RT/68MX

SYSTEMS MANUAL

MICROWARE SYSTEMS
CORPORATION
P.O. BOX 954 • DES MOINES, IOWA 50304

INDEX

TOM

The Microware Systems Corporation
P. O. Box 954
Des Moines, Iowa 50311
/(515) 279-9856
265-6121

Third Edition

Part Number RT68MXM

INTRODUCTION

Thank you for purchasing the RT/68 system. We hope that the power and versatility of the system will open new avenues of microcomputer system development and make your job easier as it has for us.

Please read this manual carefully, preferably in two passes. People have problems common to assemblers; forward references are hard to resolve. We have tried to arrange this manual in a somewhat logical order. Please pay particular attention to the handling precautions in the installation instructions as they are very important.

We hope the information contained herein is adequate to answer most questions about RT/68. However, if you encounter difficulties you may write or call us at (515) 279-9856 from 9AM to 4PM central time and we will attempt to offer the assistance you need.

We would also like to mention that if your application is for incorporation in a production system, another version that does not have the console monitor and I/O and includes support for more sophisticated interrupt and timing is available. Please write or call for details.

Several software products for RT/68 will be announced in the next few months. These include a new compiler, assembler, editor, and multi-user BASIC system. Purchasers of RT/68 will be informed as these become available.

If you develop tasks that you would like to share with others, please send us a writeup and/or listing. We will distribute a list of these to RT/68 purchasers from time to time and make copies available at cost of handling.

We also actively solicit comments and suggestions for improvement of both the RT/68 program and documentation. These will be of great value in future developments.

Again, thank you and happy programming.

The Microware Corporation

OVERVIEW

The RT/68 system is provided on a MCM6830D mask-programmed read-only-memory that is a direct replacement for the Mikbug(TM) ROM used in many M6800 systems. In addition to the functions performed by Mikbug(TM), the RT/68 ROM contains a 16-task real-time multiprogramming operating system.

RT/68 provides three modes which are mutually exclusive: Console Monitor to load, save and debug programs; Single Task Mode to execute existing Mikbug(TM) software without modification; and Multi-Task Mode which is the real time multiprogramming mode.

Sections of this manual are devoted to each of these modes. In addition, a source listing and information on installing and interfacing the ROM is included.

There are many subroutines in the ROM that may be called from a user program that can substantially save time and memory. An examination of the listing and the list of subroutines in the appendix can provide information on interfacing to these subroutines.

CONSOLE MONITOR MODE

The console monitor provides a convenient means to load, save, alter, and debug programs. Because the monitor and its initialization routine are entered automatically by the hardware restart vector, there is no need for any manual memory switch register or "bootstrap" program.

An I/O port (either a PIA or ACIA device) is used for communication of serial ASCII data to any type of terminal device. This may be a Teletype(TM), CRT terminal, hardcopy terminal, etc. In addition, a tape I/O device may be connected to the serial line and be used to load and store data in an "ASR" mode where the monitor program transmits ASCII control codes to enable and disable the tape devices. If a Teletype equipped with a tape reader control relay is used, a reader control signal is also available.

The terminal connected to the selected type of interface is referred to as the "console terminal device". All communication between the monitor and the operator is through the keyboard and display of this device.

Ten command functions are provided. They are:

Code/parameters	Function
D,aaaa,bbbb	memory Dump on console display, formatted
P,aaaa,bbbb	write (Punch) formatted object tape from mem.
B,aaaa	set Breakpoint at specified address
M,aaaa	enter memory examine/change function
E,aaaa	Execute program (single task mode)
G	Go to program on stack; return from breakpoint
L	Load memory from formatted tape
R	print Registers on stack
S	activate multi-task operating System
(ESC)	user defined

In the table above the symbols "aaaa" and "bbbb" refer to the beginning and ending addresses of the functions, respectively. These are represented as four hex characters seperated by commas. All other data entered or displayed is also in either two-character hex or four-character hex format. Entering the incorrect number of hex characters will result in an error, even if the value has leading zeros.

When the monitor is awaiting a command, it will print a $ character as a prompt.

The monitor also has logic to detect seven different types of error. An error message is printed with a code that specifies the type of error:

```
$B,012W ERR #3      (non-hex char in address)
$
```

The console monitor error codes are:

Code	Error
1	Tape load tried to write in ROM or non-existent memory
2	Checksum error on tape load function
3	Non-hex char in hex field
4	Format error in command string
5	No change in memory examine/change function
6	Illegal command code
7	Breakpoint error, SWI encountered at non-specified addr.

Note that when address parameters are required by a command, they are seperated by commas. The command may be cancelled by typing a carriage return anywhere a comma would normally be entered. Command lines are executed immediately after the last required data is entered and do not need to be terminated by a carriage return.

Each function in the console monitor is written as a subroutine which may be called from a user's program. The appendix contains information on interfacing to these routines.

On the following pages, descriptions of each console function are presented along with examples. In each example input entered by the user is underlined.

<u>Command</u>: Dump memory on console display
<u>Format</u>: D,aaaa,bbbb

<u>Operation</u>: This command causes the contents of memory locations aaaa through bbbb to be displayed on the console display device. The dump is specially formatted so it can be easily interpreted. A beginning address is printed to the left of each line followed by sixteen data bytes corresponding to the contents of memory. If the ending address of the dump is such that a line has less than 16 bytes, the last line of the dump will be shorter.

Note that if the beginning address of the dump has a zero in the least significant hex position (i.e., the address is an even multiple of sixteen) the bytes will be printed so the position on the line corresponds to the last digit of the address.

<u>Example</u>:

```
$D,0100,0146
0100 6B 6F 2F 4F 2F 6F 6F 6B 0B 4F EA 0B 2F 1F 0F 6F
0110 BF BD BF A9 BD BD BF AD BB BE BB BD BF FB BF BF
0120 6F 2B 0F 1F 6F 6E 4A 4F 2B 6F 6B 0F 4F 6F 6F 4B
0130 AD BD A8 BB B9 BB EB B9 BF BA FF BF FF FF FF FB
0140 77 7D D5 F7 F5 FD F7
$
```

<u>Command</u>: Punch (write) memory on tape
<u>Format</u>: P,aaaa,bbbb

<u>Operation</u>: A Mikbug(TM) formatted object tape of the contents of memory locations aaaa to bbbb is generated. A tape device operated in an ASR (automatic send-receive) mode that is connected to the serial data output line is used to generate the tape. The ASCII control codes for punch on (hex 12) and punch off (hex 14) are transmitted to control the device.

Before the data records are transmitted, a series of sixty null characters are transmitted. If a paper tape punch is used, these will form a six inch leader. If an audio cassette-type device is used, the null codes will provide a two second delay (at 300 baud) for the tape motor to come to full speed.

If the beginning address is <u>greater</u> than the ending address, only a leader will be generated.

An exact description of the tape format can be found in the appendix.

<u>Example</u>:

```
$P,E000,E036
S113E000C63C86118D108D70815326FA8D6A813934
S113E0102609C6348613F78007205A813126E75F24
S113E0208D338002B7A0028D1E8D2A7AA0022709A3
S10AE030A700A100260A0865
$
```

(depending on the interface circuit, the object data may not be displayed.)

Command: set Breakpoint
Format: B,aaaa

Operation: A breakpoint is set at program address aaaa. A breakpoint is a means of interrupting execution of a program at some specified address and is used as a debugging tool. When a breakpoint is encountered, the contents of all MPU registers are displayed, and the console monitor mode is entered. The registers or memory may be examined or changed. See the description of the "R" command for the register dump format.

Program execution may be resumed by removing or relocating the breakpoint and performing the "G" command. A breakpoint is removed by entering B followed by a carriage return only, and moved by typing B,aaaa where aaaa is the new address.

The breakpoint function operates by replacing the contents of the memory location with a software interrupt (SWI) opcode. This opcode is inserted just before program execution by the G, E, and S commands, and removed (and original opcode restored) just after it is encountered. This means that if the breakpoint location is displayed by the memory examine/change routine, the "true" opcode will be displayed. Also, the address of the breakpoint must be of the first byte of the instruction, and in read/write memory.

When a breakpoint is encountered, the register dump will be of the state of the MPU just prior to execution of the instruction at that address. Only one breakpoint is allowed.

If a SWI instruction at an address other than the one specified, a breakpoint error (error #7) message will be displayed. An exception to this is the program requested executive call used by the real-time executive, discussed in the section on the multi-task executive.

Note that if a breakpoint is set and not encountered, the SWI opcode will not be removed.

Example:

```
$B,0234                       (SET BREAKPOINT AT ADDRESS 0234)
$E,0100                       (EXECUTE PROGRAM BEGINNING AT 0100)

A042 D1 72 89 0336 0234       (REGISTER DUMP DUE TO BREAKPOINT)

$B                            (REMOVE BREAKPOINT)
$G                            (RESUME PROGRAM EXECUTION)
```

Command: Memory Examine/change
Format: M,aaaa

Operation: This function allows any memory locations or peripheral registers to be examined or modified on an individual or sequential basis.

The beginning address is given by aaaa in the command. The routine will print the address and data for that memory location. One of three operations may be performed:

1) The examine change function may be terminated by entering a carriage return
2) The next sequential location may be opened by entering a line feed
3) The contents of the location may be changed by entering a / character followed by the new data.

If the location's contents do not change, an error will result (error #5). This may occur if the location is non-existant, defective, or ROM. Certain peripheral control registers (PIA's, ACIA's) have control registers that are mixed read/write and read only, which may also result in an error.

In the example below, an up arrow corresponds to a line feed, and a curved arrow to a carriage return.

Example:

```
$M,1200         (OPEN FUNCTION BEGINNING AT 1200)
1200 D3↑        (ADVANCE TO NEXT)
1201 27↑        (ADVANCE TO NEXT)
1202 33/34      (CHANGE TO 34)
1203 A7/00      (CHANGE TO 00)
1204 FF↲        (CLOSE FUNCTION)
$
```

Function: Go to program on stack
Format: G

Operation: Used to initiate execution of a program whose register values are stored on a stack. The stack is defined by the contents of a RAM location called SPTMP (addresses A008-A009). This location is initialized to A042 upon system reset, or the current stack pointer value when a breakpoint or abort occurs.

This means that the G command will execute a program with a starting address stored in locations A048 and A049 (SP+6 and SP+7) after a restart; or resume program execution after an abort or breakpoint. After a restart, the G will result in Single Task Mode execution. After an abort or breakpoint, the mode will be dependent on the last mode before interruption.

Command: Load tape into memory
Format: L

Operation: This command will cause a Mikbug(TM) format tape to be loaded into memory. The format is described in the Appendix. There are three types of tape records: header, data, and end of file. Header records (as well as any other data not preceded by an S1) are ignored. The function is terminated by reading an end of file record (S9).

The ASCII codes for reader on (11) and reader off (13) are transmitted to control the reader device. Additionally, the control terminal PIA output CB2 may be used to enable a teletype paper tape reader.

Two errors may occur during a tape load. If the checksum read at the end of each data record does not agree with the checksum generated by the load routine as the block is read, an error message will be printed and the load will terminate. The tape may be backed up a block or so and another read attempted. Each block has an individual starting address so it is not necessary to start from the beginning. If an attempt is made to load non-existant, defective, or ROM memory, an error will also occur and the load stopped.

Command: Display registers
Format: R

Operation: The contents of the MPU registers on the current stack are displayed. The current stack is defined by the contents of the location SPTMP. See the "G" command for more information.

The format of the register display is:

```
A042 C1 00 33 21FF 0103
(SP  C  B  A   XR   PC)
```

The registers may be altered by changing the corresponding locations on the stack. These locations are relative to the value of the SP. the CC register will be at SP+1, the B register at SP+2, etc.

Example:

```
$R                              (REGISTER DISPLAY COMMAND)
0842 D3 45 02 1000 039A         (CONTENTS OF REGISTERS)
$M,0846                         (OPEN XR STACK LOCATION [SP+6])
0846 10/0F                      (CHANGE HI BYTE)
0847 00/FF                      (CHANGE LO BYTE)
0848 03                         (CLOSE MEM. EXAMINE/CHANGE)
$R                              (REGISTER DISPLAY COMMAND)
0842 D3 45 02 0FFF 039A         (REGISTER CONTENTS)
$
```

Command: Activate Mulitask Operating System
Format: S

Operation: This command initializes the RT/68 multitask operating system and give it control of the MPU. The executive will then begin the process of searching the task status table for the highest priority task to execute. If an initialization task is required, it should be task #1 and given highest priority, or all other tasks should be off.

The S command resets the CLOCK value to zero. It does not give the system an initial priority level. This should be established before the S command is executed. This command sets the RT mode flag.

Command: Execute Single Task
Format: E,aaaa

Operation:This command jumps to address aaaa to execute a program. If the task (program) is terminated with an RTS instruction and does not reestablish a stack, control will be returned to the monitor automatically.

Command: User Defined
Format: (ASCII escape character)

Operation: Jumps to a subroutine at a fixed address of 7ØØØ (hex). A ROM, PROM or RAM at this address may provide any desired function. This command was included with a future disk operating system bootstrap PROM in mind.

PROGRAM ABORT

When the optional abort switch is added to the 6800 system, an abort (a type of NMI interrupt) will cause an operation identical to that resulting from a breakpoint. The program is not altered in any way. Execution may be resumed by means of the G command.

The abort is not active when the console monitor is in use, and may not work under very rare circumstances if a program is out of control and changes the control terminal PIA status register.

SINGLE TASK MODE

The single task mode is a provision of the RT/68 system that allows programs that were written for Mikbug(TM) and cannot run as tasks to have a means to execute without modification.

The following list describes attributes that would make a program unsuitable for execution as a task without modification:

1. The program does not use or continuously maintain a stack. Some programs will use the stack pointer as an index register (poor practice) and therefore cannot tolerate interrupts.
2. The stack does not have enough room. Generally, if the program used interrupts, it will have enough room.
3. The program uses system I/O that must be shared with other tasks but there is no provision for coordination. Tasks cannot share a printer on alternate characters. This may or may not be a problem, depending on the I/O requirements of the other tasks.
4. There are critical software timing loops in the program. One notable example of this is the software UART input and output routines that use the control terminal of this system. This routine cannot be interrupted in the middle of a character.
5. The program uses interrupts. The interrupt driver routines must be modified. A simple task that handles the interrupt processing is required.

The single task mode allows programs with any of the above characteristics to run without any modification. Either the E command or G command (after reset) will accomplish this. Interrupts are handled as straight vectors as in Mikbug(TM).

OVERVIEW OF MULTIPROGRAMMING

A multiprogamming system is an implementation of a control framework in a computing system where several sequential processes are allowed concurrent execution by allocating processor time alternately to the processes.

A less formal definition is a scheme where a portion of total CPU time is given to each task so they have the appearance of executing simultaneously. In the RT/68 system, each process is called a _task_, and up to 16 may be active at once. All of the tasks are assigned numbers and must be present in memory. They may or may not be independent, and may control the execution of each other and communicate by means of _flags_ or _common data areas_. In many systems, the tasks may compete for system resources such as I/O devices so some degree of coordination may be required.

In addition, RT/68 is a _real-time_ system meaning that it is designed to provide control mechanisms for responding to time dependent or external events (such as interrupts). This poses a greater degree of both power and complexity.

MEMORY MAP OF A TYPICAL RT/68 CONFIGURATION

top of memory	Operating system and utility routines
	free memory
	task #4 program and data area
	task #3 program
	task #2 program
	task #2 data area
	task #1 program
	task #Ø program
bottom of memory	common data area

TASKS

A task can be defined as a complete unit of object code that can compete for system resources independently. In this sense, a task is a module that has the necessary attributes to exist as a runnable entity within the RT/68 environment.

There are no restrictions to the absolute size or function of a task. Because all tasks must be memory resident, a task must be small enough to share available system memory with other tasks.

Any program that can be run by the M6800 can be written as or modified to be a single task or in some cases, a group of tasks. Often, the process of dividing a long, complex program into smaller coordinated tasks allows for easier program development.

The reason for creating tasks in a multiprogramming system is straightforward. The MPU divides its available time between several different functions, and the main control program (executive) needs to have a means for distinguishing between each entity. By formally defining what comprises a task, a standard set of conventions allows a degree of consistency in system development that would not be possible otherwise.

We may therefore define what is a RT/68 task:

An RT/68 task has as components:

1. One or more memory patterns which are machine level instructions which perform some function independently.
2. One or more data areas in memory that are unique to the task; or shared with other tasks.
3. A unique stack area in memory which is continuously maintained by the task for its exclusive use.
4. A task status byte in the corresponding place in the system status table, which contains status information and operating parameters.
5. A task stack pointer word in the system status table which holds the current value of the task's stack position in between task executions.

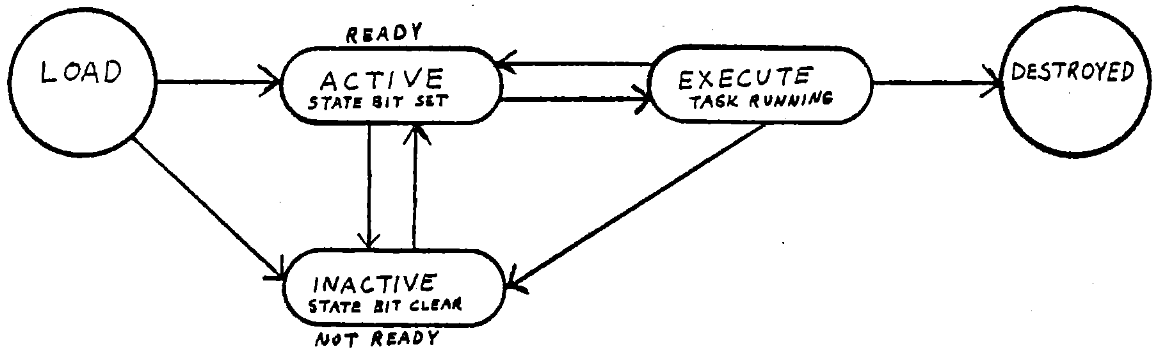
It is obvious from the definition that the stack is an integral part of the tasks structure. For more information on the stack operations of the M6800 consult the M6800 Programming Manual. It is the stack that RT/68 depends on for starting, stopping, and saving task status.

A task is prepared or created from a source program by an assembler or compiler. Normally all tasks in a system would be loaded by means of the console monitor load routine before the system is activated, but it is not difficult to create a "master task" that has the capability to load, initialize, and perhaps create other tasks.

Once a task is a part of memory, it will have one of three mutually exclusive states at any given instant. A task is either active (ready to run), inactive (not ready) or in execution (has control of the MPU and is actually running). The state diagram illustrates the possible paths between each state.

The inactive state has a dual purpose. The executive "thinks" there are always 16 tasks regardless of how many actually exist. Thus it is necessary to establish dummy status information making non-existent tasks appear to be inactive tasks, because inactive tasks may not ever run in that state.

RT/68 TASK STATE DIAGRAM



Tasks have the ability to call subroutines in the RT/68 ROM that may be used to control the state and status of themselves or other tasks. This ability is an extremely powerful tool to allow for coordinated, prioritized interaction of tasks on a time, event, and calculated basis.

For example, suppose a "main" task is running, busy calculating pi to ten thousand places, when a hardware sensor detects a fire in the computer room and generates an interrupt. Task X is activated by the RT/68 interrupt system, and polls peripherals to determine which caused the interrupt. Task X determines it was the fire detector, activates Task Y which is the fire sensor device routine, then deactivates itself and relinquishes control of the MPU.

Task Y then runs and reads temperature data from various points in the computer room, as well as the time of day. It determines that the fire is not too bad and it is the beginning hour of the second shift so someone is likely to extinguish it. Rather than activating Task Z (which is the device driver for the sprinkler system) the task reactivates the original task which still has 9,500 places to calculate.

Note that often in the preceding example, tasks made decisions that controlled the execution of other tasks by altering their state. The real-time aspects were the response to an external event (the fire sensor interrupt) and to time dependent data (determination of the time of day). More elaborate constructions can be made as well. Task Y might have played it safe and rescheduled itself to run ten minutes later to make sure the fire did indeed get extinguished. The timed task capability of RT/68 will allow this.

This example did not illustrate (but implied) the task prioritizing capability of the operating system. Had the original "main" task had a higher priority than the interrupt handler then the fire sensor task may have had to wait for the main routine to complete its calculation. Assuming this time would be much longer than the time it would take the computer room to be gutted, a classic "deadlock" situation would develop.

A deadlock occurs when two tasks(possibly more) have control over some resource, device, or task that is required by some other task and will not be released by the other task until the first releases something it needs to proceed, etc. Careful thought (and state diagrams) can prevent this situation from arising.

In addition, the executive has the capability of allocating fixed quanta of MPU time to several tasks in round-robin fashion. Had task Y been truly cautious, it might have put itself on an equal priority with the main task and allowed itself some ratio of the total MPU time to continuously keep tabs on the fire, delaying calculation of pi slightly by sharing MPU time with the main task.

An excellent tutorial article on multiprogramming coordination by Leon Presser may be found in ACM Computing Surveys for March, 1975. This journal is available from most city libraries or university libraries. It is highly reccommended for any RT/68 system programmer. Many examples given in the article are directly applicable to RT/68.

TASK STATUS TABLE

The task status table is the main data structure of the RT/68 operating system. It occupies addresses AØ5Ø thru AØ7F in the scratchpad RAM. The table contains status, operating parameters, and stack pointer values for each of the 16 possible tasks.

Each task has associated with it a 3-byte area of the table called the task status word (TSW). The task status word has as components a task status byte (TSB) and a 2-byte task stack pointer.

TASK STATUS TABLE

AØ5Ø	TSB Task Ø	TSW TASK Ø
AØ51	TSP Hi Byte	
AØ52	TSP Lo Byte	
AØ53	TSB Task 1	TSW TASK 1
AØ54	TSP Hi Byte	
AØ55	TSP Lo byte	
.	.	TSWs TASKS 2 - 14
.	.	
AØ7D	TSB Task 15	TSW TASK 15
AØ7E	TSP Hi Byte	
AØ7F	TSP Lo Byte	

To find the address of any TSW, the following formula may be used. The numbers are hexadecimal.

$A_n = AØ5Ø + (3 * n)$ where n is the desired task #

A subroutine in the RT/68 ROM can be used to locate the TSW. The task number is loaded in accumulator A and the subroutine FNDTSB (address E33B) called. This subroutine will return with the index register pointing to the TSB of the correct TSB.

The task stack pointer holds the information necessary to start and restart tasks. The section on "Task Switching" deals with this in greater detail.

The RAM area used for this table is used only by the multi-task executive. If the single task mode is being used exclusively, this RAM may be used for any other purpose.

There must be entries in the TSB portion of the status table even if the corresponding tasks do not exist, as random data might be interpreted as status information. It is a good idea to clear all TSB's that are not used by real tasks.

TASK STATUS BYTE

The task status byte contains all operational and status parameters for each task. Packed into the eight bits are the state flag, a time limit value, and a task priority level.

Bit 7 is the task state flag. It is set to indicate that the corresponding task is active and should be considered by the executive during the task selection process. Tasks may use this flag to control the execution of other tasks. State flag bits (and preferable the entire TSB) should be cleared if no task exists for the corresponding task number.

TASK STATUS BYTE (TSB)

7	6 5 4 3	2 1 Ø
FLAG	TIME LIMIT	PRIORITY
state flag 1=active Ø=inactive	time limit in ticks range 1 to 15; or Ø=unlimited time	task priority range Ø to 7

Bits 3 through 6 contain a time limit which determines the maximum length of the corresponding tasks time slice per execution turn. If the value of this number is equal to zero, the task will not be "timed out" unless an interrupt or program requested executive call occurs.

Bits Ø through 2 are the task's priority level. This is used by the executive to determine the ability of a task to run in relation to the other tasks. This is covered in detail in the "Task Switching" and "Task Scheduling" sections.

There are several subroutines in the RT/68 ROM that can be used to affect the TSB.

They are:

TSKON - set state flag of selected task
TSKOFF - clear " " " " "
FNDTSB - find the address of selected TSB
CTSKOF - clear state flag of current task TSB

In all cases except CTSKOF, the number of the desired task is loaded in ACC A before the subroutine is called. The subroutine will return with the address of the TSB in the index register. CTSKOF will automatically determine the number of the current (calling) task.

TIME SLICES

Tasks may be allocated a specific maximum amount of time to run for each of their turns. The time in between is shared by other tasks at the same priority level.(if any).

The time period for which the task may run is called a _time slice_ which is measued in units called _ticks_. A tick is the time interval between positive transitions of the external clock reference signal which is connected to the control PIA. The precise duration of a tick is the inverse of the frequency of the clock signal. For example a 100 Hz clock will result in a tick time of 1/100 second or 10 milliseconds. With this reference frequency, a task allocated 13 ticks per turn will run for 10 ms. x 13 = 130 ms.

The fact that the basic unit of time in the RT/68 system is a _relative_ unit allows flexibility in choice of time scales and resolution.

The time slice duration is determined by the time limit value packed in the task's TSB. This may range from one to fifteen ticks. If the time limit is zero, the task will not be time restricted.

When a task is started, the executive unpacks the time limit from the TSB and places it in a RAM location called TIMREM (address ØØØ2). On each clock interrupt, this value is decremented, and the task is suspended upon a one to zero _transition_. If the initial value is zero or the task changes it to zero, there will be no transition and the task will not be time limited (TIMREM is never decremented if zero).

In addition, the TIMREM value has a full byte and can represent a number as large as 255. A task may change its running time to a value greater than fifteen by altering this location.

The task may not run for its full time slice. It is possible that an interrupt occuring during its turn will cause it to be suspended before expiration of the time slice. The remaining time is then "lost".

A task may also give up whatever time remains in its turn by executing a _program requested executive call_. The subroutine below will accomplish this. It is reentrant and relocatable.

```
0800 7C A00E GIVEUP INC    RELFLG          SET RELEASE FLAG
0803 01             NOP
0804 0F             SEI                    SET IRQ MASK FOR ERROR DETECT.
0805 3F             SWI                    SOFTWARE INTERRUPT CALLS EXEC
0806 0E             CLI                    CLEAR IRQ MASK
0807 39             RTS                    RETURN
```

TASK SELECTION

There may be up to 16 tasks resident in memory at once. Each task's status information consisting of the state flag, a priority level, and time limit is packed into the Task Status Byte. The TSB, along with the value of the task's stack pointer is contained in the System Status Table. When the executive needs to select another task to run, the information in this table, along with certain system parameters, are used as the basis for the selection process.

The process of new task selection occurs:

1. When the system is first activated by means of the Console Monitor's "S" command.
2. When a non-deferred interrupt occurs.
3. When a task executes a programmed executive call.
4. When a running task's time limit expires.

The executive will select the next task on the basis of which task has the highest priority level that is greater than or equal to the system priority level, and is active as defined by it's state flag.

If more than one task is active and at the same priority level which is higher than that of any other tasks and the system priority level, all the tasks at that level will run in numerical order of their task numbers, round robin fashion.

If there are no tasks active and at a level at least equal to the system priority level, the executive will retain control until a task of sufficiently high priority becomes active (generally due to an interrupt).

In any case the only task or tasks to run are those that are active and at the highest priority level.

As an example, suppose we have a system in which there are 7 tasks with the following attributes:

Task	Priority	State
Ø	4	Active
1	2	Active
2	7	Inactive
3	4	Active
4	4	Inactive
5	4	Active
6	3	Active

System Priority = 3

When initialized, the order of execution will be Ø,3,5,Ø,3,5,Ø,3,5....... Task #1 will not run because it is at a lower priority level than both the other active tasks and the system priority level. Task 2, though of a higher priority, will not run because it is inactive, the same reason task 4 will not run. Suppose an interrupt occurs and activates task #2. Task #2 will then be the only task to run because it has a higher priority level than any other task.

TASK SWITCHING

Tasks are switched whenever the executive is activated, always due to an interrupt of one type or another. In a program-requested executive call, a SWI interrupt is executed. A time slice expiration will occur on the last NMI clock interrupt, as will a timed task interrupt.

When a task is interrupted, the MPU's hardware causes all the registers in the MPU to be pushed on the task's stack. Suppose the task's stack looked like this just before a clock interrupt that is the last of the task's time slice:

	addr	data	
	Ø11F	22	(data)
	Ø11E	3A	(data)
SP =	Ø11D	ØØ	(empty)
	Ø11C	6F	(empty)

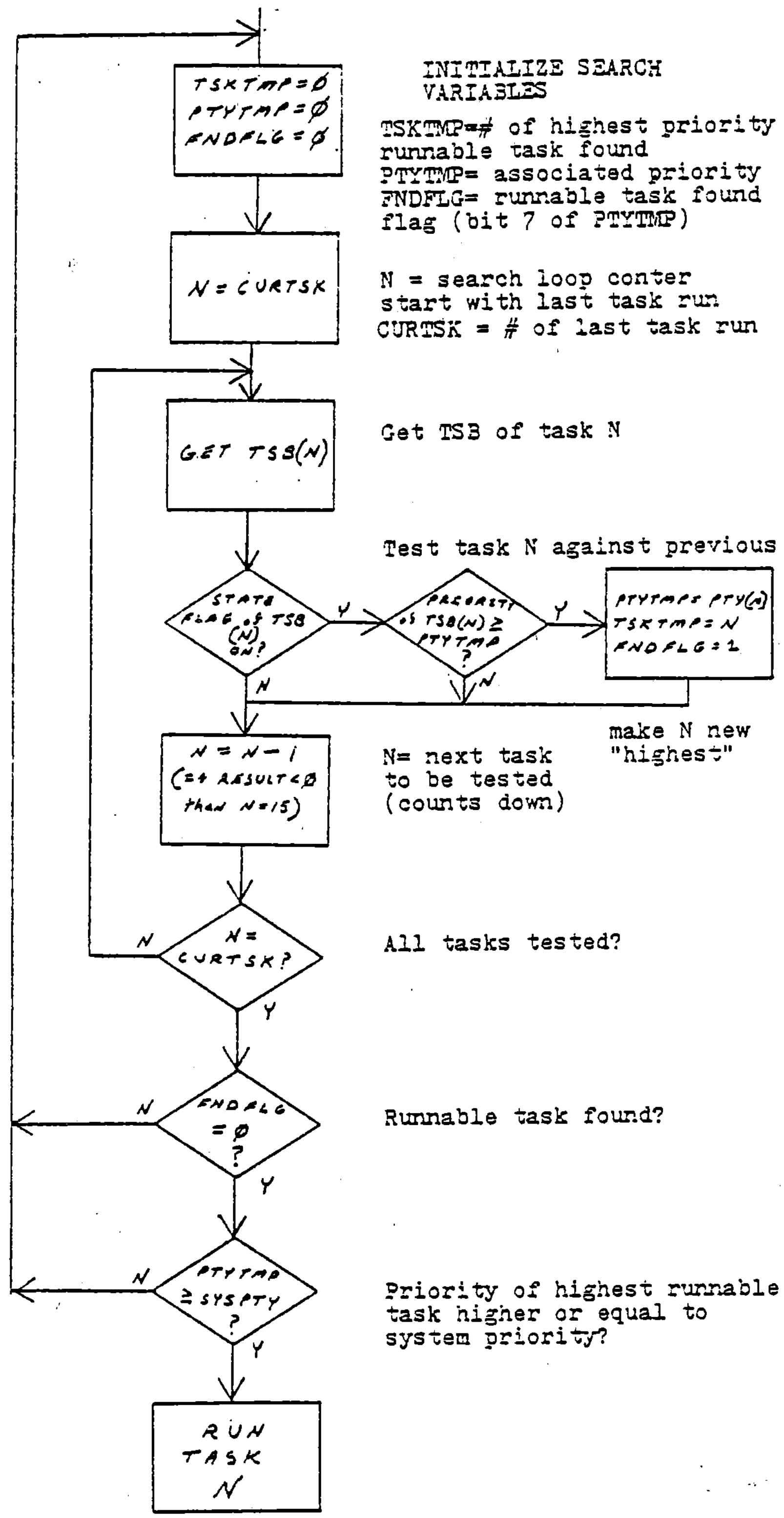
The stack pointer has a value of Ø11D which is the location of the next "empty" byte of the stack. The stack builds downward when data is pushed (placed on the stack). Now the interrupt occurs, the registers are pushed on the stack, which now looks like:

	Ø11F	22	(undisturbed)
	Ø11E	3A	(undisturbed)
	Ø11D	12	(program counter - next instr. addr.)
↓	Ø11C	1Ø	(program counter - low byte)
	Ø11B	AØ	(index register high byte)
	Ø11A	39	(index register low byte)
	Ø119	Ø1	(accumulator A)
	Ø118	22	(accumulator B)
	Ø117	D3	(condition code register)
SP =	Ø116	38	(next empty)

At this point, the stack pointer alone is sufficient information to be able to store and still be able to restore the exact state of the MPU during a later operation. The executive then loads the saved value of the stack pointer and executes an RTI instruction which causes the reverse of the process above to occur (the register "images" on the stack are pulled back into the corresponding registers) and task execution to resume.

A stack pointer value for each task is saved in the System Status Table for this purpose. Note that because the system starts tasks in this manner, when a task is first loaded the beginning address must be in the appropriate location on the stack.

TASK
SELECTION
SYSTEM
FLOWCHART



INTERRUPT PROCESSING

This section deals with a topic that has a greater potential for confusion than any other. This is because the RT/68 firmware uses interrupts for internal functions and to synthesize a fourth type of "firmware interrupt". To minimize confusion, we will define two new types of "user" interrupts and give them distinctive names.

The first is the user NMI, called UNMI. This refers to any NMI interrupt not caused by the reference clock signal or the abort switch, i.e., an NMI caused by some user device. An NMI becomes a UNMI after preprocessing by the RT/68 NMI routine, it having been determined not to be an internal interrupt.

The second is a firmware creature that is synthesized from the NMI/clock interrupt by the internal NMI handler. It is called the timed task interrupt (TTI) which is in effect a 16 bit presettable timer that causes an interrupt when it reaches zero. It is decremented every tick (but not less than zero) and implemented in software. It is given the same characteristics as other interrupts.

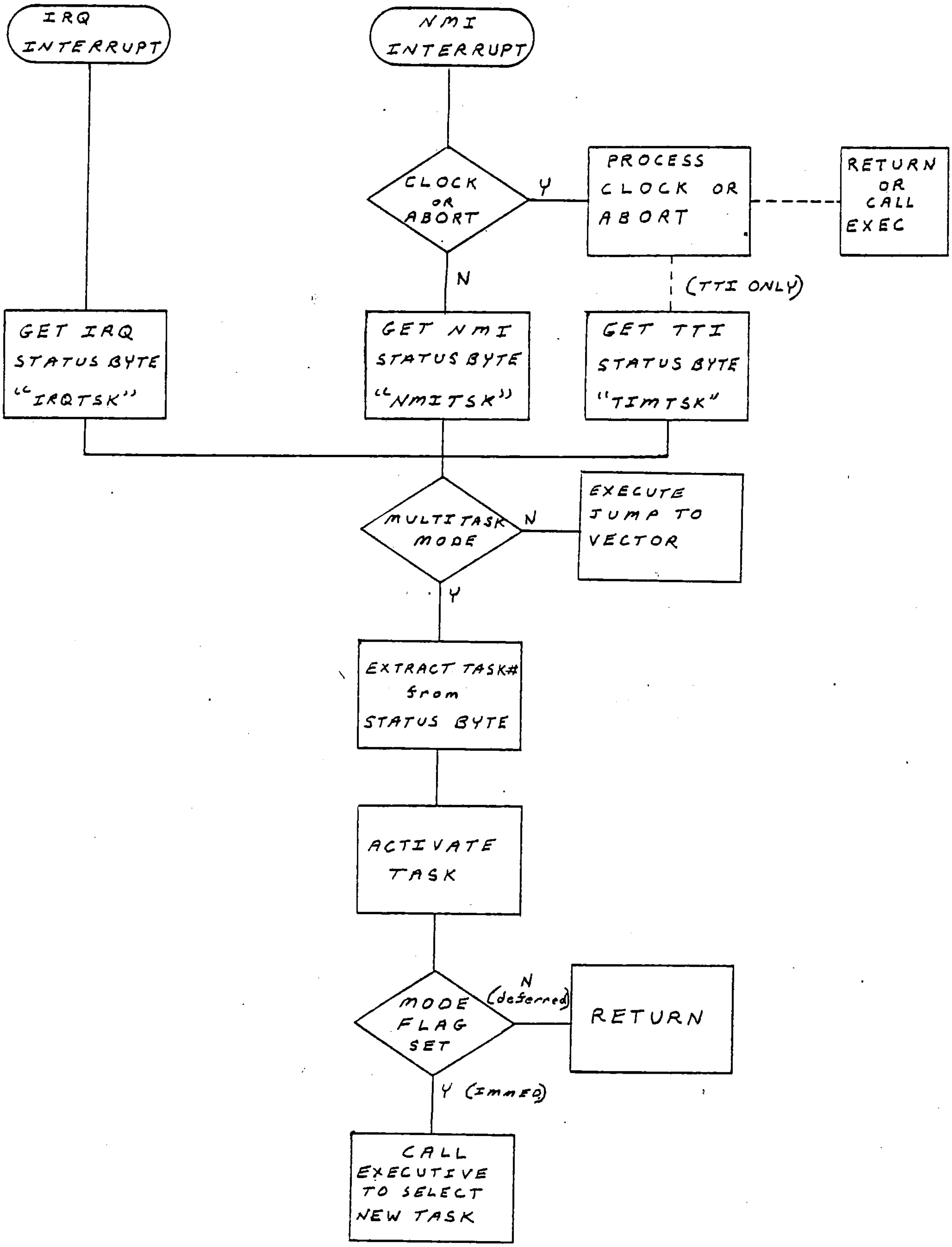
We shall also consider IRQs which in RT/68 can only be caused by user devices. We shall not consider the SWI interrupt because it is used for "internal" functions only.

Now the three type of interrupts of interest (UNMI,TTI, and IRQ) have been defined another powerful characteristic of the RT/68 system may be considered. All three interrupt types are handled by the same general service routine and may be totally software prioritized and scheduled. Each type has an associated status byte and task. The status byte contains two values: a mode flag and a task number corresponding to the task which is to perform the user's interrupt service.

The status bytes are named NMITSK, IRQTSK, and TIMTSK. Bit 7 of the byte is the mode flag. Bits Ø through 3 comprise the associated task number.

When an interrupt occurs, the executive immediately fetches the appropriate status byte. The task number is extracted and the designated task is activated by setting the task's state flag in its TSB. The mode flag of the interrupt status byte is sampled and if it is clear, the interrupted task is resumed. The interrupt has been deferred for later execution of its service task.

If the mode flag is set, the executive is called to select a new task to run. Note that the next task to execute will not necessarily be the interrupt task. The task selected will be the one that has the highest priority.



INTERRUPT SYSTEM PROCESSING FLOWCHART

A summary of the interrupt process:

1. interrupt occurs
2. system fetches either IRQTSK, NMITSK, or TIMTSK status bytes, depending on the type of interrupt.
3. The task specified in the status byte is turned on (activated) by setting its state flag.
4. The mode flag (B7) in the status byte is sampled. if clear, the interrupted task is resumed. If set, the executive is called to select the next highest priority runnable task (not necessarily the interrupting task).

The operation of this interrupt handling system is designed so it will never alter the normal heirarchy of task execution, that is it will never force a task to run that has a lower priority than some other runnable task.

<u>Interrupt example #1</u> - Prioritization following MPU hardware priority.

The example system has seven tasks numbered Ø to 6 Tasks Ø-3 are the "background" tasks that are to run round-robin between interrupts. Tasks 4,5 and 6 are the IRQ, TTI and NMI service tasks respectively. The system data values are set up as follows:

```
TSB(Ø) = 1 1ØØØ ØØ1     (active, time limit=8, priority=1)
TSB(1) = 1 1ØØØ ØØ1     (   "       "     "  =8      "     =1)
TSB(2) = 1 1Ø1Ø ØØ1     (   "       "     "  =1Ø     "     =1)
TSB(3) = 1 ØØ11 ØØ1     (   "       "     "  =3      "     =1)
TSB(4) = Ø ØØØØ Ø1Ø     (inactive   "     "  unlim.  "     =2)
TSB(5) = Ø ØØØØ Ø11     (   "       "     "  unlim.  "     =3)
TSB(6) = Ø ØØØØ 1ØØ     (   "       "     "  unlim.  "     =4)

IRQTSK = 1 xxx Ø1ØØ     (immediate-specified task = 4)
TIMTSK = Ø xxx Ø1Ø1     (deferred -specified task = 5)
NMITSK = 1 xxx Ø11Ø     (immediate-specified task = 6)

SYSPTY =   ØØØØØØØ1     (system priority = 1)
```

When this system is first activated, tasks Ø,1,2 and 3 will run round robin completing a cycle of task executions every 29 ticks (the sum of the four active task's time slices). The three interrupt service tasks do not run because they are inactive.

Assume a timed task interrupt occurs. TIMTSK is sampled, and task #5 is activated, but control returns to the interrupted task because the mode flag is clear. However, task 5 will run immediately following completion of the current task's time slice because it has a higher priority (3) than the others. Now assume that the current task is complete and the executive starts task #5. Now an IRQ interrupt occurs, the IRQTSK is sampled and task #4 is activated. The mode flag in IRQTSK is set so the executive is called to select a task. Task #5 will be selected again and run continuously because it still has the highest priority. After some time, it will complete its function, turn itself off (deactivate itself by clearing its mode flag in the TSB) and call the executive.

The executive will now execute task #4 (IRQ service) because it has become the task with the highest priority. At this point, assume an NMI interrupt occurs. NMITSK is sampled and task #6 is started. Because it has the highest absolute priority in the system and has unrestricted time, it will unconditionally run until it completes its function, deactivates itself, and calls the executive. At this time, task 4 (the IRQ service task which was interrupted) will run until completion if not interrupted again. When it is finished, tasks Ø through 3 will resume round robin operation again.

Interrupt example #2 - Interrupt task entry into queue.

This example has five tasks. Tasks Ø and 1 are the normal background tasks. Tasks 2, 3 and 4 are the interrupt service tasks.

```
TSB(Ø) = 1 1111 Ø1Ø  (active, time limit=15, priority=2)
TSB(1) = 1 Ø11Ø Ø1Ø  (active,    "     " = 6,      "   =2)
TSB(2) = Ø 1ØØØ Ø1Ø  (inactive,  "     " = 8,      "   =2)
TSB(3) = Ø 1ØØØ Ø1Ø  (inactive,  "     " = 8.      "   =2)
TSB(4) = Ø ØØØØ Ø11  (inactive,  "     " unlim.    "   =3)

IRQTSK = Ø xxx ØØ1Ø  (deferred, specified task = 2)
TIMTSK = Ø xxx ØØ11  (deferred,     "       "  = 3)
NMITSK = 1 xxx Ø1ØØ  (immediate,    "       "  = 4)

SYSPTY=Ø
```

In this case, before any interrupt, tasks Ø and 1 will alternately run until an interrupt occurs. Note that tasks 2 and 3 (the IRQ and TTI) tasks have a priority equal to the "background" tasks, as well as a time limit. When these interrupts occur, they will join the queue and execute round-robin with the other tasks. The NMI task has a higher priority and unlimited time, so after an NMI task 4 will execute continuously until completion.

```
Execution order before interrupts: Ø, 1, Ø, 1, Ø, 1.......
    "         "   after  TTI : Ø, 1, 3, Ø, 1, 3 ........
    "         "     "    IRQ : Ø, 1, 2, 3, Ø, 1, 2, 3.......
```

The preceding examples illustrate only two simple possibilities for interrupt programming. The possibilities for other structures is virtually unlimited. Consider that a task (or tasks) can dynamically alter the status bytes TIMTSK, IRQTSK, and NMITSK allowing several tasks to be associated with one type of interrupt. Alternately, one task may serve as the handler for two or three interrupts (particularly useful if the service task is reentrant) by having the status bytes specify the same task.

The interrupt system can also alter various task and the system priorities,etc.

Hardware Interrupt Considerations

The M6800 MPU has two external interrupt inputs, IRQ and NMI. These are "wire-ored" in most systems following the protocol that a peripheral requesting an interrupt "pulls down" the appropriate signal line and the software service routine polls the peripheral status registers to find the interrupting device. The device's interrupt flag then releases the line. The IRQ interrupts may by inhibited by setting the interrupt mask bit in the MPU's condition code register.

The interrupting device on the IRQ system must release the IRQ line before the interrupt mask bit is cleared or another interrupt will result due to the same occurence. Return from interrupt instructions (RTI) reload all MPU registers including the condition codes from the stack. The RT/68 system uses this instruction to start tasks. It is important therefore for the IRQ service tasks (and tasks of higher priority) to have "their" interrupt mask bit set until they clear the peripheral register's interrupt flag. In practice, the task should run with this bit set all the time, and release control to the executive following preliminary interrupt identification. A good system involves the use of one task solely to poll peripherals and clear interrupt bits, then activating a specific service task.

The NMI system may not be masked. However, it is possible to <u>pulse</u> the NMI line for about 5usec to cause an NMI. It is important that NMI tasks in a system where NMIs are wire-ored are not deferred too long because abort or clock interrupts will be inhibited as long as this line is held low. Generally the NMI system is used for critical peripherals (such as the clock) only and should not be heavily loaded. Most systems will operate optimally on the TTI and IRQ systems alone.

Hardware-Caused Interrupt Errors

Almost all M6800 MPUs manufactured until very recently have a quirk where if a SWI and NMI occur at the same time an error condition would occur. The problem is that the MPU would fetch the interrupt vector from the IRQ location instead of either the NMI or SWI location. Because RT/68 makes frequent use of all three type of hardware interrupts logic was added to the interrupt processors in RT/68 to detect and correct this error. A SWI istruction alone as used to call the RT/68 executive is not entirely reliable. The following code must be added when the SWI is used:

```
NOP
SEI
INC $AØØE
SWI
CLI
```

Page A-11 (appendix A) of the <u>M6800 Applications Manual</u> elaborates on this subject.

The code above also illustrates another glitch. The SEI instruction may not operate properly unless the preceding instruction is a NOP (no operation).

Another potential problem concerns reading of PIA data registers by routines that are not part of interrupt service. It is possible for the read of the data registers to clear the interrupt flag before the interrupt service routine polls it. This happens when the interrupt occurs during the PIA read instruction.

PIA configuration that eliminates this problem is the best solution. Otherwise, additional code to correct the problem (unidentified interrupts) may be necessary.

Interrupt Handling In Single Task Mode

When the multi-task executive is not active, the interrupt routines for NMI and IRQ use the contents of memory locations AØØ6-AØØ7 and AØØØ-AØØ1 respectively as jump destinations.

This is identical to the operation of Mikbug(TM). In single task mode the contents of memory loaction $AØØE must be zero.

Timed Task Interrupts

The TTI is a feature that allows precise generation and measurement of time. A 16 bit value in memory called TSKTMR (addr ØØØ3-ØØØ4) is decremented by the clock interrupt service routine contained in the executive at every clock interrupt (tick).

It is not decremented if zero, in which case it is inactive. Upon a transition from a count of one to zero a TTI interrupt will occur.

The range of this timer is 1 to 65,535. The absolute time is dependent on the clock input frequency. If a 60 Hz clock is used, each tick is 1/60 second long and the maximum time possible is 1092.25 seconds or 18 minutes, 12.25 seconds. The resolution of the timer using a simple load-counter-then-wait is ± 1 tick. The software can sync the clock by first giving the counter a time of one tick, then resetting the timer to the desired value as soon as it becomes zero. This is necessary because the time the counter is set is random in respect to the time the next decrement occurs, so the first interval may be a random fraction of a tick long.

Even more precise generation may be accomplished by having external sync or counters that may be reset by some PIA or other peripheral input. The SWTPC MP-T timer circuit may be used by connecting the interrupt output of the card to the clock input on the control interface card (and not connecting it to the IRQ or NMI on the system bus). Write for further details on interfacing.

Real Time Reference Clock

This 16 bit value called CLOCK (addr ØØØ5) may be used for time reference by tasks. It is incremented at each tick and is not used by the system so it may be set, cleared or otherwise changed any way desired.

Once again, the absolute time is dependent on the input clock rate.

Interrupt Service Time

It takes the RT/68 executive about 1ms. to switch tasks. This means that interrupts serviced by a simple immediate-execution system have a top rate of approximately 1K intr./sec. Scheduled (deferred) interrupt service with a master interrupt service task can reduce the number of context (task) switches and achieve much higher rates. This applies to user interrupts, system clock interrupts are serviced in about 150 microseconds.

TASK PROGRAMMING TECHNIQUES

Programming an RT/68 task at the assembly language level is not significantly different than developing any other M6800 program. There are no restrictions on the use of any instruction or memory, except those which may destroy other task's data or program areas.

There are certain requirements for programs to exist as compatible tasks in the RT/68 environment. The most important considerations are:

1. The task must maintain its stack, as well as its status data in the system status table. This implies that the task also initializes same in the proper manner.

2. There must be provisions for any necessary coordination for tasks that compete for system resources (peripheral devices, other utility tasks, memory, etc.)

3. The task must interface with the operating system and system data structures following the proper conventions.

All of the above are treated in the specific sections preceding that deal with the system features affected. The balance of this section is designed to give specific examples and rules for designing tasks according to them.

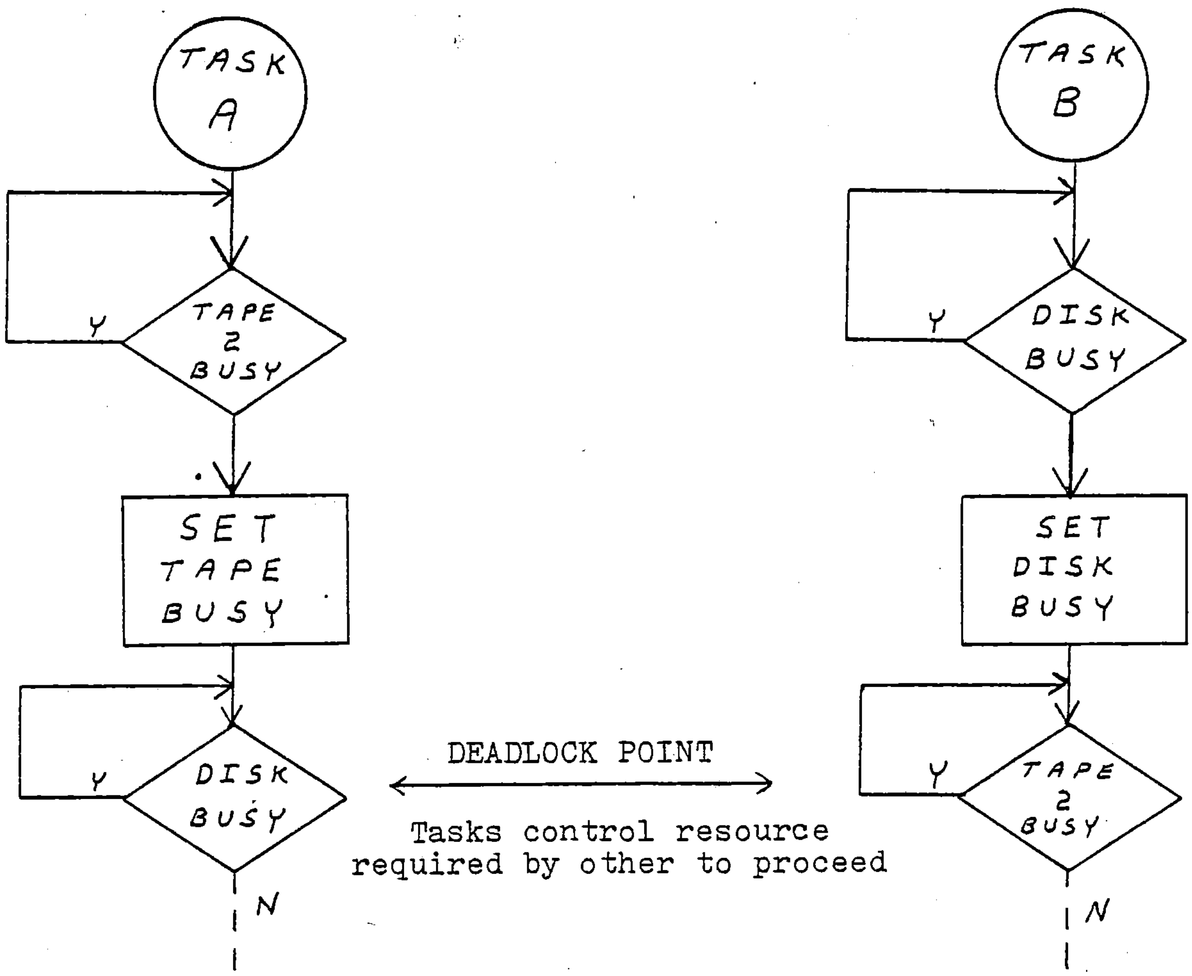
System Planning

The first step in designing applications tasks for the RT/68 environment is developing an overall system plan. The plan should address itself to the specifics of:

Memory Allocation	Function (task) prioritization
I/O coordination	Time allocation
Interrupt handling	Task intercontrol

The interdependence of tasks in a multiprogramming system, particularly when tasks divide functional parts of an overall application, is often intimate and careful consideration must be given to these effects. "Deadlock" situations can arise when tasks are competing for some resource (example - a printer) but inhibit each other from gaining control of the resource by holding another resource required by the other task.

Such a deadlock situation is illustrated in the flowchart of two competing tasks. Task A is programmed to copy a file on tape unit 2 to the disk. Task B is attempting to dump a disk file to a tape on the same drive. Both device requests from each task meet with sucess; but as soon as they are granted control of one device they are locked



into a situation where the other device cannot be used. There are two solutions to this problem. First, the devices might have been requested from a <u>resource</u> <u>allocator</u> task which when persented with the entire device requirements of each task could have scheduled each task use of the devices at non-conflicting times. An alternative method is to grant one task priority by programming the other to release control of its first device if the second is not available.

It is not necessary for the system resource being requested by competing tasks to be I/O or other hardware devices. Tasks, memory area, etc can be consided as a system resource if two or more tasks may require its use at any time.

Other factors can contribute to a more fatal type of deadlock, where task priorities have been established so that no task has a high enough priority to execute, perhaps because the system priority is higher than that of the highest runnable task. Be certain that at least one task has an active state at any given time (and presumably can activate other tasks as need be).

The idea of the master task which in effect is a super-executive for a specific system can eliminate many problems such as deadlock, device and task conflict, and perhaps allocate "utility" tasks. This task should have the highest priority in the system but not necessarily have a very long time slice. It can read the state of all tasks by simply scanning the system status table and determining if the wrong combination of tasks are in illegal or conflicting states.

A master task may also be the system interrupt handler, polling peripherals and activating tasks as necessary. In any case, best system performance can be expected if one type of interrupt is handled by one task if there are multiple devices or functions associated with a particular type of interrupt.

Total system "crashes" are invariably due to a task getting out of control and destroying the data or program areas belonging to other tasks. Large multiprogramming computers have hardware to prevent this, in a microcomputer careful programming must be substituted. Almost always out of control tasks are the result of program loops that never reach a terminating value for a loop counter. Try to avoid loops where the counter test is for equality only:

```
USUALLY:   LOOP  CLR Ø,X     BETTER:  LOOP  CLR Ø,X
                 INX                        INX
                 INC B                      INC B
                 CMP B #1ØØ                 CMP B #1ØØ
                 BNE LOOP                   BCS LOOP
                  .                          .
                  .                          .
```

Filling unused areas of memory with SWI ($3F) opcodes can occasionally save a system from destruction by resulting in a breakpoint error if an out of control task somehow fetches instructions from the area. An examination of the system using the console monitor can often result in the identification and cause of the problem.

Memory maps that are graphic illustrations of memory allocation can be another important aid in system design. The more detail shown in the map, the more useful a tool it becomes.

Real-time applications where external or timed events alter the state of the system pose greater design problems. A global concept of task concurrency must be developed by the programmer. Programs and instructions within tasks are sequential in nature; the MPU executes them in a determined and semi-continuous order. The tasks as a whole in a real-time situation are dynamic in nature and the periods of actual execution are interleaved in most cases.

The best way to conceive this is the inside/outside approach. Memory values that are parts of and used only by a specific task, or local variables, can be thought to be "inside" a task. These can be considered to be "reliable" and may be considered as any other program.

Memory locations that are shared by two or more tasks are "outside" the task and subject to change without notice. These "global" variables can even change between the execution of two sequential instructions within a task. This is because tasks are interrupted and restarted by the RT/68 executive in a transparent manner, that is there is no change in the state of the program after it is interrupted that the task can detect.

However, the value of global variables can seemingly change between instructions. For example, suppose the sequence of instructions below is being executed by task X when it is interrupted by task Y.

```
        TASK X
  .
1237    LDA A SYSTMP  (ACC A = SYSTMP = $Ø4)
1239    CMP A #$Ø5
        (interrupted here by task Y)
  .     PORTION OF TASK Y
B66Ø    INC   SYSTMP  (SYSTMP = 4 + 1 = 5)
  .     (control returned to task X)
        TASK X
123A    BEQ   LABL5   (no branch, ACC A did not = 5)
123C    .
  .     .
```

The question is, should task X have branched when the instruction at address 123A was executed? At the instant the instruction was executed the branch condition nominally was true (SYSTMP did equal 5). The condition code register was not disturbed during the time task X was inactive and when it resumed the Z bit was set.

There is no answer to the question but there are techniques to insure that tasks are not interrupted at critical momemts.

Defining the Task Stack

As discussed previously, each task must continuously maintain an individual stack area. This is generally defined in the part of the assembly source program where other data storage locations are established. The main difference in the way a task stack is set up as opposed to typical M6800 programs is the program is not required to load the stack pointer register with an initial value.

Instead, the executive obtains a value from the task's TSW to load in the SP register, and executes an RTI instruction to start the task. This loads all MPU registers from the stack and starts execution. This means the starting address of the task must be at the RAM address that corresponds to the program counter "image" on the stack. If desired, other registers may also be initialized in this manner.

How many bytes must be reserved for the stack? This de-depends on a number of factors. First, there must be enough to satisfy the task's requirements for subroutine return address nesting and data storage (PSH, PUL operations, etc.) as in any other M6800 program. The executive requires at least 20 bytes plus an additional 7 if any IRQ or NMI interrupts are caused by user devices. Reserving 32 bytes will almost always be adequate for most system requirements (above the task's requirements).

The following M6800 assembly language statements will properly define a stack:

```
Ø1  Ø4ØØ            ORG   $Ø4ØØ  low limit of stack
Ø2  Ø4ØØ            RMB   3Ø     save stack area
Ø3  Ø41E            FDB   START  form PC image on stack
Ø4  Ø42Ø     STKPRT EQU   *-8    define task stack pointer
```

Line 1 defines the lower limit of the stack (remember the stack builds "downward"). Line 2 reserves 3Ø bytes of storage for the stack, in this example addresses Ø4ØØ to Ø41D. Line 3 serves to form a 2 byte value on the top of the stack that is the initial value for the program counter (the starting address of the task). This example therefore creates a 32 byte stack. Line 4 gives the symbol "STKPTR" the correct value for the task's initial stack pointer to be used in the definition of the Task Status Table. The asterisk refers to the present value of the assembler's location counter and is equal at line 4 to the next byte following the top of the stack. This will result in a value of Ø418 in this example.

Notice that if the task's stack pointer equals this value and an RTI occurs (as does when the task is started) the MPU registers will be loaded from addresses Ø419 to Ø41F. If line 2 were changed to reserve 25 bytes instead, FCB and FDB assembler directives could be inserted afterwards to initialize the CC, ACC B, ACC A, and XR registers respectively with some values.

If more than one task is being generated by the same assembly, all TSPs and TSBs may be defined in sequence at the beginning of the source program for clarity. Do not forget that status table positions that correspond to non-existent tasks must be zeroed.

Use of System Subroutines

Exclusive of the various I/O subroutines contained in the ROM there are four important subroutines that may be called from tasks.

They are all based on the subroutine FNDTSB (address E33B). FNDTSB is called with a task number in accumulator A. It will return with the TSB for the specified task in ACC B. The index register will contain the address of the TSB. This subroutine is useful for examining or altering a task's status byte. The example below will change the priority of task 7 to level 3.

```
LDA A #7        Desired task #
JSR   $E33B     Get TSB
AND B #$F8      Mask out old priority
ORA B #$Ø3      OR with new priority
STA B Ø,X       Replace TSB
```

Two other subroutines (TSKON, addr E33C and TSKOFF, addr E335) exist and are used to turn tasks on and off by setting or clearing the task's state flags. The task number is also passed in ACC A. The example below will deactivate task #13

```
LDA A #13
JSR  $E335
```

The last subroutine, CTSKOF (addr E333) is used to turn the calling task off. It obtains the correct task number from a system variable. It is used mostly before a task passes control to the executive after completing a function or to give up the remainder of a time slice. The following code should be used to accomplish this:

```
INC    Ø        Set system  flag (addr Ø)
JSR    $E333
NOP
SEI             Set interrupt mask
INC    $AØØE    Set release flag
SWI             Call executive
JMP    ______
```

The JMP instruction following the software interrupt is included if the task has completed its function and may be called again later. The JMP will cause the task to re-execute if the destination is the beginning of the task. If the executive call is for some other reason (such as to give up unneeded remaining time in the slice, etc.) the task instructions would simply continue after the SWI instruction.

Utilizing System Data Values

The executive maintains many data values that can be read and/or modified by tasks to perform special functions. The specific memory addresses can be found on page 1 of the RT/68 listing.

SYSMOD will inhibit task switching while non-zero. This is important if the task is changing shared data that must be completely processed without interference. If an interrupt occured during this time, INTREQ will be non-zero. This should be tested and the executive called if necessary.

TIMREM represents the time remaining (ticks) of the current task. It may be changed to any time from 1 to $FF or zero, which results in unrestricted time.

SYSPTY is the current system priority level. It acts as a mask, that is, no lower priority task can be executed. Care should be taken so that this value is never greater than seven, or the system will stall.

Position Independent Code (PIC)

A powerful feature of the M6800 instruction set is the ability to create programs in position independent code. PIC is a program that may be relocated simply by moving it; it is not necessary to recompile or reassemble it. PIC must use branch-type instructions in statements that transfer program control, because the branch is to a location relative to the current value of the program counter. If the transfer must be made to an address out of range of a single branch, intermediate BRA instructions may be used. Data areas are best created in PIC on the stack using the TSX instruction and the indexed addressing mode, as no absolute RAM addresses are used.

PIC may reference external fixed addresses for common non-relocatable subroutines or data. RT/68 compliments PIC because tasks may pass data and control to each other through the executive, and addresses do not need to be "fixed" if the PIC is a complete task.

PIC allows fast and easy dynamic memory allocation.

Reentrant Code

Certain instructions and addressing modes of the M6800 support a type of program called reentrant code. A reentrant program module has the property of being able to be interrupted, entered by one or more other calling programs, and resume execution without change from the point of interruption. Use of reentrant subroutines can save substantial amounts of memory by allowing tasks to share common subroutines. For example, several tasks can share a reentrant multiplication subroutine which otherwise would need to be included in every task that used it.

The simplest type of reentrant code for the M6800 consists of code that uses no storage other than the MPU registers. The following subroutine is of this type. It multiplies the binary number in accumulator A by 10, useful in converting BCD to binary.

```
MULTEN TAB          A = B  = N
       ASL A
       ASL A
       ASL A        A = N * 8
       ASL B        B = N * 2
       ABA          A = (N * 8) + (N * 2) = N * 10
       RTS
```

If this subroutine is interrupted, the register contents are saved on the stack and another task can call the subroutine and execute it. When the original task is resumed, the register contents will be restored and execution would continue as it was before interruption.

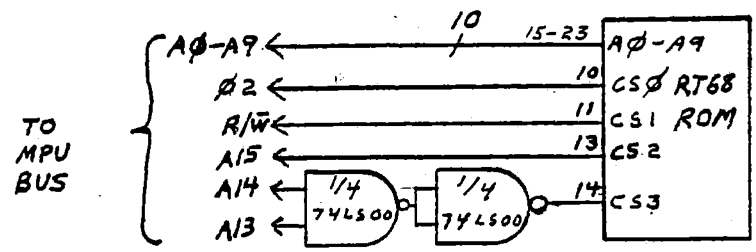
Often, however, subroutines may require more storage than is provided by the MPU registers. In this case data area on the stack can be accessed and used for intermediate storage. The PSH, PUL, INS, DES, TSX and index addressing mode allow for utilization of the task's stack as temporary storage in a reentrant manner. Each task that calls the subroutine has an individual stack (though a common stack works also) that provides the storage for its "copy" of the reentrant subroutine.

Reentrant code can be used to implement device handlers for common types of peripherals, as in the ACIA read routine below. The calling task loads X reg with the ACIA address:

```
RDACIA LDA B Ø,X    READ STATUS REG
       ASR B        SHIFT READY TO C BIT
       BCC   RDACIA BRA IF NOT READY
       LDA A 1,X    READ DATA BUFFER REG
       RTS          DONE
```

RT/68 HARDWARE CONFIGURATION

The addresses of the RT/68 ROM range from EØØØ to E3FF. However, the restart and interrupt vectors are also contained in the ROM so it must be able to respond to all addresses



MAXIMUM ADDRESS DECODE

from EØØØ to FFFF. This means that address lines A1Ø through A12 cannot be decoded. A circuit that will accomplish this is illustrated above.

If full decode is desired, a seperate PROM that has the correct interrupt vectors included can be placed at the top of memory. The vector data is found on the last page of the source listing.

Any circuit that accepts the MC6830L7-L8 Mikbug(TM) ROM will properly decode the addresses for the RT/68 ROM.

The circuits on the following pages give example configurations for several optional features. The abort switch may be connected to the control terminal PIA input CA2. The switch circuit must have a normally low, debounced function. If this feature is not used, ground the CA2 pin.

Two circuits are show that can provide a stable, precise clock signal for the RT/68 multitask executive. This is also an optional feature. Both circuits cost less than a dollar or so to construct and are extremely simple, but provide an accurate reference signal. This clock signal should be in the range of 10 to 100 Hz for optimum operation.

This signal is connected to control PIA input CA1, which also should be grounded if not used.

The level of PIA inputs PB6 and PB7 determine the number of stop bits and interface type respectively for serial I/O to the system console device. Thes must be established by jumpers to ground or +5 volts.

It is possible to use outputs from another PIA or a latch circuit connected to the control PIA to allow software control of these parameters. The circuit should guaranty a specific logic level at system initialization.

A schematic for the I/O level drivers and receivers for both current loop (teletype) and RS232 is given. This is a typical configuration and several other variations are possible. As a rule, any interface circuit that is designed for Mikbug(TM) will operate correctly.

RT/68 SYSYEM MEMORY UTILIZATION MAP

	"Images" of RT/68 ROM due to partial address decoding to allow access to interrupt vector addresses.	FFFF – E400
	RT/68 Program (ROM)	E3FF – EØØØ
	Not used - available for RAM, ROM or I/O	DFFF – AØ8Ø
	Operating system RAM: AØØØ-AØ13 = monitor temp. AØ14-AØ49 = stack AØ5Ø-AØ7F = status table (not used in single task mode)	AØ7F – AØØØ
	Not used - available for RAM, ROM or I/O	9FFF – 8ØØ8
(SWTPC port #1)	Control and/or console interface PIA	8ØØ7 – 8ØØ4
(SWTPC port #Ø)	Console Interface ACIA (if option selected)	8ØØ1 – 8ØØØ
	Not used - available for RAM, ROM or I/O	7FFF – ØØØC
	RT/68 multiprogramming exec. temp. (multi-task mode only)	ØØØB – ØØØØ

ROM INSTALLATION

CAUTION ! ! ! THE RT/68 ROM IS A MOS DEVICE AND EXTREMELY SUSCEPTABLE TO DESTRUCTION THROUGH STRAY STATIC CHARGES PRESENT ON THE HUMAN BODY. READ AND FOLLOW THE INSTRUCTIONS BELOW CAREFULLY ! ! !

1. Before handling any MOS devices, neutralize your body by connecting a metal watchband to a known good ground (cold water pipe, electrical conduit pipe, etc.) through the included 1 Megohm resistor. THE RESISTOR IS ESSENTIAL FOR YOUR SAFETY as it can prevent a shock from a defective soldering iron, etc. Don't wear nylon clothing.

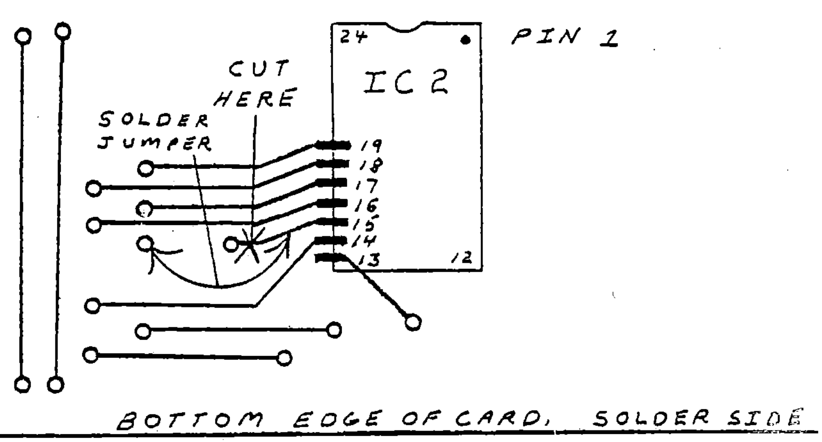
2. Remove all power from the system before removing any circuit boards or parts. Remove MOS devices before soldering any foil or wire connected to it.

3. Do not store the ROM in any nonconductive material.

For SWTPC 6800 systems:

1. After you have followed the precaution above, remove the MPU and Mikbug ROM (IC's 1 and 2 on the MP-A board).
2. Find pin 15 of the ROM (IC2) socket on the bottom of the card. A foil runs from this pin that bends slightly and then terminates about ½" from the socket in a feedthrough hole that connects to the ground foil on the top of the board. Cut the foil just before the feedthrough hole. This should be done carefully with a razor-sharp instrument. Make sure the foil is completely cut.
3. Looking at the bottom side of the card with the molex connectors on the bottom, note that there is a feedthrough hole on the same line and 3/8" to the left of the foil just cut. Jumper a wire from the cut foil (ROM side) to this hole. This modification enables address line A9.
Replace the MPU and insert the RT/68 ROM in the IC2 socket.

Location of MP-A card foil.



4. Remove the PIA (MC6820 - IC1) from the MP-C board.
5. Solder a jumper from pin 15 (PB5) to ground if PIA interface is to be used, or +5 volts for ACIA interface.
6. Solder a wire from pin 40 (CA1) to the clock generator circuit. If not used, ground the pin.
7. Solder a wire from pin 39 (CA2) to the abort switch debounce circuit. If not used, ground the pin. Tie IRQ-A to NMI.
8. Replace IC1, taking care not to bend the pins.
9. If an ACIA-type interface was selected, insert a SWTPC MP-S type interface card in I/O slot #0 and connect the terminal to it.
10. Check for poor or bridged solder connections. Correct if necessary.
11. Power up the system and the terminal. A $ should be displayed whenever the reset button is depressed.

For Motorola MEK6800D1 Evaluation Kit 1:

1. Carefully following the handling instructions on the previous page, remove all MOS devices (if socketed).
2. Move the jumper located next to the ROM (U8) from point E2 to point E1.
3. Perform steps 4 to 8 above on the control PIA (U9)
4. If ACIA interface is desired, the addresses of the ACIA (A010) and the second PIA (U10 - A008) must be reversed. The bits decoded stay the same except A3 that connects to CS1 (pin 24) of the U10 PIA which must be changed to address line A4, and CS1 (pin 10) of the ACIA which must be moved to address line A3 from A4.
5. The MEK board does not provide an internal baud rate generator for the ACIA, so one must be provided. Page 10 of the Motorola manual describes one that works well.
6. Replace all MOS devices carefully, except for the Mikbug(TM) ROM. Insert the RT/68 ROM in its place.
7. Perform steps 10 and 11 above.

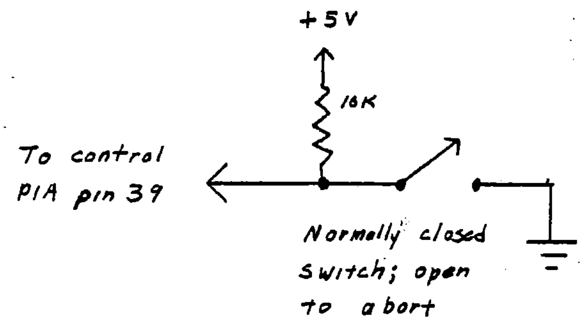
Other Mikbug-based systems:

You will need to follow the same general instructions as outlined for the two systems above. The two main concerns are enabling address line A9 for the ROM and properly configuring the control PIA.

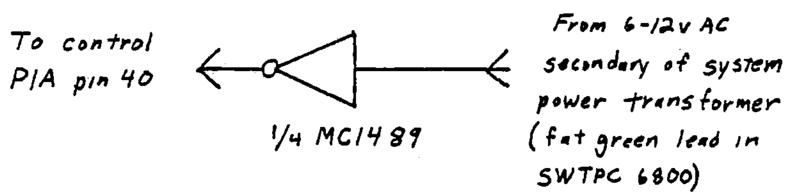
Motorola MEK6800D2 Evaluation Kit 2

The RT68 ROM is ideal for upgrading the D2 Kit. The ROM will replace JBUG directly except inputs CS1 and CS2 which must be inverted, and RS-232 interfaces added. A small circuit board and all required interface parts is available for $30.95 from Microware. Order part #DA1B. Comprehensive installation and programming information is provided with the kit.

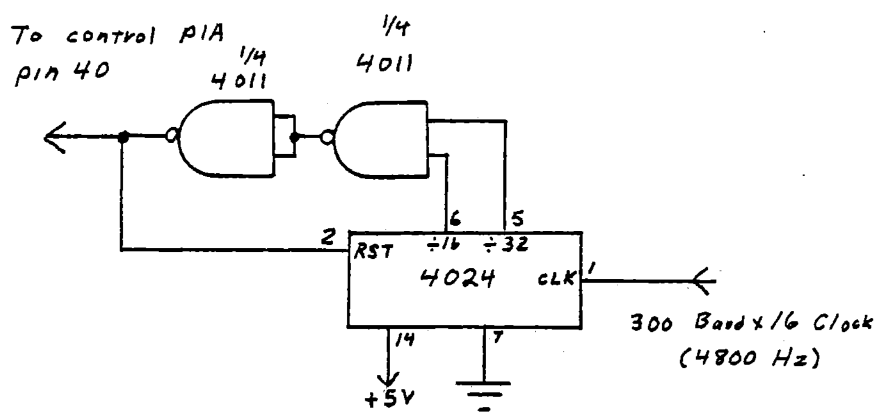
ABORT SWITCH CIRCUIT



60 Hz Real Time Clock Signal Source (AC line Ref.)



100 Hz Real Time Clock Signal Source



RT68 READ ONLY MEMORY SPECIFICATIONS

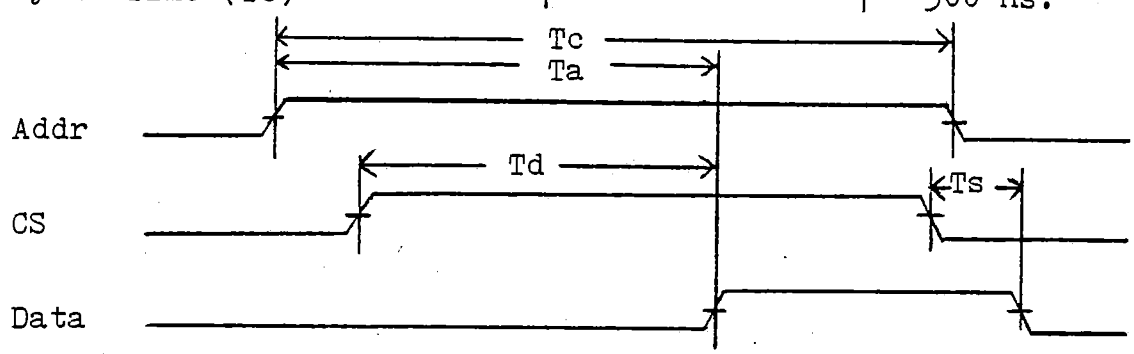
The RT68 part is a mask-programmed read-only-memory organized as 1024 8-bit bytes. It is fabricated using N-channel silicon gate process. It is completely static in operation and features tri-state data outputs.

ELECTRICAL

Power supply voltage Vcc	+5.0 volts ± 0.25v
Supply current	130 ma. max

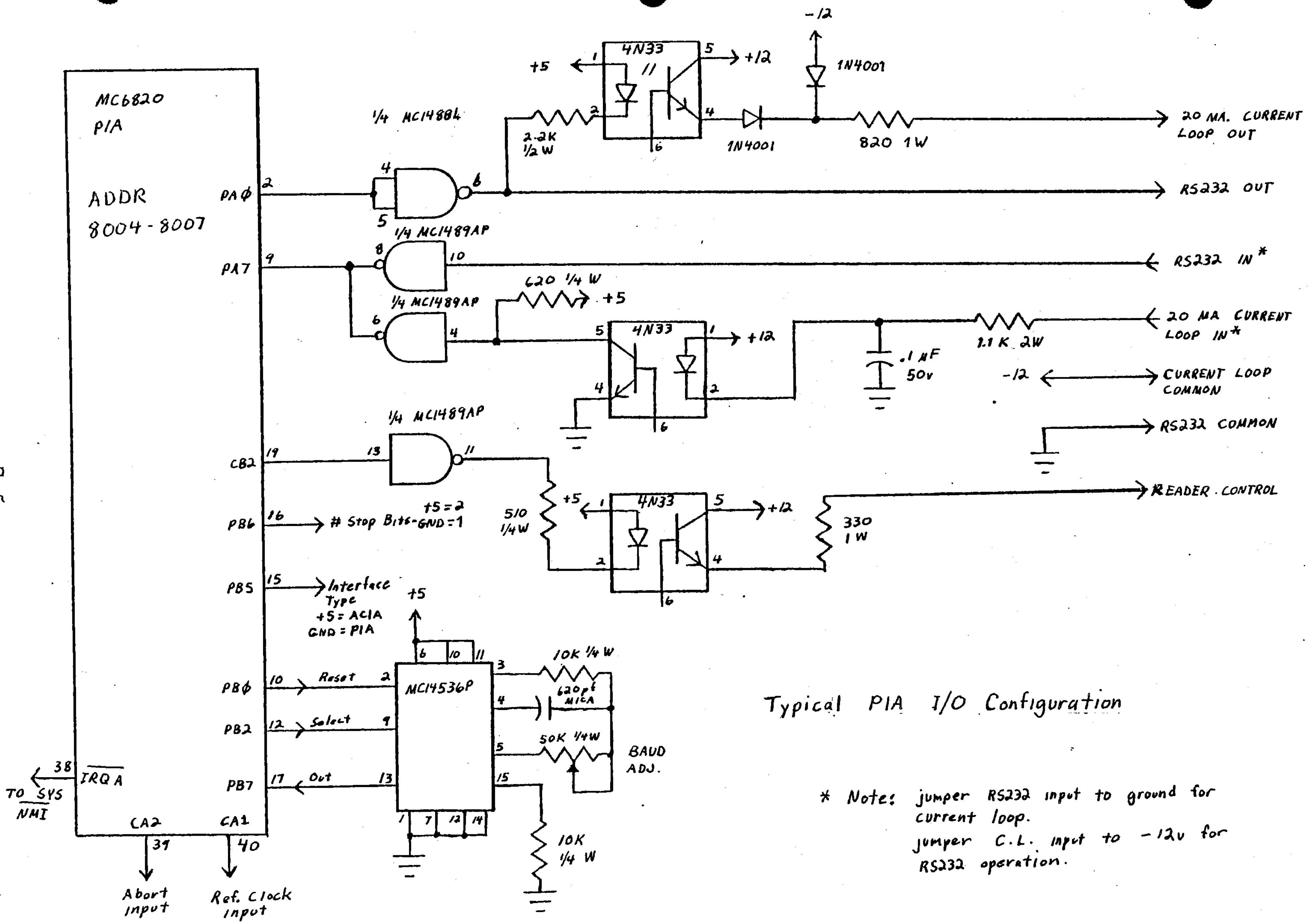
Characteristic (static)	Minimum	Maximum
Input high voltage	2.0 v	5.25 v
Input low voltage	-0.3 v	0.8 v
Input current		2.5 ua.
Output voltage high	2.4 v	
Output voltage low		0.4 v
Output leakage (deselected)		10.0 ua.
Input capacitance		7.5 pf.
Output capacitance		12.5 pf.

Characteristic (dynamic)	Minimum	Maximum
Access Time (Ta)		500 ns.
Data Delay Time (Td)		300 ns.
Data Deselect Time (Ts)	10 ns.	150 ns.
Cycle Time (Tc)		500 ns.



Temperature Range

Operating	0 - 70 C°
Storage	-65 - +150 C°



Typical PIA I/O Configuration

* Note: jumper RS232 input to ground for current loop.
jumper C.L. input to -12v for RS232 operation.

RT/68 Input/Output System

RT/68 allows use of either a standard MIKBUG-type bit serial/PIA interface or one or more ACIA type interfaces for maximum flexibility in system configuration. The selection of interface type is made at the character I/O routine level so the interface type selected is transparent to the calling software.

Most systems designed for MIKBUG will have the PIA/serial interface built in which RT/68 can use without modification. However, the ACIA interface may prove more versatile in many applications particularly when interrupt driven or high baud rate serial I/O is desired. ACIAs may also be invaluable when device of different baud rates must be interfaced, such as a 110 baud teletype and and a 1200 baud audio cassette interface.

The PIA located at address $8ØØ4 which is used in the MIKBUG interface is used by RT/68 for several functions other than I/O, so it must be present even if the interface is not used. Besides providing interrupt inputs for the real time clock and abort functions, two PIA inputs are sampled by RT/68 to determine interface options. These may be either jumper defined permanently or may be connected to other PIA outputs so the options may be selected under program control.

The table below shows the options available according to the state of the PIA inputs PB5 and PB6.

PB5	PB6	Function, interface type
Ø	Ø	I/O using PIA, transmit 1 stop bits
Ø	1	I/O using PIA, transmit 2 stop bits
1	X	I/O using ACIA, transmit 1 stop bit*

* Applies to first ACIA at $8ØØØ, may be changed by software

ACIA I/O Operation

If the ACIA option is selected, primary system I/O will be performed by an ACIA located at address $8ØØØ. The ACIA command and status register is initialized at restart to use one stop bit. The initialization is performed regardless of whether or not the ACIA is selected by PB5.

The restart routine establishes a RAM location (address $AØ12) called IOVECT as a pointer to the ACIA address which is used by the ACIA I/O subroutines to locate the ACIA. This vector may be changed either by a user program or by means of the memory examine/change function to a different address for selection of multiple ACIA interfaces.

If the ACIA vector is changed two important factors must be considered: the system only initializes the ACIA at address $8ØØØ, and the operating system will expect to do business through the new ACIA.

This requires that the address of the ACIA interfaced to the operator's terminal be restored after an operation using an ACIA connected to an audio cassette interface for example. Two short routines that accomplish this are illustrated in the appendix.

Software Selection of Interface Type

In some applications it may be desirable to have software select between an ACIA and MIKBUG/PIA interface. The PB5 input of the control PIA may be connected to an output to determine the interface type. If the selection is to be made between these interfaces, the initial state of the output must be known to the terminal may be connected to the proper interface at restart. The initialization of control registers was designed to allow either:

To initialize with terminal enabled to the PIA connect PB5 to CB2; this will automatically switch to the ACIA for tape loading.

To initialize with terminal enabled to the ACIA connect PB5 to CB2 through an inverter, this will automatically switch to the PIA interface for tape loading.

Additionally, external decoding of the serial data stream can be used to control the state of PB5 and therefore the interface type.

Parallel Data Output

It is possible to connect a PIA interface to the M6800 bus so it "looks like" an ACIA, which is useful for interfacing line printers, etc. to the operating system or software calling the RT/68 I/O routines. This is accomplished by reversing the RSØ and RS1 inputs of the PIA which are connected to the two lowest address lines. The result is having the two data registers appear to be contiguous and the two control registers contiguous in the memory address space of the PIA. The control registers of the PIAs are set up as usual, with the A data register set up as output and the B data register set up as inputs. The B register becomes the "ACIA status" register and the A data register is the "ACIA data" register.

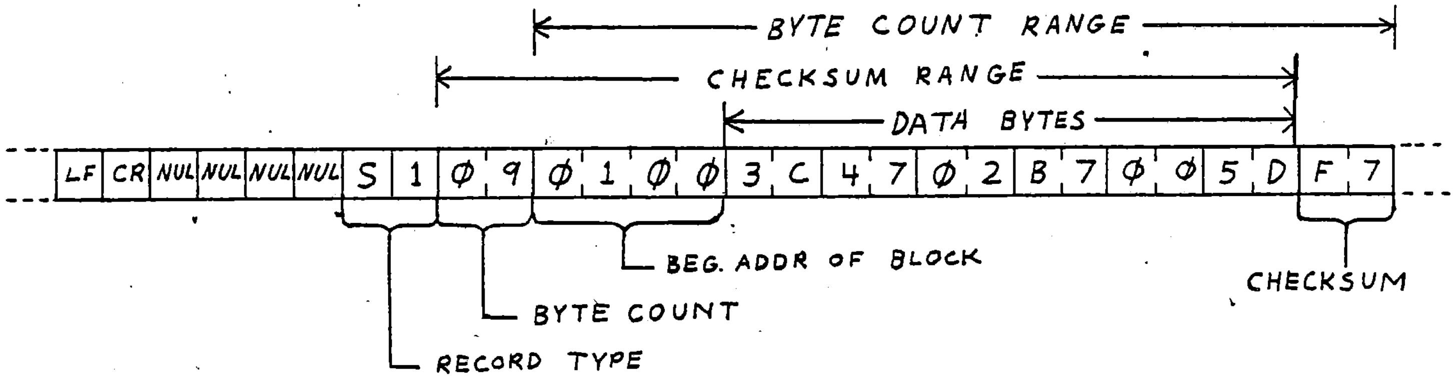
A comparative examination of the ACIA and PIA data sheets will provide information on simulating the handshaking and status signals required.

Binary Input/Output

The input character subroutine that is entered at MIKBUG entry points will strip the high order (parity) bit from the received byte as well as removing rubout ($7F) characters. However, the basic input byte routine at address $E359 will input all 8 bits without modification, which is invaluable for byte-oriented (binary) input such as binary loaders.

The output character routine will transmit the contents of accumulator A without change.

TAPE FORMAT DATA



The tape format detailed above is the same as used by almost all M6800 systems for storage of binary object data. The tape consists of strings of ASCII characters organized into records. a group of two or more records is considered to be a file.

There are three types of record used in the Motorola format: header, data, and end-of-file (EOF). The header record contains the name of the file and precedes the other records. It is identified by the SØ record type. It is not needed to load files and is neither recognized or generated by either Mikbug(TM) or RT/68. If present on a tape it will be ignored.

The data type record is illustrated above. All information on the record is represented as hex data. The beginning of the record contains the record type (S1), a byte count which covers all bytes that follow, and a beginning address of the data block. Data bytes follow which represent the object data to be stored in memory beginning at the block address and stored in sequential memory locations that follow. At the end of the record is a checksum, which is the one's compliment of the summation (mod 256) of all data bytes in the record, plus the byte count and block address. This value is checked as data is loaded and can usually detect errors that may have garbled data in the write or read process.

The end-of-file terminates a data file and consists of the characters "S9". The EOF will terminate a tape load function. The RT/68 program does not generate this type of record because it is often desirable to make files of data at non-contiguous addresses. Most systems have a provision for generating and EOF by appending an S9 in an off-line mode.

Characters between records are ignored and may include carriage return, line feed, null, control or other codes as required for device operation or readability. The RT/68 generate tape routine inserts a CR, LF and four nulls between records.

INTERFACING TO RT/68 SUBROUTINES

There are about 30 useful subroutines in the RT/68 ROM that may be called freely from user programs. The majority of these involve character, byte, and block oriented I/O to the system console device. These range from single character operations to the complex memory load and dump to/from tape.

The subroutines used in connection with the multi-task operating system are discussed in detail in the text of this manual.

All subroutines are shown in the program listing that follows. A general convention is that parameters are passed to/from the subroutine via the MPU registers, predominantly the XR and ACC A. Exceptions to this rule are the load, write tape, and dump subroutines that have parameters in BEGADR (addr. AØØ2-AØØ3) and ENDADR (addr. AØØ4-AØØ5). Subroutines that check for error conditions will place a unique error code in the loaction ERRFLG (addr. AØØF) that is the ASCII code for the error number.

Subroutines that perform I/O were written for use in the console monitor mode and as such do not have any provision for coordination in a multiprogramming environment. If one task uses the I/O device exclusively, no coordination is required. Otherwise, a device status flag should be established for each peripheral to indicate its availabity to requesting tasks.

Also, if the PIA interface is used, note that to insure correct timing, the I/O subroutine may not be interrupted for any significant amount of time. You may wish to consider other time-dependent characteristics of your peripheral devices when assigning task time limits and priorities.

RT/68 SYSTEM ENTRY POINTS

There are several points a program or task may jump to to enter various system modes or functions.

Console/system cold start	E147
Console monitor soft start/reentry	E16A or EØE3
Console monitor error entry	E1E8
RT exec cold start	E2ØC
RT exec soft start	E2F3

```
                        NAM    RT68-V2
                        OPT    O
                  *
                  *        ***************
                  *        *             *
                  *        *  R T / 6 8  *
                  *        *     M X     *
                  *        *             *
                  *        ***************
                  *
                  *  RT/68MX REAL TIME OPERATING SYSTEM
                  *  (REVISED VERSION OF RT/68MR)
                  *
                  *  COPYRIGHT (C) 1976,1977
                  *  THE MICROWARE SYSTEMS CORPORATION
                  *
                  *  RT/68 LISTING AND OBJECT MAY NOT BE
                  *  REPRODUCED IN ANY FORM WITHOUT
                  *  EXPRESS WRITTEN PERMISSION.

                  * MEMORY DEFINITIONS

                  * RT/68 EXECUTIVE USES 12 BYTES OF RAM
                  * BEGINNING AT 0. THESE ARE NOT NEEDED
                  * IN SINGLE TASK MODE AND MAY BE
                  * USED FOR ANY OTHER PURPOSE.
0000                     ORG    0
0000 0001         SYSMOD RMB    1          RT MODE 0=USER 1=EXEC
0001 0001         CURTSK RMB    1          TASK CURRENTLY ACTIVE
0002 0001         TIMREM RMB    1          TASK TIME REMAINING
0003 0002         TSKTMR RMB    2          TIMED TASK COUNTER
0005 0002         CLOCK  RMB    2          RT CLOCK COUNTER
0007 0001         INTREQ RMB    1          INTERRUPT REQUEST FLAG
0008 0001         TSKTMP RMB    1          RT EXEC TEMP VAL
0009 0001         PTYTMP RMB    1          RT EXEC TEMP VAL
000A 0001         TIMTSK RMB    1          TIMED TASK INTR STATUS
000B 0001         SYSPTY RMB    1          SYS PRIORITY LEVEL

A000                     ORG    $A000
A000 0002         IRQTSK RMB    2          IRQ TASK/VECTOR
A002 0002         BEGADR RMB    2
A004 0002         ENDADR RMB    2
A006 0002         NMITSK RMB    2          NMI TASK/VECTOR
A008 0002         SPTMP  RMB    2          SP TMP VAL
A00A 0001         RTMOD  RMB    1          RT MODE FLAG
A00B 0001         BKPOP  RMB    1          BKPT OPCODE/FLAG
A00C 0002         BKPADR RMB    2          BKPT ADDRESS
A00E 0001         RELFLG RMB    1          SWI FLAG
A00F 0001         ERRFLG RMB    1          ERROR FLAG/CODE
A010 0002         XTMP   RMB    2
A012 0002         IOVECT RMB    2          ACIA ADDRESS VECTOR
```

```
A042                   ORG    $A042
       A042     STACK  EQU    *          MONITOR STACK


                       * TASK STATUS TABLE
                       *
                       * CONSISTS OF 16 3-BYTE TASK STATUS WORDS, ONE FOR
                       * EACH POSSIBLE TASK. EACH TASK STATUS WORD CONTAINS
                       * A TASK STATUS BYTE (TSB) AND A 2-BYTE TASK STACK
                       * POINTER (TSP).
                       *
                       * THE TSB IS DEFINED AS FOLLOWS:
                       *
                       *   BIT 7    1=TASK ON  0=TASK OFF
                       *   BIT 6-3  TIME LIMIT IN TICKS (0-15)
                       *   BIT 2-0  TASK PRIORITY (0-7)
                       *
                       * THE TSP IS THE VALUE OF THE TASK'S STACK
                       * POINTER FOLLOWING THE LAST INTERRUPT, AND
                       * THEREFORE POINTS TO THE COMPLETE MPU
                       * REGISTER CONTENTS AT THE TIME THE TASK WAS
                       * INTERRUPTED. TO RESTART A TASK THE EXEC
                       * INITIALIZES THE SP FROM THE TSP AND
                       * EXECUTES AN RTI INSTRUCTION.
                       *
A050                   ORG    $A050
A050 0030       TSKTBL RMB    48


                       * DEFINE PERIPHERAL REGISTERS
8004                   ORG    $8004
8004 0001       PIADA  RMB    1
8005 0001       PIACA  RMB    1
8006 0001       PIADB  RMB    1
8007 0001       PIACB  RMB    1
8008 0001       ACIACS RMB    1
8009 0001       ACIADB RMB    1
```

```
E000                  ORG     $E000
                *  TAPE LOAD SUBROUTINE
                *
                * READS MIKBUG(TM) FORMATTED OBJECT TAPES
                * INTO RAM.
                *
                * READER DEVICE IS CONTROLLED BY EITHER ASCII
                * CONTROL CODES OR PIA READER CONTROL
                * OUTPUT.
                *
                * TWO ERRORS ARE CHECKED: CHECKSUM AND
                * NO CHANGE.
E000 C6 3C      LOAD   LDA B   #$3C      TAPE ON CONSTANTS
E002 86 11             LDA A   #$11      READER ON CODE
E004 8D 10             BSR     RDRCON    LET IT ROLL
E006 8D 70      LOAD2  BSR     INCH
E008 81 53             CMP A   #'S       LOOK FOR START OF BLOCK
E00A 26 FA             BNE     LOAD2     BRA IF NOT
E00C 8D 6A             BSR     INCH
E00E 81 39             CMP A   #'9       END OF FILE?
E010 26 09             BNE     LOAD4     BRA IF NOT
E012 C6 34      LOAD3  LDA B   #$34      TAPE OFF CONSTANTS
E014 86 13             LDA A   #$13
E016 F7 8007    RDRCON STA B   PIACB     PIA READER CTRL
E019 20 5A             BRA     OUTCH     ASCII TAPE CONTROL
E01B 81 31      LOAD4  CMP A   #'1       S1 DATA RECORD?
E01D 26 E7             BNE     LOAD2     BRA IF NOT, LOOK AGAIN
E01F 5F                CLR B             ACC B WILL GENERATE CHKSUM
E020 8D 33             BSR     BYTE      PICK UP BYTE COUNT
E022 80 02             SUB A   #2        LESS 2 FOR THE BLOCK ADDR
E024 B7 A002    LOAD5  STA A   BEGADR    SAVE IT
E027 8D 1E             BSR     BADDR     GET BLOCK START ADDR IN X

                * LOOP TO READ DATA BLOCK
E029 8D 2A      LOAD6  BSR     BYTE      GET A DATA BYTE
E02B 7A A002           DEC     BEGADR    DECR BYTE COUNT
E02E 27 09             BEQ     LOAD7     BRA IF LAST BYTE
E030 A7 00             STA A   0,X       PUT IT IN MEMORY
E032 A1 00             CMP A   0,X       BE SURE IT CHANGED
E034 26 0A             BNE     LDMERR    BRA TO ERROR
E036 08                INX               NEXT ADDR
E037 20 F0             BRA     LOAD6     NEXT BYTE

                * B ADDS CHKSM FROM TAPE TO CALCULATED CHKSUM,
                * SO BY ADDING ONE IT SHOULD ZERO
E039 5C         LOAD7  INC B
E03A 27 CA             BEQ     LOAD2     BRA IF IT DID
E03C 86 32             LDA A   #$32      TOO BAD, GET THE ERROR CODE
E03E 20 02             BRA     LODERR
E040 86 31      LDMERR LDA A   #$31      NO CHANGE ERROR CODE
E042 B7 A00F    LODERR STA A   ERRFLG
E045 20 CB             BRA     LOAD3
```

```
                      * BUILD 4 HEX CHAR VALUE (ADDRESS)
                      * RETURNS VALUE IN XR
E047 8D 0C    BADDR  BSR     BYTE      INPUT 2 LEFT CHRS
E049 B7 A004         STA A   ENDADR
E04C 8D 07           BSR     BYTE      INPUT 2 RIGHT CHRS
E04E B7 A005         STA A   ENDADR+1
E051 FE A004         LDX     ENDADR
E054 39              RTS

                      * INPUT A BYTE (2 HEX CHARS)
                      * RETURNS BINARY VALUE IN ACC A

E055 37       BYTE   PSH B             INPUT 2 HEX CHAR
E056 8D 52           BSR     INHEX     LEFT HEX CHAR
E058 48              ASL A
E059 48              ASL A
E05A 48              ASL A
E05B 48              ASL A
E05C 16              TAB
E05D 8D 4B           BSR     INHEX     RIGHT HEX CHAR
E05F 1B              ABA
E060 33              PUL B
E061 36              PSH A
E062 1B              ABA
E063 16              TAB
E064 32              PUL A
E065 39              RTS
E066 01              NOP

                      * HEX OUTPUT AUX. SUBROUTINES
E067 44       OUTHL  LSR A
E068 44              LSR A
E069 44              LSR A
E06A 44              LSR A
E06B 84 0F    OUTHR  AND A   #$F
E06D 8B 30           ADD A   #$30
E06F 81 39           CMP A   #$39
E071 23 02           BLS     OUTCH
E073 8B 07           ADD A   #$7

E075 7E E3A6 OUTCH   JMP     OUT1CH
E078 7E E350 INCH    JMP     IN1CHR


                      * PRINT DATA STRING POINTED TO BY XR
                      * AND ENDING WITH ASCII EOT ($04)
E07B 8D F8    PDATA2 BSR     OUTCH
E07D 08              INX
E07E A6 00    PDATA1 LDA A   0,X       SUBR ENTRY POINT
E080 81 04           CMP A   #4
E082 26 F7           BNE     PDATA2
E084 39              RTS
```

```
                  *
                  * CONSOLE MEMORY DUMP SUBROUTINE
                  *
                  * PRINTS BEG ADDR AND 16 BYTES OF DATA ON EACH LINE
                  * STARTING ADDR IN BEGADR
                  * ENDING ADDR IN ENDADR
                  *
E085 BD E141 DUMP    JSR     CRLF      CR AND LF
E088 CE A002         LDX     #BEGADR
E08B 8D 3B           BSR     OUT4HS    PRINT BEGINNING ADDR
E08D C6 10           LDA B   #16       BYTE COUNT FOR LINE
E08F FE A002         LDX     BEGADR    GET BEG. ADDR
E092 8D 36   DUMP1   BSR     OUT2HS    PRINT A BYTE
E094 09              DEX
E095 BC A004         CPX     ENDADR    DONE YET?
E098 26 01           BNE     DUMP2     BRA IF NOT
E09A 39              RTS
E09B 08      DUMP2   INX               ADV X TO NEXT BYTE
E09C 5A              DEC B             DEC LINE BYTE COUNT
E09D 26 F3           BNE     DUMP1     BRA IF LINE NOT DONE
E09F FF A002         STX     BEGADR    UPDATE BEGADR TO CURRENT ADDR
E0A2 20 E1           BRA     DUMP

E0A4 86 33   HBAD    LDA A   #$33      INHEX ERROR RETURN
E0A6 B7 A00F         STA A   ERRFLG
E0A9 39              RTS

                  * INPUT HEX CHARACTER. IF CHAR IS NOT
                  * HEX, THE ERROR FLAG IS SET TO THE
                  * ERROR CODE ($33 - ASCII 1)
E0AA 8D CC   INHEX   BSR     INCH      INPUT ONE HEX CHAR
E0AC 80 30           SUB A   #$30
E0AE 25 F4           BCS     HBAD
E0B0 81 09           CMP A   #9
E0B2 23 08           BLS     IHRET
E0B4 80 07           SUB A   #7
E0B6 25 EC           BCS     HBAD
E0B8 81 0F           CMP A   #15
E0BA 22 E8           BHI     HBAD
E0BC 39      IHRET   RTS

E0BD 01              NOP
E0BE 01              NOP

                  * OUTPUT BYTE (TWO HEX CHARS) POINTED
                  * TO BY XR
E0BF A6 00   OUT2H   LDA A   0,X
E0C1 8D A4           BSR     OUTHL
E0C3 A6 00           LDA A   0,X
E0C5 08              INX
E0C6 20 A3           BRA     OUTHR
```

```
                     * OUTPUT 4 HEX CHARS AND SPACE
E0C8 8D F5    OUT4HS BSR    OUT2H

                     * OUTPUT 2 HEX CHARS AND SPACE
E0CA 8D F3    OUT2HS BSR    OUT2H

                     * OUTPUT A SPACE
E0CC 86 20    OUTS   LDA A  #$20
E0CE 20 A5    BOUT   BRA    OUTCH


                     * PRINT CONTENTS OF STACK
                     * FORMAT:
                     * SP CC B A XR PC
E0D0 8D 6F    PRSTAK BSR    CRLF      PRINT CR+LF
E0D2 CE A008         LDX    #SPTMP
E0D5 8D F1           BSR    OUT4HS    PRINT SP
E0D7 FE A008         LDX    SPTMP
E0DA 08       PRTSK  INX              ENTRY TO PRINT TASK STACK
E0DB 8D ED           BSR    OUT2HS    PRINT CC
E0DD 8D EB           BSR    OUT2HS    PRINT ACC B
E0DF 8D E9           BSR    OUT2HS    PRINT ACC A
E0E1 20 03           BRA    PRSTK2    BRA OVER PATCH
E0E3 7E E16A  CONTRL JMP    CONENT    PATCH FOR ADDR. ALIGNMENT
E0E6 8D E0    PRSTK2 BSR    OUT4HS    PRINT XR
E0E8 20 DE           BRA    OUT4HS    PRINT PC +RTS
```

```
                   *   WRITE OBJECT TAPE SUBROUTINE
                   *
                   * GENERATES MIKBUG(TM) FORMATTED TAPES
                   * ON SYSTEM TAPE DEVICE (PAPER TAPE,
                   * AUDIO CASSETTE, ETC.)
                   *
                   * BEGINNING ADDRESS OF DATA IN 'BEGADR'
                   * ENDING ADDRESS IN 'ENDADR'
                   *
                   * ENTRY POINT IS 'TAPOUT' - E0EE

                   * AUX. SUBR. TO OUTPUT BYTE + UPDATE
                   * CHECKSUM.
E0EA EB 00         TAPAUX ADD B  0,X
E0EC 20 D1                BRA    OUT2H

E0EE 86 12         TAPOUT LDA A  #$12      TAPE ON CODE
E0F0 8D 83                BSR    OUTCH
                   * OUTPUT 60 NULL CHARS TO GENERATE
                   * EITHER A 6" LEADER FOR PAPER TAPE
                   * OR A 2 SECOND TAPE SPEEDUP DELAY
                   * (AT 30 CPS) FOR AUDIO CASSETTES
E0F2 C6 3C                LDA B  #60       LEADER/DELAY NULL COUNT
E0F4 4F            OUTLDR CLR A
E0F5 8D 12                BSR    JOUT1C
E0F7 5A                   DEC B
E0F8 26 FA                BNE    OUTLDR

                   * SUBTRACT BEGADR FROM ENDADR
E0FA CE A002       TOUT1  LDX    #BEGADR
E0FD A6 02                LDA A  2,X
E0FF E6 03                LDA B  3,X
E101 E0 01                SUB B  1,X
E103 A2 00                SBC A  0,X
E105 24 05                BCC    TOUT2     BRA IF BEG < END TO DUMP
E107 86 14                LDA A  #$14      PUNCH OFF CODE
E109 7E E075       JOUT1C JMP    OUTCH

                   * CALCULATE BYTE COUNT
E10C 26 04         TOUT2  BNE    TOUT3     BRA IF HIGH BYTE NONZERO
E10E C1 10                CMP B  #16
E110 25 02                BCS    TOUT4     BRA IF BLOCK < 16 BYTES
E112 C6 0F         TOUT3  LDA B  #15       SET FULL BLOCK
E114 CB 04         TOUT4  ADD B  #4        ADD FOR B.C + BEG ADDR.

                   * OUTPUT BLOCK HEADER
E116 8D 29                BSR    CRLF      OUTPUT CR,LF+NULLS
E118 08                   INX
E119 8D 29                BSR    JPDATA    OUTPUT S,1
E11B 37                   PSH B            SAVE BYTE CNT
E11C 30                   TSX
E11D 5F                   CLR B            CLEAR CHECKSUM
E11E 8D CA                BSR    TAPAUX    PRINT BYTE CNT.
E120 32                   PUL A
```

```
00328 E121 80 03           SUB A  #3        UPDATE BYTE COUNT
00329 E123 36              PSH A
00330 E124 CE A002         LDX    #BEGADR
00331 E127 8D C1           BSR    TAPAUX    OUTPUT BEG. ADDR.
00332 E129 8D BF           BSR    TAPAUX

00334                * LOOP TO OUTPUT ONE BLOCK OF DATA
00335 E12B FE A002         LDX    BEGADR    XR POINTS TO CURRENT DATA BYT
00336 E12E 8D BA    TOUT5  BSR    TAPAUX    OUTPUT BYTE
00337 E130 32              PUL A
00338 E131 4A              DEC A            DECR. BYTE COUNT
00339 E132 36              PSH A
00340 E133 26 F9           BNE    TOUT5     BRA IF BYTE COUNT NOT ZERO

00342 E135 31              INS
00343 E136 FF A002         STX    BEGADR    SAVE CURRENT ADDR
00344 E139 53              COM B            COMPL. CHKSUM
00345 E13A 37              PSH B
00346 E13B 30              TSX
00347 E13C 8D AC           BSR    TAPAUX    OUTPUT CHKSUM
00348 E13E 31              INS
00349 E13F 20 B9           BRA    TOUT1

00351                * SUBROUTINE TO PRINT CR + LF
00352 E141 CE E3D0 CRLF   LDX    #CRLSTR
00353 E144 7E E07E JPDATA JMP    PDATA1
```

C000
C147

```
                          * RT/68 CONSOLE MONITOR PROGRAM
                          *
                          * ACCEPTS COMMANDS FORM THE CONSOLE DEVICE
                          * AND EXECUTES THE APPROPRIATE  FUNCTION.


                          * ENTRY POINT FOR RESTART
E147 8E A042 INIT   LDS    #STACK   INITIALIZE PERIPHERALS
E14A BF A008        STS    SPTMP
E14D CE 8000        LDX    #$8000
E150 FF A012        STX    IOVECT   INIT ACIA VECTOR
                    * INITIALIZE CONTROL PIA
E153 6C 04          INC    4,X
E155 C6 16          LDA B  #$16
E157 E7 05          STA B  5,X
E159 6C 04          INC    4,X
E15B 86 05          LDA A  #$05
E15D A7 06          STA A  6,X
E15F 86 34          LDA A  #$34
E161 A7 07          STA A  7,X
                    *  INITIALIZE ACIA AT $8000
E163 86 03          LDA A  #3
E165 A7 00          STA A  0,X
E167 5A             DEC B
E168 E7 00          STA B  0,X      SET ACIA CSR
E16A 7F A00B CONENT CLR    BKPOP    CONSOLE ROUTINE ENTRY POINT
E16D 7F A00A        CLR    RTMOD
E170 7F A00F CONSOL CLR    ERRFLG
E173 8E A042        LDS    #STACK   INIT SP
E176 8D C9          BSR    CRLF
E178 86 24          LDA A  #'$      PRINT PROMPT
E17A 8D 55          BSR    OUTEEE
E17C 8D 2E          BSR    INEEE    INPUT COMMAND CODE

                    * COMMAND TABLE LOOKUP/EXECUTE LOOP
                    *SEARCHES FOR COMMAND CODE ON TABLE TO OBTAIN
                    *FUNCTION SUBROUTINE ADDRESS.
E17E CE E3D6        LDX    #CMDTBL-3  INIT X TO BEGINING OF TABL
E181 08      CMSRCH INX             ADV TO NEXT ENTRY
E182 08             INX
E183 08             INX
E184 E6 00          LDA B  0,X      GET CODE FROM TABLE
E186 27 0B          BEQ    CMDERR   IF ZERO, END OF TABLE
E188 11             CBA             COMMAND CODE MATCH COMPARE
E189 26 F6          BNE    CMSRCH   BACK TO ADV IF NOT
E18B EE 01          LDX    1,X      GET CMND SUBR ADDR FROM TABLE
E18D AD 00          JSR    0,X      DO IT
E18F 8D 57   TSTENT BSR    ERTEST   TEST FOR ERROR
E191 20 DD   GOCON  BRA    CONSOL   GET ANOTHER CMND

E193 C6 36   CMDERR LDA B  #'6      ILLEGAL COMMAND CODE
E195 20 56          BRA    ERROR    GOTO ERROR ROUTINE
```

```
                      * SUBR TO SET OR REMOVE BREAKPOINTS
E197 B6 A00B SETBKP LDA A  BKPOP    GET BKPT FLAG OR OPCODE
E19A 27 0A          BEQ    SBRET    IF = 0, NO BKPT ACTIVE
E19C FE A00C        LDX    BKPADR   GET ADDR
                    * SWAP FLAG/OPCODE
E19F E6 00          LDA B  0,X
E1A1 A7 00          STA A  0,X
E1A3 F7 A00B        STA B  BKPOP
E1A6 39      SBRET  RTS

                    * "D" DUMP COMMAND ROUTINE
E1A7 8D 2B   DMPCOM BSR    GET2AD
E1A9 7E E085        JMP    DUMP

E1AC 7E E350 INEEE  JMP    IN1CHR

                    * SUBR TO PREPARE FOR USER PROGRAM
                    * EXECUTION. CALLED BY G, E & S COMMANDS
                    *
E1AF 8D E6   SETRUN BSR    SETBKP   SET BKPT IF ANY
E1B1 C6 1E          LDA B  #$1E
E1B3 B6 A00A        LDA A  RTMOD    TEST IF MULTITASK MODE
E1B6 27 04          BEQ    SETRN2   BRA IF NOT MULTI
E1B8 5C             INC B           ENABLE RT CLOCK INTR
E1B9 4F             CLR A
E1BA 97 00          STA A  SYSMOD
E1BC B6 8004 SETRN2 LDA A  PIADA
E1BF F7 8005        STA B  PIACA
E1C2 39      RETURN RTS

                    * "B" BREAKPOINT COMMAND ROUTINE.
E1C3 7F A00B BKPCOM CLR    BKPOP
E1C6 8D 11          BSR    GETADR
E1C8 FF A00C        STX    BKPADR
E1CB 86 3F          LDA A  #$3F
E1CD B7 A00B        STA A  BKPOP
E1D0 39             RTS

E1D1 7E E3A6 OUTEEE JMP    OUT1CH

                    * SUBR TO READ ONE OR TWO ADDRESS
                    * PARAMETERS. COMMA LEADS ADDRESSES,
                    * (CR) CANCELS COMMAND.
E1D4 8D 03   GET2AD BSR    GETADR   GET TWO ADDRESSES
E1D6 FF A002        STX    BEGADR

E1D9 8D D1   GETADR BSR    INEEE    GET ONE ADDRESS
E1DB C6 34          LDA B  #$34
```

```
E1DD 81 0D          CMP A   #$0D
E1DF 27 8F          BEQ     CONSOL
E1E1 81 2C          CMP A   #',
E1E3 26 08          BNE     ERROR
E1E5 BD E047        JSR     BADDR

             * ERROR TEST SUBROUTINE
E1E8 F6 A00F ERTEST LDA B   ERRFLG
E1EB 27 D5          BEQ     RETURN

             * ERROR HANDLER, PRINTS MESSAGE
             * AND ERROR CODE
E1ED CE E3CA ERROR  LDX     #ERRMSG
E1F0 BD E07E        JSR     PDATA1
E1F3 17             TBA
E1F4 8D DB          BSR     OUTEEE
E1F6 20 99          BRA     GOCON

             * "E" EXECUTE SINGLE TASK COMMAND.
E1F8 8D DF   EXCOM  BSR     GETADR
E1FA 8D B3          BSR     SETRUN
E1FC FE A004        LDX     ENDADR
E1FF 6E 00          JMP     0,X

             * "G" GO TO USER PGM OR RETURN FROM
             * BREAKPOINT COMMAND ROUTINE.
E201 BE A008 GOCOM  LDS     SPTMP
E204 8D A9          BSR     SETRUN
E206 3B             RTI

             * "P" WRITE TAPE COMMAND ROUTINE
E207 8D CB   PUNCOM BSR     GET2ADR
E209 7E E0EE        JMP     TAPOUT

             * "S" COMMAND ROUTINE.
             * ACTIVATES AND INITIALIZES RT/68
             * EXECUTIVE.
E20C 7F A00E SYSCOM CLR     RELFLG
E20F 86 01          LDA A   #1
E211 B7 A00A        STA A   RTMOD
E214 CE 0009        LDX     #PTYTMP
E217 6F 00   CLOOP  CLR     0,X
E219 09             DEX
E21A 26 FB          BNE     CLOOP
E21C A7 00          STA A   0,X
E21E 8D 8F          BSR     SETRUN
E220 7E E2EA        JMP     EXEC02    JUMP TO RT EXEC ENTRY
```

```
                  * "M" MEMORY EXAMINE/CHANGE ROUTINE.
                  * AFTER BEGINNING ADDR IS ENTERED, PGM
                  * PRINTS ADDR AND DATA IN HEX:
                  *     AAAA DD
                  * A  SLASH AND NEW HEX DATA CHANGES LOACTION,
                  * A (LF) OPENS NEXT ADDR, AND (CR) CLOSES
                  * FUNCTION.
E223 8D B4        MEMCOM BSR    GETADR   GET BEG ADDR

                  * EXAMINE/CHANGE LOOP
E225 BD E141      MEM1   JSR    CRLF
E228 86 0D        MEM2   LDA A  #$0D     PRINT LF
E22A 8D A5               BSR    OUTEEE
E22C CE A004             LDX    #ENDADR
E22F BD E0C8             JSR    OUT4HS   PRINT ADDRESS
E232 FE A004             LDX    ENDADR
E235 BD E0BF             JSR    OUT2H    PRINT CONTENTS
E238 FF A004             STX    ENDADR
E23B BD E1AC             JSR    INEEE    INPUT DELIMITER
E23E 81 0A               CMP A  #$0A
E240 27 E6               BEQ    MEM2     BRA IF LF TO OPEN NEXT
E242 81 2F               CMP A  #'/
E244 27 01               BEQ    MEM3     BRA IF CHANGE
E246 39                  RTS

                  * CHANGE MEMORY LOCATION
E247 BD E055      MEM3   JSR    BYTE     READ NEW DATA
E24A 8D 9C               BSR    ERTEST
E24C 09                  DEX
E24D A7 00               STA A  0,X      STORE NEW DATA
E24F A1 00               CMP A  0,X      TEST FOR CHANGE
E251 27 D2               BEQ    MEM1     BRA IF OK TO OPEN NEXT
E253 C6 35               LDA B  #$35     ERROR CODE
E255 20 96               BRA    ERROR
```

```
                    *       REAL TIME OPERATING SYSTEM COMPONENTS
                    *
                    * CONSISTS OF:
                    *
                    *             INTERRUPT PROCESSORS
                    *             TASK EXECUTIVE
                    *             AUX. SUBROUTINES
                    *


                    * BREAKPOINT SERVICE ROUTINE
E257 30             RUNBKP TSX                 GET SP IN XR
E258 8D 1D                 BSR    ADJSTK       DECR PC ON STACK
E25A EE 05                 LDX    5,X          GET TASK PC OFF STACK
E25C BC A00C               CPX    BKPADR       COMPARE TO PRESET ADR
E25F 27 05                 BEQ    RUNBK2       BRA IF SAME
E261 C6 37                 LDA B  #$37         SET ERROR FLAG
E263 F7 A00F               STA B  ERRFLG
E266 BD E197        RUNBK2 JSR    SETBKP       REMOVE BKPT OPCODE
E269 86 16                 LDA A  #$16
E26B B7 8005               STA A  PIACA        OFF RT CLOCK + ABORT INTR
E26E BF A008               STS    SPTMP        SAVE TASK SP
E271 BD E0D0               JSR    PRSTAK       DUMP STACK
E274 7E E18F               JMP    TSTENT       ENTER CONSOLE MONITOR

                    * SUBR TO DECREMENT PC ON STACK
E277 6D 00          ADJSTK TST    0,X
E279 26 02                 BNE    ADSTK2
E27B 6A 05                 DEC    5,X
E27D 6A 06          ADSTK2 DEC    6,X
E27F 39                    RTS

                    * SWI ENTRY POINT. DETERMINES WHETHER
                    * BREAKPOINT OR PGM RELEASE FUNCTION

     E280           SINT   EQU    *            SWI VECTOR DESTINATION
E280 B6 A00E               LDA A  RELFLG       GET PGM RELEASE FLAG
E283 27 D2                 BEQ    RUNBKP       EXEC BKPT IF NOT SET
E285 7F A00E               CLR    RELFLG       RESET FLAG
E288 5F                    CLR B
E289 20 56                 BRA    EXEC09       GO TO EXEC TO SWAP
```

```
                   * IRQ INTERRUPT ENTRY POINT
                   * INCLUDES LOGIC TO DETECT AND CORRECT
                   * INTERRUPT ERROR OCCURING WHEN SWI +
                   * NMI OCCUR SIMULTANEOUSLY. (SEE P. A-10
                   * OF M6800 APPLICATIONS MANUAL)

      E28B         IRQ     EQU     *         IRQ VECTOR DESTINATION
E28B B6 A00E               LDA A   RELFLG    GET SWI FLAG
E28E 26 05                 BNE     INTBAD    BRA TO ERR CORR. IF SET
E290 CE A000               LDX     #IRQTSK   PTR TO IRQ VECTOR/STATUS
E293 20 36                 BRA     RUNINT    GOTO INTR SERVICE


                   * CORRECT SWI-IRQ COINC. ERROR
E295 30            INTBAD TSX
E296 8D DF                 BSR     ADJSTK    DECR TASK PC ON STACK


                   * NMI INTERRUPT HANDLER.
                   *
                   * TEST CONTROL PIA FOR ABORT OR CLOCK
                   * INTERRUPT AND PROCESS SAME
                   * IF NOT, EXECUTES USER INTERRUPT
      E298         NMI     EQU     *         NMI VECTOR DEST.
E298 B6 8005               LDA A   PIACA     GET PIA STATUS REG
E29B F6 8004               LDA B   PIADA     CLEAR PIA INTR FLGS
E29E 48                    ASL A
E29F 2B C5                 BMI     RUNBK2    BRA IF ABORT INTR
E2A1 24 25                 BCC     NMI5      BRA IF USER INTR
                   * HERE IF CLOCK INTR ONLY
E2A3 B6 A00A               LDA A   RTMOD     TEST SYS MOD
E2A6 27 20                 BEQ     NMI5      BRA TO USER INTR IF NOT
E2A8 DE 05                 LDX     CLOCK     INCR RT CLOCK COUNTER
E2AA 08                    INX
E2AB DF 05                 STX     CLOCK
                   * UPDATE TIMED TASK STATUS
E2AD DE 03                 LDX     TSKTMR    GET TIMED TASK COUNTER
E2AF 27 09                 BEQ     NMI3      BRA IF NOT ACTIVE
E2B1 09                    DEX               DECR COUNTER
E2B2 DF 03                 STX     TSKTMR
E2B4 26 04                 BNE     NMI3      BRA IF NOT EXPIRED
E2B6 96 0A                 LDA A   TIMTSK    GET TIMED TASK STAT BYTE
E2B8 20 1C                 BRA     RNINT3    RUN AS INTERRUPT
                   * UPDATE REMAINING TIME OF CURRENT TASK
E2BA 96 02         NMI3    LDA A   TIMREM    GET TIME LEFT
E2BC 27 05                 BEQ     NMI4      BRA IF UNLIMITED
E2BE 4A                    DEC A
E2BF 97 02                 STA A   TIMREM
E2C1 27 1A                 BEQ     EXEC01    BRA TO EXEC IF TIME UP
E2C3 96 07         NMI4    LDA A   INTREQ    TEST FOR PENDING INTR.
E2C5 26 16                 BNE     EXEC01
E2C7 3B                    RTI
E2C8 CE A006       NMI5    LDX     #NMITSK   GET NMI STAT PTR
```

```
                   * GENERAL INTERRUPT PRESERVICE
                   * SELECTS PROPER MODE, AND EITHER
                   * RUNS OR SCHEDULES INTERRUPT SERVICE
                   * TASK ACCORDING TO THE APPROPRIATE
                   * STATUS BYTE.
E2CB B6 A00A RUNINT LDA A  RTMOD
E2CE 26 04          BNE    RNINT2   BRA IF MULTITASK MODE
E2D0 EE 00          LDX    0,X      GET VECTOR
E2D2 6E 00          JMP    0,X      EXECUTE SAME AS MIKBUG

E2D4 A6 00   RNINT2 LDA A  0,X      GET INTR STATUS BYTE
E2D6 8D 54   RNINT3 BSR    TSKON    TURN SERV. TASK ON
E2D8 4D             TST A           CHK IMMED OR DEFERRED
E2D9 2A 50          BPL    INTRET   BRA IF DEFERRED
E2DB 97 07          STA A  INTREQ   SET INTR REQ. FLAG
                   * FALL THROUGH TO EXECUTIVE
```

```
                  * RT/68 MULTI-TASK EXECUTIVE PROGRAM
                  *
                  * SAVES CURRENT TASK STATUS IN TASK STATUS
                  * TABLE, THEN SEARCHES THE TABLE FOR THE
                  * HIGHEST PRIORITY RUNNABLE TASK AND STARTS
                  * IT. IF THERE IS MORE THAN ONE RUNNABLE TASK
                  * AT THE HIGHEST LEVEL, THE
                  * EXECUTIVE WILL RUN THEM ROUND-ROBIN.

                  * TEST MODE TO PREVENT MULTIPLE
                  * EXECUTION OF EXEC BY INTERRUPTS
E2DD D6 00        EXEC01 LDA B  SYSMOD
E2DF 26 4A               BNE    INTRET   BRA IF EXEC ALREADY ACTIVE
E2E1 5C           EXEC09 INC B           SET EXEC MODE
E2E2 D7 00               STA B  SYSMOD
                  * SAVE CURRENT TASK SP ON TABLE
E2E4 96 01               LDA A  CURTSK   GET CURRENT TASK #
E2E6 8D 53               BSR    FNDTSB   FIND ADDR OF TSB
E2E8 AF 01               STS    1,X      SAVE SP

                  * INITIALIZE EXEC TEMP VALUES
                  * PTYTMP = HIGHEST PRIORITY FOUND
                  * TSKTMP = TASK # FOR ABOVE
E2EA 4F           EXEC02 CLR A
E2EB 97 07               STA A  INTREQ
E2ED 97 09               STA A. PTYTMP
E2EF 97 08               STA A  TSKTMP
E2F1 96 01               LDA A  CURTSK

                  * LOOP TO SEACH THROUGH TABLE FOR
                  * HIGHEST RUNNABLE TASK
                  * STARTS WITH CURRENT TASK AND COUNTS
                  * DOWN SO LAST TASK TESTED IS THE
                  * CURRENT TASK # -1. THIS ALLOWS TASKS
                  * AT SAME PRIORITY LEVEL TO EXECUTE
                  * ROUND-ROBIN.
E2F3 8D 46        EXEC03 BSR    FNDTSB   FIND TSB
E2F5 2A 0D               BPL    EXEC04   BRA IF TASK OFF
E2F7 C4 07               AND B  #$07     MASK PRIORITY
E2F9 D1 09               CMP B  PTYTMP   COMP. TO HIGHEST SO FAR
E2FB 25 07               BCS    EXEC04   BRA IF LOWER
E2FD D7 09               STA B  PTYTMP   MAKE IT LATEST
E2FF 16                  TAB             CHANGE SET TASK#
E300 CA 80               ORA B  #$80     SET FOUND FLAG
E302 D7 08               STA B  TSKTMP
                  * ADVANCE TO NEXT TASK
E304 4A           EXEC04 DEC A
E305 84 0F               AND A  #$0F
E307 91 01               CMP A  CURTSK   SEE IF LAST TASK
E309 26 E8               BNE    EXEC03   BRA IF NOT FINISHED

                  *CHECK IF TASK FOUND IS RUNNABLE
E30B D6 09               LDA B  PTYTMP   GET HI PRIORITY
E30D D1 0B               CMP B  SYSPTY   COMPARE TO SYS PRIORITY
```

```
00703 E30F 25 D9           BCS    EXEC02   SEARCH AGAIN IF LOWER
00704 E311 96 08           LDA A  TSKTMP   TEST FOUND FLAG
00705 E313 2A D5           BPL    EXEC02   BRA IF NOT SET

00707                * RUNNABLE TASK FOUND, SET SYSTEM
00708                * PARAMETERS TO RUN IT
00709 E315 84 0F           AND A  #$0F
00710 E317 97 01           STA A  CURTSK   SET TASK #
00711 E319 8D 20           BSR    FNDTSB   GET TASK TSB
00712 E31B 54              LSR B           EXTRACT TIME LIMIT
00713 E31C 54              LSR B
00714 E31D 54              LSR B
00715 E31E C4 0F           AND B  #$0F
00716 E320 D7 02           STA B  TIMREM
00717 E322 AE 01           LDS    1,X      LOAD TASK SP
00718                * TEST FOR ANY INTERRUPT THAT OCCURED
00719                * DURING EXEC MODE
00720 E324 96 07           LDA A  INTREQ
00721 E326 26 C2           BNE    EXEC02
00722 E328 7F 0000         CLR    SYSMOD   SET USER MODE
00723 E32B 3B        INTRET RTI            RUN TASK
```

```
                     * RT EXECUTIVE AUX. SUBROUTINES
                     *
                     * ALL ARE REENTRANT SUBROUTINES THAT
                     * PASS PARAMETERS AS FOLLOWS:
                     *
                     * ENTRY:  TASK # IN ACC A
                     *
                     * RETURN: TASK # IN ACC A
                     *         TASK STATUS BYTE (NEW) IN ACC B
                     *         ADDR OF TSB IN XR

                     * SUBR TO TURN TASK ON
E32C 8D 0D           TSKON  BSR    FNDTSB
E32E CA 80                  ORA B  #$80
E330 E7 00           RESTSB STA B  0,X
E332 39                     RTS

                     * SUBR TO TURN CURRENT TASK OFF
E333 96 01           CTSKOF LDA A  CURTSK

                     * SUBR TO TURN TASK OFF
E335 8D 04           TSKOFF BSR    FNDTSB
E337 C4 7F                  AND B  #$7F
E339 20 F5                  BRA    RESTSB

                     * SUBR TO FIND TASK STATUS BYTE/WORD
E33B 36              FNDTSB PSH A
E33C 84 0F                  AND A  #$0F
E33E 16                     TAB
E33F 48                     ASL A
E340 1B                     ABA
E341 8B 50                  ADD A  #$50
E343 36                     PSH A
E344 86 A0                  LDA A  #$A0
E346 36                     PSH A
E347 30                     TSX
E348 EE 00                  LDX    0,X
E34A 31                     INS
E34B 31                     INS
E34C E6 00                  LDA B  0,X
E34E 32                     PUL A
E34F 39                     RTS
```

```
                    *  CHARACTER AND BYTE I/O ROUTINES
                    *
                    * SELECTS INTERFACE TYPE (PIA OR ACIA)
                    * ACCORDING TO LEVEL OF PIA INPUT CB5
                    * IF ACIA TYPE IS SELECTED, THE ADDRESS
                    * OF THE ACIA IS OBTAINED FROM 'IOVECT'
                    * WHICH WILL DEFAULT TO $8000

                    *READ CHAR WITHOUT PARITY OR RUBOUT
E350 8D 07          IN1CHR BSR    INBYTE   GET BYTE
E352 84 7F                 AND A  #$7F     STRIP PARITY BIT
E354 81 7F                 CMP A  #$7F     TEST FOR RUBOUT
E356 27 F8                 BEQ    IN1CHR   AGAIN IF RUBOUT
E358 39                    RTS

                    * READ 8-BIT BYTE
E359 37             INBYTE PSH B
E35A 8D 31                 BSR    IOAUX    SAVE XR + SAMPLE TYPE
E35C 26 26                 BNE    ACIAIN

                    * PIA SOFTWARE UART ROUTINE -
                    * INPUT ONE CHAR WITHOUT PARITY
E35E A6 04          PIAIN  LDA A  4,X
E360 2B FC                 BMI    PIAIN    WAIT FOR START BIT
E362 6F 06                 CLR    6,X      SET 1/2 BIT TIME
E364 8D 3B                 BSR    STRTBT   RESET TIMER
E366 8D 35                 BSR    WAITBT   WAIT FOR TIMER
E368 C6 04                 LDA B  #$04
E36A E7 06                 STA B  6,X      SET TIMER TO FULL BIT TIME
E36C 58                    ASL B           BIT COUNT=8
                    * LOOP TO INPUT 8 DATA BITS
E36D 8D 2E          PIAIN2 BSR    WAITBT   WAIT BIT TIME
E36F 0D                    SEC
E370 69 04                 ROL    4,X      SHIFT OUT DATA
E372 46                    ROR A           SHIFT IN A TO BUILD
E373 5A                    DEC B           DECR BIT COUNT
E374 26 F7                 BNE    PIAIN2   BRA IF NOT DONE
E376 8D 25                 BSR    WAITBT   WAIT FOR STOP BIT
E378 E6 06          CHKSTB LDA B  6,X      TEST FOR # STOP BITS
E37A 58                    ASL B
E37B 2A 02                 BPL    RESTOR
E37D 8D 1E                 BSR    WAITBT
                    *RESTORE REGISTERS + RETURN
E37F FE A010        RESTOR LDX    XTMP
E382 33                    PUL B
E383 39                    RTS

                    * ACIA CHAR INPUT ROUTINE
E384 E6 00          ACIAIN LDA B  0,X      GET STAT REG
E386 54                    LSR B           MOVE RDY BIT TO SIGN POS
E387 24 FB                 BCC    ACIAIN   WAIT IF NOT READY
E389 A6 01                 LDA A  1,X      READ DATA
E38B 20 F2                 BRA    RESTOR   BRA TO CLEANUP
```

```
                       * I/O SETUP SUBROUTINE
E38D FF A010 IOAUX  STX     XTMP      SAVE XR
E390 CE 8000        LDX     #$8000    LOAD XR WITH PERIPH PTR
E393 E6 06          LDA B   6,X       TEST FOR ACIA OR PIA
E395 C5 20          BIT B   #$20
E397 27 03          BEQ     AUXRET    BRA IF PIA
E399 FE A012        LDX     IOVECT    GET ACIA ADDRESS
E39C 39      AUXRET RTS

                       * SUBR TO WAIT FOR 1 BIT TIME
                       * AND RESET TIMER
E39D 6D 06   WAITBT TST     6,X
E39F 2A FC          BPL     WAITBT

                       * SUBROUTINE TO START (RESET) BIT TIMER
E3A1 6C 06   STRTBT INC     6,X
E3A3 6A 06          DEC     6,X
E3A5 39             RTS

                       * OUTPUT 1 CHARACTER SUBROUTINE TO
                       * PIA OR ACIA
E3A6 37      OUT1CH PSH B             SAVE ACC B
E3A7 8D E4          BSR     IOAUX     SETUP FOR ROUTINE
E3A9 26 15          BNE     ACOUT     USE ACIA SUBR IF TRUE

                       * PIA SOFTWARE UART CHAR OUTPUT
E3AB C6 04          LDA B   #4
E3AD E7 04          STA B   4,X       SPACE FOR START BIT
E3AF E7 06          STA B   6,X       SET TIMER FOR FULL
E3B1 C6 0A          LDA B   #10       INIT. BIT COUNTER
E3B3 8D EC          BSR     STRTBT    RESET TIMER
                       * BIT OUTPUT LOOP
E3B5 8D E6   POUT1  BSR     WAITBT    WAIT BIT TIME
E3B7 A7 04          STA A   4,X       SET BIT OUTPUT
E3B9 0D             SEC
E3BA 46             ROR A             SHIFT IN NEXT BIT
E3BB 5A             DEC B             DEC BYTE COUNT
E3BC 26 F7          BNE     POUT1     BRA IF NOT LAST BIT
E3BE 20 B8          BRA     CHKSTB    CHECK FOR STOP BIT + RTS

                       * ACIA CHAR OUTPUT ROUTINE
E3C0 E6 00   ACOUT  LDA B   0,X       GET STAT REG
E3C2 54             LSR B             SHIFT RDY BIT TO C
E3C3 54             LSR B
E3C4 24 FA          BCC     ACOUT     BRA IF NOT READY
E3C6 A7 01          STA A   1,X       STORE DATA
E3C8 20 B5          BRA     RESTOR    GO TO CLEANUP
```

```
                   * ERROR MESSAGE STRING
E3CA 20            ERRMSG FCB    $20,'E,'R,'R,$20,4.
E3CB 45
E3CC 52
E3CD 52
E3CE 20
E3CF 04

                   * CR/LF AND TAPE HEADER STRING
E3D0 0D            CRLSTR FCB    $0D,$0A,0,0,0,4,'S,'1,4
E3D1 0A
E3D2 00
E3D3 00
E3D4 00
E3D5 04
E3D6 53
E3D7 31
E3D8 04
                   *
                   * COMMAND CODE/ADDRESS TABLE
                   *
     E3D9          CMDTBL EQU    *
E3D9 42                   FCB    'B
E3DA E1C3                 FDB    BKPCOM
E3DC 44                   FCB    'D
E3DD E1A7                 FDB    DMPCOM
E3DF 45                   FCB    'E
E3E0 E1F8                 FDB    EXCOM
E3E2 47                   FCB    'G
E3E3 E201                 FDB    GOCOM
E3E5 4C                   FCB    'L
E3E6 E000                 FDB    LOAD
E3E8 4D                   FCB    'M
E3E9 E223                 FDB    MEMCOM
E3EB 50                   FCB    'P
E3EC E207                 FDB    PUNCOM
E3EE 52                   FCB    'R
E3EF E0D0                 FDB    PRSTAK
E3F1 53                   FCB    'S
E3F2 E20C                 FDB    SYSCOM
E3F4 1B                   FCB    $1B      (ESC) NEXT ROM OR USER DEFINE
E3F5 7000                 FDB    $7000
E3F7 00                   FCB    0        END

                   *  INTERRUPT VECTORS
                   *
E3F8 E28B                 FDB    IRQ      IRQ VECTOR
E3FA E280                 FDB    SINT     SWI VECTOR
E3FC E298                 FDB    NMI      NMI VECTOR
E3FE E147                 FDB    INIT     RESTART VECTOR
```

```
                    NAM    DIGCLK
                    OPT    0
               *  DIGITAL CLOCK DEMONSTRATION PROGRAM
               *
               *  WRITTEN AS TWO TASKS!
               *
               *  CLKTSK USES THE TIMED TASK CAPABILITY OF
               *  RT/68 TO KEEP TIME.
               *
               * DISTSK DISPLAYS THE TIME ON THE CONSOLE
               *  EVERY 20 SECS.
               *
               *  SET UP THE CLKTSK STACK
0070                ORG    $0070     BOTTOM OF STACK
0070 001E           RMB    30        ROOM FOR STACK
008E 0100           FDB    CLKTSK    INIT START ADDR
               *  DEFINE TSP FOR SYSTEM TABLE
     0088      TSP1   EQU    *-8
               *  CLOCK TASK MEMORY DEFINITION
0090 0001      SECS   RMB    1
0091 0001      MINS   RMB    1
0092 0001      HRS    RMB    1
0093 0001      TIMER  RMB    1       TIMER TO ACTIVATE PRINT
               *  DEFINE DISPLAY DELAY
     0014      DEL    EQU    20      PRINT DELAY IN SECS
               *  TASK #1 BEGINS HERE
0100                ORG    $0100
0100 7F 0093   CLKTSK CLR    TIMER     INITIALIZE TASK
0103 86 01          LDA A  #1
0105 B7 000A        STA A  TIMTSK    SET UP TIME TASK
0108 20 2E          BRA    CLNUP2
010A CE 0090   UPDATE LDX    #SECS     POINTER TO COUNTERS
010D 86 3C          LDA A  #60       TEST VALUE
010F C6 18          LDA B  #24       HOURS TEST VAL
0111 6C 00          INC    0,X       BUMP SECONDS
0113 A1 00          CMP A  0,X       TEST FOR OVER
0115 26 12          BNE    CLNUP
0117 6F 00          CLR    0,X       RESETS SECS
0119 6C 01          INC    1,X       BUMP MINS
011B A1 01          CMP A  1,X       CHECK FOR OVER
011D 26 0A          BNE    CLNUP
011F 6F 01          CLR    1,X       RESET MINS
0121 6C 02          INC    2,X       BUMP HRS
0123 E1 02          CMP B  2,X
0125 26 02          BNE    CLNUP
0127 6F 02          CLR    2,X       RESET HRS
               *
               *  TEST PRINT DELAY
0129 6C 03     CLNUP  INC    3,X       INCR TIMER
012B 86 14          LDA A  #DEL
012D A1 03          CMP A  3,X       TEST FOR TIME UP
012F 26 07          BNE    CLNUP2
               *  TURN ON DISPLAY TASK
0131 6F 03          CLR    3,X       RESET TIMER
```



```
0133 86 02          LDA A  #2        PUT TASK # IN ACC A
0135 BD E32C        JSR    TSKON     TURN TASK ON
               *  RESET TASK TIMER AND SUSPEND
0138 CE 003C   CLNUP2 LDX    #60       DELAY CONSTANT (60 HZ CLOCK)
013B FF 0003        STX    TSKTMR
013E BD E333        JSR    CTSKOF    SUSPEND
0141 8D 02          BSR    GIVEUP    RELEASE CONTROL (CALL EXEC)
0143 20 C5          BRA    UPDATE
               *  SUBR TO CALL EXECUTIVE
0145 7C 0000   GIVEUP INC    SYSMOD
0148 0F             SEI
0149 01             NOP
014A 7C A00E        INC    RELFLG
014D 3F             SWI
014E 39             RTS
               *  TASK TO PRINT TIME ON CONSOLE
               *
               *  SET UP STACK
0300                ORG    $0300
0300 001E           RMB    30        RESERVE STACK AREA
031E 0329           FDB    DISTSK    INIT P.C. ON STACK
     0318      TSP2   EQU    *-8       DEFINE TSP
               *  SET UP OUTPUT BUFFER
     0320      OUTBUF EQU    *
0320 0002           RMB    2         HOURS POS
0322 3A             FCB    ':
0323 0002           RMB    2         MINS POS
0325 3A             FCB    ':
0326 0002           RMB    2         SECS POS
0328 04             FCB    $04       EOT CHAR
               *  TASK BEGINS HERE
0329 96 92     DISTSK LDA A  HRS       GET HOURS
032B CE 0320        LDX    #OUTBUF
032E 8D 19          BSR    MAKDEC    MAKE ASCII BCD
0330 96 91          LDA A  MINS
0332 8D 15          BSR    MAKDEC
0334 96 90          LDA A  SECS
0336 8D 11          BSR    MAKDEC
0338 BD E141        JSR    CRLF      PRINT CR+LF
033B CE 0320        LDX    #OUTBUF
033E BD E07E        JSR    PDATA1    PRINT BUFFER
0341 BD E333        JSR    CTSKOF    SUSPEND CURRENT
0344 BD 0145        JSR    GIVEUP    CALL EXEC
0347 20 E0          BRA    DISTSK
               *  SUBR TO CONVERT BINARY VALUE IN ACC A
               *  TO ASCII AND PUT IN BUFFER
0349 C6 30     MAKDEC LDA B  #$30      ASCII ZERO
034B E7 00          STA B  0,X       ZERO MSD
034D 81 0A     MD1    CMP A  #10
034F 25 06          BCS    MD2
0351 80 0A          SUB A  #10
0353 6C 00          INC    0,X
0355 20 F6          BRA    MD1
0357 8A 30     MD2    ORA A  #$30      MAKEONE ASCII
```

```
01070 0359 A7 01           STA A  1,X
01080 035B 08              INX
01090 035C 08              INX
01100 035D 08              INX
01110 035E 39              RTS
01120      E07E     PDATA1 EQU    $E07E
01130      A00E     RELFLG EQU    $A00E
01140      E333     CTSKOF EQU    $E333
01150      E32C     TSKON  EQU    $E32C
01160      E141     CRLF   EQU    $E141
01170      0003     TSKTMR EQU    $0003
01180      000A     TINTSK EQU    $000A
01190      0000     SYSMOD EQU    $0000
01200               *  SET UP TASK STATUS TABLE
01210 A050                 ORG    $A050
01220 A050 00              FCB    0,0,0        TASK 0 NOT USED
      A051 00
      A052 00
01230 A053 80              FCB    $80          SET UP TASK 1 TSW
01240 A054 008B            FDB    TSP1
01250 A056 00              FCB    $00          SET UP TASK 2 TSW
01260 A057 031B            FDB    TSP2
01270 A059 00              FCB    0,0,0,0,0,0,0,0,0
      A05A 00
      A05B 00
      A05C 00
      A05D 00
      A05E 00
      A05F 00
      A060 00
      A061 00
01280 A062 00              FCB    0,0,0,0,0,0,0,0,0
      A063 00
      A064 00
      A065 00
      A066 00
      A067 00
      A068 00
      A069 00
      A06A 00
01290 A06B 00              FCB    0,0,0,0,0,0,0,0,0
      A06C 00
      A06D 00
      A06E 00
      A06F 00
      A070 00
      A071 00
      A072 00
      A073 00
01300 A074 00              FCB    0,0,0,0,0,0,0,0,0
      A075 00
      A076 00
      A077 00
      A078 00
```



```
      A079 00
      A07A 00
      A07B 00
      A07C 00
01310 A07D 00              FCB    0,0,0
      A07E 00
      A07F 00
01320 000A                 ORG    $000A        SET UP SYS CONSTANTS
01330 000A 00              FCB    0
01340 000B 00              FCB    0            ZERO PRIORITY
01350                      END

TOTAL ERRORS 00000
```

```
                              NAM    TAPIO
                              OPT    0
      1000                    ORG    $1000

                      * ALTERNATE TAPE I/O ROUTINES FOR RT/68
                      *
                      * USED TO READ OR WRITE OBJECT TAPES THROUGH
                      * AN ACIA OTHER THAN THE CONSOLE INTERFACE
                      * ACIA.
                      *
                      * THE ACIA ADDRESS IS DETERMINED BY THE
                      * CONTENTS OF $A014 - $A015
                      * WHEN OPERATION IS FINISHED, THE CONSOLE
                      * ACIA ADDRESS IS RESTORED.
                      *
                      *  ** POSITION INDEPENDENT CODE **
                      *
                      * INXIO IS CALLED TO INITIALIZE THE ACIA
                      * RDTAPE IS CALLED TO READ A TAPE
                      * WRTAPE IS CALLED TO WRITE A TAPE

      1000 FE A014 INXIO  LDX    ACIADR
      1003 86 03          LDA A  #$3
      1005 A7 00          STA A  0,X       MASTER RESET
      1007 86 15          LDA A  #$15
      1009 A7 00          STA A  0,X       OPER. PARAMETERS
      100B 39             RTS

      100C 8D 1A   RDTAPE BSR    SETADR
      100E BD E000        JSR    LOAD
      1011 20 0E          BRA    RESTOR

      1013 BD E047 WRTAPE JSR    BADDR     GET STARTING ADDR
      1016 FF A002        STX    BEGADR
      1019 BD E1D9        JSR    GETADR    GET ENDING ADDR
      101C 8D 0A          BSR    SETADR
      101E BD E0EE        JSR    TAPOUT

                      * SUBR TO RESTORE CONSOLE ACIA ADDR VECTOR
      1021 CE 8000 RESTOR LDX    #$8000
      1024 FF A012        STX    IOVECT
      1027 39             RTS

                      * SUBR TO CHANGE ACIA VECTOR TO DESIRED
                      * ALTERNATE ADDRESS
      1028 FE A014 SETADR LDX    ACIADR
      102B FF A012        STX    IOVECT
      102E 39             RTS
```



```
                      * DEFINE EXTERNAL REFERENCES
           A014       ACIADR EQU    $A014
           A012       IOVECT EQU    $A012
           E000       LOAD   EQU    $E000
           A002       BEGADR EQU    $A002
           A004       ENDADR EQU    $A004
           E0EE       TAPOUT EQU    $E0EE
           E047       BADDR  EQU    $E047
           E1D9       GETADR EQU    $E1D9
                             END

TOTAL ERRORS 00000
```

```
                      * BINARY LOADER FOR RT/68MX
                      * LOADS FORMATTED BINARY TAPES
                      * **POSITION INDEPENDENT CODE**
                      *
                      * MAY BE RELOCATED WITHOUT REASSEMBLY
                      *
                      * START BY: E,NNNN  NNNN=ADDR OF FIRST INSTR.

1FA0 86 11      BINLOD LDA A  #$11      READER ON CODE
1FA2 C6 3C             LDA B  #$3C
1FA4 BD E016           JSR    $E016
1FA7 8D 3E      BLOAD2 BSR    INBYTE    LOOK FOR PREAMBLE
1FA9 81 58             CMP A  #'X
1FAB 26 FA             BNE    BLOAD2
1FAD 8D 38             BSR    INBYTE
1FAF 81 39             CMP A  #'9       CHECK FOR EOF
1FB1 26 03             BNE    BLOAD3
1FB3 7E E012           JMP    $E012     TAPE OFF +RTS
1FB6 81 31      BLOAD3 CMP A  #'1
1FB8 26 ED             BNE    BLOAD2    BRA IF NOT DATA RECORD
1FBA 7F A004           CLR    CHKSUM    CLEAR CHECKSUM
1FBD 8D 28             BSR    INBYTE    GET BYTE COUNT
1FBF 4C                INC A
1FC0 16                TAB
1FC1 8D 24             BSR    INBYTE    GET BLOCK ADDR
1FC3 B7 A002           STA A  BLKADR
1FC6 8D 1F             BSR    INBYTE
1FC8 B7 A003           STA A  BLKADR+1
1FCB FE A002           LDX    BLKADR
                      * LOOP TO READ DATA RECORD + STORE
1FCE 8D 17      BLOAD4 BSR    INBYTE
1FD0 A7 00             STA A  0,X       STORE IT
1FD2 A1 00             CMP A  0,X       MAKE SURE IT CHANGED
1FD4 27 03             BEQ    BLOAD5    BRA IF IT DID
1FD6 7E E040           JMP    $E040     JMP TO TAPE OFF + ERROR
1FD9 08         BLOAD5 INX              INCR POINTER
1FDA 5A                DEC B            DECR BYTE COUNT
1FDB 26 F1             BNE    BLOAD4    BRA IF NOT END OF BLOCK
1FDD 8D 08             BSR    INBYTE    GET CHECKSUM
1FDF 7C A004           INC    CHKSUM
1FE2 27 C3             BEQ    BLOAD2    BRA IF OK
1FE4 7E E03C           JMP    $E03C     JMP TO TAPE OFF + ERROR

                      * SUBR TO READ BYTE + UPDATE CHECKSUM
1FE7 BD E359    INBYTE JSR    $E359     READ A BYTE
1FEA 36                PSH A            SAVE IT
1FEB BB A004           ADD A  CHKSUM    ADD TO CHECKSUM
1FEE B7 A004           STA A  CHKSUM
1FF1 32                PUL A            RESTORE DATA
1FF2 39                RTS

                      * MEMORY DEFINITION
     A004       CHKSUM EQU    $A004
     A002       BLKADR EQU    $A002
```