THE DESIGN AND IMPLEMENTATION OF A REMOTE TERMINAL MONITOR

A thesis

Presented to

the Department of Computer Science

University of Manitoba

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

William Reid

December 1971

ABSTRACT

This thesis describes the design and the implementation of a monitor for low-speed remote terminals. The basic features of the monitor are:

- terminal users have access to a library of programs,
- all communication with the terminals is handled by the monitor,
- any program executing in the computer can communicate with a terminal via the monitor,
- programs which are communicating with the terminals are transferred to backing storage when they are not active. This allows different programs to use the same memory area during their execution.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

The aim of this thesis is to describe the design and implementation of an on-line monitor for remote terminals. The University of Manitoba was committed to supporting remote terminals for a variety of on-line applications. Two on-line systems were considered: the Conversational Remote Batch Entry (CRBE) and the Administrative Terminal System (ATS). CRBE allows the user to edit disk data sets and submit card image data sets for batch execution. ATS provides text editing facilities. Together they required too much of the computer's resources to justify their use. Also, whenever more than one terminal system is used there is always some duplication of resources. No existing system could support all the various applications proposed for the terminals. Therefore, it was decided that the University should develop a flexible terminal monitor which would support the applications but would not make excessive demands on the computer resources. The monitor was named the Manitoba University Monitor (MUM).

The program development was done in close consultation with Dr. C. Abraham who provided the basic design and motivation of the MUM project and who was also working

simultaneously on parts of MUM such as: File handling, EDIT program, and other programs which required continuous changes in design and implementation of other parts of MUM.(1)

On-line, Real Time, Time-sharing

An on-line system is one in which a direct path is provided between the user and the computer. The path commonly consists of a telephone line and a cathode ray tube or typewriter which allows the user to send or receive information to or from the computer.

An on-line system can be passive or interactive with the terminal environment. A data acquistion system is passive with respect to the terminal environment. An interactive system produces feedback to control or change the environment. An example is a process control system. An interactive system which communicates with people may also be called a conversational system.

A real time system is an on-line system which reacts to input sufficiently quickly to affect the functioning of the environment at that time. A criterion for differentiating between real time and on-line, is how quickly does the

system "respond" to the input. For example, the response time for a man-machine interaction may be defined as the time when the last character was entered to the time when the computer types the first character of the answer. For conversational systems, this response time should be in the order of a few seconds to be considered real time.(2) The response time may vary from milliseconds in a missile guidance system to hours in a process control environment.

Any on-line system for which more than one terminal can be active simultaneously can be called a time-sharing system. Since most on-line systems have this capability, the term should not be used in this general way. It is used more commonly to refer to systems where the user's terminal appears to be connected to a dedicated computer and the user can enter, test, and execute programs.(3) The user of a time-sharing system may have access to only one programming language, or to the extreme of loading his own operating system.

MUM was designed primarily to meet the needs of a real time conversational system.

CHAPTER II

DESIGN CRITERIA

The following points were the main features felt necessary for the implementation of the MUM system.

1. The terminal user must have access to a library of application oriented programs. These programs should be able to be easily written and added to the system.
2. All communication with the terminals should be handled by the monitor. The monitor should provide a device independent interface for the application programs.
3. Any program executing under the operating system but not under the control of the monitor should be able to communicate with the terminals through the monitor.
4. Programs which are under the control of the monitor should share the same core storage. Since many terminals can be operating simultaneously, many application programs may be in use at the same time. It would require excessive core storage if each program was resident during the terminal session. Therefore, while data is being transmitted to or received from the terminal, the program can be

transferred from core storage to disk storage. This allows another program to use the same core area. To improve performance, more than one core area should be available.

5. Programs should be able to pass control to other programs.
6. The implementation should be designed to aid debugging, maintenance, support of new terminal types, and the addition of new monitor facilities.

CHAPTER III

EXTERNAL IMPLEMENTATION OF MUM

To use MUM, the user must first identify himself with an account number. If the account number is valid, he can then request to communicate with a particular application program. When the desired results are obtained, he indicates to the program that he is finished. The program then returns control of the terminal to MUM; the user can now request to communicate with another program.

Signing On or Off

The user signs on by giving his account number and an optional password. The password is to prevent unauthorized use of the account number. One can sign on implicitly or explicitly. With the implicit signon the user gives only the account number. He is signed on only for the duration of the requested program. When the program terminates, he must sign on again. To explicitly sign on the user precedes the account number with "signon". When the program terminates, he has the option of either requesting another program without the need of entering his account number or signing

off. To sign off the user enters "signoff" or another signon (either implicit or explicit).

When the user requests a program he has the option of passing initial data to the program. This saves time since the program can dispense with the first prompting message.

User Attributes

Data, describing a user's attributes, are associated with each account number. Data such as tab positions at previous signoff, whether accounting information should be displayed at signoff, whether signon messages may be abbreviated, and the current password are kept and can be modified by the user. The accumulated signon time, which can be displayed by the user, is also kept.

Monitor Requests

Monitor requests are commands which are serviced directly by MUM and are transparent to the application program. These requests are functions which are required by the user, but are independent of the application program

with which communication has been established. The requests permit the user to:

- obtain the time and the date.
- test the terminal by displaying a fixed pattern of characters or by echoing input data.
- obtain the terminal identification.
- obtain the number of signed on users.
- obtain or change the current tab positions.
- send messages to the computer operator.
- LINK or SWAP (pass control) to another application program.

To differentiate these requests from input to an application program, the user precedes the request with two question marks.

A Sample Terminal Session

Figure 1 is the script for a typical terminal session. Text in lower case was entered by the user and text in upper case was typed by the computer. The points below refer to the numbers on the left hand margin of Figure 1.

1. A user whose account number is 4 requests to communicate with the application program LIST. This

```
        ENTER ACCT #,PRG NAME
 1      4 list
        LIST PROGRAM, ENTER COMMAND
        end
        OK
 2      6S
        ENTER ACCT #,PRG NAME
 3      4 ?change,p123
        ENTER ACCT #,PRG NAME
 4      signon 4.p123 edit old demo
 5      00150      $$list 30 3
        00030      $JOB  WATFIV   REID
        00040                READ 1,X,Y
        00050       1        FORMT(F5.2/F5.2)
 6      00150      ??time
        TIME IS 21.32.30
 7      $$submit
        FILE SUBMITTED
        1M 10s
        ENTER PRG NAME
 8      ?minimum
        PRG
 9      ?no-account
        PRG
10      print inquire demo
         864 AWAITING PRINT
        ??ti
        TIME IS 21.33.15
11      find 864 prt1 2 20
             1             READ 1,X,Y
             2    1        FORMT(F5.2/F5.2)
        ***ERROR***  INVALID ELEMENT IN INPUT LIST OR DATA LIST
                        .
                        .
12      abandon
13      ??link edit o demo
14      00150      $$@  50
        00050       1        FORMT(F5.2/F5.2)
15      00050                   ##mat
        00150      $$1 50
        00050       1        FORMAT(F5.2/F5.2)
16      00150      $$s
        FILE SUBMITTED
17      i demo
         868 AWAITING PRINT
18      release
        end
        PRG
19      off
        #,P
```

FIGURE 1
Sample terminal session

is an implicit signon.

2. After ending the LIST program (see appendix A) MUM prints the time spent communicating with the program.
3. The password is changed to "P123".
4. With an explicit signon, using the new password, the user passes a request to EDIT (see appendix A) to load an old file called DEMO.
5. EDIT responds with the next available sequence number for input. The user requests to list three lines starting at line 30.
6. After listing the requested lines, EDIT again prompts the user for input. At this point the time of day is requested.
7. The file is submitted to the batch job queue.
8. The user requests that all signon messages be of minimum length. As can be seen by the next line "ENTER PRG NAME" has been shortened to "PRG".
9. The displaying of signon time is turned off.
10. The PRINT program (see appendix A) is called up and passed an inquiry about the status of the submitted job. Since an explicit signon was used no account number was required.
11. After requesting the time, using the abbreviated

form, part of a print data set for the job is listed.

12. Noticing the error, the output is abandoned.
13. Control is transferred to EDIT with a request for the file DEMO.
14. A request is made to edit line 50.
15. MT is replaced by MAT.
16. After checking the correction the file is submitted again.
17. Control is returned to PRINT and an inquiry is made on the status of DEMO.
18. The output from the job is released to be printed on the system line printer.
19. The user signs off.

CHAPTER IV

INTERNAL IMPLEMENTATION OF MUM

MUM was designed to operate on an IBM 360/65 computer under the control of the Operating System (OS), with the options multiprogramming with a fixed number of tasks (MFT) or multiprogramming with a varialbe number of tasks (MVT). The functions of job input, job scheduling, and collection and printing of job output is performed by the Houston Automatic Spooling and Priority System (HASP). HASP is not required for the execution of MUM, but certain MUM application programs interface with it. The monitor was written in 360 Assembler and supports a variety of terminals.

The basic purpose of a terminal monitor is to control the flow of data between the terminal and the program which processes the input data. With this in mind, the structure of MUM can be represented by three sections (see Figure 2).

1. Control and service routines - This section contains the monitoring routines which control the flow of data between the various routines of MUM. It also contains service routines which are used by the other MUM routines.
2. Request handlers - These routines service the

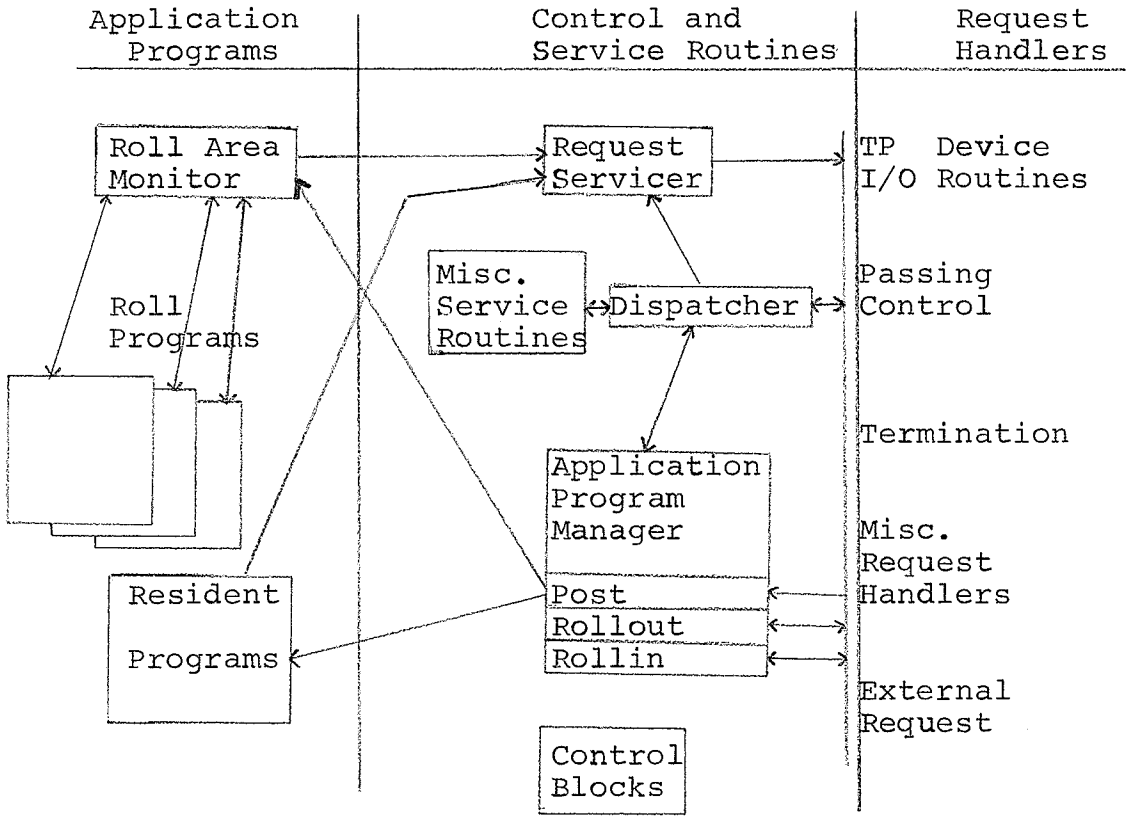


FIGURE 2

MUM program structure

application programs' requests. For example, an application program can request to communicate with a terminal.

3. Application programs - These programs, whether under the control of MUM or the operating system, process the input data and produce output data which is sent to a terminal or another program, via the monitor. The input data may also come from a terminal or another program.

The remainder of this chapter discusses the internal implementation of MUM under the three sections outlined above.

Control and Service Routines

MUM Control Blocks

A control block is a block of consecutive memory in which control information is kept for the purpose of efficient communication among the modules of a system. The control blocks in MUM serve similar purposes to control blocks in other systems. In particular, control blocks

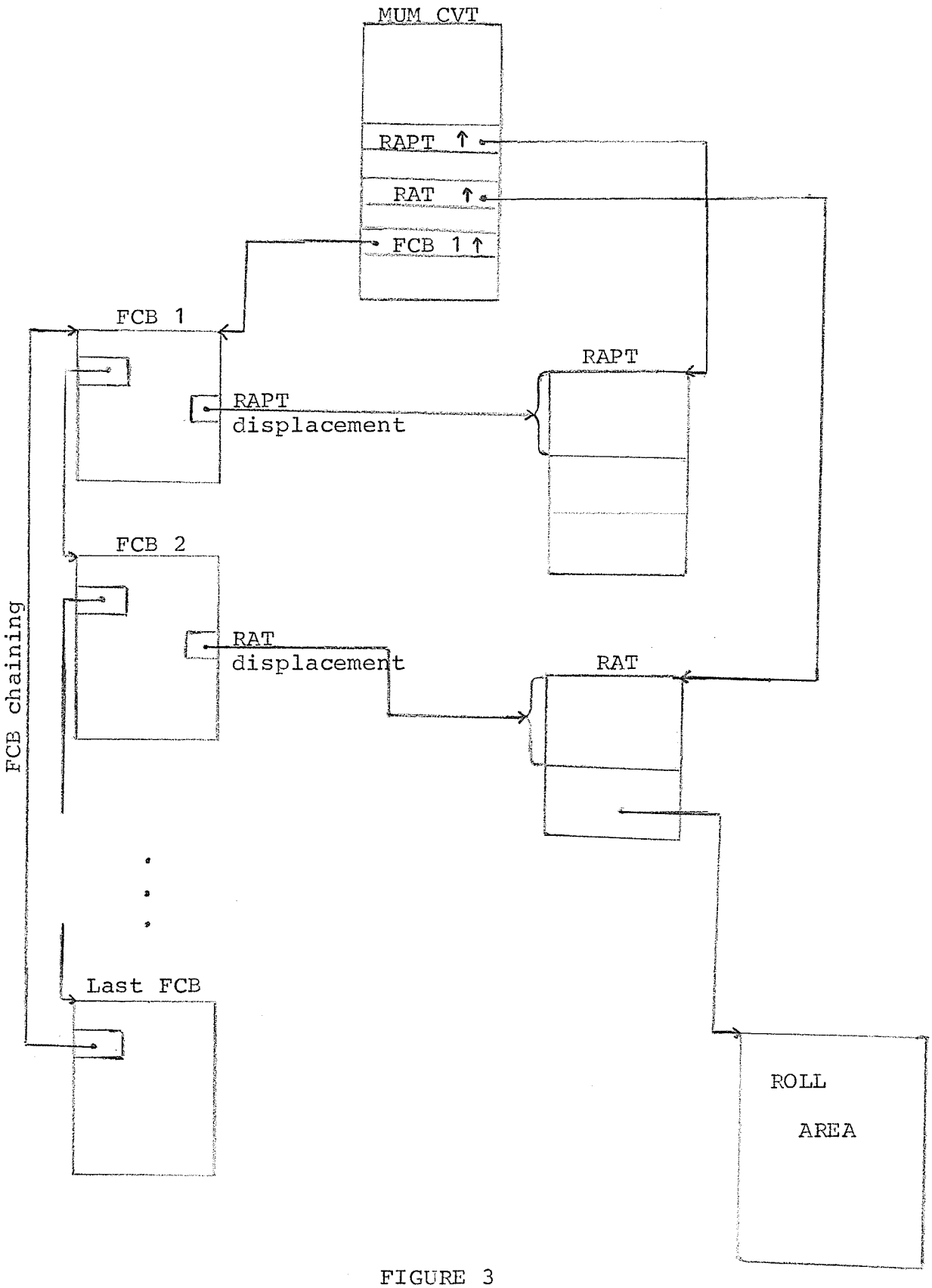


FIGURE 3
MUM control blocks

provide a means of organizing the structure of a system in an orderly manner. The main control blocks in MUM, as shown in Figure 3, are described below.

The main control block is the MUM Communication Vector Table (MCVT). As the term vector implies, it is a table consisting mostly of pointers to other control blocks. The advantage of the MCVT is that an external routine requires only one address to access all of MUM's control blocks.

The Function Control Block (FCB) was designed to maintain the information required to service a terminal transaction. One FCB exists for each terminal.

The Roll Area Table (RAT) and the Resident Application Program Table (RAPT) contain information about programs currently in core storage. The RAT is made up of one entry for each roll area in MUM. Likewise the RAPT contains an entry for each resident program-terminal association. The entries in the tables have the same displacements for common fields. This allows most routines to access the information without regard to the fact of whether the program is resident or rollable.

For a detail description of the control blocks' fields see appendix B.

Dispatcher

The dispatcher passes control of the CPU to the various components of MUM. In OS synchronization is achieved by the event control block (ECB). The program WAITs for an event to complete via the ECB. When The event completes, the ECB is POSTed with a completion code and the waiting task receives control.

If each routine in MUM were allowed to wait individually, MUM would be able to process only one terminal. To prevent this problem, a MUM routine gives control to the dispatcher with the address of the ECB when it must wait for an event to complete. The dispatcher puts the address in an ECB list and waits on all non-posted events. When an event completes the operating system passes control to the dispatcher which in turn decides which event completed. The dispatcher gives control to the MUM routine whose address is stored immediately preceding the posted ECB. The ECB list consists of a dynamic and a permanent section. In the dynamic section, the ECB addresses are deleted before control is passed to the waiting routine while in the permanent section they are left. ECBs in the permanent section are used by routines which always require an outstanding ECB.

Request Servicer

All program requests to MUM are first handled by the request servicer. Since requests can come from any program (task) in the system, the supervisor call (SVC) is used. A SVC is a CPU instruction which allows control to be passed to a resident routine in the operating system's nucleus. The instruction operand is used to determine which routine should receive control. Upon receiving control the MUM SVC routine saves register zero, register one, and the address of the task control block. Register zero contains the request code and request modifier. Certain request codes are handled immediately by the SVC routine and control is returned to the calling program. If the request is a valid MUM request, the ECB for the request servicer is posted before returning to the caller.

For test versions of MUM the SVC can be replaced by branching internally to the SVC routine in MUM. This of course prevents communication between MUM and other resident programs in the OS system.

When the request servicer receives control from the dispatcher it checks its queue for a new request. The saved contents of register one point to the communication block. Depending on the request type, the contents of the communication block are validated. If the contents are

valid, the request modifier is moved into the function control block and a branch is taken to the requested routine.

Application Program Manager

Before discussing the functions of the program manager, we should define the states in which a program can be:

active - has control of the CPU,

waiting - waiting for one or more events to complete,

ready - requires the CPU but is held pending for some reason.

From the point of view of MUM these states can be redefined to read:

active - a MUM request has been completed and MUM has posted the program,

waiting - a MUM request has been received and the program is waiting either in core storage or on backing storage (disk or drum),

ready - the request has been completed but the program is held pending because there is no free roll area.

A roll program is defined as a program which executes under the control of MUM and which may be transferred from or to backing storage whenever it is inactive.

A resident program is defined as a program which

resides in core storage for the duration of its execution. It will usually be a program which was submitted for batch execution.

The program manager keeps the status of the programs associated with MUM. For resident programs it verifies if the program is still available. For roll programs the following decisions must be made. When a roll program is required, a check must be made to determine if it is in a roll area or on backing storage. If it is in a roll area, is it active or waiting? If it is on backing storage, is there a free roll area? The decisions involved in swapping programs will be discussed in the following section.

The request handlers interface with the program manager through the following routines.

POST - the request is complete and the ECB in the communication block is posted with a completion code.

ROLLOUT - when a request can not be completed immediately the request handler indicates that the program may be rolled out. Of course for a resident program nothing happens.

ROLLIN - when a request has been completed, the request handler informs the program manager to bring the program back into core storage. For a resident

program the program manager verifies that the program is still in core. For a roll program the request is inserted in a queue until the the program is rolled into a roll area. When the program is in core the program manager passes control back to the request handler.

HOLD and FREE - for certain requests some events may occur in parallel instead of sequentially. To prevent the program from being rolled in before all events have been completed, the request handler will issue a hold for each event. When each event has completed, the request handler will issue a free. When the frees equal the holds, the program will be rolled in.

The use of these interface routines removes the distinction between a resident program and a roll program.

Swapping Programs

The current version of MUM has only one roll area so the swapping algorithm does not support more than one roll area, but the structure of MUM supports multiple roll areas.

A roll area can be in one of four states: free, active, waiting, or swapping. In the free state the area can contain a program which has been rolled out, but is still usable, or

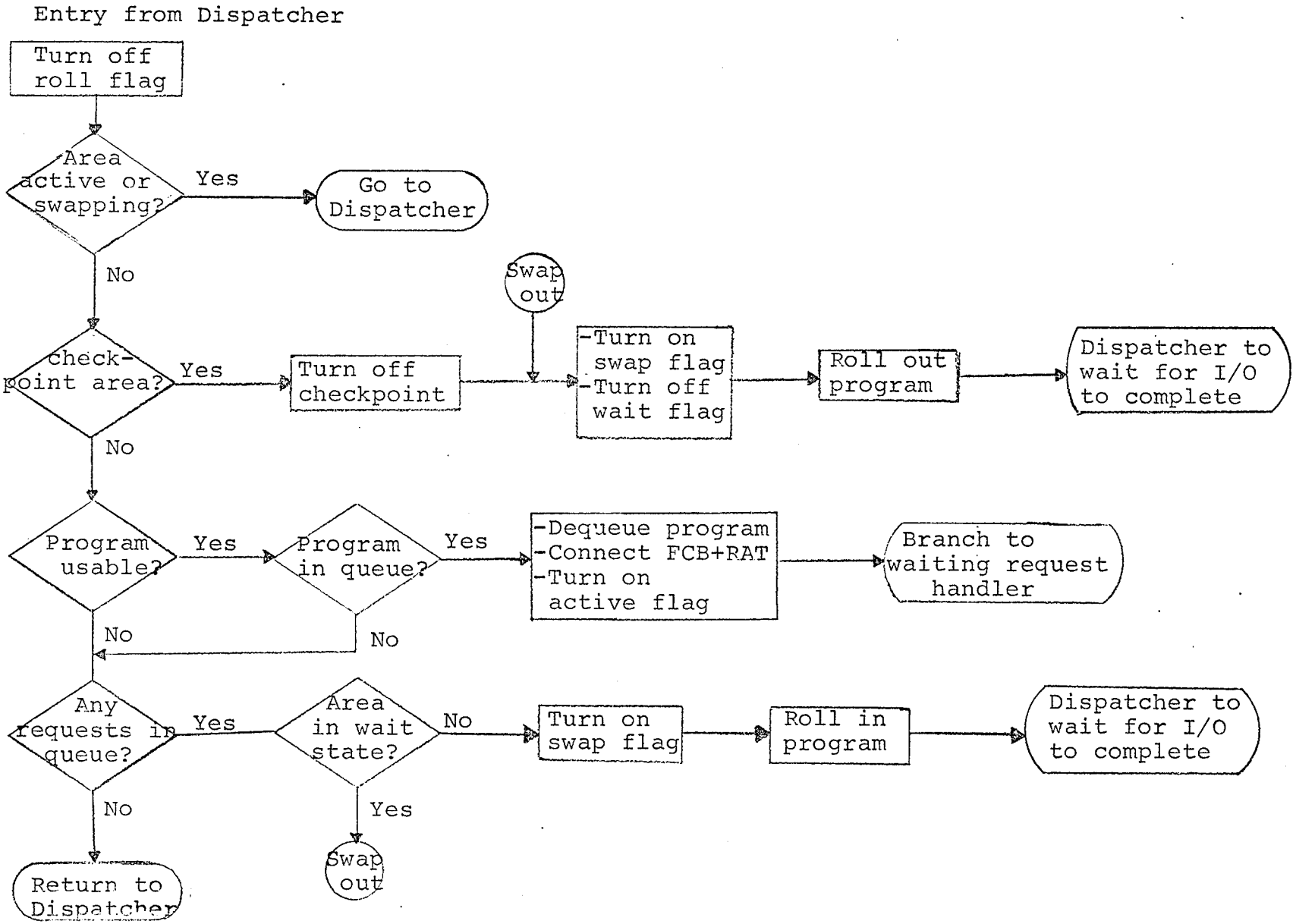


FIGURE 4

Flowchart of swapping algorithm

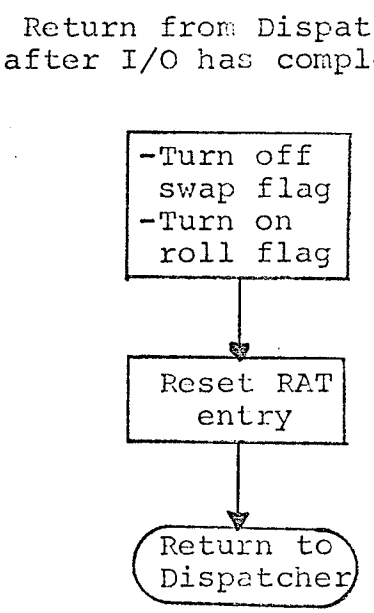


FIGURE 4 (continued)
Flowchart of swapping algorithm

a copy of an unusable program. An example of an unusable program is one that has terminated abnormally. The area must be in the free state before another program can be rolled into it. The area is active when the program, residng in it, is executing. The waiting state implies that the program has made a MUM request and can be rolled out. The area is in the swapping state when a program is being rolled in or out.

Whenever a roll area changes states or has the possibility of changing states, a roll possible flag is set. Before the dispatcher waits on its ECB list, it checks this flag. If set, control is passed to the program manager. Figure 4 gives a flowchart of the decisions to be made. As can be seen this algorithm is for one roll area. All queued requests for a program which is curently in the roll area will be given priority over requests for other programs. This is efficient, but there is the possibility of blocking the roll area to other programs, if there is always a request in the queue for the current program. Factors which will increase this possibility are a large number of terminals using a common program, the time required to proccess the transaction, and the time to transmit the information to the terminal. This situation is not handled in the current implementation, but with the current number of terminals and types of application programs the situation

could not occur.

Besides the above problem, multiple roll areas introduce other problems. As will be described later some programs can be rolled into any area while others have to be rolled into a particular area. To compensate for the fact that some programs will have a better chance of finding a free roll area, the algorithm could give preference to the programs which must be rolled into a particular free area. Similarly as above, this could prevent programs from being rolled in. Both problems can be solved by having a pre-emption count or time in queue limit which when exceeded would force the program manager to service the roll request. One other modification would be to give swapping in priority over swapping out. This is important when one roll area requires its program to be swapped out while another requires its program to be swapped in. If the swap in is done first, the execution of this program can then be overlapped with the swapping out of the other program. A pre-emption count would guarantee that a program would be rolled out within a certain time interval.

An attempt should always be made to keep frequently used programs in core. Also, the algorithm should attempt to keep in core at the same time programs that are linked together.

Miscellaneous Service Routines

Common service routine used by control routines and request handlers perform the following tasks.

- changing the program status word.
- relocation of addresses in the roll area.
- loading registers.
- moving data between buffers.
- locating a particular function control block.
- enqueuing and dequeuing function control blocks.

REQUEST HANDLERS

Request handlers receive control from the request servicer. They have the option of having the request servicer load registers with the current function control block address, the address of the entry in the application program table, and the communication block address or with the contents of the request registers zero and one and the address of the TCB of the requesting task. MUM distinguishes bewteen internal and external requests. External request handlers are external to the main MUM module. This is for ease of implementing new requests. All internal requests are implemented in the same module so they can easily share

common routines and constants. This section will only deal with internal requests.

Teleprocessing I/O Request Handler

The teleprocessing (TP) I/O request handler consists of a main independent routine and many terminal dependent routines. The main routine interprets the request, formats the message, translates it to the proper code, retries errors, interfaces with the program manager, and edits the input data. The terminal routines insert the necessary control characters, interface with the operating system, handle error conditions, and calculate message lengths. The connection between the main routine and the terminal routines is via a device interface table. It consists of a base address and half word displacementstfor each entry point of the routine. The function control block for a terminal contains an index into a list of device interface table addresses. The advantage of this method is that new terminals can be supported very easily by simply adding a small device interface routine and modifing the FCB to point to the correct address in the list of Device Interface Table addresses. No changes are required in the main MUM module.

Output to the terminal can be requested in two ways. The program may supply data which is translated and sent

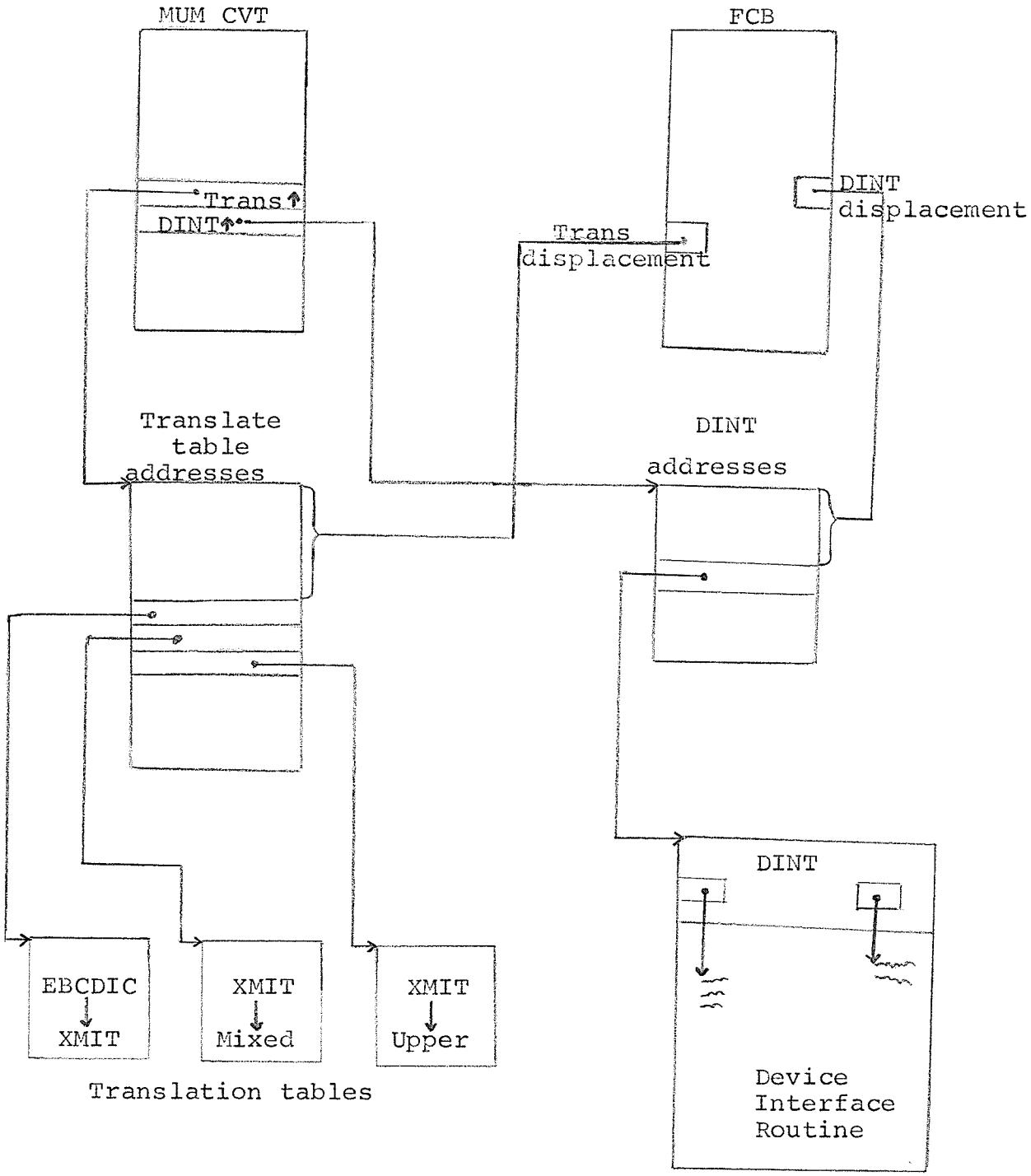
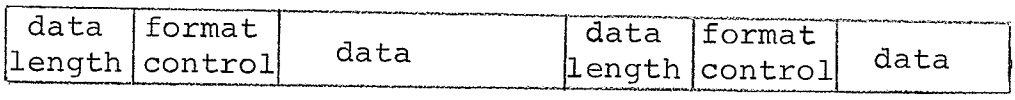


FIGURE 5

Locating translation tables and
device interface routines

directly to the terminal. In this mode the data must include all necessary control characters. The second way is for the program to supply a message with instructions for formatting. The message is in the form:



The format control character indicates if the data should begin at the beginning of the line, the number of line feeds or carriage returns that should be inserted before or after the data, and whether this data is the end of the message. In this way the program can send more than one line of output with each request. The length of the message depends on the size of the buffer assigned to the terminal.

Input from the terminal can be requested with a normal read or an interrupt read. With the interrupt read the user must first generate an interrupt before he can input data. For some terminals a read once issued can not be terminated by the computer, but with an interrupt read it can. This allows the monitor to terminate a read if there is output waiting to be sent to the terminal.

For the sake of terminal independence all data processed by the application program is in Extended Binary

Coded Decimal Interchange Code (EBCDIC). This implies that before data is sent to a terminal it must be translated to the proper transmission code. Likewise data received from a terminal must be translated to EBCDIC. Most programs require the input to be in uppercase only, therefore the main routine unless otherwise requested will translate all input to uppercase EBCDIC. To obtain the address of the required translate table the function control block contains an index into a list of table addresses. The three addresses pointed to by the index are for the tables to translate EBCDIC to transmission code, transmission code to EBCDIC, and transmission code to upper case EBCDIC. If the index is zero the transmission code is upper case EBCDIC and need not be translated. If any of the table addresses are zero the translation is not done. For infrequently used terminal types only two tables need be supplied. For upper case only the lower case table will be used and after the final editing the data will be converted from lower to upper case by logically oring the data with blanks.

The main routine will also edit the input data unless the program requests no editing. The editing consists first of checking the last significant character for a delete character. If found the read will be re-issued. A backspace character will delete the preceding character unless the

first non-backspace character after the backspace is an underline character. Characters, which are translated to a hexadecimal zero, and control characters will be deleted. A tab character will generate blanks to the next specified tab position. The editing routine attempts to estimate the location of the terminal carriage. This enables the terminal routines to insert the proper number of control characters in the output data.

The program can with the same request ask that the write be immediately followed by a read or that the program be terminated following the write.

Send Request Handler

Programs can send messages to terminals other than the one with which they are currently communicating. The program is not allowed to proceed until the message has been sent to the other terminal.

Passing Control Request Handler

The three methods of passing control of the CPU to another program are LINK, transfer control (XCTL), and SWAPping. The LINK request allows a program to pass control to another program and when that program ends control is passed back to the program which issued the LINK. XCTL

passes control to another program but control is not returned to the program which issued the XCTL. In effect, the program transferred to, replaces the program which issued the XCTL. SWAP passes control from the current program to the program which LINKed or SWAPped to it and allows control to be returned. In other words the programs exchange places.

A program can LINK or XCTL directly or by name. In the direct mode a program supplies the requested program's disk location and the program's attributes. Usually these are not known so a program can LINK or XCTL by suppling the name of the desired program.

Along with passing control of the CPU, a program can pass data. While the transfer is taking place the data is stored in the buffer assigned to the terminal.

In passing control the requested program's name may be invalid or there may be no temporary disk space available. The requesting program will be returned control with an error indication, but it has the option to also have the monitor send an error message to the terminal.

Another feature is the ability of the program to pass control of the CPU, yet remain in control of the terminal. This is done by allowing the program to intercept all TP I/O requests of the program to which control has been passed. It

can also intercept only read requests. This means that input data is first passed to it instead of the program which issued the read request.

Termination Request Handler

A program may end normally or abnormally. For a normal end, it may pass data back to the program which LINKed or XCTLed to it. An abnormal end causes the monitor to discard the program copy in core. That is, it will not roll it back to the disk or allow other terminals to use it.

After all application programs have terminated, control is passed to the monitor signon/signoff program. This is a roll program which handles the user signon and signoff. This program will prompt the user for a new program name or a new signon.

Pause Request Handler

Since MUM has no automatic time slicing, programs which may reside in the roll area for long intervals must regulate themselves. The PAUSE request allows MUM to decide if it is necessary to roll the program out to give other programs a chance at the roll area.

Miscellaneous Request Handlers

Some other functions not mentioned yet are checkpointing a program, informing the monitor of a waiting resident program, and enabling or disabling a terminal.

Application Programs

Roll Programs

Roll programs must be written in IBM/360 Assembler, with the following restrictions:

- the program must be no larger than 7294 bytes,
- address constants can not be used,
- certain system macros can not be used,
- the program should consist of only one control section.

The program should be written so that it can be relocated during MUM requests. If this is not possible then MUM will always roll the program back into the same roll area.

Roll areas consist of a resident area followed by a 7294 byte area into and from which programs are rolled. The resident area contains a register save area, control block pointers, and the start of the parameter list which is

passed to the roll program. The parameter list consists of the address of the MUM communication vector table, the address of the routine for making MUM requests, and the ECB for posting MUM requests. The ECB is the beginning of the communication block for the roll program. The rollable area begins with the remainder of the communication block, register save area, program interruption indicators, and program status word, followed by the program coding.

The roll areas interface with MUM through the roll area monitor, a common re-entrant routine. To make a MUM request the program loads register zero with the request code and register one with the parameter list address, then branches to the address specified in the parameter list. The roll area monitor gets control, saves registers, adjusts register one to point to the communication block, and then issues the MUM SVC. It then waits on the ECB. When the ECB is posted the roll area monitor returns control to the roll program. It is important to realize that when the ECB is posted another program may have been rolled into the roll area. This makes no difference since all program dependent information is in the rollable area.

Most roll programs can only communicate with one terminal at a time. This is because the status of the program is altered by each transaction. For these programs,

MUM will assign a temporary disk location for the program to which it will be rolled. In other words, as each terminal requests the program, a new copy will be created.

Programs which require no information of previous transactions need not be rolled out. Therefore, no tempoaray copy is required. Programs, which are only rolled in, can indicate that they are not reusable. The monitor will then roll in a fresh copy whenever a new request for the program is received.

Certain programs may be rolled back onto the master copy. They must keep the status of each terminal communicating with them. Programs of this nature usually want to keep the program copy on the disk current with the copy in core, so that in the event of an abnormal termination of MUM the copy is current. This would not happen if the program was being used frequently and was never required to be rolled out. To prevent this, the program indicates that it should be checkpointed whenever the roll area goes into the wait state.

Monitor Roll Programs

To decrease the monitor's core requirements certain functions are implemented as roll programs. These programs are not requested by name but are invoked by the monitor

when required. The disk locations and attributes of these programs are stored in the MUM communication vector table. Three functions currently handled by monitor roll programs are signing users on and off, associating a program name with a disk location and attributes, and servicing monitor requests.

The signon/off program consists of four parts: program code, user's attributes, user accounting information, and program directory which consists of a list of program names followed by their disk locations and attributes. When all the application programs associated with a particular terminal have ended their communication with that terminal, the monitor re-associates that terminal with the signon/off program. The signon/off program updates the accounting information, saves the current tab positions, and displays the operator's broadcast message if this user has not yet received it. It then prompts the user for a new program name or signs the user off depending on whether the previous signon was explicit or implicit. When a program is requested, the program directory is searched to determine if the program name is an entry in the directory. If found, it then XCTLs directly to the program. The signon/off program has the attribute of rollin only (no need to roll it out). Of course when any of the user information changes, the

program overrides the rollin only attribute.

As was previously mentioned, control can be passed indirectly to another program by specifying only the requested program's name. The request handler must associate this name with a disk location and attributes. Since the program directory is in the signon/off program the request handler rolls in the signon/off program and requests it to provide the required information.

Monitor requests are serviced by another roll program. To make a request transparent to the application program MUM checks the input data before it issues the rollin request for the application program. If the input data begins with two question marks, the monitor request program is rolled in instead of the application program. The monitor request program services the request, writes a message to the terminal, and issues a read request with the same attributes as the application program.

Resident Programs

Applications, which do not meet the requirements for the roll programs, can be run as separate resident programs. MUM will still handle all terminal communication. The programs will be able to make the same MUM requests as a roll program but of course it will not be swapped. This

facility is also useful for debugging MUM roll application programs, developing and demonstrating on-line applications, and monitoring long running programs.

An interface was developed to allow resident programs to communicate with a terminal as if it were standard unit record equipment. In this way only the job control language needs to be changed to permit any program to communicate with a terminal. This was done by substituting the dummy data set module (IGG019AV) with the interface routine.(4) All references to DD DUMMY would be directed to the terminal.

A resident program, similiar to a roll program, makes MUM requests via the MUM SVC. It first makes an initial request to supply the communication block address. MUM determines the job's name and waits for a terminal user to request a resident program with that name. MUM then associates the terminal with the resident program and posts the ECB in the communication block. From this point the resident program can communicate with the terminal in a manner similar to the roll program. The resident program can communicate with more than one terminal by repeating the initial request with different communication blocks.

Flow of Data and Control Through MUM

The purpose of this section is to trace the flow of data and control through MUM. First we will describe in detail the steps involved in processing a terminal transaction.

1. The terminal inputs a line of data.
2. The TP I/O request handler translates the input and requests the program manager to roll in the required program.
3. When it is possible the dispatcher gives the program manager control to initiate the rolling in of the desired program into a free roll area.
4. The TP I/O request handler receives control when the program is in core. It edits and moves the input data into the program's buffer and then posts the ECB in the communication block.
5. The roll area monitor, which was waiting on this ECB, receives control and passes control to the application program.
6. The application program processes the input data, forms the output message, and makes a terminal write-read request via the roll area monitor.
7. The request servicer gets control, validates the

request, and passes control to the TP I/O request handler.

8. The TP I/O request handler edits the output data and transmits it to the terminal.
9. The TP I/O request handler issues a read to the terminal and waits for another input message.

Now in a less detailed manner, let us follow the flow of data and control between a signon and a signoff.

1. The terminal user enters his account number, program name, and initial input data for the program.
2. The input data is passed to the current program which will be the monitor signon/off roll program.
3. After validating the account number the signon/off program searches the program directory for the requested program. When the program entry is found the signon/off program XCTLs to the requested program specifying the disk location and passing the intial input data.
4. The Passing Control request handler terminates the signon/off program and requests the rolling in of the requested application program.
5. The application program is passed the input data and from this point on can communicate with the terminal

as detailed above.

6. When the application program wishes to end it makes a termination request. The Termination request handler terminates the application program and XCTLs to signon/off program.
7. The signon/off program receives control, updates the user accounting information, and requests the user to enter a new account number and program name.

Maintenance

Since an on-line system usually is required to be operational for extended periods, most maintenance procedures should be performed while the system is on-line. While on-line, MUM allows account numbers to be added or deleted and roll programs to be modified.

The system configuration can not be dynamically altered except for the case of omitting terminals when MUM is initiated. Changes to the configuration are accomplished by modifying one module which requires a reassembly and link edit. Assembler macros make the generation of additional control blocks relatively easy.

Development Aids

Although not an integral part of a system, the ease with which modifications can be made and tested is an important consideration in the implementation of a system.

With MUM, new monitor facilities may be added as separate modules. This still enables them to share common routines that reside in the main MUM module but does not require modification to the main module.

DSECTs describing MUM's control blocks and OS Assembler macros are provided to aid in the writing of MUM programs. The use of DSECTs is very important when a modification to a control block is required. Providing that a program accesses a control block via a DSECT a reassembly of the program is all that is required to maintain its integrity.

To test a modification or develop a new application program a system can be generated which allows the card reader and line printer to simulate a terminal. This is particularly valuable when a long script is required to fully test the program. More than one terminal can be simulated if required.

A program, which resides in the MUM program library, can be added, modified, renamed, or deleted while the system is on-line. A utility program, given the program's name,

will load it onto the proper disk location.

MUM attempts to trap all abends of a roll program. A core dump is produced and control of the terminal is returned to the monitor.

A program was developed which allows the user to display or modify the contents of core storage. This has proved to be a very useful debugging tool and should have been the first application program written.

CHAPTER V

EXPERIENCE GAINED FROM IMPLEMENTATION

As a system is developed and used its good and bad features become apparent; MUM is no exception.

Communication with the Terminal User

The first thing that was noticed was the time required for a user to achieve his results. This of course depends on response time but also on the amount of information that is required to be entered by the user. On low-speed terminals the amount typed out will also affect this time. Response time was minimized in designing MUM but not enough attention was paid to the dialogue with the user. The experienced user wishes to type as little as possible and requires very little information to determine an error. On the other hand the novice wishes to enter requests using common words and requires self-explanatory messages. A well designed system should meet the requirements of both users. This could be accomplished by allowing the program to provide short and long informative messages. The monitor would display one of the messages depending on a user defined attribute. The

experienced user would still be able to receive the long message if he could not understand the short one. This method was partially implemented to decrease the time between program termination and program initiation, but was not provided as a general facility.

Command Language

The second problem was the command language implemented by the various application programs. Rules for command abbreviation, syntax, and valid delimiters varied from one program to another. This provided much freedom to the programmer but was confusing to the user. This problem could be lessened if a common command parsing routine were used by all of the application programs.

Synchronous Processing

Although individual requests to MUM are processed in an asynchronous manner, the processing of the requests involved in one terminal transaction is basically synchronous. For example the program makes a terminal I/O request, it is

rolled out, the terminal I/O is initiated, and so on. This synchronization is carried over to the function control block which is used to store information about the terminal and the program. Difficulty was encountered when the terminal was allowed to be active at the same time as the program since a number of the fields in the FCB were then used for both operations. This problem was corrected so that output operations, but not input operations, could occur asynchronously with the execution of the program. The terminal buffer is also used to keep the input data until a program is rolled in. This problem of course could be solved by having two buffers.

Buffer Pool

MUM requires buffers to be unsegmented. As the number of buffers increases, the efficient use of the buffer space decreases. This is because a large number of messages are short and unsegmented buffers must be the size of the maximum message. If the buffers were segmented a buffer pool could be maintained and requirements for a large buffer could be fulfilled with a number of small buffers.

A buffer pool would also permit the output of

continuous lines of data to be made more efficient. To accomplish this the application program would be allowed to fill additional buffers while the current buffer is being sent to the terminal. When the current buffer has been transmitted there would be another buffer waiting to be transmitted. Besides improving output efficiency, the roll program would not have to be rolled into core as frequently since it could fill more than one buffer with each rollin.

Abnormal Program Termination

Another improvement would be the handling of abnormal program termination. As MUM developed, the number of causes of abnormal termination increased. No standard method of passing an abend code was used, which of course proved a problem to the user when a program abended. Also, if MUM is required to terminate a program there should be a method of passing control to a termination routine if special termination action is required.

Passwords

As was stated earlier, the signon password can be changed by the user. Since the password is entered at each signon, there is a high probability that an unauthorized person may see the password; therefore has access not only to the use of the account number, but also the ability to change the password thus barring a valid user from signing on. This could be prevented by not allowing the user to change his password, but have him request some authorized person to change it. A more flexible method would be to require that a second password be specified whenever the signon password is changed. Since the second password would be specified infrequently, there is a smaller probability of its discovery.

Statistics

Run time statistics, which are not currently available, would also be very useful in determining system utilization. These statistics should indicate response times, utilization of terminals, error rates, activity maximums, and queue occupancy times.

Modularity

On the positive side the modular design of MUM made the addition of new facilities and terminal types very easy. This along with the ease of writing and adding application programs reduced the development time required for new applications.

Passing Control

The method of passing control was useful externally and internally. Externally it is very convenient for the user to temporarily suspend communication with one program, call another program then return and continue with the first program. Internally the ability to intercept the terminal I/O was important since modifications to the input or output formats could be done without modifing developed programs.

CHAPTER VI

CONCLUSIONS

MUM has been used successfully at the university for the past two and a half years. The current maximum number of simultaneous users is twenty-five, but the average maximum is around seventeen. The current implementation does not support more than one roll area. This is the probable cause for an observed degradation in response time when the number of signed on users approaches fifteen. The observed response time for less than ten users averages from instantaneous to 2 seconds. When response time is greater than expected, it is usually because of interference by some other task in the operating system.

User response to MUM has been good. An example of this is the fact that one out of four jobs submitted for batch execution come from MUM terminals.

The efficiency of MUM is also good. A basic system will execute in 40K bytes. The CPU overhead is approximately 1-2% for ten to fifteen active terminals.

Appendix A gives more details on the terminals supported, core requirements, and application programs.

The current status and the growth of MUM are indications that the design and implementation of the MUM

system are basically sound.

APPENDIX A: Current Status

MUM has currently been used at the university for the past two and one half years. The number and types of terminals supported are:

Operator console	1
IBM 2260 cathode ray tube (CRT)	5
IBM 2741 typewriter - leased line	8
IBM 2741 typewriter - dial line	8
IBM 1030 card and badge reader	2
Teletypewriter Exchange Service (TWX)	3
Total	26

The core requirements in units of 1024 bytes are:

Monitor	7.1
One roll area	7.1
CRT support	.5
Graphic access method for CRT	4.2
Operator console support	.2
TWX and 2741 support	1.0
1030 support	.8
Control blocks and buffers	11.7

File handler(5)	10.0
Library circulation system(9)	8.2
Access methods and buffers for library.	26.0
Total	84.0

Some of the application programs which have been developed and are in current use provide the user with the following facilities.

1. EDIT - Card image files, which are stored on disk, can be created, retrieved, and updated on-line. The files can then be submitted to HASP for batch execution.(5)
2. PRINT - The SYSOUT data sets for a batch job can be listed and then deleted or released to be printed on the system line printer.(6)
3. STAT - The batch execution queues and individual job information can be displayed.(7)
4. DSUTIL and LIST - OS data sets can be listed, deleted, renamed, cataloged, and uncataloged. Data set label information and member names in a partitioned data set directory can also be displayed.(6,8)
5. DUMP - System control blocks and core locations can be displayed or modified.(7,8)
6. ETEXT and TEXT - By suppling additional format

commands with the card image file, a formatted typed output is possible.(5) ETEXT was used to type this thesis.

7. MSG - Messages can be sent to one or more users. They are saved on disk and are displayed at the user's request.(7)
8. TABLE - Mathematical table look up and a desk calculator.(5)
9. TEACH - Computer aided instruction.(5)
10. CIRC - A roll program along with a resident module handles all the daily loans, returns, holds, inquiries, and renewals for the book circulation at the main campus library.(9)

A resident file handler provides access to the card image files used in applications 1, 6, and 9.(5)

The above description of the application programs is included for completeness and should not be considered as part of the requirements for this thesis.

APPENDIX B. MUM Control Block Layouts

The line describing each field in the control block consists of four parts:

1. Displacement - decimal (hexadecimal) displacements of each field from the beginning of control block. If blank then same as previous field. This is used where the second field is not aligned to a full word boundary.
2. Length - decimal number of bytes in field.
3. Name - name of field used in the Assembler DSECT.
4. Description - description of field.

MUM Communication Vector Table

0	4	MCVTCB	Address of MUM TCB
4	4	MCVTSTIM	Time of day when MUM was loaded
8	4	MCVTDCB	Address of DCB open list
12(C)	4	MCVTEDIT	Address of buffer edit routine
16(10)	4	MCVTHPLA	Address of HASP parameter list
20(14)	4	MCVTRESQ	Head of rollin queue for resident programs
24(18)	2		Reserved for future use
26(1A)	4	MCVTHEXT	Highest external request code
28(1C)	4	MCVTEXTR	Address of external request table
32(20)	12	MCVTREQU	Pointers for request queue
44(2C)	4	MCVTRINQ	Head of rollin queue for relocatable roll programs
48(30)	12	MCVTRAPT	Pointers for the Resident Application Program Table
60(3C)	12	MCVTRAT	Pointers for the Roll Areas Table
72(48)	4	MCVTTIP	Disk location and program attributes for the signon/off program
76(4C)	4	MCVTMREQ	Disk location and program attributes for the monitor request program
80(50)	4		Reserved for future use
84(54)	4	MCVTRSL	Address of Translate Vector Table -1
88(58)	4	MCVTDINT	Address of Device Interface Vector Table
92(5C)	4	MCVTFCB1	Address of first Function Control Block
96(60)	12	MCVTECBL	Pointers for ECB list
108(6C)	4	MCVTDECB	Address of start of dynamic part of ECB list

112(70)	4	MCVTECBE	Address of last valid entry in ECB list
116(74)	8		Reserved for future use
124(7C)	4	MCVTMAP	Address of map indicating disk swap areas in use
128(80)	2	MCVTDISP	Beginning disk location for swap areas
130(82)	2	MCVTMAPL	Number of available disk swap areas
132(84)	4	MCVTMDCB	Address of DCB for MSG program
136(88)	4	MCVTHASP	Number and address of HASP DCBs
138(8C)	2	MCVTSVC	MUM SVC instruction, NOP if SVC not available

Function Control Block

0	4	FCBNME	FCB name
4	2	FCBLNK	Can be used to link associated FCBs
6	2	FCBCHAIN	Displacement from this FCB to the next FCB, last FCB points to the first
8	2	FCBSTAT	Status bytes
10(A)	1	FCBTYPE	Code for FCB type
11(B)	1	FCBMOD	Modifier byte of a MUM request
12(C)	4	FCBQ	Used to insert FCB in a queue
16(10)	4	FCBRTN	Location to save a return address
20(14)	4	FCBECB	Can be used as an ECB
	2	FCBCOMST	Save area for disk location during a common function
22(16)	2	FCBCOMPA	Save area for program attributes during a common function
24(18)	1	FCBCOMRA	Save area for RAT or RAPT index during a common function
25(19)	1	FCBHOLD	Hold count, when zero can be rolled in
26(1A)	1	FCBPRTY	Priority of FCB
27(1B)	1	FCBDINT	Index into Device Interface Vector Table
28(1C)	2	FCBSTOR	Disk location of current program, is negative for resident programs
30(1E)	2	FCBPATR	Program attributes
32(20)	1	FCBRAT	Index into RAT or RAPT
33(21)	1	FCBMOD2	Save area for FCBMOD during ??LINK
34(22)	2	FCBSDQ	Displacement to next FCB in send queue for this terminal
36(24)	4	FCBCPU	CPU time used by user (currently not used)
40(28)	2	FCBMSGL	Message length for current write or

			completed read
42(1A)	2	FCBACCT	Account number of current user, zero if no user has signed on
44(2C)	16	FCBFHAN	Used by the file handler
60(3C)	4	FCBTAB	Current tab settings
64(40)	4	FCBSIGON	Time of day current user signed on
68(44)	1	FCBTRSL	Index into Translation Vector Table
	4	FCBUFF	Address of terminal buffer
72(48)	2	FCBUFSZ	Buffer size
74(4A)	2	FCBPOS	Postion of carriage on line, negative if unknown
76(4C)	2	FCBLNSZ	Terminal line size
78(4E)	2	FCBSTOR2	Save area for disk location during a LINK request
80(50)	2	FCBPATR2	Save area for program attributes during a LINK request
82(52)	1	FCBRAT2	Save area for RAT or RAPT index during a LINK request
83(53)	1	FCBTPST	Status for TP I/O
84(54)	1	FCBER	Error code to be posted to program when it is rolled in
85(55)	1	FCBRLN	Releative line number for DEB (not currently used)
86(56)	1	FCBDEV	Terminal characteristics
87(57)	1	FCBDEVST	Terminal status
88(58)	4	FCBIORTN	Return address when TP I/O completes
92(5C)	4	FCBIOECB	ECB to be posted when TP I/O completes

Roll Area Table

The RAT consists of one entry for each roll area. Each entry contains the following fields.

0	1	RATID	Entry id
	4	RATCB	TCB address of task controlling this roll area
4	4	RATCBA	Communication Block address
8	4	RATFCBA	Address of FCB currently associated with this area
12(C)	1	RATSTAT	Status of this roll area
	4	RATAD	Address of roll area
16(10)	4	RATQUE	Head of rollin queue for non-relocatable programs for this area
20(14)	2	RATSTOR	Disk location of current program, negative if program not usable

Resident Application Program Table

The RAPT consists of one entry for each terminal-program association. Each entry contains the following fields.

0	1	RAPTID	Entry id
	4	RAPTCB	Address of TCB for resident program
4	4	RAPTCBA	Comunication Block address
8	1	RAPTSTAT	Status of entry
	4	RAPTFCBA	Address of FCB associated with program
12(C)	8	RAPTNME	Job name of resident program
20(14)	4	RAPTRTN	Address of cancel routine
24(18)	4	RAPTECB	ECB to post when cancel request is received

Device Interface Table

0	4	DINTBASE	Content of base register for routines

The following fields contain displacements from the DINT address to the routine which does the indicated function.

4	2	DINTNEWL	New line
6	2	DINTLF	Line feed
8	2	DINTPAGE	Begin new page
10(A)	2	DINTSETW	Setup write
12(C)	2	DINTWLW	Issue write when last I/O was write
14(E)	2	DINTWLR	Issue write when last I/O was read
16(10)	2	DINTSETR	Setup read
18(12)	2	DINTRLW	Issue read when last I/O was write
20(14)	2	DINTRLR	Issue read when last I/O was read
22(16)	2	DINTINTR	Setup interrupt read
24(18)	2	DINTMSGL	Determine input message length
26(1A)	2	DINTHLTW	Halt write
28(1C)	2	DINTHLTR	Halt read
30(1E)	2	DINTERW	Analyse a write error
32(20)	2	DINTERR	Analyse a read error
34(22)	2	DINTAFTR	Exit after input has been translated

The following fields contain the character which will perform the indicated function.

36(24)	1	DINTDELL	Delete input line
37(5)	1	DINTDEL	Delete character from input

38(26)	1	DINTNL	New line
39(27)	1	DINTBS	Backspace

Communication Block

0	4	CBECB	Event Control Block
4	4	CBFCBN	Name of FCB associated with program
8	4	CBFCBA	Address of FCB
12(C)	20	CBPARM	Parameter area
	4	CBPARM1	Pararmeter one
16(10)	4	CBPARM2	Parameter two
20(14)	4	CBPARM3	Parameter three
24(18)	4	CBPARM4	Parameter four
28(1C)	4	CBPARM5	Parameter five

Parameter List Passed to Roll Programs

0	4	PLMCVT	Addrres of MUM CVT
4	4	PLMUMREQ	Address of routine for making MUM requests
8	4	PLECB	Event Control Block
12(C)	4	PLNME	FCB name
16(10)	1	PLDEV	Terminal characteristics
)	4	PLFCBA	Address of FCB
20(14)	4	PLPARM1	Parameter one, usually output buffer address
24(18)	4	PLPARM2	Parameter two, usually output length
28(1C)	4	PLPARM3	Parameter three, usually input buffer address
32(20)	4	PLPARM4	Parameter four, usually input buffer length
36(24)	4	PLPARM5	Parameter five

Roll Area

0	4	AREARAT	Address of entry in RAT
4	72	AREASAV	Save area pointed to by register 13 when program is first entered
76(4C)	40	AREAPARM	Parameter list
84(54)	32	AREACB	Communication Block
88(58)	7294	AREASIO	Rollable area
116(74)	64	AREAREG	Register save area when program is

			swapped out
180(B4)	4	AREASPIE	Spie mask and displacement from AREACB of exit routine
184(B8)	8	AREAPSW	Resume PSW for program
192(C0)	7290	AREAPRG	Program area

APPENDIX C: MUM Macros

MUM macros can be divided into the following groups:

- macros which can be used in the writing of any MUM program,
- macros which are used specifically in an application roll program,
- macros which generate and initialize control blocks.

General MUM Macros

$COMMENT - is used to place a comment card in front of the program's object deck. Besides a comment, the card contains the time and date of the assembly. Whenever the object decks are link edited the contents of the card are printed.

$DSECT - is used to give a selection or all of the available DSECTs for MUM.

$EQU - is used to give a selection or all of the groups of EQUs commonly used in MUM.

$LPSW - makes a request to MUM to modify the current program status word to give key zero, problem program key, supervisor state, or problem program state.

$PARM - contains installation specified parameters which

may be used in assembling MUM programs. An example would be to specify the MUM SVC number.

$TRUP - generates a routine which will capitalize a character string by logically oring it with blanks.

Roll Program Macros

$BEGIN - defines and initializes the data area required at the beginning of the roll program. The user can also specify the establishing of a base register, the loading of the address of the FCB and/or MUM CVT, and the saving of the address of the parameter list passed to the roll program from the roll area monitor.

$DELINK - loads register zero with the request code and modifier for a DELINK request.

$EDIT - sets up the linkage and branches to a resident buffer edit routine. This macro can be used by a roll program which wants to look at the input data before the monitor edits it.

$IF - has the effect of the following statement: IF ... THEN GO TO .... The user can check the return code from a monitor request, the device type of the terminal currently associated with the program, and the status of a LINK or SWAP request.

$LINK - loads register zero with the request code and modifier for a LINK request.

$MSG - generates a formatted message. The user can specify more than one line or part of a message.

$MUMREQ - sets up the linkage for making a monitor request. The user can also specify the registers that should be relocated in the event of resuming execution in another roll area.

$PAUSE - load register zero with the request code for a pause request.

$RSTORE1 - loads register one with the address of the parameter list passed by the roll area monitor. It loads the address from the area where the macro $BEGIN saved it.

$SEND - loads register zero with the request code to send a message to a terminal other than the one that is currently associated with the program.

$SPIE - allows the user to change the program interruptions that should be trapped and the address of the routine which will handle the interruption.

$SWAP - loads register zero with the request code and modifier for a SWAP request.

$TPIO - loads register zero with the request code and modifier to communicate with the terminal currently

associated with the program.

$XCTL - loads register zero with request code and modifier for an XCTL request.

Macros for Generating Control Blocks

The following macros with the exception of $DINT are all located within one module. This simplifies modifing the system configuration.

$ADDMCVT - generates an extension to the MUM Communication Vector Table. Installation dependent additions would be included in this macro. These additions are inserted at the end of the normal CVT by the $CVT macro.

$CVT - generates the MUM Communication Vector Table.

$DINT - generates the Device Interface Table and will also establish addressibility for the interface routine.

$ECBL - generates the ECB list. The addresses of the ECBs which will be in the permanent section are specified along with the size of the dynamic section.

$EXTR - generates the list of addresses of the external request handlers. The user also specifies if the request servicer should load the registers with the FCB address, RAT or RAPT entry address, and the CB

address.

$FCB - generates the basic function control block. All macros which generate FCBs use this macro to generate the basic section.

$OPER - generates a FCB which is used to communicate with the computer operator's console.

$RAPT - generates a resident application program table with the specified number of entries.

$RAT - generates a roll area table with the specified number of entries.

$REQU - generates the request server queue. There must be one additional entry for every task, which makes a MUM request, with a higher CPU dispatching priority than MUM.

$TEST - generates a FCB for simulating a terminal with a card reader and printer.

$2260 - generates a FCB for communicating with an IBM 2260 (CRT).

$2741 - generates a FCB for communicating with an IBM 2741 (typewriter).

APPENDIX D. Sample Application Roll Program

Below is an Assembler listing of a sample program. Following the listing is the output of the test system which simulates a terminal via the card reader and line printer.

The sample program demonstrates the use of the $BEGIN and $MUMREQ macro which are required for every application program.

The program also shows the ease of writing to the terminal as well as intercepting terminal I/O via the LINK request.

```
LOC  OBJECT CODE     ADDR1 ADDR2  STMT   SOURCE STATEMENT

                                    1 ***************************************************
                                    2 *                                                 *
                                    3 * THIS IS A SAMPLE MUM APPLICATION ROLL PROGRAM    *
                                    4 *                                                 *
                                    5 * GIVEN A CHARACTER STRING IT WILL DO THE FOLLOWING: *
                                    6 * 1. REPEAT THE CHARACTER STRING                  *
                                    7 * 2. LINK TO ITSELF PASSING REPEATED STRING      *
                                    8 * 3. THE LINKED PROGRAM WILL REPEAT STRING AND ISSUE *
                                    9 *       A WRITE END                               *
                                   10 * 4. THE WRITE IS INTERCEPTED AND THE INITIALLY    *
                                   11 *       CALLED PROGRAM WILL END WITH A WRITE REQUEST*
                                   12 *                                                 *
                                   13 ***************************************************

                                   15          $DSECT PL


                                   17+*  MUM  PARAMETER LIST PASSED TO APPLICATION ROLL PROGRAMS
000000                             18+MUMPL    DSECT
000000                             19+PLMCVT   DS    F MUM CVT ADDRESS
000004                             20+PLMUMREQ DS    F MONITOR REQUEST ADDRESS
000008                             21+PLECB    DS    F ECB
00000C                             22+PLNME    DS    CL4 FCB NAME
000010                             23+PLDEV    DS    0C DEVICE CHAR
000010                             24+PLFCBA   DS    F FCB ADDRESS
000014                             25+PLPARM1  DS    F *
000018                             26+PLPARM2  DS    F *
00001C                             27+PLPARM3  DS    F *PARAMETERS
000020                             28+PLPARM4  DS    F *
000024                             29+PLPARM5  DS    F *
000000                             30 REPEAT   CSECT
                                   31          $BEGIN  BASE=12,INPUT=BUFFER,LENGTH=L'BUFFER
                                   32+         ENTRY $RCLLSIO
000000 0000000000000000            33+$RCLLSIO DC    4F'0'
000010 00000124                    34+         DC    A(BUFFER-$ROLLSIO+12) INPUT AREA DISPLACEMENT
000014 00000064                    35+         DC    A(L'BUFFER) AREA LENGTH
000018 00000000                    36+         DC    F'0' PARMS
00001C 0000000000000000            37+         DC    15F'0' REG0-14
000058 0000006C                    38+         DC    F'108'
00005C 0000                        39+         DC    B'0000000000000000' INTERUPTION MASK
00005E 0000                        40+         DC    AL2(0)
000060 0000000000000000            41+         DC    F'0',B'00000000',AL3(0)
000068                             42+         DS    0H
000068 05C0                        43+         BALR  12,0
00006A                             44+         USING *,12
00006A 5010 C00A           00074   45+         ST    1,$MUMPARM
00006E 47F0 C00E           00078   46+         B     $MUMPARM+4
000072 0000
000074 00000000                    47+$MUMPARM DC    F'0'
000000                             48          USING  MUMPL,1        ESTABLISH ADDRESSIBILITY
000078 9823 101C           0001C   49          LM    R2,R3,PLPARM3   INPUT ADDRESS AND LENGTH
00007C 1253                        50          LTR   R5,R3           IS LENGTH ZERO?
00007E 4780 C08C           000F6   51          BZ    NOINPUT         YES BRANCH
000082 4162 3000           00000   52          LA    R6,0(R2,R3)     END OF INPUT STRING
```

```
   LOC  OBJECT CODE     ADDR1 ADDR2  STMT   SOURCE STATEMENT

000086 1A33                          53          AR     R3,R3           REPEATED LENGTH
000088 4930 C116             00180   54          CH     R3,=AL2(L'BUFFER) FIT IN BUFFER?
00008C 47D0 C032             0009C   55          BNH    OK                YES BRANCH
000090 4130 0062             00062   56          LA     R3,L'BUFFER-2   MOVE TO END OF BUFFER-2
                                     57 *                               LEAVE ROOM FOR FORMATTED MSG
000094 1155                          58          LNR    R5,R5
000096 1A53                          59          AR     R5,R3           # OF CHARACTERS TO MOVE
000098 47D0 C038             000A2   60          BNP    NOMOVE          IF NOTHING TO MOVE BRANCH
00009C 0650                          61 CK       BCTR   R5,0            -1 FOR EX
00009E 4450 C094             000FE   62          EX     R5,MOVE         REPEAT STRING
                                     63 NOMOVE   $IF    LINK,ENDMSG     BEEN LINKED TO?
0000A2 58F1 0010             00010   64+NOMOVE   L      15,16(1) FCB ADDRESS
0000A6 9102 F008       00008         65+         TM     8(15),2
0000AA 4750 C080             000EA   66+         BC     5,ENDMSG
0000AE 4140 C0AE             00118   67          LA     R4,BUFFER       RETURN BUFFER ADDRESS
0000B2 4150 0064             00064   68          LA     R5,L'BUFFER     BUFFER LENGTH
0000B6 4160 C09A             00104   69          LA     R6,NAME         NAME OF PROGRAM
                                     70          $LINK  INTER=WR        LOAD LINK REQUEST CODE
0000BA 4100 0682             00682   71+         LA     0,1666
0000BE 45A0 C060             000CA   72          BAL    LINK,MUMREQ     GO AND DO IT
0000C2 4120 C0AE             00118   73          LA     R2,BUFFER       INTERCEPTED WRITE MSG
                                     74 WRTEND   $TPIO  (W,F,E)         LOAD TPIO REQUEST CODE
0000C6 4800 C118             00182   75+WRTEND   LH     0,=H'9601'
                                     76 MUMREQ   $MUMREQ PARM=R2,REL=(LINK,12) RELOCATE LINK, BASE
0000CA 5810 C00A             00074   77+MUMREQ   L      1,$MUMPARM
0000CE 9026 1014             00014   78+         STM    R2,R2+4,20(1)
0000D2 58F1 0004             00004   79+         L      15,4(1)
0000D6 41E0 C076             000E0   80+         LA     14,*+(2*2)+6
0000DA 1FCE                          81+         SLR    12,14
0000DC 1FAE                          82+         SLR    LINK,14
0000DE 07FF                          83+         BR     15
0000E0 1ECE                          84+         ALR    12,14
0000E2 1EAE                          85+         ALR    LINK,14
0000E4 5010 C00A             00074   86+         ST     1,$MUMPARM UPDATE ONE SAVE AREA
0000E8 07FA                          87          BR     LINK

0000EA 4230 C0AC             00116   89 ENDMSG   STC    R3,MSG          STORE LENGTH FOR FORMATTED WRITE
0000EE 4120 C0AC             00116   90          LA     R2,MSG          MESSAGE ADDRESS
0000F2 47F0 C05C             000C6   91          B      WRTEND

0000F6 4120 C0A2             0010C   93 NOINPUT  LA     R2,NOINMSG
0000FA 47F0 C05C             000C6   94          B      WRTEND

0000FE D200 6000 2000 00000  00000   96 MOVE      MVC    0(*-*,R6),0(R2)

000104 D9C5D7C5C1E34040              98 NAME      DC     CL8'REPEAT'
                                     99 NOINMSG  $MSG   (ST,'NO INPUT',NL)
00010C 08D1D5D64CC9D5D7             100+NOINMSG  DC     AL.16((*M00101-(*+2))*256+2C9),C'NO INPUT'
000116                              101+*M00101  EQU    *
                                    102 MSG      $MSG   (ST,,NL)
000116 00D1                         103+MSG      DC     AL.16(209)
000118                              104 BUFFER   DS     CL100

000002                              106 R2       EQU    2               REGISTER 2
000003                              107 R3       EQU    3               REGISTER 3
```

```
 LOC   OBJECT CODE     ADDR1 ADDR2  STMT   SOURCE STATEMENT

000004                             108 R4       EQU   4              REGISTER 4
000005                             109 R5       EQU   5              REGISTER 5
000006                             110 R6       EQU   6              REGISTER 6
00000A                             111 LINK     EQU   10             RETURN REGISTER FOR BAL
                                   112          END
000180 0064                        113                 =AL2(L'BUFFER)
000182 2581                        114                 =H'9601'
```

CROSS-REFERENCE

SYMBOL LENGTH,VALUE,DEFINITION REFERENCES

```
$MUMPARM  4,74,47 45 46 77 86    $M00101  1,116,101 100    $ROLLSIO  4,0,33 32 34
BUFFER  100,118,104 34 35 54 56 67 68 73 113    ENDMSG  4,EA,89 66    LINK  1,A,111 72 82 85 87
MOVE  6,FE,96 62    MSG  2,116,103 89 90    MUMPL  1,0,18 48    MUMREQ  4,CA,77 72    NAME  8,104,98 69
NOINMSG  2,10C,10C 93    NOINPUT  4,F6,93 51    NOMOVE  4,A2,64 60    OK  2,9C,61 55    PLDEV  1,10,23
PLECB  4,8,21    PLECBA  4,10,24    PLMCVT  4,0,19    PLMUMREQ  4,4,20    PLNME  4,C,22
PLPARM1  4,14,25    PLPARM2  4,18,26    PLPARM3  4,1C,27 49    PLPARM4  4,20,28    PLPARM5  4,24,29
REPEAT  1,0,30    R2  1,2,106 49 52 73 78 78 90 93 96    R3  1,3,107 49 50 52 53 53 54 56 59 89
R4  1,4,108 67    R5  1,5,109 50 58 58 59 61 62 68    R6  1,6,110 52 69 96    WRTEND  4,C6,75 91 94
```

NO STATEMENTS FLAGGED IN THIS ASSEMBLY

```
ENTER ACCT #,PRG NAME
*IN*ON 1 REPEAT
NO INPUT
OS
ENTER PRG NAME
*IN*REPEAT ABCD
ABCDABCDABCDABCD
1S
ENTER PRG NAME
*IN*REPEAT 1111122222333334444455555666667777
1111122222333334444455555666667777711111222223333344444555556666
OS
ENTER PRG NAME
*IN*REPEAT ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRST
OS
ENTER PRG NAME
```

GLOSSARY AND ABBREVIATIONS

Abend - abnormal termination.

Buffer - An area of core storage that is used to temporarily store information during a transfer of information to or from an input/output device. It is used to compensate for a difference in the rate of flow of information, or the time of occurence of events.

CB - Communication Block.

Checkpointing - To record the status of a program so that it can be restarted later.

Control block - A block of consecutive memory in which control information is kept for the purpose of efficient communication among the modules of a system.

CPU - central processing unit.

CRT - cathode ray tube.

Data set - The major unit of data storage and retrieval in the operating system.

DCB - Data Control Block.

DINT - Device Interface Table.

EBCDIC - Extended Binary Coded Decimal Interchange Code.

ECB - Event Control Block.

FCB - Function Control Block.

HASP - Houston Automatic Spooling and Priority System.

I/O - Input/Output.

MCVT - MUM Communication Vector Table.

MFT - Multiprogramming with a fixed number of tasks.

Multiprogramming - A technique for handling numerous routines or programs seemingly simultaneously by overlapping or interleaving their execution, that is, by permitting more than one program to time-share machine components.

MUM - Manitoba University Monitor.

MVT - Multiprogramming with a variable number of tasks.

On-line system - A system in which there is a direct path between the user and the computer.

OS - IBM/360 Operating System.

RAPT - Resident Application Program Table.

RAT - Roll Area Table.

Real Time system - A system which will respond to input sufficiently quickly to affect the functioning of the environment at that time.

Re-entrant - The attribute of a program which allows the same copy of the program to be used concurrently by two or more tasks.

Resident program - A program which resides in core

storage for the duration of its execution.

Response Time - The time the system takes to react to a given input. The interval between an event and the system's response to that event.

Roll program - A program which executes under the control of MUM and which may be transferred from or to backing storage whenever it is inactive.

Time-sharing - A system where the user's terminal appears to be connected to a dedicated computer and the user can enter, test, and execute programs.

TCB - Task Control Block.

TP - teleprocessing.

XCTL - transfer control.

BIBLIOGRAPHY

A. References

1. Abraham, C. Manitoba University Monitor. Scientific Reports No. 1, University of Manitoba, Winnipeg, Canada, September 1970.

2. Martin, James. Design of Real-Time Computer Sytems. Prentice-Hall, Inc., Englewood Cliffs, New Jersy, 1967. p. 5.

3. Ibid., p. 46.

4. IBM/360 Operating System Sequential Access Methods. No. Y28-6604-1. IBM Corp. Programming Systems Publications, Poughkeepsie, N. Y. p. 78

5. Program implemented by Dr. C. Abraham.

6. Program implemented by P. A. Macdonald.

7. Program implemented by G. Neufeld.

8. Program implemented by W. Reid.

9. System implemented by the Computer Centre under the direction of W. Doran.

B. General References

Margopoulas, W. P., and others. "On Teleprocessing System Design," IBM Systems Journal, Vol. 5, No. 3, 1966, IBM Corp., Armonk, N. Y.

Martin, James. Teleprocessing Network Organization. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1970.

Thomas, Dennis V. "Teleprocessing Control Program Standard Interface." Preliminary Specifications TPS68-1, IBM Winnipeg, 1968.

van Dam, Anries and David E. Rice. "On-line Text Editing: A Survey," ACM Computing Surveys, Vol. 3, No. 3, September 1971, Association for Computing Machinery, Baltimore.

C. MUM Manuals

Collens, R. J. (ed.). MUM User's Guide. University of Manitoba, Winnipeg, Canada, February 1971.

Reid, W. "MUM Application Programmer's Manual," Ibid., 1971.

Rugger, K. MUM Implementation Guide. Ibid., February 1971.

D. Other On-line Systems

Administrative Terminal System - OS: Program Description Manual. No. H20-0582, IBM Corp. Technical Publications Department, White Plains, N. Y.

APL/360 -Operating System- User's Manual. No. GH20-0683, IBM Corp. Technical Publications Department, White Plains, N.Y.

Cocchi, H., and others. Conversational Remote Batch Entry (CRBE). No. 360D-05.1.016, IBM Corp. Program Information Department, Hawthorne, N. Y.

IBM System/360 Operating System: Time Sharing Option Guide. No. GC28-6698-3, IBM Corp. Programming Systems Publication, Poughkeepsie, N. Y.

Morrison D. T., J. Griffin, and A. M. Thurston. HOMBRE. Central Data Processing Service Bureau, Government of Canada, Ottawa, 1968.

Schroeter, Steve. WITS Users Guide. Third Edition. University of Waterloo, Canada, March 1969.

Smith, Kenneth. "Remote Visual Displays," Use of the IBM-360, University of Nebraska, Lincoln, Nebraska, 1968.

Wood, John R., and others. Interactive Applications Supervisor. No. 360d-05.2.010, IBM Corp. Program Information Department, Hawthorne, N. Y., January 1969.