SIMULATION OF IBM/370 INPUT/OUTPUT

by

Antonio C. Gellineau

SB, Massachusetts Institute of Technology
(1974)

SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENT FOR THE DEGREE OF

MASTER OF SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February, 1976

Signature of Author .....
Department of Electrical Engineering and
Computer Science, January 21, 1976

Certified by .....
Thesis Supervisor

Accepted by .....
Chairman, Departmental Committee on Graduate Students

SIMULATION OF IBM/370 INPUT/OUTPUT

by

ANTONIO C. GELLINEAU

Submitted to the Department of Electrical Engineering and Computer Science on January 20, 1976 in partial fulfillment of the requirement for the degree of Master of Science.

ABSTRACT

Simulators are useful in allowing user to develop or investigate operating systems programs which they would normally be unable to run due to lack of available equipment or privileged instruction restrictions. Through simulation of I/O devices and I/O instructions the user programs can include channel programs to the devices as if the equipment were actually attached to the system he is using. Having the privileged I/O related instructions, which are more than one third of all priviledged instructions, at his disposal, gives the user much greater flexibility in the study of operating systems programs.

This document describes simulation of the input/output system for an IBM/370 type machine. It is part of a simulator of a complete machine now being used for running student programs. The I/O simulator is software simulating S/370 hardware done on a S/370 machine.

THESIS SUPERVISOR: Stuart E. Madnick
TITLE: Professor of Management Science

ACKNOWLEDGMENTS

I would like to express my sincere appreciation to Professor Stuart Madnick for his patience and encouragement, and his suggestion that I work on this project.

To my loving wife and typist, Jo-Ann, I give special thanks for standing by me this past tough year.

Finally,to my son, Antonio C. Gellineau Jr.,with whom I've had so little time to spend,I hope that some day you will understand my efforts and surpass them in every way.

Table of contents

List of Figures

1. Introduction

A simulator provides a system that accurately reflects the behavior of all or part of the system being simulated that is necessary for the purpose of investigation and teaching. A simulator is very useful in enabling evaluation before allocation of resources (ie. money). Major design tradeoffs and performance can be evaluated before obtaining or building costly hardware. A simulator also allows a programmer to get at the "bare machine" and run operating systems programs. An excellent example is provided by the simulator described in this paper.

M.I.T. Sloan School's FOS (Family of Operating Systems) Group is in the process of completing a portion of an IBM/370 type simulator. The simulator can handle the IBM/370 machine instructions, has multiple CPU's, has a debugging package and handles all types of interrupts. This paper describes an I/O system design and implemented by the author to provide the simulator with a general facility for handling input/output functions. A fair amount of knowledge about I/O is assumed and a review of the pertinent sections in S/370 Principles of Operation under the main heading entitled "Input/Output Operations" may be helpful.

1.1 Previous Simulators and Simulation Techniques

Two previous simulators written to simulate IBM Systems are SIM360 and TRIBBLE. The former written in PL/1 to simulate IBM System/360 for running student programs and software development, and the latter written in Fortan IV to simulate IBM System/

370. SIM360 operates under a batch enviornment and TRIBBLE is designed to operate interactively. TRIBBLE is the more powerful simulator as it simulates the more powerful machine. The I/O simulator follows the method of handling I/O used in TRIBBLE.

Virtual machines, which give each user the appearance of having his own machine is one approach to having the programmer work with a bare machine. But, until 1972, when IBM VM/370 was announced, one needed to have an expensive IBM/370 model 67 available. But even the VM/370, which operates on any IBM/370 with Dynamic Address Translation, does not accurately reflect the timing and behavior of the simulated computer in the area of I/O operations and priviledged instructions. This is unfortunate since this area is the focus of interset in operating systems programs.

Emulators are another approach to simulation. These are specific hardware units built into a computer which causes the system to accept certain software programs and routines and appear as if it were another system, such as 7094 software running on an IBM/360 computer without translation. The restriction of specific hardware designed for specific systems as compared to a PL/1 simulator is clear.

Some of the advantages to be gained from a simulator are as follows:

--All privileged features may be used directly

--Sophisticated debugging and program statistics recording features may be provided

--An arbitrary configuration may be simulated
(number and types of devices, their interconnections, etc.)

--May be as accurate as necessary

--Is easily modified (editing source file rather than rewiring).

2. Users' View

2.1 Overview of I/O System

The I/O system will handle all input/output operations. The operations of the channel, subchannel, control unit, and I/O devices are simulated. Facilities for requesting and accepting of I/O interrupt are provided and device timing characteristics are also accurately reflected. Presently the system is capable of handling IBM 1403 type printers and IBM 3505 type card readers. Details on how to implement a direct access storage device (an IBM 2305 type drum with rotational position sensing) is given in appendix A.

The overall strategy is to have the I/O system structured as general as possible. To this end a mechanism, "Division of Function", is employed. Each CPU and channel are simulated as separate units as are control units and I/O devices. Then, to add a specific new device simply requires the addition of a device simulation routine for the device. A particular simulation routine will be for a particular type of device (ie.,CPU,channel, control units, printers, etc.) rather than a specific instance of that device in the simulated configuration. In this way, the I/O simulator will be a general scheme for doing I/O independent of the devices used. And because of the devision of function, changes in the nature of one simulated component (ie. CPU) have limited effects on the rest of the system.

2.2 The Complete Simulator

The FOS (Family of Operations Systems) simulator provides a

"high fidility" simulation of a computer system not unlike an IBM 360/370 or a PDP 11. The simulation includes instruction timing, interrupt generation and handling, multiple CPU's, CPU timers, Trace and debug package, channels and device control units, and a built in direct linking loader.

The simulation is divided into subsystems, one for each of the simulation functions. Global data bases provide for communication between subsystems. A subsystem will ususally consist of several program modules and several entry points. Subsystems include the trace and debug systems, instruction interpretation, I/O system, and interrupt system.

2.2.1 Events

A central theme to the simulator is the notion of an event. An event is an action of a specific type which is scheduled to occur at a specific time during the simulation. Typical events include execution of an instruction by a particular CPU, execution of an I/O instruction by a particular channel, handling of an interrupt, (ie. I/O, program, SVC, external or machine check), decrementing of the interval timer, etc. The execution of a particular event often results in the scheduling of another event.[1]

2.2.2 I/O Interaction

The user will attempt to initiate I/O through a SIO(START I/O) or an STOF (START I/O Fast Release) instruction to a given device. When the CPU event (which signals simulation of next instruction

on the given CPU) for the SIO or SIOF is popped off the event queue and decoded, the I/O system will be entered to initiate the I/O. If the device is not busy and a path exists to the device using the specified channel, the device will begin operation. It is assumed that the user has supplied the channel program, its address, an I/O interrupt handler (in case some unusual status is detected), and its address. If I/O cannot be initiated and there exists some unusual status it is indicated in a stored channel status word (CSW). If subsequent to initiation of I/O an unusual status condition is detected the channel can request an I/O interrupt.

2.2.3 Interrupt System

The interrupt structure is divided into two pieces. One part, under the entry point REQUEST, is concerned with indicating in the state of a CPU that there is a request for an interrupt. The second piece is entered before the execution of an instruction by a CPU. It checks the CPU state, comparing requested interrupts against interrupt masks to decide which, if any, interrupts should be accepted. By keeping track of PSW swaps the system traps interrupt loops. If two interrupts of the same class (ie. a two program interrupt), are accepted without a LPSW being executed in between, a weak interrupt loop is detected. An error message is printed and simulation terminates. If two interrupts of the same class occur, separated by a LPSW but not a LPSW from the old PSW for that class, eg. two program interrupts separated only by a LPSW SVC OLDPSW, a strong interrupt loop is detected. In this case a message is printed

but simulation continues.

2.2.4 Debugging System

The debugging system allows the user to monitor the progress of the simulation by displaying status information to inputted trace requests. There are two phases to the system. The first phase reads debug requests, interprets them, and sets up tables for phase two. The second phase executes requests and displays information when a particular traceable event occurs.

In connection with I/O system things that can be looked at through the TRACE facility include the old and new PSW, CSW, CAW, CHANNEL STATUS and DEVICE STATUS.

2.3 System Configuration and Creation

A user, if he wishes, may specify an arbitary configuration as mentioned earlier (the details of the system creation are described in Appendix B). If he does not want to write his own system configuration file then he can select a standard system from a set of system configuration files that exist on disk or tape.

Any Legal S/370 type configuration may be specified, with the following restrictions : 1) no more than 32 channels on any one CPU, 2) device sharing by switching between control units only, 3) it must contain only those I/O devices provided by the simulator, eg. at present only the IBM 1403 printer and the 3505 card reader. In addition to legal S/370 configurations, many configurations which are not considered legal in Principles of Operation for the S/370 may be specified (ie. Channel 0 need not be byte multiplexer),

and although they will usually operate as expected, caution is advised.

The following "simulation control cards" control the initialization of the system configuration.

2.3.1 Configuration Card

The CONFIG card indicates the number of CPU's to be used in the current simulation. This number is used to control the reading of CPU cards (see next section). If the CONFIG card is missing, an error message is printed.

2.3.2 CPU Card

Each CPU card gives the simulator certain characteristics of a CPU, such as starting address, unit number, and number of attached channels which is used to control reading of CHAN cards (see next section). For each CPU card, instances of CPU_CNTL_STRUC the general registers, and control registers are allocated, and the I/O system associated with it is initialized (ie. channel, subchannel, control units and device data bases). The instruction address field of the PSW is set to the starting address and other variables in the control structure are initialized. If too many or too few CPU cards are read an error message is printed.

2.3.3 CHAN Card

Each CHAN card provides information such as channel type, and number of attached control units. The latter controls the reading of CU cards (see next section).

For each CHAN card instances of CH_CNTL_STRUC and CH_WK_REG are allocated. If it is a selector channel the subchannel control structure (SUBCH_CNTL_STRUC) is also allocated. If not, one SUBCH_CNTL_STRUC is allocated for each CU card since for multiplexer channels there is a one-to-one correspondence between subchannel and control units while selector channels have only one subchannel. If an incorrect number of CHAN cards are read, an error message is printed.

2.3.4 CU Cards

The CU card will give the lowest and highest address that the control unit will recognize, and will indicate whether or not the attached devices will be accessible to other control units by giving the number of a Device Set Switching Unit (DSSU). This DSSU will be responsible for controling which control unit a device set will be using. The CU card will also provide the number of attached devices to control reading of DEV call,(see next section).

For each CU card, instances of the CU_CNTL_STRUC and the CU_WK_REG are allocated and if the CU is attached to the multiplexer channel the SUBCH_CNTL_STRUC is allocated and appropriate pointers set. If a new DSSU ID number is given a DSSU_CNTL_STRUC is allocated and initialized. Again if an incorrect number of CU cards is detected an error message is printed.

2.3.5 DEV Cards

Each DEV card will contain the DEVICE TYPE, the mode in which

data is to be transferred, and any device dependent information that is necessary. For example, the input or output file, print time, delay time, etc. Instances of DEV_CNTL_STRUC and DEV_WK_REG are allocated for each DEV card.

2.3.6 DATA Cards

DATA cards are used to insert hexadecimal data directly into simulated menory. The location given on the data card is converted to an actual hexadecimal value and then stored in the memory location. If the location is greater than the size of memory, an error is printed.

2.3.7 MEMORY, TIME, and INSTRUC Cards

The MEMORY card indicates the size of the memory (in bytes) for the current simulation. The INSTRUC card is used to limit the number of instructions which are executed during the simulation. The TIME card is used to set a maximum time for the simulation.

3. I/O Structure

This section deals with the I/O structure in the sense of how data bases are structured and interconnected. The section should provide the reader with a flavor for the type of information that will be flowing in the I/O system and where bits of information might be used, (ie. by the channel, by the control unit, by the device, etc.), or found, (ie. in the subchannel, device data base, etc.), The section should also facilitate reading of the detail program operation given in section 4 which describes how the I/O system operation through analysis of the major routines. Data base examples are provided throughout this section.

3.1 Data Bases

The CPU Control Structure is quite long and contains many items that have nothing to do with the I/O system. I have therefore, in this case only, presented a short form of the original data base with all information pertinent to this paper.

3.1.1 CPU

To access CPU's there exist the following PL1 declared structure which holds pointers to the CPU control structures.

```
DCL  1   AREA_STRUC BASED (CPU_AREA_PTR)
         2 I FIXED BIN,
         2 CPU_AREA (CPU_NUM REFER(AREA_STRUC.I))PTR;
```

CPU_NUM is the total number of CPU's, which has been specified on the CONFIG card (see section 2.3.1)

The CPU data base contains the following information:

CPU control structure (CPU_CNTL_STRUC)-

-- Pointer to working registers

-- Pointer to structure which holds pointers to attached channels

-- Pointer to associated queue element

-- Current Program Status Word (PSW)

-- I/O request word (IORW)

-- Loop Flags

The I/O request word (32 bits) will indicate which channels, if any, are requesting interrupt if the corresponding bit positions are "1".

A CPU data base example is given in figure 3.1.

3.1.1.1 Loop Flags

Loop Flags keep track of interrupt loop. It is one byte long (8 bits) initialized to zero with bit assignment as follows:

0 - program loop

1 - strong program loop

2 - external loop

3 - strong external loop

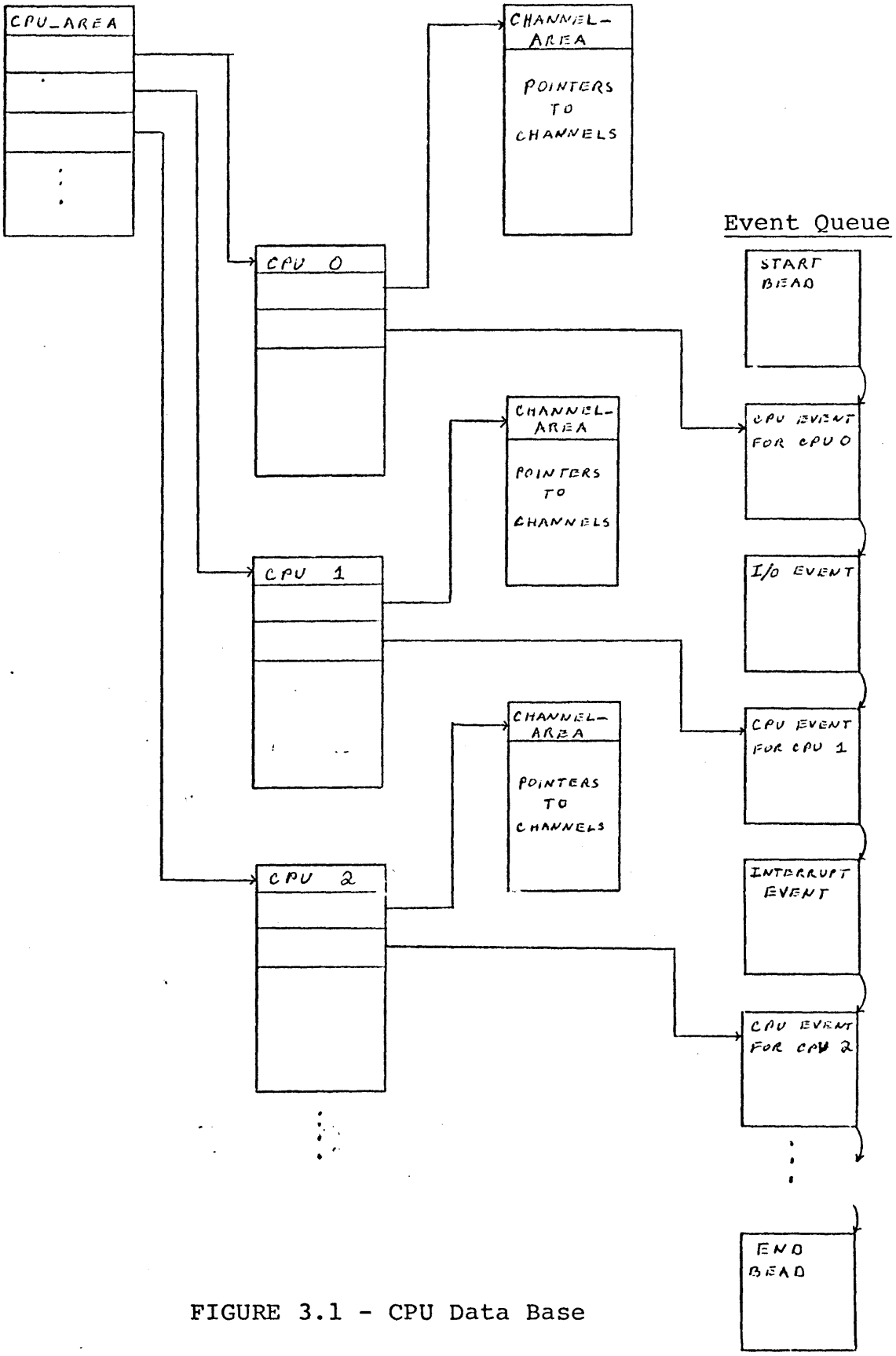

FIGURE 3.1 - CPU Data Base

4 - I/O Loop

5 - Strong I/O loop

6-7 Unassigned

Both bits associated with an interrupt class are set on whenever PSW's are swapped (ie. when an interrupt is accepted). Any LPSW turns off one of each pair while an LPSW from some old interrupt PSW location also turns off the other bit for that interrupt class. Thus, if an interrupt is about to be accepted but it is found that one of the loop flags bits corresponding to the class is on, a strong interrupt loop has been detected, and if both bits are on, a weak interrupt loop has been detected.

3.1.2 Channel and Subchannel

To access channels there exist the following PL1 declared structure which holds pointers to the channel control structure.

```
DCL  1  CH_AREA_STRUC BASED (CH_AREA_PTR)
        2 I FIXED BIN,
        2 CH_AREA (CH_NUM AFTER (CH_AREA_STRUC.I)) PTR,
```

CH_NUM is the total number of channels attached to the given CPU, and is specified by the CPU card (see section 2.3.2)

The channel data base contains the following information:

Channel Control Structure (CH_CNTL_STRUC) -

-- Pointer to working register
-- Pointer to CPU to which channel is attached
-- Pointer to first attached control unit (one with lowest address)
-- Channel Type

The working register will contain the following:

-- The state of the channel (ie. available, interrupt pending or working)
-- A pointer to the subchannel associated with the current operation
-- A pointer to the subchannel for the next interruption

The subchannel data base contains the following:

Subchannel Control Sructure (SUBCH_CNTL_STRUC)-

-- The state of the subchannel (ie. available, interrupt pending, or working)
-- I/O address
-- Current Channel Command Word (CCW)
-- Next Indirect Address Word (IDAW) address
-- Interrupt pointer

The I/O address associated with the current I/O operation is the channel and device address, as specified in a SIO or TIO, etc.

instruction. Interrupt pointer is needed in case it becomes necessary to clear a pending interrupt condition. For example, turning off a bit in the IORW or removing the associated interrupt event from the event queue.

A data base example is given in figure 3.2

3.1.3 Control Unit

The control unit data base contains the following information:

Control Unit Control Structure (CU_CNTL_STRUC)-

-- Pointer to working registers
-- Pointers to next control unit
-- Pointers to CH_CNTL_STRUC to which CU is attached
-- Lowest device address the control unit will recognize
-- Highest device address the control unit will recognize
-- Pointer to the subchannel with which the CU is associated
-- Pointer to Device Set Switching Unit (DSSU)
-- Pointer to first attached device

The CU working register contains a pointer to the device control structure that is currently being used. Device sharing among control units is controled by the DSSU which keeps track of which control units is attached to and making use of a given device set.

3.1.4 Devices

The device data base contains the following information:

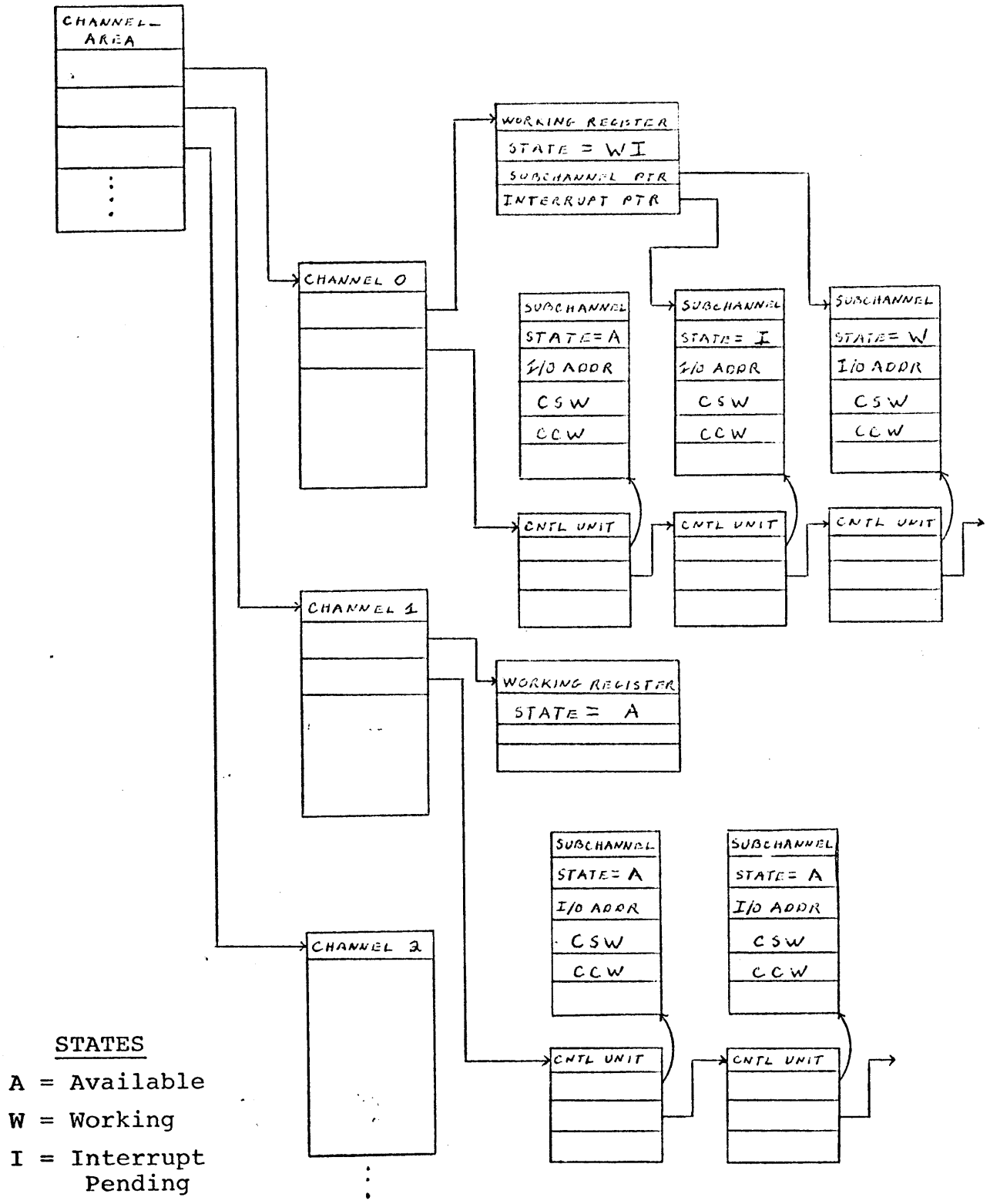

FIGURE 3.2 - Channel Data Bases

Device Control Structure (DEV_CNTL_STRUC)-

-- Pointer to working registers

-- Pointer to next device

-- Pointer to DSSU to which device is attached

-- Device type

-- Static device-dependent information

Static device-dependent information are constant characteristics of the associated device, for example print time or whether information should be transferred in burst or byte-interleave mode. The working registers contain dynamic characteristics as device status, data count, sense byte, etc. A data base example is shown in figure 3.3.

3.2 Event Queue

Event sequencing is accomplished by means of an event queue ordered primarily by time and secondarily by event priority, which is determined from the kind of event. (Thus, for instance, in the case of simultaneous interrupt requests and simulation events, the interrupt requests will be recognized before the instruction is started). The maximum time at which an event can be scheduled to start is the simulation time limit specified by the user. Any event which is to start after that is not placed on the queue.

The first and last beads on the queue have time and status such that no queue element could possibly belong before and after them, respectively. In addition, there is an element scheduled to

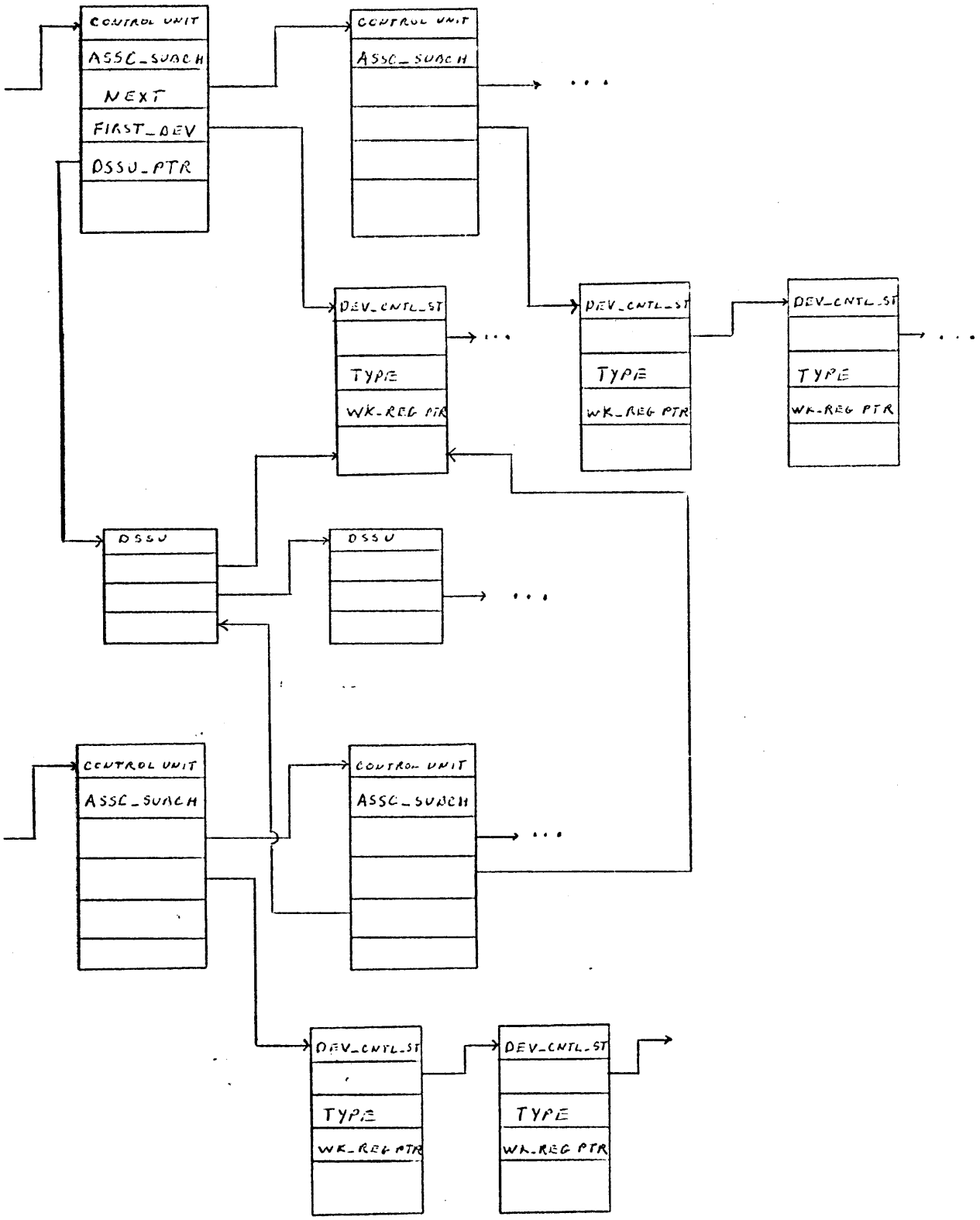

FIGURE 3.3 - Device Data Base

occur at the simulation time limit which, when executed, cause termination due to timer runout.

Other elements are placed at the proper place in the queue by entry INSERT in procedure SIMUTI. This expects to find the queue element to be inserted pointed to by CURR_Q_EL in PSTRUC. If the event is to occur too late to be scheduled, the back pointer LAST is set to NULL and a completion code of 1 is returned.

3.2.1 Queue Element

A queue element is defined as follows:

```
DCL 1 Q_EL1 Based (CURR_Q_EL)
      2 NEXT PTR,
      2 LAST PTR,
      2 TIME,
        3 A FIXED BIN (31),
        3 B FIXED BIN (31),
      2 CLASS FIXED BIN (31),
      2 TYPE FIXED BIN (31),
      2 DEST PTR,
      2 SOUCE PTR,
      2 DATA PTR;
```

NEXT and LAST are pointers to the next and previous queue element respectively. TIME is the scheduled time for occurance. CLASS tells what this particular element is concerned with. Associated with the I/O system there are : I/O events, class 12; I/O interrupts, class 6; device events, class 14 for printers. TYPE provides addtion-

al information for the specific class. For an I/O event, type can be 1,2, or 3 where "1" is "I/O to Begin" which signals that an SIO instruction has been detected and the channel needs to be awaken. "2" is an I/O start up event which signals that a SIOF has been initiated. "3" indicates command chaining is to take place. For I/O interrupt events TYPE is the number of the channel requesting an interrupt. For a printer event type can be "1", or "2" indicating a data transfer or that a device end is to be sent to the channel, respectively. DEST is a destination pointer and when used points at a subchannel control structure. It is not used for I/O event of Type "1". Source pointer in a multiprogramming environment will provide information as to which program the event is associated with or it may point at a CPU control structure. DATA is a pointer to a section that provides any other data that is necessary to process the event. For DEVICE EVENTS, for example, it would point at the associated device control structure.

4. Program Operation

This section describes in detail how the I/O simulation has been handled and implemented. Section 4.1 describes the major routines and section 4.2 shows how they interact. Section 4.3 gives the sequence of events for sample I/O operations. It is hoped that this break down will provide the reader with a simple and clear picture of the program operation.

4.1 Major Routines

The first two routines described are not I/O routines in the sense of only being used during I/O. These are routines which help control and direct the entire simulator. All other routines are in the I/O system and are only used in connection with I/O. The first two routines give a feeling for how and when the I/O system will be entered.

4.1.1 MASTER_DRIVER

Events are executed by a loop in MASTER_DRIVER located in SIMMAIN. After an event has gone to completion, the next event to be handled is the element on the top of the queue (unless simulation has been terminated). Handling the event involves removing the element from the queue, making the CPU the current one, getting the associated general registers, and doing some event dependent processing.

For a CPU simulation event, the CPU simulation routine, CPU_SIM, is called. If it returns normally, a new CPU simulation event

is scheduled for the time at which the current one is finished. If the return indicates that an interrupt loop has been detected, simulation is terminated. The returns are :

0 - normal

1 - CPU inactive

-1 - interrupt loop

For returns 0,1 a completion code of 0 is returned to the top of the loop in MASTER_DRIVER.

When the users specify some type of I/O operation, (ie. SIO, TIO, HIO, etc.) it will appear in the event queue as a CPU simulation event not an I/O event (see section 3.2.1 for events specifically associated with the I/O system). It is not until after CPU_SIM has been called and the op code for the instruction has been decoded, that the appropriate I/O system subroutine (ie. SIO, TIO, etc.) will be called.

For an interrupt event, the interrupt class and type are removed from the queue element and placed in INT_CLASS and INT_TYPE in the static external structure MISTRUC. DATA_PTR is also copied into MISTRUC to save anything that might be hung off of it. The queue element is then freed. REQUEST is called to mark the requested interrupt in the current CPU state, and since REQUEST has no abnormal returns, 0 is returned to the top of the loop by default.

If the timer runout event is executed, "SIMULATION TIMER HAS BEEN EXCEEDED " is written onto SYSPRINT and simulation is terminated.

4.1.2 CPU_SIM

The CPU simulator, CPU_SIM, is activated by the event sequencer MASTER_DRIVER when it recognizes a CPU simulation event. If the CPU in question is enabled for any requested interrupts, they are accepted by calling the entry ACCEPT, located in INTRPT, which accepts interrupts in order of priority as specified in IBM System/370 Principles of Operation. If, after accepting these interrupts, the CPU is in the wait state, CPU_SIM passes the completion code of 1 back to MASTER_DRIVER, while if an interrupt is detected loop is detected a completion code of -1 is sent back. For a normal return from ACCEPT, CPU_SIM continues on to execute an instruction.

If there is a specification exception due to an odd instruction address, the request is marked in CPU_CNTL_STRUC and accepted by looping back to the interrupt accepting stage. Otherwise, the instruction is fetched by calling entry LOG located in SIMUT1 once, unless the instruction straddles a memory boundary, in which case two calls are necessary. If accessing exceptions are raised in the instruction fetch, the interrupts are requested by LOG and accepted by looping to the interrupt accepting state.

The instruction itself is simulated by calling one of a group of routines which contain entry points labeled with the mnemonic op codes of the instructions they simulate. For example, if the instruction had been a START I/O to a printer, SIO would be called. These use the quantities such as R1, R2, ADDR1, ADDR2 computed during the instruction fetch. On return from the instruction routines, CPU_SIM updates the CPU simulation timer by the appropriate amount

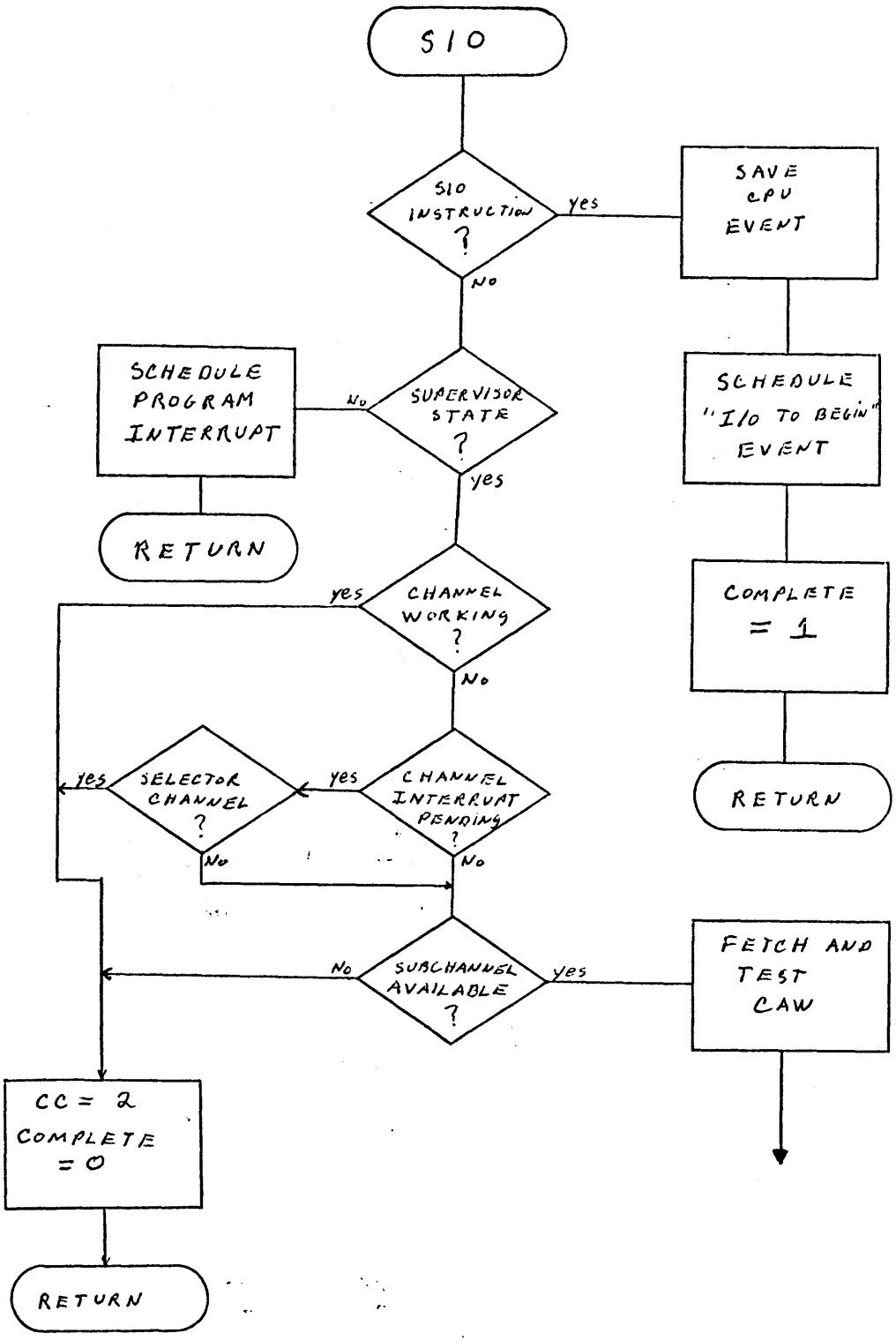

FIGURE 4.1 - SIO Routine
(continued on next page)

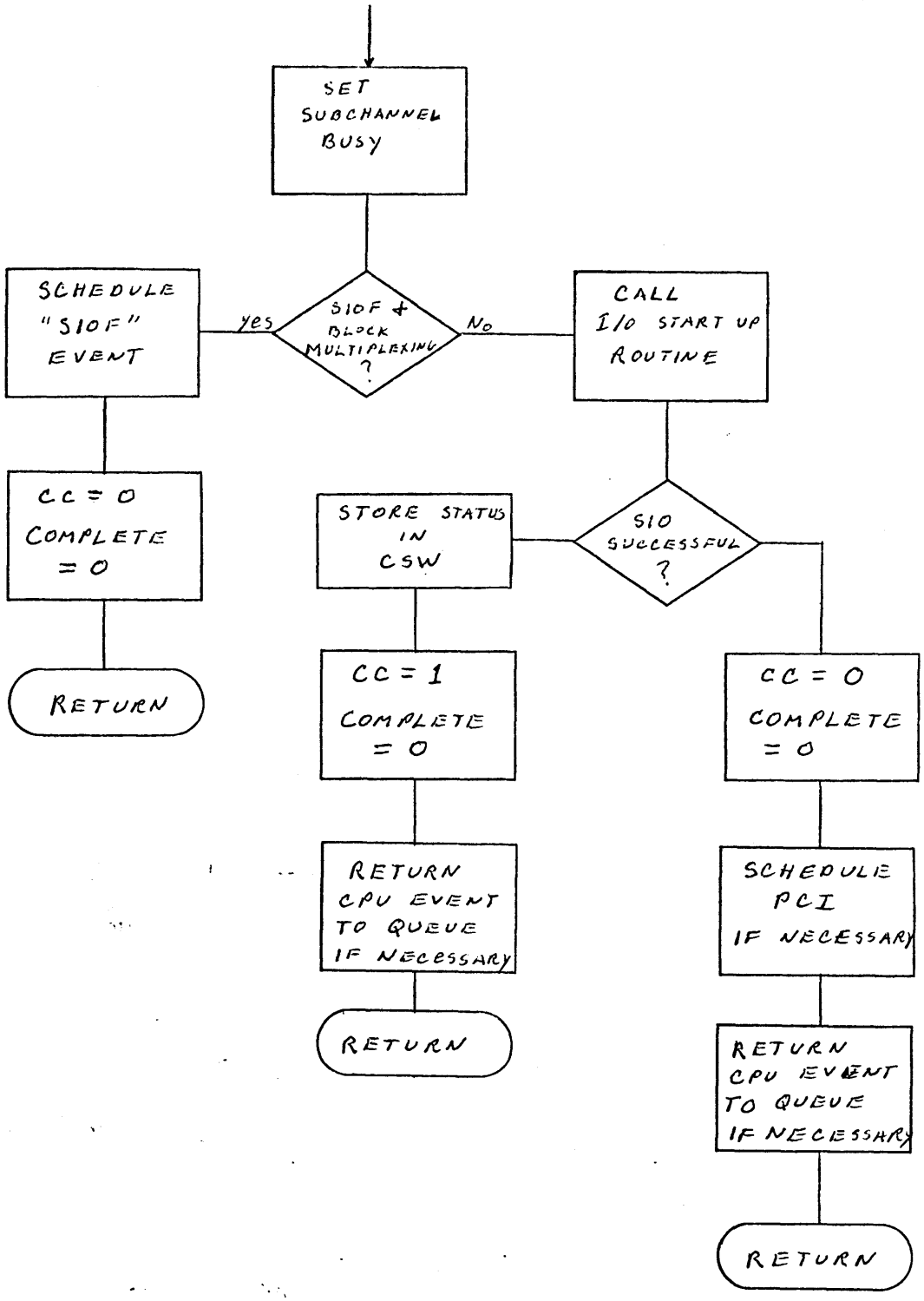

FIGURE 4.1 - Continued

of time which elasped while the instruction was being executed and passes control back to MASTER_DRIVER.

4.1.3 SIO

The START I/O (SIO) routine (flowcharted in figure 4.1) is branched to from the CPU_SIM routine when an SIO or SIOF instruction is detected, and branched to from MASTER_DRIVER when an "I/O to Begin" event is detected. Thus when a SIO instruction is detected and a branch is made to the SIO routine, this routine will then schedule an event, to occur immediately, that wakes up the channel and reports that there is "I/O to Begin". A return code of 1 is returned to indicate to MASTER_DRIVER that the original CPU instruction (SIO in this case) has not been completed and not to schedule another one for that specific CPU. If the instruction was a SIOF or an I/O to Begin event, the scheduling does not occur and processing continues.

The CPU is checked to make sure that it is operating in the supervisor state. If the channel is busy or the subchannel is not available then the condition code is set to BUSY and COMPLETE is set to zero indicating to MASTER_DRIVER that processing of the instruction is complete. Otherwise, the CAW is fetched and tested.

Testing the CAW (see figure 4.2) is just making sure that bits 4-7 (not used) and 29-31 (insures double word boundary) are zero.

The subchannel state is set to working and a check is made to see if the instruction being worked on is an SIOF. If so, the

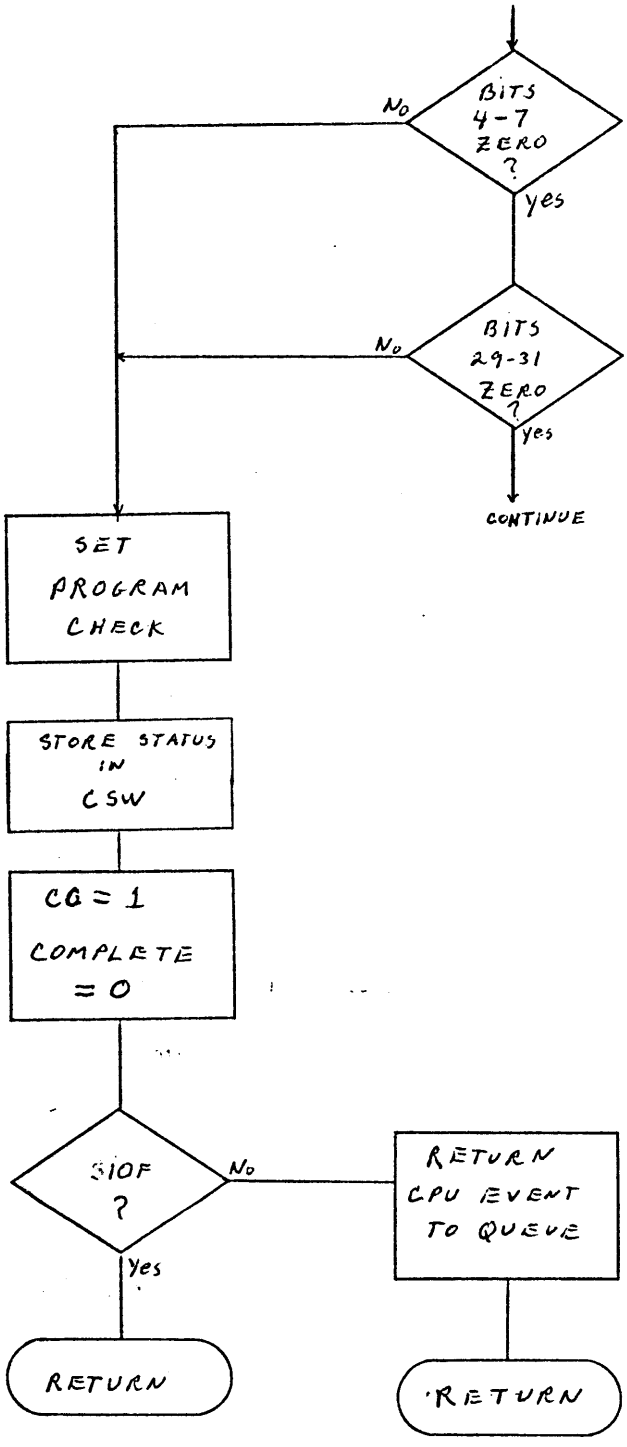

FIGURE 4.2 - Test CAW

condition code indicates that I/O has been initiated and a SIOF event is scheduled. If not, or if it is inhibited (ie. Block multiplexing bit off) then a normal SIO is performed.

I/O STARTUP is called and if upon return the subchannel is not BUSY, it is an indication that the SIO attempt was unsucessful. The unit status is OR'ed into the CSW, the subchannel is made available (ie. previous interrupt condition cleared) and the condition code indicates that a CSW has been stored. If the subchannel is BUSY upon return, then the SIO was successful and the condition code indicates I/O has been initiated. If the PCI bit is on, an interrupt is scheduled and we're done.

4.1.4 I/O START UP

The I/O START UP routine (flowcharted in figure 4.3) loads necessary data into the subchannel, sets the channel to BUSY if BURST mode is specified, and sends orders to the control unit (CU_SIM). If, upon return from CUSIM, there is no unusual status we're done. If unusual status is detected and we are not trying to initiate an SIO instruction then an I/O interrupt is scheduled. while for an SIO instruction the CSW would indicate the interrupt conditions. If PCI is the only bit on in the unit status we can still initiate I/O and therefore we simply return. Otherwise, there must be some unusual status and the BUSY bit is cleared in the channel and subchannel to indicate to the SIO routine that the I/O attempt was unsucessful. If SIOF was specified, the

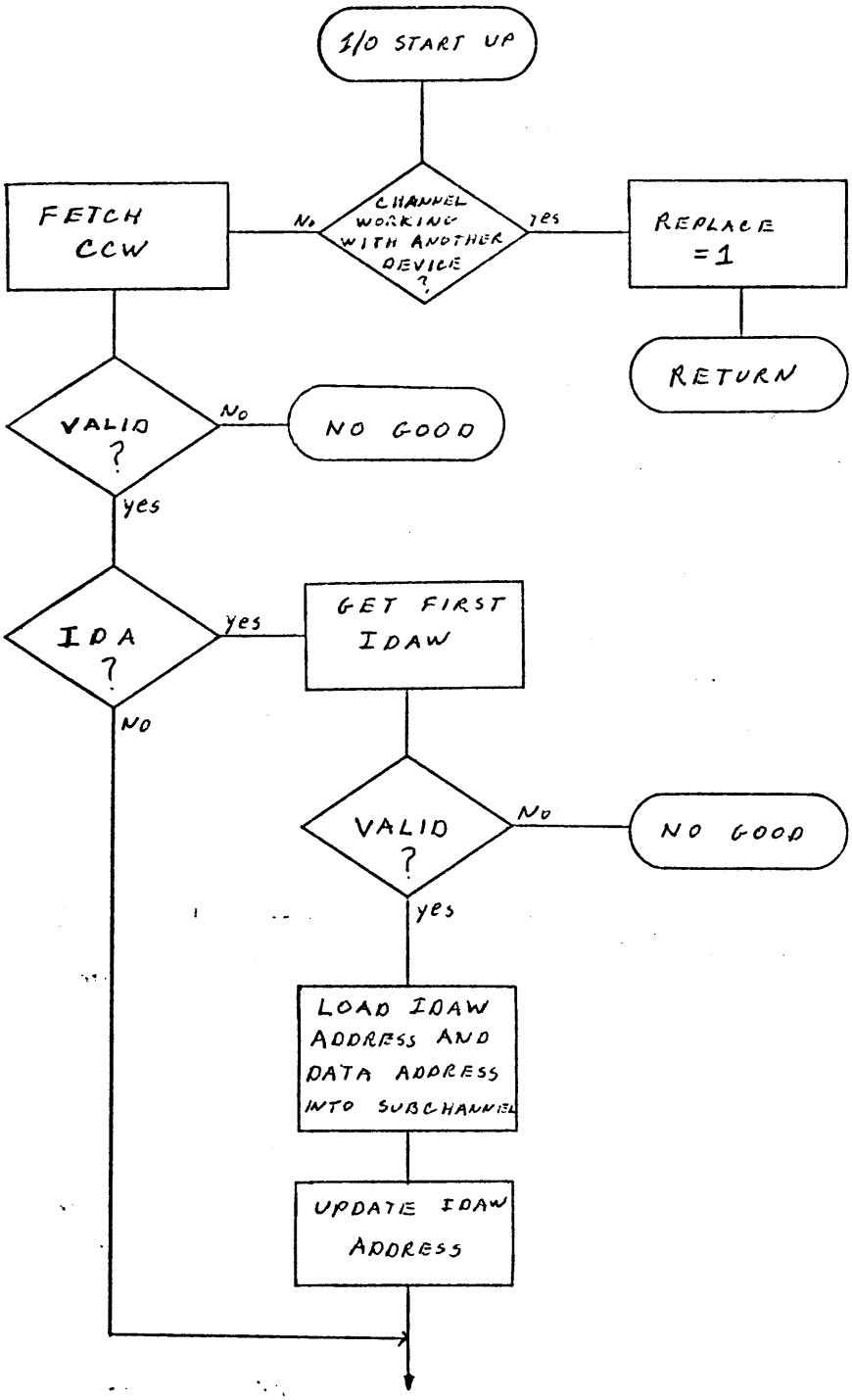

FIGURE 4.3 - I/O Start Up Routine
(continued on next page)

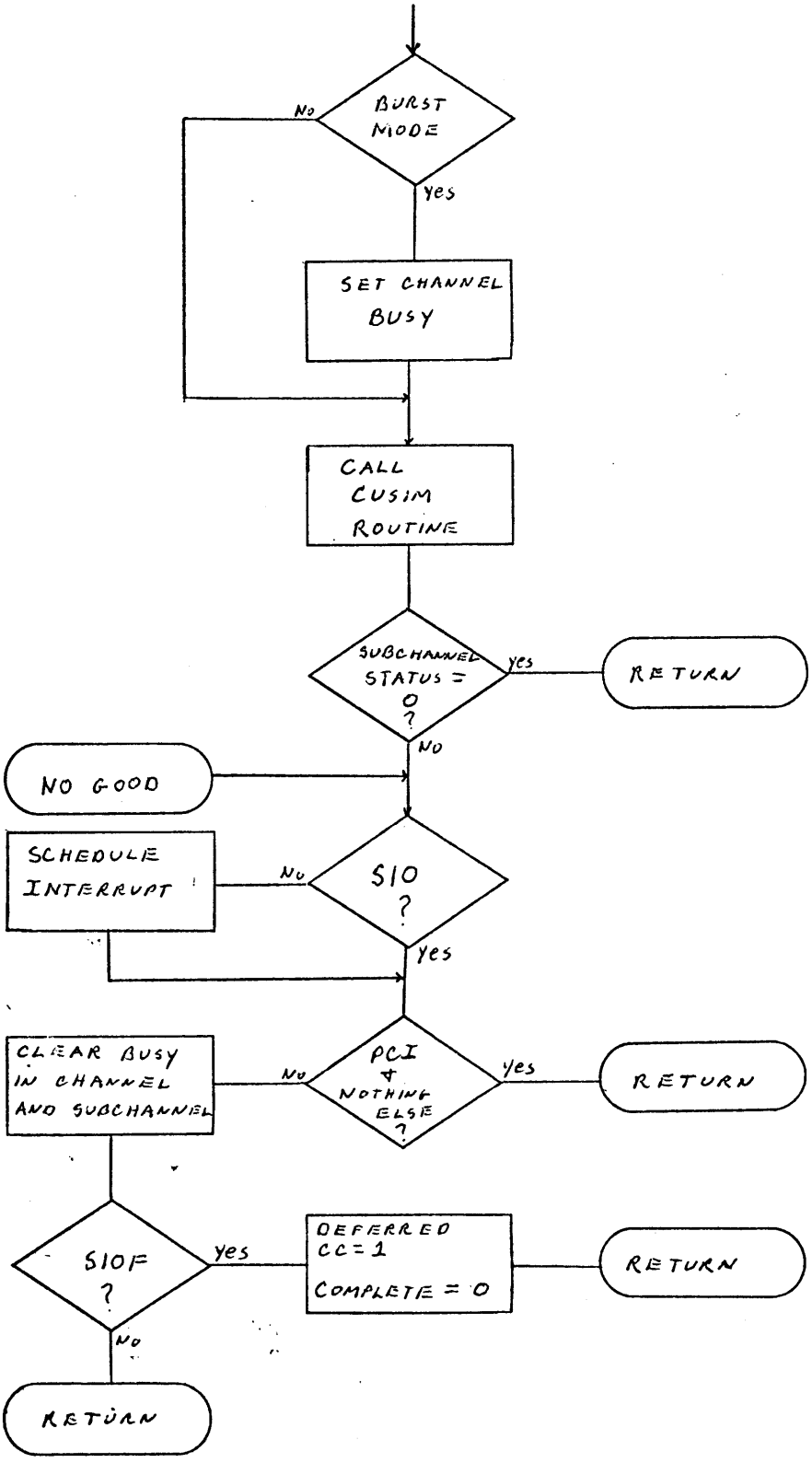

FIGURE 4.3 - Continued

deferred condition code is set to indicate that unusual status has been detected, and COMPLETE =0 indicates to MASTER_DRIVER that the CPU event can be rescheduled.

4.1.4.1 Fetch CCW

The "Fetch CCW" routine is flowcharted in figure 4.4. Before actually getting the CCW checks are made to see if its address is on a double word boundary, that its address is available to the channel, and that the keys match. After getting the CCW and it is verified that bits 38 and 39 are zero, a test for TIC (Transfer in Channel) is made. If a TIC is specified, it must not be in the first CCW of an SIO(F) operation nor can it have been specified in the previous CCW. If no program or protection check conditions are raised, the CCW is put into the subchannel, and the PCI bit is turned on in CH_STATUS if necessary.

4.1.4.2 Get First IDAW

Before the first IDAW is fetched (see figure 4.5), checks are made to see if the Indirect Data Address (IDA) is on a full work boundary, if the IDA is available to the channel, and if the keys match. Bits 0-7 of the IDAW must be zero.

4.1.5 CU_SIM

It is the job of CU_SIM (flowcharted in figure 4.6) to make sure that the addressed device exists and to send orders

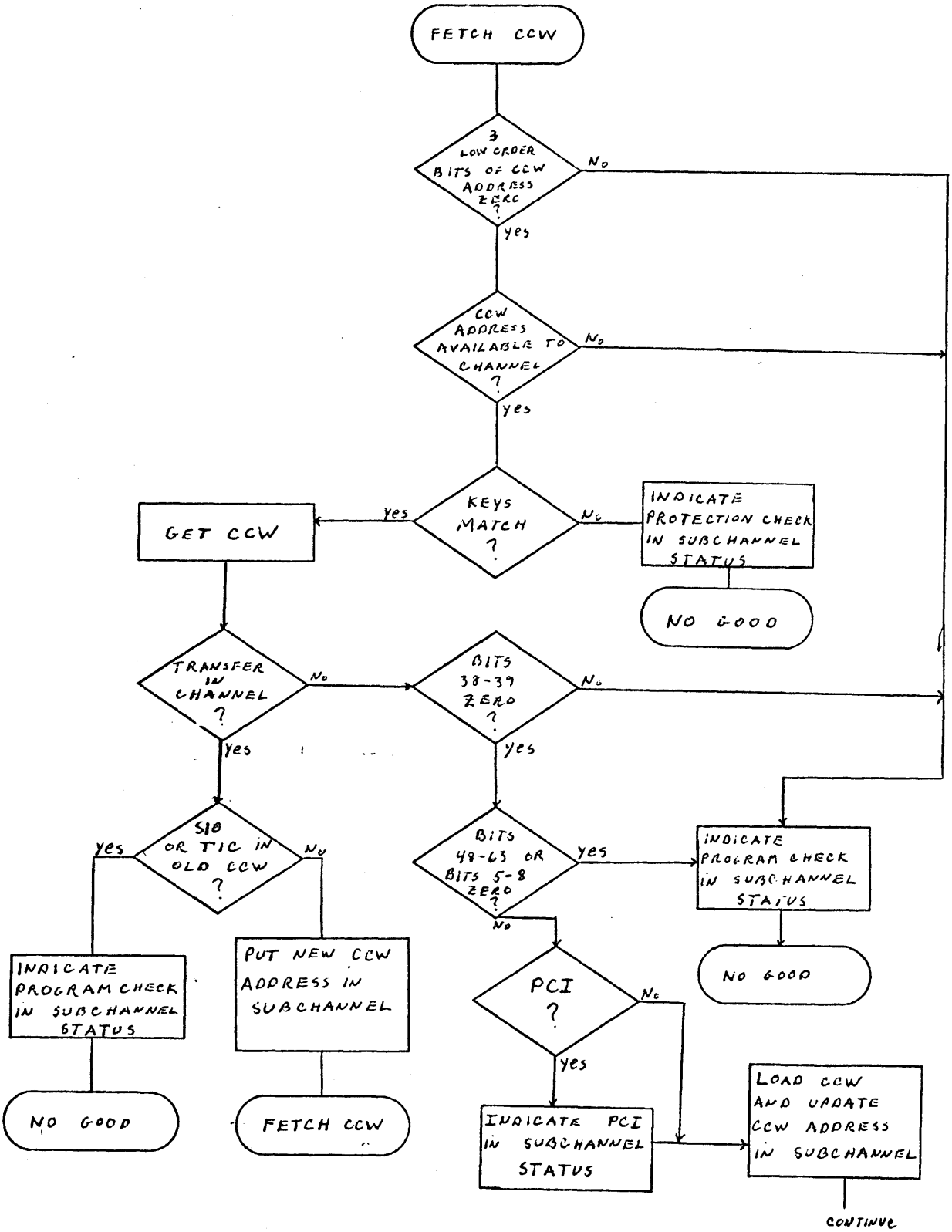

FIGURE 4.4 - Fetch CCW

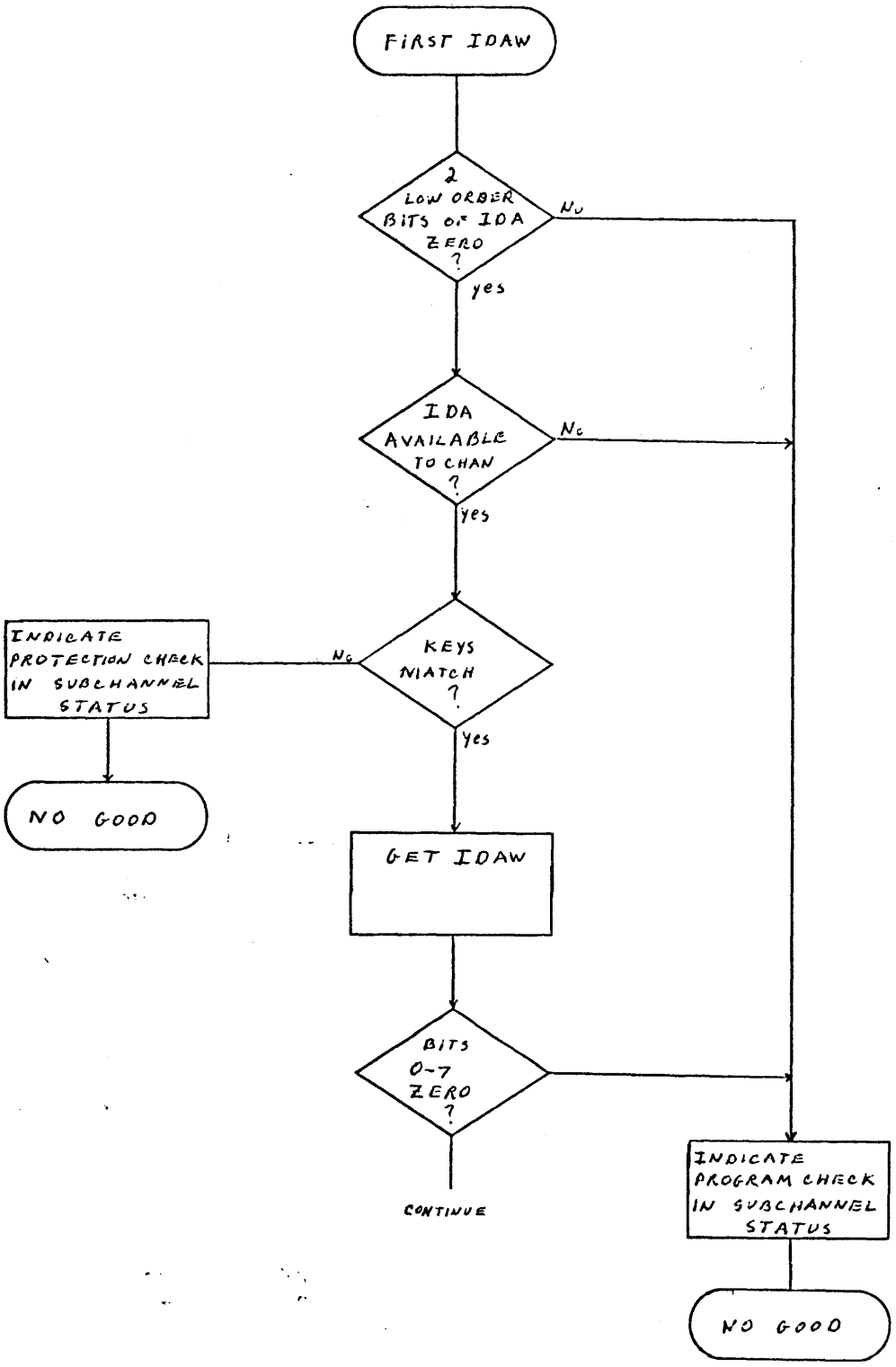

FIGURE 4.5 - Get First IDAW

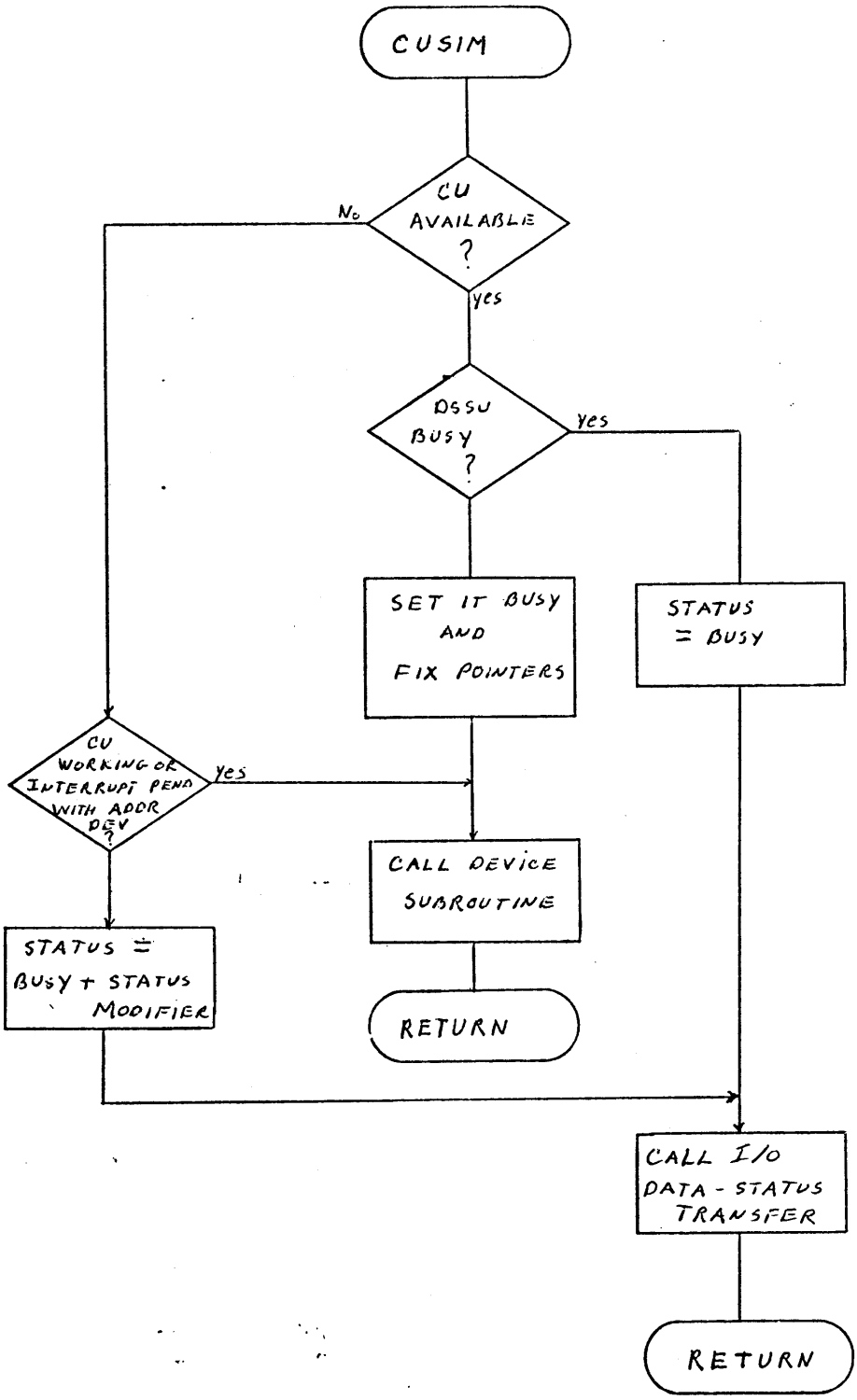

FIGURE 4.6 - CU_SIM Routine

to the device. Orders are sent only if the control unit is working with the addressed device, or if the control unit is available and the specified device is not working under another control unit through the device switching unit. Otherwise, the appropriate status is sent to the channel.

Note that this routine is entered directly from the TIO subroutine.

4.1.6 TIO

When a TEST I/O (TIO) instruction is detected the TIO routine (flowcharted in figure 4.7) is entered. If the channel or subchannel is busy or the subchannel is interrupt-pending with other than the addressed device, a condition code of busy is returned. Otherwise, if the channel is available "TIO" orders are sent to CU_SIM. If upon return from CU_SIM there is no unusual status, the condition code will indicate that the device is available and a return is made. COMPLETE =0 provides an indication to MASTER_DRIVER that processing has gone to normal completion. If the subchannel is interrupt-pending with the address device, the interrupt condition is cleared. Then, and also if any unusual status has been detected, the subchannel state is set to available and the condition code indicates that the CSW has been stored.

4.1.7 I/O DATA STATUS TRANSFER

I/O Data_Status Transfer (flowcharted in figure 4.8) handles data transfers and the transferring of status to the

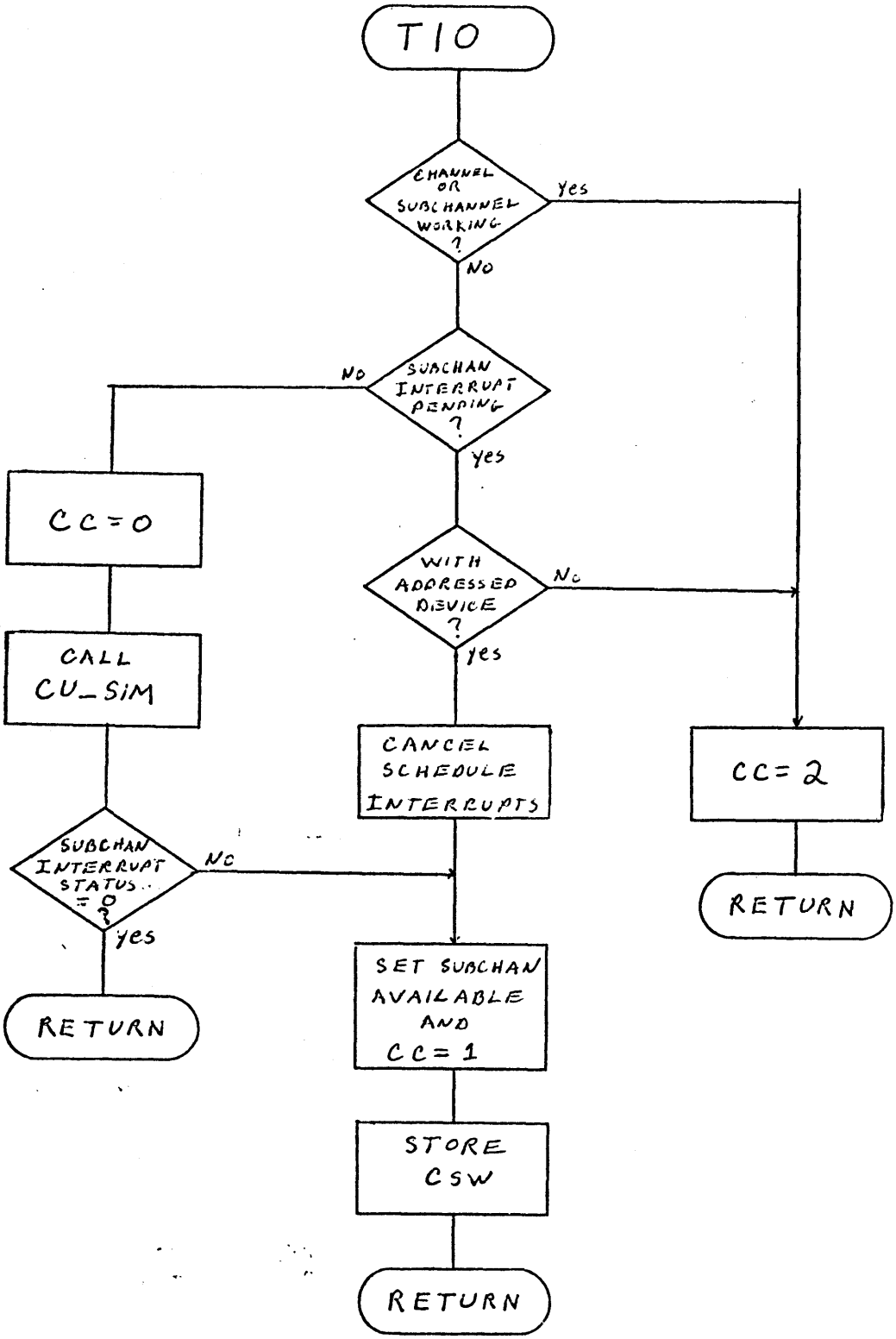

FIGURE 4.7 - TIO Routine

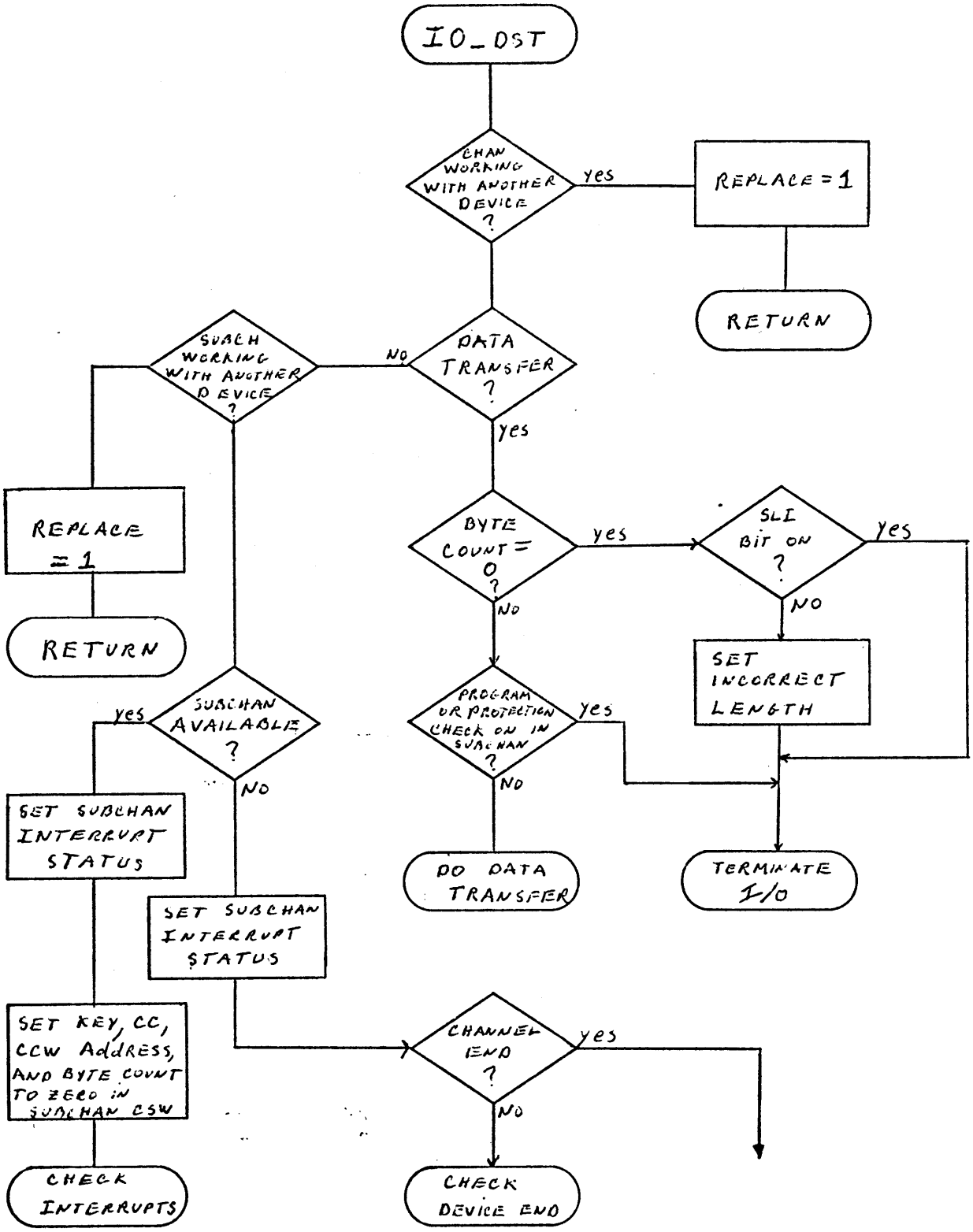

FIGURE 4.8 - I/O Data_Status_Transfer
(continued on next page)

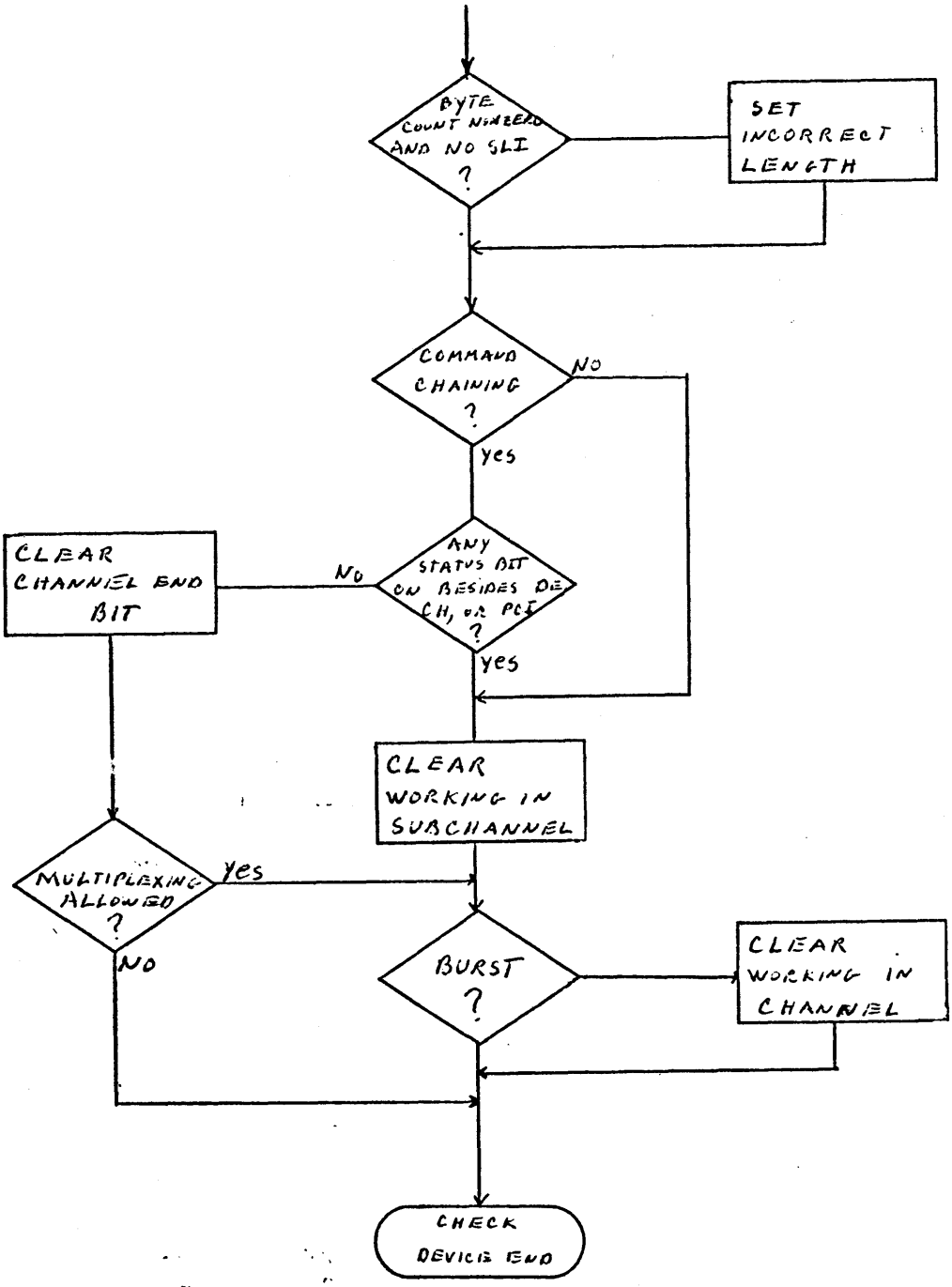

FIGURE 4.8 - Continued

channel. REPLACE =1 is an indication to MASTER_DRIVER that the event must be retried and it is replaced on the queue to be tried at a given incremental time later. If status needs to be transferred and the subchannel is available, we know that we are in the midst of executing a TIO instruction. The status field of the CSW in the subchannel is set from the device status and the other fields are set to zero. We can then check to see if any interrupt need scheduling and then return. If the subchannel is not available, status is OR'ed into the CSW status in the SUBCHANNEL.

Next a check is made to see if a channel end or device end (see section 4.1.7.1) condition exists. A channel end conditon signals that the channel has completed its current task and is able to take on another. If it exists, a check is made for incorrect length and command chaining (indicates there is another CCW to follow). If the command chaining bit (in previous CCW) is off or there are conditions to suppress it (ie. unusual status exist), the BUSY bit in the subchannel is cleared. If operation is in burst mode the BUSY bit in the channel is also cleared. If command chaining exist without any conditions to suppress it then the channel end bit is cleared.

4.1.7.1 Check Device End

If a device end is present and the subchannel state is BUSY, then command chaining is indicated and the device end bit is cleared if no unusual status is detected (see figure 4.9). A command chaining event is then scheduled. If some unusual status is present the BUSY bit in the subchannel, and channel if necessary, are cleared, and then a check for interrupts is made.

4.1.7.2 Check Interrupts

"Check Interrupts", flowcharted in figure 4.10, checks for any unusual status or if PCI = 1. If so, and we are not in the midst of a SIO, TIO, or SIOF instruction, then I (indicating interrupt is pending) is set in the subchannel state, and an I/O interrupt event is scheduled.

4.1.7.3 Do Data Transfer

If the I/O Data Status Transfer routine had been entered in order to have a data transfer performed and the byte count and channel status conditions are "all right" then the data transfer, flowcharted in figure 4.11, can be performed. A byte of data is fetched from or stored at the specified data address. If the SKIP flap is on in the CCW, no transfer is performed. If there is no

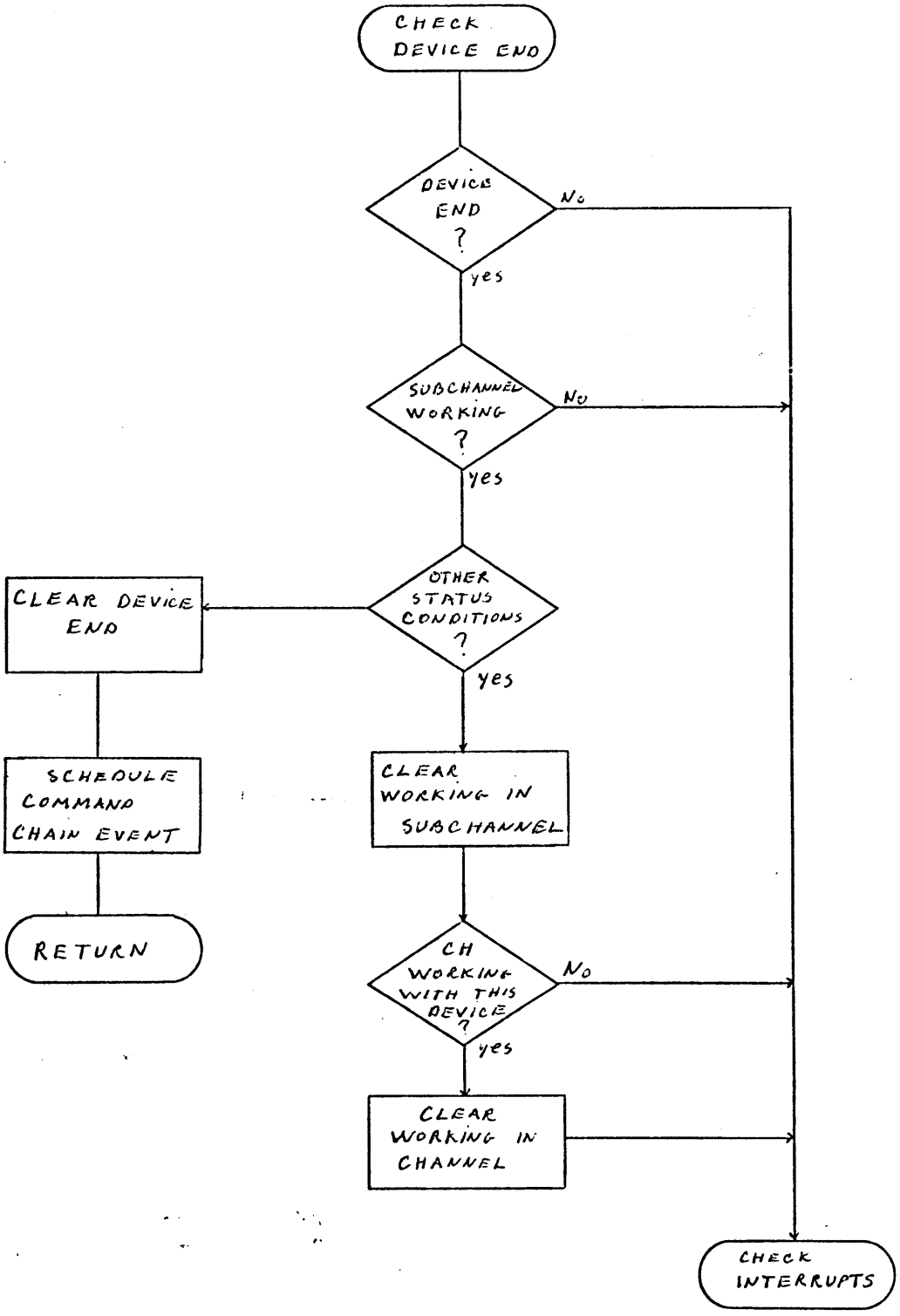

FIGURE 4.9 - Check Device End

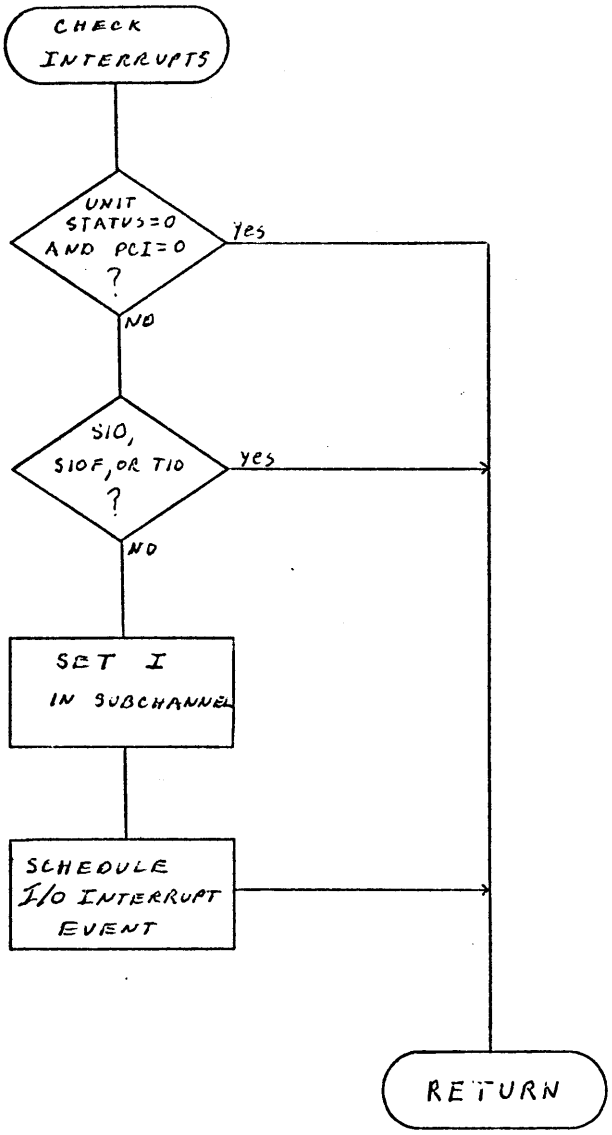

FIGURE 4.10 - Check Interrupts

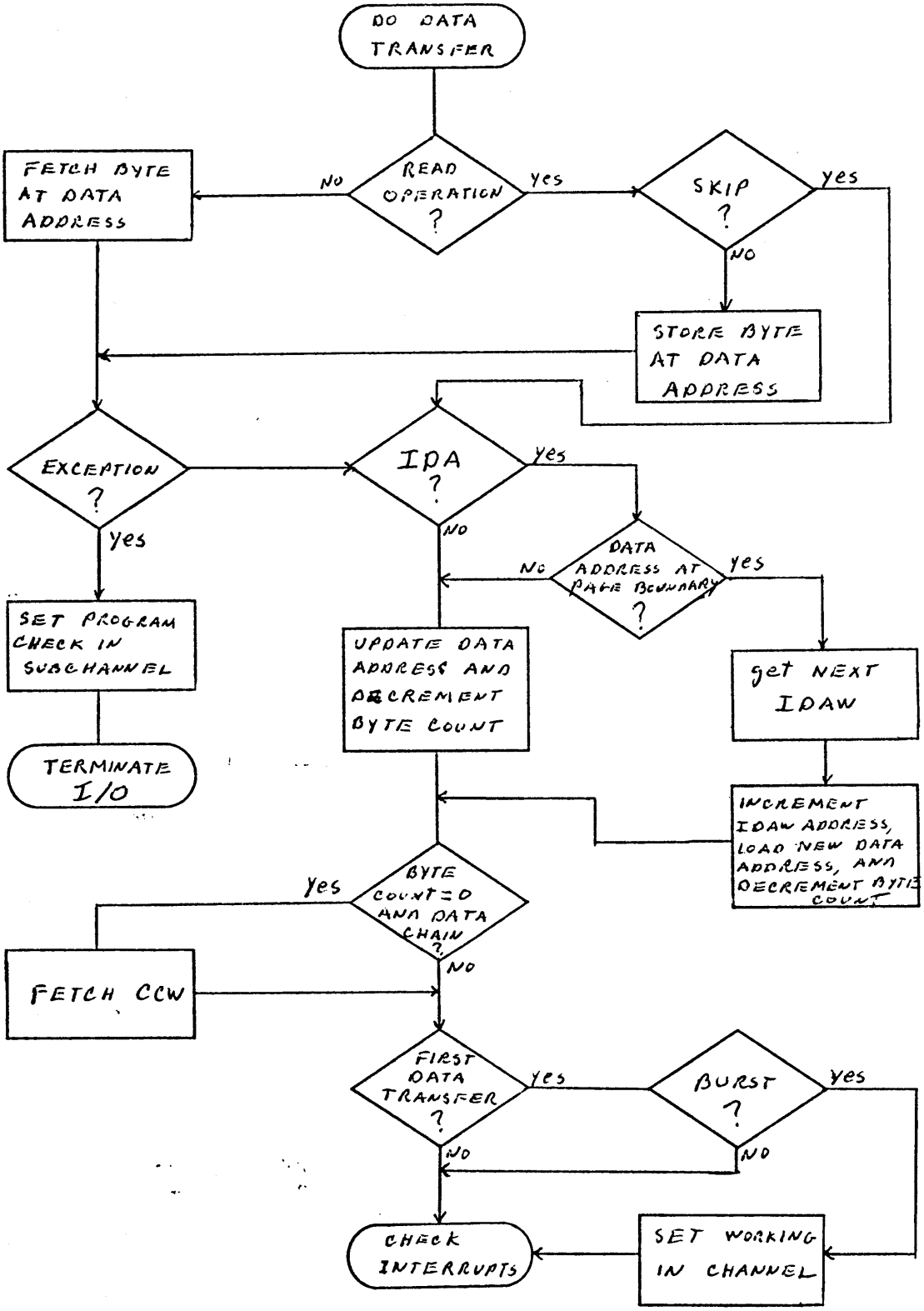

FIGURE 4.11 - Do Data Transfer

Indirect Data Address (IDA) the data address is updated and the byte count in the subchannel is decremented. If the count goes to zero and the DATA CHAIN flag is on, the next CCW is fetched. If this is the first byte transferred then the BUSY bit in the channel is set according to the transfer mode.

4.1.7.3.1 Get Next IDAW

If IDA was present and a page boundary is reached then the next IDAW is fetched (see figure 4.12). The data address must be available to the channel, the keys must match, bits 0-7 of the IDAW must be zero, and bits 21-31 of the IDAW must be zeros (ones for a read backwards command) to specify top (or bottom) of page.

4.1.8 Printer Routine

The printer routine, flowcharted in figure 4.13, is entered for a Device End event, for continued execution of a command chaining event, a SIO, a SIOF or a TIO instruction, or for handling a data transfer event. A device end event indicates that the alloted time for handling of a CCW operation at the device has terminated and it is available to go on to something else. The DEVICE END

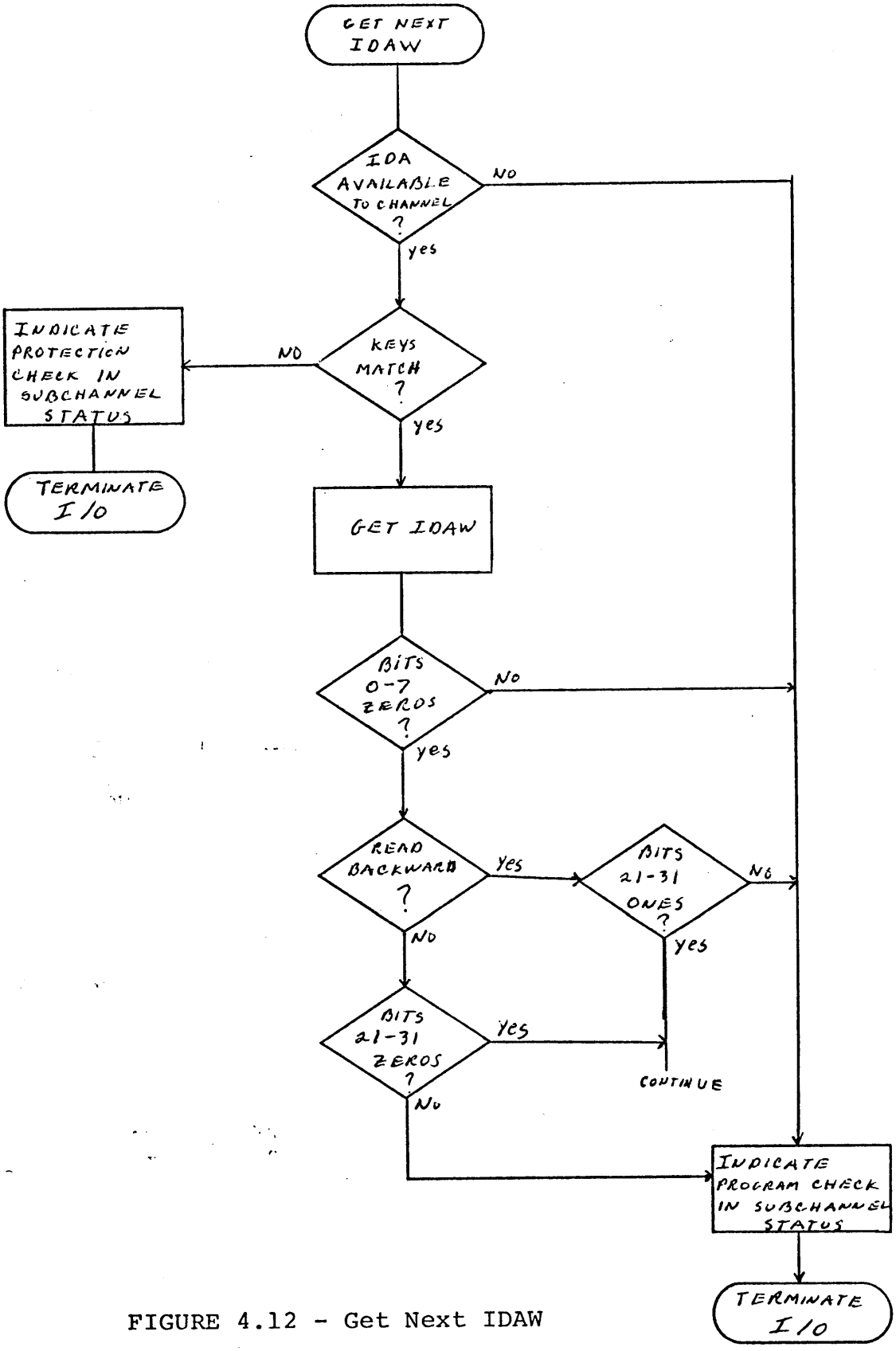

FIGURE 4.12 - Get Next IDAW

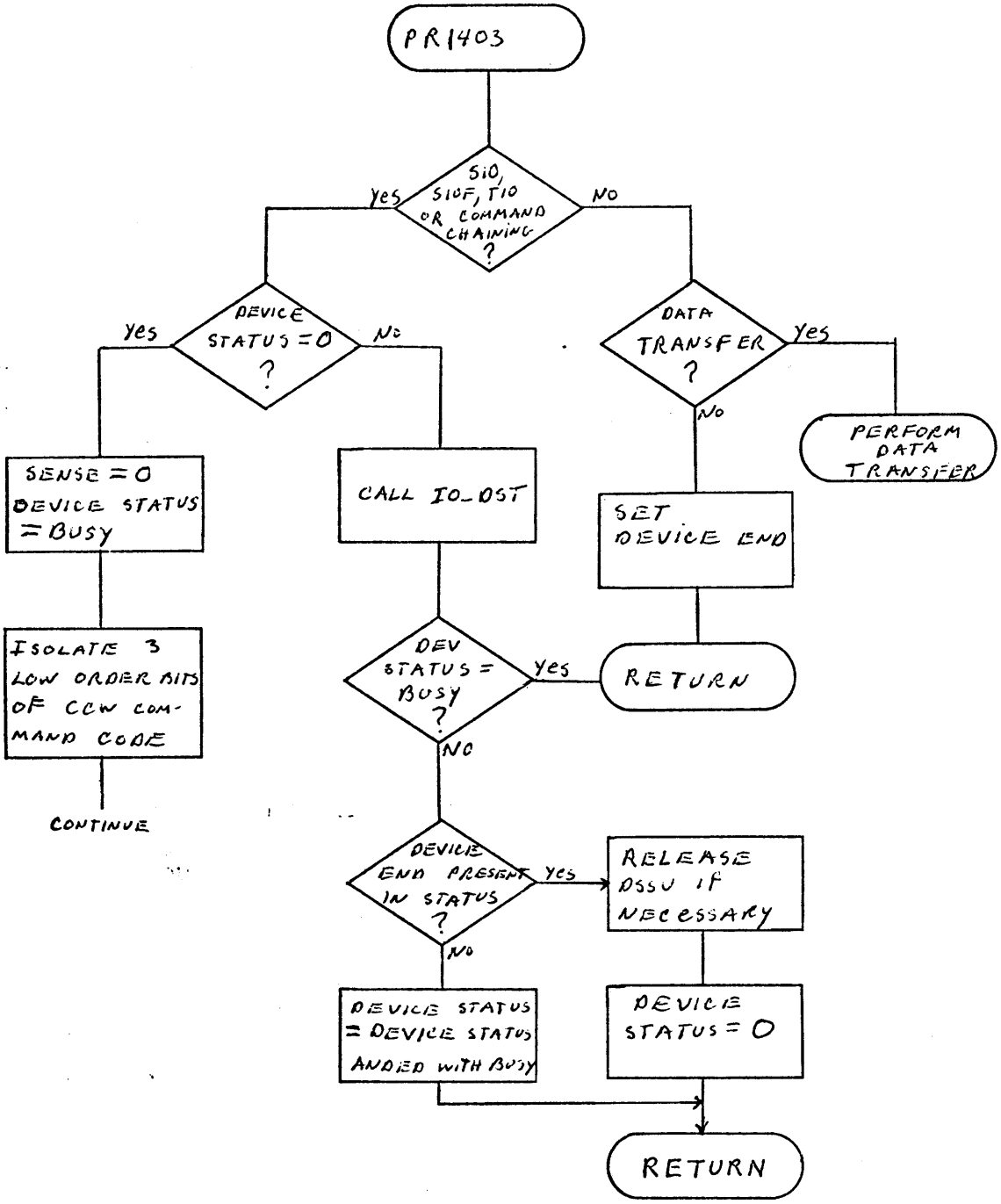

FIGURE 4.13 - 1403 Printer Routine
(continued on next page)

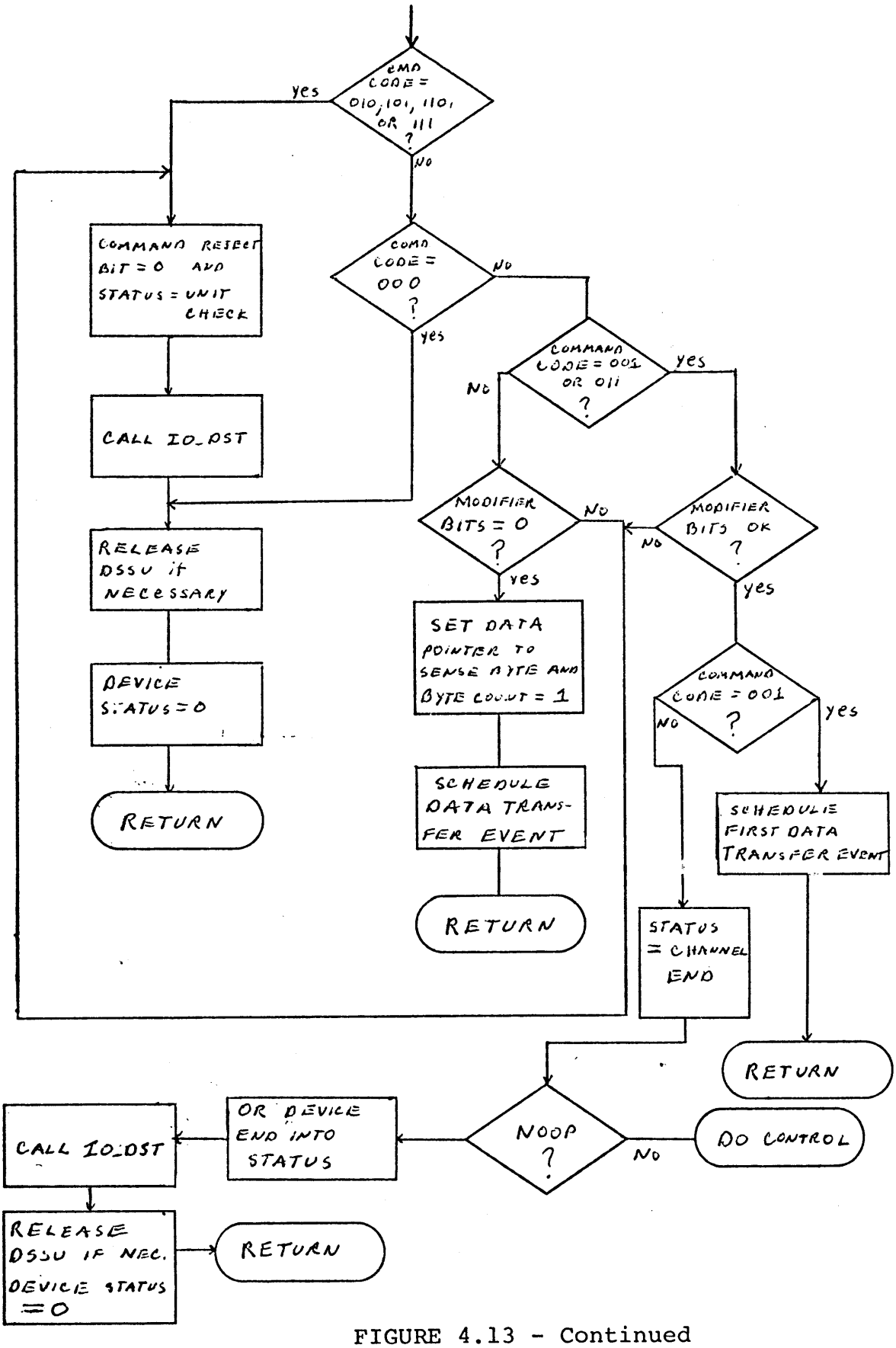

FIGURE 4.13 - Continued

bit is turned on in the status for the devices. If we do not have a data transfer or a device end event, the device status is checked. If the status is zero then the indication is to decode the CCW command code and execute it.

The BUSY bit in the device status is turned on and the three low-order bits of the command code (bits 0-7 in the CCW) are decoded to see what the command is. If decoding gives 000 for the three low-order bits, the routine will know that it is in the midst of performing a TIO. Since it is already known that the device status has nothing unusual, the BUSY bit is clear, the DSSU is released if necessary, and a return is made. This is similar to what would occur if the invalid command 010, 101, 110, or 111 is detected except that a unit check condition is sent to the channel and the command reject bit is set in the SENSE byte.

If the three low-order bits decode to a write (001) or a control (011) operation the modifier bits (first 5 bits of command code) must have a value between 0 and 3 or between 17 and 28. For a write command, a printer event is scheduled to perform the first data transfer. For a control operation, a channel end condition is indicated in the device status. If it is a NO-OP then the device

end condition is also indicated in the device staus, and the status is sent to the channel. Otherwise, the specified control operation is performed (see figure 4.14).

If, originally, there was a non-zero status then it is sent to the channel. If BUSY is the only bit on, a return is made, otherwise, a check is made for the presence of the device end condition and if found, no bit is left on in the device status. If it is not found only the BUSY bit is left on in the status for the device.

4.1.8.1 Perform Data Transfer

This subroutine, flowcharted in figure 4.14, is entered when a byte of data needs to be transfered to, or from, main memory. If there is some unusual status at the device (ie. status other than BUSY) then that status is sent to the channel. If the DEVICE END bit is on the device status is cleared, if necessary the DSSU is released and a return is made.

If there is no unusual status and the byte count is zero then one of two things happens. If the sense byte has just been transfered then Channel End and Device End are indicated in the device status and the action described above for unusual status is taken. Otherwise, the channel end condition is sent to the channel, the buffer is written into an output file along with any specified control motion (ie. line skips), and a device end

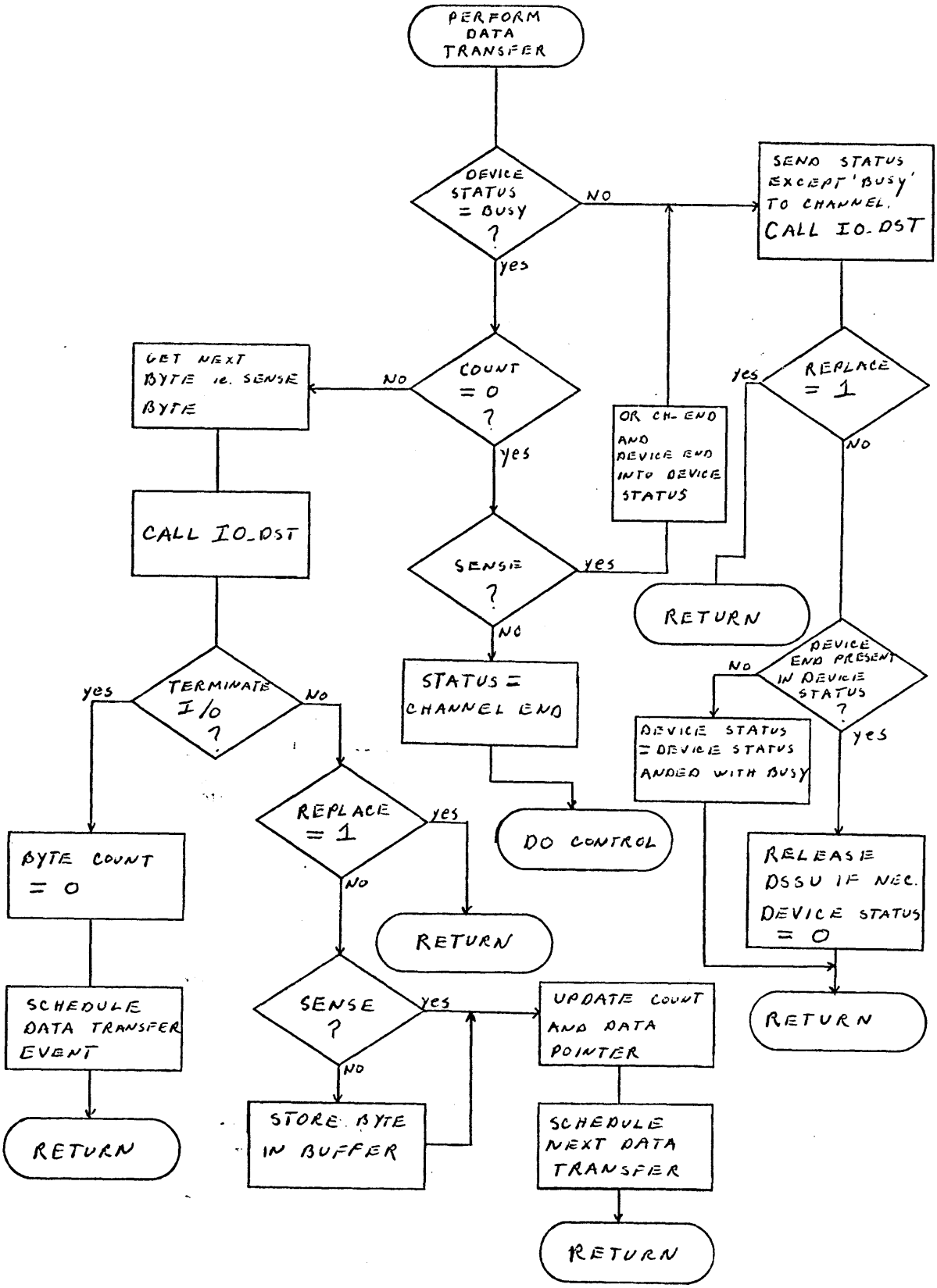

FIGURE 4.14 - Perform Data Transfer
(continued on next page)

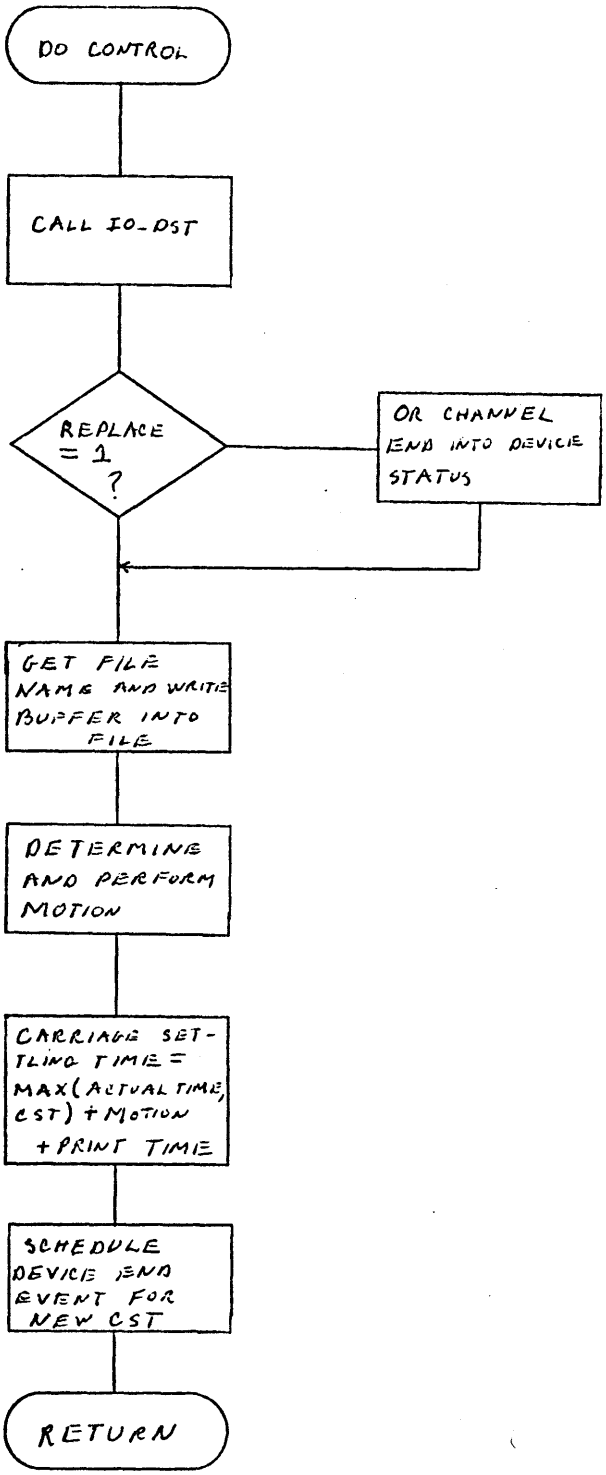

FIGURE 4.14 - Continued

event is scheduled.

If the byte count is not zero the data transfer is performed and the byte fetched during the call to I/O DATA_STATUS TRANSFER is stored in a buffer. Since the sense command is treated by I/O DATA_STATUS TRANSFER as a read operation (ie. storing the byte at an appropriate address) it should be noted that there is no need to store it in a buffer. The count and data address are updated, the next data transfer is scheduled and a return is made.

To terminate I/O the byte count is set to zero and another data transfer is scheduled. At the next attempt at a data transfer, the zero will be detected and I/O will go to a normal termination.

4.1.9 Card Reader Routine

The flowchart for a simulated 3505 type card reader is given in figure 4.15. The throughput rate simulated is 800 cards per minute (75 ms/cycle). The clutch access time is incorporated by taking into account the clutch decision print (CDP) where the CDP is the point in time after which a delay of 25 ms will occur before the start of the next feed.

This routine is entered for the same conditions as the printer routine and, as can be seen, there is much overlap between the two routines in how data is handled and status is transferred. Upon decoding the three low order bits of the command code if we find a sense command four bytes of data will be transferred to memory as compared to one with the printer. For a feed, select stacker command, first a sequence check is made. This command must

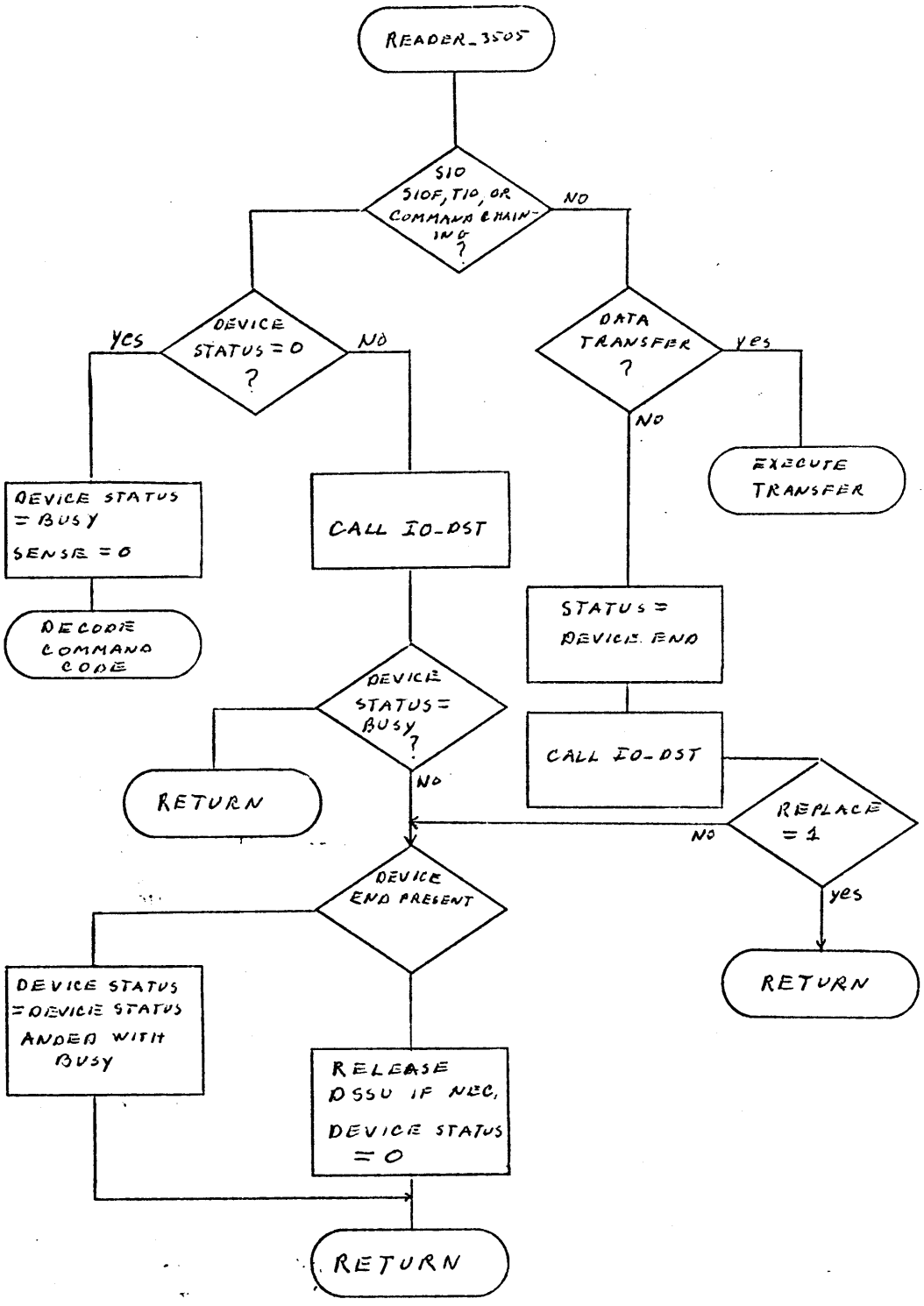

FIGURE 4.15 - Card Reader Routine
(continued on next page)

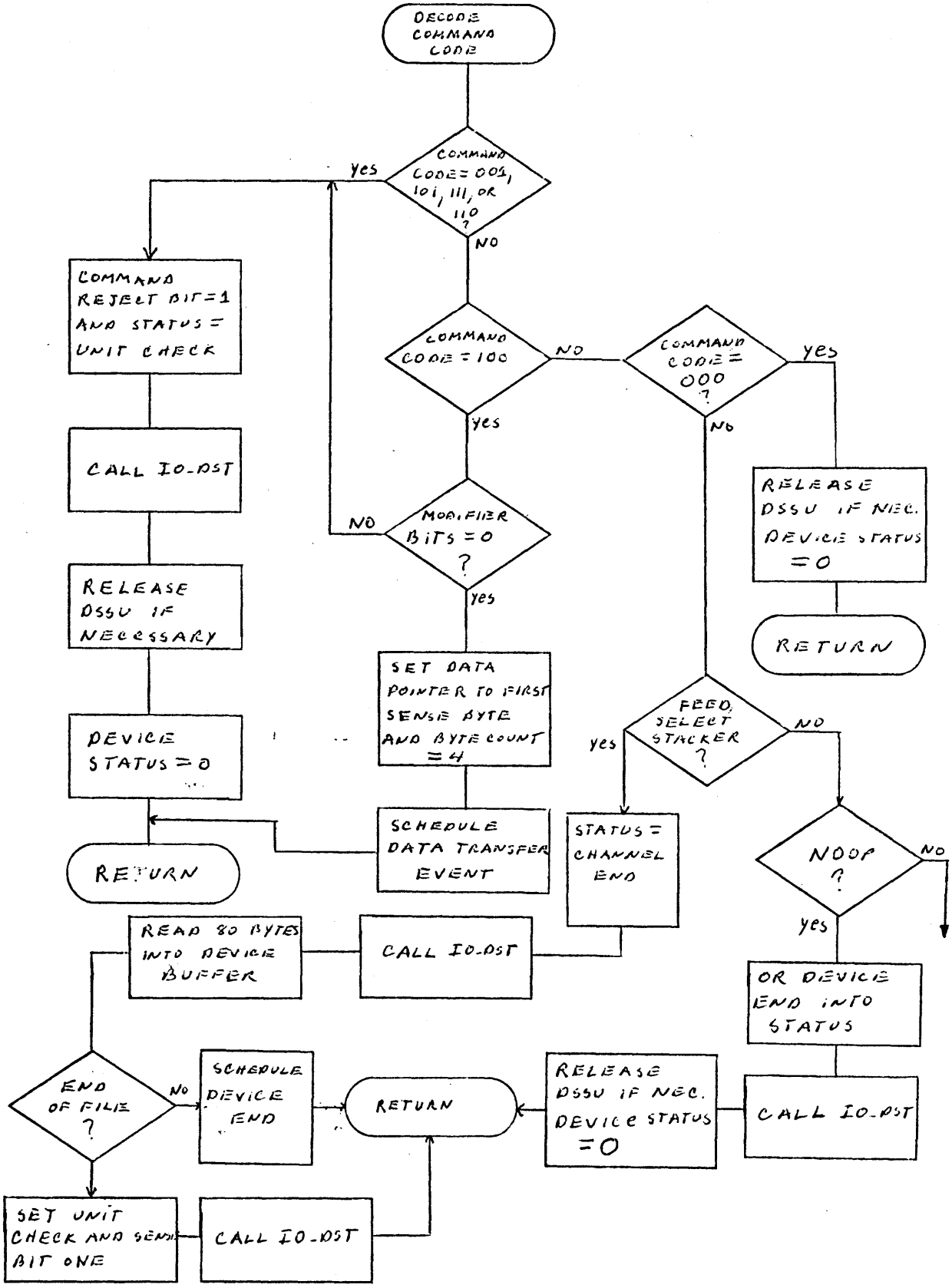

FIGURE 4.15 - Continued
(continued on next page)

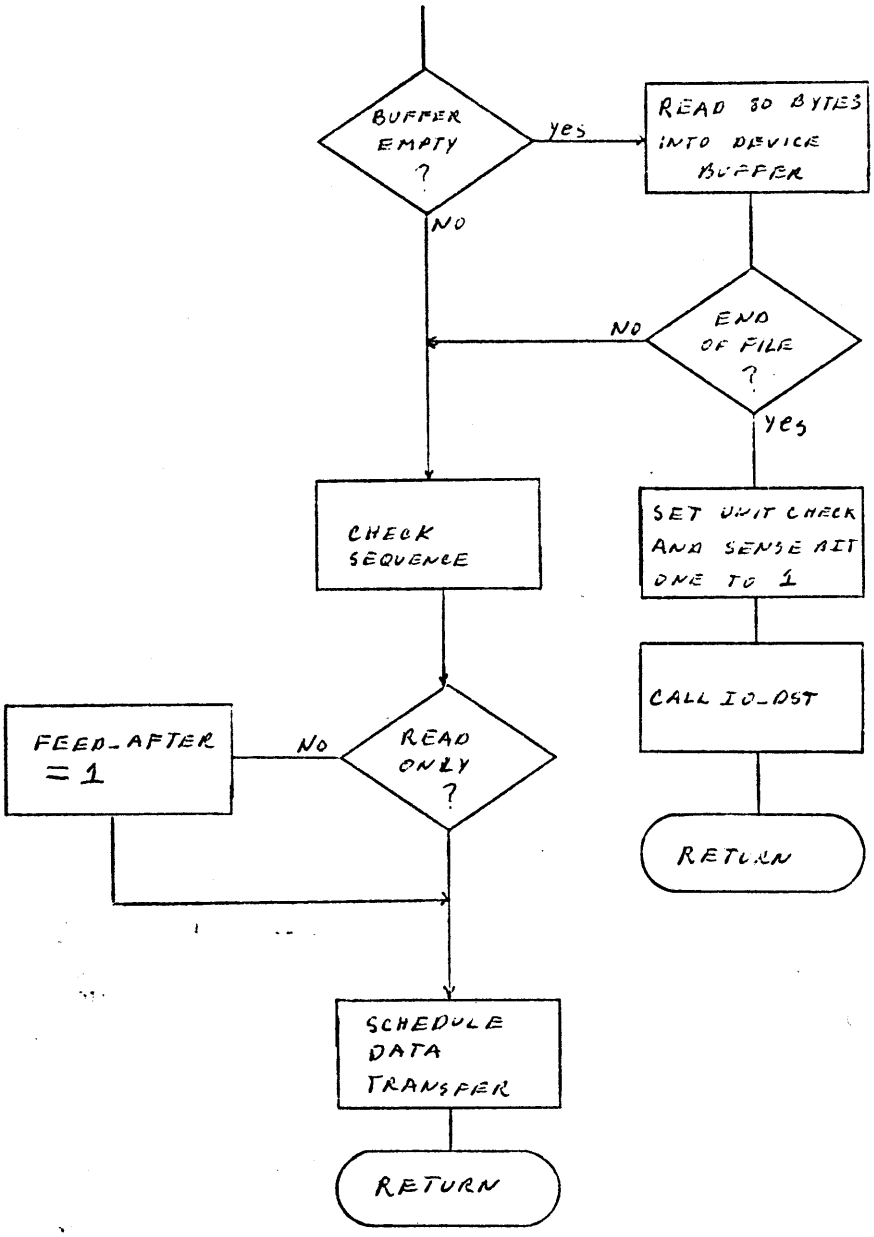

FIGURE 4.15 - Continued
(continued on next page)

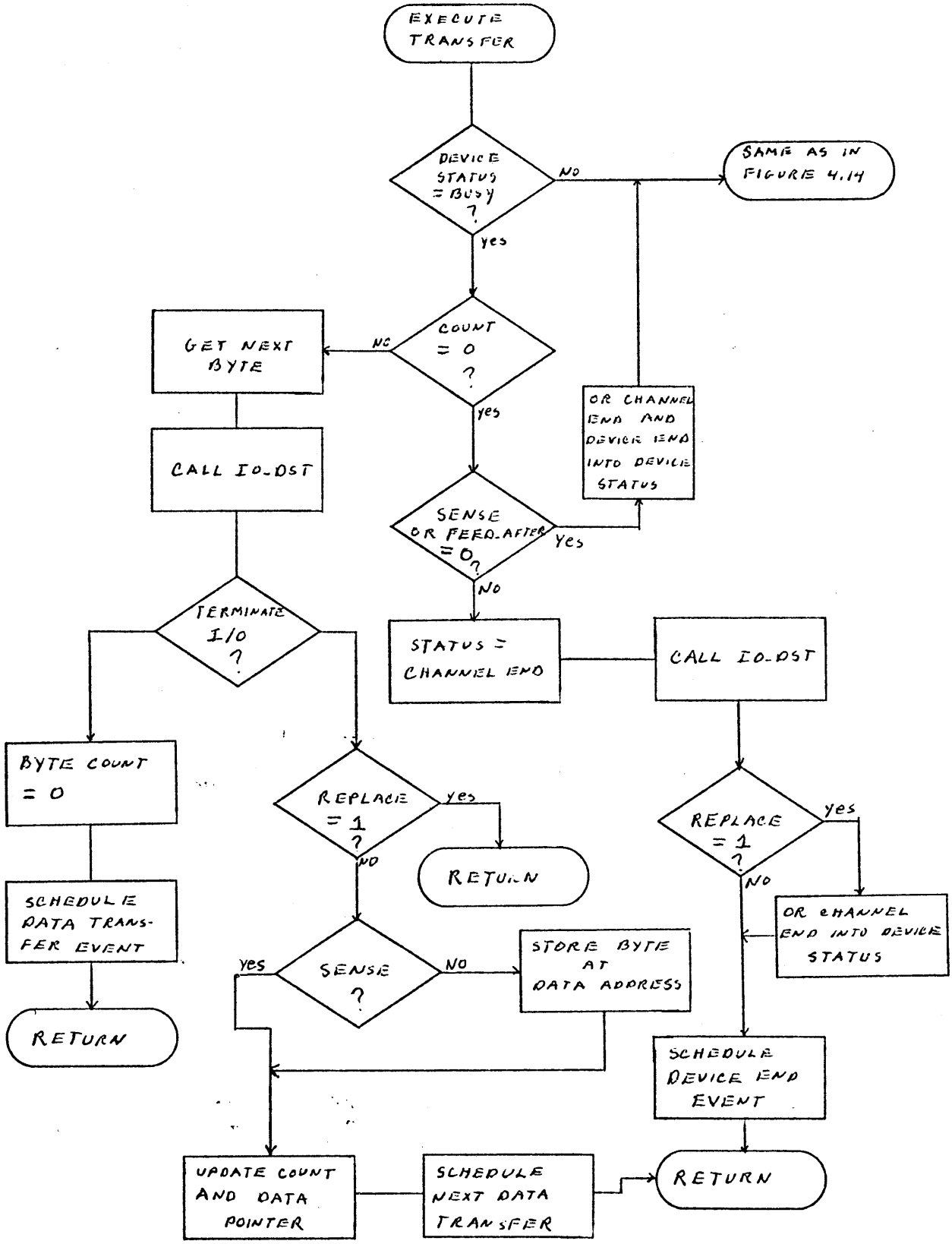

FIGURE 4.15 - Continued

follow a read only command and cannot be the first command given to the reader. Next, a Channel End is sent to the channel and a 'card' 80 bytes of data, is put in the device buffer. If there is no error a device end is scheduled to occur 67 ms from the present time plus any clutch access time needed. Of course this depends on the time the previous feed command ended which is saved in the device working register.

If we have a read command the sequence check is again made. A read only command follows a read, feed, select stacker command, a feed select stacker command or itself without command reject occuring, but if it follows itself, a unit check is indicated and unusual command sequence (sense bit 6 bytes 0) is returned on a subsequent sense operation. A read, feed, select stacker command can only follow feed, select stacker command or itself. If it follows a read only command unit check and sense bit six of byte 0 are set. FEED_AFTER, when set equal to one indicates we have a read, feed, select stacker command. It is stored in the device working register. The data transfer is then scheduled.

Upon completion of the data transfer (ie. byte count 0) CH_END conditions are sent to the channel and for a read only a device end is also indicated. For the read, feed select stacker a device end is scheduled.

4.1.10 I/O Interrupts

Figure 4.16 indicates what is done in the ACCEPT portion of the interrupts routines as far as I/O is concerned. Checks are

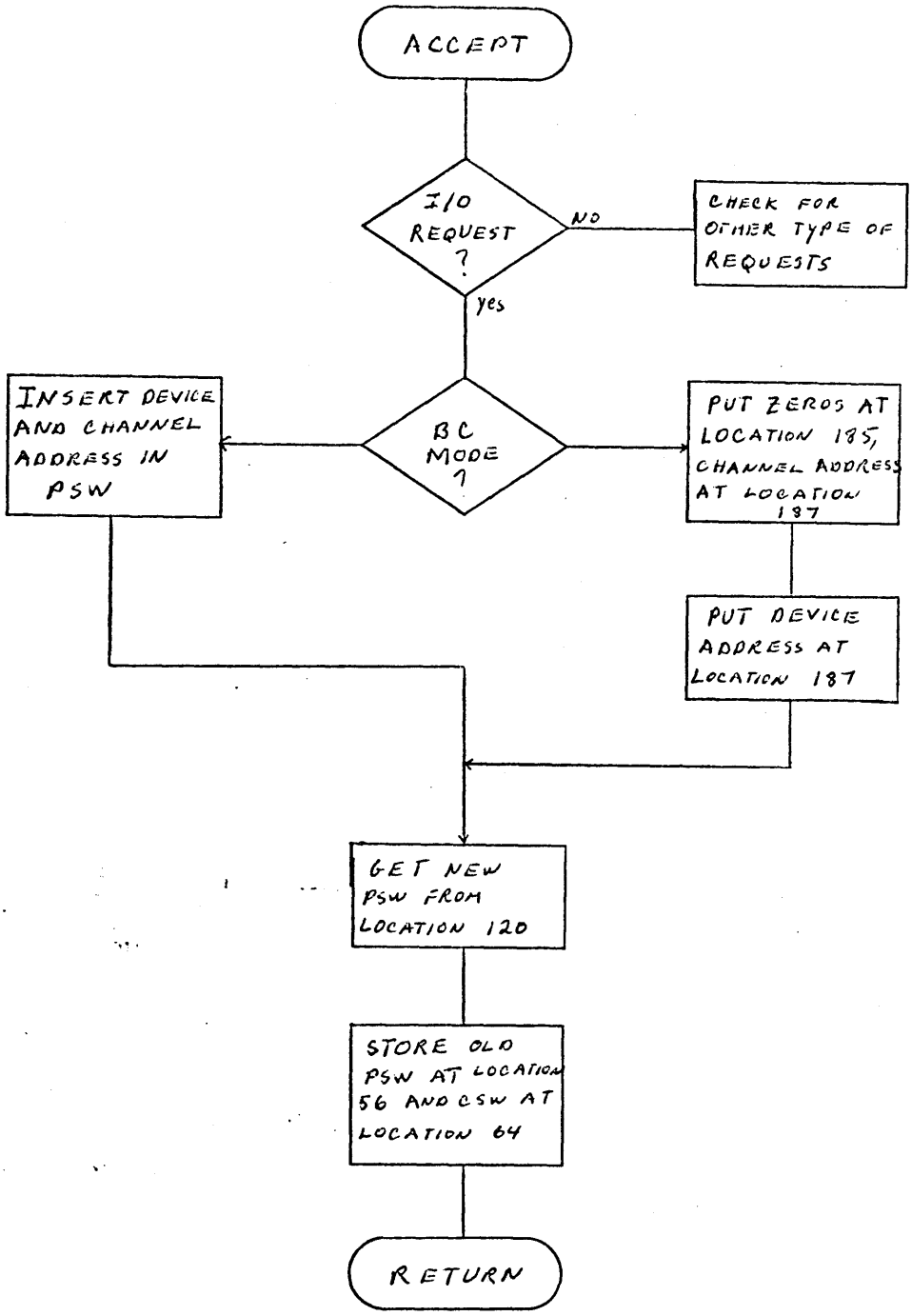

FIGURE 4.16 - I/O Portion of ACCEPT Routine

made to see whether operation is currently in basic or extended control mode (BC or EC). Depending on the control mode the action taken is that which is described in S/370 Principles of Operation in the section "Input/Output Interruptions".

4.2 System Structure and Flow

Figure 4.17 gives a feel for how the above describes routines are interconnected.

4.3 Sequence of Events for I/O Operation

Given below is what the sequence of events surrounding a SIO, TIO, and SIOF instruction would look like.

For a SIO:

1) An element is taken off of the queue and it is determined that it calls for CPU execution of an instuction.
2) Any outstanding interrupts are handling ny calling the ACCEPT routine.
3) It is determined that the CPU instruction is a SIO instruction and a branch is made to the SIO routine.
4) The SIO routine will schedule an "I/O to Begin" event and the CPU event will not be rescheduled yet.
5) The "I/O to Begin" event is eventually detected and a branch is again made to the SIO routine.

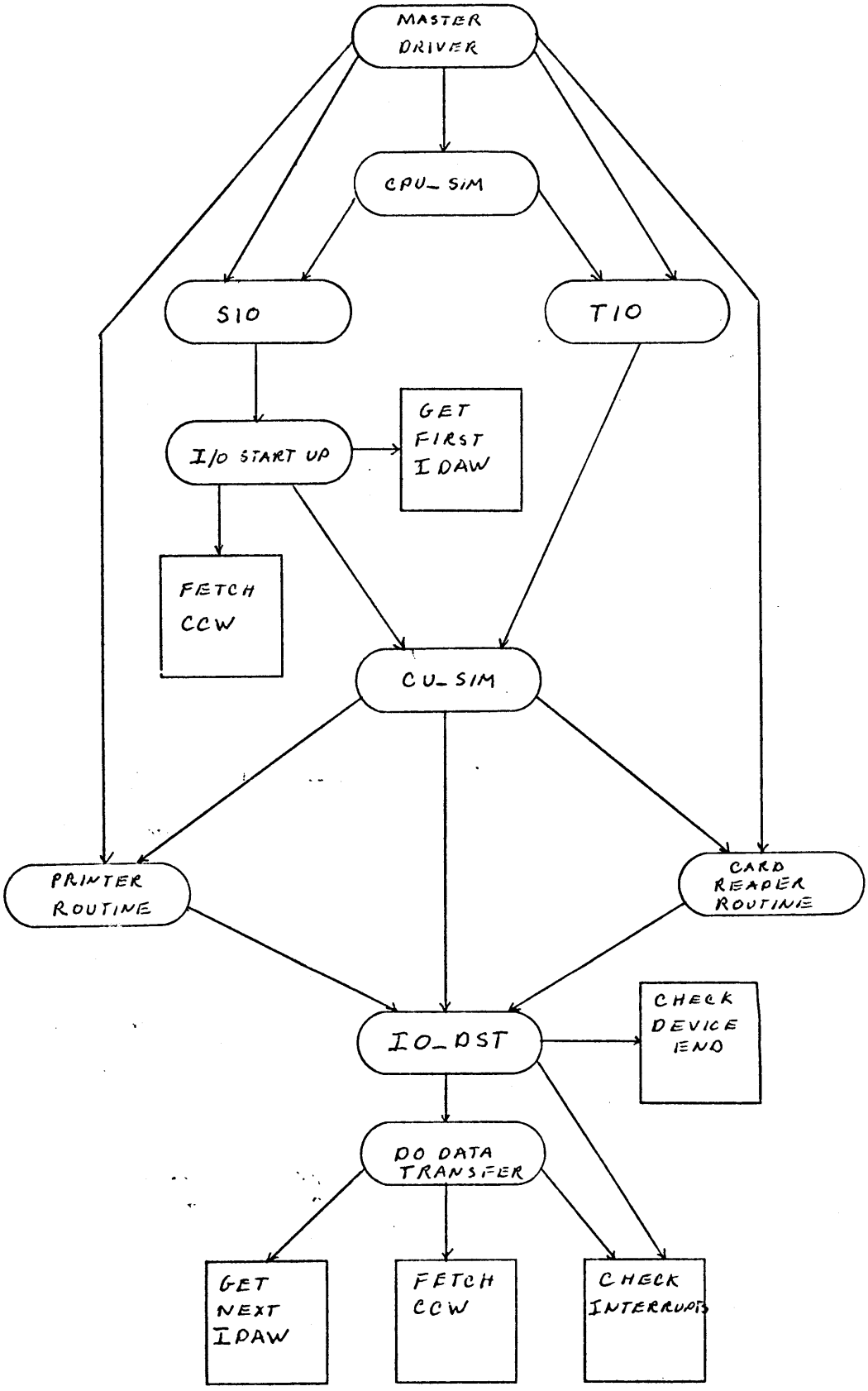

FIGURE 4.17 - Subroutine Interconnections

6) The channel and subchannel are tested for availability, and the CAW is fetched and tested. The I/O START UP routine is then called and upon return success or failure is indicated in the condition code and possibly the CSW.

7) The I/O START UP routine will fetch and test the CCW, and load it into the subchannel. CUSIM will be called to send orders to the device.

8) CUSIM checks the control unit and calls the approriate device routine.

9) For example, the printer routine would perform control operations or schedule the first data transfer and return.

For a SIOF instruction 1 and 2 would be the same and then:

3) It is determined that the CPU instruction is a SIOF instruction and a branch is made to the SIO routine.

4) The channel and subchannel are tested for availability, and the CAW is fetched and tested. Then a "SIOF" event is scheduled.

5) The "SIOF" event is eventually detected and a branch is made to the I/O START UP routine.

6) The I/O START UP routine will fetch and test the CCW, and load it into the subchannel. CUSIM is then called and upon return if anything is wrong an interrupt is scheduled and the deferred

condition code is set.

7) CUSIM checks the control unit and calls the appropriate device routine.

For a TIO instruction, steps 1 and 2 are again the same and then:

3) It is determined that the CPU instruction is a TIO and branch is made to the TIO routine.

4) The TIO routine tests the channel and subchannel for availability and then calls CUSIM routine to send orders to the device.

5) CUSIM checks the control unit and calls the appropriate device routine.

5. Conclusions

The I/O simulator provides much greater flexibility in the usefulness of the complete IBM S/370 type simulator. Increased operation at the bare machine level is readily available and should prove a useful improvement in the use of the simulator as a teaching aid and in the study of operating systems in general.

5.1 Suggestions for Improvement

1) Incorporating the idea of keys and the associated instructions (ie. IPK, ISK, SPKA, and SSK).
2) Implementing the suggested DASD.

Appendix A - Direct Access Storage Device

This description is not intended to be complete but rather to provide the reader or a person intending to implement the device with a neat and clear approach to the changes and additions necessary to add the DASD to the simulator. A detailed understanding of the IBM Systems Reference Manual for IBM 2835 Storage Control and IBM 2305 Fixed Head Storage Module is a prerequisite to implementation and more than a passing knowledge of direct access device operation has been assumed in the writing of this description.

The direct access storage device chosen for this example is the IBM 2305 Fixed Head Disk storage facility. The facility consists of an IBM 2835 Storage Control (control unit) and an IBM 2305 Fixed Head Storage Module. The 2835 interprets and executes commands from the channel, controls the channel and disk storage interface, furnishes status to the system, and performs error detection and correction. The 2305 responds to commands from the 2835, selects head, and reads or writes data. There are two models available, one having two read heads per track with approximately 5.4 million bytes per module, and the other having one read head per track with about 11.2 million bytes per module.

The major concern of the simulation routine for this device will be simulating the disk. The user must be able to format the tracks and have direct access to records. The system should require a minimum amount of the simulated disk in main memory in order to find any given record. In PL/1, record-ori-

ented transmission of input/output is well suited to these requirements, though for any given system suitable capabilities may be lacking and implementation of this type of device may be more difficult.

This discussion will only deal with the 2305 Model 2 which has 768 addressable recording tracks that the user sees as 96 cylinders having 8 tracks each. There can be a maximum of 47 records per track. For initial implementation in the simulated system, one might start with a scheme to simulate 10 cylinders with 8 tracks each and limit the number of records per track to 10. The record size could then be 1k - 14k bytes and the maximum number of records for the simulated module would be 800. These limits can easily be extended if necessary.

The working registers for the device would contain such information as the file mask, the 24 sense bytes, a buffer, the last cylinder address, the last track address, the last record address, the status for the device, and interrupt pointers. The device control block would have a pointer to the working registers along with such static information as track rotation time (10 ms), number of sectors (180), byte transfer rate (1.5 million bytes per second), device starting time, etc. The file mask is associated with the set file mask command which specifies the type of operations that can be performed in the given channel program. It is reset to zero after each chain of commands.

In creating or accessing a record, the use of keys plays

a major role. The key would be one of two things. First, it could be the key specified by the CCW during formatting of the track or, secondly, it could be the record ID which contains the cylinder, track, and record number. One of these two character strings will be the key that the device routine uses to access a record through keyed record-oriented transmission. Thus, when a track is formatted, the key that is used will always be the key that is specified in the formatting write commands unless none is given, in which case the record ID will be used as the key. The keys are stored in the following array structure.

```
DCL 1 TRACK (80),
      2 AREA (10),
        3 KEY CHARACTER(255) VARYING,
        3 EXIST FIXED BIN(31) INITIAL ((10)0),
        3 COUNT FIXED BIN(31),
        3 SECTOR FIXED BIN(31);
```

If EXIST is zero the corresponding record does not exist. If it is less than zero then the record has no specified key and the record ID is used as a key. This structure makes the search of a track for a record a simple matter and avoids the need of explicit index points for single track searches (on the 2305, if two index points are detected without completing a search then unit check is indicated along with channel and device end).

Similarly multi-track searches present no problems.

SECTOR, calculated during formatting of the track, is used with the read sector command and in rotational position sensing (RPS). RPS is easily implemented if one assumes that the track head was at sector zero at the start of the simulation and the start time is stored in the device control block. Then from a knowledge of the starting time, the present time, the rotation time, and the total number of sectors, the sector number at any given time can be determined. As an example of a procedure which uses RPS and contributes to increased channel utilization consider the following:

1) A set sector commands is given to indicate which sector number it is desired to reach.
2) The channel is freed until that sector is reached.
3) When the third sector in front of the desired one is reached (to allow for a channel reselection delay) a special device end is sent to the channel.
4) If upon detection of the device end the channel is available then the device end is accepted and the channel reconnected. If the channel is busy then the device end is rescheduled.
5) If the channel is still busy after the reselection delay has past then the device end is rescheduled for the next revolution.
6) The record is now ready to be read or written (ie. the next CCW can be executed).

Upon detection of a seek command, the seek address (cylinder and/or track number) is stored in the device working register and a channel end along with a device end are sent to the channel. For a search command, the key or record ID is saved in the working register and the array structure is checked to see if the record exists. If the search key command is specified, the cylinder and track number at which the scan through the array will start are those stored in the working register (last cylinder and last track address). Upon termination of the search a device end is scheduled for an appropriate time to reflect the simulated search (the device will appear busy to inquiries until that time) and the appropriate status is stored so that it can be indicated along with the device end when it occurs.

Of course other items as command prerequisites and file mask settings are very explicit and must be checked for during certain commands and appropriate status indicated, but the important point to note is that no actual input/output needs to be done for any of the validity checking or operations associated with the control, search, or sense commands. Operations are performed and device ends are scheduled solely from knowledge about the device and from information on hand. Only with the read and write commands need I/O actually be performed, and these operations can be handled in essentially the same manner as the printer and reader, previously described, handled them.

Appendix B - System Creation

To add a new device should present no difficulty. As seen previously, I/O devices have the same calling sequence and operate in basically the same manner with respect to data transfer to and from main storage. Since the other aspects of device operation can differ greatly, a clear understanding of both the I/O system, as described in this document and in S/370 Principles of Operation, and the device itself, through SRL publications, will be needed.

The format of the I/O System Configuration Cards is given below. It is assumed that the reader is familiar with the sections "Input/Output Device Addressing" and "Attachment of Input/Output Devices" in S/370 Principles of Operation.

1) CPU Card - (Card Type, Unit Number, Start Address, Trace information, prefixing information, Number of Channels)

2) CHAN Card - (Card Type, Channel Type, Number of Control Units)

3) CU Card - (Card Type, Low Address, High Address, DSSU Identification Number, Number of Devices)

4) DEV Card - (Card Type, Device Type, Mode, First Device Dependent Information, Second Device Dependent Information)

References

1 Simulator Program Logic Manual - L. Goodman and S. Madnick.

Operating Systems - S. Madnick and J. Donavan, McGraw Hill, 1974.

Simulation of a Multiple Processor IBM System/370 With Associated I/O Equipment - Thesis, W. Silver, May, 1975.

SIM360: A S/360 Simulator - Thesis, Wm. Mc Cray, May, 1972.

IBM System/370: Principles of Operation - Form GA22-7000-4.

IBM 1403 Printer Component Description - Form GA24-3073-9.

IBM 2821 Control Unit Component Description - Form GA24-3312-8.

IBM 3504 Card Reader/IBM 3505 Card Reader and IBM 3525 Card Punch Subsystem - Form GA21-9124-5.

OS PL/1 Checkout and Optimizing Compilers: Language Reference Manual - Form GC33-0009-3.

Reference Manual for IBM 2835 Storage Control and IBM 2305 Fixed Head Storage Module - Form GA26-1589-3.