPFDNAME              |
        +--------+--------+--------+--------+
      8 |        CPFVOL1           |xxxxxxxx|
        +--------+--------+--------+--------+
     10 |             CPFDPOS               |
        +--------+--------+--------+--------+
```

where:

CPFDNAME is the eight-character file name.

CPFVOL1 is the volume label of the disk volume containing the first record.

CPFDPOS is the position within the first volume of the first record - in the format BBCCHHRx.

CPFFDBLK

There is one CPFS file descriptor block for each open file in the Control Program File System routines (CPFILE). Its format is as follows:

```
           0        2        4        6        8
           +--------+--------+--------+--------+
        0  |     CPFNEXT     |     CPFRDEV     |
           +--------+--------+--------+--------+
        8  |              CPFNAME              |
           +--------+--------+--------+--------+
       10  |        CPFVOLID          |C*1 |C*2|
           +--------+--------+--------+--------+
       18  |             CPFFDPOS              |
           +--------+--------+--------+--------+
       20  |CPFUPDPT|CPFRDPT |CPFBYTER|xxxxxxxx|
           +--------+--------+--------+--------+
       28  |    CPFBUFAD     |     CPFPQUE     |
           +--------+--------+--------+--------+
```

where:

CPFNEXT points to the next open file.

CPFRDEV points to the real device of the current record being read.

CPFNAME is the eight-character file name.

CPFVOLID is the volume identification of the current record.

C*1 - CPFSTAT is the file status:
X'80' indicates file open for writing;
X'40' indicates file open for reading.

C*2 - CPFLOCK is the file lock (for use by writing and updating).

CPFFDPOS is the position of the current record on the real device.

CPFUPDPT is the pointer for the update function.

CPFRDPT is the pointer for the read function.

CPFBYTER is the count of the bytes remaining to be read or updated.

CPFBUFAD is the buffer address for this open file.

CPFPQUE is the queue of locked file requests (not implemented).

CPFRECRD

The following is a description of the record format of all CPFILE records on system-owned tracks:

```
           0        2        4        6        8
           +--------+--------+--------+--------+
        0  |        CNEXTVOL          |xxxxxxxx|
           +--------+--------+--------+--------+
        8  |             CNEXTPOS              |
           +--------+--------+--------+--------+
       10  |     CRECLNG     |                 |
           +--------+--------+                 |
           |                                   |  - 829 Bytes
           |              CPFDATA              |
           |                                   |
           +--------+--------+--------+--------+
```

where:

CNEXTVOL is the label of the pack containing the next record. (Note: A zero entry indicates that this is the last record.)

CNEXTPOS is the position of the next record within the pack specified by CNEXTVOL.

CRECLNG is the number of valid data bytes in CPFDATA.

CPFDATA is the actual data in the record, which may be user directory files, machine description files, or the system directory itself.

Note: All physical records are CPRECSZ bytes long (currently 829) . CRECLNG establishes the end of the valid data in the buffer. Logical records, which are of a length defined by the calling program to CPFILE, are not split over physical records.

EXTUTAB

There is one EXTUTAB for each virtual 67 in the system. It contains all the information peculiar to a virtual 67; its format is as follows:

```
0        2        4        6        8
+--------+--------+--------+--------+
|    VCR0         |    VCR1         |
/-----------------+-----------------/
.                                   .
.                                   .
.                                   .
/-----------------+-----------------/
|    VCR14        |    VCR15        |
+-----------------+-----------------+
| SHADVCR0        |E*1|E*2|COPYPAGT |
+-----------------+-----------------+
| COPYSEGT        | IMAGESGT        |
+-----------------+-----------------+
```

where:

VCR0 to VCR15 are the contents of the virtual control registers 0 to 15.

SHADVCR0 is a pointer to the shadow segment table.

LSTBYTST (E*1) is the last byte of the free storage area address reserved for the shadow segment table.

NBVSEGT (E*2) contains 0 if the virtual machine is using only segment 0, and 1 if not.

COPYSEGT contains the length of virtual segment 0 (minus 1) if the virtual machine is using only segment 0; otherwise, it contains the address of the copy of the virtual segment table currently in use.

IMAGESGT contains the first virtual segment table entry if the virtual machine is using only segment 0; otherwise, it contains the address of the image of the shadow segment table, with the unavailable bit in each entry.

IOTASK

There is one IOTASK block for each user selector channel task active in the system. A task is active from the time the user performs the SIO operation (at which time the block is created from free storage and queued onto the appropriate channel task list) until the device is freed (at which time the block is returned to free storage). Its format is:

```
       0        2        4        6        8
       +--------+--------+--------+--------+
    0  |    TASKRDEV     |    TASKRCU      |
       +--------+--------+--------+--------+
    8  |    TASKPNT      |TP*|TF* |TASKVADD|
       +--------+--------+--------+--------+
   10  |    TASKUSER     |    TASKCAW      |
       +--------+--------+--------+--------+
   18  |    TASKIRA      |    TASKMISC     |
       +--------+--------+--------+--------+
       |                 |                 |
       +--------+--------+--------+--------+
```

where:

TASKRDEV is the pointer to the real device control block for this task.

TASKRCU is a pointer to the real control unit on which this task is being executed.

TASKPNT is the pointer to the next task on the list strung on the channel.

TP*-TASKPATH contains a bit in the position corresponding to the control unit on which the task is to be executed; this bit is used to scan for availability of the control unit.

TF*-TASKFLAG contains a bit pattern to indicate task status. The following bits are defined:

X'80'	reserved for future use
X'40'	reserved for future use
X'20'	error in this I/O operation
X'10'	CP-67 I/O (paging,spooling,etc.)
X'08'	CP-67 split seek
X'04'	channel free on this interrupt
X'02'	processing CC 1 for this task
X'01'	stand-alone seek operation

TASKVADD is the address of the virtual device originating the input-output request.

TASKUSER is a pointer to the appropriate user's UTABLE block.

TASKCAW is a pointer to the real channel command list for this operation.

TASKIRA is a pointer to the routine which will be given control on any interrupt resulting from this operation. If a nonzero condition code is encountered on the SIO for this task within the CHFREE module, control will be passed to the TASKIRA, with register 0 containing the condition code. On an interrupt, register 0 will contain a zero to so indicate.

TASKMISC is a slot which may be used by the originator of the IOTASK block for whatever purposes required.

For user selector channel operations, TASKMISC holds the values of registers 6, 7, and 8 (three words) which are the addresses of the virtual channel, control unit, and device blocks respectively. These values are used to re-load the same registers upon receiving the I/O interrupt.

Note: The IOTASK for CP-initiated I/O functions is generally associated with other control blocks and is often integrated with them (for example, MVIBUFF). In these cases, only the first four doublewords of the IOTASK are present.

LOGCDATA

LOGCDATA is a description of the format of the error records saved by CP-67 for channel checks:

```
    0        2        4        6        8
  0 +--------+--------+--------+--------+
    |   LOGSNSE                |LOG |LOG|
    |                          |CODE|TYP|
   8+--------+--------+--------+--------+
    |   LOGVOLID               |LOGADDR |
  16+--------+--------+--------+--------+
    |   LOGDATE                |unused  |
  24+--------+--------+--------+--------+
    |   LOGCSW                          |
  32+--------+--------+--------+--------+
    |   LOGIOPSW                        |
  40+--------+--------+--------+--------+
    |   LOGCHLOG                        |
  64+--------+--------+--------+--------+
    |   LOGCAW                          |
  68+--------+--------+--------+--------+
```

where:

LOGSNSE, LOGCODE, LOGTYPE, LOGVOLID, LOGADDR, LOGDATE and LOGCSW are the same as in the LOGIDATA control block.

LOGIOPSW is the old I/O PSW which was stored at the time of the error.

LOGCHLOG contains the channel logout data.

LOGCAW contains the channel address word at the time of the error.

LOGIDATA

LOGIDATA is a description of the format of the error records saved by CP-67 for I/O errors:

```
        0        2        4        6        8
       0+--------+--------+--------+--------+
        |   LOGSNSE                |LOG |LOG|
        |                          |CODE|TYP|
       8+--------+--------+--------+--------+
        |   LOGVOLID               |LOGADDR |
      16+--------+--------+--------+--------+
        |   LOGDATE                |unused  |
      24+--------+--------+--------+--------+
        |   LOGCSW                          |
      32+--------+--------+--------+--------+
        |   LOGCCWS                         |
     104+--------+--------+--------+--------+
        |   LOGSKLOC                        |
     112+--------+--------+--------+--------+
```

where:

LOGSNSE contains the six I/O sense bytes.

LOGCODE contains the type of I/O or channel error.

LOGTYPE is the type of device upon which the error occurred.

LOGVOLID is the volume serial number of the device upon which the error occurred (if known to CP).

LOGADDR is the channel/unit address of the erring device.

LOGDATE contains the date and time of the error.

LOGCSW contains the channel status word at the time of the error.

LOGCCWS contains the failing CCW string (up to nine CCW's).

LOGSKLOC contains the last seek address prior to the failure.

LOGMDATA

LOGMDATA is a description of the format of the error records saved by CP-67 for machine checks:

```
       0        2        4        6        8
      0+--------+--------+--------+--------+
       |  LOGMDATE                |LOGMCODE|
      8+--------+--------+--------+--------+
       |  LOGMCPU                          |
    184+--------+--------+--------+--------+
       |  LOGMPSW                          |
    224+--------+--------+--------+--------+
       |  LOGMGRS        |  LOGMCRS        |
    352+--------+--------+--------+--------+
       |  LOGMFPRS                         |
    384+--------+--------+--------+--------+
```

where:

LOGMDATE contains the date and time of the machine check.

LOGMCODE contains the machine check code.

LOGMCPU contains the CPU logout data.

LOGMPSW contains the five old PSW's at the time of the machine check (external, SVC, program, machine check, and input-output).

LOGMGRS contains the values of the general registers at the time of the failure.

LOGMCRS contains the values of the extended control registers at the time of the failure.

LOGMFPRS contains the values of the floating point registers at the time of the failure.

MDENT

MDENT is the machine description entry created by DIRECT to describe a device in a user's virtual machine. It is pointed to by a UFDENT entry. The format of MDENT is as follows:

```
         0        2    4    6        8
         +--------+----+---+--------+--------+
      0  | MDADR  |M*1 |M*2|      MDID       |
         +--------+----+---+--------+--------+
      8  |  MDID  |xxxxxxxx| MDRELN | MDSIZE |
         +--------+--------+--------+--------+
     10  |            MDRDPASS               |
         +-----------------------------------+
     18  |            MDRWPASS               |
         +-----------------------------------+
```

where:

MDADR is the virtual device address.

M*1 - MDSTAT is the unit status information:

UNITEMP X'80' indicates temporary device allocation.

UNITDED X'40' indicates that the real device specified in MDID is to be dedicated to this user.

UNRDONLY X'20' indicates a read-only volume.

UNITRMT X'10' indicates that spooled output is to be sent to the real device specified by MDID.

UNRWRIT X'08' if on denotes that the device is shareable in write mode.

UNCONT X'04' if on denotes that the virtual card reader will read all spool files as one.

UNRWMULT X'02' if on denotes that multiple write users are allowed.

UNRDSHAR X'01' if on, denotes that the device is shareable for read-only.

M*2 - MDTYPE contains the virtual device type.

MDID contains a six-byte volume label for DASD volumes. If UNITDED or UNITRMT is on, MDID is of the form "IDccu", where "ccu" is a real device address.

MDRELN is the cylinder offset for a shared DASD device.

MDSIZE is the size of the virtual device.

MDRDPASS is an eight-byte password used to determine eligibility for read-only sharing.

MDRWPASS is an eight-byte password used to determine eligibility for write sharing.

MRDEBLOK

There is one MRDEBLOK for each multiplexer device defined in the system. The definition is contained in the REALIO module by macros. Its format is as follows:

```
        0        2        4        6        8
        +--------+--------+--------+--------+
   0    |     MRDEVPNT    |MRDEVADD|M*1 |M*2|
        +--------+--------+--------+--------+
   8    |      MUSER      |      MIRA       |
        +--------+--------+--------+--------+
  10    |     MRDEVIO     |      MTASK      |
        +--------+--------+--------+--------+
  18    |    MRPNTVIR     |M*3|  MRDCSWAD   |
        +--------+--------+--------+--------+
  20    |MRDERRCT|M*4|M*5 |M*6|M*7 |M*8 |xxx|
        +--------+--------+--------+--------+
```

where:

MRDEVPNT is a pointer to the next real device block.

MRDEVADD is the device address of this real device.

M*1 - MRDESTAT is the real device status:

X'80' indicates prepare issued (2702 only)
X'40' indicates HIO issued (2702 only)
X'20' indicates sense issued
x'10' indicates not ready
X'08' indicates enabled (2702 only)
X'04' indicates ATS terminal (2741 only)
X'02' indicates device is dedicated
X'01' disable line

M*2 - MRDEVTYP contains the real device type number.

MUSER contains the UTABLE address of the user owning this device.

MIRA is the interruption return address for this device.

MRDEVIO contains a pointer to closed files for this device (for spooling operations only).

MTASK contains a pointer to open MRIBUFF blocks for this device (for spooling operations only).

MRPNTVIR contains a pointer to the virtual device equivalent to this device (for nonspooling operations only).

M*3 - MRDESENS contains the sense byte information

(2702).

MRDCSWAD contains a pointer to the saved CSW information (2702 only).

MRDERRCT contains the count of errors on this device.

M*4 - MRRETRY is the retry counter for attempted error recovery.

M*5 - MRFTR contains device or line features, such as the SAD number 0,1,2,3, or 4.

M*6 - MRIFLAG is the flag for MRIOEXEC:

MRIDRAIN X'08' drain spooling operations
MRISPACE X'04' force printer to single space
TERMINAT X'01' terminate spooled I/O when interrupt comes in
UNSPOOL X'02' punch available for unspooled I/O, that is, accounting cards.

M*7 - MRWRTFLG is used by CONSINT to identify the terminal.

M*8 - MRDEBRCT is reserved for future use.

MRIBUFF

The following buffers and their descriptions apply to those blocks used in the "unspooling" operations associated with MRIOEXEC and the real hardware.

MRIBUFF is the buffer for spooled packed data when being handled for the real equipment. It is chained from MTASK in the multiplexer real device block (MRDEBLOK). Its format is:

```
         0        2        4        6        8
         +--------+--------+--------+--------+
         |                                   |
    0    |            IOTASK                 |
         |                                   |
         +--------+--------+--------+--------+
         |                                   |
   20    |             MRICAW1               |
         |                                   |
         +--------+--------+--------+--------+
   48    |   BB   |   CC   |   HH   |  R |xxx|
         +--------+--------+--------+--------+
   50    |MRICOUNT|                          |
         +--------+                          |
         |                                   |
         |               DATAD               |
         |                                   |
         +--------+--------+--------+--------+
         |                                   |
   390   |              MRICAW2              |
         |                                   |
         +--------+--------+--------+--------+
         |                                   |
   3E8   |               DATAP               |
         |                                   |
         +--------+--------+--------+--------+
   470   |     REGSAVE     |      BADDR      |
         +--------+--------+--------+--------+
         |                                   |
   478   |                MRIFILEC           |
         |                                   |
         +--------+--------+--------+--------+
```

where:

IOTASK is the IOTASK block associated with bringing this buffer to and from the disk; four doublewords only.

MRICAW1 are the CCW's required to bring the buffer off the disk or write it to the disk; five CCWs: SEEK, SEARCH, TIC *-8, RD or WRT, NOP.

MRINEXT is the pointer to the next buffer on the disk;

BBCCHHRx, x is device table code (index).

MRICOUNT is the pointer within the buffer to the next byte to be processed.

DATAD is the packed data read or written on disk.

MRICAW2 are the unit record CCW's required for this buffer.

DATAP is the output buffer for the PACK routine (card reader) or the output buffer for the UNPACK routine (printer and punch).

REGSAVE is a temporary register save area.

BADDR is a pointer to the unpacked input-output buffer for unit record data.

MRIFILEC is a three-doubleword pointer for DASD record address. It contains data to build the SFBLOK when the file is completed (reader only).

The following is a description of the buffer for the unit record operations chained from BADDR of the MRIBUFF block (preceding):

```
        0        2        4        6        8
        +--------+--------+--------+--------+
     0  |CUR. CCW|CUR. DAT|  CAW   |xxxxxxxx|
        +--------+--------+--------+--------+
        |                                   |
     8  |               DATA                |
        |                                   |
        +--------+--------+--------+--------+
```

MVDEBLOK

There is an MVDEBLOK for each virtual multiplexer device attached to a user's UTABLE (from VMXSTART); its format is as follows:

```
      0        2        4        6        8
      +--------+--------+--------+--------+
 0    |     MVDEVPNT    |MVDEVADD|M*1  |M*2|
      +--------+--------+--------+--------+
 8    |    MVPNTREL     |      MVIOB      |
      +--------+--------+--------+--------+
10    |               MVCSW               |
      +--------+--------+--------+--------+
18    |     MVDEVIO     |M*3  |M*4|M*5|M*6 |
      +--------+--------+--------+--------+
20    |             MVIXUSER              |
      +--------+--------+--------+--------+
```

where:

MVDEVPNT is a pointer to the next virtual device on the virtual multiplexer channel.

MVDEVADD is the virtual device address.

M*1 - MVDESTAT is the virtual device status; the bit definition is the same as the bit definition of byte 4 of a CSW, for example, CE=X'08', BUSY=X'10'.

M*2 - MVDEVTYP is the virtual device type number.

MVPNTREL is the pointer to the real terminal (MRDEBLOK).

MVIOB is the current buffer address for this device; MVIBUFF for unit record; or terminal I/O buffer.

MVCSW is the virtual CSW for this subchannel.

MVDEVIO is the pointer to closed files for this virtual device (spooling operations only); for terminals (virtual 1052), address of current CCW.

M*3 - MVSENSE is the sense information for the device.

M*4 - MVIFLAG are miscellaneous status bits:

MVIFCCW	X'01'	current CCW is first in chain
MVIFCLOS	X'02'	file closed by CONSOL function
MVIFRMT	X'04'	spooled output to go to MVPNTREL
MVIFSAV	X'08'	keep virtual card reader files after use
MVIXFER	X'10'	punch file to be made a card reader file for MVIXUSER

MVIEXIT	X'20'	MVIOEXEC has done EXIT, go to DISPATCH
MVICONT	X'40'	continuous card spooling
	X'80'	reserved for future use

M*5 - MVIOKEY is the virtual CAW storage protection key.

M*6 - MVIOBRK is a flag to indicate (X'FF') that the attention key was hit during virtual console I/O.

MVIXUSER - for punch or printer; contains userid to transfer output if MIVXFER list in MVIFLAG is on; for terminals (virtual 1052), contains current CCW being processed.

MVIBUFF

This section is a description of the various buffers used by the spooling mechanism of the Control Program.

MVIBUFF is a buffer for packed spooled data. It is chained from MVIOB in the multiplexer virtual device blocks and has the following format:

```
        0        2        4        6        8
        +--------+--------+--------+--------+
        |                                   |
     0  |             IOTASK                |
        |                                   |
        |                                   |
        +--------+--------+--------+--------+
    20  |MRICAW1                            |
        |                                   |
        |                                   |
        |                                   |
        |                                   |
        +--------+--------+--------+--------+
    48  |        MVINEXT                    |
        +--------+--------+--------+--------+
    50  |MVICOUNT|                          |
        +--------+                          |
        |                                   |
        |               DATAD               |
        |                                   |
        +--------+--------+--------+--------+
    390 |             MVICCW                |
        +--------+--------+--------+--------+
    398 |       Temporary Save Area         |
        +--------+--------+--------+--------+
        |                                   |
    3A0 |             DATAPAC               |
        |                                   |
        +--------+--------+--------+--------+
        |                                   |
    3C0 |              DATAP                |
        |                                   |
        +--------+--------+--------+--------+
    448 |MVIRECS |                          |
        +--------+                          |
        |             MVIFILEC              |
        |                                   |
        +--------+--------+--------+--------+
```

where:

IOTASK is the task control block for reading or writing the disk buffers; four doublewordsords only.

MRICAW1 are the CCW's required to write or read the buffer to secondary storage; five CCW's: SEEK, SEARCH, TIC *-8, RD or WRT, NOP.

MVINEXT is the pointer to the next record; BBCCHHRx, where x is device code.

MVICOUNT is the byte address within the following data area, DATAD, of the next byte.

DATAD is the buffer of packed data (830 bytes long).

MVICCW is the user's current CCW.

DATAPAC is the output buffer for the PACK routine (see "PACK" in Section 5).

DATAP is the input buffer for the PACK routine or the output buffer for the UNPACK routine, depending on the spooling function being performed.

MVIRECS (MVIFILEC):

> When the file is open, it is the number of records in this file (two bytes).
>
> When the file is closed, it is a three-doubleword pointer for DASD record address. It is used to build an SFBLOK when the file is completed.

PAGTABLE

There is one PAGTABLE for each user; its format is as follows:

```
0           12   15
+-----------+-+----+
|   SWPTBL  PNT    |
+-----------+-+----+
| Page Add. | |xxxx|
+-----------+-+----+
|                  |
|                  |
|                  |  up to
|                  | 256 Entries
|                  | 512 Bytes maximum
|                  |
|                  |
|                  |
+-----------+-+----+
|           | |    |
+-----------+-+----+
```

where:

SWPTBL PNT is a pointer to the SWPTABLE associated with this PAGTABLE, one fullword in size. The remainder is made up of halfword entries. Each entry describes the status and main storage address of a virtual memory page, as follows:

Bits 0 through 11 are the address of a page in real memory (if resident).

Bits 12 through 15 are a control field:

Bit 12 indicates status of the page:

0 indicates core resident.
1 indicates not in core.

Bits 13-15 are reserved for future use (they must be zero for the 360/67).

RCCWLIST

There is one RCCWLIST for each user CCW list; its format is as follows:

```
        0        2        4        6        8
        +--------+--------+--------+--------+
     0  |      VLIST      |      TADDR      |
        +--------+--------+--------+--------+
     8  | VCNT   | RCNT   | IDENT  | SCNT   |
        +--------+--------+--------+--------+
    10  |R*1| RADDR       |R*2 |R*3|RBYTE   |
        +--------+--------+--------+--------+
```

where:

VLIST is the location of CCW's in user's program.

TADDR is the real address of the next CCW list (0 if none).

VCNT is the number of user's CCW's in this list.

RCNT is the number of CCW's required to represent user's list.

IDENT is the halfword marker (used in UNTRANS); X'FFFF'.

SCNT is the number of doublewords reserved for control data.

R*1 - RCOMND is the actual CCW op-code for the channel.

RADDR is the real (translated) address for the data transfer or argument.

R*2 - RFLAG is the real flag field for the channel CCW's.

R*3 - RCNTL is the control field used by CCWTRAN and UNTRANS to identify certain types of CCW's:

RCXIS	X'80'	check for ISAM read
RCSUDO	X'40'	pseudo 2311 or 2314
RCUTIC	X'20'	untranslated TIC
RCIO	X'10'	I/O CCW
RCGEN	X'08'	CP-generated CCW
RCDATA	X'04'	CP-generated CD
RC02	X'02'	reserved for future use
RC01	X'01'	reserved for future use

RBYTE is the real CCW data count.

RCHBLOK

There is one RCHBLOK for each real channel; its format is as follows:

```
        0        2        4        6        8
        +--------+--------+--------+--------+
     0  |     RCHANPNT    |     RCULIST     |
        +--------+--------+--------+--------+
     8  |     TASKLIST    |R*1|R*2 |RCUCOUNT|
        +--------+--------+--------+--------+
    10  |RCHANADD| TASKCNT|      TASKLAST   |
        +--------+--------+--------+--------+
        |RCHCOND |R*3|R*4 |R*5|R*6 |RESERVED|
        +--------+--------+--------+--------+
```

where:

RCHANPNT is the pointer to the next channel.

RCULIST is the pointer to connected control units.

TASKLIST is the pointer to pending tasks.

RCUACT (R*1) is the active control unit mask.

RCHSTAT (R*2) are channel status bits:
X'80' indicates channel busy.
X'40' indicates rescan required in CHFREE.

RCUCOUNT is the count of attached control units.

RCHANADD is the real channel address.

TASKCNT is the count of pending tasks.

TASKLAST is the pointer to last task on this channel.

RCHCOND is channel status after a channel error (**).

R*3	RCHDATCK	count of channel data checks
R*4	RCHCONCK	count of channel control checks
R*5	RCHIFCC	count of interface control checks
R*6	RCHANCC	count of channel chaining checks

(**) channel error is defined as any error indicated by R*3, R*4, R*5, or R*6.

RCUBLOK

There is one RCUBLOK for each real control unit; its format is as follows:

```
        0        2        4        6        8
        +--------+--------+--------+--------+
     0  |     RDEVLIST    |      RCUPNT     |
        +--------+--------+--------+--------+
     8  |     RACTCHAN    |R*1|xxxxxxxxxxxxx|
        +--------+--------+--------+--------+
    10  | RCUADD | RCUSTAT|RTAILCNT|RDECOUNT|
        +--------+--------+--------+--------+
    18  |     RCUTAIL1    |     RCUTAIL2    |
        +--------+--------+--------+--------+
```

where:

RDEVLIST is the pointer to connected devices.

RCUPNT is the pointer to next control unit.

RACTCHAN is the pointer to active channel; zero value initially; filled in from RCUTAIL1 after SIO.

R*1 - RCUPATH is the path for this control unit.

RCUADD is the real control unit address.

RCUSTAT is the real control unit status (not currently used).

RTAILCNT is the tail count for this control unit (not currently used).

RDECOUNT is the count of devices on this unit.

RCUTAIL1 is the pointer to channel for tail 1.

RCUTAIL2 is the pointer to channel for tail 2 (not currently used).

RDCONPKG

There is one RDCONPKG for each CCWPKG that requires control to be returned upon completion of the associated I/O operation. Its format is:

```
     0        2        4        6        8
    0+--------+--------+--------+--------+
     |  NEXTCPRQ       |  JSRETADD       |
    8+--------+--------+--------+--------+
     |              JSREGS               |
    ___                                 ____

    ___                                 ____
     |                                   |
     +--------+--------+--------+--------+
     |  JSPARE3        |  JSPARE4        |
   48+--------+--------+--------+--------+
```

where:

NEXTCPRQ is a pointer (always zero until queued by CPSTACK).

JSREADD is the return address (becomes CPEXADD in CPEXBLOK).

JSREGS are registers for return.

JSPARE3 is a spare.

JSPARE4 is a spare.

RDEVBLOK

There is one RDEVBLOK for each real device; its format is as follows:

```
        0        2        4        6        8
        +--------+--------+--------+--------+
      0 |     RDEVPNT     |     RDEVCU      |
        +--------+--------+--------+--------+
      8 |RDEVADD |R*1 |R*2|    RDEVTASK     |
        +--------+--------+--------+--------+
     10 |          RVOLSER         |RDEVCODE|
        +--------+--------+--------+--------+
     18 |    RDEVALLN     |RDEVERCT|RDEVSTAT|
        +--------+--------+--------+--------+
     20 |    RDEVUSER     |RATTVADD|R*3 |R*4|
        +--------+--------+--------+--------+
        |         RDEVSEN          |C*0 |C*2|
        +--------------------------+--------+
        |C*3|C*4|C*7|   RDEVTMON            |
        +-------+---------+--------+--------+
```

where:

RDEVPNT is a pointer to the next device on the chain.

RDEVCU is a pointer to the real control unit.

RDEVADD is the real device address (control unit and device portions only).

R*1 - RDEVTYPE is the device type code.

R*2 - RDECUPTH is the control unit path for this device.

RDEVTASK is a pointer to the attached task block (if active).

RVOLSER is the six-character EBCDIC volume label (if DASD volume and attached to the system).

RDEVCODE is the halfword identification number (index into RDEVTABL).

RDEVALLN is the pointer to the allocation table (if CP-owned).

RDEVERCT is the error count for this device.

RDEVSTAT is the real device status:

RDEVOWND X'80' indicates CP-owned volume (DASD only).
RDEVATTD X'40' indicates dedicated (nonshared) device.

RDEVDED X'20' indicates channel, control unit, and device block dynamically created by DEDICATE.
RDEVSEEK X'08' indicates a seek is in progress.
RDEVPOSD X'04' indicates 2311,2314 comb positioned for next read/write operation.
RDEVSYS X'02' device attached to system.

RDEVUSER is the UTABLE pointer for the current user (for dedicated devices).

RATTVADD is the current user's virtual address (for dedicated devices).

R*3 - RDEVFTR Real device features. Used to describe dedicated communication lines SAD value.

R*4 - RDEVSLEN device sense byte count

RDEVSEN contains up to six sense bytes for device

C*0 - command reject counter

C*2 - bus out parity error counter

C*3 - equipment check error counter

C*4 - data check counter

C*7 - seek check (sense bit 7, byte 0) counter

RDEVTMON is the attached time for a dedicated device (MMDDYY HHMM).

RECBUF

One RECBUF block of the following format is created for each cylinder. The start of the RECBUF block's chain is RECSTART. Its format is:

```
        0        2        4
        +--------+--------+
    0   | Pointer to next |
        +--------+--------+
    4   |R*1 |R*2|CNUM|DCD|
        +--------+--------+
        |                 |
    8   |      DATA       |  - 202 Bytes
        |                 |
        +--------+--------+
```

where:

The first word is the pointer to the next RECBUF block.

(R*1) is the number of records in use on this cylinder.

(R*2) is the maximum number of records available in this cylinder.

CNUM is the cylinder number of this cylinder.

DCD is the real device code for the device for this cylinder.

The remaining bytes are:

For a 2314, two bytes for each pair of even-odd tracks. There are 15 records per pair of tracks, and each bit (0-14) indicates whether the corresponding record (1-15) is available. Bit 15 is always set to 1.

For a 2311, two bytes for each track. There are four records per track. Bits 4 through 15 are set to 1. A 1 indicates that the corresponding record is in use.

SAVEAREA

The active SAVEAREA format is:

```
0                4                8
+----------------+----------------+
| RETURN ADDRESS | CALLERS R12    |
+----------------+----------------+
| CALLERS R13    |                |
+----------------+                |
|  21 WORD REGISTER SAVEAREA      |
|      and WORKAREA               |
+---------------------------------+
```

where:

RETURN ADDRESS is the instruction address immediately following the SVC 8 call which obtained the current save area.

CALLERS R12 is the base register of the calling routine.

CALLERS R13 is the address of the active save area.

21 WORD REGISTER SAVEAREA and WORKAREA is normally used by the ENTER macro to save the caller's registers. Up to 16 registers can be saved, although only registers 0-11 are significant. Words not used for register saving can be used as a scratch area by the called program.

The inactive (available) SAVEAREA format is:

```
0                4                8
+----------------+----------------+
| NEXTSAVE       |                |
+----------------+----------------+
|                                 |
|                                 |
|                                 |
+---------------------------------+
```

where:

NEXTSAVE is a pointer to the next 24-word save area in the chain of available save areas. The pointer is updated in the last save area on the chain when a save area is released by SVC 12 or SVC 16.

SEGTABLE

CP-67 contains one SEGTABLE for each user; its format is as follows:

```
0        1                        4
+--------+------------------------+
|PAGE CNT|   PAGE TABLE ADDRESS   |
+--------+------------------------+
|                                 |
|                                 |  16 Entries
|                                 |  64 Bytes
|                                 |
|                                 |
+--------+------------------------+
|        |                        |
+--------+------------------------+
```

Each four-byte entry defines a page table, as follows:

Byte 1 - Number of page table entries (less 1).

Bytes 2-4 - Address of page table origin.

SFBLOK

SFBLOK is a control block for a closed spool file. The format is as follows:

```
0         2         4
+---------+---------+
| Pointer to next   |
+---------+---------+
|  BB     |   CC    |
+---------+---------+
|  HH     | R |Code|
+---------+---------+
|   MRDEBLOK        |
+---------+---------+
|    Userid         |
|                   |
|                   |
+---------+---------+
```

When this file is being used by MRIOEXEC, the pointer is removed from the chain and hooked up to MRDEVIO in the multiplexer real device block (MRDEBLOK).

MRDEBLOK is filled in if the spooled output is directed to a particular device.

The high-order byte of this field is also used for a repeat of the output in MRIOEXEC. An x'80' means output is directed to the MRDEBLOK address in the remaining three bytes. An x'4x' means repeat the output up to x times.

SWPTABLE

The SWPTABLE contains an eight-byte entry for each entry in a user's PAGTABLE. It is generated at LOGON time, its length depending on the size of a user's virtual memory. It is in the following format:

```
0         1         2         3         4
+---------+---------+---------+---------+
|S*1      | VPAGNO  |  KEY1   |  KEY2   |
+---------+---------+---------+---------+
|RDEVCODE|   CYL   |  HEAD   | RECORD  |
+---------+---------+---------+---------+
```

where:

S*1 has the following meaning:

X'80': Transit bit, page in transit (in)
X'40': Recompute bit, DASD address is source of page, get new DASD address if write is required
X'20': Transit bit, page in transit (out)
X'10': Shared bit, page is shared and protected
X'08': first half page was used since last SSK (if in core)
X'04': first half page was modified since last SSK (if in core)
X'02': second half page was used since last SSK (if in core)
X'01': second half page was modified since last SSK (if in core)

VPAGNO is the virtual page number of the user using the page.

KEY1 and KEY2 are the virtual keys for the bottom and top halves of this page, respectively.

RDEVCODE is the real device code of the device containing this page.

CYL, HEAD, and RECORD are the physical location of the nonresident page on the device indicated by RDEVCODE.

TREXT

This control block is built as a UTABLE extension when the user invokes tracing functions.

```
0        2        4        6        8
+--------+--------+--------+--------+
| TRSVCI | TRBRI  | TRSTI  | T*1|T*2|
+--------+--------+--------+--------+
|T*3 |T*4| UNUSED |     TRSVCIA     |
+--------+--------+--------+--------+
|     TRBRIA      |     TRSTAD      |
+--------+--------+--------+--------+
|     TRSTSV      |     TREXINS     |
+--------+--------+--------+--------+
|              UNUSED               |
+--------+--------+--------+--------+
|              TRSVLCO              |
+--------+--------+--------+--------+
|               TRPWK               |
+--------+--------+--------+--------+
|                                   |
|                                   |
/               TRLIN               /
/                                   /
|                                   |
|                                   |
+--------+--------+--------+--------+
```

TRSVCI - saved 2 bytes of next instruction.

TRBRI - saved 2 bytes of branch-to instruction.

TRSTI - saved 2 bytes of address stop instruction

T*1-TRCNSL - console tracing options.

T*2-TRPRT - printer tracing options.

T*3-TRINTF - interrupt type flag.

T*4-BRSW - processing control switch.

TRSVCIA - next instruction address.

TRBRIA - branch-to instruction address

TRSTAD - address stop location.

TRSTSV - address of executed NSI.

TREXINS - executed NSI contents.

TRSVLCO - saved location zero 8 bytes.

TRPWK - pack data work area.

TRLIN - output data buffer.

UFDENT

The following is a description of an entry in the user file directory (U.DIRECT) which contains information about the user and his access privileges to the system:

```
        0       2       4       6       8
        +-------+-------+-------+-------+
    0   |            UFDID              |
        +-------+-------+-------+-------+
    8   |           UFDPASS             |
        +-------+-------+-------+-------+
   10   |           UFDACCT             |
        +-------+-------+-------+-------+
   18   |           UFDMDEF             |
        +-------+-------+-------+-------+
   20   |U*1| U*2|xxxxxxxx|
        +-------+-------+
```

where:

UFDID is the eight-character user identification.

UFDPASS is the eight-character user password.

UFDACCT is the user accounting information.

UFDMDEF is the eight-character file name of the user's machine description file.

U*1 - UFDPRIV is the user's privilege class code.

U*2 - UFDPRIOR is the user's priority code (1-9).

UTABLE

There is one UTABLE block for each user in the system. It is the primary control block from which all user blocks are strung. It completely reflects, with the virtual I/O blocks, the status of the virtual machine. Its format is:

```
        0        2        4        6        8
        +--------+--------+--------+--------+
        |                                   |
   0    |               VGPR's              |
        |                                   |
        +--------+--------+--------+--------+
        |                                   |
  40    |               VFPR's              |
        |                                   |
        +--------+--------+--------+--------+
  60    |                VPSW               |
        +--------+--------+--------+--------+
  68    |    SEGTABLE     |    VMACHSIZ     |
        +--------+--------+--------+--------+
  70    |    VCHSTART     |VCHCOUNT| PENDING|
        +--------+--------+--------+--------+
  78    |ULOCKS  |VMSTATUS|    TIMEUSED     |
        +--------+--------+--------+--------+
  80    |    NEXTUSER     |     VTIMER      |
        +--------+--------+--------+--------+
  88    |              USERID               |
        +--------+--------+--------+--------+
  90    |      DVTOT      |     USYSTAB     |
        +--------+--------+--------+--------+
  98    |    VMXSTART     |    VMXPOINT     |
        +--------+--------+--------+--------+
  A0    |ULOCKL  |U*1 |U*2|     UTREXT      |
        +--------+--------+--------+--------+
  A8    |     CIOREQ      |NCIOREQ |DNMPAGE |
        +--------+--------+--------+--------+
  B0    |VMXCOUNT|SEGTBDSP|     ADEXTAB     |
        +--------+--------+--------+--------+
  B8    |     TIMEON               |U*3| U*4|
        +--------+--------+--------+--------+
  C0    |              ACCTNG               |
        +--------+--------+--------+--------+
  C8    |     TIMINQ      |NUMPAGES|PRIORIT |
        +--------+--------+--------+--------+
  D0    |    VTOTTIME     |U*5 |U*6 |UPIOCNT|
        +--------+--------+--------+--------+
  D8    |UVIOCNT |         UCPCOMND         |
        +--------+--------+--------+--------+
  F0    |    TIMSTAMP     |    NEXTRTMR     |
        +--------+--------+--------+--------+
  F8    |      NXTQ       |      PRVQ       |
        +--------+--------+--------+--------+
 100    |     VMUSER1     |     VMUSER2     |
        +--------+--------+--------+--------+
 108    |     VMUSER3     |     VMUSER4     |
    +--------+--------+--------+--------+
116|       USERINST           |  TRSW  |
    +--------+--------+--------+--------+
124|     VMSSIO      |     VMPNCH      |
    +--------+--------+--------+--------+
132|     VMLINS      |     VMCRDS      |
    +--------+--------+--------+--------+
140|     VMPGRD      |    RESERVED     |
    +--------+--------+--------+--------+
148|              RESERVED             |
    +--------+--------+--------+--------+
156|              RESERVED             |
    +--------+--------+--------+--------+
```

where:

VGPR's are the user's 16 general purpose registers saved on an interrupt.

VFPR's are the user's four double-precision floating point registers.

VPSW is the user's virtual PSW.

SEGTABLE is a pointer to the user's segment table.

VMACHSIZ is the size of the virtual machine (last valid address +1).

VCHSTART is a pointer to the first selector channel block.

VCHCOUNT is the number of virtual selector channels attached.

PENDING contains a bit for each channel which has a pending interrupt.

ULOCKS is reserved for future use.

VMSTATUS is a halfword containing bits reflecting the state of a user machine:

Byte 0

PAGEWAIT	X'80'	user waiting for a page or pages
IOWAIT	X'40'	user SIO being analyzed
CFWAIT	X'20'	user in console function mode
SYSOPBIT	X'10'	user is system operator
UARPQ	X'08'	reserved for future use
VIRCOMSW	X'04'	virtual console function in execution
INLOGOFF	X'02'	user in logoff process
INLOGON	X'01'	user in login process

Byte 1

X'80'	indicates that the current runuser has not been charged for virtual time
X'40'	reserved

X'08' user is runnable
X'04' user is in a Q
X'02' user is running shared system

TIMEUSED is the total time used since logon (problem state plus CP overhead).

NEXTUSER is the pointer to the next user's UTABLE.

VTIMER is the user's virtual timer.

USERID is the eight-character user identification.

DVTOT is the VTOTTIME value on entry to a queue or the virtual time used during the last time in a queue.

USYSTAB is the pointer to the table for the system which this user is sharing.

VMXSTART is the pointer to the first virtual device block on the virtual multiplexer channel.

VMXPOINT is reserved for future use.

ULOCKL is reserved for future use.

U*1 - UOPTDEF is the user options from the DIRECTORY:

RTIMR X'80' - real timer
ISAM X'40' - self-modifying DASD CCW checking
V67 X'20' - user can operate in virtual extended PSW mode

U*2 - PRCLASS is the user's privilege class and priority level:

SYSCTLOP X'80' indicates system operator.
SYSADMIN X'40' indicates system administrator.
SUBSYSOP X'20' indicates subordinate system operator.
SYSUSER X'10' indicates system user.

The low-order four bits contain the user's priority level (1-9).

UTREXT built when user invokes tracing functions.

CIOREQ is the pointer to pending console operation requests.

NCIOREQ is the number of pending console operations.

DNMPAGE is paging activity value for this user.

VMXCOUNT is the count of multiplexer devices for this user.

SEGTBDSP is the displacement of SEGTABLE from start of free storage block.

ADEXTAB is the address of the UTABLE extension, used for a virtual 67.

TIMEON is the user time on.

U*3 - TIMERMOD is the virtual timer mode switch:

```
DISCNBIT x'80'  user terminal disccnnected
PRIDISP  x'40'  request for priority dispatch
RUNCP    x'04'  virtual machine running with
                   console function read active
MSGBIT   X'20' user set MSGOFF
WNGBIT   X'10' ignore warnings
MULTCH   X'08' more than one virtual channel
may exist with same channel address
```

U*4 - PAGWCNT is the count of user outstanding page requests.

ACCTNG is the user accounting information.

TIMINQ is used by DISPATCH for scheduling.

NUMPAGES is number of pages the user has in core.

PRIORIT is priority to reenter the queue.

VTOTTIME is the total problem state time used by user since login.

U*5 - WORKSET is not presently used.

U*6 - CNTRLMOD is the status of the virtual 360/67:

EXTCM X'80' indicates the virtual machine is in extended control mode.

INVCR0 X'20' indicates that all the tables describing the third-level memory have to be rebuilt.

INVSHADT X'10' indicates that the shadow segment and page tables have to be rebuilt.

UPIOCNT is the number of page reads done for this user while in a queue. Reset to zero each time on entry to a queue.

UVIOCNT is the number of virtual SIOs issued by this user.

UCPCOMND is the last CP console function executed by the user.

| TIMSTAMP is time stamp at status change.

| NEXTRTMR next user with a real timer.

NXTQ next user in this runnable list.

PRVQ previous user in this eligible list.

VMUSER1-4 for installation use.

USERINST for saving privileged instructions.

TRSW trace switch.

VMSSIO number of selector channel SIO's.

VMPNCH number of spooled cards punched.

VMLINS number of spooled lines printed.

VMCRDS number of spooled cards read.

VMPGRD number of pages read.

VCHBLOK

There is one virtual channel block for each virtual channel on each user. Its format is as follows:

```
        0        2        4        6        8
        +--------+--------+--------+--------+
     0  |     VCHANPNT    |     VCULIST     |
        +--------+--------+--------+--------+
     8  |VCHANADD|VCUCOUNT| V*1 |xxx|V*2 |xxx|
        +--------+--------+--------+--------+
    10  |VCEUNIT |VNPNDCUI|xxxxxxxxxxxxxxxxx|
        +--------+--------+--------+--------+
    18  |               VCHCSW              |
        +--------+--------+--------+--------+
```

where:

VCHANPNT is the pointer to this user's next virtual channel.

VCULIST is the pointer to the connected control unit blocks.

VCHANADD is the virtual channel address.

VCUCOUNT is the count of virtual control units attached to this channel.

VCHSTAT (V*1) is the virtual channel status; bit definition for channel status is the same as the CSW, byte 4; for example, BUSY=X'10', CE=X'08'.

VCHFLAG (V*2) is reserved for future use.

VCEUNIT is the address of the unit for which the pending channel end, if any, occurred.

VNPNDCUI is the number of pending control unit interruptions.

VCHCSW is the virtual channel status word for channel end type interruptions.

VCUBLOK

There is one virtual control unit block for each virtual control unit; its format is as follows:

```
         0        2        4        6        8
         +--------+--------+--------+--------+
      0  |     VDEVLIST    |     VCUPNT      |
         +--------+--------+--------+--------+
      8  | VCUADD |VDECOUNT| VCUSTAT|xxxxxxxx|
         +--------+--------+--------+--------+
     10  |VCUEUNIT|VNPNDDEI|xxxxxxxxxxxxxxxxx|
         +--------+--------+--------+--------+
```

where:

VDEVLIST is the pointer to the virtual devices connected to this control unit.

VCUPNT is the pointer to the next virtual control unit in the chain from the virtual channel.

VCUADD is the virtual control unit address (no channel or device included).

VDECOUNT is the number of virtual devices attached.

VCUSTAT is the status of the virtual control unit; bit definition is the same as the CSW, byte 4; for example, BUSY=X'10'.

VCUEUNIT is the unit for which a control unit end condition, if any, is pending.

VNPNDDEI is the number of pending device interruptions.

VDEVBLOK

There is a virtual device block for each virtual device for each user in the system; its format is as follows:

```
       0        2        4        6        8
       +--------+--------+--------+--------+
    0  |     VDEVPNT     |VDEVADD |V*1 |V*2|
       +--------+--------+--------+--------+
    8  |     VPNTREAL    |VDEVREL |VDEVBND |
       +--------+--------+--------+--------+
   10  |              VDEVPOS              |
       +--------+--------+--------+--------+
   18  |              VDEVSNSE    |V*3 |V*4|
       +--------+--------+--------+--------+
```

where:

VDEVPNT is the pointer to the next device on the chain from the control unit.

VDEVADD is the virtual device address.

V*1 - VDEVSTAT is the virtual device status; bit definition is the same as the CSW, byte 4; for example, BUSY=X'10', DE=X'04'.

V*2 - VDEVTYPE is the virtual device type code.

VPNTREAL is the real device control block corresponding to this virtual device.

VDEVREL is the relocation factor within the real device for the start of this virtual device (for DASD only).

VDEVBND is the size of this virtual device (DASD only).

VDEVPOS is the current virtual arm position of this device (as BBCCHH).

VDEVSNSE is the virtual device sense information (filled when an error is detected on the virtual device to save the conditions for shared devices.)

V*3 - VDEVFLG contains miscellaneous device status bits:

TEMPDEV	X'01'	indicates a TDSK allocation
READONLY	X'02'	indicates read-only status
VSHARED	X'04'	reserved for future use
VDVENBL	X'08'	virtual 2702 line is enabled
VDVDIAL	X'10'	virtual 2702 line is in use

V*4 - VDEVSLEN is the sense byte count.

SECTION 5: SYSTEM MODULES

This section consists of descriptions of the modules contained in both CP-67 and the stand-alone utilities. They are arranged in alphabetical order according to module name. Listed below are the module names with a brief description of each. Table 3 gives the module entry points for each module.

ACCTON - for individual installations, additional processing and/or checking of users at LOGIN time

ACNTIME - computes and prints on user's terminal the total connect, virtual and actual CPU time

ACNTOFF - for individual installations, a replaceable module for accounting functions at LOGOUT time

CCWTRANS - prepares user CCW's for execution by real machine, and creates user CCW at end of operation

CFSCOM - contains the commands WNG, MSG, READY, LOGOUT, SLEEP, and DISCONNECT

CFSDBG - contains the commands DCP, DUMP, DMCP, DISPLAY, STCP, and STORE

CFSIPL - contains the commands IPL and IPLSAVE

CFSMAIN - calls user console functions and operator functions; entered during BREAK on user's terminal, or virtual machine idle state

CFSPRV - contains the commands ENABLE, DISABLE, LOCK, UNLOCK, SHUTDOWN, KILL, ACNT, DIRECT, and D_U_M_P

CFSQRY - contains the command QUERY

CFSSET - contains the command SET

CFSSPL - contains the commands TERM, CLOSE, XFER, SPACE, DRAIN, START, PURGE, SPOOL, and REPEAT

CFSTACH - contains the commands ATTACH, DETACH, and LINK

CHKCUACT - determines control unit status at channel end time based on last CCW executed by channel program and device on which it was executed

CHKPT - saves accounting records and in-core spool pointers on disk after an ABEND condition

CONSINT - initializes and identifies remote terminals and processes their interrupts

CONVRT - data conversion routines (BINHEX, FPCONV, BINDEC, etc.) for CP-user communication

CPCORE - currently contains constants for the IPL command

CPFILE - enables CP-67 to open, read, and close various internal working disk files

CPINIT - volume recognition and initialization of core (set new PSW's, compute real core size, etc.) for CP-67

CPSTACK - queues requests for CP service (CPRQUEST blocks)

CPSYM - resident loadmap of CP-67 modules and major entry points

DEDICATE - switches device from MRDEBLOK's to selector channel real device blocks for dedicated use

DIAGDSK - responds to diagnose call for a specialized I/O task on a 2311 or 2314

DIAL - removes user terminal from CP control and attaches it as a dedicated device to an existing virtual 2701, 2702, or 2703 line

DISPATCH - at completion of interrupt processing, searches for pending job (CPRQUEST queues, interrupted user, higher priority user), then either loads runnable user or enters idle condition, after totaling times in various states

DSKDUMP - for debugging; takes core dump of CP and performs a software re-IPL

EXTEND - calls PAGFREE to obtain pages for CP common buffer space, called Free Area

FREE - maintains and allocates units of system free storage, with minimum fragmentation

IOERROR - analyzes and records selected I/O errors and retries CP-generated I/O to selector channel devices

IOINT - receives control from the I/O new PSW, determines further action, and normally exits to IOTASK block's TASKIRA

IPL - virtual memory resident; simulates and interprets various IPL sequences for several devices

LINK - processes the CP console function "LINK" used by CMS for file sharing

LOGFILES - counts the number of spool file blocks awaiting processing and returns address of a message to caller

LOGIN - allocates free storage control blocks and machine resources required in setting up and logging in a new user

MRIOEXEC - entered from unit record interrupt; completes reading of cards, printing and punching of pending data in disk buffers

MVIOEXEC - handles all virtual I/O operations to user's multiplexer channel, including terminal and spooling functions

PACK - packs and unpacks blanks from the spooling data used by MRIOEXEC and MVIOEXEC

PAGEGET - allocates and deallocates DASD areas for paging

PAGTR - handles page sharing and page releasing

PAGTRANS - handles all functions which require a knowledge of the nature of the mapping device (virtual address translation, storage key setting, etc.)

PRIVLGED - simulates privileged instructions

PROGINT - entry from program interrupt new PSW; determines type of interrupt (CP-issued simulated instruction, or user-issued privileged instruction) and takes appropriate action

PSA - handles SVC, external, and machine check interrupts

QUEVIO - queues selector channel I/O requests, checks channel availability, positions DASD access arms, and initiates I/O operations

RDCONS - creates a CCW "package", according to terminal type, that can be stacked as a read request for that terminal

RDSCAN - determines whether a virtual DASD device is currently attached to the virtual machine of an active user (that is, a "link" exists)

RECFREE - handles spooling requests for available disk records in much the same way as free storage handles main memory

RESINT - performs virtual systems reset when explicitly asked or when implied by an IPL request

SAVECP - writes core-image of CP-67 onto system residence volume at end of card or tape load of CP-67 into core; procedure is reversed at IPL time (read in core image)

SCANUNIT - for a real or virtual device address, scans appropriate list and sets up pointers to various level blocks

SCHEDULE - maintains real timer and runnable list chains at logon and logoff and clock maintenance at 60 second intervals.

SCREDAT - contains 10 bytes of EBCDIC for system identification

STCONSIO - starts an I/O request to a console or stacks it if there are outstanding requests (entry via PRIMSG gives priority in the stack)

TMPSPACE - dynamically allocates DASD cylinders from devices of specified type

TRACER - performs analysis and output formatting of user specified tracing functions

UNSTIO - unstacks and reflects virtual I/O interrupts from both selector and multiplexer devices

UNTRANS - computes from hardware CSW the virtual CSW to be reflected to the user

USERLKUP - finds the entry in the U.DIRECT file for a specified userid

USEROFF - functions associated with logging off a user from the system (initiate logout sequence, delete virtual machine from system, log user off, detach nonshared I/O device)

VIOEXEC - intercepts virtual I/O commands; itself handles selector channel requests and passes multiplexer requests on to MVIOEXEC

VSERSCH - searches RDEVBLOK's for a given volume serial number

WRTCONS - allows a remote terminal to be used for output as though it were an operator's 1052 console

Table 3. System Modules with Entry Points

Module Name	Entry Point(s)
ACCTON	ACCTON
ACNTIME	ACNTIME
ACNTOFF	ACNTOFF, DEVOFF
CCWTRANS	CCWTRANS, VSMCPIR, CP6IRA
CFSCOM	WNG, MSG, READY, LOGOUT, SLEEP, DISCONN
CFSDBG	DCP, DUMP, DMCP, DISPLAY, STCP, STORE FREEPST, FRETPST
CFSIPL	CFSIPL, IPLSAVE
CFSMAIN	BREAK, BRKRD, BRKWR, COMENTRY
CFSPRV	ENABLE, DISABLE, LOCKC, UNLOCK, SHUTDOWN, KILL, CFSACNT, CFSDIR, ABEND
CFSQRY	QUERY
CFSSET	SET
CFSSPL	TERM, CLOSE, XFER, SPACE, DRAIN, START, PURGE, SPOOL, REPEAT
CFSTACH	ATTACH, DETACH, CLINK
CHKCUACT	CHKCUACT
CHKPT	CHKPT
CONSINT	CONSINT, IDENTIFY, PREPLINE, RTN41WT, RTN52WT, RTN41ND, RTN52ND, OFFHANG, OFFENT, CPIENT
CONVRT	BINHEX, HEXBIN, DECBIN, BINDEC, FPCONV, DATETIME
CPCORE	CPCORE
CPFILE	CPFOPENR, CPFOPENW, CPFCLOSE, CPFREAD, READTASK, WRITTASK, CPFDLKUP, CPFDCLOS
CPINIT	CPINIT
CPSTACK	CPSTACK
CPSYM	CPSYM
DEDICATE	DEDICATE
DIAGDSK	DIAGDSK
DIAL	DIAL
DISPATCH	DISPATCH, DSPTCHA, DSPTCHB, DSPTCHC, DISDRQ, DISIO, DISACT
DSKDUMP	DSKDUMP
EXTEND	EXTEND
FREE	FREE, FRET, FRETR
IOERROR	IOERROR, VERROR, RECERROR, MCKERR, FINDLOG, FMTLOG, LOGRETN, FINDMC, FINDIO, FMTMLOG, FMTLOGM, FMTILOG, FMTLOGI
IOINT	IOINT, IOISTVDE, IOISTVCU
IPL	IPL
LINK	LINK
LOGFILES	LOGFILES
LOGIN	LOGON, OPMSG, AUTOLOGON
MRIOEXEC	MRIOEXEC, RPUNCH, PRIRA, CRIRA, PUIRA
MVIOEXEC	MVIOEXEC, MVICLPR, MVICLPN, MVICLCR, MVIPRINT
PACK	PACK, UNPACK
PAGEGET	PAGEGET, PAGERLE

PAGTR	PAGSHARE, PAGOUT, PAGFRET
PAGTRANS	PAGTRANS, PAGUNLOK, PAGFREE, CORUSER, DRMWAIT, CORTENT, WAITPAGE
PROGINT	PROGINT, REFLECT
PRIVLGED	PRIVLGED
PSA	SVCINT, SVCINIT, EXTINT, MCHEKINT, SVCDUM
QUEVIO	QUEVIO, QUERIO, CHFREE
RDCONS	RDCONS
RDSCAN	LINKSCAN, RDSCAN, DEVSCAN
RECFREE	RECFREE, RECFRET
RESINT	RESINT, RESIRA
SAVECP	SAVECP
SCANUNIT	RUNITSCAN, VUNITSCAN
SCHEDULE	SCHEDULE, SCLOCK
SCREDAT	SCREDAT
STCONSIO	PRIMSG, STCONSIO
TMPSPACE	TMPSPACE, TMPRET, T2311, TMPERTN
TRACER	TRACER, TRINT
UNSTIO	UNSTIO
UNTRANS	UNTRANS, FREECCW
USERLKUP	USERLKUP
USEROFF	USEROFF, ADSET, ADSETOUT, RELEASE, RUNRET
VIOEXEC	VIOEXEC, VIRA
VSERSCH	VSERSCH
WRTCONS	WRTCONS, PRIORITY, OPTIME

ACCTON

Module name: ACCTON

Entry point: ACCTON

Purpose: To provide individual installations with the ability to add additional processing and/or checking of users at LOGIN time.

Entry conditions: Called from LOGIN after all other functions are complete except for message to operator and writing of LOGMSG to user.

Exit conditions: Condition code 0 - continue.
Condition code not 0 - log off user.

ACNTIME

Module name: ACNTIME

Entry point: ACNTIME

Purpose: This module computes the total connect, virtual, and actual CPU time used by the user and prints a formatted message on the user's terminal.

Registers 0-15 are saved upon entry to this module.

Entry point: ACNTIME

Entry conditions: GPR11 pointing to user's UTABLE

Exit conditions: None

ACNTOFF

Module name: ACNTOFF

Entry points: ACNTOFF, DEVOFF

Purpose: To provide individual installations with a replaceable module for performing accounting functions at LOGOUT time.

Entry point: ACNTOFF - punch accounting card for USER.

Entry conditions: GPR 11 points to the UTABLE.

Registers 0-11 are saved upon entry.

Exit conditions: None

Entry point: DEVOFF - punch accounting card for a dedicated device.

Entry conditions: GPR 11 points to UTABLE; GPR 2 points to RDEVBLOK.

Registers 0-11 are saved upon entry.

Exit conditions: None

CCWTRANS

Module name: CCWTRANS

Entry points: CCWTRANS, VSMCPIR, CP6IRA

Purpose: The CCWTRANS module prepares the user program channel command words for execution by the real machine, and creates the user program's channel status word on termination of the operation.

Entry point: CCWTRANS - translate user's virtual CCW list into an equivalent real list.

Entry conditions: GPR 1 is 0, indicating no I/O is to be performed, or it points to the IOTASK block which will represent this task. GPR 6 points to the virtual device block (VDEVBLOK) on which the operation is to be performed. The TASKCAW entry in the task block points to the user's virtual CCW list.

Exit conditions: The TASKCAW points to the real CCW list. The TASKFLAG in the task block indicates whether any multitrack CCW's are present. Other registers are preserved.

Entry point: VSMCPIR - restart ISAM I/O operation.

Entry conditions: GPR9 points to the IOTASK block. GPR10 points to the channel status word.

Registers 0-15 are saved upon entry to VSMCPIR.

Exit conditions: None

Entry point: CP6IRA - restore user's virtual core to its original condition after executing certain OS ISAM CCW's.

Entry conditions: Same as VSMCPIR

Exit conditions: None

CFSCOM

Module name: CFSCOM

Entry points: WNG, MSG, READY, LOGOUT, SLEEP, DISCONN

Purpose: Each entry point corresponds to a console command and contains logic for that command.

Entry conditions: Register 9 is used for addressing this module, and register 12 for addressing a branch table located in CFSMAIN. See "Console Functions" in Section 2 for individual command processing.

Exit condition: Return to CFSMAIN via branch table after handling command.

CFSDBG

Module name: CFSDBG

Entry points: DCP, DISPLAY, DMCP, DUMP, STCP, STORE

Purpose: Each entry point corresponds to a console command.

DCP - displays real core storage on the operator's console.
DISPLAY - displays virtual core storage, etc. on the user's terminal.
DMCP - dumps real core to the printer.
DUMP - dumps virtual core, etc. to the user's virutal printer.
STCP - stores into real core from the operator's console.
STORE - stores into the user's virtual core, etc.

Entry conditions: Register 9 contains the address of the entry point in this module. It is immediately changed to point to the beginning of csect CFSDBG, in order to provide addressability to the entire module. Register 12 contains the base address of a branch table in CFSMAIN. This table is used to branch to various subroutines in CFSMAIN. Register 13 contains the address of an 18 double word savearea. Register 11 contains the address of the UTABLE for the user issuing the command.

Exit conditions: Return is made to CFSMAIN via the branch table after handling the command.

External routines used--

Called routines:
PAGTRANS, WRTCONS, MVIPRINT, BINDEC, BINHEX, DECBIN, HEXBIN, FPCONV, DISPATCH, FREE, FRET, FREEPST, FRETPST.

Routines in CFSMAIN which are branched or branch and linked to:
SCANFLD, BADCOM, BADARG, READI.

References to other locations in the CP nucleus:
RMACHSIZ, TREBCDIC.

CFSIPL

Module name: CFSIPL

Entry points: CFSIPL, IPLSAVE

Purpose: Each entry point corresponds to a console command and contains logic for that command.

Entry conditions: Register 9 is used for addressing this module, and register 12 for addressing a branch table located in CFSMAIN. See "Console Functions" in Section 2 for individual command processing.

Exit condition: Return to CFSMAIN via branch table after handling command.

Module name: CFSMAIN

| Entry points: BREAK, BRKRD, BRKWR, COMENTRY

Purpose: The CFSMAIN module calls the user console functions and the operator functions. It is entered when a BREAK occurs on the user's terminal or the virtual machine goes idle (detected in DISPATCH).

Entry point: BREAK - entered when a user activates the attention key.

Entry conditions: GPR 6 points to the terminal's MRDEBLOK. GPR 10 points to the CSW information from the interrupt.

Registers 0-15 are saved upon entry.

Exit conditions: None. CONSOL exits by making the user runnable and returning to DISPATCH, after a BEGIN or IPL command, or if ATTN key actuated while in console function mode. Also exits immediately after a virtual console function.

Entry point: COMENTRY - entered from PROGINT when a DIAGNOSE instruction specifying a virtual console function has been detected.

Entry conditions: GPR2 points to a buffer containing the command line. GPR3 contains the number of bytes in the input line. GPR11 points to the user UTABLE.

Registers 0-15 are saved upon entry.

Exit conditions: GPR2 contains an error code as follows:

0 - No errors
4 - INVALID CP REQUEST (message not printed by CP)
8 - BAD ARGUMENT (message not printed by CP)
x - Code dependent upon specific function (error message usually printed by CP-67)

CFSPRV

Module name: CFSPRV

Entry points: ENABLE, DISABLE, LOCKC, UNLOCK, SHUTDOWN, KILL, CFSACNT, CFSDIR, ABEND

Purpose: Each entry point corresponds to a console command and contains logic for that command.

Entry conditions: Register 9 is used for addressing this module, and register 12 for addressing a branch table located in CFSMAIN. See "Console Functions" in Section 2 for individual command processing.

Exit condition: Return to CFSMAIN via branch table after handling command.

CFSQRY

Module name: CFSQRY

Entry point: QUERY

Purpose: Each entry point corresponds to a console command and contains logic for that command.

Entry conditions: Register 9 is used for addressing this module, and register 12 for addressing a branch table located in CFSMAIN. See "Console Functions" in Section 2 for individual command processing.

Exit condition: Return to CFSMAIN via branch table after handling command.

CFSSET

Module name: CFSSET

Entry point: SET

Purpose: Each entry point corresponds to a console command and contains logic for that command.

Entry conditions: Register 9 is used for addressing this module, and register 12 for addressing a branch table located in CFSMAIN. See "Console Functions" in Section 2 for individual command processing.

Exit condition: Return to CFSMAIN via branch table after handling command.

Module name: CFSSPL

Entry points: TERM, CLOSE, XFER, SPACE, DRAIN, START, PURGE, SPOOL, REPEAT

Purpose: Each entry point corresponds to a console command and contains logic for that command.

Entry conditions: Register 9 is used for addressing this module, and register 12 for addressing a branch table located in CFSMAIN. See "Console Functions" in Section 2 for individual command processing.

Exit condition: Return to CFSMAIN via branch table after handling command.

CFSTACH

Module name: CFSTACH

Entry points: ATTACH, DETACH, CLINK

Purpose: Each entry point corresponds to a console command and contains logic for that command.

Entry conditions: Register 9 is used for addressing this module, and register 12 for addressing a branch table located in CFSMAIN. See "Console Functions" in Section 2 for individual command processing.

Exit condition: Return to CFSMAIN via branch table after handling command.

CHKCUACT

Module name: CHKCUACT

Entry point: CHKCUACT

Purpose: CHKCUACT will examine the last CCW executed by a channel program and decide whether, for the device type on which the sequence was executed, the control unit is freed at channel end time.

Registers 0-4 are saved upon entry to CHKCUACT.

Entry point: CHKCUACT (BALR)

Entry conditions: GPR 6 points to the virtual channel block for which the input-output operation was executed. GPR 8 points to the virtual device block for which the operation was executed.

Exit conditions: The condition code is set nonzero if the control unit remains busy after the channel end occurring on the indicated CCW operation code. It is set to zero if the control unit may be considered free.

CHKPT

Module name: CHKPT

Entry point: CHKPT

Purpose: To save user accounting information and in-core spool pointers on disk.

Entry point: CHKPT

Entry conditions: If low core location CPID (hex '1FC') contains "CP67" or "SHUT", records will be written to disk; otherwise no action is taken.

Exit conditions: If CPID does not contain "SHUT", CP-67 will be IPL'ed by software; otherwise CHKPT will enter the wait state.

CONSINT

Module name: CONSINT

Entry points: CONSINT, IDENTIFY, PREPLINE, RTN41WT, RTN52WT, RTN41ND, RTN52ND, OFFHAND, OFFENT, CPIENT

Purpose: This module initializes and identifies remote terminals and processes all interrupts from those terminals. (Terminals presently supported are 1050, 1052, 2741-1, and 2741-2.)

Entry point: CONSINT

Entry conditions: All 1052 console interrupts are serviced via this entry point. GPR 10 is the location of the CSW associated with the interrupt, and GPR 6 is the MRDEBLOK for the interrupting device.

Exit conditions: If the previous I/O terminated normally, another I/O is initiated.
If only a CE has been received, the DE is waited for.
If an irregular ending occurred, the "ready" interrupt or the "termination-of-sense" is waited for.
Control is returned to DISPATCH via IOINT.

Entry point: IDENTIFY

Entry conditions: The 2702 lines once they are enabled have their first interrupt enter at IDENTIFY. GPR 6 contains the address of the terminal's MRDEBLOK and GPR10 points to the relevant CSW.

Exit conditions: The terminal is identified and its MRDEVTYP stored or it is indicated to be an unknown type. The line is initialized with a "Prepare" command and is waiting for a "login" attention break.

Entry point: PREPLINE

Entry conditions: GPR 6 points to the MRDEBLOK of a terminal of known device type (MRDEVTYP). The terminal line is then initialized with a "Prepare" command.

Exit conditions: The line sits in a "prepared" state waiting for a "login" attention break.

Entry points: RTN41WT, RTN52WT, RTN41ND, RTN52ND

Entry conditions: GPR6 points to a terminal MRDEBLOK of known device type that has completed an I/O operation by HIO, ATTN, or carriage return.

Exit conditions: the next I/O operation is started, if any, or the line is put in "prepare" status. Control is returned to DISPATCH via IOINT.

Entry points: OFFHANG, OFFENT

Entry conditions: GPR6 points to a terminal MRDEBLOK. A message is written to the terminal and an interrupt return address (OFFENT) is set up.

Exit condition: Return is to the caller (CCWTRAN).

Module name: CONVRT

Entry points: BINHEX, HEXBIN, DECBIN, BINDEC, FPCONV, DATETIME

Purpose: CONVRT is a collection of data conversion routines to assist CP-67 in communicating with the user.

Entry point: BINHEX

Entry conditions: GPR 1 contains the number to be converted from binary to hexadecimal notation.

Exit conditions: GPR's 0 and 1 contain the converted number in hexadecimal notation with leading zeros not suppressed.

Entry point: HEXBIN

Entry conditions: GPR 1 contains a pointer to a string of eight or fewer characters in hexadecimal notation (EBCDIC) which are to be converted. The length of the string is in GPR 0.

Registers 0-5 are saved upon entry.

Exit conditions: The condition code is set nonzero if an illegal hexadecimal character is encountered in the string; otherwise, the condition code is zero. The converted number is returned right-justified in GPR 1.

Entry point: FPCONV

Entry conditions: GPR 2 contains a pointer to a doubleword which contains the floating point word to be converted to standard floating point notation (for example, .00000000000 E 00) and GPR 1 contains a pointer to an output buffer of at least 17 characters.

Registers 0-5 are saved upon entry.

Exit conditions: The routine will fill the buffer pointed to by GPR 1 with the number in standard floating point notation.

Entry point: BINDEC

Entry conditions: GPR 1 contains a binary number to be converted to the equivalent in decimal notation.

Exit conditions: BINDEC returns the low-order eight decimal digits in GPR's 0 and 1.

Entry point: DECBIN

Entry conditions: GPR 1 points to a field containing the EBCDIC form of a decimal number which is to be converted to binary equivalent. GPR 0 contains the length of this field (in bytes).

Exit conditions: The condition code is set nonzero if the specified string contains invalid decimal information, or the length exceeds 15; otherwise, the condition code is set to zero. GPR 1 is returned with the converted number.

Entry point: DATETIME

Entry conditions: GPR 1 points to a field into which the date will be entered (as mm/dd/yy). GPR 2 points to a field into which the time will be entered (as hh.mm.ss). The date and time data are obtained from their locations in lower memory. If either pointer is zero, that parameter will not be provided.

Exit conditions: The fields are filled in as specified.

CPCORE

Module name: CPCORE

Entry point: CPCORE

Purpose: Contains only constants, no executable code. Currently contains constants for the IPL command, (DASDIPL - disk address of IPL module, DASDIPLN - disk address of SYSTEM module, and CMSTABLE - table for CMS shared system).

CPFILE

Module name: CPFILE

Entry points: CPFOPENR, CPFOPENW, CPFCLOSE, CPFREAD, READTASK, WRITTASK, CPFDLKUP, CPFDCLOS

Purpose: CPFILE is the mechanism by which CP-67 reads the various internal working disk files required, for example, system and user file directories and machine description files. Various routine entries are provided to allow opening, reading, and closing various files.

Entry point: CPFOPENR - open a file for reading.

Entry conditions: GPR 3 points to an eight-character file name.

Registers 0-8 are saved upon entry.

Exit conditions: GPR 2 points to a Control Program File System (CPFS) block which will be used to control access to the file. (Note: this must be preserved for later use in calling for actual file input-output).

Entry point: CPFOPENW - open a file for writing.

Entry conditions: This routine is not implemented yet - its calling conditions will be identical to CPFOPENR.

Registers 0-8 are saved upon entry.

Exit conditions: None

Entry point: CPFREAD - read data from a previously opened file.

Entry conditions: GPR 0 contains the number of bytes to be read. GPR 2 points to the CPFS block which was provided when the file was opened (see CPFOPENR).

Registers 2-7 are saved upon entry.

Exit conditions: GPR 1 points to the desired data (which resides in a CPFS-owned buffer) or zero if an end-of-file condition was encountered.

Entry point: CPFCLOSE - close a previously opened file.

Entry conditions: GPR 2 points to the appropriate CPFS file descriptor block.

Registers 0-5 are saved upon entry.

Exit conditions: None

Entry point: READTASK

Entry conditions: GPR 1 points to a buffer at least CPRECSZ bytes long (currently 829 bytes). GPR 2 points to the real device block. GPR 3 points to the record to be read (as BBCCHHR).

Registers 0-15 are saved upon entry.

Exit conditions: None. If the operation was not completed successfully, IOERROR is called to attempt recovery.

Entry point: WRITTASK - perform a write operation to disk.

Entry conditions: Same as READTASK

Exit conditions: Same as READTASK

Entry point: CPFDLKUP - finds specified directory entry.

Entry conditions: GPR 3 points to an eight-character file name.

Registers 3-6 are saved upon entry.

Exit conditions: Condition code=0 for file found, and GPR 1 points to DIRECTORY CPFRECRD; GPR 2 points to DIRECTORY CPFDENT. Otherwise, condition code=1 for file not found, and GPR 2 points to first empty entry.

Entry point: CFDCLOS - closes open directory file.

Entry conditions: GPR 2 points to CPFFDBLK.

Registers 0-3 are saved upon entry.

Exit conditions: None

CPINIT

Module name: CPINIT

Entry point: CPINIT

Purpose: This is the CP-67 initialization module. Its function is to create the necessary control blocks such as CORTABLE and allocation tables tased upon the hardware configuration present. For a detailed description of the functions performed see the section "Control Program Initialization" in Section 2.

Entry point: CPINIT

Entry conditions: GPR 2 contains a pointer to the allocation table address of the system residence volume. GPR 6 contains the device address of the residence volume.

Exit conditions: Exits to DISPATCH.

CPSTACK

Module name: CPSTACK

Entry point: CPSTACK

Purpose: This routine queues requests for CP execution (CPREQUEST blocks deferred pending an event occurrence) on the request stack (CPRQFST) defined in DISPATCH.

Registers 0-3 are saved upon entry to CPSTACK.

| Entry point: CPSTACK - queue CPRQUEST blocks (BALR)

Entry conditions: GPR 1 points to a CPRQUEST block.

Exit conditions: None

CPSYM

Module name: CPSYM

Entry point: CPSYM

Purpose: The CPSYM module does not contain executable code. It is essentially an in-core load map of the CP-67 nucleus. It contains the EBCDIC name and hex address of each CP module as well as some of the more important entry points and control words.

DEDICATE

Module name: DEDICATE

Entry point: DEDICATE

Purpose: This module creates from CP-67 free storage a set of RCHBLOK, RCUBLOK, and RDEVBLOK control blocks to define a dedicated (nonshared) multiplexer device. The control blocks are chained on to the existing chain of control blocks pointed to by RCHSTART. The MRDEBLOK is flagged (in MRDEFLAG) as being dedicated (MRIDED); the UTABLE address of the owning user is stored in MUSER.

Entry conditions: GPR 11 contains the UTABLE address.
GPR 1 contains the real device address.

Exit conditions:

Successful:	Condition code 0 - GPR 1 contains the address of the RDEVBLOK created.
Unsuccessful:	Condition code 1 - Nonexistent real device Condition code 2 - Device in use

DIAGDSK

Module Name: DIAGDSK

Entry Point: DIAGDSK

Purpose: This module is entered from PRIVLGED when a user has issued a diagnose call for a specialized I/O task to be performed on a 2311 or 2314. DIAGDSK checks for various calling errors; if none is present, an I/O task is made up and scheduled for execution by calls to QUERIO and DISPATCH. Upon completion, a condition code of 0 indicates to the user that the I/O has been completed with no errors (no CSW being returned to the user). Errors are signalled to the user as indicated below. The use of DIAGDSK for simple I/O provides a significant speed improvement for CMS or other users who have a CCW string of similar format.

Entry point: DIAGDSK

Entry conditions: GPR5 points to user's "R1", which must hold the device address. GPR4 points to user's "R2", which must point to a CCW-string of the following format:

```
(1)  SEEK    BBCCHHR        (below)
(2)  SEARCH BBCCHHR+2       (below)
(3)  TIC   BACK TO SEARCH
(4)  READ OR WRITE OF UP TO 4096 BYTES
          (up to 824 bytes for CMS)
(5)  NO-OP
(6)  BBCCHHR      SEEK/SEARCH ARGUMENTS (7 bytes)
```

Exit Conditions: (Upon return to user via DISPATCH)

Condition-Code (CC) = 0: I/O complete with no errors.

CC = 1: SIO failed, CSW stored.
(CSW+4 & CSW+5 returned to user)

CC = 2: Either an attempt to write on a read-only disk
(program-check returned to user)
or
other I/O error on completion
CSW (8 bytes) returned to user
(sense bytes available if user does a 'SENSE')

CC = 3: Not attached, neither 2314 nor 2311,
or invalid DIAGNOSE call by user.
Error-code returned to user in his R15, as follows:

1 = Not attached (error from VUNITSCN in CP)
2 = Device is neither 2314 nor 2311

3 = Pointer to user's CCW-string not dbl-word aligned
4 = SEEK/SEARCH arguments not within user core
5 = Read/write CCW neither read (06) nor write (05)
6 = Read/write byte-count = 0
7 = Read/write byte-count greater than 4096
8 = Read/write buffer not within user core
9 = Condition-code 2 (busy) on actual SIO as attempted by CP
10 = Condition-code 3 (not operational) on actual SIO as attempted by CP

DIAL

Module name: DIAL

Entry point: DIAL

Purpose: Attaches user's terminal as a dedicated device to an existing virtual 2701, 2702, or 2703 line in the virtual machine specified. The UTABLE and MVDEBLOK are returned to free storage and the user terminal is removed from CP control.

Registers 0-11 are saved upon entry to DIAL.

Entry conditions: Entry is from LOGIN after a DIAL command. GPR 10 points to an eight-character userid. GPR 11 points to a UTABLE.

Exit conditions: If successful, GPR 11 is set to zero.

Module name: DISPATCH

Entry points: DISPATCH, DSPTCHA, DSPTCHB, DSPTCHC, DISACT, DISDRQ, and DISIO

DISPATCH is entered when some process has been completed or cannot continue any further until some other event has completed (an I/O operation). It updates the user's control blocks to reflect his current status. If a user was running, DISPATCH attempts to restart him. If it cannot restart the running user or if there was none, DISPATCH will dequeue any CP-67 deferred work requests and start them. When all CP requests are exhausted DISPATCH will run the highest priority, runnable, and in queue user if there is one, or it will enter enabled wait state.

Entry point: DISPATCH

Entry conditions: GPR 11 points to a valid UTABLE to be charged for time spent in CP-67 since the last charge.

Entry point: DSPTCHA

Entry conditions: Same as DISPATCH except entered after processing a program interrupt from a running user where processing has not changed the virtual PSW.

Entry point: DSPTCHB

Entry conditions: Same as DSPTCHA except processing has changed the virtual PSW.

Entry point: DSPTCHC

Entry conditions: No change to user pointed to by R11.

Entry point: DISACT

Entry condtions: Charge user for CPU time used since last charge. Called when a routine has changed the status of a user.

Exit conditions: None.

Entry point: DISDRQ - drop a user from a queue (called)

Entry conditions: GPR 11 points to user to be dropped from a queue.

Exit condtions: None.

Entry point: DISIO (called)

Entry conditions: GPR 11 points to user that has had his status changed. Called by routines which have updated a user's status and are not returning or going to DISPATCH (either directly or indirectly), with GPR 11 pointing to this user.

DSKDUMP

Module name: DSKDUMP

Entry point: DSKDUMP

Purpose: This module is entered from module PSA when CP-67 issues an SVC 0 ABEND, on activation of the PSW restart button, or from PROGINT for a system program error. The module contains code to dump core to a printer, tape or disk. The dump will be of all core or of only those pages marked as *CP* or FREE in the CORTABLE.

Entry point: DSKDUMP

Entry conditions: General registers are stored at GREGS.

Exit conditions: An exit is taken by performing a software re-IPL of the system.

EXTEND

Module name: EXTEND

Entry point: EXTEND

Purpose: EXTEND is used to obtain a number of pages for CP-67 common buffer space called Free Area. It has a table, EXT1, to indicate the number of pages required, depending upon the real machine core size. It calls PAGFREE repeatedly to get these pages. It is called by FREE.

Registers 0-15 are saved upon entry to EXTEND.

Entry point: EXTEND

Entry conditions: None

Exit conditions: GPR 1 points to an area which may be incorporated into the free storage zone. GPR 0 contains the length of this area in bytes.

FREE

(See Figure 44 for an overview of FREE processing.)

Note: &TRACE(4) option must be chosen as sysgen time in order to gather statistics in FREE/FRET.

Module name: FREE

Entry points: FREE, FRET, FRETR

Purpose: To maintain and allocate units of system free storage, with minimum fragmentation. Free storage is utilized by CP-67 for I/O tasks, CCW strings, buffers--in fact, for all but real channel-control unit-device blocks, CORTABLEs, and save areas.

The most frequently used storage block sizes, some 29 in number, constituting about 99% of all FREE/FRET calls, have been allocated into ten subpools. All FREE/FRET calls for the doubleword block size listed in the left column below receive the corresponding doubleword block in the right column:

Number of double words called for	Subpool size actually used
1	1
2 or 3	3
4	4
5	5
6, 7 or 8	8
9 or 10	10
11 - 14	14
15 - 18	18
19 - 23	23
24 - 29	29

A block from the subpool chain is given priority in a call to FREE for a subpool size; a block is selected from the regular free storage chain only if none is available from the subpool chain, or if the call to FREE is for a block greater than 29 doublewords. A FRET call, likewise, is checked for subpool size; if it corresponds, the block returned is patched into the chain on a LIFO (last-in-first-out) basis, that is, push-down stack.

A special entry, FRETR, enables CPINIT and EXTEND to bypass subpool consideration whether or not the block being returned is subpool size.

Various statistical information is now kept in the FREE routine, starting at entrypoint FREELIST. The code for statistical information can be removed by revision of a SETA symbol, if speed of performance takes precedence over statistics gathering: by assigning SETA a value of 1, statistics are included; a value of 0 causes their removal. The following is a partial list of pointers, counters, and statistical quantities of interest (the names in parentheses are labels of these quantities):

- end of highest subpool block given out (ENDSUB)
- address of lowest regular block given out (BEGINREG)
- end of free area in lower core (ELOFREE)
- beginning of free area in high core (BHIFREE)
- subpool FRET calls requiring regular FRET (SBFRTREG)
- table of pointers to subpools (SUBTABLE)
- number of times subpools returned (SUBRETN)
- number of times EXTEND is called (EXTCALL)

Other statistical quantities (for debugging and operations research, only)

- maximum value attained by FREENUM (MFREENUM)
- FREE/FRET calls for sizes not in subpools (FREEUSED,FRETUSED)
- counts of satisfied and unsatisfied FREE subpool calls (SUBFREE,USUBFREE)
- count of successful FRET subpool calls (SUBFRET)

Statistical counters for each subpool size

- number of subpool blocks in use; number left (SUBLEFT)
- maximum value attained by SUBUSED (MSUBUSED)
- cumulative times spent in FREE and FRET (TIMEFREE,TIMEFRET)
- count of subpool-range sizes referenced (SIZEREF)

Entry point: FREE - allocate a region of free storage (BALR)

Entry conditions: GPR 0 contains the number of doublewords requested.

Registers 0-15 are saved upon entry.

Exit conditions: GPR 1 contains a pointer to the region of the size requested. This region will always be on a doubleword boundary.

Entry point: FRET - return a region to free storage (BALR)

Entry conditions: GPR 0 contains a count of the number of doublewords being returned. GPR 1 contains a pointer to the initial doubleword of the region. This pointer must always be on a doubleword boundary.

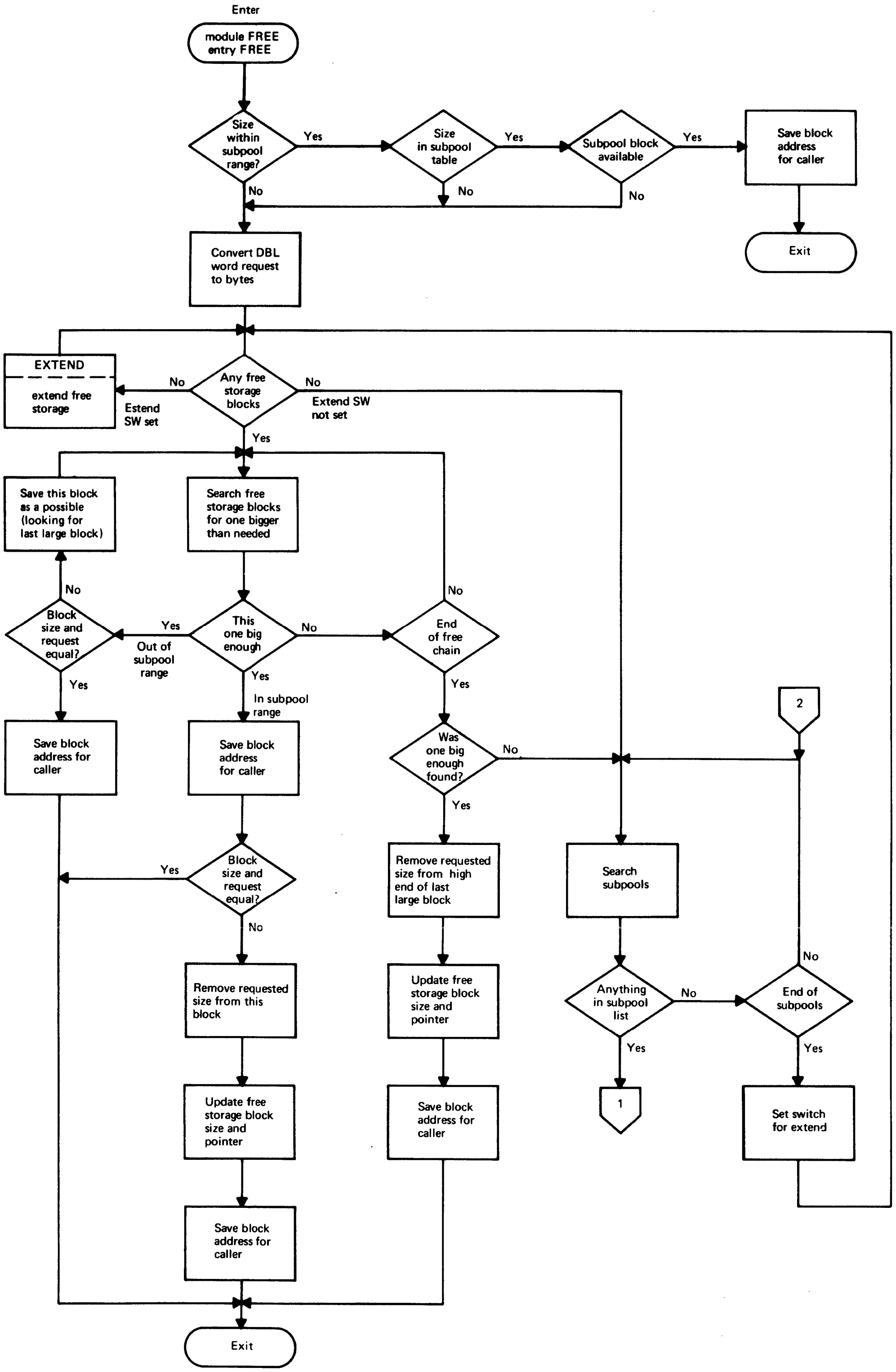
Register 0-15 are saved upon entry.

Exit conditions: None. No reference may be made to a region after it has been returned to free storage.

Entry point: FRETR - return a region to free storage. Same as FRET except does not attempt to use subpool logic (BALR)

Entry conditions: Same as FRET

Exit conditions: Same as FRET



Enter
module FREE entry FREE
Size within subpool range?
Yes
Size in subpool table
Yes
Subpool block available
Yes
Save block address for caller
Exit
No
No
No
Convert DBL word request to bytes
EXTEND
extend free storage
No
Extend SW set
Any free storage blocks
No
Extend SW not set
Yes
Save this block as a possible (looking for last large block)
Search free storage blocks for one bigger than needed
No
Block size and request equal?
Yes
Out of subpool range
This one big enough
No
End of free chain
No
Yes
Yes
In subpool range
Yes
2
Save block address for caller
Save block address for caller
Was one big enough found?
No
Yes
Block size and request equal?
Yes
No
Remove requested size from high end of last large block
Search subpools
Remove requested size from this block
Update free storage block size and pointer
Anything in subpool list
No
End of subpools
No
Yes
Yes
1
Set switch for extend
Update free storage block size and pointer
Save block address for caller
Save block address for caller
Exit

Figure 44. CP-67 FREE (1 of 2)

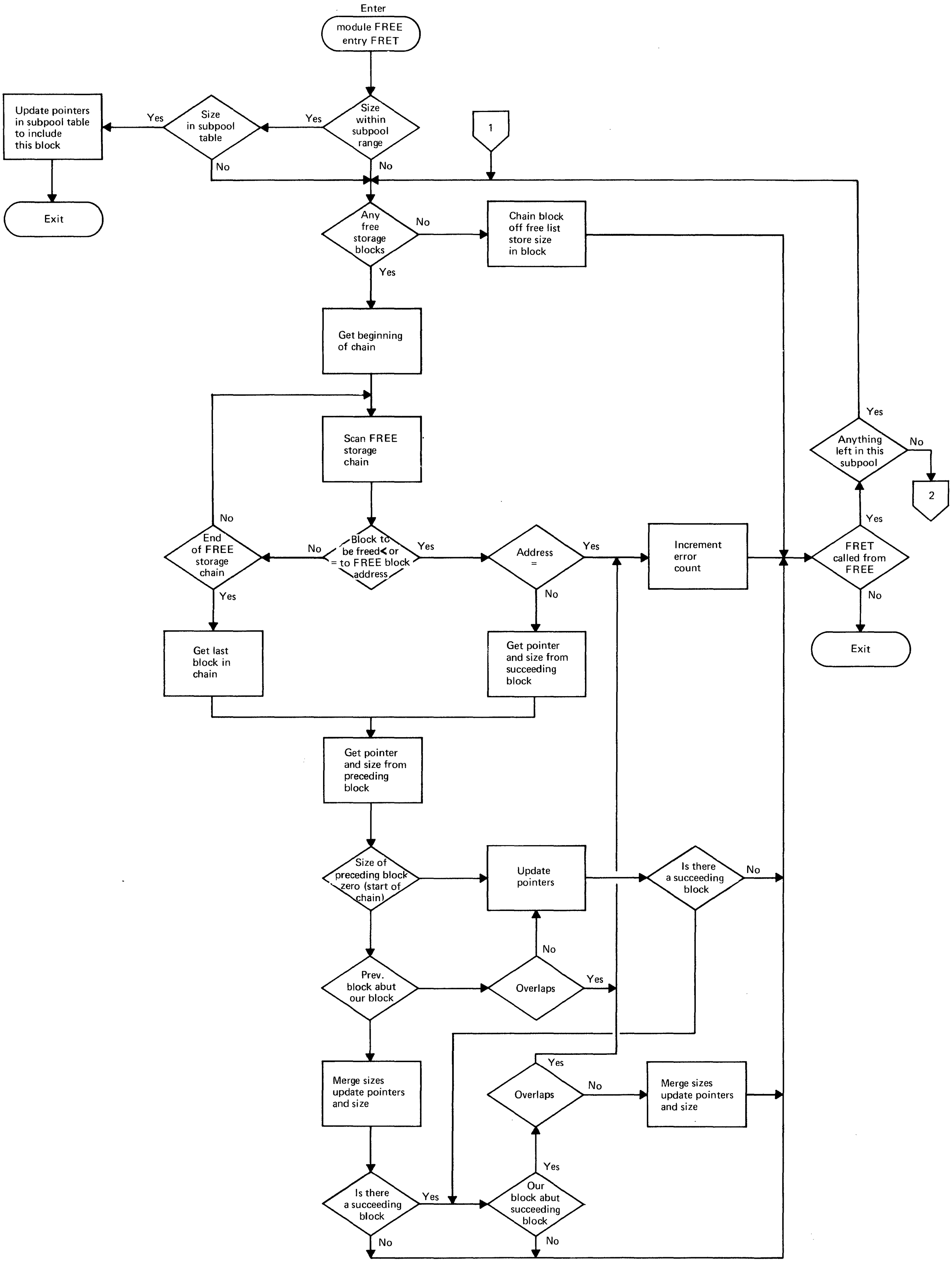

Enter
module FREE entry FRET
Size within subpool range
Yes
No
Size in subpool table
Yes
No
Update pointers in subpool table to include this block
Exit
1
Any free storage blocks
No
Yes
Chain block off free list store size in block
Get beginning of chain
Scan FREE storage chain
Block to be freed < or = to FREE block address
No
Yes
End of FREE storage chain
No
Yes
Address =
Yes
No
Increment error count
Get last block in chain
Get pointer and size from succeeding block
Get pointer and size from preceding block
Size of preceding block zero (start of chain)
Update pointers
Is there a succeeding block
No
Prev. block abut our block
Overlaps
No
Yes
Merge sizes update pointers and size
Overlaps
Yes
No
Merge sizes update pointers and size
Is there a succeeding block
Yes
No
Our block abut succeeding block
Yes
No
Anything left in this subpool
Yes
No
2
FRET called from FREE
Yes
No
Exit

Figure 44. CP-67 FREE (2 of 2)

Module name: IOERROR

Entry points: IOERROR, VERROR, RECERROR, MCKERR, FINDLOG, FMTLOG, LOGRETN, FINDMC, FINDIO, FMTMLOG, FMTLOGM, FMTILOG, FMTLOGI

On entry to each, registers 0-11 are saved.

Purpose: The IOERROR routine analyzes and retries CP-generated I/O errors incurred while paging, spooling, or reading the directory. Selected I/O errors and machine check errors are recorded on the SYSRES volume at a predefined location. Warning messages are sent to the operator when repeated I/O errors occur on a device used by CP-67 for paging, spooling, or directory space. The routine also contains code to locate and/or format the error-recording records to be used. The locate function is initiated by CPINIT. The format function (actually erasing any previous data) is performed by a special diagnose code executed by a privilege class C user only.

Entry point: IOERROR

Entry conditions: GPR 6 contains a pointer to the real device block for the device on which the error occurred. GPR 9 contains a pointer to the IOTASK block which did not execute properly.

Exit conditions: If retry is successful, control returns to the program which issued the original I/O request (return address in TASKIRA). If 64 retries of the I/O are all unsuccessful, exit is to the dispatcher, and the system will ABEND.

Entry point: VERROR

Entry conditions: GPR 8 contains a pointer to the virtual device block.

Exit conditions: None

Entry point: RECERROR

Entry conditions: Called by IOERROR and VERROR

Exit conditions: The appropriate counter in the real or virtual device block will be updated to reflect the error. If the error is the first of its type to be encountered for this device, or the error counter overflows, the CE LOGREC will be updated to reflect the latest error.

Entry point: MCKERR

Entry conditions: Entered whenever a machine check occurs, whether in supervisor or problem state.

Exit conditions: Return is to the machine check interrupt handler, MCHEKINT.

Entry point: FINDLOG

Entry conditions: Called by CPINIT

Exit conditions: Returns to caller

Entry point: FMTLOG

Entry conditions: Called by FINDLOG and "Diagnose" by customer engineer to clear and format CE cylinder

Exit conditions: If successful, return to caller. If permanent I/O error, system will ABEND.

Module name: IOINT

Entry points: IOINT, IOISTVDE, IOISTVCU

Purpose: IOINT receives control from the I/O new PSW. It saves the state of the running user's machine, if any, and determines what further action is required. In normal processing, an exit is taken to the IOTASK block's TASKIRA.

| If the &TRACE(2) and/or the &TRACE(3) options are selected in the LOCAL COPY file, then the IOINT module also generates entries in the selector and/or multiplexor trace table respectively.

Entry point: IOINT

Entry conditions: Receives control from the I/O new PSW.

Exit conditions: Exits through a call to TASKIRA followed by a transfer to DISPATCH with GPR 11 pointing to the chargeable user.

| Entry point: IOISTVCU - increment control unit pending count (BALR)

| Entry conditions: GPR 6 points to a virtual channel block.

| Exit conditions: None.

| Entry point: IOISTVDE - increment device pending count and control pending count if device count was zero (BALR)

| Entry conditions: GPR 7 points to a virtual control unit block and GPR 6 points to a virtual channel block.

IPL

Module name: IPL

Entry point: IPL

Purpose: The IPL module is responsible for simulation and correct interpretation of various IPL sequences supported for several devices. It is unique in that it resides in virtual memory. The virtual memory location is the page boundary closest to half the virtual memory size or page X'20000', whichever is the smaller.

Entry point: IPL

Entry conditions: The IPL module resides in virtual memory and its parameters are passed by the control program through the use of the first 24 bytes of page zero of that virtual memory. (Note: Since the IPL sequence destroys these bytes on the real machine, no alteration of behavior from the real machine is seen.) The information passed consists of (1) the virtual device address and (2) the virtual device type code.

Exit conditions: The IPL module transfers control to the system just IPL'ed via user's lower core location zero.

LINK

Module name: LINK

Entry point: LINK

Purpose: To dynamically attach virtual DASD devices based on a machine description entry (MDENT) found in the appropriate machine description file. Supports the LINK console function. Called from CPSTACH, link command.

Entry point: LINK

Entry conditions:
   GPR 1 points to a parameter list as follows:
         DC CL8'userid',CL8'password',XL2'XXX',XL2'YYY'
         where XXX is the virtual address to be found
         in the directory, and YYY is the address to
         be used in attaching the device.
   GPR 2 contains zero for read-only access, 1
         for read/write access.
   GPR 11 contains UTABLE address of requesting user.

Exit conditions:
   GPR 2 contains an error code as follows:
       0 - Successful, attached as requested
       1,2 - Not used
       3 - 'userid' found, address XXX not in directory
       4 - Device YYY already attached
       5 - Password is bad or the device is not shareable
           for the given access mode
       6 - 'userid' is in INLOGON state
       7 - A write link to XXX already exists. LINK denied.
       8 - The required volume is not mounted or not
           attached to system
       9 - Attached in read-only, not in write as requested
       10 - 'userid' not in directory
       11 - Address XXX not a DASD device
       12 - Directory locked

LOGFILES

Module name: LOGFILES

Entry point: LOGFILES

Purpose: This module counts the number of spool file blocks awaiting processing for the user and returns the address of a message to the caller. Called by LOGIN and CFSQRY (queue files command).

Registers 0-15 are saved upon entry to LOGFILES.

Entry point: LOGFILES

Entry condition: GPR11 pointing to user's UTABLE

Exit condition: None

Module name: LOGIN

Entry points: LOGIN, OPMSG, AUTOLOGIN

Purpose: The LOGIN module is responsible for setting up and logging in a new user, creating from free storage those control blocks required and allocating the required machine resources.

Entry point: LOGIN - to log in a new user.

Entry conditions: GPR 6 contains a pointer to the MRDEBLOK desiring entrance to the system.

Registers 0-15 are saved upon entry.

Exit conditions: GPR 11 contains the address of the new user UTABLE if the logon was successfully completed; otherwise GPR 11 contains zero.

Entry point: OPMSG - Inform system operator of LOGIN, LOGOUT activity.

Entry conditions: GPR 11 contains address of UTABLE; GPR 7 contains character string CL4' OFF' or CL4' ON'.

Registers 0-11 are saved upon entry.

Exit conditions: None

Entry point: AUTOLOGIN - sets up pointers for automatically logging on a specified user, and joins standard logon code. Called from CPINIT.

Entry conditions: Same as LOGIN

Registers 0-15 are saved upon entry.

Exit conditions: Standard LOGIN exit

MRIOEXEC

Module name: MRIOEXEC

Entry points: MRIOEXEC, RPUNCH, PRIRA, CRIRA, PUIRA

Purpose: These routines are entered when an interrupt occurs on the unit record equipment. For a reader interrupt, all of the cards are read and stored. For a printer or punch interrupt, the corresponding disk buffer is checked, and if there is pending data in the buffers, it is printed or punched.

Entry points: MRIOEXEC, PRIRA, PUIRA, CRIRA

Entry conditions: All of these entries are entered from IOINT on the appropriate interrupt. GPR 6 contains the address of the corresponding MRDEBLOK, and GPR 10 contains a pointer to a doubleword of CSW information from the interrupt.

Exit conditions: After performing their functions, all entry points return to DISPATCH with GPR 11 pointing to the user for whom the input-output operation was being processed.

Entry point: RPUNCH

Entry conditions: GPR 4 points to the buffer containing the accounting information to be punched.

Exit conditions: The accounting information is punched when a punch is available.

Module name: MVIOEXEC

Entry points: MVIOEXEC, MVICLPR, MVICLPN, MVICLCR, MVIPRINT

Purpose: The MVIOEXEC module handles all virtual input-output operations to the user's multiplexer channel. This includes terminal and spooling functions.

Entry point: MVIOEXEC - handle virtual input-output requests to the multiplexer. Called by VIOEXEC.

Entry conditions: GPR 4 points to the first half of the instruction. GPR 5 points to the second half. GPR 9 contains the virtual device address. The user's virtual CAW is pointing to the virtual CCW list.

Exit conditions: The user's condition code is set to reflect the status of his virtual device.

Entry points: MVICLPR, MVICLPN - close printer and punch files.

Entry conditions: GPR 8 contains the address of the virtual device block for which the file is to be closed.

Exit conditions: If there was an open file, it is closed; that is, it is put in the closed file chain, and real output is initiated if the corresponding device is available.

Entry point: MVICLCR - close file on card reader.

Entry conditions: Same as for MVICLPR

Exit conditions: If there was an open file, it is closed; that is, any remaining data in that file is discarded and the next file, if any, is made accessible.

Entry point: MVIPRINT - print a line on the virtual printer.

Entry conditions: GPR 1 is the address of a buffer containing data for printer output. GPR 0 contains the byte count of the data.

Exit conditions: Data is packed and put in the spooled file.

PACK

Module name: PACK

Entry points: PACK, UNPACK

Purpose: To pack and unpack blanks from the spooling data used by MRIOEXEC and MVIOEXEC.

Entry point: PACK - compress blanks from input data.

Entry conditions: GPR 1 is the address of a byte containing input data count, followed by the input data. GPR 2 is the address of the output buffer.

Registers 0-7 are saved upon entry.

Exit conditions: First byte of output buffer contains output data count, followed by output data.

Entry point: UNPACK

Entry conditions: GPR 1 contains the address of the input buffer, in the same format as the output buffer for PACK. GPR 2 contains the address of the output buffer.

Registers 0-6 are saved upon entry.

Exit conditions: The unpacked data appears in the output buffer.

PAGEGET

Module name: PAGEGET

Entry points: PAGEGET, PAGERLE

Purpose: The module handles DASD storage requirements for paging.

Entry point: PAGEGET - allocate space for one page. Called by PAGTRANS.

Entry conditions: None. Registers 0-12 are saved upon entry.

Exit conditions: GPR 2 contains device index and DASD address, if found. If not found, GPR 2 contains 0.

Entry point: PAGERLE - release paging DASD area for this user. Called by PAGOUT.

Entry conditions: GPR 11 points to this user's UTABLE and registers 0-12 are saved upon entry. GPR 5 points to a SWPTABLE entry containing the address of the record to be released.

Exit conditions: None

PAGTR

Module Name: PAGTR

Entry points: PAGOUT, PAGFRET, PAGSHARE

Purpose: This module handles functions which require a knowledge of the nature of the mapping device.

Entry point: PAGOUT - remove a user's pages from core. Called from CFSIPL or USEROFF.

Entry conditions: GPR 11 points to the UTABLE of the user whose pages are to be scrapped.

Registers 0-13 are saved upon entry.

Exit conditions: None.

Entry point: PAGFRET

Entry conditions: GPR 2 points to first page to be made available for users, and GPR 0 is count of pages.

Exit conditions: None.

Entry point: PAGSHARE - called by CONSOL for first user of a named system to bring into core and lock any shareable pages.

Entry conditions: GPR 1 = first shared page number; GPR 5 = PAGTABLE address; GPR 6 = count of saved pages; GPR 7 = address of first entry of saved SWPTABLE; GPR 11 = UTABLE address.

Registers 1-7 are saved upon entry.

Exit conditions: None. All required pages are in core and locked.

PAGTRANS

Module name: PAGTRANS

Entry points: PAGTRANS, PAGUNLOK, PAGFREE, CORUSER, CORTENT, WAITPAGE, DRMWAIT

Purpose: This module handles functions which require a knowledge of the nature of the mapping device.

Entry point: PAGTRANS - translate virtual address and page in, if required.

Entry conditions: GPR 1 contains the virtual byte address. GPR 2 contains control parameters in byte 3 as follows: BRING, to bring the page into core; LOCK, to lock the page in core (implies BRING); DEFER, to prevent return to caller until page is in core; CHANGE, to set changed bit for this page; USED, to set used bit for this page.

Registers 0-15 are saved upon entry.

Exit conditions: GPR 2 contains the real byte address.

Entry point: PAGUNLOK - unlock a virtual page (BALR)

Entry conditions: GPR 2 contains a real byte address within the page to be unlocked.

Registers 0-7 are saved upon entry.

Exit conditions: None

Entry point: PAGFREE - obtain free page for free storage. Called by EXTEND.

Entry conditions: None. Registers 3-11 are saved upon entry.

Exit conditions: GPR 1 points to the page address which can be included in the free area.

Entry point: CORUSER
Fullword containing address of first page available to users, set by PAGFREE. Initialized value is A(CPEND).

Entry point: CORTENT
Fullword containing address of end of CORTABLE, set by PAGFREE.

Entry point: WAITPAGE - reflects completion of page I/O. Called by IOINT.

Entry conditions: GPR 9 points to IOTASK; GPR 10 points to CSW.

Registers 0-15 are saved upon entry.

Exit conditions: None

Entry point: DRMWAIT - After a 2301 drum paging interrupt, stacks CPREQUEST blocks for each additional page if the I/O operation involved more than one page. Chains together any IOTASKS queued off the allocation block and calls QUERIO.

Entry conditions: GPR 9 points to IOTASK; GPR 10 points to CSW.

Exit conditions: Transfers to WAITPAGE.

PRIVLGED

Module Name: PRIVLGED

Entry points: PRIVLGED, FREEPST, FRETPST

Purpose: Provide non-I/O privileged instruction simulation.

If the &TRACE(5) option is selected in the LOCAL COPY file, then the PRIVLGED module accumulates statistics in low core (defined in STAT COPY) about the number and type of privileged instructions executed.

Entry point: PRIVLGED

Entry Conditions: R13 points at the real address of the privileged instruction.

Exit conditions: Via a GOTO to VIOEXEC if the instruction is for I/O or to DISPATCH after the instruction is simulated.

Entry point: FREEPST - creates real copies of virtual 67 page tables.

Entry conditions: Register 11 points to UTABLE.

Registers 0-4 are saved upon entry.

Exit conditions: None.

Entry point: FRETPST - releases real copies of virtual 67 page tables.

Entry conditions: Register 11 points to UTABLE.

Registers 0-5 are saved upon entry.

Exit conditons: None.

Module name: PROGINT

Entry points: PROGINT, REFLECT

Purpose: PROGINT is entered from the program interruption new PSW. It attempts to determine whether the interruption occurred from the Control Program issuing a simulated instruction (for example, SLT) or the user issuing a privileged instruction; in the latter case, control is passed to PRIVLGED, which interprets and simulates user-issued privileged instructions.

---

Entry point: PROGINT

Entry conditions: PROGINT is entered from the program interrupt new PSW.

Exit conditions:

If the interruption occurred as the result of a program interruption in the Control Program indicating program trouble, a terminal system dump occurs. If the program interruption was the result of a user issuing a privileged operation (the usual condition), control is passed to PRIVLGED.

Entry point: REFLECT - reflect an interrupt to the user.

Entry conditions: GPR 13 points to the old PSW for the interruption condition which is to be reflected. The user's registers have already been saved.

Exit conditions: After making changes in the user's UTABLE to reflect the interrupt, REFLECT transfers control to DISPATCH with GPR 11 pointing to the affected user.

Note: The use of register 13 in this case to point to the proper old PSW is a deviation from standard calling sequence practice.

PSA

Module name: PSA

Entry points: SVCINT, SVCINIT, EXTINT, MCHEKINT, SVCDUMP

Purpose: To initialize and maintain the save areas provided as a part of the calling protocol maintained within CP-67. SVCINIT is called to initialize the save areas, and SVCINT is entered by the SVC interrupt occurring, indicating a request for linkage or return by a CP-67 module. Also handles external and machine check interrupts.

If the &TRACE(1) option is selected in the LOCAL COPY file, then PSA also places entries in a trace table for CP SVC's.

Entry point: SVCINIT - initialize save area.

Entry conditions: Entered via a BALR 14,15 to initialize the save areas from CPINIT.

Exit conditions: Initialized save areas.

Entry point: SVCINT

Entry conditions: Entered via an SVC 0, 4, 8, 12, 16, or 20 to perform, respectively, DIE, DUMP, LINK, RETURN, RELEASE, or SAVEGET.

Exit conditions: See "SVC Interruptions" in Section 2.

Entry point: EXTINT

Entry conditions: Entered from the external interrupt new PSW. If the interrupt occurred because of the external interrupt pushbutton, the system operator is logged out. This allows him to log in again from an alternate console. If the interrupt occurred because of a timer interrupt, the running user, if any, is saved, and an exit is taken to DISPATCH to determine whether there is any work.

Exit conditions: Exits to DISPATCH under normal conditions with GPR 11 pointing to the interrupted user.

Entry point: MCHEKINT

Entry conditions: Upon detection of a hardware malfunction.

Exit conditions: After printing warning messages, if machine check was in CP-67 mode, terminate all processing. If machine check was in user mode, the user is informed that a machine check has occurred. The machine check is reflected back to the virtual machine, which is placed in CP console function mode; this enables a console function to be issued.

Entry point: SVCDUMP - branched to from within PSA on an SVC 0 or entered from a PSW restart. Branches to DSKDUMP to abnormally terminate.

Entry conditions: None

Exit conditions: None

QUEVIO

Module name: QUEVIO

Entry points: QUEVIO, QUERIO, CHFREE

Purposes: This module queues requests for input-output operations on the selector channels, determines whether the channels are available, prepositions access arms on direct access devices, and initiates the input-output operations.

| Entry point: QUEVIO - queue virtual task block (BALR)

Entry conditions: GPR 1 points to an IOTASK block which is to be queued. GPR 2 points to the virtual device block.

| Registers 0-14 are saved upon entry.

| Exit conditions: None. Transfer is to CHFREE, to initiate
| operation of the task.

| Entry point: QUERIO - queue real task block (BALR)

Entry conditions: GPR 1 points to an IOTASK block which is to be queued. GPR 6 points to the real device block on which the input-output operation is being performed.

| Registers 0-14 are saved upon entry.

| Exit conditions: None. Transfer is to CHFREE, to initiate
| operation of the task.

| Entry point: CHFREE - start idle channel (BALR)

Entry conditions: GPR 1 points to a real channel block for which input-output operations are to be initiated, if possible.

| Registers 0-14 are saved upon entry.

Exit conditions: None. If the operation can be started, a zero condition code from the SIO operation causes the user to be removed from the IOWAIT condition if the operation originated from a virtual machine. For a nonzero condition code, CHFREE calls the TASKIRA with the condition code indicated in GPR 0. CHFREE can also call itself recursively if it determines that the operation just initiated has freed the channel.

RDCONS

Module name: RDCONS

Entry point: RDCONS

Purpose: This module creates a CCW "package" (according to the type of terminal it is servicing) that can be stacked as a read request for that terminal. It allows the different remote terminals to be treated as though each were a 1052.

Registers 0-10 are saved upon entry to RDCONS.

Entry point: RDCONS

Entry conditions: GPR 1 contains the address of the input buffer (132 bytes). GPR 2 holds the options that are requested when RDCONS is called: EDIT, OPERATOR, or UCASE. If EDIT is specified, character or line deletions are performed as specified. If UCASE is specified, all lowercase letters are translated to equivalent uppercase letters. If OPERATOR is specified, a read is performed from the operator's terminal. GPR 3 contains the address to which the Control Program will return control after completion of the console I/O. GPR 11 points to the UTABLE of the user to whom the read is directed.

Exit conditions: The return is made from RDCONS immediately with all registers restored. At the termination of the read operation, GPR 0 contains the byte count of the input message; GPR 2 contains an error condition code, if any. Control is returned to the address specified in GPR 3 at the call to RDCONS.

RDSCAN

Module name: RDSCAN

Entry points: LINKSCAN, RDSCAN, DEVSCAN

Purpose: To determine whether a virtual DASD device is currently attached to the virtual machine of an active user (that is, a "link" exists). Definition: Two virtual devices having the same RDEVBLOK and relocation factor are the same.

Registers 0-9 are saved upon entry to this module.

Entry point: LINKSCAN

Entry conditions: GPR 11 is the UTABLE address of the current user, not to be included in the search. GPR 10 is the UTABLE address of the first user to be scanned. GPR 0 is the relocation factor of the virtual device, and GPR 1 is the address of the RDEVBLOK.

Exit conditions:
Condition code 0 - No link exists.
Condition code 1 - Read-only link(s) exists.
Condition code 2 - Read/Write link exists.

GPR 10 is the UTABLE address of the user having the link. For no link, GPR 10 is equal to GPR 11.

Entry point: RDSCAN

Entry conditions: Same as LINKSCAN

Exit conditions: Same as LINKSCAN except that if all links are read-only, no return is made until all user machines have been examined or a read/write link is encountered.

Entry point: DEVSCAN

Purpose: To determine whether any link exists to the real device regardless of the relocation factor.

Entry conditions: Same as LINKSCAN except GPR 0 is not used. To include current user in search, set GPR 10 equal to GPR 11.

Exit conditions:
Condition code 0 - No link exists.
Condition code 3 - A link exists. GPR 10 points to
UTABLE of first link encountered.

RECFREE

Module name: RECFREE

Entry points: RECFREE, RECFRET

Purpose: To handle the spooling requests for available disk records in much the same manner as free storage handles main memory.

Entry point: RECFREE - obtain free record.

Entry conditions: None. Registers 2-6 are saved upon entry.

Exit conditions: GPR 0 = 1; GPR 1 is the address of a doubleword containing the DASD record address and device code in the following format: bytes 0-1 are zero; bytes 2-3 contain the cylinder number; bytes 4-5 contain the track number; byte 6 contains the record number; and byte 7 contains the device code.

Entry point: RECFRET - return disk record to free storage.

Entry conditions: GPR 1 contains the address of a doubleword in the RECFREE format.

Registers 0-7 are saved upon entry.

Exit conditions: None

RESINT

Module name: RESINT

Entry points: RESINT, RESIRA

Purpose: This module performs a virtual system reset.

Entry point: RESINT

Entry conditions: GPR 11 points to the UTABLE of the user for whom the reset is desired.

Registers 0-11 are saved upon entry.

Exit conditions: None

Entry point: RESIRA - interrupt return address set by RESINT for IOTASKS queued up for a user to be reset; clears interrupt without resetting virtual machine status; entered from IOINT.

Entry conditions: GPR 9 points to IOTASK. RIO points to CSW.

Registers 0-11 are saved upon entry.

Exit conditions: None

SAVECP

Module name: SAVECP

Entry point: SAVECP

Purpose: This module writes the core image of CP-67 onto the system residence volume (currently specified by a REP card in the SAVECP module) at the end of a card or tape load of CP-67 into core. At IPL time the SAVECP function is reversed and it reads in the core image.

Entry point: SAVECP

Entry conditions: The module requires that the disk address be loaded with it, and that the device be a 2311 or 2314. The addressability of the module is contained in GPR 3 (not 12 as in the norm).

Exit conditions: The disk address is stored in word 0, and the address of the location containing the label is stored in GPR 2, when control is transferred to CPINIT after SAVECP-restore. After a SAVECP-save a DISK LOAD OK message is printed.

SCANUNIT

Module name: SCANUNIT

Entry points: RUNITSCN, VUNITSCN

Purpose: To accept a device address, either real or virtual, and scan down the appropriate list, setting up pointers to the various level blocks.

Registers 0-8 are saved upon entry to this module.

| Entry point: RUNITSCN - scan for real device block (BALR)

Entry conditions: GPR 8 contains the address to be searched for.

Exit conditions: GPR 6 contains the pointer to the real channel block, if found. GPR 7 contains a pointer to the real control unit block, if found. GPR 8 contains a pointer to the real device block, if found. The condition code is set as follows:

0 - all blocks found
1 - channel block not found (no pointers valid)
2 - control unit block not found (channel pointer valid)
3 - device block not found (channel and control unit pointers valid

| Entry point: VUNITSCN - scan for virtual device block
| (BALR)

Entry conditions: GPR 8 contains the address to be searched for. GPR 11 points to the user whose blocks are to be searched.

Exit conditions: Same as for RUNITSCN except pointers are to virtual blocks.

SCHEDULE

Module name: SCHEDULE

Entry points: SCHEDULE, SCLOCK

Purpose: Contains extended DISPATCH functions.

Entry point: SCHEDULE

Entry conditions: R1 is non-zero if the UTABLE pointed to by R11 is in logon and is to be added to the real timer chain if that option is specified. Otherwise, R1 is zero and the R11 UTABLE is in logoff and it is to be removed from all chains that it currently may be on.

Exit conditions: None.

Entry point: SCLOCK

Entry conditions: Entered once a minute on a call from DISPATCH to update the decimal clock and to recalculate the paging activity variable. Also once an hour it resets the elapsed binary timer and any other locations dependent on its current value.

Exit conditions: None.

SCREDAT

Module Name: SCREDAT

Purpose: Contains system identification information that may be changed for each system created.

Module name: STCONSIO

Entry points: PRIMSG, STCONSIO

Purpose: This module will start an I/O request to a console or stack it if there are outstanding requests. If entered via PRIMSG, the request is stacked ahead of all current outstanding requests.

Entry point: STCONSIO

Entry conditions: GPR 6 contains the address of the console I/O request to be started or added. GPR 8 contains the device address and GPR 11 points to the appropriate user's UTABLE.

Exit conditions: The address of the CIOREQ pointer in the UTABLE will be changed to that of the current request, if the operation can be started immediately.

Entry point: PRIMSG

Entry conditions: Same as STCONSIO

Exit conditions: Same as STCONSIO, except the operation is always started and CIOREQ entry is always altered.

TMPSPACE

Module name: TMPSPACE

| Entry points: TMPSPACE, TMPRET, TMPERTN, T2311

Purpose: TMPSPACE dynamically allocates cylinders on DASD devices from devices of a specified type.

Registers 0-11 are saved upon entry to this module.

Entry point: TMPSPACE - obtain free cylinder.

Entry conditions: GPR 0 contains the number of contiguous cylinders desired. GPR 1 contains the desired device type code. GPR2 contains the type of space desired (for example, paging or spooling space, T-disk space, or directory space).

Exit conditions: GPR 0 contains the relocation factor of the allocated cylinder. GPR 1 points to the RDEVBLOK of the selected device. If space is not available, GPR 1 is set to zero.

Entry point: TMPRET - return a cylinder to free storage.

Entry conditions: GPR 0 contains the relocation factor of the allocated cylinder. GPR 1 points to the appropriate RDEVBLOK. GPR 2 contains the number of contiguous cylinders.

Exit conditions: None

Entry point: TMPERTN - interrupt return address for an IOTASK that erases TRK 00 of a T-DISK that has been released; entered from IOINT.

Entry conditions: GPR 9 points to IOTASK. GPR 10 points to CCW.

Registers 0-11 are saved upon entry.

Exit conditions: None

TRACER

Module Name: TRACER

Entry points: TRACER, TRINT

Purpose: This module handles the analysis and formatting of user specified tracing functions. Tracing is controlled by a table extension to the UTABLE. This table is located by the UTREXT entry in the UTABLE. The trace functions are controlled by a one-byte switch named TRSW defined in the UTABLE. The trace extension block called TREXT is defined in the UTABLE COPY. It contains control words, storage areas, and output buffers for the trace function. The TREXT block is 25 double words in size.

Entry point: TRACER - output trace data

Entry conditions: GPR1 contains the address of the output buffer.

Exit conditions: The buffer is cleared to all (132) blanks after being passed for console and/or printer output.

Entry point: TRINT - trace interrupt

Entry conditions:
GPR1 - virtual old PSW address
GPR3 - interrupt code
GPR4 - SVC extended interrupt code
GPR6,7 - SVC extended old PSW contents

Exit conditions: Trace buffer has been formatted and printed by calling TRACER. All necessary instructions have been restored and any "trace-following" SVC's have been set. The virtual machine PSW is ready to run from the correct location.

UNSTIO

(See Figure 45 for an overview of UNSTIO processing.)

Module name: UNSTIO

Entry point: UNSTIO

Purpose: To unstack and reflect virtual input-output interrupts from both the selector and multiplexer devices.

Registers 1-8 are saved upon entry to this module.

Entry point: UNSTIO

Entry conditions: GPR 11 points to a user who has at least one enabled interrupt condition.

Exit conditions: The user's UTABLE and virtual page 0 have been altered to reflect the appropriate interrupts.

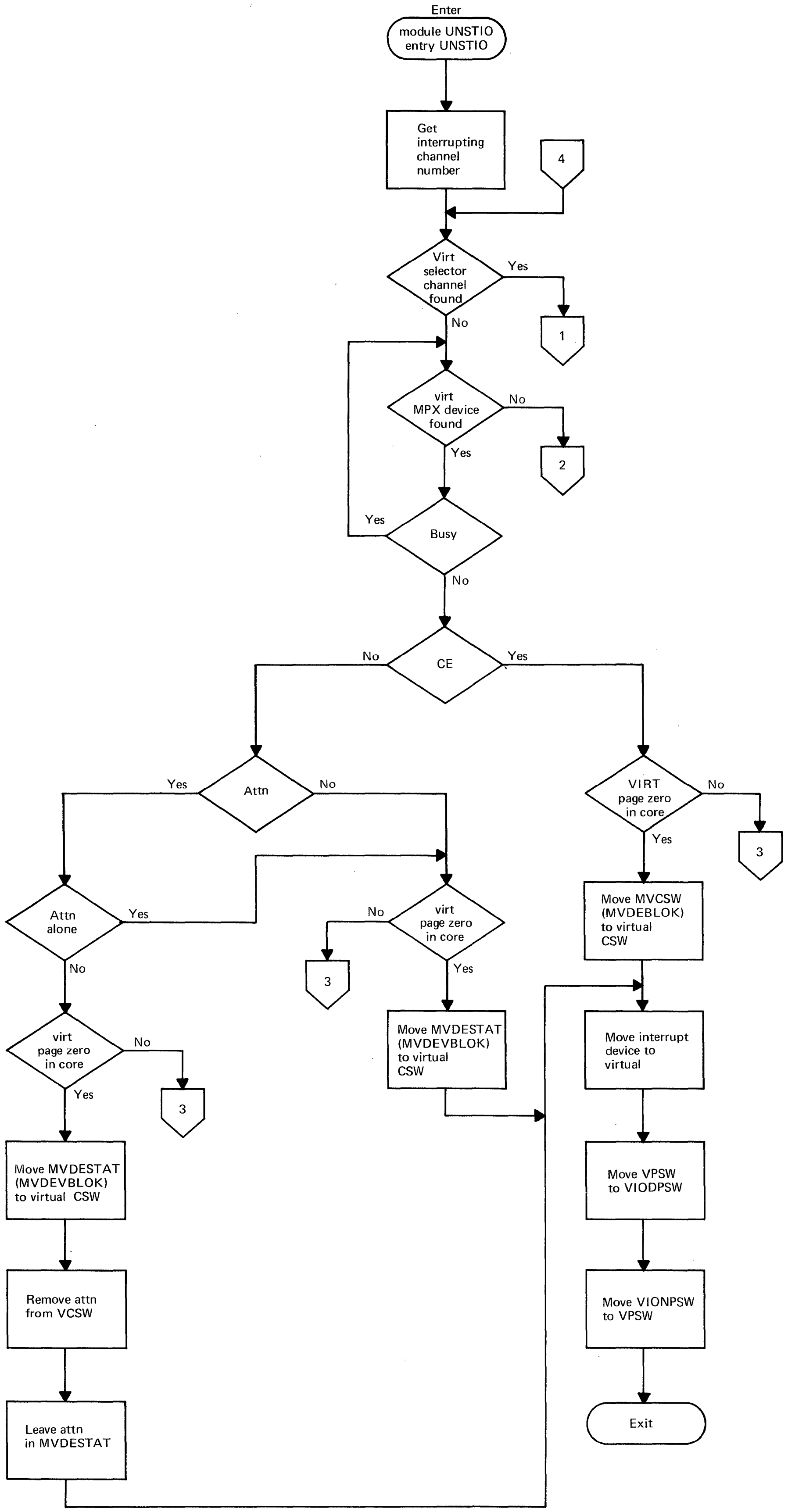


Figure 45. CP-67 UNSTIO (1 of 2)

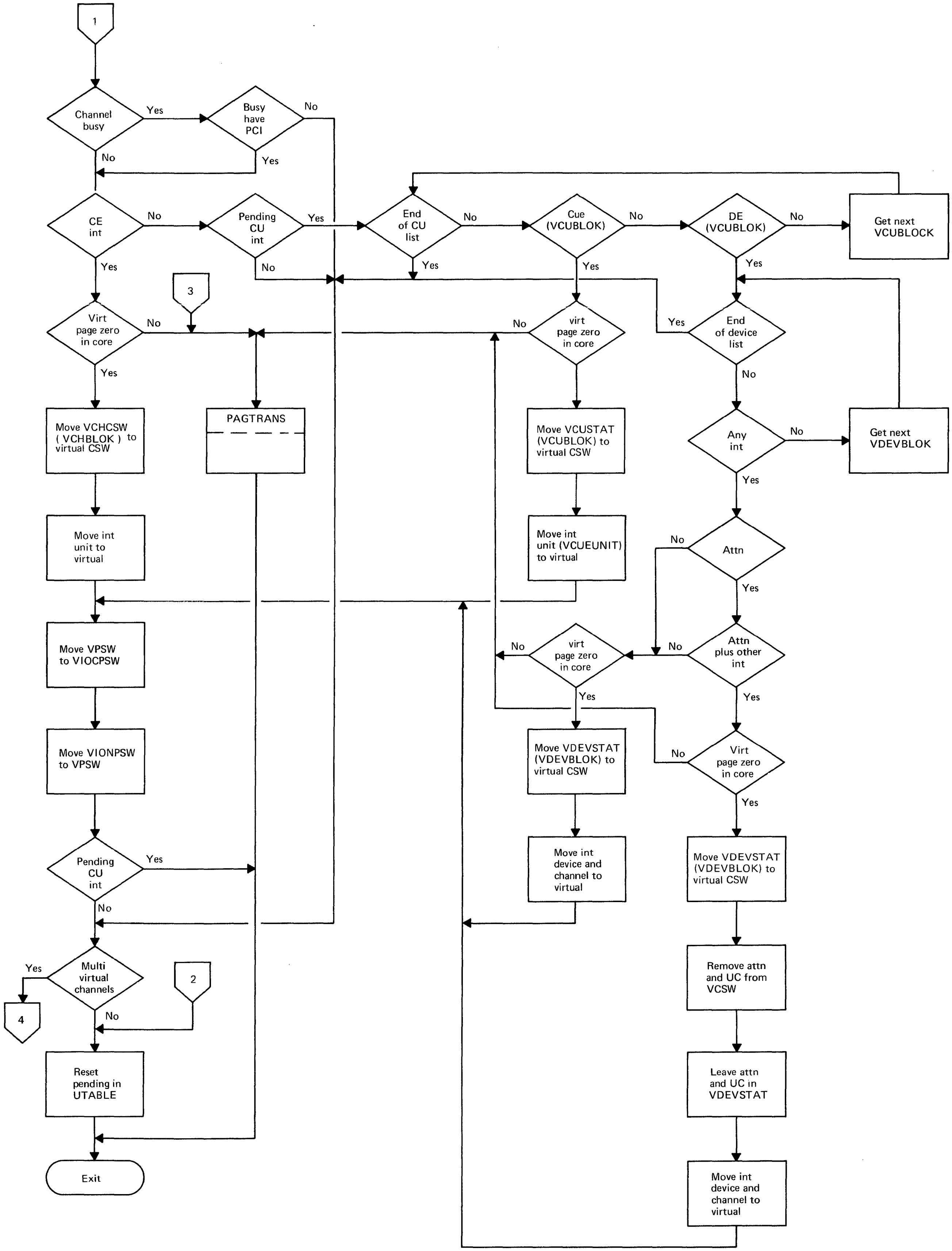


Figure 45. CP-67 UNSTIO (2 of 2)

UNTRANS

Module name: UNTRANS

Entry points: UNTRANS, FREECCW

Purpose: The module computes from the hardware CSW the virtual CSW to be reflected to the user. The real CCW's are released to free storage.

Registers 0-12 are saved upon entry to this module.

| Entry point: UNTRANS (BALR)

Entry conditions: The GPR6 is pointing to the VCHBLOK which contains the CSW.

Exit condition: The VCHBLOK contains the translated CSW.

Entry point: FREECCW

Entry conditions: GPR8 points to the user's VDEVBLOK, and GPR9 points to the IOTASK block.

Exit conditions: All I/O pages are unlocked, the RHA data is relocated, and the real CCW lists have been released.

USERLKUP

Module name: USERLKUP

Entry point: USERLKUP

Purpose: To find the entry in the U.DIRECT file for a specified userid.

Registers 0-5 are saved upon entry to this module.

Entry conditions:

GPR 1 points to an eight-byte userid.

GPR 2 points to a buffer of size greater than or equal to UFDENTLN (the size of the UFDENT DSECT).

Exit conditions:

Condition code	nonzero: Userid found in directory. Caller's buffer contains a copy of the user file directory entry (DSECT UFDENT).
Condition code	zero: Userid not found.

USEROFF

Module name: USEROFF

Entry points: USEROFF, ADSET, ADSETOUT, RELEASE, RUNRET

Purpose: The USEROFF module handles the details of logging a user off the system.

Entry point: USEROFF - Deletes the virtual machine from the system.

Entry conditions: GPR 11 contains UTABLE address.

Exit conditions: GPR 11 set to zero.

Entry point: ADSET - initiate the logout sequence.

Entry conditions: GPR 11 points to the user's UTABLE. GPR 2 is set to 1 if the logoff is forced (line error or hangup). GPR 2 is set to 2 if called by KILL or SHUTDOWN. Otherwise, GPR 2 is set to zero.

Registers 0-10 are saved upon entry.

Exit conditions: INLOGOFF bit is set in VMSTATUS and normal exit taken.

Entry Point: ADSETOUT - log user off the system.

Entry conditions: User has no outstanding I/O operations. GPR 11 points to the user's UTABLE.

Registers 0-10 are saved upon entry.

Exit conditions: GPR 11 contains zero.

Entry point: RELEASE - detach a nonshared input-output device.

Entry conditions: GPR 11 points to the user UTABLE. GPR 2 points to the RDEVBLOK of the device to be detached. If the device is a tape unit, the volume mounted is rewound and unloaded. If the device is a dedicated multiplexer unit, the selector channel real I/O blocks are returned to free storage, and the original MRDEBLOK is restored to the list.

Exit conditions: None

Entry point: RUNRET - interrupt return address for an IOTASK that rewinds and unloads a tape after being detached.

Entry conditions: GPR 9 points to IOTASK. GPR 10 points to CSW.

Registers 0-15 are saved upon entry.

Exit conditions: None

VIOEXEC

Module name: VIOEXEC

Entry points: VIOEXEC, VIRA

Purpose: VIOEXEC is responsible for intercepting virtual input-output commands and determining how they will be handled. It performs operations required for handling selector channel requests and passes multiplexer requests onto MVIOEXEC.

Entry point: VIOEXEC

Entry conditions: GPR 4 points to the first half of the input-output instruction which caused entry to VIOEXEC. GPR 5 points to the second half. The virtual CAW will point to the virtual CCW list to be executed.

Registers 0-10 are saved upon entry.

Exit conditions: Goes to DSPTCHB. The condition code in the virtual PSW is set as follows:

0 - I/O initiated or performed
1 - CSW stored
2 - device busy
3 - device not operational

Entry point: VIRA - generalized interrupt return address for IOTASK performing user-dedicated I/O operations; sets condition and stacks a virtual pending interrupt.

Entry conditions: GPR 9 points to IOTASK. GPR 10 points to CSW.

Registers 0-15 are saved upon entry.

VSERSCH

Module name: VSERSCH

Entry point: VSERSCH

Purpose: Searches RDEVBLOK's for a given volume serial number.

Registers 0-11 are saved upon entry to this module.

Entry point: VSERSCH

Entry conditions: GPR 1 points to a six-byte field containing the volume serial label desired.

Exit conditions: GPR 1 points to the desired RDEVBLOK. If the given label is not currently recognized by the system, this register will be zero.

Module name: WRTCONS

Entry points: WRTCONS, PRIORITY, OPTIME

Purpose: This module allows each remote terminal to be used for output as though it were an operator's 1052 console. It will create and stack a CCW package for a specific terminal (with a priority status, if requested).

Entry point: WRTCONS

Entry conditions: GPR 0 contains the byte count of the output message (must be nonzero). GPR 1 contains the starting address of the output message (see DFRET note below). GPR 2 contains 0 or parameters as follows: NORET specifies that no return is to be made on completion of the operation, that is, GPR 3 (below) is not set up. ALARM specifies that the audible alarm is to be given, if available, at the completion of the operation. DFRET causes the output buffer to be automatically returned to free storage at the completion of the operation. (Note: In this case, the data in GPR's 3 and 1 must be appropriate for return to the FRET routine; that is, GPR 1 is on a doubleword boundary, and GPR 3 contains the number of doublewords to return to free storage.) OPERATOR specifies that the message is to go to the operator's terminal. GPR 11 need not be established for this call. NOAUTO specifies that the message is to be written without an automatic carriage return following the message. GPR 3 contains the return address, if NORET was not specified. It contains the number of doublewords to be returned to free storage if NORET and DFRET were specified.

Registers 0-4 are saved upon entry.

Exit conditions: An immediate return is made from WRTCONS before the operation is completed. All registers are saved here. Upon completion of the operation, GPR 2 contains an error code, if any. Return (if NORET was not specified) is to the location specified by GPR 3.

Entry point: PRIORITY

Entry conditions: Same as for WRTCONS

Exit conditions: Same as for WRTCONS, except that the console write is requested to be queued ahead of any other currently stacked I/O for that terminal.

Entry point: OPTIME - writes time of day to operator's terminal.

Entry conditions: None

Registers 0-15 are saved upon entry.

Exit conditions: None

The CP-67 utility modules, all of which are stand-alone except for VDUMP, are provided as follows:

BUZZARD - upon abnormal system failure, (1) saves accounting cards for billing, and (2) saves starting DASD address of spooled printer, punch and virtual card reader files.

DIRECT - writes the user file directories onto SYSRES volume; allocates space on DASD devices used as system device.

FORMAT - formats DASD devices used as system device.

SAVESYS - writes a pageable core image copy of an operating system, such as CMS, which is run in a virtual machine under CP-67; enables the saved operating system to be IPL'ed by name.

VDUMP - runs in a virtual machine to retrieve any system ABEND dumps from the system disk.

BUZZARD

Utility module name: BUZZARD

Entry point: BUZZARD

Purpose: The BUZZARD module is responsible for performing three distinct functions. It saves the accounting records on disk after an abnormal system failure. The system LOGMSG is also saved so that when the system is re-IPL'ed with the WARM start option, this message will appear on the user's console at login time. Moreover, if there were any files in the spooled area of the system waiting to be printed or punched, it saves a table of the starting DASD address of such files and writes out the table on CP-67 system residence disk, cylinder 202. It also saves virtual card reader files in the same way. When the system is IPL'ed the next time and the WARM start option is specified, the spooled output is continued.

Entry point: BUZZARD

Entry conditions: None

Exit conditions: Prints completion message on 1052 console and goes into wait.

DIRECT

Utility module name: DIRECT

Entry point: DIRECT

Purpose: The DIRECT program writes the user file directories onto the system residence volume and allocates space on that volume and other volumes which are to be used for permanent file residence, paging, and spooling.

Entry point: DIRECT

Entry conditions: Entered from stand-alone loader. No other entry conditions.

Exit conditions: Sets WAIT state PSW after completion of all allocation and directory creation activities, and termination message to operator console.

FORMAT

Utility module name: FORMAT

Entry points: FORMAT

Purpose: To format any DASD device that CP-67 uses for a system device (that is, for residence, paging or spooling). Currently those devices are 2311, 2314, 2303, and 2301.

Entry point: FORMAT

Entry conditions: All required variable data is collected by the program interrogating the operator for (1) device type, (2) device address, (3) volume label, (4) start address (optional), and (5) end address of cylinders or tracks to be formatted.

Exit conditions: Program prints FORMAT ENDS.

Utility module name: SAVESYS

Entry point: SAVESYS

Purpose: This module is used to write a pageable core image copy of an operating system such as CMS, which is run in a virtual machine under CP-67. The operating system (such as CMS) is IPL'ed on a bare machine, with an appropriate address stop set. Then the program SAVESYS is IPL'ed from the card reader. The control card describes the core limits to be saved and the device and cylinder address of where to save it. (see Operator's Guide for procedure.)

The module SYSTEM has to be set up to reflect the page numbers and cylinder addresses where the core image was saved. This allows the user to IPL the virtual system by name, such as

IPL CMS

The advantage of IPL'ing by name is in speed since it requires less I/O and paging than normal IPL. Moreover, in order to share CMS system pages among users, it is necessary to IPL by name.

VDUMP

Utility module name: VDUMP

Entry point: VDUMP

Purpose: This module runs in a CMS virtual machine specially configured to retrieve the system ABEND dumps from disk. Only the user specified for a SYSDUMP in the SYSGEN macro can operate this program. That user's virtual machine must have defined in the CP-67 directory a special spool file reader defined as:

```
UNIT 0F1,RPRT
```

as well as a standard CMS machine. VDUMP will reside on that user's P-disk. The program uses the special reader (0F1) to access any system dumps. The dump input is then formatted and printed on the CMS printer (00E), which is spooled. As VDUMP proceeds, it prints a message indicating each 10,000 bytes of core printed as:

```
DUMPING STORAGE LOCATION xxxx
```

Upon completion, VDUMP prints END OF DUMP and closes the virtual printer.

APPENDIX A: SAVE AREAS

Register 13 normally points to a 96-byte save area. The first 12 bytes are reserved for use by the SVC handler for keeping linkage information. Modules normally use the next 12 to 16 words for saving the registers of the calling routine (the ENTER macro generates an STM of the specified register(s) into an area whose beginning is displaced 12 bytes off register 13). The remaining bytes are optionally used as a work area. The first word of an active save area will contain the interrupt return address in the calling routine. The second word contains the caller's register 12, and the third word the caller's register 13. Very seldom are more than registers 0 through 11 saved since (1) 14 and 15 are normally work registers, and (2) 12 and 13 have already been saved by the SVC handler. Inactive save areas will contain a pointer to the next inactive area in the first word of the save area. A word in the SVC handler points to the first available (inactive) save area.

Note: In OS, register 13 normally points to a 20-word save area for use by the called routine. If a called routine wishes to call, it will provide core or dynamically obtain core for its called routine's save area. In CP-67, register 13 points to a save area for the currently active routine, containing the saved registers of the calling routine and the necessary linkage information to return. The maintaining of linkage information and chains for active and available save areas is all done by the SVC handler. There is one exception to this rule: in CFSMAIN, the routine obtains its own, extra large save area, and it temporarily replaces the normal save area in the chain with the extra large one.

APPENDIX B: REGISTER USAGE

Register	
0	variable (many times count of doublewords for FREE or FRET linkage)
1	variable (many times pointer to temporary storage obtained from FREE)
2	CALL macro parameters if PARM is used
3-5	variable
6	variable (I/O routines use commonly as channel block pointer)
7	variable (I/O routines use commonly as control block pointer)
8	variable (I/O routines use commonly as device block pointer)
9-10	variable
11	pointer to the user's status table (UTABLE) for the user CP is currently working on
12	base
13	save area pointer
14	variable (some use as BAL, BAS, etc. within particular modules)
15	variable (address of entry point of currently active module or last called module, set by CALL macro)

Registers 0 and 1 are commonly used to pass arguments to subroutines. Registers 14 and 15 are not preserved over a subroutine call and therefore should not be used for any but very temporary use.

APPENDIX C: CORE LAYOUT

The following items are of particular importance in debugging CP-67. For a complete description of lower core see the listing of EQU67 COPY file from the CPMAX macro library. (EQU67 is listed in "CP-67 Equate Package - EQU67" in Section 3: Programming Conventions of this manual.)

See Figure 46 for a diagram of real low core.

Hexadecimal
Address
----

0 Eight-byte PSW restart

E External old PSW interrupt code

10 SVC old PSW interrupt code

12 Program old PSW interrupt code

14 Machine check old PSW interrupt code

16 I/O old PSW interrupt code

160 UTABLE address of the currently active or last run user

340 Address of CPSYM module. CPSYM contains a twelve-byte entry for each CP module, an eight-byte EBCDIC name, and a four-byte ADCON.

CPEND Address variable depending on system, represents highest address of permanently resident CP code. Beginning on the first 32-byte aligned boundary following CPEND is the CORTABLE, one 16-byte entry for each 4K page in the machine. Following the CORTABLE, beginning on the first following 32-byte boundary are the initial 100 96-byte save areas.

<table>
<tr><td>000</td><td colspan="8">IPLPSW</td></tr>
<tr><td>008</td><td colspan="6">IPLCCW</td><td colspan="2">EXT. INT. CODE</td></tr>
<tr><td>010</td><td colspan="2">SVC INT. CODE</td><td colspan="2">PROG. INT. CODE</td><td colspan="2">MCK. INT. CODE</td><td colspan="2">I/O INT. CODE</td></tr>
<tr><td>018</td><td colspan="8">OLD PSW'S</td></tr>
<tr><td>040</td><td colspan="8">CSW</td></tr>
<tr><td>048</td><td colspan="8">CAW</td></tr>
<tr><td>050</td><td colspan="8">TIMER</td></tr>
<tr><td>058</td><td colspan="8">NEW PSW'S</td></tr>
<tr><td>080</td><td colspan="8">SCANOUT</td></tr>
<tr><td>160</td><td colspan="4">RUNUSER</td><td>CPSTATUS</td><td>MONTHS</td><td>DAYS</td><td>YEARS</td></tr>
<tr><td>168</td><td>HOURS</td><td>MINUTES</td><td>SECONDS</td><td></td><td colspan="4">STARTIM</td></tr>
<tr><td>170</td><td colspan="4">STARTIM</td><td colspan="4">BINTIME</td></tr>
<tr><td>178</td><td colspan="8">DISPSW</td></tr>
<tr><td>180</td><td colspan="4">ASYSWRM</td><td colspan="4">ASYSINF</td></tr>
<tr><td>188</td><td colspan="4">ASYSCNSL</td><td colspan="4">CPID</td></tr>
<tr><td>190</td><td colspan="4">ARMXST</td><td colspan="4">ARDEVT</td></tr>
<tr><td>198</td><td colspan="4">AZVOL</td><td colspan="4">APRINT</td></tr>
<tr><td>1A0</td><td colspan="4">APUNCH</td><td colspan="4">AREADERS</td></tr>
<tr><td>1A8</td><td colspan="4">AMREAL</td><td colspan="4">ARCHSTRT</td></tr>
</table>

Figure 46. CP-67 Real Low Core (1 of 2)

1B0	CPUTAB		
1E0	TEMPSAVE		
220	BALRSAVE		
270	DISPAGWK		
290		DSCRO	
298	KALG	LOCKOUNT	MAXLOCK
350	CPTIME	PROBTIME	
358	WAITTIME	OVERHEAD	
360	WAITIDLE	WTPAGE	
368	WTUSR	WTUSRA	
370	KPGEX	PGREAD	
378	PGSWAP	QCOUNT	
380	INSTWRD1	INSTWRD2	
388	INSTWRD3	INSTWRD4	
390	STATUSER		

Figure 46. CP-67 Real Low Core (2 of 2)

APPENDIX D: CP-67 ABEND

The first occurrence to check for in an ABEND dump is an SVC 0 (a halfword zero in the SVC interrupt code at location hex 10), and supervisor state in the SVC old PSW (PSW at hex location 20 does not contain problem state bit, bit 01, byte 1). There are two possible SVC 0's which should be eliminated before proceeding any further: (1) an SVC 0 issued by the machine check handler when there has been a machine check while in supervisor state, and (2) the SVC 0 issued by the command handler in response to the operator command D_U_M_P.

If an SVC 0 is not found, the second possibility to check for is a program interrupt in supervisory mode. The program old PSW (hex address 28) will not contain the problem state bit.

The third possibility is that the operator has pushed the STOP and PSW RESTART buttons on the CPU. In this case there should be additional information provided by the operator on what CP-67 was doing to force the operator to take an ABEND dump.

APPENDIX E: CP-67 MEASUREMENT HOOKS

Low Core (defined in EQU67)

RUNUSER - running user
MONTHS, DAYS, YEARS, HOURS, MINUTES, SECONDS - current date and time accurate to one second
STARTIM - system IPL date and time
BINTIME - binary timer; one hour elapsed time
RUNINTIM - binary timer; one second elapsed time
LOCKOUNT - number of "locked" pages
MAXLOCK - maximum number of "locked" pages
CPTIME - CPU time in supervisor state
PROBTIME - CPU time in problem state
WAITTIME - CPU time in wait state
OVERHEAD - supervisor time not charged to users
WAITIDLE - wait time system idle
WTPAGE - wait time while paging
KPGEX - count of paging exceptions
PGREAD - pages read in
PGSWAP - pages written out
QCOUNT - pages stolen from in Q users
INSTWRD1 - installation counter
INSTWRD2 - installation counter
INSTWRD3 - installation counter
INSTWRD4 - installation counter

Low Core (defined in STAT)

STATUEXT - user external interrupts
STATUSVC - user SVC interrupts
STATUPGM - user program interrupts
STATUIOI - user I/O interrupts
STATSSK - user SSK instructions
STATISK - user ISK instructions
STATSSM - user SSM instructions
STATLPSW - user LPSW instructions
STATDIAG - user DIAG instructions
STATDDSK - user diagnose disk I/O instructions
STATSIO - user SIO instructions
STATTIO - user TIO instructions
STATHIO - user HIO instructions
STATTCH - user TCH instructions
STATWRD - virtual 67 user WRD instructions
STATSTMC - virtual 67 user STMC instructions
STATLRA - virtual 67 user LRA instructions
STATLMC - virtual 67 user LMC instructions
STATDSP - count of calls to CKUSR in DISPATCH

User Data (defined in UTABLE)

TIMEUSED - total CPU time user
TIMEON - login time (MMDDYYHHMMSS)
PRIORIT - priority to enter Q
VTOTTIME - virtual CPU time used

UPIOCNT	-	pages read while in queue
UVIOCNT	-	virtual SIO count
VMUSER1	-	installation counter
VMUSER2	-	installation counter
VMUSER3	-	installation counter
VMUSER4	-	installation counter
VMSSIO	-	selector channel SIO
VMPNCH	-	spool cards punched
VMLINS	-	spool lines printed
VMCRDS	-	spool cards read
VMPGRD	-	pages read

DISPATCH

NUMUSERS	-	current logged in user count

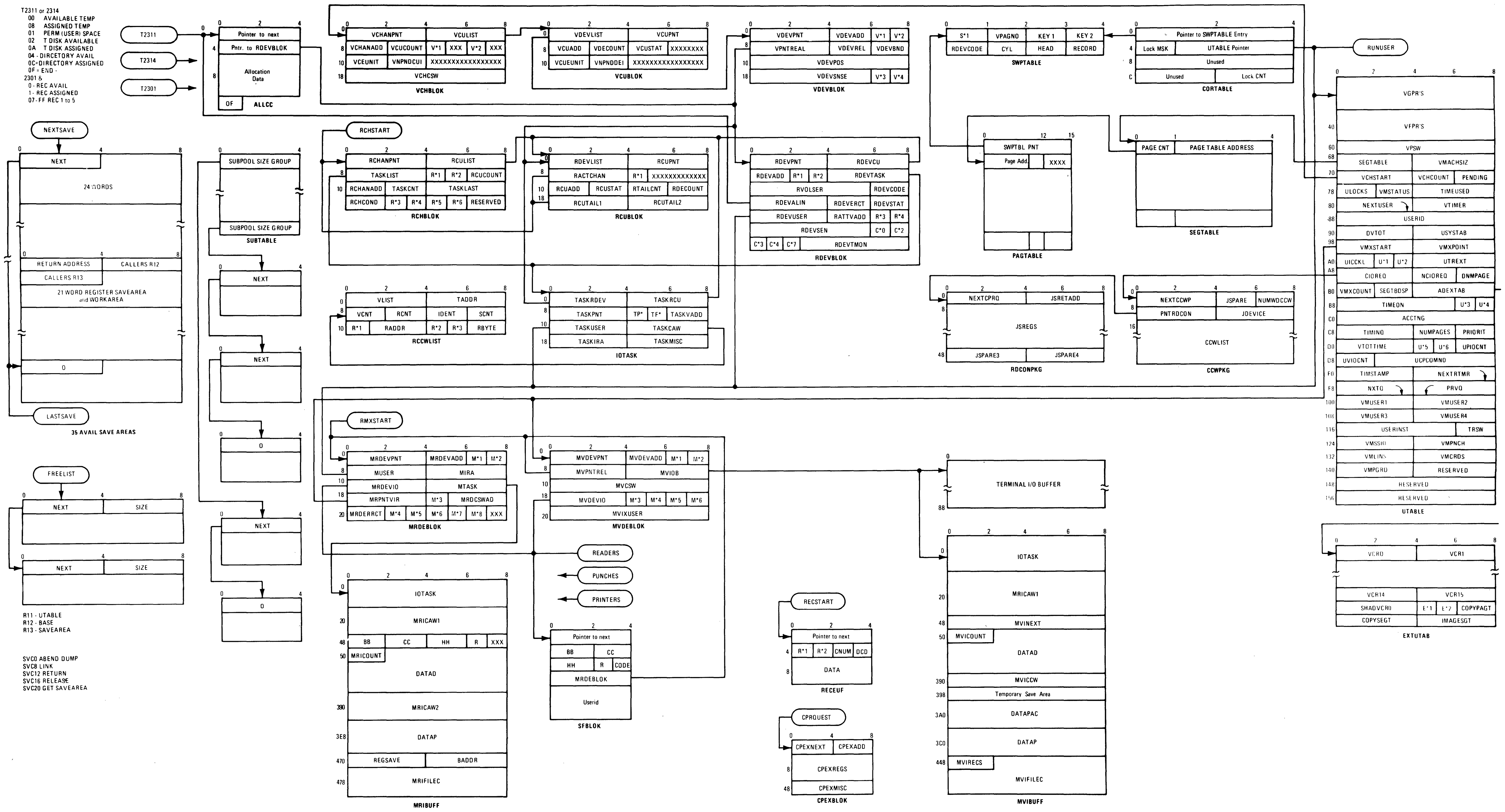
MVIOEXEC

VMIO	-	total user MPX SIO count

QUEVIO

VIOCOUNT	-	total user SIO count
RIOCOUNT	-	total CP SIO count

APPENDIX F: CP-67 CONTROL BLOCKS



APPENDIX G:
Alphabetical Listing of System Modules by Entry Point

ABEND	CFSPRV
ACCTON	ACCTON
ACNTIME	ACNTIME
ACNTOFF	ACNTOFF
ADSET	USEROFF
ADSETOUT	USEROFF
ATTACH	CFSTACH
AUTOLOGON	LOGON
BINDEC	CONVRT
BINHEX	CONVRT
BREAK	CFSMAIN
BRKRD	CFSMAIN
BRKWR	CFSMAIN
CCWTRANS	CCWTRANS
CFSACNT	CFSPRV
CFSDIR	CFSPRV
CFSIPL	CFSIPL
CHFREE	QUEVIO
CHKCUACT	CHKCUACT
CHKPT	CHKPT
CLINK	CFSTACH
CLOSE	CFSSPL
COMENTRY	CFSMAIN
CONSINT	CONSINT
CORTENT	PAGTRANS
CORUSER	PAGTRANS
CPCORE	CPCORE
CPFCLOSE	CPFILE
CPFDCLOS	CPFILE
CPFDLKUP	CPFILE
CPFOPENR	CPFILE
CPFOPENW	CPFILE
CPFREAD	CPFILE
CPIENT	CONSINT
CPINIT	CPINIT
CPSTACK	CPSTACK
CPSYM	CPSYM
CP6IRA	CCWTRANS
DATETIME	CONVRT
DCP	CFSDBG
DECBIN	CONVRT
DEDICATE	DEDICATE
DETACH	CFSTACH
DEVOFF	ACNTOFF
DEVSCAN	RDSCAN
DIAGDSK	DIAGDSK
DIAL	DIAL
DISABLE	CFSPRV
DISACT	DISPATCH
DISCONN	CFSCOM
DISDRQ	DISPATCH
DISIO	DISPATCH
DISPATCH	DISPATCH
DISPLAY	CFSDBG

DMCP	CFSDBG
DRAIN	CFSSPL
DRMWAIT	PAGTRANS
DSKDUMP	DSKDUMP
DSPTCHA	DISPATCH
DSPTCHB	DISPATCH
DSPTCHC	DISPATCH
DUMP	CFSDBG
ENABLE	CFSPRV
EXTEND	EXTEND
EXTINT	PSA
FINDIO	IOERROR
FINDLOG	IOERROR
FINDMC	IOERROR
FMTILOG	IOERROR
FMTLOG	IOERROR
FMTLOGI	IOERROR
FMTLOGM	IOERROR
FMTMLOG	IOERROR
FORCE	PLACE
FORCEA	PLACE
FPCONV	CONVRT
FREE	FREE
FREECCW	USERLKUP
FREEPST	CFSDBG
FRET	FREE
FRETPST	CFSDBG
FRETR	FREE
HEXBIN	CONVRT
IDENTIFY	CONSINT
IOERROR	IOERROR
IOINT	IOINT
IOISTVCU	IOINT
IOISTVDE	IOINT
IPL	IPL
IPLSAVE	CFSIPL
KILL	CFSPRV
LINK	LINK
LINKSCAN	RDSCAN
LOCKC	CFSPRV
LOGFILES	LOGFILES
LOGIN	LOGIN
LOGOUT	CFSCOM
LOGRETN	IOERROR
MCHEKINT	PSA
MCKERR	IOERROR
MRIOEXEC	MRIOEXEC
MSG	CFSCOM
MVICLCR	MVIOEXEC
MVICLPN	MVIOEXEC
MVICLPR	MVIOEXEC
MVIOEXEC	MVIOEXEC
MVIPRINT	PACK
OFFENT	CONSINT
OFFHANG	CONSINT
OPMSG	LOGON
OPTIME	WRTCONS
PACK	PACK

PAGEGET	PAGEGET
PAGERLE	PAGEGET
PAGFREE	PAGTRANS
PAGFRET	PAGTR
PAGOUT	PAGTR
PAGSHARE	PAGTR
PAGTRANS	PAGTRANS
PAGUNLOK	PAGTRANS
PLACE	PLACE
PLACINIT	PLACE
PREPLINE	CONSINT
PRIMSG	TMPSPACE
PRIORITY	WRTCONS
PRIRA	MRIOEXEC
PRIVLGED	PRIVLGED
PROGINT	PROGINT
PRTINIT	PLACE
PUIRA	MRIOEXEC
PURGE	CFSSPL
QUERIO	QUEVIO
QUERY	CFSQRY
QUEVIO	QUEVIO
RDCONS	RDCONS
RDSCAN	RDSCAN
READTASK	CPFILE
READY	CFSCOM
RECERROR	IOERROR
RECFREE	RECFREE
RECFRET	RECFREE
REFLECT	PROGINT
RELEASE	USEROFF
REPEAT	CFSSPL
RESINT	RESINT
RESIRA	RESINT
RPUNCH	MRIOEXEC
RTN41ND	CONSINT
RTN41WT	CONSINT
RTN52ND	CONSINT
RTN52WT	CONSINT
RUNITSCAN	SCANUNIT
RUNRET	USEROFF
SAVECP	SAVECP
SCHEDULE	SCHEDULE
SCLOCK	SCHEDULE
SCREDAT	SCREDAT
SET	CFSSET
SHUTDOWN	CFSPRV
SLEEP	CFSCOM
SPACE	CFSSPL
SPOOL	CFSSPL
START	CFSSPL
STCONSIO	STCONSIO
STCP	CFSDBG
STORE	CFSDBG
SVCDUMP	PSA
SVCINIT	PSA
SVCINT	PSA
TERM	CFSSPL

TMPERTN	TMPSPACE
TMPRET	TMPSPACE
TMPSPACE	TMPSPACE
TRACER	TRACER
TRINT	TRACER
T2311	TMPSPACE
UNLOCK	CFSPRV
UNPACK	PACK
UNSTIO	UNSTIO
UNTRANS	UNTRANS
USEROFF	USEROFF
VERROR	IOERROR
VIOEXEC	VIOEXEC
VIRA	VIOEXEC
VSERSCH	VSERSCH
VSMCPIR	CCWTRANS
VUNITSCAN	SCANUNIT
WAITPAGE	PAGTRANS
WNG	CFSCOM
WRITTASK	CPFILE
WRTCONS	WRTCONS
XFER	CFSSPL

Obtaining a Cross-Reference Chart of CP

To obtain a cross-reference chart of CP, run CP nucleus text decks through the OS linkage editor, then run the OS utility LMODMAP. This produces a cross-reference listing of all CP control sections and entry points.

READER'S COMMENT FORM

Control Program-67/Cambridge Monitor System
(CP-67/CMS) Version 3.1 PLM

GY20-0590-1

Please comment on the usefulness and readability of this publication, suggest additions and deletions, and list specific errors and omissions (give page numbers). All comments and suggestions become the property of IBM. If you wish a reply, be sure to include your name and address.

COMMENTS

fold

fold

fold

fold

- Thank you for your cooperation. No postage necessary if mailed in the U.S.A. FOLD ON TWO LINES, STAPLE AND MAIL.

GY20-0590-1

YOUR COMMENTS PLEASE...

Your comments on the other side of this form will help us improve future editions of this publication. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material.

fold

fold

FIRST CLASS
PERMIT NO. 1359
WHITE PLAINS, N. Y.

BUSINESS REPLY MAIL

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY...

IBM Corporation
1133 Westchester Avenue
White Plains, N.Y. 10604

Attention: Technical Publications

fold

fold

IBM®

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
[U.S.A. only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

Control Program-67/Cambridge Monitor System (CP-67/CMS) Version 3.1 PLM Printed in U.S.A. GY20-0590-1

IBM®

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

IBM / Technical Newsletter

Base Publ. No. GY20-0590-1

This Newsletter No. GN20-2502

Date January 24, 1972

Previous Newsletter Nos. None

Control Program – 67/Cambridge Monitor System
(CP-67/CMS) Version 3.1
CP-67 Program Logic Manual
Program Number: 360D-05.2.005

This Technical Newsletter, a part of Version 3, Modification Level 1, of Control Program – 67/Cambridge Monitor System, provides replacement pages for the subject manual. These replacement pages remain in effect for subsequent versions and modifications unless specifically altered. Pages to be inserted and/or removed are listed below.

Pages

1, 2
73, 74
181, 182
207, 208
223, 224
239, 240
339, 340

Minor additions and changes have been made to provide program support information on the IBM 3420 Magnetic Tape Unit.

A vertical rule in the left margin indicates that a change has been made to either text or illustration.

Please file this cover letter at the back of the manual to provide a record of changes.

IBM Corporation, Technical Publications Dept., 1133 Westchester Avenue, White Plains, N. Y. 10604

Printed in U.S.A.

SECTION 1: INTRODUCTION TO CP-67

CP-67 is a Control Program designed for execution on an IBM System/360 Model 67. Its objective is to create an environment in which many users can simultaneously perform work and in which each user can perform his own work under the supervision of the programming system of his choice. It achieves its objective by generating a "virtual computer" for each user and by sharing the resources of the real computer (CPU time, main storage, etc.) among the virtual computers for all users that are concurrently logged into the system.

When a user identifies himself from a terminal, the Control Program "creates" for his personal use a virtual computer from a predefined configuration. (Before the system becomes available to users, the systems administrator defines the configuration of each user's virtual machine. He may define different configurations for different users.) To the user, his virtual computer appears real and he uses it as if it were. The Control Program also provides, as part of the virtual computer, commands that parallel the functions of the buttons and switches on an operator's console. The user issues these commands from his terminal, and, thus, the terminal becomes a pseudo-console for his virtual machine.

After the Control Program has created the virtual computer, the user equips it with the programming system that gives him the desired functional capabilities. He does this by issuing a command from his terminal. CP-67 is designed so that the user can run the programming system (for example, Operating System/360) of his choice on his virtual computer. The user who desires a terminal-oriented, conversational programming system that allows him to directly monitor his work will choose CMS.

MACHINE CONFIGURATION

Devices Supported by CP-67

CP-67 is structured to run on an IBM System/360 Model 67. The minimum machine configuration for CP-67 is:

2067-1 or 2067-2 Processing Unit
Recommended feature:
#4434 Floating Storage Addressing (Model 1 only)

2365	Processor Storage
1052	Printer-Keyboard Model 7
1403	Printer

2540 Card Read Punch
3 2311 Disk Storage Drives or 2314 Direct Access Storage Facility (2 drives minimum)
2400 or 3420 Nine-Track Magnetic Tape Unit, 800 or 1600 bpi
2702 or 2703 Transmission Control or
2701 Data Adapter Unit

Terminals Supported by CP-67 as Machine Operator's Console

1051/1052 Model 1 or Model 2 Data Communication System
Features and Specifications:
Data Set Attachment (#9114)
IBM Line Adapter (#4647)
Receive Interrupt (#6100 or RPQ E27428) required
Transmit Interrupt (#7900 or RPQ E26903) required
Text Time-out Suppression (#9698) required

1056 Card Reader Model 3

2741-1,-2 Communication Terminals
Features and Specifications:
Data Set Attachment (#9114)
Data Set Attachment (#9115)
IBM Line Adapter (#4635, #4647)
Dial-Up (#3255) required
Receive Interrupt (#4708) required
Transmit Interrupt (#7900 or RPQ E40681) required
Print Inhibit (#5501) desirable

Line control for teletypewriter terminals (*) compatible with the IBM Telegraph Terminal Control Type II Adapter (8-level ASCII code at 110 bps).

Transmission Control Units Supported by CP-67

2701 Data Adapter Unit

Terminals	2701 Adapter
8-level ASCII, 110 bps*	7885

2702 Transmission Control

Terminals	Terminal Control Base	Terminal Control	Line Adapter
2741s, 1050	9696 or 7935	4615, 9684, 8200**	3233
8-level ASCII, 110 bps*	9697 or 7935	7912	3233

type for the device), a counter overflow error record is written. This error may represent the failure of a completely different channel program than the first error of this type which was recorded. If the error is neither the first encountered nor a cause of a counter overflow condition, control returns to VIOINT, and the error information is reflected back to the user's virtual machine.

The I/O error record has the following 112-byte format:

```
          ORG    LOGDATA    DEFINE I/O ERROR RECORD
LOGSNSE   DS     CL6        SENSE INFORMATION
LOGCODE   DS     CL1        FIRST ENCOUNTERED OR COUNTER
                            OVERFLOW - TYPE OF ERROR
LOGTYPE   DS     CL1        DEVICE TYPE
LOGVOLID  DS     CL6        VOLID OF DEVICE (IF AVAILABLE)
LOGADDR   DS     CL2        PHYSICAL ADDRESS OF DEVICE
LOGDATE   DS     CL6        DATE AND TIME STAMP OF ERROR
LOGCSW    DS     CL8        CHANNEL STATUS WORD
          DS     CL2        UNUSED
LOGCCWS   DS     9D         FAILING CCW STRING (UP TO NINE
                            DOUBLEWORDS)
LOGSKLOC  DS     1D         LAST SEEK ADDRESS (DASD ONLY)
```

For a 3420 device type (LOGTYPE = X'C4') 24 bytes of sense data are recorded. This is done by preserving the 24 sense bytes in the first 3 double words at LOGCCWS. The remaining 6 double words are used to contain the failing CCW string, up to the last six CCW's only. The LOGSNSE field for a 3420 is not used.

The CCW in the string which failed is flagged with an asterisk in the unused fifth byte.

After the error record is written, the pointer to the next available slot on the CE cylinder is updated. Seven logical records are contained within one 829-byte physical record. Since 15 records may be written on two tracks of a 2314, up to 1050 error records may be written on one cylinder. If the attempt to write the error record fails, it is retried eight times. Upon continued failure, an error message "** IOERROR RECORDING FAILURE ON DEV___" is sent to the operator. If there is no more room on the CE cylinder for error records, the message "**CECYL FULL; I/O ERRORS NOT RECORDED **" is sent to the operator. Errors are not recorded for users with privilege class C in order to prevent the recording of intentional errors produced by CE diagnostics. Recording will be reinitiated after the CE executes the CLEARIO function.

Main Dispatcher and Control Routine - DISPATCH

Entrance: DISPATCH is entered from routines which have completed their processing for a user or cannot continue processing until some other process has been completed. (See Figure 10.1 for DISPATCH module processing.)

Operation: DISPATCH checks for pending interruptions and determines which user is to receive control next.

Routines called: When DISPATCH determines that an I/O interruption is pending, the I/O interruption unstacking routine (UNSTIO) is called. UNSTIO updates the virtual CSW, restores virtual PSW's, and indicates the address of the interrupting device. When UNSTIO processing is completed, DISPATCH attempts to restart the current user, if runnable and if his quantum is not exhausted.

DISPATCH may be entered at 4 locations: DISPATCH, DSPTCHA, DSPTCHB, and DSPTCHC. DISPATCH is the normal entry point used by all routines that are not sure of a user's status. DSPTCHA is entered from routines which have gained control after a program interrupt for a user and have changed the user's PSW. DSPTCHB is similar to DSPTCHA except the PSW is at most changed in its condition code field. DSPTCHC is used by routines which have done some processing for a user but in no way changed his status.

Figures '25-28' illustrate the relationships of routines which process an I/O interrupt returned from a selector channel device.

CP-67 DEVICE CODES

```
********************************************************
*                                                      *
*        CP-67 DEVICE TYPE CODES                       *
*                                                      *
********************************************************
*
TYP1052   EQU   0
TYP1050   EQU   4
TYP2250T EQU    8
TYP2260T EQU    12
TYP2741T EQU    16                MPX/2702   2741
TYP 1052T EQU   20                           1052
TYP2703T EQU    24
TYP2702T EQU    24
TYP2701T EQU    24
TYPTT35T EQU    28                MDL 35 TELETYPE
TYPTTY35 EQU    TYPTT35T
TYPTIMER EQU    44               SIMULATED CHRONOLOG
TYP1403   EQU   48
TYP2540P EQU    52
TYP2540R EQU    60
TYP2671   EQU   64
TYPRMPRT EQU    X'44'            REMOTE PRINTER READER
TYPRMPUN EQU    X'48'            REMOTE PUNCH READER
TYPM20    EQU   96
TYP1800   EQU   100
TYP2311   EQU   128
TYP2314   EQU   132
TYP2302   EQU   136
TYP2321   EQU   140
TYP2301   EQU   144
TYP2303   EQU   148
TYP2250   EQU   180
TYP2260   EQU   184
TYP2400   EQU   192               GENERAL MAG TAPE
TYP2404   EQU   192
TYP2402   EQU   192
TYP2403   EQU   192
TYP3420   EQU   196
TYP7340   EQU   204
TYP2701   EQU   208
TYP2701L EQU    208              L IS A DEDICATED LINE
TYP2702L EQU    208
TYP2703L EQU    208
TYP2700L EQU    208
TYP2702D EQU    212              D IS A DIAL CONNECTED LINE
*
********************************************************
```

```
************************************************************************
*                                                                      *
*        CP-67 EQUIVALENCE AND MACHINE DEFINITION PACKAGE              *
*                                                                      *
************************************************************************
*
*              BITS IN STANDARD PROGRAM STATUS WORD
*
PROBMODE EQU   X'01'          PROBLEM MODE BIT.
WAIT     EQU   X'02'          WAIT BIT.
MCHEK    EQU   X'04'          MACHINE CHECK.
ASCII    EQU   X'08'          ASCII BIT.
*
*              BIT ASSIGNMENTS IN EXTENDED PROGRAM STATUS WORD
*
MODE32   EQU   X'08'          24/32 ADDRESSING MODE BIT.
TRANMODE EQU   X'04'          DYNAMIC TRANSLATION MODE BIT.
IOMASK   EQU   X'02'          OVERALL I/O MASK BIT.
EXTMASK  EQU   X'01'          OVERALL EXTERNAL INTERRUPTION MASK BIT.
*
*              DEFINED BITS IN CHANNEL STATUS WORD
*
ATTN     EQU   X'80'          ATTENTION BIT.
SM       EQU   X'40'          STATUS MODIFIER BIT.
CUE      EQU   X'20'          CONTROL UNIT END BIT.
BUSY     EQU   X'10'          BUSY BIT.
CE       EQU   X'08'          CHANNEL END BIT.
DE       EQU   X'04'          DEVICE END BIT.
UC       EQU   X'02'          UNIT CHECK BIT.
UE       EQU   X'01'          UNIT EXCEPTION BIT.
*
PCI      EQU   X'80'          PROGRAM-CONTROLLED INTERRUPT BIT.
WLR      EQU   X'40'          WRONG-LENGTH-RECORD BIT.
PRGC     EQU   X'20'          CHANNEL PROGRAM CHECK
PRTC     EQU   X'10'          CHANNEL PROTECTION CHECK
*
*              FLAGS DEFINED IN CHANNEL COMMAND WORDS
*
CD       EQU   X'80'          CHAIN DATA FLAG.
CC       EQU   X'40'          CHAIN COMMAND FLAG.
SILI     EQU   X'20'          SUPPRESS INCORRECT LENGTH INDICATOR FLAG.
SKIP     EQU   X'10'          SUPPRESS TRANSFER OF INFORMATION.
PCIF     EQU   X'08'          PROGRAM-CONTROILED-INTERRUPT FLAG.
*
*        FLAGS DEFINED IN FIFTH BYTE OF CCW TO AID CCW TRANSLATION
*
RCXIS    EQU   X'80'          CHECK ISAM INDICATOR
RCSUDO   EQU   X'40'               PSEUDO 2311 INDICATOR
RCUTIC   EQU   X'20'          UNTRANSLATED TIC
RCIO     EQU   X'10'          I/O CCW
RCGEN    EQU   X'08'          CP GENERATED CCW
RCDATA   EQU   X'04'          CP GENERATED CHAIN DATA
RC02     EQU   X'02'               RESERVED FOR FUTURE USE
```

LOGIDATA

LOGIDATA is a description of the format of the error records saved by CP-67 for I/O errors:

```
       0        2        4        6        8
      0+--------+--------+--------+--------+
       |  LOGSNSE                 |LOG |LOG|
       |                          |CODE|TYP|
      8+--------+--------+--------+--------+
       |  LOGVOLID                |LOGADDR |
     16+--------+--------+--------+--------+
       |  LOGDATE                 |unused  |
     24+--------+--------+--------+--------+
       |  LOGCSW                           |
     32+--------+--------+--------+--------+
       |  LOGCCWS                          |
    104+--------+--------+--------+--------+
       |  LOGSKLOC                         |
    112+--------+--------+--------+--------+
```

where:

LOGSNSE contains the six I/O sense bytes. For a 3420 device, this field is unused.

LOGCODE contains the type of I/O or channel error.

LOGTYPE is the type of device upon which the error occurred.

LOGVOLID is the volume serial number of the device upon which the error occurred (if known to CP).

LOGADDR is the channel/unit address of the erring device.

LOGDATE contains the date and time of the error.

LOGCSW contains the channel status word at the time of the error.

LOGCCWS contains the failing CCW string (up to nine CCW's. For a 3420 device, the first 3 double words contain the 24 sense bytes. The remaining 6 double words contain the failing CCW string (up to 6 CCW's).

LOGSKLOC contains the last seek address prior to the failure.

LOGMDATA

LOGMDATA is a description of the format of the error records saved by CP-67 for machine checks:

```
       0        2        4        6        8
      0+--------+--------+--------+--------+
       |  LOGMDATE                |LOGMCODE|
      8+--------+--------+--------+--------+
       |  LOGMCPU                          |
    184+--------+--------+--------+--------+
       |  LOGMPSW                          |
    224+--------+--------+--------+--------+
       |  LOGMGRS        |  LOGMCRS        |
    352+--------+--------+--------+--------+
       |  LOGMFPRS                         |
    384+--------+--------+--------+--------+
```

where:

LOGMDATE contains the date and time of the machine check.

LOGMCODE contains the machine check code.

LOGMCPU contains the CPU logout data.

LOGMPSW contains the five old PSW's at the time of the machine check (external, SVC, program, machine check, and input-output).

LOGMGRS contains the values of the general registers at the time of the failure.

LOGMCRS contains the values of the extended control registers at the time of the failure.

LOGMFPRS contains the values of the floating point registers at the time of the failure.

RDEVBLOK

There is one RDEVBLOK for each real device; its format is as follows:

```
       0        2        4        6        8
       +--------+--------+--------+--------+
     0 |     RDEVPNT     |     RDEVCU      |
       +--------+--------+--------+--------+
     8 |RDEVADD |R*1 |R*2|    RDEVTASK     |
       +--------+--------+--------+--------+
    10 |          RVOLSER         |RDEVCODE|
       +--------+--------+--------+--------+
    18 |    RDEVALLN     |RDEVERCT|RDEVSTAT|
       +--------+--------+--------+--------+
    20 |    RDEVUSER     |RATTVADD|R*3 |R*4|
       +--------+--------+--------+--------+
       |C*0|C*2 |C*3 |C*4|C*7     |RDEVTMON|
       +--------+--------+--------+--------+
       | (CONT) |     RDEVSEN     |        |
       +--------+--------+--------+--------+
       |  RDEVSEN = 24 SENSE BYTES         |
       |  FOR 3420 RDEVBLOK ONLY           |
       +--------+--------+--------+--------+
       |        |    (UNUSED)     |        |
       +--------+--------+--------+--------+
```

where:

RDEVPNT is a pointer to the next device on the chain.

RDEVCU is a pointer to the real control unit.

RDEVADD is the real device address (control unit and device portions only).

R*1 - RDEVTYPE is the device type code.

R*2 - RDECUPTH is the control unit path for this device.

RDEVTASK is a pointer to the attached task block (if active).

RVOLSER is the six-character EBCDIC volume label (if DASD volume and attached to the system).

RDEVCODE is the halfword identification number (index into RDEVTABL).

RDEVALLN is the pointer to the allocation table (if CP-owned).

RDEVERCT is the error count for this device.

RDEVSTAT is the real device status:

RDEVOWND X'80' indicates CP-owned volume (DASD only).
RDEVATTD X'40' indicates dedicated (nonshared) device.
RDEVDED X'20' indicates channel, control unit, and device block dynamically created by DEDICATE.
RDEVSEEK X'08' indicates a seek is in progress.
RDEVPOSD X'04' indicates 2311,2314 comb positioned for next read/write operation.
RDEVSYS X'02' device attached to system.

RDEVUSER is the UTABLE pointer for the current user (for dedicated devices).

RATTVADD is the current user's virtual address (for dedicated devices).

R*3 - RDEVFTR Real device features. Used to describe dedicated communication lines SAD value.

R*4 - RDEVSLEN device sense byte count

C*0 - command reject counter

C*2 - busout parity error counter

C*3 - equipment check counter

C*4 - data check counter

C*7 - seek check (sense bit7, byte0) counter

RDEVTMON is 5 bytes for the attached time for a dedicated device (MMDDYYHHMM)

RDEVSEN contains the sense bytes for the device following a unit check. All devices except 3420 have only 6 sense bytes maximum available. For 3420 devices, the RDEVBLOK is generated with 3 more double words at the end. The RDEVSEN field is considered to be 24 bytes long for 3420's with 6 unused bytes at the end.

VCUBLOK

There is one virtual control unit block for each virtual control unit; its format is as follows:

```
          0        2        4        6        8
          +--------+--------+--------+--------+
       0  |    VDEVLIST     |     VCUPNT      |
          +--------+--------+--------+--------+
       8  | VCUADD |VDECOUNT| VCUSTAT|xxxxxxxx|
          +--------+--------+--------+--------+
      10  |VCUEUNIT|VNPNDDEI|xxxxxxxxxxxxxxxxx|
          +--------+--------+--------+--------+
```

where:

VDEVLIST is the pointer to the virtual devices connected to this control unit.

VCUPNT is the pointer to the next virtual control unit in the chain from the virtual channel.

VCUADD is the virtual control unit address (no channel or device included).

VDECOUNT is the number of virtual devices attached.

VCUSTAT is the status of the virtual control unit; bit definition is the same as the CSW, byte 4; for example, BUSY=X'10'.

VCUEUNIT is the unit for which a control unit end condition, if any, is pending.

VNPNDDEI is the number of pending device interruptions.

VDEVBLOK

There is a virtual device block for each virtual device for each user in the system; its format is as follows:

```
          0        2        4        6        8
          +--------+--------+--------+--------+
       0  |     VDEVPNT     |VDEVADD |V*1  |V*2|
          +--------+--------+--------+--------+
       8  |     VPNTREAL    |VDEVREL |VDEVBND |
          +--------+--------+--------+--------+
      10  |              VDEVPOS              |
          +--------+--------+--------+--------+
      18  |              VDEVSNSE  |V*3  |V*4 |
          +--------+--------+--------+--------+
```

where:

VDEVPNT is the pointer to the next device on the chain from the control unit.

VDEVADD is the virtual device address.

V*1 - VDEVSTAT is the virtual device status; bit definition is the same as the CSW, byte 4; for example, BUSY=X'10', DE=X'04'.

V*2 - VDEVTYPE is the virtual device type code.

VPNTREAL is the real device control block corresponding to this virtual device.

VDEVREL is the relocation factor within the real device for the start of this virtual device (for DASD only).

VDEVBND is the size of this virtual device (DASD only).

VDEVPOS is the current virtual arm position of this device (as BBCCHH).

VDEVSNSE is the virtual device sense information (filled when an error is detected on the virtual device to save the conditions for shared devices.)

If the virtual device type is a dedicated 3420 tape (VDEVTYPE = X'C4') then the function of VDEVSNSE is different. Since the 3420 provides 24 sense bytes, extra space is required to contain them. This is accomplished in the following manner. When a unit check occurs on the 3420, 3 double words are obtained from CP FREE storage. The address of the 3 double work area for the 24 sense bytes is saved in the word located at VDEVSNSE in the VDEVBLOK. Once the sense data is presented to the virtual machine through a virtual sense operation, the 3 double word area is FRETed (in CCWTRAN). The function is repeated for further unit checks on the 3420 device.

V*3 - VDEVFLG contains miscellaneous device status bits:

TEMPDEV	X'01'	indicates a TDSK allocation
READONLY	X'02'	indicates read-only status
VSHARED	X'04'	reserved for future use
VDVENBL	X'08'	virtual 2702 line is enabled
VDVDIAL	X'10'	virtual 2702 line is in use

V*4 - VDEVSLEN is the sense byte count.

UPIOCNT - pages read while in queue
UVIOCNT - virtual SIO count
VMUSER1 - installation counter
VMUSER2 - installation counter
VMUSER3 - installation counter
VMUSER4 - installation counter
VMSSIO - selector channel SIO
VMPNCH - spool cards punched
VMLINS - spool lines printed
VMCRDS - spool cards read
VMPGRD - pages read

DISPATCH

NUMUSERS - current logged in user count

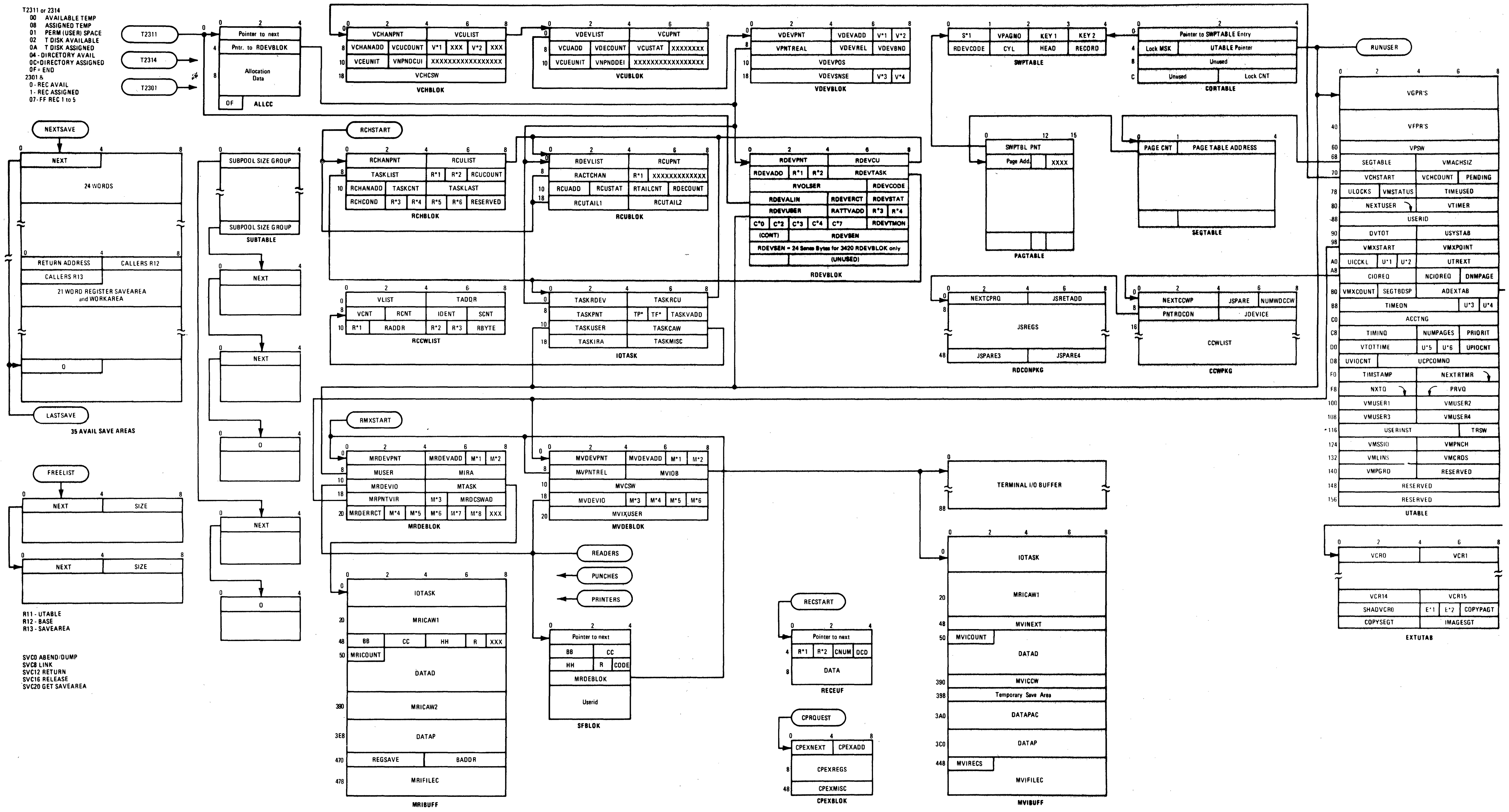
MVIOEXEC

VMIO - total user MPX SIO count

QUEVIO

VIOCOUNT - total user SIO count
RIOCOUNT - total CP SIO count

APPENDIX F: CP-67 CONTROL BLOCKS


T2311 or 2314
00 AVAILABLE TEMP
08 ASSIGNED TEMP
01 PERM (USER) SPACE
02 T DISK AVAILABLE
0A T DISK ASSIGNED
04 - DIRCETORY AVAIL
0C=DIRECTORY ASSIGNED
0F = END
2301 &
0 - REC AVAIL
1 - REC ASSIGNED
07-FF REC 1 to 5
T2311
T2314
T2301
0 2 4
0 Pointer to next
4 Pntr. to RDEVBLOK
8 Allocation Data
0F
ALLCC
NEXTSAVE
0 4 8
NEXT
24 WORDS
0 4 8
RETURN ADDRESS
CALLERS R12
CALLERS R13
21 WORD REGISTER SAVEAREA and WORKAREA
0
LASTSAVE
35 AVAIL SAVE AREAS
FREELIST
0 4 8
NEXT
SIZE
0 4 8
NEXT
SIZE
R11 - UTABLE
R12 - BASE
R13 - SAVEAREA
SVC0 ABEND/DUMP
SVC8 LINK
SVC12 RETURN
SVC16 RELEASE
SVC20 GET SAVEAREA
0 4
SUBPOOL SIZE GROUP
SUBPOOL SIZE GROUP
SUBTABLE
0 4
NEXT
0 4
NEXT
0 4
0
0 4
NEXT
0 4
0
0 2 4 6 8
0 VCHANPNT VCULIST
8 VCHANADD VCUCOUNT V*1 XXX V*2 XXX
10 VCEUNIT VNPNDCUI XXXXXXXXXXXXXXXXX
18 VCHCSW
VCHBLOK
0 2 4 6 8
0 VDEVLIST VCUPNT
8 VCUADD VDECOUNT VCUSTAT XXXXXXXX
10 VCUEUNIT VNPNDDEI XXXXXXXXXXXXXXXXX
VCUBLOK
0 2 4 6 8
0 VDEVPNT VDEVADD V*1 V*2
8 VPNTREAL VDEVREL VDEVBND
10 VDEVPOS
18 VDEVSNSE V*3 V*4
VDEVBLOK
0 1 2 3 4
S*1 VPAGNO KEY 1 KEY 2
RDEVCODE CYL HEAD RECORD
SWPTABLE
0 2 4
0 Pointer to SWPTABLE Entry
4 Lock MSK UTABLE Pointer
8 Unused
C Unused Lock CNT
CORTABLE
RUNUSER
RCHSTART
0 2 4 6 8
RCHANPNT RCULIST
8 TASKLIST R*1 R*2 RCUCOUNT
10 RCHANADD TASKCNT TASKLAST
RCHCOND R*3 R*4 R*5 R*6 RESERVED
RCHBLOK
0 2 4 6 8
0 RDEVLIST RCUPNT
8 RACTCHAN R*1 XXXXXXXXXXXXX
10 RCUADD RCUSTAT RTAILCNT RDECOUNT
18 RCUTAIL1 RCUTAIL2
RCUBLOK
0 2 4 6 8
RDEVPNT RDEVCU
RDEVADD R*1 R*2 RDEVTASK
RVOLSER RDEVCODE
RDEVALIN RDEVERCT RDEVSTAT
RDEVUSER RATTVADD R*3 R*4
C*0 C*2 C*3 C*4 C*7 RDEVTMON
(CONT) RDEVSEN
RDEVSEN = 24 Sense Bytes for 3420 RDEVBLOK only
(UNUSED)
RDEVBLOK
0 12 15
SWPTBL PNT
Page Add. XXXX
PAGTABLE
0 1 4
PAGE CNT PAGE TABLE ADDRESS
SEGTABLE
0 2 4 6 8
0 VLIST TADDR
8 VCNT RCNT IDENT SCNT
10 R*1 RADDR R*2 R*3 RBYTE
RCCWLIST
0 2 4 6 8
0 TASKRDEV TASKRCU
8 TASKPNT TP* TF* TASKVADD
10 TASKUSER TASKCAW
18 TASKIRA TASKMISC
IOTASK
0 2 4 6 8
0 NEXTCPRQ JSRETADD
8 JSREGS
48 JSPARE3 JSPARE4
RDCONPKG
0 2 4 6 8
0 NEXTCCWP JSPARE NUMWDCCW
8 PNTRDCON JDEVICE
16 CCWLIST
CCWPKG
0 2 4 6 8
VGPR'S
40 VFPR'S
60 VPSW
68 SEGTABLE VMACHSIZ
70 VCHSTART VCHCOUNT PENDING
78 ULOCKS VMSTATUS TIMEUSED
80 NEXTUSER VTIMER
88 USERID
90 DVTOT USYSTAB
98 VMXSTART VMXPOINT
A0 UICCKL U*1 U*2 UTREXT
A8 CIOREQ NCIOREQ DNMPAGE
B0 VMXCOUNT SEGTBDSP ADEXTAB
B8 TIMEON U*3 U*4
C0 ACCTNG
C8 TIMINQ NUMPAGES PRIORIT
D0 VTOTTIME U*5 U*6 UPIOCNT
D8 UVIOCNT UCPCOMND
F0 TIMSTAMP NEXTRTMR
F8 NXTQ PRVQ
100 VMUSER1 VMUSER2
108 VMUSER3 VMUSER4
116 USERINST TRSW
124 VMSSIO VMPNCH
132 VMLINS VMCRDS
140 VMPGRD RESERVED
148 RESERVED
156 RESERVED
UTABLE
0 2 4 6 8
VCR0 VCR1
VCR14 VCR15
SHADVCR0 E*1 E*2 COPYPAGT
COPYSEGT IMAGESGT
EXTUTAB
RMXSTART
0 2 4 6 8
0 MRDEVPNT MRDEVADD M*1 M*2
8 MUSER MIRA
10 MRDEVIO MTASK
18 MRPNTVIR M*3 MRDCSWAD
20 MRDERRCT M*4 M*5 M*6 M*7 M*8 XXX
MRDEBLOK
0 2 4 6 8
0 MVDEVPNT MVDEVADD M*1 M*2
8 MVPNTREL MVIOB
10 MVCSW
18 MVDEVIO M*3 M*4 M*5 M*6
20 MVIXUSER
MVDEBLOK
READERS
PUNCHES
PRINTERS
0 2 4 6 8
0 IOTASK
20 MRICAW1
48 BB CC HH R XXX
50 MRICOUNT
DATAD
390 MRICAW2
3E8 DATAP
470 REGSAVE BADDR
478 MRIFILEC
MRIBUFF
0 2 4
Pointer to next
BB CC
HH R CODE
MRDEBLOK
Userid
SFBLOK
RECSTART
0 2 4
Pointer to next
4 R*1 R*2 CNUM DCD
8 DATA
RECEUF
CPRQUEST
0 4 8
CPEXNEXT CPEXADD
8 CPEXREGS
48 CPEXMISC
CPEXBLOK
0
TERMINAL I/O BUFFER
88
0 2 4 6 8
0 IOTASK
20 MRICAW1
48 MVINEXT
50 MVICOUNT
DATAD
390 MVICCW
398 Temporary Save Area
3A0 DATAPAC
3C0 DATAP
448 MVIRECS
MVIFILEC
MVIBUFF