UNIVERSITY OF MARYLAND
COMPUTER SCIENCE CENTER

COLLEGE PARK, MARYLAND

FACILITY FORM 602
N70-36769
(ACCESSION NUMBER)
(THRU)
119
(PAGES)
1
(CODE)
CR-112367
(NASA CR OR TMX OR AD NUMBER)
08
(CATEGORY)

N70-36769

AN IOCS ALGORITHM FOR MICROPROGRAMMING

Jeffry W. Yeh

University of Maryland
College Park, Maryland

July 1970

Technical Report 70-124 July 1970
NGR-21-002-206

An IOCS Algorithm for Microprogramming

by

Jeffry W. Yeh

This research was supported in part by Singer-Link Research Assistant Scholarship in Computer Science and by Grant NGR-21-002-206 from the National Aeronautics and Space Administration.

Abstract

An Input-Output Control System (IOCS) initiates and controls the input and output processes of an operating system, thereby making it unnecessary for the user to recode any of these processes. Input-Output Control Systems usually perform the following functions: (1) file and buffer handling for the creation and maintenance of the file, the buffering of the input-output data, and the blocking or deblocking of the records; (2) input-output scheduling for the examination of the result of an I/O activity and the determination of the next I/O activity; (3) generation of the actual I/O programs, including the channel programs.

This report presents a tree-structure design of an IOCS, using double-buffers. The design includes a set of macro instructions and a set of algorithms. There are three levels in the tree-structure: the first level deals with file handling and buffering; the second level with I/O scheduling; and the third level with the device drivers. Special emphasis is placed on the design of the file and on buffering, employing double buffers for files, variable lengths for buffers, and a rotation method for buffer usage. All algorithms are presented in the form of flow charts, including an overall flow chart for the IOCS and 16 flow charts for individual algorithms. The purpose, the major objectives, the input and output, as well as the calling sequences, are stated for each flow chart.

The algorithms are prepared as to be easily convertible into sequence charts which in turn can be described in terms of Computer Design Language (CDL) statements for simulation by the CDL simulator and eventual implementation by microprogramming.

Table of Contents

An IOCS Algorithm for Microprogramming

Jeffry W. Yeh

1. Summary

This paper is a report on a study of an Input-Output Control System (IOCS). An overview of the IOCS is presented in Section 2.. The purpose, advantages, and functions of IOCS are presented in this Section. From the studies of several Input-Output Control Systems, such as the Input-Output Control System of the IBM 7000 series, the IBM 1400 series, and CDC 3000 series, a Simplified Input-Output Control System (SIOCS)has been designed. This design is presented in Sections 3 through 6.

In Section 3, the design goal and principles of SIOCS are discussed in detail with special emphasis on the evolutionary development of SIOCS. The macro-instructions of SIOCS is discussed together with several concrete examples.

Section 4 presents the functions of SIOCS. These functions are separated into three parts based on their levels of tree-structure. These are: the file and buffering algorighms, the I/O scheduling algorithms, and the unit interpretive algorithms. This section places the greatest emphasis on the file and buffering algorithms. The concept of the file and buffering and the algorithms used in handling double buffering are described in detail. The structures and formats of internal control blocks and I/O tables are presented together with a sample network of these tables and blocks.

The algorithms of SIOCS are presented in Section 5. It consists of an overall diagram for the algorithms of SIOCS together with a series of flow charts for all algorithms in SIOCS. For each flow chart, the purpose, the major objectives, the inputs and outputs, and the calling sequences of the algorithm are described in detail. These algorithms are devided into three parts according to the functions of the SIOCS, and so presented that they could be converted into sequence charts for eventual implementation by microprogramming.

In Section 6, a discussion of this study is presented. This discussion includes the remarks on the designing and microprogramming of the SIOCS.

2. Overview of Input-Output Control Systems

In the simplest digital computers, input or output operations cause computer processing to be suspended while the input/output (I/O) is in progress. In this case, no problem of synchronization or overlap of I/O time with computing time need concern the programmer. There is no way to conduct more than one I/O operation at a time on such an elementary machine configuration.

Most modern computers are much more sophisticated and powerful. They have data channels that allow one or more I/O operations to be processed simultaneously with the Central Processing Unit (CPU). However, this is possible in those programs which have segments of code that perform the following functions before an I/O operation is executed:

(a) Test and determine whether the I/O device is busy or ready to be used.
(b) If the I/O device is busy, then either transfer control to the proper routines or keep waiting until the I/O device is free.
(c) If the I/O device is free, then initiate an I/O operation and jump back to continue processing.
(d) When the I/O operation is finished, notify the user of this fact.

The programmer must be assured that the piece of data to be used in a computation has already been read in before the computation is initiated. It is quite obvious that the programming required to produce this assurance will increase the I/O preparation time and the problem of making input/output execution efficient will become much more complicated. A solution involves, at least, answers to the following questions:

(a) How can the total I/O operation time (including the device preparation and data transmission time) be minimized without wasting core storage?
(b) Under what circumstances and with what techniques can operations be made asynchronous?
(c) When is the proper initiation time for an I/O operation?

The answers to these questions involve sophisticated programming and are required in order to get maximum use of the hardware. Since these programs ought to be available to every programmer, and since it is beyond the need and/or skill of an average programmer to provide his own solution, a centralized solution has been developed. That is, to provide the programmer with an Input/Output Control System (IOCS) Which would be core resident and always available to every program.

2.1 Purpose

The IOCS eliminates the time and expense involved in writing special I/O routines and allows programmers to concentrate their efforts on the processing of data. The programmer need not concern himself with the intricacies or increasingly complex input/output hardware. Instead, he is free to write his internal process as efficiently as possible. He need only see the records that are made available to him when he issues simple requests.

The IOCS may be considered as an interface between the input/output devices and the processing program. It provides the following features:

(a) Simple manner for handling complex I/O operations,

(b) Reading/writing of data records on input/output units concurrently with processing,

(c) Scheduling the I/O operations and I/O devices.

The relationship of IOCS to the operating system and input/output devices is shown in Figure 1.

2.2 Advantages

In addition to those which we mentioned above, IOCS offers the following significant advantages:

(A) I/O operations which are easy to learn and to program

IOCS provides standard input/output routines and formats. A programmer with little training in the capability of data channels, buffering techniques, or other techniques which make input or output operations efficient, can still write efficient I/O programs by using IOCS. The following example indicates the steps needed for iniating an I/O operation within the program, where (1) the IOCS is not used, and (2) the IOCS is used.

Examples:

(1) The steps needed to initiate an I/O operations within a program not using IOCS:

(a) assign a unit to be used

(b) select a channel

(c) test the channel status

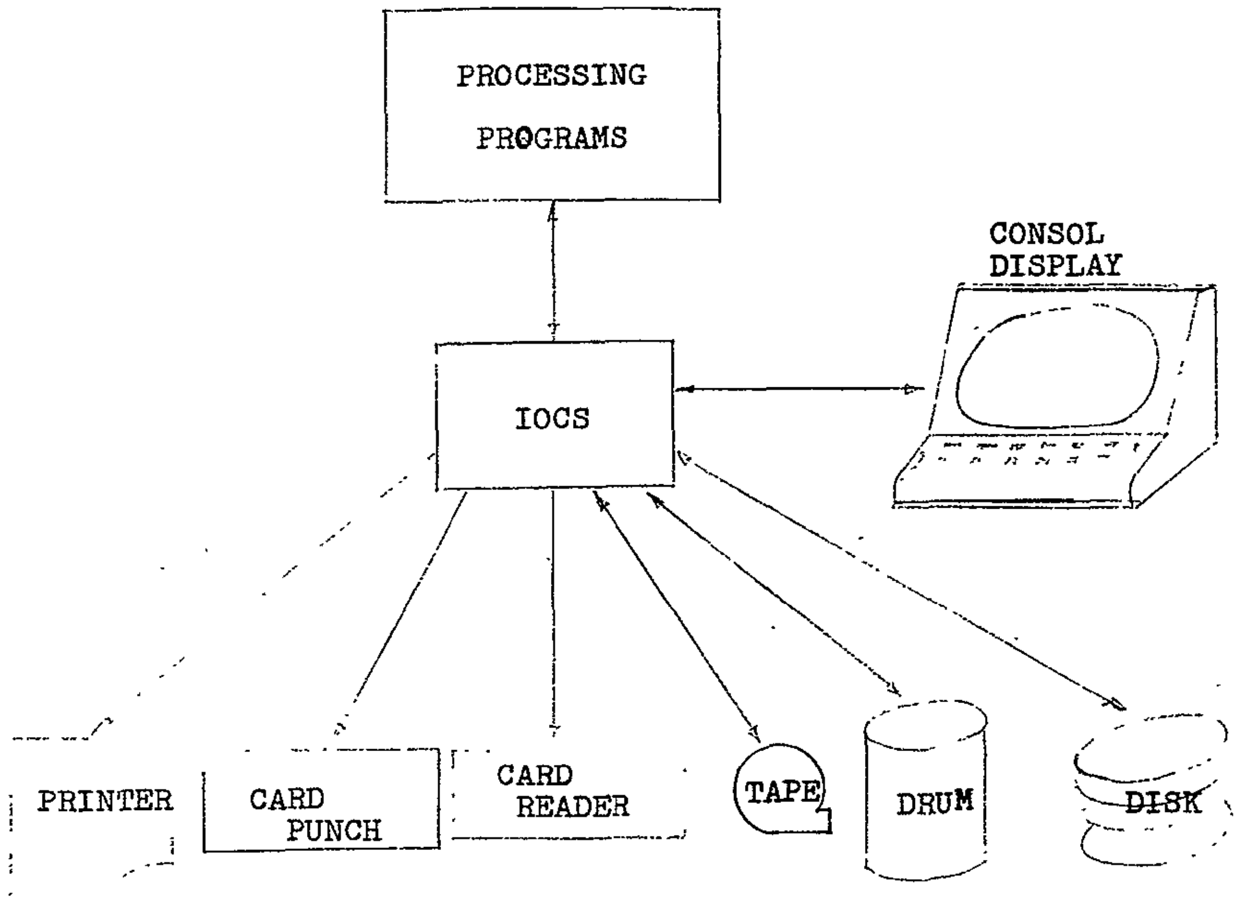


Fig. 1. The Relationships of IOCS to the Processing Programs and the I/O Devices.

(d) if the channel and unit are available for this program then connect it, otherwise either wait or transfer control to some proper routine,

(e) set up the I/O instructions and channel program,

(f) initiate the I/O instructions (set up by step (e)), when the channel and unit are connected by step (d).

(2) The steps needed to initiate an I/O command within a program using IOCS:

(a) open a file, declare the file name, file type, and device type,

(b) issue a simple I/O macro-instruction (e.g., READ, WRITE,...)

(B) IOCS Provides for Asynchronous Operations

By using an input buffering technique, IOCS allows the system to read ahead on input devices, thus diminishing waiting time. Output buffers are used to store records to be transmitted to devices currently in use without holding up computation. Also, with the assistance of the I/O interrupt routines, creation of fully overlapped I/O buffering is allowed without requiring waiting loops to process the buffered operations.

(C) Symbolic Addressing of Files and Units

Symbolic addressing allows the user to communicate with I/O devices in a very convenient way. The user creates a file by telling the system the file name and the devices which are associated with that file. Future I/O references to the file need only specify the file name. The system will match names and then will perform the I/O operation. The symbolic addressing of units also allows for flexibility when the program must be executed under a different configuration.

(D) IOCS Affords Flexibility of Operation

If a specific input-output device which a program is expected to use is out of commission or not available at the time the program is to be executed, the IOCS assigns an alternate device to it. Also, if a program expects a particular type of device to be used and does not care which actual physical unit it is, IOCS will assign an appropriate available unit to it.

2.3. Terminology

In order to discuss the functions of Input-Output Control System in some detail, the terminology used in the description of an input-output system first must be introduced and then defined with precision. Then, the general aspects of an Input-Output Control System can be discussed briefly.

The terminology presented in this section is in common use and can be interpreted reasonably precisely in the case of any given machine. These terms are classified under three major groupings:

2.3.1. Software Terminology

(a) Random and sequential input/output calls: Sequential calls include calls for the next record, message, character, etc., as well as calls for spacing and backspacing. Random calls include calls for data in nonsequential order. For example, a call to backspace the tape **file** is a random call.

(b) Record and block: The information is often written as a sequence of words or characters separated by gaps. These contiguous sequences will be called a record. A record of maximum size (when such a maximum exists) is called a block. A logical record is the sequence of related data items that the program logic treats as a record. A physical record is a set of adjacent data characters terminating with an end-of-record indicator.

(c) Blocking and unblocking: Blocking is a method of compressing data that would normally appear in several physical records into a single physical record. It is normally used in transcribing data from one physical medium to another. For example, punched cards as a primary input to a system normally is transferred immediately upon reading to an auxiliary memory device, such as magnetic tape or drum. These latter devices have characteristics that favor larger physical records than the 80 column punched cards. Thus, the information in several cards (perhaps 10) is combined, this is blocking. Unblocking is the reverse process.

(d) File: A group of records is called a file. When reading input, end-of-file is the condition that is recognized as the end of the group; for output, the end-of-file condition is written in order to delineate an output group. Since the word file is also used in a logical sense, we need two terms, logical file and physical file. These are defined analogously to the logical record and physical record.

2.3.2. Hardware Terminology

A block diagram of an input-output hardware configuration as shown in Fig. 2 will assist in the clarification of the meaning of the subsequent hardware terminology.

(a) Channel: A channel is a hardware device which is employed to transmit both control information and data between a controller and the computer. The channel must be able to inform the processor of error conditions or termination of an operation. Note that the channel is parallel computer.

(b) Controller: A controller is a hardware device which is used for selecting a satellite unit, and relaying the control orders to this particular unit. (e.g., rewind, eject sheet, read forward, position access mechanism to a given address, etc.), and transmitting data between the selected unit and the channel. It must also be able to relay exceptional or normal conditions (e.g., parity error, end-of-record, unit busy, etc.) back to the machine via the channel.

(c) Unit: A unit (I/O unit) is that part of the computer system which introduces data into or extracts data from data storage. For example, magnetic tape unit is used to send data from tape to memory or to record data from memory onto tape.

2.3.3. Terminology Used to Describe Input/Output Techniques

(a) I/O instruction, Channel command, or Control order: I/O instructions are those instructions which are interpreted and executed by the central processor. Channel commands are those words that initiate and control the action of the channel itself. Controller order are those words that initiate and control the action of a controller.

(b) Channel Program: A channel program consists of one or more channel commands that control a specific sequence of channel operations. Execution of the specific sequence is initiated by a single start I/O instruction.

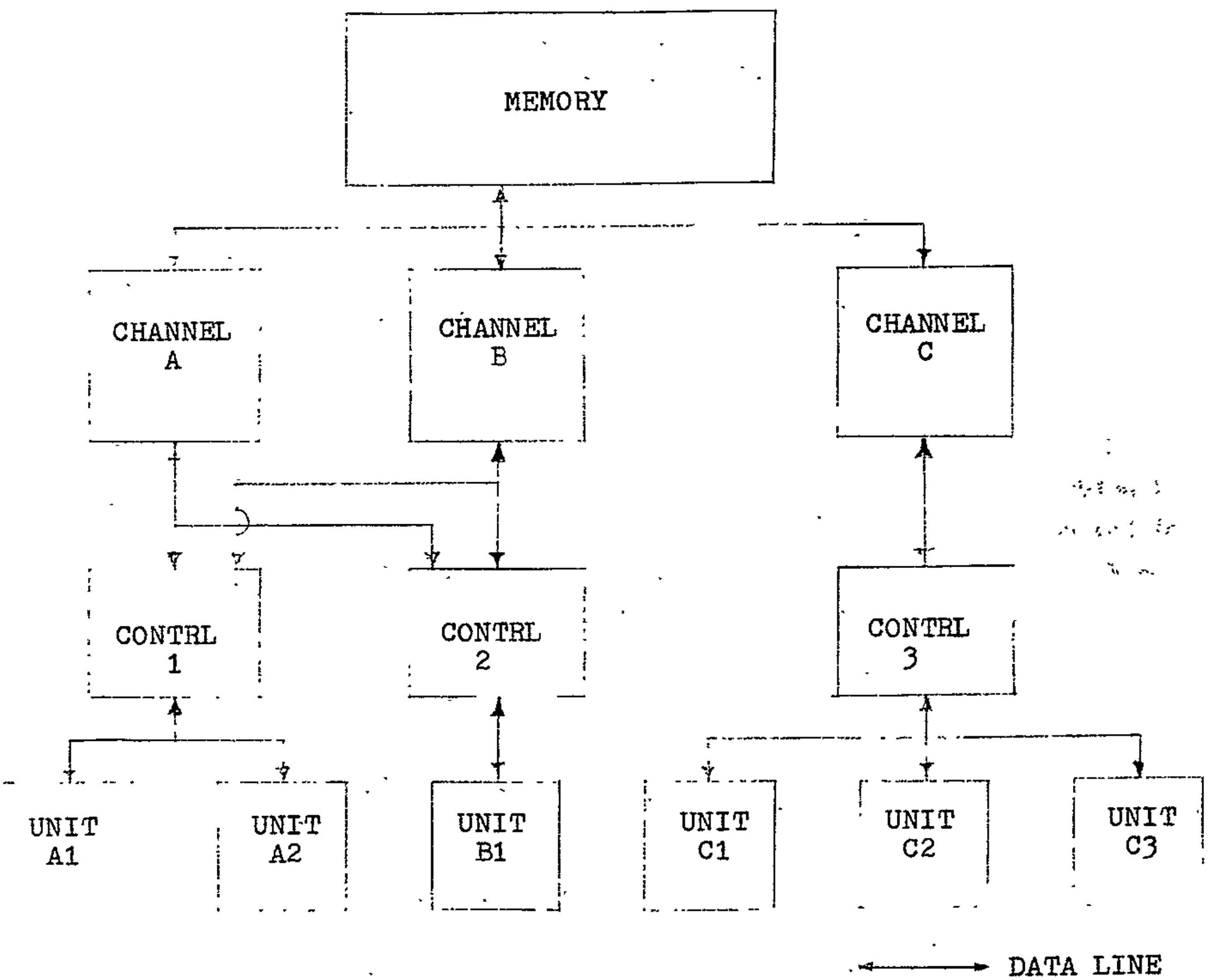


Fig. 2. An Example of The Flow of Information Through The Input-Output Hardware

(c) Buffering: An area of memory which is used to store temporary data during a transfer of information to or from an I/O device is called a buffer. Buffering is a technique which uses the storage buffers to compensate for a difference in data handling rates when transmitting data from one device to another, or to compensate for the difference in physical size natural to different hardware devices. Note that blocking is a form of buffering.

(d) Synchronous and asynchronous input/output: These are two basic operating modes for any particular input/output system. In the synchronous I/O system, the physical transaction associated with a user's input/output statement (or instruction) is carried out during the statement's execution. Control is not returned to the program until the actual transaction is completed. In an asynchronous system, the physical input/output transactions are not necessarily synchronized or interlocked with the execution of a user's input/output statement.

(e) Interrupt: An interrupt is a break in the normal flow of an instruction sequence such that the flow can be resumed from that point at a later time. An interrupt is usually caused by a signal from a source external to the Central Processing Unit. The interrupt causes an automatic transfer to a preset storage location, where action is or where some other appropriate action is taken.

(f) Trap: A trap is an automatic transfer of control to a known location. This transfer occurs when a specified condition is detected by hardware. A trap is different from an interrupt in that it is caused only by the Central Processing Unit, the program, or some internal event. When a trap condition is detected and the corresponding trap is called for, a transfer of control to a hardware-designated location occurs. Simultaneously the location from which the trap occurred is recorded. The hardware-designated location usually contains a transfer to the proper trap-handling routine, or it may ignore the trap instruction.

2.4. Functions

Having introduced the terminology, the functions of IOCS can now be described.

2.4.1. Input-Output Buffering Routines

The input-output buffering routines is based on the characteristics of the following:

(a) A standardized set of physical and logical formats.

(b) A set of internal tables describing the current status of internal buffers and the buffers themselves.

(c) A set of management routines maintaining the information contained in the internal tables. The block diagram in Figure 3 shows how the characteristics of the items described fit into the flow of the buffering routines.

Figure 3 illustrates the interactions within the buffering system, where the parameter names are explained in Table 1. Now, let us consider some typical operations as they might occur. The user's program obtains information from input unit A-2 (e.g. channel A unit 2) by calling the READ subroutine. Information is obtained from the input-processing buffer immediately. If this buffer does not contain all the information requested, it is emptied (i.e., all the data is transferred from the processing buffer into use's working area) and the next quiet buffer is called up to replace the present processing buffer and the remaining information required is obtained from this buffer. Meanwhile, the present processing buffer is placed in the available buffer pool. At this time the DISPATCHER may be notified by the 'critical amount of buffers' (sometimes called the CRITICAL BUFFERS) routine that another buffer is ready for input data from a device. Note that the DISPATCHER is the routine which manages the status of the buffers in the system. The CRITICAL BUFFERS routine is used to manage the status and the number of the quiet buffers in the quiet buffer pools.

On the other hand, the user's program may send the information to the output unit B-2 (e.g. channel B unit 2) by calling the WRITE subroutine. Information is transferred to the output-processing buffer immediately (see the right half part of Fig. 3). If this output-processing buffer cannot hold all the information requested, the buffer is filled and then place it

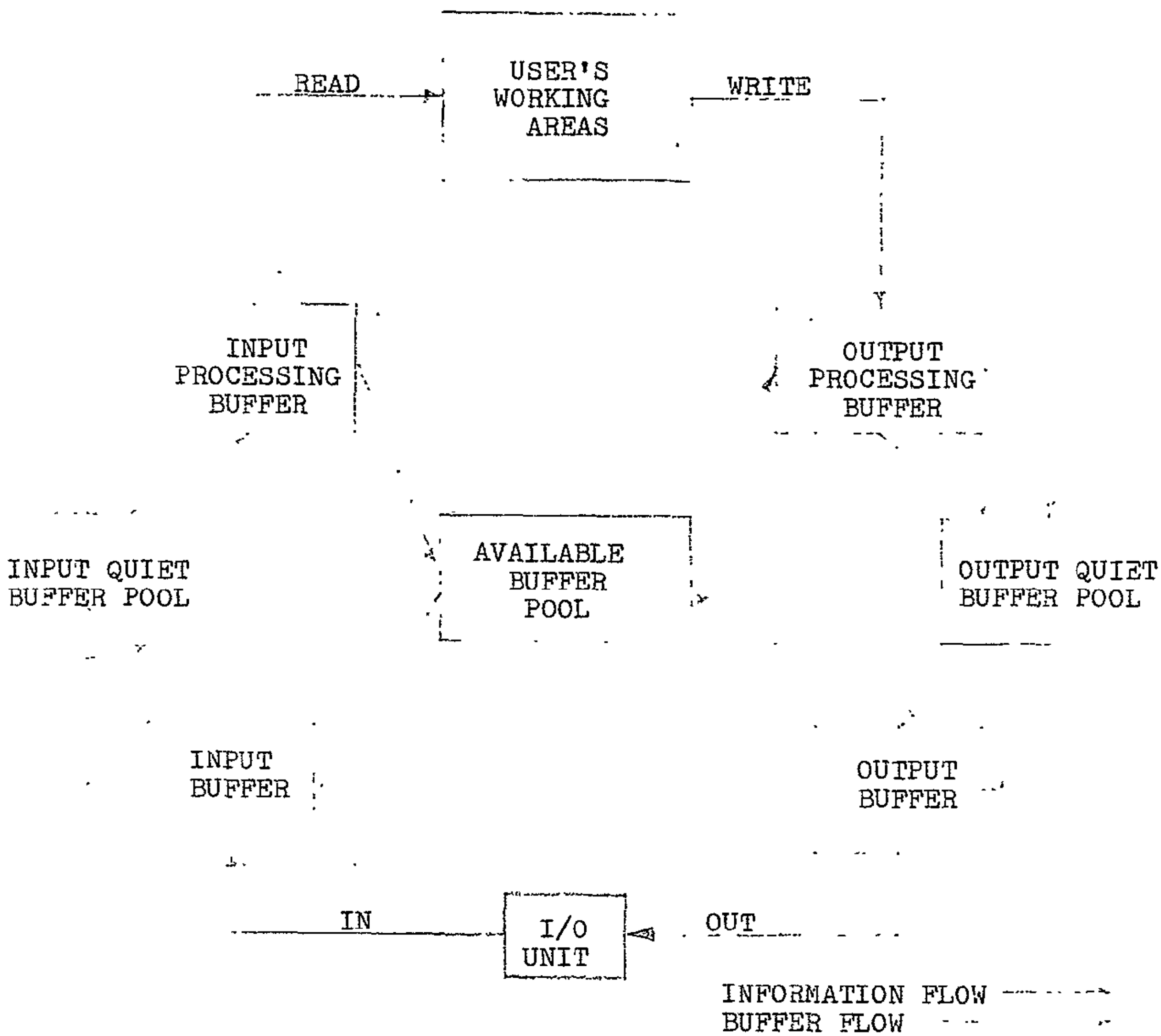


Figure 3. Block Diagram of The Buffering System

Table 1. Terms of the buffering system in Fig. 2

Term	Explanation
Working storage	That area utilized by the customer for program data, intermediate and final results.
Processing buffer	A buffer unit to or from which the user's program is in the process of transmitting data.
Input (output) buffer	A buffer unit currently being operated upon (read into ro out of) by one of the channels.
Quiet Buffer	A buffer unit containing current information coming from or being sent to one of the physical input-output units but which is currently activation.
Available buffer	A buffer unit not currently employed.

into the output-quiet buffer pool. Meanwhile, the next available buffer is called upon to hold the rest of the information. After the completion of the above operation, the DISPATCHER may be notified by the CRITICAL BUFFERS routine that another buffer is waiting for output. Note that the CRITICAL BUFFERS routine activates the DISPATCHER only when the buffers in the input (output) quiet buffer pool reach a predetermined critical amount.

2.4.2. Input-Output Scheduling Routines

The input-output scheduling routines can be divided into two parts, namely, the I/O initiation routines and the I/O completion routines (or I/O Executor).

The I/O initiation routines are activated when an I/O operation is required by a user's program (this includes the supervisor's routines). The I/O initiation routine determines the nature of the I/O request, and checks the availability of the requested I/O device. If the I/O facility is available to perform that function then the I/O action is initiated. If the I/O facility is not available, then one of two actions occur, either the system waits until the facility is available or it puts the I/O request into a waiting queue for initiation at a later time. Note that the term ' to initiate an I/O action' in this section means 'first connect the required I/O device, select the proper unit interpretive routine and then pass control to that routine'.

The I/O completion routines (I/O Executor) is the TRAP Supervisor which takes over control during trapping and finally surrenders its control back to the program which was using the input-output system. The I/O Executor determines when an I/O operation has just ended, checks for detected errors, determines which I/O operation is to be performed next, and initiates the new action.

A typical I/O Executor is shown in Fig. 4.

2.4.3. Unit Interpretive Routines (or Unit Drivers)

Unit interpretive routines are hardware dependent, and hence every type of I/O devices has a unit interpretive routine associated with it. The unit interpretive routines are called by the I/O initiation routines. If a unit interpretive routine is activated, it first checks the function code of

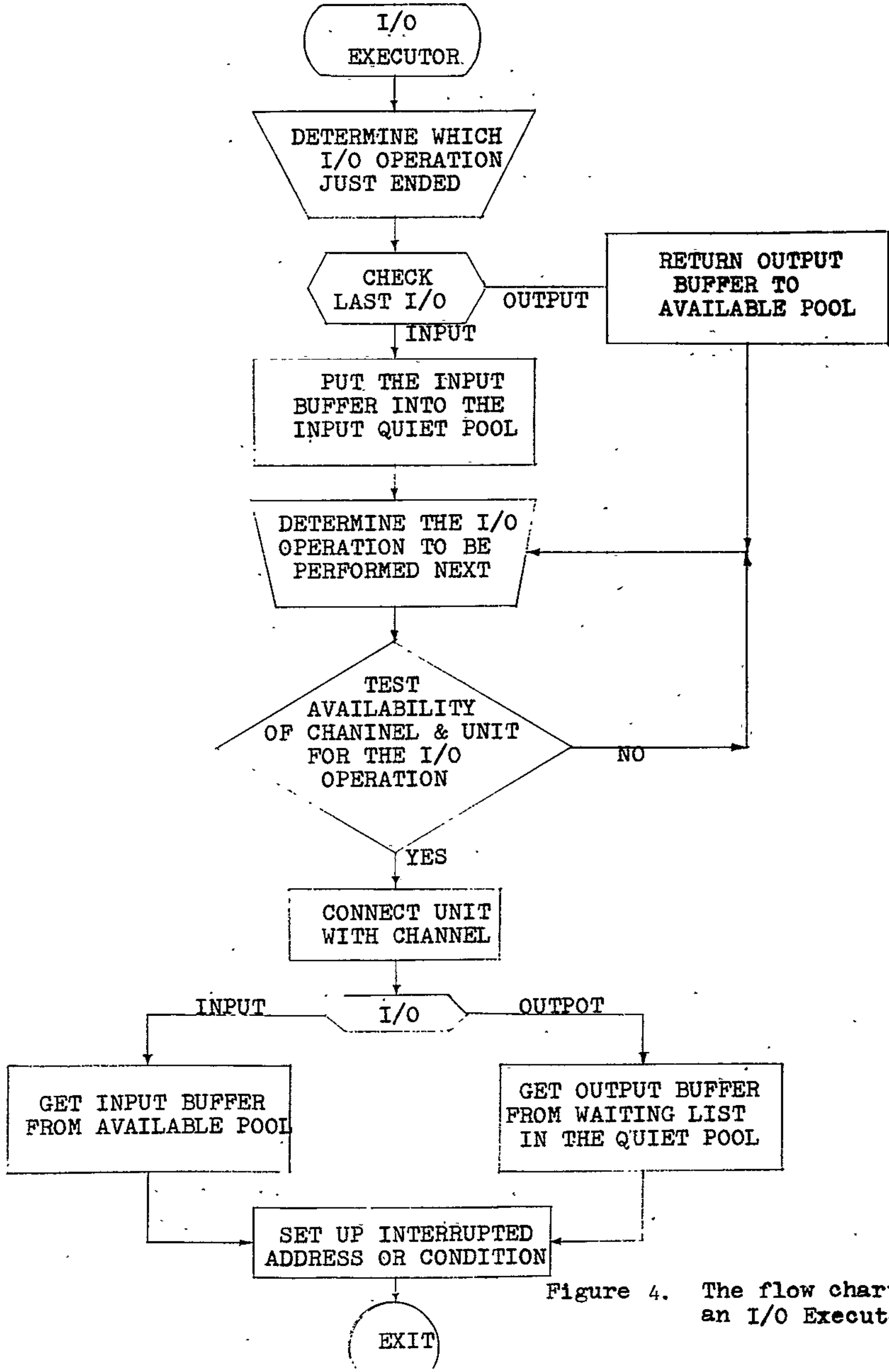


Figure 4. The flow chart fo
an I/O Executor

the I/O request and then:

(a) Sets up the I/O instructions for that I/O request,

(b) Forms a list of channel commands (i.e. forms a channel program) to be performed on the unit,

(c) Issues those I/O instructions, and

(d) Returns control to proper routine.

A typical unit interpretive routine is shown in Fig. 5.

2.4.4. Communications Among Routines

(a) The file for communication between the user's program and the input-output control system

A file is a complete set of logical records which a user may treat as a logical entity. All files must be defined and opened before they can be processed. Similarly, a file must be closed when activity in the file is to be terminated. There are two routines, READ and WRITE, which serve as communication between the file system and the buffering system. The READ routine reads the information out of the input-processing buffer into the user's working area, while the WRITE routine puts the information into the output-processing buffer. Note that a buffer can be treated as a logical record of a file. In addition, each file has a File Control Block (FCB) associated with it. The File Control Block contains several items of information about the file. This information includes the present status of the file, the processing buffer which is currently in use by this file, and the address of the Unit Control Block (UCB) of the unit to be used by the file.

(b) The use of tables for communication within the input-output control system

At the present time, most of the input-output systems are able to refer to I/O units by symbolic names. A symbolic assignment of input-output units has at least three advantages:

(1) Object programs refer to storage cells rather than to absolute unit address,

(2) Unit assignments are made by the system and need not be known

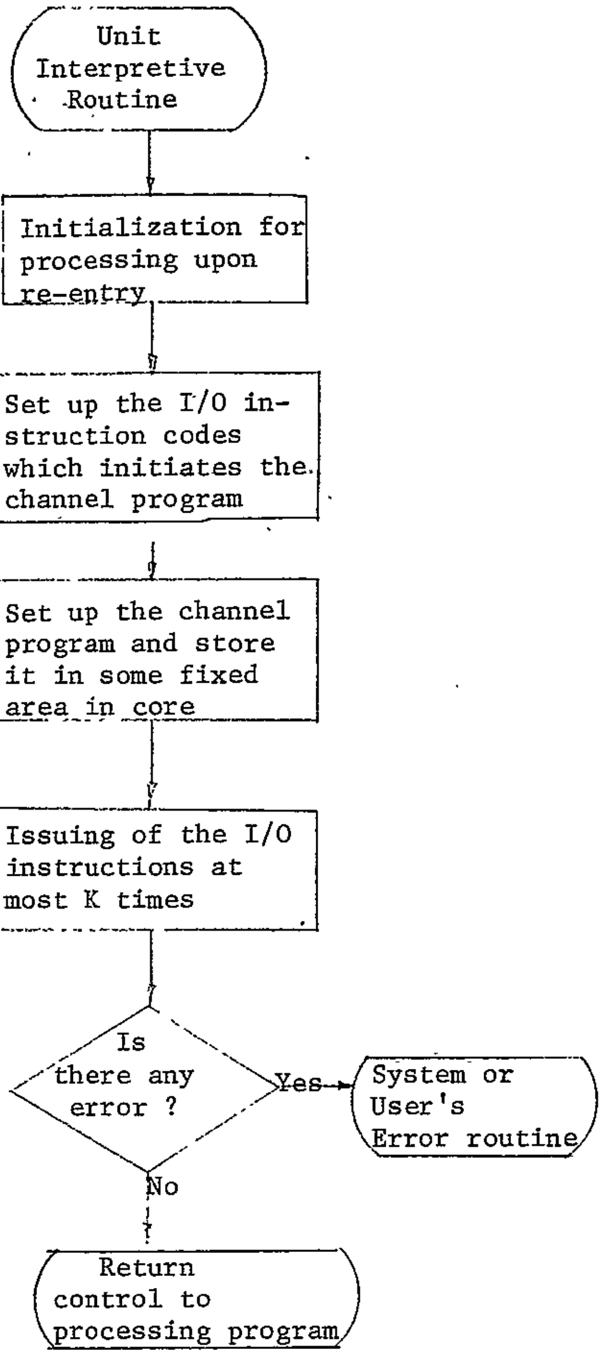


Fig. 5. The Flow Chart of an Unit Interpretive Routine

by the programmer in advance.

(3) In case the full system is not working, I/O activity can be continued, albeit at reduced efficiency. This is an example of what is called graceful degradation of the system.

In order to be able to refer to an I/O unit symbolically, use is made of a symbolic unit table. This table contains entries for all the symbolic names of the units. Each table entry contains the address of a Unit Control Block which is associated with each name. In the general case, several symbolic units can be associated with one Unit Control Block, while each physical unit has only one Unit Control Block associated with it. The Unit Control Block contains the unit address as well as the unit status, types of information in it and unit position information. In addition, for each channel there is an associated Channel Control Block (CCB). A Channel Control Block contains the channel status, the interrupt address, and the address of the Unit Control Block for the unit which is currently connected to this particular channel.

A more graphic description of the input-output system communication is given in Fig. 6.

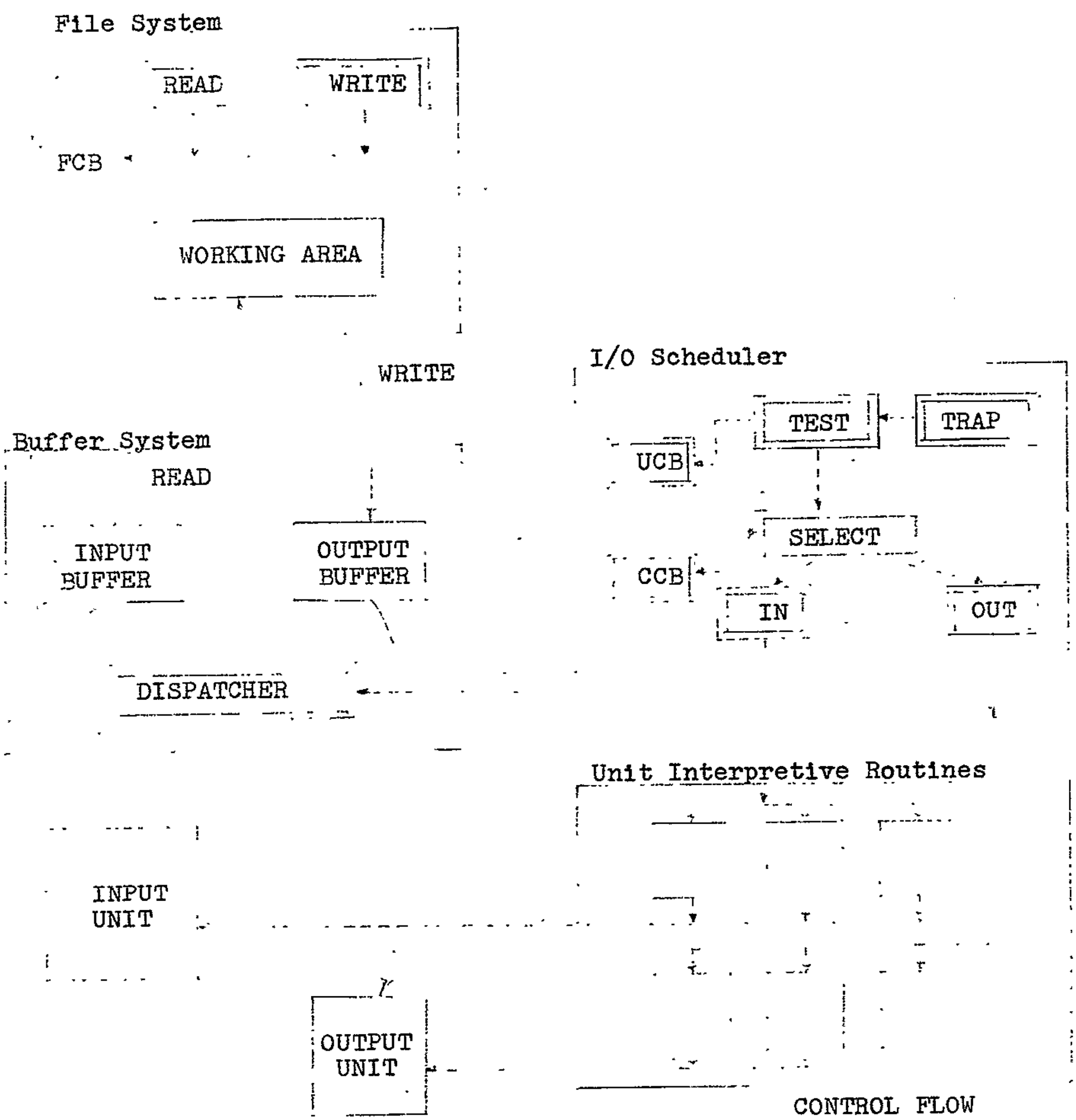


Figure 6. Input-Output Communication

3. A Simplified IOCS (SIOCS) for Microprogramming

Unified hardware-software design is the main goal of this research (See Reference [50] for detail). This paper presents one of the initial studies of this research, namely extracting the algorithms from a piece of software and presenting it in some form which is suitable for eventual microprogramming. The following sections (Sections 3 through 6) describe an Input-Output Control System named a Simplified Input-Output Control System (SIOCS), where the functions of IOCS are defined precisely in terms of their level of tree-structure. The algorithms are so presented that it is able to convert to sequence chart for eventual microprogramming.

3.1. Design Principles

A major consideration in the design of SIOCS is to make it simple yet at the same time extensible and machine independent, with a minimum number of extra restrictions and assumptions. Two basic aspects of the design of SIOCS are:

(a) The assignment of functions of an I/O system to levels in a tree-structured (or hierarchical structured system).

(b) The use of a double buffering technique.

3.1.1. Simple yet extensible and machine independent

SIOCS was chosen for the initial study of the Input-Output Control System and its implementation as a micro-program. The first consideration for designing SIOCS was to make it simple but, at the same time, extensible. SIOCS contains all the tables and control blocks which most conventional systems use. For example, SIOCS contains the channel control word for each channel, thus permitting several I/O devices to share the same channel at different times (see 3.1.2., this feature has not been implemented yet in SIOCS). The design philosophy follows the common features of the IBM 7000 series, IBM 1401, 1410, and CDC 3000 series, since these are the most popular batch processing systems. SIOCS consists of three major parts: the file and buffering system, the I/O scheduler, and the unit interpretive routines. The first two parts are emphasized and described in detail, while for the last part, only the algorithm is presented. The algorithm is designed to generate a machine executable code for any particular I/O device. SIOCS is completely defined for a given particular computer system configuration whenever the unit interpretive routines and the I/O function tables are provided, this makes SIOCS machine independent.

3.1.2. Restrictions and Assumptions

SIOCS assumes that every I/O device is connected with one channel at all times. This assumption frees SIOCS from the necessity of checking and scheduling the channel and connecting the channel with the proper I/O device every time an I/O operation is requested. SIOCS also assumes only one mode and that no automatic label is used. This mode may be considered as Binary-Coded Alphabetic (BCD, EBDIC, or Field Data). This assumption results in the necessary restriction that every tape file should be able to fit into the one physical tape reel. Since SIOCS does not check the label for each file, no file protection is implemented in SIOCS. At the present stage, only the tape operating system is implemented in SIOCS. That is, no random-access mass storage is used in the hardware configuration. One hardware constraint that should be mentioned here is that the central processor must have the ability to process I/O interrupts.

3.1.3. Levels within an I/O system

The concept of a tree-structured operating system has been proposed by Dijkstra. The important aspect of this organization is that all activities are divided into sequential processes. A tree structure of these sequential processes results in an hierarchical or ring organization. Each procedure in the system is given its level, or place in the hierarchy. Each call may be downward only. Thus, if at each level, procedures are organized about an expanding set of relevant states, the system can be exhaustively tested and proved to work. As Dijkstra and others have suggested, this may be the only way to make certain that a system can be debugged before the hardware is obsolete.

The hierarchy of levels of SIOCS to be presented in this paper can be divided into two classes: one is the levels pertaining to the I/O devices and the other is the levels of I/O programs themselves. Fig. 7 illustrates the three levels of I/O units, they are: File, Symbolic Unit, and Physical Unit; and four levels of I/O programs, they are: User's Program, Buffering Routines, I/O Scheduling Routines, and Unit Interpretive Routines.

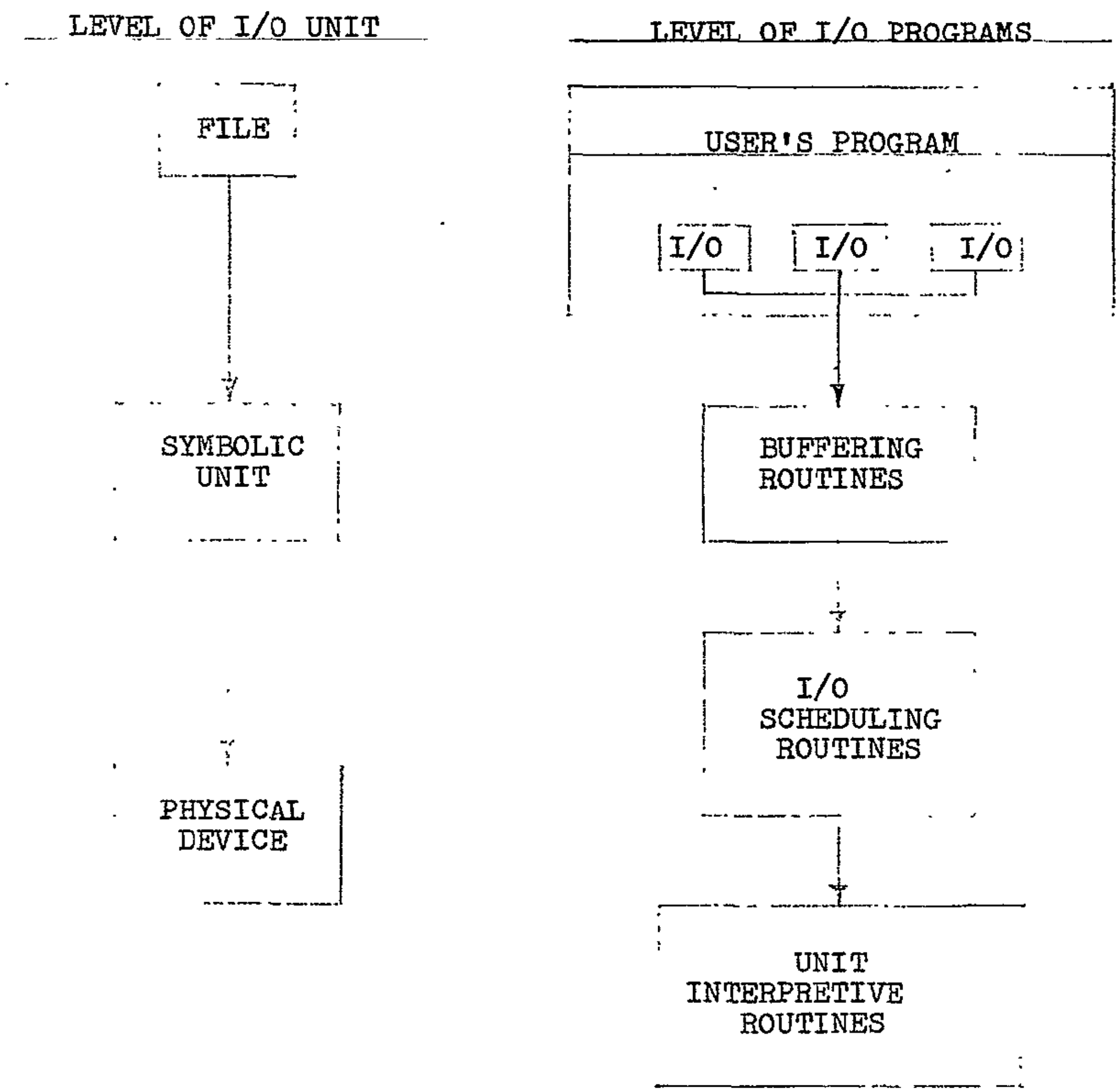


Figure 7. The Levels of I/O System

The notion of levels can best be introduced by the following example. Consider a user using SIOCS to perform I/O operations. At the user's level (User's Program), the computer is viewed as a CPU with a main memory, and several tape units for input-output purposes. Each tape unit has its own symbolic name, for instance, cardreader, printer, disk,..., etc. Whenever a user wants input (or output) from some particular tape device, he may achieve this simply by opening a file, and by assigning it to that particular tape unit. This can be thought of as assigning a name to a reel of tape and mounting this reel of physical tape to the desired tape device. Similarly, closing a file can be imagined as dismounting that reel of tape from the device. By using a simple READ or WRITE request, the user can read out, or write into that reel of tape. On the other hand, the system programmer who wrote the buffering system, at the level of Buffer Routines, need not have had any knowledge of the file system. He might view the computer as a Buffering Machine. Whenever the output buffer is full, it will automatically empty it. The buffering routine is just as simple as a routine used to assign the proper status for each file when it changes. In this manner the user need not have any specific knowledge of the internal operation of filling or emptying a buffer, but only the way in which he can interact with it. Similarly, the system programmer who wrote the I/O Scheduling Routines may assume that the user will request I/O operations very frequently and that the duties of the I/O Scheduler are: (1) to keep the I/O devices as busy as possible; (2) to respond to the I/O request as quickly as possible; (3) to report to the user immediately whenever the I/O request is finished. In the lowest level of I/O programs, the unit interpretive routines, only the knowledge of how to generate I/O instructions and channel commands is required of the programmer.

Note that only the unit interpretive routines are hardware dependent, since it is in this lowest level that the actual code for the channel programs will be generated and executed.

There are several advantages to this hierarchical organization. The most important is logical completeness at each level. It is easier for the system designers and implementers to understand the functions and interactions of each level and thus the entire system. Another advantage is debugging assistance, since whenever an error occurs it can be localized at a level and identified easily. As has been mentioned before, it may be the only method of debugging the system.

3.1.4. Buffering Algorithms

The basic characteristics of the buffering algorithms used in SIOCS are:

(a) Each file is associated with two equal-size buffers (double buffers),

(b) The buffer size is dependent upon the I/O device,

(c) Each processing buffer has a critical number associated with it.

As mentioned above, each file in SIOCS is associated with two equal-size buffers. One of these two buffers is used as an input (or output) buffer into which data is read in (or written from). The other is used as a processing buffer where current data are obtained. Fig. 8 shows the ideal model of this buffering scheme.

In Figure 8, the shaded areas represent the portions of the buffers which contain the data, while the blank areas represent empty areas of the buffer. The arrows below the two double buffers in this figure indicate the direction of the rotation of the double buffers. In the illustration above, an input device is filling the buffer A, while the buffer D is being emptied into an output device. Meanwhile, the user's program READs information from buffer B for processing. After processing, it WRITEs the information into buffer C. In this case buffer A is the input buffer, buffer B is the input-processing buffer, buffer C is the output-processing buffer, while buffer D is the output buffer. Whenever buffer A is filled and buffer B is emptied, they are interchanged. At this time, buffer B is called the input buffer and buffer A is called the input-processing buffer. Similar treatment occurs for buffers C and D.

Figure 8 shows an ideal model which assumes that the time-interval required for filling an input (or output-processing) buffer is equal to the time-interval required for emptying the input-processing (or output) buffer. Unfortunately, these conditions usually do not hold. Some basic principles to be applied for solution of this problem are:

(a) When inputting data, a sufficiently large buffer must be made available for input transmission well ahead of the active routine's immediate requirements.

(b) During output, a sufficiently large buffer must be supplied to contain the potentially large amounts of data that can be generated.

One way to apply the above mentioned two principles is illustrated as follows. Consider that a program requests input from an input device, UNT1. Let T_f be the time-interval required for UNT1 to transmit one physical record

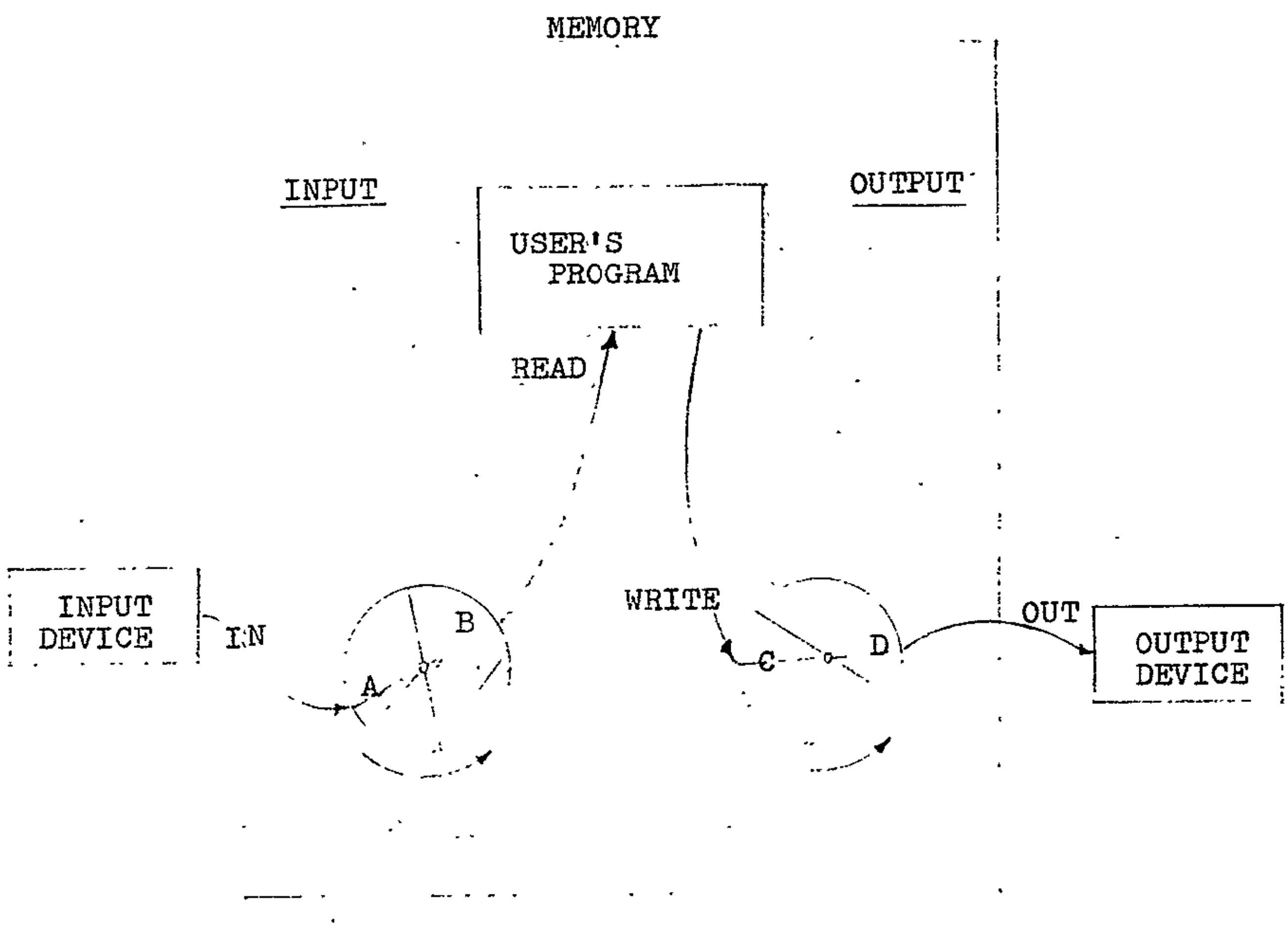


Figure 8. Ideal Model of The Buffering System

into the input buffer. Let T_e be the average time required for the program to request a physical record from the input-processing buffer and then process it.

It is clear that T_f is fixed and is dependent on the hardware device, while T_e may vary from one program to the next.

In the case where $T_e \geqslant T_f$ (that is, the processing time is greater than or equal to the I/O time), we must consider the ratio between the I/O initiation time and the actual data transmission time. Let T_i be the average time required to initiate an input operation of UNT1. As shown in Fig. 9.1, if T_i is smaller in comparison with T_f, then it will be better for the buffer size to be a small multiple (i.e., one or two) of the physical record. On the other hand, as shown in Fig. 9.2, if T_i is much larger than T_f, a large multiple of the physical size is required.

In addition to the above considerations, we must evaluate the current request and decide the best time to initiate the next input operation. We don't want to transfer data too far in advance of when the active routine will actually process that data. This could mean wasting core storage or wasting time due to the fact that the data may never really be required by this active routine. One way to insure sufficient READ AHEAD is to set up a critical amount indicator for the input-processing buffer. Whenever the available data in the input-processing buffer is less than the critical amount, the next input operation is initiated. Note that by setting up the critical amount indicator we permit the data to be transferred into the input buffer before the input-processing buffer is emptied.

An example for assigning the buffer size and the critical amount of data indicator for the processing buffer is as follows.

Let $T_f / T_e = m/n$

Then set buffer size = n* (size of physical record)

critical amount = m* (size of physical record)

Thus $T_f * n = T_e * n$ as shown in Fig. 9.3

In the case where $T_e < T_f$, program X requests input data from UNT1 very expeditiously. Even though UNT1 continuously transfers the data into buffers, the program X still has to wait for data. In this case, we only need to set the buffer size equal to the size of physical record. This makes the total execution time as small as possible. From Fig. 9.4, one may easily see the difference between two execution times.

Case 1. $T_e \geqslant T_f$

(A) $T_i < T_f$

Example : $T_e = 3 * t$, $T_f = 2 * t$, $T_i = t / 4$, t = time slice

(1) Buffer Size = 3 * physical record

(2) Buffer Size = 1 * physical record

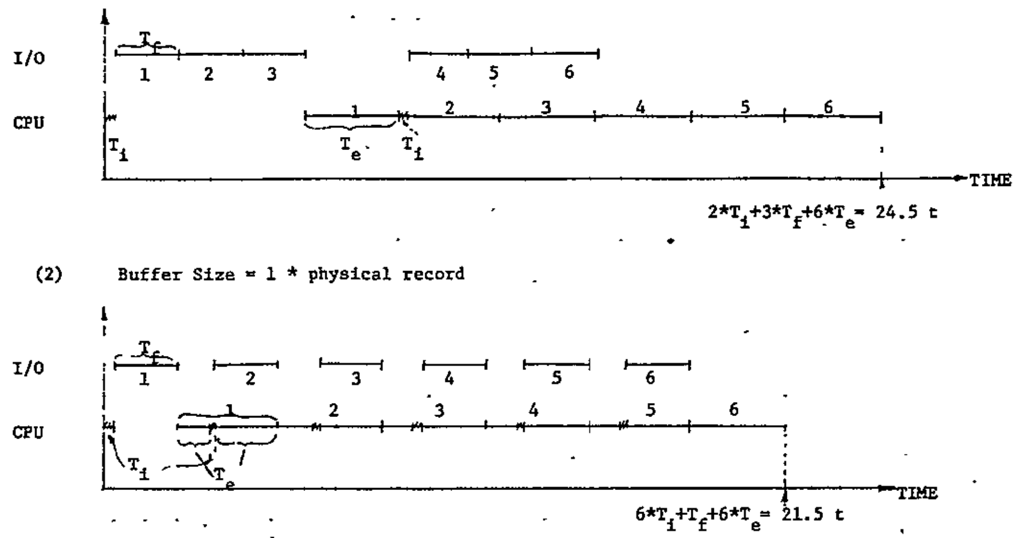


Fig. 9.1 A timing chart for an example of case $T_e \geqslant T_f$ and $T_i < T_f$

Case 1. $T_e \geqslant T_f$

(B) $T_i \geqslant T_f$

Example : $T_e = 3 * t$, $T_f = 1 * t$, $T_i = 2 * t$, t = time slice

(1) Buffer Size = 3 * physical record

(2) Buffer Size = 1 * physical record

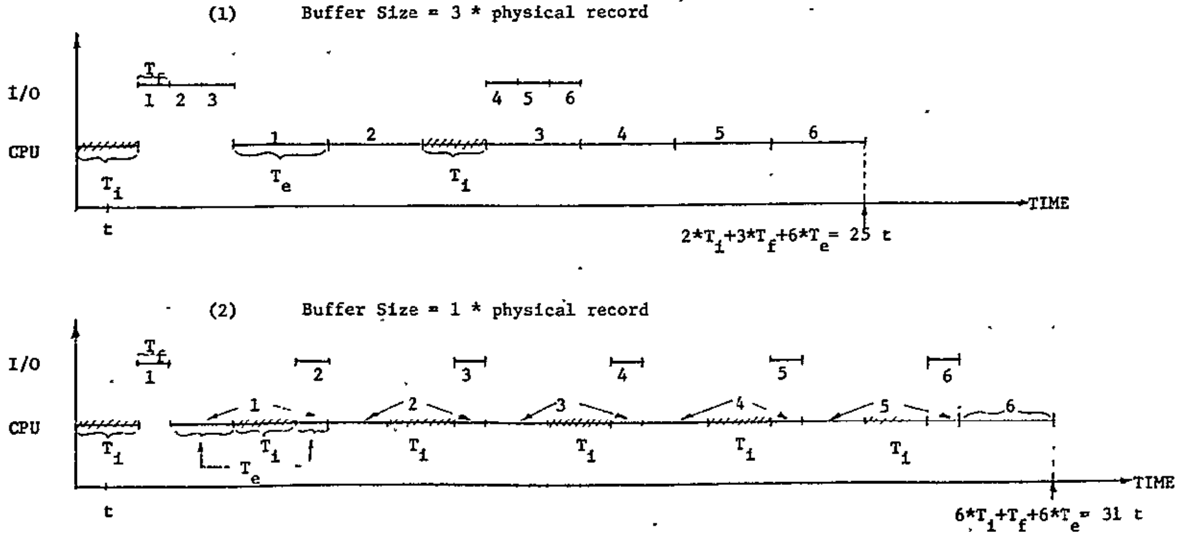


Fig. 9.2 A timing chart for an example of case $T_e \geqslant T_f$ and $T_i \geqslant T_f$

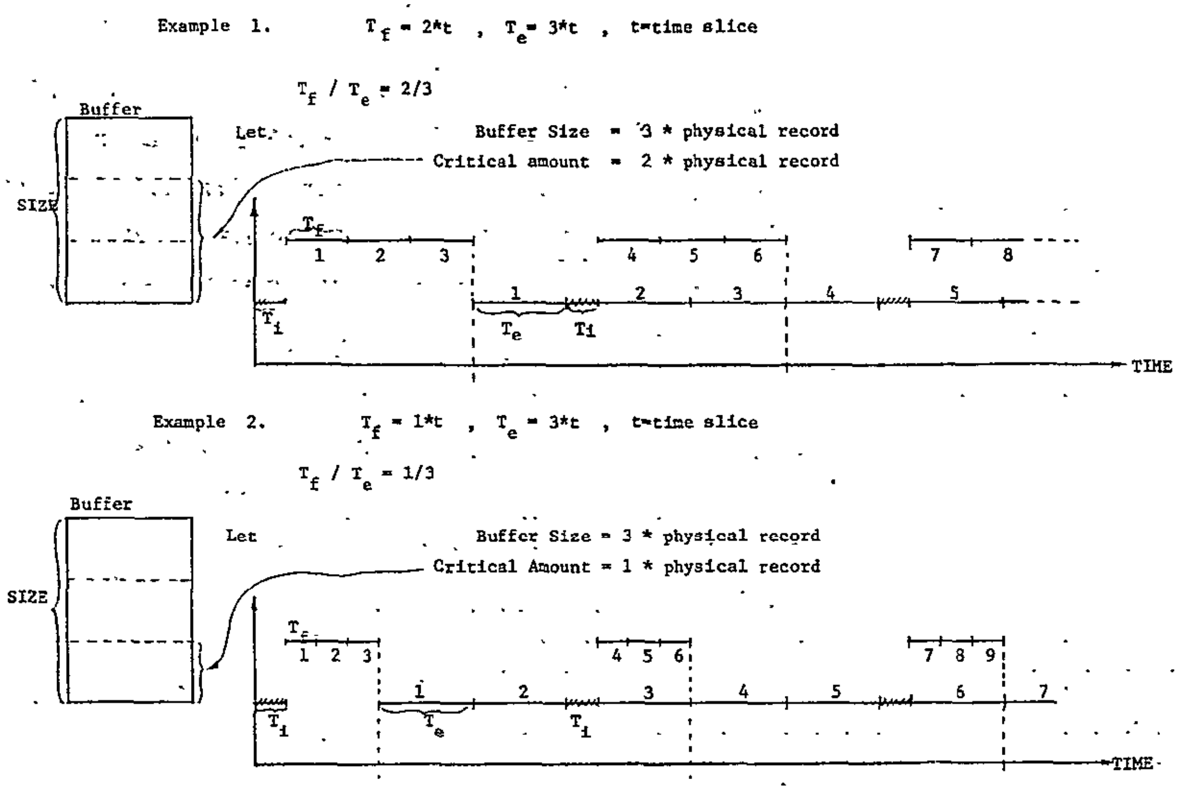


Fig. 9.3 An example for assigning the buffer size and tne critical amount indicator.

CASE 2 Te < Tf

Example: Te = 2 * t , Tf = 3 * t , t= time slice

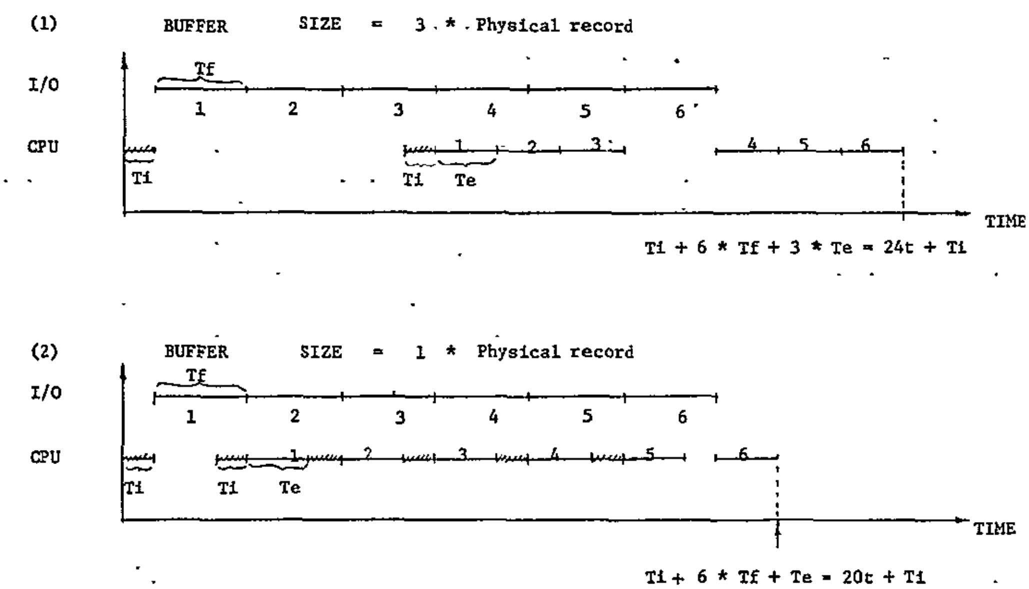


Fig. 9.4 An timing chart for an example of case $T_e < T_f$

The problems of output buffers have similar characteristics to those of the input buffers. However, note that the input-processing buffer requires a critical amount indicator while the output-processing does not. This distinction is because there is no possibility of WRITING AHEAD (that is, there is no possibility of sending out some information which has not yet been processed).

Different computer installations may have quite different collections of I/O devices, and different user patterns. After some statistical studies, an assumption can be made about user characteristics in order to fix the buffer size and the critical amount parameter associated with each I/O device. The buffering system of SIOCS was designed under the assumption that for every I/O device there is a fixed buffer size and there is a critical amount indicator associated with it. These two items of information are stored in the symbolic unit table which is generated at the system generation time. They can be changed by the system programmer.

Besides the buffering system mentioned above, SIOCS allows the user to establish his own buffering routine without reference to the SIOCS buffering system. Through these means, a user is free to play with any buffering scheme that he may choose.

3.2 Macro-instructions and Examples

3.2.1 The Macro-instructions in SIOCS

(1) File handling

(a) OPEN -- initiate processing of a file

The OPEN macro-instruction has the following format:

FILENAME	OPEN	TYPE, DEVICE (REWIND)

FILENAME is the name of the file to be opened. (i.e., the symbolic address of the File Control Block to be opened.)

TYPE is the one of the following file types which is to be assigned to the file.

IN -- input file

OUT -- output file

NONBUF -- non system buffering file

DEVICE is the device type or symbolic name of a particular unit which is to be used by the file. Such as the following:

TAPE -- magnetic tape

CARDREAD -- cardreader

PRINTER -- line printer

CARDPUNCH -- card punch

SYSUT n -- system utility unit n

SYSIN n -- system input unit n

SYSOU n -- system output n

REWIND is the tape rewind operation. This field is optional.

(b) CLOSE -- terminate processing of files

The CLOSE macro-inscruction has the following format:

	CLOSE (OPTION)	NAME1, NAME2, ...

As an option, one of the following can be specified for closing a list of files:

REWIND -- close and rewind the tape

UNLOAD -- close and remove the tape from the UNIT

The NAME n is the name of the file (i.e., the symbolic address of the File Control Block) to be closed. Several files can be closed by using one macro-instruction, note that the option field applies to every file in the list (e.g., if UNLOAD option is specified, then every file in the list is closed and unloaded).

(c) REDEF -- reassign the file type to the file

The REDEF macro-instruction has the following format:

```
REDEF  FILENAME, TYPE (REWIND)
```

FILENAME is the name of the file (i.e., the symbolic address of the File Control Block) to be redefined.

The TYPE is one the following types of the file to be assigned to the file.

IN -- designates an input file

OUT -- designates an output file

NONBUF -- don't use system buffering for this file

REWIND -- This is a rewind operation. This field is optional.

(2) Data Handling

(a) READ -- read data

The READ macro-instruction has the following format:

READ FILE, ERR, EOF, INTRUP, 1^{st} ADDR., N

FILE is the name of the file which the data is to be read from.

ERR is the address of the user's error recovery routine. If this field is blank, the system error recovery routine is assumed.

EOF is the address of user's end-of-file detection routine. If this field is blank, the system error checking routine is assumed.

INTRUP is the address of the user's interrupt routine. If this field is blank, the system interrupt routine is assumed.

1^{st} ADDR. is the address of the first word where the data are to be stored.

N is the number of words to be read in.

(b) WRITE -- write out the data

The WRITE macro-instruction has the following format:

```
WRITE  FILE, ERR, INTRUP, 1st ADDR., N
```

FILE is the name of the file to be written into.

ERR is the error return address. If this field is blank, the address of system error recovery routine is assumed.

INTRUP is the address of user's interrupt routine. If this field is blank, then the address of the system interrupt return address is assumed.

(c) Nondata request

There are four non-data request macro-instructions, namely, REWIND, MOVE, BKSP, and WEOF. The formats of these four macro-instructions are as follows:

```
REWIND   FILENAME
```

```
MOVE     FILENAME, FN
```

```
BKSP     FILENAME, RN
```

```
WEOF     FILENAME
```

FILENAME is the name of the file.

FN is the number of end-of file markers to be used.

RN is the number of en-of record markers to be used.

3.2.2 Some examples which use SIOCS

Example 1:

The following program, in the IBM 7090, will read a deck of 10 cards and copy it onto magnetic tape. After that, it writes an end-of-file on tape and rewinds it. Then the program copies the tape on the card-punch and the line-printer. The output will be 10 cards, each card consists of the first 60 columns of the original input card. A listing of the input cards will also be given.

```
CARDS   OPEN    IN, CARDREADER               .OPEN CARD INPUT FILE
TAPE    OPEN    OUT, TAPE                    .OPEN TAPE OUTPUT FILE
        AXT     10,1                         .
        READ    CARDS, ERR,,, RECORD, 14     .READS ONE CARD AND
        WRITE   TAPE, ERR,, RECORD, 14       .COPY IT TO TAPE
        TIX     *-3,1,1                      .
        WEOF    TAPE                         .WRITE END-OF-FILE
        REDEF   TAPE, IN, REWIND             .REWIND AND REDEFINE
PRINT   OPEN    OUT, PRINTER                 .OPEN PRINT OUTPUT FILE
PUNCH   OPEN    OUT, CARD-PUNCH              .OPEN PUNCH OUTPUT FILE
LOOP    READ    TAPE, ERR, EOF,, RECORD, 14  .READS 14 WDS FROM TAPE
        WRITE   PUNCH, ERR,, RECORD, 10      .PUNCH 1ST 10 WDS
        WRITE   PRINT, ERR,, RECORD, 14      .PRINT 14 WDS
        TRA     LOOP                         .
EOF     CLOSE   CARDS, TAPE, PRINT           .CLOSE ALL FILES
        CALL    EXIT                         .
ERR     TRA     SYSDMP                       .SYSTEM DUMP ROUTINE
RECORD  BSS     14                           .
```

Example 2:

This program performs the following operations:

(a) Reads a deck of cards.
(b) Copies 50 words from card-images onto two tapes.
(c) Writes an end-of-file and then rewinds both tapes.

(d) Reads tape 1 without a system buffer while it sets a counter to count how long the read operation will take.

(e) If the read operation is completed, it prints out the counter and the data on the tape, otherwise it prints out the I/O status and the counter only.

(f) Reads tape 2 using the system buffering scheme. Also, it sets up a counter to count how long the read operation will take.

(g) Prints out in the same manner as step 5.

```
CARD      OPEN    IN, CARDREADER                         .OPEN CARD INPUT FILE
TAPE 1    OPEN    OUT, TAPE, REWIND                      .OPEN WITH REWIND
TAPE 2    OPEN    OUT, SYSUT 1, REWIND                   .OPEN, REWIND SYSUT 1
          READ    CARD, ERR, EOF,, RECORD, 50            .READS 50 WDS
          WRITE   TAPE 1, ERR,, RECORD, 50               .COPY ON TAPE 1
          WRITE   TAPE 2, ERR,, RECORD, 50               COPY ON TAPE 2
          WEOF    TAPE 1                                 .WRITE END-OF-FILE
          WEOF    TAPE 2                                 .WRITE END-OF-FILE
          REDEF   TAPE 1, NONBUF, REWIND                 .REWIND AND REDEFINE
PRINT     OPEN    OUT, PRINTER                           .OPEN PRINT FILE
IN        READ    TAPE 1, ERR, EOF, INTRUP, RECORD, 50   .READS 50 WDS
          STZ     FLAG                                   .SET FLAG=OFF
          STZ     COUNT                                  .RESET COUNTER=0
LOOP      CLA     COUNT                                  .
          ADD     =1                                     .INCREASE COUNTER
          STO     COUNT                                  .BY ONE
          ZET     FLAG                                   .FLAG=?
          TRA     CHECK                                  .FLAG=ON
          TRA     LOOP                                   .FLAG=OFF
INTRUP    NOP            :STATUS: ADDRESS:               .STATUS WORD IN HERE
          STO     TEMP                                   .SAVE AC
          CLA     =1                                     .
          STO     FLAG                                   .SET FLAG=ON
          CLA     TEMP                                   .RESET AC
          TRA     INTRUP                                 .GO BACK TO PROCESS
```

```
CHECK   CLA      INTRUP                          .CHECK STATUS WORD
        CAS      COMPLT                          .
        TRA      INCOM                           .I/O IS INCOMPLETED
OUT     WRITE    PRINT, ERR,, COUNT, 51          .I/O COMPLETED, PRINT
        TRA      AGAIN-3                         .OUT
INCOM   WRITE    PRINT, ERR, COUNT, 1            .PRINT OUT COUNTER
        WRITE    PRINT, ERR,, INTRUP, 1          .PRINT OUT STATUS WD
        CLA      INTRUP                          .
        PBT      =1                              .CHECK UNIT
        TRA      EOF                             .UNIT IS SYSUT 1
AGAIN   REDEF    TAPE 2, IN, REWIND              .OTHER UNTIL, REDEFINE
        CLA      AGAIN                           .CHANGE FILENAME OF
        STA      IN                              .THE READ STATEMENT
        TRA      IN                              .
EOF     CLOSE    CARD, PRINT                     .
        CLOSE, U TAPE 1,. TAPE 2                 .CLOSE WITH UNLOAD
        CALL     EXIT                            .SYSTEM EXIT ROUTINE
ERR     TRA      SYSDMP                          .SYSTEM DUMP ROUTINE
TEMP    BSS      1                               .
COUNT   PZE      0                               .
RECORD  BSS      50                              .
FLAG    BSS      1                               .
COMPLT  OCT      =....                           .I/O COMPLETED
```

4. The Functions of SIOCS

4.1 The File and Buffering

4.1.1 Defining a File

(A) Opening a File

A FILE is a collection of related records treated as a unit. All files must be opened before they can be processed. The OPEN macro instruction opens a file and describes, in detail, an individual file (Example 1). An OPEN macro instruction must declare the file name, file type, and device used, rewind option for each file processed by IOCS.

Example 1:

	Label	Operator	Operands
	Name		Type, Device, (Rewind)
1	FILEA	OPEN	NONBUF, CARDREADER
2	DRUMA	OPEN	IN, DRUM
3	OUTAP	OPEN	OUT, TAPE, REWIND

Line 1: The OPEN macro instruction opens a card inputfile named FILEA.
Line 2: The OPEN macro instruction opens a drum input file named DRUMA.
Line 3: The OPEN macro instruction opens a tape output file, named OUTAP. This file must be rewound before opening.

(B) Closing a file

When activity on a file is to be terminated, it must be closed. At closing, all I/O activity on a file ceases. The CLOSE macro instruction closes a list of files or a single file (Example 2)

Example 2:

	Label	Operator	Operands
			Name 1, Name 2...
1		CLOSE	FILEA
2		CLOSE	DRUMA, OUTAP

Line 1: The CLOSE macro instruction closes a file which is named FILEA.

Line 2: The CLOSE macro instruction closes a list of files which contains DRUMA file and OUTAP file.

(C) Redefining a file

There are three different type of files, namely, IN (input file), OUT (output file) and NONBUF (non-buffered file). Every IN and OUT file is associated with a double-buffer, while a file which was declared nonbuf means that the file is to be read from or written onto a device without using any system buffering routines. (for details of a buffering technique used in SIOCS see the next section 4.1.2) The REDEF macro instruction is used for the redefinition of a file which was opened previously (Example 3). The advantage of using a REDEF is that it allows a file to be defined first as one type and then changed to another type later. One need not declare a change in the I/O device associated with the file. It also allows IN/OUT files to share the same buffers.

Example 3:

	Label	Operator	Operands
			Name, Type, (Option)
1		REDEF	FILEA, NONBUF
2		REDEF	DRUMA, OUT
3		REDEF	OUTAP, IN, REWIND

Line 1: The REDEF macro instruction redefines the file FILEA as a NONBUF file. If this file was so defined previously, then this macro statement is treated as a no operation.

Line 2: The REDEF macro instruction redefines the file DRUMA to be an OUT file. The operation of this macro statement are:

Old Type	Operations of the macro statement
OUT	No operation
IN	1. Redefine DRUMA as an output file 2. Use the same buffers which were used before.
NONBUF	1. Redefine DRUMA as an output file 2. Allocate a buffer area for this file

Note: Old type means previous defined type of the file

Line 3: The REDEF macro instruction redefines the file OUTAP to be an input file with rewind operation. The operations of this macro statement are:

Old type	Operations of the macro statement
OUT	1. Redefine outap as an input file 2. Use the same buffers which were used before 3. Rewind
IN	Rewind only
NONBUF	1. Redefine OUTAP as an input file 2. Allocate a buffer area for file outap 3. Rewind

4.1.2 Buffering

4.1.2.1 Buffer Area

Every file, except a nonbuffered file, is associated with a double-buffer. A double-buffer is a pair of equal-size blocks in core storage. It is referred to by two pointers IOBUF and PROBUF, and is used for intermediate storage of input/output data. (figure 10)

Whenever an IN/OUT file is opened or redefined, the SIOCS allocates two equal-size contiguous core storages, and assigns them as double-buffers for this file. The buffer size is equal to N*(physical record size of that device) depending on the device used by the file, where N is an integer factor which depends on the device data transmission rate and the memory data transmission rate. As soon as a file is closed, the double-buffer associated with it is released.

4.1.2.2 Buffer cycles

When the double-buffer is used for an input file, it can be considered as an input double-buffer, although the input status may be only temporary. Similarly, when the double-buffer is used for an output file, it can be considered as an output buffer.

(A) Input buffer cycle

The logic flow for the input buffer cycle is shown in Figure 11.

IOBUF:	A pointer which points to the current I/O buffer
IOBUFR:	The current I/O buffer
PROBUF:	A pointer which points to the processing buffer
PROBUFR:	The current processing buffer
CRTCL:	The critical number of items in the PROBUF buffer
AVBCT:	A counter of the number of available items in the PROBUF buffer

(1) At first the IOBUFR buffer (the buffer pointed by IOBUF) and the PROBUFR buffer (the buffer pointed by PROBUF) are empty, and AVBCT=0

(2) The IOBUFR buffer is filled with data from an input unit

(3) If the AVBCT=0, then the pointer IOBUF is exchanged with the pointer PROBUF. RESET the AVBCT to buffer size and start to process the new PROBUFR buffer.

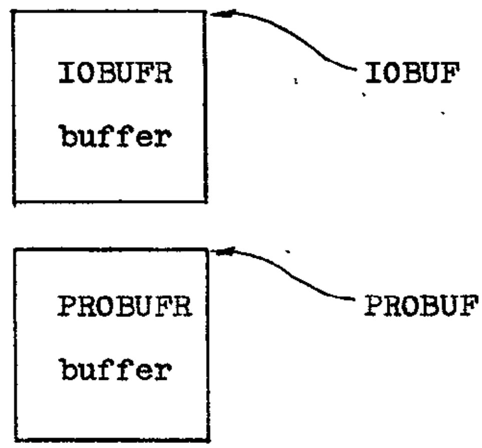


Fig. 10. An Example of a Double-Buffer

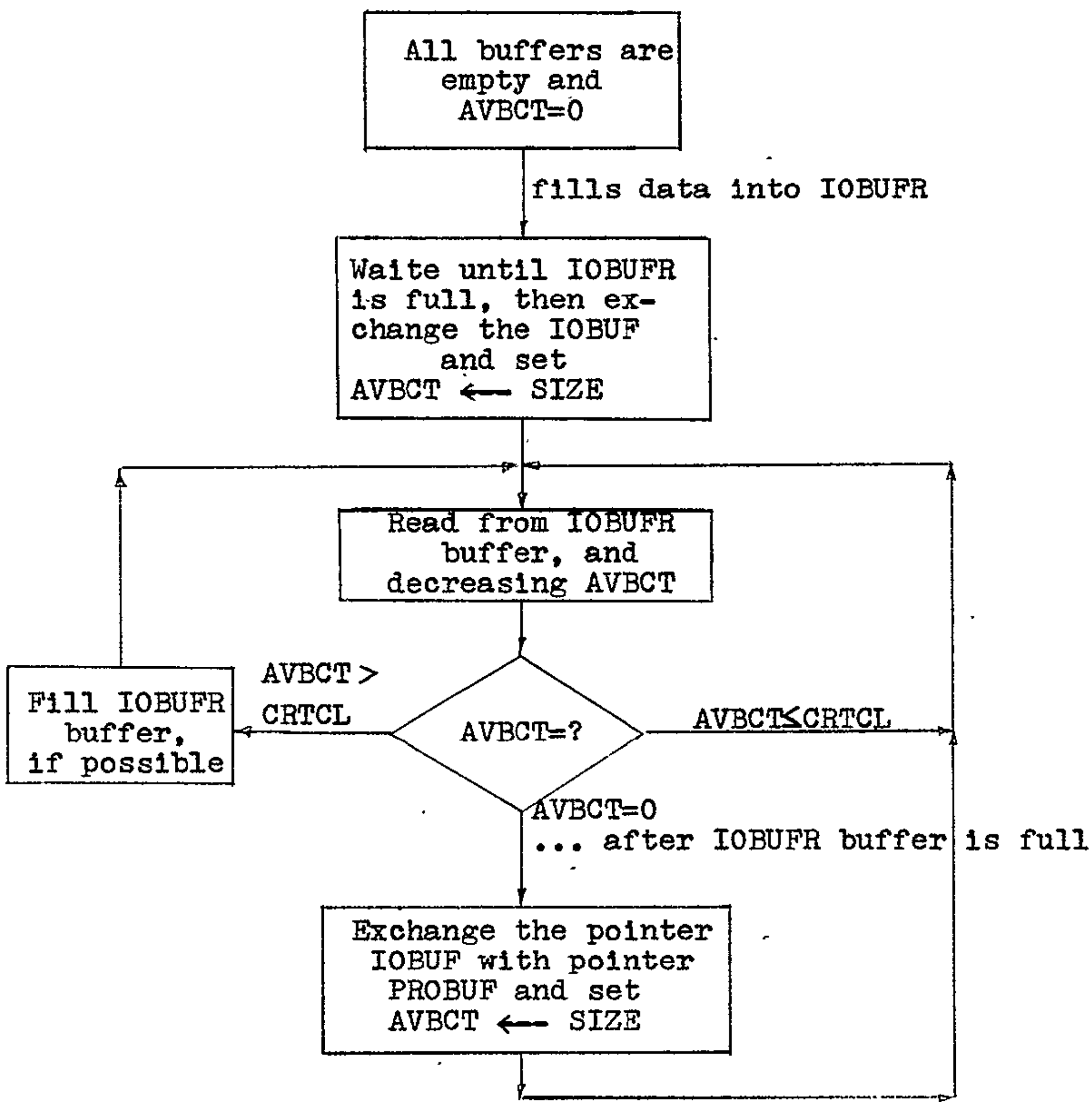


Fig. 11. The Logic for Input Buffer Cycle

(4) If the AVBCT is less than CRTCL, then set up input unit for the IOBUFR buffer and start to fill with data at the suitable time. Go to step 2.

(B) Output buffer cycle

The logic flow for output buffer cycle is shown in Figure 12. Where the definitions of IOBUF, IOBUFR, PROBUF, PROBUFR, AVBCT are the same as in input buffer cycle.

(1) At first the IOBUFR buffer and the PROBUFR buffer are empty, and the PROBUFR buffer is waiting to be filled with data.

(2) When the PROBUFR is full (i.e. AVBCT=0), exchange the pointer PROBUF with the pointer IOBUF.

(3) Set up the output unit for the IOBUFR buffer and then output data from the IOBUFR buffer to unit. Meanwhile, fill the PROBUFR buffer with data, and go back to step 2.

4.1.2.3 Buffer allocation

As has been mentioned before, double buffers are used for each file except the NONBUF file. All buffers which are used by SIOCS are initially linked in the available buffer chain. The Available Buffer-Chain Entry Table (ABC Entry Table) contains all the entries for the available buffer chain. This becomes one push-down stack. Each buffer is one stack frame. This feature of an SIOCS allows a programmer to define a file as an internal file (i.e. the core memory of the primary high speed store). An internal file has many extra advantages for the programmer of complers. In the linguistic processors, (FORTRAN, COBOL, etc.) all the push-down stacks with variable length stack frames may now be maintained through SIOCS.

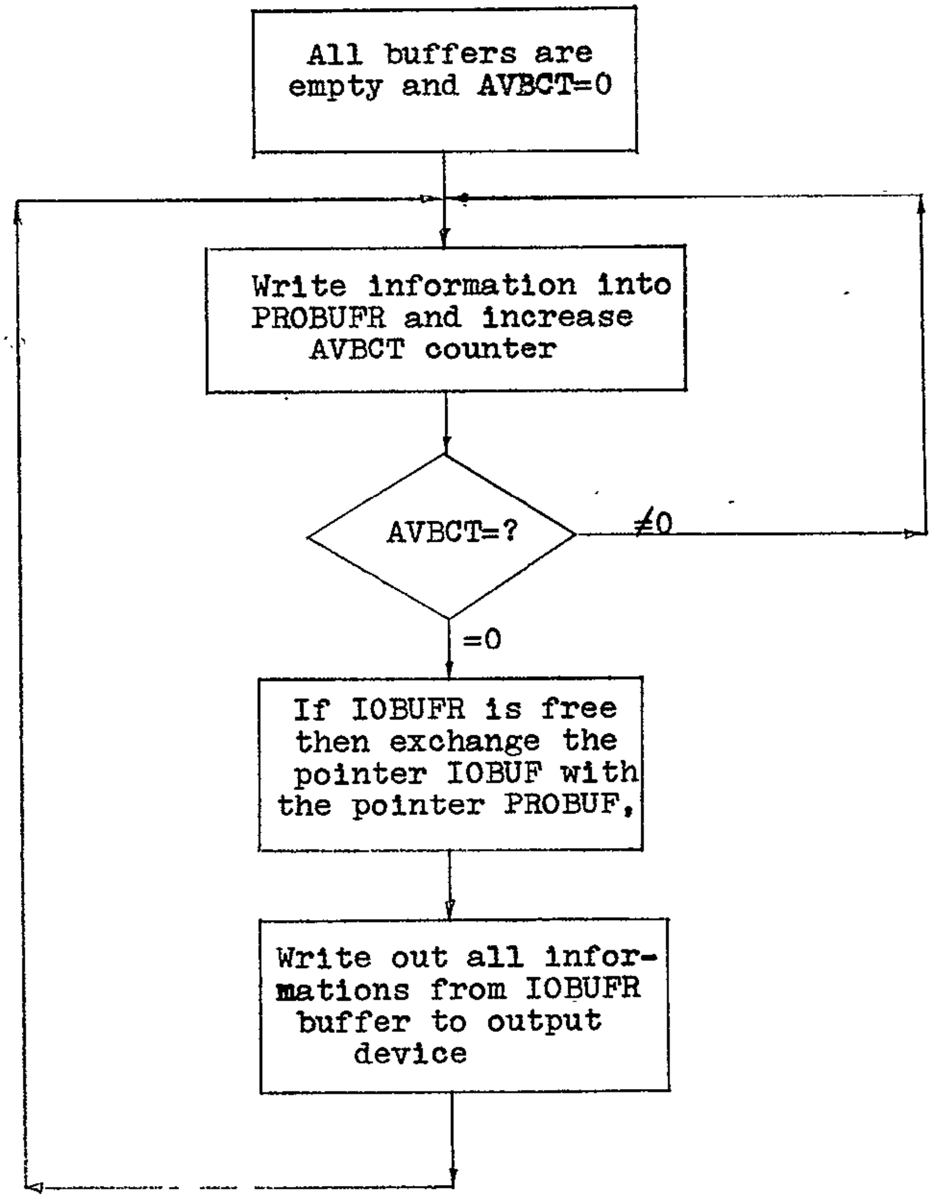


Fig. 12. The Logic Flow for Output Buffer Cycle

A PUSH-DOWN means write and a POP-UP means read. There is one entry in the ABC entry table for each of the buffers on any one size. Whenever a file is opened or redefined, the SIOCS searches the available buffer chain, obtains two buffers of proper size from one of the buffer chains and assigns them to that file. When these two buffers are no longer used, the SIOCS release them and returns them to the available buffer chain.

An example of the available buffer chain and the ABC entry table is shown in Figure 13. The entry to this chain is in the ABC entry table. The LINK field of this entry as shown in Figure 13 contains the address of the first buffer of this buffer chain.

The SIZE field of this entry describes the size of the buffer. Note that all the insertions and deletions to the chain are made at the leftend or top of stack (i.e. the end which is pointed to by the entry in ABC table).

4.1.2.4 Non system buffering

There are several different buffering techniques that have been built into our input-output buffering system. As described in the previous sections. The SIOCS buffering system employs: double buffers for files, variable lengths for buffers, and a rotation method for buffer usage. In order to allow the programmer to use any buffering technique which he considers more efficient, the SIOCS allows him to use his own buffering routine without referring to SIOCS buffering routines. This is on the assumption that a good programmer will know more about the I/O characteristics of his job than any system program could. For most cases a programmer will rely upon SIOCS. However in certain places he will, for specified files, switch over to his own, buffering system by declaring those files as NONBUF files.

4.1.3 I/O request

I/O requests can be separated into two distinct types, data transmission requests and non-data requests.

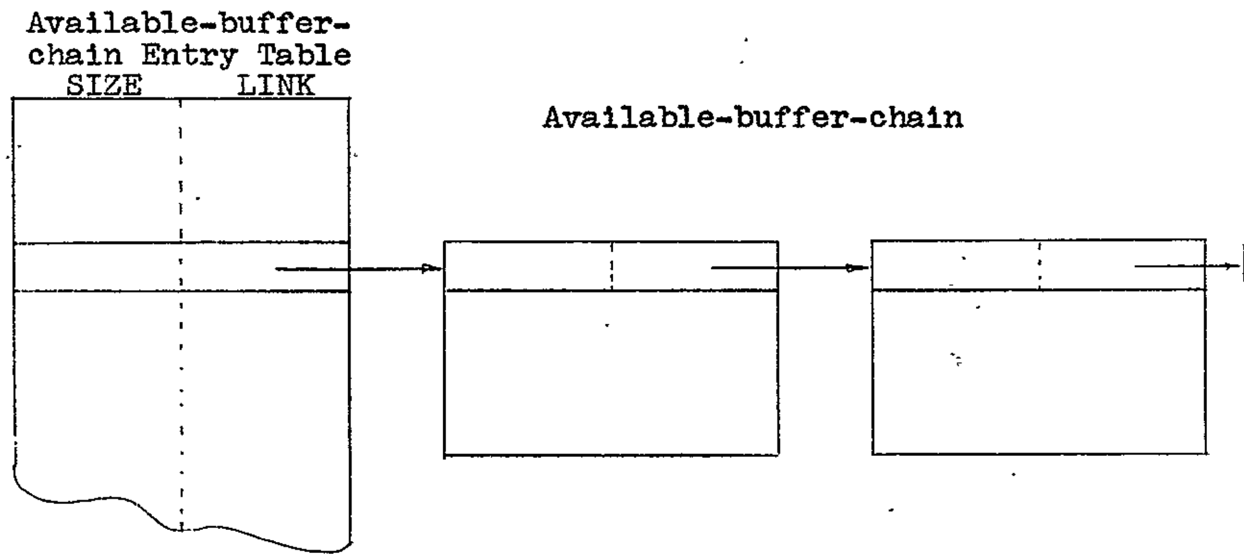


Fig. 13. The Structure of the Available Buffer Chain and the ABC Entry Table

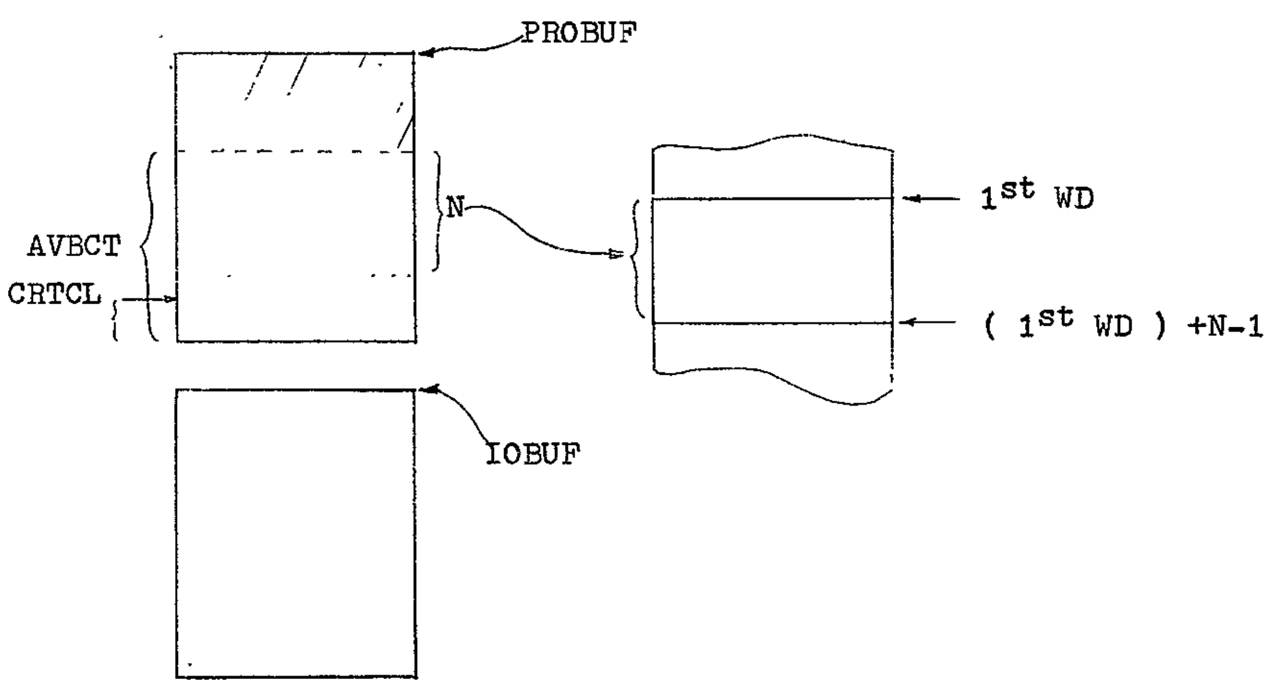


Fig. 14. Reads Input File Under Conition: AVBCT>N and CRTCL>(AVBCT-N)

4.1.3.1 Data transmission request

Figure 8 shows the data transmitted into and out of the computer. SIOCS accomplishes the actual transmission of records from the input unit to the input buffer and from the output buffer to the output unit. The macro instructions used for data transmission requests are:

LABEL	OPERATOR	OPERANDS
	READ	FILENAME, ERR, EOR, INTRUP, 1ST WD ADDR, N
	WRITE	FILENAME, ERR, INTRUP, 1ST WD ADDR, N

The READ macro instruction reads N words from the file and transfers it into consecutive memory locations (1st word address) through (1st word address+N-1)

The WRITE macro instruction writes the data from consecutive memory locations (1st word address) through (1st word address+N-1) to the file specified.

(A) Read input file

There are four conditions which can possibly occur when reading an input file

(1) Figure 14 shows the first condition AVBCT>N, and CRTCL>(AVBCT-N) when the READ macro instruction is given.

    After processing this READ macro instruction, AVBCT is decreased by N.

(2) Figure 15 shows the second condition CRTCL greater than or equal to (AVBCT-N)

    After processing this READ macro instruction, two more actions take place:

    (a) Decrease AVBCT by N
    (b) Set up an input unit for this file and initiate an input operation to fill data into the IOBUFR buffer.

(3) The third condition is that AVBCT less than N and N is less than

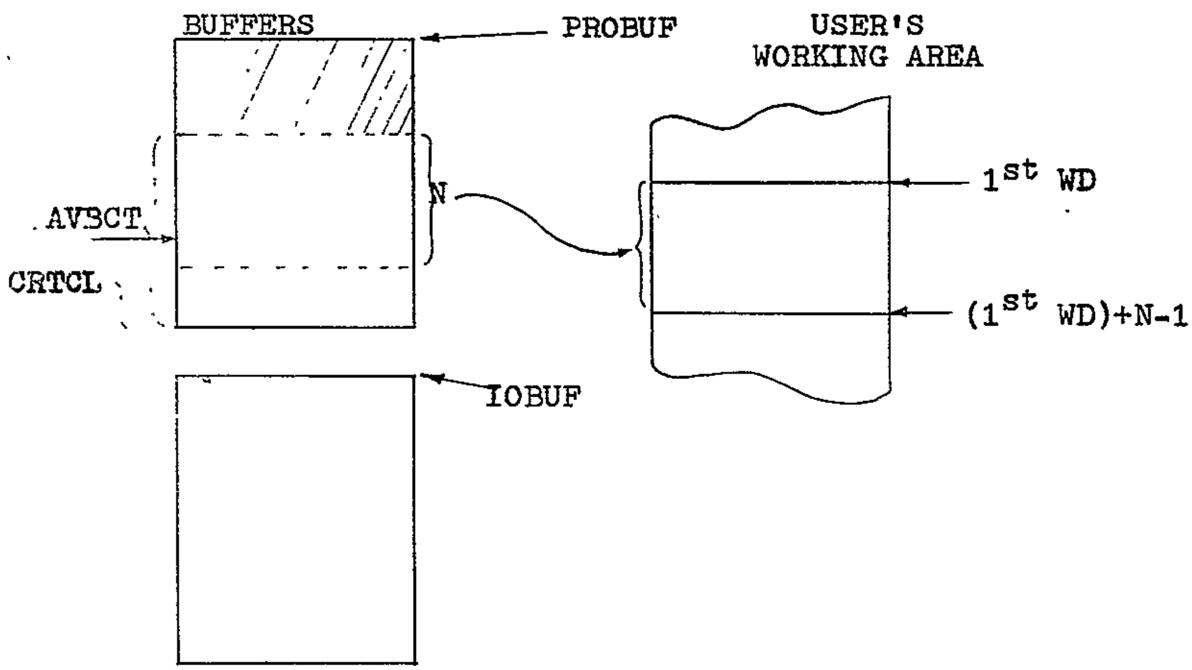


Fig. 15. Reads Input File Under Condition:

CRTCL $\geqslant$(AVBCT-N)

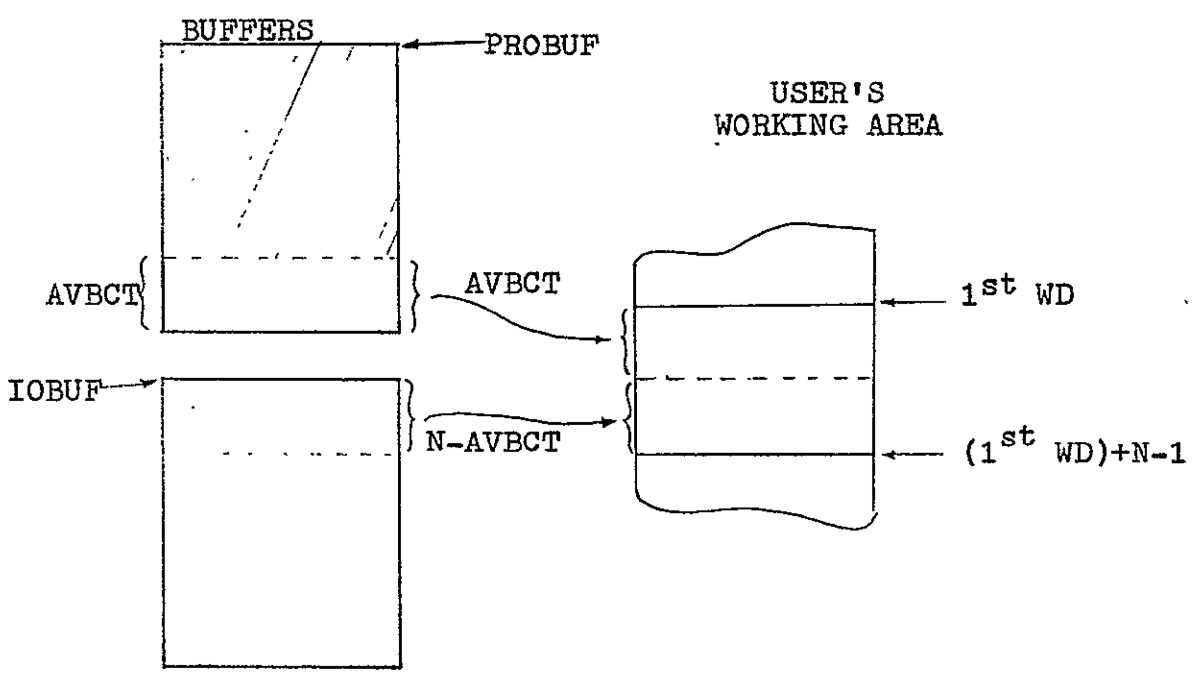


Fig. 16. Reads Input File Under Condition:

AVBCT<N$\leqslant$AVBCT+SIZE

or equal to AVBCT + (buffer size). That is, N can be contained in the total buffer space available. If this condition occurs when the IOBUFR buffer is empty (i.e. AVBCT CRTCL), then the central processor unit is forced to wait until the IOBUFR buffer is filled with data. Figure 16 shows the operations after the IOBUFR buffer is filled with data. After processing the READ macro instruction, SIOCS does the following:

(a) Exchange IOBUF with PROBUF

(b) Reset AVBCT equal to (buffer size-(N-AVBCT))

(4) When the condition N > AVBCT+(buffer size) occurs, then the following algorithm is applied:

(a) If the IOBUFR buffer is empty, then the CPU is forced to wait until the IOBUFR is filled with data

(b) Transfer AVBCT words from the PROBUF buffer to working area

(c) Exchange the pointer IOBUF with the pointer PROBUF and reset

* AVBCT equal to (buffer size),
* 1^{st} WD ADDR equal to (1^{st} WD ADDR+AVBCT),
* N equal to (N-AVBCT)

(d) Go to step A, B, C, or D depending on the conditions:

* AVBCT > N and CRTCL > AVBCT-N ---Go to step (1)
* AVBCT ≥ N and CRTCL ≤ AVBCT-N ---Go to step (2)
* AVBCT < N and N ≤ AVBCT+(buffer size) ---Go to step (3)
* N > AVBCT+(buffer size) ---Go to step (4)

respectively.

(B) Write output file

There are three conditions that can possibly occur when the WRITE output file macro instruction is given.

(1) When AVBCT > N occurs, SIOCS transfers N words from working area to output buffer, as shown in Figure 17.

(2) When the condition $AVBCT \leqslant N \leqslant AVBCT+$(buffer size) occurs, SIOCS transfers AVBCT words from working area to the PROBUFR buffer. Now test if the buffer pointed to by IOBUF is free, if it is then the remaining words are transferred from working area to to the IOBUFR buffer. However, if the OPBUFR buffer is busy, SIOCS forces the CPU to wait until the IOBUFR buffer is free, and then transfers data to that buffer, as shown in Figure 18.

After processing the data transmission, SIOCS does the following:

(a) Exchanges the pointer PROBUF with the pointer IOBUF

(b) Resets AVBCT equal to (buffer size)-(N-AVBCT)

(c) Initiates a write command to the channel to output data from the IOBUF buffer to output unit.

(3) The other condition is AVBCT+(buffer size)<N. When this condition occurs, the following algorithm is applied:

(a) Transfer AVBCT words from working area to the PROBUFR buffer and reset N equal to (N-AVBCT), increase 1^{st} WD ADDR by AVBCT

(b) If the PROBUFR buffer is busy, then the CPU is forced to wait until it is free

(c) Exchange the pointer PROBUF with the pointer IOBUF, and set AVBCT equal to buffer size

(d) Go to case A, B, or C depending on the conditions:

* $AVBCT > N$ ---Go to case (1)
* $AVBCT \leqslant N < AVBCT+$(buffer size) ---Go to case (2)
* AVBCT+(buffer size) $< N$ ---Go to case (3)

respectively.

(C) <u>READ/WRITE a non-buffer file</u>

To execute a READ/WRITE to or from a NONBUF (non-system-buffered) file is to read or write data directly from an I/O device into working area, as illustrated in Figure 19.

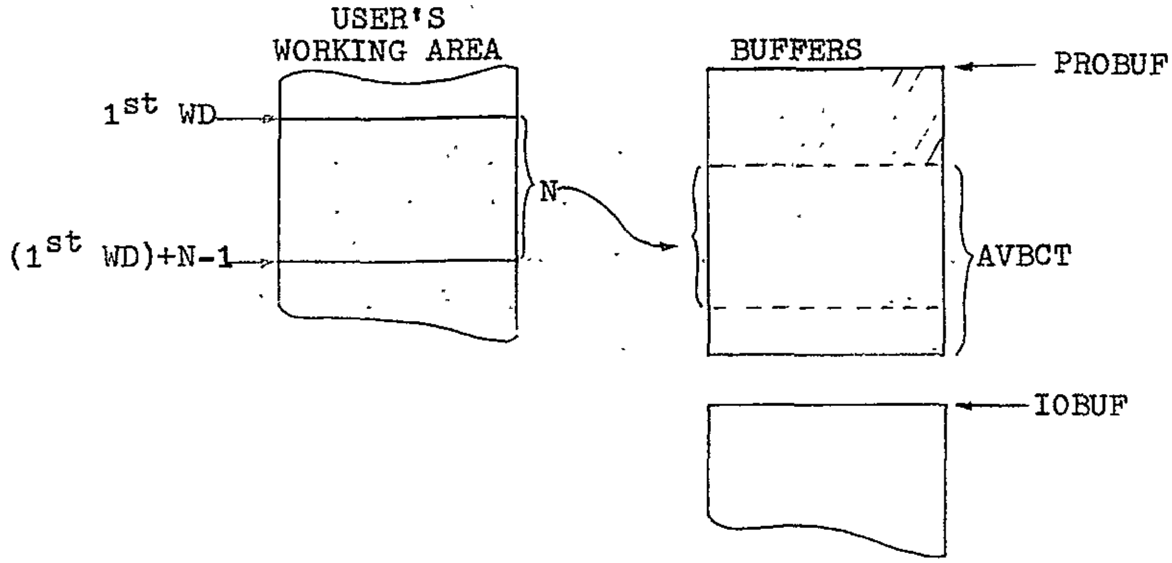


Fig. 17. Writs Output File Under Condition:

AVBCT>N

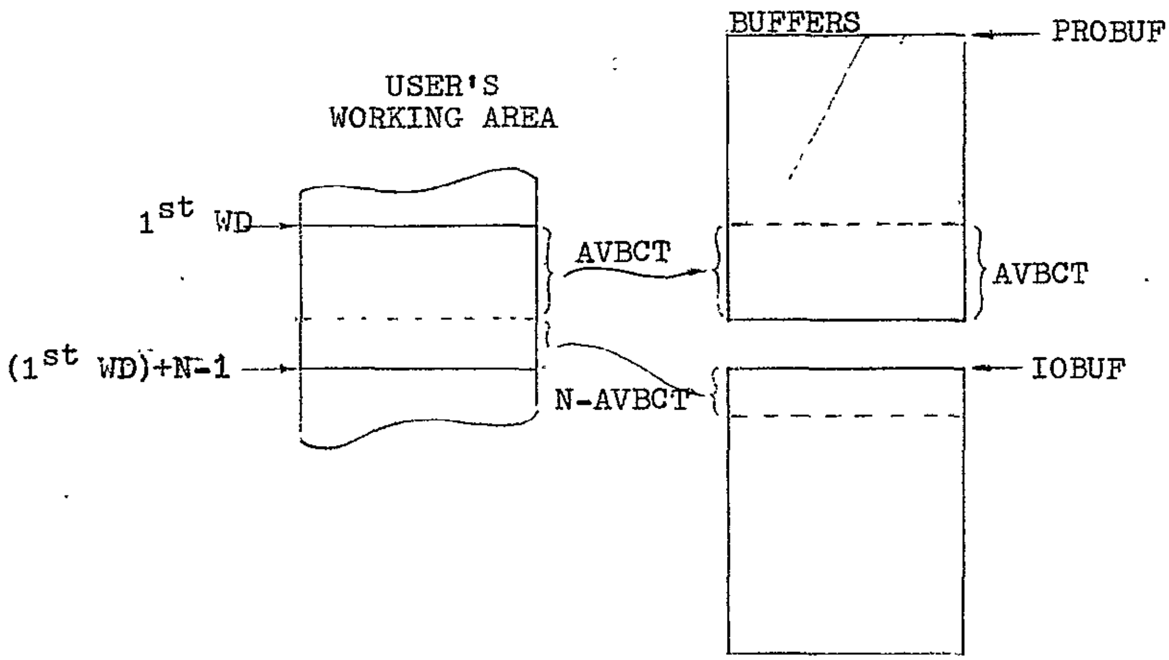


Fig. 18. Writs Output File Under Condition:
AVBCT≤N≤AVBCT+SIZE

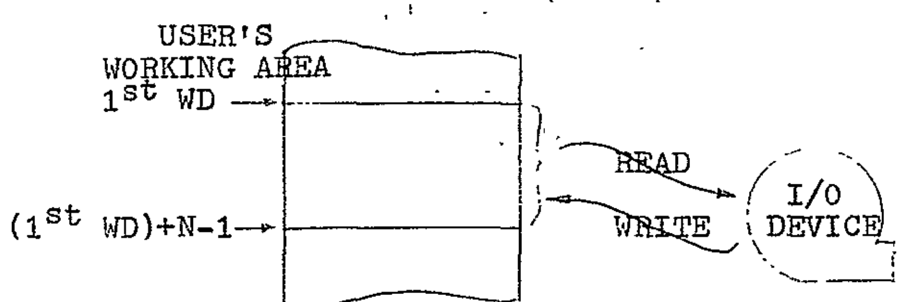


Fig. 19. Read/Write an Non-Buffer File

4.1.3.2 Non-data request

There are four macro instructions used for I/O requests which do not refer to data, these are:

1. REWIND:

Label	Operator	Operands
	REWIND	FILENAME

2. BKSP: Backspace N records

Label	Operator	Operands
	BKSP	FILENAME,N

3. MOVE: Move forward and pass N end-of-file markers

Label	Operator	Operands
	MOVE	FILENAME,N

4. WEOF: Write an end-of-file marker

Label	Operator	Operands
	WEOF	FILEMAME

4.2 The Input-Output Scheduling

Some of the functions of the I/O Scheduler are handling I/O interrupts, scheduling the operations on I/O units and channels, and checking for correct functioning of all I/O. The main purpose of this I/O Scheduler is to keep the input/output devices as busy as possible and to insure that the I/O operations are as efficient as possible.

Whenever an I/O operation is required by user's request or required by SIOCS, an I/O request-entry is generated. This I/O request-entry contains all the necessary information for that I/O operation. This I/O operation will probably not be able to be executed immediately because the channel or unit in question may be busy. In this case, the I/O request entries on each unit will be constructed. The I/O initiation routine inspects these queues when a new I/O operation is to be started on a unit.

When all I/O operations associated with one of the currently executed request-entry are completed, or an error or abnormal condition has been detected, an interrupt occurs. The I/O interrupt routine identifies the interrupted channel and records an I/O status descriptor. From the information which is stored in the I/O status descriptor, all the I/O control blocks are updated. After that, the I/O initiation routine is again called to start the next I/O operation as quickly as possible.

4.2.1 I/O Initiation

(a) I/O Request entry

The I/O request entry is a group of contiguous fields which are generated for each I/O operation requested by the IOREQU macro instruction. These fields (namely, the function code, the file name, the interrupt address, the first word address, the error reject address, and the number of words transmitted) contain the information needed to define a specific input/output operation on a particular I/O unit. The format of an I/O request

PRVSIO		NEXTIO
F.C.	FILENAME	INTRUP
N	1^{st} ADDR.	ERR

Fig. 23. The Format of an I/O Request Entry

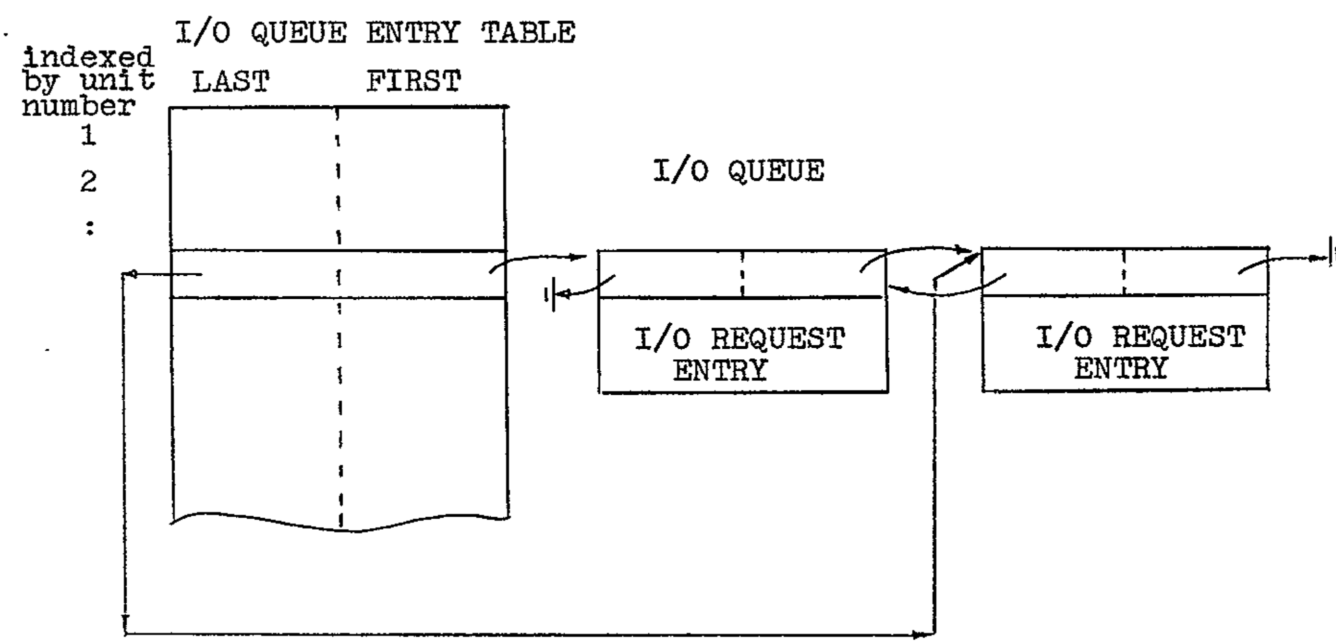


Fig. 24. An Example of I/O Request Queue

entry is shown in figure 23.

In the figure, note that the PRIVIO field of the I/O request-entry is used to stored the address of the previous I/O request for the same I/O unit, while the NEXTIO field is used to store the address of the next I/O request for the same I/O unit.

(b) I/O Request queue

The I/O request queue is a list of I/O request entries which are currently awaiting service by a particular physical device. Since an I/O request entry generated by an IOREQU macro instruction may not be able to executed until all the previous I/O requests are finished. The I/O request queue is a holding queue for I/O service. For each physical device there is one I/O request queue. An example of the I/O request queue is shown in figure 24.

The I/O request queue entry table is used to stored the addresses of the first and the last I/O request entries of each I/O request queue. The unit number is used as an index number of this table. These queue are arranged on a first-in-first out basis. Refering to figure 24, the FISTIO field of the table entry points to the first I/O request entry in queue. This entry is the most critical entry and will be serviced first when this queue is activated. The LASTIO field of the table entry points to the last entry in the queue, and all insertions to the queue are made to this end of the queue.

(c) I/O Functional table

The I/O functional table is provided for the purpose of defining the operations of a given set of I/O functions with respect to a given set of I/O devices. SIOCS was designed to be as general as possible and still be simple.

Thus, this table could be expanded as future I/O equipment is added and there is no necessity for a modification to the logic of SIOCS. For the version of SIOCS, present the I/O request functions are: READ, WRITE, REWIND, MOVE, BKSP, and WEOF, while the I/O devices are: tape, cardreader, printer, cardpunch, and the console typewriter. Figure 25 is an example of an I/O functional table.

4.2.2 I/O Completion

(a) I/O Interrupt

When there is an interrupt signal for an I/O channel, an immediate attempt is made to activate the I/O interrupt routine. During the initialization of SIOCS, the program for the I/O interrupt routine is loaded into main memories and remains resident in the memory through out all the time. If there is no other interrupt being processed, then the I/O interrupt routine for the current interrupt given control immediatly. However, if there is another interrupt routine in processing then the current I/O interrupt signal is inhibited or placed in the waiting queue.

The major functions of the I/O interrupt routine are as follows:

(1) Identify the interrupted unit and channel,
(2) Record the I/O status descriptor,
(3) Check the I/O request queue for the interrupted unit, and initiate the first I/O request in queue, in case when the queue is not empty,
(4) Call result analysis routine to analysis the I/O reult and update the I/O control blocks,
(5) Pass control back to user's interrupt routine. If the user's interrupt address is specified and
(6) Return control to user's program.

(b) I/O status descriptor

The I/O status descriptor can be implemented either by

DEVICE / FUNCTION	TAPE	CARD READER	PRINTER	CARD PUNCH	CONSOL TYPEWRITER
1 READ	Read forward one record	Read onecard	Illegal	Illegal	Read until not busy
2 WRITE	Write forward one record	Illegal	Write one line	Punch one card	Write
3 REWIND	Rewind tape	Illegal	Illegal	Illegal	Illegal
4 MOVE	Space forward pass EOF mark	Illegal	Illegal	Illegal	Illegal
5 BKSP	Back space one record	Illegal	Illegal	Illegal	Illegal
6 WEOF	Write EOF mark	Illegal	Eject page	Punch EOF card	Illegal

Fig. 25. An Example of I/O Functional Table

I/O Status Descriptor

Memory Address	Char. count	Unit number	Error field

where

Memory Address : the memory address at which the I/O is terminated

Character count : how many charaters or how many words read in or write out

Unit number : physical unit identification

Error field : error indicator

Fig. 26. The Format of the I/O Status descriptor

hardware or by software. Here we assume that this descriptor is in a channel register. This descriptor has four fields which in turn contain the information which describes the current status of the I/O operation. These four fields are memory address, word count UCB addres, and error field. The memory address field contains the memory address of the point at which the I/O was terminated. The word count represents the number of words has been transmited, while the error field indicates the error conditions which is detected in the channel or the unit. The error field may subdivided into several fields such as the standard error field, and unit error field. The standard I/O error field is used to indicate the standard I/O error such as parity error, address error, end-of-file mark encountered,... etc. The format of the I/O status descriptor is shown in figure 26.

4.3 The unit interpretive routines

For each type of I/O device, there is an associated unit interpretive routine. The unit interpretive routines are hardware dependent, so that there is no point in having one general-purpose interpretive routine. In what follows, the algorithm for the unit interpretive routine is not intended for use on any one particular device, but rather for the presentation of those operations which must be performed by any unit interpretive routine. These operations are as follows:

(a) Initialization for processing upon re-entry.

(b) The set up on the I/O instruction code. This I/O instruction can be an intiation of a sequence of channel commands which are generated by step (c) and which are executed directly by the data channel.

(c) The set up of the channel program. This channel program must be stored in some fixed area in core and will be executed directly and independently by the data channel when the I/O instructions which initiate this channel program is issued.

(d) The issuing of the I/O instruction which is set up in step (b)

(e) The return of control to the user's program.

4.4 The Elements of SIOCS and the Communication among elements

4.4.1 Files and File Control Blocks

A file is a collection of related records treated as one unit. Before the I/O is activated for a file, that file must opened. Similarly, after all I/O is completed for a file, that file must be closed.

(A) File Types

An item buffering scheme is specified by selecting one of three possible types of files. Those are: IN (input), OUT (output), and NONBUF (nonbuffering)

Type	Buffering schema
IN	Double-buffer
OUT	Double-buffer
NONBUF	No system buffering

(B) File Control Block (FCB)

For each file used in SIOCS, a File Control Block is established in core storage. It keeps the file and the buffer information, and also links the buffer area used by the file to a Unit Control Block (UCB). The following figure shows the format and information included in the FCB.

FCW1	FILE NAME				
FCW2	OC	TYPE	UCB ADDRESS		
FCW3	IOBUF		PROBUF		
FCW4	SIZE	CRTCL	BUSY	EOF	AVBCT

Where TYPE: file type

UCBADD: Address of the Unit Control Block (UCB) used by this file

IOBUF: A pointer which points to the IOBUFR buffer

PROBUF: A pointer which points to the PROBUFR buffer,

OC: An open-close indicator,

SIZE: Buffer size,

CRTCL: Critical number of the input probuf buffer,

AVBCT: Available counter which counts the available words remaining in the PROBUFR buffer,

BUSY: A buffer busy indicator,

EOF: An end-of-file indicator,

The File Control Block (FCB) is generated by the OPEN macro instruction and released upon termination of the run.

4.4.2 Buffers

A buffer is a block of core storage used to compensate for the difference in data handling rates when transmitting data from device to core or vice versa. There are two buffers for each IN or OUT file in SIOCS. These two buffers have the same size and are pointed to by the pointers OBUF and PROBUF that are stored in the third work of the FCB. The third and fourth words of the FCB contain information about this double-buffer. Figure 20 shows the format of these two buffers.

4.4.3 Units and Unit Control Blocks

A unit is an I/O device attached to a computer. SIOCS uses symbolic assignments to allow flexibility in assigning physical input/output units. When a program is written, a symbolic unit is assigned to a file. At run time, a proper physical unit is assigned to the symbolic unit. At system generation time, the number of units of each physical type is specified and the symbolic unit table is built accordingly. Also, at system generation time, all physical units are assigned to a symbolic unit by linking the unit control blocks to the symbolic unit table. The format and linkage relation between the Unit Control Blocks with symbolic unit table is explained in figure 21.

4.4.4 Channels and Channel Control Blocks

A channel is a hardware device (a small computer) designed to be

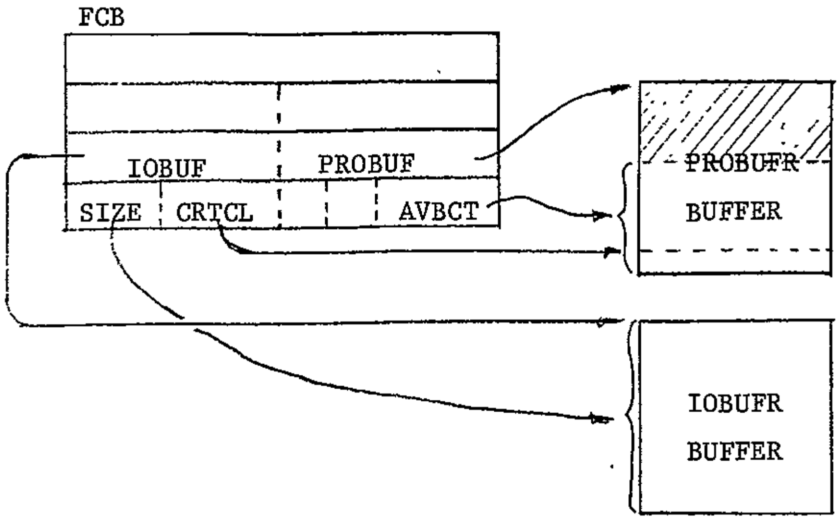
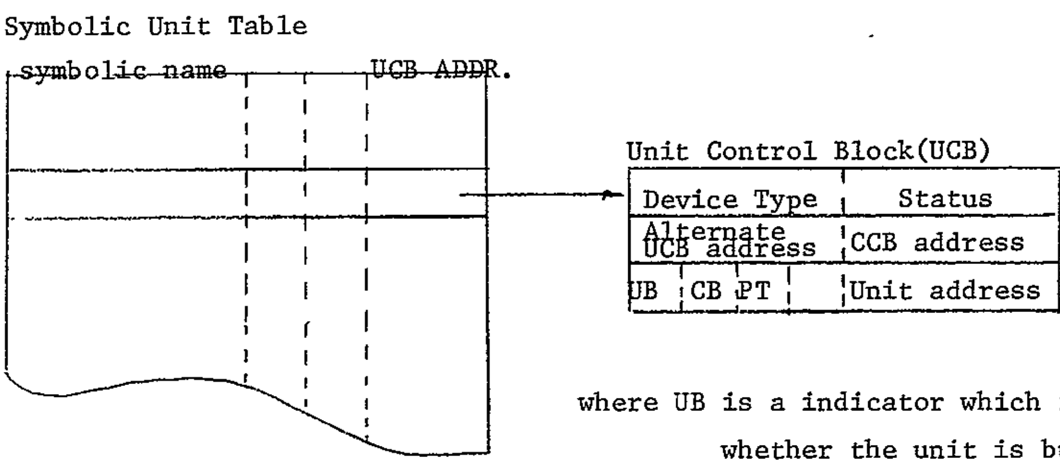


Fig. 20 The Format of Double-buffer



where UB is a indicator which indicates whether the unit is busy or not,

CB is a indicator which indicates the channel is busy or not,

PT is a indicator which indicates the unit is been protected by the system or not.

Fig. 21 The Format and the Linkage between the UCB and the Symbolic Unit Table operated in parallel with the CPU and carry out input or output operations. Each channel is associated with one Channel Control Block (CCB) in SIOCS. The Channel Control Block defines channel status and interupt selections. The format of CCB is:

CB	UNTADD	I	INTRUP

Where UNTADD is the address of UCB of the current (or last) unit using this channel

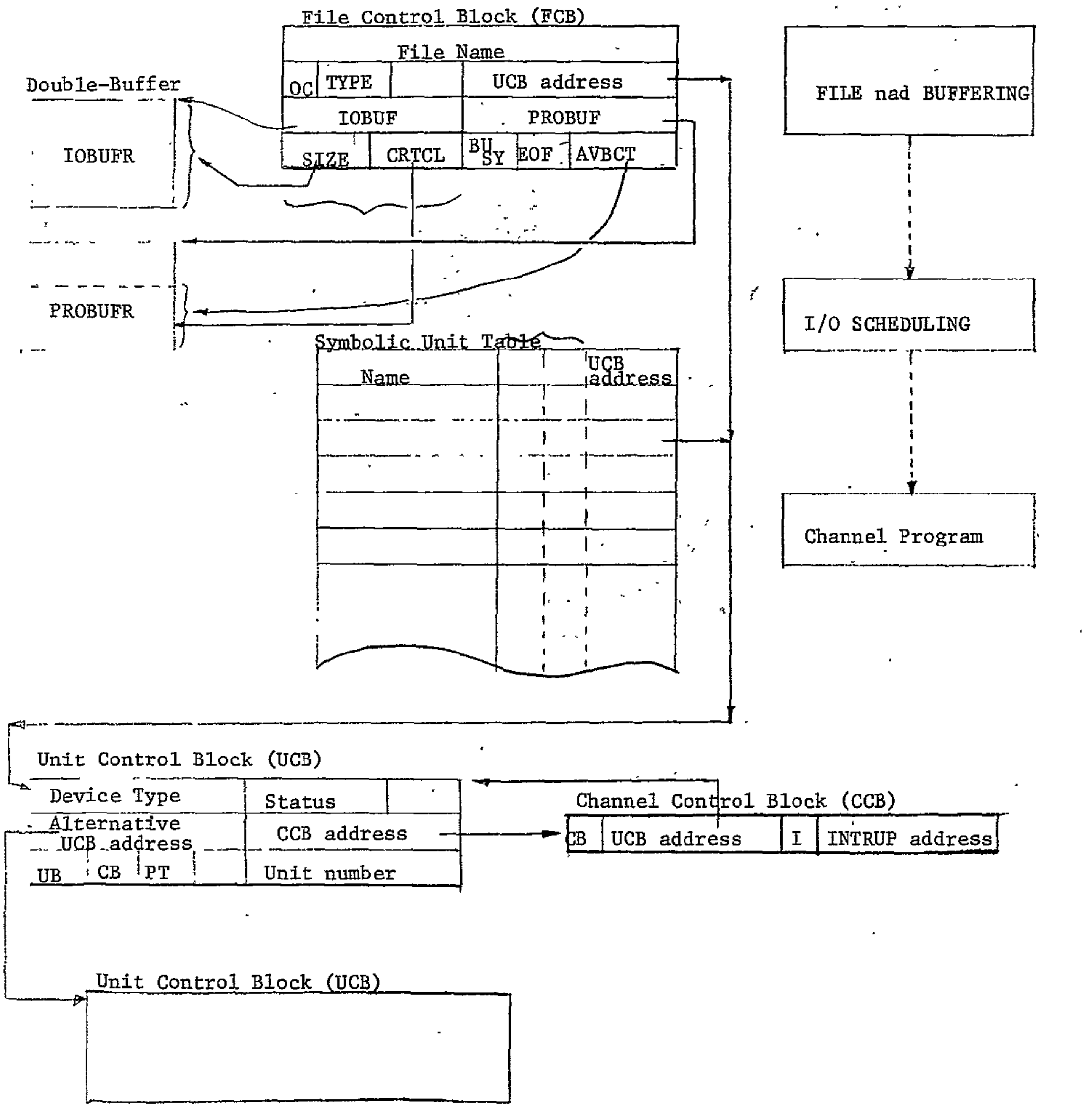
CB is a indicator which indicates whether the channel is busy or not

I is the interrupt indicator which indicates whether the interrupt is selected by the user or not

INTRUP is the entry of the user's interrupt routine

4.4.5 Communication among control blocks

As shown in Figure 22.


File Control Block (FCB)
File Name
OC
TYPE
UCB address
IOBUF
PROBUF
SIZE
CRTCL
BUSY
EOF
AVBCT
Double-Buffer
IOBUFR
PROBUFR
FILE nad BUFFERING
I/O SCHEDULING
Channel Program
Symbolic Unit Table
Name
UCB address
Unit Control Block (UCB)
Device Type
Status
Alternative UCB address
CCB address
UB
CB
PT
Unit number
Channel Control Block (CCB)
CB
UCB address
I
INTRUP address
Unit Control Block (UCB)

5. The Algorithms of SIOCS

5.1 The overall diagram of the SIOCS

The SIOCS is an interface between the operating system and the input-output devices associated with that system. All requests made by the operating system for input-output operations are directed to this interface. The SIOCS analyzes each request and takes appropriate action. This action consists of scheduling input-output operations, setting up the I/O areas associated with the I/O operations and, in general, handling all of the many and various functions needed in reading and writing tape, card, and printer, and their records. After a request has been serviced, the SIOCS returns control to the user routine. An overall functional block diagram of SIOCS is shown in Figure 27. The overall algorithm of SIOCS is shown in Figure 28.

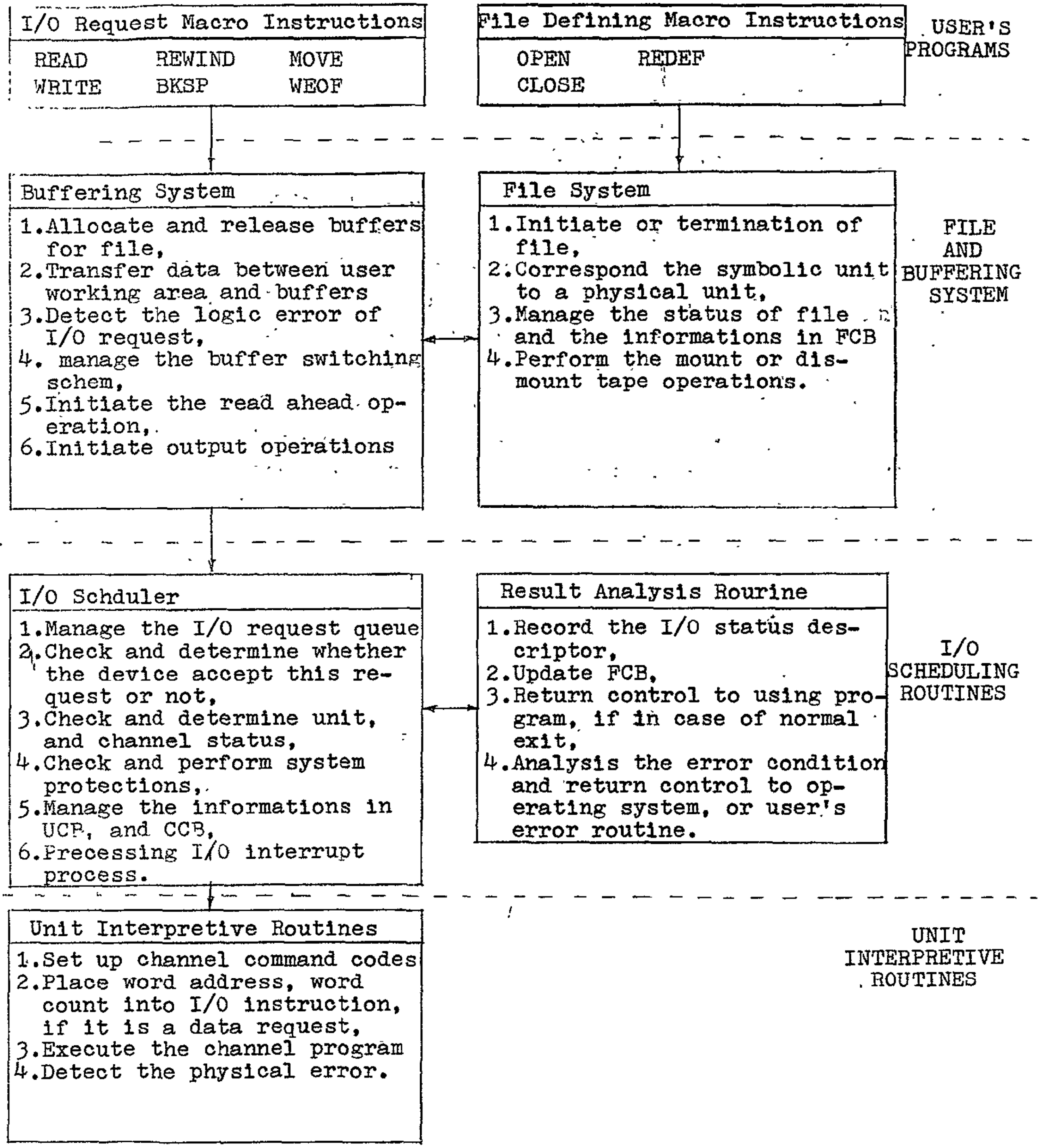


Fig. 27. An Overall Functional Block Diagram of SIOCS

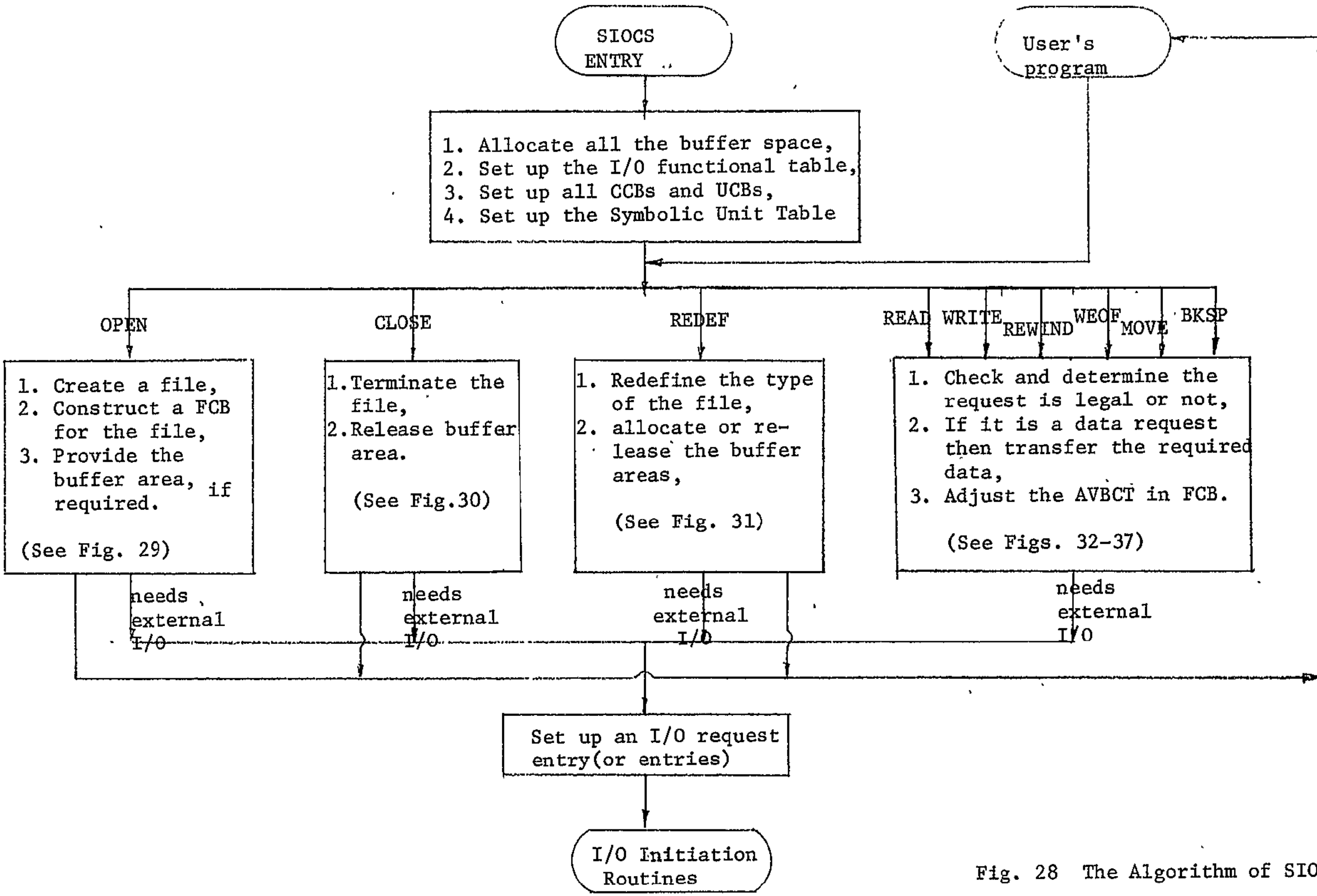


Fig. 28 The Algorithm of SIOCS

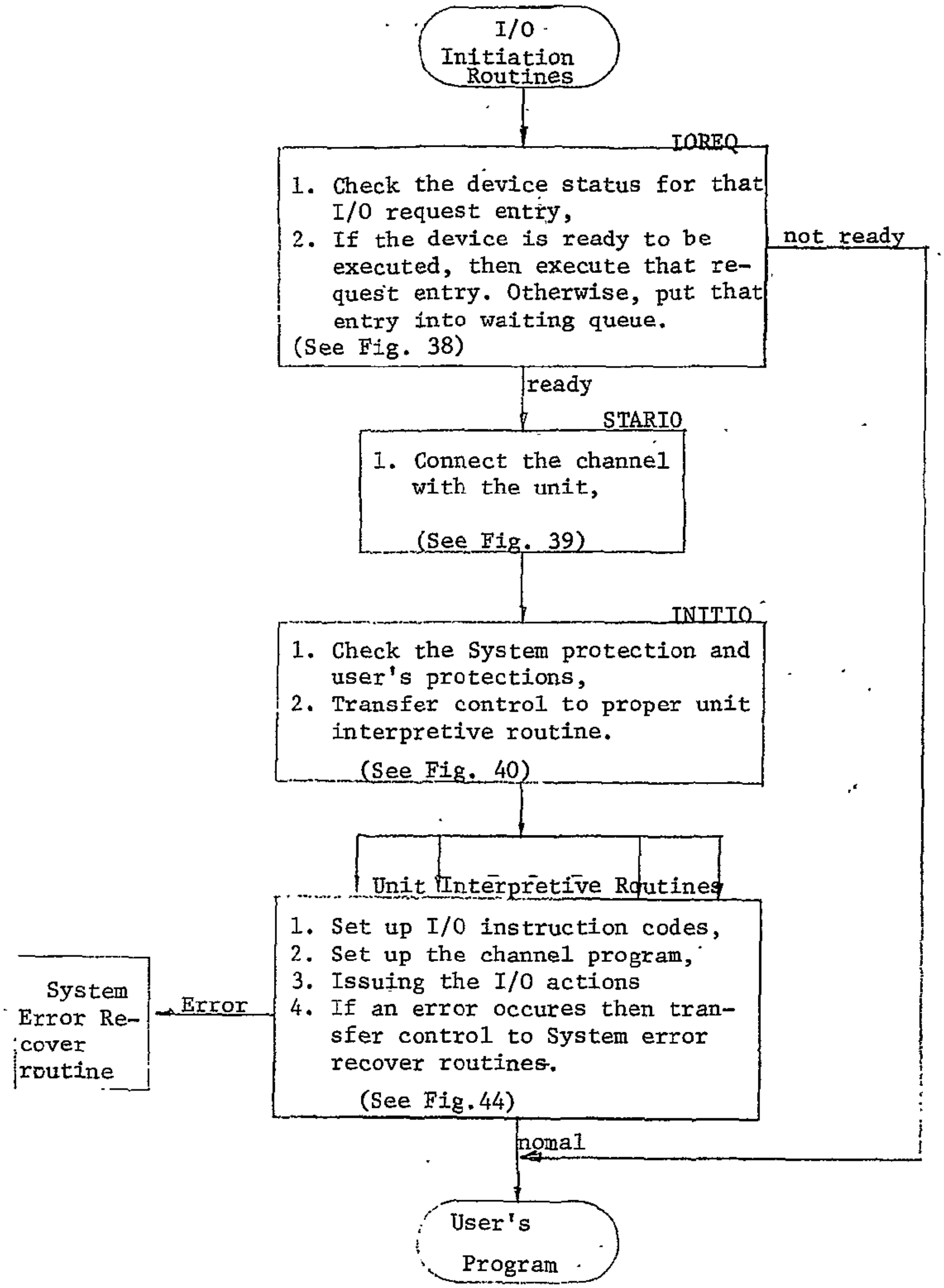


Fig. 28 The Algorithm of SIOCS, Part 2. The I/O Initiation Routines and Unit Interpretive Routines

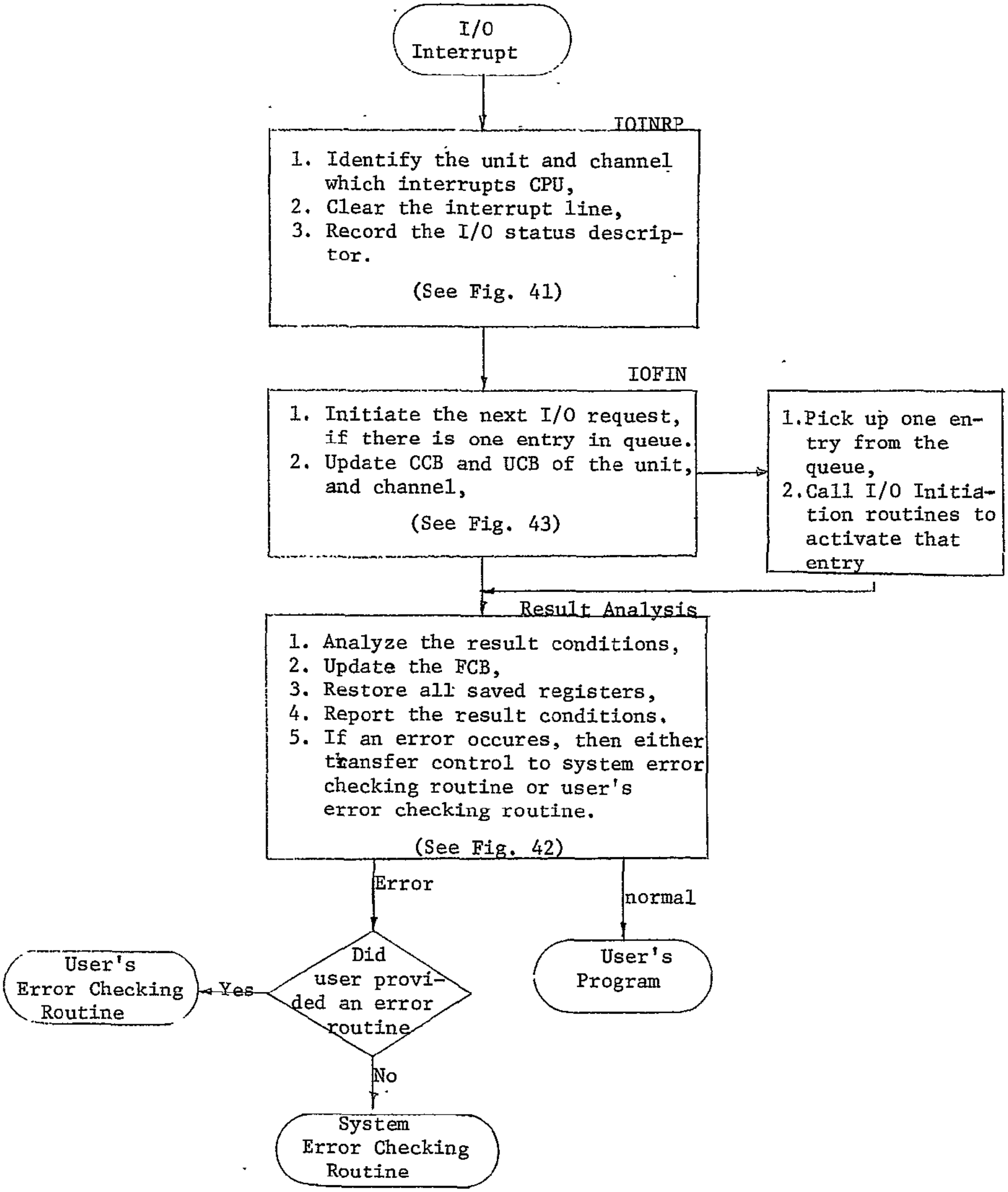


Fig. 28 The Algorithm of SIOCS, Part 3. The I/O Completion Routines

5.2 Description of each routine

SIOCS routines can be divided into three classes. These are: the file and buffering routines, the I/O scheduling routines, and the unit interpretive routines. The description of each routine presented in this section consists of its purpose, major objectives, its input and output parameters, and the algorithm in the flow chart form.

5.2.1 The file and buffering routines

There are nine major routines which manage the files and buffers in SIOCS. These nine routines can be divided into two groups. One of the groups is the file declaration routines which includes the OPEN routine, the CLOSE routine, and the REDEF routine. The other group is the I/O request routines. These consist of READ, WRITE, BKSP, WEOF, MOVE, and REWIND routines.

5.2.1.1 The file declaration routines

(a) The OPEN ROUTINE

Purpose: To create a file and initiate I/O operations for that file.

Major Objectives:
1. Generate a FCB for the file.
2. Fill in the information of FCB.
3. Mount tape, if it is a tape file.
4. Allocate two buffers with proper size and store these two buffer addresses in FCW 3.
5. Perform the REWIND operation, if the REWIND option exists.
6. Initiate a READ operation, if it is a IN file.

Calling Sequence: The OPEN routine is called by the OPEN macro instruction whose format is shown in Section 3.2.1.

Input: The sources of input to the OPEN routine are:
1. The parameters of the OPEN macro instruction (See Section 3.2.1). Those are the TYPE of file, the symbolic name of a device which is

desired, and the REWIND tape option.

2. The symbolic unit table. This table resides in core at a fixed location.
3. The UCB's for every physical unit. Each physical unit has it's own UCB. The addresses of these UCB's are stored in the symbolic unit table associsted with its symbolic names. Note that one physical unit may associate with more than one symbolic name.
4. The Available-Buffer-Chain Entry Table (or ABC Entry Table) and the available buffer chains, the structure of ABC entry table and the available buffer chains are shown in Figure 13 .

Output: The output from the OPEN routine is:

1. A FCB, a FCB is generated and assigned to the file. The FCB contains all information about the file.
2. Mount tape message. A message will be sent to operator console, if this is a tape file.
3. Initiating of READ AHEAD. If this is a IN file

Algorithm and Flow chart: As shown in Figure 29.

(b) The CLOSE routine

Purpose: Terminate the activity of files.

Major Objectives:

1. Reset the open-closed indicator to 1, to indicate that file is closed.
2. Send out all the information that remains in buffers and write an end-of-file mark, if the file to be closed is an OUT file.
3. Release the buffer used by the file and return to available buffer chains, if the file to be closed is an IN or OUT type of file.
4. Clear all information in FCB except the file name.
5. Perform the REWIND, or UNLOAD operation, if specified.

FILENAME	OPEN	TYPE,DEVICE,(REWIND)

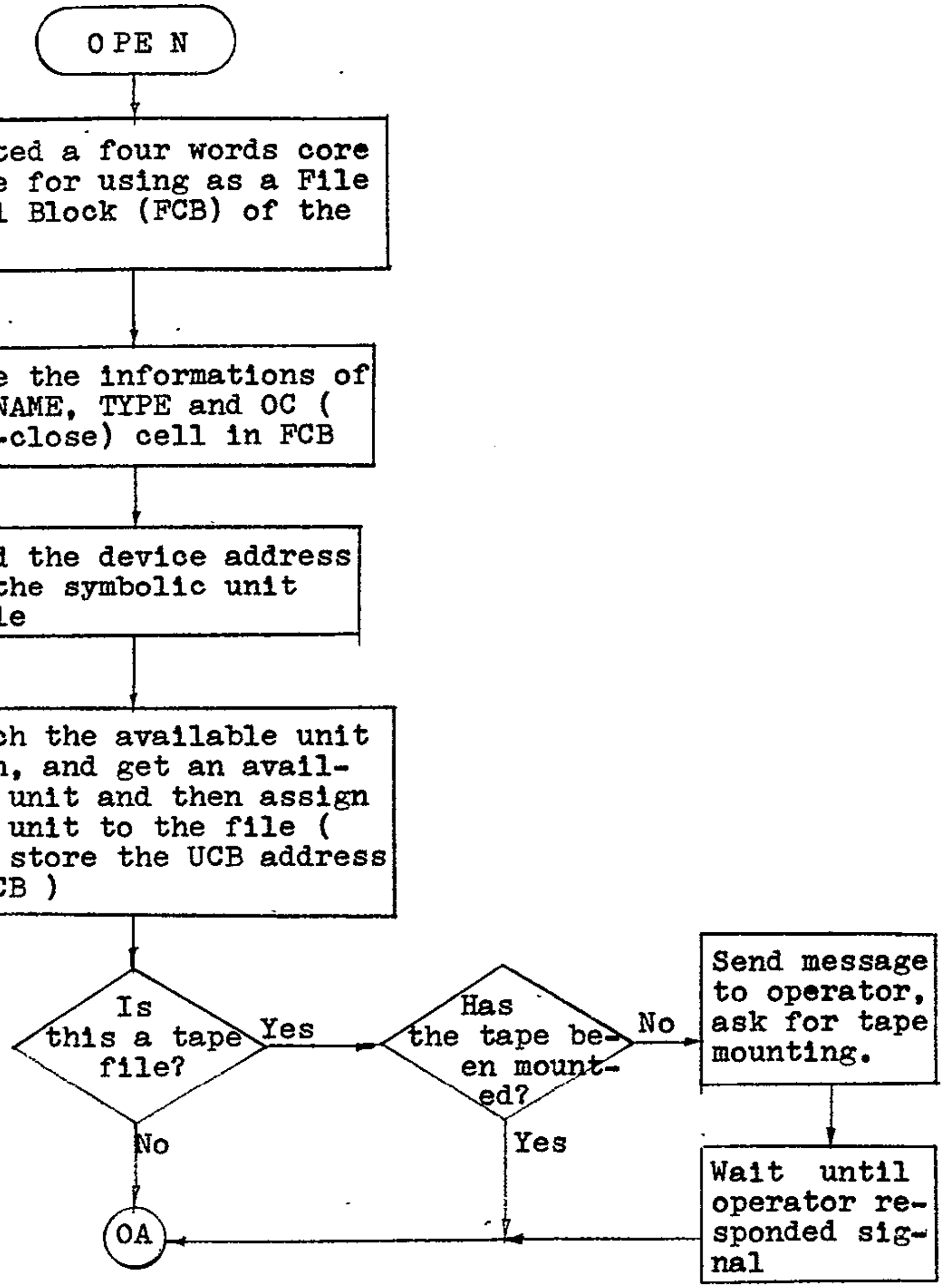


Fig. 29. The Flow Chart of the OPEN Routine

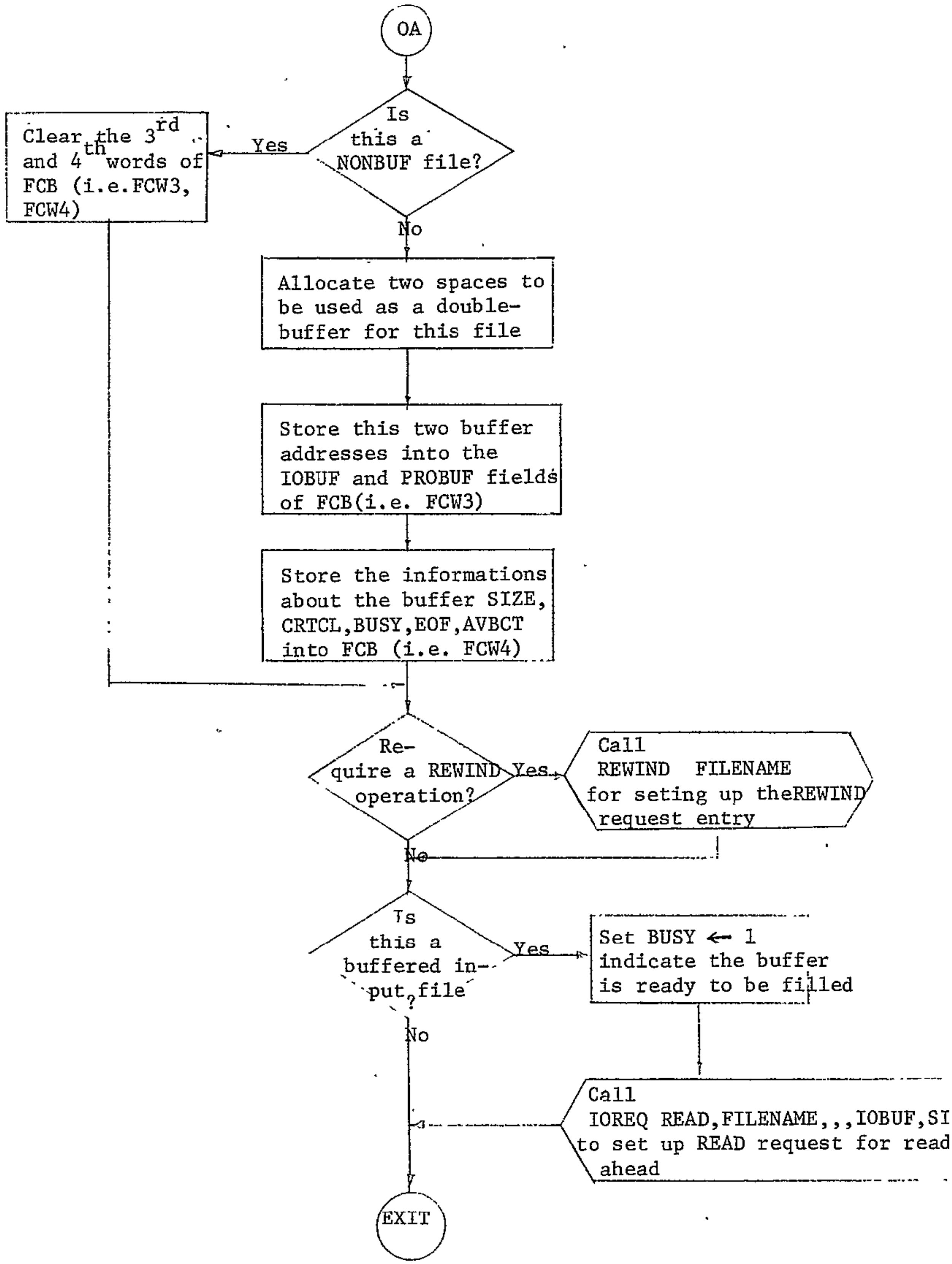


Fig. 29 The Flow Chart of the OPEN routine (cont.)

	CLOSE (,OPTION)	FILENAME.1,FILENAME2,...

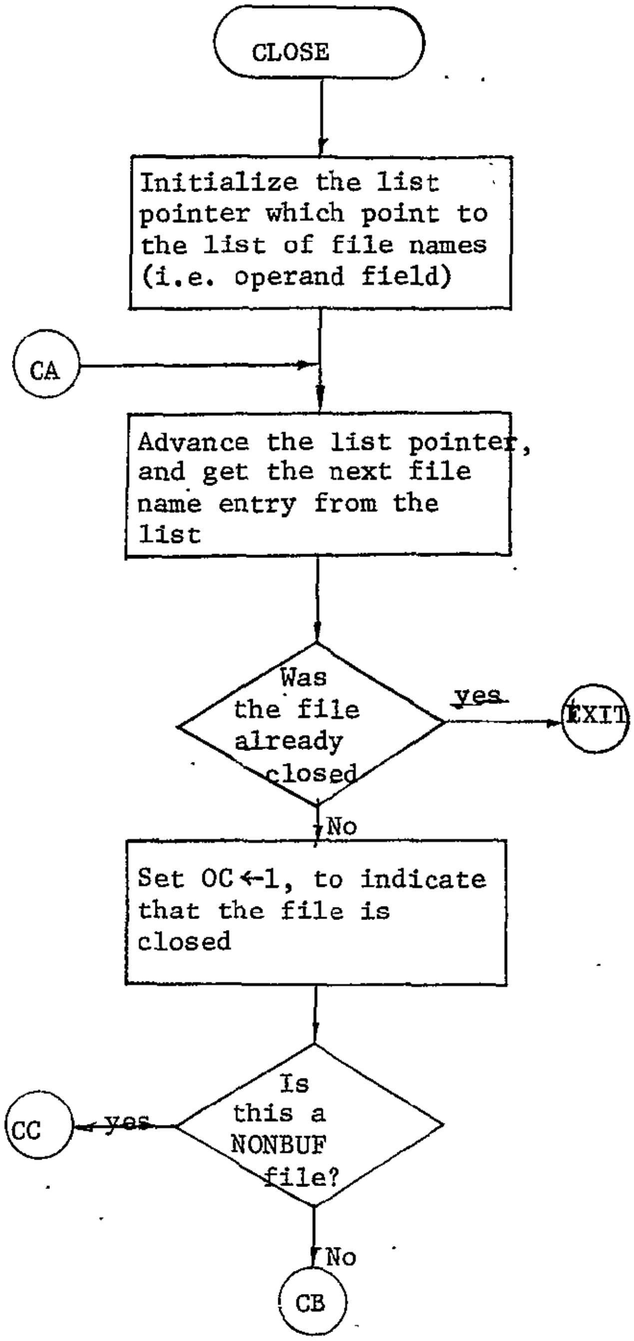


Fig. 30 The Flow Chart of CLOSE Routine

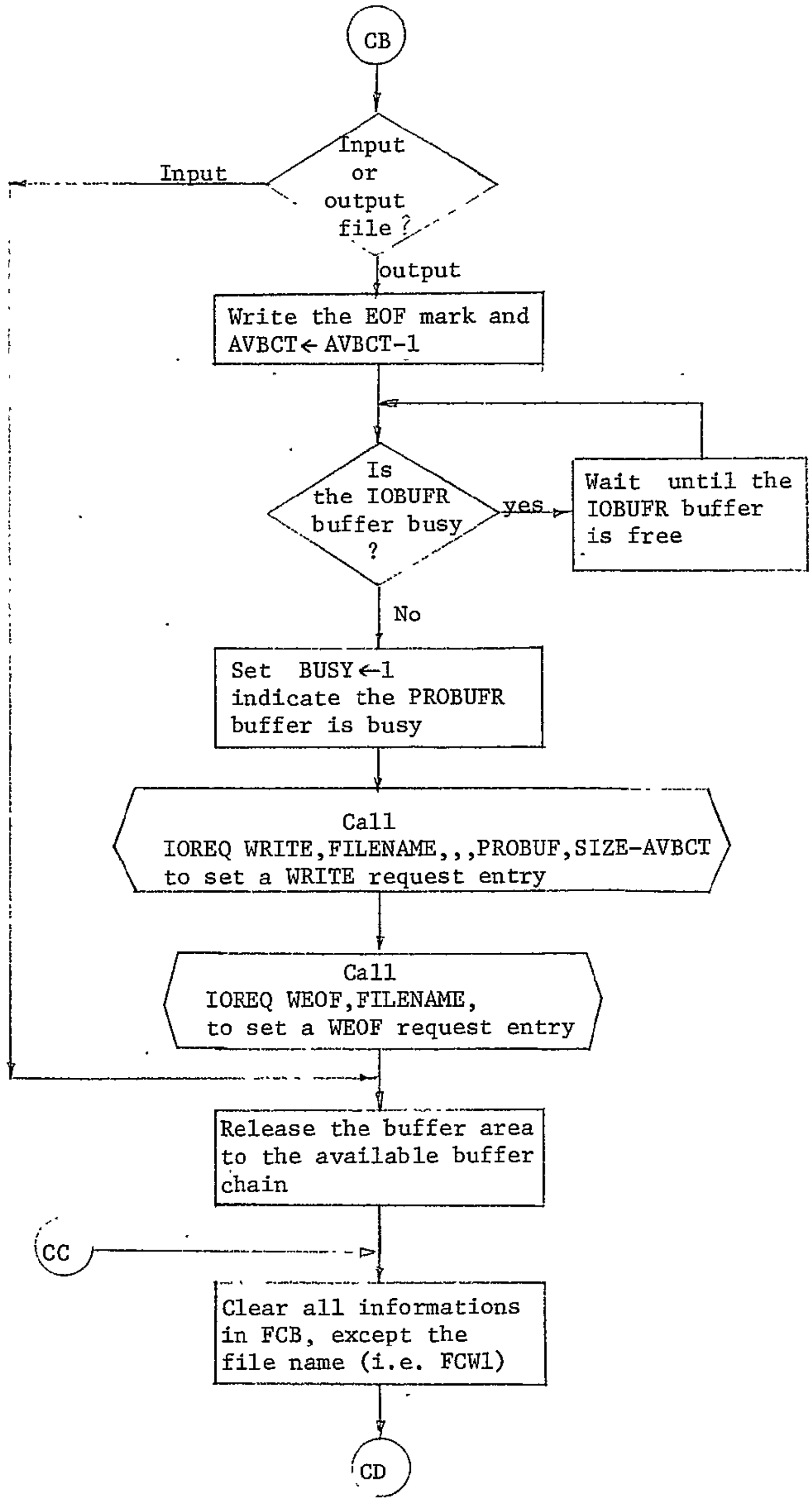


Fig. 30 The Flow Chart of CLOSE Routine (Cont.)

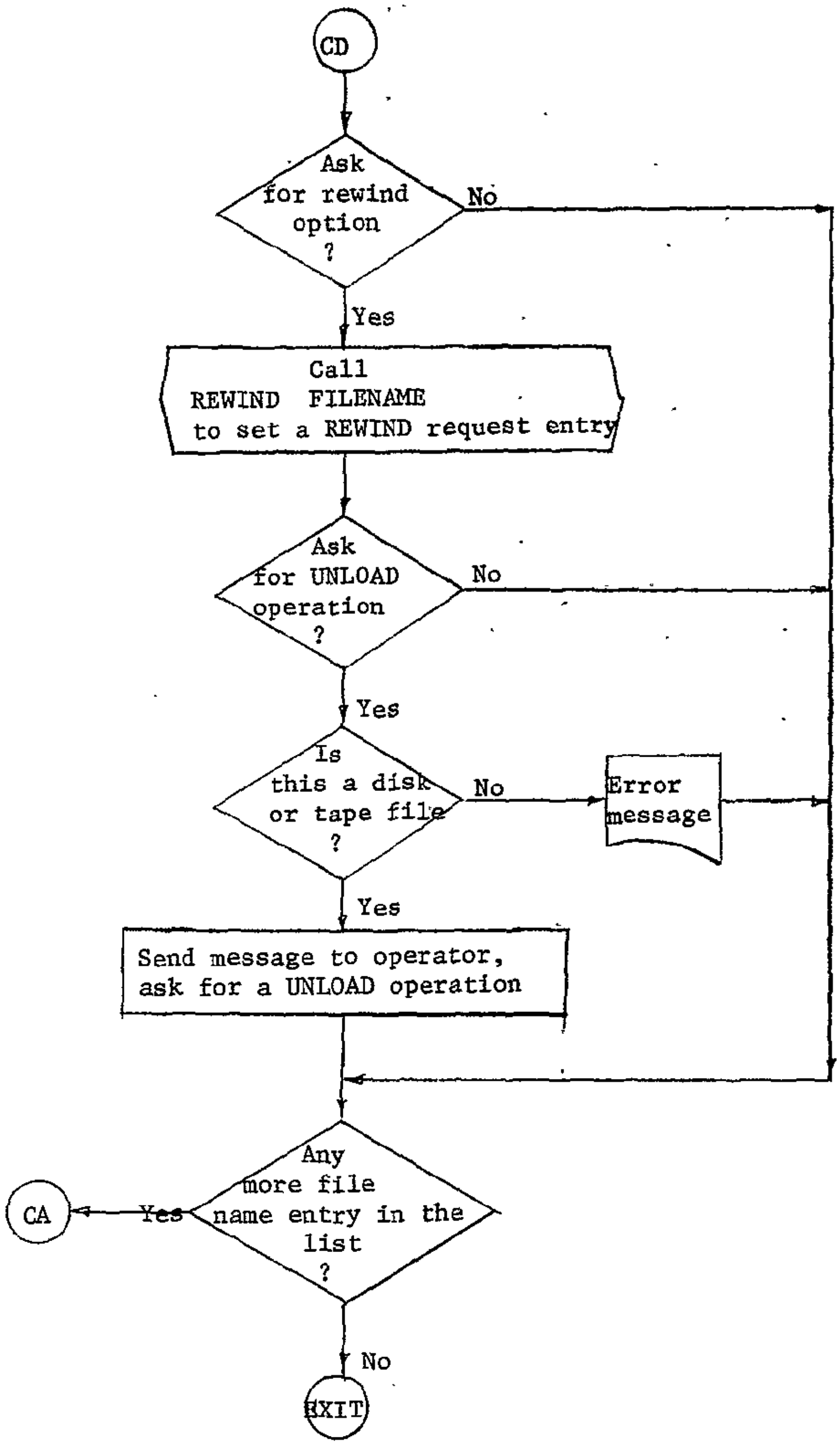


Fig. 30 The Flow Chart of-Close Routine (Cont.)

Calling Sequence: The CLOSE routine is called by the CLOSE macro instruction. The format of CLOSE macro instruction is shown in Section 3.2.1. Note that a list of files may closed by a single CLOSE macro instruction.

Inquts: The inputs to the CLOSE routine are:

1. The parameters of the CLOSE macro instruction. They are: A list of files to be closed, and options(REWIND, or UNLOAD).
2. The FCB's for each file in the list.
3. The Available-Buffer-Chain Entry Table and the available buffer chains.

Outputs: The outputs from this routine are:

1. Clear the FCB's. All FCB's of files in list are cleared and contain no information except the file names.
2. An end-of-file mark at the end of each OUT type of files in list.
3. Perform the REWIND operation for every file in list, if the REWIND option is specified.
4. Dismount tape message. A dismount message will sent to operator console, if the UNLOAD option is specified.

Algorithm and Flow chart: As shown in Figure 30.

(c) The REDEF routine

Purpose: Switch the type of the file

Major Objectives:

1. Change the file type information in FCB to the type declared in the REDEF macro instruction.
2. Do the necessary modification as shown in Table 2.
3. Perform the REWIND operation, if REWIND option exists.

new / old	IN	OUT	NONBUF
IN	No operation (if file still open)	Backspace N records, where if BUSY = 0, then $N = \frac{\text{Buffer size}}{\text{Physical record size}}$ Otherwise $N = 2 * \frac{\text{Buffer size}}{\text{Physical record size}}$	1. Release the buffers used by the file and return them to the ABC 2. Clear all buffer informations in FCB
OUT	1. Sent out all the informations that remains in the buffers 2. Rewind tape 3. Reset the AVBCT information in FCB	No operation (if file still open)	1. Sent out all the informations that remains in the buffers. 2. Release the buffers used by the file and return them to the ABC 3. Clear all buffer informations in FCB.
NONBUF	1. Allocate two buffers with proper size and store these two address into FCB. 2. Store the buffer informations into FCB 3. Initiate a READ operation, if the new type of file is IN.		No operation (if file still open)

Table 2. The Actions of the REDEF macro instruction

	REDEF	FILENAME, TYPE, (REWIND)

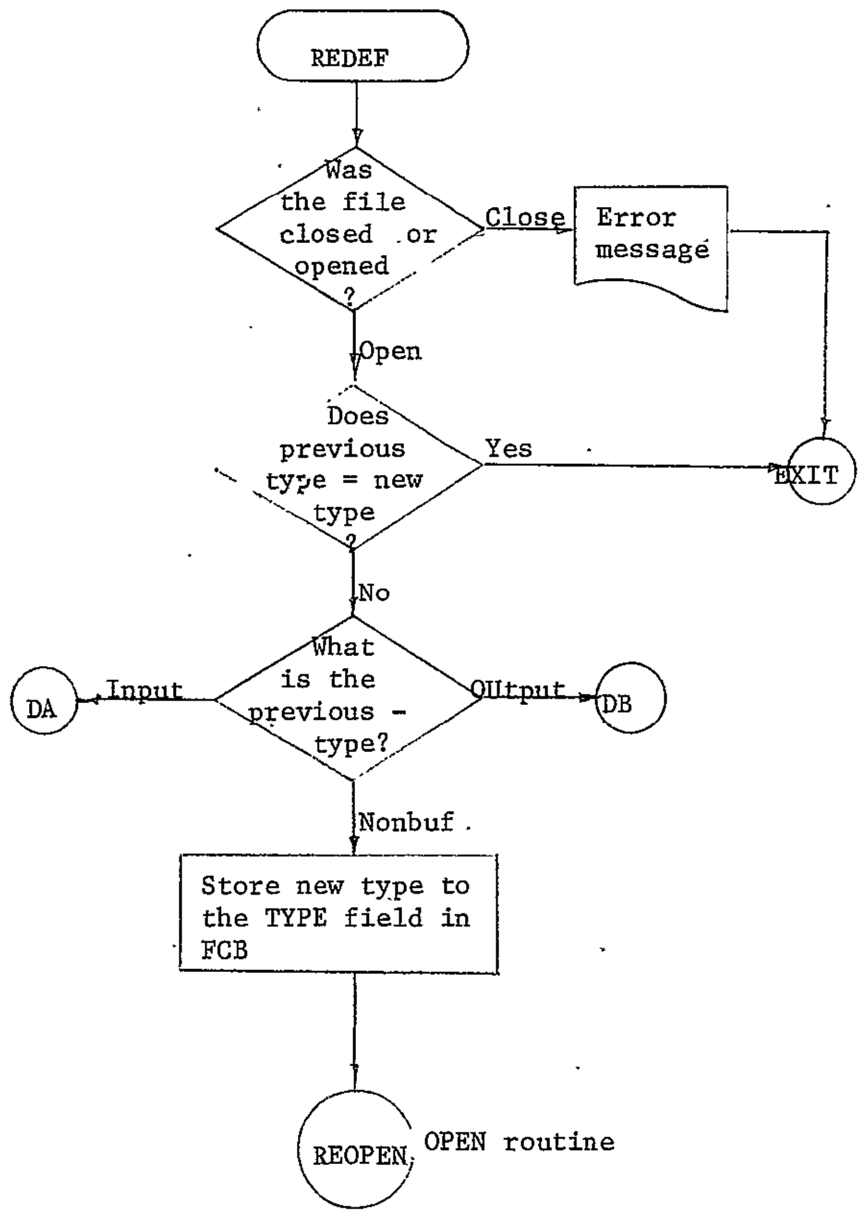


Fig. 31 The Flow Chart of REDEF Routine

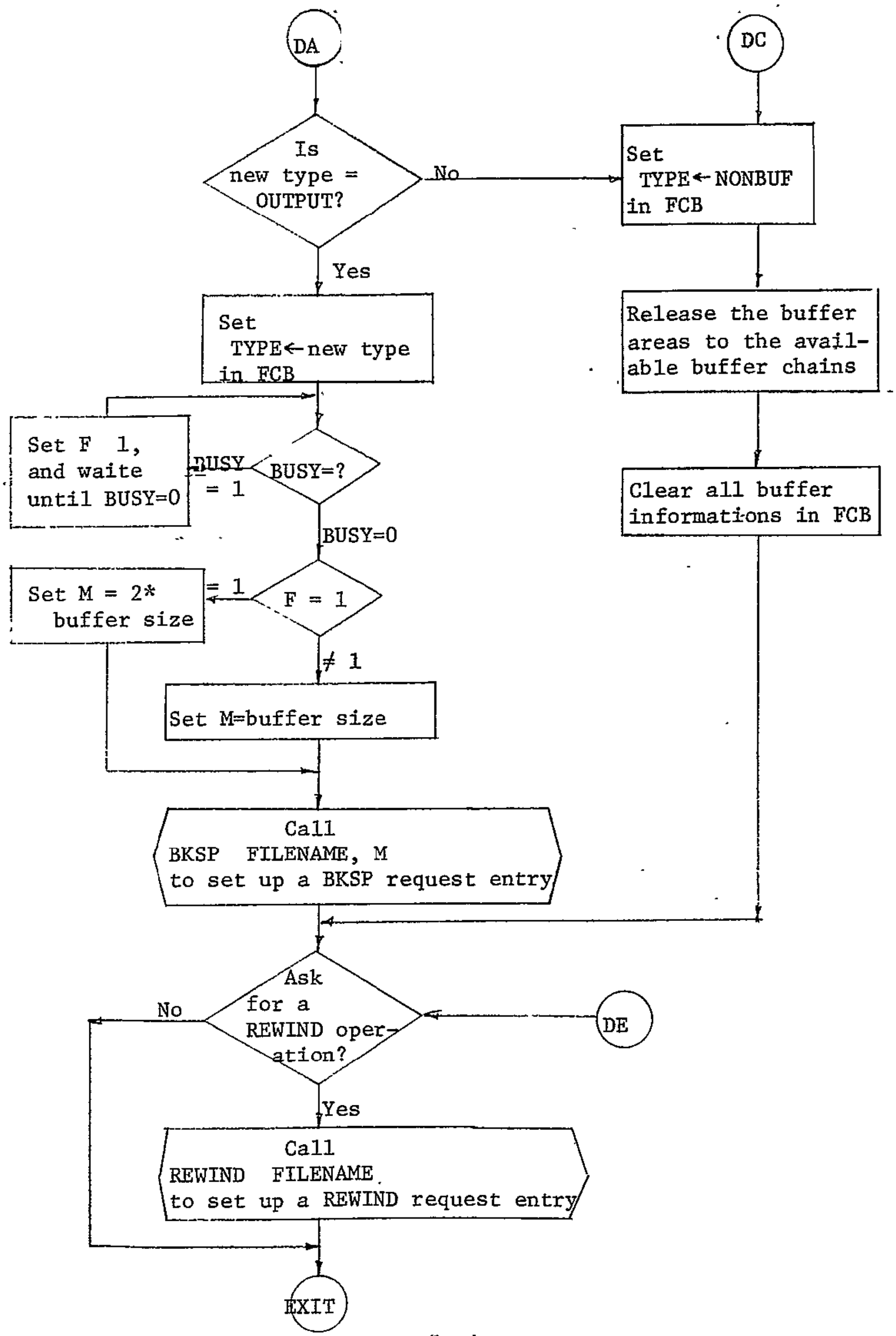


Fig. 31 The Flow Chart of REDEF Routine (Cont.)

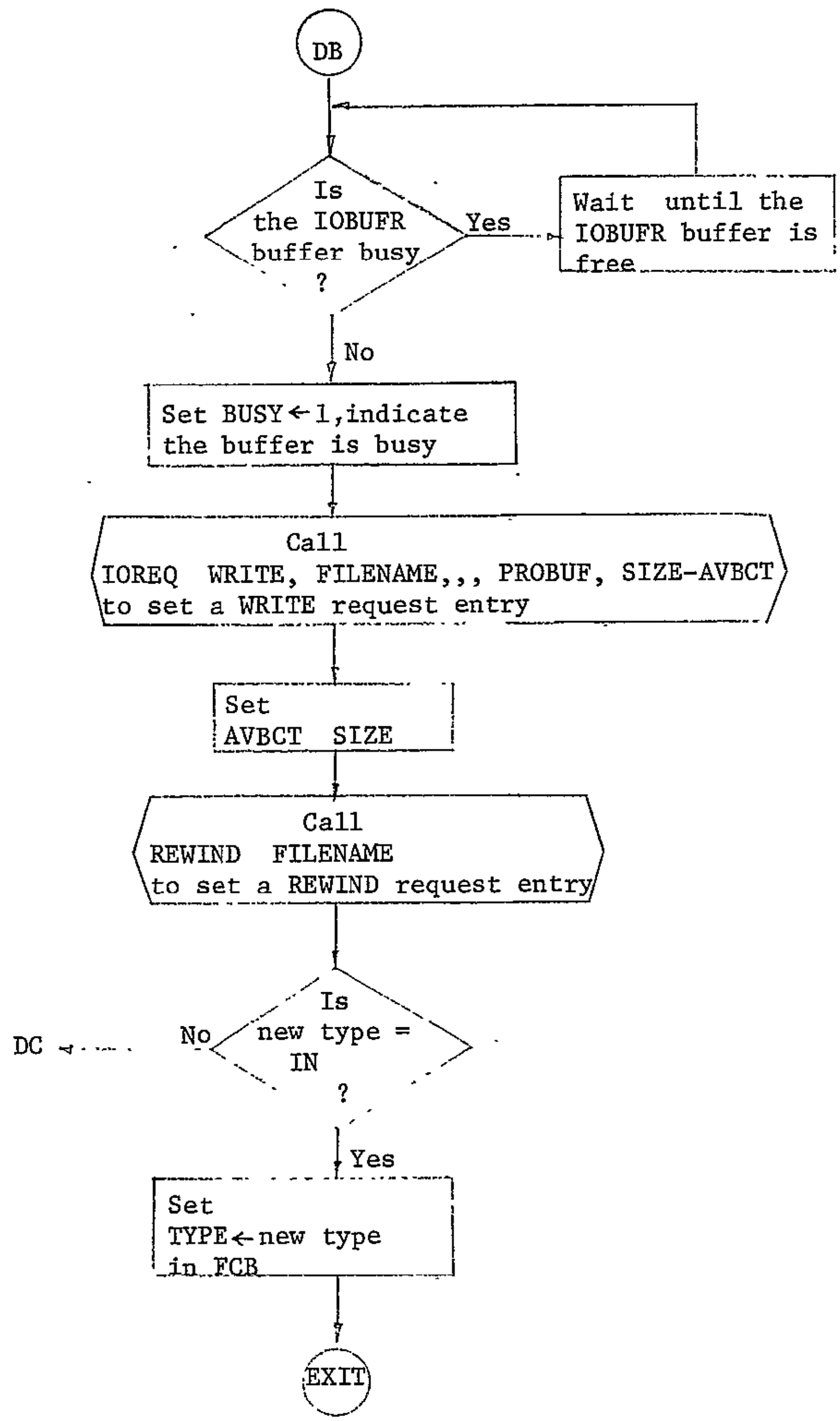


Fig. 31 The Flow Chart of REDEF (Cont.)

Note that, in Table 2 when the file changes its status from type IN to type OUT, a backspace operation is performed. This allows the user to switch input mode into output mode at any point of his file,

Calling Sequence: The REDEF routine is called by the REDEF macro instruction. The format of REDEF macro instruction is shown in Section 3.2.1.

Inputs: The inputs to the REDEF routine are:

1. The parameters of the REDEF macro instruction. They are: FILENAME of the file to be changed,
   TYPE to be changed,
   REWIND tape option.

2. The FCB of the file.

3. The Available-Buffer-Chain Entry Table and the abailable-buffer chains.

Outputs: The outputs from the REDEF routine are:

1. The FCB of the file--FCB contains the information about the present status of the file which has been redefined.

2. The Abailable-Buffer Chain Entry Table and the available-buffer chains--they are changed according to the buffer allocation or freed by REDEF routine.

Algorithm and Flow chart: As shown in Figure 31.

5.2.1.2 The I/O request routines

(a) The READ routine

Purpose: To transfer data into user's working area

Major Objectives: 1. Detect the error, if the file is closed or if it is a type OUT file

2. Transfer the requested amount of data from PROBUFR buffer into the user's specified working area (1^{st} ADDR.+N-1).

3. Initiate the read ahead operation to read in data from input device at proper time (i.e. when AVBCT CRTCL).

4. Switch the pointer IOBUF, with pointer PROBUF in FCB, whenever PROBUFR buffer is empty and IOBUFR buffer is full.

5. Adjust the buffer informations in FCB, if necessary.

6. Pass control to the I/O scheduling routines for requesting data directly input from input device into user's area, if the file is a type NONBUF file.

Calling Sequenct: The READ routine is called by the READ macro instruction. The format of the READ macro instruction is present in section 3.2.1

Inputs: The inputs to the READ routine are:

1. The parameters of the READ macro instruction

FILENAME:	The name of the desired file.
ERR:	The address of the user's error routine.
INTRUP:	The address of user's interrupt routine.
EOF:	The address of the user's end-of-file check routine.
1^{st} ADDR:	The first location in which the requested data are to stored.
N:	The number of words required by this macro instruction

Note that, if any of the ERR, INTRUP, EOF parameters

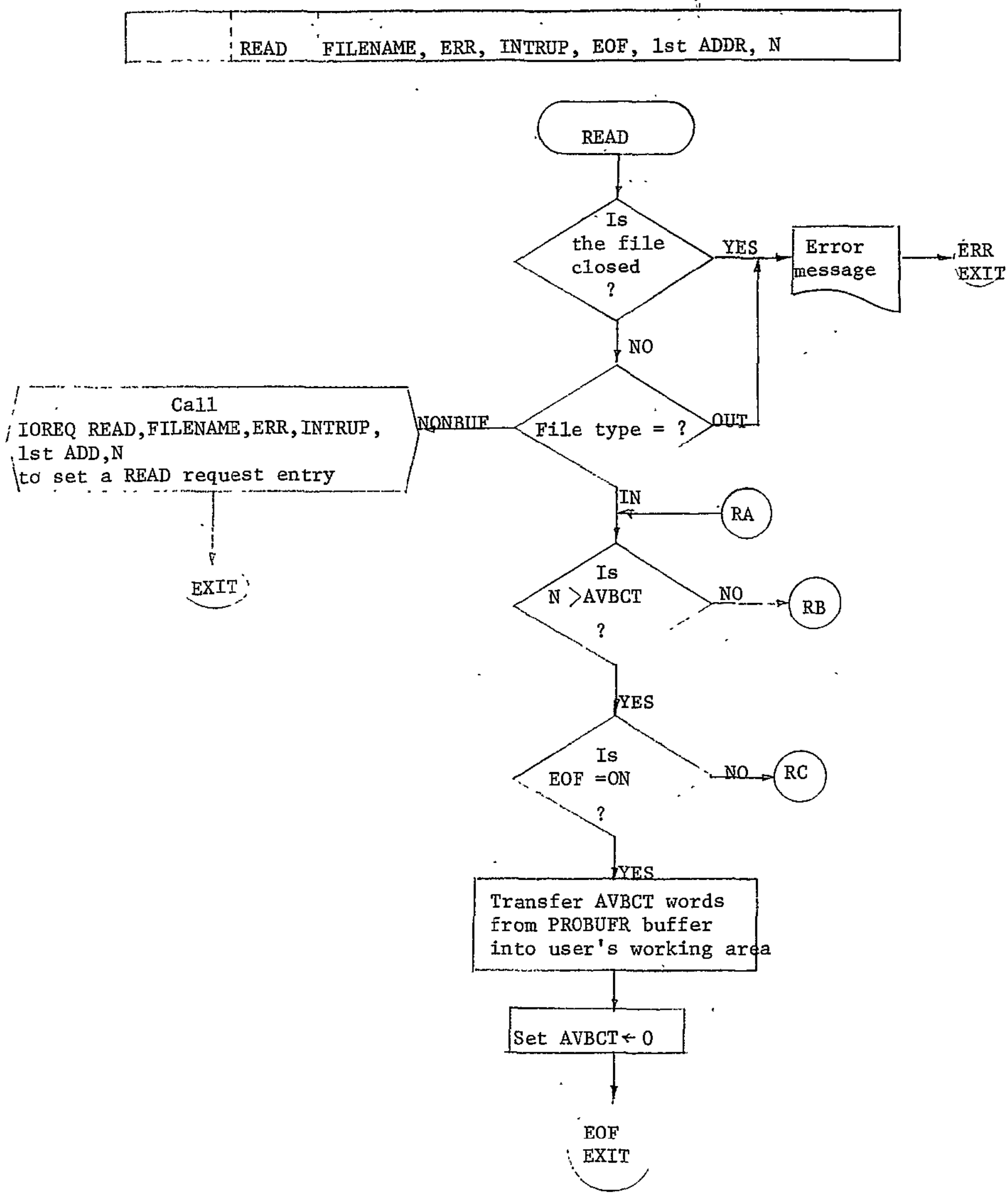


Fig. 32 The Flow Chart of READ Routine

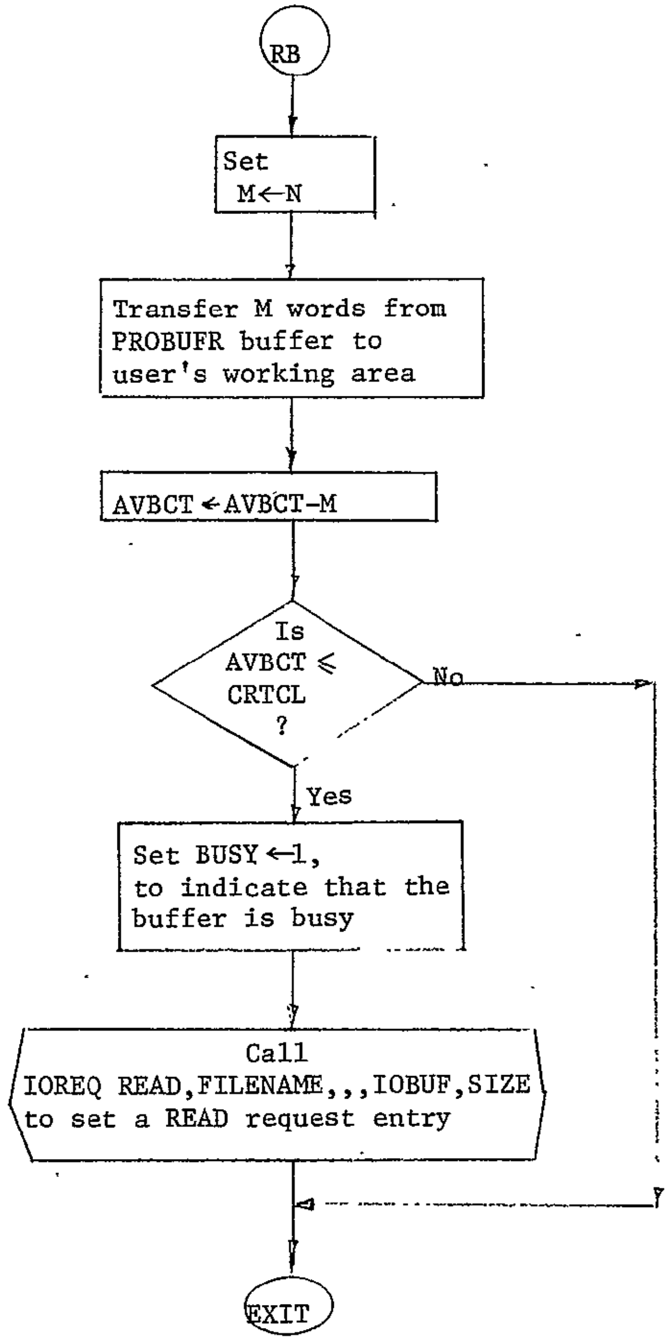


Fig. 32 The Flow Chart of READ Routine (Cont.)

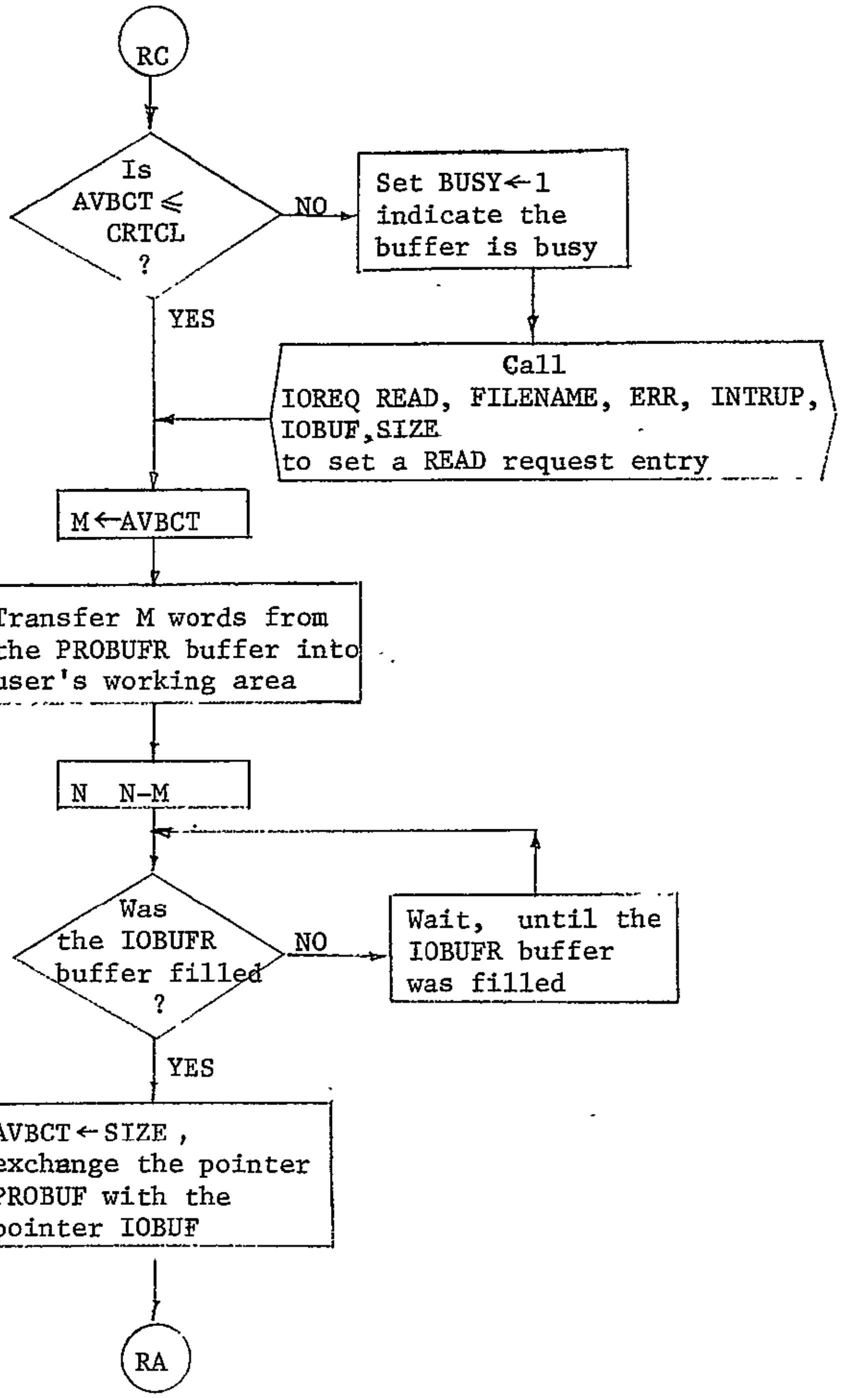


Fig. 32 The Flow Chart of READ Routine (Cont.)

is absent, then the address of system error check routine, or system end-of-file check routine will be used, respectively.

Outputs: The outputs from the READ routine are:

1. The requested data--they are transfered into the users working area.
2. The FCB of the file--the information in FCB is changed according to the present status of the file.
3. An error message--an error message will print out if the file is closed or it is an OUT file.

Algorithm and Flow chart: As shown in Figure 32.

(b) The WRITE routine

Purpose: To transfer data out of the user's working area.

Major Objectives:

1. Detect the error condition, when the file is closed or it is an IN file.
2. Pass control to the I/O scheduling routines for sending the information directly out from user's area to output device, if the file is a NONBUF file.
3. Transfer data from user's working area (i.e. location 1^{st} ADDR. through location 1^{st} ADDR. +N-1) to the PROBUFR buffer.
4. Switch the pointer IOBUF with pointer PROBUF in FCB, whenever PROBUFR buffer is full and IOBUFR buffer is empty.
5. Initiate an output operation to empty out the IOBUFR buffer. That is transfer control to I/O

scheduling routine to request an output operation from IOBUFR buffer to proper output device.

Calling Sequence: The WRITE routine is called by the WRITE macro instruction. The format of this macro instruction is shown in section 3.2.1.

Inputs: The inputs to the WRITE routine are:

1. The parameters of the WRITE macro instruction. These are,

   FILENAME: The name of the desired file to be written out.

   ERR: The address of user's error check routine.

   Note: If this field is a blank, then the address of system error check replaces it. That means the error return from this routine will be sent to system error check routine.

   INTRUP: The address of user's interrupt routine.

   Note: If this field is absent, then the address of the system intrrupt routine is used.

2. The FCB of the desired file.

Outputs: The outputs from this routine are:

1. The requested output data--these output data are now in the buffer area.

2. The FCB of the file--the contents of FCB are changed according to the status of the file.

	WRITE	FILENAME,ERR,INTRUP,1st WD ADD.,N

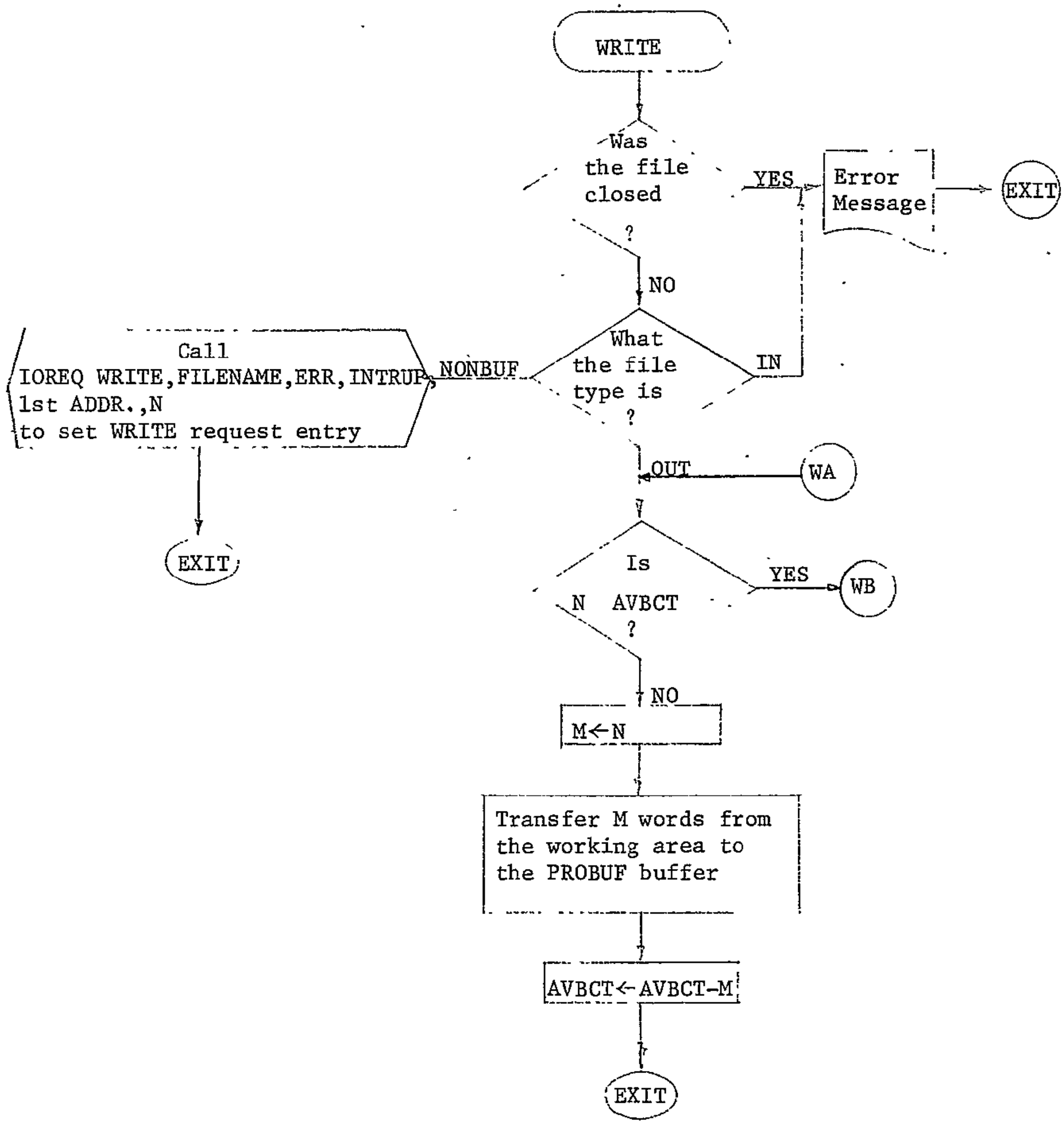


Fig. 33 The Flow Chart of WRITE Routine

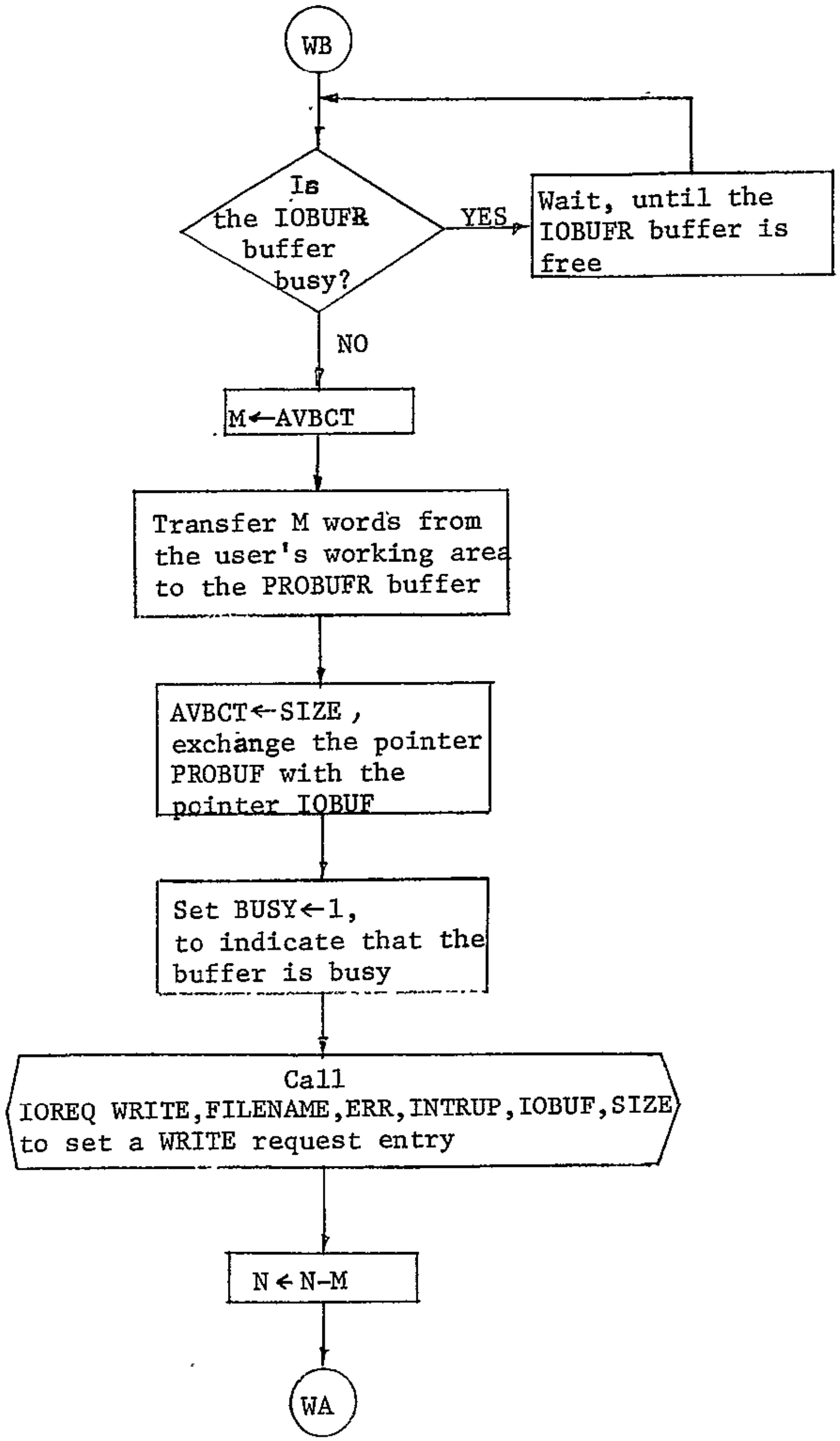


Fig. 33 The Flow Chart of WRITE Routine (Cont.)

3. An error message--if the file is closed or if it is a type IN file then an error message will sent out indicate the error condition.

Algorithm and Flow chart: As shown in Figure 33.

(c) The WEOF routine

Purpose: Write an end-of-file mark at the end of the file.

Major Objectives:

1. Detect the error condition. Send out an error message, if the file is closed or it is a type IN file.

2. Initiate a write end-of-file mark operation. That is, pass control to I/O scheduling routines to set up write end-of-file request.

Calling Sequence: This WEOF routine is called by the WEOF macro instruction. The format of the WEOF macro instruction is shown in section 3.2.1.

Inputs: The inputs to this WEOF routine are:

1. The parameter of the WEOF macro instruction--FILENAME of the file.

2. The FCB of the file specified by FILENAME.

Output: The output from this routine is an end-of-file mark delimiting the end of the file.

Algorithm and flow chart: As shown in Figure 34.

(d) The MOVE routine

Purpose: Move forward and pass end-of-file markers.

Major Objectives:

1. Check and make sure that the file is not closed.

2. Send out all the information that remains in the buffers together with an end-of-file mark, if this is an output file.

3. Set up and initiate an I/O request.

	WEOF	FILENAME

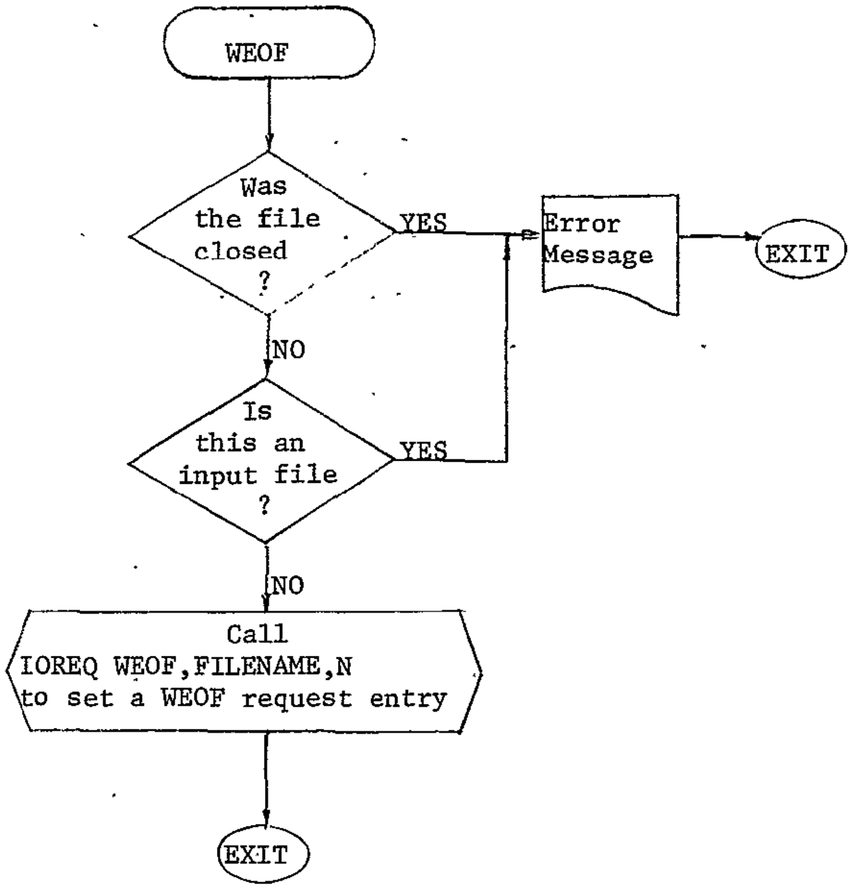


Fig. 34 The Flow Chart of WEOF Routine

4. Set up and read I/O request, if this is an input file.

5. Clear AVBCT information in FCB.

Calling Sequence: This routine is called by the MOVE macro instruction. The format of the MOVE macro instruction is shown in Section 3.2.1

Inputs: The inputs to this routine are:

1. The parameters of the MOVE macro instruction are the FILENAME and N.

2. The FCB of the file.

Output: If the file is already closed, then an error message will send out from this routine.

Algorithm and Flow chart: As shown in Figure 35.

(e) The BKSP routine:

Purpose: Move N physical records backward.

Major Objectives:

1. If an error condition is detected, sent out an error message. If the file is closed, this is an error.

2. If the length of N physical record is greater than or equal to buffer size then:

   (a) Set up a request for backspace N, records, where N_1 is the smallest integer such that N=N-(SIZE-AVBCT) /physize and N_1 is multiplier of (size/physize),

   (b) Set up a READ request, and

   (c) Adjust AVBCT.

3. If the length of N physical record is less than buffer size then adjust AVBCT only.

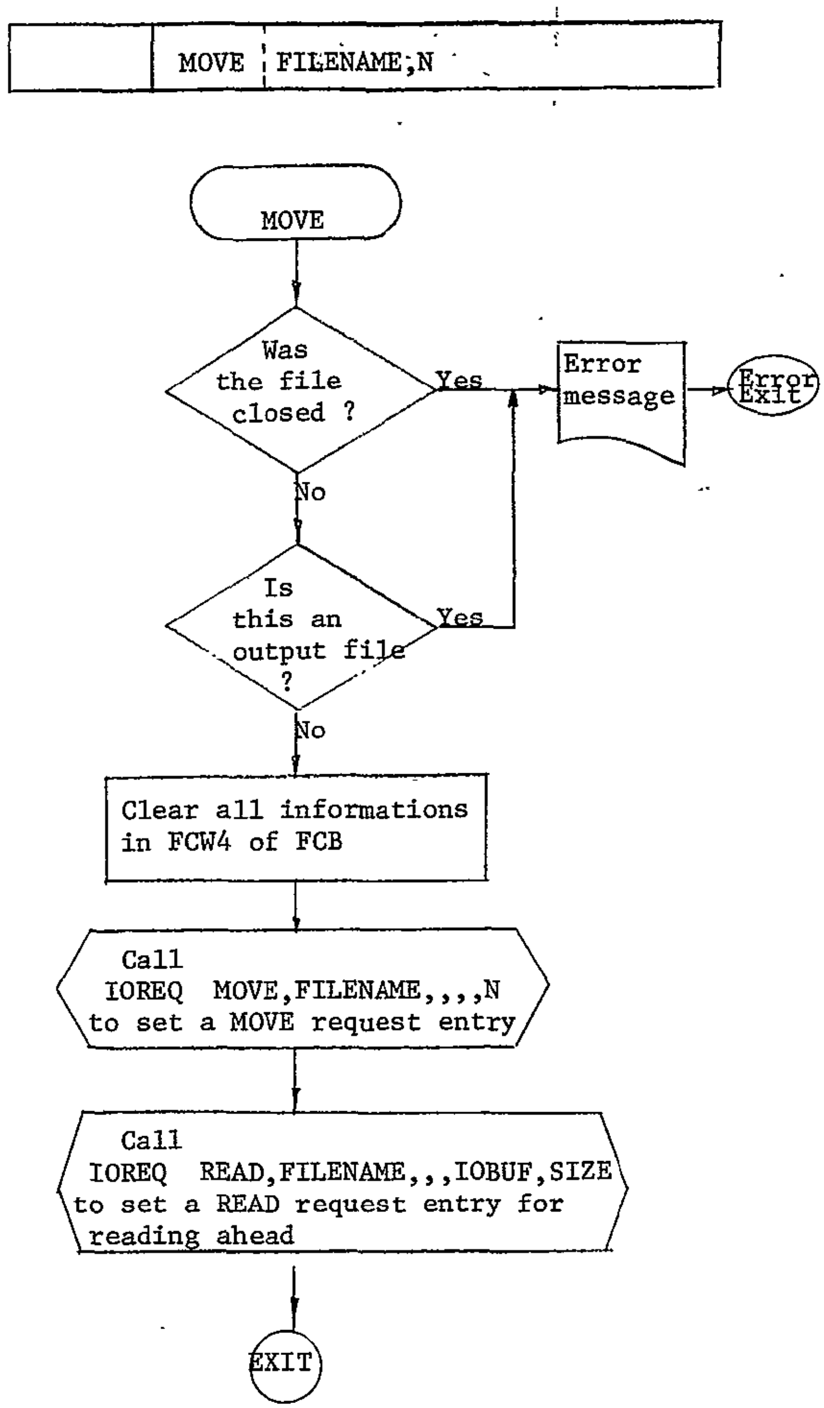


Fig. 35 The Flow Chart of the MOVE Routine

Calling Sequence: This routine is called by the BKSP macro instruction. The format of the BKSP macro instruction is shown in the flow chart (Figure 36).

Inputs: The inputs to this routine are:

1. The parameters of BKSP macro instruction are the FILENAME and N.
2. The FCB of the file.

Outputs: An error message will be sent out if the file is closed.

Algorithm and Flow chart: is shown in Figure 36.

(f) The REWIND routine

Purpose: Perform the rewind operation.

Major Objectives:

1. Check and make sure that the file is not closed.
2. Write out all the data remaining in the buffers and write out an end-of-file mark at the end, if this is an output file.
3. Set up a REWIND I/O request, if this is an input file.

Calling Sequence: This routine is called by the REWIND macro instruction. The format of the REWIND macro instruction is shown in the flow chart (Figure 37)

Inputs: The inputs to this routine are:

1. The first parameter of the REWIND macro instruction is the FILENAME.
2. The FCB of the file.

Output: An error message will be sent out if the file is closed.

Algorithm and Flow chart: As shown in Figure 37.

	BKSP	FILENAME,N

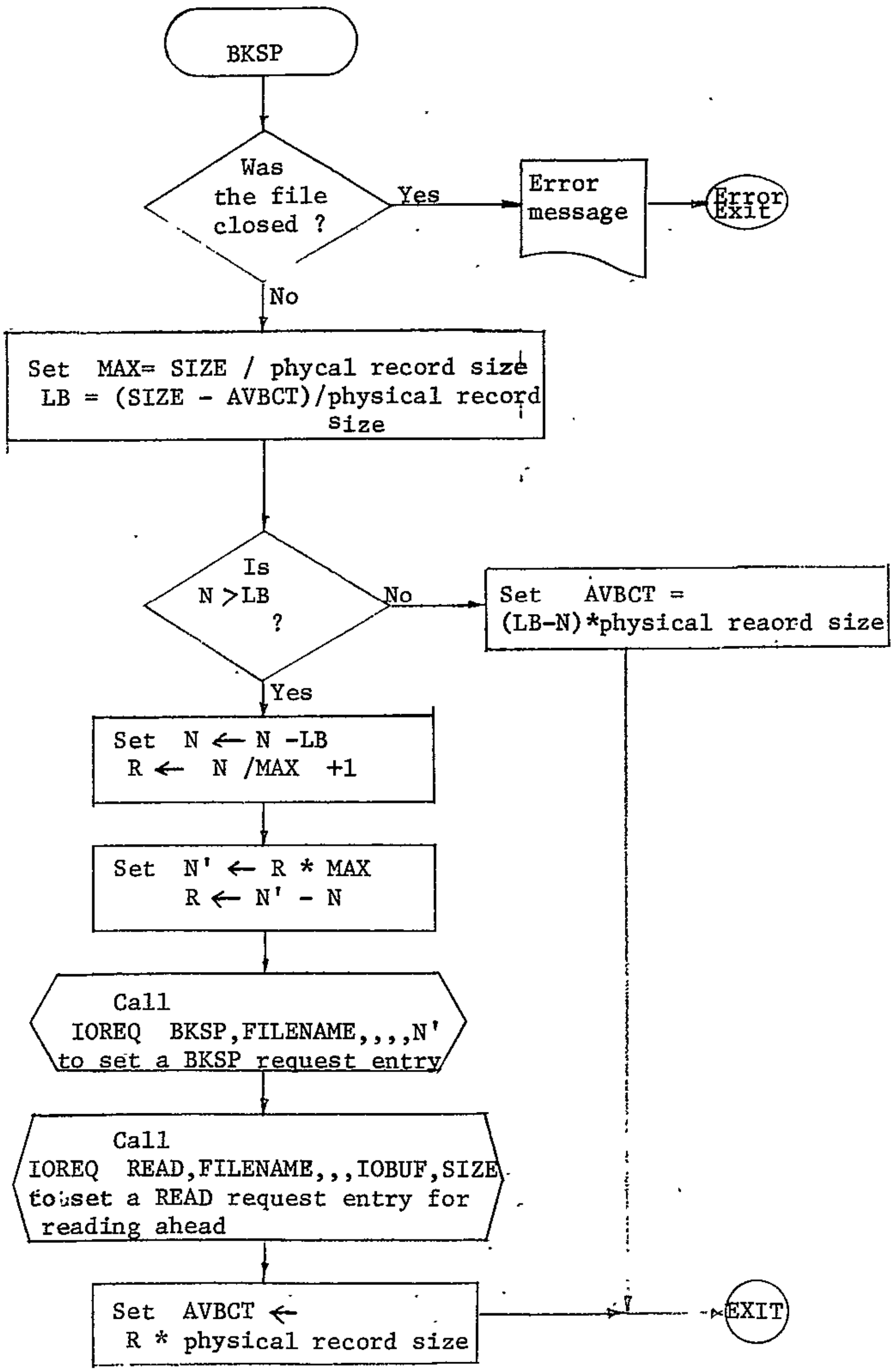


Fig. 36 The Flow Chart of the BKSP Routine

	REWIND	FILENAME

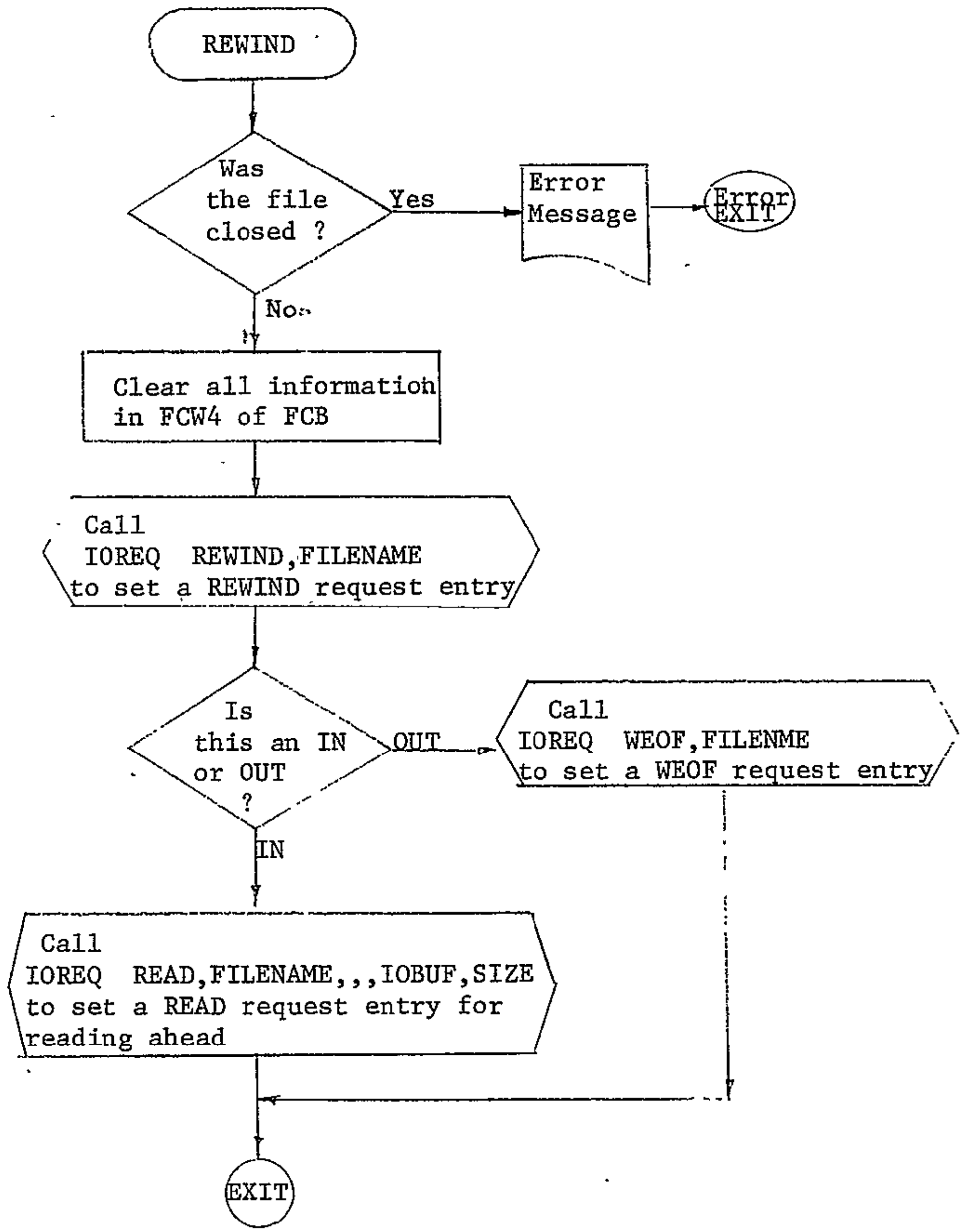


Fig. 37. The Flow Chart of the REWIND Routine

5.2.2 The I/O scheduling routines

The I/O scheduling routines can be divided into two groups, namely, I/O initiation and I/O completion. The I/O initiation group consists of three routines: IOREQU, STARIO, INITIO. The I/O completion group consists of the IOINPR routine, IOFIN routine and the RSLANL routine.

5.2.2.1 The I/O initiation routines

(a) The IOREQU routine

Purpose: Request an I/O operation

Major Objectives:
1. Check and determine whether the device can accept the request.
2. Check and determine whether the unit is busy.
3. Call the STARIO routine, if the unit is ready to accept this function. Otherwise, insert the I/O request entry into I/O request queue of the proper unit.

Calling Sequence: The IOREQU routine is called by the IOREQU instruction. The format of this IOREQU macro instruction is shown in Figure 38.

Inputs: The inputs to the IOREQU routine are:

1. The parameters of the IOREQU macro instruction. They are: I/O request function--the name of the function.

FILENAME--The name of the file

INTRUP --The address of the user's interrupt routine.

Note: If this field is absent, the address of system interrupt routine will be used.

ERR --The address of the user's error check routine

Note: If this field is blank, then the address of system error check routine will be supplied.

1st ADDR.-The first location where data will be read or written.

N --The number of words to be read into or written from memory.

Note: That N exists only when the function is a data request (e.g. READ, WRITE)

2. The FCB of the file specified by the parameter FILENAME.

3. The UCB of the unit which is used by the file.

4. I/O request table and I/O request queue (See Section 6)

5. The function acception table (See Section 6)

Outputs: The outputs from the IOREQU routine are:

1. The I/O request is inserted into an I/O request queue, if that request can not be initiated right away.

2. An error message--if the device does not accept the request function.

Algorithm and Flow chart is shown in Figure 38.

(b) The STARIO routine--

Purpose: Prepare the CCB for initiate an I/O request

Major Objectives:

1. Fill in all the information in the CCB

2. Call the INITIO routine.

Calling Sequence: This routine is called by the IOREQU routine, address of I/O request entry must be in the index register before entering this routine.

	IOREQ FUNCTION,FILENAME,INTRUP,ERR,1st ADDR,N

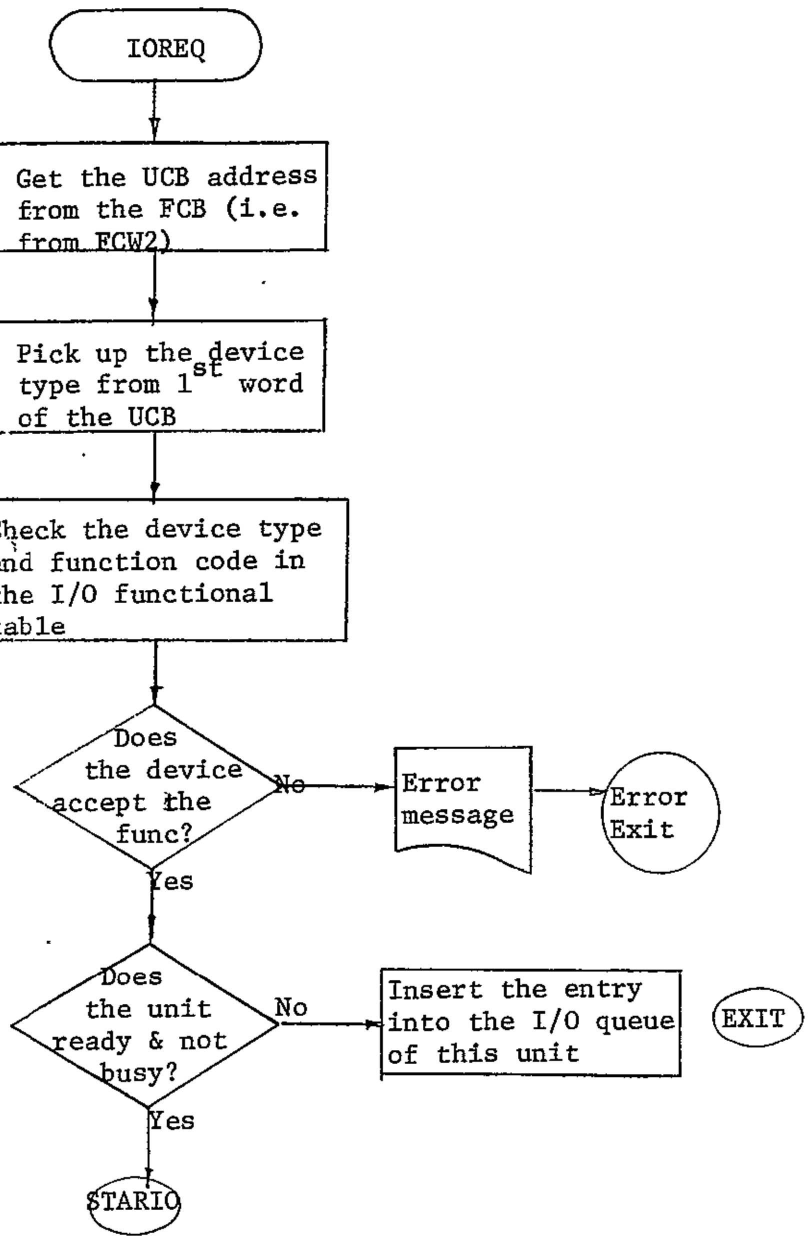


Fig. 38 The Flow Chart of IOREQ Routine

Inputs: The inputs to this routine are:

1. The I/O request entry--its address is specified by the register
2. The UCB of the unit required by this I/O request.
3. The CCB of the channel required by this I/O request.

Outputs: The output of this routine is a CCB with new information in it.

Algorithm and Flow chart: As shown in Figure 39.

(c) The INITIO routine--

Purpose: Initiate an I/O request

Major Objectives:
1. Check and determine whether the request is a data request. Transfer control to unit interpretion routines, if it is a non-data request.
2. Check and protect the system protectional unit.
3. Transfer control to proper unit interpretive routine.

Calling Sequence: This routine is called by the STARIO routine, address of I/O request entry must be stored in the index register before entering this routine.

Inputs: The inputs to this routine are:

1. The I/O request entry--the address of this entry is in the index register
2. The UCB of the unit requested by this I/O request entry.

Output: The output from this routine is an error message--if the I/O request attempt to harm the system protection unit.

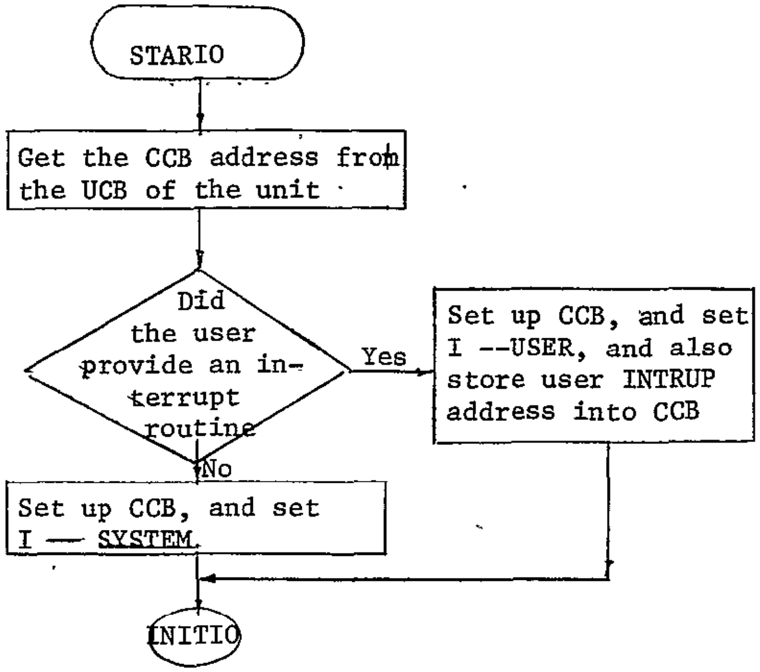


Fig. 39 The Flow Chart of the STARIO Routine

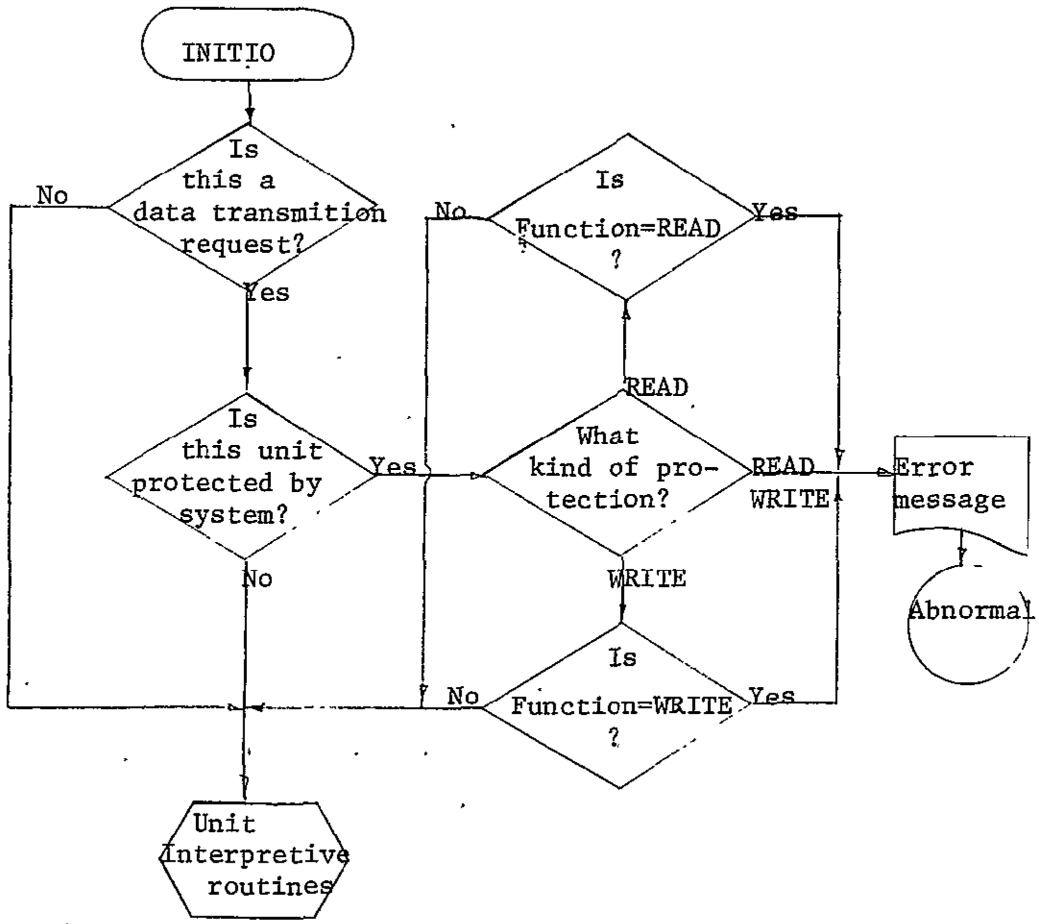


Fig. 40 The Flow Chart of the INITIO Routine

Algorithm and Flow chart: is shown in Figure 40.

5.2.2.2 The I/O completion routines

(a) The IOINRP routine

Purpose: Process the I/O interrupt

Major Objectives:

1. Disable or inhibit other occurence of an interrupt.
2. Save the contents of the program location counter and of all necessary registers.
3. Identify the interrupted unit and channel.
4. Clear the interrupt line.
5. Record the I/O result descriptor (See Section 6.1.2 for the details and description of the I/O status descriptor).
6. Transfer control to the IOFIN routine.

Calling Sequence: This routine is called when an I/O interrupt has occurred.

Inputs: The input to this routine is the I/O interrupt signal.

Outputs: The outputs from this routine are:

1. The I/O result descriptor, which is recorded and stored in some fixed location.
2. The contents of the location counter and of necessary registers, these are saved in predetermined locations.

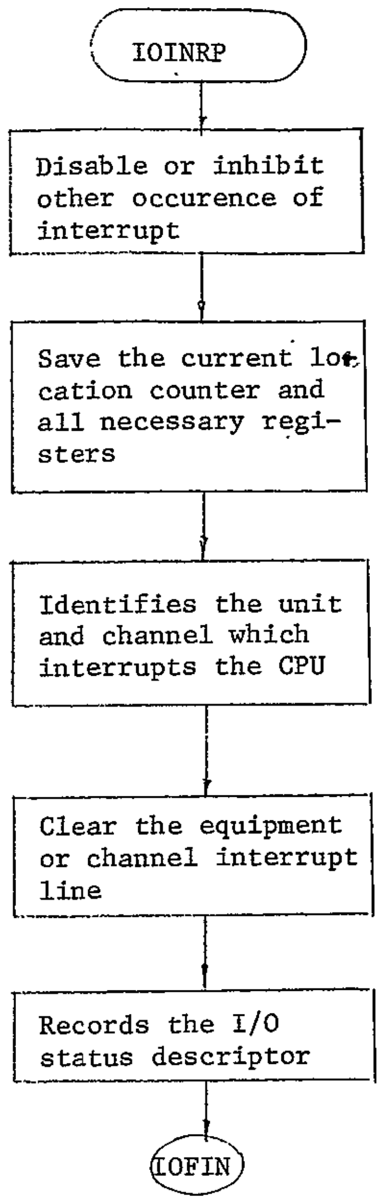
Algorithm and Flow chart: As shown in Figure 41.

(b) The result analysis routine

Purpose: Analyze the result of an I/O operation

Major Objective:

1. Analyze the error condition indicated by the error field of the I/O result descriptor.



I/O status descriptor

Memory address	Char. count	Unit number	Error field

(see Fig.26)

Fig. 41 The Flow Chart of the IOINRP Routine

2. Turn on the EOF indicator in FCB if the end-of-file bit is 1 in the I/O status descriptor.

3. Reset the busy flag to indicate that the buffer is not busy now.

4. If the user's interrupt address is not specified, enable the interrupt, restore the contents of location counter and of all saved registers, and then return control to the calling program.

5. If the user's interrupt routine is specified, store the I/O result descriptor into the first word of the user's interrupt routine, reset all contents of the saved registers, and then transfer control to user's interrupt routine.

Calling Sequence: This routine is called by the IOFIN routine, the address of the I/O status descritor must be stored in the index register before enterring this routine.

Inputs: The inputs to this routine are:

1. The I/O status descriptor, whose address is stored in the index register
2. The CCB, the UCB and the FCB which are resident in core at all times.

Outputs: The outputs from this routine are:

1. If the end-of-file condition is detected,1 in the EOF indicator of the FCB.
2. 0 in the BUSY indicator of the FCB.
3. The I/O status descriptor in the first word of the user's interrupt routine.

Algorithm and Flow chart: As shown in Figure 42.

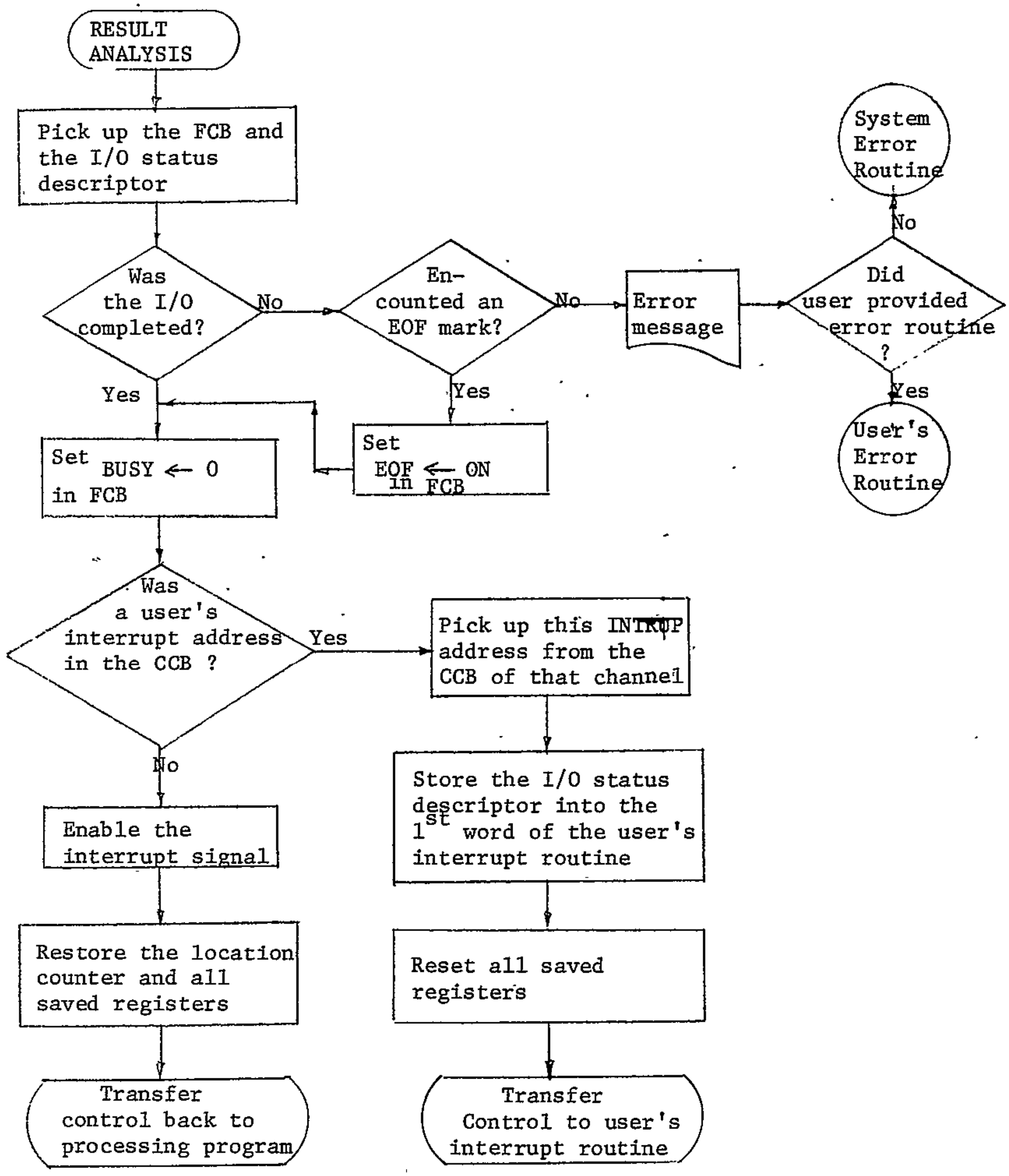


Fig. 42 The Flow Chart of the Result Analysis Routine

(c) The IOFIN routine

Purpose: Update the CCB and UCB

Major Objectives:

1. Initiate the next I/O request in the I/O request-queue for that particular unit which interrupts the processing, if there is an I/O request in that queue.
2. Update the CCB and UCB of the channel and unit which interrupts the processing, if there is no I/O request in that queue.
3. Pass control to the result analysis routine.

Calling Sequence: This routine is called by the IOINRP routine. Address of the I/O result descriptor must be stored in the index register before enterring this routine.

Inputs: The inputs to this routine are:

1. The address of the I/O request-queue entry table. This address is a known parameter.
2. The I/O request-queues, these queues are reside in the core memory at all time.
3. The I/O result descriptor, whose address is stored in the index register X.

Outputs: The outputs from this routine are:

1. If the I/O queue is empty then CCB and UCB are updated.
2. If the I/O queue is not empty then an I/O entry is picked up from I/O queue.

Algorithm and Flow chart: As shown in Figure 43.

5.2.3 The unit interpretive routine

Purpose: Set up the channel program and initiate the proper action.

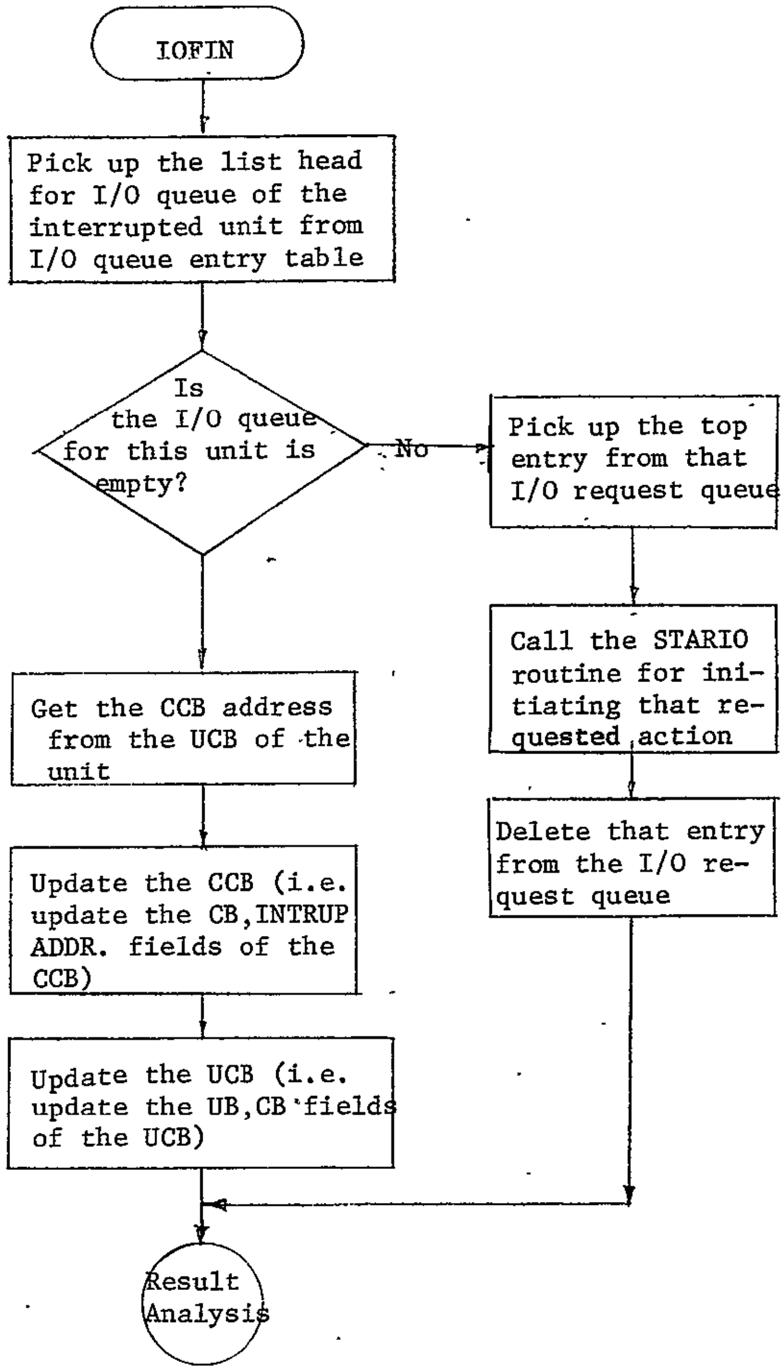


Fig. 43 The Flow Chart of the IOFIN Routine

Major Objectives:	1. Initialization for processing upon re-entry.
	2. Set up the I/O instruction codes.
	3. Set up the channel programs.
	4. Issue the I/O actions.
	5. Return control to the user's program.
Calling Sequence:	This routine is called by INITIO routines.
Input:	The input to this routine is the I/O request entry whose address is in the index register X.
Output:	An error message will send out, if the issuing of the I/O instruction has been rejected K times by the hardware.

Algorithm and Flow chart: is shown in Figure 44.

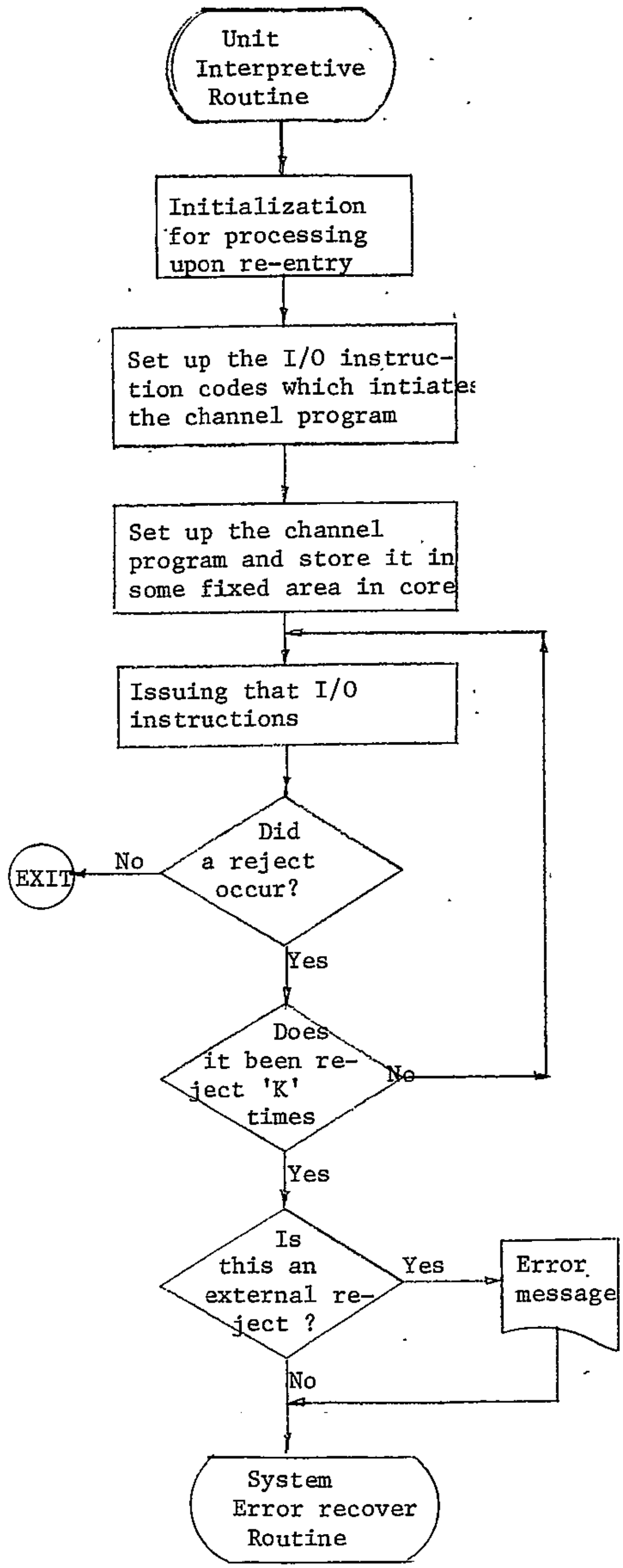


Fig. 44 The Flow Chart of the Unit Interpretive Routine

6. Discussion

This paper has demonstrated how an Input-Output Control System can be simplified and organized as a tree-structured system. The discussions on the designing and expansion of SIOCS are presented first in this section. Then it is followed by the discussion on the microprogramming of SIOCS. The microprogrammed implementation of a portion of SIOCS, the buffer allocation, has been presented in Reference [50], where the illustration of an integrated software-hardware design through microprogramming is given in great detail.

6.1 Discussion on the designing of SIOCS

(a) The tree-structure is regarded as a very important principle for designing an operating system. It is both easy to understand and easy to implement, because each level of the tree has its own goals and its own clear environment. To isolate the levels and to decide upon how many levels are most important in the design. The experience gained in designing this SIOCS indicates that the ideal solution to achieving program modularity is to divide the IOCS into four levels. The highest level (the file system) is accessed directly by the user, and only the lowest level (the unit interpretive routines) is dependent upon the hardware. The middle two levels (the buffering system and the I/O scheduling) are accessed only by the system programmer. In this manner, the system programmer may change part of the I/O scheduling for a special hardware configuration at a later time. Similarly, the system programmer may change a part of the buffering system at will in order to accommodate some special user need.

(b) Tables should be used by the IOCS to communicate within different parts of the operating system, while explicit software should be created to communicate to the outside. This choice is because the environment within the system is relatively static while the environment outside the system is always changing.

6.2 Discussion on the Expansion of SIOCS

(a) A channel scheduler should be added into I/O scheduler--the SIOCS contains a Channel Control Block (CCB) for every channel, and assumes that each unit is connected with one channel at all times. One may add a channel scheduler which allows several I/O devices to share the same channel.

(b) A disk and drum I/O capability should be added for disk and drum operations. Such information to enable an order such as seek address to be implemented must be maintained in Unit Control Block for disk or drum operations.

(c) Internal files should be added into the file system--one may introduce a fourth type of file, namely internal file, which is a list of buffers together with pointers. The third word of present File Control Block (section 4.4) may be used as a list head of an internal file. With this feature, a user may declare a particular file which is to be referenced very frequently as an internal file. (See References [39], [44], [45])

(d) One may add conversion routines into SIOCS-- This will allow I/O devices to perform the I/O function under several different modes, such as binary mode, BCD mode,...,etc.

6.3 Discussion on the Microprogramming of SIOCS

(a) The computer elements which are required for implementation of the buffer allocation routines are included in most microprogrammed computers. This means that the buffer allocation routines could be indeed microprogrammed.

(b) When the address of next micro-instruction is specified in every micro-instruction, there is a greater flexibility in the sharing of common sequences of micro-instructions among different functions. This is due to the fact that branching does not take a separate step and successive micro-instructions may be located anywhere in control memory. Furthermore, if the concepts of paging or segmenting are applied in the control memory, then a branch from page to page, or from segment to segment may be implemented very easily.

(c) In order to refer to an operand and to store temporary results, a LOCAL STORE consisting of high speed registers is required. A part of this store may be designed as a stack. This may be used for storing the micro-subroutine return address for re-entry. A stack is most useful for a linguistics processor or for any multiple buffering scheme.

(d) The basic implementation of operating system involves such queuing techniques for control block handling, table reference, internal sorting, pointer handling,etc. It is found from this study that those queuing techniques require some macro operations such as,

* Buffer allocation or general storage allocation,
* Storage release operation,
* Insertion of an item into a chain or list (this may be any type of linkage),
* Delete an item from a chain or list,
* Transfer a block of data from one area into another area within the same storage,
* Sequential search and locate an item,
* Random search and locate an item.

As demonstrated in Reference [50], the buffer allocation routine needs only 6 control words to implement the entire operation. Thus it may be worth while to add the above mentioned elementary operations into the machine language level of such microprogrammed computers as the IBM 360 family or the RCA Spectra 70.

Acknowledgement

The author wishes to express his deepest appreciation to his advisor Mr. Martin Milgram for his inspiring guidance, patience and encouragement during the preparation of this paper.

The author wishes to express special thanks to Professor Yaohan Chu for his inspiration and guidance in this project. The author also wishes to acknowledge and express his thanks to Mr. R. Pardo and Mr. G. Lindamood for their many helpful conversations and sugestions.

Finally, the author is grateful to the Singer-Link Division of Singer Company for their finacial support through Singer-Link Student Scholarship in Computer Science, and to the National Aeronautics Space Administration for their support under the contract NGR 21-002-206.

Bibliography

The bibliography which follows includes subjects related to the computer Input-Output Control System. The bibliography is arranged by subjects, alphabetically by author within each subject.

A. General operating system and I/O Control System

1. Ackerman, W. B. Plummer, W. W.
An implementation of a multiprocessing computer system
ACM 1st Symp. on O. S. Prin. (Oct., 1967)

2. Dijkstra, E. W.
The structure of the 'THE'-multiprogramming system
ACM 1st Symp. on O. S. Prin. (Oct. 1967)

3. Dijkstra, E. W.
Cooperating sequential processes
Technical U. Eindhoven, Netherlands, 1966

4. Flores, I.
Computer Software
Prentice Hall, Inc 1965

5. Flores, I.
Computer Programming
Prentice Hall, Inc. 1966

6. Fuchel, K. Heller, S.
Consideration in the design of a multiple computer system with extended core storage
ACM 1st Symp. on O. S. Prin. (Oct. 1967)

7. Lett, A. S. Kdnigsford, W. L.
TSS/360: A Time-Shareed Operating System
AFIP 1968 FJCC

8. Mealy, G. H.
Operating Systems
Rand report P-2584; or Rosen: Programming Systems and Languages, McGraw-Hill pp.516-559

9. Mealy, G. H.
The System Design Cycle
ACM 2nd Symp. on O. S. Prin. (Oct. pp. 1969) 1-7

10. Needham, R. M. Hartley, D. F.
Theory and practice in operating system design
ACM 2nd Symp. on O. S. Prin. (Oct. 1969) pp. 8-12

11. Poole, P. C. Waite, W. M.
Machine independent software
ACM 2nd Symp. on O. S. Prin. (Oct. pp. 169) 19-24

12. Ramsay, K. Strauss, J. C.
A real time priority scheduler
ACM National Meeting, 1966 pp. 161-166

13. Rosen, S. (Ed.)
IBM operating system/360 concepts and facilities
Programming Systems and Languages, pp.598-646
McGraw-Hill Book Co.

14. Trapnell, E. M.
A systematic approach to the development of system programs
AFIP 1969 SJCC

15. Van Horn, F. C.
Three criteria for designing computing system to facilate debugging
ACM 1st Symp. on O. S. Prin. (Oct. 1967)

16. Wirth, N.
On multiprogramming, machine coding, and computer organization
CACM Vol. 12 No. 9 (Sept. 1969) pp. 489-498

B. Input-Output Control System

17. Allen, T. R. Foote, J. E.
Input/output software capability for a man-machine communication and image processing system
Proc. AFIP 1964 FJCC pp. 387-396

18. Bouman, C. A.
An advanced input-output system for a cobol compiler
CACM Vol. 5 (May 1962) pp. 273-277

19. Bryant, P.
Levels of computer systems'
CACM 9,12 (Dec. 1966) pp. 873-876

20. Cohn, C. E.
Incorporation of non-standard input/output device into FORTRAN systems
CACM Vol. 9 (May 1966) pp. 343-344

21. Digri, V. J. King, J. E.
The share 709 system: Input-Output Translation
JACM Vol. 6, No. 2 (April 1959) pp. 141-144

22. Ferguson, D. E.
Input-Output Buffering and FORTRAN
JACM Vol. 7, No. 1 (Jan. 1961) pp. 1-9

23. Hassitt, A.
Data directed input-output in FORTRAN
CACM Vol. 10 (Jan. 1967) pp. 35-40

24. Mock, O. Swift, C. J.
The share 709 system: programmed input-output buffering
JACM Vol. 6, No. 2 (April 1959) pp. 145-151

25. Ossanna, J. F Mikus, L. E. Dunten, S. D.
Communications and input/output switching in a multiplex computing system
proc. AFIP 1968 FJCC pp. 231-240

26. Patel, R. M
Basic I/O handling on Burroughs B6500
ACM 2nd Symp. on O. S. Prin. (Oct. 1969)

27. Roth, B.
Channel analysis for the IBM 7090
16th ACM National Meeting (Sep. 1961)

28. Smith, R. B.
The BKS system for the PHILCO-2000
CACM Vol. 4 (Feb. 1961) pp. 104-109

29. Statland, N. Hillegass, J.
A survey of input-output equipment
Comput. Autom. 13, 7 (July 1964) pp. 16-20, 28

30. Sutherland, I. E.
Computer inputs and outputs
Scientific American 215,3 (Sep. 1966) pp. 86-109

31. Tasini, B. B. Winograd, S.
Multiple input-output links in computer system
IBM J. Res. Develop. 6,3 (July 1962) pp. 306-328

32. White, P.
Relative effects of centrol processor and input-output speeds upon thronghput on the large computer
CACM 7, 12 (Dec. 1964) pp. 711-714

33. Buffering Between Input-Output and Computer
EJCC 1952 p. 02/022

34. SEAC input-output system
EJCC 1952 p. 02/031

35. RAYDAC input-output system
EJCC 1952 p. 02/070

36. Operational compatibility of systems-conventions
CACM Vol. 4 (Jan. 1961) pp. 266-267

37. The structure of system/360 part IV channel design considerations
IBM System J. Vol. 3-2 (1964) pp. 165-180

C. Reference manuals

38. IBM S. R. L.
IBM 1410/7010 Operating System, Programming Systems Analysis Guide
Form C28-0395-, C28-0396

39. IBM S. R. L.
IBM 709/7090 Input/Output Control System
Form C28-6100-2

40. IBM S. R. L.
Input/Output Control System (on tape) specifications and operating procedures IBM 1401 and 1460
Form C24-1462-2

41. IBM S. R. L.
Input/Output Control System (on disk) specifications and
IBM 1401 and 1460
Form C24-1489-3

42. IBM S. R. L.
IBM System/360 operating system
Supervisor and data management services
Form C28-6646-2

43. IBM S. R. L.
IBM System/360 Operating System
Supervisor and data management macro instructions
form

44. IBM S. R. L.
IBM 7090/7094 Input-Output Control System, programming systems
analysis guide
Form C28-6773

45. IBM S. R. L.
IBM 7090/7094 IBSYS operating system, version 13
System Monitor, Input/Output Control System,
Form C28-6248-7, C28-6345-5,

46. CDC Computer Div.
3200 SCOPE Operating System maintenance documentation
no. M0521.0.

47. CDC Computer Div.
3100, 3200, 3300, 3500 Computer System MSOS reference manual

48. CDC Computer Div.
3100, 3200 computer system SCOPE/Disk SCOPE reference manual

D. Microprogramming and CDL (Computer Design Language)

49. Chu, Y.
A higher-order language for describing microprogrammed
computers
T. R. 68-75 (Sept. 1968) Computer Center, U. of Md.

50. Chu, Y. Pardo, O. R. Yeh, J. W.
A methodology for unified hardware-software design
T. R. 70-107 (Jan. 1970) Computer Center, U. of Md.