MOSTEK®

Z80 MICROCOMPUTER SOFTWARE

Operations Manual

MITE-80
MULTIPLE INDEPENDENT TASK EXECUTIVE

MOSTEK MITE-80 OPERATION MANUAL

Publication No. MK 79726

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

LIST OF TABLES

SECTION 1

GENERAL DESCRIPTION

1.1 INTRODUCTION

MITE-80 is the MOSTEK Multiple Independent Task Executive Z80 software package. It provides the basic services for managing the CPU's resources in an orderly fashion. MITE-80 accommodates MDX applications requiring real-time multiple asynchronous event handling.

1.1.1 FEATURES

The highlighted features of MITE-80 are:

- Simplistic data structures.
- Fast context switching between tasks.
- Up to 127 priority levels for task execution.
- Message queuing by several options.
- Handles unlimited number of tasks.
- Accommodates interrupt driven device handlers.
- Can be stored in RAM, EPROM, and/or ROM.
- A DEBUG version for assistance in application development.
- Memory Pool Manager and Timer Handler.

1.1.2 SOFTWARE CONFIGURATION

MITE-80 is designed to work with the following minimum hardware:

- MOSTEK MDX-CPU CARD.

MITE-80 DEBUG is designed to work with either a MOSTEK Development System having a FLP-80DOS software package or an MDX Card system having a FLP-80DOS/MDX software package. Refer to the FLP-80DOS/MDX Operation Manual for the MDX Cards required.

1.2 REFERENCE DOCUMENTS

FLP-80 DOS Operation Manual	MK79668
FLP-80 DOS Operation Manual	MK78557
MDX-CPUI Operation Manual	MK79612
Micro Components Data Book	MK79801

1.3 DEFINITION OF SYMBOLS USED IN THIS MANUAL

The following conventions are used throughout this manual:

1. Most hexadecimal numbers are identified by the character 'H' following the hexadecimal numbers.

2. aaaa indicates any hexadecimal number.

3. <CR> Stands for "carriage return".

4. All user input is underlined.

1.4 PRODUCT OVERVIEW

The MITE-80 software package is provided on a floppy diskette in IBM 3740 single density format. The files can be read using a MOSTEK FLP-80DOS system software package. The MITE-80 software package contains the following files:

MITE-80
MITE-80 DEBUG
MITE-80 MACROS
MITE-80 EQUATES
TIMER HANDLER
MEMORY POOL MANAGER
MITE-80 SYSTEM LINKAGES

A brief overview of each file follows:

1.4.1 MITE-80

The MITE-80 program is the multi-tasking nucleus which provides the capability for controlling multiple real-time tasks. In providing this control MITE-80 uses two data structures; a Task Control Block and a Message Block. User tasks communicate with each other through MITE-80 by a series of system services.

1.4.2 MITE-80 DEBUG

The MITE-80 DEBUG program provides the facility for interactively debugging Z80 programs executing under MITE-80. The various commands allow for displaying and modifying memory and CPU registers, for executing programs, for setting breakpoints on tasks and services, and for displaying Task Control Block and Message Block contents. MITE-80 DEBUG includes a special version of MITE-80 which contains loopdetect logic and certain integrity checks.

1.4.3 MITE-80 MACROS

A macro file is provided to aid the user in defining and developing MITE-80 Task Control Blocks (TCB) and memory pools. The macros create a TCB from user specified parameters. The macros will also generate code which will install the TCB into the MITE-80 system.

1.4.4 MITE-80 EQUATES

An equate file is provided to facilitate using MITE-80. The file includes globals for all of the MITE-80 system entry points and data structures and defines the entries in each data structure.

1.4.5 MITE-80 TIMER HANDLER

A Timer Handler is provided for control over the MK3882 Counter Timer Circuit (CTC) Chip. The timer can be used for applications requiring event time delays or for event watchdog alerts. The timer executes as a MITE-80 task. A maximum time duration of 13.9 minutes in increments of 12.8 milliseconds is provided. The time increment is a user-configurable parameter.

1.4.6 MITE-80 MEMORY POOL MANAGER

A Memory Pool Manager is provided as a way to allocate and deallocate memory blocks. The user can configure an area of memory into up to 252 memory pools,

with each pool consisting of a unique memory block size. A macro is provided to aid the user in configuring the pools.

1.4.7 MITE-80 SYSTEM LINKAGES

A system linkage file is provided which contains linkage addresses for the debug version of MITE-80. The user can link unique application programs to MITE-80DEBUG with this file.

1.5 DOCUMENT FORMAT

The following sections detail the areas of functional description, data structures, system services, configuration requirements, and debug usage.

SECTION 2

FUNCTIONAL DESCRIPTION

2.1 OVERVIEW

This section provides an operational overview of MITE-80.

2.2 TASK IDENTIFICATION

An application is divided into modules, with each module being a defined processing chore. Each module can be identified as a task within MITE-80. All the tasks share memory and compete for the CPU's processing time. MITE-80 allocates the CPU's time based on information provided about each task. Each task is uniquely identified by a Task Control Block (TCB). The TCB contains task information of status, priority, link to next TCB, message queue pointer, stack pointer, and a task name.

2.3 MESSAGE IDENTIFICATION

Each task is a defined process which may produce intermediate and/or final results required by other tasks. The means by which this data is moved throughout MITE-80 is with a Message Block (MB). An MB is used to pass data to another task or to request service of another task. Each MB is constructed by the task that needs to send information to another task. MITE-80 passes these messages from task to task based on information provided in the MB and in the service call. The MB contains message information of status, priority, queuing, sending task identification, receiving task identification, and any optional data that the sending task may need to provide the receiving task.

2.4 TASK COMMUNICATION

All inter-task communication is performed by a message block. These message blocks are used to pass results to another task, to initiate an I/O request, to create a task, and for any other user-to-user or user-to-system communication. MITE-80 provides several system services which the user can use within each task

for communicating between tasks. The services include the facilities to send, receive, forward, return, wait for, and cancel messages.

2.5 PRIORITY

A priority field is part of both the TCB and MB data structures. MITE-80 uses these priority fields to determine the order of processing importance. The TCB's priority defines the task's level of execution. The MB's priority defines the message's level of importance in a message queue. The priority value ranges from 0 to 126, with 0 being the highest priority and 126 being the lowest priority. Task priority levels 0 thru 15 are normally reserved for system tasks but may be used by high priority user tasks. Each TCB can be assigned a different priority level and any number of TCBs can have the same priority level. The determination of a TCB's priority is dependent on the application requirement's urgency to process the task. Each MB can be assigned a different priority level and any number of MBs can have the same priority level. The determination of an MB's priority is dependent on the application's requirement. In most cases the MB's priority level will be identical to the sending task's TCB priority level.

2.6 TASK STATES

A task can be in any one of three possible states; running, waiting, or ready. At any instant of time there is only one task running; all other tasks are either waiting or ready.

2.6.1 RUNNING TASK

The running task is the task currently using the CPU's resources. The running task was the current highest priority ready task.

2.6.2 READY TASK

A ready task is a task that would be running, but is not because a higher priority task is running. Each ready task must become the current highest priority ready task before MITE-80 will re-allocate the CPU's resources to it.

2.6.3 WAITING TASK

A waiting task is a task that has requested a system service and has specified

the "wait" option. When the event has satisfied the wait option, the waiting task then becomes a ready task.

2.7 MESSAGE QUEUING

The queuing of MB's by MITE-80 is accomplished by two algorithms; message priority and queue type. The priority and the queue type each have two user-specifiable options. For the priority, either specified priority or task priority can be used. And for queue type, either LIFO or FIFO can be used. Both the priority and queue type option are specified in the MB by the task prior to calling a MITE-80 service.

2.7.1 SPECIFIED PRIORITY

A specified priority is a priority level specified by the MB's issuing task. The specified priority can be any priority level of 0 to 126. The higher the priority level (the lower the number), the higher the MB will be placed in the receiving task's message queue.

2.7.2 TASK PRIORITY

A task priority is a priority level specified by MITE-80 when the MB, with a priority of 127, is issued by the task. MITE-80 will take the issuing task's TCB priority value and place it in the MB's priority field. The MB's priority level will then be the same as the issuing task's priority. This feature is only valid for the Send Message services.

2.7.3 LIFO QUEUING

LIFO is a Last-In-First-Out queuing concept. The MB is placed at the top of all MB's having the same priority level within the MB's receiving task message queue.

2.7.4 FIFO QUEUING

FIFO is a First-In-First-Out queuing concept. The MB is placed at the bottom of all MB's having the same priority level within the MB's receiving task message queue.

2.8 THEORY OF OPERATION

2.8.1 OVERVIEW

MITE-80 determines which task gets current use of the CPU resources based on information contained in each of the TCBs within the system. The task whose TCB identifies it as in a ready state and is the current highest priority task will be the running task.

2.8.2 CONTEXT SWITCHING

Whenever an external event happens or whenever an MB is processed by a MITE-80 service, MITE-80 will re-evaluate which task should be the running task. If it is determined that the running task is still the highest priority ready task, the CPU resources are returned to the running task to continue processing. If it is determined that a higher priority task is now a ready task, the lower priority running task will become ready. The higher priority ready task will then become the running task. This task switching is known as context switching. Whenever context switching occurs, MITE-80 performs a CPU register image save and restore operation. The running task's appropriate main or full Z-80 registers are all pushed onto its stack. The ready task's stack is popped to restore its task's Z-80 registers. MITE-80 then gives the CPU resources to the higher priority ready task which then becomes the new running task.

2.9 METHODS OF TASK COMMUNICATION

Communication between tasks can occur either by the transmitting of MBs or by the posting of an interrupt event.

2.9.1 TRANSMITTING MBs

MITE-80 provides several services which are task callable and allow for the transmitting of an MB between tasks; M8SN, M8RCV, M8RET, M8FWD and M8CAN.

2.9.1.1 M8SN

This service is a Send Message service which allows a task to send an MB to another task.

2.9.1.2 M8RCV

This service is a Receive Message service which allows a task to receive an MB

from its MB queue.

2.9.1.3 M8RET
This service is a Return Message service which allows a task that has received an MB to return that MB to the sending task.

2.9.1.4 M8FWD
This service is a Forward Message service which allows a Task to send an MB to another task without altering the MB's sender and receiver pointers.

2.9.1.5 M8RSN
This service is a Resend Message service which allows a task which has previously sent a message to a task to resend that message with a minimum of overhead.

2.9.1.6 M8CAN
This service is a Cancel Message service which allows a task that has previously sent an MB to now cancel, or kill, that MB in the receiving task's MB queue.

2.9.1.7 M8FIND
This service is a Find Message service which allows a task to find another task given the other task's name.

Further information on these and other MITE-80 services is outlined in Section 4, System Services.

2.9.2 EVENT POSTING
MITE-80 provides a service for the waiting and posting of interrupt events; M8WINT and M8PINT.

2.9.2.1 M8WINT
This service is a Wait for Interrupt service which allows a device driver to create a wait until interrupt process is complete. The task will become a waiting task.

2.9.2.2 M8PINT
This service is a Post after Interrupt service which allows an Interrupt Service

Routine or another task to post an interrupt process completion state. The waiting task will become a ready task.

2.10 REGISTER USAGE

A task can use all of the Z-80 registers or only the main set consisting of PC, SP, A, B, C, D, E, F, H, L, IX, and IY. The task's TCB contains a register usage designation. When a context switch is required of this task, MITE-80 will PUSH or POP the appropriate registers as specified in the task's TCB.

2.11 STACK USAGE

Each task has its own stack. When a task becomes a running task, MITE-80 will POP the task's specified registers off the task's stack before CPU control is given to the task. When the task transitions to a wait or ready state, MITE-80 will PUSH the task's specified registers onto the task's stack before the next task is allowed to run. Each task must allocate a stack area sufficient for its unique task requirements (e.g. routine nesting levels) and task registers used.

SECTION 3

DATA STRUCTURES

3.1 INTRODUCTION

This section defines the MITE-80 data structures of Task Control Block and Message Block.

3.2 TASK CONTROL BLOCK (TCB)

The Task Control Block contains the task state information required by MITE-80 in order to make resource allocation decisions. Each task within the system must have its own TCB. The TCB is 10 bytes in length.

3.2.1 STRUCTURE

The TCB's information is structured as:

Field #	# of Bytes	Offset	Name	Field	Data Type	Source
1	1	0	STAT	Task status	Bit encoded	User/MITE-80
2	1	1	PRIO	Task priority	Binary	User
3	2	2	LINK	Next TCB address	Binary	MITE-80
4	2	4	MPTR	Message pointer	Binary	MITE-80
5	2	6	SPTR	Stack pointer	Binary	User
6	2	8	NAME	Name of task	Binary or ASCII	User

Each of these fields is further defined as follows:

3.2.1.1 STAT

The Status byte (STAT) contains information which indicates task state, register usage, and task's message state. The byte is user specified and is bit encoded as follows:

Bit Name	Bit	Definition
WAIT	7	0 = if task not waiting for message 1 = if task is waiting for message
IWAT	6	0 = if task not waiting for an interrupt event 1 = if task is waiting for an interrupt event
----	5	0, reserved for future use
INTQ	4	0 = no interrupt has been posted since last M8WINT call 1 = interrupt has been posted since last M8WINT call
DBUG	3	0 = task running normally 1 = task currently being debugged
MROB	2	0 = if task uses all registers 1 = if task uses only main register set
MHBS	1	0 = if message not sent yet 1 = if message sent but task has not yet run
TCBB	0	0 = if this is not a TCB 1 = if this is a TCB

3.2.1.2 PRIO

The Priority byte (PRIO) contains information which determines where in the TCB queue this TCB is to be placed. The byte is user specified. The priority of an active task should NEVER be modified. If the priority of an active task must be changed, then the task should be cancelled and re-created. Priority level 0 is reserved for MITE-80 system use and must NOT be specified in user tasks. User tasks can have priority level assignments from 0 to 126 inclusive. The priority byte is bit encoded as follows:

Bit	Definition
7-1	00H = priority level of 0 to FCH = priority level of 126
0	1 = always set for TCB

3.2.1.3 LINK

The Link word (LINK) is used by MITE-80 to implement a singly linked list of all TCBs within the TCB queue. The terminator for the list is a null TCB with a priority of 127. The null TCB is linked into the TCB queue by MITE-80 when the TCB queue is created. The null TCB is provided by MITE-80. The Link word's contents are maintained by MITE-80.

3.2.1.4 MPTR

The Message Pointer word (MPTR) is a message list head. All messages sent to this TCB are linked to this message list. All messages to be processed (receive message) by this TCB are delinked from this message list, one MB at a time. The message pointer is initialized to point to a null message block at task creation time by MITE-80. The null message block is provided by MITE-80. The message pointer word's contents are maintained by MITE-80.

3.2.1.5 SPTR

The Stack Pointer word (SPTR) is a pointer to the task's stack area. MITE-80 uses the task's stack to maintain the context of the task. Whenever a context switch occurs, the task registers (including the program counter) are saved on the stack and the Stack Pointer is saved in the TCB's SPTR field. The task registers are retrieved from the stack when the task regains control of the CPU resources. When a task is created, it is the user's responsibility to set-up the TCB, initialize the registers on the stack, and provide sufficient stack space for the new task. Macros have been provided with the MITE-80 package to facilitate setting up the TCB and stack.

The task registers are popped from the stack in the following order; HL', DE', BC', AF', IY, HL, DE, BC, AF, IX, PC. If the task status byte (STAT) indicates task register usage of "Main Registers Only" (MROB bit = 1), then the HL' through AF' registers are not popped from the stack. MITE-80 assumes that these registers do not exist on the stack. This option saves 8 bytes of stack space and reduces system latency.

3.2.1.6 NAME

The task Name word (NAME) is used to identify the task. The task name can range from 0H to FFFFH. Since ASCII characters are included within this range, the following allocation of task names is recommended:

Range (HEX)	Allocate For
0000 - 1FFF	User binary task names
2000 - 7FFF	ASCII task names, both user and system task names (e.g. LP for Line Printer)
8000 - FFFF	User binary task names

The task name is specified by the user for each user task. For binary task names, byte 8 is the least significant byte of task name and byte 9 is the most significant byte of task name. For ASCII task names, byte 8 is the first character of task name and byte 9 is the second character of task name.

3.3 MESSAGE BLOCK (MB)

The Message Block contains message identity information required by MITE-80 in order to route the MB from the issuing task to the recipient task. Each message transmitted within the system must have its own MB. The MB is 8 bytes in length with optional user data expanding its length as necessary to fulfill user application requirements.

3.3.1 STRUCTURE

The MB's information is structured as follows:

Field #	# of Bytes	Offset	Name	Field	Data Type	Source
1	1	0	STAT	Message status	User defined	User
2	1	1	PRIO	Message priority	Bit encoded	User *
3	2	2	LINK	Next MB address	Binary	MITE-80
4	2	4	RPTR	Receiver pointer	Binary	MITE-80
5	2	6	SPTR	Sender pointer	Binary	User *
6	N	8	DATA	Data	User defined	User

* denotes fields for which the user may request MITE-80 to supply the value.

Each of these fields is further defined as follows:

3.3.1.1 STAT

The Status byte (STAT) is used by the receiving task to inform the sending task information about how the message was handled. This byte is defined by the user. The byte can be used for process status results, error conditions, message number tagging, or for any use required by the application.

3.3.1.2 PRIO

The Priority byte (PRIO) contains information which determines where in the receiving task's message queue the sending task's MB is to be placed. The byte is user specified, but the calling task may request MITE-80 to insert a priority value. The byte contains a priority value and a queuing directive. The priority value can be assigned a level 0 to 127, and the queuing can be either LIFO or FIFO. The priority byte is bit encoded as follows:

Bit Name	Bit		Definiton
PRIO	7-1	00H =	priority level of 0
			to
		FCH =	priority level of 126
		FEH =	indicates to MITE-80 to replace this value with sending task's TCB priority level.
FIFO	0	0 =	LIFO queue at PRIO value in message list
		1 =	FIFO queue at PRIO value in message list

NOTE: On LIFO queue, MITE-80, sets the bit after the LIFO queuing operation. A sending task priority of 127 (FEH) can only be specified when using a M8SN[W] Service.

3.3.1.3 LINK

The Link word (LINK) is used by MITE-80 to implement a singly linked list of all MB's within the MB queue. The terminator for the list is a null MB with a priority of 127. The null MB is linked onto the MB queue by MITE-80 when the MB queue is created. The null MB is provided by MITE-80. The link word's contents are maintained by MITE-80.

3.3.1.4 RPTR

The Receiver Pointer word (RPTR) is set to the address of the receiving task's TCB by MITE-80 when the message is sent by the M8SN or M8SNW. The field is used by the M8RSN and M8RSNW to determine the task to which a message is to be re-sent.

3.3.1.5 SPTR

The Sender Pointer word (SPTR) is used by MITE-80 for determining to which task the MB is to be returned. The word is user specified, but the calling task may request MITE-80 to insert the calling task's TCB address in this field. The user can specify this word to return the MB to the sending task or to a different task. If the word is zero when a M8SN or M8SNW service is called, MITE-80 will insert the sending task's TCB address into this field.

3.3.1.6 DATA

The Data bytes (DATA) are used for transmitting user information between tasks. The Data bytes are all user specified and are not affected by MITE-80. The DATA field can be used for passing process directives, process results, I/O vector information, error codes, data blocks, or any user application requirement.

SECTION 4

SYSTEM SERVICES

4.1 INTRODUCTION

This section describes the MITE-80 Services callable from a user task.

4.1.1 OVERVIEW

The Services provided allow a user to perform the following:

A. Send and Receive message blocks.
B. Forward message blocks.
C. Return message blocks.
D. Cancel message blocks.
E. Find task control blocks.
F. Interrupt control handling.
G. Create and Cancel tasks.

The following table summarizes the Services:

SERVICE	DESCRIPTION
M8SN	Send a message to a task.
M8SNW	Send a message to a task and wait for a message to be available for the sending task.
M8RSN	Resend a message to a task.
M8RSNW	Resend a message to a task and wait for a message to be available for the sending task.
M8RCV	Receive a message if one is available for calling task.
M8RCVW	Receive a message, otherwise wait till one is available for calling task.
M8FWD	Forward a message to a task.
M8FWDW	Forward a message to a task and wait for a message to be available for the forwarding task.
M8RET	Return a message to the sending task.
M8RETW	Return a message to the sending task and wait till a message is available for calling task.
M8CAN	Cancel a message sent to a task.
M8FIND	Find the receiver address of the specified task name.
M8WINT	Wait for an interrupt event to be posted.
M8PINT	Post an interrupt event which is completed.

4.2 M8SN - SEND MESSAGE

4.2.1 FORMAT

The calling sequence is:

```
LD      DE,<message receiver address>
LD      BC,<message block address>
CALL    M8SN
```

4.2.2 DESCRIPTION

The M8SN Service will send an MB to a task and return to the calling task. Registers BC must contain the address of the MB to be sent, and registers DE must contain the address of the receiving task's TCB. The contents of registers DE are loaded into the MB's RPTR field by the M8SN Service. The contents of the BC registers are loaded into the SPTR field of the message if the SPTR field contains a zero. CPU control will be immediately returned to the calling task if the task is still the highest priority task ready to run. This service is also used to install a task into the system by having registers DE contain the MITE-80 TCB queue address (M8TCBQ) and registers BC contain the "to be installed" task's TCB address.

4.3 M8SNW - SEND MESSAGE & WAIT

4.3.1 FORMAT

The calling sequence is:

```
LD      DE,<message receiver address>
LD      BC,<message block address>
CALL    M8SNW
```

The Service will return:

```
HL = <message block address>
```

4.3.2 DESCRIPTION

The M8SNW Service will send an MB to a task and return to the calling task when an MB is available. Registers BC must contain the address of the MB to be sent, and registers DE must contain the address of the receiving task's TCB. The contents

of registers DE are loaded into the MB's RPTR field by the M8SN Service. The contents of the BC registers are loaded into the SPTR field of the message if the SPTR field contains a zero. CPU control will be returned to the calling task only when an MB is available for the calling task. On return, registers HL will contain the address of the MB to process. This service is equivalent to an M8SN followed by a M8RCVW.

4.4 M8RSN - RESEND MESSAGE

4.4.1 FORMAT

The calling sequence is:

```
LD      BC,<message block address>
CALL    M8RSN
```

4.4.2 DESCRIPTION

The M8RSN Service will resend an MB to a task and immediately return to the calling task. The MB to be resent is an MB that has its RPTR correctly specified. Registers BC must contain the address of the MB to be resent. This Service is similar to M8SN except that this Service uses the contents of the RPTR field of the MB as the receiving task address. CPU control will be immediately returned to the calling task if the task is still the highest priority task ready to run.

4.5 M8RSNW - RESEND MESSAGE & WAIT

4.5.1 FORMAT

The calling sequence is:

```
LD      BC,<message block address>
CALL    M8RSNW
```

The Service will return:

```
HL = <message block address>
```

4.5.2 DESCRIPTION

The M8RSNW Service will resend an MB to a task and return to the calling task

when an MB is available. The MB to be resent is an MB that has its RPTR correctly specified. Registers BC must contain the address of the MB to be resent. This Service is similar to M8SNW except that this Service uses the contents of the RPTR field of the MB as the receiving task address. CPU control will be returned to the calling task only when an MB is available for the calling task. On return, registers HL will contain the address of the MB to process. This service is equivalent to a M8RSN followed by a M8RCVW.

4.6 M8RCV - RECEIVE MESSAGE

4.6.1 FORMAT

The calling sequence is:

```
CALL M8RCV
```

The Service will return:

```
HL     = <message block address> or 0000H
Z Flag = set if no message received or,
         reset if a message received
```

4.6.2 DESCRIPTION

The M8RCV Service will get the next MB, if one is available, for this task to process and immediately return to the calling task. If an MB is available for the calling task, the Service will provide the MB's address; otherwise, an indication ofno MB's available is given. CPU control will be returned to the calling task. On return, registers HL will contain either the address of the next MB to process or zeroes if no MB's are available. The Zero Flag will be reset if an MB is received; otherwise, the flag will be set.

4.7 M8RCVW - RECEIVE MESSAGE & WAIT

4.7.1 FORMAT

The calling sequence is:

```
CALL M8RCVW
```

The Service will return:

HL = <message block address>

4.7.2 DESCRIPTION

The M8RCVW Service will return to the calling task only when an MB is available for the task. On return, registers HL contain the address of the MB to be processed. If no MB exists for the calling task, CPU control is given to the next highest priority ready task.

4.8 M8FWD - FORWARD MESSAGE

4.8.1 FORMAT

The calling sequence is:

```
LD      DE,<message receiver address>
LD      BC,<message block address>
CALL    M8FWD
```

4.8.2 DESCRIPTION

The M8FWD Service will forward an MB to a task and return to the calling task. The difference between M8FWD and M8SN is that M8FWD leaves the MB intact and does not use or affect the MB's RPTR field. Registers BC must contain the address of the MB to be forwarded, and registers DE must contain the address of the receiving task's TCB, that is the "to be forwarded to" task. CPU control will be immediately returned to the calling task if the task is still the highest priority task ready to run. This Service is used to forward a receiver MB on to another task without altering the MB being forwarded. Note that neither the RPTR or SPTR fields are affected by this service.

4.9 M8FWDW - FORWARD MESSAGE & WAIT

4.9.1 FORMAT

The calling sequence is:

```
LD      DE,<message receiver address>
LD      BC,<message block address>
CALL    M8FWDW
```

The Service will return:

HL = <message block address>

4.9.2 DESCRIPTION

The M8FWDW Service will forward an MB to a task and return to the calling task when an MB is available. The difference between M8FWDW and M8SNW is that M8FWDW leaves the MB intact and does not use the MB's RPTR field. Registers BC must contain the address of the MB to be forwarded, and registers DE must contain the address of the receiving task's TCB, that is the "to be forwarded to" task. CPU control will be returned to the calling task only when an MB is available for the calling task. On return, registers HL will contain the address of the MB to process. This Service is used to forward a received MB on to another task without the MB being altered. This service is equivalent to an M8FWD followed by a M8RCVW. Note that neither the RPTR or SPTR fields are affected by this service.

4.10 M8RET - RETURN MESSAGE

4.10.1 FORMAT

The calling sequence is:

```
LD      BC,<message block address>
CALL    M8RET
```

4.10.2 DESCRIPTION

The M8RET Service will return a received MB to the sending task. The Service is normally used by a task that has received a MB, and has to return the MB to the sender task. Registers BC must contain the address of the MB to be returned. CPU control will return to the calling task if it is still the highest priority ready task. Note that neither the RPTR or SPTR fields are affected by this service.

4.11 M8RETW - RETURN MESSAGE & WAIT

4.11.1 FORMAT

The calling sequence is:

```
LD      BC,<message block address>
CALL    M8RETW
```

The Service will return:

```
HL = <message block address>
```

4.11.2 DESCRIPTION

The M8RETW Service will return a received MB to the sending task and will get the next MB, if available, for the calling task. This Service is normally used by a task that has received an MB (either by M8RCV or M8RCVW), has processed the MB, has to return the MB to the sender task, and needs to receive the next MB to process. Registers BC must contain the address of the MB to be returned. On return, registers HL contain the address of the next MB to be processed. If no MB exists for the calling task, CPU control is given to the next highest priority ready task. This Service is equivalent to M8RET followed by a M8RCVW. Note that neither the RPTR or SPTR fields are affected by this service.

4.12 M8CAN - CANCEL MESSAGE

4.12.1 FORMAT

The calling sequence is:

```
LD      DE,<message receiver address>
LD      BC,<message block address>
CALL    M8CAN
```

The Service will return:

```
HL = <message block address> or 0000H
Z Flag = reset if service successful, or set if
         unsuccessful
```

4.12.2 DESCRIPTION

The M8CAN Service will cancel an MB from a task's MB queue. Registers BC must

contain the address of the MB to be cancelled, and registers DE must contain the address of the receiver's TCB. On return if the MB is cancelled, registers HL will contain the address of the cancelled MB and the Zero Flag will be reset. Otherwise, if the MB is not cancelled, registers HL will be zeroed and the Zero Flag will be set. An unsuccessful cancel of an MB is a result of the MB not being on the specified receiver's task MB queue.

This service can also be used for cancelling a task from the system by having registers DE contain the MITE-80 TCB queue address and by having registers BC contain the TCB address of the task to be cancelled.

4.13 M8FIND - FIND RECEIVER

4.13.1 FORMAT

The calling sequence is:

```
LD      DE,<queue header>
LD      BC,<name of entry>
CALL    M8FIND
```

The Service will return:

```
HL = <receiver TCB address>, or 0000H
Z Flag = reset if service successful, set if unsuccessful
```

4.13.2 DESCRIPTION

The M8FIND Service will search a queue for a match to an entry name and return the address of entry if found. Registers DE must contain the address of the queue header to be searched, and registers BC must contain the entry name to be searched for. On return, registers HL will contain the address of the entry name's TCB and the Zero Flag will be reset. Otherwise, if the search is unsuccessful, registers HL will be zeroed and the Zero Flag will be set. This Service is used to find the address of a receiving task.

4.14 M8WINT - WAIT FOR INTERRUPT

4.14.1 FORMAT

The calling sequence is:

```
CALL M8WINT
```

4.14.2 DESCRIPTION

The M8WINT Service is for use by device driver tasks. The driver task conditions the device's Interrupt Service Routine (ISR) to interrupt when the current operation is complete: for example, at end of block, or on carriage return character detection. The driver calls this Service which will place this task in an interrupt wait state until the ISR posts a complete status (see M8PINT Service). This service will return immediately if an ISR has already posted the calling task's interrupt bit (INTQ).

4.15 M8PINT - POST INTERRUPT

4.15.1 FORMAT

The calling sequence is:

```
PUSH  PC
PUSH  IX
PUSH  AF
PUSH  BC
PUSH  DE
PUSH  HL
LD    IX,<address of TCB to post>
JP    M8PINT
```

4.15.2 DESCRIPTION

The M8PINT Service is for use by Interrupt Service Routines (ISR). This Service is the complement to M8WINT Service. When an ISR completes the requested operation the ISR will normally jump to the M8PINT Service. The M8PINT service will save the task's IY register contents on the task's stacks. M8PINT issues a RETI for the ISR at operation completion time.

SECTION 5

USING THE SYSTEM SERVICES

5.1 OVERVIEW

This section provides information and examples on how to use the various MITE-80 system services. The areas covered include:

A. Establishing and Cancelling a Task.
B. Sending and Receiving MBs.
C. Tasks waiting states.
D. ISR processing.

5.2 ESTABLISHING A TASK

5.2.1 DESCRIPTION

To establish a task into the system a TCB must first be created. This is accomplished by either constructing one in RAM, or by transferring a pre-constructed one in ROM to RAM. The TCB's address is then sent to the MITE-80 TCB queue and is then placed in the TCB queue at the specified priority level.

5.2.2 EXAMPLE A

To configure a TCB whose address is NEWTCB into the system, and a TCB queue name of M8TCBQ, the code sequence would be:

```
LD      DE,M8TCBQ       ;Set-up TCB Queue addr
LD      BC,NEWTCB       ;Set-up TCB addr
CALL    M8SN            ;Place NEWTCB into system
```

On return from the system service the TCB is then configured into the system, and the Task is available for use. Note that the Send Message Service, M8SN, is used to establish a new TCB into the system. The TCB address replaces the MB address

and the TCB queue address is the message receiver address.

Care must be taken when installing a task. If the "to be installed task" is of a higher priority than the task installing it, then CPU control will be given to the newly installed task.

5.2.3 EXAMPLE B

Prior to installing a task, its initial register values must be set-up on the task's stack since MITE-80 pops the task's stack before CPU control is given to the new task. If the task does not require any initial register value set-ups, then these registers need not be initialized. However, the minimal stack set-up requirement is the loading of the task's entry point into its stack's Program Counter location.

For a task which uses only the main register set, has an entry point label of UTSP, the code sequence to initialize the PC in the task's stack would be:

```
        LD      HL,UTASK        ;UTASK ENTRY POINT
        LD      (UTSP+12),HL    ;INIT UTASK'S SP PC LOCATION
```

5.2.4 PROGRAMMING NOTES

The TCB constructed MUST adhere to the TCB data structure.

The Task's TCB must NOT be changed once it is placed into the TCB queue. If the TCB has to be changed, then cancel the TCB, alter it, and send it again to the TCB queue.

The task's stack must be initialized; at minimum the task's entry point must be loaded into its stack's PC location, prior to installing the task.

Tasks should normally be installed using the M8SN service.

5.3 CANCELLING A TASK

5.3.1 DESCRIPTION

To cancel a task from the system the TCB's address must be known. The address can be obtained by using the M8FIND service to find the address by using the TCB's name for comparison. Once found, the TCB address is then used to cancel the TCB from the TCB queue.

5.3.2 EXAMPLE

To cancel a TCB whose name is AB from the system, and a TCB queue name of M8TCBQ, the code sequence would be:

```
LD      DE,M8TCBQ       ;Set-up TCB queue addr
LD      BC,'BA'         ;Set-up TCB name
CALL    M8FIND          ;Find TCB address
JP      Z,TNIS          ;Jump if TCB not in system
LD      B,H             ;Move TCB addr to BC
LD      C,L             ;
CALL    M8CAN           ;Cancel task 'AB'
```

A check is made after the M8FIND service call to assure that the TCB was still in the system. On return from the M8CAN service the TCB has been cancelled and the Task is no longer available for use.

5.3.3 PROGRAMMING NOTES

Care must be taken when cancelling a task:

If conditional cancelling is desired then the TCB must first be checked to assure that the Task does not have a MB to process and that it is not waiting for a post interrupt event.

If unconditional cancelling is desired then any unprocessed MBs for this TCB will be LOST. Requeuing a task by means of the M8FWD Service will allow unprocessed MBs to be found again.

5.4 SENDING A MESSAGE

5.4.1 DESCRIPTION

To send a message to a task a MB must first be created. This is accomplished by either constructing one in RAM, or by transferring a pre-constructed one in ROM to RAM. If the receiver's address is not known, the M8FIND service is used to find the address. Once the MB is constructed several options exist in which to send the MB to the receiver; send and immediate return, or send and wait for another MB. Regardless of the option selected, the service will queue the MB at the specified priority in the receiver Task's MB queue.

5.4.2 EXAMPLE A

To send a MB whose address is TMB2 to a receiver task MC whose TCB address is MCTCBA the code sequence would be:

```
LD      DE,MCTCBA       ;Set-up tasks MC's TCB addr
LD      BC,TMB2         ;Set-up MB addr
CALL    M8SN            ;Send TMB2 to Task MC
```

In this example the calling task will retain CPU control if it is still the highest priority ready task after sending the MB. The calling task can then continue processing its other requirements.

5.4.3 EXAMPLE B

If in Example A further processing is to be suspended after sending the MB until another MB is available for the sending task, then the code sequence would be:

```
LD      DE,MCTCBA       ;Set-up tasks MC's TCB addr
LD      BC,TMB2         ;Set-up MB addr
CALL    M8SNW           ;Send TMB2 and wait till another MB is
                        ;available
```

In this example the calling task will retain CPU control only if another MB is available for the calling task to process and if it is still the highest priority ready task after sending the MB. When a MB is available for the task, CPU control will return to the instruction immediately following the call instruction and the address of the next MB to process will be in registers HL.

5.4.4 EXAMPLE C

If in Example A the sending task wanted to send the MB to the last task to which it had been sent, then the code sequence would be:

```
LD      BC,TMB2      ;Set-up MB addr
CALL    M8RSN        ;Send TMB2 to Task MC
```

In this example the M8RSN service will retrieve the MCTCBA value from the MB's RPTR field and set-up the DE registers. For MBs with their RPTR field already initialized, this service can be used to minimize execution time and memory overhead on service set-up requirements. The wait option, as shown in Example B, is also available by using the M8RSNW service call.

5.4.5 EXAMPLE D

If in Examples A or B the receiver's TCB address was not known, the Find Receiver Service could be used to get the value. The code sequence would be:

```
LD      DE,M8TCBQ      ;Set-up TCB queue addr
LD      BC,'BA'        ;Set-up TCB name
CALL    M8FIND         ;Find TCB address
JP      Z,TABNIS       ;Jump if TCB 'AB' not in system
EX      DE,HL          ;Move TCB addr to DE
LD      BC,TMB2        ;Set-up MB address
CALL    M8SN (or M8SNW)          ;Send TMB2 to Task AB
```

In this example the system TCB queue (M8TCBQ) is searched for Task's 'AB' address before the Send message service is set-up and called. Also, a check is made on return from the M8FIND service to assure that task AB's TCB is still in the system.

5.4.6 PROGRAMMING NOTES

The MB constructed MUST adhere to the MB data structure.

The MB must NOT be changed once it is sent to the receiver. If the MB has to be changed, then cancel the MB, alter it, and send it again to the receiver.

The wait option will suspend (wait state) the sending task if no MBs are currently available for the sending task to process.

On return from the service with a wait option specified, the MB address returned is the highest priority MB to process, NOT necessarily the MB that was last sent with the service call wait option. If a task must know which MB has been received, then a MB tag numbering or process level status scheme in the MB's data or STATUS field should be used.

5.5 RECEIVING A MESSAGE

5.5.1 DESCRIPTION

To receive a message from any task a receiving task must condition its TCB to receive messages. If a task does not condition itself to receive messages, then all MBs sent to the task will be queued in the task's MB queue. To receive messages a task must request an MB from its queue everytime an MB is to be processed. There are no service set-up requirements. Two options exist on the service call: an immediate return regardless of whether or not an MB is available, and a wait option until a MB is available. Regardless of the option selected, if a MB is returned to the calling task it will be the highest priority MB in the task's queue. The receiving task is responsible for returning the MB back to the sending task, if required of the application.

5.5.2 EXAMPLE A

To receive a MB the code sequence would be:

```
      CALL     M8RCV      ;Get next MB if available
```

In this example the calling task will retain CPU control. On return, register HL will contain either zeros for no MB available, or the address of the MB to process. Also, the Zero Flag is set if no MB is available.

5.5.3 EXAMPLE B

To receive an MB and return to caller only if an MB is available, the code sequencewould be:

```
CALL    M8RCVW      ;Get next MB or wait if none available
```

In this example the calling task will retain CPU control only if an MB is available. Otherwise, the calling task will be suspended (wait state) until an MB is available. When an MB is available for the task, the task will then be in a ready state. On return the HL registers contain the address of the MB to process. The M8RCVW service will generally be the first service call made by a MB processing task.

5.5.4 PROGRAMMING NOTES

The MB received will be the highest priority MB from the calling task's MB queue.

An immediate knowledge of whether the MB was initially sent by the receiving task or by another task can be determined. If the SPTR field does NOT match the receiver Task's address, then the MB was sent to it by another task. If the SPTR field DOES match the receiver task's address, the receiving task was the original sender task of the MB.

A task CANNOT receive any messages to process until a Receive Message Service is called. For MBs sent to tasks which have not yet called a receive message service, their MBs will be queued in the task's MB queue.

5.6 FORWARDING A MESSAGE

5.6.1 DESCRIPTION

To forward a message to another task, the receiver's address must be known. If theforward receiver's address is not known, the M8FIND service can be used to find the address. Once the receivers address is known, it is placed in the DE registers and the MB to be forwarded is left intact. The MB can then be forwarded to the receiver task with one of the two options, either forward and immediate return, or forward and wait for another MB. Regardless of the option selected, the service will queue the MB at the specified priority in the forwarded receiver task's MB queue. The forward service is used in applications where the MB specified receiving task cannot process or handle the MB, and the MB is forwarded on, intact, to another task for processing.

5.6.2 EXAMPLE

To forward an MB whose address is TMB2 to a receiver task MC whose TCB address is MCTCBA, the code sequence would be:

```
LD      DE,MCTCBA       ; Set-up tasks MC's TCB addr
LD      BC,TMB2         ; Set-up MB addr
CALL    M8FWD           ; Forward TMB2 to Task MC
```

In this example the calling task will retain CPU control if it is still the highest priority ready task after forwarding the MB. The forwarding task can then continue processing its other requirements. If the forwarding task only wanted CPU control and another MB was available to process, then the M8FWDW service would be called. Note that the received MB was left intact, thus allowing the next receiving task to return the MB to the original sender task. Any MB can be forwarded any number of times before it is returned, if required by the application, to the original sender task.

5.6.3 PROGRAMMING NOTES

The MB forwarded MUST remain intact (unmodified) if the MB is to be returned to the original sender task.

The wait option will suspend (wait state) the forwarding task if no more MBs are currently available for the forwarding task to process.

A forwarded MB will NOT be returned to the task forwarding the MB, since the MB's fields of RPTR and SPTR (receiver and sender pointers) remain intact from the original sender task.

The MB to be forwarded must NOT have a priority of 127 (FEH) since queue by task priority only works for M8SN and M8SNW.

5.7 CANCELLING A MESSAGE

5.7.1 DESCRIPTION

To cancel a message sent to a task the receiver's TCB address and the "to be cancelled" MB address must be known.

5.7.2 EXAMPLE

To cancel an MB whose address is TMB2 from a receiver whose TCB address is MCTCBA, the code sequence would be:

```
LD      DE,MCTCBA      ;Set-up task's TCB addr
LD      BC,TMB2        ;Set-up MB addr
CALL    M8CAN
```

On return from M8CAN, a check should be made to verify that the MB was cancelled from the receiver task's MB queue. If the Zero Flag is reset the cancel request was successful; otherwise, the Zero Flag will be set to indicate an unsuccessful request.

5.7.3 PROGRAMMING NOTES

Always check the Z Flag on return from the M8CAN Service to verify that the MB was cancelled.

An unsuccessful cancel request indicates that the MB does not exist in receiver's MB queue (which could mean that the message is currently being processed by the receiver).

5.8 ISR PROCESSING

5.8.1 DESCRIPTION

Two services facilitate ISR processing; M8WINT AND M8PINT. M8WINT is for tasks such as I/O drivers which condition an ISR to start and execute until a terminating event occurs (end of block, carriage return, character present, etc). This will place the task into an interrupt wait state until an ISR process posts the event being completed, M8PINT. The waiting task then becomes a ready task.

5.8.2 EXAMPLE

For a data entry driver to wait for an event of a character being entered, the code sequence would be:

```
DEDRVR   CALL    M8RCVW          ;Get next MB or wait if none
         :                       ;available
         :                       Decode MB request command
         :
         LD      HL,DBFFRC       ;Driver buffer count addr
         LD      (HL),0          ;Initialize count to zero
         LD      HL,DBSPP        ;Driver buffer starting pointer
         LD      (DBFFRP),HL     ;Init buffer pointer to start addr
         LD      C,DEPRTI        ;Get Data Entry Port Addr for
                                 ;Interrupt
         LD      A,INTE          ;Get interrupt enable bit code
         OUT     (C),A           ;Enable card's interrupt
         CALL    M8WINT          ;Wait for character
         :
         :

; Data Entry Driver's ISR
DEDISR   PUSH IX                 ;Save the registers
         PUSH AF
         PUSH BC
         PUSH DE
         PUSH HL
         IN A,(DEINP)            ;Input character
         LD (DECTS),A            ;Save in temp area
         LD IX,DETCB             ;Get Data Entry Driver TCB address
         JP M8PINT               ;Post "character in" event complete
```

5.8.3 PROGRAMMING NOTES

The posting interrupt event task MUST push the main register set, except for the IY register, before jumping to M8PINT.

Always JUMP to the M8PINT service.

If the wait and post interrupt services are used for non-driver ISR purposes, such as a synchronization mechanism between task-task, then the wait and post processes within these tasks MUST follow the same guidelines.

A task performing an M8WINT will be placed into an interrupt wait state until another task or ISR process posts the event completion by an M8PINT service.

SECTION 6

MITE-80 DEBUG

6.1 INTRODUCTION

This section describes the functions and operation of MITE-80 DEBUG, a software package which provides the user with a means of debugging programs running under the control of MITE-80. MITE-80 DEBUG is a combination of 2 primary programs; a modified MITE-80 and a modified DDT-80 (Designer's Development Tool 80). The modified MITE-80 portion is the application version of MITE-80 with checkpoint features added for debug purposes. The modified DDT-80 portion is the DDT-80 package of FLP-80DOS with additional commands added for MITE-80 service debug purposes.

6.2 SOFTWARE CONFIGURATION

MITE-80 DEBUG is a program that is provided on a diskette, named M80DDT.BIN[X], where X is 16, 32, 48, or 64; the memory size in K (1024 bytes) of the debug system. MITE-80 DEBUG is approximately 4000 bytes in size and resides at the top of RAM as specified by [X]. In addition to program area, MITE-80 DEBUG uses 256 bytes of RAM for scratch RAM and temporary storage. This RAM resides at locations FF00H-FFFFH. An additional RAM area is used for a historical record of services performed and requires 512 bytes of RAM (included in the 4000 bytes.)

The 256 byte RAM area is used by MITE-80 DEBUG for temporary storage, a push down stack, and system data structures. This RAM also holds an image, or map, of all user's internal CPU registers, and a jump vector for all MITE-80 Services.

The 512 byte RAM area is used as a task history table. This table is a circular list which contains the calling task registers, task address, and service name for each MITE-80 service as it is entered or exited. This table provides a historical trail of pertinent information as tasks utilize the MITE-80 services.

To preserve the state of the CPU for a user's program while debugging, MITE-80 DEBUG keeps an image or map of all the user's registers. This image or map is referred to as the User Register Map. MITE-80 DEBUG installs or makes the CPU registers equal to the user register map when control is transferred from MITE-80 DEBUG to a user program. MITE-80 DEBUG saves the user register map when MITE-80 DEBUG is commanded (breakpoint) to interrupt a user program. MITE-80 DEBUG allows modification to this register map with the display and/or update memory command. The user register map resides in the 256x8 RAM area, locations FFE6H thru FFFFH, as follows:

Location	User Register
FFFF	Program Counter (MSB)
FFFE	Program Counter (LSB)
FFFD	A
FFFC	F
FFFB	I
FFFA	IF
FFF9	B
FFF8	C
FFF7	D
FFF6	E
FFF5	H
FFF4	L
FFF3	A'

Location	User Register
FFF2	F'
FFF1	B'
FFF0	C'
FFEF	D'
FFEE	E'
FFED	H'
FFEC	L'
FFEB	IX (MSB)
FFEA	IX (LSB)
FFE9	IY (MSB)
FFE8	IY (LSB)
FFE7	Stack Pointer (MSB)
FFE6	Stack pointer (LSB)

6.3 COMMAND FORMATS

MITE-80 DEBUG operation is similar to that of DDT-80, and many of the commands are identical to those of DDT-80. Further information on DDT-80 can be found in the FLP-80DOS Operation Manual. Users who are already familiar with FLP-80DOS DDT-80 need only be concerned with the added commands and operational differences that MITE-80 DEBUG provides. However, all of the commands are outlined in this section with those commands identical to DDT-80 summarized. MITE-80 DEBUG recognizes commands which consist of three parts:

1. A single letter identifer.
2. An operand, or operands separated by commas or blanks.
3. A terminator to either abort the command or cause it to be executed.

In order to execute MITE-80 DEBUG, enter at the monitor level:

$M80DDT<CR>

MITE-80 DEBUG will then prompt with a colon (:). Any of the MITE-80 DEBUG commands may then be executed. MITE-80 DEBUG echos the command letter, prints a space, and then waits for the user to key-in the appropriate operand(s) in the format described below. A command is not executed until terminated by a carriage return (or one of the special terminators described below for display and update commands) and may be aborted at any time by a period. MITE-80 DEBUG automatically supplies a line feed for the carriage return.

MITE-80 DEBUG may also be loaded and entered directly from DDT-80 by looking up the track and sector address from a PIP directory command and using the stand-alone loader in DDT-80 to load MITE-80 DEBUG. When using this method, the entry point and load address for MITE-80 DEBUG can be determined from the following table:

VERSION	ENTRY POINT
16K	2B00
32K	6B00
48K	AB00
64K	CB00

6.3.1 COMMAND IDENTIFIERS

The following summarizes the 17 different single letter identifiers recognized as command identifiers:

1. B - Insert a breakpoint in the user's program (must be in RAM) which transfers control back to MITE-80 DEBUG. This allows the user to intercept his program at a specific point and examine memory and CPU registers to determine if his program is working correctly.

2. C - Copy the contents of a block of memory to another location in memory.

3. D - Display the history table of last Tasks executed and MITE-80 services used.

4. E - Transfer control from MITE-80 DEBUG to a user's program.

5. F - Fill memory limits with an 8 bit data pattern.

6. H - Perform 16 bit hexadecimal addition and/or subtraction.

7. J - Snap shot a Task Control Block and/or a Message Block.

8. K - Breakpoint on a Task and/or a MITE-80 service.

9. L - Locate all occurrences of an 8 or 16 bit data pattern.

10. M - Display, update, or tabulate the contents of memory.

11. O - Set the offset constant.

12. P - Display and/or update the contents of an I/O port.

13. Q - Quit MITE-80 DEBUG and return to the system Monitor.

14. R - Display the contents of the user registers.

15. V - Verify that 2 blocks of memory are equal.

16. W - Software single step (walk).

17. X - Duplicate output to printer device.

All of the above commands, with the exception of the D, J, K, and X commands, are identical to the corresponding DDT-80 commands. A colon (:) prompt is used instead of a period (.) as in DDT-80.

6.3.2 COMMAND OPERANDS

MITE-80 DEBUG command operands are described in this section. The user is referred to the FLP-80DOS Operation Manual for additional information and examples.

A command operand represents 4 hexadecimal digits; e.g. aaaa. MITE-80 DEBUG allows arithmetic expressions (addition and/or subtraction). The 4 hexadecimal digits, aaaa, can be calculated with a string of additions and/or subtractions. The values in the string may be entered in one of the following forms:

1. 0-9, A-F — hexadecimal digits

2. : — a prefix of ":" before an alphacharacter specifies that the next 1 or 2 characters are mnemonic and is equivalent to 4 hex digits

3. $ — represents current address +1. This is valid for the M command and is used to calculate relative jump displacements.

4. L — a prefix of "L" before a character specifies that the ASCII value of the next 1 or 2 characters is to be used.

5. R — a prefix of "R" specifies the relative address. This causes the offset specified by the 'O' command to be added to the number entered.

6. xxxxxx — xxxxxx is 4 to 6 alphanumeric characters which represent a MITE-80 Service call label which is to be used as the command operand. This form is only valid for the 'K' command.

An equal sign '=' may be entered at any time within the string to display the operand value as 4 hexadecimal digits.

Examples of typical operands are:

1. 4F7F — The operand value is equal to 4F7FH.

2. :PC — The mnemonic PC is equivalent to address FFFEH and the operand value is equal to the contents of FFFEH and FFFFH.

3. 5038-5000 — The operand value will be 38H.

4. 5038-5000=0038 — The same as 3 except '=' was entered to display the operand value.

5. 5038-$ — If current address = 5000H, then $=5001H and the operand value equals 37H.

6. LAB=4241 — Operand is equal to the ASCII value of 'AB'.

7. R100=1100 — Assuming offset = 1000H.

8. M8SN — The operand value is used to obtain the address of Send Message Service.

A mnemonic is composed of 1 or 2 characters following a colon (:) and represents a 4 hexadecimal digit address. Table 6-1 lists the mnemonics recognized by MITE-80

DEBUG. Mnemonics are equivalent to a 4 hex digit address and the data at that address may represent either a single or double byte value (marked by * in the table). Table 6-2 lists the MITE-80 service call labels recognized by MITE-80 DEBUG. The labels are equivalent to 4 hex digits which represent the address of the service. If a command requires more than one operand, those operands have to be separated by either a blank or a comma.

TABLE 6-1

MNEMONICS RECOGNIZED BY MITE-80 DEBUG.

Unrecognized mnemonics are resolved with a value of zero.

MNEMONIC	ADDRESS REPRESENTED BY THE MNEMONIC	DATA SAVED AT THAT ADDRESS
:PC*	FFFE	User's PC Register
:A	FFFD	User's A Register
:F	FFFC	User's F Register
:I	FFFB	User's I Register
:IF	FFFA	User's IFF Register
:B	FFF9	User's B Register
:C	FFF8	User's C Register
:D	FFF7	User's D Register
:E	FFF6	User's E Register
:H	FFF5	User's H Register
:L	FFF4	User's L Register
:A'	FFF3	User's A' Register
:F'	FFF2	User's F' Register
:B'	FFF1	User's B' Register
:C'	FFF0	User's C' Register
:D'	FFEF	User's D' Register
:E'	FFEE	User's E' Register
:H'	FFED	User's H' Register
:L'	FFEC	User's L' Register
:IX*	FFEA	User's IX Register
:IY*	FFE8	User's IY Register
:SP*	FFE6	User's SP Register

* = 2 byte mnemonics

TABLE 6-2

MITE-80 SERVICE CALL LABELS RECOGNIZED BY MITE-80 DEBUG

Service Label	Represents Service of
M8SN	Send a message to a task.
M8SNW	Send a message to a task and wait for a message to be available.
M8RSN	Resend a message to a task.
M8RSNW	Resend a message to a task and wait for a message to be available.
M8RCV	Receive a message if one is available for calling task.
M8RCVW	Receive a message, otherwise wait until one is available.
M8FWD	Forward a message to a task.
M8FWDW	Forward a message to a task and wait for a message to be available.
M8RET	Return a message to the sending task.
M8RETW	Return a message to the sending task and wait until a message is available.
M8CAN	Cancel a message sent to a task.
M8FIND	Find the receiver address of the specified task name.
M8WINT	Wait for an interrupt event to be posted.
M8PINT	Post an interrupt event which is completed.
M8ENAI	Enable CPU interrupts.
M8DISI	Disable CPU interrupts.

6.3.3 COMMAND TERMINATORS

The command terminator immediately follows the operand(s) and signals MITE-80 DEBUG that the command has been entered. Depending on the terminator, MITE-80 DEBUG will do one of the following:

Terminator	Action
1. <RETURN KEY>	Carriage return. MITE-80 DEBUG will perform the command entered.
2. ^	Carat or up arrow. This terminator is valid only for the M and P commands. When updating a memory location (M) or a port (P), it signals MITE-80 DEBUG to display the contents of the location or port just updated.
3. .	Period. MITE-80 DEBUG will abort the command, enter the command mode and be ready to accept another command.
4. /	Slash. This terminator is valid only for the M command. This causes the data entered to replace the old data, then return to the command mode. If no data was entered, it is treated as a carriage return.

NOTE--Anytime erroneous input is detected, a question mark (?) is printed and MITE-80 DEBUG returns to the command mode.

6.4 DETAILED COMMAND DESCRIPTIONS

This section describes each MITE-80 DEBUG command in detail. The command format is shown, followed by a description and examples. However, for those commands

which are identical to DDT-80, their sections are only highlighted and the user is referred to FLP-80DOS Operations Manual for additional information and examples. For the purpose of this section, the conventions used are:

1. aaaa,...,zzzz — denotes 4 hexadecimal digit operand value.
2. t — denotes the command terminator; carriage return, carat, period, or slash.
3. _____ — underline denotes the portion of the command entered by the user.
4. <CR> — Denotes the carriage return character (0DH).

6.4.1 B COMMAND, BREAKPOINT COMMAND

The breakpoint command causes the setting of a "trap" or breakpoint within the user's program. Upon encountering the breakpoint, the user's program will transfer control back to MITE-80 DEBUG where the registers, I/O ports, memory contents,service call histogram, TCB's amd MB's may be inspected. Breakpoints may be set only in RAM, not ROM.

6.4.1.1 FORMATS

:B aaaat — Set breakpoint at address aaaa.

:B Raaaat — Set breakpoint at relative address aaaaH.

:Bt — Clear any previous breakpoint.

6.4.1.2 DESCRIPTION

The user types the command identifier B followed by the address where it is desired to place a breakpoint. Upon entering carriage return MITE-80 DEBUG proceeds to:

1. Remove any pre-existing breakpoint by restoring user's code.

2. Extract and save 3 bytes of the user's program at the breakpoint address.

3. Place the 3 bytes at the breakpoint address with a breakpoint sequence. (This sequence consists of a 3 byte JP instruction to return to the breakpoint entry of MITE-80 DEBUG).

MITE-80 DEBUG then types a line feed and a colon ":" to return to the command mode. The user may now initiate execution of his program by using the execute command. When the address specified by the breakpoint command is encountered, control is transferred to MITE-80 DEBUG where the following actions are taken.

1. The three bytes of user code replaced by the trap instruction are restored.

2. All registers are recorded in RAM storage within MITE-80 DEBUG.

3. MITE-80 DEBUG types: the breakpoint address (Program Counter), and the values of the A and F registers for short format output or all internal CPU registers for long format output. If the offset command "O" had been executed, the relative PC is also printed.

4. MITE-80 DEBUG waits for the user to enter a "." to return to command mode or carriage return/line feed to begin "walking". (See W command)

A breakpoint can be cleared by executing its address or entering the B command with no operands. If the user misses a breakpoint while executing a program, the 3 bytes of breakpoint code must be replaced by executing the address of the inserted breakpoint (using E command). If the RESET Switch was used, then MITE-80 DEBUG must be reloaded along with the user program being debugged. The set break point command and execute command are closely related and are normally used together during the debug process for executing sections of a program and then

evaluating the registers for correct data.

There are certain characteristics of the MITE-80 DEBUG breakpoint facility which the user should be aware of during debugging. The only difference between MITE-80 DEBUG and DDT-80 breakpoint is if the MITE-80 DEBUG package is loaded into RAM instead of PROM, then care must be taken to not breakpoint within MITE-80 DEBUG itself. Refer to the FLP-80DOS Operations Manual for common characteristics.

6.4.2 C COMMAND, COPY MEMORY BLOCKS

The copy command permits any block of memory data to be moved to any area of memory. The move may be forward or backward and the new block may or may not overlap with the original memory block.

6.4.2.1 FORMAT

:C aaaa,bbbb,cccct	Copy memory locations aaaa through bbbb inclusive to the memory block starting at address cccc.

6.4.2.2 DESCRIPTION

The user enters the command identifier C followed by the starting address aaaa and ending address bbbb of the block to be moved, followed by the starting address cccc of the block receiving the data. The operands may be absolute or relative and are separated by commas or blanks. Upon terminating with a carriage return, MITE-80 DEBUG prints a line feed, performs the requested copy operation, and then prints a colon ":" to indicate that it is ready to accept another command. The data copied is not displayed.

6.4.2.3 EXAMPLES

:C 100,200,1200<CR>	Copy memory location 100H through 1300H.
:C 100,200,150<CR>	Copy memory locations 100H 150H through 250H (overlapping copy).

:O 100<CR>	Set relative offset to 100H.
:C R0,R100,R50<CR>	This would be the same as the previous copy example.

Entire programs or subroutines may be moved around in this way and still execute properly in their new locations if they contain self relocating code (i.e. use only relative jumps). Care should be taken to copy complete instructions on both ends of the block when copying programs, and any relative branch instructions contained within a block to be moved should not branch outside the block. If the second operand entered (bbbb) is smaller than the first (aaaa), a question mark prints out (?) and control returns to command mode.

If the MITE-80 DEBUG package is loaded into RAM instead of PROM, care must be taken not to copy memory locations into the MITE-80 DEBUG memory locations.

6.4.3 D COMMAND, DISPLAY HISTORY TABLE

The History Table display command allows display of the history table of MITE-80 services most currently executed by a user program. The MITE-80 service, task calling the service, type of action (exit or entry), and task's main register set contents are displayed by this command. Tasks with a priority in the range 0 to 15do not have their service calls recorded in the history table.

6.4.3.1 FORMATS

:D t	Display entire history table.
:D aaaat	Display last aaaa history entries.

6.4.3.2 DESCRIPTION

The user enters the command identifier D for history display followed by an

optional 1 to 2 hex digits for the number of last history entries to be displayed. Upon terminating with a carriage return, the specified number of last history entries will be displayed with a heading to label the table's contents. The maximum number of history entries is a user-specified system generation value. For console devices having 24 display lines, the user should specify 22 entries which result in a maximum hex value of 16H. For each MITE-80 service called during the program execution, an entry into the history table will occur. When the maximum number of table entries has been reached, the next entry will replace the oldest entry in the table. The table will always contain the last 32 service calls executed. Once the history table is displayed, a colon ':' will be displayed to indicate that it is ready to accept another command.

6.4.3.3 EXAMPLE

```
:D 5<CR>

ADDR TCB NAME PC   IX   AF   BC   DE   HL  SERVICE
A000 4241 AB aaaa aaaa aaaa aaaa aaaa aaaa M8SN   E
BF52 504C LP aaaa aaaa aaaa aaaa aaaa aaaa M8RCVW X
BF52 504C LP aaaa aaaa aaaa aaaa aaaa aaaa M8RCVW E
A000 4241 AB aaaa aaaa aaaa aaaa aaaa aaaa M8SN   X
A000 4241 AB aaaa aaaa aaaa aaaa aaaa aaaa M8RCVW E
```

The history table heading label and contents are:

ADDR — is the TCB address of the calling task in hexadecimal.

TCB — is the calling task name in hexadecimal.

NAME — is the calling task name in ASCII, a non printable ASCII character will be denoted by '.' (period).

PC — is the program counter contents and represents the location within the task calling the service.

IX-HL is the register contents of the task on entry to or exit from the service.

SERVICE is the MITE-80 service called. The next character is the direction between task and service, E indicates entry to service, X indicates exit from service.

6.4.4 E COMMAND, EXECUTE A USER'S PROGRAM The execute command is used to begin execution of all programs during debug sessions.

6.4.4.1 FORMATS

:E aaaat Transfer control to the program starting at address aaaa.

:E t Transfer control to the address specified by PC in the register map.

6.4.4.2 DESCRIPTION

To cause execution of a program, the user types the identifier E followed by the desired entry address of the program. Upon terminating with a carriage return MITE-80 DEBUG will load the user's internal registers from the saved register map then transfer control to the program entry point. (It is therefore possible to enter a program with preset values in the registers if desired.) Since the register map is used for saving internal registers when a breakpoint is encountered, the contents of the register map reflects the effect of the last instruction before the breakpoint was encountered. If no entry address is specified after the E command, MITE-80 DEBUG will transfer control to the address specified by PC in the user's register map.

6.4.4.3 EXAMPLES

:E 1200<CR> Execute the program starting at location 1200H.

To return control to MITE-80 DEBUG, the user's program must encounter a breakpoint. If the RESET button is pressed, then the reloading of MITE-80 DEBUG will be required in order to return to debugging.

$M80DDT<CR>	(User pressed RESET and enters Monitor).
:	Enter MITE-80 DEBUG.
:M :PC<CR>	Examine user's program counter (PC).
:PC 62FF 1220<CR>	Set user's PC to 1220H.
:E<CR>	Execute program starting at location 1220H.

The execute command may also be used together with the breakpoint command to execute portions of programs while debugging.

6.4.5 F COMMAND, FILL MEMORY COMMAND

The fill command permits a block of memory to be filled with a data constant.

6.4.5.1 FORMAT

:F aaaa,bbbb,cct	Fill memory locations aaaa through bbbb inclusive with cc.

6.4.5.2 DESCRIPTION

The user enters the command identifier F followed by the starting address aaaa and ending address bbbb, followed by the data cc. The operands are separated by commas or blanks. Upon terminating with a carriage return, MITE-80 DEBUG prints a line feed, performs the requested fill operation and then prints a colon ":" to indicate that it is ready to accept another command.

6.4.5.3 EXAMPLES

:F 100,FFF,5A<CR>	Insert a 5A in every memory location between and including 100H and FFFH.
:O 100<CR>	Set relative offset to 100H.
:F R0,REFF,5A<CR>	Fill same addresses as first fill example.

If the MITE-80 DEBUG program is loaded in RAM instead of PROM, then care must be taken not to fill memory in its area.

6.4.6 H COMMAND, HEXADECIMAL ARITHMETIC

The arithmetic capability of MITE-80 DEBUG allows hexadecimal addition and subtraction.

6.4.6.1 FORMAT

:H +aaaa=bbbb+...+yyyy=zzzzt	Perform hexadecimal arithmetic.

6.4.6.2 DESCRIPTION

The user enters the command identifier and then enters the arithmetic expression. Only + and - are legal operations. If the sign of the first operand is omitted, it is assumed +. The equal sign causes the 4 digit (least significant 4 digits) result to be displayed. When the terminator is entered MITE-80 DEBUG returns to accept another command. All operands may be absolute or relative (with 'R' prefix).

6.4.6.3 EXAMPLES

:H 5000-4FFF=0001<CR>	Subract 4FFFH FROM 5000H.

:H 5000+4FFF=9FFF<CR> Add 4FFFH to 5000H.

6.4.7 J COMMAND, SNAP SHOT COMMAND

The Snap Shot command allows the displaying of the MITE-80 queue structure of TCB's and MB's along with their respective contents.

6.4.7.1 FORMATS

:J t Snap shot the entire queue.

:J Lxxt Snap shot the task named "xx".

:J aaaat Snap shot the hex task name aaaa

6.4.7.2 DESCRIPTION

The user enters the command identifier J for snap shot followed by an optional operand which specifies the desired task to be snap shot. If the TCB name is omitted, the entire TCB queue will be snap shot. For a specified task name snap shot, the display will show the TCB's contents followed by all MBs and their contents which exist in the TCB's MB queue. The MBs which are present in the tasks queue are displayed in the order as they appear in the queue. The snap shot tabulation may be stopped at any time by entering a period "." on the console or temporarily suspended by entering a space " " on the console.

6.4.7.3 EXAMPLE

:J L1B<CR>

```
ADDR ST PR LINK MPTR SPTR NAME *** TCB ***
0104 05 05 56A7 0184 0178 4231 1B
```

```
 PC   IX   AF   BC   DE   HL   IY
07EF 0000 0000 0000 0000 0000 0000

ADDR ST PR LINK RPTR SPRT DATA *** MB ***
0184 00 03 2152 0004 0084 018E

ADDR ST PR LINK RPTR SPTR DATA *** MB ***
2152 00 09 FF2C 0104 157F 59AC
```

In this example a task whose name is '1B' is snap shot. The TCB contents are displayed along with the MB's present in the task's MB queue. The last MB will always have its link point to the MITE-80 null MB, which is not displayed. This is also true for the last TCB. The structure header labelling is as follows:

For TCBs:

ADDR	is the Address in memory where the TCB is located.
ST	is the Status field.
PR	is the Priority field.
LINK	is the Link field.
MPTR	is the Message Pointer field.
SPTR	is the Stack pointer field.
NAME	is the Name field, in hex followed by the ASCII equivalent characters. If non-printable characters exist, then a '.' (period) will be displayed.
PC-IY	the register contents of the task, the main register set.
A'F'-H'L'	is the alternate register set.

For MBs:

ADDR	is the Address in memory where the MB is located.
ST	is the Status field.
PR	is the Priority field.
LINK	is the Link field.

RPTR	is the Receiver Pointer field.
SPTR	is the Sender Pointer field.
DATA	is the Data field, only the first 2 bytes of the field is displayed.

6.4.8 K COMMAND, SERVICE BREAKPOINT COMMAND

The Service Breakpoint command allows for the setting of a "trap" or breakpoint on a specified task and MITE-80 Service. Upon encountering the service breakpoint, the program will transfer control back to MITE-80 DEBUG where the registers, I/O ports, memory contents, service call history, TCBs and MBs many be inspected. The service breakpoints may be set on tasks and/or services regardless of whether the programs reside in RAM or ROM. Multiple breakpoints are allowed.

6.4.8.1 FORMATS

:K aaaa,bbbbt	Breakpoint on MITE-80 service bbbb for task name aaaa.
:K ,bbbbt	Breakpoint on MITE-80 service bbbb for all tasks.
:K ,ALLt	Breakpoint on all MITE-80 services for alltasks.
:K t	Clear all previous service breakpoints.

6.4.8.2 DESCRIPTION

The user types the command identifier K followed by the task name and MITE-80 service where it is desired to place a breakpoint. The operands are separated by commas or blanks. Upon entering carriage return, MITE-80 DEBUG proceeds to mark the selected task and service as a breakpoint. Eash task name can have different and multiple services specified as breakpoints.

MITE-80 DEBUG then types a line feed and a colon ":" to return to the command

mode. The user may now initiate execution of his program by using the execute 'E' command. When the task name and the service specified by the breakpoint command is encountered, control is transferred to MITE-80 DEBUG where the following actions are taken:

1. All user registers are recorded in RAM storage within MITE-80 DEBUG.

2. MITE-80 DEBUG displays the same information as provided in the D command's history display: TCB address, task name, service, direction of service process, task address, and task register contents.

3. MITE-80 DEBUG displays a colon ':' indicating it is ready for the next command.

A service breakpoint can be cleared only by entering the K command without any operands. The breakpoint is not cleared by execution of the trap.

There are certain characteristics of the K command breakpoint facility which the user should be aware of during debugging:

1. The breakpoint all task feature is only valid on tasks whose priority is greater than 15 (0FH). This is done so that the higher priority tasks, such as system tasks like the timer handler, are not prevented from executing when required. If breakpointing on these higher priority tasks is desired, then the breakpoint must be explicitly set for the task name.

2. An error indication is given if the user attempts to breakpoint a task name and/or service not in the system.

3. Tasks which are created after a :K ,bbbb type command will not breakpoint.

6.4.8.3 EXAMPLES

For command inputs:

:K LAB,M8SNW<CR>	breakpoint on M8SNW for task name 'AB'.
:K 4F27,M8RCV<CR>	breakpoint on M8RCV for hex task name 4F27.
:K LCD,ALL<CR>	breakpoint on all services for task name 'CD'.
:K ,ALL<CR>	breakpoint on all services for all tasks.
:K <CR>	clear all breakpoints.

For a session:

```
:K LAA,M8SN<CR>

:E 0<CR>

*** BREAK POINT, TCB=004E 4141 AA
ADDR TCB NAME PC   IX   AF  BC   DE   HL  SERVICE
004E 4141 AA 016A 0000 0000 014E 00CE 0000 M8SN   X
```

The table heading label and contents are:

ADDR	is the TCB address of the calling task, in hexadecimal.
TCBNAME	is the calling task name in hexadecimal and ASCII, non-printable ASCII character will be denoted by a '.' (period).
PC	is the program counter contents and represents the location within the task calling the service.
IX-HL	is the register contents of the task on entry to or exit from the service.
SERVICE	is the MITE-80 service trapped on and the direction between taskand service, E indicates entry to service, X indicates exit from service.

6.4.9 L COMMAND, LOCATE 8 OR 16 BIT DATA PATTERN The locate command permits locating every occurrence of an 8 or 16 bit data pattern in a block of memory.

6.4.9.1 FORMATS

:L aaaa,bbbb,cct	Locate and print the address of every occurrence of cc from aaaa to and including bbbb.
:L aaaa,bbbb,cccct	Locate and print the address of every occurrence of the 16 bit pattern cccc from aaaa to and including bbbb.

6.4.9.2 DESCRIPTION

The user enters command identifier L followed by the starting address aaaa and ending address bbbb, followed by the data cc to be located. The operands are separated by commas or blanks. Upon terminating with a carriage return, MITE-80 DEBUG prints a line feed, then every address between aaaa and bbbb which contains cc is printed on the console. For 16 bit patterns, the address of the most significant byte is displayed. When the operation is complete, MITE-80 DEBUG prints a colon ":" to indicate that it is ready to accept another command.

6.4.9.3 EXAMPLE

:L 0,750,35<CR>	Locate every occurrence of 35H from address 0H thru 750H.
0052 35	Every Location containing 35H from 0H thru 750H is
00F3 35	printed.
0542 35	
0750 35	

6.4.10 M COMMAND, DISPLAY AND UPDATE MEMORY

This command allows display and/or modification of specified memory locations or the CPU registers.

6.4.10.1 FORMAT

:M aaaat

6.4.10.2 DESCRIPTION

The user enters the command identifier M. MITE-80 DEBUG collects the command and prints a space. The user then enters the operand aaaa followed by a terminator. MITE-80 DEBUG responds by printing the memory address on the next line. This is followed by the contents of the particular address in hexadecimal. If the content is to be changed, the new value is entered. The new value entered is an operand as previously described except that the appropriate number of hexadecimal digits (2 or 4) is selected.

6.4.10.3 EXAMPLE

If the memory location 5001H is to be changed to FFH:

:M 5001<CR>

5001 A3 FF<CR> — one memory location was changed, therefore the least significant 2 hex digits are used as the operand.

5002 A4 . — The period exits the M command and will allow for another command to be entered.

If the PC register is to be changed to 7F50H:

:M :PC<CR> — The PC register is a 4 hex digit (16 bit) register, therefore the least significant 4 hex digits are used as the operand.

:PC 7F50 . — Exit the M command.

For additional information and examples on the M Command, refer to the FLP-80DOS Operation Manual, MK78557.

6.4.11 M COMMAND, TABULATE MEMORY

This command allows the user to display, but not change, a block of memory. Up to 16 values are printed per line.

6.4.11.1 FORMAT

:M aaaa,bbbbt tabulate memory locations aaaa through bbbb.

6.4.11.2 DESCRIPTION

The user enters the command identifier M followed by the starting (aaaa) and ending (bbbb) addresses of the memory block separated by a comma or a blank. Upon terminating with a carriage return MITE-80 DEBUG prints a line feed, and then prints the contents of aaaaH to bbbbH inclusive with up to 16 values per line. MITE-80 DEBUG then returns to the command mode. The tabulation may be stopped at any time by entering a period "." on the console. It may also be suspended by depressing the space bar and resumed by depressing any other character key. When the 'R' prefix is used, the relative address is printed before absolute.

6.4.11.3 EXAMPLES

```
:M 4100,4127<CR>      display memory locations 4100H through 4127H inclusive.

4100 2B 90 12 20  00 B7 A5 21  10 94 04 20  CA B7 44 18  +.. ...!... ..D.
4110 81 11 34 21  07 94 17 45  12 55 A5 18  21 80 C5 55  ..4!...E.U..!..E
4120 90 0C A5 81  09 21 40 22  .....!@"

:                     MITE-80 DEBUG waiting for next command.

:O 4100<CR>           set offset to 4100.

:M R0,R27<CR>

'0000 4100 2B 90 12 20  00 B7 A5 21  10 94 04 20  CA B7 44 18 +.. ...!... ..D.
'0010 4110 81 11 34 21  07 94 17 45  12 55 A5 18  21 80 C5 55 ..4!...E.U..!..E
'0020 4120 90 0C A5 81  09 21 40 22  .....!@"
```

6.4.12 O COMMAND, SET OFFSET CONSTANT

The offset command is used to set a constant. This constant is added to any operand entered with an 'R' prefix.

6.4.12.1 FORMAT

:O aaaat set offset equal to aaaa.

6.4.12.2 DESCRIPTION

The user enters the command identifier O followed by the offset aaaa. Upon terminating with a carriage return, MITE-80 DEBUG prints a line feed, saves the 16 bit offset, and then prints a colon ":" to indicate that it is ready to accept another command. The offset can be cleared by entering the O command with no operands. After the offset has been set, both relative and absolute addresses are printed any time addresses are displayed and until the offset is cleared.

6.4.12.3 EXAMPLE

:O 1200<CR> Sets offset to 1200H.

6.4.13 P COMMAND, DISPLAY AND/OR MODIFY PORTS

This command allows the user to display and/or modify any of the possible 256 I/O ports. The reader should note that some ports are output only and cannot be read.

6.4.13.1 FORMAT

:P aat Display port aa.

6.4.13.2 DESCRIPTION

The user enters the command identifier P followed by the port address aa and a terminator. MITE-80 DEBUG responds by printing the port address and the value at the port. If the value at the port is to be changed, the user enters the new value. The new value entered is a 2 hexadecimal digit operand. When the user is

examining and/or modifying a port, the terminator signals the action MITE-80 DEBUG is to take.

6.4.13.3 EXAMPLE

:P D1<CR>	Program PIO Port 1AH (D1H) for BIT MODE.
D1 FF CF^	CFH sets 1A Control (D1H) to BIT MODE. Port D1 is output only.
D1 CF ^	Displays same port with new value.
D0 00 AA^	Output value AAH to Port D0H, then re-examine port.
D0 AA .	Exit the P command.

For additional information on terminator options and other examples, refer to the FLP-80DOS Operation Manual.

6.4.14 Q COMMAND, QUIT

The quit command is used to exit M80DDT and reboot the FLP-80DOS Monitor.

6.4.14.1 FORMAT

:Q

6.4.14.2 DESCRIPTION

The user enters Q to exit. The Monitor prints the reboot message.

6.4.14.3 EXAMPLE

:Q <CR> exit MITE-80 DEBUG.

MOSTEK FLP-80DOS VX.X (DATE)	Monitor reboot message
$	Enter Monitor (Monitor prompts with $).

6.4.15 R COMMAND, DISPLAY CPU REGISTERS

The display CPU registers command allows the user to examine the contents of all user registers to the console.

6.4.15.1 FORMATS

:R t	Print the contents of the CPU registers.
:R 1t	Print a heading to label the CPU registers on one line, on the next line print the contents of the CPU registers.
:R 1,xt	Print a heading to label the CPU registers and set the long/short flag as follows: x=0 SHORT, x=1 LONG. Long causes all registers to be printed after breakpoint and single step. Short causes only PC and AF to be printed. The LONG/SHORT FLAG remains set until changed by the 'R' command.

6.4.15.2 DESCRIPTION

The user enters the command identifier R. If the user wants a heading to be printed that labels the register contents, the operand of 1 needs to be entered. If no heading is desired, then no operand is entered. If the 'O' command has been used to set an offset, the relative PC is also printed.

6.4.15.3 EXAMPLES

:R <CR>	Display contents of CPU registers.
A000 0100 0104	CFB3 C09A FFEE EDF6 9C3E C3DC FE9B D6ED F1BE FFB4
R 1<CR>	Display contents of CPU registers with heading.

```
 PC   AF  I IF  BC   DE   HL  A'F' B'C' D'E' H'L'  IX   IY   SP
A000 0181 0104 CFB3 0010 C09A FFEE EDF6 C3DC FE9B D6EC F1BE FFB4
```

For further information and examples, refer to the FLP-80DOS Operation Manual.

6.4.16 V COMMAND, VERIFY MEMORY

The Verify command allows for the comparing of two memory blocks to detect any differences.

6.4.16.1 FORMAT

:V aaaa,bbbb,cccct	Compare memory location aaaa to bbbb with the memory starting at cccc.

6.4.16.2 DESCRIPTION

The user enters command identifier V followed by the starting address aaaa and ending address bbbb, followed by the starting address cccc of the second memory block. The operands are separated by commas or blanks. Upon terminating with a carriage return, every address from aaaa to bbbb is compared with the corresponding address starting at cccc. Any discrepancies are printed on the console (address data address data). When the comparison is complete, MITE-80 DEBUG prints a line feed and a colon ":" to indicate that it is ready to accept another command.

6.4.16.3 EXAMPLES

:V 0,FF,1000<CR>	Compare every location from 0H to FFH inclusive with
0000 AA 1000 BB	locations starting at 1000H. All differences are dis-
0038 55 1038 54	played on the console.

6.4.17 W COMMAND, WALK THROUGH A PROGRAM

The walk command, also known as software single-step, allows stepping through a program which is contained in RAM. The user's registers are saved and displayed after each step.

6.4.17.1 FORMAT

:W aaaa,nn,xxxt	Begin software single-step at address aaaa, for nn steps, xxx = HD requests register heading.

6.4.17.2 DESCRIPTION

The user enters the command identifier W followed by the starting address aaaa, the number of steps to take nn, and the options operand xxx. The operands are separated by commas or blanks. Upon terminating with a carriage return, MITE-80 DEBUG begins "walking" through the user's program (RAM resident). After each step the user's registers are displayed (See 'R' command). When nn steps have been taken, MITE-80 DEBUG waits for the user to enter a carriage return, line feed, space, or a period ".". A carriage return causes the next instruction to be executed and wait again for input. A line feed causes a register heading to be printed before the register print out. A space causes single stepping to continue for 256 instructions or until another space is entered to stop stepping. If nn is omitted, the default is 1. If aaaa is omitted, the last value of the user's program counter (:PC) is used to begin "walking". The stepping may always be stopped by entering any of the characters described above. When the address entered is relative, the PC' is also printed (relative PC).

Restrictions to W Command:

1. Only operates with programs in RAM.

2. Cannot CALL or RESTART to an address one or two locations prior to the CALL or RESTART.

3. Walking through self-modifying code will generally not work.

6.4.18 X COMMAND, DUPLICATE OUTPUT TO PRINTER DEVICE The duplicate command allows the duplicating of MITE-80 DEBUG output to console to also be outputted to the printer. This command is useful when outputting a snap shot (J Command) or history table (D Command) results.

6.4.18.1 FORMAT

:X t	Duplicate output to printer device along with console device.
:X t	Output to console device only.

6.4.18.2 DESCRIPTION

The user enters the command identifier X followed by the terminator of carriage return. M80DDT will then duplicate all console device output to the printer device as well if the console device was last designated for output only. If the console device and printer device were last designated for output, then output will be limited to the console device only. The output duplicate capability is useful for dumping large amounts of output from commands such as J (Snap Shot) or D (History Table), or M (Memory). The X command would be entered just prior to the command which will produce the large output, and then the X command is entered again after output to limit output back to the console device. On loading MITE-80 DEBUG, the output is always defaulted to the console device. If output is given to the printer device and the device is off-line, the console device will beep for 5 seconds. If the user does not bring the printer device on-line within the 5 seconds, the output will be automatically switched to the console device.

6.4.18.3 EXAMPLE

Input Device	Output Device (C = Console) (P = Printer)		
C	C	:B <CR>	Remove all breakpoints.
C	C	:X <CR>	Duplicate output to printer device.
C	C&P	:D 15<CR>	Display last 21 (decimal) history table entries.

			:
			:
			The history table is output to the printer and console
			:
			:
C	C&P	:X <CR>	Limited output to console device.
C	C	:K LTA,M8RET<CR>	Breakpoint on M8RET for task name 'TA'.

6.4.19 PROGRAMMING NOTES

The following is a list of items in MITE-80 DEBUG that could affect a program the user is writing and debugging:

The user stack pointer is set by MITE-80 DEBUG on power-up and reset (SP=FFAAH).

MITE-80 DEBUG uses 6 locations on the user's stack for temporary storage when transferring control to a user program (E command). The user's stack is left unaffected and the stack pointer points to the correct value. The user needs to be aware that 6 locations past the stack pointer are used.

When a breakpoint (B or K command) has been entered and not encountered while running the program, the user must press reset to regain control. MITE-80 DEBUG must be reloaded into RAM if the reset button is used. The breakpoint must be cleared by executing the address at which the breakpoint was inserted or by typing "B"<CR>.

To clear a program of all breakpoints, the user must remember that MITE-80 DEBUG has 2 different breakpoint commands, B and K, and that both must be individually cleared.

M80DDT can be re-entered after depressing the RESET button if there was no disk in drive 0 and the resident DDT-80 printed 'DSK ERR'. To re-enter M80DDT enter:

.E XB0B<CR>

Where X is 2, 6, A, or C for the 16K, 32K, 48K, and 64K version respectively of M80DDT. This allows the use of the M, J, and D commands to examine the state of MITE-80 immediately before the RESET button was depressed. NOTE: do NOT attempt to execute from this point without rebooting.

SECTION 7

CONFIGURATION REQUIREMENTS

7.1 OVERVIEW

This section outlines the requirements for configuring a MITE-80 system. The areas covered include memory requirements, initialization requirements, user specified configuration parameters, and MITE-80 macros.

7.2 MEMORY REQUIREMENTS

In order to compute the memory required for a MITE-80 system, the user must first itemize all the software modules which will comprise the system such as MITE-80 nucleus, user tasks, utilities, device drivers, TCBs, MBs, and stack areas.The determination of RAM or ROM program residency must also be made. Specific MITE-80 memory requirements follow and are outlined in ROM for program requirements and in RAM for volatile requirements. However, the ROM area can be a RAM area requirement if the user's application is an all RAM configuration.

7.2.1 MITE-80 NUCLEUS

The MITE-80 nucleus requires the following memory area:

ROM = 500 bytes
RAM = 14 bytes

The ROM area provides the memory necessary for all of the system services. The RAM area provides the memory necessary for the executive's current system state information needs. MITE-80 can reside anywhere within the 64K address range of the Z-80.

7.2.2 USER TASKS

User tasks are tasks which perform the user's application functions. These tasks

can reside in either RAM, or ROM, or a combination of both. Tasks are readily identifiable to MITE-80 by their TCB. Tasks require MBs in order to communicate to other tasks.

7.2.3 UTILITIES

Utilities are modules which are directly callable from a task. That is, an MB is not required to be used either to communicate with the task or to identify it to the system. Utilities perform a user specified function. Utilities can reside in ROM, or RAM, or a combination of both.

7.2.4 DEVICE DRIVERS

Device drivers are tasks which provide interface and operation to a peripheral device and the user's tasks. The drivers can reside in either RAM, or ROM, or a combination of both. MOSTEK supplied drivers are tasks and follow the TCB and MB rules. For user designed drivers, it is recommended that a task design approach be followed since priority levels can be easily changed as user application development conditions change.

7.2.5 TASK CONTROL BLOCK

Each task residing within the MITE-80 system must have a TCB. TCBs must reside in RAM. An initialization process or a task can construct or transfer a TCB from ROM to RAM. The size of a TCB can vary from task to task. However, the minimum TCB size is 10 bytes. Any additional TCB bytes are defined by and dependent on the specific task.

7.2.6 MESSAGE BLOCK

All inter-task communication is performed using an MB. MBs must reside in RAM. A task can construct or transfer an MB from ROM to RAM prior to calling a system service. The size of an MB can vary from MB to MB. However, the minimum MB size is 8 bytes. Any additional MB bytes are defined by and dependent on the specific task's application. The total amount of RAM required is the total number of bytes of all the MBs which will be active in a system at any given time. Care should be taken when calculating the total MB memory size. MBs which are known will be inactive while others are active until their process is complete can share the same memory area. The resultant memory size can then be less than the total of all MBs. The MB RAM size need only be as large as the maximum number of MBs

active at any given time.

7.2.7 STACK

Each task must have a stack area. The stack must reside in RAM. The size of each stack is dependent on the task's needs for nesting levels. However, the minimum stack size is 16 bytes. This area is used by MITE-80 for register saving on context switching and for register saving during interrupt occurrences. Care should be taken when calculating the total stack size. Tasks can not share stack areas. The total stack size is the sum of all the individual task's stack sizes.

7.3 TCB MACROS

To assist the user in configuring TCBs, 2 macros are provided: MTCB and ETCB. The MTCB macro will build a TCB from user specified parameters at the macro defined location. ETCB macro will build a TCB from user specified parameters and will also build code to transfer the TCB from ROM to RAM. This latter macro is recommended for use during initialization processes as it will queue the TCB into the MITE-80 system for the user. Both TCB macros require the use of MOSTEK's MACRO-80 Assembler.

7.3.1 MTCB MACRO

This macro builds a TCB from user specified parameters starting at the location where the macro is used.

7.3.1.1 FORMAT

```
MTCB NAME,RS,PR,STKS,EP,[IX],[AF],[BC],[DE],[HL],[IY],[AF'],[BC'],[DE'],[HL']
```

The parameter fields are:

NAME	the name of the task, in 2 ASCII characters beginning with a non-numeric character or a valid decimal or hexadecimal constant.
RS	the register set used in the task, M for main register set only, or A for full (all) register set. If this parameter is omitted, M is assumed.
PR	the priority of the task. If this parameter is omitted, the priority of the creating task is assumed.
STKS	the stack size of the task, in hexadecimal or decimal.
EP	the task entry point address.
IX-HL'	initial values for registers IX through HL' (optional).

If any of the parameters are incorrectly specified, an error message will be printed. The TCB will still be built with any valid parameters. For those parameters in error their field entries will have zeroes.

7.3.1.2 EXAMPLE A

For a task name of TA using the main register set, a priority of 25, a stack size of 100 bytes, and which is to begin execution at the label "ENTRY", the macro would be coded as:

```
        MTCB    TA,M,25,100,ENTRY
```

The resultant assembly code would be:

Assembly Code			User Parameter	TCB Field
	DEFS	100-16	100,M	Stack size minus main register set area.
T0001	DEFW	0	--	Initial main register set up
	DEFW	0	--	
	DEFW	0	--	
	DEFW	0	--	
	DEFW	0	--	
	DEFW	ENTRY	ENTRY	Task entry point
TCBTA	DEFB	005H	M	Status

	DEFB 033H	25	Priority (25 * 2 + 1)
	DEFW 00000H	--	Link
	DEFW 00000H	--	Message Pointer
	DEFW T0001	--	Stack Pointer
	DEFM 'TA'	TA	Task Name

7.3.1.3 EXAMPLE B

For a task name of 4F7CH using the entire register set, a priority of 65, a stack size of 40H bytes and which begins at the label "BEGIN", the macro would be coded as:

```
MTCB    4F7CH,A,65,40H,BEGIN
```

The resultant assembly code would be:

Assembly Code		User Parameter	TCB Field
	DEFS 040H-22	40H	Stack size minus full set area reserved
T0001	DEFW 0		
	:		
	:		
	DEFW BEGIN	BEGIN	Task Entry Point
TCB4F7CH	DEFB 001H	A	Status
	DEFB 083H	65	Priority (65 * 2 + 1)
	DEFW 00000H	--	Link
	DEFW 00000H	--	Message Pointer
	DEFW T0001	--	Stack Pointer
	DEFW 4F7CH	4F7CH	Task Name

7.3.1.4 PROGRAMMING NOTES

The MTCB macro should not be used to define a TCB which will initially reside in ROM.

If the task requires no pre-defined values in its registers prior to the task execution, then these need not be concerned with the macro call. However, the Program Counter (parameter EP, entry point) MUST be set-up in the macro call.

7.3.2 ETCB MACRO

This macro builds a TCB from user specified parameters and also creates code which when executed will transfer the TCB from ROM to RAM. The code generated can be part of a user's initialization program. An initial Tasks stack contents are also built from user specified parameters. To install a task defined by an ETCB macro, the user must execute the macro (i.e. the ETCB macro generates the instructions necessary to create and install the task).

7.3.2.1 FORMAT

ETCB,ADDR,NAME,RS,PR,STKS,EP,[IX],[AF],[BC],[DE],[HL],[IY],[AF'],[BC'],[DE'],[HL']

The parameter fields are:

ADDR	address of RAM to load TCB ROM image.
NAME	the name of the task, in 2 ASCII characters beginning with a non-numeric character or a valid decimal or hexadecimal constant.
RS	the register set used in the task, M for main register set only, or A for full (all) register set. If this parameter is omitted, M is assumed.
PR	the priority of the task. If this parameter is omitted, the priority of the creating task is assumed.
STKS	the stack size of the task, in hexadecimal or decimal.
EP	the task entry point address.
IX-HL'	initial values for registers IX through HL' (optional).

If the task does not require initial values in the registers, then the optional parameters need not be specified. If only certain registers require initial values, only their values need be specified. However, the proper parameter positions must be adhered to by using commas (,) to separate them, including any non-specified ones.

If any of the parameters are incorrectly specifed an error message will be printed. The TCB will still be built with any valid parameters. For those parameters in error their field entries will have zeros. Also, entries which are not specified will have their entry set to a default value of zero. Trailing parameters, however, will NOT have their entries set to zero if they are omitted.

7.3.2.2 EXAMPLE A

For a task to be loaded at address 1000H and with a name of LT using the main register set, a priority of 30, a stack length of 32, an entry point of LTEP, with an initial register values of IX = IOVECT, BC = 4752H, and IY = IBUFF, the macro would be coded as:

```
ETCB 1000H,LT,M,30,32,LTEP,IOVECT,,4752H,,,IBUFF
```

The resultant code generated by the macro will transfer the specified TCB and stack contents from ROM to RAM starting at the RAM address specified by address parameter. Once the TCB and stack are transfered into RAM, the TCB will be installed into the MITE-80 TCB queue.

7.3.2.3 EXAMPLE B

Multiple ETCB macros can be defined in succession and will occupy the RAM area immediately following the last RAM location used from the previous macro. A task's TCB and stack area will occupy a RAM area different from the last defined address contained in ADDR, then ADDR can be re-initialized with the new desired RAM transfer address. An example follows:

```
ETCB  4000H, . . . . .        Task A
ETCB  , . . . . . .           Task B
ETCB  6000H,. . . . . .       Task C
```

In the above code sequence, Task A's TCB and Stack contents will be transfered to RAM starting at RAM location 4000H. Task B's TCB and stack contents will be transferd to RAM starting at the next available RAM address after the end of Task A's TCB. Task C's TCB and stack pointer will be transfered to RAM starting at RAM location 6000H.

7.4 TASK INSTALLATION

The following procedures will serve as a guideline for those users electing to install a task into the MITE-80 system with their own designed code as opposed to the ETCB Macro generated code. A TCB must first be created either by direct coding or by the use of the MTCB Macro. The TCB must then be transferred into RAM.The TCB is then installed into the MITE-80 system by sending it to the MITE-80 TCB queue as follows:

```
LD      DE,M8TCBQ              ;MITE-80 TCB QUEUE ADDRESS
LD      BC,<TCB Address>       ;"TO BE INSTALLED" TASK'S TCB ADDR
CALL    M8SN
```

Note that in this code sequence the MITE-80 Send Message Service is used to install the task. The MB Address is the to be installed task's TCB address, and the Receiver Pointer is the MITE-80 TCB Queue adddress (M8TCBQ).

Care must be exercised when installing a task. The task's stack must already be installed in RAM (refer to STACK INSTALLATION paragraph which follows).

7.5 STACK INSTALLATION

The following procedures will serve as a guideline for those users electing to install a task's stack with their own designed code as opposed to the ETCB Macro generated code. The initial stack contents must be loaded into the proper stack

locations in order to assure proper register set-up at task run time. This order is important since MITE-80 pops the registers off the task's stack prior to providing CPU control to the task. To assist the user, the following table provides a referenced stack label offset to the corresponding register position:

Stack Pointer Offset (Decimal)		
Main Set Only	Full Set	Register Position
-	+ 0	L'
-	+ 1	H'
-	+ 2	E'
-	+ 3	D'
-	+ 4	C'
-	+ 5	B'
-	+ 6	F'
-	+ 7	A'
+ 0	+ 8	IY least significant byte
+ 1	+ 9	IY most significant byte
+ 2	+10	L
+ 3	+11	H
+ 4	+12	E
+ 5	+13	D
+ 6	+14	C
+ 7	+15	B
+ 8	+16	F
+ 9	+17	A
+10	+18	IX least significant byte
+11	+19	IX most significant byte
+12	+20	PC least significant byte
+13	+21	PC most significan byte

Note that since the Z80 uses a push down stack concept, the initial stack pointer (that is the initial value loaded into the Task TCB's SPTR field) will be pointing to the IY registers least significant byte if only Main Register set is used, or the L' register if the full register set is used.

If a task does not require any initial register values to be set-up, then the user need not be concerned with their set-up procedures. However, the PC value is the minimum stack installation the user MUST perform. The PC value will contain the task's entry point. The entry point is the first location where CPU execution will begin when MITE-80 gives CPU control to the task.

7.5.1 EXAMPLE

To initialize a stack with a PC of entry point TASKEP, main registers, and registers BC = 407FH, the code sequence for stack pointer TASKSP installaton would be:

```
LD      HL,TASKEP              ;TASK ENTRY POINT ADDRESS
LD      (TASKSP+12),HL         ;INITIALIZE PC STACK VALUE
LD      HL,0407FH              ;INITIAL BC VALUE
LD      (TASKSP+6),HL          ;INITIALIZATION BC STACK VALUE
```

7.5.2 PROGRAMMING NOTES

When installing a task, the installing task should be of a higher priority than the to be installed task, if the installing task wants immediate CPU control returned.

A task's stack MUST be initialized before the task's TCB is installed.

Stack initialization at minimum MUST initialize the stack's initial PC value (Entry Point).

SECTION 8

MEMORY POOL MANAGER

8.1 INTRODUCTION

This section describes how to configure and use the MITE-80 Memory Pool Manager software module. This Memory Manager provides a way to allocate and deallocate different size memory blocks from a various number of memory pools. Application uses of memory pools are buffer areas, temporary storage, and Message Blocks or TCBs.

8.1.1 FEATURES

A. Provides central control over memory pools.

B. Allows up to 252 pools.

C. Each pool can have a different size memory block.

D. Up to 256 bytes allowed per memory block.

E. Unlimited number of memory blocks allowed for each pool.

8.1.2 SOFTWARE CONFIGURATION

The Memory Manager operates as a user callable routine. It is re-entrant and can be used by as many tasks as the user has configured into the MITE-80 system. The routines are provided in relocatable object format, and as such can reside at any user desired memory location. This module consists of the following programs:

A. M8MMAL - Allocate a block of memory from a given pool to the caller.

B. M8MMDE - Deallocate a block of memory from the caller and return it to the appropriate pool.

C. MPOOL - A system generation MACRO to assist the user in defining and developing the desired number of pool's and the desired block size within each pool. The macro requires the use of MOSTEK's MACRO-80 Assembler.

The amount of memory required by the Memory Manager is dependent on the number and types of pools configured. The following equations can be used to calculate the RAM and program memory sizes required of a specific user configuration:

For Program Memory:

Total Bytes = 121 + (3 x Number of Pools)

For RAM Area:

```
Total Bytes = (6 x Number of Pools)
            + (Pool #1 Block Size x Pool #1 Number of Blocks)
            + (Pool #2 Block Size x Pool #2 Number of Blocks)
            + (Pool #3 Block Size x Pool #3 Number of Blocks)
            + . . . . . .
            + (Pool #N Block Size x Pool #N Number of Blocks)
```

8.1.3 POOL CONFIGURATION

A memory pool is a contiguous area of memory. A Memory block is a contiguous area of memory within the pool. Multiple blocks of the same memory size comprise a memory pool. A memory block size can range from 4 to 256 bytes, and the number of blocks within a pool is limited only by user allotted memory. Multiple pools are permitted and each pool should have a different block size. However, all blocks within a pool must have the same block size. The number of pools allowed is limited to 252 pools.

8.2 CALLING CONVENTIONS

The format, description, and examples of how to allocate and deallocate a memory block from a user program are outlined in this section. Two user calls are provided for memory pool access;

M8MMAL - allocate a memory block from a pool.

M8MMDE - deallocate a memory block back to a pool.

8.3 M8MMAL - ALLOCATE MEMORY

8.3.1 FORMAT

The calling sequence is:

```
LD   BC,<Number of bytes desired>
CALL M8MMAL
```

The Service will return:

HL = address of block or 0000H
Z flag = reset if available, set if unavailable

8.3.2 DESCRIPTION

The M8MMAL Service will search for an available memory block from a pool whose block size is equal to or greater than the caller's requesting size, and return the address of the allocated memory block if one is available. The Service will search all pools starting with the pool being closest in memory block size to the request until an available block is found or until no more pools exist. Registers BC must contain the number of bytes, in binary, desired to be allocated. On return, registers HL will contain the starting address of the memory block allocated to user, and the Zero Flag will be reset. Otherwise if no memory block is available, then registers HL will be zeroed and the Zero Flag will be set. The Memory Manager uses 10 bytes of the calling task's stack for register saving. All registers are preserved with the exception of HL and the Zero Flag.

8.3.3 EXAMPLE A

To request a memory area of 32 bytes the code sequence would be:

```
LD    BC,00020H     ;Set-up memory request for 32 bytes
CALL  M8MMAL        ;Request memory block allocation
JR    Z,NBA         ;Jump if no block available
```

Note in this example that on return the Zero Flag was checked for block availability.

8.3.4 EXAMPLE B

To request a memory area of 253 bytes the code sequence would be:

```
LD    BC,000FDH
CALL  M8MMAL
JR    Z,NBA
```

8.3.5 PROGRAMMING NOTES

The stack of the calling task MUST have 10 bytes available for use by the M8MMAL Service.

It is recommended that the calling task always check for memory block availability on return from Service.

A Task using the allocated memory block MUST limit the use of the memory block to the area within the requested size.

The address range of Memory Block allocated is the starting address (HL register on return) plus the request size - 1.

8.4 M8MMDE - DEALLOCATE MEMORY

8.4.1 FORMAT

The calling sequence is:

```
LD    HL,<allocated memory block address>
LD    BC,<original number of bytes requested>
CALL  M8MMDE
```

The Service will return:

Z flag = reset (0) if deallocation successful, set if unsuccessful.

8.4.2 DESCRIPTION

The M8MMDE Service will return a memory block back to the proper memory pool. The deallocated memory block will then be available for use by another task. Registers HL must contain the memory block address allocated by the M8MMAL Service and the BC registers must contain the original number of bytes requested when the M8MMAL Service was called. On return the Zero Flag will be reset if the deallocation was successful. Otherwise the Zero Flag will be set to indicate an unsuccessful deallocation (this can result from a bad parameter being passed). The Memory Manager uses 10 bytes of the calling task's stack for register saving. All registers are preserved and restored with the exception of the Zero Flag.

8.4.3 EXAMPLE A

To Return an allocated memory area of 32 bytes at address 4CA0H, saved at temporary location MBSAVE, the code sequence would be:

```
LD    HL,(MBSAVE)  ;Set-up allocated memory block address.
LD    BC,00020H    ;Set-up original byte count requested.
CALL  M8MMDE       ;Return memory block.
JR    Z,DEFAIL     ;Check if deallocation failed.
```

Note in this example that on return the Zero Flag was checked for deallocation failure.

8.4.4 PROGRAMMING NOTES

The stack of the calling task MUST have 10 bytes available for use by the M8MMDE Service.

It is recommended that the calling task always checks for deallocation failure on return from Service.

A deallocation failure can occur from not providing the original allocated memory block address and the original number of bytes requested from the M8MMAL Service.

8.5 M8POOL - MACRO

A macro is provided to assist the user in defining and developing memory pools. The macro will produce the specified number of pools and blocks along with its internal parameter requirements. The macro requires the use of the MOSTEK MACRO-80 Assembler.

8.5.1 FORMAT

```
M8POOL  Size1,Block1,Size2,Block2,Size3,Block3,...,SizeN,BlockN
```

Where:
- Size1 - size of blocks for first pool
- Block1 - number of blocks in first pool
- Size2 - size of blocks for second pool
- Block2 - number of blocks in second pool
- Size3 - size of blocks for third pool
- Block3 - number of blocks in third pool
- .
- .
- .
- SizeN - size of blocks for Nth pool
- BlockN - number of blocks in Nth pool

8.5.2 DESCRIPTION

The M8POOL macro will produce the specified number of pools with their respective

block sizes all in one contiguous memory pool area. If multiple pools are specified, then all of the pools will be located in one contiguous memory area. The size parameters must be non-repeating, in ascending order, and in the range of 4 to 256. The block parameters must be a value greater than 0. If not, then the size is ignored and that pool is not generated. Macro errors will result if illegal conditions are specified.

8.5.3 EXAMPLE A

Correct use of macro is:

```
M8POOL    16,14,64,2,256,10
```

This will create 3 pools;
the first with 14 blocks of 16 bytes each,
the second with 2 blocks of 64 bytes each,
the third with 10 blocks of 256 bytes each.

Using the equation provided in the SOFTWARE CONFIGURATION Section, the total RAM area required for these pools would be:

```
    18 = 6 x 3 for (6 x Number of Pools)
+  224 = 16 x 14 for (Pool #1 Block Size x Pool #1 Number of Blocks)
+  128 = 64 x 2 for (Pool #2 Block Size x Pool #2 Number of Blocks)
+ 2,560 = 256 X 10 For (Pool #3 Block Size x Pool #3 Number of Blocks)
  2,930 = Total RAM bytes required
```

8.5.4 EXAMPLES B

Incorrect uses of macro is:

```
M8POOL    16,4,64,2,32,12
```

The Size parameters are not in ascending order.

```
M8POOL    16,8,64
```

An invalid number of parameters is specified.

8.6 POOL CONSTRUCTION

To construct a pool or pools in a system, the user should follow this example procedure:

```
        .
        .
INCLUDE MPOOL
        .
        .
        .
ORG     04000H
M8POOL  16,12,32,4,64,8,128,8
        .
        .
```

The above sequence of code will create 4 pools starting at location 4000H.

SECTION 9

TIMER HANDLER

9.1 INTRODUCTION

This section describes how to configure and use the MITE-80 Timer-Handler Software Module. The Timer Handler provides a means to time events using the MK3882 Counter Timer Circuit (CTC) Chip. Application uses of the timer are event timeouts and watchdog alert timeouts.

9.1.1 FEATURES

The Timer Handler features are:

A. Multiple timer handling capability.
B. User specified timer tick rate.
C. Control of 1 or multiple MK3882 CTC Chips.
D. Time range of 12.8 milliseconds to 13.98 minutes
E. Post interrupt event option on time-out occurrence.
F. Cyclic timer option on time-out occurrence.

9.1.2 SOFTWARE CONFIGURATION

The Timer Handler operates as a task executing under MITE-80. User tasks communicate with the Timer Task by use of a Message Block (MB). The Handler is provided in relocatable object format, and as such can reside at any user desired memory location. This module is named M80TH. The module is re-entrant and can control multiple CTC channels.

The macro file M80SYS.MAC contains two macros to assist the user in defining the Timer Handler's Task Control Block:

A. MTHTCB - a system generation macro to assist the user in defining and developing the Timer Handler's TCB in RAM.

B. ETHTCB - a system generation macro to assist the user in defining and developing the Timer Handler's TCB as a ROM based skeleton which is copied RAM at execution time.

The Timer Handler requires an MK3882 CTC chip for timing purposes. The Timer Handler can accommodate any number of CTC chips with from 1 to 4 timer channels active within each chip. Each timer channel requires its own Timer Handler TCB which can specify a different timer count rate. The tick rate is fixed at 12.8 milliseconds. This rate combined with 2 other user specified parameters (timer rate and delay count) can produce a maximum delay time of 13.98 minutes. Applications requiring time delays greater than the maximum delay will have to use an additional counter which is maintained and kept by the Task requiring the capability.

9.1.3 MEMORY

The memory required by the Timer Handler is 195 bytes of program area and a minimum 66 bytes of RAM area. The program area is for the timer initialization and Timer Handler. The RAM area is for the timer handler's TCB and stack. Each additional Timer Handler TCB requires 18 bytes of RAM plus 20 bytes for a stack.

9.2 CALLING CONVENTIONS

The format, description, and examples on how to set-up a timer MB and use of the timer handler are outlined in this section.

9.2.1 TIMER MESSAGE BLOCK

Tasks requiring use of the Timer Handler communicate with it by an MB. The Timer Handler MB format is:

Field #	# of Bytes	Offset	Name	Field	Data Type	Source
1	1	0	STAT	Status	Binary	User
2	1	1	PRIO	Priority	Bit Encoded	User
3	2	2	LINK	Next MB Addr	Binary	MITE-80
4	2	4	RPTR	Timer Handler Pointer	Binary	User
5	2	6	SPTR	Sender Pointer	Binary	User
6	1	7	RQST	Request Code	Bit Encoded	User
7	1	8	PERD	Period of time	Binary	User

Field numbers 1 through 5 represent the standard MB format and fields 6 and 7 are the timer handler's data field. Each field is further defined as follows:

9.2.1.1 STAT

The Status byte (STAT) is used by the Timer Handler for keeping record of the remaining timer delay counts for this specific MB request. The user provides this value, range of 01H (for 1 tick) to FFH (for 255 ticks) and 00H (for 256 ticks), and the timer handler will decrement the count once for each timer rate count. The total delay time is:

Delay Time = Number of Delay Counts (STAT Field) x Timer rate count.

Refer to the MTHTCB MACRO paragraph of this section for information on the timer rate count.

9.2.1.2 PRIO

The Priority byte (PRIO) contains user provided information on where in the timer

handler's message queue this MB is to be placed. Refer to Section 3, Data Structures, for additional information on the Priority byte.

9.2.1.3 LINK

The Link word (LINK) is used by MITE-80 to link the MB into the task's message block queue.

9.2.1.4 RPTR

The Receiver Point word (RPTR) is the task name the user has assigned the Timer Handler TCB. For example; TH, T0, T1, etc. This field contains the TCB address of the Timer Handler.

9.2.1.5 SPTR

The Sender Pointer word (SPTR) is the task name of the sending task's TCB. This field contains the TCB address of the sending task.

9.2.1.6 RQST

The Request Code byte (RQST) is the user-specified timer option which is to be used for this timer service request. The Request Code determines what action the Timer Handler is to take with the MB when the STAT byte (delay count) becomes zeroed, and what type of return service is to be used. There are 3 Request Code options and are:

Request Code	Time-Out Action
00H	M8CAN and M8RET
01H	M8CAN and M8PINT
02H	M8PINT

Request Code 00H. When delay timeout occurs, this request code will cause the Timer Handler to cancel the MB time request and return the MB to the sending task.

Request Code 01H. When delay time-out occurs, this request code will cause the Timer Handler to cancel the MB time request and post an interrupt event complete status for the sending task.

Request Code 02H. When delay time-out occurs, this request code will cause the Timer Handler to RESET the MB's FIFO bit and to post an interrupt event complete status for the sending task and will restart the delay time-out for the period specified in the PERD field (Period). Tasks using this request code MUST perform a M8WINT before delay time-out occurs. This sequence will be repeated every time a delay time-out occurs until the sending task cancels the MB from the Timer Handler's MB queue.

9.2.1.7 PERD

The Period byte (PERD) is required only if Request Code (RQST) of 02H is used. This field is used by the Timer Handler to reload the STAT field (delay counts) when delay time-out occurs. This user specified value is the new delay time-out value to be used whenever the delay time-out occurs. The Period value range is 01H, 02H, . . . , FFH, 00H.

The delay time is:

Delay Time = Period (PERD) x Timer Rate Count

9.3 USING THE TIMER TASK

To use the timer task, a using task must first construct an MB to be sent to the Timer Task. This MB can either be contructed in RAM or already exist in ROM and be transferred to RAM. The MB is sent to the Timer Task just as any MB is sent between tasks.

9.3.1 FORMAT

The calling sequence is:

```
LD      DE,<Timer TCB address>
LD      BC,<MB address>
CALL    M8SN (or M8SNW)
```

9.3.2 DESCRIPTION

The calling sequence is identical to that used for sending any MB to a task. The M8SN or M8SNW Service can be used. CPU control will be returned to the calling task if M8SN Service is used and it is still the highest priority ready task. Care should be taken when specifying the time-out delay value since a tick down count could occur moments after the MB is sent to the Timer Handler Task. An additional delay count of 1 should be added to the desired delay time-out value to guarantee a minimum delay (STAT or PERD fields).

9.3.3 EXAMPLE

To use a timer service of return MB when a minimum delay time-out of 1 second occurs (assume timer count rate of 51.2 milliseconds) the MB would be:

```
TMB  DEFB   21        ; STAT - DELAY COUNT OF ABOUT 1 SECOND
     DEFB   03CH      ; PRIO - OF 30
     DEFW   0000H     ; LINK
     DEFW   TCBTH     ; RPTR - TIMER HANDLER TASK'S TCB
     DEFW   EWTCB     ; SPTR - ERROR WATCHDOG TASK'S TCB
     DEFB   0         ; RQST - RETURN MESSAGE UPON TIME-OUT
```

The total delay time is:

Delay time = 21 x 51.2 = 1075.2 milliseconds.

The delay count of 21 was determined by:

1 second/51.2 milliseconds = 19.53 counts or 20 counts.

An additional 1 count (51.2 milliseconds) is added to guarantee a minimum timer delay of 1.024 seconds and a maximum delay of 1.0752 seconds. The result is a delay count of 21.

The program code sequence would be:

```
LD   DE,TCBTH    ; TIMER HANDLER TASK
LD   BC,TMB      ; TIMER MESSAGE BLOCK
```

```
        CALL M8SN       ; SEND MB TO TIMER
```

9.4 MTHTCB MACRO

A macro is provided to assist the user in defining and developing a RAM based Timer Handler TCB. The macro will produce the specified timer parameters in the proper TCB data structure. The macro requires the use of the MOSTEK MACRO-80 Assembler.

9.4.1 FORMAT

```
        MTHTCB    NAME,TICK,VECTOR,PORT,PRIO,TMCNT
```

Where:

NAME - the name of the Timer Handler task in 2-3 ASCII characters or a hexadecimal or decimal number.

TICK - the timer tick rate in binary from 01 (for 1 tick) to FFH (for 255 ticks) and 00H (for 256 ticks) with each tick equaling 12.8 milliseconds. Default value is 4 for 51.2 milliseconds.

VECTOR - the Timer Handler interrupt vector value (must be even). Default label is (NM)VECT.*

PORT - the port number for this Timer Handler in 2 hexadecimal digits or a global label. Default label is (NM)PORT.

PRIO - the priority level at which the Timer Handler task will execute. Default value is (NM)PRIO.

TMCNT - the time constant used to load the CTC. Default is (NM)MD25 for the MD series 2.5 MHz system (See CTC Technical Manual).

* NOTE--(NM) indicates the first two characters of the NAME parameter.

9.4.2 DESCRIPTION

The MTHTCB will produce a Timer Handler TCB from the user-specified parameters. Each one of the parameters MUST be separated by a comma. The default value will be used for any parameter which the user has NOT specified except for the NAME field which will generate an error. A macro error message will result if any illegal parameter value is specified. Care must be exercised when defining multiple timer TCBs for the same CTC Chip. The CTC Chip rules for interrupt vector addressing (offset from first channel) MUST be adhered to. Also, the user MUST adhere to CTC Chip port addressing (offset from first channel). Refer to the MK3882 Counter Timer Circuit Techinical Manual for further information.

9.4.3 EXAMPLE A

To configure a timer TCB of the name TH with a tick rate of 512 milliseconds, vector of FOH, I/O port of DCH, and at a priority of 8 for MD series 4.0 MHz, the macro call would be:

```
MTHTCB  TH,40,0F0H,0DCH,8,THMD40
```

The resultant TCB code would be:

```
            DEFS    6
T0001       DEFS    10      ; STACK AREA
            DEFW    TCBTH   ; TCB ADDRESS
            DEFW    M80TH   ; ENTRY POINT
            DEFB    0F0H    ; VECTOR FOR 'TH' ISR
            DEFB    0DCH    ; PORT FOR 'TH' CHANNEL
            CALL    TH?ISR
TCBTH       DEFB    005H    ; STAT
            DEFB    011H    ; PRIO OF 8
            DEFW    0000H   ; LINK
            DEFW    0000H   ; MSG BLK POINTER
            DEFW    T0001   ; STACK POINTER
            DEFW    'TH'    ; TIMER HANDLER NAME
            DEFB    40      ; TICK COUNT OF 512 milliseconds
            DEFB    40      ; CURRENT TICK COUNT
            DEFB    200     ; CTC TIME CONSTANT
```

The characters specified in the NAME parameter are used to construct a label for the TCB with the first 3 characters always being "TCB".

9.4.4 EXAMPLE B

To configure four Timer TCBs all working from the same CTC Chip whose vector address is AOH and the port address is BCH with unique timer features as follows:

	Timer 0	Timer 1	Timer 2	Timer 3
Name	T0	T1	T2	T3
Priority	5	10	30	15
Tick Rate	.192 seconds	.512 seconds	3.008 seconds	1.472 seconds

The macro calls would be:

```
MTHTCB  TH0,15,0A0H,0BCH,5
MTHTCB  TH1,40,0A2H,0BDH,10
MTHTCB  TH2,235,0A4H,0BEH,30
MTHTCB  TH3,115,0A6H,0BFH,15
```

Note that in this example the Timers TH0, TH1, TH2, and TH3 all adhere to the CTC Chip's vector and I/O port addressing offset rules.

9.5 ETHTCB MACRO

A macro is provided to assist the user in defining and developing a RAM based Timer Handler TCB. The macro will produce the specified timer parameters in the proper TCB data structure. The macro requires the use of the MOSTEK MACRO-80 Assembler.

9.5.1 FORMAT

```
ETHTCB    ADDR,NAME,TICK,VECTOR,PORT,PRIO,TMCNT
```

Where:

ADDR - the address at which the timer's TCB is to be installed. (If this parameter is missing the TCB will follow the previous 'E' type TCB in memory).

NAME - the name of the Timer Handler task in 2-3 ASCII characters or a hexadecimal or decimal number.

TICK - the timer tick rate in binary from 01 (for 1 tick) to FFH (for 255 ticks) and 00H (for 256 ticks) with each tick equaling 12.8 milliseconds. Default value is 4 for 51.2 milliseconds.

VECTOR - the Timer Handler interrupt vector value (must be even). Default label is (NM)VECT.*

PORT - the port number for this Timer Handler in 2 hexadecimal digits or a global label. Default label is (NM)PORT.

PRIO - the priority level at which the Timer Handler task will execute. Default value is (NM)PRIO.

TMCNT - the time constant used to load the CTC. Default is (NM)MD25 for the MD series 2.5 MHz system (See CTC Technical Manual).

* NOTE--(NM) indicates the first two characters of the NAME parameter.

9.5.2 DESCRIPTION

The ETHTCB will produce a Timer Handler TCB from the user specified parameters. Each of the parameters MUST be separated by a comma. The default value will be used for any parameter which the user has NOT specified except for the NAME field which will generate an error. A macro error message will result if any illegal parameter value is specified. Care must be exercised when defining multiple timer TCBs for the same CTC Chip. The CTC Chip rules for interrupt vector addressing

(offset from first channel) MUST be adhered to. Also, the user MUST adhere to the CTC Chip port addressing (offset from first channel). Refer to the MK3882 Counter Timer Circuit Technical Manual for further information.

9.6 INSTALLATION

Once the Timer TCBs are contructed, they must be installed into the MITE-80 system before any timer sevice request can be made. Installing the Timer Task can be performed within a user's initialization task. The Timer TCBs can be installed along with other TCBs as outlined in the M8TCB macro Section. If the TCB is installed separately then the code sequence would be:

```
LD    DE,M8TCBQ
LD    BC,<Timer TCB Address>
CALL  M8SN
```

Note that the Timer TCB must be in RAM and the Timer Handler's stack must contain the Timer Task entry point in the Program Counter (PC) register locations within the stack. The entry point value is loaded in the stack by the MTHTCB macro.

Since the ETHTCB is "self-installing" its TCB should NOT be installed as specified above!

9.7 PROGRAMMING NOTES

When multiple timer Handlers are configured for the same CTC Chip, care MUST be exercised when specifying the I/O ports and vector addresses.

Add an additional count of 1 to the MBs STAT and PERD fields for timer rate count padding, if the timer must run for at least the specified period.

SECTION 10

MITE-80 SYSTEM FILES

10.1 INTRODUCTION

All of the system files provided on the MITE-80 diskette are outlined in this section. A majority of the files are provided in relocatable object format, while other files are in Z80 Assembler source format.

10.2 FILE LIST

The filenames are as follows:

Filename	Software Module
M8OBE80.OBJ	MITE-80 nucleus, relocatable object.
M8TCBQ.OBJ	TCB Queue header, relocatable object.
M80TH.OBJ	Timer Handler, relocatable object.
M80LNK.OBJ	MITE-80 DEBUG Linker module, absolute object.
M8MMGR.OBJ	Memory Pool Manager, relocatable object.
M80SYS.EQU	System equates, source.
M80SYS.MAC	System macros, source.
M80ESY.MAC	System executable macros, source.
M80DDT.BIN[X]	MITE-80 DEBUG, binary.(where X=16,32,48,64).

The system routines of MITE-80, MITE-80 DEBUG, Timer Handler, and Memory Pool Manager have all been discussed in previous sections of this manual. The other files are outlined as follows.

10.2.1 TCB QUEUE HEADER - M8TCBQ.OBJ

This file contains the initial MITE-80 Queue structure required to execute

MITE-80. It contains the null TCB to which all tasks are threaded.

10.2.1.1 USING M8TCBQ.OBJ

The M8TCBQ.OBJ file is used by linking this relocatable object file with the application files. An example procedure follows:

```
$LINK APPL,...,M8TCBQ,... TO USYS
```

10.2.2 LINKER MODULE - M80LNK.OBJ

This MITE-80 DEBUG link file, M80LNK.OBJ, provides the user application with the required linkages for MITE-80 debugging. This absolute object file contains the linkage addresses for debug purposes of all MITE-80 system services.

10.2.2.1 USING M80LNK.OBJ

The M80LNK.OBJ file is used by linking this absolute object file with the application object files. An example procedure follows:

```
$LINK APPL,...,M80LNK,... TO USYS
```

10.2.3 SYSTEM EQUATES - M80SYS.EQU

All MITE-80 system equates that a user program would require are contained in one file, M80SYS.EQU. This file is provided as a development aid and has all global and value equates for MITE-80 references defined. This file can be included in every user program that has references to MITE-80 services and labels. The labels defined in the M80SYS.EQU file are reserved as MITE-80 labels. When developing programs, the user must use ONLY these labels for MITE-80 references in order to prevent multiple defined label errors from occurring.

10.2.3.1 USING M80SYS.EQU

The M80SYS.EQU file can be used as an INCLUDE file in a user program. A listing of M80SYS.EQU is provided at the end of this section.

10.2.3.2 EXAMPLE

```
        :
        :
        INCLUDE   M80SYS.EQU    ;MITE-80 GLOBALS & EQUATES
        GLOBAL    ULAB1         ;USER REQUIRED
        GLOBAL    ULAB2         ;GLOBALS & EQUATES
ULAB3   EQU       0AAH
        :
        :
```

10.3 SYSTEM MACROS - M80SYS.MAC

The system macro file, M80SYS.MAC, contains all of the MITE-80 system macros in Z80 assembler source format. The purpose and use of these macros has been outlined in previous sections. All of the macros provided require the use of the MOSTEK MACRO-80 Assembler. The macros provided in the M80SYS.MAC file are:

MACRO	PURPOSE	REFER TO
MTCB	builds a TCB and stack	Section 7
M8POOL	builds memory pool	Section 8
MTHTCB	builds timer TCB and stack	Section 9

10.3.1 USING M80SYS.MAC

The M80SYS.MAC file is used as an INCLUDE file in a user program.

10.3.2 EXAMPLE

```
INCLUDE         M80SYS.MAC      ;MITE-80 SYSTEM MACROS
MTCB            ..........      ;TCB A DEFINED
MTCB            ..........      ;TCB B DEFINED
```

10.4 SYSTEM EXECUTABLE MACROS - M80ESY.MAC

The system executable macro file, M80ESY.MAC, contains all of the MITE-80 system macros that create executable code. The file is provided in source form. The macros provided in this file are:

MACRO	PURPOSE	REFER TO
ETCB	Builds a TCB, stack and installation code.	Section 7
ETHTCB	Builds Timer Handler TCB, stack, and installation code.	Section 9

10.4.1 USING M80ESY.MAC

The M80ESY.MAC is designed to be used in an absolute program segment, and generates executable code to transfer the TCB and stack structures from ROM/EPROM to RAM.

10.4.2 EXAMPLE

```
PSECT   ABS             ; ABSOLUTE PROGRAM SEGMENT
ORG     XXXXH           ; XXXX=LOCATION OF EXECUTABLE TCBS
 .
 .
INCLUDE M80ESY.MAC      ; MITE-80 SYSTEM EXECUTABLE MACROS
 .
 .
ETCB    ...........     ; TCB A DEFINED
```

```
ETCB    ..........      ; TCB B DEFINED
```

10.5 MITE-80 DEBUG - M80DDT.BIN[X]

The following binary executable files contain the respective MITE-80 DEBUG program which will reside at the upper end of memory:

```
M80DDT.BIN[16]  -  16K MITE-80 DEBUG SYSTEM
M80DDT.BIN[32]  -  32K MITE-80 DEBUG SYSTEM
M80DDT.BIN[48]  -  48K MITE-80 DEBUG SYSTEM
M80DDT.BIN[64]  -  64K MITE-80 DEBUG SYSTEM
```

The user must load the application tasks to be debugged and then execute the appropriate M80DDT file for the particular user's debug system configuration.

10.5.1 EXAMPLE

```
$GET USYS<CR>          - Application to debug

$M80DDT[x]<CR>         - Execute MITE-80 DEBUG for system configuration x
```

APPENDIX A

MITE-80 SYSTEM EQUATES

```
MITE-80 EQUATES   MOSTEK MACRO-80 ASSEMBLER  V2.2 PAGE   1
 LOC   STMT-NR SOURCE-STMT PASS2 M80SYS M80SYS M80SYS REL
                COPYRIGHT 1979 MOSTEK CORP

      * * * * * * * * * * * * * * * * * * * * * * * * *
      *         TITLE: MITE-80 SYSTEM EQUATES         *
      *                                               *
      *         ID:     VERSION 1.0                   *
      *                                               *
      *         DATE:   20-AUG-79                     *
      *         PROGRAMMERS:                          *
      *         PHIL MATHEWS                          *
      *         R.E. LEE                              *
      * * * * * * * * * * * * * * * * * * * * * * * * *

       16         GLOBAL CURTCB
       17         GLOBAL CURPRI
       18         GLOBAL CURNAM
       19         GLOBAL M8TCBQ
       20         GLOBAL NXTTCB
       21         GLOBAL M8SN
       22         GLOBAL M8SNW
       23         GLOBAL M8RSN
       24         GLOBAL M8RSNW
       25         GLOBAL M8FWD
       26         GLOBAL M8FWDW
       27         GLOBAL M8RCV
       28         GLOBAL M8RCVW
       29         GLOBAL M8CAN
       30         GLOBAL M8RET
       31         GLOBAL M8RETW
       32         GLOBAL M8FIND
       33         GLOBAL M8WINT
       34         GLOBAL M8PINT
          *
       36         GLOBAL RENTRY
```

```
MITE-80 EQUATES   MOSTEK MACRO-80 ASSEMBLER  V2.2 PAGE   2
LOC   STMT-NR SOURCE-STMT PASS2 M80SYS M80SYS M80SYS REL
          ;
=0000  38 STAT    EQU  0
=0000  39  TCBB:  EQU  0
=0001  40  MHBS:  EQU  1
=0002  41  MROB:  EQU  2
=0003  42  BKPT:  EQU  3
=0006  43  IWAT:  EQU  6
=0007  44  WAIT:  EQU  7
=0001  45 PRIO    EQU  1
=0000  46  FIFO:  EQU  0
=0002  47 LINK    EQU  2
=0004  48 RPTR    EQU  4
=0004  49 MPTR    EQU  RPTR
=0006  50 SPTR    EQU  6
=0006  51 OPTR    EQU  SPTR
=0008  52 NAME    EQU  8
          ;
=00FE  54 NULPRI  EQU  127*2
          ;
          ; TIMER HANDLER TCB DEFAULTS
          ;
=00FE  58 THVECT  DEFL 0FEH
=007F  59 THPORT  DEFL 07FH
=0001  60 THPRIO  DEFL 1          ;TIMER HANDLER PRIORITY
=0004  61 THLOOP  DEFL 4          ;TIMER LOOP CONSTANT
=007B  62 THSD25  DEFL 123        ;SD 2.486 MHZ TIME CONSTANT
=007D  63 THMD25  DEFL 125        ;MD 2.5 MHZ TIME CONSTANT
=00C8  64 THMD40  DEFL 200        ;MD 4.0 MHZ TIME CONSTANT
```

MOSTEK®

Z80·F8 Covering the full spectrum of
3870 microcomputer applications.

1215 W. Crosby Rd. • Carrollton, Texas 75006 • 214/323-6000
In Europe, Contact: MOSTEK Brussels
150 Chaussee de la Hulpe, B1170, Belgium;
Telephone: 660.69.24

PRINTED IN USA May 1980
Publication No. MK79726