PLEXUS

Plexus Sys3 UNIX 3.0 Release Notice

98-40032.1 June 3, 1983

Plexus Sys3 UNIX 3.0 Release Notice

98-40032.1 June 3, 1983

PLEXUS COMPUTERS INC

2230 Martin Ave

Santa Clara, CA 95050

408/988-1755

Copyright 1983
Plexus Computers Inc, Santa Clara, CA

All rights reserved.

No part of this manual may be reproduced in any form
without written permission from the publisher.

Printed in the United States of America

CONTENTS

1. INTRODUCTION

Sys3 is the implementation of the UNIX SYSTEM III operating system by Plexus Computers Inc. This document, which accompanies Release 3.0 of Sys3 for the P/35 and P/60, is a collection of information about Sys3.

In this document, "SYSTEM III" refers to the stock software provided by Western Electric. "Sys3" refers to the equivalent software provided by Plexus. "V7" is the Version 7 equivalent of Plexus software.

This document has five chapters. The first chapter contains the introduction. The second describes in detail how to reload Sys3 Release 3.0; the third chapter tells how to install Sys3 Release 3.0 as an upgrade from 1.0 or 1.1 (Z8000 versions). Optional and site-dependent procedures are described in Chapter 4, and Chapter 5 gives general information about Sys3.

This manual was written to describe every ordinary case of Sys3 3.0 installation and reload, and not everything in this manual applies to every site. You probably won't need to use every chapter of this manual. See section 1.2 for guidelines.

1.1 Purpose

This document describes installation procedures for release 3.0 of Plexus Sys3. It is intended as a supplement to the Plexus User's Manual. This document also contains some usage and troubleshooting information.

If you encounter any problems with this software or documentation, please contact:

Marketing Technical Support
Plexus Computers Inc
2230 Martin Ave
Santa Clara, CA 95050
408/988-1755

1.2 How to Use This Manual

If you are receiving a new Plexus P/35 or P/60, Sys3 Release 3.0 will normally be already on disk, and you need only start up the system as described in section 2.3 (and in the Plexus User's Manual).

This document mainly contains procedures for reloading Sys3 (described in Chapter 2), and for moving from Sys3 1.0 or 1.1 (Z8000 versions) to Sys3 3.0 (MC68000 version).

Follow the reload procedures (Chapter 2) if

1. your system has a new disk and system software must all be reloaded; or

2. your system has experienced a catastrophic failure such that all the software is lost.

Follow the procedures outlined in Chapter 3 if you are moving from Sys3 Release 1.0 or 1.1 (Z8000 versions).

1.3 Release 3.0 Overview

Plexus UNIX Sys3 Release 3.0 consists of a Release Tape and this release document.

The Release Tape comprises at least 21 files. Files 0-19 are blocked at 1024 bytes per record; file 20 is blocked at 10240 bytes per record; file 21 through the end of the tape are blocked at 5120 bytes per record. Most of the tape files are copies of Release 3.0 standalone programs. These are for backup and emergency purposes, in case the disk copies of the standalones become inaccessible and you need to run the standalone programs from tape. File 20 has a dump of a standard release 3.0 system.

1.4 Differences Between Sys3 1.1 (Z8000) and 3.0 (MC68000)

Sys3 Release 3.0 (MC68000) differs from Sys3 Release 1.1 (Z8000) in the following ways:

1. The compiler (**cc**), assembler (**as**), debugger (**adb**), and linker (**ld**) deal with the MC68000 processor instead of the Z8000. When porting C programs, keep in mind that integers and pointers are 32 bits in the MC68000 and 16 bits in the Z8000.

2. Maximum user address space is 2 megabytes in this release. Split instruction and data (as on the Z8000) is no longer necessary.

3. Floating point format does not conform to the IEEE standard in this release.

1.5 Of Special Note for Sys3 3.0

The following are of special note for Sys3 3.0:

1. The file system format may change for compatibility with SYSTEM V.

2. Floating point format may change to IEEE floating point format in the next release.

3. All changes to the root file system should be monitored so that the next release can be brought up quickly and painlessly.

4. No considerations for software performance were taken for this release. The C compiler optimizer is not hooked up, nor is it completely reliable.

5. For the next release, you should be able to look forward to

- smaller object files and faster program execution;
- less UNIX kernel overhead;
- IEEE format floating point;
- a much improved C compiler and optimizer;
- larger (8 megabyte) user address space.

2. RELOADING SYS3

This chapter gives procedures for the basic steps required to reload Sys3. Remember to follow these reload procedures only if

1. your system has a new disk and system software must all be reloaded; or

2. your system has experienced a catastrophic failure such that all the software is lost.

The first section of this chapter is a checklist for reloading Sys3 3.0. Each subsequent section of this chapter corresponds to one item in the checklist.

If you are receiving a brand new Plexus system, you don't have to follow ANY procedures except start the system (section 2.3).

2.1 Checklist for Reloading Sys3

This section gives a checklist for reloading Sys3. Each step in this checklist is described in a separate subsection below.

1. Verify that the data on disk block 0 is correct by running **dconfig**.

2. Install the Release Tape.

3. Startup the system.

4. The following steps are optional:

   a. Set the console default baud rate.

   b. Enable the accounting package.

   c. Add additional disk and special files.

5. Go to multi-user state.

2.2 Run dconfig

Shut down your system, press reset, and obtain the primary boot prompt. Then run standalone **dconfig**; see the Plexus User's Manual for instructions. If you need to get the **dconfig** program off tape, you will have to mount a tape.

If you intend to use **uucp**, you may install the system node name using **dconfig** at this time.

2.3 Install the Release Tape

Follow these directions to load the system software onto a new disk. This procedure destroys any previous contents of the disk.

To load the tape, do the following:

1. Turn on system power. Press reset button.

2. Wait for "PLEXUS SELFTEST REV X.X COMPLETE" message. The system informs you about the disk and tape driver names in use on your system (e.g., pd, pt), tells you about the various boards (e.g., Ethernet, ICPs), and tells the memory size. Then the boot message appears. The boot message is "PLEXUS PRIMARY BOOT REV 2.0". After the boot message comes the ":" prompt.

3. The disks come preformatted from the factory. Only in the event of a major catastrophe will you be required to reformat the disks. See the Plexus User's Manual for instructions on how to do this if necessary.

4. Make a file system on the disk with **mkfs**. To do this, mount the Release Tape and respond as indicated in **bold** below. The file system size is given in 1024-byte blocks.

   NOTE: **mkfs** DESTROYS THE DATA ON THE DESIGNATED FILE SYSTEM, SO USE CAUTION!!!

   The sequence is

   ```
   : mkfs

   $$ mkfs /dev/dk1 18000 1 500
   isize = 4496
   m/n = 1 500
   ```

   When the **mkfs** finishes executing, the system prints the message "Exit 0" and re-executes the selftest. Thus you again see the "SELFTEST COMPLETE" message, initial information lines, and primary boot message.

5. Restore the file system onto the disk. Your response is in **bold**. The sequence is

   ```
   : restor

   $$ restor r /dev/dk1 +20
   Spacing forward 20 files on tape
   ```

The final remark from the restor program before it commences to restore the file system is

Last chance before scribbling on /dev/dk1.

Respond with a <return> when you are ready to restore the file system. To abort the process, hit the reset button. The restoration of the file system should take about 30 minutes for a P/60, 60 minutes for a P/35. Then the selftest is done again, and the boot prompt sequence reappears.

2.4 System Startup

Obtain the primary boot prompt, a colon (:), by pressing the reset key if necessary. Once you have the boot prompt, type <return>. The machine should respond

```
: /sys3
```

The following lines appear:

```
SYS3/x.x: sys3.yy
real mem = xxxxx bytes
avail mem = xxxxx bytes
sys3
single-user
#
```

This response includes the amount of memory in the system, and the amount available for user processes. Note that for this release, this number is wrong by one page. The "xx" after "SYS3/" is the release number of the software. The "yy" after "sys3." is the number of users supported by the software.

These messages indicate that

1. The bootstrap program has been executed from the processor PROM;
2. Plexus Sys3 has been loaded from the system disk; and
3. The Sys3 operating system is ready for use.

The system is now in "init state 1" and only the system console is active.

While in init state 1, examine the date and correct it if necessary. (See **date**(1)). Also while in init state 1, perform configuration procedures as described in the Plexus User's Manual. You may also want to run the program **fsck**(1) to check the integrity of the file system(s). See the Plexus User's Manual for more information.

2.5 Go to Init State 2 (Multi-user)

The last required step in reloading Sys3 is to go to multi-user state. Type

/etc/init 2

Messages reporting the startup of **cron**, the startup of **errdemon**, and the initialization of the Intelligent Communications Processors (ICPs) appear on the system console. The message "ICP software initialization complete" appears on /dev/tty0, /dev/tty8, /dev/tty16, /dev/tty24, and /dev/tty32. The last two messages are

multi-user
type ctrl-d

Respond by typing d with the control key depressed. The message:

login:

should appear on your console and on all active terminals.

3. UPGRADING TO SYS3 REL 3.0 FROM REL 1.0 or 1.1

If you currently have Sys3 1.1 on the Z8000, upgrading to Sys3 3.0 should be relatively easy. The files are, for the most part, identical in function, and many are identical in form. The following files may have been modified by your installation for Sys3 1.1, and will need to be modified similarly for Sys3 3.0:

/etc/rc
/etc/passwd
/etc/group
/etc/checklist
/etc/ttytype
/etc/inittab
/usr/lib/crontab

4. OTHER PROCEDURES

This chapter describes site-dependent and other procedures that may be necessary to complete the installation of Sys3. Included here are instructions for setting the console default baud rate and setting up for the commands **csh** and **vi**. See the Plexus User's Manual for detailed instructions on other procedures such as shutdown, adding line printers, and enabling accounting.

4.1 Setting the System Console Default Baud Rate

Setting the system console default baud rate is optional. You will want to do it if for any reason you don't want to use the default 9600 baud rate.

The Sys3 software does not need to use the processor board switches to determine the baud rate of the console or any other terminal. The baud rate can be set by software with the **getty**(8) command within the file /etc/inittab. On boot and selftest, however, the baud rate must be set to match the terminal. For details, please read about the **getty**(8) and **init**(8) commands in the Plexus Sys3 UNIX Programmer's Manual -- Vol 1.

The baud rate for a terminal, ttyXX, is taken from the processor switches only if a line like the following is put in the file **/etc/inittab**:

```
2:XX:c:/etc/getty ttyXX b
```

The `b' argument to **/etc/getty** directs it to get the baud rate from the processor switches.

4.2 Setting Up for csh and vi

The commands **csh** and **vi** require that you perform certain setup tasks. These are outlined below.

csh

Two files are provided along with **csh**: **login** and **cshrc**. These are examples of the **.login** and **.cshrc** files that set the working environment of the C-shell. See **csh**(1) for more information.

Users desiring to have the C-shell as their login shell must have their entries in **/etc/passwd** changed appropriately. The environment variable SHELL in the **.login** file should still be set to **/bin/sh**, however, because some Sys3 programs use this variable in determining which shell to use, and do not work correctly otherwise (see below).

The file **/etc/cshprofile** is a command file that implements several startup features: message of the day, mail notification, and news. It also sets the time zone (TZ) automatically. The default time zone is Pacific

Standard Time. If you are in a different time zone, the line `setenv TZ' in **/etc/cshprofile** must be modified to reflect this.

Csh checks the format of shell scripts before executing them. It uses the presence of a shell comment in line 1 (`#' followed by text) to distinguish a **csh** shell script from a Sys3 **sh** shell script: scripts having the comment are treated as C-shell scripts. C-shell scripts are executed by the C-shell; Sys3 shell scripts are executed by the Sys3 shell. When the C-shell encounters a Sys3 shell script, it automatically invokes the Sys3 shell to execute it. However, several stock Sys3 commands are Bourne (Sys3) shell scripts with first line comments, so they look like C-shell scripts to the C-shell. Sys3 shell scripts may contain commands unintelligible to the C-shell. so when **csh** encounters these, it becomes confused and aborts. Therefore, if you use the C-shell, you should set your environment variable SHELL to **/bin/sh.** This causes the C-shell to invoke the Bourne shell by default. If you really want the C-shell to execute your shell scripts, you can call them from another C-shell, either by executing another C-shell or from within the script.

The following files have been altered for use with the C-shell:

/usr/bin/greek
/usr/bin/man
/usr/bin/mancvt
/usr/bin/mm
/usr/bin/mmt
/usr/bin/mvt
/usr/bin/osdd
/usr/bin/sccsdiff
/usr/bin/spell
/usr/bin/typo
/usr/bin/uupick

vi

Vi requires two files: **/etc/termcap** and **/etc/ttytype.** The **termcap** file already exists in the right place. You must modify the file **/etc/ttytype** according to your terminal configuration. The environment variable TERM must also be set to the terminal type for each TTY port. See the **login** and **cshrc** files in **/usr/plx** for examples of how to do this. **Vi** has both a manual page and an associated document in the directory **/usr/man/docs/ex** as well as in the Plexus Sys3 UNIX Programmer's Manual -- vol 2C: see these for more information.

5. SYS3 GENERAL INFORMATION

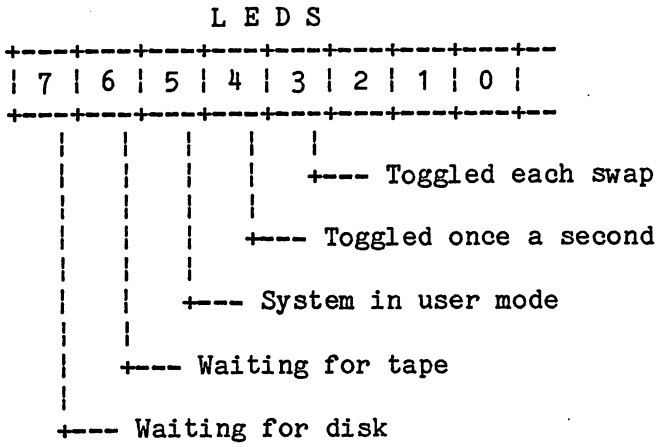
This chapter contains general information about Sys3, including disk blocking data and comparisons with V7 and SYSTEM III.

5.1 Sys3: 16, 32, or 40 Users

Sys3 comes in three versions: a 16-user system, a 32-user system, and a 40-user system. The operating system limits the number of logins to the maximum allowed, either 16, 32, or 40.

5.2 Processor Board LEDs

The LEDs on the processor board as shown in the the figure titled "Processor Board Option Selectors" of the Plexus User's Manual have the following meaning when Sys3 is running.



LEDs 0-2 are reserved.

5.3 Disk Blocking

Sys3 presents a logical file system blocked in 1024-byte blocks. This means that all disk blocks, including the super block, are 1024 bytes long, all blocked disk I/O is done with 1024-byte blocks, and commands that report or use block values assume the values are 1024-byte blocks. These commands include **acctdusg, acctdisk, du, df, find, /etc/fsck, ls -s, /etc/mkfs,** and **sum.**

Note that the blocking factor is independent of the physical size of a sector on the disk.

5.3.1 Disk Organization

The default mapping of minor disk device (/dev/dkX) number to physical sections of disks is as follows. You can override the default and define your own mapping via the **dconfig** program described in Chapter 4 of this document. The disk starting sector and size in sectors are given below. The numbers are in 512-byte sectors.

Device Number	Starting Sector	Size
/dev/dk0	0	1000000
/dev/dk1	0	40000 (last 4000 sectors are swap)
/dev/dk2	40000	to end of disk
/dev/dk3	60000	to end of disk
/dev/dk4	80000	to end of disk
:	:	:
/dev/dk15	300,000	:

sectorspp The amount of space available on the disk varies with different disks.

Users frequently want to create other file systems, in addition to the 18000K block root file system. To make a second file system spanning the rest of the disk, you need to run the **mkfs** (1) program. Type

mkfs /dev/dk2 nblks 1 500

where "nblks" is one of the following:

Size of /dev/dk2	
Disk	IMSP
22 Mbyte 8"	43
36 Mbyte 8"	13405
72 Mbyte 8"	47473
72 Mbyte 14"	48085
145 Mbyte 14"	116170

"IMSP" is the name of the P/35 and P/60 disk controller.

The **mknod** for **/dev/dk2** (which ordinarily must be done before the **mkfs**) is done by Plexus at the factory. Once **/dev/dk2** is made, you must do the standard **mkdir** and **mount** steps;

To create additional file systems (devices **/dev/dk3** and higher), you must perform **mknods** to create the logical disk devices for each new file system; then make each file system on its device using **mkfs**; then do the **mkdir** and **mount** steps.

5.3.2 Special Devices

Sys3 defines and uses the following special devices:

/dev/dk[0-2]	Disk, blocked I/O
/dev/rdk[0-2]	Disk, unblocked I/O
/dev/mt0	Magnetic media, blocked I/O
/dev/rmt0	Magnetic media, unblocked I/O
/dev/nrmt0	Magnetic media, unblocked, no rewind
/dev/tty[0-40]	Serial ports
/dev/lp	Line printer port
/dev/pp[0-4]	Parallel ports
/dev/console	System console
/dev/ic[0-4]	ICP download ports
/dev/swap	Swap device, used by ps(1) and a few other commands
/dev/mem	Memory
/dev/kmem	Kernel data space within memory
/dev/liomem	Special I/O ports on processor board
/dev/mbmem	System's bus memory address space
/dev/mbiomem	System's bus I/O address space
/dev/null	Bit bucket
/dev/error	Provides access to error records in the kernel
/dev/prf	Provides access to activity information in the kernel

You may add additional disks as needed with the **mknod**(1) command.

5.4 Comparisons of Sys3 1.1 and Sys3 3.0

Sys3 3.0 is a full implementation of SYSTEM III with the exceptions noted below.

5.4.1 Plexus Additions to SYSTEM III

Sys3 has the following additions, which are not part of SYSTEM III.

/usr/plx/dumpdir	Lists contents of dump/restor tape.
/etc/openup	Enables efficient access to key files.
/usr/plx/arcv6	Converts **ar** (1) files from UNIX Version 6 format to Sys3 format.
/usr/plx/tape	Efficient tape manipulation program.

ms	Document preparation macros available in Version 7.
File locking	Allows a program exclusive access to a file. See **locking**(2).
date	The clock is battery powered. Must be reset only if the processor board is removed.

Sys3 also includes many of the programs that were on the Sys3 Release 1.0 Addenda tape, including those based on commands from the University of California at Berkeley, Rev. 2.0, such as **csh** and **vi**.

5.4.2 Incompatibilities between V7 and Sys3

Here is a partial list of incompatibilities between Sys3 and V7 software. All incompatibilities but the programs in **/stand** are also part of SYSTEM III.

basename	"basename /f/abcde xe" prints nothing; V7 printed last member.
chdir	Replaced by cd.
date	Argument format is different.
du	Block total includes indirect blocks; V7 did not.
echo	"-n" no longer recognized. Replaced by \c.
kill	"1" arguments no longer kill all active processes.
ld	No longer recognizes SYMDEF symbol table in archive.
login:root	Default PATH variable does not include "." for root and su. Therefore, to execute files in the current directory, user must type ./<file>.
mount	Warns you if you have an incorrectly labeled or unlabeled file system. Label with **/etc/labelit**.
open(2)	Does not accept null length file.
overlays	A program can no longer overlay part of its code.
ps	Some of the flags have changed, notably the "ax" flags.
/usr/dict	Directory and files no longer exist.
su	See login:root.

sum — The summing algorithm has changed. The -r option uses the V7 algorithm, but gets a different second word.

umask — Set such that created files are 666 and dirs 777. umask = 0.

uucp — The node name is now embedded within the operating system, but may be changed via **dconfig**. See **uname** (1). In V7, the node name was defined by the file **/usr/lib/uucp/L.sysname**.

5.4.3 Known Bugs in Stock SYSTEM III

The following are known bugs in the standard SYSTEM III UNIX.

calendar — Does not recognize "today" and "tomorrow" as advertised.

checkcw — When given many file arguments, it may not be able to open some of them.

/etc/cron — Goes awry when date changed or is wrong.

csplit — csplit does not always create files of the proper number of lines if given a file with an excessively long line.

dd — Does not swap bytes (conv=swab) for small block sizes such as bs=2.

tp — Cannot write a tape with r option.

5.4.4 Known Deficiencies in Plexus Sys3

The following are known deficiencies in Plexus Sys3.

adb — Running a program may change some of the terminal modes.

cref — "-a" flag: does not recognize comments /* ... */ as comments; considers instructions, condition codes as symbols.

"-c" flag: unsigned, int, etc. treated as symbols.

ps — Will not work with -c option.

stty — Raw mode (**stty raw**) exists for compatibility with V7 and works as documented. It clears all flags and sets CS8. However, because there is not a one to one correspondence between **stty** settings in V7 and Sys3, the command **stty -raw** does not simply reverse the effects of a **stty raw**. **stty -raw** sets BRKINT, IGNPAR, ISTRIP, IXON, OPOST, ISIG, CS7, INPCK, and ICANON; it also resets EOF and EOL and clears PARENB.

sysdef	Not implemented.
timex	The values for disk0, disk1, and disk2 are not to be believed.
/etc/volcopy	Prints erroneous info when doing copy disk to tape and told 2400 feet, 1600 bpi.
graphics	Most SYSTEM III graphics capabilities are not yet part of the Plexus Sys3 release.

5.4.5 Known Problems with Sys3 3.0

The following are known problems with Plexus Sys3 3.0;

structures	Structures with bit fields can cause bad code from the C compiler. Filling out fields to 32 bits seems to get around this problem.
documents	Aside from this release notice and the attached new manual pages, no new hardcopy documentation is provided for Sys3 3.0.
admin -i	When used with the "-a" option causes an error. Use "-i" and "-a" separately to get around this.
pack, pcat, unpack	Cause unpacking error.
reform +s	The "+s" option causes an error message when used on a non-SCCS or empty file.
sact	Does not work when file is non-SCCS or non-existent.
division by 0	Works differently from on the Z8000. The MC68000 traps it, causing a floating point error signal.
uucp	Does not work.
profiling	Kernel profiling does not work.
efl	Not available.
gettytab	Not released.
indirect system call	Not supported.
uts	Can't relink kernel.
games	Do not work.
/etc/rc	For 40-user system, you need to add a line to /etc/rc to inform you about downloading the fifth ICP during

the boot process.

5.4.6 Not Provided - Not Applicable

The following list defines programs, libraries and other software that are not provided because they are:

1. specific to non-PLEXUS hardware; or

2. replaced by equivalent software.

/bin/kas	/stand/boot2	/usr/lib/lib2A.a
/bin/kasb	/stand/iltd	/usr/lib/lib2B.a
/bin/kun	/stand/rf11boot	/usr/man/man1/kas.1
/bin/kunb	/stand/rk11boot	/usr/mdec
/etc/fscv	/stand/rl11boot	/usr/mdec/copy
/etc/stcntrl	/stand/rp03boot	/usr/mdec/dldr
/etc/stload	/stand/rp04boot	/usr/mdec/iltd
/etc/stproto	/stand/rs04boot	/usr/mdec/list
/etc/vlx	/unixhpht	/usr/mdec/mboot
/lib/as2	/unixhptm	/usr/mdec/rf11booti
/lib/c0	/unixrktm	/usr/mdec/rk11boot
/lib/c1	/unixrlht	/usr/mdec/rkf
/lib/fc0	/unixrltm	/usr/mdec/rp03boot
/lib/fc1	/unixrpht	/usr/mdec/rp04boot
/lib/fcrt0.o	/unixrptm	/usr/mdec/rs04boot
/lib/fmcrt0.o	/usr/bin/sdb	/usr/mdec/tapeboot
/stand/boot1	/usr/include/sys.s	/usr/mdec/tboot

5.4.7 Not Provided - No Source

The following commands are documented in SYSTEM III but were omitted from the SYSTEM III source:

/bin/primes
/bin/factor

5.4.8 Not Provided - Not Yet Available

The following are not provided in this release but will likely be provided in a future one.

/etc/config
/etc/crash
/usr/bin/efl
/usr/bin/f77
/usr/lib/f77pass1
/usr/lib/libF77.a
/usr/lib/libI77.a

6. SYS3 3.0 MANUAL PAGES

This section contains new manual pages for Sys3 3.0 commands that differ significantly from Sys3 1.1 versions. These include **a.out**(5), **adb**(1), **cc**(1), and **ld**(1). Also included is a new page for the MC68000 assembler **as**(1). These pages are in Volume 1 format so you can insert these into your current manuals.

NAME

a.out - assembler and link editor output

DESCRIPTION

A.out is the output file of the assembler as and the link editor ld. Both programs will make **a.out** executable if there were no errors in assembling or linking, and no unresolved external references.

This file has four sections: a header, the program text and data segments, relocation information, and a symbol table (in that order). The last two sections may be missing if the program was linked with the **-s** option of ld(1) or if the symbol table and relocation bits were removed by strip(1). Also note that if there were no unresolved external references after linking, the relocation information will be removed.

The sizes of each segment (contained in the header, discussed below) are in bytes and are even. The size of the header is not included in any of the other sizes.

When an **a.out** file is loaded into memory for execution, three logical segments are set up: the text segment, the data segment (initialized data followed by uninitialized, the latter actually being initialized to all 0's), and a stack. The text segment begins at location 0 in the core image; the header is not loaded. If the magic number (the first field in the header) is 107 (hexadecimal), it indicates that the text segment is not to be write-protected or shared, so the data segment will be contiguous with the text segment. If the magic number is 108 (hexadecimal), the data segment begins at the first 0 mod 2K byte boundary (Z8000) or the first 0 mod 4K byte boundary (MC68000) following the text segment, and the text segment is not writable by the program; if other processes are executing the same **a.out** file, they will share a single text segment. For the Z8000 only, if the magic number is 109 (hexadecimal), the text segment is again pure (write-protected and shared); moreover, the instruction and data spaces are separated. The text and data segment both begin at location 0. See the Zilog Z8000 Instruction Manual for restrictions that apply to this situation.

The stack will occupy the highest possible locations in the core image: on the Z8000, from FFFE (hexadecimal) and growing downwards; on the MC68000, from 1FFFFC and growing downwards. The stack is automatically extended as required. The data segment is only extended as requested by the brk(2) system call.

The start of the text segment in the **a.out** file is hsize;

the start of the data segment is hsize+St (the size of the text), where hsize is 10 (hexadecimal).

The value of a word in the text or data portions that is not a reference to an undefined external symbol is exactly the value that will appear in memory when the file is executed. If a word in the text or data portion involves a reference to an undefined external symbol, as indicated by the relocation information (discussed below) for that word, then the value of the word as stored in the file is an offset from the associated external symbol. When the file is processed by the link editor and the external symbol becomes defined, the value of the symbol will be added to the word in the file.

Header-Z8000

The format of the **a.out** header is as follows:

```
struct      exec  {
        short  a_magic;     /* magic number */
        unsigned     a_text;/* size of text segment */
        unsigned     a_data;/* size of data segment */
        unsigned     a_bss; /* size of bss segment */
        unsigned     a_syms;/* size of symbol table */
        unsigned     a_entry;/* entry point of program */
        unsigned     a_stamp;/* version stamp */
        unsigned     a_flag;/* set if relocation info stripped */
};
```

Header-MC68000

The format of the header on the MC68000 is as follows:

```
struct      bhdr  {
        long   fmagic;      /* magic number */
        long   tsize;       /* size of text segment */
        long   dsize;       /* size of data segment */
        long   bsize;       /* size of bss segment */
        long   ssize;       /* size of symbol table */
        long   rtsize;      /* size of text relocation info */
        long   rdsize;      /* size of data relocation info */
        long   entry;       /* entry point of program */
};
```

Relocation-Z8000

If relocation information is present, it amounts to two bytes per relocatable datum. There is no relocation information if the ``suppress relocation'' flag (a_flag) in the header is on.

The format of the relocation data is:

```
struct    r_info      {
     int  r_symbolnum:11;
          r_segment:3;
          r_pcrel:1;
};
```

The r_pcrel field is not used.

The r_segment field indicates the segment referred to by the text or data word associated with the relocation word:

00 indicates the reference is absolute;
02 indicates the reference is to the text segment;
04 indicates the reference is to initialized data;
06 indicates the reference is to bss (uninitialized data);
10 indicates the reference is to an undefined external symbol.

The field r_symbolnum contains a symbol number in the case of external references, and is unused otherwise. The first symbol is numbered 0, the second 1, etc.

The start of the relocation information is

hsize + a_text + a_data

Relocation-MC68000

Relocation information, if it is present, is given for each datum to be relocated.

The format of the relocation information is:

```
struct    reloc       {
     unsigned rsegment:2;/* RTEXT, RDATA, RBSS, or REXTERN */
     unsigned rsize:2;          /* RBYTE, RWORD, or RLONG */
     unsigned rdisp:1;          /* 1 => a displacement */
     unsigned relpad1:3;/* unused portion of relocation tag */
     char relpad2;    /* unused portion of relocation tag */
     short rsymbol;   /* id of the symbol of external relocations */
     long rpos;       /* position of relocation in segment */
};
```

The rsegment field indicates the segment referred to by the

relocated datum.

00 indicates the reference is to the text segment;
01 indicates the reference is to initialized data;
02 indicates the reference is to bss (uninitialized data);
03 indicates the reference is to an undefined external symbol.

The rsize field indicates the size of the datum:

00 indicates the datum is one byte;

01 indicates the datum is one word;

02 indicates the datum is a long.

The field rsymbol contains a symbol number in the case of external references. The first symbol is numbered 0, the second 1, etc. The start of the text relocation information is

tsize + dsize + ssize

The start of the data relocation information is

hsize + tsize + dsize + ssize + rtsize

Symbol Table-Z8000

The symbol table on the Z8000 consists of entries of the form:

```
struct      nlist     {
     char   n_name[8];
     int    n_type;
     unsigned         n_value;
};
```

The n_name field contains the ASCII name of the symbol, null-padded. The n_type field indicates the type of the symbol; the following values are possible:

```
000  undefined symbol
001  absolute symbol
002  text segment symbol
003  data segment symbol
004  bss segment symbol
037  file name symbol (produced by ld)
040  undefined external symbol
041  absolute external symbol
042  text segment external symbol
043  data segment external symbol
044  bss segment external symbol
```

The start of the symbol table on the Z8000 is:

hsize+2(a_text+a_data)

if relocation information is present, and

hsize+a_text+a_data

if it is not.

If a symbol's type is undefined external and the value field is non-zero, the symbol is interpreted by the link editor ld(1) as the name of a common region whose size is indicated by the value of the symbol.

Symbol Table-MC68000

The symbol table on the MC68000 consists of entries of the form:

```
struct      sym       {
     char   stype;              /* symbol type */
     char   sympad;             /* pad to long align */
     long   svalue;             /* value */
};
```

The symbol follows each entry and is null-terminated. The stype field indicates the type of the symbol; the following values are possible:

```
000     undefined symbol
001     absolute symbol
002     text segment symbol
003     data segment symbol
004     bss segment symbol
037     file name symbol (produced by ld)
024     register name
040     external bit or'd in
"%08x"  format for printing a value
```

The start of the symbol table on the MC68000 is

hsize + tsize + dsize

If a symbol's type is undefined external and the value field is non-zero, the symbol is interpreted by the link editor ld(1) as the name of a common region whose size is indicated by the value of the symbol.

SEE ALSO

as(1), ld(1), nm(1), strip(1).

NAME

adb - debugger

SYNOPSIS

adb [-w] [objfil [corfil]]

DESCRIPTION

Adb is a general purpose debugging program. It may be used to examine files and to provide a controlled environment for the execution of UNIX programs.

Objfil is normally an executable program file, preferably containing a symbol table; if not then the symbolic features of adb cannot be used although the file can still be examined. The default for objfil is **a.out**. Corfil is assumed to be a core image file produced after executing objfil; the default for corfil is **core**.

Requests to adb are read from the standard input and responses are to the standard output. If the **-w** flag is present then both objfil and corfil are created if necessary and opened for reading and writing so that files can be modified using adb. Adb ignores QUIT; INTERRUPT causes return to the next adb command.

In general requests to adb are of the form

[address] [, count] [command] [;]

If address is present then dot is set to address. Initially dot is set to 0. For most commands count specifies how many times the command will be executed. The default count is 1. Address and count are expressions.

The interpretation of an address depends on the context it is used in. If a subprocess is being debugged then addresses are interpreted in the usual way in the address space of the subprocess. For further details of address mapping see ADDRESSES.

EXPRESSIONS

. The value of dot.

+ The value of dot incremented by the current increment.

^ The value of dot decremented by the current increment.

" The last address typed.

integer An octal number if integer begins with a 0; a

hexadecimal number if preceded by #; otherwise a decimal number.

integer.fraction
A 32 bit floating point number.

'cccc' The ASCII value of up to 4 characters. \ may be used to escape a '.

< name The value of name, which is either a variable name or a register name. Adb maintains a number of variables (see VARIABLES) named by single letters or digits. If name is a register name then the value of the register is obtained from the system header in corfil. The register names are r0 ... r15 fcw seg pc for the Z8000, d0 ... d7 a0 ... a7 ps pc for the MC68000.

symbol A symbol is a sequence of upper or lower case letters, underscores or digits, not starting with a digit. The value of the symbol is taken from the symbol table in objfil. An initial _ or ~ will be prepended to symbol if needed.

_ symbol In C, the ``true name'' of an external symbol begins with _. It may be necessary to utter this name to distinguish it from internal or hidden variables of a program.

routine.name
The address of the variable name in the specified C routine. Both routine and name are symbols. If name is omitted the value is the address of the most recently activated C stack frame corresponding to routine. (Not implemented in the MC68000.)

(exp) The value of the expression exp.

Monadic operators:

*exp The contents of the location addressed by exp in corfil.

@exp The contents of the location addressed by exp in objfil.

-exp Integer negation.

~exp Bitwise complement.

Dyadic operators are left associative and are less binding than monadic operators.

e1+e2 Integer addition.

e1-e2 Integer subtraction.

e1*e2 Integer multiplication.

e1%e2 Integer division.

e1&e2 Bitwise conjunction.

e1|e2 Bitwise disjunction.

e1#e2 E1 rounded up to the next multiple of e2.

COMMANDS

Most commands consist of a verb followed by a modifier or list of modifiers. The following verbs are available. (The commands ? and / may be followed by *; see ADDRESSES for further details.)

?f Locations starting at address in objfil are printed according to the format f. dot is incremented by the sum of the increments for each format letter (q.v.).

/f Locations starting at address in corfil are printed according to the format f and dot is incremented as for ?.

=f The value of address itself is printed in the styles indicated by the format f. (For i format ? is printed for the parts of the instruction that reference subsequent words.)

A format consists of one or more characters that specify a style of printing. Each format character may be preceded by a decimal integer that is a repeat count for the format character. While stepping through a format dot is incremented by the amount given for each format letter. If no format is given then the last format is used. The format letters available are as follows:

o 2 Print 2 bytes in octal. All octal numbers output by adb are preceded by 0.
O 4 Print 4 bytes in octal.
q 2 Print in signed octal.
Q 4 Print long signed octal.
d 2 Print in decimal.
D 4 Print long decimal.
x 2 Print 2 bytes in hexadecimal.
X 4 Print 4 bytes in hexadecimal.
u 2 Print as an unsigned decimal number.
U 4 Print long unsigned decimal.

f 4 Print the 32 bit value as a floating point number.
F 8 Print double floating point.
b 1 Print the addressed byte in octal.
c 1 Print the addressed character.
C 1 Print the addressed character using the following escape convention. Character values 000 to 040 are printed as **@** followed by the corresponding character in the range 0100 to 0140. The character **@** is printed as **@@**.
s *n* Print the addressed characters until a zero character is reached.
S *n* Print a string using the **@** escape convention. *n* is the length of the string including its zero terminator.
Y 4 Print 4 bytes in date format (see *ctime*(3C)).
i n Print as Z8000 instructions. *n* is the number of bytes occupied by the instruction. This style of printing causes variables 1 and 2 to be set to the offset parts of the source and destination respectively.
Z n Prints as Z8000 assembler listing. (Not available on the MC68000.)
a 0 Print the value of *dot* in symbolic form. Symbols are checked to ensure that they have an appropriate type as indicated below.

/ local or global data symbol
? local or global text symbol
= local or global absolute symbol

p 2 Print the addressed value in symbolic form using the same rules for symbol lookup as **a**.
t 0 When preceded by an integer tabs to the next appropriate tab stop. For example, **8t** moves to the next 8-space tab stop.
r 0 Print a space.
n 0 Print a new-line.
"..." 0
Print the enclosed string.
^ *Dot* is decremented by the current increment. Nothing is printed.
+ *Dot* is incremented by 1. Nothing is printed.
- *Dot* is decremented by 1. Nothing is printed.

new-line
Repeat the previous command with a *count* of 1.

[?/]**l** *value mask*
Words starting at *dot* are masked with *mask* and compared with *value* until a match is found. If **L** is used then the match is for 4 bytes at a time instead of 2. If no match is found then *dot* is unchanged; otherwise *dot* is

set to the matched location. If mask is omitted then -1 is used.

[?/]w value ...
Write the 2-byte value into the addressed location. If the command is W, write 4 bytes. Odd addresses are not allowed when writing to the subprocess address space.

[?/]m b1 e1 f1[?/]
New values for (b1, e1, f1) are recorded. If less than three expressions are given then the remaining map parameters are left unchanged. If the ? or / is followed by * then the second segment (b2,e2,f2) of the mapping is changed. If the list is terminated by ? or / then the file (objfil or corfil respectively) is used for subsequent requests. (So that, for example, /m? will cause / to refer to objfil.)

>name
Dot is assigned to the variable or register named.

! A shell is called to read the rest of the line following !.

$modifier
Miscellaneous commands. The available modifiers are:

- <f Read commands from the file f and return.
- >f Send output to the file f, which is created if it does not exist.
- r Print the general registers and the instruction addressed by pc. Dot is set to pc.
- f Print the registers as double precision values.
- b Print all breakpoints and their associated counts and commands.
- c C stack backtrace. If address is given then it is taken as the address of the current frame (instead of r14 (Z8000) or fp (MC68000). If C is used then the names and (16 bit) values of all automatic and static variables are printed for each active function (Z8000 only). If count is given then only the first count frames are printed.
- e The names and values of external variables are printed.
- w Set the page width for output to address (default 80).
- s Set the limit for symbol matches to address (default 255).
- o All integers input are regarded as octal.
- x Changes default output to hexadecimal.
- d Reset integer input as described in EXPRESSIONS.
- q Exit from adb.

v Print all non zero variables in octal.
m Print the address map.

:modifier
Manage a subprocess. Available modifiers are:

bc Set breakpoint at address. The breakpoint is executed count-1 times before causing a stop. Each time the breakpoint is encountered the command c is executed. If this command sets dot to zero then the breakpoint causes a stop.

d Delete breakpoint at address.

x Delete all breakpoints.

e If address is given, set breakpoint upon exit from the routine on stack whose address is address. If address is not given, set breakpoint upon exit from current routine. This is useful for looking at the values returned (in **r7** or **rr6** or **rg4**) by a procedure.

p Similar to **b**, except that address is assumed to be a procedure name, and the breakpoint is positioned after the stack frame has been set up.

q Does for **p** what **d** does for **b**.

r Run objfil as a subprocess. If address is given explicitly then the program is entered at this point; otherwise the program is entered at its standard entry point. count specifies how many breakpoints are to be ignored before stopping. Arguments to the subprocess may be supplied on the same line as the command. An argument starting with < or > causes the standard input or output to be established for the command. All signals are turned on on entry to the subprocess.

cs The subprocess is continued with signal s (see signal(2)). If address is given then the subprocess is continued at this address. If no signal is specified then the signal that caused the subprocess to stop is sent. Breakpoint skipping is the same as for **r**.

ss As for **c** except that the subprocess is single stepped count times. If there is no current subprocess then objfil is run as a subprocess as for **r**. In this case no signal can be sent; the remainder of the line is treated as arguments to

the subprocess.

k The current subprocess, if any, is terminated.

VARIABLES

Adb provides a number of variables. Named variables are set initially by adb but are not used subsequently. Numbered variables are reserved for communication as follows.

0 The last value printed.
1 The last offset part of an instruction source.
2 The previous value of variable 1.

On entry the following are set from the system header in the corfil. If corfil does not appear to be a **core** file then these values are set from objfil.

b The base address of the data segment.
d The data segment size.
e The entry point.
m The ``magic'' number (0405, 0407, 0410 or 0411).
s The stack segment size.
t The text segment size.

ADDRESSES

The address in a file associated with a written address is determined by a mapping associated with that file. Each mapping is represented by two triples (b1, e1, f1) and (b2, e2, f2) and the file address corresponding to a written address is calculated as follows:

b1<address<e1 => file address=address+f1-b1
otherwise

b2<address<e2 => file address=address+f2-b2,

otherwise, the requested address is not legal. In some cases (e.g. for programs with separated I and D space) the two segments for a file may overlap. If a **?** or **/** is followed by an ***** then only the second triple is used.

The initial setting of both mappings is suitable for normal **a.out** and **core** files. If either file is not of the kind expected then, for that file, b1 is set to 0, e1 is set to the maximum file size and f1 is set to 0; in this way the whole file can be examined with no address translation.

In order for adb to be used on large files all appropriate values are kept as signed 32 bit integers.

FILES

/dev/mem

/dev/swap
a.out
core

NOTES

Plexus adb differs from the standard SYSTEM III adb in the following ways.
The number of registers and certain register names have changed.
Format option **i** prints Z8000 or MC68000 instructions instead of PDP-11 instructions.
A new **z** format option is provided for disassembly (Z8000 only).
The **a** (ALGOL stack backtrace) modifier is not supported.
A new modifier **x** changes default output to hexadecimal.
Four new modifier options are provided: **e**, **p**, **q**, and **x** (**e** available on Z8000 only).

SEE ALSO

ptrace(2), a.out(5), core(5).

DIAGNOSTICS

``Adb'' when there is no current command or format. Comments about inaccessible files, syntax errors, abnormal termination of commands, etc. Exit status is 0, unless last command failed or returned nonzero status.

BUGS

A breakpoint set at the entry point is not effective on initial entry to the program.
Local variables whose names are the same as an external variable may foul up the accessing of the external.
The MC68000 disassembly (**i** option) is not always correct.

NAME

as - MC68000 assembler

SYNOPSIS

as [-g] [-o OBJFILE] [-e] [-l] FILE

DESCRIPTION

As assembles the concatenation of the named files. The options may be placed in any order.

The "-l" option causes a listing to be created in a file whose name has ".list" substituted for ".s".

The "-g" option causes undefined symbols to be made global.

The "-e" option causes only external symbols to be placed in the ".o" file.

The output of the assembler is by default placed on the file with ".o" substituted for ".s" in the current directory; the "-o" flag causes the output to be placed on the named file. If there were no unresolved external references, and no errors detected, the output file is marked executable; otherwise, if it is produced at all, it is made non-executable.

FILES

/usr/tmp/as*	temporary
a.out	object
xref	cross reference

SEE ALSO

ld(1), nm(1), adb(1), a.out(5)
UNIX MC68000 Assembler Reference Manual

DIAGNOSTICS

Diagnostics are meant to be self explanatory and are accompanied by either the offending file or the appropriate line number. If the listing option is used, then the error messages also are placed in the listing file.

NAME

as - Z8000 assembler

SYNOPSIS

as [-l] [-m] [-n] [-o OBJFILE] [-u] [-v] [-x] FILE

DESCRIPTION

As assembles the concatenation of the named files. The options may be placed in any order.

The "-l" option causes a listing to be created on the standard output.

The "-m" option places a tab at the left margin of each line when the listing option "-l" has been selected.

The "-n" option causes a listing to be created on the standard output, but causes a narrower listing than the "-l" option. This is sometimes useful for 80 column paper, but does not contain line numbers.

The "-u" option causes all undefined symbols in the current assembly to be made undefined-external.

The "-v" option puts the assembler in verbose mode, which causes the files to be listed during the first pass, along with the errors encountered.

The "-x" option causes a cross refererence to be created on the file named "xref". The cross reference consists of an alphabetical listing of all user symbols along with the value, line number, and line numbers for all references.

The output of the assembler is by default placed on the file "a.out" in the current directory; the "-o" flag causes the output to be placed on the named file. If there were no unresolved external references, and no errors detected, the output file is marked executable; otherwise, if it is produced at all, it is made non-executable.

FILES

/usr/tmp/as*	temporary
a.out	object
xref	cross reference

SEE ALSO

ld(1), nm(1), adb(1), a.out(5)
UNIX Z8000 Assembler Reference Manual by Craig C. Forney

DIAGNOSTICS

Diagnostics are meant to be self explanatory and are

accompanied by either the offending file or the appropriate line number. If the listing option is used, then the error messages also are placed in the listing file.

BUGS

Cross reference ("-x" option) is not implemented. Only a single file may be assembled at a time.

NAME

cc, pcc - C compiler

SYNOPSIS

cc [option] ... file ...
pcc [option] ... file ...
ncc [option] ... file ...

DESCRIPTION

Cc is the UNIX C compiler. Pcc is another name for cc. Ncc is a SYSTEM III-compatible version of the C compiler; its optimizer does a slightly better job. Ncc represents the latest version of cc and may not be available in some releases. These commands accept several types of arguments:

Arguments whose names end with **.c** are taken to be C source programs; they are compiled, and each object program is left on the file whose name is that of the source with **.o** substituted for **.c**. The **.o** file is normally deleted, however, if a single C program is compiled and loaded all at one go.

In the same way, arguments whose names end with **.s** are taken to be assembly source programs and are assembled, producing a **.o** file.

The following options are interpreted by cc and pcc. See ld(1) for link editor options.

-c Suppress the link edit phase of the compilation, and force an object file to be produced even if only one program is compiled.

-p Arrange for the compiler to produce code that counts the number of times each routine is called; also, if link editing takes place, replace the standard startoff routine by one that automatically calls monitor(3C) at the start and arranges to write out a **mon.out** file at normal termination of execution of the object program. An execution profile can then be generated by use of prof(1). (Z8000 only)

-O Invoke an object-code optimizer.

-S Compile the named C programs, and leave the assembler-language output on corresponding files suffixed **.s**.

-E Run only the macro preprocessor on the named C programs, and send the result to the standard output.

-P Run only the macro preprocessor on the named C programs, and leave the result on corresponding files suffixed **.i**.

-C Comments are not stripped by the macro preprocessor.

-Dname=def
-Dname Define the name to the preprocessor, as if by **#define**. If no definition is given, the name is defined as 1.

-Uname Remove any initial definition of name.

-Idir Change the algorithm for searching for **#include** files whose names do not begin with / to look in dir before looking in the directories on the standard list. Thus, **#include** files whose names are enclosed in "" will be searched for first in the directory of the file argument, then in directories named in **-I** options, and last in directories on a standard list. For **#include** files whose names are enclosed in <>, the directory of the file argument is not searched.

Other arguments are taken to be either link editor option arguments, or C-compatible object programs, typically produced by an earlier cc or pcc run, or perhaps libraries of C-compatible routines. These programs, together with the results of any compilations specified, are linked (in the order given) to produce an executable program with the name **a.out**.

The loader (ld(1)) accepts 8-character symbols, and the first character of each symbol is an underbar (`_'), which cc prefixes at compile time. Therefore, symbol names in program modules that are to be linked must be unique within the first seven characters.

FILES

Files with `[n]' are ncc versions.

file.c	input file
file.o	object file
a.out	linked output
/lib/cpp	preprocessor
/usr/lib/[n]ccom	compiler, pcc
/lib/[n]c2	optional optimizer
/lib/crt0.o	runtime startoff
/lib/mcrt0.o	startoff for profiling
/lib/libc[n].a	standard library, see (3)
/usr/include	standard directory for **#include** files

SEE ALSO

B. W. Kernighan and D. M. Ritchie, The C Programming Language, Prentice-Hall, 1978.
B. W. Kernighan, Programming in C-A Tutorial.
D. M. Ritchie, C Reference Manual.
adb(1), as(1), ld(1), prof(1), monitor(3C).

DIAGNOSTICS

The diagnostics produced by C itself are intended to be self-explanatory. Occasional messages may be produced by the assembler or the link editor.

BUGS

If a #define line contains a continuation (\ newline), cc miscounts the number of lines in the program.

The following SYSTEM III options are not supported: -f, -g, -d, -B, and -t.

Cc does not allow more than 250 switches in a case statement. (Z8000 only)

NAME

ld - link editor

SYNOPSIS

ld [-sulxXrdnim] [-o name] [-t name] [-V num] file ...

DESCRIPTION

Ld combines several object programs into one; resolves external references; and searches libraries (as created by ar(1)). In the simplest case several object files are given, and ld combines them, producing an object module which can be either executed or become the input for a further ld run. (In the latter case, the **-r** option must be given to preserve the relocation bits.) The output of ld is left on **a.out**. This file is made executable if no errors occurred during the load and the **-r** flag was not specified.

The argument routines are concatenated in the order specified. The entry point of the output is the beginning of the first routine.

The loader accepts 8-character symbols, and the first character of each symbol is an underbar (`_'), which cc prefixes at compile time. Therefore, symbol names in program modules that are to be linked must be unique within the first seven characters.

If any argument is a library, it is searched exactly once at the point it is encountered in the argument list. Only those routines defining an unresolved external reference are loaded. If a routine from a library references another routine in the library, the referenced routine must appear after the referencing routine in the library. Thus the order of programs within libraries is important.

The symbols **_etext**, **_edata** and **_end** (**etext**, **edata** and **end** in C) are reserved, and if referred to, are set to the first location above the program, the first location above initialized data, and the first location above all data respectively. It is erroneous to define these symbols.

Ld understands several flag arguments which are written preceded by a -. Except for -l, they should appear before the file names.

-s ``Strip'' the output, that is, remove the symbol table and relocation bits to save space (but impair the usefulness of the debugger). This information can also be removed by strip(1). This option is turned off if there are any undefined symbols.

-u Take the following argument as a symbol and enter it as undefined in the symbol table. This is useful for loading wholly from a library, since initially the symbol table is empty and an unresolved reference is needed to force the loading of the first routine.

-l This option is an abbreviation for a library name. **-l** alone stands for **/lib/libc.a**, which is the standard system library for C and assembly language programs. **-l**_x_ stands for **/lib/lib**_x_**.a**, where _x_ is a string. If that does not exist, _ld_ tries **/usr/lib/lib**_x_**.a** A library is searched when its name is encountered, so the placement of a **-l** is significant.

-x Do not preserve local (non-**.globl**) symbols in the output symbol table; only enter external symbols. This option saves some space in the output file.

-X Save local symbols except for those whose names begin with **L**. This option is used by _cc_ to discard internally generated labels while retaining symbols local to routines.

-r Generate relocation bits in the output file so that it can be the subject of another _ld_ run. This flag also prevents final definitions from being given to common symbols, and suppresses the ``undefined symbol'' diagnostics.

-d Force definition of common storage even if the **-r** flag is present.

-n Arrange that when the output file is executed, the text portion will be read-only and shared among all users executing the file. This involves moving the data areas up to the first possible 2K-byte (Z8000) or 4K-byte (MC68000) boundary following the end of the text. On the MC68000, this option is on by default; use **-N** to turn it off.

-i When the output file is executed, the program text and data areas will live in separate address spaces. The only difference between this option and **-n** is that here the data starts at location 0. This option is meaningful only on the Z8000; it does nothing on the MC68000.

-m The names of all files and archive members used to create the output file are written to the standard output. (Z8000 only)

-o The _name_ argument after **-o** is used as the name of the

ld output file, instead of **a.out**.

-t The name argument is taken to be a symbol name, and any references to or definitions of that symbol are listed, along with their types. There can be up to 16 occurrences of **-t**name on the command line. (Z8000 only)

-V The num argument is taken as a decimal version number identifying the **a.out** that is produced. Num must be in the range 0-32767. The version stamp is stored in the **a.out** header; see a.out(5). (Z8000 only)

FILES

/lib/lib?.a	libraries
/usr/lib/lib?.a	more libraries
a.out	output file

SEE ALSO

ar(1), as(1), cc(1), a.out(5).