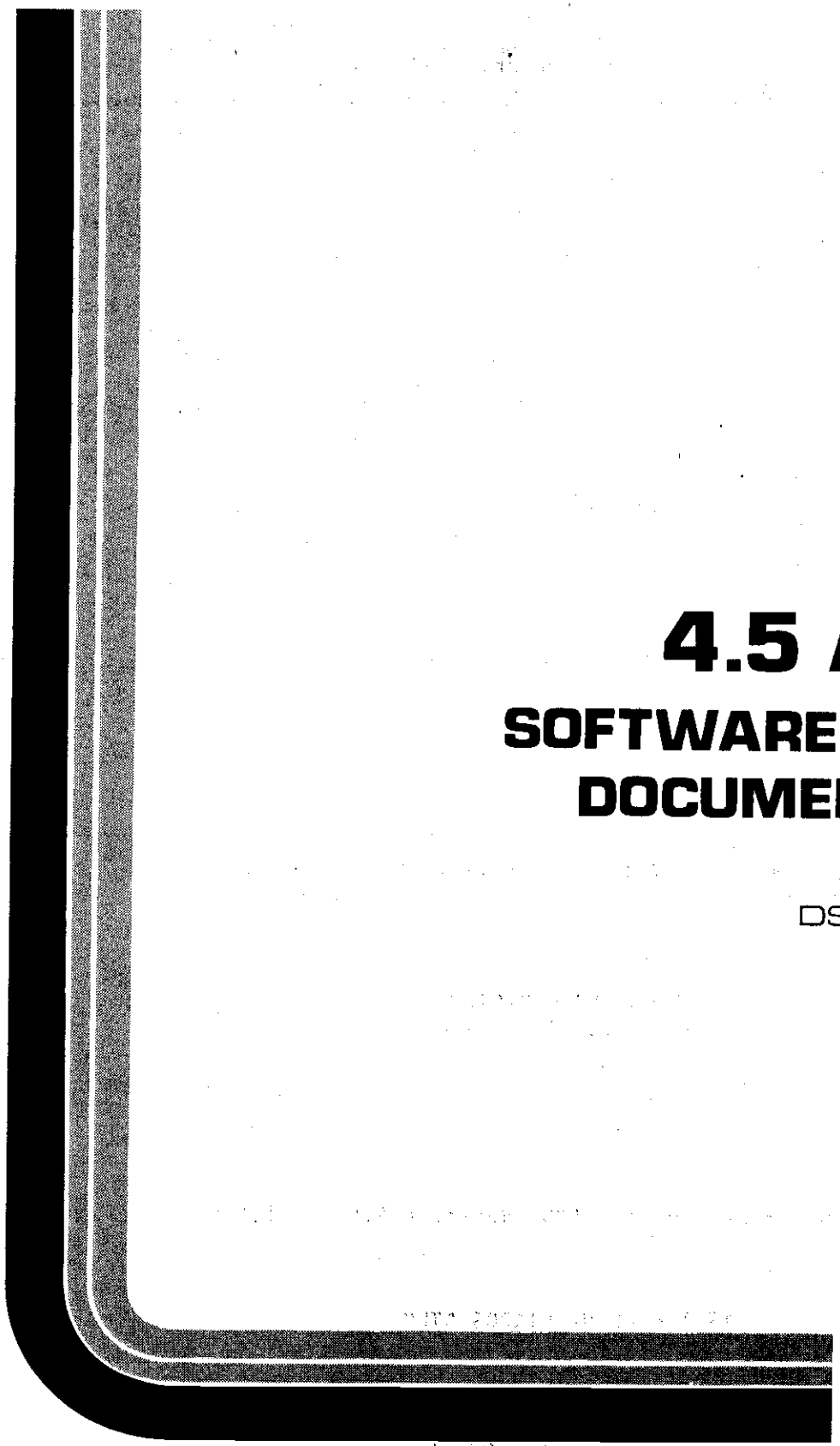

4.5 AMOS
SOFTWARE UPDATE DOCUMENTATION

DSS-10000-05

alpha micro

AMOS 4.5 SOFTWARE UPDATE DOCUMENTATION PACKET

These documents reflect AMOS Versions 4.5 and later

ALPHA MICROSYSTEMS
17881 Sky Park North
Irvine, CA 92714

April 1981
AMOS Release 4.5

MASTER TABLE OF CONTENTS
FOR THE AMOS SOFTWARE UPDATE DOCUMENTATION PACKET

BASIC Programmer's Information Section:

BASORT - BASIC Subroutine for Sorting Random and Sequential Files	Revised
COMMON - BASIC Subroutine to Provide Common Variable Storage	Unchanged
FLOCK - BASIC Subroutine to Coordinate Multi-user File Access	Revised
SPOOL - BASIC Subroutine for Spooling Files to the Line Printer	Revised
XLOCK - BASIC Subroutine for Multi-user Locks	Revised
XMOUNT - BASIC Subroutine to Mount a Disk	Unchanged

(For a complete list of Alpha Micro software documentation, see A Guide to the Alpha Micro Software Documentation Library.)

April 1981

AMOS VERSION 4.5 RELEASE NOTES

This document describes the changes and additions that have been made for Version 4.5 of the Alpha Micro system software. It describes only those changes and additions made since the last release (4.4B); you may find general system documentation in the appropriate manuals. (The document A Guide to the Alpha Micro Software Documentation Library in this documentation packet lists all software documentation available from Alpha Micro.)

This release contains a number of major new programs. Besides the new Alpha Micro Electronic Mail System, AlphaMAIL, this release also includes a terminal driver building program, many enhancements to the assembly language programming system (including the addition of an object file library generator), and new features in the AlphaBASIC programming system (including two new file modes). See Section 3.0, below, for information on the changes for this release.

Please read the documentation supplied with this release before converting to AMOS Version 4.5. (NOTE: If you are currently running under AMOS Version 4.3 or earlier and are using a disk that runs under control of the AM-410, it is very important that you read AMOS Version 4.4 Method of Handling Bad Disk Blocks in the "System Operator's Information" section of this documentation packet before running any of the 4.5 software. AM-410 users must be aware of the fact that some software contained on pre-4.4 Releases of AMOS is not compatible with AMOS Version 4.5 because of the changes in the bad block handling.)

1.0 THE RELEASE MECHANISM

Because of the large amount of software now available from Alpha Micro, we cannot distribute it all on one floppy disk. We are now providing all the software we supply on one 5-megabyte or one 15-megabyte hard disk which is automatically updated for each release. For floppy disk users, things are a bit more involved. With each of the releases you will receive a new System Disk. This will contain all of the standard system software. To receive the additional software, you must specifically order one of the additional diskettes. These diskettes are divided up as follows: a LISP/PASCAL diskette containing the LISP and PASCAL programs and their associated files; a Driver Source Diskette containing sources to device, terminal, and interface drivers; and a Miscellaneous Program Diskette containing various BASIC and MACRO programs. You may order these disks through your dealer. See the separate document dealing with disks, Disks Available from Alpha Micro in this packet.

2.0 DOCUMENTATION INCLUDED WITH THE 4.5 RELEASE

Included with this release is the standard AMOS Software Update Documentation Packet.

The following new manuals and documents have been issued since Release 4.4:

AlphaMAIL User's Manual, (DSS-10000-06): This book describes the use and installation of AlphaMAIL, the Alpha Micro Electronic Mail System.

AlphaVUE/TXTFMT Training Guide, (DSS-10000-03): This book introduces the new AMOS user to AlphaVUE, the screen-oriented text editor, and TXTFMT, the text formatting program.

Change Page Packet #2 for the "AlphaBASIC User's Manual", (DSS-10000-07): These change pages update the BASIC manual for Release 4.5.

Change Page Packet #1 for the "AlphaPASCAL User's Manual", (DSS-10000-10): These change pages contain additional information about AlphaPASCAL Version 2.0.

Change Page Packet #2 for the "AMOS System Commands Reference Manual", (DSS-10000-09): These change pages update the system command reference sheets for Release 4.5.

In addition, the following manuals have been revised for AMOS Release 4.5:

AlphaFIX User's Manual, (DWM-00100-69, Revision A01).

AMOS Assembly Language Programmer's Manual, (DWM-00100-43, Revision B00).

AMOS Monitor Calls Manual, (DWM-00100-42, Revision B00).

ISAM System User's Guide, (DWM-00100-06, Revision A02)

TXTFMT User's Manual, (DWM-00100-07, Revision B00).

We provide a complete list of the documentation applicable to this release in A Guide to the Alpha Micro Software Documentation Library, in this packet. NOTE: You may order a four-binder set of Alpha Micro documentation that includes three volumes of software documentation and the one-volume Alpha Micro Integrated Systems User's Guide, by ordering part number PDB-00001-00 from the Alpha Micro Sales Order Department. (This set includes all Alpha Micro software documentation except the AlphaLISP User's Manual.) You may also order individual manuals by their own part numbers.

Also, notice the Master Table of Contents that lists all documents in the AMOS Software Update Documentation Packet; this is the first document in this packet.

3.0 NEW SOFTWARE FOR RELEASE 4.5

AMOS Release 4.5 features several major new programs:

AlphaMAIL - AlphaMAIL, the Alpha Micro Electronic Mail System allows users on an Alpha Micro computer system to exchange mail in the form of AMOS files. Messages sent can take the form of memos, letters, reports, random data files, or files containing binary data. The AlphaMAIL Operator can send one message to multiple users, and can specify how many days the system will hold the message. For more information, see the AlphaMAIL User's Manual, (DSS-10000-06).

NEWTRM - To help our users build their own terminal drivers for terminals that are not currently supported by Alpha Micro, we have developed NEWTRM. This interactive program asks questions about the characteristics of the terminal you need a terminal driver program for, and then produces the assembly language source (.MAC) file for the appropriate driver. (NOTE TO FLOPPY DISK USERS: NEWTRM is a PASCAL program-- you must have AlphaPASCAL Version 2.0 on your system in order to use it.) For more information, see the document Building a New Terminal Driver (The NEWTRM Program) in the "System Operator's Information" section of the AMOS Software Update Documentation Packet.

LIB - As one part of the general enhancements made to the AMOS assembly language programming system (discussed in Section 4.0, below), this release features the object file library generator, LIB. LIB allows you to define libraries of object file routines which all assembly language programmers on your system can make use of. LIB allows you to modify as well as create library files. For information on using LIB, refer to Revision B00 of the AMOS Assembly Language Programmer's Manual, (DWM-00100-43).

Magnetic Tape File Backup - The magnetic tape file backup system consists of the programs FILTAP, TAPFIL, and TAPDIR. These programs allow you to perform disk file-oriented backup on a magnetic tape unit connected to an AM-600 Magnetic Tape Formatter Interface. You may back up and restore random files, sequential files, and multiple disk surfaces on a single tape. For information, see the document The Magnetic Tape File Backup Programs in the "System Operator's Information" section of the AMOS Software Update Documentation Packet.

For the first time on a general release, we are releasing the AM-120 Auxiliary I/O Controller support software-- CAL120, TIME, DATE, and the AM-120 driver, AM120.DVR. The AM-120 board contains several features including a clock/calendar with battery backup, power fail detection and handling, two serial ports, and three eight-bit parallel output ports and three eight-bit parallel input ports.

For information, see the Software Installation Instructions for the AM-120 in the "System Operator's Information" section of the AMOS Software Update Documentation Packet.

This release also contains software support for the Alpha Micro AM-710 128K byte memory board which detects and reports parity errors. The PARITY program enables parity error detection for this memory board. For information on PARITY, see Software Installation Instructions for the AM-710 Memory Board in the "System Operator's Information" section of the AMOS Software Update Documentation Packet.

4.0 ENHANCEMENTS TO EXISTING PROGRAMS FOR RELEASE 4.5

In addition to the new programs discussed above, this release also contains many enhancements to existing software:

4.1 The Monitor

The monitor now provides a new output control command, Control-R. When you type a Control-R, the monitor repeats back the current contents of the input buffer. This is extremely useful when you type several lines of characters ahead while your terminal is displaying program output and then want to remember exactly what you entered. (Or, if you are using a printing terminal, you will find this command useful when repeated rubouts have obscured the text.)

Another change in the 4.5 monitor allows command files to accept lower case characters as well as upper case.

Finally, several new monitor calls have been added:

AMOS	Executes AMOS commands without exiting current program.
FMARKR	Read in reverse to find file mark on specified magnetic tape unit.
JWAITC	Sets your job into the wait state.
LCS	Converts one character in R1 to lower case.
OPENA	Opens a logical dataset for appending.
PCALL	Invokes program as subroutine.
RLSE	Releases control of a semaphore and allows waiting job to access source.
RQST	Requests control of a semaphore to access source or to wait in wait chain.
UCS	Converts one character in R1 to upper case.
WAKE	Wakes a job out of sleep state.

See Revision B00 of the AMOS Monitor Calls manual, (DWM-00100-42) for more information.

4.2 The AlphaBASIC Programming System

This release includes two versions of the AlphaBASIC run-time package: RUN.PRG and RUNSML.PRG. RUNSML.PRG is much smaller than RUN, but is identical to it except that it does not support the trigonometric functions or the EXP, LOG, LOG10, FACT, or exponentiation (^) operations. Since many business packages do not make use of those mathematical functions, many users will be able to save a considerable amount of memory (over 1000 bytes) by using the smaller RUNSML rather than the standard RUN program to execute BASIC programs.

The only time you will see a difference between using RUN or RUNSML is if your program makes use of the functions not supported by RUNSML. If you are using RUN.PRG, such functions will execute normally; if you are using RUNSML.PRG, you will see the error message:

?Unsupported function

For compatibility with existing command files, users who do not need the extra mathematical functions may want to rename RUNSML.PRG to RUN.PRG. Before doing so, however, you will probably first want to rename RUN.PRG to a new name (e.g., RUNOLD.PRG) to save it in case you need it in the future.

Two new file modes have been added to COMPIL and BASIC: FORCED'RANDOM and APPEND. FORCED'RANDOM mode was added to aid those of you who are writing applications that use file locking to permit users to concurrently update the same files. If you open a random file in FORCED'RANDOM mode, every time your program READs a file record, BASIC will force a disk access even if that record is already in memory, and every time your program WRITEs a logical record, BASIC will force a disk write operation even if the buffer is not full. FORCED'RANDOM mode ensures that a record retrieved by your program contains the latest updates to that record.

APPEND mode has been added to make the use of sequential files more convenient. If you open an existing sequential file in APPEND mode, BASIC will position the file pointer to the end of that file and allow you to write information to the end of that file.

This release of BASIC contains a new error trapping procedure for a program interrupt caused by a Control-C. If a Control-C has been trapped by an error handling routine, the routine RESUME statement will cause the program to resume to the line following the one that was interrupted by the Control-C.

Other new features for COMPIL include:

If an error occurs during program compilation, COMPIL will not produce a .RUN file.

COMPIL now supports the use of Include Files. That is, COMPIL can fetch source code from a specified file while you compile another program file, and insert it into your compiled program file.

COMPIL optionally reports a message if it encounters any unmapped variables in your program file. (This allows you to make sure that all variables were defined via MAP statements.)

And, finally, considerable work has been done to enhance AlphaBASIC's error detection and reporting. For example, instead of stacking and discarding some errors, COMPIL now reports errors as soon as they are encountered. A variety of syntax errors that earlier were unreported are now detected and reported.

4.3 The Alpha Micro Assembly Language Programming System:

The assembly language programming system has been considerably enhanced and changed for this release. See Section 1.1 of the AMOS Assembly Language Programmer's Manual, (DWM-00100-43) for more information on the features added to the AMOS assembly language programming system for Release 4.5. (This manual has been completely rewritten for this release.)

4.3.1 MACRO

MACRO provides an optional symbol cross reference listing as part of the standard assembly listing.

MACRO supports the use of local symbols.

New psuedo opcodes allow you to: cause undefined symbols to be automatically EXTERNed; modify the name of MACRO output disk files; enable and disable output to the listing file; enable and disable symbol output to the cross reference listing; and end macro expansion.

Additionally, MACRO now supports a parameterized assembly option that allows you to specify a value on the MACRO command line that can be examined during the assembly process (using the new pseudo opcode NVALU). This is especially useful when used with the conditional assembly directive pseudo opcodes.

4.3.2 LINK and SYMBOL

LINK and SYMBOL have been rewritten for Release 4.5. Their output display has been changed to provide more information, and their functions have been considerably expanded.

Both LINK and SYMBOL accept a large number of option requests that allow you to do such things as: generate a load map file that indicates how the linked files will be loaded into memory; include equated symbols in the symbol table file; and, specify a library file, an optional file, or a required file.

4.3.3 DDT and FIX

With this release, DDT and FIX both support local symbols.

4.4 TXTFMT

To increase the flexibility of TXTFMT so that it suits your particular documentation needs, we have added four new commands to TXTFMT: /HEADER NO EMBED, /HEADER EMBED, /NUMBER HEADER, and /NO NUMBER HEADER.

When TXTFMT encounters section titles that are level three or deeper, it brings the next line of text up to the same line as the section header, separating the header from the text by a hyphen. (That is, it embeds the header in the surrounding text.) The /HEADER NO EMBED (/HNE) command disables this default format, and tells TXTFMT not to embed these headers. The /HEADER EMBED (/HE) command re-enables header embedding. (NOTE: Section titles are specified via the /HEADER LEVEL n command.)

The /NUMBER HEADER (/NMH) command tells TXTFMT to number pages with section oriented numbers. For example, if the text at the top of the page is in Section 4.0, and the page is the 25th page in that section, the page number for that page is: 4-25. The /NO NUMBER HEADER (/NNMH) command disables section-oriented page numbering.

Other enhancements include: the /FOOTER command accepts leading spaces and change bars work with unformatted text, underscores, and lists. See the TXTFMT User's Guide, (DWM-00100-07) for more information. (This manual has been completely rewritten for this release.)

4.5 Miscellaneous:

The generalized terminal driver program, TRM.DVR, no longer has to be loaded into system memory. (In previous releases, your system initialization command file had to include a SYSTEM TRM.DVR[1,6] command line to include the TRM driver in system memory.) NOTE: However, If your BASIC programs access the TRM driver, the driver must be loaded into system memory since BASIC performs its own memory allocations in user memory.

Because of changes to AlphaBASIC, SYSTAT now displays the name of the BASIC program a user is running, rather than just "RUN".

HASHER now accepts input in the same format as the DSKCPY command. See the HASHER reference sheet in the AMOS System Commands Reference Manual, (DWM-00100-49) for more information.

The APPEND command extension defaults have been changed to make the command easier to use. (The input file extension default is now the output file extension.)

LOAD has been changed to use the appropriate extension as the default if you specify a file in an ersatz device. (For example, if you say "LOAD

BAS:NEW", LOAD will load into memory NEW.BAS, since you have specified BAS:, the BASIC Library account.)

PDLFMT now accepts apostrophes within label names (thus making PDLFMT more useful for designing BASIC programs, since AlphaBASIC also allows apostrophes within variable names).

FORCE now checks to see if a job is guarded before forcing input to that job; if the job is guarded, the forcing job receives an error message and the operation aborts. (NOTE: A job "guards" itself by using the SET GUARD command.)

5.0 PROGRAMS INCLUDED WITH VERSION 4.5

The following describes all of the standard system software as of AMOS Version 4.5. FLOPPY DISK USERS NOTE: Not all of the programs described below are included on the standard 4.5 System Diskette. Due to space limitations, some of the programs have been placed on separate diskettes. Any such programs are noted in the descriptions below. See Disks Available from Alpha Micro for information on the separate diskettes. The AM-500 System Disk (a 5-megabyte pack) and the AM-410 System Disk (a 15-megabyte pack) contain all of the software described below, plus additional software described in the Disks Available writeup.

Programs in Account [1,4]:

AMSORT.SYS	Generalized sort module used by SORT.PRG and BASORT.SBR. This module must not be run directly.
APPEND.PRG	Program to append sequential data files.
ASCDMP.PRG	Dumps physical disk blocks in ASCII onto the user's terminal.
ATTACH.PRG	Attaches terminals to jobs.
BADBLK.PRG	Lists the contents of BADBLK.SYS, which contains disk certification data.
BASIC.PRG	The interactive AlphaBASIC compiler.
BAUD.PRG	Sets interface baud rate. Used with AM-300 and AM-310 only.
BITMAP.PRG	Defines disk-type device bitmap size during system startup.
BMVR.PRG	Programs 2708 EPROMs using a CROMEMCO Bytesaver board.
CDC210.PRG	Bootstrap program for the AM-210/CDC combination.
CLKFRQ.PRG	Defines the line clock frequency during system startup.
CPMCPY.PRG	Copies files between CP/M floppy diskettes and the AMOS system.
CPMDIR.PRG	Allows you to see the directory of a C/PM floppy diskette.
COMPIL.PRG	The disk-based AlphaBASIC compiler.
COPY.PRG	Copies files between devices.
CREATE.PRG	Creates a contiguous file.
CRT410.PRG	Certifies disks running on the AM-410 Hard Disk Controller.
DATE.PRG	Sets or displays the system date. (Used by AlphaAccounting, the line printer spooler, and the AM-120 Auxiliary I/O Controller.)

DDT.PRG	Symbolic debugger for assembly language programming.
DEL.PRG	Deletes memory modules.
DEVTBL.PRG	During system startup, defines the devices connected to the system. During normal operation, lists the connected devices.
DIAG2.PRG	Floppy disk drive and controller diagnostic.
DIAG3.PRG	Memory diagnostic (allows start and end address to be user specified).
DIAG4.PRG	Memory diagnostic.
DING.PRG	Program to sound the terminal bell. (Used in command files.)
DIR.PRG	Lists directory of files on a device.
DIRSEQ.PRG	Alphabetizes a directory by filename.
DO.PRG	Parameterized command file processor. (Processes DO files.)
DSKANA.PRG	Disk analyzer program. Checks the directory structure on a device for errors.
DSKCPY.PRG	Copies a literal disk image from one drive to another.
DSKDDT.PRG	Octal debugger for physical disk blocks.
DSKDMP.PRG	Dumps the contents of a physical disk block on the user's terminal.
DSKFIL.PRG	Lists the physical disk blocks occupied by a sequential file.
DSKPAK.PRG	Packs the contiguous files on a disk.
DUMP.PRG	Dumps memory, bitmaps, directories, etc., in octal or hex.
DYSTAT.PRG	Dynamic system status display for use with a VDM-1 board.
EDIT.PRG	Character-oriented text editor.
EMC.PRG	Support program for the AlphaMAIL system. Should not be run directly.
EPO.PRG	Support program for the AlphaMAIL system. Should not be run directly.
ERASE.PRG	Erases disk files.
EXIT.PRG	Exits a command file prematurely.
FILCOM.PRG	Binary file comparison program.
FILDMP.PRG	Dumps a disk file in octal on the user's terminal.
FILTAP.PRG	Copies disk files to a magnetic tape unit running under control of the AM-600 Magnetic Tape Formatter Interface.
FIXDVR.PRG	Configures a floppy disk driver for the proper controller, density, and device.
FIXMTM.PRG	Configures the MTM printer driver.
FIX.PRG	Screen-oriented, symbolic debugger for assembly language programs.
FLTCNV.PRG	Floating-point conversion module used by various AlphaBASIC XCALL subroutines. This module must not be run directly.
FMT200.PRG	Formats disks on the AM-200 floppy disk subsystem.
FMT210.PRG	Formats disks on the AM-210 floppy disk subsystem.
FMT400.PRG	Formats disks on the AM-400 disk subsystem.
FMT500.PRG	Formats disks on the AM-500 disk subsystem.
FORCE.PRG	Forces commands into a terminal input buffer.
GLOBAL.PRG	Produces a cross-reference listing of global symbols used within assembly language programs.
GOTO.PRG	Performs branching within command or DO files.
HASHER.PRG	Generates a disk hash total.

HEDLOD.PRG	Defines the head load time for floppy disks during system startup.
HELP.PRG	Displays brief instructions concerning various programs.
HWKLOD.PRG	Bootstrap program for the AM-500.
IBMCPY.PRG	Program to read IBM-3740 format diskette.
IBMDIR.PRG	Program to list the directory of an IBM-3740 format diskette.
ICMLOD.PRG	Bootstrap program for the ICOM floppy board.
ISAM.PRG	The Indexed Sequential Access Method control module. Used by AlphaBASIC and others. This module must not be run directly.
ISMBLD.PRG	Builds ISAM files.
ISMCOM.PRG	Compresses the top level of ISAM files.
ISMDMP.PRG	Dumps the data from an ISAM file.
ISMFIX.PRG	Converts Version 4.2 ISAM files to AMOS Versions 4.3 and later format.
JOBMEM.PRG	Defines a job's memory partition during system startup on a system that uses bank switching for memory management.
JOBPRI.PRG	Sets a job's scheduling priority.
JOBS.PRG	During system startup, defines the jobs to be available on the system. During normal operation, lists the user's job name.
KILL.PRG	Aborts a specified job.
LABEL.PRG	Changes or displays disk labels.
LIB.PRG	Object file library generator for assembly language programs.
LINK.PRG	The linking loader for assembly language object files.
LISP.PRG	The AlphaLISP programming language. (Available on the AlphaLISP/AlphaPASCAL diskette.)
LOAD.PRG	Loads files into memory.
LOG.PRG	Allows the user to gain access to the system, and to move between disk accounts.
LOGOFF.PRG	Allows the user to leave the system.
LOOKUP.PRG	Performs test for valid file specification within command files.
LPTINI.PRG	Sets up the line printer spooler. Should be run only during system startup.
LPTSPL.PRG	The line printer spooler program. Should not be run directly.
MACRO.PRG	The macro assembler.
MAC0.OVR	Overlay used by the macro assembler.
MAC1.OVR	Overlay used by the macro assembler.
MAC2.OVR	Overlay used by the macro assembler.
MAC3.OVR	Overlay used by the macro assembler.
MAC4.OVR	Overlay used by the macro assembler.
MAC5.OVR	Overlay used by the macro assembler.
MAKE.PRG	Create an empty sequential file in preparation for use with EDIT.PRG.
MAP.PRG	Lists the contents of the user's memory partition.
MDO.PRG	Parameterized command file processor used by the operating system. This module must not be run directly.
MEMDEF.PRG	During system startup, defines the memory management banks for a system that bank switches memory. During normal

	operation, prints the memory configuration of the system on the user's terminal.
MEMERR.PRG	Initializes memory error detection on the Piiceon SuperMem memory board. Also initializes the AM-710 memory board.
MEMORY.PRG	Used to allocate memory partitions on a system that does not use memory management. Also displays the user's current memory allocation.
MODFLG.SYS	Support module for AlphaMAIL. Cannot be run directly.
MODIAG.PRG	Support program for AlphaMAIL. Should not be run directly.
MONGEN.PRG	Links device drivers into the system monitor to generate a new system monitor.
MONTST.PRG	Boots a specified system monitor with a specifed initialization file. Used for system testing.
MOUNT.PRG	Mounts a device, making it available for use.
MTSTAT.SYS	Used by the mag tape software to store tape drive status.
PARITY.PRG	Enables parity error detection for AM-710 memory boards.
PASS.PRG	Allows the user to change his own password.
PAUSE.PRG	Temporarily pauses within a command file. The command file may be continued by use of the CONT program.
PDLFMT.PRG	Program Design Language, a utility used to help design projects.
PERLOD.PRG	Bootstrap program for the AM-200, PERSCI combination.
PPN.PRG	Lists PPNs defined on a disk.
PRINT.PRG	Sends a print request to the line printer spooler.
PRINTR.INI	Sample parameter file for the LPTINI program.
PULSE.PRG	Support program used by AlphaMAIL. Should not be run directly.
QDT.PRG	Octal debugger which works with absolute memory locations.
QUEUE.PRG	During system startup, defines the number of additional monitor queue blocks to be allocated. During normal operation, displays the number of free queue blocks available.
RAZA.PRG	Random read-write test for the AM-500.
REDALL.PRG	Test program which reads all blocks on a device.
RENAME.PRG	Utility to rename files.
REVIVE.PRG	Program to wake up a job which has been suspended. (See SUSPND.PRG.)
REWIND.PRG	Rewinds mag tapes.
RNDRED.PRG	Test program which does random seeks and reads on a device.
RUN.PRG	The AlphaBASIC Runtime package.
RUNSML.PRG	Version of the AlphaBASIC Runtime package that does not contain several mathematical functions: smaller than RUN.PRG.
SAVE.PRG	Saves a memory module on the disk.
SCNWLD.SYS	Wildcard support module for various system utilities.
SEND.PRG	Allows the user to send a message to another terminal.
SET.PRG	Utility to set various parameters.
SIZE.PRG	Displays a file's size on the user's terminal.
SKIP.PRG	Skips files on mag tapes.
SLEEP.PRG	Puts the user's job to sleep for the specified number of seconds.
SMDLOD.PRG	Bootstrap loader for the AM-410 disk subsystem.

SORRF.SYS	Module used by the sort utilities to sort random files. This program must never be run directly.
SORSVA.SYS	Module used by the sort utilities to sort sequential files. This program must never be run directly.
SORT.PRG	Stand-alone sort utility.
SRCCOM.PRG	Source level file comparison utility.
SUSPND.PRG	Suspends a job's activity. (see REVIVE.PRG)
SYMBOL.PRG	Creates a symbol file for use with the symbolic debugger.
SYSACT.PRG	Allows maintenance of a disk's accounting structure. Creates and deletes PPNs.
SYSLPT.INI	Sample initialization file showing how to set up the line printer spooler. The system is capable of booting using this initialization file.
SYSMEM.PRG	Defines bank switchable system memory.
SYSTAT.PRG	Displays the current system status.
SYSTEM.INI	A command file containing the instructions to the operating system for configuring the system during system startup.
SYSTEM.MON	The system monitor.
SYSTEM.PRG	During system startup, defines the programs to be made sharable via loading them into system memory. During normal operation, displays the contents of system memory.
TAPDIR.PRG	Allows you to look at the directory of a magnetic tape whose contents were created using FILTAP.
TAPE.PRG	Reads and writes magtapes using the AM-600 Magnetic Tape Formatter Interface.
TAPFIL.PRG	Copies files from magnetic tape to disk. See FILTAP and TAPDIR.
TIME.PRG	Sets and displays the current time of day. Used by the AM-120 Auxiliary I/O Controller.
TLGRAM.PDL	Example program for the Program Design Language Formatter (PDLFMT).
TODCNV.PRG	Time of day conversion module used by TIME.PRG and others. This module must not be run directly.
TRACE.PRG	Manipulates the trace (:T) flag within command files.
TRIDDT.PRG	Debugger used to display TRIDENT formatter status.
TRIINI.PRG	Initializes the AM-400 interface board and the TRIDENT formatter.
TRILOD.PRG	Bootstrap program for use with the AM-400.
TRISET.PRG	Used to configure the AM-400 when running a mix of different size TRIDENT disk drives.
TRMDEF.PRG	During system startup, this program defines the terminal configuration connected to the system. During normal operation, it lists that terminal configuration.
TXTFMT.PRG	Text formatter program.
TYPE.PRG	Utility to dump a sequential file on the user's terminal in ASCII.
T80INI.PRG	Version of the TRIINI program for T-80 drives.
T80LOD.PRG	Bootstrap program for the AM-400, TRIDENT T-80 combination.
U.PRG	Accepts and stores a single command line.
VUE.PRG	A screen-oriented text editor.
WAIT.PRG	Program to stall until the specified job is idle.
WNG210.PRG	Bootstrap program for the AM-210/WANGCO combination.
WNGLOD.PRG	Bootstrap program for the AM-200/WANGCO combination.

XY.PRG	Performs cursor positioning and other extended terminal functions. Used primarily within command files.

Programs in account [1,6]:

200DVR.DVR	Device driver for the AM-200 floppy disk subsystem.
210DVR.DVR	Device driver for the AM-210 floppy disk subsystem.
ACTIV.TDV	Terminal driver for the ACT-IV terminal.
ADDS.TDV	Terminal driver for the ADDS terminal.
ADM1.TDV	Terminal driver for the Lear Siegler ADM-1 terminal.
ADM2.TDV	Terminal driver for the Lear Siegler ADM-2 terminal.
ADM3.TDV	Terminal driver for the Lear Siegler ADM-3 terminal.
ADM31.TDV	Terminal driver for the Lear Siegler ADM-31 terminal.
ADM41.TDV	Terminal driver for the Lear Siegler ADM-41 terminal.
AM100T.IDV	Interface driver for the AM-100/T on-board serial ports.
AM120.IDV	Interface driver for the AM-120 on-board serial ports.
AM300.IDV	Interface driver for the AM-300 serial interface board.
AM310.IDV	Interface driver for the AM-310 communications board.
CEN.DVR	Device driver for the Centronics printer.
DIABLO.DVR	Device driver for the Diablo Hytype II printer.
DMEDIA.TDV	Terminal driver for the Datamedia 1520 terminal.
ECHO.MAC	Source files used by NEWTRM.PCF; do not modify.
HAZEL.TDV	Terminal dirver for the Hazeltine 1500 series terminals.
HWK500.DVR	Device driver for the AM-500.
IMSIO.IDV	Interface driver for the IMSAI SIO serial board.
IQ140.TDV	Terminal driver for the Soroc IQ-140 terminal.
LPR.DVR	Device driver for the AM-320 High-Speed Printer Interface.
MEM.DVR	Device driver to allow manipulation of the user's memory partition.
MTM.DVR	Device driver for the Multiterm printer.
MTU.DVR	Device driver for the AM-600 Magnetic Tape Formatter Interface.
M40.TDV	Terminal driver for the Teletype Model-40 printer.
PS3.IDV	Interface driver for the Processor Tech 3P+S serial board.
QUM.DVR	Device driver for the Qume Sprint 3 printer.
RES.DVR	Device driver for accessing system memory.
SIL700.TDV	Terminal driver for the TI Silent-700 terminal.
SMD410.DVR	Device driver for the AM-410 disk subsystem.
SOROC.TDV	Terminal driver for the SOROC IQ-120 terminal.
TABDEF.MAC	Source file used by NEWTRM.PCF. Do not modify.
TDV1.MAC	Source file used by NEWTRM.PCF. Do not modify.
TDV2.MAC	Source file used by NEWTRM.PCF. Do not modify.
TELTYP.TDV	Terminal driver for standard Teletype like device.
TELVID.TDV	Terminal driver for the Televideo terminal.
TRIT25.DVR	Device driver for the TRIDENT T-25 disk drive.
TRIT50.DVR	Device driver for the TRIDENT T-50 disk drive.
TRIT80.DVR	Device driver for the TRIDENT T-80 disk drive.
TRI300.DVR	Device driver for the TRIDENT T-300 disk drive.
TRM.DVR	Device driver to allow access to the terminals connected to the system as devices.
VARDEF.MAC	Source file used by NEWTRM.PCF. Do not modify.

Programs in Account [2,2]:

BATCH.CMD	Loads those programs commonly used within command files into the user's memory partition.
COM.DO	Compiles a program, automatically calling the correct compiler program, based on the program extension.
CONT.DO	Continues execution of a command file after the PAUSE command has been used.
CPY410.CMD	Copies System Disks on AM-410 based systems.
CPY500.CMD	Copies System Disks on AM-500 based systems.
EMAIL.CMD	Runs the AlphaMAIL general interface program, EMAIL.
NEWTRM.CMD	Runs the terminal building program, NEWTRM. (You must log into DSK0:[1,6] to run NEWTRM.)
RES.CMD	Loads commonly used programs into the user's memory partition.
SYSCPY.CMD	Duplicates a System Disk on floppy disk based systems.
UMAIL.CMD	Support command file for EMAIL.

Programs in Account [7,0]:

DDB.FXO	Overlay file for FIX, used to display data in DDB format.
DIR.FXO	Overlay file for FIX, used to display directories.
EA.FXO	Overlay file for FIX, used to display effective addresses.
HELP.FXO	Overlay file for FIX, used to display help message.
JCB.FXO	Overlay file for FIX, used to display job control block contents.
LABELS.FXO	Overlay file for FIX, used to display symbol table.
MAP.FXO	Overlay file for FIX, used to display the user's memory map.
MENU.VUE	Function menu for the VUE editor.
NEW.FXO	Overlay file for FIX, used to clear the user's memory map.
RAD50.FXO	Overlay file for FIX, used to display data in RAD50 notation.
RPN.FXO	Overlay file for FIX, used to perform calculation in reverse-polish notation.
TRMDEF.FXO	Overlay file for FIX, used to display data in a terminal definition block.
TYPE.FXO	Overlay file for FIX, used to type ASCII text files.

Programs in Account [7,2]:

ACT.RUN	Used by the OPR program. Should not be run directly.
CHK.RUN	Used by the OPR program. Should not be run directly.
DTB.RUN	Used by the OPR program. Should not be run directly.
OPR.CMD	Invokes the OPR program.
OPR.RUN	The main AlphaMAIL System maintenance program. Only for use by the AlphaMAIL Operator.
REC.RUN	Used by the OPR program. Should not be run directly.
STA.RUN	Used by the OPR program. Should not be run directly.
UPD.RUN	Used by the OPR program. Should not be run directly.

Programs in [7,4] (Available on the AlphaLISP/AlphaPASCAL Diskette.)

DIFF.LSP	Sample LISP program.
DOCTOR.LSP	Sample LISP program.
ILISP.LSP	Sample LISP program.
LISP.LSP	The LISP library.
METEOR.LSP	Sample LISP program.

Programs in [7,5] (Available on the AlphaLISP/AlphaPASCAL Diskette.)

CMPLIR.PCF	The AlphaPASCAL compiler.
DEMO.PAS	AlphaPASCAL demonstration program (source file).
DEMO.PCF	AlphaPASCAL demonstration program (compiled and linked version of DEMO.PAS).
NEWTRM.PCF	Terminal driver building program.
PLINK.PCF	The AlphaPASCAL linker.

Programs in Account [7,6]:

BASORT.SBR	AlphaBASIC subroutine to sort disk files.
COMMON.SBR	AlphaBASIC subroutine to store variables in common storage.
EMAIL.RUN	AlphaMAIL interface program for the general user.
FLOCK.SBR	AlphaBASIC subroutine to perform interprocess file locking functions.
IDIN.SBR	Support subroutine for AlphaMAIL.
PRIV.SBR	Support subroutine for AlphaMAIL.
READIN.SBR	Support subroutine for AlphaMAIL.
SPOOL.SBR	AlphaBASIC subroutine to send print requests to the line printer spooler.
UMAIL.RUN	Support program for AlphaMAIL. Should not be run directly.
WRTBIN.SBR	Support subroutine for AlphaMAIL.
XLOCK.SBR	AlphaBASIC subroutine to perform simple interprocess locking.
XMOUNT.SBR	AlphaBASIC subroutine to mount disks.

Programs in Account [7,7]:

ISUSYM.MAC	Definition file used when linking to ISAM.
SYS.MAC	Defines items used for communicating with the operating system.

April 1981
Revision A05

DISKS AVAILABLE FROM ALPHA MICRO

1.0 DISKS AVAILABLE

The disks currently available from Alpha Micro are listed below. Listed after the disks' descriptions are their part numbers. To obtain the prices for these disks, please contact your Alpha Micro Dealer.

WANGCO AM-200 System Diskette (STD format)	PDB-00104-06
WANGCO AM-200 System Diskette (AMS format)	PDB-00104-07
PERSCI AM-200 System Diskette (STD format)	PDB-00104-02
PERSCI AM-200 System Diskette (AMS format)	PDB-00104-03
CDC AM-210 System Diskette (Double Density, AMS)	PDB-00104-11
AM-500 System Disk (5-Mbyte disk pack)	PDB-00104-12
AM-410 System Disk (15-Mbyte disk pack)	PDB-00104-13
Miscellaneous Program Diskette	PDB-00104-19
Driver Source Diskette	PDB-00104-20
AlphaLISP/AlphaPASCAL Diskette	PDB-00104-21

Because of the large amount of software now available from Alpha Micro, it is no longer possible to fit all of the System programs onto one single-density floppy diskette. Thus, floppy diskettes other than just the System Diskette are available for some of the standard floppy devices (e.g., Wangco and Persci). These additional diskettes contain software and sources not included in the single-density System Diskettes.

The AM-410 and AM-500 System Disks and the CDC AM-210 System Diskette contain all of the single-density System Diskette software, plus all of the software contained on the three other non-System Diskettes. The three non-System Diskettes are supplied in STD (128 byte/sector) format only.

Trident users wishing to update their systems from a System Diskette should choose the System Diskette for their type of floppy drive, in whichever format is most convenient.

The following sections discuss the overall organization of the Alpha Micro software by disk account, and discuss what types of software and sources are available on the various single-density floppy diskettes.

(Changed 30 April 1981)

2.0 SOFTWARE ORGANIZATION

The general system release software is organized into thirteen different categories. Each category of software resides in a separate disk account:

[1,2] This is the System Operator's account, OPR:. It is the account from which the System Operator runs various system management and maintenance programs. Only the AM-410 System Disk has a file in this account: BADBLK.SYS (which contains disk certification data for that disk).

[1,4] This is the System Program Library account, SYS:. It is the account that contains all of the system software. You execute a program in this account by typing its name at AMOS command level.

[1,6] This is the Device Driver Library account, DVR:. It contains all terminal, interface, and general device drivers.

[2,2] This is the Command File Library account, CMD:. It contains a number of useful command and DO files.

[7,0] This is the Library account, LIB:. It contains auxiliary files for system software (such as the AlphaFIX overlay files).

[7,1] This is the Help File Library account, HLP:. It contains the HELP files. Any text file with a .HLP extension that resides in this account can be accessed via the HELP command from anywhere on the system.

[7,2] This is the AlphaMAIL account, BOX:. The AlphaMAIL Operator uses this account in maintaining and managing the AlphaMAIL system.

[7,4] This is the AlphaLISP Library account, LSP:. It contains AlphaLISP programs.

[7,5] This is the AlphaPASCAL Library account, PAS:. It contains AlphaPASCAL programs, as well as the AlphaPASCAL compiler, linker, and standard library.

[7,6] This is the AlphaBASIC Library account, BAS:. It contains various assembly language subroutines callable by your BASIC programs, and contains compiled BASIC programs. Any .RUN or .SBR file in this account can be accessed by BASIC from anywhere on the system.

[7,7] This is the MACRO Library account, MAC:. It contains various assembly language files that can be accessed by your assembly language programs.

[10,1] This account contains source files to various miscellaneous programs.

[10,2] This account contains source files for a number of terminal and device drivers.

Each one of the accounts above is included in the AM-500 and AM-410 System Disks and the double-density floppy System Diskettes.

3.0 SINGLE-DENSITY FLOPPY DISKETTES

These paragraphs discuss what accounts listed above are included on which single-density floppy diskettes:

3.1 The Single-Density System Diskette

The System Diskette is actually two diskettes that contain all of the system software, drivers, and command files distributed with the current release. (That is, these two diskettes contain accounts [1,4], [1,6], [2,2], [7,0], and [7,7].)

We have taken care to make sure that the diskette labeled "Part 1 of 2" contains the software necessary to boot your system under the minimal system initialization command file contained on the diskette. There is also enough room on the first diskette to allow you to copy files and to edit your SYSTEM.INI file.

The four different System Diskettes differ only in the device driver the monitor has been generated with. Each System Diskette contains a SYSTEM.MON file that has been generated to access a particular System Device.

3.2 The Single-Density Non-System Diskettes

The other three single-density floppy diskettes contain the rest of the accounts listed in Section 2.0 above that are not included in the single-density System Diskette.

The Miscellaneous Programs Diskette - This floppy diskette contains all of the files in accounts [7,1], [7,2], and [10,1].

The Driver Sources Diskette - This floppy diskette contains all of the files in account [10,2].

The AlphaLISP/AlphaPASCAL Diskette - This floppy diskette contains all of the files in [7,4] and [7,5]. It also contains the system software and command files necessary to use AlphaLISP and AlphaPASCAL.

(Changed 30 April 1981)

April 1981
Revision A04

A GUIDE TO THE ALPHA MICRO SOFTWARE DOCUMENTATION LIBRARY

Alpha Micro software documentation is grouped into two categories: 1., individual manuals that you can purchase separately; and 2., documents that are packaged together in the AMOS Software Update Documentation Packet. (NOTE: The AMOS Software Update Documentation Packet was previously known as the "AM-100 documentation packet.")

A number in parentheses after a document title (e.g., DWM-00100-20) indicates that the document is a manual that can be purchased separately; the number is the part number by which you may order that document.

Refer to Section 3.0 for an alphabetic list of all Alpha Micro software documentation. (In that list we indicate those documents that are part of the AMOS Software Update Documentation Packet by following their titles with the code "AMOS PKT." You may only order those documents by ordering the entire AMOS Software Update Documentation Packet, part number DSS-10000-05.)

1.0 THE ALPHA MICRO SOFTWARE MANUALS

This section lists those documents not included in the AMOS Software Update Documentation Packet or as documentation for the AlphaAccounting system.

NOTE: We are in the process of writing an AMOS System Operator's Guide, which contains information necessary to the person in charge of system management (e.g., system software installation, modifying the system initialization command file, formatting disks, memory management, performing diagnostic tests on disks and memory, creating user accounts, etc.).

(Changed 30 April 1981)

TITLE: AlphaBASIC User's Manual

(DWM-00100-01, Revision B00)

READER: Aimed at all BASIC users. Assumes that you have had some prior experience with BASIC and with programming.

TOPICS: Discusses the operation of AlphaBASIC in both interactive and compiler modes. Lists all AlphaBASIC commands and functions, and discusses the various data formats supported by AlphaBASIC. Also discusses the use of MAP statements for mapping data structures into memory. Talks about advanced BASIC programming techniques (e.g., the AlphaBASIC file I/O system, calling external assembly language routines, chaining to other programs, print using, error trapping and using the ISAM system). Contains many program examples.

TITLE: AlphaFIX User's Manual

(DWM-00100-69, Revision A01)

READER: Advanced assembly language programmer, who already fully understands the use of MACRO, LINK, and DDT.

TOPICS: Explains the use of FIX, the Alpha Micro screen-oriented debugger program for machine language programs. Lists all the FIX commands and modes.

TITLE: AlphaLISP User's Manual

(DWM-00100-05)

READER: This manual is written for the LISP programmer who is already quite familiar with the LISP language.

TOPICS: Discusses the functions available in AlphaLISP, along with a description of the data types recognized. Instructions are included on operating the language processor, along with a group of sample programs written in AlphaLISP.

TITLE: AlphaMAIL User's Manual

(DSS-10000-06)

READER: Aimed at all users of the system who are authorized users of AlphaMAIL. Assumes no prior experience with electronic mail systems but some familiarity with the Alpha Micro system is required. Further instructions for an experienced user acting as the AlphaMAIL Operator are also provided.

TOPICS: Gives full operating instructions for AlphaMAIL, along with a summary of all AlphaMAIL commands. Discusses the general user features of sending, receiving, printing and forwarding messages. Further discusses the maintenance system available to the Operator controlling AlphaMAIL.

TITLE: AlphaPASCAL User's Manual

(DWM-00100-08, Revision B00)

READER: This manual is aimed at the PASCAL programmer who is already familiar with standard PASCAL (i.e., Wirth and Jensen PASCAL).

TOPICS: Describes the AlphaPASCAL Version 2.0 compiler and linker. This implementation of PASCAL is fully compatible with the AMOS file system and the AlphaVUE text editor. The manual contains an introduction to AlphaPASCAL, information on compatibility with previous versions of AlphaPASCAL, and complete operating instructions for the AlphaPASCAL compiler, linker, and run-time package. The book also contains a complete summary of AlphaPASCAL, including information on all functions, procedures, data types, file-handling techniques, assembly language subroutines, and information on writing and modifying an external library.

TITLE: AlphaVUE/TXTFMT Training Guide

(DSS-10000-03)

READER: Written for all users who are new to AlphaVUE and TXTFMT, the Alpha Micro text processors. Assumes no prior experience with computers or text editors.

TOPICS: A tutorial containing exercises and demonstrations that teach the new user how to use AlphaVUE and TXTFMT. The emphasis of the book is on creating documents in a business environment. For complete information on AlphaVUE and TXTFMT, turn to the reference manuals: AlphaVUE User's Manual and TXTFMT User's Manual.

TITLE: AlphaVUE User's Manual

(DWM-00100-15, Revision B00)

READER: Aimed at all users of the system who want to create text files. Assumes no prior experience with text editor programs.

TOPICS: Gives full operating instructions for VUE, along with a summary of all VUE commands. Discusses the creation of the VUE initializaton file, VUE.INI, and describes those display features of your CRT terminal that are required by VUE.

TITLE: AMOS Assembly Language Programmer's Reference Manual

(DWM-00100-43, Revision B00)

READER: Written for the experienced assembly language programmer who wants to become familiar with the AM-100 assembly language programming system. Does NOT teach assembly language. For information on the instruction set used by the AM-100 processor, refer to the WD16 Microcomputer Reference Manual, (DWM-00100-04).

TOPICS: Discusses assembly language programming on the AMOS system. Describes operation of the AMOS macro-assembler, MACRO, the linkage editor, LINK, and the symbol-table file program, SYMBOL. Discusses use of the object file library generator, LIB and the global cross reference program, GLOBAL. Discusses types of expressions, data and statements recognized by the assembler, and gives information on pseudo-operations, macros and writing relocatable code. Also gives detailed operating instructions for the symbolic debugger program, DDT.

TITLE: AMOS Monitor Calls Manual

(DWM-00100-42, Revision B00)

READER: Aimed at the advanced systems programmer, who wants to interface assembly language programs to the monitor via those monitor calls available to user programs.

TOPICS: Describes in detail the 70+ monitor routines resident in the operating system that are available to user programs. The manual also discusses the file service system, the terminal service system, the structure of a memory partition, and the format of various data structures used by the system (e.g., Job Control Blocks, Dataset Driver Block, Master File Directory, etc.).

TITLE: AMOS System Commands Reference Manual

(DWM-00100-49, Revision A01)

READER: Written for the experienced user of the AMOS system who wants a quick reference guide to all commands on the system. We assume that the reader is already very familiar with the AMOS system software.

TOPICS: Contains two- or three-page summaries of all AMOS commands. Also contains a chart of the ASCII character set.

TITLE: AMOS User's Guide

(DWM-00100-35, Revision A01)

READER: This manual is written for the beginning user of the system who has had some prior experience with computers, but who is new to the AMOS system.

(Changed 30 April 1981)

TOPICS: Written in two parts: Part I deals with turning the system on and off, typing commands, logging in and specifying files. Part II covers the major system commands, and discusses command files and DO files, special wildcard commands and file backup procedures. Command descriptions are ordered by type and function.

TITLE: Introduction to AMOS

(DWM-00100-65)

READER: This manual was written for the person who wants some background information on the AMOS system in particular and computers in general. Assumes little computer experience.

TOPICS: NOT an operations or demonstration manual. Written in three parts: Part I deals with elementary computer concepts, defining terms such as "files," "hexadecimal," "CPU," and "program." Part II talks about some of the programs available on the AMOS system, such as AlphaBASIC, VUE, TXTFMT, and ISAM. Part III gives an overview of the Alpha Micro operating system, giving a very general introduction to the Terminal Service System, Job Scheduler, Command Processor, and other major portions of AMOS.

TITLE: ISAM System User's Guide

(DWM-00100-06, Revision A02)

READER: This manual is aimed at the assembly language programmer who wants to use the ISAM (Indexed Sequential-files Access Method) package from within an assembly language program. Assumes thorough knowledge of assembly language, and prior experience with ISAM. NOTE: For complete information on using ISAM from within BASIC programs, refer to the AlphaBASIC User's Manual.

TOPICS: Describes the ISAM calls available to the assembly language programmer for modifying and using ISAM files, as well as the various ISAM programs that the user can run from the monitor level for building and dumping ISAM files. Also discusses the process of building and dumping ISAM files. Contains brief discussion of using ISAM from within a BASIC program.

TITLE: TXTFMT User's Manual

(DWM-00100-07, Revision B00)

READER: Written for all users of the system who want to create formatted documents. Assumes experience with one of the text editing programs, AlphaVUE or EDIT.

TOPICS: Lists all of the text formatting commands recognized by TXTFMT. Also discusses TXTFMT operation and error messages.

TITLE: WD16 Microcomputer Reference Manual

(DWM-00100-04)

READER: This manual is aimed at the experienced assembly language programmer who wants to become familiar with the instruction set used by the AM-100 processor.

TOPICS: The manual does NOT contain information on assembly language programming, or on using the AMOS assembler. It does list all of the instructions and addressing modes used by the WD16 microprocessor. For information on assembly language programming on the AMOS system, refer to the AMOS Assembly Language Programmer's Reference Manual.

2.0 THE AMOS SOFTWARE UPDATE DOCUMENTATION PACKET

The documents in the AMOS Software Update Documentation Packet (DSS-10000-05) are organized into four major groups:

1. User's Information
2. System Operator's Information
3. System Programmer's Information
4. BASIC Programmer's Information

For a list of the documents included in the AMOS Software Update Documentation Packet, refer to the Master Table of Contents; this document is the first document in the AMOS Software Update Documentation Packet.

3.0 AN ALPHABETIC LIST OF ALL ALPHA MICRO SOFTWARE DOCUMENTATION

NOTE: The code AMOS PKT following a title indicates that the document is part of the AMOS Software Update Documentation Packet.

AlphaAccounting Release Notes - Version 1.3 (DWM-00100-61), available only to users licensed for the AlphaAccounting software package.

AlphaBASIC User's Manual (DWM-00100-01, Revision B00)

AlphaFIX User's Manual (DWM-00100-69, Revision A01)

AlphaLISP User's Manual (DWM-00100-05)

AlphaMAIL User's Manual (DSS-10000-06)

AlphaPASCAL User's Manual (DWM-00100-08, Revision B00)

The AlphaVUE/TXTFMT Training Guide (DSS-10000-03)

AlphaVUE User's Manual (DWM-00100-15, Revision B00)

AMOS Assembly Language Programmer's Reference Manual (DWM-00100-43, Revision B00)

AMOS Monitor Calls Manual (DWM-00100-42, Revision B00)

AMOS Release Notes - Version 4.5 (AMOS PKT)

AMOS Software Update Documentation Packet (DSS-10000-05)

AMOS System Commands Reference Manual (DWM-00100-49, Revision A01)

AMOS User's Guide (DWM-00100-35, Revision A01)

AMOS Version 4.4 Method of Handling Bad Disk Blocks (AMOS PKT)

BASORT - BASIC Subroutine for Sorting Random and Sequential Files (AMOS PKT, Revision A01)

Building a Terminal Driver (The NEWTRM Program) (AMOS PKT)

Change Page Packet #1 for AlphaBASIC User's Manual (DSS-10000-04)

Change Page Packet #2 for AlphaBASIC User's Manual (DSS-10000-07)

Change Page Packet #1 for AlphaPASCAL User's Manual (DSS-10000-10)

Change Page Packet #2 for AMOS System Commands Reference Manual (DSS-10000-09)

COMMON - BASIC Subroutine to Provide Common Variable Storage (AMOS PKT)

Configuring Floppy Disk Drivers (AMOS PKT, Revision A01)

Defining Non-system Disk Devices (AMOS PKT, Revision A01)

Defining Switchable System Memory (AMOS PKT, Revision A02)

Disk Drivers and Formats (AMOS PKT, Revision A03)

Disk Labeling Procedures (AMOS PKT)

Disk Maintenance Procedures for the System Operator (AMOS PKT, Revision A03)

Disks Available from Alpha Micro (AMOS PKT, Revision A05)

The DUMP Command (AMOS PKT, Revision B00)

EDIT: A Character-oriented Text Editor (AMOS PKT, Revision A01)

FLOCK - BASIC Subroutine to Coordinate Multiuser File Access (AMOS PKT, Revision A01)

Generating System Monitors (AMOS PKT, Revision A01)

A Guide to the Alpha Micro Software Documentation Library (AMOS PKT, Revision A04)

Important Notice for LISP Users (AMOS PKT)

Introduction to AMOS (DWM-00100-65)

I/O Programming for the Alpha Micro Computer (AMOS PKT, Revision A01)

ISAM System User's Guide (DWM-00100-06, Revision A02)

The Magnetic Tape File Backup Programs (AMOS PKT)

Memory Management Option (AMOS PKT, Revision A01)

New Command File and DO File Features (AMOS PKT, Revision A01)

Program Design Language Formatting System (AMOS PKT)

Setting Up the Line Printer Spooler (AMOS PKT, Revision A02)

Software Installation Instructions for the AM-120 (AMOS PKT)

Software Installation Instructions for the AM-170 Memory Board (AMOS PKT)

(Changed 30 April 1981)

Software Notice for AM-410 Users (AMOS PKT, Revision A03)

SPOOL - BASIC Subroutine for Spooling Files to the Line Printer (AMOS PKT, Revision A02)

The System Initialization Command File (AMOS PKT, Revision A04)

Terminal Service System (AMOS PKT)

TXTFMT User's Manual (DWM-00100-07, Revision B00)

Using the Magnetic Tape Utility Programs (AMOS PKT)

WD16 Microcomputer Reference Manual (DWM-00100-04)

XLOCK - BASIC Subroutine for Multiuser Locks (AMOS PKT, Revision A01)

XMOUNT - BASIC Subroutine to Mount a Disk (AMOS PKT, Revision A01)

USER'S INFORMATION

This section contains the following documents:

New Command File and DO File Features, Revision A01

The DUMP Command, Revision B00

Important Notice for LISP Users

EDIT: A Character-oriented Text Editor, Revision A01

Program Design Language Formatting System

AMOS 4.5 SOFTWARE UPDATE DOCUMENTATION PACKET

These documents reflect AMOS Versions 4.5 and later

ALPHA MICROSYSTEMS
17881 Sky Park North
Irvine, CA 92714

April 1981
Revision A01

NEW COMMAND FILE AND DO FILE FEATURES

Command and DO files play an important part in extending the power and range of the AMOS command language. This document describes several new features and programs that greatly increase the flexibility of your command files. Among other things, these new command file features allow you to: change the command file trace flag as many times as you wish within a command file; conditionally transfer control to various portions of a command file based on whether a specific file exists; and, exit a command file, perform various AMOS commands, and then resume execution of that command file. We also describe several new AMOS level commands that are particularly useful as command file elements.

For detailed information on the concepts of command files and DO files, refer to Chapter 8, "Command Files and DO Files," in the AMOS User's Guide, (DWM-00100-35). In brief, a command file is a special kind of text file which can contain a series of AMOS commands, specifications of other command files, test data, etc. To execute all of the commands and data in a DO file or a command file, just enter the name of the file at AMOS command level. Command files are extremely useful for performing entire sequences of commands that you use frequently (e.g., commands that do disk backup or that compile a series of BASIC programs).

DO files are a type of command file that allow you to specify arguments when you invoke the DO file that are substituted into that DO file where special parameter symbols appear. This allows you to use one DO file with a wide range of files or programs. Since DO files are just one type of command file, when we mention "command files" in the rest of this document, we will be talking about both standard command files and DO files.

Special symbols may appear in command files that allow you to ask the user of the file for input, and to display messages to that user. In addition, you may use the :T, :S and :R symbols to choose whether or not users of the command file see program output that result from the actions of your command file. Except for these special symbols, when AMOS processes a command file, it treats most of the elements of the file as if you were entering each command file line from a terminal, line by line.

For more information on any command discussed below, refer to the reference sheet for that command in the AMOS System Commands Reference Manual, (DWM-00100-49, Revision A01 and later).

(Changed 30 April 1981)

1.0 THE COMMAND FILE TRACE FLAG (:T AND TRACE)

Command files have always allowed you to choose whether or not the user of your command file sees the contents of the file while it is being processed by AMOS. AMOS checks the status of the command file "trace flag" to determine whether or not to display the contents of the command file. Until the advent of the TRACE command, the only way to affect the trace flag was by including the :T symbol at the place in the command file where you wanted users to see command file contents. The main disadvantage to this was that you could only use the :T symbol once to turn the trace flag on in a command file, and then could not turn it off again. (Of course, if you wanted the trace flag off, you could simply omit the :T from the command file.)

NOTE: When the trace flag is off, you can still allow users to see program output and messages by use of the special :S, :R, and :<> symbols; however, they will not see comments or command lines in the command file. For example, if the line:

```
.ERASE *.BAK (RET)
```

appears in the command file when the trace flag is off, if no :R precedes the command line, the user of the file does not see the command line OR the information output by the ERASE program. If a :R precedes the command line and the trace flag is off, the user does not see the command line, but does see the output produced by ERASE. NOTE: Remember to always include a :R in your command file if the trace flag is off but you want the user of the command file to see any output generated by the commands in the command file.

1.1 The TRACE Command

The TRACE command allows you to turn the trace flag on or off at any point within the command file as many times as you wish. You may also use TRACE to switch the current state of the trace flag, regardless of whether it is on or off. You may only use TRACE in a command file.

To use TRACE, enter it in your command file in one of the following three formats:

TRACE ON	Turns the command file trace flag ON. Users see all of the contents of your command file, including comments and all program command lines. :S and :R are ignored when TRACE ON is in effect.
TRACE OFF	Turns the command file trace flag OFF. Users see only messages enclosed with the :<> symbols. If you include the :R symbol, users also see program output. (The :S symbol turns off the :R symbol.) (NOTE: :S and :R work with TRACE OFF as if no TRACE or no :T were in the command file.)

TRACE SWITCH	Switches the current status of the command file trace flag. If the flag is OFF, this command turns it ON and vice versa.

If you do not include an argument on the TRACE command line, AMOS ignores the command.

As an example of the use of the TRACE command, let's create a command file that does disk backup:

```
; Command file to backup HAWK System Disk onto another disk.
LOG DSK0:1,2
:< Make sure nobody else is running on DSK0: or your backup disk>
ERASE *.BAK[]
DIRSEQ
;
TRACE ON
DSKCPY
DSK0:
DSK1:
:<
All done. Remove cartridge and label it. Log back into your
account.>
```

Not having a :T or TRACE OFF makes sure that the user of the command file won't see the output of the cleanup functions we perform. (The user still sees those messages bracketed by the :<> symbols.) The TRACE ON ensures that the user will see that we are running the DSKCPY program.

2.0 LOOKING UP FILES FROM WITHIN A COMMAND FILE (LOOKUP)

There are many occasions when a command line in a command file causes AMOS to search for a file. If AMOS fails to find the specified file, execution of the command file does not abort, but continues even though the necessary file was not found. Sometimes this can be a severe inconvenience if several subsequent command lines assume that the nonexistent file exists.

2.1 The LOOKUP Command

The LOOKUP command allows you to search for a file from within a command file, and then to perform several actions based on the results of that search. If a file is not found, you may resume execution of the command file at the command line following the lookup, or you may cause the command file to terminate execution. You may also choose whether to display your own error message or a standard AMOS error message if the file is not found.

By using LOOKUP in combination with the GOTO command, you can choose which portions of the command file to execute based on the results of the file lookup. This allows you to perform conditional branching within a command file.

You may only use the LOOKUP command within a command file. To use LOOKUP, enter it into your command file in one of the following formats:

LOOKUP Fspec	where Fspec specifies the file you want to search for. If the file is found, LOOKUP continues execution of the command file; if it is not found, LOOKUP displays the appropriate AMOS error message (e.g., ?Cannot OPEN STDMOD.BAD - file not found), and returns the user of the command file to AMOS command level.
LOOKUP Fspec Msg	where Fspec specifies the file you want to search for, and Msg is a message supplied by you. If the file is found, LOOKUP just continues execution of the command file. If the file is not found, LOOKUP displays your message (instead of the appropriate AMOS error message) and returns the user of the command file to AMOS command level.
LOOKUP Fspec/	where Fspec specifies the file you want to search for, and the "/" symbol tells LOOKUP not to abort command file execution if the file is not found. If the file is found, LOOKUP skips over the next line in the command file and resumes execution at the line past it. If the file is not found, LOOKUP continues execution at the next line after the LOOKUP command, and displays the appropriate AMOS error message.
LOOKUP Fspec/Msg	where Fspec specifies the file you want to search for, "/" tells LOOKUP not to abort command file execution if the file is not found (see the paragraph above), and Msg is a message supplied by you. If the file is found, LOOKUP skips over the next line in the command file and resumes execution at the line past it. If the file is not found, LOOKUP resumes execution at the next line in the command file, and displays the specified message. (If you include both the "/" symbol and a message, the message must follow the slash on the LOOKUP command line.)

If you omit portions of the file specification, LOOKUP assumes the device and account the user of the command file is logged into and a .PRG extension.

The LOOKUP "/" option becomes especially useful when you use the GOTO and EXIT statements to select certain portions of the command file to be

executed as a result of the LOOKUP operation. (See Section 4.0, "Transferring Control Within a Command File (GOTO, EXIT)," for an example of conditional branching within a command file.)

As an example of the use of the LOOKUP command, let's create a DO file that installs a BASIC program into an account. (The user of the DO file provides the name of the program to install as an argument when he or she invokes the DO file. This argument gets substituted into the DO file for the $0 symbol, which is a special DO file parameter symbol.)

```
; Command file to install $0.BAS from project library account
; into the account user is logged into.
;
:R
LOOKUP $0.BAS[300,0] ?That BASIC program doesn't exist. Try again.
COPY =[300,0]$0.BAS
;
TRACE ON
COMPIL $0.BAS
RUN $0.RUN
```

If the user of the DO file doesn't specify an argument, or gives an incorrect file specification, the LOOKUP command above catches it and ends command file execution.

NOTE: The command file above provides a good example of why you often need to check to see if a proper file specification has been given. If the example above did not use the LOOKUP command, and if the user of the command file supplied no argument at all (thus causing a space to be substituted for the $0 symbol), the COPY command would copy ALL .BAS files from [300,0] over to the account the user is logged into.

3.0 TEMPORARILY INTERRUPTING COMMAND FILE EXECUTION (PAUSE, CONT)

It would often be convenient to temporarily exit a command file, perform various cleanup or housekeeping functions, and then resume use of the file. The PAUSE and CONT commands allow you to do so.

The PAUSE command causes the temporary interruption in the execution of the command file in which it appears. You may then execute AMOS commands, invoke other command files, use a text editor, etc. To resume execution of the command file, use the CONT command.

3.1 The PAUSE Command

You may only use PAUSE within a command file. Enter the PAUSE command in your command file where you want to temporarily interrupt execution of the file. You may optionally include a message on the PAUSE command line which is displayed when the command file pauses. For example:

```
PAUSE Type a K; COPY old .BAS files to [40,1]; type CONT to resume backup
```

This is what happens when a PAUSE statement is processed in a command file you are using:

1. If a message has been included on the PAUSE command line, PAUSE displays that message to you.

2. Whether or not it has displayed a message, PAUSE now stops and waits for you to type a character. If you type a RETURN, PAUSE resumes execution of the command file. If you type anything but a RETURN, PAUSE returns you to AMOS command level.

3. Once PAUSE returns you to AMOS command level, you can now run any programs or command files you want. To resume execution of the command file (at the point after the PAUSE command), just type CONT at AMOS command level. (The CONT command may also appear within a command file.)

When PAUSE interrupts execution of a command file, it saves the elements of the command file past the PAUSE command line in a special disk file named CNT.CMD. (This file appears in the device and account the user of the command file is logged into.) When you use the CONT command, it loads into your memory partition the CNT.CMD file that is in your account and resumes executing that command file. (This means, of course, that to resume execution of a PAUSEd command file, the user of the file must be logged into the device and account where the command file was originally interrupted.) The CNT.CMD file always contains the most current command file that has been PAUSEd in that account. For example, if you use a command file that is interrupted because of a PAUSE command, the rest of that file is stored as CNT.CMD in your account.

If, after performing various commands, you do not use a CONT command before invoking another command file that also uses PAUSE to interrupt execution, the contents of CNT.CMD (the first command file) are replaced by the contents of the second command file. If you use the CONT command now, you resume execution of the second command file, not the first. When a second PAUSE causes the current contents of CNT.CMD to be written over (because a CONT has not been used to resume execution of the previous command file), you see the following message:

```
%Supersedes existing file
```

No harm is done if you do not resume command file execution by using a CONT command before another PAUSE command occurs, but you do lose the previous contents of CNT.CMD when the new command file replaces them.

As an example of the use of PAUSE, let's create a command file that performs "housekeeping" functions on a disk:

```
; This command file cleans up the disk and performs a backup.
:R
:<
If disk labeled BACKUPA is in drive, type a RETURN; otherwise,
type anything else, go put pack in drive, and then type CONT
when you are ready to resume backup.>
;
PAUSE
; Proper backup pack (DSK5:) is now in drive.
MOUNT DSK5:
:<
If you want to save everything on the disk, type a RETURN.
If there are any scratch files you don't want to save, type
anything but a RETURN.  Then erase your old, working files
from all accounts on the disk.  To resume backup, type CONT.

>
PAUSE Enter a character:
;
; Disk is ready to back up.  Chain to another command file
; that does actual backup.
BACKUP
:<
All done.  You may remove BACKUPA from drive.>
```

3.2 The CONT Command

If a command file has been temporarily interrupted as the result of a PAUSE command (that is, if a CNT.CMD file appears in the account you are logged into), you may resume execution of that command file by using the CONT command. For example:

```
.CONT RET
```

If there is a CNT.CMD file, CONT tells AMOS to process it. When the entire command file has been processed, CONT erases the CNT.CMD file from the disk. If no CNT.CMD file exists in the account, CONT displays the message:

```
?Can't continue
```

and you are returned to AMOS command level.

4.0 TRANSFERRING CONTROL WITHIN A COMMAND FILE (GOTO, EXIT)

Since the LOOKUP command allows you to choose which line of a command file to execute, it now becomes possible to use a transfer command, GOTO, to select which portion of a command file to execute based on the results of a file lookup. Used in combination with one another, the GOTO, EXIT, and LOOKUP commands allow your command files to perform conditional branching.

4.1 The GOTO Command

You may only use the GOTO command within a command file. The GOTO command allows you to transfer control from one portion of your command file to another. The GOTO command line must contain both the GOTO command and an argument which is the name of the label to branch to. For example:

```
:R
; Command file to compile BASIC programs.
;
LOOKUP TAXTBL.BAS/?Couldn't find file.  Are you in right account?
; If file not found, go to NOFILE.
GOTO NOFILE
TRACE ON
COMPIL TAXTBL.BAS
RUN TAXTBL.RUN
;
EXIT *That's all...Returning you to AMOS command level*
;
;NOFILE
:<
We're going to enter VUE so you can create TAXTBL.BAS.  Type
an X if you don't want to create the file; otherwise,>
PAUSE Hit RETURN when ready:
;
VUE TAXTBL.BAS
Y
```

In the example above, the GOTO command line contains the argument NOFILE. NOFILE is the label of the portion of the command file to which the GOTO command transfers control.

There are some conventions you must follow in setting up a GOTO, its argument, and the label that designates the portion of the command file to which you want to transfer.

1. GOTOs must precede the labels they branch to. That is, GOTO statements may only transfer <u>forward</u> in the file.

2. An argument may not contain trailing spaces. That is, the end of an argument must be either a RETURN or a semicolon (which identifies the start of a comment). That means that if you include a comment on the GOTO command line, it must begin directly after the argument.

3. The command file label may either be a comment (that is, begin with a semicolon) or a valid, executable command file element.

   a. If a label is a comment, the argument in the GOTO command line that refers to the label must not begin with a semicolon. (See the sample command file above.)

   b. If a label is not a comment, it must be a valid command file element (e.g., a program name, a command file specification, etc.).

   If the GOTO statement directs you to a label that is not a comment or a valid command file element, the command file resumes execution after the label.

4. You may begin a label with spaces, a semicolon, or spaces followed by a semicolon. (There may be no spaces between a semicolon and the rest of the label.) These are ignored when GOTO compares an argument to the label it selects.

5. Labels may be of any length (as long as they fit on one line), and must be the only thing on the line.

If GOTO cannot find the specified label, the user of the command file sees:

?Label not found

and is returned to AMOS command level.

4.2 The EXIT Command

Whenever you create conditional branches within a command file, you face the problem of separating portions of the command file so that users not deliberately transferred to a labeled portion do not "fall into" that section of the command file as they proceed through the file. The EXIT statement allows you to create one or more points in the file which cause the user to gracefully be returned to AMOS command level. You may only use EXIT within a command file. The sample command file above demonstrates the use of the EXIT statement. As another example, consider the DO file below, which does different things with a file, based on the extension of that file:

```
; Command file that handles text files.
; If file .TXT file, format it.
:R
LOOKUP $0.TXT/?not .TXT file
GOTO NOTTXT
TRACE ON
TXTFMT $0.TXT
TRACE OFF
;
;NOTTXT
LOOKUP $0.LST/?not .LST file
GOTO NOTLST
PRINT $0.LST
EXIT *Your file is formatted and the .LST version is printed.*
;
;NOTLST
EXIT ?Couldn't find a .TXT or .LST file of that name.
```

5.0 ADDITIONAL USEFUL COMMANDS (BATCH, COM)

In addition to the commands we discussed above, two other new commands exist that are particularly helpful when used within command files. You may also use these commands at AMOS command level.

5.1 The BATCH Command

A command file executes faster if the programs it accesses are already loaded into memory when that command file needs them. The BATCH command loads into your memory partition programs that are frequently used when making use of the new command file features (GOTO, LOOKUP, EXIT, TRACE, PAUSE, and LOAD). (NOTE: GOTO, LOOKUP, EXIT, TRACE, and PAUSE are re-entrant, and may be placed into system memory by the System Operator.) If you use BATCH from within a command file rather than at AMOS command level, you will want to put BATCH at the front of the command file. BATCH takes up about 1K of your memory partition. To use BATCH, enter the command followed by a RETURN:

```
.BATCH (RET)
```

5.2 The COM Command

The COM command processes a file based on its extension. To use the command, enter COM followed by the name of the file you want to affect. You may not include the extension of the file. In addition, the file must appear within the account you are logged into, but you may specify a different device. For example:

```
.COM STD1:MNMENU RET
```

COM begins looking for the disk file in this order:

1. .MAC file? Then assemble with MACRO.
2. .BAS file? Then compile with BASIC.
3. .PAS file? Then compile with PRUN CMPILR.
4. .TXT file? Then format with TXTFMT.

If you omit portions of the file specification, COM assumes the account and device you are logged into. (Of course, if COM is used within a command file, COM uses the account and device of the user of the command file as the defaults.)

If COM can't find the file you have specified, or if the file does not have one of the extensions listed above, you see:

```
?Filename is not a compilable file
```

where Filename is the file you specified on the COM command line.

May 1980
Revision B00

THE DUMP COMMAND

1.0 INTRODUCTION

This document describes the DUMP utility program. (Also, see the reference sheet on DUMP in the AMOS System Commands Reference Manual, (DWM-00100-49), for a brief summary of DUMP command format.)

DUMP gives you a simple method for examining data, either in memory or on the disk. You can use DUMP to take a look at the contents of memory, the contents of a block on the disk, a disk Master File Directory, a disk User File Directory, or a disk bitmap. You can use DUMP to look at the contents of both random and sequential files. DUMP is re-entrant, and may be loaded into system memory by the System Operator.

Some uses of DUMP require that you give it one or more numeric arguments. These arguments must be in the number base the system is using for your numeric displays (usually octal). (You can use the SET command to change this number base from octal to hexadecimal, and vice versa. See the SET reference sheet in the AMOS System Commands Reference Manual (DWM-00100-49), for more information on SET.) Some uses of DUMP require that you supply keywords which select the DUMP function you want to use. You may abbreviate these keywords by giving just as many characters as will uniquely identify that keyword. (For example, you may enter DI instead of DIRECTORY.)

NOTE: Previous versions of this document used the term "disk records." The use of the word "record" can cause some confusion since it is sometimes used in other documentation to mean different things. In the interests of clarity, therefore, we have adopted the convention that the 512-byte groups of data into which AMOS organizes the disk are called "disk blocks," not "disk records." We have changed this document and the DUMP program accordingly. (We have retained the earlier DUMP format "DUMP RECORD"; it performs exactly the same function as the new "DUMP BLOCK" command.)

2.0 DUMP FUNCTIONS

DUMP allows you to select six different display functions. The DUMP format you use selects the specific function you want to perform:

2.1 Displaying Memory

FORMAT:

.DUMP Address1 Address2 (RET)

To display memory, give DUMP two memory addresses in the number base the system is using for your numeric displays. (If your system is a bank-switched system, remember that you may only display memory addresses that are in the memory bank within which your job resides.) For example:

.DUMP 110000 120002 (RET)

DUMP displays the contents of memory from the first address to the second address, inclusive. If you do not supply a second memory address, DUMP displays only the first 16 bytes of data. NOTE: DUMP rounds the starting address down to the nearest multiple of 16.

A DUMP memory display looks something like this:

```
110000:006562 020056 052040 062550 066400 071557 020164 067543 rm. The most co
110020:066555 067157 072040 070171 020145 063157 062040 071551 mmon type of dis
110040:066160 074541 072040 065541 071545 072040 062550 063040 play takes the f
```

Let's take a look at the first line of this display:

1. The first number on the left (ending with a colon) is the memory address that contains the first byte of data on the line. In this example, memory addresses 110000 and 110001 contain the two bytes of data 006562.

2. Each group of six digits after the memory address represents two bytes (16 bits) of data in octal form. (If the system is using hexadecimal for your numeric displays, DUMP displays the data in groups of eight bits-- one byte.)

3. On the far right of the display is a field that gives the ASCII form of the data. It displays the same data as the numbers in the center of the display, but translated into their character representation. Non-printing ASCII characters (such as Control-characters) appear as dots.

2.2 Displaying a Disk File

FORMAT:

.DUMP Filespec (RET)

To display a random or sequential disk file, type DUMP followed by the specification of the file you want to see. Then type a RETURN. For example:

```
.DUMP DSK1:PROJCT.OBJ[12,45] RET
```

If you omit portions of the file specification, DUMP assumes the device and account you are logged into and a .PRG extension. DUMP displays the entire file in the same form as the memory display (see above).

DUMP precedes the display with a message that tells you the number of the block you are seeing. For example:

```
Block number 12033 of DSK0:DATA.DAT[35,4]
```

If you are dumping a sequential file, DUMP also displays the next block link. For example:

```
Block number 784 of DSK1:PROJCT.OBJ [12,45], next block link is 11027.
```

2.3 Displaying a Disk Block

FORMAT:

```
.DUMP BLOCK Block-number1 {Block-number2} {Devn:} RET
```

To display the data in a disk block, type DUMP BLOCK. (You may also use the format DUMP RECORD.) Now enter the number of the block you want to see and the specification of the logical unit that holds that block.

If you want to see the contents of several blocks, enter a second block number. Type a RETURN. DUMP now displays the data from the first to the second block, inclusive. For example:

```
.DUMP BLOCK 1355 1360 HWK2: RET
```

If you do not specifiy a device, DUMP assumes the device you are logged into. The DUMP display looks much like the memory display above. The addresses on the left side of the display give the relative position of the data in each block.

2.4 Displaying a Bitmap

FORMAT:

```
.DUMP BITMAP {Devn:} RET
```

To display a disk bitmap, enter DUMP BITMAP. Now enter the specification of the logical unit whose bitmap you want to see. (If you omit the device

specification, DUMP assumes the device you are logged into.) Now type a RETURN. For example:

```
.DUMP BITMAP DSK5: RET
```

You now see a display of ones and zeros that represents the bitmap of the disk. (A bitmap is a disk allocation map that marks which disk blocks are in use and which are available.) Every block on the disk is represented in the bitmap by a one (if in use) or a zero (if empty). Each line of the display begins with a number which ends in a colon; this octal or hex number is the number of the disk block represented by the first one or zero on that line.

At the end of the display, DUMP prints the total number of free blocks (in decimal) on the disk. (An STD-format diskette has a total of 512 blocks; a Hawk hard disk has 9696 blocks, and a Phoenix hard disk has about 29088 blocks.) For example:

```
.DUMP BITMAP HWK1: RET

                    Bitmap Dump of HWK1:

000000: 11111111111111011110111111111011111111101111111111111111111111
000100: 11111111110111111111011111111111111111110011111111110011111111
000200: 11110111000110060110010000101010111111110001000111111111111111
000300: 11111111111111111111110111111111111011111111111100111111111111
                                .
                                .
                                .
022500: 00000000000001000000000001110000001010111111000000000011000000
022600: 00000000000000000000010000000000000000000000000001110000000000
022700: 00000000000000000000000000000

4850 free blocks
.
```

2.5 Displaying a Disk Master File Directory

FORMAT:

```
.DUMP MFD {Devn:} RET
```

Every disk contains a Master File Directory in Block 1 that contains a list of all accounts on the disk and the starting block number of all individual account directories (the User File Directories, or UFDs). To see the MFD of a disk, type DUMP MFD. Now enter the specification of the logical unit whose MFD you want to see. (If you omit the device specification, DUMP assumes the device you are logged into.) Finally, type a RETURN. For example:

.DUMP MFD HWK3: RET

The display you see looks something like this:

```
Master File Directory Dump of HWK3:

000000    [1,2]     002110
000010    [11,5]    002105
000020    [50,1]    000024
             .
             .
             .
000170    [0,0]     000000
000200    [0,0]     000000
.
```

Each line represents one User File Directory. The number on the left gives the relative address of the MFD entry in the disk block. The characters in the center give the account PPN. The number on the right gives the disk block at which the UFD for that account begins.
Nonexistent accounts appear as:

[0,0]

This is a normal part of the MFD display.

2.6 Displaying a Disk User File Directory

FORMAT:

.DUMP DIRECTORY Block-number {Devn:} RET

or:

.DUMP DIRECTORY [p,pn] {Devn:} RET

To display the directory for an individual account (i.e., the User File Directory, or UFD), type DUMP DIRECTORY followed by the number of the block at which the UFD starts. Or, you may simply enter the PPN associated with that account, using the standard AMOS PPN format of [p,pn]. (If you use [p,pn] format, and DUMP is not able to find the account you specified, you see: ?Illegal user code.)

Now enter the specification of the logical unit that contains the UFD you want to see. Type a RETURN. For example:

.DUMP DIRECTORY 002105 SMD4: RET

or:

.DUMP DIRECTORY [110,2] SMD4: RET

(Changed 1 May 1980)

If you omit the device specification, DUMP assumes the device you are logged into. The display may look something like this:

Directory dump of block 2105, next block link is 2564

Addr	Filename	Size	Active	Link
000002	SYSINO TXT	74	000166	002306
000016	:80RLD TXT	21	000034	002421
000032	PHONX LST	34	000053	002446
000046	:80ST MAC	27	000227	002650
000062	DOCMAN BAS	21	000413	002654
000076		0	000000	000000
000112		0	000000	000000

The first line of the display tells you the number of the block you are looking at. The next line tells you what disk block contains the next section of the directory. (A "next block link" of 0 indicates that the block you are displaying is the last block in the directory.) The rest of the display gives information about the directory entries.

You see this information for each directory entry:

Addr — The position (in bytes) of the directory entry relative to the start of the block.

Filename — The name and extension of the file.

Size — The number of disk blocks in the file.

Active — The number of active data bytes in the last block of the file.

Link — The address of the first disk block of the file.

NOTE: Directory entries in which the filename begins with the characters ":80" represent files that have been deleted from the directory. These are normal elements of the DUMP DIRECTORY display. The next time the system writes a new entry into the directory, it overwrites the first deleted entry.

May 1980

IMPORTANT NOTICE FOR LISP USERS

Several new functions and enhancements have been added to LISP in AMOS Release 4.4. These features include improved error reporting and the addition of functions to the Extended Library to handle breakpoints.

1.0 ERROR HANDLING

When LISP reports an error, it now displays the user function in which the error occurred. For example:

```
*(DE DOUBLE (X) (PLUS XX))
DOUBLE
*(DOUBLE 2)
UNBOUND VARIABLE - EVAL IN DOUBLE
                   =========

XX
*
```

2.0 NEW FUNCTIONS

Three new functions have been added to LISP: RETFROM, BREAK, and UNBREAK. In addition, we have added the variable BREAKFNS (which is maintained by BREAK and UNBREAK).

2.1 RETFROM

The call (RETFROM fn val) causes the most recent call of function fn to return with value val. If the specified function is not active, LISP generates an error message. For example:

```
*(DE F1 (X) (PROGN (F2) X))
F1
*(DE F2 () (RETFROM @F1 5))
F2
*(F1 7)
5
```

NOTE: The call (RETFROM PROG val) behaves exactly the same as (RETURN val).

2.2 BREAK (added to the Extended Library)

The call (BREAK fn1 fn2 ...) causes execution of a program to be interrupted if an attempt is made to call any of the specified functions. You may then single-step execution of the interrupted function by typing a line-feed, or resume execution by typing (RESUME). NOTE: fn1, fn2, ... are not evaluated.

2.3 UNBREAK (added to the Extended Library)

The call (UNBREAK fn1 fn2 ...) restores the specified functions so that they no longer interrupt program execution when called. (That is, this function clears breakpoints set via the BREAK function.) NOTE: fn1, fn2, ... are not evaluated.

2.4 BREAKFNS (added to the Extended Library)

BREAKFNS is a variable which contains a list of all functions which will interrupt program execution when called. BREAKFNS is maintained by BREAK and UNBREAK; therefore, you should not directly modify this variable. (See BREAK and UNBREAK, above.)

July 1979
Revision A01

EDIT - A CHARACTER-ORIENTED TEXT EDITOR

1.0 INTRODUCTION

There are two text editing programs available on the AMOS system: VUE (a screen-oriented text editor) and EDIT (a character-oriented editor). For information on VUE, see the manual AlphaVUE User's Guide (DWM-00100-15). Unfortunately, a manual for EDIT does not yet exist. This document gives only a brief summary of the EDIT commands.

Character-oriented text editors were originally designed to be used on non-CRT terminals. Because these kinds of editors were designed to be used on hard copy terminals that do not permit fast display, the emphasis of such an editor is not on display, but on speed and power.

When you edit a text file, a text editor brings a copy of the file into memory and allows you to make your editing changes to the copy in memory; then the editor writes your changed file back out to the disk. EDIT maintains a pointer (called DOT) that points to your current position in the copy of the text file that is in memory. Most commands that you give to EDIT reference this pointer to see what text to affect.

You do not see any of the text in memory unless you explicitly ask EDIT to display one or more lines of text. You advance throughout the text in memory by using the various EDIT commands to move DOT. EDIT commands are one or two characters long, and some require arguments (e.g., you follow the search command with a string of text for which to search).

2.0 EDITING A NEW FILE

Before you can begin to enter text into an empty file, you must create the file by using the MAKE program. Type MAKE and the specification you want to assign to the new file. For example:

```
.MAKE DSK1:NEWFIL.TXT[100,2] RET
```

You may only create a file in your own account or in an account within your own project.

After you have used the MAKE command, you can now use EDIT to enter text into the file (see below).

3.0 EDITING AN EXISTING TEXT FILE

You may use EDIT on any sequential file that contains ASCII characters. To edit an existing file, type:

.EDIT Filespec (RET)

where Filespec is the name of the file you want to edit. The default EDIT extension is .MAC. After you hit RETURN, you see the EDIT prompt: *. You are now ready to enter EDIT commands. DOT initially points to the first character in your file. When you exit EDIT, the editor renames your original disk file to a .BAK extension (for BACKUP), and saves your edited copy under the original file's name and extension.

4.0 SPECIAL CHARACTERS

When entering text and commands to EDIT, you may use the RUB key (also labeled RUBOUT, DEL, DELETE, etc.) to erase single characters, and a Control-U to erase an entire line of input. EDIT itself has a group of commands that you must use to delete those characters and lines already part of the file you are editing.

EDIT uses the Escape key (labeled on your keyboard as ESC, ALT MODE, etc.) rather than a RETURN as a command delimiter; this allows you to enter carriage returns as part of your text. When you type an Escape to EDIT, you see the character displayed on your terminal as a dollar sign, $. Whenever you see a $ in this document, the symbol indicates an Escape.

5.0 THE COMMANDS

You may enter EDIT commands either in upper or lower case. You may enter the commands one at a time, ending your input with two Escapes. For example:

```
*C$$
*
```

After the EDIT prompt symbol, *, we entered the Character-advance command, C, which moved DOT ahead in the file by one character-position. To tell EDIT that the command line was complete, we entered two Escapes. EDIT responds with another prompt to let us know that it is ready for another command.

You may also enter commands as a group. For example:

```
*LKT$$
```

The example above tells EDIT to move DOT to the beginning of the next line (L), kill (that is, delete) from DOT to the end of the current line, and type (that is, display) the characters from DOT to the end of the current line.

A line consists of all of the characters between two carriage-return/linefeed character pairs.

Some commands take numeric arguments (e.g., "3D" says delete the three characters after DOT); numeric arguments are decimal numbers, and always precede the command to which they apply. If a command takes a text argument (e.g., "Stext" says search for the word "text"), you must end the text argument with one or two Escapes. (One Escape tells EDIT that the last command is complete, but that it may not yet take action upon the current input line. Two Escapes tell EDIT to go ahead and act upon the current command line.) For example:

```
*IThe "I" command tells EDIT to insert text$10T$$
```

The command line begins with an Insert command, I. EDIT will insert into your text file (at the current DOT position) all text (including carriage returns) following the I command up to a single or a double Escape. The single Escape above terminates the text entry string; next is a display command, 10T, that tells EDIT to display the 10 lines of characters that occur after the current position of DOT. The double Escapes tell EDIT to go ahead and act upon the entire string of commands. You may enter as many lines of commands and text as you wish; EDIT will not take action upon the input until you hit two Escapes.

If you want to cancel a string of input, you can do so by typing a Control-C (as long as you type the Control-C before entering the double Escapes).

5.1 A SUMMARY OF THE EDIT COMMANDS

Below is an alphabetical list of the EDIT commands. (Remember, DOT is the pointer that marks your position in the file.)

A	APPEND - Appends one or more records of the input file to the data buffer if there are at least 2000 free bytes of memory left, and DOT has not reached the end of the file.
C	CHARACTER ADVANCE - Moves DOT forward one character (e.g., C$$).
nC	CHARACTER ADVANCE - Moves DOT forward by "n" characters (e.g., 3C$$).
-nC	CHARACTER ADVANCE - Moves DOT backward by "n" characters (e.g., -5C$$).
0C	CHARACTER ADVANCE - Moves DOT backward to the beginning of the current line.

D DELETE - Deletes the first character after DOT.

nD DELETE - Deletes the next "n" characters after DOT (e.g., 3D$$).

-D DELETE - Deletes the character just behind DOT (e.g., -D$$).

-nD DELETE - Deletes the the previous "n" characters behind DOT (e.g., -20D$$).

0D DELETE - Deletes characters from the beginning of the line up to DOT.

HD DELETE - Deletes entire buffer; that is, deletes as much of the file as is in memory.

E EXIT - Exit to monitor. Outputs data buffer, and rest of input file. Renames new file to original file's name and extension, and renames original file to a .BAK extension.

EG EXIT AND GO - Exits to monitor and, if it is a .BAS or a .MAC file, processes the text file as is appropriate for its filetype. An EG$$ command used on a .BAS file tells the monitor to load in BASIC and compile the file; the EG command used on a .MAC file tells the monitor to load in MACRO and assemble the file.

EQ EXIT AND QUIT - Exits to monitor, but doesn't make the editing changes you entered; the original file is left as is, untouched, and is not renamed to .BAK.

F FREE MEMORY - Prints decimal number of free bytes left in your memory partition.

Gx GET AUXILIARY - Gets auxiliary buffer "x" where the symbol x may be the letters A-Z. Inserts the buffer into the file at the current position of DOT. DOT is moved forward the number of characters inserted.

Itext$ INSERT - Inserts specified text into the file at the current position of DOT. You may insert carriage returns and other special symbols except for those Control-characters discarded by AMOS on input (See Special Insert, below). Remember that the text is not actually inserted until you type two Escapes. For example:

```
*IThis is all one
input; all of the
characters, even the
carriage returns, can
be entered with one
insert command$$
```

nI SPECIAL INSERT - You may insert special Control-characters not usually accepted by EDIT, by preceding the I command with the decimal ASCII code of the character you want to insert (e.g., 12I$$ inserts an ASCII character 12-- a form-feed).

0J JUMP - Jumps DOT back to the beginning of the data buffer (i.e., back to the beginning of the portion of your file that is in memory).

nJ JUMP - Jumps DOT to immediately in front of the "nth" character in the data buffer.

ZJ JUMP - Jumps DOT to the end of the buffer.

K KILL - Kills the characters from DOT to the end of the current line.

nK KILL - Kills the next "n" lines of text past DOT.

0K KILL - Kills from the beginning of the current line to DOT.

-K KILL - Kills from the beginning of the previous line to DOT.

-nK KILL - Kills from the start of the "nth" line behind DOT up to DOT itself.

HK KILL - Kills the entire data buffer.

L LINE ADVANCE - Advances DOT to the beginning of the next line.

nL LINE ADVANCE - Advances DOT forward "n" lines. DOT is positioned at the start of the line.

0L LINE ADVANCE - Moves DOT back to the start of the current line.

-L LINE ADVANCE - Moves DOT back to the start of the previous line.

-nL LINE ADVANCE - Moves DOT backward "n" lines from the current position of DOT, and positions DOT to the start of the line.

Linefeed LINE ADVANCE AND TYPE - Typing a linefeed (Control-J) performs same function as LT$$; that is, advances to the front of the next line, and displays that line. A backspace (Control-H) performs the same function as the -LT$$ command; that is, moves DOT back to the start of the previous line, and displays that line.

Ntext$ WHOLE FILE SEARCH - Searches the current data buffer, beginning at DOT, for the first occurrence of "text". If the search within the current data buffer is not successful, EDIT writes that data buffer out to the disk, and brings in more text; DOT is reset to the beginning of that buffer, and the search begins again. This process continues until "text" is found, or until

the end of the file has been reached. If "text" is found, DOT is positioned just after it; if "text" is not found, EDIT displays an error message: [SEARCH FAILED], and any commands occurring in the input string after the search command are aborted.

nNtext$ WHOLE FILE SEARCH - Same as above, except that search stops at the "nth" occurrence of "text" (e.g., 10Nsilicon$).

N$ WHOLE FILE SEARCH - Same as Ntext$, except that it uses the last text string that you entered to a search command.

nN$ WHOLE FILE SEARCH - Same as nNtext$ except that the last search string entered is used.

R REVERSE - Same as -C command.

nR REVERSE - Same as -nC command.

0R REVERSE - Same as 0C command.

-R REVERSE - Same as C command.

-nR REVERSE - Same as nC command.

Stext$ SEARCH - Searches the data buffer beginning with DOT for the first occurrence of "text". Positions DOT just after "text" if it finds it; otherwise it displays the message: [SEARCH FAILED], DOT is positioned to the front of the buffer, and the rest of the commands in the input string are aborted.

nStext$ SEARCH - Same as Stext$, but EDIT searches for the "nth" occurrence of the search string "text".

S$ SEARCH - Same as Stext$ above, but EDIT uses the last search string entered.

nS$ SEARCH - Same as nStext$, but uses the last search string entered.

FSoldtext$newtext$$

SEARCH-AND-REPLACE - Searches for "oldtext" and replaces it with "newtext". The command FNoldtext$newtext$$ performs the same function, but on the entire file, rather than on just the current data buffer.

T TYPE - Displays the characters from DOT to the end of the line.

nT TYPE - Displays "n" lines of characters starting from DOT.

0T TYPE - Displays the characters from the beginning of the current line up to DOT.

-T TYPE - Displays the characters from the start of the previous line up to DOT.

-nT TYPE - Displays from the "nth" line behind DOT up to DOT.

Vx VERIFY - Verifies auxiliary buffer contents, where the symbol "x" is a character from A-Z. Lists the contents of the auxiliary buffer.

Xx SAVE - Saves, in auxiliary buffer "x", the characters from DOT to the end of the current line. (X is a character A-Z.) The previous contents of the auxiliary buffer are lost.

nXx SAVE - Saves "n" lines past DOT in auxiliary buffer "x". ("x" is a character A-Z.)

0Xx SAVE - Saves from the beginning of the current line to DOT in auxiliary buffer "x".

-nXx SAVE - Saves, in auxiliary buffer "x", from the start of the "nth" line previous to the line DOT is in up to DOT.

;text$ SEMICOLON INSERT - Performs same function as the Itext$ command, except that a semicolon is placed at the start of the inserted text.

TABtext$ TAB INSERT - Performs same function as the Itext$ command, except that a TAB character is placed at the start of the inserted text.

SPACEtext$ SPACE INSERT - Performs same function as the Itext$ command, except that a space is placed at the start of the inserted text.

n<...> REPEATS - All of the commands within the angle brackets are repeated "n" times (e.g., 10<FSprimpt$print$>$$ tells EDIT to search for the word "primpt" and replace it with "print" ten times). All EDIT commands can be executed in a repeat, including other repeats. The maximum nesting level for repeats is eight. An error message and an abort occurs if you exceed the nesting limit; search failures also abort repeats.

If you omit "n" the group of commands repeat endlessly until an error occurs, or until a Control-C is typed. Often used to replace ALL occurrences of an item (e.g., <FSregistrar$register$>$$).

January 1979

PROGRAM DESIGN LANGUAGE FORMATTING SYSTEM

1.0 INTRODUCTION

The Program Design Language Formatting System is a tool that helps you to produce a program-design document.

The first step in creating a program-design document is to use one of the Alpha Micro text editors (EDIT or VUE) to write a document. Write the document in the form set by the Program Design Language-- PDL (see Section 2.0, "Program Design Language").

When you exit the text editor, and are again at the AMOS command level, you may use the Program Design Language Formatting System (PDLFMT) to transform your text file into a finished program-design document.

PDLFMT produces a document which contains the following:

1. Table of Contents.

2. Formatted Design Listing - Each procedure in the program is listed on a separate, numbered page, with the page numbers that refer to other procedures in the margin. Each page indents the text to show control-structure nesting.

3. Reference Trees - Indented listing shows how procedure references are nested.

4. Cross Reference - Alphabetical listing of all sections and procedures. An index of where the sections and procedures appear, and where references to them appear (page and line numbers).

2.0 PROGRAM DESIGN LANGUAGE

A program design written in PDL has this form:

```
/T  Design Title
/S  Section Name
Text describing the section.....
.................................
/P  Procedure Name
Text giving the procedure design...
.................................
/P  Procedure Name
.........  Text  ..................
.................................
/S  Section Name
.........  Text  ..................
/R
Procedure name
Procedure name
.................................
```

NOTE: the slash (/) must be the first character on the line.

1. /T Design Title - Specifies the name of the program design. This title appears on every page of the finished document. (/T is called the title command.) The title command must always be the first command in the design, and must always be present.

2. /S Section Name - Specifies the start of a new section of procedure designs. The section name specified will appear on the pages of the finished design document as subtitles. After the section command is free-form text that describes the section. (This text may be any length or form that you want, but no line of it may begin with a /.)

3. /P Procedure Name - Specifies the start of a procedure design, and assigns Procedure Name as the name of that design. Any time the Procedure Name occurs as a statement within a procedure design, PDLFMT considers that occurrence as a reference to the procedure design.

4. /R Reference Tree - Specifies the start of a list of procedure names on successive lines. Each procedure name is a root of a reference tree listing. The Reference Tree command is optional, but if you do include it in your document, it must be the last command in the design.

2.1 Procedure Design

A procedure design consists of a sequence of statements. You may label each statement. (Labels are an alphanumeric identifier followed immediately by a colon.) PDLFMT will indent labels by -2 in the final document.

You may also precede each statement by one of the keywords: IF, ELSEIF, ELSE, ENDIF, DO, ENDO, or ENDDO. PDLFMT uses these keywords when it formats your design document. The rest of the line after an IF or ELSEIF is considered a condition; PDLFMT will not consider it a potential procedure reference.

Statements are sequences of text that end with carriage returns. They may contain embedded comments enclosed in parentheses. If the text of a statement (ignoring the comments) matches the name of a procedure or section, PDLFMT considers it a reference to that procedure or section.

You may continue statements on one or more lines by placing an ampersand (&) at the beginning of succeeding lines.

2.2 Control Structures

You can use the keywords IF, ELSEIF, ELSE, ENDIF, DO, and ENDDO (or ENDO) to indicate a variety of control structures. The paragraphs that follow give some idea of the possibilities.

2.2.1 The IF Construct - The IF construct provides the means for indicating condition execution. It corresponds to the classic IF...THEN...ELSE construct found in Algol-60 and PL/I, augmented by the ELSEIF of languages such as Algol-68. The ELSEIF is used to prevent excessive indentation of levels when cascaded tests are used.

The general form of the construct is:

```
IF condition
 one or more statements
 ...
ELSEIF condition
 one or more statements
 ...
ELSEIF condition
 one or more statements
 ...
ELSE
 one or more statements
 ...
ENDIF
```

NOTE: You are allowed any number (including zero) of ELSEIFs, and you are allowed one ELSE at the most.

2.2.2 The DO Construct - Use the DO construct to indicate repeated execution, and for case selection. Indicate the iterative DO by:

```
DO iteration criteria
   one or more statements
   ...
ENDDO
```

You can choose the iteration criteria to suit the problem. Typical criteria begin with the words WHILE, UNTIL, or FOR. WHILE denotes a continuation criteria which is checked before each iteration. UNTIL denotes a termination criteria that is checked after each iteration. FOR denotes a range of items over which the one or more statements are to be applied. Examples:

```
DO WHILE THERE ARE INPUT RECORDS
DO UNTIL "END" STATEMENT HAS BEEN PROCESSED
DO FOR EACH ITEM IN THE LIST EXCEPT THE LAST ONE
```

Provision for premature exit from a loop and premature repetition of a loop are frequently useful. To accomplish this, you can take the statement UNDO to mean that control is to pass to the point following the ENDDO of the loop. Likewise, CYCLE can be taken to mean that control is to pass to the iteration criteria test. If you want to apply UNDO or CYCLE to an outer loop in a nest of loops, you may label any DO and place the label after the UNDO or CYCLE.

You can indicate case selection by:

```
DO CASE selection criteria
```

In general, labels are used in the body of the DO to indicate where control passes for each case:

```
   DO CASE OF TRANSACTION TYPE
ADD:
      CREATE INITIAL RECORD
DELETE:
      IF DELETION IS AUTHORIZED
         CREATE DELETION RECORD
      ELSE
         ISSUE ERROR MESSAGE
      ENDIF
CHANGE:
      INCREMENT CHANGE COUNT
      CREATE DELETION RECORD
"OTHER":
      ISSUE ERROR MESSAGE
   ENDO
```

3.0 OPERATING INSTRUCIONS

Call PDLFMT from the AMOS command level by typing:

.PDLFMT Filespec(RET)

where Filespec specifies a design file prepared with a text editor. If you omit a filename extension, PDLFMT assumes a .PDL extension. The formatted design document is placed in the file Filespec.LST.

A demonstration file, TLGRAM.PDL, is included with PDLFMT.PRG on the dealer distribution disk. For a demonstration of PDLFMT, type:

.PRINT TLGRAM.PDL(RET)
.PDLFMT TLGRAM(RET)
.PRINT TLGRAM(RET)

4.0 ERROR MESSAGES

X IS AN ILLEGAL COMMAND - BYPASSING LINE

PDLFMT found a command /X in the design document, but X is not a legal command. Use /T, /S, /P, or /R.

REPEATED DEFINITION: xxxxxxx, REFERENCES WILL BE TO LAST OCCURRENCE

xxxxx is a section or procedure name that occurred in a section or procedure command more than once.

5.0 ACKNOWLEDGMENTS

PDLFMT is based on PDL and its processor, as described in "PDL - A Tool for Software Design," by Stephen H. Caine and E. Kent Gordon of Caine, Farber, and Gordon, Inc.

AMOS Software Update Documentation
AMOS Release 4.5
April 1981

SYSTEM OPERATOR'S INFORMATION

This section contains the following documents:

The System Initializaton Command File, Revision A04

Setting Up the Line Printer Spooler, Revision A02

Memory Management Option, Revision A01

Defining Switchable System Memory, Revision A02

Configuring Floppy Disk Drivers, Revision A01

AMOS Version 4.4 Method of Handling Bad Disk Blocks

Software Installation Instructions for the AM-120

Software Installation Instructions for the AM-710 Memory Board

Software Notice for AM-410 Users, Revision A03

Disk Labeling Procedures

Disk Maintenance Procedures for the System Operator, Revision A03

Defining Non-system Disk Devices, Revision A01

Disk Drivers and Formats, Revision A03

Generating System Monitors, Revision A01

Using the Magnetic Tape Utility Programs

The Magnetic Tape File Backup Programs

Building a Terminal Driver (The NEWTRM Program)

AMOS 4.5 SOFTWARE UPDATE DOCUMENTATION PACKET

These documents reflect AMOS Versions 4.5 and later

ALPHA MICROSYSTEMS
17881 Sky Park North
Irvine, CA 92714

THE SYSTEM INITIALIZATION COMMAND FILE

April 1981
Revision A04

This document reflects AMOS versions 4.5 and later

ALPHA MICROSYSTEMS
17881 Sky Park North
Irvine, CA 92714

Table of Contents

1.0 INTRODUCTION

The Alpha Micro Operating System has been designed so that you can "customize" it for your particular hardware set-up (that is, adapt it to run with your terminals, your disks, etc.) In fact, you MUST do that initial system software installation if your system software has not already been set up so that it conforms to your machine's hardware configuration. Because a user's hardware and system needs can keep changing as a system grows, Alpha Micro has created a simple mechanism for adapting the operating system to reflect those changes: the system initialization command file (the SYSTEM.INI).

The SYSTEM.INI is a special kind of command file. (A command file is a text file that contains system commands; the system reads and obeys the instructions it reads from the file.) Whenever the system is powered up or reset, it consults the SYSTEM.INI to find out what devices you use on the system and what special programs and functions you want to add to the operating system area of memory.

The rest of this document discusses the elements of the SYSTEM.INI and the changes that you can make to the file to reflect changes in your system configuration. (NOTE: While you read this document, keep in mind that the terms "operating system," "monitor," and "system" are roughly interchangeable in the pages that follow.)

1.1 Modifying the SYSTEM.INI

CAUTION: Let's assume that you have a SYSTEM.INI file that gets your system up and running, and you want to change it to reflect some hardware additions or changes. Before you edit the file, it's a good idea to make a backup copy of your System Disk so that you can bring the system up again, even if something goes wrong with your modified SYSTEM.INI.

If you change the file that works (and somehow your new SYSTEM.INI doesn't work), you won't be able to get the system up and running off of that disk. Before you can again use the disabled disk as a System Disk, you'll have to bring the system up off of another System Disk, and transfer over a copy of a good SYSTEM.INI. The system will not come up unless you have a valid SYSTEM.INI file on your System Disk.

If you are running the system off physical drive zero (i.e., the fixed-platter in a CDC Hawk hard-disk system, the first fixed disk in the CDC Phoenix hard-disk system, or Drive Zero in a floppy-disk system) you can make a copy of the good SYSTEM.INI under a different name, edit THAT version of the file, and then use the MONTST command to test the new copy. This

procedure leaves you with a valid SYSTEM.INI under its original name, so that if you have to reset the computer, the system will be able to come up under the control of the original, valid SYSTEM.INI.

To use the MONTST command, type MONTST, SYSTEM.MON, a comma and the name of the modified command file:

```
.MONTST SYSTEM.MON,NEWSYS.INI (RET)
```

If you are not running off physical drive zero, you will not be able to use the MONTST command, and will have to modify the SYSTEM.INI file itself; in that case, make sure that you have a valid System Disk that you can bring the system up on before you change your SYSTEM.INI file.

To change the SYSTEM.INI to reflect your own needs and hardware configuration, edit it with one of the text editors on the system (EDIT or VUE). The SYSTEM.INI resides in area [1,4] of the System Disk. After you change the file, reboot the system by pressing the reset button or by using the MONTST command (see above).

1.2 System Start-up

The monitor performs a certain set of procedures while it is coming up. If one of these steps fails, the system will not come up. These steps are:

1. When you press the reset button, the Alpha Micro CPU starts executing instructions at the address set up in its header. (If the CPU is an AM-100, this address is the address of the PROM on the disk controller board; if the CPU is an AM-100/T, the address is the address of a PROM on the CPU board itself.)

2. The program in the PROM transfers itself down into RAM between 31K-32K. (If your memory in these locations is bad or nonexistent, the system start-up will not proceed beyond this step.) If the phantom memory option is installed on the disk controller board, the phantom memory now becomes active.

3. The PROM program (the bootstrap loader) is now in RAM and it begins to execute. It reads in the operating system skeleton monitor, which is a file called SYSTEM.MON in the [1,4] System Disk program area. The loader reads SYSTEM.MON into memory beginning at location zero and extending as far as necessary.

4. When SYSTEM.MON is in memory, it executes the initialization routine (INITIA) within the monitor itself. The purpose of INITIA is to scan memory to determine how much is available, and then to set up a user memory partition in the last 8K of memory. (This user partition is temporary, and is just used to execute the system start-up functions under control of the SYSTEM.INI.)

5. Once INITIA is through setting up the user partition, the monitor reads in the system initialization command file, SYSTEM.INI. Each line in the SYSTEM.INI represents one system function or parameter which determines the characteristics of the running monitor.

6. The monitor executes the commands in the SYSTEM.INI just as it would the commands in any other command file. Because this command file is the system initialization file, however, the monitor performs some of the commands in the SYSTEM.INI differently than it would the same commands after the system is completely up. The execution of certain system commands (e.g., JOBS, TRMDEF, DEVTBL, etc.) performs the actual system generation.

During system start-up, certain programs cause the monitor to create new areas at the end of itself; these areas include terminal definition blocks, terminal drivers, job control blocks, device tables, memory bank tables, system queues, and disk bitmaps. Your SYSTEM.INI may optionally specify a list of programs to be added to the resident monitor area of memory during system start-up.

The size of the monitor is not fixed, but is expanded during system start-up. This is why INITIA allocates the initial user partition in the top 8K portion of memory. This gives the monitor room as it expands so that it won't overlap the user partition. (The monitor executes the SYSTEM.INI in that user partition.)

NOTE FOR BANK-SWITCHED SYSTEMS: If your system uses memory management (i.e., it bank switches memory), the system uses the top 8K of Bank Zero to process the SYSTEM.INI. You can go ahead and allocate that portion of bank zero to a job as a user memory partition (via the JOBMEM command), but DO NOT try to use that job to run anything until after the SYSTEM.INI is fully processed and the system is up and running. (That is, do not use a FORCE command within the SYSTEM.INI to force input to the job.) NOTE: For information on memory management via bank switching, see the Alpha Micro Integrated Systems User's Guide, (DWM-00101-00).

2.0 A SAMPLE SYSTEM INITIALIZATION COMMAND FILE

This page and the next contain a typical SYSTEM.INI file for a system that uses memory management. The sections that follow base their discussions on this sample SYSTEM.INI.

```
:T
JOBS JOB1,JOB2,JOB3,JOB4,SPOOL
;
TRMDEF TRM1,AM300=1,ADM3,100,100,200
TRMDEF TRM2,AM300=2:6,SIL700,100,80,30
TRMDEF TRM3,AM300=3:16,SOROC,100,100,100
TRMDEF TRM4,AM300=5:16,SOROC,100,100,100
TRMDEF PRNTR,AM300=6:10,SOROC,100,100,20        ; Define printer
TRMDEF NULL,PSEUDO,NULL,25,25,2                 ; Define
;                                                 pseudo-terminal.
;
MEMDEF 100,0,14                 ; 32K switchable (Bank 0)
MEMDEF 101,14,0                 ; 32K switchable (Bank 1)
MEMDEF 102,14,0                 ; 32K switchable (Bank 2)
MEMDEF 101,3,0                  ; 32K switchable (Bank 3)
MEMDEF 102,3,0                  ; 32K switchable (Bank 4)
;
SYSMEM 4:100000-160000          ; Define 24K as switchable
;                                 system memory.
;
MEMERR 250                                      ; Initialize memory
;                                                 boards to detect
;                                                 double-bit errors.
DEVTBL DSK1,DSK2,DSK3,DSK4,DSK5,HWK0,HWK1
DEVTBL AMS0,AMS1,TRM,RES,MEM,/MTM               ; Define devices.
;
BITMAP HWK,606,0,1
BITMAP AMS,39,0,1
BITMAP DSK,1818,0/S             ; Put DSK bitmaps in
BITMAP DSK,1818,1/S             ; switchable system memory.
BITMAP DSK,1818,2/S
BITMAP DSK,1818,3/S
BITMAP DSK,1818,4/S
BITMAP DSK,1818,5/S
;
QUEUE 20                        ; Add 20 more queue blocks.
CLKFRQ 60                       ; Set clock frequency
;
SYSTEM HWK.DVR[1,6]
SYSTEM VUE.PRG[1,4]
SYSTEM
;
SET DSKERR                      ; Enable full disk error reporting
SET GUARD                       ; for this job.
```

```
;
ATTACH TRM2,JOB2
JOBMEM JOB2 1:100000-177376     ; Give JOB2 32K of memory
KILL JOB2                       ; Initialize JOB2
FORCE JOB2 LOG DSK2:22,2        ; Log JOB2 into the system.
FORCE JOB2 SET DSKERR           ; Enable full disk error reporting
;
ATTACH TRM3,JOB3
JOBMEM JOB3 2:100000-177376     ; Give JOB3 32K of memory
KILL JOB3                       ; Initialize JOB3
;
ATTACH TRM4,JOB4
JOBMEM JOB4 3:100000-177376     ; Give JOB4 32K of memory
KILL JOB4                       ; Initialize JOB4
;
ATTACH NULL,SPOOL               ; Attach line printer spooler
;                                 job to pseudo-terminal.
KILL SPOOL                      ; Initialize spooler job.
;
FORCE SPOOL                     ; Force input to spooler job.
MEMORY 4K                       ; Give spooler 4K sharable memory.
LOG 1,2                         ; Log job into system.
LPTINI PRNTR.INI                ; Set up line printer spooler.

WAIT SPOOL                      ; Wait for SPOOL to finish before
;                                 proceeding.
MOUNT DSK1:                     ; Mount the six Phoenix
MOUNT DSK2:                     ; logical units.
MOUNT DSK3:
MOUNT DSK4:
MOUNT DSK5:
;
MEMORY 0                        ; Give JOB1 rest of sharable
                                ; memory not used by monitor.
```

The following sections discuss the elements of the SYSTEM.INI file and their functions.

3.0 THE TRACE FUNCTION (:T)

The first line in a SYSTEM.INI is:

```
:T
```

This turns on the trace function of the command file processor. That is, the ":T" tells the monitor to display the command file on a terminal while it is processing the SYSTEM.INI. When the monitor first finds the SYSTEM.INI, the process of reading and processing the file is initially under control of the first job (to which no terminal has yet been attached.) As soon as a terminal is defined (by the first TRMDEF command), the monitor displays the remainder of the SYSTEM.INI on that terminal screen as it executes the file.

If you don't want the system to display the SYSTEM.INI as it processes it, omit the :T from the file.

NOTE: The semicolons in the example above indicate comment lines. The system does not process comment lines, but does display them as it processes the SYSTEM.INI if a :T appears in your file.

4.0 ALLOCATING JOBS (THE JOBS COMMAND)

The first command in the SYSTEM.INI (JOBS) tells the monitor what jobs to allocate in the system, and gives a name (1 to 6 characters) to each job. Each job named in the JOBS line causes one JCB (Job Control Block) area to be allocated in system memory. (Each job's JCB maintains information about that job for the system.)

If you wish to allocate more jobs than will fit on one line, you may have as many JOBS commands as you wish as long as they are before the first TRMDEF command in the file.

Each job allocated takes up about 150 words of system memory. Note that the JOBS command does not automatically associate a terminal with a job; this is done using the TRMDEF command to define a terminal, and the ATTACH command to associate that terminal to a specific job. You must explicitly attach terminals to jobs in this way. In the sample SYSTEM.INI in Section 2.0, the JOBS command line looks like this:

```
JOBS JOB1,JOB2,JOB3,JOB4,SPOOL
```

This line tells the system to allocate JCBs for five jobs.

5.0 DEFINING TERMINALS (TRMDEF)

After the JOBS command(s) must come one TRMDEF command for each terminal you want connected to your system. Every terminal has a name by which it is referenced by the monitor (1 to 6 characters), a specific hardware interface to which it is connected, and a terminal driver (a program that does any necessary character conversions). The TRMDEF command also specifies the size of the various buffers that are used in the data transfers between the terminal and the computer. The TRMDEF command takes this form:

```
TRMDEF Name,Interface,Terminal,In-width,In-buffer,Out-buffer{,HOG}
```

When the monitor processes a TRMDEF command line, it builds a terminal definition unit in system memory which includes all of the elements above. The system then loads in the correct terminal driver and interface driver and links them to the definition unit; then it executes the interface driver which performs any necessary interface initialization.

NOTE: The buffer size values that you specify in your TRMDEF command lines affect the total size of the monitor.

After the monitor has finished processing the SYSTEM.INI, the TRMDEF command performs a different function. After the system is up and running, TRMDEF becomes a user command. At this time, TRMDEF displays on the screen the current terminal configuration of the system in a form similar to the original TRMDEF command lines in the SYSTEM.INI. The octal number that follows each terminal name is the absolute address in the monitor of the terminal definition unit for that terminal. (That information is sometimes useful when debugging the terminal service system; the general user can ignore it.)

In the sample SYSTEM.INI in Section 2.0, the first TRMDEF command line looks like this:

```
TRMDEF TRM1,AM300=1,ADM3,100,100,200
```

Now we'll discuss the different elements of the TRMDEF command line:

5.1 Name

The terminal name consists of one to six alphanumeric characters chosen by you. Every terminal on the system must have a different name, although you may choose to use a terminal name that duplicates a job name or a program name. The system uses the terminal name to identify the terminal that you want to attach to a job or that you want to access using the TRM device driver.

5.2 Interface

The interface is the hardware board that connects the terminal to the system bus. The interface statement gives the name of the terminal interface and its I/O port address on the system. (The I/O port address follows the name of the terminal interface, and is separated from it by an equals sign--e.g., PS3=1.) As the system processes each TRMDEF command line, it loads the proper interface driver into system memory from area [1,6] of the System Disk. (If the driver is already in memory because of a previous TRMDEF command line, the system does not load it in again.) Interface drivers are the programs that actually transfer data between the terminal and the terminal interface boards; these programs have the extension .IDV and must reside in account [1,6] of the System Disk. The interface drivers have the same name as the interface boards they work with. The currently defined interface drivers available with the system are:

5.2.1 PS3

Noninterrupt driver for the Processor Technology 3P+S serial interface board. The interface statement must include the octal address of the control status port for the serial side of the board (e.g., PS3=0, PS3=20, etc.)

5.2.2 IMSIO

Noninterrupt driver for the IMSAI SIO-2 serial interface board. The interface statement must include the octal address of the control status port for the selected side (A or B). The address of side A is 3 greater than the board I/O port address, while the address of side B is 5 greater than the I/O port address. Examples: IMSIO=3, IMSIO=5, IMSIO=23, IMSIO=25, etc.

5.2.3 AM100T

Interrupt-driven driver for the two serial ports contained on the AM-100/T CPU. The command format is identical to that for the AM310 driver, except that the I/O port must be either 0 or 1.

5.2.4 AM300

Full interrupt driver for the six-port Alpha Micro serial interface board. If you are using an AM-300 board, you may optionally include a code that selects the terminal baud rate. (We give these codes below.) The interface statement includes an I/O port address (1-6) and the optional baud rate code (separated from the I/O port address by a colon). The baud rate code is an octal number (0-17). If you omit the code, the AM-300 driver assumes a rate of 19200 baud. An AM300 interface statement takes the form:

AM300=I/O port address{:baud rate code}

Some examples of AM-300 interface statements:

```
AM300=1          (port 1 at 19200 baud)
AM300=2          (port 2 at 19200 baud)
AM300=3:6        (port 3 at 300 baud)
AM300=5:12       (port 5 at 2400 baud)
```

The baud rate codes that you can specify in an interface statement are:

:0	50	baud
:1	75	baud
:2	110	baud
:3	134.5	baud
:4	150	baud
:5	200	baud
:6	300	baud
:7	600	baud
:10	1200	baud
:11	1800	baud
:12	2400	baud
:13	3600	baud
:14	4800	baud
:15	7200	baud
:16	9600	baud
:17	19200	baud

5.2.5 AM310

Fully interrupt-driven driver for the four-port Alpha Micro communications controller. If you are using the AM-310, you may optionally include a code that selects the terminal baud rate. The interface statement includes an I/O port address (0-3) and an optional baud rate code (separated from the I/O port address by a colon). The baud rate code is actually a two-byte command. The low byte is sent to Mode Register 1 (MR1) of the Programmable Communications Interface on the AM-310 board. The high byte is sent to Mode Register 2 (MR2). (For more information, see the AM-310 Technical Manual.) If you omit the baud rate code, the AM-310 driver assumes a rate of 19200 baud. The interface statement for the AM-310 board takes this form:

```
AM310=I/O port address{:baud rate code}
```

Some examples of AM-310 interface statements:

```
AM310=0          (the first port at 19200 baud)
AM310=3:37316    (the fourth port at 9600 baud)
```

The baud rate codes that you can specify are:

:30316	50	baud
:30716	75	baud
:31316	110	baud
:31716	134.5	baud
:32316	150	baud
:32716	300	baud
:33316	600	baud
:33716	1200	baud
:34316	1800	baud
:34716	2000	baud
:35316	2400	baud
:35716	3600	baud
:36316	4800	baud
:36716	7200	baud
:37316	9600	baud
:37716	19200	baud

5.2.6 AX310

Fully interrupt-driven driver. Allows you to run with multiple AM-310 interface boards in your system. The command format is identical to that of the AM310 driver, and the baud rate codes you may specify are the same.

5.2.7 AM120

The AM120 driver allows you to use the two serial I/O ports on the AM-120 Auxiliary I/O Controller for terminals or printers. You must also use this driver if you want to use the AM-120 parallel ports in interrupt driven mode. For more information on the AM-120 board, see the document Software Installation Instructions for the AM-120 in the "System Operator's Information" section of the AMOS Software Update Documentation Packet.

The interface statement portion of the TRMDEF for this interface board is identical to that of the AM-310 board, and uses the same optional baud rate codes as the AM-310 (see Section 5.2.5, above). For example:

```
TRMDEF TERM1,AM120=1:37316,SOROC,100,100,100
```

5.2.8 PSEUDO

You normally use the PSEUDO driver with either the PSEUDO or NULL terminal specifications in a TRMDEF command line. This sets up a software interface driver that communicates with a pseudo terminal for those occasions when you have a job that doesn't need a real, hardware-controlled terminal for processing (e.g., a print spooler job). The PSEUDO interface driver is built into the monitor, and does not reside in area [1,6] of the System Disk.

5.3 Terminal

The terminal statement tells the system what kind of terminal is connected to the interface board, and thus what kind of terminal driver to load into system memory from area [1,6] of the System Disk. Different terminals process characters differently. A terminal driver is the program that does the necessary code conversion and character processing required by the particular terminal that it supports. It is the terminal driver, then, that takes care of the special functions (e.g., cursor control, Control-U, rubout, null characters after RETURNs, etc.) that differ between terminal types.

Terminal drivers have the extension .TDV and are sharable; that is, a given driver is loaded only once into system memory, no matter how many terminals of the same type are defined. Some of the terminal drivers currently available on the system are:

5.3.1 ADM3

The driver for the Lear Siegler ADM3 dumb terminal. When this driver receives a rubout, it tells the terminal to backspace and erase the character in that position. It processes a Control-U by backspacing and erasing the entire line. It also reverses the case condition of the RUB key so that you do not need to shift rubouts, but you do need to shift underscores. Most people prefer this convenience, but you can disable this feature by setting the BIT 8 switch on your terminal to position "1."

5.3.2 SOROC

Similar to the ADM3 driver, but contains the codes for the SOROC CRT-terminal.

5.3.3 HAZEL

Codes for the HAZELTINE 1500, 1510, and 1520 CRT-terminals.

NOTE: If you used earlier versions of the HAZEL driver, you were not able to use the standard VUE commands because the terminal driver had to make some special character translations to allow you to use the cursor control keys of the 1510 and 1520 models. (For example, an end-of-line command was a Control-G instead of the standard Control-N.) For the sake of convenience, the current HAZEL driver allows you to use the standard VUE control commands; however, you may not use the cursor control keys. (To move the cursor, use the Control-H, Control-J, Control-K, and Control-L commands.) The sources for the HAZEL driver are available on the "Driver Sources Diskette" and on the Phoenix and Hawk System Disk packs.

5.3.4 ACTIV

Codes for the ACT-iv CRT-terminal.

5.3.5 DMEDIA

Codes for the Data Media ELITE 1520 CRT-terminal.

5.3.6 ADDS

Codes for the ADDS REGENT 100 terminal.

5.3.7 TELTYP

Driver for standard KSR and ASR Teletypes. Rubouts echo by typing the rubbed out characters between backslashes. A Control-U echoes as "^U" followed by a carriage return/line feed, which takes the printing mechanism to the next line. The line that was ended with a Control-U is ignored by the computer. This driver does no other code conversions, and no null characters are appended after a line feed.

5.3.8 SIL700

Driver for the Texas Instruments Silent-700 terminal. This driver is identical to the TELTYP driver above, except that it adds 8 null-characters at the end of every line-feed to prevent character overrun.

5.3.9 PSEUDO

Driver for the software-controlled pseudo terminals. This driver merely stops echoing of input characters and allows buffering of input to and from the controlled job. Use it only with the PSEUDO interface statement. The PSEUDO terminal driver is built into the monitor, and is not in the [1,6] area of the System Disk.

5.3.10 NULL

Driver identical to the PSEUDO driver above, except that it discards the terminal output from the job, instead of buffering it to wait for some other job to pick it up. Use this terminal driver when you want to control a job whose terminal output is of no importance (e.g., the print spooler). When using this driver for a job like the line printer spooler, you will usually use the FORCE command to send commands and data to that job; make sure that the buffer sizes you define in the pseudo-terminal's TRMDEF statement (see below for information on buffers) are large enough to accept the lines of data that you are going to FORCE to the job.

5.3.11 Other Terminal Drivers

Other terminal driver programs exist that support a variety of terminals and printers. Refer to the current AMOS Release Notes for a list of the terminal drivers in account DSKO:[1,6].

5.3.11.1 Building Your Own Terminal Driver (NEWTRM)

With Release 4.5, Alpha Micro offers a new program, NEWTRM, that you use outside of the SYSTEM.INI to build your own terminal driver for a particular terminal. See Building a New Terminal Driver (The NEWTRM Program) in the "System Operator's Information" section of the AMOS Software Update Documentation Packet for more information.

5.4 In-width

The in-width statement specifies the maximum terminal line-width allowed before the system begins to discard input characters. Allowing a large width, such as 100, gives an added margin of safety when typing long lines.

5.5 In-buffer

There are times when the system cannot immediately process characters that you type from the keyboard. Instead, it stores the characters in an input buffer until it can get around to them. The in-buffer statement specifies the size of this buffer. The number that you specify, then, is also the number of characters that you can type ahead of the system before it starts to discard characters. When you've reached the end of the type-ahead buffer, the system echoes any additional characters as bell codes and discards them. If you want to be able to type ahead a full line, make this parameter at least as large as the in-width value.

5.6 Out-buffer

The out-buffer statement specifies the size of the terminal output buffer. This is the buffer that holds the characters that the system sends to the terminal. The terminal empties this buffer at its own speed. The system allocates two output buffers of the size specified in the out-buffer statement. The system allows a job to stay active until it fills these buffers; then the job is put into the terminal output wait state. In general, specify larger output buffers for faster terminals, and specify smaller output buffers (perhaps only 10 characters or so) for slower terminals.

For more details on these input and output buffers, see the document, Terminal Service System. Remember: large buffers result in a larger resident monitor size.

5.7 HOG

The last element of the TRMDEF statement is an option that you may wish to include for terminals that run noninterrupt hardware, such as the 3P+S or IMSAI SIO boards. The system limits the output of these boards to 60 characters per second whenever any job is demanding CPU time.

The HOG statement tells the system to use any time remaining in the job's scheduled quantum to output characters to the terminal at maximum speed, instead of giving that time over to another job for its task. This, of course, reduces the total system throughput, since the time spent waiting for the terminal interface to become ready is lost for any other tasks.

Interrupt-driven interface boards (such as the AM-300) ignore the HOG statement; they always run at maximum speed regardless of CPU demands.

There is a bug in HOG. If the output buffer is less than 512 and you fill the out-buffers up, printing will stop until you type a character. Because of this problem, don't use an out-buffer of less than 513 when you include the HOG statement in a TRMDEF command line.

6.0 MEMORY MANAGEMENT (MEMDEF)

The sample SYSTEM.INI in Section 2.0 makes use of the memory management option. For detailed information on memory management and the use of the MEMDEF statement, see the Alpha Micro Integrated Systems User's Guide, (DWM-00101-00) and the document Memory Management Option (in the "System Operator's Information" section of the AMOS Software Update Documentation Packet). Briefly, however-- memory management enables you to expand the amount of memory the system as a whole can access by allowing you to address more than 64K of memory. In one method of memory management (called "bank switching"), each user is still limited to a maximum of 64K, but the system can select between several different sets of memory banks (i.e., it can bank switch memory).

To tell the system that you are bank switching memory, use the MEMDEF command in the SYSTEM.INI to define switchable memory banks. The example in our SYSTEM.INI:

```
MEMDEF 100,0,14          ; 32K switchable (Bank 0)
MEMDEF 101,14,0          ; 32K switchable (Bank 1)
MEMDEF 102,14,0          ; 32K switchable (Bank 2)
MEMDEF 101,3,0           ; 32K switchable (Bank 3)
MEMDEF 102,3,0           ; 32K switchable (Bank 4)
```

defines five switchable memory banks of 32K each, and a sharable, non-switchable area of 32K which contains the monitor and programs that all users can access.

(The numbers that follow the MEMDEF statements depend on the type of memory boards you are using, and on the particular memory configuration you are setting up. For the example above, we used three Piiceon 64K memory boards.

Refer to the memory management documentation mentioned above for instructions.)

If you are not bank switching memory (for example, if you do not have more than 64K of memory on your system), do not include the MEMDEF command in your SYSTEM.INI.

7.0 SWITCHABLE SYSTEM MEMORY (SYSMEM)

As the variety of devices that you can add to your system grows, it becomes more likely that you will have a greater number of DEVTBL entries and BITMAP commands in your SYSTEM.INI. Each new type of device that you add to the system increases the size of your monitor, because the software and control tables for that device must be incorporated into the monitor area of memory. Beginning with AMOS version 4.3, if your system uses memory management, you may now set aside an area of switchable memory for the use of the monitor. Currently, the only use you may make of this "switchable system memory" is to place bitmaps in it. (See Section 11.0, "The Disk Bitmap (BITMAP)," for information on placing bitmaps in switchable system memory.) In the future, you may be able to allocate other sections of the monitor to switchable memory.

SYSMEM tells the system what area of switchable memory you want to set aside for system use. SYSMEM takes this form:

```
SYSMEM Bank#:StartAddress-EndAddress
```

where Bank# indicates the memory bank you want to allocate to system memory, and StartAddress and EndAddress give the beginning and ending memory addresses within that bank of the block you want to set aside. Be sure and allocate as much memory as you need for the bitmaps you want to place into switchable system memory. Place the SYSMEM command after the MEMDEF commands and before the BITMAP commands.

After the system is up and running, SYSMEM becomes a user command. SYSMEM followed by a RETURN tells you what switchable area of memory is set aside for the system.

There are some important restrictions on the use of SYSMEM; for more information, refer to Defining Switchable System Memory in the "System Operator's Information" section of the AMOS Software Update Documentation Packet.

8.0 MEMORY ERROR DETECTION (MEMERR)

The MEMERR command enables double-bit error detection for a Piiceon 32K-word memory board. It also initializes the Alpha Micro AM-710 128K byte memory board.

8.1 Piiceon 32K Word Memory Boards

MEMERR initializes the Piiceon board by instructing it to abort when a double-bit memory error occurs. Make sure that the Piiceon memory board is properly jumpered for the I/O error port you specify in the MEMERR command line. The error interrupt-enable jumper (jumper 54) must be installed on the memory board.

NOTE: MEMERR was designed to be used with the AM-100/T CPU. If you are using the 32K-word Piiceon memory boards as 64K-byte memory boards (that is, if you are using the AM-100 CPU), you can still use MEMERR if you use an unused error interrupt-enable jumper other than jumper 54 and enable the new line on the AM-100 board. (Jumper 54 is an AM-100/T interrupt line.)

You may assign the same I/O port number to more than one memory board because the system only issues write-status commands to the memory boards.

If you don't use MEMERR, the Piiceon 32K-word memory board automatically corrects single-bit errors, but ignores double-bit errors; if you use MEMERR, the memory board still corrects single-bit errors, but causes the system to halt on a double-bit error.

If the system halts, look at the error light on the Piiceon memory board (a red LED). If the light is on, the system halt occurred because of a double-bit memory error.

Put MEMERR after the TRMDEF commands in your SYSTEM.INI. Include the number of the I/O error port you have assigned to the memory board(s). (This I/O port is usually 250, octal or A8, hex.) For example:

```
MEMERR 250
```

If double-bit errors are frequent on your system, you may want to replace the memory board on which the errors occur.

8.2 AM-710 128K Byte Memory Board

The AM-710 memory board also requires the use of MEMERR. If your system contains AM-710 and Piiceon 32K word memory boards, include on the MEMERR command line the I/O error port assigned to the Piiceon boards (as in the example above). If your system contains only AM-710 memory boards, you must not supply any argument to the MEMERR command. For example:

```
MEMERR          ; No Piiceon boards, only AM-710 boards.
```

Note that the AM-710 board also requires the use of the PARITY command (discussed below).

9.0 PARITY ERROR DETECTION (PARITY)

The AM-710 memory board requires the presence of the PARITY command in your SYSTEM.INI in order to enable parity error detection and reporting. Place the PARITY command after the MEMERR command (also required). PARITY takes this form:

PARITY I/O-port{,I/O-port2,...I/O-portN}

where I/O-port identifies the I/O port address of the one or more AM-710 boards in your system. (One I/O port address must appear for each AM-710 board.) For information on parity error handling, see the document Software Installation Instructions for the AM-710 Memory Board in the "System Operator's Information" section of the AMOS Software Update Documentation Packet.

10.0 THE DEVICE TABLE (DEVTBL)

Following the TRMDEF command lines is the DEVTBL command. It defines the devices that your system can access. If you have more than one disk drive on your system, specify them in the DEVTBL command line. (The system already knows that the System Disk, DSK0:, is present, so don't put DSK0: in the DEVTBL command line.) List all sharable devices before a slash; all private, unsharable devices after it. (A sharable device is one that all users can access, such as a disk drive; a non-sharable device is one that only one user at a time can access, such as a magnetic tape unit.)

If your system has more devices than will fit on one DEVTBL command line, you can have as many DEVTBL command lines as you want, as long as they are not separated by intervening commands.

As the system processes the DEVTBL command line, it builds a device table in system memory. The file system consults the device table for device assignments. (If your devices run under the control of the AM-410, DEVTBL also builds a table in memory that keeps track of any alternate tracks assigned for those devices.)

Here are some sample device names that the system recognizes:

AMS0	(Floppy disk drive that uses Alpha Micro AMOS format.)
STD1	(Floppy disk drive that uses IBM-standard format.)
DDA1	(Floppy disk drive that uses double-density, double-sided AMS format.)
DSK0	(Logical unit zero of the System Device.)
HWK1	(Logical unit 1 of a hard disk drive that runs under control of the AM-500 controller.)
SMD5	(Logical unit 5 of a hard disk drive that runs under control of the AM-410 controller.)
MTU0	(Magnetic tape unit that runs under control of the AM-600.)
TRM	(The generalized terminal service driver. Allows input and output to any terminal connected to the system.)
RES	(Driver that allows you to use system memory as a device: e.g., .DIR RES:LOG.PRG.)
MEM	(Driver that allows you to use user memory partition as a device: e.g.,.COPY MEM: =WRKFIL.PRG.)

NOTE: Several commands on the system require that you have both MEM and RES defined in your DEVTBL as system devices.

After the system is fully up and running, the DEVTBL command becomes a user command that tells you what devices are in the device table in system memory; it also tells you which devices are sharable among users.

11.0 THE DISK BITMAP (BITMAP)

To randomly access information on a disk, the AMOS file structure needs a disk allocation map (a bitmap). The BITMAP command sets up these bitmaps. If you have more than one type of disk controller on the sytem, each kind of controller must have its own device name and separate bitmap areas. (Floppy disk drives, which may use disks in several different formats, must have a different device defined for each type of format.)

The BITMAP command specifies the device name, the number (in decimal) of words that the bitmap buffer needs, and the list of drive numbers that are to share this particular bitmap area. The Hawk hard disk drive requires 606 words per logical device; the Phoenix hard disk drive requires 1818 words per surface. (For information on the number of words needed for the bitmap of a particular floppy disk device, see Configuring Floppy Disk Drivers in the "System Operator's Information" section of the AMOS Software Update Documentation Packet.) You may have as many BITMAP commands as you want; as many as one for each disk drive. Each BITMAP command creates a separate bitmap area.

In the sample SYSTEM.INI in Section 2.0, one BITMAP command line looks like this:

```
BITMAP HWK,606,0,1
```

We could replace it with two BITMAP command lines to create two separate bitmap areas for drives 0 and 1:

```
BITMAP HWK,606,0
BITMAP HWK,606,1
```

The monitor builds one sharable bitmap area in memory for each BITMAP command line it encounters. The BITMAP command specifies the disks to be accessed by SYSTAT when SYSTAT prints the number of free blocks left on the devices on the system.

11.1 Switchable System Memory Option (/S)

Beginning with AMOS version 4.3, you may use the SYSMEM command to set aside part of switchable memory for use by the monitor, thus reducing monitor size. (See Section 7.0, "Switchable System Memory (SYSMEM)," for more information.)

For the present, bitmaps are the only monitor elements that can be placed in switchable system memory. Designate a specific bitmap as one which is to be placed in switchable system memory by ending the BITMAP command line with a /S. For example:

```
BITMAP DSK,1818,0,1,2,3,4,5/S
```

Make sure you have enough room in the switchable system memory you have allocated.

12.0 THE MONITOR QUEUE (QUEUE)

The monitor has a general purpose queue system that several commands use, and which is also available to user programs. The queue contains a fixed number of eight-word blocks which are assigned and then returned during the course of processing. The number of queue blocks that you need depends upon the size of your monitor, and the tasks that it performs.

The monitor initially contains 20 blocks; you may add more by using the QUEUE command in the SYSTEM.INI. Place the QUEUE command before any SYSTEM commands. The QUEUE command allocates additional queue blocks. So, the example in the sample SYSTEM.INI file, QUEUE 20, adds 20 blocks to the basic queue size of 20 to give a total queue size of 40 blocks.

For information on how your assembly language programs can access the monitor queue, refer to AMOS Monitor Calls Manual, (DWM-00100-42).

13.0 THE CLOCK FREQUENCY (CLKFRQ)

The AM-100 CPU board contains a real-time clock that several programs (e.g., the AlphaBASIC compiler and DYSTAT) refer to when they calculate time intervals; the system also uses this clock to perform job scheduling and timekeeping functions. The CPU board contains an external input for the line clock frequency that is connected to an AC line of approximately 10 volts.

The system has to know what frequency is being applied to the clock input line on the CPU board so that the programs that refer to the clock can know how to convert the clock tick count into actual time in seconds. This frequency is usually 60 Hz (the standard line frequency in the United States). Since many systems are shipped overseas, however, where the standard line frequency is 50 Hz, you must specify which frequency you are using.

The CLKFRQ command specifies the frequency (in Hz) that is being applied to the external clock input. The system stores this value so that programs that need to convert clock ticks to real time will be able to find out what frequency the clock is operating at. The CLKFRQ has nothing to do with the actual frequency at which the computer runs, and changing the CLKFRQ value does not affect the speed of the system. The CLKFRQ command in the sample SYSTEM.INI in Section 2.0 was:

```
CLKFRQ 60
```

If you do not include the CLKFRQ command in the SYSTEM.INI, the system stores a zero in the monitor location reserved for the clock frequency. In this case, programs trying to convert clock ticks into actual time aren't able to do so.

You may place the CLKFRQ command anywhere in the SYSTEM.INI. (NOTE: If you are going to reset the system date and time from the AM-120 board clock/calendar, you must place the CLKFRQ command before the DATE and TIME commands in your SYSTEM.INI. See the next section.)

14.0 RESETTING THE SYSTEM DATE AND TIME FROM THE AM-120 (DATE AND TIME)

The AM-120 Auxiliary I/O Controller contains as one of its features a clock/calendar with battery backup. If your system contains an AM-120, you may include the DATE and TIME commands in your SYSTEM.INI to reset the system date and time from the AM-120 clock/calendar. If you do so, you must place the DATE and TIME commands before the final SYSTEM command in the SYSTEM.INI. You must also place the CLKFRQ command before the DATE and TIME commands. For example:

```
CLKFRQ 60
DATE
TIME
SYSTEM HWK.DVR
SYSTEM
```

For information on DATE and TIME, see the DATE and TIME reference sheets in the AMOS System Commands Reference Manual, (DWM-00100-49). For information on the AM-120, see Software Installation Instructions for the AM-120 in the "System Operator's Information" section of the AMOS Software Update Documentation Packet.

15.0 INCORPORATING PROGRAMS IN SYSTEM MEMORY (SYSTEM)

You may incorporate programs into the system monitor by using the SYSTEM command in the SYSTEM.INI file. These programs may be system programs or your own programs. When the system reads the SYSTEM.INI, it loads into system memory the programs you've specified in the SYSTEM commands. These programs actually become part of the monitor, and so dynamically increase its size as they are loaded into memory. (NOTE: Before Release 4.5, you were required to place the generalized terminal driver, TRM.DVR[1,6], in system memory. This is no longer necessary.)

CAUTION-- the programs to be included in the monitor must be re-entrant (that is, sharable by more than one user). If they are not re-entrant, there is a possibility of system failure when two users attempt to access the same program. Many of the AMOS programs are re-entrant. Check with the AMOS System Commands Reference Manual, (DWM-00100-49), to see if a particular command program is re-entrant.

The most common use of the SYSTEM command in the SYSTEM.INI is to include the AlphaBasic runtime package (RUN.PRG) in the monitor so that each user does not need to load RUN into his own memory partition. You may also include the interactive compiler (BASIC.PRG) itself if you expect heavy development work by more than one user. If users on your system will be making extensive use of the screen-oriented text editor, VUE, you may want to load it into sharable memory via SYSTEM.

You might also want to use the SYSTEM command to include any realtime routines in the monitor that must be in memory at all times so that they can process asynchronous events (such as data collection from interrupting devices). For example, the DYSTAT program runs a continuously changing system display on a video monitor, and must be in memory at all times once it has begun execution so that it can update the display.

Another reason to use the SYSTEM command is if you want to include frequently-called user subroutines in the monitor. (You can locate such subroutines by name via the SRCH and FETCH monitor calls from assembly language programs.) Again, if these programs are to be shared by several users, they MUST be re-entrant.

To include programs in the system monitor, use one SYSTEM command for each program. Follow each SYSTEM command with the file specification of the program you want to include. Example:

```
SYSTEM VUE.PRG[1,6]
SYSTEM RUN.PRG
```

Note that you may include programs outside of the [1,4] account (the account that the system is initially brought up under). If you do not specify an extension, the system assumes a default extension of .PRG. You must place any SYSTEM commands after all other commands in the SYSTEM.INI that expand the monitor size.

The SYSTEM command has another use beside the inclusion of programs in the system monitor. A SYSTEM command alone on a line in a SYSTEM.INI (that is, not followed by a file specification), tells the system that monitor expansion is finished. The system then flags the monitor as up and running. The system also sets a flag in the system communication area that indicates that the system initialization is done except for final cleanup. Various commands (including SYSTEM) test this flag to see which mode they may operate in. For example, before the system is up and running, the JOBS command allocates new jobs on the system; after system initialization, the JOBS command displays the jobname of the user that typed the JOBS command.

Whether or not you include any programs in the monitor, your SYSTEM.INI must have a SYSTEM command without a file specification to tell the operating system that the system has been initialized. This SYSTEM command must be after any other commands that expand the monitor size (and that includes any other SYSTEM commands that are followed by a file specification).

After the system is up and running, the SYSTEM command performs a new function as a user command. After system initialization, the SYSTEM command tells you what programs are in system memory (that is, what programs are a part of the monitor), and the total size (in decimal words) of the monitor. For more information on the programs in system memory, you can use the MAP command. (For information on MAP, refer to Chapter 11, "Memory Commands," in the AMOS User's Guide, (DWM-00100-35).)

16.0 SYSTEM INITIALIZATON CLEANUP

After the monitor processes the SYSTEM commands in the SYSTEM.INI, the system is technically up and running. There are a couple of things still left to do, however, before the initialization procedure is complete. You may now include any commands in the SYSTEM.INI that you want the monitor to perform automatically at the time of system start-up. These commands are all commands that you can enter from the keyboard for yourself, but it's sometimes convenient to have the monitor perform them automatically every time you power up or reset the system. For example, you can have the monitor mount the disks that you are going to use, connect terminals to jobs, etc. You can also use the FORCE command to force input to a particular job. (You can use this feature to login a user, run a business program, etc., all without direct operator intervention.) We discuss some of these commands below.

After you've included the functions you want performed automatically, there is one last thing to do before system initialization is complete-- put a MEMORY 0 command into the SYSTEM.INI to deallocate the temporary user partition in the top 8K of memory that we have been using to process the SYSTEM.INI. The MEMORY 0 command must be the last command in the file. (Include the MEMORY 0 command even if your system bank switches memory.)

Now we'll discuss some of the commands you may want the system to perform automatically for you at system start-up. Remember that unlike some of the commands discussed in earlier sections, all of these commands are legal user commands, and may be used outside of the SYSTEM.INI.

16.1 Setting Options (SET)

The SET command can perform a variety of system functions. For example, the SET BPI command sets the bits-per-inch data density value used by the magnetic tape transport driver. The examples in our sample SYSTEM.INI:

```
SET DSKERR
SET GUARD
```

tell the system to report any soft disk errors that occur (SET DSKERR) and to guard the terminal of the job processing the SYSTEM.INI from any messages sent by other terminals (SET GUARD).

Note that the SET command only affects the job that used it. For example, the SET DSKERR command above only affects the job the system comes up under. (For information on forcing commands to other jobs, see Section 16.5, "Forcing Input to a Job (FORCE).")

See the SET reference sheet in the AMOS System Commands Reference Manual, (DWM-00100-49), for more information on SET.

16.2 Attaching Jobs (ATTACH)

When the system first begins to process the SYSTEM.INI, it automatically attaches the first job listed in the JOBS command and the terminal defined by the first TRMDEF command. Except for this special case, however, the system does not automatically link jobs with terminals. (When a job is linked to a terminal, the job and terminal are "attached." When a job is not linked to a terminal, the job is "detached.") A detached job must have a terminal attached to it before it can do terminal input or output. A detached terminal, on the other hand, can be accessed through terminal service calls or the general TRM driver. You cannot attach a job to a detached terminal from that terminal itself; you must do it from another terminal.

To attach jobs and terminals, you must use the ATTACH command. Once a job is attached to a terminal, it uses that terminal for input and output.

You can use the ATTACH command in several different ways:

```
ATTACH Terminal,Job
```

This command attaches the terminal and job named. If the terminal or job are already attached to other units, the ATTACH command detaches them before it attaches them to each other.

```
ATTACH Job
```

This ATTACH command attaches the user's own terminal to the job named (and detaches it from the current job).

```
ATTACH
```

This use of the ATTACH command lists the terminals that are currently attached, and the jobs to which they are attached. For more information on ATTACH, see the ATTACH reference sheet in the AMOS System Commands Reference Manual, (DWM-00100-49).

16.3 Allocating Memory (JOBMEM)

Systems that bank switch memory require that you use the JOBMEM command to allocate memory to a user memory partition. Do not use JOBMEM to allocate memory to the job the system is coming up under. (For complete information on JOBMEM, refer to the JOBMEM reference sheet in the AMOS System Commands Reference Manual, (DWM-00100-49), and the document Memory Management Option in the "System Operator's Information" section of the AMOS Software Update Documentation Packet.)

The JOBMEM command in the SYSTEM.INI takes the form:

```
JOBMEM Jobname Bank-#:StartAddress-EndAddress
```

where Jobname specifies the job you are allocating memory to. Bank-#: selects the memory bank the user partition will reside in, and StartAddress and EndAddress select the beginning and ending addresses of the memory block in that bank you want to allocate.

After the system is up and running, JOBMEM becomes a user command. You can use JOBMEM to tell you what areas of memory are allocated to your own job and to other jobs on the system.

There are some important restrictions on the use of JOBMEM; refer to the JOBMEM documentation mentioned above for complete instructions on its use. Also see the Alpha Micro Integrated Systems User's Guide, (DWM-00101-00), for information on bank switching.

16.3.1 Allocating Memory in Bank Zero (MEMORY)

NOTE: If your system does not bank switch memory, do not use the JOBMEM command. If you are going to allocate memory within the SYSTEM.INI, use the FORCE and MEMORY commands. For example:

```
FORCE JOB2 MEMORY 32K.
```

Note that even if your system bank switches memory, the last command in the SYSTEM.INI is always MEMORY 0; this restores all sharable memory not used by the monitor to the job the system is coming up under. For example, if your MEMDEF commands have set up a sharable area of 32K and switchable banks of 32K, if the monitor actually only takes up 29K, you can add the remaining 3K (32K-29K) in the sharable portion to the job that has the first block of memory in Bank Zero.

In a bank-switched system, use the MEMORY command for all jobs whose memory is located in Bank Zero or sharable memory. When we talk about setting up

the line printer spooler, you will see that we use a MEMORY 4K command to allocate memory to the job running the spooler. That's because our sample SYSTEM.INI is setting up the spooler to run in sharable memory.

16.4 Killing Jobs (KILL)

To properly initialize the jobs on the system, you should KILL the jobs that you have defined. A KILL command sends a Control-C to the specified job; this puts the job at the monitor level, ready to receive and send data. The KILL commands must appear after any ATTACH commands, but before any FORCE commands are used to send commands or data to the job. Do not kill the job that the system is coming up under (i.e., the first job in the JOBS command line).

Use one KILL command for each job on the system except for the job the system is coming up under:

```
KILL JOB2
KILL JOB4
```

16.5 Forcing Input to a Job (FORCE)

The FORCE command gives you a way of sending input to another job. To send one line of input to another job, use the FORCE command followed by the jobname and the input. For example, our sample SYSTEM.INI contains this line:

```
FORCE JOB2 LOG DSK2:2,2
```

The line above logs JOB2 into the system under account DSK2:[2,2]. You can also send several lines of input to a job by typing a carriage return after the jobname. Example:

```
FORCE JOB2
```

After that point, all lines of text that follow (up to a blank line) will be sent to the specified job. (A blank line is a carriage return alone on a line.) Example:

```
FORCE JOB2
LOG DSK2:2,2
ORDER.PRG
```

16.6 Mounting Disks (MOUNT)

The system automatically mounts the System Disk (DSK0) for you at the time of system start-up. If you wish it to mount other disks as well, include one MOUNT command for each disk drive you want to mount. Example:

```
MOUNT DSK1:
MOUNT STD0:
```

You MUST mount a disk before you can read or write data to it. REMEMBER: Never mount a disk while another user is accessing it.

16.7 Setting Head Load Time (HEDLOD)

The HEDLOD command sets the head load time for a Persci floppy disk that runs under control of the AM-200 or AM-210 floppy disk controller boards.

The number that follows the HEDLOD command selects the number of real-time clock ticks that the AM-200 board must keep the disk drive heads loaded after a data transfer. The example in the SYSTEM.INI file in Section 2.0 is:

```
HEDLOD 1800
```

This tells the floppy disk controller to keep the heads loaded for 1800 clock ticks (30 seconds when the real-time clock is operating at 60 Hz) after any data transfer. The HEDLOD command does not affect disk drives that do not allow software control of head load timing. You may omit this line if you are not running with a Persci floppy disk.

16.8 DYSTAT

DYSTAT is a system program that reports on the dynamic status of the operating system by way of a continuous display on a video monitor. DYSTAT requires that your system contain a memory-mapped video board. Omit the DYSTAT command if you do not want to run this display.

NOTE: DYSTAT runs asynchronously without regard to user job. If you are going to use DYSTAT, you must include it in the monitor (by using a SYSTEM command in the SYSTEM.INI) so that it is in system memory at all times. If you try to execute the DYSTAT command and it is not in system memory, the system fails the first time the job scheduler calls on DYSTAT for an update because it sees that DYSTAT is no longer in memory. (Remember that the system executes system commands in the partition of the user that requested the command; after it is finished, it usually deletes the program from the partition.)

The DYSTAT program also requires that the TODCNV routine is in system memory so that it can display the current time in the header line of the DYSTAT display. Load TODCNV into system memory by using a SYSTEM command in the SYSTEM.INI, just as you loaded DYSTAT.

For more information on DYSTAT, see the DYSTAT reference sheet in AMOS System Commands Reference Manual, (DWM-00100-49).

16.9 Setting Up the Line Printer Spooler

You can set up the line printer spooler from AMOS command level (and doing so is a good technique for debugging the spooler if you have problems bringing it up). However, usually you will want to bring up the spooler at the time of system start-up by placing the proper instructions in the SYSTEM.INI.

The sample SYSTEM.INI in Section 2.0 shows the following 6 lines:

```
FORCE SPOOL
MEMORY 4K
LOG 1,2
LPTINI PRINTR.INI

WAIT SPOOL
```

These six lines set up the system line printer spooler. (A spooler is a program that sets up a queue (or waiting line) for a particular program. When a line printer spooler is in control, requests for use of the printer are placed into a queue. As the printer becomes available, the spooler looks at the request at the top of the list, finds the file, and sends it to the printer. The spooler then removes that request from the queue. The printer prints files in the order of their requests in the queue.)

Setting up a line printer spooler is a good example of the kinds of things you can ask the SYSTEM.INI file to do at the time of system start up. For a detailed explanation of how to set up the line printer spooler, see the document Setting Up the Line Printer Spooler in the "System Operator's Information" section of the AMOS Software Update Documentation Packet.

SETTING UP THE LINE PRINTER SPOOLER

April 1981
Revision A02

This document reflects AMOS versions 4.5 and later

ALPHA MICROSYSTEMS
17881 Sky Park North
Irvine, CA 92714

Table of Contents

1.0 INTRODUCTION

The line printer spooler takes care of handling requests to use the printer by "spooling" a printer request into a queue. (As the printer becomes available, it prints the file specified by the next request in the queue.) This allows your job to print files at the same time as other jobs on the system without tying up your job until the printer is free. You simply enter your printer request (using the PRINT command) and then go on to other things.

The spooler program has a number of special features. For example, it can: print multiple file copies, print a banner page that identifies the listing, delete a file after it is printed, recognize wildcard file specifications, remember what type of form should be mounted on a printer, identify a printer by name, etc. In addition, the spooler (LPTSPL.PRG) is re-entrant, which means that you may load it into system memory.

(NOTE: However, loading LPTSPL.PRG into system memory is not usually a good idea. In setting up a line printer spooler, you do not run LPTSPL.PRG directly; instead, you run the LPTINI program which calls LPTSPL. Since LPTINI must be run in each memory area in which you are setting up a spooler, and since it loads LPTSPL.PRG into memory over itself, placing LPTSPL.PRG in system memory does not save you any memory space in the individual area in which you are setting up the spooler and just increases the size of your monitor area.)

You may either set up the spooler in system memory or (if your system bank switches memory) you may place it in a bank-switched memory partition (thus reducing the size of the monitor in system memory). You may also install more than one line printer spooler (which you may want to do if you have more than one printer). Installing multiple printer spoolers gives you multiple printer queues; one for each printer that is handled by a spooler.

For more information on the features and operation of the line printer spooler, see the documentation on the PRINT command in the AMOS System Commands Reference Manual, (DWM-00100-49) and the section titled "Printing Files (PRINT)," in the AMOS User's Guide, (DWM-00100-35).

The purpose of this document is to help you set up the line printer spooler on your own system. To that end we have included on your System Disk two text files: SYSLPT.INI and PRINTR.INI. SYSLPT.INI is a sample system initialization command file much like the regular SYSTEM.INI file with which you are already familiar, except that it includes those command lines necessary for bringing up the spooler program. (See Section 2.0, below, for a discussion of changing the SYSTEM.INI.) The PRINTR.INI is a sample spooler parameter file that contains information necessary for customizing the spooler program for your own use.

You may install multiple spoolers by setting up multiple spooler jobs. Each job can have a different spooler parameter file; this enables you to spool to several different printers on the system at the same time.

Before attempting to set up the line printer spooler, read this entire document carefully. Then modify the PRINTR.INI file that we have supplied to reflect your own needs or create your own spooler parameter file. At that point you can either use the SYSLPT.INI file to boot the system up with the new line printer spooler, or you can change your existing SYSTEM.INI file as outlined below. If you do not want the line printer spooler to be set up automatically when the system comes up under the control of the SYSTEM.INI or the SYSLPT.INI, you may set up the spooler yourself from the keyboard by following the instructions in Section 6.0, "Troubleshooting the Spooler."

2.0 MODIFYING THE SYSTEM.INI TO SET UP THE SPOOLER

The discussions below assume that you are familiar with the document titled The System Initialization Command File which appears in the "System Operator's Information" section of the AMOS Software Update documentation packet; that document contains information on the JOBS, TRMDEF, DEVTBL, ATTACH QUEUE, MEMORY, and FORCE commands, and discusses how to modify a SYSTEM.INI.

Take a look at the sample system initialization command file we provide (SYSLPT.INI):

```
:T
JOBS JOB1,SPOOL
TRMDEF TERM1,AM300=1,SOROC,100,100,80
TRMDEF TERM6,AM300=6:16,TELTYP,100,100,100
TRMDEF DUMMY,PSEUDO,NULL,30,30,2
DEVTBL DSK1,TRM,MEM,RES
BITMAP DSK,606,0,1
QUEUE 15
SYSTEM
CLKFRQ 60
;
ATTACH DUMMY,SPOOL
KILL SPOOL
FORCE SPOOL
MEMORY 4K
LOG 1,2
LPTINI PRINTR.INI

WAIT SPOOL
;
MOUNT DSK1:
MEMORY 0
```

This sytem initialization command file assumes that you do not want to bank switch memory. The following sections discuss this initialization command file, and we go through each step of installing a line printer spooler.

2.1 Setting Up the Spooler Job

The first step in setting up the line printer spooler is to define the job in which it is to run. The SYSLPT.INI file above creates job SPOOL in which to run the spooler program by including SPOOL in the job definition command (JOBS). You may use any unused job to control the spooler program.

2.2 Defining a Pseudo-terminal

So that we do not tie up a terminal when entering the commands to SPOOL that will set up the line printer spooler, we define a pseudo-terminal with which to run the job. A pseudo-terminal requires no actual hardware I/O device. The terminal definition (TRMDEF) command:

```
TRMDEF DUMMY,PSEUDO,NULL,30,30,2
```

defines the pseudo-terminal named DUMMY. Be careful that the buffer sizes you specify (in this case, 30,30,2) are large enough to contain the commands that you are going to be forcing to the job via the FORCE command. (See the document The System Initialization Command File for information on the PSEUDO interface driver, the NULL terminal driver, and the TRMDEF command.)

2.3 Allocating Additional Queue Blocks

The line printer spooler uses the system queue blocks to store requests for printing. When the system is initially brought up, 20 of these queue blocks are allocated by the system. Two of the blocks are used for every printer defined on the system, and three for every file request. Because queue blocks are required by other portions of the operating system, the spooler will not allow the number of free queue blocks to go below six. Therefore, with the standard allocation of 20 queue blocks, you may queue a total of four printer requests at any one time. If you want to be able to queue a larger number of requests, use the QUEUE command in the system initialization command file to allocate additional blocks. In the example above, we have used the QUEUE command to allocate an additional 15 blocks for a total of 35.

NOTE: If you are setting up more than one line printer spooler, make sure to allocate enough extra queue blocks. For example, if you are setting up two line printer spoolers, you will need twice as many queue blocks as if you were setting up one spooler.

2.4 Attaching a Terminal to the Spooler Job

No jobs and terminals are automatically assigned to one another except for the job under whose control the system comes up (the first job on the JOBS line) and the terminal defined by the first TRMDEF command. We must explicitly attach any other jobs and terminals by using the ATTACH command. So, we attach the spooler job and the pseudo-terminal:

```
ATTACH DUMMY,SPOOL
```

2.5 "Killing" the Spooler Job

To properly initialize a job, you must always use the KILL command on that job after attaching a terminal to it but before forcing any commands to it:

```
KILL SPOOL
```

2.6 Using the FORCE command to Set Up the Line Printer Spooler

Once the spooler job has been allocated, you have attached a terminal or a pseudo-terminal to the job, and you have used the KILL command on the job, you can start the spooler program itself. All commands issued to the spooler job during the spooler initialization go through the FORCE command.

Now that we have attached the pseudo-terminal DUMMY to the job SPOOL, we can send the commands to the job that will get the line printer spooler program up and running. FORCE the following sequence to the job:

```
FORCE SPOOL
MEMORY 4K
LOG 1,2
LPTINI PRINTR.INI
```

The following sections discuss each element of this FORCE sequence:

2.6.1 "FORCE SPOOL" - The first line of this sequence tells the system that we are starting a set of commands that we want to send to job SPOOL. (Because a carriage return appears at the end of the job name, AMOS knows that more than one line of input follows; a blank line signals the end of the group of commands and data that we are forcing to the job.)

2.6.2 "MEMORY 4K" - The first thing to do is to allocate the spooler job some memory so that it will be able to bring up the line printer spooler program. This spooler program requires at least 4K bytes of memory in which to work. On a bank-switched system, you must use the JOBMEM command (NOT the MEMORY command) before the FORCE command if you want to allocate the spooler memory in a bank-switched partition. For example:

```
;
ATTACH NULL,SPOOL
JOBMEM SPOOL 3:111530-123030
KILL SPOOL
FORCE SPOOL
LOG OPR:
LPTINI PRINTR

WAIT SPOOL
```

NOTE: We have determined that, on the average, 4K of memory is the usual amount of memory required to bring up the spooler program. This amount may vary depending on the particular device driver or terminal driver you are using to run the printer. (See Section 6.0, "Troubleshooting the Spooler Program," for instructions on determining the exact amount of memory required by your particular use of the line printer spooler program.) For information on the JOBMEM command, see The System Initialization Command File and Memory Management Option in the "System Operator's Information" section of the AMOS Software Update Documentation Packet.

2.6.3 "LOG OPR:" - The spooler job is no different from any other job on the system-- it has to be logged in to a disk account and must have memory if it is to run a program. Here we log the spooler job into the System Operator's account, OPR: (DSK0:1,2).

NOTE: If you do not log the spooler job into an account, it will not be able to run the spooler program. In that event, you will not see an error message; the system will simply not be able to bring up the spooler.

We log the spooler into account [1,2] because it must reside in the System Operator's account in order to override the normal AMOS account protection. This is necessary if a user wishes the PRINT command to delete files in an account outside of his or her own project.

IMPORTANT NOTE: Although we show the system initialization command file logging the spooler job into account [1,2], you may not wish to do so. (If your System Operator account is password protected, you would have to include the password in the SYSTEM.INI file, which would then make that password accessible to ingenious but unauthorized users.) To avoid placing the password to account [1,2] in the SYSTEM.INI file, you may log the spooler into any disk account (most commonly [1,4]); however, note that the LPTINI program will always transfer the spooler job to account [1,2] once the program is running.

2.6.4 "LPTINI PRINTR.INI" - Now we bring in the LPTINI program. LPTINI reads the specified parameter file (in this case, PRINTR.INI), and brings in the actual line printer spooler program, LPTSPL. The parameter file, PRINTR.INI tells LPTINI how to customize the spooler program for your own needs. (See below for information on PRINTR.INI). The file specification that you provide to LPTINI may that of any text file that you have created as long as it is a valid spooler parameter file. The file specification default extension is .INI, and the default account is [1,4].

2.6.5 "BLANK LINE" - The blank line (a carriage return alone on a line) signals the end of the forced input to SPOOL.

2.6.6 "WAIT SPOOL" - The next line:

```
WAIT SPOOL
```

tells the system to wait until SPOOL is finished executing LPTINI before it proceeds with bringing up the system; this allows the spooler program to finish any tricky allocations and initializations.

3.0 SETTING UP MULTIPLE SPOOLERS

To set up another spooler, you may basically follow the same instructions you followed when setting up the first spooler. Just make sure that you define a new job for the new spooler and that you assign it its own area of memory. (You may use the same pseudo terminal for the second spooler as you did for the first.) The following portion of a SYSTEM.INI shows setting up two spoolers on a bank switched system. Notice that each spooler job uses a different .INI file-- SPOOL uses PRINTR.INI, and SPOOL2 uses PRNTR2.INI. Each spooler parameter file specifies a different system printer.

```
;
ATTACH NULL,SPOOL
JOBMEM SPOOL 3:111530-123030
KILL SPOOL
FORCE SPOOL
LOG OPR:
LPTINI PRINTR

WAIT SPOOL
;
ATTACH NULL,SPOOL2
JOBMEM SPOOL2 3:123032-134340
KILL SPOOL2
FORCE SPOOL2
LOG OPR:
LPTINI PRNTR2

WAIT SPOOL2
;
CLKFRQ 60
MEMORY 0
```

4.0 THE SPOOLER PARAMETER FILE

The commands that appear in the spooler parameter file set the default information used by the spooler program when dealing with the specific job to which the parameter applies. You can set up several printers on the system, all maintaining a separate printer queue, by setting aside a job for each printer (as we did in the example above), and creating a different parameter file for each job. By merely changing the name of the printer to be used and the printer specification, you may initialize as many separate printer queues as you would like.

We have provided a sample parameter file for you, PRINTR.INI:

```
DEVICE=TRM:TERM6
NAME=LPT0
DEFAULT=TRUE
OPERATOR=JOB1
FORMFEED=TRUE
FORMS=NORMAL
BANNER=TRUE
HEADER=FALSE
LPP=56
WIDTH=132
```

You can create your own file (named with any valid filename) using the system text editors, VUE or EDIT. The format used by the elements of the file is:

```
command=argument
```

where argument is a value or attribute to be assigned to the command. Only one command may appear on each line of the file. Some commands take a boolean argument (a true or false value). LPTINI understands the following boolean arguments:

TRUE	FALSE
T	F
ON	OFF
YES	NO
Y	N
1	0

Here is a list of the parameter file commands:

4.1 DEVICE Command

The format of the command is:

```
DEVICE=devspec
```

where devspec is the specification of the device that is going to be used as the printer. If you are using a terminal as a printer, your command line might look something like this:

```
DEVICE=TRM:TERM6
```

which uses the generalized terminal driver, TRM.DVR. The terminal must have been defined in a TRMDEF command line in the system initialization command file, and must have a .TDV program in account DSK0:[1,6].

If you are using a Centronics device driver, the command line might look like this:

```
DEVICE=CEN:
```

which selects a device defined in the DEVTBL command of your system initialization command file. Whatever device specification you use for this second format, the device specified must be in your DEVTBL command line in the SYSTEM.INI, and must have a .DVR program in account DSK0:[1,6].

4.2 NAME Command

The format of this command is:

```
NAME=string
```

where string is a one- to six-character name that you want to assign to the device specified by the DEVICE command. Since more than one line printer spooler can be set up on a system, each handling a different printer, giving a name allows you to specify a particular printer to the PRINT command.

4.3 DEFAULT Command

This command takes the format:

```
DEFAULT=boolean
```

If the argument for this command evaluates to TRUE, the spooler program defines the printer defined by the DEVICE command as the default printer to be used when no printer is specified to the PRINT command. This command is an optional one; use it only when more than one printer is being defined on the system. If you omit DEFAULT from all spooler parameter files on the system or if all DEFAULT commands are set to false, the default printer is that printer with the least number of blocks waiting to be printed.

To specify a non-default printer when you use the PRINT command, enter the name of the printer (as specified in the NAME command above) followed by an equal sign. Then enter the specification selecting the files you want to print. For example:

```
.PRINT DIABLO=*.LST (RET)
```

4.4 OPERATOR Command

The format of this command is:

```
OPERATOR=jobname
```

where jobname specifies the job to which the spooler will send error messages and requests for forms changing. If you omit the OPERATOR command, the spooler will use the first job on the JOBS line of the SYSTEM.INI as the operator job.

4.5 FORMFEED Command

This command takes the form:

```
FORMFEED=boolean
```

The command sets the form feed switch default to /FF if the argument evaluates to TRUE or sets it to /NOFF if the argument evaluates to FALSE. (See above for a list of legal boolean arguments.) The /FF switch tells the spooler to perform special form feed handling; this ensures that the printer

is always at top-of-form when the spooler begins to print a new listing. For some applications (such as check printing), it is not desirable to have a final form feed output at the end of each listing; the /NOFF switch disables this final form feed. If you omit the FORMFEED command, the spooler program sets the default to /FF.

4.6 FORMS Command

The format of this command is:

FORMS=formname

where formname is one to six characters that you choose to identify a type of form (e.g., CHECKS, 2PART, etc.). If you omit the FORMS command, the spooler uses the default formname of NORMAL.

The purpose of this command is to allow you specify the kind of forms that should be mounted on the particular printer defined by this parameter file. The PRINT command then checks printer requests against this to see if the forms should be changed; if the printer request specifies a different form than the one mounted on the printer, an error message occurs informing the user that he must change the forms.

4.7 BANNER Command

The format of this command is:

BANNER=boolean

This command sets /BANNER as the default switch if the command argument evaluates to TRUE or sets it to /NOBANNER if the argument evaluates to FALSE. (See the table above for legal boolean arguments.) If BANNER is set to true, a banner page will be printed at the front of each listing. The banner identifies the file printed, the printer on which the listing was made, the date the listing was printed, etc. If you omit the BANNER command, the spooler sets the default switch to /BANNER.

4.8 HEADER Command

This command takes the form:

HEADER=boolean

The HEADER command sets /HEADER as the default switch if the command argument evaluates to TRUE or sets it to /NOHEADER if the argument evaluates to FALSE. The /HEADER switch tells the spooler program to print page headers. Page headers are titles printed at the top of every page which

give the name of the file, the date on which it was printed, and the current page number. The /NOHEADER switch disables the printing of page headers. If you omit the HEADER command, the default switch is /NOHEADER.

4.9 LPP Command

This command takes the form:

LPP=number

where number specifies the default number of lines per page. The spooler program uses this value in determining where to print page headers. If you omit the LPP command, the spooler uses the value of 56. You may override the value set by the LPP command by using the /LPP switch when you use the monitor PRINT command.

4.10 WIDTH Command

This command takes the form:

WIDTH=number

where number specifies the default width (in characters) of the printed line. The spooler program uses this value in determining the width of page headers. If you omit the WIDTH command, the spooler uses the value of 132. You may override the value set by the WIDTH command by using the /WIDTH switch when you use the monitor PRINT command.

NOTE: The values you give to WIDTH must range between 80 and 132. If you specify a number less than 80, the spooler uses 80; if the number is greater than 132, the spooler uses 132.

5.0 SPECIFYING FORMS

You can specify the type of form to be used on the printer by including the FORMS command in the spooler parameter file at the time of spooler initialization. The PRINT command compares the forms specified in print requests against this default form to make sure that they match.

After the spooler program is up, you can change that forms default by using the SET command:

.SET FORMS printername formname RET

where printername specifies the specific printer on which the form must be mounted, and formname gives the form type. For example:

.SET FORMS TI810 2PART RET

Once a form has been set using the SET command, the PRINT command checks all print requests sent to that printer to make sure that the proper type of forms is being specified.

If the form you specify in the PRINT command is not mounted on the printer when the file is selected for printing, the spooler sends this message to the terminal attached to the operator job:

;LPTSPL - Please mount form formname on printername

This message repeats once a minute until the spooler is notified that the form has been mounted (via the SET command). Since both LPTINI and PRINT use the default formname NORMAL, you can omit the FORMS command in the parameter file if you only use one kind of paper in your printers.

6.0 TROUBLESHOOTING THE SPOOLER PROGRAM

In the event that you have been unable to get the spooler up and running properly, you may find this section helpful.

Once the system has been booted, run the SYSTAT program to see what the spooler job is doing; if all has gone well you should see that the spooler job is running LPTSPL in an EW state. If the job is not running LPTSPL, or if it is in a ^C state, something has gone wrong.

You can run the LPTINI program on your own terminal to see what is happening. Be warned that this procedure will lock up your terminal; but it should give you some idea of what went wrong. You will have to reboot the system to gain control of your terminal after you are finished.

To see why the spooler did not initialize properly, log into [1,2] and allocate yourself the same amount of memory as the spooler job. Now run the LPTINI program, specifying the same parameter file that you used earlier in attempting to bring up the spooler. If you see nothing on the terminal after about 20 seconds or so, everything is probably allright. If you have another terminal connected to the system, run SYSTAT to see if the spooler job is now in EW state. If it is, something is wrong with the way you set up the spooler in the SYSTEM.INI. Check over your SYSTEM.INI file for some error in setting up the job, defining the pseudo-terminal, allocating memory, or defining the output device. If you still see no reason why the initialization process should have failed, try attaching the spooler to a real terminal (not the pseudo-terminal). By doing this, you will be able to see any error messages generated during the initialization process. If you are using a serial output device, and a TRMDEF command exists for your printer device (that is, it is defined as a terminal), you may attach the spooler job directly to the printer.

Chances are, you won't get that far. Most errors are caught by the LPTINI program which displays an appropriate error message. These are the messages displayed by LPTINI:

MEMORY ALLOCATION FAILED

You did not allocate enough memory to the job. LPTINI wasn't even able to load itself and its impure areas. Allocate more memory.

?Invalid command X

You specified command or argument "X" in your spooler parameter file which LPTINI was not able to recognize. Check the spelling in the file for errors. Make sure that your parameter command arguments are in the proper form (e.g., numeric or boolean).

?Bad DEVICE specification

The device you specified as the printer in your DEVICE command is not a valid device specification. Make sure that the device you specified is in the device table (the DEVTBL line of the SYSTEM.INI), and the the device has a driver program in DSK0:[1,6].

?Nonexistent job name specified for OPERATOR

The job you specified in your OPERATOR command does not exist. Check your spelling, and examine a SYSTAT display to see a list of the valid jobs on the system.

?Insufficient memory to run spooler, expand memory by n bytes

You did not specify enough memory to run the line printer spooler program. Increase your memory allocation by the amount specified. The amount of memory required by the spooler program depends on the particular device to which you are printing, and so may exceed the recommended amount of 4K.

Index

April 1981
Revision A01

MEMORY MANAGEMENT OPTION

This document describes how to set up and use more than 64K of memory in your Alpha Micro computer system by bank switching memory. It will not tell you how to address your memory boards; in order to implement memory management you will have to refer to documentation supplied by the board manufacturer(s). For more information, refer to the Alpha Micro Integrated Systems User's Guide, (DWM-00100-00).

Become familiar with this documentation before trying to implement memory management on your system. Also, try to become familiar with the addressing of the various memory boards you will be using in your system. If you have problems implementing memory management, refer to Section 3.0 to see if the problem is explained there.

For a full explanation of memory management, see Section V, "Memory Management," of the Integrated Systems User's Guide, (DWM-00101-00).

1.0 SETTING UP A MEMORY BANK

Unless the system initialization command file (SYSTEM.INI) contains MEMDEF commands, the operating system (AMOS) will not recognize any memory beyond the first 64K (1K = 1024; 64K = 65536). There is one MEMDEF command for each bank of memory to be defined. The MEMDEF commands must be before the final SYSTEM command in the SYSTEM.INI.

Due to the way AMOS works, there must be a totally sharable portion of memory starting at location 000000. This sharable portion must be large enough to hold AMOS, which requires a minimum of 11K (this size increases as more jobs, terminals, devices, and system programs are defined). You may also place the line printer spooler in sharable memory, which requires between 4 and 5K. Typically, the sharable portion will be 16K with no system programs, 32K if RUN.PRG is in system memory, or 48K if BASIC.PRG and RUN.PRG are in system memory. If sharable memory takes up some amount xK, then the largest bank-switched portion of memory can take up (64 - x)K of memory. Bank-switched memory refers to that portion of memory past sharable memory. There may be multiple banks of various sizes, as long as the largest bank is no larger than (64 - x)K, and there is no sharable memory past xK.

Bank-switched memory must be physically addressed to the memory address that it will occupy when that bank of memory is active. (Refer to the manufacturer's specifications for the appropriate switch or jumper settings.) For instance, assuming that sharable memory is from 0K to 31K and bank-switched memory starts at 32K, no bank-switched portion of memory can be addressed to a physical address below 32K. In this situation, assuming 16K memory boards, there would be one sharable memory board addressed for

0K-15K, one sharable memory board addressed for 16K-31K, and then the bank-switched memory boards. Two banks of 32K would have two boards in one bank addressed for 32K-47K and for 48K-63K respectively, and in the second bank a board addressed for 32K-47K and a board for 48K-63K.

In addition to the physical memory address, all bank-switched memory boards have an I/O port address associated with them. This I/O port is what is used to turn the various memory boards on and off. There may be more than one I/O port possible per board, depending on the manufacturer of the memory board.

This scheme allows memory boards to be turned on or off by sending a value to the I/O port of the memory board. More than one bank may use the same I/O port, as long as the value for turning on a particular bank is different than the value for turning on a different bank.

1.1 Defining a Memory Bank (MEMDEF)

There is one MEMDEF command for each memory bank. Memory bank numbering starts at 0 and is incremented by one for each MEMDEF statement. Therefore, three MEMDEF instructions in the initialization file define Banks 0, 1, and 2.

The form of the MEMDEF command is as follows:

MEMDEF board0-adr,on-cnst,off-cnst/board1.../boardN-adr,on-cnst,off-cnst

1. All operands are in octal, unless the SET HEX command has been executed.

2. "boardX-adr" is the I/O port address that the board is set up on.

3. "on-cnst" is the on constant-- the value that is used to turn the piece of memory at this I/O port on.

4. "off-cnst" is the off constant-- the value that is used to turn the piece of memory at this I/O port off. This will usually turn off the entire board except for any memory that is sharable or set up for a different I/O port.

5. The slash (/) allows multiple memory addresses to share the same bank of memory. This is especially convenient when using a large memory board (32K - 64K) that resides all at one I/O port but has different "on" and "off" values for smaller pieces of memory (8K to 16K).

Assuming a 16K sharable portion residing at the front of a 64K memory board, the remaining 48K of memory is switchable, and so are two other 64K memory boards with only 48K active on them; the first board is set up at I/O port 100, the second board is set up at 101, the third board is set up at 102; the "on" constant for memory on all boards in Bank Zero is 1, for memory in

Bank 1 is 2, for memory in Bank 2 is 4; the "OFF" constant for all memory is 0; the MEMDEF commands would then be as follows:

```
MEMDEF 100,1,0/101,1,0/102,1,0        ; Bank Zero
MEMDEF 100,2,0/101,2,0/102,2,0        ; Bank 1
MEMDEF 100,4,0/101,4,0/102,4,0        ; Bank 2
```

Note that memory from 0K to 15K is not set up to be switchable. Also note that a piece of all memory boards is in all banks. The boards would be set up in the following manner:

Board 1: I/O port 100	16K-31K	On constant = 1	Off constant = 0
	32K-47K	On constant = 2	(all boards)
	48K-63K	On constant = 4	
Board 2: I/O port 101	16K-31K	On constant = 2	
	32K-47K	On constant = 4	
	48K-63K	On constant = 1	
Board 3: I/O port 102	16K-31K	On constant = 4	
	32K-47K	On constant = 1	
	48K-63K	On constant = 2	

	Board 1	Board 2	Board 3
0-15K	shared	unused	unused
16-31K	Bank 0	Bank 1	Bank 2
32-47K	Bank 1	Bank 2	Bank 0
48-63K	Bank 2	Bank 0	Bank 1

Note that this is but one possible memory configuration and is set up in a strange way for demonstration purposes. A more efficient use of 64K memory boards is possible if different portions of the board are set up at different I/O addresses or (if the entire board is at one I/O address) if different portions have different "on" constants.

This scheme allows the use of different manufacturer's boards in the same system, as long as they can be set up at various I/O addresses and the largest piece turned on by a particular constant is small enough so that it does not overflow into sharable memory.

After the system is booted, the MEMDEF command takes on a new function. The MEMDEF command now allows you to see how memory is configured. The MEMDEF command will display a map of all active banks as a string of characters, with each character representing 1K of memory. Assuming a 13K monitor and the above example memory definition, here is what MEMDEF would display after you have booted the system:

```
.MEMDEF (RET)

          0K      8K      16K     24K     32K     40K     48K     56K     64K
          |       |       |       |       |       |       |       |       |
BANK 0    MMMMMMMMMMMMMSSSBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BANK 1    MMMMMMMMMMMMMSSSBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BANK 2    MMMMMMMMMMMMMSSSBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
.
```

"M" stands for "monitor" memory (memory used by the monitor), "S" stands for "sharable" memory (memory that is not monitor memory and is not bank switched), and "B" stands for "bank-switched" memory. Since monitor and sharable memory can be accessed by all banks, this area of memory is treated as a part of all banks. It is always active. The area of memory used by the monitor and sharable programs is also called "system memory."

Programs can be loaded into system memory using the SYSTEM command in the SYSTEM.INI file. The spooler can reside in system memory or in bank-switched memory. Any remaining sharable memory after the system is booted up that is not used by the monitor or programs in system memory is added to the beginning of Bank Zero and is available to the job the system comes up under. (The last command in the SYSTEM.INI file, MEMORY 0, allocates all available memory in Bank Zero to the job the system comes up under.)

There is a good reason why there is no memory appearing at "64K." As you can see, the counting of memory starts at 0K, not 1K. If there were memory at "64K," it would really be "65K," which is 66560 bytes of memory, since 1K = 1024.

The MEMDEF display is also a handy way to verify that you have jumpered the boards properly. Memory which the system cannot find will be missing from the display. Bad memory boards can sometimes be located in this fashion, too. Of course, if the bad board is in sharable memory or Bank Zero, the system may not come up at all.

2.0 ALLOCATING MEMORY TO JOBS

With this memory management system, the MEMORY command becomes obsolete except for allocating memory in Bank Zero. Do not use the MEMORY command to allocate memory for any job other than the first job in Bank Zero, the job the system comes up under. Instead, use the JOBMEM command to allocate bank-switched memory.

(However, remember to use the MEMORY command if you are allocating the line printer spooler in system memory. See Setting Up the Line Printer Spooler in the "System Operator's Information" section of the AMOS Software Update documentation packet for information on allocating the spooler in either system or bank-switched memory.)

Use the JOBMEM command as follows:

JOBMEM jobnam Bank-number:StartAddress-EndAddress

1. "jobnam" is the name of the job for which memory is to be allocated. The user's current job is used if "jobnam" is not specified.

2. "Bank-number" is the number of the memory bank that the job is to reside in.

3. "StartAddress" is the starting memory location for the job.

4. "EndAddress" is the ending memory location for the job.

If you specify no arguments, JOBMEM tells you how much memory is allocated to your job. If you specify a job name with no memory argument, JOBMEM tells you how much memory is allocated to the specified job.

Here are some sample JOBMEM statements:

```
.JOBMEM JOB2 1:40000-177376 (RET)
.JOBMEM JOB3 2:40000-77776 (RET)
.JOBMEM JOB4 2:100000-137776 (RET)
.JOBMEM JOB5 2:140000-177376 (RET)
.JOBMEM JOB5 (RET)
CURRENT MEMORY ALLOCATION IS 2:140000-177376
.JOBMEM (RET)
CURRENT MEMORY ALLOCATION IS 0:32370-177376
```

The first example gives the job called JOB2 all of the memory in Bank 1. The second through the fourth examples allocate the 48K of memory in Bank 2 in 16K partitions to three different jobs. The fourth job (JOB5) really gets 16K minus 256 bytes of memory (16128 bytes total) due to the fact that the top 256 bytes of memory are not accessible. (The top 256 bytes of memory are reserved for the I/O ports.) The fifth example lists the memory allocation for the job called JOB5. The final example lists the memory allocation for the current job.

2.1 JOBMEM Error Messages

?Memory allocation format error

JOBMEM is confused by the format of the information you've given it; it may be that you've made a spelling error.

?Non-existent bank number

If you've specified a bank number greater than the number of MEMDEF commands in the SYSTEM.INI, you'll see this message.

?Non-existent job

You've specified a job that does not exist. Check the JOBS command line in your SYSTEM.INI or run SYSTAT to see a list of the jobs on the system.

?Allocation overlaps monitor or system memory

Monitor and system memory is sharable between users; that means that you must not allocate any part of it to an individual user. Reduce your user allocations, or remove some of the programs that you've previously added to system memory by deleting the appropriate SYSTEM commands in your SYSTEM.INI.

?Illegal memory range (end is below base)

You probably entered your ending memory address before you entered the starting address. The starting address of the memory block you want to allocate must be less than the ending address.

?Allocation is not within requested bank's defined memory

You've asked for memory in a valid bank, but the memory addresses that the bank is set for do not include the addresses that you've tried to allocate. Check your MEMDEF commands, and the addressing of your memory boards.

?Requested allocation would overlap job Jobname

You've tried to allocate memory to one job that belongs to another. The job you've overlapped is named Jobname. (e.g., ?Requested allocation would overlap job JOBA.)

3.0 HINTS, RESTRICTIONS, AND WARNINGS

The next few paragraphs discuss some of the things that you should avoid doing when setting up your system for bank-switched memory. We also discuss some of the miscellaneous pieces of information that you should know to set up your system in this way.

When power is initially supplied to the system, or whenever the reset button is pressed, Bank Zero must turn itself on and all other banks must turn themselves off. Any other configuration may cause system failure either immediately after AMOS starts executing the SYSTEM.INI file, or when the MEMDEF statements in the SYSTEM.INI file are executed.

All memory in a particular bank must be contiguous. That is, you cannot have a piece of switchable memory starting at one location and ending at another, a "hole" where there is no memory, and then another piece of switchable memory. This is also true of sharable memory, and all memory in a system that does not use bank switching. You can, however, have sharable memory up to some point, a "hole," and then bank-switched memory. This is not the case with Bank Zero, which must always be contiguous with sharable memory.

If you are allocating a line printer spooler and there is not enough sharable memory for the spooler, your spooler will not work. Make sure that the spooler's "end adr" is lower than the top of sharable memory if you are placing the spooler in system memory.

If too many modules are loaded into the monitor with the SYSTEM command, monitor memory will extend into the bank-switched area, and the system will not work properly. You can use the MEMDEF command after you boot the system to check this out.

The MEMDEF commands must come before the final SYSTEM command in the SYSTEM.INI. It is recommended that the MEMDEF commands be placed right after the TRMDEF commands.

MEMDEF does not check to see if an I/O address with the same on/off constants was previously allocated. It is up to the System Operator to make sure that SYSTEM.INI is set up properly and that a single piece of memory is not allocated to two different banks.

When the MEMDEF display shows the last column of bank-switched memory at "62K" (instead of at "63K" where it belongs), you may have a problem with your disk controller board(s). The 8131 driver IC for the bootstrap PROM must be removed from any board controlling a disk device that is not the system device. PHANTOM must be removed from these boards also. Only the board which controls the system device can have the bootstrap PROM and PHANTOM enabled.

All memory addresses must be on word boundaries. In octal, memory addresses on word boundaries end with 0, 2, 4, or 6. In hex, they end with 0, 2, 4, 6, 8, A, C, or E.

With this memory management system, there is no way for a user to have a portion of memory in one bank, and another portion in another bank, unless he is running a specially written assembly language program and uses the BNKSWP monitor call. The user must be in sharable memory when he issues this call. (There is more information on the BNKSWP call in the AMOS Monitor Calls Manual, (DWM-00100-42).)

A bank of memory may be composed of part or all of several different boards, even if the boards are made by different manufacturers.

April 1981
Revision A02

DEFINING SWITCHABLE SYSTEM MEMORY

The Alpha Micro Operating System (AMOS) allows you to change the configuration of your system by changing the system initialization command file (SYSTEM.INI) that controls the system start-up process. When you add a new disk device or terminal, you add information about that device to the SYSTEM.INI.

As more device controllers and drivers become available, you will probably have a greater variety of devices on your system. Each time you add a new kind of device, however, your monitor becomes larger because the SYSTEM.INI instructs the monitor to incorporate within itself the software necessary to handle the new device. The larger your monitor becomes, the less memory is available to user memory partitions. The area of memory used by the monitor is called system memory.

If your system does not use memory management (that is, if you do not bank switch memory), there is nothing you can do about reducing the size of your monitor except remove programs that have been loaded into system memory via the SYSTEM command in the SYSTEM.INI or change the configuration of your system.

Beginning with AMOS Version 4.3, if your system uses memory management, you may now reduce the size of your monitor by placing bitmaps into bank-switched memory. To do this, you must first set aside an area of switchable memory as system memory. (For information on bank-switched systems, refer to Memory Management Option, in the "System Operator's Information" section of the AMOS Software Update documentation packet and see the Alpha Micro Integrated Systems User's Guide, (DWM-00101-00).) At this time, bitmaps are the ONLY portions of the monitor that you can place in switchable system memory.

1.0 THE SYSMEM COMMAND

The first step in defining switchable system memory is to make sure that your SYSTEM.INI contains valid MEMDEF commands. MEMDEF commands set up switchable memory banks that allow the system to access more than 64K of memory. (Individual users are still restricted to a maximum memory partition of 64K.) The switchable portions of these banks are usually allocated to user jobs, but you will be setting aside part of this area for your switchable system memory.

Now that your system is bank switching memory, you can use the SYSMEM command in the SYSTEM.INI to tell the system which part of a memory bank outside of the sharable monitor area you want to assign to the system

bitmaps. (NOTE: When you use SYSMEM in the SYSTEM.INI, you may only use it to define switchable system memory.)

The SYSMEM command takes this form:

SYSMEM Bank#:StartAddress-EndAddress

where Bank# identifies the specific memory bank in which you want to allocate switchable system memory, StartAddress and EndAddress select the beginning and ending addresses of the block of memory you want to allocate to switchable system memory.

Switchable system memory may not overlap sharable memory. That is, you cannot allocate to switchable system memory any memory locations in the non-switched area of memory. (The non-switched area of memory is used by the monitor.)

You may not allocate to switchable system memory the last 256 bytes of the 64K address space. (These bytes are reserved for the I/O ports.) The highest memory address you can allocate, then, is 177376.

Once the system is up and running, you can use SYSMEM at AMOS command level to find out what area of memory is set aside as switchable system memory. For example:

```
.SYSMEM (RET)
System memory allocations are:
     3:100000-166774
```

2.0 DEFINING SWITCHABLE SYSTEM MEMORY

To define switchable system memory, edit the SYSTEM.INI and make these changes:

1. First, you must define the area of switchable memory you want to set aside for system memory. Enter the SYSMEM command for this purpose. SYSMEM commands must appear after any MEMDEF commands, but before any BITMAP commands.

   NOTE: You can use more than one SYSMEM command in your SYSTEM.INI if you want to allocate more than one area of switchable system memory.

   SYSMEM uses the same format as the JOBMEM command. For example:

   SYSMEM 2:100000-116150

The command above allocates 7272 bytes to Bank Two (the third memory bank on the system) for bitmaps. (When you use SYSMEM, remember that the BITMAP commands refer to the number of decimal words you need to reserve for bitmaps, NOT the number of bytes. One word is two bytes, so double the bitmap size to find the number of bytes you need to allocate with the SYSMEM command.)

2. Next, identify those bitmaps you want to place in switchable system memory by including a /S at the end of the appropriate BITMAP command lines in your SYSTEM.INI. For example:

```
BITMAP DSK,1818,0,1,2,3,4,5/S
BITMAP AMS,39,0,1
BITMAP HWK,606,0,1/S
```

The BITMAP commands above tell AMOS to reserve 39 words in the monitor area in Memory Bank Zero for the bitmaps for devices AMS0: and AMS1:. The other two BITMAP commands select the switchable option by including a /S at the end of the command line. AMOS therefore knows that the bitmaps for DSK0:, DSK1:, DSK2:, DSK3:, DSK4:, DSK5:, HWK0:, and HWK1: are to be placed in switchable system memory.

3. At the time of system start-up, AMOS automatically places the bitmaps you have previously designated (via the /S option on the BITMAP command line) into the switchable system area you have defined.

3.0 ERROR MESSAGES

If you make an error in defining switchable system memory, you see the following error messages:

?System memory not allocated - monitor memory will be used

You tried to place a bitmap in switchable system memory (via the /S option in the BITMAP command line), but AMOS couldn't find any switchable system memory. (You can also see this message if you did not allocate enough switchable system memory to hold the designated bitmaps.) AMOS therefore places the bitmap in the area of sharable memory used by the monitor. Check to see that SYSMEM commands are present in your SYSTEM.INI.

?Memory allocation format error

SYSMEM didn't understand the format of your SYSMEM command line. For example, did you leave out the colon after the bank number?

?Nonexistent bank number

You've given SYSMEM a bank number larger than the total number of MEMDEF commands in your SYSTEM.INI. (That is, you've referred to a bank number that does not exist.)

?Allocation overlaps monitor memory

You must not allocate to switchable system memory any of the sharable memory area. (Sharable memory is memory that contains the monitor and that all users can access.) Type MEMDEF followed by a RETURN to see the memory bank configuraton for your system. This display tells you which areas of memory you can allocate to switchable system memory.

?Illegal memory range (end is below base)

Ending address of the block of memory you allocate to switchable system memory must be greater than the starting address.

?Allocation is not within requested bank's defined memory

You've specified a valid bank number to the SYSMEM command, but that bank is not addressed for the memory locations you've requested. Check the addressing of your memory boards and check the MEMDEF statements in the SYSTEM.JNI.

?Allocation overlaps memory previously allocated to a job

You've already tried to allocate to switchable system memory an area that has already been allocated (via the JOBMEM command) to a user job. Check your bank number and memory addresses in the SYSMEM command. If they're all right, check the memory allocations for the jobs on your system.

May 1980
Revision A01

CONFIGURING FLOPPY DISK DRIVERS

1.0 INTRODUCTION

The AMOS system supports several different kinds of disk devices. Because each type of device has its own characteristics and requirements, a separate device driver program must exist for each kind of disk device you use on your system.

The disk driver program links AMOS's generalized disk service routines with the physical disk device. One disk driver exists for each kind of hard disk that you can run under control of the AM-500, AM-400, and AM-410 hard disk controllers.

In the past, several different floppy disk drivers existed: one for each combination of drive type and disk format. A much larger number of floppy disk formats are now available. For this reason, rather than providing a separate disk driver for each possible combination of device type and disk format, we have given you the ability to configure your own floppy disk drivers.

1.1 New Alpha Micro Disk Formats

Before Release 4.2, only three floppy disk formats were available: IBM-compatible (STD), Alpha Micro format (AMS), and Image format (IMG). With the introduction of a new floppy disk controller (the AM-210) that can handle disks in double-sided, double-density formats, a new range of floppy disk formats is now possible. The FIXDVR program allows you to configure your own floppy disk drivers based on the particular disk type, floppy disk controller, and disk format you want to use. You must use FIXDVR to configure a driver for each different combination of device, controller, and format that occurs on your system.

2.0 USING FIXDVR

To run the FIXDVR program, log into the Device Driver Library account, DSK0:[1,6]. Then type FIXDVR followed by a RETURN:

```
.LOG DSK0:[1,6] RET
.FIXDVR RET
```

(Changed 1 May 1980)

FIXDVR now begins to ask you a series of questions, so that it can determine how to configure a floppy disk driver that matches your particular combination of disk type, disk controller, and disk format:

1. Controller Type (A) AM-200, (B) AM-210 or (C) Icom:

   Enter the letter A, B, or C to select the type of floppy disk controller you are using for your floppy disk drive. If you select the Icom controller, FIXDVR skips down to question #4 (see below).

2. Drive type (A) Persci, (B) Wangco, or (C) CDC:

   Enter the letter that selects the type of disk drive you are using.

3. Double-density?

   FIXDVR asks this question only if you have already specified the AM-210 as your disk controller. Enter a Y for Yes, or an N for No, depending on whether you plan to use the driver on double-density disks.

4. Format (A) STD, (B) AMS, or (C) IMG:

   Enter the letter that selects the disk format you want the driver to use. You may not specify the AMS format if the driver is to use single-density format on a drive running under control of the AM-210 disk controller. If you have previously selected the Icom controller, FIXDVR now skips down to question #6. (If you are using the Icom controller, you may not specify the AMS format.)

5. Double-sided?

   FIXDVR asks this question only if you have already specified the AM-210 as your disk controller. Enter a Y for Yes, or an N for No, depending on whether you plan to use the driver on double-sided disks.

6. Enter new driver name:

   Enter the name that you want to give to the driver program. The standard names that you might want to use are listed below in Section 4.0.

FIXDVR now displays this message:

New driver is now in memory, bitmap size is nn

The driver that you created is now in memory. Use the SAVE command to save it on the disk. For example:

.SAVE DDS.DVR RET

(Changed 1 May 1980)

The command above saves the driver onto the disk (in the account you are logged into, DSK0:[1,6]) as the file DDS.DVR. If you do not specify an extension, the SAVE command saves the file under the extension .DVR (which indicates a device driver program).

2.1 FIXDVR Error Messages

You may see the following error messages when using FIXDVR:

1. ?Could not find xxxxxx.DVR

   FIXDVR could not find the necessary file. If you are configuring a driver for the AM-200, FIXDVR requires that 200DVR.DVR be in DSK0:[1,6]; a driver for the AM-210 requires 210DVR.DVR in DSK0:[1,6]. A driver for the Icom controller requires that ICMDVR.DVR be in DSK0:[1,6].

2. Please enter Y or N

   Several of the questions that FIXDVR asks require that you answer with a Y or N for Yes or No.

3. ?Invalid reponse

   Several of the questions that FIXDVR asks require that you enter a letter to select an option (e.g., an A to select STD format).

4. ?Invalid device

   You have a bad version of 200DVR.DVR, 210DVR.DVR, or ICMDVR.DVR in DSK0:[1,6].

5. ?Icom does not support AMS format

   You tried to format an Icom floppy diskette in AMS format. (Icom floppy drives only support STD and IMG format.)

6. ?AM-210 does not support single-density AMS format

   You may not use single-density AMS format on a device that runs under the control of the AM-210 floppy disk controller.

7. ?AM-200 does not support CDC floppy disks

   You may only run CDC floppy disks under the control of the AM-210 floppy disk controller.

3.0 MODIFYING THE SYSTEM INITIALIZATION COMMAND FILE

You must modify the system initialization command file (SYSTEM.INI) to:

1. Add the device for which you have just defined a device driver to the DEVTBL command line. NOTE: If you have too many devices on the DEVTBL command line to fit all on one line, you may use several DEVTBL command lines. For example:

```
DEVTBL DSK1,AMS0,AMS1,AMS2,AMS3,DDS0,DDS1
DEVTBL TRM,MEM,RES,/MTM
```

2. Add BITMAP commands to the SYSTEM.INI to define bitmaps for the new devices you are adding to the system. Use the bitmap size given in the final FIXDVR message. For example, if you are defining a driver for a two-drive device, and the format requires a bitmap of 39, you might have seen this message:

New driver is now in memory, bitmap size is 39

Now you must add the appropriate BITMAP command to the SYSTEM.INI. For example:

```
BITMAP AMS,39,0,1
```

4.0 STANDARD DISK DRIVER NAMES

Below is a list of some of the standard names you can assign to the device driver defined by FIXDVR:

Driver Name	Characteristics
STD	Single-density, Single-sided, STD format
SDS	Single-density, Double-sided, STD format
DSS	Double-density, Single-sided, STD format
DDS	Double-density, Double-sided, STD format
AMS	Single-density, Single-sided, AMS format
SDA	Single-density, Double-sided, AMS format
DSA	Double-density, Single-sided, AMS format
DDA	Double-density, Double-sided, AMS format
SSI	Single-density, Single-sided, IMG format
SDI	Single-density, Double-sided, IMG format
DSI	Double-density, Single-sided, IMG format
DDI	Double-density, Double-sided, IMG format

5.0 FORMATTING DISKS

When you format a disk, you will use either the FMT200 or the FMT210 formatting programs (depending on whether you are running the device containing that disk under control of the AM-200 or the AM-210 Floppy Disk Controller).

NOTE: If you are using an AM-200 controller, make sure that the AM-200 format-enable switch is turned ON before you format a disk. The Icom controller is not capable of formatting diskettes; if you are using an Icom controller, you must buy preformatted disks.

The device specification you give to the formatting program will identify the driver (and thus the format) used for that device. For example, if you want to format a disk in Drive One of a device that uses the DSS driver (double-density, single-sided, STD format), enter:

```
.FMT210 DSS1: RET
```

FMT210 then formats the disk in the proper format, and you see:

```
BEGIN FORMATTING
EXIT
.
```

May 1980

AMOS VERSION 4.4 METHOD OF HANDLING BAD DISK BLOCKS

1.0 INTRODUCTION

AMOS Version 4.4 supports a new method of handling bad areas on devices such as the Phoenix disk drive. This class of high-performance disk drives, because of the high density of data on such devices, may contain flaws which prevent contiguous allocation of the disk area. As disk drives incorporating the newer technologies become available, the need for error tolerance becomes increasingly important. The Phoenix disk drive is the first disk supported on the Alpha Micro system where the media is not guaranteed to be 100% good.

AMOS Version 4.3 supported an interim solution to this problem by simply marking all bad blocks as "in-use," thereby preventing their use within a file. Although this method solved most of the problems posed by the media flaws, it introduced others, such as the possible fragmentation of storage on a surface, preventing the use of that surface as a single random file. Because of the actual distribution of bad blocks on the Phoenix drive, this did not turn out to be a major problem. However, recognizing that this was not the optimal solution, we reserved fifteen spare or "alternate" tracks on each Phoenix drive for use as alternate storage. To better support future disk drives, and to eliminate the possibility of surface storage fragmentation, we have implemented a new method of handling media flaws which uses these spare tracks.

2.0 THE NEW METHOD OF HANDLING BAD BLOCKS

The new technique of handling bad blocks is to flag bad tracks and translate these tracks to the spare tracks at the time of disk access. This method results in several advantages:

1. The entire disk surface is available for use. That is, no "bad" blocks can exist in the middle of a surface, preventing complete use of the surface as a random file.

2. DSKCPY-type utilities can be used on the surface, speeding up backup time.

3. Disk maintenance utilities, such as DSKANA and DSPAK, do not need to treat the disk as a special case; it can be accessed the same way as any other device.

4. The actual choice of bad tracks versus bad blocks is device dependent, making the new method easily expandable for future devices.

The technique used is as follows:

1. The certification program for the disk creates the file BADBLK.SYS which contains a list of bad tracks or blocks, depending on the particular device. Currently, the Phoenix uses bad tracks.

2. The DEVTBL command reserves space for an Alternate Track Table in system memory. DEVTBL does this automatically, with no change in the command format. The size of the Alternate Track Table is device dependent; currently, the Phoenix causes 30 bytes to be allocated for each surface.

3. The MOUNT command reads the BADBLK.SYS file into the Alternate Track Table whenever a disk is mounted.

4. Whenever a disk access is requested, the disk driver scans the Alternate Track Table to see if the requested block is in a track which is marked as bad. If so, a translation is performed to access the requested block within the alternate track assigned to that bad track.

   The actual allocation of alternate tracks is device dependent; currently, the Phoenix allocates tracks 808-822 as alternate tracks. The first bad track on a Phoenix is assigned to track 808; the second is assigned to track 809, and so on.

5. When the system is being booted, the bootstrap routine must read in BADBLK.SYS to handle the case where SYSTEM.MON is allocated on an alternate track.

At the current time, the Phoenix is the only device that uses the BADBLK.SYS technique of checking for bad areas.

3.0 CONVERTING TO THE NEW METHOD

Because of the advantages of the new method of handling media flaws, most Phoenix users will want to convert their disks to use the new format. The old format is still usable under AMOS Version 4.4; however, the old format may not be supported in future releases.

YOU MUST CONVERT ALL DISKS TO THE NEW FORMAT!

(Of course, if you are a first-time Phoenix user as of AMOS Release 4.4, your disks are already using the proper system.)

To convert a disk surface to the new format, follow this four-step procedure:

1. First, be certain that you are running under Version 4.4 of AMOS. (Use the SYSTAT command to see which version of AMOS you are

using.) The following procedures will have no effect if you are running under AMOS Version 4.3 or earlier.

2. Create a backup copy of the surface to be converted on a certified, but otherwise blank cartridge. (In the case of converting a cartridge disk, you will need to clear off a fixed surface so that you can place your backup there.)

3. Certify the surface to be converted, using the CRT410 program.

4. Copy the files from the backup disk to the newly certified disk via COPY.

Before AMOS Version 4.4, Phoenix users were restricted to using COPY when backing up a Phoenix surface. Now, after you have converted a disk, you may use DSKCPY from that time on to copy that converted disk to a certified, but blank, cartridge or surface when you want to do a backup, as long as the backup disk has also been certified via the 4.4 or later CRT410.

(Remember that DSKCPY makes a literal image; any data on the backup disk will be destroyed during the copy.)

April 1981

SOFTWARE INSTALLATION INSTRUCTIONS FOR THE AM-120

1.0 INTRODUCTION

This document describes the software that accompanies the AM-120 Auxiliary I/O Controller board and gives instructions for installing that software. Some of the programs described below are new and some are programs from previous software releases that we have modified to take advantage of the features of the AM-120 board.

The AM-120 Auxiliary I/O Controller board provides the following features:

1. Two full RS-232 serial I/O ports (one of which contains a remote reset line). Connecting a terminal to the port with the remote reset line allows you to reboot the system from that terminal.

2. Three 8-bit parallel output ports and two 8-bit parallel input ports.

3. An interval timer. The length of the timer's interval is software selectable.

4. Power failure detection and handling.

5. A clock/calendar with battery backup. This clock/calendar maintains both the time and the date (including day of the week) even when your system is not on.

6. Memory error interrupt.

For more information on the AM-120, and for information on physically installing the board in your system, see the document Installation Instructions AM-120, (PDI-00120-XX).

2.0 THE SOFTWARE INCLUDED WITH THE AM-120

Below is a list of the software we supply with the AM-120 board:

CAL120.PRG	Calibration program for time-of-day clock oscillator on the AM-120.
DATE.PRG	Reads and sets system date from the AM-120.
TIME.PRG	Reads and sets system time from the AM-120.
AM120.IDV	Interface driver for the AM-120.
AM120.MAC	The source code for the AM120.IDV interface driver.

Note the command reference sheets attached to this document for the CAL120, DATE, and TIME programs. The .PRG files listed above must appear in account [1,4] of your System Disk, and the .IDV file must appear in account [1,6] of your System Disk.

3.0 THE AM120.IDV INTERFACE DRIVER

The AM120.IDV interface driver allows you to use the two serial ports on the AM-120 for terminals or printers. You must also use the driver if you want to use the parallel ports in interrupt driven mode, the memory error interrupt, or the interval timer. (The AM120.IDV interface driver handles all of these interrupts.)

We have designed the AM120.IDV driver so that installing an AM-120 board in a system that uses an AM-100 CPU allows you to use the AM-120 parallel ports, memory error interrupt and interface timer in exactly the same way as you would use those features on the AM-100/T CPU board. This means you may use software originally written for the AM-100/T on the AM-120 with no modification. NOTE: To use the parallel ports in interrupt driven mode, or the interval timer on a second AM-120 board in your AM-100 based system, you must modify the AM120.IDV program by inserting your own code that handles the function you want to perform. We have provided the source code for the interface driver to allow you to do so.

If your system uses an AM-100 CPU, the AM120.IDV driver allows up to two AM-120 boards in the same system. If your system uses an AM-100/T CPU, the interface driver allows one AM-120 board in your system; that board must be physically addressed to the "alternate second board address," and you must reference the serial ports on it as ports 2 and 3. (In this case, the AM-100/T acts as the first auxiliary I/O controller board in the system.) (See the document Installation Instructions AM-120 for information on installing the AM-120 board.)

4.0 USING THE AM-120 SERIAL I/O PORTS

To tell the system that you want to use a terminal or printer that is connected to one of the serial ports on the AM-120, you must include an appropriate TRMDEF statement in your system initialization command file that references the AM-120 interface driver. (For information on the TRMDEF statement and on modifying the SYSTEM.INI file, see the document The System Initialization Command File in the "System Operator's Information" section of the AMOS Software Update documentation Packet.) The TRMDEF statement for the AM120.IDV is very similar in form to the TRMDEF statement for the AM-310 interface board. The interface statement portion of the TRMDEF statement takes this form:

AM120=I/O-port-address{:baud-rate-code}

where you may optionally supply a baud rate code for the terminal. The I/O-port-address is a number from 0-3 and selects the serial port you want to use. (If you are using one AM-120 board with an AM-100, you have serial ports 0 and 1 available; if you are using two AM-120 boards, you have serial ports 0-3 available.) As an example, a TRMDEF statement for a terminal connected to the AM-120 on a system using an AM-100 CPU might look like this:

```
TRMDEF TERM1,AM120=1:37316,SOROC,100,100,100
```

The optional baud rate codes that you can specify are:

:30316	50	baud
:30716	75	baud
:31316	110	baud
:31716	134.5	baud
:32316	150	baud
:32716	300	baud
:33316	600	baud
:33716	1200	baud
:34316	1800	baud
:34716	2000	baud
:35316	2400	baud
:35716	3600	baud
:36316	4800	baud
:36716	7200	baud
:37316	9600	baud
:37716	19200	baud

5.0 SUPPORTING UTILITY PROGRAMS

We have modified the DATE and TIME programs so that you can set the system date and time from the AM-120 clock/calendar, and you can reset the date and time maintained by the AM-120.

On a system with an AM-120, DATE also maintains the day of the week as part of the date. The DATE program now supports European date format as well as American format. (A date in European format looks like this: 26 January 1982; a date in American format looks like this: January 26, 1982.) You may optionally request that DATE automatically reset the date at midnight.

A new program, CAL120, allows you to use a high-frequency counter to calibrate the AM-120 oscillator.

For information on DATE, TIME, and CAL120, see the command reference sheets for DATE, TIME, and CAL120 in the AMOS System Commands Reference Manual (DWM-10010-49).

6.0 INTERNAL FORMATS FOR DATE AND TIME

The formats that AMOS uses for storing the system date and time are the same whether or not your system contains an AM-120 board.

AMOS stores the time of day in two words in system memory as the number of system clock ticks since midnight. (The "system clock" is not the AM-120 clock/calendar or interval timer, but is the clock on the CPU board derived from the AC line frequency.) For information on these two words in system memory (TIME and TIME+2), see the AMOS Monitor Calls Manual, (DWM-00100-42).

AMOS stores the system date in two words in system memory. These two words contain the month, the day, the year, and some flag bits. The byte at the word DATE contains the month, the byte at the word DATE+1 contains the day, and the byte at DATE+2 contains the last two digits of the year. The byte at DATE+3 contains some flag bits, only two of which are currently used. Within the flag byte, bits 0-6 are currently unused; a one in bit 7 means that an AM-120 is present in the system; and, a one in bit 6 means that DATE is using the European date format.

NOTE: Some programmers have in the past assumed that DATE+3 will always contain zero. This is not true; programs that depend on this assumption may experience problems when used on a system that uses the AM-120 software.

April 1981

SOFTWARE INSTALLATION INSTRUCTIONS FOR THE AM-710 MEMORY BOARD

The AM-710 memory board is a 128K byte memory board produced by Alpha Micro. One of the features of this board is its ability to detect parity errors. This document discusses the changes you must make to your system initialization command file when installing the AM-710 memory board. We also discuss what happens when a parity error occurs.

For technical information on the AM-710 board, see the Alpha Micro Integrated Systems User's Guide, (DWM-00101-00). That manual will also tell you how to set the AM-710 I/O port address and how to address the boards.

NOTE FOR BANK SWITCHING SYSTEMS: If your system bank switches memory, you will also want to consult the Alpha Micro Integrated Systems User's Guide for information on what on- and off-constants to specify in the MEMDEF commands that define the memory banks made up of your AM-710 memory boards.

1.0 ENABLING PARITY ERROR DETECTION FOR THE AM-710

To enable the AM-710 memory board's parity error detection feature, you must use the PARITY command. Although you may use this command at AMOS command level, you will almost always want to use it within your system initialization command file, SYSTEM.INI. (For information on changing your system initialization command file, see The System Initialization Command File, in the "System Operator's Information" section of the AMOS Software Update Documentation Packet.)

Place the PARITY command in your system initialization command file after the MEMERR command. Type PARITY followed by the I/O port addresses of the AM-710 boards in your system (separating the addresses by commas). For example:

```
PARITY 100,101,102
```

This I/O port address is three digits for octal numbers and two digits for hexadecimal numbers. (NOTE: Do not enter hex numbers unless you have used the SET HEX command for your job.) If you have more I/O port addresses than will fit on one command line, you may enter multiple PARITY command lines. For example:

```
PARITY 101,102,103,104,105
PARITY 106,107,110,111,112
```

2.0 THE MEMERR COMMAND

The AM-710 memory board requires the presence of the MEMERR command in the system initialization command file as well as the use of the PARITY command. If your system uses only AM-710 boards, do not specify an address after the MEMERR command. For example:

```
MEMERR
PARITY 100,101,102
```

If your system contains Piiceon 32K word boards as well as one or more AM-710s, specify the MEMERR port address required by the Piiceon memory boards. For example:

```
MEMERR 250
PARITY 100,101,102
```

(If your system uses Piiceon 32K word memory boards, refer to the Alpha Micro Integrated System User's Guide for information on the error I/O address to supply to the MEMERR command.)

3.0 PARITY COMMAND ERROR MESSAGES

Section 4.0 below discusses the error messages you can see when a parity error actually occurs. In addition, there are several other messages you can see that result from the improper use of the PARITY command itself:

?There is no AM-710 at port address xxx
: Where xxx is an I/O port address you specified on your PARITY command line. This address did not match the jumpered I/O port address of any of the AM-710 boards in your system. Check the PARITY command line to make sure that you entered the port address correctly; then check the memory boards to make sure that their port addresses are jumpered correctly.

?Command format error
: You did not supply any I/O port addresses, or in some other way used an improper format.

4.0 WHAT HAPPENS WHEN A PARITY ERROR OCCURS?

Your system can respond in a variety of ways when a parity error occurs depending on your system's particular configuration:

AM-100 CPU and AM-710 memory boards - Memory error detection is not supported.

If your system contains an AM-100/T CPU (Pre-Revision F) and one or more AM-710 memory boards:

If a parity error is detected, the monitor sends a "5" to the status display port and halts the CPU.

If your system contains an AM-100/T CPU (Revision F or later) (or an AM-100 CPU and an AM-120 Auxiliary I/O Controller) and one or more AM-710 memory boards:

If a parity error occurs, the monitor will send a "9" to the status display board, and will display the message:

?AM-710 parity error for job xxxxxx

on the Operator's terminal. (The Operator's terminal is the terminal attached to the job the system came up under.) If the job in whose memory parition the parity error occurred is attached to a real terminal (as opposed to a pseudo terminal), that user sees the message:

?Parity error

The monitor then halts that job.

When a parity error is reported, the System Operator should inform all users still running that a memory error has occurred, and ask them to finish up what they were doing and logoff. Then the System Operator should open the computer and inspect each AM-710 memory board.

When a parity error occurs, you can only identify which memory board encountered the error by looking at each AM-710 memory board to see if its LED indicators are lit. Make note of the memory board on which the parity error occurred.

At this point you must decide whether to continue operating with the memory board that generated the parity error or whether to swap the board out for another one. Parity errors can reflect a transient problem that surfaces once and then is never seen again. You may therefore wish to continue running the system as is until a parity error occurs again. If the parity error occurs on the same memory board more than once or twice in a great while, you probably will want to remove the questionable memory board and return it to your dealer for inspection and/or repair.

To reset an AM-710 memory board's parity error indicator, you must perform a hardware reset on your system.

4.1 Other Memory Errors

Note that if your system uses Piiceon 32K word memory boards as well as AM-710 boards, in addition to parity errors reported by AM-710 boards, you may also encounter other types of memory errors reported by the Piiceon boards. If a Piiceon 32K word memory board (only for use with the AM-100/T or AM-100/AM-120) detects an uncorrectable memory error, the monitor sends a

"5" to the status display port and halts the system. You may see which Piiceon board encountered the error by looking at the boards' LED indicators as discussed in the section above.

SOFTWARE NOTICE FOR AM-410 USERS

April 1981
Revision A03

This document reflects AMOS versions 4.4 and later

'Alpha Micro', 'AMOS', 'AM-100',
'AlphaBASIC', 'AlphaPASCAL', and 'AlphaLISP'

are trademarks of

ALPHA MICROSYSTEMS
Irvine, CA 92714

ALPHA MICROSYSTEMS
17881 Sky Park North
Irvine, CA 92714

Table of Contents

1.0 INTRODUCTION

Disks that run under the control of the AM-410 Hard Disk Controller differ somewhat from the other hard disks supported by Alpha Micro. They therefore require different techniques for data backup, disk formatting, and disk initialization. (As of this release, the only disk that runs under control of the AM-410 is the 90-megabyte CDC Phoenix hard disk.) The purpose of this document is to acquaint you with these devices.

The technology that gives the Phoenix disk a much higher density of data than more conventional drives (e.g., the CDC Hawk), also makes media flaws (i.e., bad disk blocks) a much more likely possibility. Therefore it is necessary to use the CRT410 command to certify a Phoenix logical unit before you use it. This certification process identifies any bad tracks on the disk, formats the disk, and initializes it.

IMPORTANT NOTE:

> As of AMOS version 4.4, the system handles devices that contain media flaws differently than earlier versions of the operating system. (That is, AMOS now uses a "bad track" instead of a "bad block" method of keeping track of media flaws.) For more information, refer to AMOS Version 4.4 Method of Handling Bad Disk Blocks, in the "System Operator's Information" section of the AMOS Software Update documentation packet.
>
> The information in this document reflects AMOS 4.4; it assumes that if you have Phoenix devices that were certified with a pre-4.4 version of CRT410, you have converted your Phoenix disk surfaces via the instructions given in AMOS Version 4.4 Method of Handling Bad Disk Blocks. If you have not done this conversion, refer to the previous version of this document in your 4.3 AMOS Software Update documentation packet, Software Notice for AM-410 Users, for instructions on backing up and analyzing your Phoenix surfaces.

CRT410 certifies the Phoenix media by checking each disk track. It creates the file BADBLK.SYS[1,2] and lists any bad tracks on the media in that file. AMOS assigns spare Phoenix tracks (called alternate tracks) to the bad tracks, to be used in their place whenever the system tries to access a block on the bad tracks. The BADBLK command displays the contents of BADBLK.SYS[1,2].

1.1 Important Notes

There are several things you must keep in mind before you begin to use the Phoenix disk:

1. You must not format or initialize a disk that runs under control of the AM-410. That means that you must not use the SYSACT initialize command. (After you use CRT410 to certify the disk, however, you may use SYSACT to add user accounts to the disk.)

2. As of AMOS Version 4.4, you may use the AMOS disk diagnostic tests REDALL, RAZA, and RNDRED on the Phoenix.

3. As of AMOS version 4.4, you may use DSKCPY or COPY to back up a Phoenix surface. Both DSKCPY and COPY have been changed so that they will not write over the BADBLK.SYS[1,2] file on the backup disk, and will not write into bad disk tracks.

4. We have increased the thoroughness and reliability with which CRT410 searches for bad disk areas. This means that certifying a disk now takes longer than under AMOS Version 4.3. certification.

2.0 THE PHOENIX DISK CONFIGURATION

The CDC Phoenix disk contains 5 fixed platters of 15 megabytes each and a removable 15-megabyte cartridge. Each of these platters is a separate logical unit and must be accessed as such. The cartridge always bears the highest unit number for that physical device. (For example, if you have a single Phoenix you are using as a non-System Device, then SMD0: - SMD4: are fixed disks; the cartridge is SMD5:.) The one exception to this rule is the Phoenix physical device being used as the System Device. If the system is running off the cartridge, the System Disk cartridge is DSK0: and the fixed disks are logical units DSK1:-DSK5:. (A second Phoenix on this system follows the general rule above; the cartridge is DSK15: and the fixed disks are DSK10:-DSK14:.)

Each Phoenix logical unit contains 808 (#0-807) tracks, with 36 sectors per track, to give a total of 29088 (decimal) sectors (or disk blocks) on each unit. The Phoenix also contains 15 spare (or "alternate") tracks, #808-822. Each disk block contains 512 bytes. The Phoenix requires a bitmap size of 1818 words.

Note that AMOS allows you to place bitmaps in switchable system memory. (See the document Defining Switchable System Memory, in the "System Operator's Information" section of the AMOS Software Update documentation packet, for information on using this technique to reduce the monitor size.)

The device driver program for the Phoenix disk is SMD410.DVR[1,6] on the System Disk. The bootstrap loader program is SMDLOD.PRG in account [1,4] on the System Disk.

(Changed 30 April 1981)

The paragraphs below discuss adding the Phoenix to your system either as the System Device or as a non-System Device. (Hardware settings on the disk controllers on your system define the device that the system attempts to boot from.)

If you have several Phoenix disks, note that the first physical device contains logical units zero through five (e.g., DSK0:-DSK5:), a second Phoenix device contains logical units ten through fifteen (e.g., DSK10:-DSK15:), a third Phoenix contains logical units twenty through twenty-five, and so on.

2.1 Using the Phoenix as a Non-System Device

If you are not using the Phoenix disk as the System Device (that is, if you boot the system off some other type of device), you will need to follow some simple steps before you can access the Phoenix disk:

1. Rename the Phoenix driver program from DSK0:SMD410.DVR[1,6] to DSK0:SMD.DVR[1,6].

2. Now you need to define the Phoenix disk as a system device. To do this, edit your system initialization command file, SYSTEM.INI, and add the devices SMD0, SMD1, SMD2, SMD3, SMD4, and SMD5 to the DEVTBL command line. This adds the Phoenix to your system device table. (For information on using the DEVTBL command to define devices and on using the BITMAP command to define disk bitmaps, see The System Initialization Command File, (DWM-00100-09, Rev A03), in the "System Operator's Information" section of the AMOS Software Update documentation packet.)

3. Add BITMAP commands to the SYSTEM.INI to define bitmap areas for the Phoenix. (The Phoenix requires a bitmap size of 1818 decimal words.) Remember that you can set up your system so that part of system memory resides in a switchable area of a memory bank. You can thus place your bitmaps in switchable system memory and so reduce the size of the monitor. Now reboot the system with your new SYSTEM.INI. You can now access the Phoenix.

If you want more information on adding new disk devices to your system, refer to the document Defining Non-System Disk Devices in the "System Operator's Information" section of the AMOS Software Update documentation packet. Section 4.0 of that document, "Building a System on a New Device," explains how to convert the Phoenix to the System Device after you have added it to your system as a non-System Device.

The configuration of your Phoenix non-System Device is as follows: The five fixed platters are SMD0:, SMD1:, SMD2, SMD3:, and SMD4:. The removable cartridge is SMD5:.

2.2 Using the Phoenix as the System Device

If the Phoenix is your System Device, you will want to copy all of the system software from the System Disk cartridge down to one of the fixed disks:

1. Turn on the computer, holding down the reset button. CPU power must be on whenever the Phoenix disk is cycled up. Therefore, always turn the computer on before cycling up the Phoenix; always cycle down the Phoenix before turning off your CPU. If you do not follow this procedure, it is quite possible that the data on your Phoenix disk will be damaged. NOTE: We recommend that you always leave your Phoenix drive powered up when you are not using it. This allows the filter system to continually guard the disk surfaces. (Of course, if you are not using your system, you will probably want to write-protect the drive, cycle it down, and turn off your CPU even though you leave the drive powered up.)

2. Insert the Phoenix System Disk Update cartridge according to the instructions accompanying the drive. Cycle it to READY status.

3. The logical unit the system boots off of is ALWAYS known as DSK0:. The System Disk Update cartridge contains the programs SYSTEM.MON[1,4] and SYSTEM.INI[1,4]; therefore the system recognizes it as a System Disk and tries to boot the system off the cartridge. Because you are running off the cartridge, the fixed platters are units DSK1:-DSK5: and the cartridge is DSK0:. (NOTE: Even after you install the system software onto a fixed platter, if you reset the computer with the System Disk Update cartridge mounted, the system tries to boot off the cartridge.)

4. To install the system software on the first fixed platter, use the CPY410 command. This command certifies DSK1: and copies the contents of the cartridge down onto that fixed disk.

5. You can now remove the cartridge and insert a data pack. The next time you reset or turn on the system, the system boots off the fixed disk on which you have installed the system software.

   NOTE: If you do not want to remove the cartridge from the drive, but do not want the system to boot off the cartridge, erase file SYSTEM.MON[1,4] from the cartridge. Now the system cannot recognize the cartridge as a System Disk and therefore will not attempt to boot off it.

   However, it is most important that you always keep an intact System Disk cartridge (i.e., it contains recent system software and SYSTEM.MON and SYSTEM.INI) around so that you can re-install the system should the fixed disk become damaged.

6. The fixed platters are now logical units DSK0:-DSK4: and the non-System Disk cartridge is DSK5:.

3.0 THE CERTIFICATION PROGRAM, CRT410

The CRT410 program certifies a Phoenix logical unit by reading, writing, and verifying every block of the disk. It also formats and initializes the disk.

3.1 Hints and Restrictions

There are several things you should keep in mind before using the certification program:

1. Only the System Operator may run CRT410. Log into the System Operator's account, [1,2], before using the program.

2. You may ONLY use CRT410 on disks that run under control of the AM-410 Controller.

3. Run CRT410 on every logical unit of a Phoenix disk before the first use of that device. Make sure that the logical unit is not write-protected.

4. A logical unit does not have to be mounted before you certify it; CRT410 mounts the disk for you.

5. CRT410 writes data in every byte of the logical unit you are certifying. If there is data on that unit, make sure that you back it up onto another device before using CRT410.

6. CRT410 communicates directly with the AM-410 Controller without going through the Phoenix driver program. Therefore, you MUST NOT run CRT410 at the same time as any other program that accesses devices that run under control of the AM-410.

3.2 Sample Use of CRT410

Below is sample output of a typical disk certification. The next section discusses the questions that CRT410 asks and the messages that it displays.

```
.CRT410 SMD0: (RET)
CAUTION: This program writes to all blocks

Enter maximum acceptable number of bad tracks: 40 (RET)
?15 bad tracks is maximum
Enter maximum acceptable number of bad tracks: 10 (RET)
Display current track? (Y or N):Y (RET)
Current track is: 0
Enter serial number (10 char. max): PAYROLL1 (RET)

Begin certification of SMD0:
Current track is: 1
Current track is: 2
          .
          .
          .
?Track 7 did not verify
?Track 8 did not verify
          .
          .
          .
Current track is: 14
Current track is: 15
          .
          .
          .
Current track is: 807

?2 bad tracks detected
Certification complete
.
```

3.3 Using CRT410

To use CRT410, enter CRT410 followed by the specification of the logical unit you want to certify. For example:

```
.CRT410 SMD5: (RET)
```

You now see the following message:

```
CAUTION: This program writes to all blocks.
```

(If you do not want to continue the certification, you may enter a Control-C at this point.)

CRT410 now asks you several questions:

1. Enter maximum acceptable number of bad tracks:

   Give CRT410 the maximum number of bad tracks that you will accept on the disk you are certifying. If the number of bad tracks that CRT410 finds exceeds this value, CRT410 tells you so and then aborts the certification, returning you to AMOS command level:

   ```
   ?Device has exceeded maximum number of errors
   .
   ```

2. Display current track? (Y or N):

   If you want CRT410 to tell you as it verifies each track, enter a Y; otherwise, enter an N. If you answer Y, CRT410 now displays this message:

   ```
   Current track is: 0
   ```

   NOTE: Asking CRT410 to display the number of the track it is currently verifying greatly increases the length of time it takes to certify a disk surface.

3. Enter serial number (10 char. max):

   You may optionally give CRT410 a ten-character alphanumeric I.D. for the logical unit you are certifying. CRT410 writes this identifier into the BADBLK.SYS file for that logical unit.

After answering the questions above, CRT410 begins to certify the disk. You see this message:

```
Begin certification of Devn:
```

where Devn: is the device specification you supplied on the CRT410 command line.

If you asked CRT410 to tell you its track position as it certifies, you now see a list of messages that can look something like this:

```
Current track is: 1
Current track is: 2
Current track is: 3
      .
      .
      .
```

and so on. When CRT410 encounters a track that does not verify, it tells you so. For example:

```
?Track 15 did not verify
?Track 16 did not verify
      .
      .
      .
```

When finished certifying the disk, CRT410 tells you how many bad tracks it found. For example:

```
?4 bad tracks detected
Certification complete
.
```

3.4 How CRT410 Certifies a Disk

CRT410 follows these procedures when it certifies a disk:

1. CRT410 creates a file named BADBLK.SYS in account [1,2] on the disk you are certifying. This file will contain a list of all of the bad tracks on the disk. If you have specified a serial number, CRT410 writes that information to this file.

2. CRT410 writes one data pattern in every byte on the first track of the disk. Then it reads each byte on that track and verifies the data. (CRT410 checks for CRC errors as well as data verification errors.)

   If any data does not verify, CRT410 places the number of the track in the BADBLK.SYS file and tells you that the track is bad. For example:

   ```
   ?Track 35 did not verify
   ```

   CRT410 writes a total of four data patterns, following the procedure above for each data pattern.

3. CRT410 now moves on to the next disk track and performs these operations again, reporting any bad tracks that it finds (and entering their numbers into the BADBLK.SYS file). CRT410 verifies every track. When it finishes, CRT410 tells you that it is done and how many tracks are bad. For example:

   ```
   ?3 bad tracks detected
   Certification complete
   ```

4. Now CRT410 computes a hash total for the BADBLK.SYS file and stores it in the file. This value provides a validity check that other programs (e.g., BADBLK) can use to make sure that the BADBLK.SYS file is complete and healthy. When AMOS accesses a disk block, it checks to see if that block occurs on a bad track; if so, AMOS uses the alternate track assigned to that bad track instead.

If you interrupt the CRT410 program by typing a Control-C, you see:

```
?Certification incomplete
^C
.
```

and CRT410 intentionally writes a bad hash total to the BADBLK.SYS file. (A bad hash total tells other programs that may later look at the file that the data in the file is not complete and is not to be trusted.)

3.5 CRT410 Error Messages

Below are the error messages you can encounter when using CRT410.

?You must be logged into PPN [1,2] to run CRT410

Because it writes data into every byte on the disk, CRT410 is a dangerous program to run. You must be logged in as the System Operator to certify a disk. Log into the System Operator's account, [1,2], before trying to use CRT410.

?15 bad tracks is maximum

You specified a number greater than 15 as the maximum number of bad disk tracks you would accept on the certified disk. However, the BADBLK.SYS file cannot handle more than 15 bad disk tracks; so, enter a number less than 15.

?Track 0 did not verify. (First track must verify.)

The first track of the disk did not verify. CRT410 cannot continue the certification if the first disk track does not verify, so it now stops the certification and returns you to AMOS command level.

?Track n did not verify.

CRT410 marked track n in the BADBLK.SYS file as a bad track.

?Device has exceeded maximum number of errors

CRT410 found more bad tracks than the value you specified as the maximum number of bad tracks that you would accept. CRT410 now aborts and returns you to AMOS command level.

?S100 data transfer error

An error occurred with the AM-410 controller board. CRT410 aborts and returns you to AMOS command level. If you receive this error several times, you may have hardware problems.

?Nonexistent device

Your device specification on the CRT410 command line is invalid; the system believes that the device does not exist. Check your spelling and try again.

?Certification incomplete
You typed a Control-C to interrupt the disk certification. CRT410 now intentionally writes a bad hash total to the BADBLK.SYS file to let other programs know that the data in the file is incomplete and not to be trusted.

4.0 THE BADBLK PROGRAM

The BADBLK program allows you to see the contents of the BADBLK.SYS file created by the certification program, CRT410. BADBLK also verifies the BADBLK.SYS hash total. In future releases, BADBLK will allow you to modify BADBLK.SYS to rebuild a damaged disk.

BADBLK checks the specified disk to see if it was certified by a pre-4.4 version of CRT410; if it was, BADBLK displays the number of bad blocks on the disk. If the disk was certified by a 4.4 version of CRT410, BADBLK displays the number of bad tracks. (The messages you see will tell you if BADBLK is displaying the number of bad blocks or tracks.)

NOTE: Although AMOS version 4.4 handles disks that have been certified via the "bad block" or the "bad track" method, future releases may not support the old bad block system. Therefore, you must convert all Phoenix surfaces certified by pre-4.4 CRT410 over to the new system by certifying those surfaces with CRT410 versions 4.4 or later.

4.1 Using BADBLK

To use BADBLK, type BADBLK followed by the specification of the device whose BADBLK.SYS file you want to see. Then type a RETURN. For example:

```
.BADBLK SMD1: (RET)
```

If BADBLK found the BADBLK.SYS file, it tells you so:

```
SMD1: BADBLK.SYS[1,2]
```

BADBLK now tells you the serial number associated with that device and the number of tracks or blocks marked as bad on that disk. For example:

```
Serial number: INVENTORY2
Number of bad blocks: 0
```

If there are any blocks or tracks listed in the BADBLK.SYS file, BADBLK lists them for you. For example:

```
Number of bad tracks: 3

35      36      37
```

BADBLK exits and returns you to AMOS command level:

```
EXIT
.
```

NOTE: Track numbers are decimal; block numbers are octal.

4.2 BADBLK Error Messages

You can see the following BADBLK error messages:

?File not found: Devn:BADBLK.SYS
: BADBLK was not able to find the BADBLK.SYS file for the disk you specified. Make sure that the device you specified is a Phoenix disk and that the disk has been certified (i.e., you've run CRT410 on that device).

CAUTION: HASH TOTAL DID NOT VERIFY
: The BADBLK.SYS file contained a bad hash total. This indicates that the data in that file is not to be trusted. Use COPY to copy all files off the logical unit containing that BADBLK.SYS file, being careful not to overwrite the BADBLK.SYS file on the new disk. Then re-certify the disk.

You may also see several system error messages if your device specification is invalid. For example:

?Cannot INIT Devn: - device does not exist
: The system did not recognize the device specification you gave. Check your spelling and try again.

?Cannot READ Filespec - disk not mounted
: The system is unable to read the device you specified on the BADBLK command line because it is not mounted. Use the MOUNT command to mount the disk and try again.

5.0 THE DSKANA PROGRAM

For the most part, the new method of handling disk flaws is transparent. For that reason, DSKANA is able to treat any disk that was certified with a 4.4 or later version of CRT410 in the same way that it does a device that does not contain a BADBLK.SYS[1,2] file.

Index

May 1980

DISK LABELING PROCEDURES

Although removable disk cartridges provide an extremely convenient and portable method of backing up and exchanging data, problems sometimes occur in identifying just what information is on a cartridge if that disk pack does not contain a label on the outside of the pack, or if it is improperly labeled.

To solve this problem, Alpha Micro has developed a set of software which allows you to establish and verify a permanent, identifying label on each disk. The LABEL program writes data to a disk that serves as identification for that disk. Other programs (e.g., MOUNT, XMOUNT.SBR, or your own assembly language programs) read the disk identification information on the disk and tell you which disk is currently mounted.

Being able to permanently label a disk results in several advantages:

1. You may easily determine what disk is mounted, even if you are using the system from a remote site.

2. Your programs can verify that the correct disk is mounted before they begin to change data on that disk.

At the current time, the disk labeling system consists of these new and modified programs:

LABEL	Labels a disk by writing identifying data onto Block Zero of that disk. Also displays a disk's label.
MOUNT	Displays the name of the disk that was just mounted or a list of all mounted disks.
XMOUNT	A BASIC subroutine that your BASIC programs can use. It returns the contents of the disk identification field so that your BASIC programs can verify that the correct disk is mounted.

In future releases, additional programs will make use of the disk label.

This document contains information on the LABEL and MOUNT commands. Also, see the MOUNT and LABEL reference sheets in the AMOS System Commands Reference Manual, (DWM-00100-49). For information on XMOUNT, refer to XMOUNT - Basic Subroutine to Mount a Disk, in the "BASIC Programmer's Information" section of the AMOS Software Update documentation packet.

The last section of this document describes the exact format of the disk label, so that your assembly language programs can access the label information.

1.0 LABEL

The LABEL program gives you a way to label a disk with descriptive information and to display that information. Disk labels are stored in Block 0 of the disk, and are used to allow both you and your programs to verify that the correct disk has been mounted.

There are three situations in which you will want to use LABEL:

1. A disk has never been labeled and you want to give it a label.
2. A disk has a label and you want to display that information.
3. A disk has a label and you want to change it.

1.1 Labeling a Disk

To give a disk a label, enter LABEL followed by the specification of the logical device that holds the disk; then type a RETURN. For example, if you want to label the disk in logical device DSK5:, enter:

.LABEL DSK5: (RET)

Now LABEL asks you for the following information:

Volume Name:

Enter up to 40 characters that describe the disk. The MOUNT program displays this field when it gives the list of the mounted disks on the system. The purpose of this information is primarily to give you a way to identify the contents of the mounted disk.

Volume ID:

Enter up to ten characters as the Volume ID. This field is used by programs to determine if the proper disk has been mounted. The XMOUNT subroutine (an assembly language routine callable by your BASIC programs) returns this field. The MOUNT program displays this field (as well as the Volume Name) when it lists the disks mounted on the system.

Installation:

Enter the name of your installation or company. This field, which may contain up to 30 characters, identifies the site where the disk was created. This information will be particularly useful if you exchange disks among different installations.

System:
Enter the name of the computer system this disk was created on. This field may be up to 30 characters in length, and is especially useful when an installation has more than one computer system.

Creator:
The name of the person who created the disk. This field may contain up to 30 characters.

When you finish entering the requested information, LABEL returns you to AMOS command level. The disk is now labeled with the information you specified. The label also contains the date on which you labeled the disk.

1.2 Displaying a Disk Label

Once a disk contains a label, you may use the LABEL command to display that label. Just enter LABEL followed by the specification of the disk whose label you want to see. Now type a RETURN. For example:

```
.LABEL DSK3: (RET)
Currently labeled as:

Documentation Archives (ARCHIVE01)
Created on 1-Jan-80 at Alpha Microsystems on System B by Jack Smith
Last access: 5-Apr-80

Volume Name: ^C
```

The creation date is the date the disk was labeled; the date of last access is the date the disk was last mounted.

After LABEL displays the label information, type a Control-C to return your terminal to AMOS command level.

1.3 Changing a Disk Label

To change a disk label, enter LABEL followed by the specification of the disk whose label you want to change. Now type a RETURN. For example:

```
.LABEL HWK1: (RET)
Currently labeled as:

Test Data (TESTDATA01)
Created on 17-Apr-80 at Computer Products, Inc. on System 1 by Anne B.
Last access: 21-Apr-80

Volume Name:
```

Now, LABEL asks you for the new labeling information. Answer each question with the new information you want to place in the label. (See Section 1.1, "Labeling a Disk," for information on these fields.) When you enter the last question, LABEL returns your terminal to AMOS command level, and the disk label contains the new information.

1.4 LABEL Error Messages

If you specify a nonexistent device, you see:

?Cannot INIT Devn: - device does not exist

where Devn: is the disk specification you gave. Check your spelling and try again. If you still see this message, use the DEVTBL command to see a list of the valid devices on the system.

If the LABEL command line is not in proper format (for example, if you type LABEL followed by a RETURN), you see:

?File specification error

Check the format of your command line and enter it again. Make sure that you include the colon after the device specification.

2.0 MOUNTING A LABELED DISK

Whenever you change a floppy disk or a hard disk cartridge (whether labeled or not), you must always use the MOUNT command to inform AMOS that the bitmap in memory for that device is no longer valid. If you do not mount a disk when you change it, AMOS has no way of knowing that it may be using the wrong bitmap when it writes data to that device; severe damage to the data on the disk could result. NOTE: Never mount or unmount a disk when other users are accessing that disk; you will damage the file structure on the disk by doing so.

You may use MOUNT to mount a disk, unmount a disk, or to display a list of all mounted disks on the system. NOTE: As of AMOS Release 4.4, you may use the /W switch to tell MOUNT to wait until the specified device is ready before mounting the disk. For more information on using MOUNT to mount or unmount a disk, see the MOUNT reference sheet in the AMOS System Commands Reference Manual, (DWM-00100-49).

When MOUNT sucessfully mounts a disk, if that disk has a disk label stored in Block 0, you see:

x (y) mounted

where "x" is the Volume Name, and "y" is the Volume ID specified when the disk was labeled. (For more information on these two fields of the disk label, see Section 1.1, "Labeling a Disk," above.)
For example:

```
.MOUNT DSK1: RET
System Disk (SYS001) mounted
```

If you want to see a list of all disks that are mounted on the system, type MOUNT followed by a RETURN. For example:

```
.MOUNT RET
Disks mounted:

DSK0:  System Disk (SYS001)
DSK1:  Payroll Data (PRD001)
DSK2:  Payroll Data (PRD002)
DSK3:  Development Disk (DEV001)
DSK4:
DSK5:  Backup Disk #3 (BCK003)
AMS0:  Transfer Disk (TRN001)
```

Each line of the display gives the following information: device specification, Volume Name, and Volume ID. (For example, the Volume Name of the disk in device DSK0: is "System Disk"; its Volume ID is "(SYS001)".)

3.0 CONTENTS OF THE DISK LABEL

Systems programmers may be interested in the exact format of the disk label. Note that not all of the defined fields are presently used by the disk identification software; these unused fields will be used by future releases of Alpha Micro software.

Field	Size	Contents
Header	2 words	125252 : 052525
Volume Name	40 bytes	ASCII text
Volume ID	10 bytes	ASCII text
Creator	30 bytes	ASCII text
Installation	30 bytes	ASCII text
System Name	30 bytes	ASCII text
Creation Date	4 bytes	System date format
Access Date	4 bytes	System date format
Backup Date #1	4 bytes	System date format
Backup Vol. ID #1	10 bytes	ASCII text
Backup Date #2	4 bytes	System date format
Backup Vol ID #2	10 bytes	ASCII text

The fields are used as follows:

Header	Used to flag that this disk is labeled. If the flag words are not correct, certain programs will ignore the label.
Volume Name	An ASCII string that describes the disk. This field is designed to be a description of the contents of the disk (e.g., Archives Disk - Jan/March 1980).
Volume ID	A short ASCII string that describes the disk. This field is used by programs checking for the proper disk being mounted (e.g., ARCHV1).
Creator	An ASCII string that describes who created this disk (e.g., John Hoolihan).
Installation	An ASCII string that names the site where the disk was created (e.g., Acme Computers, Inc.). This is useful when installations interchange disks.
System Name	An ASCII string that gives the name of the particular computer system, within an installation, on which the disk was made (e.g., Purchasing).
Creation Date	The original date on which the disk was labeled.
Access Date	The date the disk was last MOUNTed.
Backup Date #1	The date of the last backup (the "father" backup). This field gives the date of the most recent backup. We do not use this field at this time, but it is reserved for future use.
Backup Volume ID #1	The volume ID field of the disk on which the most recent backup (the "father" backup) exists. This field is not used at this time, but it is reserved for future use.
Backup Date #2	The date of the "backup before last" or "grandfather" backup. This field is not used at this time, but is reserved for future use.
Backup Volume ID #2	The volume ID field of the disk on which the "grandfather" backup exists. This field is not used at this time, but it is reserved for future use.

DISK MAINTENANCE PROCEDURES FOR THE SYSTEM OPERATOR

April 1981
Revision A03

This document reflects AMOS versions 4.5 and later

ALPHA MICROSYSTEMS
17881 Sky Park North
Irvine, CA 92714

Table of Contents

1.0 NOTE TO THE SYSTEM OPERATOR

This document discusses some of the maintenance procedures that you, as the System Operator, must perform on the disks used on the system. We've aimed this discussion at the System Operator (the person in charge of the details of system administration and maintenance) because we assume that most general users of your system will not be concerned with disk maintenance, and because many of the commands we talk about below can be fatal to your data if used carelessly or without full understanding of command operation.

The next few pages tell you how to: 1. format disks; 2. initialize disks; 3. label and identify disks; 4. allocate and change user accounts on a disk; 5. pack a disk; 6. perform disk diagnostic tests; 7. recover from disk errors; and, 8. perform disk backup.

The System Operator should establish a regular schedule for disk backup and disk diagnostic tests. For example, an hour every Tuesday and Friday morning might be set aside for disk diagnostic tests and disk backup. If you are changing or creating a lot of data or if you are particularly concerned about the security of your data, you might want to back up your disks once or twice every day.

Because many of these procedures must be done when only the System Operator's job is accessing the disks, it is wise to schedule them during a time when other users are off the system (for example, early morning or weekends).

NOTE:

Previous versions of this document used the term "file record." However, the use of the word "record" can cause some confusion because it is sometimes used in other documents to mean several different things. To help make our terminology more consistent, we have adopted the following conventions:

A physical record is the sector on the disk. This is the actual, physical grouping of data on the disk. The hard disk devices currently supported by Alpha Micro use a physical record size of 512 bytes. Physical record sizes for floppy disk devices vary depending on the device, and may range from 128 bytes to 512 bytes.

A disk block is the logical grouping of data on the disk that AMOS uses when reading from and writing to the disk. AMOS always transfers data one disk block at a time. A disk block may be made up of one or more physical records. Each disk block has a number associated with it that AMOS uses to reference that block. Disk blocks (except in the special case of devices that use the IMG device driver) are always 512 bytes long.

A logical record is the logical grouping of data on the disk as structured by your programs, and has little to do with the physical records or disk blocks on the disk, except that a logical record may not be larger than a disk block. (For example, a BASIC program might set up a data file in which every logical record is just large enough to contain customer addresses and names-- 60 bytes, for instance.)

1.1 Important Note to Phoenix Hard Disk Drive Users

As of AMOS Version 4.4, bad block handling has changed for devices that run under control of the AM-410 (e.g., the Phoenix hard disk drive). Make sure that you read the document AMOS Version 4.4 Method of Handling Bad Blocks in the "System Operator's Information" section of the AMOS Software Update Documentation Packet before you use DSKCPY, CRT410, or DSKANA on a Phoenix disk, if that disk was certified by a pre-4.4 version of CRT410.

1.2 Maintaining a Disk

There are four stages in the life of a disk:

1. When it is brand new, you set a disk up for initial use. First you format the disk; then you initialize it. IMPORTANT NOTE: Do not format or initialize disks that run under control of the AM-410 Hard Disk Controller. Instead, use the CRT410 command to certify the disk. See Software Notice for AM-410 Users in the "System Operator's Information" section of the AMOS Software Update Documentation Packet.

   Next you may decide to use the LABEL program to write an identifying label to the disk. (Of course, even if you don't use LABEL, you will always put a physical label on the outside of a hard disk cartridge or a floppy diskette to identify that disk.)

   Finally, you will need to establish the account structure on the disk by allocating user accounts.

2. After you have set the disk up, you use it. That is, you transfer files to the disk as well as create new files on it. During this time you may allocate additional user accounts and change existing ones. You may also use LABEL to re-label the disk as the need arises.

3. You maintain and protect the data on the disk by performing frequent disk backups and by running diagnostic tests that look for device and media problems. If diagnostic tests indicate problems, you may have to reconstruct the data on a damaged disk.

4. After a disk has been in use for some time, you may want to recycle the disk by initializing it (or re-certifying it in the case of

disks that run under control of the AM-410); this clears all data from the disk. Before re-using a disk, however, run some disk diagnostic tests on it to be sure that the disk media is healthy. After initializing or re-certifying the disk, the old account structure is gone, so allocate new user accounts.

2.0 FORMATTING DISKS

A new floppy or hard disk must be formatted and initialized (or certified) before you can write data to that disk. Formatting a disk sets the disk up so that it is organized into a specific pattern; it is then ready to receive data written in that same pattern. Whenever the system performs an action on a file, it knows the format it must use to read or write that file because of the device specification you include in the specification of the file. For example:

```
.COPY DSK1:=AMS0:NEWFIL.TXT (RET)
```

tells the system to read the file NEWFIL.TXT in AMS format (since you specified device AMS0:), and copy it to Drive One of the System Device (in whatever format is used by DSK1:). Disks to be used in device AMS0: must be formatted in AMS format; disks to be used in device DSK1: must be formatted in whatever format is used by the System Device.

You only need to format disks that have never been formatted before or whose format you wish to change (e.g., to change a floppy disk from STD to AMS format). New floppy disks usually come preformatted in STD format. You should NOT format these disks if you plan to use them in STD format. (When you format a disk in a disk drive, that disk may take on certain characteristics of that drive. So, when you buy a disk already formatted, you should refrain from formatting the disk if you can; this helps to ensure that you can read that disk on different disk drives.)

ICOM floppy disk users may not format disks because the ICOM disk controller does not support disk formatting. ICOM floppy disk users must therefore buy their disks preformatted.

NOTE: Formatting a disk destroys all data on that disk.

2.1 Disk Certification

Before we continue with our discussion of disk formatting, it is important to mention again this important warning: if a disk runs under control of the AM-410 (e.g., a Phoenix disk), you must not format or initialize that disk. Instead, you must certify it via the CRT410 command. CRT410 also formats and initializes the disk. See Software Notice for AM-410 Users in the "System Operator's Information" section of the AMOS Software Update Documentation Packet.

(Changed 30 April 1981)

2.2 Disk Formats

Alpha Micro supports several different disk formats. For information on the disk formats available, and for information on converting floppy System Devices from one format to another, refer to the documents titled Disk Drivers and Formats, Configuring Floppy-Disk Drivers, and Defining Non-System Disk Devices, in the "System Operator's Information" section of the AMOS Software Update Documentation Packet.

2.3 The Disk Formatting Programs

After you have decided on the format you want your disks to use, you must use the proper formatting program to format those disks. The major disk formatting programs used by Alpha Micro are:

FMT500	The formatting program for disk devices used under the control of the AM-500 Hard Disk Controller (e.g., the CDC Hawk hard disk).
FMT200	The formatting program for disk devices used under the control of the AM-200 Floppy Disk Controller (e.g., Persci and Wangco floppy disks). Formats single sided, single density floppy disks.
FMT210	The formatting program for disk devices used under the control of the AM-210 Double Density Floppy Disk Controller (e.g., Wangco disks). Formats single- and double-sided, and single- and double-density floppy diskettes.
FMT400	The formatting program for disk devices used under the control of the AM-400 Hard Disk Interface (e.g., Century Data Trident disks). Formats each logical unit as a separate device.

To use one of the formatting programs listed above, enter the name of the appropriate formatting program followed by the specification of the device holding the disk you want to format. Your device specification tells the formatting program which disk driver program you are going to be using on the disks in that device, and therefore which disk format to use. (For information on configuring a floppy disk driver for your particular combination of disk controller, disk drive and disk format, see Configuring Floppy Disk Drivers in the "System Operator's Information" section of the AMOS Software Update Documentation Packet.) Suppose that you are using a Persci drive to read and write AMS-format floppy disks. To format a disk in Drive Zero in AMS format, enter:

```
.FMT200 AMS0: RET
```

The formatting program now says:

```
BEGIN FORMATTING
```

When the program has finished, you see:

```
EXIT
.
```

The formatting programs FMT200, FMT210, FMT400, and FMT500 all work in the same way as our example above. FMT500 requires that you mount the disk before you format it (use the MOUNT command); the other formatting programs do not require that the disk be mounted.

IMPORTANT NOTE: No other job may run on the system while you format a disk, so make sure all users are off the system before you use one of the formatting programs listed above.

3.0 LABELING AND IDENTIFYING A DISK

Of course you will always place a physical label on the outside of a disk pack or floppy diskette to identify that disk. You may also write a label to the disk itself. The LABEL program allows you both to establish and display a disk label.

The disk label helps you to identify the disk. Another method of checking the contents of the disk is to create a hash total for that disk. (A hash total is a number that uniquely identifies a group of data.) The HASHER program allows you to generate a hash total for a specific disk. The hash totals for two disks will only be the same if the contents of those disks are identical.

3.1 Labeling the Disk

Labeling a disk results in two advantages: you can determine what disk is mounted, even if you are at a remote site, and your programs can check to be sure that the proper disk is mounted before changing data on that disk.

To label a disk, enter LABEL followed by the specification of the device that contains the disk you want to label. For example:

```
.LABEL DSK3: RET
```

Now LABEL asks you for the following information:

Volume Name:
Enter up to 40 characters. This field describes the disk (e.g., Payroll Data from Jan-March 1980). The MOUNT program displays this information when it mounts the labeled disk.

Volume ID:
Enter up to ten characters as the Volume I.D. This field is used by programs to determine if the proper disk has been mounted. (MOUNT displays this field, too, when it mounts the disk.)

Installation:
Enter the name of your installation or company. This field may contain up to 30 characters, and identifies the site where the disk was created (e.g., CompuWord, Inc.).

System:
Enter the name of the computer system this disk was created on. This field may be up to 30 characters in length.

Creator:
Enter the name of the person who created the disk. This field may contain up to 30 characters.

When you finish entering the required information, LABEL returns you to AMOS command level.

Once a disk contains a label, you can use LABEL to display that information. Type LABEL followed by the specification of the device that contains the disk whose label you want to see; then type a RETURN. For example:

```
.LABEL HWK1: (RET)
```

Since the disk already has a label, LABEL displays that information for you. For example:

```
Currently labeled as:

Documentation Archives (ARCHIVE01)
Created on 12-APR-80 at BeachTogs, Inc. on System C by J.K.Milne
Last access: 8-JUN-80

Volume Name:
^C
```

Notice that LABEL incorporates into the disk label the date the disk was labeled (creation date) and the date the disk was last mounted ("last access"). Now LABEL begins to ask you for the various label fields. If you do not want to change any of the information in the label, type a Control-C. Otherwise, you may answer the questions as in the paragraphs above.

For more information on LABEL, refer to Disk Labeling Procedures in the "System Operator's Information" section of the AMOS Software Update Documentation Packet.

(Changed 30 April 1981)

3.2 Identifying a Disk

The HASHER program allows you to generate a hash total for a specified disk. This gives you a number that you can compare against the hash total of another disk to see if the disks contain identical data. You will probably find HASHER to be particularly helpful when you are making multiple copies of a disk via DSKCPY. If the hash total of the master disk does not match the hash totals of the disks copied to, the copies are not perfect.

To use HASHER, enter HASHER followed by a RETURN:

```
.HASHER (RET)
```

When the system asks for the input drive, enter the name of the disk for which you want hash totals:

```
Input drive: DSK2: (RET)
```

Now you see:

```
[Hashing nnnn blocks]
```

where nnnn is the number of blocks on the specified disk.

When HASHER is finished, you see:

```
Hash is: nnn
```

where nnn is the hash total for the specified disk.

NOTE TO HAWK HARD DISK USERS: When you use DSKCPY on a Hawk disk, DSKCPY allows you to use the special fast copy mode or the slower /0 mode. (If you use the DSKCPY fast copy mode for Hawk devices, no other user may run on the system while the copy is taking place; the /0 mode, while slower, allows users to continue running on the system.) DSKCPY optionally generates a hash total for a disk, but it generates the hash total differently depending on whether you are using the fast copy mode or the /0 mode. That means that the hash total differs for the same disk, depending on the mode in which the disk copy was made.

To allow you to use HASHER to generate a disk hash total that is compatible with one generated by DSKCPY, HASHER also has a Hawk fast copy and a /0 mode. Just as with DSKCPY, HASHER does not allow you to run other users on the system if you use the default Hawk fast copy mode. You see:

```
%All other users will be suspended while HAWK hash is running.
Hit return to continue or control-C to abort:
```

If no other users are on the system, you may type a RETURN. If you want HASHER to generate a hash total in the same way that DSKCPY does when it uses the /0 mode, use the HASHER /0 switch. For example:

```
.HASHER /0 (RET)
```

You may use HASHER with the /O switch while other users are running on the system. For more information on HASHER, see the HASHER reference sheet in the AMOS System Commands Reference Manual, (DWM-00100-49).

4.0 USING THE SYSACT COMMAND

The SYSACT command allows you to perform several disk maintenance functions. You may use SYSACT to: 1. initialize (that is, clear) a disk; 2. allocate user accounts on a disk; 3. change account passwords; 4. delete user accounts; and, 5. display a list of accounts (and passwords) on a disk.

4.1 Initializing a Disk

After you have formatted a disk, you must initialize it if you are going to build an account structure on that disk. If you are simply going to transfer a literal image of another disk onto the newly formatted disk (via DSKCPY) you do not need to initialize it. (Remember that if you are using a disk that runs under control of the AM-410, you must not format or initialize it.)

Initializing a disk writes zeros in the Master File Directory and the bitmap. If you wish to clear the data on a disk, you do not need to reformat it (unless you want to change the format that it uses); just re-initialize it.

NOTE: Once you have initialized a disk, there is no way to access the data on that disk-- you have, in effect, erased the disk.

To initialize a disk:

1. If the disk drive you are using allows it, write-protect all devices that you are not going to initialize.

2. Log into the System Operator's account:

   .LOG [1,2] RET

3. Mount the disk you want to initialize. Type MOUNT followed by the specification of the device holding the disk. Type a RETURN. For example:

   .MOUNT AMS1: RET

4. Type SYSACT followed by the specification of the device that holds the disk you want to initialize:

   .SYSACT AMS1: RET

   You now see the SYSACT prompt, *.

5. Be very sure that you have specified the correct device. To make sure that the disk is empty, use the SYSACT L command; type an L followed by a RETURN:

```
*L (RET)
```

If the disk has already been initialized and is in use, you see a list of PPNs and passwords.

If the device you specified does not hold the disk you want to initialize, type an E and a RETURN to exit SYSACT and return to AMOS command level.

6. If you have specified the correct device, type an I followed by a RETURN:

```
*I (RET)
```

7. SYSACT now tells you:

```
Initializing the disk clears all files - enter Y to confirm:
```

SYSACT initializes the disk only if you enter a Y followed by a RETURN. When it has finished, SYSACT displays its prompt symbol.

8. To exit SYSACT and return to AMOS command level, type an E followed by a RETURN:

```
*E (RET)
.
```

4.2 Building the Disk Account Structure

After you have initialized a disk, that disk is now ready to receive data. You can now begin to build accounts and files on the disk. Use SYSACT to allocate user accounts.

All files on a disk are associated with an account on that disk. Usually, you may not write any files to the disk until a user account exists to hold those files. (However, you can copy files onto an initialized disk that has no account structure if you copy from the System Operator's account, [1,2]. In this case, the COPY command allocates the proper accounts for you on the new disk as it copies over the files.)

You can only access a file if you: 1. log into the account which contains the file; or, 2. specify the account the file belongs to within that file's specification.

Each account on the disk has a directory associated with it (called a User File Directory or UFD) that lists the files in that account. Every disk has one Master File Directory (called the MFD) that maintains a list of all UFDs on that disk. When you first initialize a disk, SYSACT creates the MFD on the disk, but no UFDs exist. As you use SYSACT to allocate user accounts, SYSACT creates the UFDs for those accounts.

4.2.1 Allocating User Accounts

1. LOG into the System Operator's account:

   .LOG [1,2] RET

2. Mount the disk you want to use the SYSACT command on.

   .MOUNT DSK1: RET

3. Type SYSACT followed by the specification of the device holding the disk you want to allocate user accounts on. Then type a RETURN.

   .SYSACT DSK1: RET

   You now see the SYSACT prompt, *.

4. To see a list of the SYSACT commands you can use, type an H followed by a RETURN:

   *H RET

5. To allocate an account, type an A followed by a space and the project number and programmer number of the account you want to allocate. (Separate the numbers with a comma.) These numbers must be octal, and must not be greater than 377. For example, suppose you want to allocate the user account DSK1:[100,1]:

   *A 100,1 RET

   Now SYSACT asks you if you want to assign a password to that account:

   PASSWORD:

   You may enter a password of six characters or less. If you enter just a RETURN, SYSACT assigns no password to the account.

6. If you want to allocate another account, use the A command again. Follow the procedure above until you have added all accounts that you wish to allocate.

7. To exit from SYSACT and return to the monitor, enter an E after the prompt; then type a RETURN:

```
*E (RET)
.
```

You may not create more than 63 user accounts on a single disk; that is the maximum number of entries in the Master File Directory.

4.2.2 Changing and Deleting User Accounts

Once you have created user accounts on a disk, you can begin to create and transfer files on that disk. At any time you may change or delete user accounts by using the SYSACT command.

NOTE: Always erase any files in an account before deleting it. SYSACT won't let you delete an account if there are files in that account.

1. Log into the System Operator's account:

```
.LOG [1,2] (RET)
```

2. Mount the disk on which you want to use SYSACT:

```
.MOUNT AMS1: (RET)
```

3. Type SYSACT followed by the specification of the device that contains the disk whose accounts you want to change or delete. Type a RETURN:

```
.SYSACT AMS1: (RET)
```

   Now you see the SYSACT prompt, *.

4. If you want to see a list of all accounts already allocated on that disk, type an L followed by a RETURN.

5. To change the password of an account:

   a. Enter a C followed by a space and the project-programmer number of the account whose password you want to change. For example, to change the password of account [110,6]:

```
*C 110,6 (RET)
PASSWORD: MILO (RET)
```

   The new password for account [110,6] is now MILO.

b. To remove a password entirely from the account, enter just a RETURN:

```
*C 110,6 (RET)
PASSWORD: (RET)
```

6. To delete a user account, enter a D followed by a space and the project-programmer number of the account you want to delete:

```
*D 110,6 (RET)
```

7. To exit from SYSACT, type an E followed by a RETURN:

```
*E (RET)
.
```

5.0 DISK DIAGNOSTIC TESTS

There are a number of diagnostic tests you can run to check your disk media, the disk controller and the physical device itself. (NOTE: As of AMOS Version 4.4, you may use any of the disk diagnostic programs below on a disk that runs under control of the AM-410.)

Before running any of the tests that we discuss below, it is a good idea to use the SET DSKERR command:

```
.SET DSKERR (RET)
```

If you do not use SET DSKERR, the system reports only hard errors. Once you have SET DSKERR, the system reports any soft errors that occur, and tells you at what disk location the error occurred. (NOTE: The system makes an exception for hard disks used with the AM-500 Hard Disk Controller. Even if you use SET DSKERR, the system reports only hard errors for such devices. However, using SET DSKERR does tell the system to report the disk location at which the hard error occurred.)

REMEMBER: SET DSKERR only affects error reporting for the job that used the SET command.

A soft error is a read-error. When a soft error occurs, the system has to retry reading the data in a specific disk location. The system does not report soft errors unless you use SET DSKERR. When a set number of soft errors have occurred at the same disk location (usually eight), the system reports a hard error. An occasional soft error is not in itself an indication of serious problems, but frequent soft errors may indicate maladjustments in the physical device or disk controller, or problems with the disk media itself. The particular message the system uses to report a soft error depends upon the type of device; check with Appendix A of the AMOS User's Guide, (DWM-00100-35), for a list of soft error messages. For example, if the system had to retry reading a disk block on a floppy disk drive five times, you might see something like this:

```
CRC Error - AMS1: Block 145
CRC Error - AMS1: Block 145
CRC Error - AMS1: Block 145
CRC Error - AMS1: Block 145
CRC Error - AMS1: Block 145
```

A hard error occurs when the system has repeatedly tried to read the same disk location, but has failed to do so. A hard error is a serious matter, since it indicates that the system has given up trying to read the disk block affected.

If any of the disk diagnostic tests are not able to complete an analysis because of a hard error, they tell you so:

```
?Cannot READ Filespec - device error
```

If you have used the SET DSKERR command, the system tells you where on the disk the hard error occurred. For example:

```
AM500 ERROR CODE 4 FOR DRIVE 1 BLOCK 12 (CYLINDER 0 HEAD 0 SECTOR 12)
```

To see what the error codes for a specific disk drive mean, consult the hardware documentation that accompanied that drive. (For example, the error message above occurred on a CDC Hawk hard disk running under control of the AM-500 Hard Disk Controller. For that specific disk drive, an error 4 is a CRC Error-- a Cyclic Redundancy Check error.)

For information on recovering from disk errors reported by the diagnostic tests, refer to Section 6.0, "Recovering From Disk Errors."

5.1 REDALL and RNDRED

Both REDALL and RNDRED perform read tests on a specified hard or floppy disk. REDALL reads all disk blocks (or the number you specify) beginning with the first block on the disk. RNDRED performs random-read tests.

Neither REDALL nor RNDRED alter the data on your disk; they merely read the data and report any read errors that occur.

To use REDALL:

1. Enter REDALL followed by the specification of the device you want to read. Type a RETURN.

   ```
   .REDALL DSK1: RET
   ```

   The command above tells REDALL to read all blocks on the specified disk.

2. If you don't want REDALL to read all blocks on the disk, follow the disk specification with the number of blocks you want read:

   ```
   .REDALL DSK2:100 RET
   ```

(Do not put a space between the device specification and the number of blocks.) The command above tells REDALL to read the first 100 blocks on DSK2:.

3. REDALL now tells you the number of blocks it is reading:

```
.REDALL DSK3: RET
Reading 9696 blocks
EXIT
.
```

4. REDALL exits when it finishes reading the blocks. If any errors occurred, REDALL tells you so by displaying the appropriate error message on the screen.

To use RNDRED:

1. Enter RNDRED followed by the specification of the device you want to test. Type a RETURN:

```
.RNDRED AMS1: RET
```

RNDRED now randomly selects a disk track and performs a seek and read operation on a random block of that track. RNDRED continues on, selecting and reading disk locations at random. You see nothing on the screen unless RNDRED finds an error.

2. When you wish to exit RNDRED, type a Control-C; otherwise, RNDRED continues until you reset the system.

RNDRED and REDALL can display the usual system error messages that result from an invalid device specification. For example:

```
.RNDRED DSK2: RET
?Cannot READ DSK2: - device not mounted

.REDALL ASM2:200 RET
?Cannot READ ASM2: - device does not exist
```

5.2 DIAG2

DIAG2 tests floppy disks by performing read/write tests. It does not verify write operations and does not destroy the data on your diskette.

To use DIAG2:

1. Enter DIAG2 followed by the specification of the device you want to test. Type a RETURN:

```
.DIAG2 STDO: RET
```

2. DIAG2 waits until you are ready. Then it proceeds with the test:

```
Hit return when ready: RET
Test 1 - track 0 read/write
Test 2 - track 76 read/write
Test 3 - random seek-verify 500 times
Test 4 - speed seek tracks 0 and 76 10 times
EXIT
.
```

If any errors occur, DIAG2 displays the appropriate error messages. If you give DIAG2 an invalid device specification, you can also see standard system error messages (e.g., - device does not exist).

5.3 DSKANA

Use of the DSKANA command is a very important part of your disk maintenance routine. DSKANA analyzes the data on a specified disk and rewrites the bitmap. DSKANA also reports lost and mislinked disk blocks, inconsistent block counts, and other file errors. Use DSKANA frequently on every disk on the system. (You might make it a practice to use DSKANA on every disk just before you back it up.)

NOTE:

NEVER use DSKANA while other users are accessing the specified disk; to do so may damage the bitmap and the files on the disk. Make sure that the disk you are analyzing is write-enabled; DSKANA must be able to rewrite the bitmap out to the disk. Before you use DSKANA, you must log into account [1,2].

5.3.1 Displaying the DSKANA Option Summary

DSKANA operates in several different modes. For example, the default mode tells DSKANA to display a list of PPNs as DSKANA analyzes the disk accounts and then to give the final disk analysis messages that report the results of the analysis. If you would like more information (such as a display of all disk blocks and files on the disk, including the blocks and files in which file errors (if any) occurred), use the List (/L) option. If you want just a list of the disk blocks and files in which file errors (if any) occurred, use the Errors only (/E) option. If you want to send the DSKANA display to a file, specify an output file.

If you become confused about the DSKANA options, enter DSKANA followed by a RETURN:

```
.DSKANA (RET)
```

DSKANA displays a summary of the modes and options available to you.

5.3.2 The DSKANA Default Mode

Log into [1,2]. Now, enter DSKANA followed by the specification of the device that contains the disk you want to analyze. Then type a RETURN. For example:

```
.DSKANA DSK1: (RET)
```

You now see:

```
[Begin analysis of Devn]
```

where Devn is the device you specified. If you are not using DSKANA on a device that runs under the control of the AM-410 hard disk controller, you see nothing more for some minutes, except for a list of PPNs as DSKANA proceeds through the accounts on the disk. Then you see some messages that tell you the results of the analysis. For example:

```
.DSKANA HWK1: (RET)
[Begin analysis of HWK1]

[1,2]
[1,4]
[10,6]
   .
   .
   .
[110,1]
[300,20]
[The following blocks were marked in use but not in a file]
      1767   1772   2562   3456   6265   10270  11555  11567
     11661  12272  12303

[The following blocks were in a file but not marked in use]

[Rewriting BITMAP]

no file errors
.
```

Below we discuss the messages that you see at the end of the disk analysis:

1. [The following blocks were marked in use but not in a file]

   It is quite likely that numbers will appear beneath this message; if they do, it is nothing to worry about. These are the addresses of disk blocks that the system has previously marked as being in use during intermediate operations. In fact, one reason to run DSKANA frequently is that it reclaims these temporarily allocated blocks so that they can be used by files.

2. [The following blocks were in a file but not marked in use]

   A list of numbers under this message is an indication of problems in the disk file structure. Somehow the block linking structure of the disk has gone astray. If you are using a disk that runs under control of the AM-410 and that was certified via a pre-4.4 version of CRT410, this message can also indicate that a block marked as bad in the BADBLK.SYS file was not marked in use in the bitmap. (The pre-4.4 version of CRT410 marks all bad blocks in use in the bitmap.) You must take immediate steps to restore the integrity of the data on your disk. (See Section 6.0, "Recovering From Disk Errors.")

3. [Rewriting BITMAP]

   After performing its disk analysis, DSKANA always rewrites the bitmap so that it reflects the true block allocation on the disk. Before it reconstructs the bitmap, DSKANA compares the bitmap hash total with the total stored in the bitmap itself. If the two do not agree, DKSANA tells you so:

   [BITMAP on disk had a bad hash total]

   (A hash total is a computed value used to check the integrity of a group of data.) This message can be an indication of read/write errors, but is not necessarily anything to worry about. It would be a good idea to use the SET DSKERR command so that you will be made aware of any soft disk errors that occur in the future. If this message occurs more than once in a great while as you use DSKANA, you may have a hardware problem.

4. no file errors

   If you see the message above, you know that DSKANA has completed the disk analysis, and that the file structure on the disk is intact.

   If the message instead says something like:

   5 file errors

   you have a serious problem. The file structure on the disk is in error, and you are going to have problems in recovering the data on the disk. For information on coping with this problem, see Section

6.2, "DSKANA File Errors." The first thing you will have to do is to run DSKANA again with the /L or /E switch, so that you can see where on the disk the file errors occurred. You must use the /L or /E options if you are to see the file error messages that indicate exactly what is wrong with the file structure on the disk.

NOTE TO PHOENIX DRIVE USERS:

If you are using a device that runs under the control of the AM-410, before DSKANA begins its analysis, it looks for the file BADBLK.SYS[1,2] on the disk. This file contains a list of any bad blocks or tracks on that disk. If the disk was certifed by an AMOS 4.4 or later version of CRT410, DSKANA ignores the BADBLK.SYS file and analyzes the disk in the same way that it analyzes any other disk.

If the disk was certified by a pre-4.4 version of CRT410, DSKANA must take into account the information in BADBLK.SYS. First, CRT410 checks the hash total of BADBLK.SYS. If the hash total is bad, you see:

```
[BADBLK.SYS contains a bad hash code]
```

You then know that the original certification was not allowed to finish or that the file was damaged in some way. If you want to see if anything else is wrong with the disk, let the disk analysis continue. Or, you may exit DSKANA by typing a Control-C. In either case, you should use COPY to copy all files off the disk, since DSKANA is using information in BADBLK.SYS that is of doubtful integrity. Then re-certify the disk.

If the BADBLK.SYS hash total was OK, and if you are using the /L switch (see below), DSKANA now prints the numbers of any bad blocks. For example:

```
[bad disk blocks]
     334   335   2035
```

Now DSKANA continues on with the disk analysis, displaying a list of PPNs as it analyzes the disk accounts.

5.3.3 Using the DSKANA List Option

If you want to see more information on how DSKANA is proceeding with its analysis of the disk besides just the PPNs of the accounts it is analyzing, select the List option by including the /L switch at the end of the command line. For example:

```
.DSKANA AMS1:/L RET
```

The analysis proceeds as in the example above, but now you also see: 1. the disk address of the account, and 2. a list of all files in the account along with the disk addresses used by the blocks in those files.

For example, you might see the following information for a small account:

```
[200,1]
Directory        6627
INDEX   HLP      11430
GLOSRY  TXT      3444    3445    3446    3447    3450    3451    3452
                 3607    3610    3611    3614    3615
HEADER  TXT              4130    4632
```

The account [200,1] has three files: INDEX.HLP, GLOSRY.TXT, and HEADER.TXT. The directory for the account appears at disk address 6627 and takes up only one disk block. INDEX.HLP takes up one disk block (11430). GLOSRY.TXT takes up twelve disk blocks. HEADER.TXT takes up two disk blocks (4130 and 4632).

If a file error is on the disk, you see a file error message in the appropriate spot in the DSKANA display. For example:

```
[Begin analysis of DSK3:]

[1,4]
Directory          143     354     712     1126    4010
ME       BAS       144     145     146     147     150
Block    0 - block reserved for system use only in DSK3:ME.BAS[1,4]
```

(For a list of the file error message, see Section 6.2.1, "DSKANA File Error Messages.")

5.3.4 Using the DSKANA Errors Only Option

If you want to see a list of only the disk blocks and files in which errors occurred, use the /E option. For example:

```
.DSKANA HWK1:/E (RET)
```

The display you see looks just like the one you would see if you were using no switches at all, except that if DSKANA finds a file error, you see the disk block and file in which the error occurred. For example:

```
.DSKANA DSK3:/E (RET)
[Begin analysis of DSK3]

[20,1]
[30,4]
Block   731 - Block used in previous file in DSK3:GLIDX.RUN[30,4]
[30,5]
                .
                .
                .
```

5.3.5 Specifying an Output File

You may tell DSKANA to send its display to a disk file by specifying an output file on the DSKANA command line. For example:

```
.DSKANA DSK0:ERRORS.LST=DSK1:/L (RET)
```

If the specified file already exists, DSKANA deletes it before beginning the analysis. When DSKANA is finished, you may use the TYPE command to look at the file or you may use the PRINT command to print it. DSKANA always writes the current date into the file as the first line of that file. For example:

```
Disk analysis list file on 7/12/80
```

6.0 RECOVERING FROM DISK ERRORS

Disk errors can come to your attention in one of two ways: 1. A program on the system (e.g., one of the disk diagnostic programs) can report a soft or hard error on the disk; or 2. DSKANA can report file errors (indicating that the blocks on the disk are incorrectly linked).

In either case, you must immediately do what you can to restore the integrity of the data on the disk. Remember: the procedures below are aimed at effecting a partial recovery of your data. Once the linking structure of your disk or the disk media itself goes wrong, retrieving the data on that disk is difficult. The most effective measure is a preventive one: that is, run DSKANA regularly, so that if trouble does occur, you catch it before it has done major damage to your data. Make frequent backups so that you can easily restore damaged data.

6.1 Handling Hard and Soft Disk Errors

If the system has trouble reading a disk location (a soft error), it retries that read operation eight times before it gives up and declares that disk block to be unreadable. If you do not have the SET DSKERR option in effect, the system does not report these retries; instead, after eight soft errors occur, the system reports a hard error.

If you begin to see soft errors when reading a particular disk, it is a good idea to follow the disk cleanup procedures below before the soft errors can develop into hard errors.

6.1.1 Cleaning Up the Disk

If you have made no changes to the files on the disk since your last backup was made, all you have to do at this point is clear the disk by formatting it (using the appropriate formatting program-- see Section 2.3, "The Disk Formatting Programs") and initializing it (see Section 4.1, "Initializing a Disk"). Then use the REDALL program to read the freshly initialized disk to make sure that the disk is OK. If the disk media seems to be healthy, you

can copy your backup onto the empty disk. Remember, however, that in the case of disks which run under the control of the AM-410, you may NOT format or initialize the disk. Instead, you must re-certify the disk with CRT410.

If you have changed some files since your last backup, you must attempt to save the data on the damaged disk. If the system can still read the disk blocks (that is, if you have soft rather than hard errors), use the COPY command to copy all files over to a good disk. (Use the REDALL program to make sure that the output disk is good before copying over to it.)

If you are dealing with a hard error, the system is not able to read at least one of the disk blocks. Use the SET DSKERR command. Then run REDALL to try to read the disk. REDALL then tells you which disk blocks are unreadable. Write those bad block numbers down.

Now there are a couple of things you can try. You may be able to fix the hard error by simply recomputing the block CRC (Cyclic Redundancy Check). Use the DSKDDT program to do so:

1. Enter DSKDDT followed by the specification of the questionable logical device. Then type the number of the disk block you want to check. Hit RETURN. For example, to check block #20 on DSK1:, enter:

   .DSKDDT DSK1:20 (RET)

2. DSKDDT loads the specified block into memory. Now type an E followed by a RETURN (the DSKDDT Exit command):

   E (RET)

   DSKDDT now writes the block back out to the disk, recomputing the CRC in the process.

3. Do this for all bad disk blocks on the disk.

4. Now, use REDALL on the disk again. With any luck, the disk is now healthy again. NOTE: This procedure does not ensure that your data is intact. (Fixing the CRC error may actually cause some data in the block to be lost; however, correcting the CRC does allow the system to read the block.) You may want to dump the restored block with the DUMP BLOCK command to see if you need to modify the data in it.

6.1.2 Getting Rid of Bad Disk Blocks

If using DSKDDT did not fix the hard errors on the disk, you must take more stringent measures and get rid of the bad blocks. To find out what files the bad blocks belong to, use DSKANA with the /L or /E options. DSKANA exits when it finds the first bad block, but at least it tells you which file contains that first bad block. For example, if block 12 is bad in the

file VARSET.BAS, the DSKANA display for that file might look something like this:

```
VARSET BAS  5       6       7       10      12 AM500 ERROR CODE 4
FOR DRIVE 1 BLOCK 12 (CYLINDER 0 HEAD 0 SECTOR 12)
```

Keep track of the files in which the bad blocks appear. You can use the DSKFIL command to check an individual file for a hard error.

Using DSKCPY to copy the damaged disk to another disk won't solve the problem-- DSKCPY exits and returns you to AMOS command level when it encounters a hard error.

Simply erasing the bad files will not help either. If you erase a file that contains a bad block and then run DSKANA again, DSKANA frees up the bad block so that the system can allocate it to another file.

The only solution is to erase the bad files and then to use the COPY command to copy the good files over to another disk.

1. Make sure that the disk you are copying to has no hard errors. (Use the REDALL program or the DSKANA program.)

2. Erase the bad files by using the ERASE command. (Be very careful not to run DSKANA again on the bad disk after this point, or the bad blocks will be freed again.)

3. Log into the System Operator's account and copy the good files on the disk over to the backup disk:

```
.LOG [1,2] (RET)
.COPY AMS2:=AMS1:[] (RET)
```

   The COPY command above copies all accounts on AMS1: over to AMS2:. If two files exist on AMS1: and AMS2: with the same name, extension, and account number, the command above deletes the file on AMS2: and replaces it with a copy of the corresponding AMS1: file. If you don't want COPY to replace duplicate files, use the /NODELETE switch. (For example: .COPY AMS2:=AMS1:[]/NODELETE) Because you are logged into the System Operator's account, the command above copies all files in all accounts on AMS1: over to the corresponding accounts on AMS2:. If the corresponding account does not exist on AMS2:, the COPY command creates it, transferring to it any password associated with the source account.

4. Now is the time to restore the files that you erased. Copy them over to the good disk from your most recent backup disk.

5. Back up the good disk.

6. Now that you have copied all of the good files over to the new disk, get rid of the bad blocks on the original disk by formatting the disk. For example:

```
.FMT500 DSK1: (RET)
BEGIN FORMATTING
EXIT
.
```

7. After you have formatted the disk, use the REDALL program to see if the original hard errors resulted from actual physical damage to the disk. If everything is OK, you can initialize the disk and begin to use it again.

If hard and soft errors are frequent occurrences on your system, you ought to take a look at the disks themselves. (For example, if you are using floppy disks, are you storing them correctly? Are they scratched or dusty?) You might also check your disk controller board and the physical device itself for maladjustments.

6.2 DSKANA File Errors

For the purposes of this discussion, we assume that you have already handled any soft or hard errors on the disk, and that the only problem with the disk at this point is in the linking of the disk blocks.

If DSKANA reports file errors (e.g., 5 file errors) or lists disk blocks under the message: [The following blocks were in a file but not marked in use], the block linking structure of the disk is in error. You must take immediate steps to recover the data on your disk.

Run DSKANA again, but this time use the /L switch so that you can see what disk blocks were assigned to each file or use the /E switch if you just want a list of the blocks and files where the errors occurred.

Look for the file error messages in the display (see Section 6.2.1, "DSKANA File Error Messages"). Their location in the display indicates which disk blocks are incorrectly linked, and this tells you which files are bad. For example, if part of the DSKANA display looks something like:

```
.DSKANA DSK3:/E (RET)
[Begin analysis of DSK3]

[20,1]
[30,4]
Block   731 - Block used in previous file in DSK3:COPYR.LST[30,4]
[30,5]
                .
                .
                .
```

then you know that a disk block used by the file COPYR.LST was also used by another file. Two files cannot share the same disk block.

If you are fortunate enough to have a very recent backup disk, and the files you have changed since that backup are okay, you can simply copy those changed files from the damaged disk over to the backup disk, which now becomes your original. (Of course, this assumes that the backup disk is all right. You might make it a practice always to run DSKANA before backing up a disk, so that you know that your backup disk is always good.) Now initialize the damaged disk; this clears the disk.

If you are not so lucky as to have a recent backup disk, you must do what you can to salvage the data on your original disk:

1. Make a disk backup so that you have a copy with which to work.

2. Now that you have found out what files are bad, use the ERASE command to erase those files from the disk. If you do not do so, the errors in the linking structure will propagate, and you will lose even more data.

3. After clearing the bad files from the disk, run DSKANA again to make sure that all problems have been cleaned up.

4. Once no more file errors show up, you must set about restoring the files you have erased. If you have old backup disks that contain good copies of the files that you have just erased, restore those files on your disk by copying them over from the backup disks.

Make a final backup copy of your newly restored disk.

The discussion above assumes that damage has been done only to your files; if the linking structure of the disk directories themselves is bad, you may have to copy off of the disk whatever files you are able to, and then initialize the disk and start over with a new account structure.

A systems expert may be able to reconstruct disk directories by using DSKDDT to actually change the binary data on the disk. (Directory entries are not stored in straight ASCII, but in a special packed format called RAD50; you will have to make the conversion yourself.) Using DSKDDT to reconstruct directories is very dangerous; do not try it except as a last resort, and be sure to make a backup first!

(Changed 30 April 1981)

6.2.1 DSKANA File Error Messages

Below is a list of the file error messages that DSKANA can display when you use the List (/L) or Errors only (/E) options. The location of the messages in the display tells you the disk location of the bad block link.

1. Block used in previous file

   The last disk block listed in the file where this message appears also exists in another file. Since two files cannot share a disk block, this message means that the system made an error in allocating disk space to the two files.

2. Block marked as bad

   A block marked as bad in the BADBLK.SYS file has been mistakenly allocated to a file.

3. This file has a bum block count

   The actual disk block count for the file where this message appears does not match the block count assumed by the file itself. The system made an error in allocating blocks to this file.

4. Illegal block link

   A link in the file where this message appears points to an invalid disk address (e.g., to a disk block that does not exist).

5. Block reserved for system use only

   A link in the file where this message appears points to a disk block that is reserved for system use. The system has mistakenly allocated a disk block that should not be allocated to a file.

6. Block creates endless loop in file

   The linking structure of this file is such that eventually the disk blocks point back to themselves. That is, block-A points to block-B which points back to block-A.

7. Device error on Devn:

   This block contains a hard error that the system could not recover from.

8. [unable to locate BITMAP for rewrite]

   DSKANA couldn't find the bitmap area in memory for the device being analyzed. This means that the bitmap in memory may be invalid.

9. BITMAP rewrite error code XXXXX

The bitmap could not be written back out to the disk. The number you see is the error code that indicates what the problem was. For a list of these error codes, see Chapter 6, "The File Service System," in the AMOS Monitor Calls Manual, (DWM-00100-42).

7.0 PACKING THE DISK

Besides explaining the concept of "packing the disk," this section explains why packing a disk is necessary, and gives information on the two commands you can use to pack a disk: DSKPAK and COPY (with the /PACK option).

NOTE: Before packing a disk, make sure the linking structure of the disk is intact. We recommend that you use DSKANA before you pack a disk.

The next few pages refer frequently to sequential and random files. If you are not familiar with these terms, see Chapter 5, "Introduction to Files," in Introduction to AMOS, (DWM-00100-65).

7.1 When to Pack a Disk

When AMOS writes file blocks out to the disk, it follows this allocation scheme:

1. If a block belongs to a sequential file, AMOS searches for the first free disk block on the disk beginning with the front of the disk, and writes the block there. Sequential file blocks thus tend to be located toward the front of the disk.

2. If AMOS is trying to write out a random file, it searches for the LAST area on the disk in which the file will fit, and writes the file there. Random files thus tend to be located toward the end of the disk.

This scheme leaves an area in the middle of the disk for new file blocks. When you delete a file from the disk, the disk blocks that made up that file are now free for use by other files. "Packing" the disk consolidates these free areas on the disk by sliding the random files down toward the end of the disk and sliding the sequential file blocks up toward the front of the disk. This allows the system to make efficient use of the free space on that disk.

You especially need to reduce fragmentation of free space on the disk if you make use of a number of random files. If you only use sequential files, you will not need to pack the disk very often because the system, as it allocates disk blocks for sequential files, fills in any "holes" left by deleted sequential files.

On the other hand, if you use large random files, you will probably want to pack the disk quite often, perhaps before every disk backup. It's particularly important to consolidate free space if you use random files because when it comes time to allocate space for a random file, the total number of free blocks on a disk doesn't matter-- it's the number of free blocks that appear in a contiguous group that counts. For example, it is quite possible to get a device full error when allocating a random file of 50 blocks, even though you have 200 blocks free on that disk-- the system has to find 50 contiguous disk blocks.

7.1.1 Displaying the Bitmap

If you want to have some idea of how much in need of packing your disk is, you can take a look at the bitmap of that disk. A bitmap is a map of your disk. That is, it tells the system what blocks on the disk are available and which are in use by a file. If you look at the bitmap, you see a matrix of 1s and 0s which represents the free and used blocks on that disk. Each block on the disk is represented by a one if that block is in use and a zero if it is free.

If all of the 1s are clustered together in large groups at the beginning and end of the disk, with only occasional 0s scattered among the groups, the data on the disk is efficiently allocated. If, however, the 1s and 0s seem to be randomly mixed on the disk, you should pack the disk. To see a display of your bitmap, enter DUMP BITMAP followed by the specification of the device whose bitmap you want to see. For example:

```
.DUMP BITMAP DSK0: (RET)
```

You now see on the screen the bitmap of the disk in device DSK0:. To freeze the display, type a Control-S; to resume the display, type a Control-Q. To interrupt the display, type a Control-C. At the end of the display, the DUMP command tells you how many blocks are available on the disk:

```
3411 free blocks
```

7.2 DSKPAK

DSKPAK packs all random files on the disk. That is, it moves all random files toward the end of the disk, consolidating the free area in the middle of the disk for new random files. It does this by sliding random files down to occupy the area left by deleted files. If there are no sequential file blocks in the random file area, DSKPAK causes all of the random files to form one contiguous area at the end of the disk.

Do NOT run DSKPAK while other users are accessing the specified disk. To run DSKPAK, enter DSKPAK and the specification of the device you want to pack:

```
.DSKPAK DSK1: (RET)
```

(Changed 30 April 1981)

When DSKPAK is finished, you see the AMOS prompt.

7.3 COPY (the /PACK Option)

You can use the COPY command to pack the data on your disk if you use the /PACK option.

NOTE: It is VERY important that no other job be allowed to access the disk while you are packing it. If fact, it is good practice never to pack a disk while other jobs are running on the system.

To pack all files on the disk, log into the System Operator's account, [1,2]. Then type: COPY/PACK, the specification of the device you want to pack, an equal sign, the specification of the same device, and a RETURN. For example:

```
.LOG [1,2] (RET)
.COPY/PACK DSK1:=DSK1:[] (RET)
```

The command above copies every file on DSK1: over to itself. This causes the system to reallocate blocks for all files on the disk, writing over areas left by deleted files. (This command does NOT pack the User File Directories or the Master File Directory of the disk.) To completely pack the disk, you might have to perform this command several times.

NOTE: To achieve maximum packing of the disk, use the COPY command from the System Operator's account to copy all files from one disk to another freshly initialized disk. (In this case, you don't need the /PACK option since you are not copying files over to themselves.)

8.0 DISK BACKUP

The most important procedure in your disk maintenance routine is disk backup. You will have noticed that the discussions of the disk diagnostic tests assume that you have recent backups of all of your disks. Backups are your only assurance that you can at least partially recover from disaster.

Make disk backup a regular procedure on your system. You should encourage all users to back up the files in their own accounts to either their own data disk or to a communal data disk. You should do regular backups of the System Disk and any important data disks.

8.1 The COPY Command

Before using COPY to back up files, you will probably want to use the SET command to set DSKERR:

```
.SET DSKERR (RET)
```

Setting this option tells the system to report any soft disk errors to your job.

Individual users may back up the files in their own accounts by using the COPY command. For example, to transfer copies of all files in account DSK0:[300,1] to the same account on DSK1:, enter:

```
.LOG DSK0:[300,1] RET
.COPY DSK1:= RET
```

Individual users may also back up accounts that are within the same project. For example, to back up all accounts in Project 100 from DSK0: to DSK1:, log into an account in Project 100 and specifiy wildcard PPNs in the COPY command:

```
.LOG DSK0:[100,0] RET
.COPY DSK1:[]=[100,*] RET
```

To back up all accounts from one device to another (for example, DSK0: to DSK1:), log into the System Operator's account:

```
.LOG [1,2] RET
.COPY DSK1:=[] RET
```

Because you are logged into the System Operator's account, the command above acts very differently than the usual COPY command:

1. The command above copies all files in all accounts on DSK0: over to DSK1: regardless of whether those accounts are in Project 1.)

2. Even though there is no wildcard PPN symbol on the left side of the equal sign, the command above copies all files in all accounts on DSK0: over to their corresponding accounts on DSK1: (not into the single account you are logged into, [1,2]).

3. If the destination account does not exist on DSK1:, the command above creates it. The command above transfers over to a new account any password associated with the corresponding source account files are being copied from.

To back up specific accounts, enter the correct account specifications. For example:

```
.LOG [1,2] RET
.COPY DSK1:=[200,*],[100,*] RET
```

8.2 The DSKCPY Command

IMPORTANT NOTE: As of AMOS Version 4.4, you may use DSKCPY on devices that run under control of the AM-410 hard disk controller, but only if the disk you are copying from and the disk you are copying to have both been certified with an AMOS 4.4 or later version of CRT410. If either disk has been certified with a pre-4.4 version of CRT410, you must use COPY to back up the disk.

Use DSKCPY to make a literal image of one disk onto another. You may use DSKCPY on any type of disk; however, you may not copy between disks of different types. (We say that two devices are of the same type if they use the same device driver program. All devices that use the same device driver appear in the DEVTBL line of the system initialization command file with the same three-character device code. For example, DSK1, DSK2, and DSK3 use the same device driver; AMS0, AMS1, AMS2, and AMS3 use the same device driver.) So, you can use DSKCPY to copy between any two devices that share the same device code (e.g., from AMS1: to AMS2; STD3: to STD1:, DSK3: to DSK2:, etc.)

Besides copying and verifying data from one disk to another, DSKCPY also optionally generates a hash total for the disk copied to. To generate a hash total, use the /H switch. (For example: .DSKCPY/H.) (For full information on DSKCPY, see the DSKCPY reference sheet in the AMOS System Commands Reference Manual, (DWM-00100-49).

IMPORTANT NOTES:

1. Never run DSKCPY while other jobs are accessing the disks you are copying between.

2. A common mistake in using DSKCPY is to accidentally reverse the input and output device specifications. This has the effect of copying your empty disk onto your original disk. To avoid this situation, if your disk device allows it, always write-protect the disk from which you are planning to copy.

3. Before you copy to a disk, make sure that the disk is empty or that it does not contain any data that you need. Make sure that the disk is in the proper format. If the disk already contains data (that is, it is not brand new), you do not need to reformat it. (However, remember that using DSKCPY writes over any data already on a disk.)

4. If you use DSKCPY to copy a disk that has soft errors, DSKCPY will display the proper error message when it encounters the soft error, and then will attempt to copy the bad block over to the output disk, recomputing the CRC in the process. Since it had trouble reading the bad block, the copy that DSKCPY makes of the block may contain garbled data.

   If you use DSKCPY to copy a disk that has one or more hard errors, DSKCPY aborts the copy when it encounters the first hard error, and returns you to AMOS command level.

If you suspect that your disk has hard or soft errors, follow the procedures outlined in Section 6.1, "Handling Hard and Soft Disk Errors."

5. You can use DSKCPY to copy both System Disks and data disks. (A System Disk is a disk the system can boot from-- that is, it contains elements of the operating system necessary for system operation. A data disk is any disk that is not a System Disk.)

We have divided the information below into instructions for backing up hard disk devices and floppy disk devices.

We have additionally divided those instructions into information on using DSKCPY on: 1. Multiple-unit devices; and 2. Two-unit System Devices.

If you are using DSKCPY on a device that has more than two logical units (for example, besides DSK0: and DSK1:, your system also has DSK2: and DSK3:), backup procedures are the same regardless of whether that device is a System Device as long as you avoid copying to the drive containing the System Disk.

However, if you are using DSKCPY on a System Device that contains only two logical units (DSK0: and DSK1:), using DSKCPY becomes more complicated because you MUST use DSK0: (the drive reserved for the System Disk) during the backup procedure. This can be tricky because intermediate steps in the backup may require that you write over the System Disk or replace that disk with a backup disk.

8.2.1 Important Note for Hawk Hard Disk Drive Users

DSKCPY uses a special fast copy mode for Hawk devices only. This mode gives a disk copy in approximately one-third the time of the traditional Hawk disk copy (about 6 minutes versus 18 minutes.) However, it does require that no other user be using the AM-500 controller while the disk copy is taking place. When you use DSKCPY on a Hawk device, you see:

```
%All other users will be suspended while HAWK copy is running
Hit return to continue or a control-C to abort:
```

If no other users are accessing disks that run under the AM-500, you may type a RETURN to continue; otherwise, type a Control-C to exit DSKCPY.

If it is not convenient for all users to stop running on the system, you may use the /O switch to tell DSKCPY to use the slower copy mode for the Hawk device. If you use the /O switch, you do not see the message above; other users may run on the system, but the disk copy will take about 13 minutes.

If you use the /H switch to generate a hash total for the copied to Hawk disk, the hash total will differ for the same disk, depending on whether you used the Hawk fast copy mode or the /O mode. The /O switch has no effect when DSKCPY is being used on a non-Hawk device.

As you read the instructions in the sections below, remember that if you are using DSKCPY on a Hawk device, you must decide whether to use the fast copy mode or the /O mode.

8.2.2 The Hard Disk Multiple-Unit Device

The instructions below assume that you are working with a disk in which at least one logical unit is an unremovable, permanently fixed disk. We assume that you are copying between disk cartridges or from the fixed disk to a cartridge. The instructions below also apply to using DSKCPY on a device that contains only two logical units (e.g., HWK0: and HWK1:) if that device is not a System Device, since you do not have to worry about writing over the System Disk.

If you are using a hard disk device that contains more than two logical units within one physical device (e.g., the Century Data Trident which can contain DSK0: through DSK18:), you will treat that single device as a multiple-unit system. That is, you can back up each logical unit separately by copying one logical unit to another. (If you have two of these kinds of devices you will probably want to back up by copying from one drive to another. Be advised, however, that backing up several hundred megabytes of data is a very slow process!)
To copy either System Disks or data disks:

1. Write-protect the disk you are copying. (For example, if you are going to make a backup copy of the disk in device DSK2:, write-protect DSK2:.) If possible (that is, if you can arrange to perform the DSKCPY when no other users are going to be accessing the disks), write-protect ALL disks except the one you are copying to.

2. If you need to change disk cartridges, do so. This may require cycling-down the disk drive, but you do NOT need to turn off the drives or the computer.

3. Enter:

```
.DSKCPY (RET)
```

4. DSKCPY now asks you which devices to copy between:

```
Input Drive:
Output Drive:
```

For input drive, enter the specification of the device you are copying from. For output drive, enter the specification of the device you are copying to. For example, if you are copying from AMS1: to AMS3:, enter:

```
Input Drive: AMS1: (RET)
Output Drive: AMS3: (RET)
[Copying 616 blocks]
[Duplication and verification completed]
.
```

DSKCPY tells you how many disk blocks it is copying, and gives you progress reports on the status of the disk copy. When you see the AMOS prompt, DSKCPY is done.

8.2.3 The Hard Disk Two-unit System Device

If you are using DSKCPY on a System Device, and that device has only two logical units (DSK0: and DSK1:), using DSKCPY becomes a little more complicated because you cannot avoid using DSK0:. (If you are copying from a fixed disk to a disk cartridge, follow the procedures above for multiple-drive systems.)

To copy a data disk, you must first back up the System Disk (the fixed disk), copy the data disk down from a cartridge onto the System Disk, and then copy that back up to another cartridge. Then you must restore the System Disk from the backup. This procedure requires that you change cartridges several times, so to make sure that no mix-ups occur, carefully label all cartridges so that you can be sure which cartridge is the System Disk backup, which is the original data disk, and which is the data disk backup.

8.2.3.1 Backing Up the System Disk

1. Write-protect DSK0: and DSK1:.

2. Insert a backup disk cartridge to which you will copy the System Disk. (This may require cycling-down the disk drive, but do not turn it or the system off.)

3. Write-enable DSK1:.

4. Use the DSKCPY command to copy from the System Disk (DSK0:) to the cartridge:

```
.DSKCPY (RET)
Input Drive: DSK0: (RET)
Output Drive: DSK1: (RET)
[Copying 9696 blocks]
[Duplication and verification completed]
.
```

5. Remove the cartridge. Label and date it (e.g., SYSTEM DISK BACKUP 1/23/80).

8.2.3.2 Backing Up the Data Disk

A. PART I:

1. Write-protect DSK1: and DSK0:.
2. Place the data disk you want to copy into DSK1:.
3. Load DSKCPY into memory:

```
.LOAD DSK0:DSKCPY[1,4] RET
```

4. Before proceeding, make sure that you have backed up your System Disk. The next step writes over everything on DSK0:, and without a valid system backup you will not be able to get your system up and running again.
5. Write-enable DSK0:.
6. Copy the data disk onto DSK0:

```
.DSKCPY RET

Input Drive: DSK1: RET
Output Drive: DSK0: RET
[Copying 9696 blocks]
[Duplication and verification completed]
.
```

7. DO NOT TURN OFF THE SYSTEM OR THE DISK DRIVES.

B. PART II:

1. Write-protect DSK0: and DSK1:.
2. Place the data disk backup cartridge into DSK1:, but do NOT TURN OFF THE SYSTEM OR THE DRIVES. (Remember, the System Disk on DSK0: is temporarily gone.)
3. Write-enable DSK1:.
4. Copy DSK0: to DSK1:.

```
.DSKCPY RET
Input Drive: DSK0: RET
Output Drive: DSK1: RET
[Copying 9696 blocks]
[Duplication and verification completed]
.
```

5. Remove the backup cartridge. Date and label it (e.g., DATA BACKUP 4/20/81). DO NOT TURN OFF THE DRIVES OR THE SYSTEM.

8.2.3.3 Restoring the System Disk

After copying a data disk, you MUST restore the data originally on DSK0:.

1. Write-protect DSK0: and DSK1:.
2. Place the System Disk backup cartridge into DSK1:. (Do NOT turn off the drive or the system.)
3. Write-enable DSK0:.
4. Copy DSK1: to DSK0:.

```
.DSKCPY (RET)
Input Drive: DSK1: (RET)
Output Drive: DSK0: (RET)
[Copying 9696 blocks]
[Duplication and verification completed]
.
```

5. Remove the System Disk backup disk and store it in a safe place. The system is now up and ready for normal use.
6. Delete the DSKCPY program from memory:

```
.DEL DSKCPY (RET)
```

8.2.4 The Floppy Disk Multiple-Unit System

These instructions apply to all situations where you can avoid writing to DSK0: of the System Device--that is, you are using: 1. a floppy disk device that contain more than two logical units (e.g., you have AMS2: and AMS3: as well as AMS0: and AMS1:); 2. a floppy disk device that has only two logical units, but that device is not a System Device (since you don't have to worry about writing over the System Disk); and 3. a two-unit System Device to copy System Disks.

To copy either System Disks or data disks:

1. Do not turn off the computer or the disk drives. Do not remove the System Disk.
2. Write-protect the drives you will not be copying to.
3. Insert the original and backup disks in the drives. (The original disk is the disk you want to copy; the backup disk is the empty disk you will be writing to.)
4. Type DSKCPY followed by a RETURN.

5. DSKCPY now asks which disks you want to copy between:

```
Input Drive:
Output Drive:
```

For input drive, enter the specification of the device you are copying from. For output drive, enter the specification of the device you are copying to. For example, if you are copying from AMS1: to AMS2:, enter:

```
Input Drive: AMS1: (RET)
Output Drive: AMS0: (RET)
[Copying 616 blocks]
[Duplication and verification completed]
```

6. DSKCPY is now done. Write-enable the disks.

8.2.5 The Floppy Disk Two-unit System Device

The instructions below apply to a situation where you MUST use DSK0: of a two-unit System Device. (That is, you must copy a non-System Disk and you only have two drives with which to do it.)

To copy a System Disk, follow the instructions above for multiple-unit systems; leave the System Disk in DSK0:, and copy it to DSK1:.

To copy a data disk:

1. Do not turn off the system or the disk drives. Leave the System Disk in DSK0:.

2. Write-protect DSK1:.

3. Insert the input disk into DSK1:.

4. Use the LOAD command to load the DSKCPY program:

```
.LOAD DSK0:DSKCPY[1,4] (RET)
```

5. Remove the System Disk. Insert the backup disk into DSK0:.

6. Enter DSKCPY followed by a RETURN:

```
.DSKCPY (RET)
```

7. For input drive, enter the specification of the device you are copying from. For output drive, enter the specification of the device you are copying to. For example:

```
Input Drive: DSK1: (RET)
Output Drive: DSK0: (RET)
[Copying 616 blocks]
[Duplication and verification completed]
.
```

8. When DSKCPY has finished, remove the backup disk and reinsert the System Disk. Mount the System Disk.

9. Delete DSKCPY from your memory partition:

```
.DEL DSKCPY (RET)
```

Index

(Changed 30 April 1981)

(Changed 30 April 1981)

April 1981
Revision A01

DEFINING NON-SYSTEM DISK DEVICES

There are many times when you need to configure your system to access a disk device other than the System Disk. Examples of this would be adding an AM-500 and Hawk disk drive to an AM-400 Trident based system, or adding an AMS-format floppy to an STD-format system. In both of these cases, the procedure is essentially the same. In addition to defining the device on your system, it is often necessary to transfer data from one type of device to another, or to convert a System Disk from one device or format to another. This document will cover both the definition of new disk devices and the transfer of data.

1.0 DEFINING NEW DISK DEVICES

To add a new disk device to your system, follow the procedure outlined below. It is a good idea to create a copy of your SYSTEM.INI file and modify that copy, rather than to modify the SYSTEM.INI file itself. The modified copy can be tested using the MONTST program, without risking creating a version of your SYSTEM.INI file which will not properly boot the system. Then, when you are sure that your new system initialization command file works, rename it to SYSTEM.INI. Regardless of these precautions, you should have a backup copy of your System Disk, just in case your changes don't work.

1. Add the device driver program to those in the [1,6] area on DSK0:. Each device to be used on the system must have a device driver program in this area. The device driver program must have a unique three-character name and an extension of .DVR. After locating the driver program for the device you wish to use, it should be renamed to a three-character name. For instance, the driver for the AM-500 is distributed as HWK500.DVR on standard System Disks; before use it should be renamed to a three-character name such as HWK.DVR. If you are defining a floppy disk, you must use the FIXDVR program to create a new driver for your particular configuration of drive, format and controller. See the document titled Configuring Floppy Disk Drivers in the "System Operator's Information" section of the AMOS Software Update documentation packet for more information.

2. Using one of the system text editors, VUE or EDIT, make these changes to your SYSTEM.INI file:

   a. Add the new device name to your device table; that is, add the three-character device name defined in step #1 to the DEVTBL command line. This defines the new device name, and allows I/O to take place through the new device driver. Disk devices are

always sharable devices, and should therefore be defined before the / in the DEVTBL command line. If you cannot fit the new device names all on one line, you may use more than one DEVTBL command.

You must include both the device name and all valid unit numbers in the DEVTBL command line. Thus, if you are adding an AMS-format floppy driver to your system and wish to be able to reference both drives 0 and 1, you must add the device codes AMS0 and AMS1 to the DEVTBL command line.

b. To allow the system to write information on the new device, you must define a bitmap for the new device. A bitmap is the method by which the system allocates space on the device. To add a new bitmap, place a BITMAP command for the new device immediately after any other BITMAP commands. The BITMAP command requires that you specify the device code, the bitmap size, and the units (drives) to which this bitmap will be applied.

The device code is the three-letter code you defined in step #1.

The bitmap size is dependent on the particular device you are defining. If you are working with a floppy disk drive, the FIXDVR program will tell you the bitmap size. For other devices, see the documentation accompanying the particular device.

Each BITMAP command actually defines a buffer in system memory which is used for reading and writing the bitmap to and from the device. This bitmap buffer may be shared by multiple units of the same device, resulting in a saving of system memory. In exchange for this saving, however, I/O must be done more frequently to the device to update the buffer than if the bitmap buffers were not shared.

For example, to define an AMS device (which has a bitmap size of 39 words) and to share the buffer between units 0 and 1, add the command:

```
BITMAP AMS,39,0,1
```

If you do not wish to share the bitmap buffer, but wish to have separate buffers for each unit, add two lines which read:

```
BITMAP AMS,39,0
BITMAP AMS,39,1
```

c. You may wish to include the driver program in your system memory area for faster response while working with the new device. To do this, insert the command:

SYSTEM xxx:DVR[1,6]

into the SYSTEM.INI file just above the final SYSTEM command line. In almost all cases, this is not really necessary, as the system will fetch the driver from the disk each time it is needed if it is not already in memory, and will load it into the memory partition of the user who is requesting access to that device. However, including it in system memory will speed up the response time of the system. NOTE: Never include the driver for your System Device in system memory-- since that driver was MONGENed into your monitor, it is always in system memory anyway, and adding it via the SYSTEM command just takes up uneccessary space.

IMPORTANT NOTE: There are several situations where you must put a non-System Device driver into system memory:

i. If you are using the /S option of the BITMAP command to place bitmaps for non-System Devices in switchable memory, you must place the drivers for those devices in system memory.

ii. Some programs (notably BASIC, RUN, COMPIL, and AlphaVUE), do not follow standard memory module conventions, and therefore require that the device driver of any non-System Device be in system or user memory if you are going to access that device while using those programs. (You may place the driver into system memory using the SYSTEM command, or the individual user may load it into his own memory partition via the LOAD command before invoking one of the programs listed above.)

3. After performing the preceding steps, boot up the system using the MONTST command:

.MONTST SYSTEM,TEST.INI (RET)

(Remember that to use the MONTST command your System Disk (DSK0:) must be on the fixed disk of the System Device, and your job must be operating in the first memory partition on the system (Bank Zero for bank switching systems).)

You should now be able to access the new devices for both reading and writing. Once you are satisfied with the operation of the device and your changes to your SYSTEM.INI file, rename yur temporary initialization command file to SYSTEM.INI and reset the system. You are now up and running.

2.0 TRANSFERRING DATA TO AND FROM THE DEVICE

Once a device has been defined on the system using the procedures described above, any of the standard system utilities may access it. If you are starting with a new device with no data on it, use the correct formatting program to write formatting information onto the device. You may then use the SYSACT program to initialize the device and to create accounts on that device. You may then use the COPY command to to copy data onto the new device.

If you wish to read existing data off of the new device, merely mount the device and use COPY to transfer the data.

3.0 BUILDING A SYSTEM ON A NEW DEVICE

Once you have defined the new device on your system, you may wish to create a System Disk for the device so that you can boot your system from the new device. To create a new System Disk, follow the procedure outlined below:

1. Format and initialize the device on which you want to build a system.

2. Copy your current System Disk over to the new device.

3. Log into the new device on account [1,4] and create a new monitor. Use the MONGEN program to do this. (See Generating System Monitors in the "System Operator's Information" section of the AMOS Software Update documentation packet.)

   When MONGEN asks you for the driver name, specify the driver for the device for which you are building the System Disk. After you run MONGEN, save the new monitor onto the new device, but don't reboot the system at this point.

4. Edit the SYSTEM.INI command file on the new System Disk and change the BITMAP commands for all DSK devices to match the bitmap size and configuration of the new device. Remove the old specifications for the device from the DEVTBL command. If you wish, you may add the current system device as a peripheral to the new System Disk by using a method similar to the one you used to create your current System Disk.

5. You may now use the MONTST program to test your new System Disk. If it successfully boots, you may wish to change your hardware configuration to boot off the new device; you may find information regarding this in the installation instructions accompanying the device.

May 1980
Revision A03

DISK DRIVERS AND FORMATS

1.0 INTRODUCTION

This document discusses the disk drivers and formats supported by Alpha Micro software and how to use them. One mark of the AMOS system's flexibility is that it allows you to customize your system software for your particular hardware configuration by selecting the disk driver programs and disk formats that you want to use.

2.0 DISK FORMATS

The format of a disk refers to the way that the data on a disk is structured. This discussion concerns itself with two main areas: the physical and the logical disk format.

The physical format means the number of bytes in each sector on the disk (which in turn dictates the number of sectors on each track). The logical format refers to the way in which AMOS reads the physical format, including blocking factor and file structuring.

We call the sector a physical record, because it reflects the physical organization of data on the disk. AMOS imposes a logical organization on the disk (regardless of the physical attributes of the device); we call these logical units of data disk blocks. Except in the special case of devices that use the IMG device driver, disk blocks are always 512 bytes long. For hard disk devices currently supported by Alpha Micro, the size of the disk block is the same as the physical record size (512 bytes); floppy disk devices sometimes use a physical record that is smaller than a disk block.

The following paragraphs discuss the disk formats that Alpha Micro currently supports:

1. Standard (STD) Floppy Disk Format -- This format comes in both single- and double-density, and single- and double-sided versions. All are designed to use IBM-compatible diskette formatting. Single-density diskettes are formatted with 128 bytes per sector and 26 sectors per track. Double-density diskettes are formatted with 256 bytes per sector and 26 sectors per track. Double-sided diskettes for both densities merely have twice the number of tracks per diskette (154 instead of 77). The format for both densities is 512 bytes per block, with single-density blocked at 4 sectors per disk block, and double-density blocked at 2. Both densities have

sectors mapped out to read every 5th sector to improve rotational latency time, thereby allowing one 512-byte disk block to be read within one revolution of the disk. The AMOS file system is fully supported on this format. Single-density, single-sided diskettes are supported on both the AM-200 and AM-210 floppy disk controllers; double-density and double-sided diskettes are supported on the AM-210 only.

2. AMS Floppy Disk Format -- This format also comes in both single- and double-density, and single- and double-sided versions. All are formatted in 512-byte sectors. Single-density has 8 sectors per track; double-density has 16. The physical records are interleaved on a 3-to-1 basis to improve latency for sequential accesses. Double-sided diskettes merely have twice the number of tracks per diskette (154 instead of 77). The logical format is the same as the physical format, with each disk block being one 512-byte physical record. The AMOS file system is fully supported on this format. Single-density, single-sided AMS format diskettes are supported on the AM-200 controller; only double-density diskettes (both single- and double-sided) are supported on the AM-210.

3. Image (IMG) Floppy Disk Format -- The physical format is the same as standard format, with 128- or 256-byte sectors and 26 sectors per track. The logical format is the same as the physical format. Image format is supplied to allow the programmer to read a non-AMOS diskette (such as one created on an IBM computer) by physical record number, and to retrieve the data under the programmer's own file structure. The AMOS file system will operate under this format, but is extremely inefficient due to the small disk block size (128 bytes) and the fact that one full revolution must occur between sequential reads. Image format for single-density, single-sided diskettes is supported by the AM-200 controller; both single- and double-density, single- and double-sided diskettes are supported by the AM-210.

4. Trident Hard Disk Format -- The Century Data Trident drives have their own special format. All drives use a 512-byte physical record which is also their disk block size. The T-25, T-50, and T-200 drives all have 22 sectors per track; the T-80 and T-300 have 32. The T-25 has 408 tracks per surface; all other Trident hard disks have 815.

5. Hawk Hard Disk Format -- The CDC Hawk drive has its own special format which gives a physical record size of 512 bytes, 812 tracks, and 12 records per track.

6. Phoenix Hard Disk Format -- The CDC Phoenix drive has its own special format which gives a physical record size of 512 bytes. Each surface has 808 tracks, with 36 records per track. (Actually, each surface has 822 tracks; fifteen of those tracks are spare or "alternate" tracks that are used in case bad tracks are detected on the surface.)

3.0 HARDWARE DISK DEVICES

The AMOS system supports several different disk devices. Each device has its own peculiar traits, and must be understood for proper system configuration. The following paragraphs describe the various hardware devices currently supported:

1. Persci Floppy Disk -- Runs under the control of the AM-200 floppy disk controller. The Persci supports all single-sided, single-density floppy disk formats described above. The Persci is capable of formatting diskettes.

2. Wangco Floppy Disk -- Runs under the control of the AM-200 or AM-210 floppy disk controller. With the AM-200, the Wangco supports all single-sided, single-density floppy formats described above. With the AM-210, it supports all single-sided, single- and double-density formats, except for single-density AMS format. The Wangco is capable of formatting diskettes.

3. CDC Floppy Disk -- Runs under the control of the AM-210 floppy disk controller. It supports all single- and double-sided, single- and double-density formats, except for single-density AMS format. The CDC is capable of formatting diskettes.

4. Icom Floppy Disk -- This is the original system offered with the AM-100 computer, but it is no longer in extensive use because of the limitations it imposes. The Icom disk runs under control of its own controller board, and does not have the capability of formatting diskettes. The Icom supports only single-density, single-sided STD and IMG formats.

5. Trident Hard Disk Subsystem -- The Century Data Trident is a special, large disk-pack subsystem that runs under control of the AM-400 interface board and the Century Data 1150-A formatter unit. The only format supported on this device is the Trident format. The unit will reformat disk packs (one surface at a time). The T-25, T-50, and T-80 Tridents have 5 surfaces; the T-200 and T-300 have 19 surfaces each. (The T-200 and T-300 are formatted as nineteen individual disks; the T-25, T-50, and T-80 are formatted as five individual disks.)

6. Hawk Hard Disk Subsystem -- The CDC Hawk is a special disk-pack subsystem that runs under control of the AM-500 Hard disk Controller board. The only format supported is Hawk format. The AM-500 is capable of formatting disks.

7. Phoenix Hard Disk Subsystem -- The CDC Phoenix is a special disk subsystem that runs under control of the AM-410 Hard disk Controller board. The only format supported is Phoenix format. The AM-410 is capable of formatting disks.

3.1 DISK DRIVER PROGRAMS

A driver is a program that links the generalized disk service routines of the monitor with the physical disk device. You can generate your own version of the monitor (via the MONGEN program) that incorporates a specific disk driver into the monitor for use as the System Device (device DSK:). The disk drivers reside in account DSK0:[1,6], and can be called in as needed if you have more than one type of device on your system, or if you wish to operate with several different types of formats on one device. In other words, your System Device may be a Persci in single-density standard format, but you may use a separate driver (called AMS, for instance), to read an AMS-format diskette. In this instance you would generate a single-density standard format driver to use with MONGEN when you create the monitor. The AMS driver exists in account [1,6] of the system disk, and is available for use when you want to read AMS-format diskettes.

Drivers are supplied to you on your System Disk in the [1,6] account; they have a six-character name that represents the device type they handle. Below are listed the disk drivers that are currently available, along with the devices they support:

```
200DVR - AM-200 with Persci or Wangco
210DVR - AM-210 with Wangco or CDC
ICMDVR - Icom floppy disk
TRIT25 - AM-400 with Trident T-25
TRIT50 - AM-400 with Trident T-50
TRIT80 - AM-400 with Trident T-80
TRI300 - AM-400 with Trident T-300
HWK500 - AM-500 with CDC Hawk
SMD410 - AM-410 with CDC Phoenix
```

The drivers above all have the .DVR extension. Since all devices must be identified by a three-character device name, you will have to rename or copy over the particular drivers to some chosen three-character device code; this code will also have to be added to your SYSTEM.INI file in both the DEVTBL and BITMAP command lines. You may call these drivers any three-character name you choose, except for the System Disk defined when you use MONGEN to generate a new monitor, which will always have the device code of DSK.

The floppy drivers must be configured by the FIXDVR program prior to use. See the document Configuring Floppy Disk Drivers in the "System Operator's Information" section of the AMOS Software Update documentation packet for information on configuring floppy drivers.

3.2 FORMATTING PROGRAMS

Formatting programs are provided for those devices that support formatting. The following is a list of the programs provided:

FMT200 - All formats supported by the AM-200
FMT210 - All formats supported by the AM-210
FMT400 - All formats supported by the AM-400
FMT500 - Hawk format for CDC Hawk disks

(NOTE: Use the CRT410 program to format and initialize disks that run under control of the AM-410.)

3.3 MONGEN

The MONGEN program is used to generate a new system monitor by overlaying the disk driver area in an existing monitor with a different disk driver from the list above. (See Generating System Monitors in the AMOS Software Update documentation packet.) You do not need to rename the disk driver used in the MONGEN procedure to a three-character device code; you may reference it directly by its six-character name (e.g. TRIT80 for a Trident T-80). The MONGEN procedure automatically renames this device so that it can be referenced as DSK in the new system. Note that you must configure floppy drivers with FIXDVR prior to the MONGEN procedure, however.

July 1979
Revision A01

GENERATING SYSTEM MONITORS

1.0 INTRODUCTION

The MONGEN program allows you to generate system monitors for any disk hardware by inserting the necessary disk driver into an existing monitor.

To build a new monitor, you need an existing monitor (AMOS version 3.1 or later), and the disk driver for the specific device that you are going to use as the System Disk. The monitor that you will normally use is the file SYSTEM.MON located in account [1,4] of the System Disk. The disk driver to be used will be one of the drivers in DSK0:[1,6]. MONGEN will insert the driver into the monitor (overlaying the old driver), and leave the new monitor in memory. You may then test the new monitor directly from memory (via the MONTST command), or you may save it onto a disk (via the SAVE command).

2.0 USING THE MONGEN PROGRAM

Type MONGEN followed by a RETURN:

```
.MONGEN (RET)
```

The MONGEN program responds with:

```
INPUT MONITOR NAME:
```

enter the file specification of the monitor program you are going to use as the foundation of your new monitor. You may specify the default system monitor DSK0:SYSTEM.MON[1,4] by entering just a carriage return. If you enter a file specification, the default device and account is DSK0:[1,4]; if you want to use a monitor not in that account, include device and account specifications.

MONGEN locates the specified monitor and loads it into your memory partition. Be sure that you have enough room to accomodate the monitor and disk drivers, as well as the MONGEN program itself, and also enough room for a couple of disk buffers. Typically this requires about 16K bytes of user memory.

Now MONGEN asks for the specification of the disk driver you want to insert into the monitor:

```
NEW DISK DRIVER NAME:
```

(Changed 1 July 1979)

Enter the file specification of the disk driver program you want to use. You may NOT enter just a carriage return. The default device is DSK0:, the default account is [1,6], and the default extension is .DVR.

MONGEN locates the disk driver program and loads it into memory above the previously loaded monitor. MONGEN now inserts the driver into the proper area of the monitor (thus overlaying the original disk driver). The new monitor is now complete in your memory partition.

Now MONGEN asks for a name to be given to the new monitor:

NEW MONITOR NAME:

Enter a one- to six-character name (the default extension is .MON); this is now the name of the new monitor in memory. MONGEN now exits, leaving the new monitor as a module in your memory partition. You can test the new monitor by using the MONTST program, or you can save the monitor as a disk file by using the SAVE command.

NOTE: MONGEN does not affect the currently running monitor either in memory or on the System Disk. Nor does MONGEN test the new monitor; it merely builds a new monitor as a module in your memory partition.

3.0 DISK DRIVER PROGRAMS

The hard disk drivers currently available are listed below. These programs are on your System Disk in account [1,6].

SMD410 - CDC Phoenix 90-megabyte hard disk (512-byte sectors).
HWK500 - CDC Hawk 10-megabyte hard-disk (512-byte sectors).
TRIT25 - Calcomp Trident 25-megabyte hard-disk (512-byte sectors).
TRIT50 - Calcomp Trident 50-megabyte hard-disk (512-byte sectors).
TRIT80 - Calcomp Trident 80-megabyte hard-disk (512-byte sectors).
TRI300 - Calcomp Trident 300-megabyte hard-disk (512-byte sectors).

In addition, the system supports a number of floppy disk drivers. With the advent of the AM-210 Floppy Disk Controller, the system now supports double-sided and double-density diskettes. Because the possible combinations of device type, floppy disk controller type, and disk format have greatly increased the number of possible floppy disk drivers, we now provide a program, FIXDVR, that you can use to configure the floppy disk drivers you need. See Configuring Floppy Disk Drivers in the "System Operator's Information" section of the AMOS Software Update documentation for information on using FIXDVR, and for a list of the possible floppy disk drivers you can create. (For information on the physical disk formats used by the various floppy disk drivers, see Section 2.0, "Disk Formats," in the document titled Disk Drivers and Formats in the "System Operator's Information" section of the AMOS Software Update documentation packet.)

USING THE MAGNETIC TAPE UTILITY PROGRAMS

October 1979

This document reflects AMOS versions 4.3 and later

ALPHA MICROSYSTEMS
17881 Sky Park North
Irvine, CA 92714

Table of Contents

1.0 INTRODUCTION

Alpha Micro has developed several magtape utility programs that support the AM-600 Magnetic Tape Transport Formatter Interface. These programs-- TAPE, REWIND, SET, and SKIP-- are all programs that you can invoke at AMOS command level by entering the proper command. For information on accessing the magnetic tape driver (MTU.DVR) directly from within your own assembly language programs, see the AMOS Monitor Calls Manual, (DWM-00100-42). For brief summaries of the tape utility programs, refer to the AMOS System Commands Reference Manual, (DWM-00100-49).

You will be most likely to use your magnetic tape transport to transfer files to the AMOS system from other computer systems. Magnetic tape also provides inexpensive backup media. The TAPE program works only with sequential disk files that contain fixed-length records. It does not support any random-access file format on the tape. TAPE can convert your data from ASCII to EBCDIC form and vice versa. See Section 5.0, "Transferring Data Between Disk Files and Tape," below, for more information on the kind of data you can transfer.

2.0 SETTING YOUR SYSTEM UP TO USE MAGNETIC TAPE UNITS

Make sure that you have defined the magnetic tape units as system devices in the DEVTBL command line of the system initialization command file, SYSTEM.INI. Magnetic tape units are not sharable, so place them after a slash on the DEVTBL command line. For example:

```
DEVTBL DSK1,STD0,STD1,MEM,RES,/MTU0,MTU1,MTU2,MTU3,PTP
```

In the example above, the devices MTU0, MTU1, MTU2, and MTU3 are the magnetic tape units. (The example above includes a paper tape punch, PTP, which also is not sharable.) The AM-600 supports up to eight tape units, numbered 0 through 7.

The program MTSTAT.SYS must be in system memory. (You must place it in system memory by entering a SYSTEM MTSTAT.SYS command line in the SYSTEM.INI.) The magnetic tape driver program uses MTSTAT.SYS to determine tape density and tape speed. If MTSTAT.SYS is not in system memory, when you try to perform a magnetic tape unit operation, you see:

```
MTSTAT.SYS NOT FOUND
```

Make sure that the magnetic tape driver, MTU.DVR, is in account DSK0:[1,6]. Once you have modified your SYSTEM.INI to include information on the magnetic tape units, you are ready to use the magnetic tape utility programs. Reboot the system with your new SYSTEM.INI.

2.1 Using System Commands to Access the Magnetic Tape Unit

Besides the magnetic tape utility programs discussed in this document, MTU.DVR allows you to use several AMOS commands to access the magnetic tape units. To look at files on magnetic tape, use the ASCDMP command. Enter:

.ASCDMP Devn: (RET)

where Devn: selects the magnetic tape unit you want to access. ASCDMP displays (in ASCII form) the tape block currently in position at the tape transport read heads. To display the next block, use ASCDMP again. NOTE: If you try to read past the end of the file, ASCDMP blows up; type a Control-C to exit ASCDMP.

You may use the TYPE command to display a magnetic tape file. Enter:

.TYPE Devn: (RET)

where Devn: selects the magnetic tape unit you want to access. TYPE displays the file currently in position at the transport read heads. To display the next file, use TYPE again.

You may use COPY to copy from a disk file to the magnetic tape. Enter:

.COPY Devn:=Filespec (RET)

where Devn: selects the magnetic tape unit you want to access and Filespec selects the disk file you want to copy to the tape. Make sure that Filespec is a valid AMOS file specification. You may not use COPY to copy data from the magnetic tape to a disk file.

2.2 Concepts

Before we describe the tape utility programs, it might be a good idea to define a few of the terms that appear in the following discussion:

1. Magnetic Tape Transport - A transport is the hardware that reads and writes data on a magnetic tape (also known as a tape unit). (The mechanisms within the transport that actually read and write the data are called heads.)

2. Magnetic Tape - The magnetic tape is the media on which the transport acts. Tapes are either 7-track or 9-track; the tape utility programs work correctly on either type. The utility programs assume that your tape is capable of recording at least 800 bits per inch. The standard magnetic tape length is 2,400 feet, but the tape utility programs function on any length of tape supported by your tape transport.

3. BPI - The BPI (bits-per-inch) value specifies the density of the data on the magnetic tape (either 1600 or 800 bits per inch). When

the system starts up, the magnetic tape driver assumes that the BPI you will be using is 1600. If this value is incorrect, you must use the SET BPI command to enter the correct BPI. You may record at either 800 or 1600 BPI, but when you read a tape you must read it at the BPI it was recorded at.

4. Density - When we talk about tape density, we are talking about the BPI a tape was recorded at. This figure gives the density of the data on the disk.

5. Load Point - When the magnetic tape is positioned so that the metallic film at the start of the tape is at the transport read heads, that tape is said to be at load point. Until the tape is at load point, you cannot read or write data on the tape.

6. Mounted - The magnetic tape is mounted when it is correctly threaded on the tape drive. See the documentation accompanying your tape transport for information on mounting tapes. (NOTE: Mounting a tape is not the same as mounting a disk. When you mount a disk, you must actually use the MOUNT command to mount it. If you use MOUNT on a magnetic tape unit, it does no harm, but accomplishes no purpose, either.)

3.0 SETTING TAPE DENSITY

When you use the tape utility programs to read or write data on a magnetic tape, the system must know the density of the data you want to read or write. When the system starts up, the default data density is 1600 BPI. If you use the TAPE command to transfer data from the tape to disk files, make sure that the current BPI value is the same as the density value under which the data was originally recorded. If you use the TAPE command to write data from the disk to the magnetic tape, make sure that the current BPI value is set to the density you want to have the data recorded at.

To set the tape density value, use the SET BPI command. At AMOS command level, enter:

```
.SET BPI Devn:NNNN (RET)
```

where Devn: selects the magnetic tape unit you want to affect. NNNN is the BPI value you want to use. For example:

```
.SET BPI MTU0:800 (RET)
```

(SET BPI sets a flag in the magnetic tape module, MTSTAT.SYS. The tape density does not get changed to the specified value until you actually read from or write to the magnetic tape unit. At that time, MTU.DVR reads the BPI value from MTSTAT.SYS and sends it to the transport.)

3.1 SET BPI Error Messages

If you give an invalid device specification or incorrect BPI value to the SET BPI command, you see:

> The format for the command is: SET BPI MTU*:XXXX
> Where * = a tape drive in the range 0 thru 7 and
> XXXX is either 800 or 1600.

Check your device specification. (You can see a list of the valid system devices, including the magnetic tape units, by typing DEVTBL followed by a RETURN.) The AM-600 supports up to eight tape drives (unit numbers 0-7). Make sure that you have specified a BPI value of either 800 or 1600.

4.0 REWINDING A TAPE

To rewind a magnetic tape, use the REWIND command. Enter:

```
.REWIND Devn: (RET)
```

where Devn: selects a magnetic tape unit. Make sure that no other user is accessing the specified tape unit.

This program returns a tape to load point. If the tape is already rewound, using REWIND does no harm.

Make sure that the tape you want to rewind is correctly mounted. Type REWIND followed by the specification of the unit whose tape you want to rewind. Type a RETURN. For example:

```
.REWIND MTU1: (RET)
```

You now see one of the following messages:

1. Tape is rewinding now
   This message is the most common and indicates that the tape is in the process of rewinding. The rewind will be finished in a moment.

2. Tape is already rewinding
   Someone else has already begun to rewind the tape. The rewind will be finished in a moment.

3. Tape cannot be rewound - it is at load point.
   The tape has already been rewound (or has never left load point). Check to see that you are specifying the correct tape unit.

REWIND returns you to AMOS command level while the tape is still rewinding.

4.1 REWIND Error Messages

If you use REWIND incorrectly, you see one of the following error messages:

?File specification error
The system didn't understand the format of your command line. For example, you typed REWIND followed by a RETURN, without a device number specified. Re-type the line.

?Cannot OPEN Devn: - device does not exist
The system didn't recognize your specification of the magnetic tape unit. Check your spelling. Type DEVTBL followed by a RETURN for a list of the devices (including magnetic tape units) defined on the system.

Tape cannot be rewound - it is at load point
REWIND cannot rewind the tape; it is already at load point. Check to see that you have specified the correct device.

5.0 TRANSFERRING DATA BETWEEN DISK FILES AND TAPE

The TAPE utility program copies data from disk files to magnetic tape and vice versa.

If you are reading from a magnetic tape created on another system, chances are that the data is incompatible with the AMOS system. For example, AlphaBASIC requires that all data records it handles end with a carriage return/line-feed pair; if these are not present, BASIC may not be able to read the tape data you have written into a file. The screen-oriented text editor, VUE, requires that each line end with a carriage return/line-feed pair, and that the line be less than 510 characters. TAPE can convert your data from ASCII to EBCDIC and vice versa; beyond that, any data conversion and manipulation required to make your data compatible with the AMOS system is left up to you.

5.1 How TAPE Writes Data to a Tape

Before we discuss the actual operation of the TAPE program, it is important to understand how TAPE actually copies data to the tape from a disk file and vice versa. Records on tape are organized sequentially; that is, records are written one after the other and are read back the same way. (TAPE does not support a random-access file format.) Records are grouped on the tape into blocks. (The exact number of records per tape block is set by you when you originally write the data to the tape; this is called the blocking factor.)

5.1.1 Organization of Data on the Tape - When it writes data to a tape, MTU.DVR magnetizes a short amount of tape at the end of every tape block. This magnetized interval is called an Inter-record Gap (IRG), and when the tape drive reads the data back again, the tape drive automatically recognizes an IRG as marking the end of a tape block. The hardware itself begins reading after the IRG and stops when it reaches the next one.

Because an IRG takes up room on the tape that might otherwise be used to store data, it is a good idea to make the tape blocks relatively long in comparision to the IRG. In this way, you can make more efficient use of the tape by minimizing the amount of wasted space on it.

When TAPE finishes copying the data from your disk file to the magnetic tape, it writes a special symbol to the tape called an EOF, or end-of-file marker. This marker designates the end of the file so that when you use TAPE to copy the data back to a disk file, TAPE knows when the end of the file has been reached.

The tape transport detects the end of the tape. When you use TAPE to write data to the tape, TAPE rewinds the tape one block when it reaches the end of the tape; then it writes an EOF to indicate the end of the file.

5.1.2 Reblocking Data - When you write data to a tape from disk, several of the disk file records form one tape block. The number of records per block is called the blocking factor. For example, if you write 25 records per tape block, the blocking factor is 25. Of course, the actual amount of data per tape block also depends upon the number of characters in each disk file record.

When you write data to a tape, the TAPE program asks you if you want to do reblocking. What this means is that you can choose the number of data records to write to a tape block. (This number does not have to be the number of data records per disk block that appear in the disk file.) TAPE is able to calculate the number of characters per tape block after it asks you the length of your data records and the blocking factor you want to use.

The ability to change the blocking factor of the data records when transferring data from disk to tape is an important factor in making efficient use of the magnetic tape.

If you do not want to do reblocking, TAPE assumes that your data records are 512 bytes long, and that you want to write one record per tape block. When you write data to a tape, make sure that you make note of the length of your data records and the blocking factor you use. Also remember to write down the number of tape blocks you are transferring. When you transfer the data back to a data file you will have to know all of this information.

5.2 Transferring Data from a Disk File to Tape

When you write data from a disk file to a tape, you may translate the data from ASCII to EBCDIC or vice versa. TAPE requires that the file be a sequential file with fixed-length records. Make sure that the current system BPI value for the unit you are using is the data density you want to use to record the data on the tape.

To write data to a tape from a disk file, enter:

.TAPE (RET)

Now TAPE begins to ask you a series of questions. You can exit TAPE and return to AMOS command level at any time by typing a Control-C in response to any of the questions below.

1. The screen clears and you see:

```
This is the magnetic tape program.
It can copy files from disk to tape or it can copy files
from tape to disk.
*
*
*
Which do you want to do?
1- Copy a file from disk to tape.
2- Copy a file from tape to disk.


Type the number of the option you wish.
Answer?
```

   Enter a 1, followed by a RETURN. (If you enter anything but a 1 or a 2, TAPE asks the question "Which do you want to do?" and then displays the example above again.)

   (If you change your mind about wanting to write to tape, type a Control-C to interrupt TAPE and return you to AMOS command level.)

2. Now the screen clears and TAPE displays the message:

```
Tape can do character code conversion.
What type of conversion do you want to do?
1 - None.
2 - Convert the ASCII file to an EBCDIC file.
3 - Convert the EBCDIC file to an ASCII file.


Type the number of the option you wish.
Answer?
```

   TAPE is able to translate your date from ASCII to EBCDIC form, and vice versa. Because you are transferring data from a disk file to

the tape, this question is asking if you want to translate the data in the file before you write it to tape.

Enter the number that selects the conversion (if any) that you want to perform. You must enter a 1, 2, or 3 (or a Control-C to exit to AMOS command level); any other answer causes TAPE to re-display the example above.

3. The screen clears and you see:

```
Type in the name of the disk file you want copied to tape.
Answer?
```

Enter a valid AMOS file specification. You may only enter the specification of a sequential file.

4. Now the screen clears and TAPE displays this message:

```
Which tape drive are you using?
(For example - MTU0: MTU1: MTU2: etc.)
Answer?
```

Give the specification of the magnetic tape unit you want to use (e.g., MTU5:). If you decide at this point that you do not want to continue with the TAPE operation, type a Control-C to exit to AMOS command level.

5. The screen clears and you see:

```
Tape can do reblocking of the disk file.
This means you can specify the length of the record
and the number of records in a tape block.


Do you want to do reblocking?
Answer yes or no -
```

Answer YES or NO, depending on whether or not you want to do reblocking. (See Section 4.1., "Reblocking," for a discussion on reblocking.) If you want to do reblocking, the screen clears and TAPE displays this message:

```
What is the size of the record?
(For example: 10, 20, 512 or any size you want)
Answer?
*
*
*
What is the number of records in a block?
Answer?
```

TAPE assumes that the sequential file you want to copy to tape has fixed-length records. Give the length of the record (in bytes,

including any record delimiters such as carriage returns and line-feeds.)

If you enter just a RETURN, TAPE assumes a record length of 512.

Now give the number of these records you want to fit into one tape block. This is called the blocking factor.

6. TAPE now clears the screen and tells you how many characters you are writing in a tape block. For example:

```
You are writing 2500 characters in a tape block.
```

If this number does not reflect what you really want to do, you may type a Control-C to exit from TAPE to AMOS command level. For example:

```
You are writing 0 characters in a tape block.
```

The message above indicates that you entered invalid data to the reblocking questions. Exit TAPE via a Control-C, and try again.

7. Next you see:

```
*
*
*
Is the tape loaded?
Type return if it is
```

When you are sure that the tape is physically mounted on the tape drive, and that the tape is at load point, type a RETURN. (If you do not want to start writing at the beginning of the tape, the tape does not have to be at load point.)

8. Now TAPE asks:

```
Is the tape drive on-line?
Type return if it is
```

Make sure that the tape drive is on-line. (Check with the documentation that accompanied your tape transport to see what button or switch to push to put the drive on-line.)

Now the TAPE program begins its data transfer. When the copy is finished, TAPE clears the screen and you see a message that tells you how many tape blocks were written and how many errors occurred. For example:

```
There were 100 tape blocks written or read.
There were 0 errors.
```

Your tape drive detects any parity or CRC errors and reports them back to the AM-600 controller.

9. Remember to make note of the size of the data records, the number of data records per tape block, and the total number of tape blocks you wrote. Also remember the BPI value you recorded the tape at. When you transfer the data back to a disk file, you will need this information.

10. If you do not rewind the tape, you can write another file beginning at the next position on the tape. (You can keep using TAPE without rewinding the tape until the tape is full.)

5.3 Transferring Data from Tape to Disk Files

When you copy data from tape to a disk file, TAPE requires that the file be a sequential file with fixed-length records. Make sure that the current BPI value for the drive you are using is the value originally used to record the data.

To write data to a disk file from a magnetic tape, enter:

```
.TAPE (RET)
```

Now TAPE begins to ask you a series of questions:

1. The screen clears and you see:

```
This is the magnetic tape program.
It can copy files from disk to tape or it can copy files
from tape to disk.
*
*
*
Which do you want to do?
1- Copy a file from disk to tape.
2- Copy a file from tape to disk.

Type the number of the option you wish.
Answer?
```

Enter a 2. (If you enter anything but a 1 or a 2, TAPE asks the question "Which do you want to do?" and then displays the example above again.)

(If you change your mind about wanting to write to a disk file, type a Control-C to interrupt TAPE and return to AMOS command level.)

2. Now the screen clears and TAPE displays the message:

```
Tape can do character code conversion.
What type of conversion do you want to do?
1 - None.
2 - Convert the ASCII file to an EBCDIC file.
3 - Convert the EBCDIC file to an ASCII file.

Type the number of the option you wish.
Answer?
```

TAPE is able to convert your tape data from EBCDIC to ASCII form, and vice versa. Because you are transferring data from tape to a disk file, this question is asking if you want to translate the data on the tape before you write it to disk. Enter the number that selects the conversion (if any) that you want to perform. (You must enter a 1, 2, or 3; any other answer causes TAPE to re-display the example above.) To interrupt TAPE and return to AMOS command level, type a Control-C.

3. The screen clears and you see:

```
Type in the name of the disk file the tape is to be
copied into.
Answer?
```

Enter a standard AMOS file specification (e.g., DSK0:MAIL.DAT[110,2]). (The default extension is .DAT. TAPE assumes the account and device you are currently logged into.)

4. The screen clears and you see:

```
Which tape drive are you using?
(For example - MTU0: MTU1: MTU2: etc.)
Answer?
```

Give the specification of the magnetic tape unit you want to use (e.g., MTU5:). If you decide at this point that you do not want to continue with the TAPE operation, type a Control-C to exit to AMOS command level.

5. TAPE now displays this message:

```
You must supply blocking information on the tape file.
This means you must say how many characters are in a
record and how many records are in a tape block.

What is the size of the record?
(For example: 10, 20, 512, or any size you want)
Answer?
```

Give the number of bytes in each data record. Include any record delimiter bytes, such as carriage returns and line-feeds.

NOTE: When writing data from a disk file to tape, blocking information is optional. However, when you write data from tape to a disk file, you MUST supply blocking information. TAPE has no way of knowing the blocking factor and record length you supplied when you originally wrote the data onto the tape. You must now supply that information to successfully retrieve your data from the tape.

After you answer the question above, TAPE asks:

```
*
*
*
What is the number of records in a block?
Answer?
```

Give the blocking factor you used when you originally wrote the data to the tape. (For example, if you wrote 100 data records per tape block, enter the number 100.)

You MUST enter an answer for the questions above on blocking information. If you enter just a RETURN, TAPE re-displays the questions.

6. Now you see a message that tells you how many characters TAPE has calculated to be in a tape block. For example:

```
You are reading 10000 characters in a tape block.
```

If this number is not correct (for example, it is 0), there was something wrong with your answers to the questions on blocking above. Type a Control-C to exit to AMOS command level and try again. After this message, TAPE asks you for the number of blocks you want to read from the tape. For example:

```
You are reading 2500 characters in a tape block.
*
*
*
How many tape blocks do you want to read?
Answer?
```

Enter the number of tape blocks you want to transfer to a disk file. TAPE will read no more than the number of blocks you specify, but under certain circumstances will read less. For example, if the end of the tape is reached before TAPE has read the specified number of blocks, it will stop. If an end-of-file marker is reached, TAPE stops reading.

NOTE: Some tapes contain a special header block at the front of the tape as the first file. This header file may be only one block long. If you use TAPE on such a tape, it reads this file first and stops, reporting that it read only one block. Use TAPE again without rewinding the tape, and you will read the first of your files on the tape.

(NOTE: See Section 6.0., "The SKIP Command," for information on using SKIP to skip over files and header files.)

7. Now TAPE asks:

```
*
*
*
Is the tape loaded?
Type return if it is
```

Type a return when you are sure that the tape is physically mounted on the tape drive reel. Make sure that it is at or past load point. Type a RETURN. (If you do not want to start reading at the front of the tape, it does not have to be at load point.)

8. Now you see:

```
Is the drive on-line?
Type return if it is
```

Type a RETURN when you are sure that the tape drive is on-line.

9. TAPE now transfers over the data from the tape to the disk file. It clears the screen and tells you how many tape blocks it read from the tape and how many errors occurred. For example:

```
There were 200 tape blocks written or read.
There were 0 errors
```

Your tape drive detects any parity or CRC errors and reports them back to the AM-600 controller.

10. If you do not rewind the tape, you can use TAPE again to copy the next file on the tape. In this way, you can read a tape that contains many files, even if those files consist of records of different sizes grouped with different blocking factors.

5.4 TAPE Error Messages

If you give TAPE invalid device specifications you see the standard system error messages. For example:

?Cannot READ Devn: - device does not exist

You gave an invalid specification to TAPE's question: Which tape drive are you using?. Make sure that the magnetic tape units are defined as devices in the system device table.

?Memory allocation failed

When you specified the number of characters per tape block, you gave too large a number. (For example, you specified a record size of 100,000.) Check the validity of the blocking information you gave to TAPE.

You are writing 0 characters in a tape block.

This message indicates that you made an error when answering TAPE's questions on data blocking. Type a Control-C to exit to AMOS command level and try using TAPE again.

?Cannot OPEN Filespec - file already exists

You tried to write data from the tape to a file, but a file with the same specification already exists. TAPE returns you to AMOS command level. Check your file specification to make sure that you entered the correct device and account specifications. Try using TAPE again, but enter a different filespec.

?Cannot READ Filespec - file not found

You tried to write data from a disk file to the tape, but the system cannot find the specified file. TAPE returns you to AMOS command level. Check your file specification to make sure that you entered the correct account and device specifications. Try using TAPE again, but enter a different filespec.

?Cannot READ Devn: - file not found

You are trying to write data from the magnetic tape to a disk file, but TAPE cannot read the tape. You usually see this message when you try to read from a blank tape or when the tape is not positioned at the start of a file.

?Cannot WRITE Devn: - device not ready

The tape drive is not on-line. Push the on-line switch or button on the tape transport.

?Cannot READ Devn: - device error

You see this message if the system BPI value is different than that at which the tape was recorded. (Use the SET BPI command if you need to change the system BPI value.) You can also see this message if there is a bad spot on the tape.

6.0 THE SKIP COMMAND

The SKIP command tells MTU.DVR to skip to the next end-of-file marker on the tape. Use SKIP to skip over initial header files at the front of the tape and to skip over files on the tape you don't want to read.

IMPORTANT NOTE: Make sure that no other user is accessing the tape drive you are using when you use SKIP.

To skip to the beginning of the next file, enter:

```
.SKIP Devn: (RET)
```

where Devn: selects the magnetic tape unit you want to use. For example:

```
.SKIP MTU4: (RET)
```

If the tape is currently positioned at the front of a file, the SKIP command causes the tape drive to skip over the entire file to the beginning of the next one. If the tape is currently positioned in the middle of a file, the SKIP command causes the tape drive to move to the beginning of the next file.

When SKIP finishes, it returns you to AMOS command level.

6.1 SKIP Error Messages

If you supply an invalid device specification, you see:

```
?File specification error
```

Make sure that the magnetic tape unit you specified is a valid system device. (Type DEVTBL followed by a RETURN to see a list of all system devices, including the magnetic tape units.)

Index

April 1981

THE MAGNETIC TAPE FILE BACKUP PROGRAMS

1.0 INTRODUCTION

The Alpha Micro magnetic tape backup system consists of three programs: FILTAP, the program to transfer files from disk to tape; TAPFIL, the program to restore files to disk from tape; and TAPDIR, the program to list the contents of a magnetic tape. These three programs when used with the AM-600 Magnetic Tape subsystem allow you to easily and rapidly back up and restore both sequential and random files. The programs have full wildcarding capability and allow you to back up multiple disk surfaces on a single tape. The software also allows you to split a single backup across multiple reels of tape.

NOTE: To use these programs, you must use Version 4.5 or later of the magnetic tape driver, MTU.DVR[1,6], supplied on the disk with the programs.

These programs store and read data on the magnetic tape in a special variable-length record format developed by Alpha Micro. This format was optimized for the characteristics of the Alpha Micro computer and its magnetic tape subsystem; it was not intended that this format be used for data interchange with other, non-Alpha Micro computers. If you wish to transfer data to other computers using the magnetic tape subsystem, you should use the TAPE program; see Using the Magnetic Tape Utility Programs in the AMOS Software Update Documentation Packet for more details on that program.

IMPORTANT NOTE: It is important to remember that this set of programs was designed as a mechanism for backing up disk files. That means that FILTAP writes files to the tape along with their full device and account specifications. (It also writes the date and time of backup.) Therefore, when using TAPFIL to read a tape, if you want to access a specific file you MUST specify the disk and account from which the file was backed up, as well as the magnetic tape drive containing the tape you want to access. For instance, if you stored a file on tape from DSK2:, you must specify the device specification "DSK2:" when you restore the file from the tape. We give specific examples in the sections below.

1.1 Wildcarding Features

All three programs have been designed to function as much as possible like their disk-oriented counterparts (COPY and DIR) to make the magnetic tape software as easy to use as possible.

FILTAP, TAPFIL, and DIRTAP use wildcard symbols and specification defaults in the same way that COPY and DIR do. This is because, like COPY and DIR, FILTAP, TAPFIL, and DIRTAP are "wildcard commands."

Wildcard commands differ from other AMOS commands in that besides accepting the standard AMOS file specification:

Devn:filename.extension[p,pn]{/switches}

they also accept a variety of wildcard specifications. A wildcard file specification allows you to select multiple files with only one file specification. These file specifications can contain the special wildcard symbols *, ?, [], and ALL:. For example, to specify all files in all accounts of DSK0: that have a .BAS extension, wildcard commands permit the file specification:

DSK0:*.BAS[]

(In the example above, the symbol * stands for all possible filenames.) In addition, the wildcard commands allow you to set account, device, and switch specification defaults. For example, the wildcard command line:

.FILTAP [100,2]*.BAS,*.TXT[117,6],*.LST RET

sets the default account specification to [100,2]. The command line thus selects all .BAS and .LST files in account [100,2], and all .TXT files in account [117,6]. (Notice that the account specification setting the default occurs before the filename and extension rather than after it, as is the case with a standard AMOS file specification.)

1.1.1 Switches and Wildcard Commands - "Switches" are option requests. Each switch must begin with a slash, /. Remember that wildcard commands recognize two types of switches: "file" switches and "operation" switches. An operation switch applies to all of the file specifications on an entire command line no matter where it appears on that command line.

A file switch may apply to only specific file specifications, depending on where it appears on the command line. If a file switch appears at the end of a file specification, it applies only to the files selected by that specification. For example:

.FILTAP *.BAS,*.TXT/QUERY,*.LST RET

in the command line above, the /QUERY switch (a file switch) applies only to those files selected by the *.TXT file specification. If a file switch appears before a file specification, it becomes the default switch, and applies to all of the files selected by the following file specifications unless it is overridden for a particular file specification by another switch, or until a new default is set. For example:

.FILTAP *.BAS,/QUERY*.TXT,*.LST,*.MAC/NOQUERY,*.PRG RET

the /QUERY switch affects all files selected by the specifications *.TXT, *.LST, and *.PRG, but not the files selected by the specification *.MAC.

Because wildcard file specifications are extremely powerful, they can have very widely ranging effects. If you are not familiar with the way in which wildcard commands work, be sure to read Chapter 9 of the AMOS User's Guide, (DWM-00100-35) before attempting to use wildcard specifications.

2.0 WRITING DISK FILES TO TAPE - THE FILTAP PROGRAM

The FILTAP program writes disk files to the magnetic tape. This program accepts a standard wildcard file specification which specifies the files to back up.

The FILTAP program operates in two modes: /APPEND and /NOAPPEND mode. In /APPEND mode, FILTAP searches for the last file on the tape and starts writing the new files immediately after any data already on the tape. In /NOAPPEND mode, FILTAP does not look to see if any data is already on the tape, but just starts writing files at the beginning of the tape. The default mode of operation is /APPEND. Of course, if the tape is blank (i.e., a new tape), you should specify the /NOAPPEND mode.

Call the FILTAP program by giving a wildcard specification for the files you wish to back up:

.FILTAP{/switches} Filespec1{/switches}{,Filespec2{/switches}...} RET

The Filespec default is *.* and the account and device you are logged into. The default switches are /NOQUERY/APPEND.

The FILTAP program now asks you for the device code and unit number of the magnetic tape drive you want to write to:

Enter tape unit number:

(FILTAP assumes a device code of MTU.)

FILTAP now writes the files to tape, listing each file as it transfers it. (NOTE: Because FILTAP also writes the date and time of the backup to the tape, you should use the system commands DATE and TIME to set the system date and time before you use FILTAP.)

2.1 Example

For example, to back up all files from disk device DSK0: to the magnetic tape drive MTU0: (starting at the beginning of the tape), use the following command:

```
.FILTAP DSK0:[]/NOAPPEND (RET)
Enter tape unit number: 0 (RET)
AMSORT.SYS[1,4] to MTU0:AMSORT.SYS[1,4]
                .
                .
                .
Total of 1282 files transferred
.
```

2.2 Writing to Multiple Tapes

If all of the files you specified will not fit on one tape, FILTAP displays the following message:

```
%Tape is full, please mount another tape then type RETURN to
%  continue, or type control-C to abort copy
```

If you wish to continue backing up files on another reel of tape, wait for the current tape to finish rewinding, mount a new reel of tape, then type RETURN on your terminal. The backup will continue on the new reel. If you wish to abort the backup, type a Control-C.

2.3 FILTAP Switches

FILTAP provides the switches below:

/QUERY	or /Q	Ask user for confirmation before copying files (file switch).
/NOQUERY	or /NOQ	Don't ask for confirmation (default, file switch).
/APPEND	or /A	Write files to tape at the end of existing files (default, operation switch).
/NOAPPEND	or /NOA	Write files at beginning of tape (operation switch).

2.4 Error Messages

You may see the following error messages when using the FILTAP program:

?Cannot find DSK0:SCNWLD.SYS[1,4] or MEM:SCNWLD.SYS

The FILTAP program needs this file to be able to process wildcard symbols in your file specification. This message can indicate that SCNWLD.SYS does not exist, or that you do not have enough memory to load the file into your partition.

?Cannot READ Devn - device does not exist
?Cannot READ Devn - device is not mounted

You tried to copy to or from a device that is not listed in the DEVTBL command in your SYSTEM.INI, does not have a driver in area [1,6] of the System Disk, is not file-structured, or is not mounted. ("Devn:" is the device you specified.)

%No file-oriented device corresponding to Devn: is mounted

You specified a device, but left off the unit number. FILTAP cannot find a logical unit that matches your specification. Try mounting the device.

%Tape is full, please mount another tape then type RETURN to
% continue, or type control-C to abort copy

There is no more room on the current reel of tape. Mount another reel and type RETURN to continue the backup process, or type a control-C to abort the backup procedure.

3.0 RESTORING DISK FILES FROM TAPE - THE TAPFIL PROGRAM

Use the TAPFIL program to transfer files back to the disk from tape. The files must have been written to the tape via FILTAP. TAPFIL provides full wildcarding, allowing easy selection of the files to be restored, as well as automatic renaming facilities.

To use TAPFIL, enter the TAPFIL command followed by an output specification, an input specification, and any optional switches:

.TAPFIL{/switches} outspec = {inspec1{/switches}{,inspec2{/switches}...}}RET

The output specification defaults to the input specification. The input specification defaults to *.* and the device and account you are logged into. The default switches are /NOQUERY/DELETE.

TAPFIL now asks you for the device code and unit number specifying the tape drive you wish to read from:

Enter tape unit number:

TAPFIL assumes a device code of MTU.

TAPFIL then rewinds the tape and starts searching for the specified files. As TAPFIL finds them on the tape, it transfers the files to the disk and accounts you have specified.

The output specification you supply to TAPFIL is the specification of the file(s) you wish to create. TAPFIL provides full wildcarding. Just as with the COPY command, you may not copy files from one account to another unless: 1) the account you are copying from is in the same Project as the account to which you are copying; or, 2) you are logged into the disk account into which you are copying the files; or, 3) you are logged into the System

Operator's account, [1,2]. (As with the COPY command, logging into account [1,2] gives you certain privileges. The default account specification of the Outfilespec when you are logged into [1,2] is the wildcard account, []. Also, if you are logged into [1,2], TAPFIL will create the account you are copying to if it does not exist.)

The input specification is a list of the files you wish to copy from the tape. The input specification must give the exact specification of the file you wish to copy, including device and account of the file as it is stored on the tape.

3.1 Example

For example, assume you are logged into DSK0:[140,1] and you wish to copy a file from tape that was backed up from your own account (DSK0:[140,1]). Enter:

```
.TAPFIL = FILE.DAT (RET)
Enter tape unit number: 0 (RET)
MTU0:FILE.DAT to FILE.DAT
Total of 1 file transferred
```

Note that in the example above, the output specification defaulted to the input specification, and the input device and account defaulted to the device and account you are currently logged into.

If you want to copy a file from tape that was backed up from another device and account (DSK2:[1,4] for example), you would enter the following command:

```
.TAPFIL = DSK2:TEST.BAS[1,4] (RET)
Enter tape unit number: 0 (RET)
MTU0:DSK2:TEST.BAS[1,4] to TEST.BAS
Total of 1 file transferred
```

If you want to restore all the files stored on a tape to their original device and account, you would log into [1,2] and enter the following command:

```
.TAPFIL = ALL:[] (RET)
```

If you want to return all the files stored on a tape back to their accounts of origin in Project 110 of DSK0:, you would log into an account in Project 110 and enter the following:

```
.TAPFIL DSK0:[]=DSK3:[110,*] (RET)
```

If you include a filename and/or extension in your output specification, you can rename the copies of the files you are writing to disk. For example:

```
.TAPFIL DSK3:[]*.OLD = DSK1:[300,1?]*.MAC (RET)
```

copies all .MAC files from the tape backed up from accounts [300,1?] on DSK1: to the same accounts on DSK3:, and renames the file extensions from .MAC to .OLD.

3.2 Restoring from Multiple Tapes

If you are restoring files from multiple tapes created by the FILTAP program, you must enter separate TAPFIL commands for each tape.

3.3 TAPFIL Switches

TAPFIL provides the following switches:

/QUERY	or /Q	Ask user for confirmation before copying files (file switch).
/NOQUERY	or /NOQ	Don't ask for confirmation (default, file switch).
/DELETE	or /D	Copy over to an existing file, thereby deleting it (default, file switch).
/NODELETE	or /NOD	Don't copy over to any existing files (file switch).

3.4 Error Messages

You may see the following error messages when using the TAPFIL program:

?Cannot find DSK0:SCNWLD.SYS[1,4] or MEM:SCNWLD.SYS

The TAPFIL program needs this file to be able to process wildcard symbols in your file specification. This message can indicate that SCNWLD.SYS does not exist, or that you do not have enough memory to load the file into your partition.

?Cannot READ Devn - device does not exist
?Cannot READ Devn - device is not mounted

You tried to copy to or from a device that is not listed in the DEVTBL command in your SYSTEM.INI, does not have a driver in area [1,6] of the System Disk, is not file-structured, or is not mounted. ("Devn:" is the device you specified.)

%No file-oriented device corresponding to Devn: is mounted

You specified a device, but left off the unit number. TAPFIL cannot find a logical unit that matches your specification. Try mounting the device.

?Tape is not file structured
The tape you are trying to read was not written by the FILTAP program. The TAPFIL program can only read tapes written by FILTAP. Check to make sure you have mounted the correct reel of tape.

?Missing output specification
You omitted the equal sign in your TAPFIL command line; TAPFIL couldn't tell which information was your input specification and which was your output specification.

?More than one output specification
You may not supply more than one output specification.

?Files may not be transferred to RES:
You may only add programs to system memory by using the SYSTEM command within your system initialization command file, SYSTEM.INI.

%Not copied - Destination file already exists
You tried to copy to an existing file while the /NODELETE option was in effect.

?You are not logged in under [1,2], can't create [p,pn]
You cannot copy from an account to a nonexistent account unless you are logged in under [1,2]. If you copy to a nonexistent account while logged under [1,2], TAPFIL will create the account.

?Output MFD is full
The Master File Directory only has room for 64 entries. The transfer in progress would have created a new account, but there is no room in the MFD.

%Bypassing BADBLK.SYS[1,2]
% BADBLK.SYS exists to prevent bad blocks
% on a device from being allocated, and
% should never be directly accessed.
You cannot copy the BADBLK.SYS file, since this would lead to corruption of the file system.

?Device full
There is no more room on the disk.

?Cannot OPEN Devn: - protection violation
TAPFIL could not transfer the files. You are not allowed to write into an account you are not logged into unless the project number of that account is the same as the project number of the account you are copying from. You must either log into the System Operator's account, [1,2], or the account you are copying into to accomplish the transfer.

4.0 LISTING THE CONTENTS OF A TAPE - THE TAPDIR PROGRAM

The TAPDIR program displays a list of the files that have been stored on a tape. (NOTE: You may only use TAPDIR on a tape that was written via FILTAP.) The TAPDIR program has been designed to be as similar to the DIR program as possible.

To create a list of the files on a tape, enter the following command:

.TAPDIR{/switch}{Listfilespec=}{inspec1{/switch}{,inspec2{/switch}}...} RET

Where the optional Listfilespec specifies where the output listing is to be placed. If you specify no listfile or equal sign, the display goes to your terminal. By specifying a listfile, you can send the display to a disk file or printer.

The optional inspec allows you to select the files you wish to include in the directory listing. The default is the device and account you are logged into, and a file specification of *.*.

TAPDIR now asks for the device code and unit number of the drive containing the tape that you want to get a directory listing of:

Enter tape unit number:

Now TAPDIR creates the output listing, showing the relative position of each file on the tape, the full file specification, the size of the file, whether the file is a linked (L) or contiguous (C) file (that is, whether it is a sequential or a random file), and the date and time that the file was written to the tape. At the end of the listing, TAPDIR gives the total number of files and blocks that it has found.

4.1 Example

For example, to list all the files on a tape on your terminal, enter the following command:

```
.TAPDIR ALL:[] RET
Enter tape unit number: 0 RET
    1  DSK0: SYS     MAC   140,1      16 L  14-May-80  14:52:23
    2  DSK0: NBSORT  MAC   140,1       4 L  14-May-80  14:52:25
    3  DSK0: FILTAP  MAC   140,1      23 L  14-May-80  14:52:25
    4  DSK0: JANE    DAT   140,1      99 C  14-May-80  14:52:27
Total of 4 files in 142 blocks
```

To create a file (DIRECT.LST) in the account and device you are logged into that contains a list of all data (.DAT) files on the tape, enter the following command:

.TAPDIR = ALL:*.DAT[] RET

4.2 DIRTAP Switch

DIRTAP provides the following switch:

/KILL or /K — Delete and replace existing Listfile if it has the same specification as your Listfilespec. (Operation switch.)

4.3 Error Messages

You may see the following error messages when using the TAPDIR program:

?Cannot find DSK0:SCNWLD.SYS[1,4] or MEM:SCNWLD.SYS
: The TAPDIR program needs this file to be able to process wildcard symbols in your file specification. This message can indicate that SCNWLD.SYS does not exist, or that you do not have enough memory to load the file into your partition.

?Cannot READ Devn - device does not exist
?Cannot READ Devn - device is not mounted
: You tried to copy to or from a device that is not listed in the DEVTBL command in your SYSTEM.INI, does not have a driver in area [1,6] of the System Disk, is not file-structured, or is not mounted. ("Devn:" is the device you specified.)

%No file-oriented device corresponding to Devn: is mounted
: You specified a device, but left off the unit number. TAPDIR cannot find a logical unit that matches your specification. Try mounting the device.

?Tape is not file structured
: The tape you are trying to read was not written by the FILTAP program. The TAPDIR program can only read tapes written by FILTAP. Check to be sure you've mounted the correct reel of tape.

?More than one output specification
: You may not supply more than one output specification.

?Device full
: There is no more room on the disk.

%No such files
: TAPDIR was unable to find any files matching your input specification.

April 1981

BUILDING A TERMINAL DRIVER (THE NEWTRM PROGRAM)

Because of the device independent nature of the Alpha Micro system, adding a new kind of terminal is a simple matter-- you need only modify your system initialization command file to contain a terminal definition statement for the terminal, and add a terminal driver program for the new terminal to your System Disk. The terminal driver program defines the terminal to the system, giving information about the particular characteristics of the device. Because terminals differ so widely, certain functions may be useful only for certain types of terminals. The two main types of terminals are hard copy terminals, which output data to a permanent display via a printing mechanism, and CRT terminals (also called video display terminals, or VDT), which output data to a video screen. The terminal drivers for CRT terminals are the most difficult to write, since the programmer must worry about cursor positioning and other screen-oriented functions.

Alpha Micro supports a wide range of terminals by providing a large number of terminal driver programs (in both source and assembled form). In the past, if you wanted to use a terminal for which Alpha Micro does not supply a terminal driver, you had to try to modify an existing terminal driver to perform the functions of the new terminal, or had to laboriously write your own driver program. Either case required that you understand quite a bit about AMOS assembly language and terminal drivers.

Beginning with Release 4.5, Alpha Micro is providing a tool to help our users develop their own terminal driver programs for any kind of terminal they would like to add to the system. The tool is a terminal driver building program called NEWTRM. This program provides you a skeleton terminal driver, and allows you to customize the skeleton terminal driver according to the needs of your particular terminal. You customize the terminal driver by answering a number of questions with yes or no, values, or parameters which describe your terminal.

1.0 THE NEWTRM PROGRAM

Using NEWTRM to build a new terminal driver will take you about half an hour and will require that you are familiar with the characteristics of the terminal you need the driver for. It is a good idea to have the user's manual that accompanied your terminal at hand while you use NEWTRM so that you can look up the information that NEWTRM requests.

NEWTRM uses the following files. Do not erase or modify them.

DSK0:TABDEF.MAC[1,6] VARDEF.MAC[1,6]
DSK0:TDV1.MAC[1,6] DSK0:TDV2.MAC[1,6]
DSK0:ECHO.MAC[1,6] DSK0:NEWTRM.PCF[7,5]

To use NEWTRM, log into the Driver Library account, DSK0:[1,6]:

.LOG DVR: (RET)

Now, run the terminal driver building program NEWTRM by entering:

.NEWTRM (RET)

NEWTRM now gives a set of instructions:

Terminate all input lines with carriage returns.
All numeric input is in decimal.
Separate numeric answers on the same line with spaces.
All NUMERIC answers default to 0.
If you have any problems, refer to the NEWTRM documentation.

Next, NEWTRM asks a long series of questions. Your answers to these questions will tell NEWTRM how to customize the skeleton driver to form a terminal driver for your particular terminal. The result will be a .MAC file that you can assemble to produce an assembled terminal driver program.

You may enter your answers in either uppercase or lowercase letters, or a combination of both. Also, any response of yes or no you enter may be abbreviated to the first letter of the word. For example, to a yes-no question you wish to answer in the affirmative, you may enter any of the following:

YES
Yes
Y
y

We discuss each of the questions asked by NEWTRM below:

1.1 What is the name of the driver?

NEWTRM first asks for the name of the new terminal driver. Choose a one- to six-character name for the driver. You should use a name that readily identifies the type of terminal supported by the driver (for example, "HAZEL" for a Hazeltine terminal). Do not supply a file name extension (that is, if you want to use the name SOROC, enter just "SOROC", not "SOROC.TDV"). For example:

What is the name of the driver? SOROC (RET)

You will later add this terminal driver name to the system initialization command file TRMDEF statement that will define your terminal. (For information on modifying the system initialization command file to add a new terminal to your system, see The System Initialization Command File in the

"System Operator's Information" section of the AMOS Software Update documentation packet.)

1.2 Enter the number of nulls required after a line feed:

Some terminals require that one or more null characters be sent to them following a linefeed. These null characters serve as a delay so that the terminal does not print the first few characters following the linefeed on that same line, before actually performing the linefeed. Usually only hard copy terminals require this delay. For example, your terminal might require that 6 nulls be sent after a linefeed to ensure that the beginning of the next line actually follows the linefeed. Enter 0 if no delay is required.

1.3 Enter the number of nulls required after a form feed:

Some terminals, again usually just hard copy terminals, require that one or more null characters be sent to them following a form feed. These null characters serve as a delay so that the terminal does not print the first few characters following a form feed on the bottom of that same page, before actually performing the form feed. Your terminal, for instance, might require that 4 nulls be sent to it after a form feed in order to process the form feed before printing further characters. Enter 0 if no delay is required.

1.4 Does your terminal have a keyboard?

Some terminals are output-only devices, and have no keyboard. Enter yes or no. If your answer is no, NEWTRM will ask you no further questions. Instead, it will say "Please wait...", then finish building the terminal driver. For information on what to do next, once you are returned to AMOS command level, see Section 2, "Finishing Up."

1.5 Is RUBOUT a shift-underline?

Answer yes if the RUBOUT key (sometimes labeled RUB, DEL, or DELETE) is sent by pressing the Shift and Underscore keys simultaneously. This is the only way that a shifted rubout is generated. Otherwise, you must have a separate key for RUBOUT that you don't have to shift for; in this case, answer no.

1.6 Is your terminal a CRT?

Answer yes if your terminal has a video display screen. (CRT means cathode-ray tube, a common reference to the actual device which electronically displays characters on the screen. Often known as a VDT, or Video Display terminal.) Answer no if your terminal is a "hard copy"

terminal; that is, if it prints a permanent copy of its interaction with you and the system on paper. If you answer no, NEWTRM will ask you no further questions. Instead, it will say "Please wait...", then finish building the terminal driver. For information on what to do next, once you are returned to AMOS command level, see Section 2, "Finishing Up."

1.7 Enter the number of rows on the screen:

Enter the number of horizontal rows that may be displayed on the terminal at one time. This number is usually 16, 24 or 25 rows.

1.8 Now enter the number of columns:

Enter the number of vertical columns that may be displayed at one time. This number is usually 32, 40, 64, 72, 80 or 132 columns.

1.9 For the cursor positioning command, is the row sent first?

Following the cursor positioning command from the terminal driver, if your termimal must receive the ROW byte from the terminal driver before it receives the COLUMN byte, answer yes. If it must receive the COLUMN byte first, then the ROW byte, answer no.

1.10 Enter the positional offset from 1,1:

NEWTRM assumes that the home position on the video dsplay terminal is 1,1 (row 1, column 1.) However, many terminals make the home position 0,0, and some use 32,32, so NEWTRM accepts an offset value relative to 1,1. This offset value is added to 1,1. If the home position on your terminal is 0,0, enter -1. If the home position on your terminal is 32,32, enter 31.

1.11 Enter the decimal ASCII value of the function leadin code:

Some terminals use function codes, while others use control codes, to perform operations such as clear screen or position cursor. If your terminal accepts function codes, it requires a function leadin code to tell it to recognize the following input as a function code. NEWTRM requires from you the decimal equivalent of the ASCII value for the leadin code. A typical leadin code, for example, is an ESCAPE. You would enter the ASCII value for an ESCAPE, which is 27 (base 10). If your terminal uses only control codes (e.g., Control-K (^K) for cursor up), enter a 0.

1.12 Enter the delay (in clock ticks) required after functions:

Some terminals, whether using function codes or control codes, require a time delay to complete execution after receiving commands to CLEAR (clear screen), EOS (erase to end of screen), EOL (erase to end of line), or even to position the cursor. Some terminals may also require a time delay after any insertion or deletion of lines or characters. If your terminal requires a time delay after any of these operations, enter the number of clock ticks required to delay until the command taking the longest to execute is complete. You can calculate this number of clock ticks by:

1. Knowing the current baud rate setting of your system. (Say 19,200 baud.)

2. Counting the number of characters the terminal needs to generate to complete that longest operation. (Say it takes 1920 characters to do the longest operation on your terminal.)

3. Dividing the number of characters that need to be sent by the baud rate to obtain the decimal part of a second. (1920/19200 = 0.1 seconds.)

4. Knowing the cycles per second (Hertz, or Hz) of A.C. power used by your system as a reference frequency. In North America and a few other areas, 60 Hz is used, but most of the rest of the world uses 50 Hz. (Say 60 Hz.)

5. Finding the time of one cycle (one clock tick), in seconds. For 60 Hz, one cycle is approximately 0.01667 seconds. For 50 Hz, it is 0.02 seconds. (Say 0.01667 seconds per clock tick.)

6. Dividing the decimal part of a second, previously obtained, by the length of one clock tick to find the number of clock ticks required. (0.1/0.01667 = 6 clock ticks.)

The result, rounded up to the nearest whole clock tick (in our case, still 6 clock ticks), is the number of clock ticks required to delay before sending further information to the terminal while the terminal is performing a time-consuming function. NEWTRM causes the terminal driver to accomplish this delay by using the SLEEP monitor call and simply doing nothing for the duration of that operation.

If your terminal does not require a delay after receiving any command, enter a 0.

1.13 Does the terminal have insert and delete line functions?

Your terminal will have both or neither, but not just one or the other. Enter yes if it has both, or no if it has neither.

1.14 Does the terminal have erase to end of screen?

Enter yes if the terminal has the ERASE to END OF SCREEN (EOS) feature, or no if it does not.

1.15 Does the terminal have erase to end of line?

Enter yes if your terminal has the ERASE to END OF LINE (EOL) feature, or no if it does not.

1.16 Do you want function keys translated?

Some terminals have special keys called function keys. They might be labeled F1, F2, F3 and so on. When pressed, they send unique codes to the terminal driver to request the performance of equally unique functions. Those codes transmitted by these function keys can be "translated" to their equivalent codes in the Alpha Micro text editor, AlphaVUE. By answering yes to this question, you cause NEWTRM to set certain flags in the terminal driver so that VUE responds directly to the various function keys. If you do not have function keys, or do not want them translated, answer no. The next question will not be asked.

1.17 Enter the decimal ASCII value of the function key leadin code:

You are asked this question only if you answer yes to the previous one. To recognize the unique code sent from the terminal which indicates that the key pressed is a function key, the terminal driver must first receive the function leadin code. This function leadin code is typically an ESCAPE, having a decimal ASCII value of 27. If your terminal has function keys, enter the decimal ASCII value of the function leadin code it transmits to the driver. If your terminal does not have any function keys, but rather incorporates terminal functions into control codes, enter a 0.

1.18 Enter the delay required between function key characters:

When you press a function key, the terminal returns to the terminal driver a sequence of characters, one at a time, to accomplish the specified function. Each separate character is sent at the repeat rate of the terminal (the rate at which characters are repeated when, on the various terminals, you either hold down a key or hold down both a key and the REPEAT key). The terminal driver determines whether or not the character sequence is that of a function key by first testing for the function key leadin code character. If it finds that character, it then performs a counting loop internally, then looks at a buffer. Function key characters alone are found in the buffer at the end of that loop; if the terminal driver finds a character in the buffer, it knows a function key is being transmitted. If the length of the loop time is too short, the terminal driver won't find the character in

the buffer whether or not it is a function key character. Much too long a loop time simply delays the system. The length of time the terminal driver does its timing loop is determined by your answer to this question. Determine the delay count required by:

1. Finding which is lower: 1) the "effective repeat baud rate" (the maximum repeat rate (in characters per second) the terminal can accomplish, multiplied by 10); or 2) the actual baud rate the terminal will be operating at. (Say the "effective repeat baud rate" is 600 (for 60 characters per second). Say the actual baud rate of the terminal is 19,200. The lower value is 600.)

2. Dividing 1 by that lower value. (1/600 or 0.001666....)

3. Dividing the result by a constant value depending on which processor your system uses. For the AM-100, that constant is .0000075, and for the AM-100/T, that constant is .000004333.... (Say it is an AM-100. 0.001666.../0.0000075 = 222.222 (approximately).)

4. Taking the final result and rounding it up to the nearest integer. (223.)

Enter that value (in our example, 223) in response to the question.

NOTE: Sometimes, if the value as determined above is slightly too low (due to overhead or other factors), and when the function key is pressed several times rapidly in succession (or the REPEAT key is also used), the function key translation routine cannot translate the function key properly. Changing the value determined above to a slightly higher value should solve the problem.

1.19 Are there terminating characters sent by function keys?

If you answered yes to the question, "Do you want function keys translated?", NEWTRM asks you this question (and the following question if you answer yes to this one).

Some terminal function keys send an "end-of-key" code that tells the system the function key transmission is over. This code may be a single character, or a combination of two or more characters. Typically, the code is <EOT>, <ETX>, <CR>, <CR><LF>, or something similar. The terminal driver needs to know if this terminating code is sent (but doesn't care what characters the code consists of); if the code is sent, the driver discards all of its characters. Answer yes to this question if the function keys send a terminating code of one or more characters. Answer no if no terminating code characters are sent.

NOTE: Some terminals allow you to set the character or characters of the terminating code with a manual switch built into the terminal itself. You may set this switch into any position available; in any case, the terminal driver discards it. However, we recommend that you do not select the <EOT> (i.e., the <End of Transmission>, or Control-D) switch position if your

terminal operates over a telephone line. This is because <EOT> will cause some modems, which are between the terminal and the system running the terminal driver program, to disconnect the phone line.

1.20 Enter the number of terminating characters:

If you answered no to the question, "Are there terminating characters sent by function keys?", this question will not appear. If you answered yes, now enter the number of characters in the function key terminating code. For example, if the terminating code is <ETX>, enter a 1; if the terminating code is <CR><LF>, enter a 2. The terminal driver will ignore that many characters following the character unique to the function key. The terminal driver does not care what those characters are, but must know exactly how many of them are in the terminating code.

1.21 Are there values sent by function keys that should be discarded?

Some terminal function keys send more than the function leadin code and the character unique to the given key. Any characters coming before these two should be discarded. For example, the BEEHIVE DMxx series returns four characters, <STX><ESC><ASCII character><ETX>. For that terminal, <ESC> is the leadin character, the ASCII character is unique to a given key, and <STX> must be discarded. <ETX> is the function key terminator, also to be discarded, but which is handled in a different way, based on your answers to the previous two questions. Answer yes to the current question if your terminal transmits characters (including the terminating character) other than the leadin character and the unique character. If you answer no, NEWTRM skips to the second question below.

1.22 Enter the decimal ASCII value of the character to discard:

If you answer yes to the question "Are there values sent by function keys that should be discarded?", NEWTRM asks you this question. Answer by entering the decimal equivalent of the ASCII value of the character to be discarded, then type a RETURN. Do not enter the value of any terminating characters. NEWTRM will repeat the question until you identify all the characters (other than terminating characters) to be discarded. After you have finished, type just a RETURN when the question is asked again; NEWTRM will go on to the next question.

1.23 Is the XXX command implemented on the terminal?

Where XXX is a command name. NEWTRM begins a four question loop which it repeats for each of 27 commands your terminal may implement (that is, electronically support based on its construction). For example, you first see:

Is the clear screen command implemented on the terminal?

If you answer yes, NEWTRM then asks:

Does the clear screen command require a delay?

Whether you answer yes or no to this question, but dependent upon information asked of you earlier, NEWTRM may ask you:

Does the clear screen command use the standard leadin code?

And finally, NEWTRM requests the command code or codes your terminal transmits for the clear screen command:

Enter the decimal ASCII value(s) of the command code(s):

Then NEWTRM repeats the sequence for the second command:

Is the cursor home command implemented on the terminal?

If you answer no this time, NEWTRM skips the three questions dependent on a yes and inquires about the next command. And goes on in this manner to ask you about each of the 27 commands listed below, basing the secondary questions upon your response to Is the XXX command implemented on the terminal?. Here are the commands NEWTRM asks you about:

Command Name	TCRT Function
clear screen	0
cursor home	1
cursor return (is not asked; it is an ASCII standard)	2
cursor up	3
cursor down	4
cursor left	5
cursor right	6
lock keyboard	7
unlock keyboard	8
erase to end of line	9
erase to end of screen	10
protect field (reduced intensity)	11
protect field (normal intensity)	12
enable protection of fields	13
disable protection of fields	14
delete line	15
insert line	16
delete character	17
insert character	18
read cursor address	19
read character at current cursor address	20
start blinking field	21
end blinking field	22
start line drawing or alternate character set	23
end line drawing or alternate character set	24
set horizontal position	25
set vertical position	26
set terminal attributes	27
cursor positioning	none

The commands in the above list are the only ones that NEWTRM asks you about (except for cursor return, which is standard for all ASCII terminals, and which therefore is automatically provided by NEWTRM). They are the TCRT commands supported by Alpha Micro. The numbers to the right of the commands in the list above are the TCRT functions for each command. (NOTE: Cursor positioning, the last on the list, has no TCRT function.) If you wish to implement additional terminal commands, the source file which NEWTRM creates for you must be modified "by hand" to contain those additional commands.

NOTE: Some terminals (such as the ADM-31) implement the end blinking field and/or the end line drawing field as an end attribute command. If this is the case for your terminal, enter the single end attribute command sequence when either the end blinking field or the end line drawing field sequences are requested by NEWTRM.

1.23.1 Does the XXX command require a delay? - Answer the first additional question with a yes or no. The delay referred to is the delay you entered in response to the question, "Enter the delay (in clock ticks) required after functions:"

1.23.2 Does the XXX command use the standard leadin code? - This second question is not asked unless you have indicated, in response to an earlier question NEWTRM asked you, that there is a leadin code. Answer yes if the command in question is implemented as a multiple character sequence. Answer no if the command is implemented as a control character.

1.23.3 Enter the decimal ASCII value(s) of the command code(s): - Enter the decimal equivalent of the ASCII value(s) of the command code(s). Do not enter the leadin code. NEWTRM automatically places it if the command requires a leadin code. Enter the values by separating them with a space, then type a RETURN. You can enter a maximum of 10 values per command.

This is the last of the secondary questions. Now NEWTRM asks you about the next command in the list, until it has inquired about all 27. Then it goes on to the question below.

1.24 Does the cursor positioning command require a leadin code?

Answer yes if your terminal requires the same leadin code for the cursor positioning command sequence as it does for the 27 commmands above. Answer no if either no leadin is used or if a different leadin is required.

1.25 Enter the cursor positioning command sequence:

Do not enter the leadin code. If you answered yes to the question above, saying the same leadin code is required, NEWTRM automatically places it. Enter the decimal value of each byte, separated by spaces, until a maximum of 10 bytes are entered. Type a RETURN after you enter all the bytes of the sequence.

1.26 Enter a code returned by a function key:

This question and the next one are only asked if you have indicated earlier that your terminal has one or more function keys that are to be translated into AlphaVUE commands. These two questions alternately repeat until you enter just a RETURN in response to this question.

Enter the ASCII value (in decimal) returned by a specific function key that uniquely identifies that key. Then type RETURN. For example, assume that the leadin code for your terminal's function keys is <ESC>. Say that one of the terminal's cursor positioning keys sends <ESC><A> when pressed. In response to this question you would enter 65, the decimal ASCII value for A. (See Section 5.0 for a complete decimal equivalent ASCII chart.)

1.27 Now type a VUE command that corresponds to the function key:

This question follows the previous question for each of the function keys. It correlates the terminal's function keys to specific operations within VUE. In response to this question, press the actual key (plus the Control-key) that VUE uses for the specific operation you want to correlate to the function key. Do NOT type a RETURN. (NEWTRM expects you to type a VUE command, and is waiting for it. If you type a RETURN, NEWTRM does not process the RETURN until it again asks you, "Enter a code returned by a function key:", at which time NEWTRM thinks you are telling it there are no more function codes to enter.) For example, an IBM terminal sends <ESC><A> from a function key to indicate MOVE CURSOR UP. To make VUE correspond to that function key of your IBM terminal, enter a Control-K in response to this question, because ^K is the VUE command for MOVE CURSOR UP.

2.0 FINISHING UP

NEWTRM has asked its last question. You will see "Please wait...", then:

```
The driver is complete.  You may assemble and test it now.
.
```

NEWTRM has created the source file for your terminal driver. Now you must assemble the program by using the MACRO command. Enter:

```
.MACRO SOROC (RET)
```

where "SOROC" is the name you used in telling NEWTRM what terminal driver to build. When MACRO finishes and returns you to AMOS command level, rename the .PRG file it produced to the terminal driver extension, .TDV:

```
.RENAME/D *.TDV=SOROC.PRG (RET)
```

You now have a finished terminal driver program, customized for your terminal.

3.0 ERROR MESSAGES

NEWTRM has one error message. You may also see standard AlphaPASCAL error messages. Typing a Control-C (^C) closes the partially completed output file and aborts NEWTRM.

?Bad answer - try again
: You typed an answer that did not start with a Y, y, N or n. NEWTRM repeats the question.

4.0 WORKSHEET FOR YOUR TERMINAL

Use this section as a worksheet for your specific terminal. You will find the information you need in your terminal user's manual. If NEWTRM asks about a feature your user's manual doesn't mention, the terminal probably does not implement that feature. You may find commands using different terminology in the user's manual. Comparing this document's text to the manual's, you will be able to find the information NEWTRM is asking for.

NEWTRM WORKSHEET FOR THE _____ TERMINAL

NEWTRM uses the following files. Do not erase or modify them.

DSK0:NEWTRM.PCF[7,5]	DSK0:TDV1.MAC[1,6]
DSK0:ECHO.MAC[1,6]	DSK0:TDV2.MAC[1,6]
DSK0:TABDEF.MAC[1,6]	DSK0:VARDEF.MAC[1,6]

Log into the Driver Library account, DSK0:[1,6]:. Run NEWTRM by entering:

.NEWTRM (RET)

Remember, these are the rules:

Terminate all input lines with carriage returns.
All numeric input is in decimal.
Separate numeric answers on the same line with spaces.
All NUMERIC answers default to 0.
If you have any problems, refer to the NEWTRM documentation.

Also, remember that you may use any of these combinations for yes or no responses: YES, Yes, yes, Y, y or NO, No, no, N or n.

What is the name of the driver (6 char. or less; no extension)?	_____
Enter the number of nulls required after a line feed:	__
Enter the number of nulls required after a form feed:	__
Does your terminal have a keyboard?	Y N
Is RUBOUT a shift-underline?	Y N
Is your terminal a CRT?	Y N
Enter the number of rows on the screen:	__
Now enter the number of columns:	__
For the cursor positioning command, is the row sent first?	Y N
Enter the positional offset from 1,1:	__
Enter the decimal ASCII value of the function leadin code:	__
Enter the delay (in clock ticks) required after functions:	__
Does the terminal have insert and delete line functions?	Y N
Does the terminal have erase to end of screen?	Y N
Does the terminal have erase to end of line?	Y N
Do you want function keys translated?	Y N
Enter the decimal ASCII value of the function key leadin code:	__
Enter delay required between function key characters:	__
Are there terminating characters sent by function keys?	Y N
Enter the number of terminating characters:	--
Are values sent by function keys that should be discarded?	Y N
Enter the decimal ASCII value of the character to discard:	__

Now summarize your terminal's commands by placing checkmarks and codes in the appropriate columns for each command below.	IS THE COMMAND IMPLE-MENTED? Y : N	DOES THE COMMAND REQUIRE A DELAY? Y : N	USE THE STANDARD LEADIN CODE? Y : N	PLEASE ENTER UP TO 10 COMMAND CODES IN DECIMAL ASCII VALUES (SPACED APART):
clear screen	_ : _	_ : _	_ : _	_ _ _ _ _ _ _ _ _ _
cursor home	_ : _	_ : _	_ : _	_ _ _ _ _ _ _ _ _ _
cursor up	_ : _	_ : _	_ : _	_ _ _ _ _ _ _ _ _ _
cursor down	_ : _	_ : _	_ : _	_ _ _ _ _ _ _ _ _ _
cursor left	_ : _	_ : _	_ : _	_ _ _ _ _ _ _ _ _ _
cursor right	_ : _	_ : _	_ : _	_ _ _ _ _ _ _ _ _ _
lock keyboard	_ : _	_ : _	_ : _	_ _ _ _ _ _ _ _ _ _
unlock keyboard	_ : _	_ : _	_ : _	_ _ _ _ _ _ _ _ _ _
erase to end of line	_ : _	_ : _	_ : _	_ _ _ _ _ _ _ _ _ _
erase to end of screen	_ : _	_ : _	_ : _	_ _ _ _ _ _ _ _ _ _
protect field (red. intensity)	_ : _	_ : _	_ : _	_ _ _ _ _ _ _ _ _ _
protect field (norm. intensity)	_ : _	_ : _	_ : _	_ _ _ _ _ _ _ _ _ _
enable protection of fields	_ : _	_ : _	_ : _	_ _ _ _ _ _ _ _ _ _
disable protection of fields	_ : _	_ : _	_ : _	_ _ _ _ _ _ _ _ _ _
delete line	_ : _	_ : _	_ : _	_ _ _ _ _ _ _ _ _ _
insert line	_ : _	_ : _	_ : _	_ _ _ _ _ _ _ _ _ _
delete character	_ : _	_ : _	_ : _	_ _ _ _ _ _ _ _ _ _
insert character	_ : _	_ : _	_ : _	_ _ _ _ _ _ _ _ _ _
read cursor address	_ : _	_ : _	_ : _	_ _ _ _ _ _ _ _ _ _
read char. at cursor address	_ : _	_ : _	_ : _	_ _ _ _ _ _ _ _ _ _
start blinking field	_ : _	_ : _	_ : _	_ _ _ _ _ _ _ _ _ _
end blinking field	_ : _	_ : _	_ : _	_ _ _ _ _ _ _ _ _ _
start line draw or alt.char.set	_ : _	_ : _	_ : _	_ _ _ _ _ _ _ _ _ _
end line draw or alt.char.set	_ : _	_ : _	_ : _	_ _ _ _ _ _ _ _ _ _
set horizontal position	_ : _	_ : _	_ : _	_ _ _ _ _ _ _ _ _ _
set vertical position	_ : _	_ : _	_ : _	_ _ _ _ _ _ _ _ _ _
set terminal attributes	_ : _	_ : _	_ : _	_ _ _ _ _ _ _ _ _ _
cursor positioning	_ : _	_ : _	_ : _	_ _ _ _ _ _ _ _ _ _

Does the cursor positioning command require a leadin code? Y N
Enter cursor positioning sequence (to 10 spaced ASCII values): _____

Enter a code returned by a function key (to 10 ASCII values): _____
Now press a VUE command that corresponds to the function key: _
Enter a code returned by a function key (to 10 ASCII values): _____
Now press a VUE command that corresponds to the function key: _
Enter a code returned by a function key (to 10 ASCII values): _____
Now press a VUE command that corresponds to the function key: _
Enter a code returned by a function key (to 10 ASCII values): _____
Now press a VUE command that corresponds to the function key: _
Continue to repeat for more function keys to correspond.

Assemble and test the driver now:

```
_.MACRO drivername RET
```

Rename the .PRG file produced to the terminal driver extension, .TDV:

```
_.RENAME/D *.TDV=drivername.PRG RET
```

5.0 DECIMAL EQUIVALENT ASCII CHART

VALUE	CHARACTER	VALUE	CHARACTER	VALUE	CHARACTER
0	NULL	48	0	96	`
1	SOH	49	1	97	a
2	STX	50	2	98	b
3	ETX	51	3	99	c
4	ECT	52	4	100	d
5	ENQ	53	5	101	e
6	ACK	54	6	102	f
7	BEL	55	7	103	g
8	BS	56	8	104	h
9	HT	57	9	105	i
10	LF	58	:	106	j
11	VT	59	;	107	k
12	FF	60	<	108	l
13	CR	61	=	109	m
14	SO	62	>	110	n
15	SI	63	?	111	o
16	DLE	64	@	112	p
17	DC1	65	A	113	q
18	DC2	66	B	114	r
19	DC3	67	C	115	s
20	DC4	68	D	116	t
21	NAK	69	E	117	u
22	SYN	70	F	118	v
23	ETB	71	G	119	w
24	CAN	72	H	120	x
25	EM	73	I	121	y
26	SS	74	J	122	z
27	ESC	75	K	123	{
28	FS	76	L	124 |	|
29	GS	77	M	125	}
30	RS	78	N	126	~
31	US	79	O	127	DEL
32	SP	80	P		
33	!	81	Q		
34	"	82	R		
35	#	83	S		
36	$	84	T		
37	%	85	U		
38	&	86	V		
39	'	87	W		
40	(	88	X		
41	)	89	Y		
42	*	90	Z		
43	+	91	[		
44	,	92	\		
45	-	93	]		
46	.	94	^		
47	/	95	_		

AMOS Software Update Documentation
AMOS Release 4.5
April 1981

SYSTEM PROGRAMMER'S INFORMATION

This section contains the following documents:

I/O Programming on the Alpha Micro Computer, Revision A01

Terminal Service System

AMOS 4.5 SOFTWARE UPDATE DOCUMENTATION PACKET

These documents reflect AMOS Versions 4.5 and later

ALPHA MICROSYSTEMS
17881 Sky Park North
Irvine, CA 92714

May 1980
Revision A01

I/O PROGRAMMING ON THE ALPHA MICRO COMPUTER

Whenever a program must communicate with an I/O device, whether within an interface driver, a device driver, or a hard-coded program, it is said to be using I/O programming. The Alpha Micro central processors (the AM-100 and the AM-100/T) both use a memory-mapped I/O technique. Except for a few minor differences, these two processors are completely compatible; those minor differences occur in I/O programming. This document outlines the various methods and techniques used in I/O programming, and explains the differences between the two Alpha Micro processors.

The Alpha Micro processors both contain 256 separate I/O locations, each eight bits wide. These I/O locations are mapped into memory locations FF00 (hex) through FFFF (hex). These locations may be accessed as bytes (AM-100 and AM-100/T) or as words (AM-100/T only). When byte operations are performed on the I/O locations, the two processors behave identically; it is only when word operations are performed that differences appear.

For word operations, the AM-100 uses only the low-order half of the word during a write operation. During a read, the I/O byte is replicated in both halves of the resulting word.

The AM-100/T, on the other hand, will properly read and write words to the I/O locations, providing that the I/O controller handles word operations. Currently, no Alpha Micro interface boards handle 16-bit transfers. However, to leave this feature available for future use, a slight change in the way that I/O word operations are performed was necessary. On the AM-100/T processor, a read-word operation to a controller handling byte-reads only, returns the resultant byte in only the upper half of the word. This differs from the AM-100, which returns the byte in both halves on a read-word operation. Likewise, word-mode I/O writes use only the data in the upper byte when using byte-oriented controllers.

Therefore, when programming byte-oriented devices (all currently available devices are byte-oriented), you should avoid word-mode instructions whenever possible to avoid any confusion. If word-mode instructions must be used, then only the high-order byte should be used. If word-mode I/O to byte-oriented controllers is restricted to the high-order byte, the AM-100 and AM-100/T are completely compatible.

(Changed 1 May 1981)

Instructions that generate a read-modify-write sequence (format 7 and 10 op codes using destination modes 1 through 7) ignore the read portion of the sequence and do only the write. This was engineered into the design of the processors to avoid confusion with I/O devices which use the same I/O location for two different functions based on whether a read or a write operation is being performed. Therefore, if you wish to read the contents of an I/O location with a format 10 op code, it may only be done in the source field of the instruction. When writing to an I/O location, any format 7 op code or format 10 op code with destination modes 1-7 may be used if such an instruction makes sense without the read operation.

Examples:

1.	CLRB	@#^H0FF04	Clears I/O location 4.
2.	CLR	@#^H0FF04	AM-100: Clears I/O location 4. AM-100/T: Clears I/O location 4 on a byte-oriented controller. Clears locations 4 and 5 on word-oriented controllers.
3.	TSTB	@#^H0FF05	Won't work without read-- writes junk to I/O location 5.
4.	BIT	#2,@#^H0FF06	Won't work without read-- writes junk to I/O location 6.
5.	BIT	@#^H0FF06,#2	AM-100: Will work fine. AM-100/T: Will not work-- contents of I/O location 6 are read into upper byte only.
6.	BIT	@#^H0FF06,#2_8.	Will work fine on all processors. The AM-100 brings data into both halves; the AM-100/T into the upper half.
7.	CMP	@#^H0FF07,R0	Will not work unless controller is word-oriented and the controller is on the AM-100/T.
8.	CMPB	@#^H0FF08,R0	Will work fine on both processors.
9.	BISB	#2,@#^H0FF09	Will not work. Requires a read-modify-write sequence for proper operation. Writes junk into I/O location 9.

Example #6, above, is the only change likely to be required in converting an existing driver to run on the AM-100/T. (The underscore, _, used in example #6 is a MACRO expression operator which tells MACRO to shift the expression (2) the specified number of bits. Hence, a shift of eight (decimal) bits puts the 2 into the upper half of the word.)

(Changed 1 May 1981)

April 1979

TERMINAL SERVICE SYSTEM

1.0 INTRODUCTION

The terminal service system incorporated in the AMOS monitor is a flexible and efficient set of routines and drivers for interfacing a variety of different terminals with different interface boards. You may write your own drivers for terminals and interfaces not supported by Alpha Micro. This document describes the general structure and function of the terminal service system, but does not go into details on how to write user-defined drivers. We have made available the sources to TRMSER, FILSER and several terminal drivers for those individuals who want to write their own terminal driver programs. Details on the monitor calls used within terminal and interface drivers are in the AMOS Montitor Calls Manual, (DWM-00100-42). For a general overview of the Alpha Micro Operating System, see Part III of Introduction to AMOS, (DWM-00100-65).

2.0 GENERAL STRUCTURE

The monitor contains a general terminal processing routine called TRMSER whose function is to link user programs and monitor processes to the outside world of interactive terminals; this is done purely on a data basis, without regard to terminal or interface hardware. TRMSER processes data on a character-by-character basis. Monitor calls are available to your programs for passing characters and full buffers of data between the terminals and the system. Think of TRMSER as a telephone operator who switches calls back and forth between sources and destinations without regard to the type of telephone in use or the name of the person using that telephone. TRMSER also provides the synchronous link to the asynchronous world of the terminal hardware.

TRMSER is a monitor routine that is embedded in the operating system skeleton monitor, SYSTEM.MON. In addition to the general TRMSER routine in the monitor, there must exist one or more routines called drivers that take the data from TRMSER and translate it as necessary into the specific codes required by the hardware and then route it to the terminal through the interface board. These drivers reside in account [1,6] of the System Disk, and are automatically loaded into system memory in response to the terminal definition (TRMDEF) command lines in the systm initialization command file (SYSTEM.INI file) at the time of system startup. Driver programs MUST be reentrant; only one copy of a driver is loaded into memory regardless of the number of terminals or interface boards of that type defined on the system.

The terminal service system uses two general types of drivers: interface drivers and terminal drivers. Interface drivers contain the routines necessary to get data characters to and from the interface boards that plug into the S-100 bus. Terminal drivers contain routines that process each

character that goes to or from the terminal. Terminal drivers handle code character conversions, echoing functions, line-feed null characters, cursor control, and special functions as required by the type of terminal in use.

3.0 INTERFACE DRIVERS

Interface drivers link the TRMSER routines and the actual hardware responsible for getting characters to and from the terminal device. The interface drivers are assembly language programs with an .IDV extension. The filename of the interface driver appears in the TRMDEF command line of the SYSTEM.INI file, and tells the system what kind of interface is being used by the terminal defined by that command line. Typical drivers are for the AM-300 board, the Processor Technology 3P+S board and the IMSAI SIO-2 board. The interface driver handles all initialization sequences for the board if required, and also sets up interrupt processes if the board supports it. Those boards which are not interrupt driven get put into the clock scanner queue for asynchronous access every clock tick. A special interface driver exists on the system called the PSEUDO interface driver; it controls no hardware at all, but instead represents a software-controllable interface for inter-job communication and control.

4.0 TERMINAL DRIVERS

Terminal drivers customize the handling of character input and output based on the type of terminal being used. They are assembly language programs that have the .TDV extension. The filename of the driver is the name by which the terminal type is referenced in the TRMDEF statements in the SYSTEM.INI file. Typical terminal drivers are for the ADM3, the Soroc, the Teletype, the Multiterm and the Silent 700.

The terminal driver processes all input and output chracters, and determines if these characters need special handling because of the type of terminal being used. The terminal driver handles echo control and different methods of character deletion. For example, most CRT terminals have the ability to back up and erase the character being deleted, while hard copy terminals (such as the Teletype) must explicitly echo the character, usually in a format that distinguishes the characters from those accepted as input.

Terminal drivers may also be written for software-controlled ports, and two such drivers are built into the monitor already. The PSEUDO and NULL terminal drivers are used in conjunction with the PSEUDO interface driver, and provide a means for passing characters straight through to the controlling job or discarding output characters that are unimportant. Terminal drivers are usually unconcerned with the type of interface used to physically tie the terminal to the computer.

5.0 INTERSYSTEM DRIVER LINKS

The relationship between the different elements of the terminal service system can seem confusing at first; nevertheless, efficient systems-level programming requires a thorough understanding of the links that exist between these items. The following units are referenced in further discussions:

1. JOB - A job is the unit that controls the operation of one task or a series of tasks running on the system. A job is independent of any other jobs running on the system unless it is tied to them by special user software. Every job on the system has a unique name one to six characters long.

2. TERMINAL - A terminal is the hardware device used to physically transfer data into the system, and get data from the system to the user on a character-by-character basis. Terminals do not themselves have names. Typical terminals might be a Teletype, ADM3, Soroc, etc.

3. TERMINAL DEFINITION - A terminal definition unit is a block of memory in the system area set up by a TRMDEF statement. It is the basic unit by which a terminal in the system is referenced when attaching that terminal to a specific job, or when using the terminal as an I/O device under control of the TRM device driver. The terminal definition unit has a unique name one to six characters long.

4. INTERFACE DRIVER - An interface driver is the program that transfers characters back and forth between the terminal and the hardware interface board to which the terminal is physically connected. The interface driver has a name one to six characters long that is referenced by the TRMDEF statements in the SYSTEM.INI file. Interface drivers reside in account DSK0:[1,6], and have the extension .IDV.

5. TERMINAL DRIVER - A terminal driver is the program that performs the character code conversions required by the terminal in use. The program has a name (one to six characters long) that is referenced only in the TRMDEF statement of the SYSTEM.INI. Terminal drivers reside in account DSK0:[1,6] and have an extension of .TDV.

6. DEVICE DRIVER - A device driver is a program that allows the system to communicate with any I/O device connected to the system. Device drivers are written for disks, tape units, printers and terminals. The handling of terminals as devices for use by the generalized file service system is done through the TRM device driver, and not through the terminal drivers themselves. Device drivers have a one to three character name that is referenced in the device table statement (DEVTBL) in the SYSTEM.INI, and in user file specifications (e.g., AMS1:FILNAM.TXT). Device drivers reside in account DSK0:[1,6] and have the extension of .DVR.

The terminal definition unit contains the links to the defined interface driver and to the defined terminal driver; it thus is the basic unit by which terminals are referenced on the system. When a terminal is attached to a job, the JCB (Job Control Block) and the terminal definition unit become linked to each other. A job is considered to be detached if it is not linked up to a terminal definition unit, and a terminal is considered to be detached if it is not linked to a JCB. A job may only be linked to one controlling terminal, and vice versa.

A job performs I/O operations through the particular device driver referenced by the device specified in the file specification. A job performs terminal operations through the linked terminal definition unit for the terminal that is controlling that job. A detached job is placed into terminal wait state if it attempts to perform a terminal input or output operation. Since I/O operations differ in structure and usage from terminal operations, performing I/O operations to a terminal must be done through some mechanism other than directly into the terminal definition unit. From a system standpoint, the terminal definition unit performs differently than a device driver. To allow this, a general device driver has been written called TRM which allows terminals to be accessed as devices, as opposed to being accessed only as job controlling terminals. This operation will be described later.

5.1 Terminal Input Characters

Terminal input characters are processed through a complex chain of events. When a terminal keyboard character is struck by the operator, it is transferred to the hardware interface which then passes it to the interface driver routine. The interface driver routine reads in the character and then passes it to the TRMSER processor. TRMSER puts the character into the input buffer to wait for pickup by the program or monitor. As an asynchronous event, if echoing is not suppressed or is local to the terminal, TRMSER passes the character back to the terminal driver (when it is about to be echoed) to again allow the terminal routine to perform special functions. An example of this is the special echoing of Control-U characters for line deletion or rubouts for character deletion. The terminal routine then passes the character (or the converted character) back to TRMSER to be sent to the output processor.

5.2 Terminal Output Characters

Terminal output characters can come from two main sources: 1. characters to be echoed from the input processor; and, 2. characters to be output (generated by the monitor or user program) as messages or data to the user. Both are handled differently from a buffering standpoint, but eventually are presented to a common output routine in TRMSER to be sent to the terminal. Each character for output goes from TRMSER to the terminal driver for possible output code conversion or character translation. An example of this would be the null sequence sent after every line-feed for timing

purposes to the Silent 700 terminal driver. The terminal driver processes the character and then sends it back to TRMSER for position processing. TRMSER then passes the output character (or converted character) from the terminal driver to the interface driver where it is physically output to the terminal.

6.0 USING TERMINALS AS I/O DEVICES

Most programs (including the print spooler) perform input and output operations to I/O devices rather than to the controlling terminal. In some instances it is desirable to perform these operations on a terminal rather than a specific I/O device defined by its own device driver. One example would be the printing of data on a Multiterm or Teletype, or the use of these terminals as the output device of the printer spooler. Any terminal may be accessed as a device through the general device driver called TRM. The TRM device driver acts as a software link between the format required by the FILSER file service system and the TRMSER terminal service system. Any terminal can be considered a device by using the device code TRM and using the name given the terminal definition unit as the filename (the extension and PPN are ignored in the file specification).

For example, suppose you have a Teletype connected to an AM-300 board on port number two. Your TRMDEF command in the SYSTEM.INI file might look like this:

```
TRMDEF TELLY,AM-300=2:2,TELTYP,80,100,20
```

This Teletype may then be accessed as an output device by the file specification for any I/O operations requiring a specific device:

```
TRM:TELLY
```

To output directly to this device from BASIC, you would first open the device:

```
OPEN #file,"TRM:TELLY",OUTPUT
```

and then display data as follows:

```
PRINT #file,variable-list
```

The variable list may contain text and variables that you want to print as well as PRINT USING masks.

If you are planning on using terminals as I/O devices or to spool to them, it might save some space to include the TRM.DVR program in system memory during system startup. You would use the command:

```
SYSTEM TRM.DVR[1,6]
```

before the last SYSTEM command in the SYSTEM.INI file. It is not necessary to do this, however, since FILSER automatically loads the TRM driver into your memory partition when it is needed if it is not in system memory.

AMOS Software Update Documentation
AMOS Release 4.5
April 1981

BASIC PROGRAMMER'S INFORMATION

This section contains the following documents:

BASORT - BASIC Subroutine for Sorting Random and Sequential Files, Revision A02

COMMON - BASIC Subroutine to Provide Common Variable Storage

FLOCK - BASIC Subroutine to Coordinate Multi-user File Access, Revision A01

SPOOL - BASIC Subroutine for Spooling Files to the Line Printer, Revision A02

XLOCK - BASIC Subroutine for Multi-user Locks, Revision A01

XMOUNT - BASIC Subroutine to Mount a Disk, Revision A01

AMOS 4.5 SOFTWARE UPDATE DOCUMENTATION PACKET

These documents reflect AMOS Versions 4.5 and later

ALPHA MICROSYSTEMS
17881 Sky Park North
Irvine, CA 92714

April 1981
Revision A02

BASORT - BASIC SUBROUTINE FOR SORTING RANDOM AND SEQUENTIAL FILES

BASORT is an external assembly language subroutine, callable from AlphaBASIC, which can sort both random and sequential files. (A random file is one in which the records are physically grouped together in one area of the disk. A sequential file's records are not necessarily contiguous on the disk.) If your memory partition is large enough (that is, if the entire file will fit into user memory at one time), BASORT will perform an internal memory-based heap sort. If there is not enough room in user memory for the entire file, BASORT does a disk-based polyphase merge-sort. The combination of these two modes results in a respectably fast sort utility.

The BASORT package consists of three modules--BASORT.SBR, AMSORT.SYS, and FLTCNV.PRG--all of which must be in memory before BASORT is used. BASORT.SBR is automatically loaded into user memory upon direction from the BASIC program, but AMSORT.SYS and FLTCNV.PRG must be loaded by the user into system or user memory. Once AMSORT.SYS and FLTCNV.PRG are loaded, they are re-entrant; BASORT.SBR is not. (For more information on loading programs into system memory, see The System Initialization Command File document in the "System Operator's Information" section of the AMOS Software Update Documentation Packet.)

1.0 SORTING RANDOM FILES

When you use BASORT to sort random files, BASORT sorts the file onto itself (that is, it replaces the original, unsorted file with a file containing the sorted data). Therefore, if you wish to retain a backup copy of the unsorted file, you must create a separate copy to be sorted.

BASORT for random files is called via:

```
XCALL BASORT,CHANEL,RECCNT,RECSIZ,K1SIZ,K1POS,K1ORD,K2SIZ,
            K2POS,K2ORD,K3SIZ,K3POS,K3ORD,K1TYPE,K2TYPE,K3TYPE
```

Where:

CHANEL	File channel on which file to be sorted is open for random processing.
RECCNT	Number of records in the file you are sorting.
RECSIZ	Size of the records in the file you are sorting.
K1SIZ	The size, in bytes, of sort key #1.
K1POS	The first character position occupied by key #1.

(Changed 30 April 1981)

K1ORD Sort order of key #1: 0 for ascending sequence, 1 for descending seqeuence.

K2SIZ The size, in bytes, of sort key #2.

K2POS The first character position occupied by key #2.

K2ORD Sort order of key #2. (See K1ORD, above.)

K3SIZ The size, in bytes, of sort key #3.

K3POS The first character position occupied by key #3.

K3ORD Sort order of key #3. (See K1ORD, above.)

K1TYPE The data type of key #1. Key types are:

0 = String
1 = Floating Point
2 = Binary

K2TYPE The data type of key #2. (See K1TYPE, above.)

K3TYPE The data type of key #3. (See K1TYPE, above.)

Keys are the elements of the data records you wish to base your sort on (i.e., customer name, order number, etc.). If you want to use less than three keys, all entries in the XCALL command line for the unused keys must be zero. If the key types are omitted, BASORT assumes string data type.

All arguments in the XCALL command line are numeric, but may be passed as either floating point or string values. For example, "99" is a valid entry.

■ The first character in a record is considered to be position 1.

2.0 SORTING SEQUENTIAL FILES

When you sort a sequential files, you must specify both an input and an output file. If you wish to sort a file back onto itself, you may specify the same file for both input and output. Before BASORT is called, the file must be opened for input. BASORT leaves the file open for output.

Call BASORT for sequential files via:

```
XCALL BASORT,CHAN1,CHAN2,RECSIZ,K1SIZ,K1POS,K1ORD,K2SIZ,K2POS,
            K2ORD,K3SIZ,K3POS,K3ORD
```

Where:

CHAN1	The file channel on which the input file is open.
CHAN2	The file channel on which the output file is open.
RECSIZ	The size, in bytes, of the largest record in the file, including the terminating carriage return/linefeed characters.
K1SIZ-K3ORD	The same as for random files.

If you are in doubt about the size of the largest record to be sorted, make RECSIZ larger than necessary; too small a value results in truncation of data records.

NOTE: Sequential files contain only ASCII data. For that reason, when you sort sequential files you do not have to specify the data type of the sort keys; BASORT knows that all keys in a sequential file are strings.

3.0 BASORT ERROR MESSAGES

?AMSORT.SYS not found in memory

The sort utility routine, AMSORT.SYS, must be loaded into user or system memory before calling BASORT.SBR.

?Bad channel number in XCALL BASORT

The channel number you passed to BASORT was invalid. This error can occur if the file is not open, or if the value given as channel is not an integer.

?File improperly open in XCALL BASORT

When you call BASORT, the file you wish to sort must be open for INPUT or RANDOM processing.

?FLTCNV.PRG not found in memory

The floating-point conversion module, FLTCNV.PRG, must be loaded into user or system memory before calling BASORT.SBR.

?Illegal value in XCALL BASORT

One of the arguments to the BASORT call was invalid. Check the key sizes and positions to make sure they fit into the record size which you specified. Also make sure that you have given valid key types.

?Read file error in XCALL BASORT

An error occurred during a read operation while sorting your file.

?Write file error in XCALL BASORT

An error occurred during a write operation while sorting your file.

?Wrong record size in XCALL BASORT

The record size you specified when calling BASORT does not match the record size you specified when you OPENed the file.

COMMON - BASIC SUBROUTINE TO PROVIDE COMMON VARIABLE STORAGE

1.0 INTRODUCTION

COMMON is an assembly language routine that allows you to place data into a common storage area in memory. This is useful for passing data between chained programs, passing messages between jobs, or any other function that requires a data area accessible to more than one program or person. By assigning a name to each packet of information within the common area, you can have several of these packets in common storage ready to be retrieved by other users or programs at various times.

1.1 THE COMMON SUBROUTINE

You can call COMMON to send data to the common area via:

```
XCALL COMMON,SEND,MSGNAM,VAR
```

You can call COMMON to retrieve data from the common area via:

```
XCALL COMMON,RECV,MSGNAM,VAR
```

Where:

SEND A one-byte binary variable that contains zero. You usually define SEND as follows:

```
MAP1 SEND,B,1,0
```

RECV A two-byte binary variable, where the first byte must be set to one, and the second byte functions as a flag that indicates whether or not COMMON found the requested packet of information. If COMMON did not find that packet, it returns a zero in this byte; otherwise it is non-zero. You usually define RECV as:

```
MAP1 RECV
MAP2 F'RCV,B,1,1
MAP2 RCVFLG,B,1,0
```

MSGNAM A six-character string that specifies the name of the packet to be sent or received.

VAR A variable to hold the data to be sent or received. The variable must represent data that is less than 151 bytes long.

You may load COMMON into either system or user memory. If you load COMMON into a user's memory partition, only that user can access the data stored by COMMON. If you load COMMON into system memory (making the data accessible to all users), be sure that MSGNAM is unique for each packet.

At this point, programs 1 and 2 have both been delayed. Since no other programs are present, the reasons for their delays will remain unchanged. DEADLOCK has occurred.

But DEADLOCK will not occur if program 2 requests permission to open files 1001 and 1002 for exclusive use in the same order as program 1. For DEADLOCK to occur, program 1 must be granted permission to open file 1001 for exclusive use, but be delayed permission to open file 1002 for exclusive use. However, if program 1 is granted permission to open file 1001 for exclusive use, the corrected program 2 (see program 1) will not be allowed to execute lines 21-990; thus it will be unable to obtain permission to open file 1002 for exclusive use. DEADLOCK cannot occur.

5.0 BIBLIOGRAPHY

SHAW, A.C. (1974). The Logical Design of Operating Systems, Prentice-Hall, Inc., Englewood Cliffs, N.J.

4.0 PREVENTING DEADLOCK

NOTE: For the purposes of the following discussion, permission to open a file or use a record will be referred to as possessing a resource.

The possession of a resource by some job XYZ can directly or indirectly cause the execution of other jobs to be delayed. It is then possible for one of these delayed jobs to possess a resource needed by job XYZ, thus causing execution of job XYZ to be delayed also. This is known as a DEADLOCK. None of the jobs involved can proceed since each requires a resource owned by one of the other jobs involved. The situation is permanent because none of the jobs involved can proceed until one of the other jobs proceeds and relinquishes a needed resource.

DEADLOCK can only occur if a job requests more than one resource simultaneously. There is a simple method of preventing DEADLOCK which in most cases is feasible to implement: that is, ALWAYS request resources in the same order.

Here is a simple illustration of the principle. First we consider what can happen if resources are requested in differing order in two programs:

```
10  !PROGRAM 1
20  XCALL FLOCK,0,2,RET,1001
21  XCALL FLOCK,0,2,RET,1002
100 REMARK  ** BODY OF PROGRAM **
990 XCALL FLOCK,1,0,RET,1002
991 XCALL FLOCK,1,0,RET,1001
992 END

10  !PROGRAM 2
20  XCALL FLOCK,0,2,RET,1002
21  XCALL FLOCK,0,2,RET,1001
100 REMARK ** BODY OF PROGRAM **
990 XCALL FLOCK,0,2,RET,1001
991 XCALL FLOCK,0,2,RET,1002
992 END
```

Consider the following sequence of execution:

1. Program 1 executes lines 10 and 20, obtaining exclusive permission to open file 1001.

2. Program 2 executes lines 10 and 20, obtaining exclusive permission to open file 1002. It then executes line 21, and must be delayed because Program 1 already has exclusive permission to open file 1001.

3. Program 1 executes line 21, and must be delayed because Program 2 already has exclusive permission to open file 1002.

```
10 !REORGANIZATION PROGRAM
20 XCALL FLOCK,0,0,RET,1001
21 XCALL FLOCK,0,0,RET,1002
22 OPEN #1001,"INDEX",RANDOM,512,KEY1
23 OPEN #1002,"DATA",RANDOM,512,KEY2
30 XCALL FLOCK,4,2,RET,1001
31 XCALL FLOCK,4,2,RET,1002
32 CALL REORGANIZE  !  REORGANIZE INDEXED DATA FILE
33 XCALL FLOCK,6,0,RET,1002
34 XCALL FLOCK,6,0,RET,1001
40 CLOSE #1001 : CLOSE #1002
41 XCALL FLOCK,1,0,RET,1001
42 XCALL FLOCK,1,0,RET,1002
50 END
60 REORGANIZE:
70   REMARK *** SUBROUTINE GOES HERE ***
80   RETURN


10  !INQUIRY PROGRAM
20  XCALL FLOCK,0,0,RET,1001
21  XCALL FLOCK,0,0,RET,1002
22  OPEN #1001,"INDEX",RANDOM,512,KEY1
23  OPEN #1002,"DATA",RANDOM,512,KEY2
30  INPUT "EMPLOYEE #",EMP$
31  IF EMP$="" THEN LEAVE
40  CALL LOOKUP !LOCATE EMP$ IN INDEX FILE, RETURN EMPLOYEE REC # IN KEY2
41              !XCALL FLOCK,0,0,RET,KEY1 IS IN EFFECT WHEN LOOKUP RETURNS
42  IF KEY2=0 THEN ?"EMPLOYEE NOT ON FILE" : GOTO 30
50  XCALL FLOCK,3,4,RET,1002,KEY2
51  IF RET <> 1 THEN 55
52   INPUT "DO YOU WISH TO WAIT? ",ANSWER$
53   IF ANSWER$ <> "Y" AND ANSWER$ <> "YES" THEN 30
54   XCALL FLOCK,3,0,RET,1002,KEY2
55  READ #1000,EMPLOYEE'RECORD
56  XCALL FLOCK,5,0,RET,1002,KEY2
57  XCALL FLOCK,5,0,RET,1001,KEY1
60  CALL DISPLAY !  DISPLAY EMPLOYEE'RECORD
70  GOTO 30
80  LEAVE:
90   CLOSE #1001 : CLOSE #1002
91   XCALL FLOCK,1,0,RET,1001
92   XCALL FLOCK,1,0,RET,1002
100  END
200 LOOKUP: REMARK **SUBROUTINE GOES HERE**
299   RETURN
300 DISPLAY: REMARK **SUBROUTINE GOES HERE**
399   RETURN
```

```
10 ON ERROR GOTO ABORT
20 XCALL FLOCK,0,0,RET,1000
21 OPEN #1000,"FILE",RANDOM,6,KEY
30 XCALL FLOCK,3,0,RET,1000,1
31 XCALL FLOCK,3,0,RET,1000,2
32 KEY = 1 : READ #1000,X
33 KEY = 2 : READ #1000,Y
34 XCALL FLOCK,5,0,RET,1000,2
35 XCALL FLOCK,5,0,RET,1000,1
40 PRINT X-Y
50 CLOSE #1000
51 XCALL FLOCK,1,0,RET,1000
60 END
70 ABORT:
71  XCALL FLOCK,2,0,KEY
72  ON ERROR GOTO 0
```

3.3 Improved File Interlocks

In Section 3.2 we said that file-open interlocks can incur long delays upon any users trying to access a file after one user has opened it and therefore locked them out. Nevertheless, it is sometimes necessary to lock an entire file for exclusive use. For example, if file XYZ is becoming full, you might wish to copy the file XYZ into a new, larger file TEMP, and then delete XYZ and rename TEMP to XYZ. Or, as another example, you might wish to reorganize an index and data file. Obviously, during these maneuvers, you want assurance that no other user can access the file.

Action 4 obtains exclusive access to a file by obtaining exclusive access to all the records of that file. Exclusive access is relinquished by using Action 6. Action 3, Mode 0 or 4, is necessary before reading a sequence of records in order to avoid the interconsistency problem. If Action 4 is used, it is necessary to use Action 3, Mode 0 or 4, before reading individual records which won't be used for updating. This is because a user who has exclusive use of a file can re-create it, which requires that all other users with the file open must then reopen it. Action 3 performs the necessary reopenings.

3.3.1 Example - Here are two partial programs which illustrate the use of improved file interlocks:

```
5   ON ERROR GOTO ABORT
10  XCALL FLOCK,0,0,RET,1000
20  OPEN #1000,"FILE",RANDOM,6,KEY
30  KEY = 1
40  XCALL FLOCK,3,2,RET,1000,KEY
50  READ #1000,X
60  X=X+1
70  WRITE #1000,X
80  XCALL FLOCK,5,0,RET,1000,KEY
90  CLOSE #1000
100 XCALL FLOCK,1,0,RET,1000
110 END
120 ABORT:
130  XCALL FLOCK,2,0,RET
140  ON ERROR GOTO 0
```

3.2.2 The Interconsistency Problem - Here is how the programs of Section 1.3 could be rewritten to incorporate Record-Update interlocks:

```
10 ON ERROR GOTO ABORT
20 XCALL FLOCK,0,0,RET,1000
21 OPEN #1000,"FILE",RANDOM,6,KEY
30 KEY = 1
31 XCALL FLOCK,3,2,RET,1000,KEY
32 READ #1000,X : X=X+1 : WRITE #1000,X
33 XCALL FLOCK,5,0,RET,1000,KEY
40 KEY = 2
41 XCALL FLOCK,3,2,RET,1000,KEY
42 READ #1000,X : X=X+1 : WRITE #1000,X
43 XCALL FLOCK,5,0,RET,1000,KEY
50 CLOSE #1000
51 XCALL FLOCK,1,0,RET,1000
60 END
70 ABORT:
71  XCALL FLOCK,2,0,KEY
72  ON ERROR GOTO 0
```

Since the second program does not update 'FILE', it requests permission to open it using Mode 0 with Action 0. This enables other programs which read but do not update 'FILE' to open and process 'FILE' simultaneously.

3.2 Record-Update Interlocks

Most programs open files when the programs begin, and close those files when they end. The programs may not actually need the files to be open throughout execution, but by not repeatedly opening and closing the files, the programs avoid many undesirable delays.

File-open interlocks that are set lock out the entire file; if a file is open throughout the run of a program, and thus unavailable to programs run by other users, serious or annoying delays can result.

Although file-open interlocks do prevent concurrency problems, they generally reduce concurrency far more than is necessary. Typically, file-open interlocks lock out the entire file to prevent access to the single record. Locking out an entire file to prevent access to a single record is like using a sledge hammer to drive a push-pin. All that is actually necessary is to delay any other user attempting to modify the record until the user originally accessing the record is done.

Consider an example of application in which you and several other users are interactively updating an employee record file. Assume files are kept open only where required. Once you display an employee's record, it is necessary that all the other users wait for you to finish making changes to that record before they can, in turn, access it; otherwise two users might concurrently attempt to update the same employee record. This results in the multiple update problem described in Section 1.2. In other words, all other users must wait for one user to enter changes to the employee's record before any other user can access and modify that record. This is called a record-update interlock, and is a far less severe restriction to all the users accessing a file than a file-open interlock is.

Actions 3 and 5 of FLOCK permit concurrent access to individual records to be controlled. Action 3, Mode 0 or 4, is used before reading a sequence of records which will not be used for updating, in order to prevent interconsistency errors (see Section 1.3). Action 5 is used after the sequence of reads. Action 3, Mode 2 or 6, is used before reading records which will be used for updating. Action 5 is used again after rewriting the records.

3.2.1 The Multiple Update Problem - Here is how the program of Section 1.2 could be rewritten to incorporate Record-Update interlocks:

```
5   ON ERROR GOTO ABORT
10  XCALL FLOCK,0,2,RET,1000
20  OPEN #1000,"FILE",RANDOM,6,KEY
30  KEY = 1
40  READ #1000,X
50  X=X+1
60  WRITE #1000,X
70  CLOSE #1000
80  XCALL FLOCK,1,0,RET,1000
90  END
100 ABORT:
110  XCALL FLOCK,2,0,RET
120  ON ERROR GOTO 0
```

3.1.2 The Interconsistency Problem - Here is how the programs of Section 1.3 could be rewritten to incorporate file-open interlocks:

```
10  ON ERROR GOTO ABORT
20  XCALL FLOCK,0,2,RET,1000
30  OPEN #1000,"FILE",RANDOM,6,KEY
40  KEY = 1 : READ #1000,X
50  X = X+1 : WRITE #1000,X
60  KEY = 2 : READ #1000,X
70  X = X+1 : WRITE #1000,X
80  CLOSE #1000
90  XCALL FLOCK,1,0,RET,1000
100 END
110 ABORT:
120  XCALL FLOCK,2,0,RET
130  ON ERROR GOTO 0
```

```
10  ON ERROR GOTO ABORT
20  XCALL FLOCK,0,0,RET,1000
30  OPEN #1000,"FILE",RANDOM,6,KEY
40  KEY = 1 : READ #1000,X
50  KEY = 2 : READ #1000,Y
60  PRINT X-Y
70  CLOSE #1000
80  XCALL FLOCK,1,0,RET,1000
90  END
100 ABORT:
110  XCALL FLOCK,2,0,RET
120  ON ERROR GOTO 0
```

The above programs will now function correctly in a concurrent environment. While the first program is updating 'FILE', no other programs can have 'FILE' open. This prevents the second program from reading 'FILE' when it is in a partially updated state.

(Changed 30 April 1981)

The problems outlined in Sections 1.2 and 1.3 can be solved by using FLOCK to any of the above levels of complexity. In your design you are free to trade off complexity for performance, so long as you use a single level of complexity consistently for any given data file.

3.1 File-Open Interlocks

Using just Actions 0 through 2, it is possible to implement a very simple file access coordination scheme which solves the problems of Sections 1.2 and 1.3. Action 0, Mode 0 or 4, is used before opening a file for input only (that is, opening a file for RANDOM processing, upon which only READs will be performed). Action 0, Mode 2 or 6, is used before opening a file for output (i.e., a file open for RANDOM processing, upon which READs or WRITEs will be performed, or a file which may be re-created). Finally, Action 1 is used after closing any file, and Action 2 is used before any abnormal termination points in the program.

3.1.1 The Multiple Update Problem

- Here is how the program of Section 1.2 could be rewritten to incorporate file-open interlocks:

```
10 XCALL FLOCK,0,2,RET,1000
20 OPEN #1000,"FILE",RANDOM,6,KEY
30 KEY = 1
40 READ #1000,X
50 X=X+1
60 WRITE #1000,X
70 CLOSE #1000
80 XCALL FLOCK,1,0,RET,1000
90 END
```

The program now will function correctly in a concurrent environment. If any other programs have 'FILE' open when line 10 is executed (and have correctly informed FLOCK of the fact with Action 0), FLOCK will make the above program wait until the other program closes 'FILE'. Furthermore, no more programs will be allowed to open 'FILE' until the above program reaches line 80.

The above program has no provisions for the user typing ^C, or for other errors occurring which will abort execution. This can be corrected by further rewriting the program, as follows:

A Return-Code greater than 1 is an indication of some programming error. For calls to FLOCK which do not use modes 4 or 6, the statement:

```
IF Return-Code>1 THEN PRINT "FLOCK Error" : STOP
```

should be used while debugging. For calls which use modes 4 or 6, Return-Code = 1 should be checked to determine if FLOCK was able to immediately satisfy the request. Modes 4 and 6 are generally used in this way to allow the user to cancel a request which may involve a lengthy delay.

2.2 Queue Block Requirements

The FLOCK subroutine builds its dynamic tables out of monitor queue blocks. It is very important, BEFORE running any BASIC program using FLOCK, to ensure that the monitor is configured to make an adequate number of these queue blocks available. A good rule of thumb is to assume that each request for which permission has been granted requires three queue blocks until permission is relinquished.

NOTE: The monitor is initially generated with 20 free blocks in the available queue. At any time in the SYSTEM.INI file prior to the final SYSTEM command, you may execute the 'QUEUE nnn' command which will allocate 'nnn' more queue blocks for general use.

Once the system is up and running, no more queue blocks can be added to the monitor. You must give your best guess at your total requirements before running the program. The QUEUE command takes on a new life once the system is running. If you type the QUEUE command the system will respond by typing back the current number of free queue blocks in the available queue list. It is by this method that you may keep tabs on the operation of your system as far as queue block usage.

3.0 USING FLOCK

There are three levels of increasing complexity with which FLOCK subroutine calls may be incorporated into a program system:

1. Use Actions 0 through 2 to implement file-open interlocks (see Section 3.1).

2. Use Actions 0 through 2 to implement file-open interlocks and use Actions 3 and 5 to implement individual record-update interlocks (see Section 3.2).

3. Use Actions 0 through 2, 4, and 6 to implement complete file interlocks and use Actions 3 and 5 to implement individual record-processing interlocks (see Section 3.3).

Action 5, Mode 0: Informs FLOCK that processing of 'Record' of 'File', for which permission was granted by Action 3, has been completed. If data has been buffered for output, it is written to disk.

Action 6, Mode 0: Informs Flock that exclusive processing of 'File', for which permission was granted by Action 4, has been completed. Any succeeding programs which are granted use of 'File' by Actions 3 or 4 will automatically reopen 'File'. This is done in case exlusive processing of 'File' has caused it to be re-created. If data has been buffered for output, it is written to disk.

2.1.2 File - File specifies a file-channel number. File is ignored by Action 2 and may be omitted if 'Record' is also omitted. The file specified may be either RANDOM or SEQUENTIAL for Actions 0 and 1, but must be a RANDOM file for all other actions.

In order for FLOCK to function properly, file-channel numbers should denote specific and unique files. This means you must systematically assign file-channel numbers to your files when designing applications programs, being careful to assign the same numbers to the same files and different numbers to different files.

File-channel numbers 1 through 999 have been reserved for use by Alpha Micro software. Although there is nothing to prevent your programs from using these numbers, you are advised against doing so in conjunction with FLOCK so that no conflict can arise between your application programs and any present or future Alpha Micro software on your system.

2.1.3 Record - Record specifies a logical record number. Record is ignored and may be omitted for Actions 0 through 2, 4, and 6.

2.1.4 Return-Code - Return-Code denotes a variable in which FLOCK places a number that indicates the success or failure of an action:

Code 0: Successful (All actions)
Code 1: Resource unavailable (Actions 0, 3, 4)
Code 2: Open request has already been granted (Action 0)
Code 3: Permission to open must first be granted (Actions 1, 3-6)
Code 4: Duplicate request for use of some record in file (Actions 3, 4)
Code 6: Permission to use some record in file must first be granted (Actions 5, 6)
Code 100: Unimplemented Action
Code 101: File-channel number must be open in AlphaBASIC for RANDOM processing (Actions 3-6)

(Changed 30 April 1981)

Action 0, Mode 6: Requests permission to open 'File' for exclusive use. If the request cannot be immediately granted, Return-Code 1 is returned.

Action 1, Mode 0: Informs FLOCK that 'File' has been closed. Implicitly informs FLOCK that any processing of records in 'File' has been completed (i.e., Actions 5 or 6 are performed automatically as necessary).

Action 2, Mode 0: Informs FLOCK that abnormal program termination is about to occur. Performs Action 1 as necessary.

Action 3, Mode 0: Requests permission to read 'Record' of 'File' for non-exclusive use (i.e., record will not be used to update file). Permission to open 'File' must already be granted. The request is placed in a first-come-first-served queue and the program is delayed until the request can be granted.

Action 3, Mode 2: Requests permission to read 'Record' of 'File' for exclusive use (i.e., record will be used to update file). Permission to open 'File' must already be granted. The request is placed in a first-come-first-served queue and the program is delayed until the request can be granted.

Action 3, Mode 4: Requests permission to read 'Record' of 'File' for non-exclusive use (i.e., record will not be used to update file). Permission to open 'File' must already be granted. If the request cannot be immediately granted, Return-Code 1 is returned.

Action 3, Mode 6: Requests permission to read 'Record' of 'File' for exclusive use (i.e., record will be used to update file). Permission to open 'File' must already be granted. If the request cannot be immediately granted, Return-Code 1 is returned.

Action 4, Mode 2: Requests permission to read/write all records of 'File' for exclusive use (i.e., processing will update and possibly re-create file). Permission to open 'File' must already be granted. The request is placed in a first-come-first-served queue and the program is delayed until the request can be granted.

Action 4, Mode 6: Requests permission to read/write all records of 'File' for exclusive use (i.e., processing will update and possibly re-create file). Permission to open 'File' must already be granted. If the request cannot be immediately granted, Return-Code 1 is returned.

The READ-WRITE-READ-WRITE sequence in the first program can be considered as steps in a single update operation. To prevent the situation outlined above from occurring, we need a mechanism for preventing access to a collection of data during such an update operation. Otherwise, we may retrieve a collection of data which has only been partially updated. In actual applications, this can lead to accessing nonexistent records through an index file, incorrect totals on reports, inconsistent reports, etc.

2.0 THE FLOCK SUBROUTINE

Use FLOCK to prevent the kinds of problems we discussed in the paragraphs above. FLOCK provides a way to synchronize attempts at accessing files and devices so that you can avoid partially updating or scrambling data.

2.1 Flock Calling Sequence

The calling sequence for FLOCK in BASIC is:

```
XCALL FLOCK, Action, Mode, Return-Code, File, Record
```

Where:

1. Action, Mode, File, and Record are all either floating point expressions which evaluate to positive integer values, or string expressions which represent positive integer values.

2. Return-Code is a 6-byte floating point variable.

2.1.1 Action & Mode - Action, modified by mode, specifies the action to be performed by FLOCK:

Action 0, Mode 0: Requests permission to open 'File' for non-exclusive use. The request is placed in a first-come-first-served queue and the program is delayed until the request can be granted.

Action 0, Mode 2: Requests permission to open 'File' for exclusive use. The request is placed in a first-come-first-served queue and the program is delayed until the request can be granted.

Action 0, Mode 4: Requests permission to open 'File' for non-exclusive use. If the request cannot be immediately granted, Return-Code 1 is returned.

```
10 OPEN #1,"FILE",RANDOM,6,KEY
20 KEY = 1 : READ #1,X
25 X = X+1 : WRITE #1,X
30 KEY = 2 : READ #1,X
35 X = X+1 : WRITE #1,X
40 CLOSE #1 : END
```

```
10 OPEN #1,"FILE",RANDOM,6,KEY
20 KEY = 1 : READ #1,X
30 KEY = 2 : READ #2,Y
40 PRINT X-Y
50 CLOSE #1 : END
```

If the values in records one and two of 'FILE' are identical, then they should continue to be identical if the first program (which increments the values in both records by one) is executed. Hence, if the values in records one and two are identical, and we execute both of the above programs concurrently, we would like the second program to print zero, thus:

X	USER #1	REC #1	#2	USER #2	X	Y
-	OPEN #1,"FILE",RANDOM,6,KEY	5	5			
5	KEY = 1 : READ #1,X	5	5			
6	X = X+1 : WRITE #1,X	6	5			
5	KEY = 2 : READ #1,X	6	5			
6	X = X+1 : WRITE #1,X	6	6			
6	CLOSE #1 : END	6	6			
-		6	6	OPEN #1,"FILE",RANDOM,6,KEY	-	-
-		6	6	KEY = 1 : READ #1,X	6	-
-		6	6	KEY = 2 : READ #1,Y	6	6
-		6	6	PRINT X-Y	6	6
				0		
-		6	6	CLOSE #1 : END	6	6

However, under some circumstances it is possible for the second program to print 1, rather than 0:

X	USER #1	REC #1	#2	USER #2	X	Y
-	OPEN #1,"FILE",RANDOM,6,KEY	5	5			
5	KEY = 1 : READ #1,X	5	5			
6	X = X+1 : WRITE #1,X	6	5			
6		6	5	OPEN #1,"FILE",RANDOM,6,KEY	-	-
6		6	5	KEY = 1 : READ #1,X	6	-
6		6	5	KEY = 2 : READ #1,Y	6	5
6		6	5	PRINT X-Y	6	5
				1		
6		6	5	CLOSE #1 : END	6	5
5	KEY = 2 : READ #1,X	6	5			
6	X = X+1 : WRITE #1,X	6	6			
6	CLOSE #1 : END	6	6			

(Changed 30 April 1981)

X	USER #1	REC #1	USER #2	X
-	OPEN #1,"FILE",RANDOM,6,KEY	5		
-	KEY = 1	5		
5	READ #1,X	5		
6	X=X+1	5		
6	WRITE #1,X	6		
6	CLOSE #1	6		
6	END	6		
-		6	OPEN #1,"FILE",RANDOM,6,KEY	-
-		6	KEY = 1	-
-		6	READ #1,X	6
-		6	X=X+1	7
-		7	WRITE #1,X	7
-		7	CLOSE #1	7
-		7	END	7

NOTE: In this example, the value in record 1 is initially 5.

However, under some circumstances it is possible for record 1 to only be incremented by 1, instead of 2, after being accessed by two users:

X	USER #1	REC #1	USER #2	X
-	OPEN #1,"FILE",RANDOM,6,KEY	5		
-	KEY = 1	5		
5	READ #1,X	5		
5		5	OPEN #1,"FILE",RANDOM,6,KEY	-
5		5	KEY = 1	-
5		5	READ #1,X	5
5		5	X=X+1	6
5		6	WRITE #1,X	6
5		6	CLOSE #1	6
5		6	END	6
6	X=X+1	6		
6	WRITE #1,X	6		
6	CLOSE #1	6		
6	END	6		

To prevent this situation from occurring, we need a method of preventing overlap in READ-modify-WRITE sequences on shared data.

1.3 The Interconsistency Problem

Consider the following two programs:

April 1981
Revision A01

FLOCK - BASIC SUBROUTINE TO COORDINATE MULTI-USER FILE ACCESS

1.0 INTRODUCTION

Some special file protection is required when your system has two or more users, and those users can run a program or programs which access the same files. FLOCK is a BASIC subroutine which protects files from this potential concurrent access, and protects one user from accessing information that another user is updating at the same time. The remainder of Section 1 in this document describes in detail the potential problems of multi-user file access. Section 2 details how you can use the FLOCK ("File Locking") subroutine from a BASIC program to coordinate shared file access and processing. Section 3 gives you some schemes to implement FLOCK in your programs. Section 4 discusses the hazard of "Deadlock," and how to avoid it. Section 5 is a bibliography.

1.1 FLOCK Program Requirements

The FLOCK.SBR program is an external assembly language subroutine which is callable from BASIC. FLOCK only functions properly if it is loaded into system memory (via the SYSTEM command in the system initialization command file, DSK0:SYSTEM.INI[1,4]). FLOCK also requires that you have FLTCNV.PRG in system memory.

1.2 The Multiple Update Problem

Consider the following program:

```
10 OPEN #1,"FILE",RANDOM,6,KEY
20 KEY = 1
30 READ #1,X
40 X=X+1
50 WRITE #1,X
60 CLOSE #1
70 END
```

The purpose of this program is to increment record 1 of 'FILE' by one. If two users execute this program concurrently, we wish the value in record one to be incremented by two, thus:

FLOCK - BASIC SUBROUTINE TO COORDINATE MULTI-USER FILE ACCESS

Table of Contents

ALPHA MICROSYSTEMS
17881 Sky Park North
Irvine, CA 92714

FLOCK - BASIC SUBROUTINE TO COORDINATE MULTI-USER FILE ACCESS

April 1981
Revision A01

This document reflects AMOS Versions 4.5 and later

April 1981
Revision A02

SPOOL - BASIC SUBROUTINE FOR SPOOLING FILES TO THE LINE PRINTER

1.0 INTRODUCTION

SPOOL is an assembly language routine that you can call from BASIC to spool a disk file to the line printer. (To "spool" a file is to insert it into the printer queue, after which you can continue to do other things while your file waits in the queue for its turn to be printed.) You can specify to SPOOL which printer you want the file to be printed on, the number of copies to print, the form to print it on, the lines per page and the width (measured in characters) per line. Also you can specify any combination of switches to turn on or off the banner option, the delete option (which deletes the file from the printer queue after printing), the header option, the formfeed option, or the wait option.

The current version of SPOOL (AMOS versions 4.2 and later) is fully compatible with earlier versions of SPOOL. In other words, the only information that you must supply to SPOOL is the specification of the file you want to print; all other parameters are optional.

2.0 THE SPOOL SUBROUTINE

Call SPOOL via:

```
XCALL SPOOL,FILE$,PRINTER$,SWITCHES,COPIES,FORM$,LPP,WIDTH
```

where:

FILE$	A string variable or expression that gives the specification of the file you want to print. If you specify a file which does not exist, SPOOL doesn't tell you that it can't find the file (but, of course, doesn't print anything).
PRINTER$	A string variable or expression that gives the name of the printer to which you want to send the file. If PRINTER$ is a null string, SPOOL uses the default printer.
SWITCHES	A floating point variable or expression that specifies various control switches and flags that affect the printing of the file. You must load FLTCNV.PRG into system or user memory if you are going to use the SWITCHES argument.

(Changed 30 April 1981)

The control switches that SPOOL uses are exactly the same as the switches used by the monitor PRINT command. (See the AMOS System Commands Reference Manual (DWM-00100-49), for information on PRINT.)

Each switch you can use has a numeric code associated with it (see below). For example, the BANNER switch code is 1; the DELETE switch code is 4. Set control switches by putting the sum of the appropriate switch codes into the SWITCHES variable. For example, if you want to use the BANNER and DELETE switches (to tell the line printer spooler program to print a banner page and delete the file after printing it), load SWITCHES with 5 (BANNER code + DELETE code). If you omit SWITCHES, SPOOL uses the default switches for the selected printer.

Switch codes:

BANNER	1
NOBANNER	2
DELETE	4
NODELETE	8
HEADER	16
NOHEADER	32
FF	64
NOFF	128
WAIT	256

COPIES — A floating point variable or expression that specifies the number of copies to be printed. If you omit COPIES or it is zero, the line printer spooler program prints one copy.

FORM$ — A string variable or expression that specifies the form on which the file is to be printed. If you omit FORM$ or it is a null string, the line printer spooler uses the NORMAL form.

LPP — A floating point variable or expression that specifies the number of lines per page. SPOOL only uses this value if you have specified the HEADER switch in the SWITCHES variable. If you omit LPP, the spooler program uses the default value for the specified printer.

WIDTH — A floating point variable or expression that specifies the width (in characters) of the print line. If you omit WIDTH, the spooler program uses the default value for the specified printer.

2.1 Error Messages

The SPOOL subroutine returns no error messages except:

?No spooler allocated

If you see the message above, it means that no line printer spooler program is currently running on the system.

April 1981
Revision A01

XLOCK - BASIC SUBROUTINE FOR MULTI-USER LOCKS

XLOCK is an assembly language routine that your BASIC program can call to set and test "locks." Include the XLOCK.SBR in system memory by using the SYSTEM command within the SYSTEM.INI file. NOTE: You MUST include XLOCK in system memory.

A lock is a tool to help you synchronize attempts to access devices and files. You can imagine the problems that result when you have two users trying to update the same record of the same file at the same time. A lock is an entity created by a program to help it keep track of whether a certain device, file, etc., is in use at the specific time that the program wants to access it. The general way that the locking system works is this:

1. When you want to prevent access to something (a file, a device, etc.) while your program accesses it, you create (that is, "set") a system lock on that resource.

2. Whenever you want to access a device or file, your program tries to set the lock associated with that item; if it is already set, you know that another user's program is using the device or file.

3. When you are finished accessing a device or file, you destroy (that is, "clear") the lock so that other programs can now access the resource.

Note that a system lock is NOT a security device-- it's a convenience. If a program wants to allow its users to write to a file without checking to see if another user is there first, it can do so (and run the risk of create chaos). A system lock simply provides a convenient way to help a program keep its users from conflicting in their attempts to use system resources. The only job that can clear a lock is the job that originally set the lock. BASIC does not automatically clear locks when a program exits, so be careful that your program clears any locks it has set before it exits. (For more background information on why locks are necessary, see the document FLOCK: BASIC Subroutine to Coordinate Multi-user File Access, in the "BASIC Programmer's Information" section of the AMOS Software Update documentation packet.)

1.0 THE XLOCK SUBROUTINE

Call XLOCK from BASIC via:

```
XCALL XLOCK, MODE, LOCK1, LOCK2
```

(Changed 30 April 1981)

Where:

MODE The function you want to perform. These modes are:

Mode 0: Set lock and return.
Mode 1: Set lock. (Wait if already locked; then set).
Mode 2: Clear lock (if set by your job).
Mode 3: Return list of all system locks and the jobs that set them.

(See below for a discussion of each mode.)

LOCK1 The first digit of the lock code. (See below.)

LOCK2 The second digit of the lock code. (See below.)

Use MAP statements at the front of your program to define MODE, LOCK1, and LOCK2 as two-byte binary variables. (They may not be floating point or string variables.) For example:

```
MAP1 MODE, B, 2
MAP1 LOCK1, B, 2
MAP1 LOCK2, B, 2
```

Before you call XLOCK, your BASIC program must first set up the correct values for MODE, LOCK1, and LOCK2.

2.0 THE LOCKS

A system lock is a two-level numeric lock; the number representing either level may be from 1 to 65535. (A value of zero in either position acts as a wildcard. That is, any number will match in that position when it comes to clearing or setting that lock.) Some typical locks are:

```
1,1
1,2
4,0
100,100
```

The numbers you choose are up to you. You may choose to assign some meaning to the numbers (for example, the first number might be the file-channel number of the file you want to lock, and the second number might be the number of the record within that file that you want to lock.)

Since both numbers in the lock may range from 1 to 65535, the actual possible number of unique locks is 65535 * 65535. But, every time you create a lock, the system sets aside a block in the monitor queue in system memory for that lock. Since there are initially only 20 queue blocks available, it's a good idea to keep the number of locks to a minimum. A good rule is that a program should not have more than two or three locks active at any one time. As you clear a lock, that queue block becomes

available again. (So, in essence, every time you set a lock you create it, and every time you clear a lock, you destroy it.)

3.0 THE MODES

The MODE argument in the XLOCK call line can contain one of four values (0-3) which selects one of the four possible locking modes:

3.1 MODE 0 (Lock and Return)

This mode tells XLOCK to create a lock with the value LOCK1,LOCK2. If the lock already exists (i.e., some other job is accessing the file or device you want to use), XLOCK returns with MODE equal to the number of the job that set the lock. (A job number is assigned to each job in the order that the jobs were defined in the JOBS command in the system initialization command file. For example, the first job defined in the JOBS command line is Job #1. The SYSTAT command lists the jobs in this order.) If the lock does not already exist, XLOCK creates it and returns with a zero in MODE. You've now set the lock.

3.2 MODE 1 (Lock and Wait)

This XLOCK mode is identical to MODE 0, except that if the lock already exists, XLOCK tells the system to put your job to sleep until the lock is cleared. That means that your job will be in an inactive state (except for waking at every clock tick to test the status of the lock) until the job that originally set the lock clears it. If you use this mode, take into consideration the fact that another user may be waiting for the same lock; it's possible that the lock might be cleared and then grabbed up either by the same or another job before your job wakes up.

3.3 MODE 2 (Clear Lock)

XLOCK clears the lock specified by LOCK1 and LOCK2 and returns to your program. A zero returned in MODE indicates that the lock you tried to clear wasn't set by your job; a one returned indicates that you sucessfully cleared one lock; a number greater than one indicates that you cleared more than one lock (which means that LOCK1 or LOCK2 were originally set to zero--the wildcard value). You may never use XLOCK to clear a lock that was not set by your job. (NOTE: If you attach your terminal to another job, XLOCK considers you a new job.)

3.4 MODE 3 (List Locks)

MODE 3 returns a complete list of all the locks set on the system and the numbers of the jobs that set them. When you use MODE 3, LOCK2 must represent a mapped array large enough to hold the expected data. When XLOCK returns from a MODE 3 call, MODE contains the number of locks that are set on the system, LOCK1 contains your job number, LOCK2 contains one three-word entry for each lock that is set on the system. (You must set up this entry as three binary words in a MAP statement.) The first two bytes hold the job number; the second and third words hold the actual LOCK1 and LOCK2 values of the specified lock. The following is an example of how to set up the MAP statement for a MODE 3 call:

```
10 MAP1  MODE, B, 2
20 MAP1  MYJOB, B, 2
30 MAP1  LISTARRAY
40   MAP2  LOCKENTRY(25)
50     MAP3  JOBNUMBER, B, 2
60     MAP3  LOCK1, B, 2
70     MAP3  LOCK2, B, 2
80      ! Start of Program goes here
100 MODE = 3
110 XCALL XLOCK, MODE, MYJOB, LISTARRAY
120     ! Rest of program goes here
```

4.0 WILDCARDS

A system lock consists of two numbers, the values of LOCK1 and LOCK2. If either of these two numbers is a zero, that number is a wildcard and any number between 1 and 65535 will match it. (A wildcard is a symbol that is matched by any other symbol.)

You can use wildcards for various reasons. For example, suppose that you decide that the LOCK1 value is going to represent a particular file and that the LOCK2 value will represent a particular record in that file. If you want to stop all references to that file while your program is accessing it, you would set the lock with a zero in LOCK2 and the number representing your file in LOCK1. Anyone who tries to set a lock that has the same LOCK1 value as your lock won't be able to do so; the system will tell him that that lock already exists (since your wildcard in LOCK2 will match any number he may try in that position). No one (including yourself) will be able to set a lock with the same LOCK1 value until you clear the lock. Note that setting a lock with both numbers zero will prevent anyone from setting a lock, since the system will say that all possible locks are already set.

5.0 PROGRAMMING EXAMPLES

The following is a small sample demonstration program that you may want to use to experiment with XLOCK, and to get a feeling for how it works. It asks you for the values of MODE, LOCK1, and LOCK2, and then reports back on the results of the locking operation you asked for. Remember: MODE = 0 sets a lock, MODE = 1 sets the lock after waiting for it to be cleared; MODE = 2 clears the lock, and MODE = 3 displays the locks set.

```
 10  ! Sample Program to Illustrate File Locking
 15  MAP1  COUNTER, F
 20  MAP1  MODE, B, 2
 25  MAP1  LOCK1, B, 2
 30  MAP1  LOCK2, B, 2
 35  MAP1  LOCKARRAY
 40   MAP2  LOCKENTRY(25)
 45     MAP3  JOB, B, 2
 50     MAP3  L1, B, 2
 55     MAP3  L2, B, 2
 60  START:
 65      INPUT "MODE, LOCK1, LOCK2: ",MODE,LOCK1,LOCK2
 70      FLAG = MODE
 75      IF MODE = 3 GOTO DISPLAY
 80      XCALL XLOCK, MODE, LOCK1, LOCK2
 85      PRINT "Mode = ";MODE :
 90      IF FLAG = 0 AND MODE <> 0 PRINT "Lock already set."
 95      IF FLAG = 2 AND MODE = 0 PRINT "You didn't set that lock."
100      IF FLAG = 2 AND MODE = 1 PRINT "You cleared the lock."
105      IF FLAG = 2 AND MODE > 1 PRINT "You cleared more than one lock."
110      GOTO START
115 DISPLAY:
120      XCALL XLOCK, MODE, LOCK1, LOCKARRAY
125      PRINT "Your job number is: ";LOCK1
130      PRINT "Current locks in use = ";MODE
135      IF MODE = 0 GOTO LOOP
140      FOR COUNTER = 1 TO MODE
145        PRINT SPACE(5);
150        PRINT STR(L1(COUNTER))+","+STR(L2(COUNTER));
155        PRINT SPACE(4) : PRINT "(Job";JOB(COUNTER);")"
160      NEXT
165 LOOP:
170      PRINT : GOTO START
```

XLOCK is often used to lock individual records within a file so that more than one user can update that file at the same time. LOCK1 might contain a number that represents the particular file you want to open for mulit-user updating (perhaps by containing the file's file-channel number). LOCK2 might hold a number that represents the specific record within the file that you want to update.

5.1 Calculating Record Numbers

We assume that you will usually be using XLOCK to control multi-user updating of random files. (For information on random files, see Chapter 15 of the AlphaBASIC User's Manual, (DWM-00100-01).) If you are going to be locking a specific file record, you need to understand the relationship between disk blocks and file records. A record (sometimes called a "logical record") is a grouping of data that you define; you also define the length of that record. Just as an example, let's define a file record that contains 6 bytes for a customer ID number, 24 bytes for a customer name, 10 bytes for the name of the customer's sales contact, and 10 bytes for the customer phone number. This file record would then contain 50 bytes. A disk block is a physical grouping of data on the disk that is always 512 bytes long. AMOS always transfers disk information in this 512-byte block. BASIC unblocks a disk block into smaller groups-- your logical records. For example, one disk block (512 bytes) would contain 100 of the logical records we defined above (50 * 100 = 500) with 12 bytes left over. No logical record is ever larger than a disk block. NOTE: You specify the size of your logical record in the OPEN statement for the file.

The reason for our explanation above is this: if you want the LOCK2 value to contain the number of the record you are updating, it must contain the relative number of the disk block being used, and not the logical record number. When BASIC unblocks a disk block into logical records, it brings the entire disk block into your memory partition. Even if you are only updating one logical record in that disk block, the entire disk block remains in your memory area until you either close the file or read a logical record that is in a different disk block. What this means is that more than one user could try to write out the same disk block at the same time even though they are updating different logical records. So, you must prevent access, not only to the logical record that you are updating, but to the entire disk block that contains it.

You must calculate the relative disk block number yourself by dividing the logical record number by the blocking factor. (The blocking factor is the number of logical records that can fit in one disk block.) In the example above where we had logical records 50 bytes long, the blocking factor is 10. Remember that each disk block is 512 bytes long and will be blocked to contain as many logical records as will fit.

If one of your lock digits is the disk block number, you can prevent access to the entire disk block; no one can access any of the logical records in the disk block until you clear the lock.

REMEMBER: The lock wildcard symbol is a zero, so calculate your disk blocks beginning with one instead of zero. Before you unlock the lock on a disk block, force the system to write that record by reading a logical record that falls outside of that disk block. (NOTE: You may also use the RANDOM'FORCED mode in your OPEN statement to force BASIC to perform a disk read or a disk write every time you access the file. See Chapter 15 of the AlphaBASIC User's Manual for more information.) The sample program below may help to clarify the last few paragraphs.

5.2 Sample Program to Illustrate File Record Locking

```
10      ! Sample Program to Illustrate File Record Locking
15      ! Remember to load XLOCK.SBR before running!
20  MAP1  MODE, B, 2                          ! Define locking variables
25  MAP1  LOCK1, B, 2
30  MAP1  LOCK2, B, 2
35  MAP1  LOGICAL'RECORD                      ! Define logical record
40    MAP2  CUST'ID,F,6                       ! contents-- 50 bytes
45    MAP2  CUSTOMER,S,24                     ! of customer info.
50    MAP2  CONTACT,S,10                      ! Customer ID is actually
55    MAP2  PHONE,S,10                        ! logical record number.
60  MAP1 RECORD'SIZE,F,,50
65      ! Scratch variables:
70  MAP1  RECORDNUM,F                         ! Logical record number
75  MAP1  FLAG,F
80  MAP1  QUERY,S,1
85      ! Begin program:
100 START:
105     LOOKUP "CUSTID.DAT",FLAG              ! If file doesn't exist,
110     IF FLAG = 0 THEN GOTO FILE'ERR        !   report error and exit.
115     OPEN #100, "CUSTID.DAT",RANDOM,RECORD'SIZE,RECORDNUM
120     LOCK1 = 100                           ! "100" represents CUSTID file
125     PRINT "Welcome to the Customer Maintenance Program."
130 LOOK:
135     INPUT "Please enter customer identification number: ",RECORDNUM
140        ! Note: Customer ID is just number of that logical record.
145        ! Calculate relative disk block number (assumes logical
150        ! records begin with zero):
155     LOCK2 = (RECORDNUM/10)+1
160        ! Lock the disk block used by the record.
165     XCALL XLOCK,MODE,LOCK1,LOCK2
170     READ #100, LOGICAL'RECORD
175     PRINT "Customer information:"
180     PRINT TAB(5);"Customer ID#: ";CUST'ID
185     PRINT TAB(5);"Customer name: ";CUSTOMER
190     PRINT TAB(5);"Sales contact: ";CONTACT
195     PRINT TAB(5);"Phone #: ";PHONE
200 UPDATE:
205     INPUT "Do you wish to change any info? ";QUERY
210     IF UCS(QUERY) = "N" THEN GOTO LOOP
215     PRINT "Customer ID: ",CUST'ID
220     INPUT "Enter customer name: ";CUSTOMER
225     INPUT "Enter sales contact: ";CONTACT
230     INPUT "Enter phone number: ";PHONE
235     WRITE #100, LOGICAL'RECORD
240        ! Force BASIC to bring different disk block into memory.
245        ! (If we are in first disk block, since blocking factor is
250        ! 10, record number >= 10 will force in next disk block)
255     IF LOCK2 = 1 THEN RECORDNUM = 10 ELSE RECORDNUM = 0
260        ! Now bring in different disk block:
265     READ #100, LOGICAL'RECORD
```

```
270         ! Release the lock.
275       MODE = 2
280       XCALL XLOCK, MODE, LOCK1, LOCK2
285 LOOP:
290       INPUT "Do you wish to see info on another customer? ",QUERY
295       IF UCS(QUERY) = "Y" THEN GOTO LOOK
300 EXIT:
305       PRINT "Returning you to AMOS..."
310       CLOSE #100
315       END
320 FILE'ERR:    ! Oops. File didn't exist.
325       PRINT "File error.  Please see System Operator."
330       END
```

May 1980
Revision A01

XMOUNT - BASIC SUBROUTINE TO MOUNT A DISK

1.0 INTRODUCTION

XMOUNT is an assembly language routine that allows you to mount a disk from within a BASIC program without leaving BASIC. You should call it whenever you change a disk and your BASIC program is going to sort files or create new files on the newly changed disk. (You must always mount a disk after you've changed it and before you write to it; otherwise the system will think that the old disk is still in the drive. When it comes time to write information out to the new disk, the disk's bitmap will be wrong, and the system will try to write to the new disk as if it had the same areas free as the old one.) Besides bringing into memory the proper bitmap, XMOUNT also loads in the alternate track table, if any, for the specified device.

IMPORTANT NOTE: NEVER mount or unmount a disk while someone is accessing that disk.

The XMOUNT program is fully re-entrant, so you may load it into system memory via the SYSTEM command in your SYSTEM.INI. (See The System Initialization Command File in the "System Operator's Information" section of the AMOS Software Update documentation packet for information on the SYSTEM.INI.)

1.1 THE XMOUNT SUBROUTINE

You can call XMOUNT to mount a disk via:

```
XCALL XMOUNT,DEV,VOLID$
```

Where:

DEV — String variable that represents a device specification (e.g., "DSK1:"). You may optionally follow the device specification with "/U" to unmount the device (e.g., "DSK0:/U").

VOLID$ — String variable in which the volume ID of the mounted device will be returned. This variable must be 10 bytes long. If it is not specified the labels block will not be read. This variable is ignored if the /U option is used.

If you specify the unmount option, the "U" must be uppercase. When you unmount a disk, you prevent BASIC and most system programs from accessing that device.

(Changed 1 May 1980)

SOFTWARE DOCUMENTATION READER'S COMMENTS

We appreciate your help in evaluating our documentation efforts. Please feel free to attach additional comments. If you require a written response, check here: ☐

NOTE: This form is for comments on software documentation only. To submit reports on software problems, use Software Performance Reports (SPRs), available from Alpha Micro.

Please comment on the usefulness, organization, and clarity of this manual:

Did you find errors in this manual? If so, please specify the error and the number of the page on which it occurred.

What kinds of manuals would you like to see in the future?

Please indicate the type of reader that you represent (check all that apply):

- ☐ Alpha Micro Dealer or OEM
- ☐ Non-programmer, using Alpha Micro computer for:
  - ☐ Business applications
  - ☐ Education applications
  - ☐ Scientific applications
  - ☐ Other (please specify):

    _____
- ☐ Programmer:
  - ☐ Assembly language
  - ☐ Higher-level language
  - ☐ Experienced programmer
  - ☐ Little programming experience
  - ☐ Student
  - ☐ Other (please specify):

    _____

NAME: _____ DATE: _____

TITLE: _____ PHONE NUMBER: _____

ORGANIZATION: _____

ADDRESS: _____

CITY: _____ STATE: _____ ZIP OR COUNTRY: _____

FOLD

FOLD

PLACE
STAMP
HERE

CUT ALONG LINE

alpha
micro

17881 Sky Park North
Irvine, California
92714

ATTN: SOFTWARE DEPARTMENT

FOLD

FOLD