AMOS™

ALPHA MICROSYSTEM AM-100

TIMESHARING FLOPPY DISK OPERATING SYSTEM

OPERATOR'S MANUAL

ALPHA MICRO SYSTEMS™

17875-N Sky Park North
Irvine, CA 92714

AMOS™

ALPHA MICROSYSTEM AM-100
TIMESHARING FLOPPY DISK OPERATING SYSTEM

OPERATOR'S MANUAL

INDEX

INTRODUCTION TO THE ALPHA MICRO OPERATING SYSTEM

The ALPHA MICRO operating system is a disk-based timesharing system which gives the user greater throughput than any other personal computer system currently available. The experienced user will find that many features are supported which have previously been found only on large computer installations. The user who has not been exposed to large system practices can expect to undergo a necessary learning process before the full capabilities of the system can be realized. This section will give a brief introduction to some of the general features found in the operating system and supporting routines.

MULTI-TASKING - several different jobs (sometimes called tasks) may be run concurrently on the system in different memory partitions performing different user or system functions. These tasks may be controlled from separate terminals or they may be under control of a single terminal. Special system commands allow the controlling of multiple jobs from a single user terminal making the feature of the system available on a minimal hardware configuration. Some ongoing tasks (such as the print spooler) may run without direct control from any specific terminal. Command language features allow the automatic initiation of a series of tasks within a jobstream similar to batch mode on larger computers. Any job may initiate another job automatically upon completion of its current task or optionally start up a totally independent task in another jobstream. The number of concurrent tasks that the system will support depends on the type of functions being performed and on the amount of main memory available. Disk swapping is not supported in the floppy-disk version and all tasks must be resident in main memory during execution.

MULTI-USER - the system supports multiple user jobs under an account number structure which protects each user's files from alteration by other users. Each user account contains its own file system on disk and filenames may be duplicated in different user areas with no confusion or interaction. Each user job runs in its own memory area which must be allocated dynamically by system commands when the user logs on. Accounts may optionally have a password associated with them which prevents unauthorized access to the system. There is a system priviledged account number [1,2] which is used when performing system manager functions. These functions may not be performed by non-priviledged users thereby protecting the integrity of the system resources.

TIMESHARING - when multiple jobs are requesting CPU or I/O resources, the monitor schedules the sharing of these resources based on a real-time clock which is a built-in feature of the CPU board. This clock interrupts the processing on a recurring interval (typically once every 10-20 milliseconds) which allows the monitor to evaluate the current system demands and give increments of time to each job requesting it in sequence. The frequency of the clock is user selectable by a capacitor on the CPU board or by an external trigger input. Software commands allow the setting of priorities to different jobs for the control of the scheduling function. Jobs in I/O or sleep states do not require CPU time and are bypassed by the job scheduler giving full time to those jobs that actually need it.

DISK MANAGEMENT - user programs reference disk data by filename within user account numbers without regard to the physical location of that file on the disk. Most files are sequential in nature and do not need to be preallocated prior to their use. Upon deleting any file the operating system immediately reclaims all space used by that file and makes it available to all users thereby eliminating the need for a disk file compression program. Under the current version, random files must be preallocated as a contiguous area. This restriction is expected to be lifted in the next major release of the file system. The operating system supports multiple random access devices and the system disk may be any one of these devices. A garbage collection program may be run which analyzes the linking structure of the disk and recreates the bitmap if necessary. This is normally only required after major system crashes or disk destruction by programs still undergoing development.

DEVICE INDEPENDENCE - internal to the monitor are a set of file service routines which handle all I/O transfers to and from the peripheral devices. User programs (and internal monitor routines also) access the peripheral devices as "datasets" without regard to the actual type of device in most cases. A dataset may be reassigned dynamically at run time to operate with a different device if desired. Implementing a new device into the system merely involves writing a device driver to perform the physical read and write functions to the new device and putting the driver onto the disk in a specific account reserved for transient device drivers. All programs and system commands will then operate with that device without modification or even reassembly.

COMMAND LANGUAGE - all commands entered into the jobstream (usually from the user terminal) are processed by the monitor which then initiates the desired system or user task. A unique command language allows a group of commands to be created in an ASCII file (using the text editor) and given a name. When that name is entered into the jobstream the ASCII file is located and the commands are then executed in sequence as if they had been entered directly by the user on his terminal. These command lines may in turn call other command files or perform direct system functions as required. The command file may also contain parameters to be entered into the specific job being executed when they are requested as normal terminal input. These command files may be placed into the public command account [2,2] where they are available to all users or may be kept in the individual user's account. This system defines a simple yet flexible method of easily implementing new commands and multi-task functions without the regeneration of the system monitor programs.

STANDARD FILE SPECIFICATION FORMAT

The standard format for specifying a device and/or filename for the operating system is as follows:

```
DEVX:FILNAM.EXT[Y,Z]
```

where	DEV	is the 3 character code that identifies the device the user wishes to access
	X	is the device number
	FILNAM	is the name of the file on the specified device the user wishes to access
	EXT	is the extension to the desired filename
	Y,Z	is the project-programmer (PPN) code that identifies the disk area that the user wishes to access

Not all of the information above is necessary to access a file in most cases. Several system defaults exist that the user can take advantage of. Not every device requires the full format for access. In general, only the disk requires the full format. Those devices that are not file structured require only the DEVX: format. Non-file structured devices are devices like the line printer, paper tape reader, and cassette. File structured devices are random access devices like the disk or memory.

Several system defaults exist for the user. If no device code is entered the system will default to DSK:. If no device number is entered the system will default to device zero. If no extension is appended to the filename then one is usually defaulted to. The exact default extension is determined by the program being run, and will be mentioned in the descriptions of the system programs that follow later. If no PPN is entered the system will default to either a particular PPN or to the PPN the user is currently logged in under. Note that device numbers are needed only when there is more than one device asssigned to the code (EXAMPLE: DSK0 and DSK1), and the user wishes to access a device other than the default device. Usually this means device zero. However, the system will also default to the disk number of the disk the user is logged in under if the disk is the specified device. Confused? Then a few examples may help. To wit:

EX. 1) Assume that the user is logged in under PPN [101,4] on disk 0. Then the following are all equivalent.

```
DSK0:SAM.ASC[101,4]
DSK:SAM.ASC[101,4]
SAM.ASC[101,4]
SAM.ASC
SAM
```

The last example is equivalent if and only if the program being run defaults to an extension of .ASC if none is entered.

EX. 2) Assume that the user is logged in under PPN [10,10] on disk 1. Then the following are all equivalent.

```
DSK1:TRASH.LST[10,10]
TRASH.LST[10,10]
TRASH.LST
```

The above examples apply to system programs only. The system EXEC has rules of its own, and they will be discussed later.

One last item to be covered is the format for filenames and extensions. A filename may be from 0 to 6 alphanumeric characters long. An extension may be from 0 to 3 alphanumeric characters long. Most system programs have default extensions, and the user is better off using them since the use of defaults cuts down on the chance of an error appearing in the command line.

There are only two device codes that the system requires to be standard for proper system operation. One of them is DSK for the system disk, and the other is LPT for the system line printer. All others are assigned by the user when he creates an initial command file for the system EXEC. The list of devices and codes assigned to the monitor include both shareable and nonshareable devices. In general, disks are shareable and non-file structured devices are not. Memory is not shareable either. Since a disk swapping system is too slow for a floppy disk system, memory is partitioned instead into one or more jobs. Each user has a memory partition which can be expanded to meet his needs if more memory is available. Once assigned the partition becomes transparent to the user. More on that subject later.

SYSTEM EXECUTIVE FEATURES

The system EXEC has a slew of features that the user had best become very familiar with if he wants to know what he is telling the system to do. Remember this: this system is complex and powerful. One command line can initiate a lot of activity. As a result, the following TRUTH should be commited to memory:

THERE ARE NO "MINOR" OPERATOR ERRORS WITH THIS SYSTEM!

If it is at all possible the system will do whatever the user tells it to do. If it is not, then an error message results that explains why. There are no cryptic error codes in this system. all errors are reported to the user is plain english. No attempt will be made to list these messages since no explanation is necessary. When you get one you will know why.

Built into the EXEC is a series of editing and program control features that are summarized below. Since they are built into the EXEC they are a part of all system programs including the EXEC.

RUBOUT	Deletes the previous character from the line. If the system has a CRT then the character is physically erased from the screen. Otherwise, it is relisted for the user's reference. Rubouts can be chained together to delete a series of characters back to but not beyond the beginning of the line.
CTRL-U	A CONTROL-U will erase the current line all the way back to the beginning of the line. It acts as a series of RUBOUTS.
CTRL-C	A CONTROL-C will abort a job currently running in the user'S memory partition and immediately terminate any terminal output that may be buffered for display.
CTRL-S	A CONTROL-S will suspend the user's program at any time. Terminal input is still accepted, but not processed. This feature is useful when the user has a CRT, and he wishes to look at a particular portion of terminal output before it disappears.
CTRL-Q	A CONTROL-Q will resume a suspended job. Q is used instead of R because it is easier to reach with one hand.

The general format for running a program is to enter the name of the program as the first item of a command line followed by a seperator and any arguments the user wishes to pass to the program. A command line is a line of characters entered from the terminal or a command file while in EXEC mode. EXEC mode is distinguished from program mode by the presence of a PERIOD at the start of the line. This character is transmitted whenever the system enters IDLE mode, and is waiting for new instructions from the user. Legal seperators are SPACES and TABS (CONTROL-I). Any number of consecutive seperators counts as a single seperator. The line is terminated by a CARRAIGE RETURN (CR) / LINE FEED (LF) pair. Insertion of a CR will automatically insert and echo a LF immediately after it.

Once a command line is entered the EXEC generates a rather involved search for the program. The search is outlined below, and continues until a match is found. At that point the program is loaded (if not in memory) and executed. The order of searching is as follows:

1) Search resident program area in EXEC for FILNAM.PRG
2) Search user memory for FILNAM.PRG
3) Search DSK0 area [1,4] for FILNAM.PRG
4) Search the disk and area the user is logged in under for FILNAM.PRG
5) Search user memory for FILNAM.CMD
6) Search DSK0 area [2,2] for FILNAM.CMD
7) Search the disk and area the user is logged in under for FILNAM.CMD

If all of the searches are unsuccessful then the EXEC will echo the filename back to the user's terminal bracketed by question marks. Usually only the filename of the program the user wishes to run is entered. However, the EXEC allows the user to direct or limit the search by entering explicitly such things as the device name and number (DSKX:), the extension, and the PPN along with the filename in standard file specification format. Note that if a filename of more that 6 alphanumeric characters is entered only the first 6 are used. If the extension is entered and it is not a .PRG extension, then the EXEC assumes it is a command file. Only programs with extensions of .PRG are considered binary run modules.

EXEC COMMAND FILES

Command files are ASCII files that mimic terminal input to the EXEC and any programs initiated by the EXEC. Command files are generated by the editor. Input to the EXEC is exactly the same as if the user was entering the data from the terminal instead. In addition, the presence of a command file activates another group of EXEC commands that add flexibility to the command file processor. These commands are never seen by system programs. The optional data they generate can be transmitted to a program, but the commands themselves are transparent to all programs except the EXEC. The commands are described below:

:T	TRACE flag on. When the trace flag is on all monitor level output and program level output will echo as it it retrieved from the command file. This flag is reset when a command file is first initiated. A :T overrides all :S commands.
:R	REVIVE flag on. When this flag is on all program level output will echo as it is retrieved from the command file. This flag is reset when a command file is first initiated.
:S	SILENCE. This command resets the REVIVE flag.
:<...>	MESSAGE. Everything between the two angle brackets will be echoed to the user'S terminal regardless of the state of the TRACE or REVIVE flags. None of it ever gets to a program. The message may span more than one line. The EXEC will bypass everything after the ">" until it finds a CR/LF pair. At that point it resumes input.
:K	KEYBOARD INPUT. Allows the user to enter one line of data or commands to either the EXEC or to a program. Command file execution stops until the line is entered. Once a CR is typed at the user's terminal the line is processed, and input from the command file is resumed.
:P	PARTIAL KEYBOARD INPUT. Allows the user to enter only part of a line of commands or data from the keyboard. Everything between the :P and the CR/LF pair is transmitted to the EXEC or the program. The CR/LF pair is replaced with a space so that input may continue on the line from the next line in the command file.

All of these commands must be at the very beginning of a line or they are not treated as command file secondary commands by the EXEC. Command files may run any program on the system including other command files. There is no limit (except the size of the user's memory partition) on the nesting level of command files.

JOB PARTITIONING AND DISK PARTITIONING

As was mentioned earlier, memory is partitioned into jobs. Each job is assigned a terminal or a pseudo terminal. The assignment of pseudo terminals allows the user to run from more that one job partition at once from a single terminal. Pseudo terminal jobs are attached to hard terminals by the user. Any number of jobs may be assigned in memory limited only by the amount of memory the user has and the number of jobs he has uses for. The entire memory partition is contained in a memory map in the EXEC. If for any reason this map gets destroyed and the EXEC is still alive enough to function reasonable well it will tell the user about it. If such an error message occurs the user should immediately take steps to save the integrity of the disks, and reboot the system as soon as possible. Because of the existance of a number of programs that allow the user access to other job partitions the system is not 100% foolproof. It is intended for use by serious individuals who know reasonable well what they are doing.

Along with the memory map is a disk BITMAP, one for each disk on the system. This BITMAP contains data that indicates to the EXEC what disk records are currently occupied or unoccupied. Associated with each BITMAP is a double precision checksum. As long as the checksum is good the user is free to write or erase on the disk. The checksum is not interrogated during a read. Each write or erase on the disk modifies the bitmap, computes the checksum, and compares it to what EXEC computed the checksum would be. If a match is found then the write or erase is executed. This occurs once for each record written or erased. If the bitmap ever goes bad it must be recreated before the disk can be altered. There are two ways of doing this. One way is to run DSKANA to see if the disk is recoverable. If it is not then SYSACT must be run to reinitiallize the disk. If the format of the disk is bad it must first be reformatted into IBM compattable format via FORMAT, initiallized by SYSACT, and then recreated from the backup disk. YOU HAD BETTER HAVE A BACKUP DISK! Those user's who are not familiar with floppy disks will soon learn that rule!

The disk is also partitioned into areas. Each area has its own PPN and directory associated with it. Disk areas are dynamically altered as the need arises. As long as there is room on the disk for a particular file then that file can be entered into the user's partition. As the size of a user's area expands or contracts the potential area for other areas contracts or expands. there are no fixed limits on the size on a disk partition except those imposed by the physical limitations of the medium. Each disk PPN is contained in a master directory along with the area password and beginning directory block number. In order to gain access to the disk the user must log into a PPN, and issue the password associated with that PPN if it has one. Otherwise the user has no system access. Once logged into a PPN area he may read any file in any area, but he may only erase or write into files in his own area or any other area that has the same project number as his area. The project number is the upper half of the PPN. The programmer number is not interrogated during a write or an erase. Any attempt to write or erase in an area that does not have the same project number as the user's PPN generates an error message and a job abort. There is one exception to this rule: a user logged in under PPN 1,2 may read, write, and erase in ANY disk area. He may also run all priviledged system programs. System priveledged programs are those that may be run by a user logged in under PPN 1,2 and ONLY by a user logged in under PPN 1,2.

The system has several areas preassigned on disk zero for efficient operation. These preassigned areas are:

[1,2]	Reserved for the system administrator.
[1,4]	Public program storage area.
[1,6]	Device driver storage area.
[2,2]	Public command file storage area.
[7,7]	Public system library file storage area.

The system library file is used by the assembler. Device drivers are modules that interface a particular device at a particular address to the system file handler. The 3 alpha character name of the driver is its device code. The driver is accessed explicitly whenever the associated device code appears in a command line, provided that the code is first inserted into the device table at system initiallization time. In general, all area codes from [1,1] to and including [7,7] are reserved for present or future system functions. Users should not generate areas with these codes for their own use.

BOOTSTRAPPING THE SYSTEM

Once the address of the PROM that contains the bootstrap routine is entered into the CPU BOOTSTRAP ADDRESS register the system may be booted at any time by hitting the RESET and RUN switches. As soon as the EXEC is in memory it looks for its associated initial command file. When found it will perform the tasks assigned by the command file. Such tasks usually involve initiallizing the line printer spooler, initiallizing the device table and disk bitmaps, assigning jobs and job partitions, and logging in each job to a default PPN, among others. Once an EXEC has been generated systems may be customized to fit user needs by editing the initial command file. More than one monitor may be generated, each with its own initial command file. The boot routine looks for SYSTEM.MON. The initial command file associated with it is assigned by SYSGEN.PRG. Other monitors may be loaded via MONTST.PRG.

TERMINAL BUFFERING

Each terminal on the system has a group of 32 character buffers associated with it. these buffers are used for terminal I/O storage. All data entered by the user via the terminal is stored in a buffer or group of buffers. The EXEC and all programs look to the buffers for data. If none is available they wait until a line of data has been entered. Since the buffers are filled on a real time basis, it is not necessary for the user to wait until a line of data has been processed before entering one or more new lines. The system will buffer the data and transmit it to the EXEC or currently running program as needed. This is called "LOOK AHEAD" terminal processing. As the data is used the buffer area that the line occupied is reclaimed. The buffers are used for input, echoing of input, and program or EXEC output. Input has priority. No program is allowed to hog all of the buffers for output. It is remotely possible, however, for the user to fill all of the buffers by running a program that requires a lot of output and doing a lot of LOOK AHEAD input typing at the same time. If the data cannot be stored the data is not echoed, and the terminal bell or beeper is rung instead. Data that is inputted from the terminal is placed in an output buffer immediately for echoing. If a program is outputting data at the same time the two sets of data will intermix in random fashion depending upon the timing between the terminal processor and the program. MORAL: if you want to read what you have just entered, you had best wait until program output is done. If your confidence level is high enough you can chance it, hit the CR key, and wait to see what happens.

Data may be entered in either upper or lower case. The EXEC treats them both the same, the editor does not. The EXEC will output all messages in upper case. Most other programs will accept upper or lower case and output in upper case. What is entered is what is echoed at all times.

MONITOR STARTUP

Whenever the system is bootstrapped the user will notice the running of the initial command file that is associated with the monitor. This section and the next will deal with those programs that are usually run in a monitor command file along with others that are in the same general catagory of programs. Some of these programs can also be run by the user. Others are run only once when the monitor is initiallized. Each program is discussed seperately, and the discussion will include information on whether or not the user may run the program after the monitor has been initiallized.

RSVOPC.PRG

RSVOPC may be one of the first programs run when the monitor comes up. It would be embedded in the monitor if it is used. The function of RSVOPC is to trap out all reserved op codes and simulate those that are currently defined in CPU MICROM #3. In this way the presence or absence of the floating point MICROM has no effect on the software except for speed. Undefined op codes will cause an error message and a job abort. This program is run once to place its base address in the reserved op code trap location. It is never run by the user.

DYSTAT.PRG

DYSTAT is the dynamic status program that uses a video monitor to display status information to the user on a real time basis. The module has two default addresses, one for the controller's ram assignment, and one for the I/O port. These addresses can be changed by including them in the command line. This module is also embedded in the monitor. It may be run by the user at any time to reinitiallize the display or to change addresses. the program remembers the last set of addresses entered.

The display consists of one line of data per job assigned. The line of data is made up of the name of the job, the PPN the job is logged in under, the priority level the job is running under, the name of the last program run by the job, and the current status of the job. Also included is an arrow to show the currently active job. the two character symbol used for the current status column comes from the following list

TI	Terminal input wait state
TO	Terminal output wait state
LD	Program load state
SL	Sleep state (inactive)
DS	disk access in progress
IO	I/O access other that terminal or disk
EW	External wait (for input)
FW	Foreground processor wait state
RN	Running
^C	At monitor level waiting for a new command

Also included in the lower left hand corner is the current time in seconds. If the current status symbol is enclosed in brackets then the job has been suspended by the user by either the CONTROL-S function or by the SUSPND program.

JOB.PRG

This program is run only as a part of the initial startup sequence. It assigns a job by name to the job table, assigns it a hard or pseudo terminal, and assigns it an initial login PPN. One job, named JOB1, is already set up in the job table, and should not be reassigned. This program is run once for each job to be assigned. It must NEVER be run by the user.

DEVTBL.PRG

DEVTBL is the program that allocates device codes to the device code table. Devices that have more than one unit also have the unit numbers assigned. All devices listed prior to the slash are sharable devices. All devices that are listed to the right of the slash are non-shareable devices. A shareable device is one that can be used by more than one user at the same time. Non-shareable devices can be used only by one user until released by the system. An example of the former is the disk. An example of the latter is the line printer. This program may also be run by the user to list the devices assigned to the device table by running the program with no input arguments. It may not be run to alter the device table after initial startup.

BITMAP.PRG

BITMAP is the program run to assign a disk bitmap area in the EXEC for each disk controller on the system. It must never be run by the user, and must never be run until the device table has been assigned at startup time. Each line of input contains the device code, the bitmap size for the device, and the unit numbers currently active. Each device has its own bitmap in memory to save time reading the disk every time a new record is added or deleted. The bitmap is rewritten once for every file added to or deleted from the disk. While a file is open the bitmap in core is the only one used. Only one bitmap is assigned to a controller. If more than one unit is assigned to a disk controller then they share the bitmap area through a swapping scheme every time a unit change is executed and a write or erase is active. Reading from the disk has no effect on the bitmap in memory or on bitmap swapping.

LPTSPL.PRG

LPTSPL is the line printer spooler. It is assigned about 2K of memory when the monitor is initiallized. It is run only once during startup, and never by the user. The spooler grabs the line printer driver from area 1,6 on disk 0 and places it in memory within its job partition. This allows the user to use the spooler or access the line printer directly without having to share the driver. Once the spooler is activated it goes into an external wait state until PRINT assigns it a module to list. LPTSPL contains a queue table that allows the user to spool up to 16 different modules for listing on the printer without having to tie up the user's job or terminal. Printer spooling runs in parallel with all other jobs. Once a module is placed in the queue for printing via PRINT the user must not alter or erase the module from the disk until the spooler has completed the listing of it. Unspecified results may occur if the user violates this rule.

JOB PARTITIONING AND ALTERATION PROGRAMS

The following group of programs allows the user to gain access to and alter some of the parameters for any job he chooses from the list of those currently assigned. Some of them are run during monitor startup time, but all may be run by the user.

ATTACH.PRG

ATTACH is the module the user runs when he wishes to attach his terminal to a job with a pseudo terminal assignment. Attaching to a job that is assigned a hard terminal can cause system problems and much inter-user friction. ATTACH has one argument as an input. It is the job name. Once attached the job stays attached until the user detatches it. All terminal output from the attached job will go to the user's terminal. To run ATTACH the user types:

```
ATTACH   JOBNAME
```

where JOBNAME is the name of the job he wishes to attach his terminal to.

DETACH.PRG

DETACH is the program the user runs to detach his terminal from a job he has attached to. It is run the same way ATTACH is run.

FORCE.PRG

FORCE is the program the user runs to force terminal input to another job. FORCE has two modes. One mode allows the user to force one line of input to the specified job. The other allows the user to force multiple non-blank lines of input to a job. To force a single line of data or commands into a job (again a job with a pseudo terminal assignment only unless you like making enemies) the user types:

```
FORCE   JOBNAME   COMMAND/DATA STRING
```

where JOBNAME is the name of the job the user wishes to force, and the rest is the garbage he wishes to force down that job's throat. BY eliminating the data after the jobname FORCE can be placed in the multiple line mode. In this mode every line after the "FORCE JOBNAME" line is forced into the job until a blank line (I.E. CR only) is entered. Naturally the user should first attach the job'S output to his terminal via the ATTACH program. The blank line returns the user to his own job partition.

WAIT.PRG

WAIT is a program that allows the user to stall the execution of any more programs in his job partition until another job has completed a task. A task is considered to be complete when it goes to terminal input state, sleep, or external wait state. The format for waiting on another job is:

```
WAIT JOBNAME
```

where JOBNAME is the name of the job the user wishes to wait for.

KILL.PRG

KILL is a program that allows the user to kill the program currently running in another job partition. If the job partition has more than one program batched for execution then the next program in the batch will be executed after the killing of the current one. One KILL is needed for each program. The format for killing another job's programs is:

```
KILL JOBNAME
```

where JOBNAME is the name of the job the user wishes to kill.

SUSPND.PRG

SUSPND allows the user to put another job to sleep, thus effectively removing him temporarily from the CPU request queue. The format is the same as for KILL or WAIT.

REVIVE.PRG

REVIVE will revive a job the user has put to sleep via SUSPND. The format is the same as for SUSPND.

SETPRI.PRG

SETPRI is the program the user runs to set the priority level of his own or another job. SETPRI can also be used to print out the current priority of the user's own or another job. The four formats are as follows:

```
SETPRI          - Prints the current job's priority level
SETPRI N        - sets the current job's priority to "N"
SETPRI JOBX     - prints the priority level of JOBX
SETPRI JOBX N - sets the priority level of JOBX to "N"
```

where N is in the range 1 to 256. One is the lowest priority, and 256 is the highest. All jobs are initially set to priority level one. JOBX is, as usual, the name of the job if it is not the user'S own.

MEMORY.PRG

MEMORY is the program that the user runs to assign some memory to his job partition. The size of a job partition may be altered at any time by running MEMORY. The format for running MEMORY is:

```
MEMORY   XXX
```

where XXX is the number of 16-bit words the user wishes assigned to him. If there is a contiguous hunk of memory that large, you got it. By adding the letter K after the XXX the program will add three zeros after the XXX to give the user 1000 word chunks instead. All is not quite that simple, however. Consider the possibility of two or more jobs running at one time, each with a partition in memory of some size. If you are below someone else and you want a larger hunk of memory, where do tou get it from? The guy above you cannot be moved if he is running a program, and let's assume that there isn't enough of a contiguous area at the top for the size partition you want. All is lost, right? Wrong. There is a way out if you have the patience. First, either you kill everyone above you in memory or execute WAIT on each job above you and batch in two commands into your terminal. Both are MEMORY commands. The first is with XXX = zero, and the second is with XXX = the size you now want. When MEMORY sees a size argument of zero it will remove the job from the job queue and set to reclaim all memory assigned to the job. The second MEMORY call, after everyone is done (a SUSPND call can insure that they stay done), will shift everyone that was above you down and give you a hunk of memory from the top. In order to force the user into a zero MEMORY call first, MEMORY will not allow the user to expand his job partition upwards unless the last MEMORY call was a zero call. The user can make his job partition smaller at any time. The EXEC will move jobs above him down one at a time as they finish any programs they may be running.

There are three programs the user can run with no memory assigned to his job partition. One of them is MEMORY, of course. The other two are SYSTAT and LOG. If the user attempts to run any other program without a hunk of memory assigned to him the EXEC will automatically grant him the largest contiguous chunk of memory it can find in the system and assign it to his job partition.

SYSTAT.PRG

SYSTAT is a program that gives the user the current status of each job in the system, much like DYSTAT does on the video monitor. SYSTAT will also give the user the number of free records left on each of the system disks, and clears the bitmaps from memory. This last step allows the user to change disks without having the bitmap for the old one in memory. The user should make it a point to run SYSTAT every time he inserts a disk into ANY drive. SYSTAT has no arguments to enter into it, so the format is just SYSTAT. The data printed out includes the following: the job's name, the logged in PPN, the EXEC queue address in memory, the run status, the last program run, the size in words, and the base memory address for the partition. There is one such line for each job assigned to the EXEC. Below all of this is a list of active system disk drives and the number of free records left on each disk.

SYSTEM MONITOR GENERATION

Generating a new monitor is relatively easy. Once all of the object files that go into making the EXEC are created and linked there is relatively little to do to make a new monitor. The only time the user needs to relink the EXEC is when he wishes to change a module within it. Usually this will be the disk driver that is to be assigned as the system disk. The only other time that the user would probably want to generate a new monitor is to have one handy in a different job table configuration for some other mode of system use. By using a command file for the linking of the EXEC most of the difficult part is done. In order to change the system disk driver the user must rename the current disk driver (SYSDSK.OBJ) to something else and rename the new disk driver to SYSDSK.OBJ. The new EXEC may now be created. To change any other device driver in the system the user need only rename them and include them in area [1,6] on disk 0. If the new driver is the line printer driver (LPT.DVR) and the system is using the line printer spooler (LPTSPL) then the system will have to be rebooted. That is the only other action that may be needed to change a driver. Adding a driver requires editing the initial command module for the monitor to add the driver code to the driver table (and possibly the bitmap table if it is a disk controller).

Once the EXEC and drivers are set up, and the user has created an initial command file for the monitor he wishes to generate he must run SYSGEN.PRG. The user must be logged into the area that has the EXEC and command file he wishes to use for his new monitor before running SYSGEN. Initiallizing SYSGEN requires that the user type SYSGEN followed by a CR. After a moderate pause SYSGEN will ask the user for the initial command file. Enter the name and extension of the module you have created. SYSGEN will then ask you for the resident programs you wish to have embedded within the monitor. Entries are one name per line (SYSGEN defaults to an extension of .PRG for each name entered) until all programs have been entered. A CR response only will terminate this phase of SYSGEN. At that point SYSGEN will ask you for the name you wish to assign to this monitor. A default extension of .MON exists here. If a monitor already exists in the user's area with the entered name SYSGEN will automatically erase it and enter the new monitor in its place.

That is it except for a discussion about resident programs. One of them may be mandatory. It is RSVOPC, which is mandatory if and only if the third CPU MICROM is not in the system. It is used to simulate the block moves and floating point op codes, all of which lie in MICROM #3. DYSTAT is also mandatory if the user has a video monitor he wishes to use with this monitor for status display. Other than that none are required. Any program embedded within the monitor is available to ALL users at the same time. It is a requirement, therefore, of all programs that are monitor resident that they be purely reentrant. A purely reentrant program is one that can be shared by more than one user at the same time. It must not have any data storage areas within it. All data access must be done via CPU registers, the stack, or external buffers. A program that is resident in the monitor can be run by all users without having to replicate the program in each user's area. Not many programs qualify as purely reentrant. The BASIC run-time module does. It would be to the user's advantage to include this program as a resident program if there are two or more jobs that will be running basic at the same time. As a resident program it will occupy memory only once, not once for each user, thus freeing up more memory for the BASIC programs.

MONTST.PRG

MONTST is the program the user runs when he wishes to run under a monitor other than SYSTEM.MON. Once the system is on the air MONTST may be run at any time. the format for running MONTST is:

```
        MONTST  FILNAM
```

where FILNAM is the name of the monitor the user wishes to run. A default extension of .MON exists if none is entered. MONTST will load and start the specified monitor if it can be found.

SYSACT.PRG

SYSACT is a priviledged user program. That is, it may be run ONLY by a user logged in under PPN [1,2]. This program is the system accounts maintainance program. It has several functions which are described below:

H	Prints out the list of functions performed by SYSACT.
L	Lists the current account PPNs and their passwords.
A	Adds a new PPN and password to the system. The format is "A PPN" where PPN is the new account to be added. SYSACT will then come back and ask for a password if the account is not already active. Passwords are from 0 to 6 alphanumeric characters long. A CR entry only makes the account a system "freesy". Any user can log into it without having to give a password.
C	Change the password of an active account. Works the same as the "A" function except that the account is not added.
D	Deletes an account from the system. The account had better not have any files in it or they are lost forever. So are the disk blocks they occupy until the user runs DSKANA to reclaim them. DSKANA will also reclaim any directory blocks assigned to the account.
I	Initiallizes the disk. This one wipes out the whole thing! To be sure that is what the user wishes this command will ask the user if that was his real intention. A "Y" answer will initiallize the disk to account 1,2 only with no password, no bitmap, and no files. Any other answer gets a nasty reply instead.
E	Exit to monitor. If the account record has been altered via an A, D, C, or I entry it will be rewritten at this time.

To run SYSACT the user must first log in under PPN [1,2]. He the types "SYSACT" followed by a CR. When SYSACT responds with an asterisk enter one of the above commands (followed by a space and the PPN if A or C). If the command is illegal SYSACT will tell you to type "H" for help. SYSACT will keep on responding with asterisks until the user finally enters an "E".

SYSTEM UTILITY MODULES

There are a lot of utility modules in this system with more to come later. Some of them were written as diagnostic aids during system creation, and they are included for the user for the same reason. Many of the programs accept what is called "wild card" filename formats. This format allows the user to act upon a group of files with one or more common characters without having to enter all of the filenames explicitly. Just as it is easy to list a group of files via a wild card filename, it is also easy to erase them. With this in mind the user should read the following section very carefully. It deals with the rules of wild card filenames. After that comes a list of system programs, alphabetically for the most part, with the editor, assembler, and assembly support programs in seperate sections at the end.

WILD CARD FILENAME RULES

Wild card filenames are acceptable to many of the system programs. They make it possible for the user to make one or more character positions in either the filename or extension (or both) "don't care" (I.E. always match) positions. Two characters are used to invoke wild card filenames. one is the question mark, and the other is the asterisk. The question mark will wild card one and only one character position. It may be used once for each position the user wishes to always match upon. The asterisk will wild card the position it occupies AND ALL POSITIONS AFTER IT to the end of the filename or extension. As a general rule, any program that accepts wild card input will ALWAYS default to an extension of ".*" (I.E. all extensions are a match) if no extension is explicitly entered. REMEMBER THAT! A few examples of wild card usage will best illustrate its usage.

EX.1 ABC?.PRG

All filenames that are four characters long, start with ABC, and have extensions of .PRG will match.

EX.2 ???.TXT

All filenames that are three characters long with extensions of .TXT will match

EX.3 FA*.PRG

All filenames FROM 3 TO 6 CHARACTERS LONG that start with FA and have extensions of .PRG will match. Note especially the variable field length that is applicable once the asterisk is invoked.

EX.4 SAM.*

All files with filenames of SAM and ANY extension will match. This includes files with no extensions.

EX.5 *.LST

All files with extensions of .LST will match including files with no filename.

EX.6   SAM

Same as example 4 since ".*" is an automatic default extension

EX.7   *.*   or   *

ALL FILES MATCH!!

BMVR.PRG

BMVR is the program the user runs to program PROMS via the CROMEMCO BYTESAVER. It makes 64 passes over the PROM with the data the user specifies. To run BMVR the user types "BMVR" followed by a CR. The program will then request the starting memory address for the data to be written. Enter the OCTAL address followed by a CR. BMVR then asks for the PROM address. Enter the OCTAL starting address of the PROM followed by a CR. Make sure the write swith is on.

CLEAR.PRG

CLEAR will clear out the video monitor display memory and port status. The memory base address is assumed to be 144000 OCTAL, and the port is assumed to be at 177704 OCTAL.

COPY.PRG

COPY is the program the user runs to copy a file or group of files from one area and/or disk to another, or to copy a file or group of files to any other output device on the system. The general format is OUTPUT FILE = INPUT FILE with a wild card option available. The general format is as follows:

```
COPY DEVX:FILNAM.EXT[Y,Z]=DEVA:FILNAM.EXT[B,C]
```

The full format is required only for disk devices. Note that a file can be renamed at the same time. A protection violation error message occurs if the user attempts to copy outside of his disk and/or PPN unless he is logged in under PPN [1,2]. Note that only one file at a time is transferred with this format. Note also that COPY does not alter or erase the input file. The format for wild card COPY is a little different, and is as follows:

```
COPY /X=DEVX:[Y,Z]FILEA,FILEB,...,FILEX
```

or

```
COPY DEVN:/X=DEVM:[A,B]FILEA,FILEB,...,FILEX
```

In the first example the copy will go to the user's disk and PPN. In the second example the copy will go to the user's PPN on disk N, or to DEV unit N if it is not the disk. The "/X" is required to activate wild card format. Each of the input files may use full wild card format, and each has a default extension of ".*". Note that the input PPN comes after the device code, not after the filenames.

In either mode COPY will copy the input file(s) to the output file(s). With wild card format the names of the input files become the names of the output files. If the specified output device is the disk and the files already exist in the disk area they are first deleted before the copy occurs. This is true in both modes. In wild card mode the files are listed as they are copied.

DEL.PRG

DEL is the program the user runs to delete modules from his job partition. Since the system has the ability to place programs and/or files in memory and leave them there, a mechanism must be created to delete them. This program is it. Normally when the user runs a program it is fetched from the disk, executed, and deleted from memory when it finishes. LOAD and UNPACK allow the user to place files or programs in his memory partition permanently until he deletes them explicitly. DEL does the deletion. The format to run DEL is:

```
DEL  FILEA,FILEB,...,FILEX
```

where each filename entered may be in wild card format, and each filename entered has a default extension of ".*".

DIR.PRG

DIR is the program the user runs to get a listing of part or all of his disk directory, someone else's disk directory, or all directories on a disk. Each line of output consists of the requested information about the file being listed. The first file listed in each PPN also lists the PPN at the end of the line. At the end of the entire listing is a totals line that consists of the total number of files listed and the total number of disk records occupied by the files listed, with both numbers in decimal.

There are many options to DIR, and a few defaults too. DIR will default to a listing of all the files and their sizes in the user's disk and PPN. From there on things get complicated. DIR allows the user to enter wild card filenames, options other than name and size, other PPNs, wild card PPNs, and even allows the user to chain groups of these as individually processed entries. Rather than try to explain all of the options in detail, a summary of available options followed by several examples will be used.

First, if no filenames are explicitly entered then DIR defaults to *.* for an entry. If no disk drive is entered then DIR defaults to the user's disk. If no PPN is entered then DIR defaults to the user's PPN. If no option switch is entered then DIR defaults to name and size. The presence of a slash on the input line places DIR in option select mode. The list of options follows:

M	List all modules. This switch overrides any filenames entered.
S	List the number of records each file occupies.
L	List the file type, linked or contiguous.
C	List contiguous files only. To be listed the file must match by both wild card name and file type = C.
B	List the base record number of the files listed.
T	List totals only. This switch suppresses file name printouts. Only the total files and records printout is listed.

The user selects his options by entering a slash (/) followed by the switches he wishes activated. The list is terminated by a space or tab. If no slash or filenames are entred DIR will default to /MS.

PPNs may be entered explicitly in the standard [X,Y] format, or they may be entered in wild card format to list a group of PPNs. To wild card either half of a PPN the user need only replace the X or Y with an asterisk. It is all or nothing per each half.

Filenames can be entered in wild card format with default extensions of ".*", with each name in a list of names seperated from the other by a comma. Other options are seperated from each other by a space or tab. Whenever DIR finds a [X,Y] entry, wild card or not, it considers it to be the end of the options input line for that listing. Each PPN entered breaks the input line into individual input lines. Any options in one section do not carry over to another. The final totals listing reflects the sum of all sections, however. Now for a few examples.

EX.1
```
DIR
```

Lists all modules and their sizes in the user's logged in area.

EX.2
```
DIR  *.LST
```

Lists all files with extensions of .LST in the user's area.

EX.3
```
DIR FILEA,FILEB,...,FILEX
```

Lists the specified files in the user's area. Wild card format is allowed on each with a default extension of ".*" on each.

EX.4
```
DIR /MT
```

Lists totals only for all files in the user's area

EX.5
```
DIR *.LST /T [101,4]
```

Lists the totals only for all files with extensions of .LST in area [101,4] of the user's disk.

EX.6
```
DIR DSK0:*.LST [*,*] DSK1: *.TXT[*,*]
```

Lists all files with extensions of .LST on disk 0, and all files with extensions of .TXT on disk 1. Note that DIR allows a space after the DSKX: and before the [X,Y] entries if the user wishes.

EX.7   DIR /MT [*,*]

Lists the totals for all files on the user's disk.

EX.8   DIR DSK0: /MT [*,*] DSK1: /MT [*,*]

Lists the totals of all files on disks 0 and 1. Note that the space after the option switches is required.

EX.9   DIR /MTC [*,*]

Lists the totals for all of the contiguous files on the user's disk.

EX.10   DIR DSK0:*.MON[1,4] DSK0: SAMPLE [*,10] DSK1: *.TXT [101,*]

Lists all files on disk 0 in area [1,4] with extensions of .MON. Lists all files on disk 0 in any area with a programmer number of 10 that has a filename of SAMPLE and any extension. Lists all files on disk 1 in any area with a project number of 101 that has an extension of .TXT. Lists the totals for all of them combined.

If DIR comes upon a disk area that is empty, and that area was referenced by a non-wild card PPN, then DIR will give a directory empty message for each area so encountered.

DSKANA.PRG

DSKANA is the disk analysis program that finds bad files, reclaims lost records, and recreates the bitmap. At the end of program execution DSKANA will list, on the terminal, the number of records that were marked in use but not found to be assigned to any file; the number of records that were assigned to some file or another but not marked in use in the bitmap; and the number of file errors it found. DSKANA reads every user's directory and every record of every file in those directories. It keeps track of what the bitmap should be, and compares it to what is was. It rewrites the bitmap it computes at the end of the program. To run DSKANA the user types:

```
        DSKANA  DSKX:
```

where "X" is the drive number the user wishes to analyze. If the user finds any file errors listed at the end of the program he will have to rerun DSKANA in full listing mode to find out what files are bad. Bad files will have to be removed from the disk, and the user MUST run DSKANA immediately to fix the bitmap. Once that has been done the bad files can be copied from the backup disk (remember the backup disk?). To run DSKANA in full list mode the user types:

```
        DSKANA DSKX:  /L
```

A full listing of all PPNs, all directories and the records they occupy, all files and the records they occupy, and any file errors will appear on the terminal.

DSKCPY.PRG

DSKCPY is the program the user runs when he wishes to create that all important backup disk. To run DSKCPY the user types "DSKCPY" followed by a CR. DSKCPY will respond with a request for the input drive number (0-7), and the output drive number (0-7). Only a single number is entered for each request. No "DSK:" is to be entered in front of the number. The user may remove the disk from drive zero any time after initiallizing DSKCPY and before entering the output drive number. This makes copying from drive one to drive zero the easiest way to go on a two drive system. DSKCPY will copy all of the input drive to the output drive, reread both of them at the end and verify the copy. Once DSKCPY finishes the user should reinsert the system disk if he removed it from drive zero. In any case, since one of the disks has been changed SYSTAT must be run after a DSKCPY to clear the bitmap from memory.

DSKDMP.PRG

DSKDMP is a program that allows the user to list the contents of any disk record on his terminal in octal. The format for running DSKDMP is:

```
        DSKDMP  DSKX:  NNN
```

where DSKX is the drive and NNN is the octal record number the user wishes to dump. Data appears as 16 bit words, in octal, 8 words per line.

DSKFIL.PRG

DSKFIL will locate a file on the disk and in the PPN the user specifies, and list the records it occupies on the disk in octal. The format is:

```
DSKFIL DEVX:FILNAM.EXT[Y,Z]
```

The usual defaults apply, and a default extension of .PRG also applies. The record numbers appear in the order the file occupies them.

ERASE.PRG

ERASE is the program the user runs to erase files from his disk. ERASE accepts filenames in wild card format with a default extension of ".*", so BE CAREFUL! The format is:

```
ERASE  DSKX:FILEA,FILEB,...,FILEX
```

The DSKX: is only needed if the user wishes to erase on a drive other than the one he is currently logged in under. If it is used then the files must be in the same PPN or no erase occurs. Each file erased is listed on the terminal as it is being erased.

FILCOM.PRG

FILCOM allows the user to compare the contents of two files and list the locations that do not match on the terminal. The format is:

```
FILCOM  FILEA  FILEB  XXX
```

where FILEA is the name of one of the files to be compared, FILEB is the name of the other, and XXX is the number of mismatches the user wishes to terminate the program after. XXX is a decimal entry, and if none is entered a one is assumed. the default extension for each filename is .PRG. FILCOM brings both files into memory before starting the compare. Errors are listed one per line with a line consisting of the octal byte address of the mismatch followed by the filename and octal data for each of the files.

FILDMP.PRG

FILDMP will dump the entire contents of a disk file in octal on the user's terminal. The format is:

```
FILDMP  DSKX:FILNAM.EXT[Y,Z]
```

with the usual defaults and a default extension of .OBJ. The listing is in octal bytes, 16 bytes per line until the file is exhausted.

FORMAT.PRG

FORMAT will reformat a disk in IBM compatable format of 26 records per track, 128 bytes per record, and 77 tracks. The entire disk is rewritten. Once formatted it must be reinitiallized by SYSACT before it can be used.

THIS PROGRAM WILL ONLY RUN WITH THE ALPHA MICRO DISK CONTROLLER!

To run FORMAT the user types: "FORMAT" followed by a CR. FORMAT will then request the drive number that contains the disk to be formatted. The entry consists of a single octal number from 0 to 7. Some time prior to giving the dirve number the user must engage the FORMAT ALLOW switch on the controller. If that switch is not on formatting cannot be performed.

LOAD.PRG

LOAD will get a program or file from the disk and place it in the user's job partition as a permanent memory module. If the program or file to be loaded is already in memory it will be deleted and replaced. The format is:

```
LOAD    DSKX:FILNAM.EXT[Y,Z]
```

The usual defaults apply, and a default extension of .PRG also applies if none is entered. The only program that cannot be loaded is LOAD.PRG. If you try it you will crash your system. LOAD will print the octal base address that the file is loaded into when the load is complete.

LOG.PRG

LOG is the program the user runs to log himself in under a PPN. It is the only other program the user may run without being logged in. The other is SYSTAT. To run LOG the user types:

```
LOG  X,Y          or          LOG [X,Y]
```

Either format is allowed. LOG will search the currently logged in disk for the entered PPN. If it cannot find it LOG will search all system disks starting with drive 0 for the specified PPN. Once found LOG will test to see if a password is needed before granting entry. If not, the user is in and LOG will print where the user came from and where he is by drive and PPN on the user's terminal. If a password is needed LOG will ask for it. The user must enter the password correctly or no access to the PPN is allowed. The password the user enters is not echoed for security reasons. If the user wishes to limit the PPN search to a particular drive he may do so by placing the drive number in front of the PPN thusly:

```
LOG  DSKX:Y,Z          or          LOG  DSKX:[Y,Z]
```

Either format is allowed. X, of course, is the drive number LOG is to search on. If the user merely wishes to know where he is currently logged in he types "LOG" followed by a CR. The PPN and drive number are listed on the user's terminal.

LOGOFF.PRG

LOGOFF is the program the user runs when he wishes to leave the system and keep other users out of his area while he is away. The format is to type "LOGOFF" followed by a CR. LOGOFF will tell the user what PPN he was in when he logged off.

MAP.PRG

MAP will display the files that are residing in the user's partition as permanent files. MAP runs much like DIR with wild card filenames and a list of option switches. The format is:

```
        MAP  FILEA,FILEB,...,FILEX   /OPTIONS
```

Full wild card format and a default extension of ".*" exists for each filename. The slash informs MAP that option switches are coming up. The user may use as many of them as he wishes. The list of switches is terminated by a CR. The option switches are:

F	Print free memory. The S switch is needed to get the size.
S	Print the size in words of each file listed.
B	Print the octal base address for each file.
M	Print all modules. Overrides any filenames also present.
U	Print the files that match in the user's partition.
R	Print the monitor resident files that match.
H	Print the "hashmark" of each file listed.

If no filename and no option switches are entered then MAP will ddefault to a "/MSFU" condition. All printouts are by filename, decimal size in words, octal base memory address, and a special "hashmark" format. The hashmark is a specially constructed checksum that includes all of the data in the file and the size of the file in the computation. It is printed out as a 12 digit number three digits at a time with a "-" seperator. It is intended as a check on the validity of data within a file that has been loaded into memory.

MAKE.PRG

MAKE is the module the user runs when he wishes to make a module on the disk in his area. Since the editor will not allow the user to edit a module that does not exist, he has to run MAKE first or copy some other module or even rename some other module. Somehow or other he has to create the module before he can edit it. MAKE creates a module with no data in it. The format is "MAKE FILNAM.EXT" with a default extension of .MAC if none is entered.

PACK.PRG

PACK is a module that allows the user to pack a bunch of modules into a single module. This allows the user to load the whole group (or subgroup) via UNPACK without having to load each one seperately. PACK will store each module with its name, size, data, a special 20 byte header, and a checksum character at the end. Modules so packed may be outputted to non-file structured devices and loaded from them by name. Files packed onto the disk will occupy less disk area than they would individually. Every file must occupy whole records even if they are not long enough to fill the last record. This means that every file will have some unused room at the end of the last record unless its size is a multiple of 128. By packing those modules into a single module most of that unused space can be reclaimed. Modules so packed cannot be accessed individually except through the UNPACK module. To run PACK the user types:

```
        PACK  DEVX:FILNAM.EXT
```

where DEVX: is any output device on the system that uses a medium that can also be read by the system (I.E. disk, cassette, paper tape, etc.). The FILNAM.EXT is needed only if the device is the disk, and a default extension of .PAK will apply if none is entered. Once PACK has been initiallized it will respond with an asterisk. The user then enters the name of the first file he wishes to output. When that file is written another asterisk appears, and the user enters another filename. This goes on indefinitely until the user enters a CR only for a filename. At that point PACK closes the output file and exits. Each filename entry has a default extension. That default is initially set to .PRG, but PACK will remember and default to the last explicitly entered extension the user enters.

PRINT.PRG

PRINT is the program that transfers filenames to the line printer spooler for printing via the spooler. The format for running PRINT is:

```
        PRINT  DSKX:FILNAM.EXT[Y,Z]
```

The usual system defaults apply, and a default extension of .LST also applies if none is entered. PRINT will check to see if the spooler is active. If it is not an error message appears. If it is then PRINT places the filename into the spooler queue and exits back to the user. If the user tries to overflow the spooler queue table an error message occurs, and the filename is not stored. The use of the spooler allows the user to print listings and still have parallel access to the system for other things.

RENAME.PRG

RENAME allows the user to rename a file in his disk area to a new name and/or extension. The format is:

```
RENAME  OUTFIL.EXT=INFIL.EXT
```

A default extension of .MAC exists for the output filename, and a default extension exists for the input filename that is the extension used for the output filename, default or otherwise. A module can also be renamed in memory by immediately preceeding the output filename with "COR:". Otherwise the disk is assumed and an error occurs if it is not or if the file cannot be found. The equals character is mandatory.

SAVE.PRG

SAVE is the program the user runs to save modules in his memory partition into his disk partition with the option to rename the original module if it is already on the disk. Filenames may be entered in full wild card format. The format for running SAVE is:

```
SAVE  FILEA,FILEB,...,FILEX   /EXT
```

where a default extension of ".*" exists for each filename if none is entered. The "/EXT" entry selects the rename function. If the rename function is not selected then modules are saved or replaced, whichever is applicable. If the rename function is selected the modules that are to be replaced are first renamed with their original filenames and the extension entered after the slash. SAVE checks to see if the backup file also exists, and if it does SAVE will delete it before doing the rename. All deletes, renames, and saves are listed on the user's terminal.

SLEEP.PRG

SLEEP allows the user to put his job to sleep for a specified time. To run SLEEP the user types: "SLEEP XXX" where XXX is the number of clock ticks (in decimal) the user wishes to sleep for. If the line clock is tied to a 60 cycle AC source then 60 ticks equals one second.

SIZE.PRG

SIZE allows the user to find out how big a disk file is in bytes. To run SIZE the user types:

```
SIZE  DSKX:FILNAM.EXT[Y,Z]
```

The usual system defaults and a default extension of .PRG apply. The size is typed on the console in decimal bytes.

REDALL.PRG

REDALL is a diagnostic program that will read all of the records on a disk, and report any read errors on the terminal. Read errors are reported by the EXEC. To run REDALL the user types:

```
REDALL DSKX:   NNN
```

where DSKX: is the disk the user wishes to read. NNN is the maximum number of records the user wishes REDALL to read. If no NNN is entered REDALL will read the entire disk.

RNDRED.PRG

RNDRED is a random disk track seeking diagnostic. It will select track numbers randomly, perform a seek and read on a random record on the track, and list any read errors it finds. To run RNDRED the user types:

```
RNDRED DSKX:
```

where DSKX: is the disk the diagnostic is to run on. This program will run forever or until the user enters a control-c on the console. RNDRED will not stop immediately after a control-c, but it will stop in a few seconds worst case.

TYPE.PRG

TYPE will list a disk file on the user'S terminal much like PRINT will list it on the line printer. The format for running TYPE is:

```
TYPE  DSKX:FILNAM.EXT[Y,Z]
```

The usual system defaults and a default extension of .LST apply. The file is listed exactly as it exists, and is assumed to be an ASCII file.

TXTFMT.PRG

TXTFMT is a program that takes an ASCII file, crunches it per the directives within the file, and creates a new file with the same filename and an extension of .LST. If the list file already exists then TXTFMT will delete it first before making the new one. Obviously then, the user cannot run TXTFMT on a file with an extension of .LST. To run TXTFMT the user types:

```
TXTFMT FILNAM
```

The file must exist in the user's disk area, and it must have an extension of .TXT. DO NOT ENTER AN EXTENSION AFTER THE FILENAME OR IT WILL BE ERASED! The list of formatting options follows. The first two options in the list are mandatory and MUST be the first two items in the file or some very unspecified results will occur. Each option is preceeded by a slash, and must be the very first thing on the line. Each is terminated by a CR. Those that do not accept arguments are to be terminated immediately. Those that do are to be terminated after the argument.

LINESIZE X	Sets the size of the output line to X decimal characters per line maximum.
PAGESIZE Y	Sets the size of the output page to Y decimal lines per page. A form-feed terminates the page.
INDENT X	Indents the next line X decimal characters.
SINGLE	Sets single spacing on output.
DOUBLE	Sets double spacing on output.
MARGIN X	Offsets the left hand margin X characters inward from the first character position.
FORMAT	Sets formatted output mode.
UNFORMAT	Sets unformatted output mode.
LINE X	Skips X decimal lines. A one is NOT assumed.
PAGE X	If no X is entered then the next line starts a new page. If X is entered then a new page is started if there are less than X decimal lines left on the current page.
CENTER ARG	Centers the rest of the line after the command.

The LINESIZE and PAGESIZE are required at the start of the file. TXTFMT will default to FORMAT and SINGLE. In unformatted mode data is transmitted as it appears, but the TXTFMT commands are still active. This allows the user to enter unformatted mode, use the TXTFMT commands if he wants to, and do a FORMAT to get back to formatted mode. In formatted mode leading spaces and tabs on a line are ignored. TXTFMT takes the data and makes lines of its own with no regard at all to the way the user entered data into the file. Words are never hyphenated to make a new line. Periods at the end of input lines get two spaces appended to them on output. The CENTER command ignores spaces and tabs that preceed the data to be centered, but not trailing spaces and tabs. In most cases it takes 4 to 6 trailing spaces after a CENTER argument to make the data look properly centered. In formatted mode tabs make no sense and should be avoided. In unformatted mode they are converted to spaces MODULO 8 by the line printer driver in the same manner as all the other files listed on the printer and terminal. This manual was formatted via TXTFMT.

UNPACK.PRG

UNPACK allows the user to unpack modules that were packed by PACK. The format for running unpack is:

```
UNPACK  DEVX:FILNAM.EXT[Y,Z]   FILEA,FILEB,...,FILEX
```

where DEVX: is the input device, FILNAM.EXT[Y,Z] is the file to be unpacked if the input device is the disk, and FILEA to FILEX are the files to be unpacked from the input device (and file) and loaded into memory. A default extension of .PAK exists for the input filename. The usual system defaults also apply if no device and/or PPN is entered. The filenames of the files to be unpacked allow full wild card format and have default extensions of ".*" for each. UNPACK will test to see if a module to be unpacked is in the input file. If it is UNPACK will test to see if it is already in memory. If it is the unpack is bypassed and a bypass message occurs. Unpack then goes on to the next file in the input file. If a file to be unpacked is not already in memory UNPACK will load it into memory as a permanent module and tell the user about it.

EDIT.PRG

EDIT is the system editor. It has commands to operate in both line and character mode. EDIT works with disk files only. EDIT contains a command buffer that expands to meet the input from the keyboard until a double altmode is entered. Altmode, or escape on some terminals, echoes as a dollar sign. Edit also traps the control-c input from the EXEC to prevent the user from exiting from EDIT via the control-c. Instead, the control-c will terminate a command string input or execution. A control-c will also terminate a listing if one is in progress. EDIT also allows upper and lower case inputs, and will recognize commands in either case. EDIT has 26 auxilliary buffers that can be used as temporary storage areas for data.

Most EDIT commands are single characters. Some require arguments, such as inserts and searches. Insert and search arguments are terminated by a single altmode character. A line of commands is terminated by a pair of altmodes. EDIT takes no action on a string of commands until the double altmode is entered. The user may buffer up as many lines of commands and/or insertion data as he wishes. EDIT will store it all until a double altmode is entered, and then operate upon the commands and data in the order they were entered until the command buffer is empty. The rubout and control-u features still operate as usual. While inserting characters into the command buffer, however, a control-c will kill the entire contents of the command buffer. If the user has made a mistake several lines back that he now recognozes a control-c spares him from having to execute it or reboot the system.

To run EDIT the user types: EDIT FILNAM.EXT where FILNAM.EXT is a file on the user's disk in the user's area. A default extension of .MAC exists if none is entered. As soon as EDIT finds the file on the disk it opens it for input, and reads in as much of it as it can into the user's memory partition. EDIT will leave about 2000 bytes of free memory if the file is too large to fit as a whole. A message will occur if this happens to inform the user of exactly how much room is left. Once the file is in memory EDIT opens a file for output in the user's area with the same name as the input file, but with an extension of .TMP. When the user exits from EDIT all data in the data buffer any any unread data from the input file are written to the output file. EDIT then looks for a file in the user's area with the input filename and an extension of .BAK. If it finds it the file is deleted. EDIT then renames the input file extension to .BAK, and renames the output file .TMP extension to the extension of the original input file. Notice that neither the input file nor the original backup file are altered until the user exits. In case of system or operator error no files are lost.

Once EDIT has been initiallized it echoes an asterisk on the user's terminal to indicate that it has an empty command buffer and is eagerly awaiting commands from the user. The user then enters commands, singularly or strung out as a series of commands, and terminates the process with a pair of consecutive altmodes. EDIT will then process the commands. Upon completion another asterisk is echoed on the console. While a string of commands or a listing is in progress the user may terminate the process and clear the command buffer by entering a control-c. If at any time while entering lines into the command buffer EDIT finds itself with less than 100 bytes of free memory left to store the data it will beep the terminal and ignore the line.

What follows is a list of commands and what they do. Commands are listed alphabetically. Many commands accept numeric arguments in front of them. These arguments direct EDIT to operate upon a designated subset of the data buffer, usually referenced to a pointer called DOT. Numeric arguments range from -65535 to +65535 with the plus always implied and zero having special significance. AN "H" modifier allows execution across the whole data buffer without referencing itself to DOT. Command execution always stops at the boundaries of the buffer if the user tries to go beyond them. In the following list the small letters are numeric modifiers, and the capital letters are the commands themselves. A dollar sign ($) always represents an altmode.

A	APPEND. Appends one or more records of the input file to the data buffer if there is at least 2000 free bytes of memory left and the input file pointer has not reached the end of the file.
C	CHARACTER ADVANCE. Moves DOT forward one character.
nC	ADVANCE DOT forward n characters.
-C	ADVANCE DOT backwards one character.
-nC	ADVANCE DOT backwards n characters.
0C	ADVANCE DOT backwards to the beginning of the current line.
D	DELETES the first character after DOT.
nD	DELETES the next n characters after DOT.
-D	DELETES the character in front of dot.
-nD	DELETES the previous n characters in front of DOT.
0D	DELETES characters from the beginning of the line to DOT.
HD	DELETES the whole buffer.
E	EXIT to monitor. Outputs data buffer and rest of input file, does rename functions, and exits.
F	FREE memory printout. Prints number of free bytes in memory in decimal.
Gx	GET auxilliary buffer "x", where x= A to Z, and insert it into the data buffer at DOT. DOT is moved forward the number of characters inserted.
Itext$	INSERTS text into the data buffer at DOT. Advances DOT the number of characters inserted. Any character except RUBOUT, CONTROL-C, CONTROL-U, or ALTMODE may be inserted. The insertion may be any number of lines long, or it may be as few as zero characters long.
0J	JAMS DOT back to the beginning of the data buffer.
nJ	JAMS DOT immediately in front of the nth character in the data buffer.
ZJ	JAMS DOT to the end of the buffer.

K	KILLS from DOT to the end of the current line.
nK	KILLS n lines of text from DOT.
0K	KILLS from the beginning of the current line to DOT.
-K	KILLS from the beginning of the previous line to DOT.
-nK	KILLS from the start of the -nth line from DOT to DOT.
HK	KILLS the whole buffer.
L	ADVANCES DOT to the beginning of the next line.
nL	ADVANCES DOT forward n lines. DOT is positioned at the start of the line.
0L	ADVANCES DOT backwards to the start of the current line.
-L	ADVANCES DOT backwards to the start of the previous line.
-nL	ADVANCES DOT backwards n lines from DOT, and positions DOT at the beginning of the line.
Ntext$	WHOLE FILE SEARCH. Searches the current buffer, starting at DOT, for the first occurrance of "text". If the search is unsuccessful in the current buffer it is written out, the data buffer is cleared, a new buffer full of data is read in from the input file, DOT is reset to the beginning of the buffer, and the search goes on. If "text" is found DOT is positioned just after it. If "text" is not found an error message occurs and any commands after the search in the command buffer are aborted.
nNtext$	Same as above, except that the search stops after the nth occurrance of "text".
N$	Same as Ntext$ except the last explicitly entered search argument is used.
nN$	Same as nNtext$ except that the last explicitly entered search argument is used.
R	Same as "-C".
nR	Same as "-nC".
0R	Same as "0C".
-R	Same as "C".
-nR	Same as "nC".
Stext$	SEARCHES the data buffer starting at DOT for the first occurrance of "text". Positions DOT just after "text" if the search is successful. If it isn't, then an error message occurs, DOT is positioned at the beginning of the buffer, and the remainder of the command buffer is aborted.
nStext$	Same as above except EDIT searches for the nth occurrance of "text".
S$	Same as "Stext$" except that the last explicitly entered search argument is used instead.
nS$	Same as "nStext$" except that the last explicitly entered search argument is used instead.

T	TYPE the data buffer contents from DOT to the end of the current line.
nT	TYPE n lines of data buffer text starting from DOT.
0T	TYPE the data buffer contents from the beginning of the current line to DOT.
-T	TYPE data buffer contents from the start of the previous line to DOT.
-nT	TYPE the data buffer contents from the start of the nth line from DOT to DOT.
Vx	VERIFY auxilliary buffer "x" contents, where x= A to Z. Lists the buffer contents on the terminal.
Xx	SAVE from DOT to the end of the current line in auxilliary buffer "x", where x=A to Z. The previous contents of the buffer are lost.
nXx	SAVE n lines of text from DOT in auxilliary buffer "x".
0Xx	SAVE from the beginning of the current line to DOT into auxilliary buffer "x".
-nXx	SAVE from the start of the nth line from the current line to DOT into auxilliary buffer "x".
;text$	SEMICOLON INSERT. Same as an "Itext$" insert except that the semicolon is also inserted into the data buffer.
TABtext$	TAB INSERT. Works the same as a semicolon insert.
SPACEtext$	SPACE INSERT. Works the same as a semicolon insert.
n<....>	REPEATS. All of the commands within the angle brackets are repeated n times. All EDIT commands can be executed in a repeat including other repeats. The maximum nesting level for repeat commands is eight. An error message and an abort occurs if the user overflows the nesting level. Search failures also abort repeats.

The end of a line is defined by a line feed. The beginning of a line is defined as the first character after a line feed. A line is terminated by a CR if it is inserted from the console, and EDIT inserts a CR/LF pair for any CR entry. Notice that it is possible to kill the current line without moving DOT by executing a "0KK". Similarly, it is possible to list the contents of the current line without moving DOT by executing a "0TT".

MACRO.PRG

MACRO is the system macro assembler. This is a discussion on how to run MACRO from the console, not a discussion on programming. Once the user has created an ASCII file that he wishes to assemble he runs MACRO. MACRO is a five phase assembler with overlays for phases 1 to 4. Phase 0 is the first phase. As each phase other than phase zero is called into the user's partition a message notifying the user that the phase is active is printed on the user's console. To run MACRO the user must be logged in under the disk and PPN that contains the file to be assembled. The system library file, SYS.MAC, resides on disk 0 in area [7,7]. MACRO will always look for SYS.MAC there when a program calls for it. To run MACRO the user types:

```
        MACRO  FILNAM.EXT   /OPTIONS
```

where FILNAM.EXT is the name of the file to be assembled. A default extension of .MAC exists if none is entered. There are nine option switches the user can select if he chooses. The presence of a slash informs the assembler that a list of options follows. Options are terminated by a CR. For now let's assume that no options were selected. The following sequence of events then takes place.

1) PHASE 0 processes the input string from the terminal and sets the appropriate option switches, if any.

2) PHASE 1 builds a symbol table and macro table in memory. If there are any symbol or macro errors then the total number of errors is reported at the end of the phase.

3) PHASE 2 builds an object file and reports the total number of assembly errors it detects. The object file is given a filename that is the name of the file being assembled with an extension of .OBJ. The file is placed on the disk in the user's area.

At this point if the program has no global symbol referrences (I.E. no INTERN or EXTERN type symbols) then the assembler calls in the PHASE 4 overlay which sets up and calls LINK.PRG to generate a program file on the disk. PHASE 3 is called in if one of the option switches that sets the list file switch was selected. PHASE 3 would occur in time between PHASE 2 and PHASE 4 if PHASE 4 was called. The list file generated by PHASE 3 has the same name as the file being assembled with an extension of .LST, and the file is either written on the disk or listed on the user's terminal depending on the option selected.

The option switches that are available are listed below. The user may use as many of them as he wishes. Note that some switches automatically select the list switch too, and the "B" switch must be the LAST switch selected if it is selected at all.

L	Creates a list file on the disk.
T	Creates a list file on the user's terminal.
E	Suppresses all but error lines in list file. Use with T or L.
O	Bypasses PHASE 1 and 2. Goes directly to PHASE 3. Sets the list switch on. Assumes a valid object file already exists.

C	Lists conditional assembly code. Sets L switch on.
X	Expands and lists all macro code. Sets L switch on.
H	Sets L switch on and selects HEX numeric output instead of OCTAL.
P	Forces program file generation by PHASE 4 unconditionally.
B	Sets bottom header. Sets L switch on. Takes all data on the input line between the "B" and the end of the line and appends it to the bottom of every page of the listing.

LINK.PRG

LINK is the program that generates a program file from one or more object files. The program file is written on the disk in the user's area. It has the same name as the FIRST object file in the list of object files to be linked, and an extension of .PRG. To run LINK the user must be logged in under the PPN that contains all of the object files to be linked. He then types:

```
        LINK  FILEA,FILEB,FILEC,...,FILEX
```

Each filename has a default extension of .OBJ. If the user cannot fit all of the object file names on one line then he can continue on the next line by terminating the last filnam in the first line with a comma. As many lines as the user needs can be chained in this manner. Errors are listed on the user's terminal, the most common error being a missing global definition somewhere. LINK also lists out the name of the object file being processed as LINK gets to it.

SYM.PRG

SYM runs exactly like LINK. The only difference is that SYM creates a symbol file instead of a program file. A symbol file has an extension of .SYM, and is used as an optional input to DDT.PRG. Otherwise, it is useless.

DDT.PRG

DDT is the system dynamic debugging and patching program. It allows the user to run his program and examine or alter program data or flow at any point in the program. To run DDT the user types:

```
        DDT FILNAM.EXT
```

where FILNAM.EXT is the name of the program file the user wishes to debug. DDT will default to an extension of .PRG if none is entered. Once the program is found DDT looks for the program and its symbol file in memory. If either is there it it deleted. DDT then looks for the program file on the disk. DDT aborts if the file is not there. If it is, DDT loads it into memory. DDT then looks for the program's symbol file and loads it too if it is on the disk. If it isn't DDT will create one in memory. DDT then looks for SYS.SYM in area [7,7] of disk 0. If it finds SYS.SYM it too is loaded. The presence of symbol tables allows DDT to print out program relative memory references, EXEC calls, and absolute memory references in EXEC symbollically instead of numerically. The symbol tables make life a little easier for the user to debug his program. they do not affect the program in any way.

Once DDT has all the garbage it can find in memory DDT prints the program's base memory address and size in bytes on the terminal. DDT also overrides the RUBOUT, CONTROL-U, CONTROL-S, and CONTROL-Q features of the EXEC. A CONTROL-C still acts the same. All the others are gone except RUBOUT which has a special meaning in DDT. DDT runs in image mode not line mode. Inputs are not usually terminated by a CR in DDT. A CR is a command to DDT. Once the user executes a DDT command or fills in any arguments a command requires DDT does its own terminal formatting. If the user enters data erroneously in DDT he can erase it with a single RUBOUT. A RUBOUT will erase ALL of the entry, not just the last character, and echoes an "XXX" followed by a tab on the terminal. ALTMODE selects a subset of the DDT command set and echoes as a dollar sign. Illegal DDT commands echo as a question mark plus a tab.

What follows is a list of DDT commands and a brief description of what they do. The user is encouraged to experiment. All numeric inputs are in octal.

SLASH (/)

Examine memory. Format is XXX/ where XXX is the numeric or symbollic address the user wishes to examine. The data is printed out symbollically as it would appear in a listing. When symbollic entry is used DDT allows the use of a +/- octal numeric offset for convenience. DDT will not display a location that is out of the program's boundaries when examining in program relative mode.

EQUALS (=)

When used after an examine it displays the contents of the address in octal.

CARRAIGE RETURN (CR)

When used after an examine (with or without an equals) it closes the examined location and echoes a CR/LF. The contents of the location are not altered unless the user enters new data prior to the CR. The new data may be either octal data or a symbollic op code expression. Op code entries are symbollically the same as the op code printouts with symbollic addresses allowed if the symbol table is in memory. Note that a symbollic op code entry may generate data for more than one location.

LINE FEED (LF)

Same as a CR except that DDT will open and display the next op code after closing the curent address(es). Note that symbollic op code displays will automatically advance the examine pointer to the next op code. The equals character forces the advance to a single word if no new data is entered.

UP-ARROW (^)

Same as a LF except that the previous address is opened for display. Note that backing up does not guarantee that the examine boundary will fall on a valid op code boundary.

AMPERSAND (@)

Indirect examine. Treats the contents of the last examined location or CPU register as a program relative address and displays it.

TAB

Treats the CONTENTS of the last examined location or CPU register as an absolute memory reference and displays it. All subsequent examines are considered absolute addresses instead of program relative addresses until an ALTMODE-R resets DDT to relative mode.

PERCENT (%)

Register examine. Format is: %XX= where XX is the CPU register the user wishes to display. The register argument must be from the following list: R0, R1, R2, R3, R4, R5, SP, PC. The display is in octal, and the register contents may be changed by entering new octal data followed by a CR.

ALTMODE-G ($G)

Allows the user to start the program being debugged at relative address 0. Echoes a tab after the $G, and accepts one line of input terminated by a CR. This line of input is passed to the program as it would be if the user entered it from the console. The PROCEED and SINGLE INSTRUCT commands are not legal until at least one $G has been entered. The user may execute an $G at any other time to restart the program if he wishes.

ALTMODE-B ($B)

Sets or lists breakpoints. There are 8 breakpoints in the DDT breakpoint table. They are numbered 0 to 7. this command accepts two arguments. The first is a numeric or symbollic program relative address that the user wishes to set a breakpoint at. It is placed just in front of the altmode. The other is the breakpoint number the user wishes to set or display. It is placed just in front of the B. The following list will explain all.

$B	Lists all active breakpoints in the table by number and symbollic or numeric address.
$XB	Lists breakpoint X if it is active.
TAG$B	Sets a breakpoint at address TAG. The first inactive table entry is used.
TAG$XB	Sets a breakpoint at address TAG. Table entry X is used whether it was already active or not.

ALTMODE-C ($C)

Clears one or all breakpoints from the table. Accepts two arguments in the same manner as $B with the effect as follows:

$C	Clears all active breakpoints from the table.
$XC	Clears breakpoint X if it was active.
TAG$C	Clears the breakpoint at address TAG if it exists.
TAG$XC	Same as $XC.

The user should know that DDT will not allow odd address arguments or breakpoint numbers greater that 7 for $B or $C.

ALTMODE-P ($P)

Proceed from last breakpoint. This command is only legal if a breakpoint has been reached in the program. When executed program operation resumes until another breakpoint is reached. This command accepts a 16 bit argument between the altmode and the P which will inform DDT as to how many times the last encountered breakpoint address must be reached before breaking again at that address. Only the last encountered breakpoint address may be set, and a default of one is used if no count is entered. The argument is an octal numeric.

ALTMODE-X ($X)
BACKSLASH (\)

These two commands are the same. The backslash is for convenience. This is the single instruct command. It is valid only after a breakpoint has been reached. It will execute one CPU op code and break on the next. Although it is legal for the user to single instruct through the EXEC it is strongly recommended that he avoid the temptation. Timing restrictions on I/O devices and the fact that single instruct mode disables interrupts may cause unspecified results.

ALTMODE-R ($R)

Resets examine mode to program relative. Used after a TAB command to get back to normal mode of operation.

ALTMODE-D ($D)

Translates the specified expression or address(es) into decimal and displays them. Accepts one of two arguments, but not both. One is the expression to translate and the other is the number of addressess to translate. The following table explains the formats.

$D	Displays the last examined address in decimal.
$XD	Displays X words in decimal starting at the last examined address.
EXP$D	Displays the decimal value of EXP. Uses as many words as necessary to display the full value. EXP can be a numeric, a symbol, or an op code expression.

ALTMODE-EQUALS ($=)

Same as $D except display is in octal. all other comments apply.

ALTMODE-M ($M)

Prints the absolute base address and the size of the program in bytes.

ALTMODE-ASTERISK ($*)

Displays the contents of the last examined location or CPU register in unpacked RAD50 format.

ALTMODE-DOUBLE QUOTE ($")

Displays the contents of the last examined address or CPU register as two ASCII characters.

ALTMODE-POUND SIGN ($#)

Displays the contents of the last examined address or CPU register as two OCTAL bytes. The low byte of the word is displayed first.

ALTMODE-A ($A)

Displays a string of bytes as an ASCII string of characters. Terminates when a null byte is found, and adjusts the examine pointer to the next even address. The formats are:

$A	Display from the last examined memory location.
TAG$A	Display from address TAG.

COLON

Allows the user to enter a label into the symbol table. The location so labeled must be in the program. Labels are, as usual, 1 to 6 alphanumeric characters long with the first always an alpha character. The format is LABEL: The value given to LABEL is the value of the last examined address. Once defined the label may be referenced symbollically by the user during program patching. Any program references to the labeled address can also be done symbollically.

DDT allows the user to expand the size of the program to accept program patches. Patches may be placed at the end of the program after the last valid location in the program. DDT will automatically expand the program to fit the patches. Program patching is done symbollically, and it even allows the use of labels. New labels are inserted into the symbol table as the user defines them. A symbol may not be referenced until the user defines it. The patch may also be called from the main part of the program symbollically after the patch has been made and if a label was used.

When the user is ready to exit from the program he executes a CONTROL-C. DDT will save the altered program and symbol table in memory. This allows the user to run SAVE.PRG if he wishes to save the altered program. As a word of caution, the user should not save a program that has been partially run. It is a good idea to debug the program once more, put it the patches, and save the program without running it. This insures that there are no data storage areas that have been altered from their initial conditions. If the program exits on its own while being run the user should NEVER save it if he used breakpoints anywhere in the program. Breakpoints are not cleared until the program goes back to DDT.

APPENDIX A - OPERATIONS HINTS

For the benefit of users who have never experienced working on or with a multi-user system or a floppy disk system, a few helpful hints are included herein. This is not meant to be a list of procedures to be followed religiously. It is intended to be illustrative of the types of problems a novice user might not be expecting. Since anything can happen, and usually does, these procedures should keep lost time and data to a minimum.

Consider for a momemt the variables that conspire to make life miserable for the unexpecting programmer or user. Large computer systems are based in dust free rooms with temperature and humidity well controlled. A.C. power is usually a dedicated circuit with little or no noise on it. Well trained personnel perform rigidly adhered to sequences that maximize system throughput and minimize errors. Now, what about your system? Where is it? In the basement? The garage? The den? Does it use the same A.C. circuit as the television or the blender or the plug that your wife uses when she vacuums the living room? Do you get the message? Expect the worst. Now, how do you protect yourself without spending all of your time protecting yourself? The following procedures, based upon lots of experience, are a basis for a system of survival. They are guidelines that should be tailored to meet the particular circumstances you find yourself in. The system's environment, the system's users, the number of consecutive users, and the importance of the data are all variables that must be considered. Eventually you will have a system of your own, either rigid or flexible. In the meantime here is one you can evaluate.

1) Never enter more data into a buffer via an EDIT insert command than you can see on the CRT. Remember, since the command buffer expands dynamically until you enter a double altmode, you need enough room in memory for both the command buffer AND enough room left over to insert the data into the data buffer. Besides, if you forget the leading "I", you won't know about it until you enter the double altmode.

2) Never edit for more than 30 to 40 minutes without exiting from EDIT. This saves the results of all your hard work on the disk. Besides, if there isn't enough room on the disk 'tis better to know about it before you create the whole thing. Once a system error aborts you from EDIT the content of the data buffer is lost, and I mean LOST!

3) NEVER CHANGE A DISK WITHOUT RUNNING SYSTAT! This insures that the bitmap in memory is the same one that goes with the disks you have on the system.

4) Every so often you should run DSKANA to make sure the disk is error free, and then run DSKCPY to back it up. Do this at least once a day if you do a lot of work. A command file is useful for this. The ":K" command allows you to stop the execution of a command file until you change disks. The EXEC will ignore an entry of CR only, and the message command (:<...>) allows you to give yourself instructions so you don't have to memorize what to do.

5) Backup disks should be of the grandfather, father, and son variety. At a minimum it means you need two backup disks for each system disk. The active system disk is the "son". The first backup is the "father", and the second backup is the "grandfather". When making a new backup leave the "father" alone, and do your backup on the "grandfather". This way if there is a system crash while you are doing a backup you haven't lost the whole kabaj. Date each backup as it is created. The dates determine the "age" of the backup.

6) Create a backup disk IMMEDIATELY after creating a new program and before you assemble it (or try running it under BASIC if it is that type of program). You never know what you may have created until you try it. Once the program sort of runs you can live a little more dangerously.

7) Know the other people on the system with you. If you don't trust their abilities, go away until they have left. If at all possible, reboot the system when you get back on.

8) NEVER leave the system without running LOGOFF, and change your password periodically. There are plenty of people in this world who delight in discovering and changing things like user passwords. They call themselves cute. We have other names for them, don't we?

9) Lock up your disks when you are not around. The less accessable they are to others the less chance there is of someone "borrowing" one to recover from an error he made.

10) Leave a copy of the system disk around in case someone does make a fatal error that clobbers it. The system disk should contain system programs, public command files, and public data (such as SYS.MAC) only. Only the system disk should contain users [1,1] to [7,7]. This is a security measure and a way to prevent a lot of confusion. Imagine what would happen if there were two [2,2] areas with different command files that had the same names? The disk that is assigned as disk 0 is the one that EXEC would look to when one of them was run. There is no reason to copy the system onto every disk unless one disk drive is all you have.

11) If one disk drive is all the system has then invest in a TARBELL cassette interface. Use PACK and UNPACK to save your data. Paper tape is too bulky and slow to be a reliable and efficient backup medium. Rubber bands and careless handling in general can destroy paper tape quite easily. If the cassette reader is functioning properly then cassettes are much less apt to become unreadable.

Don't get the idea that the system is so unreliable that it never stays up long enough to be worth running. This system has been tested under a variety of conditions. It is not unusual for it to run continuously for days without a fatal error. The above procedures are meant as pessimistic guidelines only. It is your data, how many chances are you willing to take with it?

APPENDIX B - SYSTEM BUS ERRORS

The following error conditions cause a system bus error via EXEC. A bus error causes a program abort, and can be recognized by the printout

```
BUS ERROR - PC @XXXXX
```

whrer "XXXXX" is the address of PC when the error occurred. The list of error conditions is:

A)	A memory time-out trap.
B)	A power fail trap via the HALT op code.
C)	A power-up after a recoverable power fail.
D)	A parity error trap.
E)	A reserved op code trap.
F)	An illegal op code trap (I.E. LEA or JSR with DM0).
G)	An XCT op code error trap.
H)	An XCT trace trap if not running DDT.
I)	A floating point error trap if not running BASIC.

The user who has the 16-bit to 8-bit S100 bus CPU boards should note that error conditions A to D cannot happen on his system. The boards and the systems they go into do not have the hardware to support them.

APPENDIX C

IMSAI 8080 PROCESSOR BOARDS

These boards are synchronized to the 8Ø8Ø CPU clock. They run, therefor, at 2 mhz. The boards use microm state codes "8" - "A" and "D" - "F" (see appendix D) to minimize memory multiplexing. Word reads and writes take 3 memory cycles. Byte reads and writes take only one. All memory operations are transparent to the user except for the time involved.

The I/O is another matter. The 8Ø8Ø I/O ports Ø - "FF" are mapped to memory addresses "FFØØ" - "FFFF" respectively. All I/O signals are generated by the hardware, but in order to maintain proper I/O operation two rules are followed:

1) Only byte reads and writes are done to I/O.
2) If a read-modify-write is attempted to an I/O port the read portion is ignored. Only the write is done.

These rules make for some interesting results; They are:

1) For byte operations only the appropriate byte, per address LSB, is gated through. The CPU places the same data on both halves of the DAL on a write. The I/O mapping hardware selects the lower byte in all cases. During a read the I/O mapping hardware also places the data on both halves of the DAL. The CPU selects one or the other per the address LSB.

2) For word operations only the lower half of the word is selected during an I/O write. During an I/O read the I/O data byte appears on both halves of the inputted word.

3) DM1 - DM7 addressing for format 7 and 1Ø op codes generates a read-modify-write sequence. Although some op codes do not use the data read (or rewrite the original data) microm space limitations require that all of them do a read-modify-write with DM1 - DM7 addressing. Only format 7 and 10 destination addressing modes 1-7 create read-modify-write sequences. All others, including format 1Ø SM1-SM7 addressing, do not. Since the hardware ignores a read-modify-write read cycle to I/O, the reading of an I/O port can only be done using the source field of a format 1Ø op code. Either word or byte ops can be used (see 1 and 2 above). When writing to an I/O port any format 7 op code or format 1Ø op code with DM1 - DM7 addressing can be used if it makes sense without the read. A few examples should help clarify this.

```
EX 1)   CLR @#"FFØ4"   ; I/O port 4 is cleared.

EX 2)   CLRB @ "FFØ4"  ; I/O port 4 is cleared

EX 3)   TSTB @# "FFØ5" ; won't work without the read.  Will write garbage
                                                       into port 5.
```

```
EX 4)  BIT #2@#"FFØ6"    ;won't work without the read; will write garbage
                                                      into port 6.

EX 5)  BIT @#"FFØ6,#2    ;works fine.  Read only to I/O.

EX 6)  CMP @#"ffØ8,RØ    ;won't work unless same data is
                        ; in both bytes of RØ

EX 7)  CMPB @#"FFØ9",R1    ;works fine.  Lower half of R1
                          ; compared to data read from port 9.

EX 8)  MOV RØ@#"FF1Ø"   ;Lower half of RØ written into I/O
                       ; port "1Ø"

EX 9)  MOVB RØ,@#"FF1Ø"  ;same as example 8.

EX 10) BISB #2,@#"FF11"  ;won't work.  Requires a read-
                        ;  modify-write for proper operation.
                        ;  writes garbage into port "11".
```

To overcome the type of problem with I/O that appears in example 1Ø above the user must first read the I/O port, set the bit on, and then write it back. This limitation of no read-modify-writes is an I/O limitation only. It is engineered in to prevent unspecified results from occurring in those I/O devices that use the same port for two different functions.

See the manual that comes with boards for information on:

1) Power up/halt option selection.
2) User bootstrap address selection
3) 7 channel DMA/interrupt priority selection
4) 8Ø8Ø to AM100 CPU switching
5) Valid front panel operations in AM100 mode
6) Available software for the AM100

With the 8080 processor boards IEN sets the interrupt enable immediately, and IDS disables it immediately. IEN is therefore an instruction that can be interrupted at its conclusion, and IDS cannot be interrupted. Usually the I2 flag is not set or cleared until the next instruction begins execution. That is not the case with this processor. WF1 also sets 12 immediately, but this is true of all processors that use the AM100 microprocessor chip set, not just this one.

JUMPERS

There are jumpers on both CPU boards that the user must become familiar with before his system will run. Most of the options have been preselected to run the floppy disk DOS software, and references will be made to them when appropriate.

CPU BOARD #1

The socket at 1J3 selects the power up, halt, and external address register options. The pins are set up as follows:

PIN #	FUNCTION
1	connected to a pad immediately below the pin.
2	connected to a pad immediately below the pin.
3	external address register bit 3
4	external address register bit 1
5	external address register bit Ø
6	external address register bit 5
7	external address register bit 2
8	external address register bit 4
9	halt option jumper #1
10	power up option jumper #1
11	power up option jumper #2
12	halt option jumper #2
13-16	grounds

Pins 3-12 have 4.7K pull up resistors to +5 volts. They are, therefore, jumper-to-zero (ground) selects for ones. The socket is wired at the factory to provide power up option Ø, halt option Ø, and the address required for the system floppy disk drive(s).

CPU BOARD #2

There is one jumper socket and four individual jumpers on this board. One of the individual jumpers is just to the left of 2Z47, and is labeled "NO 8080". This jumper is a ground enable for I.C. at 2Z47. When installed, as it is at the factory, it provides clocks to the bus that mimic the clocks generated by the 8080 CPU board. It is to be installed <u>only</u> if the 8080 CPU is <u>not</u> on the bus.

The second jumper is just to the right of socket 2J3, and is labeled "SW1". It is designed for a <u>optional</u> switch or a jumper. This switch, or jumper, when enabled will ground $\overline{\text{PHOLD}}$ on the bus. This will tri-state the 8080 CPU, if it is present, and enable the AM100 CPU. When the jumper is out the AM100 CPU is disabled via the DMA control logic (at level 7), and the 8080 CPU is enabled. A word of caution is in order for those who have DMA controlled devices that get control of the bus via PHOLD. A wiring change will be needed on those devices to run on the AM100 CPU. The AM100 CPU has provisions for up to 7 prioritized DMA devices. Each device has a DMA request pin (J104 to J1-10) and a DMA acknowledge pin (J1-56 to JI-62). The request lines are jumper selected to the left of 2Z42, and the acknowledge lines are jumper selected to the right of 2Z42. More on these jumpers later.

The seven DMA lines are prioritized with Ø having the lowest priority, and 6 having the next to highest. Level 7, the highest level, is reserved for CPU select via PHOLD. DMA level 7 can be jumpered to J1-63 to allow other system boards to check and see which CPU has control of the bus. Associated with the DMA lines is a signal called $\overline{\text{DMARCVD}}$ which goes low one clock time before a DMA grant line goes low. Any DMA grant causes this line to go low. This signal informs all DMA devices that a DMA request is about to be honored. To avoid switching DMA grants to a higher request while any grant is active, no DMA device should issue a request while this line is low.

The third individual jumper is associated with $\overline{\text{DMARCVD}}$. It is located just below 2Z31, and is labeled "DMAG". When the jumper is installed it ties the signal to J1-64 (if you haven't already figured it out, J1 is the 100 pin bus connector).

The fourth jumper is just below Socket 2J3, and is labeled "HALT". It is tied to 2R2 and 2J3-16. This is where the halt switch goes if one is used. The halt line (13) is disabled when the jumper is in. Note that this is different than the other jumpers. This line is tied to ground via 2J3-16 to 2J3-1 at the factory to disable the halt switch.

The socket at 2J3 has one input pin, three output pins, and four grounds connected to it. All the other pins are not used. The pins are set up as follows:

PIN #	FUNCTION
1,2,3,8	grounds
7	real time clock input from 2Z5-7
10	real time clock output to 2Z23-3
11	processor select. Goes to 2R1 and the processor select switch (SW1).
16	halt. Goes to 2R2 and the halt switch jumper.

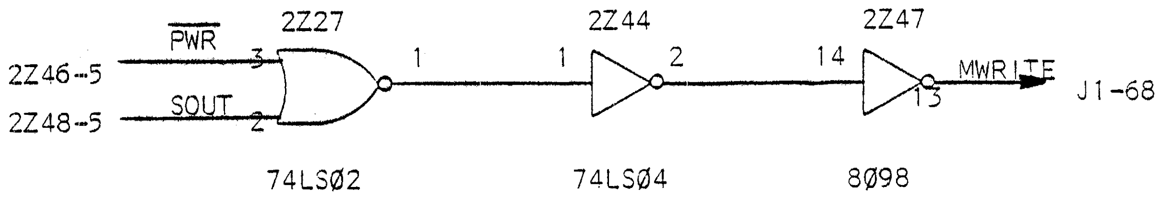
Socket 2J3 is wired at the factory as follows:
2J3-1 to 2J3-16   2J3-7 to 2J3-10   2J3-8 to 2J3-11
This selects the AM100 CPU at all times, disables the halt and selects the on-board real time clock. Note that this clock must be present at 2J3-10 from either the on-board clock or from an external source. If no real time clock is desired then 2J3-10 must be grounded. The clock signal at 2J3-10, if used, must be of sufficient duration to set the 7474 flop at 2Z23, but need not be any longer. 2Z23 is reset by the CPU when the interrupt is honored.

Immediately to the left of 2Z42 is a series of jumper sets labeled 0-7. Six of them are sets of three, and one is a set of two. These jumpers select lines J1-4 to J1-11 as either interrupt lines or DMA lines. Each line can be either a DMA request line or an interrupt request line, but not both. They select DMA levels 0-6 or interrupt levels 0-7 (DMA level 7 is preselected as CPU select). The center set of eight holes are connected to J1-4 (level 0/) to J1-11 (level 7). The right hand set of holes are the interrupt selects. The left hand set of holes are the DMA selects. The interrupts and DMA lines are both prioritized in the same way.

The highest priority interrupt that is active generates a 3-bit code that is placed on the CPU bus when an interrupt acknowledge is executed. The code goes to bus lines 1-3, and a zero is placed on line 4. This gives eight levels of interrupts out of a possible total of 16. The vectored interrupt selects the first work in the table for level Ø, and the eighth word in the table for level 7. Note that the interrupt lines do not have corresponding acknowledge lines as the DMA requests do. DMA acknowledge is a hardware function, and interrupt acknowledge is a software function. Whenever a DMA request line is connected, the corresponding DMA acknowledge line must also be connected. The jumpers for the DMA acknowledge lines are to the right of 2Z42. They tie the acknowledge lines to J1-56 (level 7) through J1-63 (level Ø).

There is a signal generated by the IMSAI 8080 front panel as an option that is used by some memory and peripheral boards for writing to memory. The signal is called MWRITE. If the front panel is not on the bus then this signal must be generated some other way. The following circuit can be installed on board #2 to generate MWRITE.

$$\text{MWRITE} - \overline{\text{SOUT}} \cdot \text{PWR}$$



Note that the 8Ø98 I.C. at 2Z47 is ground enabled for the gate that is used in the above circuit. The above circuit must be disconnected, therefore, if the front panel is reinstalled.

While we are on the subject of the front panel, it should be noted that only the following front panel operations are legal when the AM100 CPU is on the bus and has bus access: RESET, EXT. CLR. RUN, STOP, and SINGLE STEP. The single step function is really a single memory or I/O byte access, not a single instruct.

The pull-up resistors that are on the data in bus on the 8080 CPU are not replicated on the AM100 CPU. The data in bus will float if an access to a nonexistent memory location or I/O port is made. The value of the returned data cannot be guaranteed.