BNWL-1440
Special
Distribution

A REAL-TIME NUCLEAR FUELS INVENTORY
AND CRITICALITY CONTROL SYSTEM

V. L. Crow

January 1971

AEC RESEARCH &
DEVELOPMENT REPORT

BNWL-1440

BATTELLE

BATTELLE MEMORIAL INSTITUTE

NORTHWEST

PACIFIC NORTHWEST LABORATORIES

3000 STEVENS DRIVE, P. O. BOX 999, RICHLAND, WASHINGTON 99352

NOTICE

This report was prepared as an account of work sponsored by the United States Government. Neither the United States nor the United States Atomic Energy Commission, nor any of their employees, nor any of their contractors, subcontractors, or their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness or usefulness of any information, apparatus, product or process disclosed, or represents that its use would not infringe privately owned rights.

PACIFIC NORTHWEST LABORATORY

RICHLAND, WASHINGTON

operated by

BATTELLE MEMORIAL INSTITUTE

for the

UNITED STATES ATOMIC ENERGY COMMISSION UNDER CONTRACT AT(45-1)-1830

A REAL-TIME NUCLEAR FUELS INVENTORY
AND CRITICALITY CONTROL SYSTEM

V. L. Crow

COMPUTERS AND CONTROL DEPARTMENT
SYSTEMS AND ELECTRONICS DIVISION

January 1971

BATTELLE MEMORIAL INSTITUTE
PACIFIC NORTHWEST LABORATORIES
RICHLAND, WASHINGTON 99352

TABLE OF CONTENTS

	<u>Page</u>
I INTRODUCTION	1
II COMPUTER HARDWARE DESCRIPTION.	2
III INVENTORY AND CRITICALITY CONTROL SOFTWARE	7
A. Requirements	7
B. Structure of the Data Base	9
C. Basic Operations	15
D. Auxiliary Operations	21
E. Supervisory Operations	22
IV UTILITY SOFTWARE	26
A. Program Preparation and Modification	26
B. Assembly, Compilation and Program Loading.	27
C. Disc File Maintenance.	27
V MONITOR SOFTWARE SYSTEM	28
A. Requirements.	29
B. Design Philosophy.	29
C. Overview of Monitor Operation.	34
D. System Service Features	43
APPENDIX A - INVENTORY/CRITICALITY CONTROL SOFTWARE TECHNICAL DESCRIPTION.	A-1
A1, Transfer Origin Program	A-7
A2, Transfer Termination Program	A-18
A3, Material Mixing Program	A-25
A4, Sub-Batch Creation Program	A-32
A5, Binder Addition and Removal	A-37
A6, Batch Status Type Out.	A-41
A7, Hood Status Type Out	A-43
A8, Criticality Control System Maintenance Program	A-45

TABLE OF CONTENTS (continued)

	<u>Page</u>
APPENDIX B - MONITOR SYSTEM SOFTWARE	
TECHNICAL DESCRIPTION	B-1
B1, Fixed Head Disc Input/Output	B-2
B2, System Startup Routine.	B-5
B3, System Monitor Program.	B-7
B4, Program Scheduler	B-11
B5, Executive Control Programs.	B-14
B6, Real-Time Clock and Calendar	B-32
B7, Program Suspension and Termination.	B-34
B8, Disc Data File Input/Output	B-38
B9, Input/Output Initialization and Formatting.	B-46
B10, Input Interrupt Response	B-61
B11, Output Interrupt Response.	B-71
B12, Input/Output Termination	B-78
APPENDIX C - SAMPLE COMPUTER/OPERATOR DIALOGUE FOR	
INVENTORY/CRITICALITY CONTROL PROGRAMS	

LIST OF FIGURES

<u>Figure No.</u>		<u>Page</u>
1.	Central Computer Facility.	3
2.	Typical Alpha-numeric Remote Terminal.	4
3.	Computer Hardware Configuration	6
4.	Schematic Floor Plan of Work Area.	8
5.	Fixed-Head Disc Data File Structure	10
6.	Typical Task-hogram Relationship.	35
7.	Fixed-Head Disc Allocation	40
A-1	Operator File Entry	A-2
A-2	Parameter File	A-2
A-3	Batch File Entry	A-3
A-4	Hood File Entry	A-4
A-5	Active Transfer File Entry	A-5
A-6	Transfer Log File Entry	A-6
A-7	Transfer Origin Program	A-11
A-8	Transfer Termination Program	A-19
A-9	Material Mixing Program	A-26
A-10	Material Mixing Subroutine	A-28
A-11	Batch Type Subroutine.	A-29
A-12	Sub-Batch Creation Program	A-33
A-13	Binder Addition and Removal Program	A-38
A-14	Batch Status Typeout Program	A-42
A-15	Hood status Typeout Program	A-44
A-16	Criticality Control System Maintenance Program	A-46
B-1	Fixed Head Disc Read Driver	B-3
B-2	Fixed Head Disc Write Driver	B-4
B-3	System Startup Routine	B-6
B-4	System Monitor	B-8
B-5	Task List (queue).	B-9
B-6	Program Scheduler.	B-12
B-7	Program Directory.	B-13

LIST OF FIGURES (continued)

<u>Figure No.</u>		<u>Page</u>
B-8	Foreground Executive	B-15
B-9	Foreground Task Status Block	B-16
B-10	Background Executive	B-18
B-11	Background Status Block.	B-19
B-12	Executive Check Routine.	B-21
B-13	Save Header If Necessary	B-23
B-14	Save Background If Necessary	B-25
B-15	Load Foreground Program.	B-26
B-16	<i>Dynamic</i> Core Directory	B-27
B-17	Input Task's Data Area	B-28
B-18	Remove Task's Data Area	B-30
B-19	60 Hz Clock Interrupt Routine	B-33
B-20	EXIT Routine	E 3 5
B-21	ABORT and ALARM Routines	B-36
B-22	User Disc Data File Directory	B-39
B-23	Add/Delete User Data File	B-40
B-24	Open/Close Disc Data File	B-42
B-25	Read (Write) Disc Data File.	F3-43
B-26	Advance (Backspace) File Pointer	B-45
B-27	Structure of User's 1/0 Request.	B-47
B-28	Function Key Status Table Entry	B-48
B-29	Device Status Table Entry	B-48
B-30	Input/Output Scheduler	E 5 0
B-31	Input and Output Initializers	B-51
B-32	Typical Input Initializer	B-53
B-33	Typical Input Fomtter.	B-55
B-34	Format Handler for Input	B-56
B-35	Typical Output Fomtter	B-58
B-36	Format Handler for Output	B-59
B-37	Remote Terminal Input Interrupt Response	B-62
B-38	Common Input Interrupt Response	B-63

LIST OF FIGURES (Continued)

<u>Figure No.</u>		<u>Page</u>
B-39	Paper Tape Input Interrupt Response . . .	B-65
B-40	Function Key Response . . .	B-67
B-41	Teletype Input Interrupt Response Subroutine . . .	B-69
B-42	Remote Terminal Output Interrupt Response . . .	B-72
B-43	Remote Terminal Output Interrupt Subroutine. . .	B-73
B-44	Common Output Interrupt Response . . .	B-75
B-45	Paper Tape Output Interrupt Response . . .	B-76
B-46	Input/Output Complete Subroutine . . .	B-79
B-47	BTC #2 Interrupt Response . . .	B-80

I INTRODUCTION

The operation of the FFTF Fuel Fabrication Demonstration Facility in the 308 Building of the Hanford Engineering Development Laboratory requires handling of large amounts of fissile uranium and plutonium oxide. These materials are processed in a series of glove boxes in several rooms in the 308 Building as part of the FFTF Project fuels development and fabrication demonstration program. Because of the economic value of these fuel materials and because of the possibility of a critical incident being initiated by an accumulation of too much material in one location, an accurate accounting system is enforced at all points in the fabrication process.

Because of the necessity to accurately account for the amount of fissile material present at each process position in the fabrication facility, a digital data handling system has been installed in the 308 Building. Each time fissile material is transferred from one process station to another, information pertaining to the transfer is input via a remote alphanumeric terminal to the central computing system. The system maintains a running inventory of the contents of each process station. Each time a material transfer occurs the inventory for the involved process station is updated. Therefore, the system can provide, in real-time, information about the contents of any process station. In addition, criticality control is effectively assisted by the system in that a proposed transfer of material from one point to another can be checked against preset criticality safety limits before the transfer is authorized. Also, the facility provides an effective record keeping system in that information about each individual material transfer is recorded and is therefore available for playback at some future time. This feature provides traceability and accountability for all fissile material passing through the fabrication facility.

The computer system was designed around the philosophy that an on-line real-time facility was required to maintain credibility of data stored and retrieved in a dynamic process environment. In

addition, multiprogramming and time-sharing techniques were employed to provide different process operators with simultaneous and asynchronous access to the computer system. Also, a foreground and background technique was employed in order to provide users with the ability to perform certain batch processing functions such as program preparation, system maintenance, and execution of certain analytical programs concurrent with the inventory/criticality control function.

The purpose of this document is to acquaint the reader with the general operation of the inventory/criticality control system and to describe the techniques used in its implementation. Detailed technical descriptions of the monitor system and of the inventory/criticality control software are contained in the appendices.

II COMPUTER HARDWARE DESCRIPTION

Figure 1 shows the equipment contained in the central computer facility on June 30, 1970. All of the equipment shown is manufactured and marketed by Systems Engineering Laboratories of Ft. Lauderdale, Florida. In addition, the system includes several alphanumeric remote terminals as shown in Figure 2. These terminals consist of KSR-35 teletypewriters manufactured by Teletype Corporation of Skokie, Illinois and modified for use in this system by Battelle-Northwest personnel. The terminals are connected to the computer system by means of a teletype multiplexer interface which was designed and constructed at Battelle.

The teletypewriters were modified to include six back-lighted function keys and two function lights on a side panel to the right of the keyboard. These lights and keys were included to simplify the process of communication between the computer system and the operators at remote locations in the fabrication facility. For example, if a technician wishes to transfer material from one location to another, he need only depress a single function key to inform the computer system of his desire. The computer may then respond on the remote

BNWL-1440

FIGURE 1. Central Computer Facility

FIGURE 2. *Typical Alpha-numeric Remote Terminal*

terminal's printer with queries as to additional information it needs before it can authorize the transfer.

Shown in Figure 3 is a simplified block diagram of the computer hardware system. The central unit in the system is an SEL Model 810B digital computer with 16,384 16-bit words of core memory. The other units depicted in Figure 3 are peripheral input/output devices connected to the central computer. These include the following:

- A console teletypewriter used for communication between the computer and the central computer operator

A paper tape reader and punch used for input and output of computer program information

- A quarter million word random access fixed-head disc memory used primarily as an extension of the central computer's core memory for rapid storage and retrieval of program and data information
- A magnetic tape controller and two IBM compatible magnetic tape drives used for storage and retrieval of long term data
- A 300 line-per-minute printer used for output of tabular information
- A remote teletype multiplexer and several remote terminals as described above for communication between the computer system and operators in the fabrication facility.

All I/O hardware shown in Figure 3 is operated in the interrupt mode. This means that whenever an I/O device requires service it will send a signal to the central computer indicating that service is required. This will cause the computer to generate a program branch to a service routine which will take the appropriate action required by the I/O device. In this way, time shared asynchronous operation of all I/O devices in the system can be realized.

FIGURE 3. Computer Hardware Configuration

III INVENTORY AND CRITICALITY CONTROL SOFTWARE

A. Requirements

The primary purpose of the Inventory/Criticality Control System is to maintain an accurate, up-to-date account of the amount and position of fissile material everywhere in the 308 Building fuel fabrication area. Figure 4 depicts a schematic floor plan of the fuel fabrication area, showing the flow of fuel pin component material through the area. During the fabrication process, fissile material is taken from the fuel material storage vault and transferred to a series of glove boxes, as shown in Figure 4. The fabrication process proceeds in a series of logical steps, with each step being performed in one of the glove boxes. In order for any system to keep track of the material as it moves through the fabrication facility, it is necessary that each glove box, and each item of material, be given a unique identification code. Then, each time a material transfer occurs, the system can record the point of origin, the point of destination, and the identification of the material involved.

In order to satisfy the needs of the fabrication facility, it was decided that the computer software system responsible for inventory/criticality control should have the following characteristics:

It must maintain a running inventory of each glove box in the system, including the identity, quantity, and composition of each fissile item in each box

Authorized operators should be the only persons allowed to transfer fissile material between points in the facility

- Operators wishing to transfer material should be queried as to their origin, destination, material identification, and material quantity, and a criticality safety check should be made on each pending transfer

FIGURE 4. Schematic Floor Plan of Work Area

- At the completion of a material transfer, the operator should "check in" with the computer system so that the information received at the point of origin may be verified
- If the computer system detects a potentially unsafe condition, such as an unauthorized material transfer, supervisory personnel should be notified
- If the material is modified but not transferred to a new location, the computer system should be so informed
- Operators should be able to query the computer system to obtain the position or status of a particular item, or the contents of a particular glove box
- Supervisory personnel, but not operators, should have the capability to examine and modify certain system parameters, such as criticality safety limits for the glove boxes, and the identification of authorized operators
- A record of each material transfer, containing all available information, should be maintained and be output to supervisory personnel on demand.

B. Structure of the Data Base

In order to accommodate the above requirements, a software system was developed which consists of a number of foreground programs, operated with remote teletype terminals placed at various locations along the fabrication line, and several background programs, operated from the central computer facility. These programs operate on a common data base which is maintained on a portion of the fixed-head disc memory. A map of this area of disc memory is shown in Figure 5. Note that the area is divided into six data files, each containing a different type of information about the fabrication facility and its current status. Identification of the various items contained in these files is as follows:

<u>OPERATOR FILE:</u>	CONTAINS LIST OF PAYROLL NUMBERS OF PERSONS AUTHORIZED TO TRANSFER FISSILE MATERIAL WITHIN THE SYSTEM.
<u>BATCH FILE:</u>	ONE ENTRY FOR EACH CURRENT LOT, BATCH, AND SUB-BATCH IN THE SYSTEM: CONTAINING IDENTIFICATION, AND WIGHT PERCENT CONCENTRATIONS OF UO ₂ , PUO ₂ , MODERATOR, PU IN PUO ₂ , U IN UO ₂ , AND U ₂₃₅ IN U.
<u>HOOD FILE:</u>	<p>ONE ENTRY FOR EACH GLOVE BOX AND HOOD IN THE SYSTEM, CONTAINING IDENTIFICATION, CRITICALITY WEIGHT LIMITS, AND CURRENT WEIGHT TOTALS FOR FISSILE MATERIAL, MODERATOR, AND HOLDUP MATERIAL.</p> <p>SUB-ENTRIES FOR EACH MATERIAL BATCH, LOT, OR SUB-BATCH IN EACH HOOD, CONTAINING IDENTIFICATION, MATERIAL WEIGHT, AND MODERATOR WEIGHT FOR EACH.</p>
<u>ACTIVE TRANSFER FILE:</u>	ONE ENTRY FOR EACH MATERIAL TRANSFER WHICH HAS BEEN INITIATED BUT NOT TERMINATED, CONTAINING IDENTIFICATION OF MATERIAL, ORIGIN POINT, ETC.
<u>TRANSFER LOG:</u>	ONE ENTRY FOR EACH MATERIAL TRANSFER WHICH HAS BEEN PROPERLY TERMINATED, CONTAINING ALL AVAILABLE INFORMATION ABOUT THE TRANSFER.
<u>PARAMETER FILE:</u>	A LIST OF PARAMETERS USED BY THE CRITICALITY CONTROL SYSTEM IN MAINTAINING THE ABOVE FILES.

FIGURE 5. Fixed-Head Disc Data File Structure

■ Location I.D.

Each location is assigned an alphanumeric identification code by supervisory personnel. This code consists of a one or two digit number which may or may not be followed by a letter; for example:

2, 12, 23B

■ Material I.D.

Material received from outside sources is in bulk quantities, called lots. There are four varieties of lots, namely, plutonium, depleted uranium, natural uranium, and enriched uranium. These are assigned lot identification numbers consisting of three letters and three digits; for example:

PUL001, UDL123, USL456, UEL789

The first two letters of the material identification code designate the type of material, as follows:

PU = plutonium

Up = depleted uranium

US = natural uranium

UE = enriched uranium

During the fabrication process, portions of lots are taken through the line, and may be combined with portions of other lots to form batches. These **batches** are assigned **batch numbers** by the computer system as they are created from their parent lots. The format of a **batch number** is similar to that of a lot number, with the first two letters depicting the material type. Thus, possible batch numbers are:

PU 001, Up 123, US 456, UE 789

For batches consisting of **mixtures** of uranium and plutonium dioxides, the following convention is used for the first two characters of the batch I.D. number:

FD = Pu + depleted uranium

FS = Pu + natural uranium

FE = Pu + enriched uranium.

During the fabrication process, batches of material may be subdivided to form sub-batches. These sub-batches are identified by appending a decimal point and a single digit to the parent batch number; for example:

HE 365.1, HE 365.2, and FE 365.3 are sub-batches of batch FE 365.

■ Operator I.D.

Authorized operators are identified to the system by their company payroll numbers; a five digit integer. (00000-99999)

The six data files containing the data base for the Inventory/Criticality Control system are structured as follows:

1. Operator File

This file contains a list of payroll numbers; one for each person who is currently authorized to transfer or modify fissile material within the fabrication system.

2. Batch File

For each material lot, batch, and sub-batch currently identified to the computer system, this file contains one entry consisting of the following:

- a. A batch type code, identifying the material as plutonium, depleted uranium, etc.
- b. The lot, batch, or sub-batch identification number.
- c. The composition of the material, expressed in weight percents, as follows:
 - 1) PuO_2 in the Batch
 - 2) Pu in PuO_2
 - 3) UO_2 in the Batch
 - 4) U in UO_2
 - 5) U_{235} in U
 - 6) Moderator in the Batch

3. Hood File

For each unique physical area defined to the computer system this file contains the following entries:

- a. The identification ~~code~~
- b. The net weight criticality safety limit
- c. The moderator weight criticality safety limit
- d. ~~The total~~ net weight currently present at ~~this~~ location
- e. The total ~~moderator~~ weight currently present at ~~this~~ location
- f. The weight of holdup ~~material~~ currently present at ~~this~~ location
- g. The serial number of the transfer ~~now~~ in progress to this location, if any
- h. The ~~number~~ of different material lots, ~~batches~~, and/or sub-batches ~~now~~ contained in this location
- i. For each lot, batch, and/or sub-batch in ~~the~~ location, the following subentries:
 - 1) The lot, batch, or ~~sub-batch~~ identification ~~number~~
 - 2) The net weight
 - 3) The ~~moderator~~ weight.

4. Active Transfer File

This file contains one **group** of entries for each material transfer which has been initiated by an operator, but has not yet been terminated:

- a. The serial number of the **active** transfer
- b. The identification of ~~the~~ origin location
- c. The identification of ~~the~~ destination location
- d. The identification of the ~~material~~ lot, batch, or sub-batch **being** transferred

- e. The weight of the material being transferred
- f. The ~~time~~ of day (hour and minute) when the transfer originated
- g. The calendar date on which the transfer originated
- h. The payroll ~~numbers~~ of the two operators who initiated the transfer.

5. Transfer Log

This file contains one group of entries for each transfer which has been ~~completed~~, as follows:

- a. The serial number of the ~~completed~~ transfer
- b. ~~The~~ identification of the material lot, ~~batch~~, or sub-batch which was transferred
- c. ~~The~~ weight of the material which ~~was~~ transferred
- d. The calendar date upon which the transfer ~~took~~ place
- e. The identification of the origin location
- f. The payroll ~~numbers~~ of the two operators who originated the transfer
- g. The ~~time~~ of day (hour and ~~minute~~) when the transfer originated
- h. The identification of the destination location
- i. The payroll numbers of the two operators who ~~completed the transfer~~
- j. The ~~time~~ of day (hour and minute) when the transfer was ~~com-~~pleted.

6. Parameter File

This file contains ~~information~~ ~~used~~ by the system to update and maintain the other five files, and contains the following:

- a. The last assigned transfer serial number (0-999)
- b. The number of transfers which are currently active, i.e., originated but not completed (0-10)

- c. The last assigned batch number for each of the seven different types of material batches (0-999)
- d. The number of entries currently in the transfer log file (0-100).

The information contained in the six data files described above represents a complete picture of the status of the fuel-fabrication facility at any time, as well as a historical record of operation, involving material transfers. The programming of the Inventory/Criticality Control System, then, must provide the means of properly updating the contents of these files, and reporting their contents in printed form.

C. Basic Operations

The programs described below are employed by operators of the Fuel Fabrication Facility at various stages of the fuel pin fabrication process. The operators activate these programs and operate them via the remote teletype terminals which are placed at strategic locations along the fabrication line.

1. Material Transfer Origination

This program becomes operational when the operator requests through a remote terminal that a material transfer be initiated. Upon activation, the program checks the parameter file to see if 10 transfers are currently in progress. If so, a message is output to the operator indicating that he must wait until a transfer is completed, since only 10 transfers are allowed to be active in the system at any one time. If less than 10 transfers are currently active, the computer system will query the operator for his payroll identification number. When the operator enters this, it is checked against the content of the operator file to see if the operator has previously been authorized by supervisory personnel to transfer material. If so, the computer system requests that a second payroll number be entered, thus requiring that two

authorized operators initiate each material transfer. If either payroll number is not to be found in the operator file, the computer system will indicate that an unauthorized payroll number has been entered and no further communication between the computer terminal and the operator is allowed. If two valid payroll numbers are entered, the system then queries the operator for the identification of the point of origin. The operator then answers and is in turn queried for the identification of the destination point for the transfer. At this point the hood file is checked to see if a transfer is already in progress to the destination location. If this is the case, then the transfer currently being initiated is not allowed and the operator is informed that he should verify the error condition that exists by pressing the reset key on his remote teletype terminal. He is given 15 seconds to respond in this manner and if he does not, an alarm type out will appear on the console typewriter at the central computer location. This alarm will be repeated every ten seconds until the condition is verified by actuation of the reset key at the remote teletype terminal. If no transfer is in progress to the requested destination, the operator will be asked to input the identification number for the material he wishes to transfer. This being done, the weight of the material about to be transferred is requested by the remote terminal and when the operator inputs this quantity, a series of checks are made against entries in the various systems files.

First, a check of the hood file is made to see if the batch about to be moved is indeed present in the origin hood. If so, the weight of the material requested for transfer is compared with the weight of the batch presumed to be in the origin station according to the hood file. If either of these tests fail, the operator is asked to input correct information.

When the system determines that enough material exists at the origin station to allow the transfer, it then looks up the destination station in the hood file and computes two quantities given the proposed transfer weight and the amount of material now at the destination station. These two quantities are the amount of moderator and the net rate of material that would be present in the destination station if the transfer now being initiated were to take place. Once computed, these two quantities are compared against preset criticality weight limit and if either of them exceed the preset criticality limit, an error condition exists. In the case of an error condition, the operator is notified of the condition and is given 15 seconds to respond by pressing the reset key on his remote terminal. If he does not respond in the allotted time an alarm message is typed on the console typewriter at the central computer location and is repeated every ten seconds until someone acknowledges the error condition.

If the computer determines that the proposed transfer can be made safely, it obtains the next available trip number from the parameter file and issues it to the operator along with a message that his transfer has been approved and is assumed to be underway. The program then updates the parameter file, the active transfer file, and the hood file according to the information which has been input by the operator. The program then terminates releasing the remote terminal upon which the dialogue occurred to general availability.

2. Material Transfer Termination

This program is initiated by the operator by pressing the "END TRANSFER" key on a remote teletype terminal. The program's first function is to scan the parameter file to determine if any material transfers are active. If no transfers are active, the program simply exits as it would be impossible to terminate a transfer if none had been

originated. If one or more material transfers are in progress, the program asks for two authorized operator payroll numbers in the same manner as the material transfer origination program described above. Following the input of the second payroll number, the operator is asked for the transfer serial number of the transfer he wishes to terminate. When this has been input, it is checked against the entries in the transfer file and if a match is not found an error message is output to the operator. If a match is found the operator is asked the identification of the destination station. When this is input, it is compared with the entry in the transfer file corresponding to the destination that was given at the origin of the transfer in progress. If the two destination identification numbers do not correspond the operator is informed that he is at the wrong location. An error condition then exists and the operator has 15 seconds to verify this condition by pressing the reset key on his remote teletype station. If he fails to respond within 15 seconds, an alarm is typed out on the console typewriter at the central computer location.

If the destination given at this time properly corresponds with the destination identification given at the origin of the transfer, the operator is asked for the weight in grams about to be entered into the destination station. If this weight does not correspond within 10 grams to the weight entered for this transfer at the origin an error condition exists and the operator has 15 seconds to respond by pushing the reset key. Again, if no response is made within the allotted time, an alarm is typed at the central computer location and the system repeats the alarm until the condition is corrected. In this case the transfer remains active until it is properly terminated.

When the transfer has been properly terminated, the remote teletype will indicate that the transfer is complete and the program will proceed to update the hood file, the transfer log, and the parameter file and will remove this entry from the active transfer file. This being done, the program terminates and the remote terminal is released to general availability.

3. Material Modification

The following programs are employed when the composition and/or identification of material within a process station is to be modified but no physical transfer of material from one place to another is to occur. Each of the following operations is initiated by the operator pressing the "OPERATION" function key on a remote teletype terminal.

In order to combine two different lots, batches, or sub-batches that are present in a process station, to produce a new batch of material, the operator types MIX. This program first asks the operator to enter the identification of the hood containing the batches which are to be combined. It then requests the identification number and weight for each of the two materials which are to be combined. Following the input of this information, the program checks the contents of the batch file and the hood file to determine if the information just input by the operator is valid. If not, the appropriate diagnostic message is output to the operator and the program terminates. If all input information is valid, the program proceeds to calculate the weight and composition of the newly created batch. It then assigns a new batch number based on the current information in the parameter file and the composition of the new batch and places this information in the batch file. The hood entry for the station in which this operation takes place is then updated and the new information is written back into the hood file. The parameter

file is then updated and the operator is informed of the newly assigned batch number. The MIX program then terminates.

To create a sub-batch in a specified hood from a batch in that hood, the operator types SUBBCH which initiates the sub-batch program. This program begins execution by asking the operator for the identification of the station in which the sub-batch is to be created. Following the response it asks for the identification of the batch from which the sub-batch is to be created and the weight of the sub-batch. The program then verifies the information just input against information existing in the batch and hood files and, if a discrepancy exists, a diagnostic message is output to the operator and the program terminates. If the information input by the operator is valid, the program assigns a sequential batch number appending this to the batch number of the batch from which the sub-batch is being created. It then updates the proper entry in the hood file by entering the new batch in that entry and subtracting the weight of the sub-batch from the existing batch in the hood. Following this it outputs a message to the operator indicating that the information has been accepted properly, and terminates.

To add or remove binder material to or from a sub-batch in a specified hood the operator types BINDER. This program first asks the operator whether he wishes to add or remove BINDER and the operator responds by typing A or R respectively. The program then queries the operator for the hood identification in which the operation is to take place, the identification of the sub-batch in that hood to which BINDER is to be added or from which BINDER is to be removed, and the weight of the binder to be added or removed. The program then compares the information input with existing information in the batch and hood files and checks the input information for validity. If the information is invalid, the program

terminates. If the information input is valid, the proper entry in the hood file is updated with the modified binder weight and the batch entry in the batch file is modified to show the new composition of the batch following the addition or removal of the binder. The program then outputs a message to the operator indicating that the information is accepted, and terminates.

D. Auxiliary Operations

The operations described below do not affect the contents of any information file in the system but simply output various items from those files in order that operators may review the particular status of the system at any time. These operations, like those described above, may be initiated from any remote teletype in the system. The operator initiates the following operations by depressing the operation function key on a remote terminal.

1. Status Output

To output the status of any particular hood in the system, the operator types HOOD. The system then queries the operator for the identification number of the hood of interest. Following the operator's response, the remote terminal displays the material identification number of each item in the hood, along with the weight in grams of each item that is contained in the hood. In addition, if a material transfer is in progress to that location the transfer serial number is displayed.

The operator may inquire as to the current status of any material lot, batch, or sub-batch defined to the system by typing BATCH. This program queries the operator for the identification number of the material item of interest. It then outputs the entry in the batch file corresponding to the identification number typed by the operator. As we have seen in subsection B above, this information consists of the weight percent composition of the lot, batch, or sub-batch. In

addition, the program scans the hood file and outputs the identification number of each location that contains an amount of the material of interest. Following output the program terminates.

2. Tim and Date Output

The computer system contains a real-time clock which once initiated by supervisory personnel keeps a running tabulation of the current time and calendar date. This information may be output at any time on any remote teletype terminal by simply typing TIMDAT. This feature provides a convenient means for dating status printouts and records of material transfers made as described above.

E. Supervisory Operations

Many of the operations necessary for the maintenance of the Inventory/Criticality Control System are of a nature such that only supervisory personnel should execute them. Therefore, programs responsible for these operations have been written such that they may only be initiated from the console teletype in the central computer facility and then only if a "privileged operation" enabled key-lock switch is turned on.

1. Time and Date Input

This program is initiated from the console teletype by typing INTIME. It queries the operator for the current time of day and for the current date. When the operator inputs this information the real-time clock and calendar of the software monitor system are set to the values which the operator input and the program terminates.

2. Material Identification

Material lots are initially identified by supervisory personnel by typing LOT on the console teletype. This initiates the program which queries the supervisor for the

identification number to be assigned to the material lot being defined. Following the supervisor's response, the weight percent composition of the lot is requested. The supervisor responds by inputting the weight percents of plutonium dioxide in the lot, plutonium in the plutonium dioxide, uranium in the uranium dioxide, and uranium 235 in the uranium. This information is then entered into the system batch file and the program terminates.

3. Status Output and File Maintenance

The following functions are performed under the control of a privileged background program called CRITIC. This program is initiated by typing CRITIC on the console teletype. Its first task is to query the supervisor for the desired function. The supervisor may perform any of the following operations by typing one of the following codes:

- ADDO - This operation is used to add a payroll number to the list of authorized payroll numbers in the operator file. The supervisor is queried for the payroll number and when he enters it, it is copied as he types it to the operator file and CRITIC then requests the next function.
- DELO - This function is used by the supervisor to delete an existing operator payroll number from the operator file. The supervisor is queried for the payroll number to be deleted and when he enters it, it is removed from the operator file.
- LISO - This operation causes the contents of the operator file to be output to the line printer producing a list of the payroll numbers of all the persons currently authorized to transfer material.
- ADDH - This function is used by the supervisor to define a new material location to the system thus making a new entry in the hood file. When the function is initiated,

it queries the supervisor for the new hood identification and its net weight and moderator weight, criticality safety limits. It then creates the new entry in the hood file and returns control to the CRITIC program.

- DELH - This function enables the supervisor to delete a hood from the hood file. The supervisor is queried for the identification of the hood to be removed. When he responds, this hood and all information associated with it is removed from the hood file and control returns to the CRITIC program.
- LISH - This function causes a listing of all hoods defined in the hood file along with criticality safety limits to be output to the line printer.
- LISB - This function causes the entire contents of the batch file to be output to the line printer, showing all lots, batches, and sub-batches in the file and the percent composition by weight of each.
- CLEN - This function is used by the supervisor to indicate to the system that a hood has been cleaned. The supervisor is queried for the identification of the hood to be cleaned and when he responds all material weight levels of that hood are set to zero in the hood file. Control then returns to CRITIC.
- CORR - This function allows the supervisor to correct an error made by an operator in transferring material by allowing the supervisor to add a specified weight of a specified material to hood. The supervisor is queried for the hood identification and for the batch identification and weight to be added to the hood. Following his response, the proper adjustments are made to the hood and batch files and control returns to the CRITIC program.

DELB - This function allows the supervisor to remove a specified Lot, **batch**, or sub-batch from **the** batch file provided that that particular material is not present in any hood in the system. Upon initiation of this function, the supervisor is queried for the batch identification. Following his response, the hood file is **searched** and if the material is found to be present in one or more of the hoods, the supervisor is **informed** of the location of the **material** and no action is taken to remove the **batch** from the batch file. If, however, the batch cannot be found in any hood, it is purged from the batch file. In either case control returns to the CRITIC program.

- TRAN - This function causes the entire **contents** of the active transfer file to be output to the line printer. This gives the supervisor all pertinent **information** about any transfers which have been originated but have not yet **been terminated**.
- TLOG - This function causes the **entire contents** of the transfer log to be output to the line printer resulting in a **complete** historical printout on all transfers which have **been terminated** since the last time that the transfer log was PURGED.
- PURG - **Since** the transfer log file *can* only contain 100 entries, **it** is necessary from time to time to empty it out **and thus** keep it from overflowing. The PURG function **accomplishes this** by zeroing out the entire contents of the transfer log file.
- CI - This function causes a **complete inventory** by hood of **every hood** defined in the system. It shows a **complete breakdown** by **batch** of each item in each hood and also indicates whether or not a transfer is in progress to each hood.

- QUIT - This function causes a termination of the CRITIC program and releases the console teletypewriter to general availability.

IV UTILITY SOFTWARE

Most computer installations have a need for the ability to prepare and install new software and update existing routines, while continuing to operate existing systems. The computer system at the 308 Building is no exception, and the utility routines described below facilitate this requirement:

A. Program Preparation and Modification

Source programs on magnetic or paper tape are difficult to modify due to the nature of the media. For this reason, the UPDATE program was developed for use under control of the operating system. This program provides the following features:

- Inter-device copying of source statements
- Deletion of unwanted statements
- Insertion of new statements
- Listing of source files
- Magnetic tape positioning by file and record number
- Control statements and corrections input from any text-oriented device in the system
- Dynamic device reassignment

Initial preparation of source programs may be accomplished by using UPDATE to copy source statements from the console teletype keyboard to paper or magnetic tape. Correction of an existing program is effected by copying the program from one device to another with control statements, containing line number arguments, being input from a third device specifying changes to be included in the new version of the program.

B. Assembly, Compilation and Program Loading

In order to convert programs written in assembly language or FORTRAN IV to machine-language programs operating in the disc-core environment of the computer system, the following language translators are employed:

- A symbolic macro-assembler, to convert SEL 810B assembly-language coded programs into object data
- An ASA FORTRAN IV compiler, to convert FORTRAN statements to object data
- A linking, relocating loader program to convert object data output by the assembler and compiler into machine-language instructions in core memory, and append any required library subroutines.

The three processors described above were included in the system by modifying existing versions of Systems Engineering Laboratories software to make these vendor-supplied programs compatible with the required multi-programming environment. In addition, SEL-supplied subroutines were included in the system's relocatable library.

The SEL programs used are as follows:

- SEL 810A/B FORTRAN IV Compiler, Catalog #044004F
- SEL 810A/B O.S. macro-assembler, Catalog #3B4001A
- SEL 810A/B standard load/dump package, Catalog #300001D
- SEL 810A FORTRAN-coded library, Catalog #044002B
- SEL 810A machine-coded library, Catalog #044003G

C. Disc File Maintenance

The files on fixed head disc which contain program and inventory data require updating from time to time. The system includes utility programs which facilitate maintenance of the following disc areas:

The relocatable subroutine library

- The program file
- The **data** file area

The relocatable subroutine library is created by a utility which copies object data from paper or magnetic tape to the proper **area** of fixed-head disc. To update this disc area, the operator simply uses the utility to copy new library **routines** onto the area.

The **fixed-head** disc program file consists of a group of **core-image** blocks, where each block is a copy of a foreground task or **background** program which may be transferred into core and executed on demand. **The system** maintains a directory of these program blocks on the disc, and a utility program is available which allows the **operator** to add new programs to the file, delete **programs**, modify existing entries, **dump** entries to paper tape in absolute, **core image** format, and list the **contents** of the **program** directory.

The user data file area of disc is similar in structure to the program area, except that each **entry** in this area contains binary data rather than executable program **images**. The directory for this area of disc contains an entry for each **named** file in the area, **and** a utility program is available which allows the **operator** to add and delete named data files **and** to list the **contents** of the data file directory.

MONITOR SOFTWARE SYSTEM

Whenever a general purpose digital **computer** is employed with various operations being performed simultaneously, **it** is desirable that the different **software components** be coordinated by a central, supervisory program, called a "**monitor**." The monitor may also contain service programs which are available to any **application** program operating under control of the system.

A. Requirements

The monitor software system for the 308 Building computer system **was** required to **comply** with the specialized needs of the application programs described in Sections III and IV of this document, and therefore was designed to include the following features:

- Simultaneous execution of several different **inventory/criticality control application programs**, with operator **communication** through remote **terminals**
- Simultaneous and asynchronous operation of a single application program by **two or more** operators through their respective **remote terminals**
- Operation of the utility software **components** described in Section IV, concurrent with the execution of the **inventory/criticality control programs**

Simultaneous use of any and **all input/output** devices in the system **by** any user, precluding the possibility of inter-user interference

- Rapid response to any operator's request, regardless of the **number** of operators currently requesting action
- A **convenient method** for development of **programs** to run under control of the **monitor system**.

B. Design Philosophy

In order for a general-purpose, stored program digital computer to **meet** application requirements such as **those** given above, it must be so configured and **programmed** as to appear to its users to be doing several operations at the same time. Since such a machine **is** not capable of performing **more** than one single instruction during one instruction time period, we must **enable** the computer to "time-share" its facilities among several users by rapidly switching the context of the system from user to user in

such a way as to give each user the impression that he has the total system ~~under~~ his singular control. This can be done because the speed at which the central processor operates is very fast in relation to the speed of operation of human beings and of their computer ~~input/output equipment~~.

For example, suppose user "A" wishes to operate a program which involves teletypewriter input and output, while user "B", at the same time, is operating a different ~~program~~ which accepts input from the paper tape reader and outputs a listing on the line printer. The actual sequence of events within the ~~computer's~~ software system might then be as follows:

The ~~program~~ requested by user "A" is loaded into the ~~computer's~~ core ~~memory~~ from its storage area on the fixed-head disc storage unit, and control of the machine passes to this program. The program requests that a message be typed out on the teleprinter. The monitor software system responds to this request by placing the character codes for the specified ~~message~~ in a core ~~memory~~ buffer ~~area~~ assigned to the teleprinter. It then sends the first ~~char-~~acter of the message to the teleprinter by executing an output instruction, and enables a hardware interrupt ~~line~~ connected between the teleprinter and the central processor. This interrupt line will cause a pulse to be sent to the hardware logic of the central processor when the teleprinter has finished typing the first character of the ~~message~~. This pulse will cause the central processor to "drop what it is doing" and automatically transfer control to an interrupt processing routine for the teleprinter. This interrupt processing routine will fetch the next character of the message from the core ~~memory~~ buffer and send it to the teleprinter, and then return control of the central processor to whatever program it was executing when the interrupt occurred, at the point of interruption. Thus, the entire message is typed out, one character at a time, at the natural speed of the teleprinter. The program requesting the message, at this point, has nothing to do

but wait for the message to be completed, which may take several seconds to type out.

The key to understanding the operation of any time-sharing computer system is that in processes such as the teleprinter output process described above, the computer can supply information for output, and accept information from an input device, at a rate much faster *than* the device is capable of running. In the above example, the teleprinter will accept characters for output at a rate of ten per second, or one character every one hundred milliseconds. However, it takes the central processor only about one hundred microseconds to go through the program steps necessary to supply each character to the teleprinter for output. Thus only one tenth of one percent of the total available computer time is consumed during the typeout process. The monitor software system is so configured as to make use of part of this spare time by allowing several users to operate different programs simultaneously. Thus, in our example, the monitor would detect that user A's program is awaiting the completion of an output operation, and would then load the program requested by user "B" into core memory, having first saved the context of user A's program on a section of the fixed-head disc storage unit. Control would then transfer to the program requested by user "B," which would execute until it requested an input operation involving the paper tape reader. This input operation would proceed, driven by a character-oriented interrupt line similar to that used by the teleprinter of user A's program. At the completion of an input or output operation in either program, the monitor is informed of the completion, and the appropriate program is loaded, if it is not already in core, and control transfers to it. Control remains in a user's program until an input or output operation is requested, or until the program informs the monitor that it has completed its function.

Since the speed at which input/output devices operate is so much slower than the speed at which the computer is capable of executing program instructions, all of the input/output devices in the system can run at their maximum rates without having to wait for information to be supplied to them by the computer. For this reason, neither user in the above example is aware that the other user is also operating. To each user it appears that he has total control of the computer facility.

As can be seen by the above discussion, it is the manner in which input and output operations are handled that enables us to "time-share" a computer's facilities. However, we must obviously include in our software system some means of keeping track of what needs to be done at each instant, and handle the switching of control from one user program to another. We also need to provide each user with a means of informing the system that his program requires input and output operations and various other functions provided by the system. Therefore, our monitor software system design includes the following component routines:

- A program scheduler, which responds to requests typed in from keyboard terminals indicating that specified user programs are to be activated
- Input and output handlers for the fixed-head disc storage unit, to enable the system to save and restore the context of a user's program and load core memory with another user's program and data
- A monitor routine, to check for changes in the status of various user programs caused by the operation of the program scheduler and the input/output routines, and dictate the appropriate system response
- Executive routines, to carry out the proper response to stimuli detected by the monitor routine, by loading user programs from the program file area on the fixed head disc storage unit into

core memory, updating their status, and transferring control to them at the proper point in their execution path

- A clock/calendar routine, to enable the computer to keep track of the time and date, and to measure time delays requested by user programs
- Input and output routines, to provide users with a means of performing data transfers to and from peripheral equipment.

In most multiprogramming systems, each application routine which runs under control of the monitor software system is set up as a discrete program unit, called a "task." These systems must then provide enough multiprogramming ability to enable all tasks to operate simultaneously. This requires an area of core memory which can be dedicated to each task for status keeping purposes, as well as an area of disc storage to be employed in saving and restoring the context of each task whenever another task requires core memory for its own execution. This approach also requires that, if more than one user wishes to employ a given task at the same time, then a separate copy of the required program must be provided for each user. Since the 308 Building inventory/criticality control system requires a large number of separate application programs, and since each of these programs must be made available to several operators simultaneously, the core and disc requirements of a classical approach to multiprogramming would have been prohibitive. Therefore, a monitor software system was designed which allows up to 50 different programs to be available for use by any operator at any time, but with the restriction that only eight operators may engage in simultaneous operation.

In order to realize the above design structure, the system was set up with eight task status areas in core memory, just as a classical multiprogramming system with eight application program would be. However, the system was given the ability to dynamically assign any of the 50 application programs stored on

the fixed-head disc program file to any of the eight task status areas to form "task-programs." Note that several task status areas may be combined with a single program.

In addition, a foreground/background capability was included in the design, which reserves one task status area for programs designated as "background" programs, which are not shareable, and may be run only under control of the console teletypewriter in the central computer facility. This background task is maintained at a priority lower than that of other tasks, which means that if several tasks are awaiting computer time, the background task will be at the "end of the waiting line," so to speak. The background task is used with programs such as the FORTRAN IV compiler, relocating loader, UPDATE, and other language processors, thus providing the system with batch processing capability in the background, in addition to inventory and criticality control in the foreground.

Figure 6 shows a typical configuration of the dynamic task-program relationship, with two users time-sharing program "A," three other users time-sharing program "B," and one other user in the background, involved with program "C."

Overview of Monitor Operation

1. Task and Background Scheduling

Whenever the monitor system is operative but idle, all eight foreground task status areas, as well as the background status area, are available for use by any operator who wishes to run an application program or background job. When the system is in this idle state, the monitor program is in a loop, scanning all nine status areas in turn, looking for an active task. Assume that, at this time, an operator in the central computer facility depresses a teletypewriter key indicating that he wishes to activate a program. The teletype interrupt response routine will be entered, and will

FIGURE 6. Typical Task-Program Relationship

pass control to the program scheduler routine. This routine will search *the* group of task status areas for an unoccupied area. Since no tasks are currently active, *it* will, in this case, pick task number zero. The scheduler will set task zero's task status area to the "occupied" condition, and will link to this task a foreground program called "FKEY 1." Control then returns to the point of interruption.

The next time the monitor program checks task status area zero, *it* will detect that program "FKEY 1" has been scheduled for execution, and will pass control to the foreground executive routine, which will load program "FKEY 1" from the program file on the fixed head disc storage unit into core memory and transfer control to *it*. "FKEY 1" will inquire, via the console teletypewriter, which program the operator wishes to run. Following the operator's response, "FKEY 1" transfers control to the program scheduler routine, which schedules the requested program in the same way as "FKEY 1" was scheduled. If the requested program is a foreground program, the scheduler will pick the first available foreground task status area and link *it* to the desired program, forming an active foreground "task-program." If, on the other had, the requested program is of the background variety, the scheduler will link the program to the special background status area. In either case, control will then return to program "FKEY 1," which then terminates.

As seen by the above example, operators effect the initiation of desired programs contained on the disc program file by requests made through the program scheduler routine, which alters the contents of an available task status area, setting *it* "active," and linking *it* to a program, so that the monitor, in its next scan of the group of status areas, will inform the proper executive that the newly scheduled program is to be loaded into core and its execution is to commence. This

procedure **may** continue to be repeated as long as status areas are available for linkage to **programs**.

2. Multiprogramming and Time Sharing

When the monitor system is processing several active tasks, the task status area of each active task contains several machine words **which** specify the state of the task at any time. These **words** contain the following specific information:

- The identification number of the program currently linked to the task
- The address of the dedicated area of fixed head disc storage where **the** context: of the **task-program** is to be saved when another task requires use of core memory
- The current entry address to the task-program, **which** is the **point** to which control will transfer when the program is activated

The code number of the **remote terminal** currently associated with the task

- The code **number** of the last **input/output** device used by the task
- A flag indicating the task's **input/output** status
- **The** time of day, as kept by the 60 Hz **real-time** clock, at which the **task** is to be re-activated **following** any specified **time** delay
- The core location of any flag used to conditionally **suspend** the operation of **the** task.

The **monitor** routine, in **scanning** the status areas of all the tasks once, **completes** what is referred to as a "system cycle." **Thus**, during each system cycle, the **monitor** gives each active task a chance to **assume** control of the computer

for a while. When the monitor finds a task whose status area indicates that it is not waiting for the completion of an input/output operation, is not in the midst of a time delay, or is not suspended awaiting some condition, it transfers control to an executive routine which will process the *task* as follows:

The executive routine first checks to see if the task about to receive control is still in core memory from a previous system cycle. If so, then control may immediately transfer to the program associated with the task. If this is not the case, then further testing of the task status area is necessary. In this case, the executive routine looks at the identification number of the program associated with the current task, and looks this program up in the program directory table. This table contains an entry for each program stored on the disc program file, which includes a flag to indicate whether or not the program is currently in core memory. Note that, if more than one operator is using a particular program, as shown in Figure 6, then the program is being time-shared, and it may have been brought into core at the request of another task at a previous point in the current system cycle. In this case, the executive routine will load the current task's temporary information from its dedicated area of disc over the low 111 locations of the program in core memory, having first saved this area on the previous task's disc file. Thus, one program may be shared by several tasks without the necessity of loading a separate copy of the entire program for each task.

If the program desired by the current task is not in core, then the executive routine will load it from the disc program file before restoring the temporary information to the program. Then, in any case, the executive updates the task status area and program directory, so that an accurate picture of the

system's status is maintained at all times. Finally, the executive routine transfers control to the task-program at the address given in the **task** status area.

The above description is valid for the case of foreground programs. The background **task** is treated somewhat differently, in that the entire program area of core memory, rather than the small temporary area, is saved and restored to the disc in the case of the background. Also, since there is only one background task, **background** programs cannot be time-shared as foreground programs *can*.

Once a **task** program receives control, it executes until it needs to transfer control to the monitor system for the purpose of performing an input/output transfer, or initiating a program delay. Since, during the transfer of data or delay interval, the task-program cannot continue its execution, the control of the computer is not returned to the task-program, but is returned to the monitor routine so that other active tasks may make use of the time which is not needed by the current task-program during the data transfer or delay interval. As we have seen in the example at the beginning of this section, it is this feature of the system which makes multiprogramming and time-sharing possible.

As the above discussion shows, the fixed-head disc is used as an extension of core memory, enabling the monitor system to run several user programs at once, and give the entire program area of core memory to *each* one in turn. Figure 7 shows a schematic representation of how the disc is allocated, and it is the "swap areas" which are used by the monitor system in saving and restoring the core memory context of the various tasks. The disc, as Figure 7 shows, also contains storage areas for programs which run under control of the monitor system, relocatable subroutines used by FORTRAN programs in the background batch-processing system,

FIGURE 7. Fixed-Head Disc Allocation

a data file area used by both foreground and background programs for information storage and retrieval, and a core-image copy of the monitor system itself, which may be loaded into core memory during the system startup procedure.

3. Input/Output Considerations

When a program operating under the monitor system requires transfer of **data** to or from a peripheral device, it transfers control to a monitor service routine called the Input/Output Scheduler. This routine receives from the calling program the **code name** of the device to be involved in the transfer. If the device is currently unavailable, as **would** be the case if it were in use by another program, the I/O Scheduler will inform the monitor system that the current task-program should be suspended until the requested device becomes available to it, and will then return control to the monitor's task status scanning loop. The status of each peripheral device in the system is kept in an entry word in a device status table, and the requesting task is simply set in a waiting condition in which its task status area indicates that it is to be re-entered when **the** device **status** changes to **the** "not busy" condition. Thus, if several task programs operating concurrently request the use of one input/output unit, the system maintains a "first come, first serve" procedure, with each task waiting its turn to use the device.

When a requested device is available to the requesting task program for use, the I/O Scheduler transfers control to either an input or output initializing routine, depending on the specified direction of the data transfer. This routine, in the case of output, will obtain formatting information from the calling task-program, and will format the output data as ASCII characters in a dedicated buffer in the monitor system area of core memory. One such buffer exists for each input/output device in the system. Following formatting, the

initializer begins output by sending the first character to the peripheral unit, or, in the case of magnetic tape, by initializing the block transfer channel to begin automatic direct memory access by the tape unit. Then, the requesting task is set in a waiting status until the transfer has been completed, and control returns to the monitor routine, which looks for another active task in order to make use of the time taken by the data transfer operation. In the case of an input transfer, the input initializing routine starts the transfer of data from the unit into the dedicated core memory buffer for the device, then suspends the task and passes control to the monitor.

For character-oriented devices, such as typewriters, data transfers proceed on a character interrupt basis, using the hardware interrupt facility of the 810B to inform the computer that the unit is ready for each character transfer. This continues until a terminating character is found, at which point the interrupt service routine sets the device's status so that the monitor will detect that the transfer is complete. In the case of units connected to the automatic block transfer channel, character transfer is completely automatic, and a single interrupt at the end of the block transfer informs the system that the transfer is finished. In either case, the monitor will, upon detecting that the transfer is complete, will cause the requesting task-program to be re-activated and entered. If the transfer just completed was an output transfer, the task-program continues its execution at this point. In the case of an input transfer, the data in the device's dedicated core memory buffer is converted to binary data in the user's core area before the task program continues.

Input/output transfers to the data file area of the fixed-head disc storage unit are not handled like other device transfers, but are performed directly through the use of the disc

input/output handlers. This is done because the disc input/output is fast enough to make suspension of the task-program unnecessary during disc input/output transfer operations.

D. System Service Features

In addition to input/output transfers, the monitor system provides several other services which may be requested by user programs. These include the following:

- Arithmetic operations involving double-precision integer arguments
- Specific program time delays with tenth-second resolution
- Creation and removal of named data files on the fixed-head disc, and input, output, and positioning operations involving these files
- Task-program termination
- Scheduling of one program by another

A detailed description of the operation of each of the component routines which comprise the monitor software system may be found in **Appendix B** of this document.

APPENDIX A

INVENTORY/CRITICALITY CONTROL
SOFTWARE TECHNICAL DESCRIPTION

APPENDIX A

INVENTORY/CRITICALITY CONTROLSOFTWARE TECHNICAL DESCRIPTION

The following sections describe the operation of each application program employed in the inventory/criticality control system. As pointed out in Section III of this document, the data base for these programs consists of six files maintained on the fixed head disc memory. Figures A-1 through A-6 detail the structure of each of these files. The content of each word of a typical file entry is shown; also the format which would be used if an entry were to be output under the control of a FORTRAN write statement is shown. (The format designation DI stands for double precision integer.)

<u>WORD</u>	<u>CONTENTS</u>	<u>FORMAT</u>
0	AUTHORIZED OPERATOR PAYROLL NUMBER	DI
1		

2 WORDS PER ENTRY,
55 ENTRIES PER SECTOR,
LENGTH = 1 SECTOR.

FILE NAME: "OPER"

FIGURE A-1. Operator File Entry

<u>WORD</u>	<u>CONTENTS</u>	<u>FORMAT</u>
0	LAST TRANSFER SERIAL NUMBER	I
1	NUMBER OF TRANSFERS CURRENTLY ACTIVE	I
2	LAST PU BATCH SERIAL NUMBER ASSIGNED	I
3	LAST UD BATCH SERIAL NUMBER ASSIGNED	I
4	LAST US BATCH SERIAL NUMBER ASSIGNED	I
5	LAST UE BATCH SERIAL NUMBER ASSIGNED	I
6	LAST FD BATCH SERIAL NUMBER ASSIGNED	I
7	LAST FS BATCH SERIAL NUMBER ASSIGNED	I
8	LAST FE BATCH SERIAL NUMBER ASSIGNED	I
9	NUMBER OF ENTRIES NOW IN THE "TLOG" FILE	I

10 WORDS PER ENTRY
1 ENTRY PER SECTOR
LENGTH = 1 SECTOR

FILE NAME: "PARM"

FIGURE A-2. Parameter File

<u>WORD</u>	<u>CONTENTS</u>	<u>FORMAT</u>
0	BATCH TYPE NUMBER*	I
1	BATCH IDENTIFICATION NUMBER	4A2
2		
3		
4		
5	WEIGHT % PUO ₂ IN BATCH TIMES 100	I
6	WEIGHT % PU IN PUO ₂ TIMES 100	I
7	WEIGHT % UO ₂ IN BATCH TIMES 100	I
8	WEIGHT % U IN UO ₂ TIMES 100	I
9	WEIGHT % U ₂₃₅ IN U TIMES 100	I
10	WEIGHT % MODERATOR IN BATCH TIMES 100	I

*BATCH TYPE NUMBERS ARE AS FOLLOWS:

1 = PU	PLUTONIUM
2 = UD	DEPLETED URANIUM
3 = U S	NATURAL URANIUM
4 = UE	ENRICHED URANIUM
5 = FD	PLUTONIUM + DEPLETED URANIUM
6 = FS	PLUTONIUM + NATURAL URANIUM
7 = FE	PLUTONIUM + ENRICHED URANIUM

11 WORDS PER ENTRY
 1 ENTRY PER SECTOR
 LENGTH = 100 SECTORS
 FILE NAME: "BTCH"

FIGURE A-3. Batch File Entry

<u>WORD</u>		<u>FORMAT</u>
0	HOOD IDENTIFICATION NUMBER	2A2
1		
2	NET WEIGHT SAFETY LIMIT IN CENTIGRAMS	D I
3		
4	MODERATOR WEIGHT SAFETY LIMIT IN CENTIGRAMS	D I
5		
6	TOTAL NET WEIGHT PRESENT IN CENTIGRAMS	D I
7		
8	TOTAL MODERATOR WEIGHT PRESENT IN CENTIGRAMS	D I
9		
10	HOLDUP MATERIAL WEIGHT PRESENT IN CENTIGRAMS	D I
11		
2	SERIAL # OF TRANSFER IN PROGRESS TO TH HOOD*	I
13	NUMBER OF BATCHES NOW IN THIS HOOD	I
14		
15	BATCH IDENTIFICATION NUMBER	4A2
16		
17		
18	NET WEIGHT OF BATCH IN CENTIGRAMS	D I
19		
20	WEIGHT OF MODERATOR IN BATCH IN CENTIGRAMS	D I
21		
22	(ROOM FOR UP TO 12 BATCHES IN HOOD)	

*IF NO TRANSFER IS IN PROGRESS TO THE HOOD,
WORD 12 IS ZERO.

14-111 WORDS PER ENTRY,
1 ENTRY PER SECTOR,
LENGTH = 32 SECTORS.

FILE NAME: "HOOD"

FIGURE A-4. Hood File Entry

<u>WORD</u>	<u>CONTENTS</u>	<u>FORMAT</u>
0	TRANSFER SERIAL NUMBER	I
1	ORIGIN HOOD IDENTIFICATION NUMBER	2A2
2		
3	DESTINATION HOOD IDENTIFICATION NUMBER	2A2
4		
5		
6	MATERIAL BATCH IDENTIFICATION NUMBER	4A2
7		
8		
9	MATERIAL WEIGHT IN CENTIGRAMS	D I
10		
11	TIME OF DAY WHEN TRANSFER ORIGINATED	I
12		
13		
14	DATE TRANSFER ORIGINATED	4A2
15		
16	FIRST OPERATOR PAYROLL NUMBER	D I
17		
18	SECOND OPERATOR PAYROLL NUMBER	D I
19		

20 WORDS PER ENTRY,
10 ENTRIES PER 2 SECTORS,
LENGTH = 2 SECTORS.

FILE NAME: "TRAN"

FIGURE A-5. Active Transfer File Entry

<u>WORD</u>		<u>FORMAT</u>
0	TRANSFER SERIAL NUMBER	I
1	MATERIAL BATCH IDENTIFICATION NUMBER	4A2
2		
3		
4		
5	MATERIAL WEIGHT IN CENTIGRAMS	D I
6	DATE OF TRANSFER	4A2
7		
8		
9		
10	ORIGIN HOOD IDENTIFICATION NUMBER	2A2
11		
12	FIRST ORIGIN OPERATOR PAYROLL NUMBER	D I
13		
14	SECOND ORIGIN OPERATOR PAYROLL NUMBER	D I
15		
16	TIME OF DAY WHEN TRANSFER ORIGINATED	I
17		
18	DESTINATION HOOD IDENTIFICATION NUMBER	2A2
19	FIRST DESTINATION OPERATOR PAYROLL NUMBER	D I
20		
21	SECOND DESTINATION OPERATOR PAYROLL NUMBER	D I
22		
23	TIME OF DAY WHEN TRANSFER WAS COMPLETED	I
24		

25 WORDS PER ENTRY,
 4 ENTRIES PER SECTOR.
 LENGTH = 25 SECTORS.

FILENAME: "TLOG"

FIGURE A-6: Transfer Log File Entry.

Al, ar Origin Program

This program is initiated when an operator depresses function key No. 3 on a remote terminal. Upon initiation, the program reads the PARM file into core memory. It examines the number of transfers which are currently in progress. If ten transfers are in progress, a new transfer is not allowed and the operator is informed of this situation. No further dialogue is permitted and the operator must reinitialize the transfer origin program at some later time after at least one of the active transfers in the system has been completed. If less than ten transfers are active when this program is initiated, the program then checks to see if a new transfer would exceed the 100 entry capability of the transfer log file. If this is the case, the operator is informed of the condition and the program terminates.

If neither of the above situations is in effect, the operator is queried for an authorized payroll number. Upon response, the program reads the OPER file into core and attempts to locate the payroll number that was just typed in the contents of this file. If the payroll number cannot be found, the program assumes the operator operating the terminal is not authorized to transfer material and a message is output to this effect followed by program termination. If the payroll number is found in the file, the terminal then queries for a second payroll number which is again checked against the contents of the OPER file for proper authorization. If the second payroll number is also authorized, still another check is made to see if the operator has entered the same authorized payroll number twice, and if he has, he is requested to input a different authorized payroll number before proceeding.

Following successful input of two authorized payroll numbers, the operator is asked to identify the origin and destination stations for the transfer which is pending. Following the operator's response, the program begins reading the HOOD file, one entry at a time, searching for the hood that the operator specified as the origin for the transfer. If the designated origin is not found in the HOOD file, the program checks to see if the operator has specified an external origin. The operator does this by

entering a single digit for his location designator indicating that he wishes to transfer material into the fuel fabrication system from some external, uncontrolled point. If this is not the case, an error condition exists since the operator has specified an undefined origin point. In this case the operator is re-queried for point of origin and destination. If the origin is external, or if the origin hood specified by the operator is found in the HOOD file, the HOOD file is rewound and a second search is made for the designated destination point. This search is done in precisely the same manner as was the previous search for the origin point. If the operator has specified an external destination, indicating that he wishes to remove material from the fuel fabrication facility to some external uncontrolled point, no further validity checking of his proposed transfer to the destination is performed. If the destination indicated is not external, the contents of the HOOD file entry for the designated destination point are examined, and if that hood already contains 12 different batches of material, then the transfer is not allowed and the operator is informed of the error condition. Control then transfers to the error processing portion of the program. If less than 12 batches are present in the destination hood, a second test is made to see if another transfer is already in progress to that point. If so, the operator is informed of this situation and control passes to the error processing routine.

If neither of the above situations are in effect, or if the destinationpoint is external, the operator is then queried for the identification number of the material batch which he wishes to transfer. Following his response, the program searches the BTCH file for the indicated material. If the designated batch is not located in this file, the operator is informed of the situation and is asked to reenter the batch identification. If the indicated batch is valid and if the origin point is not external, then the origin hood file entry is examined to determine if the designated batch is indeed present at the origin station. If not, the operator is informed of the situation and asked to retype the batch identification.

If the program determines that the designated batch is present in the origin hood or if the origin is external, the operator is next queried for the weight of the material which he wishes to transfer. Following his response, the weight of the moderator present in the material about to be transferred is computed from the weight percent composition of the designated batch as indicated by the BTCH file. Then, if the destination point is not external, this moderator weight is added to the weight of the moderator already present in the destination hood and the sum is compared to the moderator criticality safety limit for the hood. If the limit is exceeded, the operator is informed of the condition and the program branches to the error processing routine. If it is determined that the proposed transfer would not violate the moderator criticality safety limit for the destination station, the net weight is added to the net weight of the material already in the destination station and this sum is compared against its corresponding criticality safety limit. Again, if the limit would be exceeded by the transfer, the operator is so informed and the error routine is entered.

If the destination is external or if neither the moderator nor net weight criticality safety limit would be exceeded at the destination station and if the origin is not external the program calculates the amount of the batch being removed from the origin station that would remain in that station if the transfer were to proceed. If the absolute value of the result of this subtraction is less than 10 grams it is assumed that the operator is removing the entire batch from the origin station. Therefore, the sub-entry for this batch is removed from the hood file entry for the origin station. In this case, any small difference, positive or negative, between the weight of the material being removed and the original content of the hood is appended to the holdup material total for the origin hood. However, this total is not allowed to become negative. If the proposed transfer would leave more than 10 grams of the batch being transferred in the origin hood, then the batch is not removed from the hood file entry but instead the net weight and moderator weight for the batch in the origin hood are reduced by the

proper amount. If, however, the net weight requested for transfer exceeds by 10 grams or more the weight of the batch in the origin hood, then the operator is informed that an insufficient quantity of that batch is present and the program returns to the point at which the operator is queried for batch identification.

If a sufficient quantity of the material is present in the origin station, then all the proper conditions for the transfer are in effect and the transfer is allowed. The program reduces the total net and moderator weights for the origin station and writes the updated data for the origin station onto the hood file. Then, if the destination is external, the operator is informed that the transfer has been authorized and the program terminates. If the destination is not external, the program reads the PARM file from disc, updates the transfer serial number, increments the current number of active transfers, and writes the PARM file back on to the disc. It then informs the operator that the transfer has been authorized and issues the transfer serial number just obtained for this transfer. The program then reads the TRAN file, updates it with the information concerning the transfer just authorized, and writes it back to the disc. Finally, the HOOD file entry for the destination station is read from the disc and the transfer serial number just issued is entered into the HOOD file entry indicating that a transfer is now in progress to this station. The HOOD file entry is then written back to the disc and the program terminates.

If an error condition is detected, for example, if a pending transfer would violate the criticality safety limit, an error routine is entered which first tells the operator via his remote terminal to push the reset key on that remote terminal thereby terminating the transfer origin program and acknowledging that the error has occurred. The program then goes into a loop, flashing the red function light on the remote terminal off and on at tenth second intervals for a period of about 10 seconds. If the program has not been aborted by pressing the reset key at the end of this 10 second interval, an alarm is typed on the console teletypewriter at the central computer facility notifying supervisory personnel

FIGURE A-7: Transfer Origin Program

FIGURE A-7 (continued)

FIGURE A-7 (continued)

FIGURE A-7 (continued)

FIGURE A-7 (continued)

FIGURE A-7 (continued)

that an error has ~~been~~ detected and ~~has~~ not been acknowledged by the operator. ~~The~~ program then repeats ~~the~~ entire error sequence as described above and continues to repeat this sequence ~~until someone acknowledges~~ the error condition by pushing the reset key on ~~the remote~~ teletype thereby aborting the transfer origin **program.**

A2, Transfer Termination Program

This routine, which is flow charted in Figure A8, is initiated each time function key No. 4 on a remote terminal is depressed. It begins execution by reading the parameter file from fixed head disc into core and checking to see if any transfers are currently in progress. If no transfers are in progress, then obviously no transfer can be terminated. The operator is so informed and the program terminates. If one or more transfers are in progress the program queries the operator for two different authorized payroll numbers reading the operator file and checking for authorization for the payroll numbers in the same way that the transfer origin program described above operates. Next, the operator is asked to input the serial number of the transfer he wishes to terminate. Following his response, the program reads the transfer file into core memory and searches this file for the transfer serial number that the operator has just typed. If this number is not to be found, the operator is so informed and asked to reenter the transfer serial number.

If the transfer serial number is located in the transfer file, the contents of the associated transfer file entry are saved in temporary storage. The operator is then queried for the identification of the destination hood for the transfer he is terminating. His response is then checked against the contents of the transfer file to determine if the operator's destination agrees with the destination that was given at the origin of the transfer being terminated. If not, the operator is informed that he is in the wrong place and control passes to an error processing routine to be described below.

If the operator's response to the previous question is in agreement with that input by the operator at the origin of the transfer, the program then asks for the weight of the material to be entered into the destination hood. The operator's response to this question is compared with the transfer file entry to determine if the value just input by the operator agrees with the weight that was input by the operator at the origin of the transfer. If there is more than 10 grams difference between the weight input at the origin and at the destination, the operator is so informed and the error processing routine is entered.

FIGURE A-8. Transfer Termination Program

FIGURE A-8 (continued)

FIGURE A-8 (continued)

FIGURE A-8 (continued)

If the weights are in agreement, then all the necessary conditions for proper termination of the transfer have been met. In this case the program proceeds to search the hood file for the destination hood. Having found it, the program will backspace the hood file one sector in preparation for an update of that file at a later time. The program then performs a similar search of the batch file, searching for the batch number of the material being entered into the destination hood. As soon as the proper batch file entry has been input to core memory the program returns to the hood file entry and updates the net weight figure for the destination hood. Then, using the composition data from the batch file entry just input, the program computes the weight of the moderator being entered into the destination station and updates this figure in the hood file entry. The program then zeroes the transfer number word in the hood file entry indicating that a transfer is no longer in progress to the destination hood.

The hood file entry is then searched to determine whether or not a portion of the batch being entered already resides in that hood. If so, the batch weight and the weight of the moderator in the batch are simply added to the corresponding figures already present in the hood entry. If not, the batch identification weight and moderator weight are appended to the hood file entry and the number of batches now present in the hood is incremented. Then, in either case, the updated hood file entry is written back on to the disc. Following this, the parameter file is updated by decrementing the number of active transfers incrementing the number of transfer log entries and writing the parameter file on to the disc. The entry in the active transfer file associated with the transfer being terminated is then removed from the contents of this file by a pushup process. Next, the program searches the disc for the end of the transfer log file. When it locates the end of this file it appends the necessary data to make a new entry for the transfer just terminated and writes the updated transfer log file entry back onto the disc. The operator is then informed that the transfer is complete and the program terminates.

If an error condition is detected by the aforementioned program, it transfers control to an error processing routine which first sets up an

output message for the console teletype containing pertinent information regarding the error condition which will be of use to supervisory personnel. The program then enters a loop which is identical to the error processing routine of the transfer origin program described above. This routine tells the operator to press the reset key on his remote teletype in order to acknowledge the error condition. The program then flashes the red light on the remote teletype off and on for a ten second period. It then outputs an alarm message for supervisory personnel on the console teletype in the central computer facility and repeats the above process until someone presses the reset key on the remote teletype indicating that the error condition has been acknowledged and aborting the transfer termination program.

A3, Material Mixing Program

The routine flowcharted in Figure A-9 is initiated from a remote terminal or the console teletype by the control command MIX. The routine begins execution by asking the operator for the identification number of the hood in which the material combination operation is to be performed. Following the operator's response the program searches the hood file on disc for the corresponding entry. If no such entry is found, the operator is informed of the error condition and the program terminates. If an entry is found in the hood file, the program checks to see if that hood already contains 12 items. If so, the capacity of the hood file entry would not allow another item; the operator is so informed and the program terminates.

If the hood contains less than 12 items, the disc is backspaced one sector in preparation for an update of the hood file entry at a later point in the program. The program then proceeds to query the operator for two lot or batch identification numbers and the weight of each, which are to be combined to form a third batch. Following the operator's response, the program verifies that both items that the operator requested are present in the hood of interest and that a sufficient quantity of each exists in the hood to allow the proposed mixture to take place. If any of the above conditions are not met, the operator is so informed and the program is terminated.

Otherwise, the program will search the batch file, inputting the two entries corresponding to the component lots or batches. The weight percent composition of the two components along with their respective weights are then passed to subroutine MIXER which calculates the composition and weight of the resultant batch. This subroutine, flowcharted in Figure A-10, is described below. Next, the program calls subroutine BATCHN, flowcharted in Figure A-11, which calculates the material type designation for the new batch. When this subroutine returns, the parameter file is read and the next serial number for the material type determined by subroutine BATCHN is obtained. The updated parameter file is then written back onto the disc.

FIGURE A-9. Material Mixing Program

FIGURE A-9 (continued)

FIGURE A-10. *Material Mixing Subroutine*

FIGURE A-11. Batch Type Subroutine

The program then searches for the end of the batch file and when it is found the composition parameters for the new batch are appended to the batch file. The net and moderator weights and batch identification for the new batch are then appended to the hood file entry of interest and the number of batches in the hood is incremented.

The program then examines the remaining weight for each of the component batches or lots and if this value for either batch is less than 10 grams, then it is assumed that the entire component batch has been depleted and the batch subentry is removed from the hood entry. The updated hood file entry is then written onto the disc, the operator is informed of the new batch identification number, and the program terminates.

Subroutine MIXER, flowcharted in Figure A-10, is a FORTRAN coded subroutine which is called by the material mixing program to determine the composition and weight of the batch formed by the combination of the two component batches. The calling program passes as parameters to the subroutine the weights of the component batches and the percent composition of each as indicated by the entries in the batch file. Also passed are the locations where the corresponding values for the new batch are to be returned. The subroutine converts the single and double precision integer values passed to it as arguments to floating point format, performs the necessary multiplications of weights by weight percents to determine resultant weights and compositions, converts the results to integer format, and returns to the calling program.

Subroutine BATCHN, flowcharted in Figure A-11, is a FORTRAN coded subroutine used to determine the material type of the material formed by a combination of any two types of materials. The subroutine receives four arguments which are the type numbers for the two component batches and the return locations for the new batch type number and alphabetic designation. The program determines the type of the new batch based on the following rules:

Plutonium combined with uranium will produce a mixed oxide batch

Enriched uranium combined with anything else will produce an enriched batch

- Depleted uranium combined with plutonium or natural uranium will produce a depleted batch
- If the two component materials are of identical type, then of course the resultant material will also be of this type.

A4, Sub-Batch Creation Program

This program is initiated by the control command SUBBCH typed on a **remote** or console teletype. The program begins execution by querying the operator for the identification of the hood in which the sub-batch is to be created from a batch resident in that hood. Following the operator's response, the program searches the hood file for the hood of interest. If the hood is not found, the operator is informed of the error condition and the program terminates. If the hood is found, its contents are checked and if that hood already contains 12 batches, sub-batches or lots, then the hood file entry cannot hold **another** item, the operator is so **informed** and the program terminates.

Otherwise, the operator queried for the identification **number** of **the** batch **from** which the sub-batch is to be formed. The program checks the **format** of the operator's response and if it is improper the operator is asked to use the proper **format** and is again queried for the batch identification. If he makes the proper response, the program searches the contents of the hood entry read in previously for the batch identified by the operator. If it is not to be found, the operator is **informed** that that batch is not in the hood and the program terminates.

Otherwise, the program proceeds to search the batch file on disc looking for the batch of interest. **When** it has been found, the operator is queried for the weight of the sub-batch which is to be **formed**. His response is subtracted from the weight of the batch that exists in the hood. The difference is saved in a temporary location and the absolute value of the difference is taken. If this difference is less **than** 10 grams, then it is assumed that the entire batch is to be converted to a **sub-batch**. Therefore, any difference is added to the hood's total holdup material and the program proceeds. However, if the difference is more **than** 10 grams, the sign of the difference is checked and if it is negative **the** operator is informed an insufficient quantity of that batch **exists** in **the** hood to form the sub-batch which he desires. He is then asked to input a new weight for the sub-batch. If the difference is **positive** and greater than 10 gram, then the difference is stored as the new weight of

FIGURE A-12. Sub-Batch Creation Program

Figure A-12 (continued)

FIGURE A-12 (continued)

the original batch in the hood entry. A new moderator weight for this amount is computed and stored in the entry and the program then moves the hood entry index to the end of the batch list. Then, in any case, the program scans the batch file counting the number of sub-batches of the batch of interest which are already in existence. When the program finds the end of the batch file, it checks the counter and if this counter is zero, that means that no sub-batches for the batch of interest previously exists. Therefore, the sub-batch number for the sub-batch being created is set to unity. If the counter is not zero, it is checked to see if it is equal to nine. If so, then nine sub-batches exist for this batch and no further sub-batch creation is possible since the sub-batch designation consists of the single digit 1 through 9. The operator is informed of this fact and the program terminates. If the counter is not zero and not nine, then the sub-batch number for the sub-batch being created is set equal to the counter, the batch file is back-spaced one sector and the new sub-batch information is written onto the batch file.

The program then computes the moderator weight of the sub-batch in the hood and enters the sub-batch identification weight and moderator weight into the hood entry at the index point previously determined. The hood file is then written onto the disc, the operator is informed that the process is complete, and the program terminates.

A5, Binder Addition and Removal

The program flowcharted in Figure A-13 is initiated whenever the control message BINDER is input to a console or remote teletype. The program begins by asking the operator to type A if he wishes to add binder to a batch in the hcod or R if he wishes to remove binder. If he responds other than A or R the question is repeated. If he responds A indicating binder is to be added, a parameter called "sign" is set equal to unity. If the operator's response is R indicating binder removal, "sign" is set to -1.

The operator is then queried for the identification number of the hood where the transaction is to take place. Following his response he is queried for the batch identification number of the batch in that hood to which binder is to be added. The program then searches the hood file for the hood of interest. If not found, the operator is so informed and the program terminates. If the hood is found, the program then searches the batch file for the indicated batch. Again, if the item is not found, the operator is informed of its absence and the program terminates.

When the batch is found in the batch file, the hood file entry previously read is searched for the batch of interest and if not found, the operator is informed that the batch is not present in the hood and the program terminates. If the batch is present in the hood, the operator is queried for the weight of the binder to be added to the batch. His response is multiplied by the "sign" parameter so that if binder is to be removed the quantity is converted to a negative number. The result is then added to the total weight of moderator present in the hcod. The result of this calculation is compared with the moderator criticality safety limit for that hood and if the limit would be exceeded by the proposed transaction, the operator is informed of the error condition and the program terminates.

If the moderator limit would not be exceeded, then the total moderator weight for the hood is adjusted by the proper amount but is not allowed to go negative. The binder weight being input is then added to the

FIGURE A-13. Binder Addition and Removal Program

FIGURE A-13 (continued)

~~total~~ net weight of the ~~material~~ present in the hood. Following this, the program updates the net and ~~moderator~~ weight for the batch's ~~entry~~ in the hood of interest. Then ~~new batch composition parameters~~ for the batch are calculated. The program then updates the appropriate files by ~~writing~~ the ~~hood~~ file ~~entry~~ onto the ~~hood~~ file and the ~~batch~~ file entry onto the batch file. It informs the ~~operator~~ that the process is ~~complete~~ and terminates.

A6, Batch Status Type Out

This program, which is flowcharted in Figure A-14, is **initiated** by the control **command** **BATCH** typed on a console or a **remote** teletype. It **begins execution** querying the operator for the batch identification number of interest. Following his response, the program searches the batch file for the indicated batch. If the batch is not found, the operator is **in-**
fo& of its nonexistence and the program **terminates**.

When the program locates the batch, it types out the weight percents contained in the batch file **entry**. The program then sets a flag $T = 0$ and enters a loop reading the hood file one sector at a time. Each time a hood file entry is brought into core its contents are scanned for the batch of interest. If the batch is found to be in the hood, the program checks flag T and if it is still 0 it is first set to -1 and a header is **typed** out. The program then types out the identification number of the hood that contains the batch of interest. In any case, the program then returns to read the next sector of the hood file. The program continues in **this** manner reading hood file entries, searching them for the **batch** of interest, and if **they** contain that batch, typing their identification numbers.

When the program finds the end of the hood file it again checks flag T , and if this flag is still 0, this means that the batch of interest **has** not been found to be present in any hood in the system and the operator is so **informed**. If flag T has been set, the program skips this type out. In either **case** the program **terminates at this** point.

FIGURE A-14: Batch Status Typeout Program

A7, Hood Status Type Out

This program, flowcharted in Figure A-15, is initiated when an operator types the control message HOOD on a console or remote teletype. It first queries the operator for the identification number of the hood of interest. Then it searches the hood file for this hood. If the hood is not found, the operator is informed of this condition and the program terminates. When the hood is found the entry is checked to see if any batches are present in the hood. If not, the operator is informed that the hood is empty and the program terminates.

If one or more items are present in the hood, a counter is set to the negative of this number. The program then enters a loop, typing out the batch identification number and weight for each of the items in the hood, incrementing the counter following each type-out operation. When the counter has incremented to zero, the program terminates.

FIGURE A-15: Hood Status Typeout Program

A8, Criticality Control System Maintenance Program

This FORTRAN coded program is flowcharted in Figure A-16. Since it is a privileged background program, it can only be initiated by an operator typing the control message CRITIC on the console teletype in the central computer facility and then only if the privileged operation enable keylock switch is in the ON position.

The program begins execution by asking the operator for the desired function that is to be performed. The operator responds by typing a function code of from two to four characters. The program then compares the operator's response with an alphanumeric array containing each of the valid function codes. If it is determined that the operator has entered an invalid function code, he is asked to repeat his input. A computed GO TO statement then causes a program transfer to a section of the program which will perform the indicated function.

The control message ADDO indicates the operator wishes to add a new payroll number to the authorized payroll number file on the fixed head disc. Upon receipt of this directive the program queries the operator for the payroll number. Following his response it reads the operator file into core, appends the new payroll number, and rewrites the operator file. It then returns to ask for the next function.

The control message DELO indicates to the system that the operator wishes to delete a payroll number from the operator file. Following the input of this directive, the program queries the operator for the payroll number, reads the operator number into core, and searches for the payroll number to be removed. It removes the payroll number by a compressing operation on the file. When this is complete, the file is rewritten on the disc and the program requests the next function.

The directive LISO indicates that the operator wishes a listing of the contents of the operator file to be output to the line printer. The program responds by printing a heading, the current time, and the current date on the printer, reading the operator file and then entering a loop in which operator payroll numbers are obtained from the file contents and

(Offpage connectors give associated statement numbers)

FIGURE A-16. Criticality Control System Maintenance Program

FIGURE A-16 (continued)

FIGURE A-16 (continued)

FIGURE A-16 (continued)

FIGURE A-16 (continued)

FIGURE A-16 (continued)

FIGURE A-16 (continued)

listed on the printer until the *end* of file is encountered. The program then requests the next function.

The directive ADDH indicates that the operator wishes to add a hood to the hood file and specify criticality safety limits for that hood. The program responds by *querying* the operator first for the new identification number of the hood to be added, then for the criticality safety Limit weights for net weight and *moderator* in the hood. The program then searches the *hood* file until the end of file is encountered. Following this it appends the new hood identification and safety limits to the *end* of the hood file and writes the new entry back onto the disc. The program then requests the next function.

The directive DELH indicates that the operator wishes to *remove* a hood from the hood file. The operator is again queried for the identification number of the *hood* to be *removed*. Following his response, a flag II is set equal to zero. The program then enters a *loop* reading, one sector at a time, the *hood* file, looking for the hood of interest. *When* it finds the hood, it sets the flag $II = 1$. Following *this*, each time a sector is read *from* the hood file it is written back onto the file on the previous sector position, thus *removing* the indicated hood *from* the file. The process *continues* until the end of file has been copied whereupon the *program* requests that the operator input the next function.

The directive LISH indicates the operator wishes a listing of each hood in the system containing identification number and criticality safety *limits*. The program *first* outputs a heading on the *line* printer. It then reads the hood file, one *sector* at a time, and outputs the required *information* to the line printer. This process continues until an *end* of file is encountered on the hood file *whereupon* the program requests input of the next function.

The directive LISB causes the program to transfer control to subroutine BLIST for the purpose of listing the contents of the batch file. Subroutine BLIST begins by printing a heading, the *time* and the date on the line printer. It then enters a *major loop* setting a counter equal to 1 and initializing the batch file. The program then proceeds to scan the

batch file ~~seven~~ consecutive times, incrementing the counter on each scan. During each scan each entry in batch file encountered is checked to see if its type number is equal to the value of the counter. If ~~so~~, the parameters for that batch are listed. If not, that batch is skipped in the current pass of the batch file. This process results in a listing of the entire contents of the batch file ordered by batch type. When the process is complete, subroutine BLIST ~~returns~~ control to the main program which requests input of the ~~next~~ function code.

The directive CI causes the program to transfer control to subroutine INVENT for the purpose of a complete listing of the contents of the hood file. The subroutine ~~begins~~ by printing a heading, current time and current date on the line printer. It then enters a loop reading the contents of the hood file one sector at a time and outputting the hood's identification number, total net weight present, moderator weight present, and holdup material present. If any batches are present in the hood the batch file entry corresponding to the batch identification in ~~that~~ hood is input and the program computes the amount of each component of each batch that is present in the hood and outputs its value to the line printer. This process continues for each hood in the system and when the end of file on the hood file has been encountered the subroutine returns control to the main program which requests the next function.

The directive CLEN indicates to the system that a hood has been cleaned and all variable information in the hood file entry for the hood other than criticality safety limits are to be set to zero. The program queries the operator for the identification of the hood to be cleaned. It then reads the corresponding sector in the hood file, sets the appropriate values in this entry to zero, rewrites the entry onto the hood file, and requests the next function.

The directive CORR is used when the operator wishes to correct the contents of a hood by adding a batch of material to that hood. The program queries the operator for the hood identification, the batch identification and the weight to be entered. The program then searches the hood file for the hood of interest and checks the weight which the operator indicates

should be entered against the criticality safety limits of that hood. If the criticality safety limit would be exceeded by the operation, the operator is so informed and the program does not allow the operation to take place. If the limit would not be exceeded, the program inputs the batch file entry corresponding to the batch of interest and from *the* values in this entry computes the new hood contents after the proposed modification. It updates the hood file entry and writes the updated entry back onto the hood file. The next function is then requested.

The directive PURG indicates to the system that the operator wishes to purge the contents of the transfer log file. This is accomplished by reading the parameter file, setting the number of transfer log entries in the parameter file to zero, and rewriting the parameter file.

The directive DELB indicates that the operator wishes to delete a batch entry from the batch file. The operator is first queried for batch identification number to be deleted. The program then searches *the* hood file for hoods containing this batch, and if one is found, the operator is informed that the batch should not be deleted because it physically exists in a hood. If the batch is not found in any hood in the system, then the program reads the batch file until the batch to be removed is found. It raves it from the batch file by a compressing process following which the program requests the next function.

The directive TLOG indicates that the entire contents of the transfer log file are to be output to the line printer. A heading is first printed followed by the current time and date. The program then enters a loop reading transfer log file sectors one at a time and outputting the four entries on each sector until the end of file is encountered, at which time the next function is requested.

The directive LOT indicates to the system that the operator wishes to define a new plutonium or uranium oxide material lot. This program is a separate external program from CRITIC and is therefore entered from CRITIC by means of the "chain" feature in the operating system. Upon execution, the lot program queries the operator for the identification number of the lot to be created. Following his response, the composition

of the new lot is requested and the operator replies by inputting the appropriate values of weight percents for the new lot. The program then searches for the end of the batch file and when it is found the new lot is appended as a new entry on the batch file. The program then returns to the main CRITIC program by means of the operating system "chain" feature at which point CRITIC requests the next operator function.

To release the console typewriter and terminate the CRITIC program the operator should type the directive `QUIT` which causes the CRITIC program to call system subroutine `EXIT`, thereby terminating.

APPENDIX B

MONITOR SYSTEM SOFTWARE
TECHNICAL DESCRIPTION

APPENDIX B**MONITOR SYSTEM SOFTWARE
TECHNICAL DESCRIPTION**

This appendix describes the operation of each of the routines which comprise the monitor software system. Flow charts are included for each of the routines and are intended to be used as a guide in understanding the overall operation of each routine. Since these routines are actually components of a closely integrated system, the reader may find that it is necessary to review the information on several related routines in order to familiarize himself with one particular routine. In addition, it may be necessary to refer from time to time to the system program listings in order to gain an efficient understanding of the detailed operation of a routine.

B1, Fixed Head Disc Input/Output

In order to facilitate the operation of the core/disc structure of this system, input/output routines for the fixed head disc are provided. Figure B-1 depicts the disc input routine RDSC. This routine is called from any system program and upon entry the hardware registers of the 810B contain the sector number, core address and number of words to be transferred. The routine begins by setting block transfer channel number 1 with the base address and word count. The sector number is then decoded to form track and sector disc addresses. A command is then given to the disc to transfer the required data into core. At this point, the program waits until the transfer operation is complete. A hardware test then determines whether or not a checksum error has occurred on the reading operation. If no error has occurred, the program returns to the calling routine. In the event of an error, five attempts are made to read the data properly. If the error persists, control transfers to the alarm routine in order that operating personnel can be informed of the failure.

Data is written from core onto the fixed head disc by means of the routine WDSC flowcharted in Figure B-2. Again, upon entry to this routine, the hardware registers contain the sector number, core base address and number of words to be transferred. Block transfer channel number 1 is set up with the base address and word count as in the read routine. Also, the track and sector addresses are formed from the sector number given. At this point, the program checks to see if the hardware write protect switches on the fixed head disc are set for the track requested. If so, operating personnel are informed of this condition via the alarm routine. If no track protect situation exists, then the data is output from core to disc via the block transfer channel. This program then waits for the transfer to complete. Following the write operation, the data just written is read so that any checksum error occurring during the write operation can be detected. If no error is found, control returns to the calling routine. If an error condition is detected, the program attempts five times to write the data properly and if the error persists, the alarm routine is entered.

FIGURE B-1. Fixed Head Disc Read Driver

FIGURE B-2. Fixed Head Disc Write Driver

B2, System Startup Routine

Each time the computer system is started, a core image copy of the entire monitor system is brought into core from fixed head disc and control transfers to the routine shown in Figure B-3. The purpose of this routine is to initialize the system in preparation for normal operation. The startup routine's first task is to read the directories for the program area and user data file area into core memory, since the system assumes that these directories are core-resident at all times during system operation. Next, all entries in the task list are initialized to the dormant status. Following this, the status word of all input/output devices are cleared, thus indicating that all I/O devices are available for use. Any system flags used during the normal operation of the system are then set to an initial condition. Programmable input/output interrupts for various devices in the system are enabled and the typewriter in the central computer facility indicates to the operators that the system is ready for use. The startup routine then transfers control to the system monitor program.

FIGURE B-3. System Startup Routine

B3, System Monitor Program

The routine flowcharted in Figure B-4 is responsible for the direction of all major functions of the operating system. Control transfers here following the completion of any operation that the system performs. Upon entry, the monitor program enables input/output interrupts for all I/O devices in the system. A priority flag is then set such that the monitor program is in the foreground mode. Following this, an entry is obtained from the task list which is shown in Figure B-5. This task list is a nine-word table, where each word corresponds to one foreground task or to the background. Bit zero of each word is the active/dormant indicator for the corresponding task. If bit zero is set, the task is active, i.e., is in the process of performing some program function. If bit zero of the task list word is not set, then the task is dormant, i.e., is not currently associated with any operating program. Bit 1 of a task list entry is set to indicate that a task is active but is awaiting the completion of an input/output operation.

When the monitor obtains an entry from the task list it first checks to see if the entry obtained was for a background program. If so, and if the monitor is not in the background mode, it ignores this entry and resets the task list pointer to the top of the list. If the entry obtained is not for background program, then the monitor program checks to see if the task is dormant. If so, the next entry is obtained from the list. If the task list entry indicates the task is not dormant, it is then checked to see if the task is awaiting completion of an input/output operation, and if so, this entry is ignored and the next entry is obtained.

If the foreground task entry from the task list indicates that the task is neither dormant nor awaiting I/O completion, then the monitor passes control to the foreground executive program EXEC. The foreground executive program returns control to the monitor at this point if it determines that the task which the monitor has selected cannot be executed at this time. In this case the monitor obtains the next entry from the task list.

FIGURE B-4. System Monitor

WORD	0	1	15
0	A	W	ADDRESS OF TASK 0 STATUS BLOCK
1	A	W	ADDRESS OF TASK 1 STATUS BLOCK
2	A	W	ADDRESS OF TASK 2 STATUS BLOCK
3	A	W	ADDRESS OF TASK 3 STATUS BLOCK
4	A	W	ADDRESS OF TASK 4 STATUS BLOCK
5	A	W	ADDRESS OF TASK 5 STATUS BLOCK
6	A	W	ADDRESS OF TASK 6 STATUS BLOCK
7	A	W	ADDRESS OF TASK 7 STATUS BLOCK
8	A	W	ADDRESS OF BACKGROUND STATUS BLOCK

A = Active: If bit 0 is set, task is active,
if not, task is dormant

W = Waiting: If bit 1 is set, task is suspended
awaiting completion of an I/O
operation

FIGURE B-5. Task List (queue)

After the monitor has made a complete pass through the task list in the foreground mode and has determined that no foreground task can execute at this time, then it sets the priority flag to the background mode and makes another pass to the task list. At this time, when it encounters the entry for the background, it checks this entry to see if the background is dormant or awaiting an I/O operation. If neither of these conditions is the case, it checks to see if the background has been set idle by the operator. If not, the background executive program, BKEX, is entered. This program will return control to the monitor if it determines that the background cannot execute at this time. If the background is dormant, awaiting I/O, idle, or if the background executive program returns, the monitor program then enters a loop wherein the contents of the address set in the 810B console switches is displayed in the accumulator. This feature is effected by loading the accumulator with the contents of the indicated address and entering a wait loop such that the time the program remains in this loop is long with respect to the time it spends in other portions of the monitor program. When the monitor comes out of this loop, it reinitializes the task list pointer to the top of the list and repeats the entire procedure.

B4, Program Scheduler

The routine flowcharted in Figure B-6 is responsible for the initialization of any program which runs under control of this operating system. The scheduler is most often entered in response to a user request via a function key or teletype keyboard interrupt. The routine calling the scheduler passes to it the six-character name of the program which is to be executed. The scheduler's first function is to search the program directory, shown in Figure B-7, for the name of the requested program. If the name is not found in the directory the scheduler takes an error return to the calling program. If the program is found in the directory, the scheduler checks the program type. If the program is privileged, that is, if bit 2 of word zero of the directory entry is set, then the scheduler tests to see if the request for this program was made by a privileged program. Privileged programs can only be initiated and operated by an operator at the console teletype in the central computer facility and then only if the privileged operation enable key-lock switch is in the ON position. If a privileged program has been requested from an unprivileged terminal or program, then the error return is taken from the scheduler. Otherwise, the scheduler checks to see if the requested program is a background program. If so, it checks the task list to see if the background is currently dormant. If a background program has been requested and the background is not dormant, then a busy return is taken from the scheduler. If the program requested is a foreground program, the scheduler searches the task list looking for a dormant entry. If no dormant entry can be found in the task list, this means that all eight foreground tasks are currently associated with programs and the busy return is taken from the scheduler. Otherwise, the first dormant entry in the task list encountered by the scheduler is taken as the task entry for the program being scheduled. The scheduler then sets the selected task or background active by setting bit-0 of the associated entry in the task list. The task status block for the selected foreground program(or, in the case of a background program, the background status block) is then initialized for the desired program. Figures B-9 and B-11 depict foreground and background status blocks respectively. Following task status block initialization, the scheduler returns to the calling routine.

FIGURE B-6. Program Scheduler

C = In Core : If bit one is set, the program is in core.

B = Background: If bit two is set, the program is a background program. If not, it is a foreground program.

P = Privileged: If bit three is set, the program is privileged.

FIGURE B-7. Program Directory

B5, Executive Control Programs

The routine flowcharted in Figure B-8, the foreground executive program, is responsible for checking the condition of the current foreground task's status block to determine whether or not control should transfer to the program currently associated with that task. The foreground executive first enters the executive check subroutine described below. When this subroutine returns, the foreground executive checks to see if the current foreground task is active and if not it returns control to the system monitor. If the task is active, the foreground executive compares the current re-entry address in word 2 of the current task's status block with the starting address for foreground programs to determine if this is the initial entry of the current program. If this is the initial entry, subroutine SBIN is entered to save the 8K background area on disc if necessary. Following this, subroutine SHIN is entered to save the task header of any other task which may have been in core when the monitor was entered at the start of the current system cycle. A core status word is then set with the task number of the current task and subroutine LFP is employed to load the requested foreground program into core from disc. Once the program is in core, control transfers to the starting address of the program just loaded.

If the foreground executive determines that this is not the initial entry to the current foreground program then the task number of the foreground task now resident in core, if any, is obtained. If this is equal to the current task number this means that the program which is to now receive control has not been overlayed since the last major system cycle. In this case, no loading or restoring of the program is necessary and control passes to the program at the re-entry address given in word 2 of the foreground task status block. If, on the other hand, the task now in core is not equal to the current task, then subroutine SBIN is entered to save the background on disc in case background is core resident at this time. Subroutine SHIN is entered to save the header of any foreground task which may be in core and subroutine LFP is employed to load the current foreground program into core. Following this, the header of the current task

FIGURE B-8. Foreground Executive

WORD	0	1	6	7	15
0		PROGRAM BASE MAP		CURRENT PROGRAM NUMBER	
1				ABORT FLAG	
2				PROGRAM RE-ENTRY ADDRESS	
3				DISC SECTOR OF TASK HEADER SAVE AREA	
4				CURRENT BASE ADDRESS OF DYNAMIC DATA AREA	
5				SIZE OF CURRENT DYNAMIC DATA AREA	
6				DISC SECTOR OF DYNAMIC DATA SAVE AREA	
7				CORE ADDRESS OF FLAG FOR CONDITIONAL TASK RE-ACTIVATION	
8				TIME OF DAT TO RE-ACTIVATE TASK IF WAITING FOR A SPECIFIED TIME INTERVAL	
9				(TWO WORDS)	
10				CURRENT I/O DEVICE NUMBER	
11				CURRENT REMOTE TERMINAL I/O DEVICE NUMBER	
12				BIT MASK FOR CONDITION FLAG SPECIFIED BY ADDRESS IN WORD SEVEN	

FIGURE B-9. Foreground Task Status Block

is restored from fixed head disc, overlaying the first 111 locations of the program just loaded. The core status word is then set with the current task number and the program is re-entered at the entry address given in word 2 of the status block.

Figure B-10 is a flowchart of the background executive control routine which serves the same purpose for a background program as does the foreground executive routine, just described, for a foreground program. This routine also begins execution by entering the executive check subroutine EXC to determine if the background is active. If the background is not active, control returns to the system monitor. An active background causes the background executive to inquire whether or not the current background program is at its initial entry. If so, subroutine SHIN saves on disc the header of any foreground task which may now be in core. Then the data areas of any foreground tasks which may be executing are saved on their respective disc areas. Following this, the background executive checks the chain flag, a system flag which if set indicates that one background program link has requested that control pass to another background program link in the same program chain. If the chain flag is set, the executive clears it and then checks to see if the previous background program is still in core. If it is, or if the chain flag is not set, the requested program is loaded from the disc program file. If, however, the chain flag is set and background has been overlayed by a foreground program on a previous minor system cycle, then the 8K background area is restored from its save area on fixed head disc prior to loading the current program from the program file. The executive then clears the in-core status of any program which may have been overlayed by the previous loading operation. Then, the background program just loaded is set to the in-core status. This program status is maintained in word 0 of the program directory entry for each program. Following the update of the program status, the background program is entered at its initial entry point which is given in word 6 of the program directory entry.

FIGURE B-10. Background Executive

WORD	0	1	6	7	15
0		PROGRAM BASE MAP		CURRENT PROGRAM NUMBER	
1		ABORT FLAG			
2		PROGRAM RE-ENTRY ADDRESS			
3		DISC SECTOR OF 8K BACKGROUND SAVE AREA			
4		(NOT USED)			
5		BACKGROUND PROGRAM STARTING ADDRESS			
6		(NOT USED)			
7		CORE ADDRESS OF FLAG FOR CONDITIONAL TASK RE-ACTIVATION			
8		TIME OF DAY TO RE-ACTIVATE TASK IF WAITING FOR A SPECIFIED TIME INTERVAL (TWO WORDS)			
9					
10		CURRENT I/O DEVICE NUMBER			
11		CURRENT REMOTE TERMINAL I/O DEVICE NUMBER			
12		BIT MASK FOR CONDITION FLAG SPECIFIED BY ADDRESS IN WORD SEVEN			

FIGURE B-11. Background Status Block

If the background executive determines that the current entry to the background program is not the initial one, it checks to see if the background has remained in core from a previous system cycle and if so, it is re-entered at the address given in word 2 of the background status block. If the background is not in core then any data areas of foreground program are saved on disc, subroutine SHIN is employed to save the header of any foreground task which may be core resident and the background is restored from its 8K save area on disc. The in-core status of any program which may have been overlayed by this restoring of the background is then cleared and the in-core status of the background program being entered is set. The executive then restores the background program's environment and re-enters it at the address given in word 2 of the background status block.

Figure B-12 is a flowchart of the executive check routine which is used by both the foreground and background executives in determining whether or not a task or background program should be entered at the current time. Upon entry, the routine checks word 1 of the foreground or background status block to determine if an abort request has been filed for the program. If so, the abort routine is entered. If not, the program looks at word 7 of the status block to determine if a flag has been set for conditional reactivation of the task or background program. If word 7 is negative, then no such flag has been set and the task has not been deactivated on condition. If, however, word 7 is positive, then it is assumed that word 7 contains the address of a flag which, when masked with the contents of word 12 of the status block, will reactivate the previously suspended task providing the result of the logical masking operation is zero. If this test has been requested, and if the result of the masking operation is not zero, the executive check routine returns to the executive which called it. Otherwise, the current task is set active.

If the task has not been suspended awaiting the condition of a flag, the routine checks words 8 and 9 of the status block to determine if a time delay has been requested. If word 8 is negative, then no time delay is requested and the task is active. If, however, word 8 is positive,

FIGURE B-12. Executive Check Routine

then it is assumed that words 8 and 9 of the status block contain a double precision number equal to a time of day, as kept by the 60 Hz. real-time clock, at which the task is to be reactivated. If this is the case, the contents of words 8 and 9 are compared against the current time of day to determine if it is past time for the task to reactivate. If this time condition is not met, the executive check routine returns to the executive which called it. Otherwise the task is set active.

If the task is set active, the check routine determines whether or not the current task is a background program. If it is, then the starting address is obtained from word 6 of the associated program directory entry. If, however, the program is of the foreground type then the starting address is set equal to octal 20,000, since this is the standard starting address for all foreground programs. An initial entry flag is then set by subtracting the starting address just determined from the program re-entry address present in word 2 of the status block. Therefore, this flag's word will be zero if the program of interest is being entered for the first time. The executive check routine then increments its return location indicating that the task being checked is active and returns to the executive program which called it.

Figure B-13 depicts the executive subroutine SHIN which is used by both the foreground and background executive to save the header of any foreground task which is about to be overlayed. As pointed out in the body of this document, foreground programs are not saved and restored in their entirety when overlaying becomes necessary but rather only the first 111 core locations of the foreground program are saved and restored to and from the disc.

Upon entry, the SHIN routine obtains the task number of the task now in core. Several conditions are possible at this point. If core is empty, that is, if no program is currently core resident, or if the background is currently core resident or if the task that is core resident is the current task or if a task is in core that has previously been set dormant, then the SHIN routine takes no action and returns to the calling program. If, however, none of the above conditions are in effect, SHIN

FIGURE B-13. Save Header If Necessary

employs the disc write driver to write the first 111 words of the upper 8K core module onto the proper save area of fixed head disc.

Figure B-14 is a flowchart of executive subroutine SBIN. The purpose of this routine is to save the background core area if this area contains an active background program which is about to be overlayed by a foreground program. The status of the background is obtained from the last entry in the task list. If the background is dormant, no action is taken by this routine. If the background is not dormant, the task number of the last task which used core is obtained and if this is not the background then no saving of the background is necessary and no action is taken by the program. If however, it is determined that the background is in core and not dormant then the "in-core" status bit in the program directory entry for the background program now in core is cleared. Following this, the core status word is set to "empty" and the upper 8K module is written onto the disc background save area. SBIN then returns to the executive routine which called it.

The routine flowcharted in Figure B-15, LFP, is a subroutine used by the foreground executive to load foreground programs into core from disc prior to their execution. The program first checks the directory entry for the desired program to determine if it is already in core. If so, the current task's data area is read into core provided that this operation would not overflow the allotted space in core for program data. If this is the case, then LFP takes a busy return from the foreground executive and control returns directly to the monitor program. Otherwise control returns to the foreground executive routine. If the desired program is not in core then the program checks to see if any other program is core resident at the current time. If not, the "in-core" status bit in the program is loaded from the disc program file into core memory and its in-core status bit is set. LFP then branches to the point where the data area for the incoming program is loaded.

The executive routines save and restore the data areas of the foreground program by means of the routines illustrated in B-17 and B-18 which operate on the dynamic core directory depicted in Figure B-16. If

FIGURE B-14. Save Background If Necessary

FIGURE B-15. Load Foreground Program

WORD FIGURE B-16. *Dynamic Core Directory*

FIGURE B-17. Input Task's Data Area

a number of foreground programs which are currently active have requested the use of the dynamic core data area then the dynamic core queue in the dynamic core directory will contain a list of task numbers. At the top of this list will be the number of the task whose data have been core resident for the longest period of time. This task is termed the "eldest" task.

To input a task's data area from its save area on fixed head disc, the executive programs call subroutine DCIN. This routine begins execution by obtaining the required number of words from the task's status block. If not enough available space exists in the dynamic core area and no other tasks are using dynamic core at the time, then the task has requested use of the core area whose size exceeds the maximum capacity of the system and subroutine alarm is entered informing the operator and aborting the task. If tasks are present in the dynamic core area then the data area for the eldest task is saved on its allocated disc area. Subroutine DCRE is employed to remove the data area of the eldest task from core in order to create more available space. If there is still insufficient space for the incoming task's data, the above process is repeated until enough space is made available. When the program determines that there is enough room for the incoming task's data, the base address of the available core area is set into the task status block. If this is not the initial entry to the task, its data are then restored from disc. Then, the task number of this task is placed in the dynamic core queue as the youngest task in the queue. The dynamic core base address and word count for available space are then adjusted in the dynamic core directory and the program returns to the executive routine which called it.

Subroutine DCRE, illustrated in Figure B-18, is employed to remove the data area of a task from dynamic core. This is done when a task using dynamic core is terminated for any reason or when subroutine DCIN, just described, requires more space than is currently available in order to input another task's data. DCRE begins execution by updating the task's

FIGURE B-18. Remove Task's Data Area

status block to inform the system that the task being operated upon no longer has data which are core resident. It then finds this task's number in the dynamic core queue and if this is not the youngest task, it begins a compressing operation on the dynamic core area to remove the task's data area. It then compresses the dynamic core queue to remove this task's number from the queue. Then, in any case, the dynamic core base address and word count for the next available area are adjusted and the program returns to the routine which called it.

B6, Real-Time Clock and Calendar

Once every 60th of a second a hardware circuit in the computer which is connected to the AC power supply causes an interrupt on a high priority interrupt channel. As a result of this interrupt, the routine illustrated in Figure B-19 is entered. This routine increments a counter which has initially been set to a value of -6 so that the counter will overflow once every tenth of a second. When the counter overflows it is reset to its initial value and a tenth-second counter is incremented. This counter is maintained as a double precision integer word and if the clock has been properly initialized by operating personnel via the INTIME operation described elsewhere in this report, then this word represents the time of day in tenths of a second. After incrementing this counter, the routine checks to see if the counter's value corresponds to 24 hours. If so, this means that it is now midnight, and the calendar portion of the routine is entered, updating the current day of year, day of month, month and year as required. In any case, an interrupt return is then taken from the routine.

FIGURE B-19. 60 HZ Clock Interrupt Routine

B7, Program Suspension and Termination

As pointed out previously, the user of this system has the option at any point in his program of suspending his program for a given period of time or until some condition, as indicated by a flag word in core, is met. He also may terminate his program at any time or the program may be terminated by the system in the event of an error condition.

The routine flowcharted in Figure B-20, EXIT, is entered from the user's program when he desires one of the above operations. The program begins by disabling I/O interrupts temporarily so that it may proceed uninterrupted. (These interrupts will be re-enabled when the program enters the monitor.) EXIT then checks the form of the calling sequence used by the user program. If a final exit from a program has been requested, then routine ABT is entered. Otherwise, the program checks to see if the task should be deactivated until a specific condition is met as indicated by a flag word in core. If this is the case the specified flag location is obtained from the calling sequence and set into word 7 of the current task's status block. If neither of the above conditions are in effect, then the calling program has requested a time delay. The specified delay interval in tenths of a second is obtained from the calling sequence, added to the current time of day as kept by the real-time clock, and stored in words 8 and 9 of the current status block. Then, in any case, the return address from subroutine EXIT is stored in word 2 of the status block so that the executive may transfer control back to the user's program at the proper point when reactivation occurs. The contents of the hardware registers at the time EXIT was entered are then saved in the task's header area in the case of a foreground program, or in the background register save area for background program. Then control passes to the system monitor. Shown in Figure B-21 is a flow-chart of the routine responsible for terminating all programs operated under control of this system. This routine has two entry points: ALARM, which types a specified message on the console typewriter in the central computer facility prior to terminating the program; and ABT which is identical except for the fact that no such message is typed. The program

FIGURE B-20. EXIT Routine

FIGURE B-21. ABORT and ALARM Routines

disables I/O interrupts so that it may proceed uninterrupted until control passes to the monitor. It sets the current task to the dormant status by clearing bit 0 of the task's entry in the task list. The program then looks at the task's status block word 10 to determine if the task has used an I/O device prior to termination. If so, the device status of any device used by the terminating program is cleared and any hardware interrupt which may be associated with that device is disconnected. Then, in any case, the current remote terminal associated with the terminating program, as indicated by word 11 of the task's status block, is set to the "not busy" condition. Following this, if the terminating program has been using any of the data files on fixed head disc, these files are closed by clearing the status word of each associated file in the user disc file directory. Then any data area which may be in use by the terminating program is removed from the dynamic core area by means of subroutine DCRE described above in Section B5. Following this the task status block for the terminating program is initialized and control passes to the system monitor.

B8, Disc Data File Input/Output

The routines flowcharted in Figures B-23 through B-26 are entered from user programs in order that data files on fixed head disc may be created, deleted, made available for use, released to general availability, read, written and positioned. These programs operate on the user disc data file directory illustrated in Figure 22.

In order to create a new data file on fixed head disc, the user program enters subroutine NAME shown in Figure B-23. This routine checks the data file directory to see if the requested program name is already on file. If so, this routine takes no action; otherwise it checks to see if one more entry in the directory would exceed the directory's capacity. If this is the case, an alarm is output and the calling task is aborted. Otherwise, the number of words desired for the file being created is obtained from the calling sequence. This word count is converted to a number of disc sectors by dividing by 111, and the result is checked against entries in the directory to determine if enough disc space is available to create the desired file. If not, an alarm is typed out and the calling program is aborted. Otherwise, a new entry is made in the disc data file directory and the entry is initialized. The program then writes the updated directory back onto the disc and returns control to the user program.

In order to remove a data file from the system, the user program enters subroutine DNAM illustrated in Figure B-23. DNAM searches the data file directory for the indicated name. When it is found, it checks this entry's status word to determine whether or not the file mentioned is open for use by the task which requested deletion. If this is not the case, then the calling program is in error, an alarm is output and the program is aborted. Otherwise the DNAM program changes the four character file name for this entry to "DELE" so that the compress operation included in the file maintenance program described in the body of this report will recognize this entry and will remove it by compressing the file area and the directory. The updated directory is then written onto the disc and control passes back to the user program.

FIGURE B-23. Add/Delete User Data File

In order for a user program to access any of the named data files on the disc, it must first open each file it wishes to access. This feature is included in the system to prevent simultaneous access of a file by two concurrently operating user programs. In order to open a file, the user program enters subroutine OPEN illustrated in Figure B-24. This routine looks up the requested file in the data file directory and if examination of that entry's status word indicates that the file is currently open for a different task, then the requesting task is suspended and set to reactive on the condition that the requested file is made available. Control then passes to the system monitor. If the requested file is not open or is already opened by the requesting task, the file is rewound by setting the next available sector for data input/output to be equal to the first sector of the file. The file is then set open for the current task by setting bit 0 of the file status word and by setting the current task number into bits 11 through 16 of this status word. Control then passes back to the calling user program.

In order that a file opened by the procedure just described be closed and released to general availability, the user program enters subroutine CLOS illustrated in Figure B-24. This routine searches the directory for the indicated file name. When it is found the routine checks its status word to determine if the file is open for the requesting task. If not, then no action is taken; otherwise the file is rewound by setting the next sector for data input/output to be equal to the first sector of the file and the file is closed by setting the status word to 0. Control in either case will then return to the user program.

Data is read from and written to previously created data files on fixed head disc by means of the routine flowcharted in Figure B-25. Because the only difference between a read operation and a write operation is the call to the appropriate disc input/output routine, only one flowchart is depicted. The read and write routines are, however, two physically separate programs.

Subroutine READ (or WRIT) is entered from the user's program. The requested file for data input/output is found in the data file directory.

FIGURE B-24. Open/Close Disc Data File

FIGURE B-25. Read (Write) Disc Data File

If this file is not opened for the requesting task, an alarm is output and the task is aborted. Otherwise, the core base address and number of words to be transferred are obtained from the calling sequence. The next available sector for data input/output is obtained from word 3 of the data file directory entry for this file. This information is then passed to the proper disc input/output driver and the transfer of data is effected between core and disc. The number of words transferred are then divided by 111 to determine the number of disc sectors involved in the transfer and this amount is added to word 3 of the directory entry thus updating the current file position. Control then returns to the user program.

The user has the capability to advance or retard word 3 of the directory entry for any named file, thus effectively advancing or backspacing this file similar to the manner in which a magnetic tape may be advanced or backspaced a given number of records. Figure B-26 is a flowchart of the advance routine, and the backspace routine is identical except for the direction which the pointer is moved. Subroutine ADVN is entered from the user program and looks up the requested file in the data file directory. If the file is not open for the current task, an alarm is output and the task is aborted. Otherwise, the number of sectors to be advanced are obtained from the calling sequence and are added to word 3 of the directory entry for the proper file. Control then returns to the user program.

FIGURE B-26. Advance (Backspace) File Pointer

B9, Input/Output Initialization and Formatting

The routines flowcharted in Figure B-30 through B-36 operate in response to a user program request for data input/output. The format of this user request is depicted in Figure B-27. Throughout the operation of the routines described below, the status of all input/output devices is maintained in a table. A typical entry of this table is shown in Figure B-29. In addition, remote teletype terminals have a status table with one entry for each terminal as depicted in Figure B-28.

The first item of the input/output calling sequence as shown in Figure B-27 is a transfer of control from the user's program to the input/output scheduler flowcharted in Figure B-30. This program begins execution by disabling input/output interrupts so that input/output initialization then can proceed uninterrupted. It then saves hardware register contents in the header of the current task. Next the calling sequence is examined to determine whether or not the device code in word 1 is "CR" indicating current remote terminal I/O. If so, the device number to be used for input/output is obtained from word 11 of the task or background status block. If the I/O request is not for a remote teletype terminal, then the device code for the requested device is generated by looking up the device name in a logical equipment table which is simply a sequential list of the two character device names that are used by the system. Then, in either case, the device status table entry for the collected device is examined. If bit 0 of this entry is set, then the device is busy, that is, it is in use by another program, and subroutine EXIT is employed to deactivate the current task and set this task for reactivation when the selected device's entry in the status table indicates that the device is not busy. Control passes through subroutine EXIT to the monitor. If the device is not busy, the return address in word 2 of the current program task status block is set equal to word 2 of the I/O request. Then the device number obtained is set into the task status block word 10. Following this, control returns to the calling program at word 2 of the input/output request.

As can be seen from Figure B-27, word 2 of the input/output request calling sequence is a transfer of control to either the input or output

WORD

0	CALL TO I/O SCHEDULER
1	TWO CHARACTER DEVICE NAME
2	CALL TO INPUT OR OUTPUT INITIALIZER
3	CORE BASE ADDRESS OF DATA*
4	NUMBER OF 16-BIT WORDS TO BE TRANSFERRED*
5	ADDRESS OF FORMAT SPECIFICATION TABLE†
6	ERROR RETURN LOCATION
7	NORMAL RETURN LOCATION

**If word 4 = 0, word 3 contains the
code for a positioning function*

†If word 5 = 0, data is unformatted binary

*FIGURE B-27. Structure of User's
I/O Request*

*	TASK # OF TASK NOW USING REMOTE TERMINAL
---	--

**Bit 0 set if busy, not set if free*

FIGURE B-28. Function Key Status Table Entry

B		A	O	I	TASK # OF TASK NOW USING DEVICE
---	--	---	---	---	---------------------------------------

B: BIT 0 SET IF DEVICE IS BUSY
 A: BIT 2 SET FOR BINARY CLEAR FOR ASCII
 O: BIT 3 SET IF PERFORMING OUTPUT
 I: BIT 4 SET IF PERFORMING INPUT

FIGURE B-29. Device Status Table Entry

initializer routine. These routines are flowcharted in Figure B-31. If the input/output call is for an output operation, then the output initializer IOIO is entered. This routine picks up the base address for data transfer, the number of 16 bit computer words to be transferred, and the address of the format specification table from the calling sequence. It then transfers control to the particular formatter associated with the specified output device. A typical output formatter is flowcharted in Figure B-35, to be described later. When the formatter returns, IOIO sets the output bit, that is, bit 3 of the device status table entry for the selected device. Next, the busy bit is set into the devices status word indicating the device is now busy and if the output request indicates that binary data transfer is to take place, that is, by an absence of the address of the format specification table in word 5 of the calling sequence, then the binary bit in the status table entry is also set. Next, the character counter for the selected device is set to zero and finally, the input/output wait bit, i.e., bit 1 of the current task's entry in the monitors task list, is set to a 1 indicating that an input/output operation is in progress for the current task. Control then transfers to the system monitor.

If an input request is made, then word 2 of the I/O request will instead be a transfer of control to the input initializer routine IOII. In this case, I/O interrupts are again disabled and the status of the device connected to the current task is obtained from the entry in the device status table. If bit zero, the busy bit, is set, then this entry to IOII is the second entry for this request. This means that a previously initiated input request is now complete and that the device's core buffer contains one record of ASCII information. In this case, the base address, word count, and format specification table address parameters are obtained from the input request sequence and control passes to a particular input formatting routine for the specified device. Figure B-33 depicts a flowchart of a typical input formatter to be described later. When the selected formatter returns, the device status is cleared, the task's hardware registers which were previously saved in its header are restored, input/

FIGURE B-30. Input/Output Scheduler

FIGURE B-31. Input and Output Initializers

output interrupts are enabled and control returns to the user program at the return address specified by word 2 of the task's status block.

If, at the entry to routine IOII, the busy bit in the selected device's status table entry is not set, then this indicates that a first entry to IOII is being made. In this case, control transfers to routine III which picks up the base address, word count, and format specification table address parameters from the input/output calling sequence. These parameters are passed to a particular input initializing routine for the selected device. Figure B-32 is a flowchart of a typical input initializing routine and is described below. When the selected input initializing routine returns, then the return address for the program, i.e., word 2 of the task status block of the current task is set to be equal to word 2 of the input request. Note that this will cause control to transfer to IOII once again when the task is reactivated. However, when this occurs the busy bit in the selected device status table entry will be set and the second entry path described above will be taken. Following the setting of the return address, III will set the input and busy bits, i.e., bits 4 and 0 into the selected device status table entry. Then if word 5 of the input calling sequence specifies that binary information rather than formatted alphanumeric information is to be input, the binary bit, bit 2, of the status word is also set. Then, as in the case for output, the device's character counter is cleared, the input/output wait bit, bit 1 of the current task's task list entry, is set and control passes to the monitor routine.

Figure B-32 shows a flowchart of a typical input initializer sub-program. One such routine exists in the system for each input device. The purpose of these routines is to prepare the device and associated software for input of ASCII or binary information to the device's dedicated core memory buffer. These routines begin execution by examining word 4 of the input calling sequence to determine whether or not the user has requested execution of a special peripheral positioning function. As indicated in Figure B-27, if word 4 of the input request is 0, then word 3 contains a code number from 0 to 11 which specifies a particular

FIGURE B-32. Typical Input Initializer

positioning operation. Examples of positioning operations are rewinding magnetic tape, punching leader on paper tape and positioning the line printer to top of form. If such a function is requested, the initializer initiates the function and returns control to the calling routine. If no function request is indicated, then word 5 of the calling sequence is examined to see if unformatted binary data is to be input. If not, then the device's dedicated buffer is filled with packed ASCII blanks. If binary data is indicated, the buffer is left unchanged; then a hardware command is issued to start the input operation on a specified device and control returns to the general input initialization routine.

Figure B-33 shows a flowchart of a typical input formatter subprogram. Like the input initializer just described, an input formatter exists for each input device in the system. These routines are called from the input initializer routine IOII on its second entry. The purpose of these routines is to convert alphanumeric information in the device's buffer to binary information in the core area specified by the user program. The formatters first examine the calling sequence to determine if alphanumeric or binary information transfer was requested. If binary transfer was indicated then the formatters simply move the binary data in the device buffer into the specified area of the user's program. If, however, binary data transfer is not requested then the formatters transfer control to the "format handler for input" subroutine flowcharted in Figure B-34. When the format handler returns, the formatter passes control back to the general input initializer routine.

This format handler is a routine common to all input devices in the system. As shown in Figure B-34, it begins execution by initializing a number of flags, counters and pointers to be used in decoding the users format specifications. Input requests may specify one and only one format specification word. This word, whose address appears in word 5 of the user's input calling sequence, identifies the type of data conversion desired by the user. The user may specify conversion of any of the following data forms: single precision integer, double precision integer, packed ASCII text, one character per word ASCII text, or octal

FIGURE B-33. Typical Input Formatter

FIGURE B-34. Format Handler for Input

integer. The format handler proceeds to convert the data in the device buffer to the specified form, placing it in the area of core designated by the user's calling sequence. The program checks for a number of possible error conditions in the input data and if it detects one it decrements the task return address held in word 2 of the task's status block. If no error is found, the return address is not decremented. This causes control to transfer to the error return location of the input request as shown in Figure B-27 when the task is re-entered if an error in input has occurred; otherwise the normal return location will receive control when the task is reactivated following input. The format handler then returns control to the selected input formatter.

Figure B-35 shows a typical output formatter routine. A routine such as this exists for each output device in the system. The purpose of these routines is to handle the conversion of data present in the user's program from its internal form to data suitable for device output in the device's dedicated core buffer. These formatters are entered from the output initialization routine IOIO. Upon entry the user's output request is examined to determine whether or not he has requested a peripheral positioning function. If he has, this is handled in exactly the same way as in the input initializer routine of Figure B-32 described above. If data transfer is requested, the program examines the calling sequence to determine if the data to be output are binary or formatted alphanumeric. If binary data output is called for, then the data are simply moved from the user's area of core into the specified device's buffer. Following this, a word count is set into a dedicated location for the selected device. The formatter routine then signals the device to begin output and returns to the output initializer. If, on the other hand, formatted data output is requested, then the formatter transfers control to the format handler for output depicted in Figure B-36.

The format handler for output, FHO, is a routine common to all output devices in the system. Its purpose is to convert user's data into ASCII characters in the output device's dedicated core buffer according to specifications given by the user in a format specification table whose

FIGURE B-35. Typical Output Formatter

FIGURE B-36. Format Handler for Output

address is present in word 5 of the user's output calling sequence. This format specification table consists of a list of format specification words where each word specifies the manner in which a particular portion of the specified data area is to be output. In addition, the format specification table can itself contain certain types of data; namely, packed ASCII text, so that the user may have greater freedom in formatting his output.

FHO begins execution by initializing all necessary flags, counters and pointers used in conversion of the output data. It then obtains the first format specification word from the specified table. If this word indicates the end of the format specification table, a terminating character is placed in the output device's core buffer and control returns to the output formatter specified. If the end of the format specification table is not indicated, then FHO proceeds to convert the first data item in the user's specified core area to ASCII for output under the control of the format specification word just obtained. In the special case where the format specification word indicates that text is to be output directly from the format specification table, then the user's data is temporarily ignored and the indicated text is transferred from the format specification table to the device's buffer. Following conversion, the program loops back to obtain the next format specification word and the process repeats. If the program encounters the end of the user's specified data before the end of the format specification table is found, it continues to scan the table outputting any text which may be imbedded in the table. In this case it will ignore any requests from the table to output data since no more data remains to be output. When FHO finds the terminator for the format specification table, it places a terminator in the device's core buffer and returns to the device's output formatter.

B10, Input Interrupt Response

After the input initialization process described in the previous Section, data is input to a core buffer dedicated to the specified input device. This input process varies considerably from one device to another, and for this reason the particular process for each input device in the system is described herein.

Shown in Figure B-37 are interrupt response routines for remote teletype terminals. Routine 1131 handles a group of four remote terminals which are connected to one hardware unit. Routine 1132 performs the same function for a single fifth remote teletype connected to a different hardware unit. Routine 1131 is entered in response to a priority interrupt on hardware level 9. After saving hardware registers it sets up a series of TEU instructions which test the four remote terminal subunits associated with this unit. Each unit is tested to determine whether or not it is responsible for causing the interrupt. When a subunit is found which passes this hardware test, an input instruction to that subunit causes a word to be transferred into the 810B's accumulator. If this word is negative this indicates that a function key on the corresponding remote terminal has been depressed and in this case, subroutine IIF is entered. If, on the other hand, the incoming word is positive, this indicates that a key on the teletype's keyboard has been depressed and subroutine SIIM is entered. In either case, following the return from the selected subroutine the program continues to check remote terminal subunits until all four have been tested, in order to prevent the possibility of missing an interrupt due to the simultaneous occurrence of interrupts by two different subunits. 1132 performs a similar function on a single subunit associated with hardware priority interrupt level 10.

Priority interrupt level 11 is simultaneously connected to several unrelated sources. These are the paper tape reader, the line printer operation complete flag and the console teletype keyboard. In response to an interrupt on this level, routine II0I, flowcharted in Figure B-38, is entered. The routine proceeds to determine the cause of interrupt by performing a series of hardware tests on the various connected devices

FIGURE B-37. Remote Terminal Input Interrupt Response

FIGURE B-38. Common Input Interrupt Response

until the device causing the interrupt is located. If the paper tape reader caused the interrupt, control transfers to routine II02 flowcharted in Figure B-39. If the line printer was the source of the interrupt, this indicates that an output transfer request for the line printer has been completed; the line printer is disconnected from the interrupt and subroutine IOC, described in Section B12, is entered to indicate to the system that the output transfer to the line printer has been completed. If neither of the above devices caused the interrupt, the console teletype keyboard is tested. If it did not cause the interrupt, then the interrupt must have been erroneous and is therefore ignored. If, however, the program determines that the console teletype keyboard was the source of the interrupt, then a character is read from that keyboard. This character is examined to determine whether or not the operator has pressed a key on the teletype keyboard representing a pseudo-function key. Since the console teletype keyboard is not equipped with function keys, several of the keys on the keyboard are assigned equivalence with function keys on remote terminals. If this is the case, subroutine IIF is entered. Otherwise, the program checks to see if the key which was depressed is the spacebar. If so, and if no input request is active for the teletype keyboard as indicated by its entry in the device status table, then the system assumes that the operator wishes to invert the status of the "background program hole" condition, and does so. Otherwise, the routine enters subroutine SIIm for further processing of the incoming character.

The paper tape input interrupt response routine II02 illustrated in Figure B-39 is entered if II01 determines that the priority level 11 interrupt was caused by the paper tape reader. II02 first checks the device status table entry for the paper tape reader to determine if the input data are in binary form. If so, it checks the paper tape reader software start flag. This flag will have been set if a start code equal to an octal 377 character has previously been read. If the start code has been read, the incoming character is stored in the paper tape reader's software buffer. Next, the associated character counter is incremented and if it indicates that the buffer is full, that is, if the proper number of characters have been read to satisfy a word count specified by the

FIGURE B-39. Paper Tape Input Interrupt Response

input request in the user's program, then a terminator is stored in the buffer and subroutine IOC is entered to indicate to the system that the input is complete. If binary input is indicated and the start flag is not set, the program then checks to see if the current character is the start code. If not, no further action is taken. If the start code is detected in this case, the start flag is set. If the device status table entry for the paper tape reader indicates that the incoming information is not to be interpreted as binary, then the routine checks to see if a special editing character is being read. Editing characters, such as tab and rubout, cause different functions to be performed on the incoming data. For example, a tab character causes a string of spaces to be inserted in the paper tape reader's buffer to bring the number of characters to the next preset software tab stop. If the character is not an editing indicator, the routine checks to see if it is a carriage return. If so, termination of the input paper tape record is indicated and the routine stores a terminating character in the buffer and enters subroutine IOC to indicate completion of the input transfer. Otherwise, the incoming character is stored in the buffer and the character counter is incremented. Then in any case, control returns to the interrupted routine at the point of interruption.

If, in one of the previously described interrupt service routines, it is determined a function key has been depressed on a remote teletype terminal or if a pseudo-function key has been depressed on the console teletype in the central computer facility, then IIF, the function key response routine, illustrated in Figure B-40, is entered. This routine obtains the status word for the remote or console teletype of interest from the function key status table. If the busy bit in the status word is set, this indicates that the remote terminal or console teletype is in use by some active program. If this is the case and if the incoming function key code indicates that the key which was depressed is not the abort (RESET) key, then no action is taken by the function key response routine. If, however, the program determines that the abort key has been depressed then the task number of the task or background program now using the remote terminal of interest is obtained from the function key

FIGURE B-40. Function Key Response

response routine. If, however, the program determines that the abort key has been depressed then the task number of the task or background program now using the remote terminal of interest is obtained from the function key status word. Then word 1 of the associated task or background status block is set non-zero indicating that this task or background program is to be aborted the next time control passes to the monitor routine. If, on the other hand, the status word indicates that the terminal or console teletype is not busy, then the abort key, if pressed, would be ignored because since no program is associated with the teletype then none could be aborted. If the status word indicates that the terminal is not busy and a key other than the abort key caused the interrupt, a desire on the part of the operator to input and execute a particular program has been indicated. In this case, the number of the function key which was depressed is decoded according to a function key program table. This table provides the 6 character name of a program on the disc program file which is associated with the function key. IIF executes the program scheduler subroutine described earlier in this appendix. If the scheduler is unable to schedule the desired program then IIF takes no action. If, however, the scheduling operation is successful, then by virtue of the operation of the scheduler program, a new task has been created and the desired program has been linked to it. In this case, the terminal causing the interrupt is set busy with the scheduled task by placing its task number into the associated function key status table entry and setting bit 0 in that entry. Also, the device number of the terminal which caused the function key interrupt is set into word 11 of the status block for the task which has just been initialized. This action serves to link the task or background just initialized with the proper remote terminal or console teletype. In any case, control then returns to the routine which called subroutine IIF.

The teletype input interrupt response subroutine SIIM, flowcharted in Figure B-41, is entered from an interrupt response routine if the incoming character causing the interrupt is determined not to be a function key but rather to be a key on a teletype keyboard. SIIM first checks the associated device status table entry to determine if an input data transfer

FIGURE B-41. Teletype Input Interrupt Response Subroutine

request is in effect for the device. If not, the routine takes no action and returns immediately to the interrupt response module. If an input transfer request is in effect, SIIM checks to see if the incoming character is an editing character such as rub out, up arrow, or tab. If so, the proper editing function is performed on the device's core buffer. If the character does not indicate an editing function, the routine checks for carriage return. If this is the case, then the terminating character is stored in the device's buffer and subroutine IOC is entered to indicate to the system that the operator has completed typing of the input record. If the character is not a carriage return, it is stored in the character buffer for the device and the device's character counter is incremented. The character just input is then output back to the teletype in an echo-plexing operation. This is necessary because all teletypes in this system are operated in the full duplex mode. In any case, SIIM returns control to the interrupt response routine which called it.

B11, Output Interrupt Response

The routines described in this section are employed to output data to devices from their core buffers. These routines are initiated in response to output hardware interrupts which have been set up by the initialization routine described in Section B9. The response to output interrupts from remote teletype terminals is performed by the routines flowcharted in Figure B-42. Routine 1031 responds to interrupts from four remote teletype subunits of a particular output unit. Routine 1032 responds to output interrupts from a fifth single teletype unit. 1031 sets up a hardware TEU test of each of the four subunits associated with this unit. As each unit is tested, the TEU instruction determines whether or not a particular subunit was responsible for the output interrupt. When a responsible subunit has been located, subroutine SIRM, described below, is entered. When all four subunits have been tested control returns to the point of interrupt. Response routine 1032 performs the same function on its single associated output teletype terminal.

Figure B-43 illustrates the remote terminal output interrupt subroutine SIRM, which is entered from the output remote terminal interrupt response routines described in the above paragraph. Upon entry, this routine checks the device status table entry for the selected device to determine whether or not an output data transfer request is currently active for this device. If not, control returns immediately to the interrupt response routine and no action is taken. If, however, the busy and output bits of the associated device status table entry word are set, indicating that an output request is in effect for the device, then the next character, as indicated by the character counter for the device, is obtained from the device's core buffer. This character is examined and if found to be equal to a terminating character, then subroutine IOC, described in Section B12, is entered to inform the system that output has been completed. If the character from the buffer is found to be other than a terminating character, then an output instruction is performed in order to type the character on the selected typewriter. Following this, the device's character counter is incremented so that a

FIGURE B-42. Remote Terminal Output Interrupt Response

FIGURE B-43. Remote Terminal Output Interrupt Subroutine

subsequent interrupt will reference the next sequential character in the buffer. Control then returns to the interrupt response routine which called subroutine SIRM.

Figure B-44 is a flowchart of the routine I001. This routine is entered when an interrupt occurs on hardware priority level 12. The cause of this interrupt may be either the paper tape punch or the console teletype printer. TEU instructions are used by the routine to determine which device caused the interrupt. If the cause was the paper tape punch, control passes to routine I002 described below. If the cause of the interrupt was the console teletype keyboard, I001 obtains the next character from the console teletype's core buffer as indicated by the character counter dedicated to the console teletype. If this character is a terminator, subroutine I0C returns, I001 disconnects the console teletype from priority interrupt level 12 by means of a CEU instruction. If the current character from the output buffer is not a terminator, then the character is output to the console teletype with an AOP instruction. Then the character counter for the teletype is incremented so that a subsequent interrupt will process the next sequential character in the teletype's buffer. Then, in any case, control returns to the interrupted program at the point of interruption.

If routine I001, described above, determines that the cause of the interrupt on priority interrupt level 12 was the paper tape punch, then control transfers to the paper tape output interrupt response routine I002 flowcharted in Figure B-45. This routine first issues a CEU instruction to the paper tape punch which turns on its power. The next character, as indicated by the punch's character counter, is then obtained from the core buffer for the punch. Following this, the program checks the punch's entry in the device status table to determine if binary output has been requested. If so, the character counter is examined to see if the proper number of characters corresponding to the word count specified by the user has been reached. If so, the program checks a flag which indicates whether or not the punch has been responding to a user request for trailer output. If so, a CEU command is issued to turn off the punch power. Then, in

FIGURE B-44. Common Output Interrupt Response

FIGURE B-45. Paper Tape Output Interrupt Response

either case for the paths taken on end of output, subroutine I0C is entered to inform the system of the completion of the output to the punch. Then, I002 issues a CEU to disconnect the punch from interrupt level 12 and returns to the point of interrupt. If binary output is indicated but the character count indicates that the end of output has not yet been reached, then the character obtained from the buffer is output to the paper tape punch and control returns to the point of interrupt. If the punch's entry in the device status table indicates that binary output has not been specified, then the character from the punch's core buffer is examined to see if it is a buffer terminator. If it is, the program branches to the point at which subroutine I0C is entered as described above. Otherwise the character is output to the punch and control returns to the point of interrupt.

Output requests for the line printer are not handled on a character interrupt basis as are output requests for the paper tape punch and teletype units. Instead, output is done entirely in the initializing and formatting routines by means of successive AOP instructions. This is possible since the line printer hardware includes a buffer capable of storing an entire line of text. This enables the line printer to accept characters via AOP instructions as fast as the 810B computer is capable of transmitting them. Therefore, if the line printer were to be operated on a character interrupt basis, no significant time would be available between character interrupts for programs to proceed. Thus, no advantage would be realized in operating the line printer on a character interrupt basis.

Similarly, the magnetic tape units are operated under control of a high speed block transfer control channel. This hardware channel controls the input and output of characters to and from the magnetic tape units completely independent of any software operations. Therefore, no character input or output interrupts are required for magnetic tape.

B12, Input/Output Termination

Figure B-46 is a flowchart of the input/output complete subroutine IOC. This subroutine is entered from the interrupt response routine described in the previous two sections when these routines determine that an input or output transfer has been completed by an I/O device. IOC first obtains the device status table entry for the device of interest. It examines the input bit, bit 4, and if this bit is set, it is cleared. If the input bit is not set, then IOC clears the output and busy bits, bits 3 and 0. Note that in the case of input the busy bit is not cleared, since the completion of input by a device does not represent the end of the entire input process; but rather simply indicates that data is present in the input device buffer and is ready for conversion and transmission to the user by the input formatting routines. In the case of output, contrarily, entrance to subroutine IOC indicates that the entire output operation has been completed, since formatting occurs before transmission to the output device. Following the update of the device's status table entry, IOC obtains the task number of the task which has been using the device of interest from the device status table entry for that device. The input/output wait bit, i.e., bit 1 of this task's entry in the monitor's task list is then cleared. This indicates to the monitor that the task is no longer awaiting completion of an input/output operation and control may therefore return to the task the next time its turn comes up in the monitor's cycle. Following this operation, the input/output complete subroutine returns control to the routine which called it.

The input/output complete subroutine described above is employed to inform the system of the completion of a data transfer involving any device other than magnetic tape. In the case of magnetic tape the end of a data transfer operation will cause an interrupt on priority interrupt level 8 which is connected to the hardware block transfer control channel number 2 which is employed in magnetic tape input/output. An occurrence of this interrupt causes transfer of control to the routine illustrated in Figure B-47, the BTC number 2 interrupt response routine, IBT2. This routine begins operation by disabling its own interrupt level, i.e. level 8. It

FIGURE B-46. Input/Output Complete Subroutine

FIGURE B-47. BTC #2 Interrupt Response

then checks the device status table entry for the tape transport of interest to determine whether the transfer just completed is an input or an output transfer. In the case of output, the status words for all mag tape units are cleared. For input, on the other hand, only the input bits of these device status words are cleared. The reason for this operation is the same as for the similar operation described above for subroutine IOC. Next, IBT2 clears the input/output wait status of the task for which the magnetic tape transfer has been completed. Then it performs a TEU instruction in order to test the magnetic tape control unit for parity error condition. If no parity error is detected, control returns to the point of interrupt. If, however, a parity error is found to be in effect, then it is assumed that the magnetic tape record just read or written is in error. In this case, the return address in word 2 of the proper task or background status block is decremented so that the error return, rather than the normal return, will be taken when control returns to the user's program at the completion of the input/output process. Control then returns to the point of interrupt.

APPENDIX C

SAMPLE COMPUTER/OPERATOR
DIALOGUE FOR INVENTORY/CRITICALITY
CONTROL PROGRAMS

1917-1918

1917-1918

1917-1918
1917-1918
1917-1918
1917-1918

BNWL-1440

HOOD
HOOD ID: 32
PU777 55.00 GRAMS
PU888 2 105.00 GRAMS

Typical Hood Status Typeout

BATCH
RATCH ID: PU777

WEIGHT PERCENTS:

PU02 IN RATCH = 98.90
PU IN PU02 = 88.00
U02 IN BATCH = 0.00
U IN U02 = 0.00
U235 IN U = 0.00
MODERATOR = 9.10

RATCH PRESENT IN
31
32

Typical Batch Status Typeout

HOOD
 HOOD ID: 32
 PU777 55.00 GRAMS
 PU888 2 105.00 GRAMS
 US777 500.00 GRAMS

SUBBCH
 HOOD ID: 32
 BATCH ID: US777
 SUB-BATCH WEIGHT IN GRAMS: 900.0 NOT ENOUGH MATERIAL IN HOOD,
 SUB-BATCH WEIGHT IN GRAMS: 250.00 OK.

HOOD
 HOOD ID: 32
 PU777 55.60 GRAMS
 PU888 2 185.00 GRAMS
 US777 250.00 GRAMS
 US777 2 258.80 GRAMS

*Operation of Sub-Batch Creation Program, Showing Response
 to an Operator Error, with Hood Status Typeout Before and After*

BNWL-1440

HOOD
HOOD ID: 33 NO BATCHES PRESENT.

OPERATOR 1: 77777
OPERATOR 2: 88888
ORIGIN: 1
DESTINATION: 33
BATCH ID: PU888
WEIGHT IN GRAMS: 500.00

TRANSFER AUTHORIZED.. ,TRIP # 31

OPERATOR 1: 66666
OPERATOR 2: 55555
TRIP #: 31
DESTINATION: 33
WEIGHT IN GRAMS: 500.00 TRANSFER COMPLETE

HOOD
HOOD ID: 33
PU888 500.00 GRAMS

*Proper Operation of the Transfer Origination and Transfer Termination
Programs, Showing Hood Status Typeout Before and After*

HOOD
 HOOD ID: 33
 PU888 250.00 GRAMS
 PU888 3 259.80 GRAMS
 US777 2 250.00 GRAMS

~~MX~~
 HOOD ID: 33
 FIRST LOT ID: PU888 3
 WEIGHT IN GRAMS: 125.00
 SECOND LOT ID: US777 2
 WEIGHT IN GRAMS: 125.00
 NEW BATCH ID IS FS 010

HOOD
 HOOD ID: 33
 PU888 250.00 GRAMS
 PU888 3 125.00 GRAMS
 US777 2 125.08 GRAMS
 FS 010 250.00 GRAMS

*Operation of Material Mixing Program, with Hood
 Status Typeout Before and After*

OPERATOR 1: 44444
OPERATOR 2: **33333**
ORIGIN: 1
DESTINATION: **33**
BATCH ID: **PU888**
WEIGHT IN GRAMS: 17250.08
THAT **WOULD** EXCEED THE NET WEIGHT LIMIT.
TRANSFER REQUEST DENIED. **..PRESS** "RESET" KEY TO ACKNOWLEDGE.

*Improper Operation of the Transfer Origination Program, **Which**
Results in a Warning that the Proposed Transfer **Would** Violate
Criticality Safety Limits*

UNAUTHORIZED TRANSFER ORIGINATED BY OPERATORS 44444 AND **33333**
FROM STATION 1 TO STATION **33**

*Typeout on Console Teletype in Central Computer Facility Resulting
from a failure by the Operator in the Above **Example** to Acknowledge
His Error by Pressing His **"RESET"** Key*

BATCH
 BATCH ID: FE 878 2

WEIGHT PERCENTS:

PU02 IN BATCH = 25.00
 PU IN PU02 = 88.00
 U02 IN BATCH = 75.88
 U IN U02 = 88.13
 U235 IN U = 29.99
 MODERATOR = 0.00

BATCH PRESENT IN
 25

BINDER
 ADD OR REMOVE? (A OR R): A
 HOOD ID: 25
 BATCH ID: FE 070 2
 BINDER WEIGHT IN GRAMS: 14.80 OK.

BATCH
 BATCH ID: FE 070 2

WEIGHT PERCENTS:

PU02 IN BATCH = 23.82
 PU IN PU02 = 88.00
 U02 IN BATCH = 71.47
 U IN U02 = 88.13
 U235 IN U = 29.99
 MODERATOR = 4.71

BATCH PRESENT IN
 25

*Operation of Binder Addition and Removal Program, Used for
 Binder Addition, with Batch Status Typeout Before and After*

Distr-1

DISTRIBUTION

OFFSITE

No. of
Copies

29	<u>AEC Division of Reactor Development and Technology</u> Director, RDT Asst Dir for Nuclear Safety Analysis & Evaluation Br, RDT:NS Environmental & Sanitary Engrg Br, RDT:NS Research & Development Br, RDT:NS Asst Dir for Plant Engrg, RDT Facilities Br, RDT:PE - Components Br, RDT:PE Instrumentation & Control Br, RDT:PE Liquid Metal Systems Br, RDT:PE Asst Dir for Program Analysis, RDT Asst Dir for Project Mgmt, RDT Liquid Metals Projects Br, RDT:PM FFTF Project Manager, RDT:RE Asst Dir for Reactor Engrg, RDT Control Mechanisms Br, RDT:RE Core Design Br, RDT:RE (2) Fuel Engrg Br, RDT:RE Fuel Handling Br, RDT:RE Reactor Vessels Br, RDT:RE Asst Dir for Reactor Tech, RDT Coolant Chemistry Br, RDT:RT Fuel Recycle Br, RDT:RT Fuels & Materials, Br, RDT:RT Reactor Physics Br, RDT:RE Special Technology Br, RDT:RT Asst Dir for Engrg Standards, RDT EBR-II Project Manager, RDT:PM
1	<u>AEC, Chicago Patent Group</u> G. H. Lee
5	<u>AEC, DTIE</u> For UC-32 Distribution
1	<u>United Nuclear Corporation</u> R. F. DeAngelis
1	<u>Bechtel Corporation</u> J. J. Teachnor

Dist-2

OFFSITE (continued)No. of
Copies

1	<u>Westinghouse Electric Corporation</u> Atomic Power Division Advanced Reactor Systems D. C. Spencer
4	<u>AEC Site Representatives</u> Argonne National Laboratory Atomics International General Electric Company Westinghouse Electric Corp
3	<u>Argonne National Laboratory</u> R. A. Jaross IMFBR Program Office N. J. Swanson
1	<u>Atomics International</u> FFTF Program Office
1	<u>Liquid Metal Engineering Center</u> A. E. Miller
2	<u>Babcock & Wilcox Company</u> Atomic Energy Division S. H. Esleeck G. B. Garton
5	<u>General Electric Company</u> Advanced Projects Operation Kark Cohen (3) Nuclear Systems Programs D. H. Ahmann (2)
2	<u>Gulf General Atomic Inc.</u> General Atomic Division D. Coburn

ONSITE - HANFORDNo. of
Copies

1	<u>AEC, Chicago Patent Group</u> R. K. Sharp
---	---

Distr-3

ONSITE - HANFORD (continued)

No. of
Copies

4	<u>AEC, RL</u> <u>D. W. Gossard</u> <u>H. K. Mitchell</u> <u>M. R. Schneller</u> <u>J. M. Shivley</u>
2	<u>RDT Assistant Director for Pacific Northwest Programs</u>
33	<u>Battelle-Northwest</u> <u>E. R. Astley</u> K. M. Buxness M. R. Compton V. L. Crow (5) J. L. Deichman P. J. Dionne G. E. Driver M. D. Erickson (6) R. E. Falkoski L. H. Gerhardstein E. N. Greenwell R. R. King R. E. Mahan G. W. Main T. A. Seim N. B. Smith R. W. Stewart Technical Information (5) Technical Publications (2)
42	<u>WADCO</u> <u>J. K. Anderson</u> R. O. Budd C. A. Burgess J. J. Cadwell A. W. DeMerschman J. B. Edgar W. H. Esselman E. A. Evans W. M. Gajewski R. A. Harvey (5) J. N. Judy R. L. Junkins H. T. Knight G. A. Last

Distr-4

ONSITE - HANFORD (continued)

No. of
CopiesWADCO (continued)

F. J. Leitz
W. R. Markillie (5)
J. M. Norris
D. G. Ott
H. G. Powers
W. H. Payzant
B. G. Rieck
W. E. Roake
C. D. Swanson
W. E. Taylor
L. D. VanHees
J. R. Walker
J. C. Watkins
N. P. Wilburn
B. Wolfe
FFTF File - 703 (5)