a.out.h — Header File

Include all COFF header files
#include <coff/a.out.h>

a.out.h includes all header files needed to generate COFF output.

See Also

arcoff.h, file formats, header files
Gircyc, G.R.: *Understanding and Using COFF*. Sebastopol, Calif, O'Reilly & Associates, Inc., 1990.

abort() — General Function (libc)

End program immediately
#include <stdlib.h>
void abort()

abort() terminates a process with a core dump, creating a file called **core**, and prints a message on the screen. It is normally invoked in situations that "should not happen". For example, **malloc()** invokes **abort()** if it discovers a corrupt storage arena.

Where possible, **abort()** executes a machine instruction that causes the processor to trap. If the signal associated with the trap is caught or ignored, the dump will not be produced.

See Also

_exit(), core, exit(), libc, stdlib.h
ANSI Standard, §7.10.4.1
POSIX Standard, §8.1

abs() — General Function (libc)

Return the absolute value of an integer
#include <stdlib.h>
int abs(*n*) int *n*;

abs() returns the absolute value of integer *n*. The *absolute value* of a number is its distance from zero. This is *n* if *n***>=0**, and -*n* otherwise.

Example

This example prompts for a number, and returns its absolute value.

```
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>

main()
{
      extern char *gets();
      extern int atoi();
      char string[64];
      int counter;
      int input;

      printf("Enter an integer: ");
      fflush(stdout);
      gets(string);
```

```
	for (counter=0; counter < strlen(string); counter++) {
		input = string[counter];

		if (!isascii(input)) {
			fprintf(stderr,
				"%s is not ASCII\n", string);
			exit(EXIT_FAILURE);
		}

		if (!isdigit(input))
			if (input != '-' || counter != 0) {
				fprintf(stderr,
					"%s is not a number\n", string);
				exit(1);
			}
	}

	input = atoi(string);
	printf("abs(%d) is %d\n", input, abs(input));
	exit(EXIT_SUCCESS);
}
```

See Also

fabs(), floor(), int, libc, stdlib.h
ANSI Standard, §7.10.6.1
POSIX Standard, §8.1

Notes

On two's complement machines, the **abs()** of the most negative integer is itself.

ac — Command

Summarize login accounting information
ac [**-dp**] [**-w** *wfile*][*username ...*]

One of the accounting mechanisms available on the COHERENT system is login accounting, which keeps track of the time each user spends logged into the system. Login accounting is enabled by creating the file **/usr/adm/wtmp**. Thereafter, the routines **date**, **login**, and **init** write raw accounting data to **/usr/adm/wtmp** to record the time, the name of the terminal, and the name of the user for each date change, login, logout, or system reboot.

The command **ac** summarizes the accounting data that have accumulated for your system. By default, it prints the total connect time found in **/usr/adm/wtmp**. If its command line includes a user's login identifier, **ac** prints a summary only of that user's activity.

ac recognizes the following command-line options:

-d Itemize the output into daily periods. **ac** defines a day as beginning at midnight.

-p Print a summary for every user on your system.

-w Read data from *wfile*. By default, **ac** reads its data from **/usr/adm/wtmp**.

See Also

commands, date, init, login, sa, utmp.h

Notes

File **/usr/adm/wtmp** can become very large; therefore, you should truncated it periodically. Special care should be taken if you have enabled login accounting and your system has limited amounts of free disk space.

accept() — Sockets Function (libsocket)

Accept a connection on a socket
#include <sys/types.h>
#include <sys/socket.h>
int accept(*socket*, *address*, *addrlen***)**
int *socket*, **addrlen***; struct sockaddr** **address***;**

accept() accepts a connection on a socket. It extracts the first connection request on the queue of pending connections, creates a new socket with the same properties as *socket*, and allocates a file descriptor for the newly created socket. It is used with connection-based types of sockets, currently with **SOCK_STREAM**.

socket gives a file descriptor that identifies a socket. It must have been returned by a call to **socket()**, have been bound to an address by a call to **bind()**, and be listening for connections after a call to **listen()**.

If no connections are pending on the queue and *socket* is not marked as non-blocking, **accept()** blocks the calling process until it can establish a connection. If *socket* is marked non-blocking and no connections are pending on the queue, **accept()** returns an error, as described below. The accepted socket may not be used to accept more connections; however, the original *socket* remains open.

address gives the address of the connecting entity, as known to the "communications layer". Its exact format is dictated by the domain in which communication occurs.

addrlen points to an integer that gives the number of bytes available at *address*. Upon return, that integer contains the number of bytes to which *address* actually points.

The function **select()** can perform the same action as **accept()**: simply select the socket for reading.

If all goes well, **accept()** returns the file descriptor for the accepted socket, which is a non-negative integer. If something goes wrong, **accept()** returns -1 and set **errno** to an appropriate value. The following lists the errors that can occur, by the value to which **accept()** sets **errno**:

EBADF *socket* is somehow invalid.

ENOTSOCK
socket references a file, not a socket.

EOPNOTSUPP
socket references a socket that is not of type **SOCK_STREAM**.

EFAULT
addr contains an illegal address.

EWOULDBLOCK
The socket is marked non-blocking, and no connections are present to be accepted.

Example

For an example of this function, see the Lexicon entry for **libsocket**.

See Also

bind(), connect(), libsocket, listen(), select()

access() — System Call (libc)

Check if a file can be accessed in a given mode
#include <unistd.h>
int access(*filename, mode***) char ****filename***; int** *mode***;**

access() checks whether a file or directory can be accessed in the mode you wish. *filename* is the full path name of the file or directory you wish to check. *mode* is the mode in which you wish to access *filename*, as follows:

F_OK	File exists
R_OK	Read a file
W_OK	Write into a file
X_OK	Execute a file

The header file **unistd.h** defines these values, which may be logically combined to produce the *mode* argument.

If *mode* is **F_OK**, **access()** tests only whether *filename* exists, and whether you have permission to search all directories that lead to it.

access() returns zero if *filename* can be accessed in the requested mode, and a nonzero value if it cannot. Note that the return value is the opposite of the intuitive value, i.e., zero means success rather than failure.

access() uses the *real* user id and *real* group id (rather than the *effective* user id and *effective* group id), so set user id programs can use it.

Example

The following example checks if a file can be accessed in a particular manner.

```
#include <unistd.h>
#include <stdio.h>

main(argc, argv)
int argc; char *argv[];
{
      int mode;
      extern int access();

      if (argc != 3) {
            fprintf(stderr, "usage: acc dir_name/file_name mode\n");
            exit(EXIT_FAILURE);
      }

      switch (*argv[2]) {
      case 'x':
            mode = X_OK;
            break;

      case 'w':
            mode = W_OK;
            break;

      case 'r':
            mode = R_OK;
            break;

      case 'f':
            mode = F_OK;
            break;

      default:
            fprintf(stderr, "Bad mode. Modes: f, x, r, w\n");
            exit(EXIT_FAILURE);
            break;
      }

      if (access(argv[1], mode))
            printf("file %s cannot be found in mode %d\n", argv[1], mode);
      else
            printf("file %s is accessible in mode %d\n", argv[1], mode);
      exit(EXIT_SUCCESS);
}
```

See Also

libc, path(), unistd.h
POSIX Standard, §5.6.3

Notes

When the superuser **root** executes **access()**, it always returns readable/writable/executable for any file that exists, regardless of permissions.

Note that **access()** used to be declared in header file **<access.h>**. It is now prototyped in header file **<unistd.h>**, to comply with the POSIX standard. **<access.h>** is obsolete and has been dropped from COHERENT beginning with release 4.2.

acct() — System Call (libc)

Enable/disable process accounting
#include <acct.h>
acct(*file*)
char ****file*;**

Process accounting records who initiates each system process and how long each process takes to execute. These data can be analyzed, to administer the system most efficiently.

The system call **acct()** enables or disables process accounting. If *file* is not NULL, then accounting is turned on; if *file* is NULL, however, then process accounting is turned off.

It is usual, but not necessary, that *file* be **/usr/adm/acct**. *file* must exist. When enabled, the system appends a raw accounting data record in the format described by **acct.h** to *file* as each process terminates.

acct() is restricted to the superuser.

See Also

ac, acct.h, accton, exit(), libc, sa, times(),

Diagnostics

Successful calls return zero. **acct()** returns -1 for errors, such as nonexistent *file* or invocation by a user other than the superuser.

Notes

The system writes accounting records for a process only when the process exits. Processes that never terminate and processes running at the time of a system crash do not produce accounting information.

acct.h — Header File

Format for process-accounting file
#include <acct.h>

Process accounting is a feature of the COHERENT system that allows it record what processes each user executes and how long each process takes. These data can be used to track how much each user uses the system.

The function **acct()** turns process accounting off or on. When process accounting has been turned on, the COHERENT system writes raw process-accounting information into an accounting file as each process terminates. Each entry in the accounting file, normally **/usr/adm/acct**, has the following form, as defined in the header file **acct.h**:

```
struct acct {
       char       ac_comm[10];
       comp_t     ac_utime;
       comp_t     ac_stime;
       comp_t     ac_etime;
       time_t     ac_btime;
       short      ac_uid;
       short      ac_gid;
       short      ac_mem;
       comp_t     ac_io;
       dev_t      ac_tty;
       char       ac_flag;
};

/* Bits from ac_flag */
#define    AFORK      01     /* has done fork, but not exec */
#define    ASU        02     /* has used superuser privileges */
```

Every time a process calls **exec()**, the contents of **ac_comm** are replaced with the first ten characters of the file name. The fields **ac_utime** and **ac_stime** represent the CPU time used in the user program and in the system, respectively. **ac_etime** represents the elapsed time since the process started running, whereas **ac_btime** is the time the process started. The effective user id and effective group id are **ac_uid** and **ac_gid. ac_mem** gives the average memory usage of the process. **ac_io** gives the number of blocks of input-output. **ac_tty** gives the controlling typewriter device major and minor numbers.

For some of the above times, the **acct** structure uses the special representation **comp_t,** defined in the header file **types.h**. It is a floating point representation with three bits of base-8 exponent and 13 bits of fraction, so it fits in a **short** integer.

See Also

acct(), accton, header files, sa

accton — Command

Enable/disable process accounting
/etc/accton [*file*]

One of the accounting mechanisms available on the COHERENT system is *process accounting*, Process accounting records each process, who initiates it, and how long it takes to execute.

The command **accton** turns process account on or off. To turn on process accounting, issue the command **accton** followed by a *file* argument; COHERENT then begin to write accounting data into *file*, By convention, *file* should be **/usr/adm/acct**. To turn off process accounting, issue the command **accton** without any arguments.

The command **sa** summarizes the data that will have been written into *file*.

See Also

ac, acct, acct.h, commands, init, sa

Notes

As the accounting file can become very large, you should truncate that file from time to time. You should take extra care to monitor the growth of that file should you enable process accounting on a system with a limited amount of free disk space.

acos() — Mathematics Function (libm)

Calculate inverse cosine
#include <math.h>
double acos(*arg***) double** *arg***;**

acos() calculates the inverse cosine. *arg* should be in the range of -1.0, 1.0. It returns the result, which is in the range of from zero to π radians.

Example

This example demonstrates the mathematics functions **acos()**, **cabs()**, and **tan()**.

```
#include <errno.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#define display(x) dodisplay((double)(x), #x)

dodisplay(value, name)
double value; char *name;

{
      if (errno)
            perror(name);
      else
            printf("%10g %s\n", value, name);
      errno = 0;
}

main()
{
      extern char *gets();
      double x;
      char string[64];

      for(;;) {
            printf("Enter number: ");
            if(gets(string) == NULL)
                  break;

            x = atof(string);
            display(x);
            display(acos(cos(x)));
            display(cabs(sin(x),cos(x)));
      }
}
```

See Also

cos(), errno, errno.h, libm, perror()
ANSI Standard, §7.5.2.1
POSIX Standard, §8.1

add_history() — Editing Function (libedit)

Add a line to history buffer
void add_history(*line***)**
char ****line*;**

The function **add_history()** adds *line* to a "history" buffer, from which it can be retrieved by the function **readline()**. *line* must have been returned by **readline()**.

See Also

libedit, readline()

address — Definition

An **address** is the location where an item of data is stored in memory.

On the i8086, a physical address is a 20-bit number. The i8086 builds an address by left-shifting a 16-bit segment address by four bits, and then adding it to a 16-bit offset address. The segment address points to a particular chunk of memory. The i8086 uses four segment registers, each of which governs a different portion of a program, as follows:

CS	Address of code segment
DS	Address of data segment
ES	Address of "extra" segment
SS	Address of stack segment

SMALL-model programs use only the offset address; hence, their pointers are only 16 bits long, equivalent to an **int**. LARGE-model programs use both segment and offset addresses. Their addresses are 20 bits long, which must be stored in a 32-bit pointer, equivalent to a **long**. COHERENT 286 supports SMALL model.

On the i80386, addresses start as 32 bits. Segment registers are used to look up a segment descriptor. The descriptor's base then defines the address within a four-gigabyte virtual address space. The page tables are then used to translate this to a physical address. For details, see the *Intel 386 Programmers Manual*.

On the M68000, an address is simply a 24-bit integer that is stored as a 32-bit integer. The upper eight bits are ignored; this is not true with the more advanced microprocessors in this family, such as the M68020. The M68000 uses no segmentation; memory is organized as a "flat address space," with no restrictions set on the size of code or data.

On machines with memory-mapped I/O, such as the 68000, some addresses may be used to control or communicate with peripheral devices.

Example

The following printes the address and contents of a given byte of memory.

```
#include <stdio.h>

main()
{
      char byte = 'a';
      printf("Address == %x\tContents == \"%c\"\n",
            &byte, byte);
}
```

See Also

data formats, pointer, Programming COHERENT

Administering COHERENT — Overview

To administer a COHERENT system, you must know how to do the following:

- Perform backups, manage archives and purge old files.
- Set up and manage complex system, such as mail, UUCP, and the print spooler.
- Attach peripheral devices, such as terminals, modems, and printers.
- Install third-party software.
- Configure the kernel, and add or configure device drivers.
- Act as a resource person for other users.

Overview Lexicon Articles

Many users who have purchased COHERENT for their personal use will find some of these tasks to be confusing or daunting. This is especially true if they have had no previous exposure to UNIX or similar operating systems. Such a person will find the following Lexicon articles to be helpful:

backups
When and how to back up your system, using tape or floppy disks.

booting
How booting works. In particular, it shows how to boot a kernel other than the default kernel.

CD-ROM
Introduce how to use CD-ROM drives under COHERENT.

console
This introduces the device **/dev/console**. It also lists the many escape sequences with which you can change the appearance and behavior of the console.

device drivers
The suite of device drivers available under COHERENT. This article also gives a

floppy disks
Information about floppy disks. This describes the floppy-disk devices available under COHERENT, how to format floppy disks, and how to record data on a floppy disk using a COHERENT file systems, a **tar** archive, or an file systems.

hard disk
This gives basic information about hard disks. In particular, it discusses the devices by which hard disks are accessed, and how to partition a hard disk.

IRQ This article lists the IRQs available on the IBM PC.

kernel This introduces the *kernel*, which is the master program of COHERENT. It also gives examples of how to configure and patch the kernel.

keyboard
This introduces the suite of keyboard drivers available for the COHERENT keyboard.

lpsched
This command is the daemon for the **lp** print spooler. For an overview of **lp** and the other print spoolers, see the Lexicon entry for **printer**.

mail This gives an overview of the COHERENT mail system — both commands and configuration files.

modem
This describes how to add a modem to your COHERENT system. It also introduces the communications programs available under COHERENT.

printer This describes how to add a printer to your system. It also gives an overview of the various print spoolers available with COHERENT, and how to configure each to work with a variety of printers.

RS-232
This presents the design and pin-out of the RS-232 plug, which is the standard plug for serial and parallel ports on the IBM PC and its clones.

security
This article discusses the problem of system security — that is, how to let your users but keep the "crackers" out.

tape This introduces tape devices. It describes how to access tape, and goes into some detail on how to manage tape archives.

terminal
This describes how to plug a terminal into your system, and configure it correctly.

tboot The tertiary boot is the program that loads the COHERENT kernel into memory and launches it. This article describes it. You probably will never need to work with **tboot**— but you never know.

virtual console
COHERENT supports virtual consoles, whereby several console sessions can be run on the same physical device. This describes how to set up and manage virtual consoles on your system.

System Files

The COHERENT system is controlled by *system files* and *daemons*. System files contain the information that controls the minute-to-minute operation of the COHERENT system. A daemon is a program that the system runs to manage a peripheral device or perform some other task that does not require the intervention of a human. COHERENT's system files and daemons are described in the following Lexicon articles:

/usr/lib/mail/aliases
This file holds the aliases by which your system is known to other systems.

atrun This daemon executes other commands at a preset time. A user can use the command **at** to spool another command for execution at a later time.

/etc/boottime
This file records the date and time your system was last booted.

/etc/brc
COHERENT executes this script when your system enters single-user mode. It performs maintenance chores.

/etc/checklist
This file lists the devices to check with **fsck** when you boot COHERENT.

/usr/lib/mail/config
This file performs overall configuration of **smail**.

/usr/lib/uucp/config
This file performs overall configuration of UUCP.

/usr/spool/mlp/controls
This file holds the data base for the MLP print spooler.

core This Lexicon entry describes the format of a core file — that, the file that a program dumps when it fails catastrophically.

/etc/cron
This daemon reads a data base of commands to execute periodically, and executes each when its time comes round at last.

/etc/d_passwd
This file holds the passwords that control access to your system via peripheral devices. For example, you can set an extra password in this file for all users who may attempt to log in via modem.

/usr/lib/uucp/dial
This file holds the information by which UUCP dials a modem.

/etc/dialups
This file names every peripheral device that requires an additional password.

/usr/lib/mail/directors
Name the director routines that **smail** uses, and configure them.

/etc/domain
This file names the mail domain to which your system belongs.

/etc/drvld.all
This file names the loadable drivers to load when you boot your system.

$HOME/.forward
This File lets you set a forwarding address for mail.

/etc/getty
This daemon initializes a serial port, watches the port, and assists any user who attempts to log into your system.

/etc/group
This file define groups of users on your system.

/etc/hosts
This file gives the name and address of every host on your local network.

/etc/hosts.equiv
This file names “equivalent hosts” on your local network — that is, the hosts that have identical (or nearly identical) sets of users.

/etc/hosts.lpd
This file holds the name and domain of your local host.

/usr/lib/hpd
This daemon is a spooler daemon for a laser printer.

/etc/inetd.conf
This file configures the Internet daemons.

/etc/init
Command helps to bring COHERENT into multi-user mode. It also helps users to log in.

$HOME/.kshrc
This script configures the Korn shell to suit your tastes.

$HOME/.lastlogin
This file records the date and time you last logged in to your COHERENT system.

login This command logs a user in to your COHERENT system. Its Lexicon article also describes the entire convoluted process of managing an enabled port and logging a user in.

/etc/default/login
This file sets default values for logging in.

/usr/adm/loginlog
This file logs failed attempts to log in.

/etc/logmsg
This file holds the COHERENT login prompt. If you do not like the prompt

```
Coherent 386 login:
```

and a beep, you can change it by editing this file.

/usr/lib/lpd
This daemon manages the MLP print spooler.

/etc/conf/mdevice
This file describes the device drivers currently available on your system.

/etc/mnttab
This file holds the mount table — that is, the table that describes which file systems are mounted, and what directories they are mounted on.

/etc/motd
This file holds the message of the day — a message that is printed on each user’s terminal when she logs in.

/etc/mount.all
This file names the disk devices to mount when your system enters multi-user mode.

/etc/conf/mtune
This file names the set of variables in the kernel and its device drivers that you can “tune,” to modify the kernel’s behavior.

/etc/networks
This file describes remote networks that your system can contact.

/etc/nologin
This file, if it exists, prevents users from logging in. It is used during special periods of time, such as when you wish to shut the system down.

/etc/passwd
This file describes every user who has permission to log into your system.

/usr/lib/mail/paths
This file holds the information by which your system routes mail to other systems.

/usr/lib/uucp/port
This file describes the serial ports through which UUCP can dial out from your system.

/etc/profile
This script sets up the default environment for each user on your system.

$HOME/.profile
This script holds commands that are executed when a given user logs in to your COHERENT system.

/etc/protocols
This file names the Internet protocols that your system supports.

/usr/bin/ramdisk
This script lets you build a RAM disk on your system.

/etc/rc
This script is executed when your system enters multi-user mode. It normally performs standard housekeeping chores.

/usr/lib/mail/routers
This file names the routing programs that **smail** uses, and configures them.

/etc/conf/sdevice
This file holds the information by which device drivers are configured when you build a kernel.

/etc/serialno
This file holds your system’s serial number, which you entered when you first installed COHERENT.

/etc/services
This file lists the Internet services that your system supports.

/etc/shadow
This file holds each user’s password.

/etc/conf/stune
This file sets the values of tunable kernel variables.

/usr/lib/uucp/sys
This file describes the remote systems that you can contact via UUCP, and how to contact them.

term This Lexicon article describes the format of a compiled **terminfo** file.

/etc/termcap
This file holds **termcap** terminal-description data base.

terminfo
This article describes the **terminfo** terminal-description language. Its data base is kept in directory **/usr/lib/terminfo**.

/usr/lib/mail/transports
This file names the transport routines that **smail** can use, and configures them.

/etc/trustme
This file names of trusted users — that is, users who can log in even if file **/etc/nologin** exists.

/etc/ttys
This file describes terminal ports — that is, the ports via which a user can log in. This includes both serial ports and pseudo-ttys.

/etc/update
This daemon periodically flushes all buffered information to disk.

/etc/usrtime
This file holds the time, day of the week, and terminal line by which each user can log into your COHERENT system.

/etc/utmp
This file notes every login event that has not yet concluded — that is, a user has logged in but not logged out again. You can examine this file to see who is using your system at this moment.

/etc/uucpname
This file sets your system's UUCP name — that is, the name by which it is known to all other systems.

/etc/default/welcome
This script is executed whenever a user logs in for the first time. It gives the new user some basic information and advice.

/usr/adm/wtmp
This file notes every login event that has concluded — that is, a user has logged in and logged out again. You can examine this file to see who has logged into your system in the past, and for how long.

Finally, the following header files also hold information on file formats:

acct.h	Format for process-accounting file
ar.h	Format for archive files
canon.h	Portable layout of binary data
coff.h	Define format of COHERENT 386 objects
l.out.h	Define format of COHERENT 286 objects
mtab.h	Currently mounted file systems
utmp.h	Login accounting information

For a fuller description of each file and its contents, see its entry in the Lexicon.

See Also

COHERENT, Programming COHERENT, Using COHERENT

alarm() — System Call (libc)

Set a timer
#include <unistd.h>
alarm(*seconds***)**
unsigned *seconds***;**

alarm() sets a timer. After *seconds*, the COHERENT kernel sends signal **SIGALRM** to the process that invoked **alarm()**. Setting *seconds* to zero turns off the alarm timer.

By default, signal **SIGALRM** terminates the process. However, a program can invoke the system call **signal()** to catch this signal, or ignore it. Because of scheduling variation and the one-second granularity, the action of **alarm()** is predictable only to within one second.

alarm() is useful for such things as timeouts. For example, a process on a dial-in port might hang up the line after a sufficient time has elapsed with no user response.

alarm() returns the previous alarm value, which represents the time remaining from the previous call. Time remaining is superseded by the new alarm value.

See Also

libc, signal(), sleep(), unistd.h
POSIX Standard, §3.4.1

Notes

A process can set only one alarm at a time.

alias — Command

Set an alias
alias [*name***[=***value ...***]]**

The command **alias** is used by the Korn shell **ksh** to set or display an alias.

When called without an argument, **alias** lists all aliases that have been set so far. When called with a *name* argument alone, it lists alias of *name*, assuming one has been set.

When called with one or more arguments of the form *name***=***value*, it established *name* as an alias for the command *value*. For example, the command

```
alias FOO="echo bar"
```

establishes the string **FOO** as an alias for the command **echo bar**. Thereafter, when you type **FOO** on the shell's command line, it will execute the command **echo bar** and so echo the string **bar** on your terminal.

The Korn shell sets a number of aliases by default. See the Lexicon entry for **ksh** for a list of these aliases and their settings.

See Also

commands, ksh, unalias

aliases — System Administration

File of users' aliases
/usr/lib/mail/aliases

File **/usr/lib/mail/aliases** holds aliases for users' addresses — either on your system, or on other systems. The command **smail** reads this file when it figures out how to deliver a mail message.

An *alias* is a "nickname" for a user. Once you have established an alias for a user, you can use that alias to send mail to her; this spares you the trouble of typing that person's convoluted e-mail address. An alias can also name an entire group of users; when you use the alias to send a mail message, every person in the group receives a copy.

The format of each alias is

```
alias_name:           target
```

where *alias_name* gives the alias to which you mail your message, and *target* is name to which where **smail** actually directs the message. *target* can be a login identifier on your local system; a mail address of a user on another system; or a cluster of users either on your system or on remote systems.

For example, consider the user whose e-mail address is **ivan@lepanto.mwc.com**. If you add the entry

```
ivan: ivan@lepanto.mwc.com
```

to file **/usr/lib/mail/aliases**, then whenever you send mail to **ivan**, the routing program **smail** will automatically "expand" the address from **ivan** to **ivan@lepanto.mwc.com**, and dispatch the message properly. This spares you needless work, and eliminates the errors that would occur if you typed long addresses by hand.

Please note that **smail** ignores differences in case when it compares a name with an alias. If a line begins with a white-space character, **smail** assumes that that line is a continuation of the previous line. **smail** ignores strings within parentheses, as well as any text that appears after the pound sign '#'. Thus, you can use '#' to embed a comment within **aliases**.

Examples

The following gives an example form of **aliases**:

```
# this whole line is a comment
# "mail programmers" sends mail to local users joe, jack, and bill
programmers:          joe jack bill

# same as above
programmers:          joe jack
                      bill

# same as above
programmers           joe jack
                      bill

# same as above
programmers           joe   # Joe Smith
                      jack  # Jack Thomas
                      bill  # Bill Williams

# and yet another way; note use of parentheses to comment text
programmers           joe (Joe Smith) jack (Jack Thomas)
                      bill (Bill Williams)

# send a message to someone on another system.
# this uses ‘‘bang-path’’ addressing
joe:                  boston!widget!js

# send a message to users on both your and another system
programmers:          boston!widget!js   # Joe Smith
                      chicago!gadget!jt  # Jack Thomas
                      bill               # Bill Williams

# all members of "programmers" group work at site "widget"
programmers!widget           joe jack bill
```

To tell **smail** to use the contents of another file to expand an alias, use the following form:

```
fredlist              :include:/usr/lib/mail/fredlist
```

smail adds each entry in **/usr/lib/mail/fredlist** to the alias for **fredlist**.

You can also tell **smail** to read another alias file, and include its contents in the list of aliases to be expanded. For example, the following instruction

```
:include:/usr/lib/mail/morealiases
```

when embedded within **/usr/lib/mail/aliases**, tells **smail** to add the contents of **/usr/lib/mail/morealiases** to those of **/usr/lib/mail/aliases** as a regular alias file.

All aliases are recursive, so you must be careful when you define them. For example, the entries

```
bill:          joe
joe:           bill
```

causes an infinite loop. **smail** attempts to detect infinite loops, and to guess what you intended to do. The following example illustrates how you can use an alias to deliver mail to a remote user as well as to a local user who has the same name as the alias being expanded. **smail** expands the alias

```
mylogin:              mypc!mylogin  mylogin
```

to

```
mypc!mylogin mylogin
```

even though the second occurrence of **mylogin** matches the alias name.

Both forms of file inclusion are recursive, too, and may lead to infinite loops if handled carelessly.

See Also

Administering COHERENT, mail [overview], smail

Notes

Beginning with release 4.2.14 of COHERENT, **smail**'s aliases are kept in the form of a DBM data base. This is a simple data base that uses a hash table to speed the retrieval of information. If you change your file of aliases, you must invoke either the command **newaliases** or the command **smail -bi** to rebuild the binary data base of aliases.

For details on what a DBM data base is, see the Lexicon entry for **libgdbm**. For details on how to use **newaliases** or **smail**, see their respective entries in the Lexicon.

alignment — Definition

Alignment or packing of fields within a structure

Alignment refers to the fact that some microprocessors require the address of a data entity to be *aligned* to a numeric boundary in memory so that *address* modulo *number* equals zero. For example, the M68000 and the PDP-11 require that an integer be aligned along an even address, i.e., *address***%2==0**. In the MS-DOS world, this is called "packing".

Generally speaking, alignment is a problem only if you write programs in assembly language. For C programs, COHERENT ensures that data types are aligned properly under foreseeable conditions. You should, however, beware of copying structures and of casting a pointer to **char** to a pointer to a **struct**, for these could trigger alignment problems.

Processors react differently to an alignment problem. On the VAX or the i8086, it causes a program to run more slowly, whereas on the M68000 it causes a bus error.

See Also

#pragma, data types, ld, Programming COHERENT

Notes

The COHERENT preprocessor instruction **#pragma** lets you set alignment to conform to Intel's Binary Compatibility Standard (BCS). For details, see the Lexicon entry for **#pragma**.

alloc.h — Header File

Define the allocator
#include <sys/alloc.h>

alloc.h defines manifest constants and structures that are used internally with memory allocation.

See Also

header files

Notes

This header file is obsolete, and will be dropped from a future release of COHERENT. Its use is strongly discouraged.

alloca() — General Function (libc)

Dynamically allocate space on the stack
alloca(*memory***)**
int *memory***;**

The function **alloca()** allocates *memory* number of bytes dynamically on the stack. The allocated memory disappears automatically as soon as the program exits from the function within which the memory was allocated.

For example, consider the function:

```
foo(some_string)
char *some_string;
{
      char *cp;
      . . .
      cp = alloca(strlen(some_string) + 1);
      strcpy(cp, some_string);
      . . .
}
```

Here, the call to **alloca()** allocates enough space upon the stack for **some_string** plus the terminating NUL character. When function **foo()** returns, the allocated memory vanishes.

This routine is popular in Berkeley and GNU circles because it is much faster than **malloc()**, and the programmer does not need to call **free()** to de-allocate the memory.

See Also

calloc(), libc, malloc(), realloc()

almanac — Command

Print an almanac entry for this date
almanac [*month day***]**

The command **almanac** prints on the standard output an almanac entry of noteworthy births, deaths, and events that occurred on this date. *month* and *day* give the date whose listing you wish to see. *month* must be the name of the month. For example, the command

```
almanac November 23
```

prints something like the following on your screen:

```
BIRTHS:
1221: Alfonso X (el Sabio), monarch and music collector, Toledo.
1876: Manuel de Falla, composer, Cadiz.
1887: William Henry Pratt (Boris Karloff), actor.
1888: Adolph "Harpo" Marx, comedian & musician, New York City.
DEATHS:
1585: Thomas Tallis, composer, Greenwich.
EVENTS:
1923: Height of German inflation: 4.2 trillion marks to the dollar.
1935: First "Porky Pig" cartoon premieres.
```

If you do not supply any arguments on the command, **almanac** prints an almanac listing for today.

almanac reads its information from the files **almanac.birth**, **almanac.death**, and **almanac.event**, which are kept in directory **/usr/games/lib**. Each has the same format: the date encoded by the first three letters of the name of the month (with an initial capital letter), followed by the day of the month, followed by the body of the entry. For example:

```
Nov 23 1221: Alfonso X (el Sabio), monarch and music collector, Toledo.
```

You are encouraged to modify these files to suit your tastes and interests.

See Also

commands

Notes

almanac does not check for bogus dates before it reads its data files. It also is quite rigid in how it expects its data base to be laid out.

The data files reflect the tastes of the person who compiled them, and can be rather idiosyncratic.

ANSI — Definition

Standards for information

The American National Standards Institute (ANSI) includes a standards committee called X3, which writes and publishes standards for information-processing systems. Relevant ANSI standards include the following:

X3.4-1977
Code for Information Interchange

X3.64-1979
Additional Controls for Use with ANS Code for Information Interchange

X3.159-1989
Programming Language C

Published ANSI standards are available from:

American National Standards Institute, Inc.
1430 Broadway
New York, NY 10018

See Also

C language, POSIX Standard, Programming COHERENT,
The C Language, tutorial
Mark Williams Company: *ANSI C: A Lexical Guide*. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1988.

apropos — Command

Find manual pages on a given topic
apropos *topic* **[** *topic* **...]**

This command implements a simplified version of the Berkeley command **apropos**. It prints every line in the file **man.index** that contains a *topic*. In this way, you can find what manual pages are available on a given topic. For example, the command

```
apropos daemon
```

prints something like the following:

```
daemon         Definition
hpd            Spooler daemon for laser printer
lpd            Spooler daemon for line printer
lpshut         Turn off the printer daemon despooler
```

You can also use **apropos** to nudge your memory when you cannot recall the name of a given command or library function.

apropos normally reads its information from the index files kept in directory **/usr/man**. **apropos** assumes that every file in that directory that ends in the string **.index** is an index file. For details on index files and their format, see the Lexicon entry for **man**.

If the environmental variable **MANPATH** is set, **apropos** searches the index files in each directory that it names. **MANPATH** must name one or more directories, with directories separated by a colon ':'.

Files

/usr/man/*.index

See Also

commands, help, man, Using COHERENT

ar — Command

The librarian/archiver
ar *option* **[***modifier***][***position***]** *archive* **[***member ...***]**

The librarian **ar** edits and examines libraries. It combines several files into a file called an *archive* or *library*. Archives reduce the size of directories and allow many files to be handled as a single unit. The principal use of archives is for libraries of object files. The linker **ld** understands the archive format, and can search libraries of object files to resolve undefined references in a program.

Options and Modifiers

The mandatory *option* argument consists of one of the following command keys:

d Delete each given *member* from *archive*. The **ranlib** header is updated if present.

m Move each given *member* within *archive*. If no *modifier* is given, move each *member* to the end. The ranlib header is modified if present.

p Print each *member*. This is useful only with archives of text files.

q Quick append: append each *member* to the end of *archive* unconditionally. The ranlib header is not updated.

r Replace each *member* of *archive*. If *archive* does not exist, create it. The optional *modifier* specifies how to perform the replacement, as described below. The ranlib header is modified if present.

t Print a table of contents that lists each *member* specified. If none is given, list all in *archive*. The modifier **v** tells **ar** to give you additional information.

x Extract each given *member* and place it into the current directory. If none is specified, extract all members. *archive* is not changed.

The *modifier* may be one of the following. The modifiers **a**, **b**, **i**, and **u** may be used only with the **m** and **r** options.

a If *member* does not exist in *archive*, insert it after the member named by the given *position*.

b If *member* does not exist in *archive*, insert it before the member named by the given *position*.

c Suppress the message normally printed when **ar** creates an archive.

i If *member* does not exist in *archive*, insert it before the member named by the given *position*. This is the same as the **b** modifier, described above.

k Preserve the modify time of a file. This modifier is useful only with the **r**, **q**, and **x** options.

s Modify an archive's ranlib header, or create it if it does not exist. This must be used for archives read by the linker **ld**.

u Update *archive* only if *member* is newer than the version in the *archive*.

v Generate verbose messages.

Note that because **ar** was created before UNIX established the standard of introducing an option with a hyphen. Therefore, the syntax of options to **ar** differs from most other COHERENT commands: **ar** expects all options and modifiers to be clumped together as its first argument, without an introductory hyphen. For example, to use the option **r** with the modifiers **c** and **s** on library **libname.a** and objects **file1.o** through **file3.o**, type the following command:

```
# RIGHT!
ar rcs libname.a file1.o file2.o file3.o
```

The syntax

```
# WRONG!
ar r -s libname.a file1.o file2.o file3.o
```

creates an archive named **-s**, which you may have some trouble removing.

ar reads the environmental variables **ARHEAD** and **ARTAIL** and appends them to, respectively, the beginning and end of its command line. For example, to ensure that **ar** is always executed with the **c** modifier, insert the following into your **.profile**:

```
export ARHEAD=c
```

Library Structure

All archives are written into a specialized file format. Each archive starts with a "magic string" called **ARMAG**, which identifies the file as an archive. The members of the archive follow the magic number; each is preceded by an **ar_hdr** structure. For information on this structure, see **ar.h**. The structure is followed the data of the file, which occupy the number of bytes specified by the variable **ar_size**.

See Also

ar.h, commands, ld, nm, ranlib

Notes

Each library that you create should have a name that begins with "lib" and ends with ".a". The prefix "lib" lets you call that library with the **-l** option to the command **cc**; and the linker **ld** ignores archives whose names do not end in **.a**.

ar.h — Header File

Format for archive files
#include <ar.h>

An *archive* is a file that has been built from a number of files. Archives are maintained by the command **ar**. Usually, an archive is a library of object files used by the linker **ld**.

The header **ar.h** describes the format of an archive. All archives start with a magic number **ARMAG**, which identifies the file as an archive. The members of the archive follow the magic number, each preceded by the structure **ar_hdr**:

```
#define DIRSIZ  14
#define ARMAG   0177535          /* magic number */

struct ar_hdr {
        char    ar_name[DIRSIZ]; /* member name */
        time_t  ar_date;         /* time inserted */
        short   ar_gid;          /* group owner */
        short   ar_uid;          /* user owner */
        short   ar_mode;         /* file mode */
        size_t  ar_size;         /* file size */
};
```

The structure at the head of each member is immediately followed by **ar_size** bytes, which are the data of the file.

To enhance the performance of **ld**, the command **ranlib** provides a random library facility. **ranlib** produces archives that contain a special entry named **__.SYMDEF** at the beginning.

All integer members of the structure (everything but **ar_name**) are in canonical form to ease portability. See **canon.h** for more information.

See Also

ar, canon.h, header files, ld, ranlib

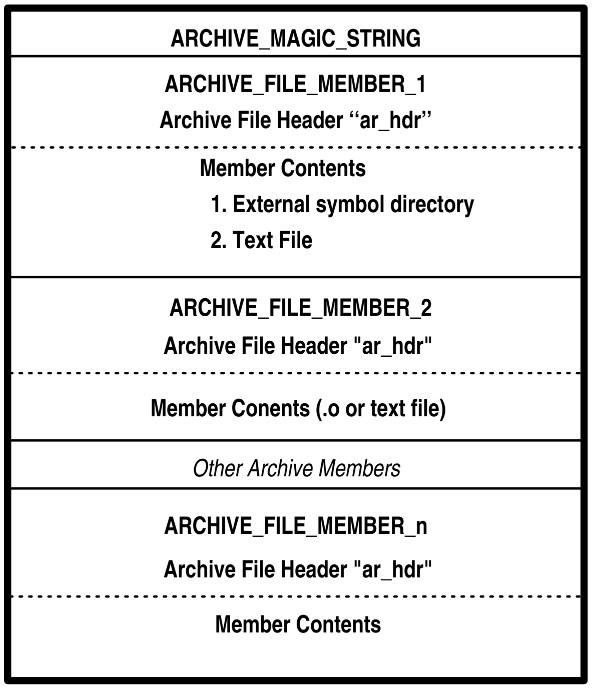
arcoff.h — Header File

COFF archive-file header
#include <coff/arcoff.h>

arcoff.h declares the structure **ar_hdr**, which is the header to a member of an archive. **ar_hdr** is structured as follows:

```
struct ar_hdr
{
       char    ar_name[16];                 /* file member name - '/' terminated */
       char    ar_date[12];                 /* file member date - decimal */
       char    ar_uid[6];                   /* file member user id - decimal */
       char    ar_gid[6];                   /* file member group id - decimal */
       char    ar_mode[8];                  /* file member mode - octal */
       char    ar_size[10];                 /* file member size - decimal */
       char    ar_fmag[2];                  /* ARFMAG - string to end header */
};
```

The COFF common-archive format has the following structure:



See Also

a_out.h, file formats, header files
Gircyc, G.R.: *Understanding and Using COFF*. Sebastopol, Calif, O'Reilly & Associates, Inc., 1990.

arena — Definition

An *arena* is the area of memory that is available for a program to allocate dynamically at run time. It is divided into *allocated* and *unallocated* blocks. The unallocated blocks together form the “free-memory pool”.

To allocate a portion of the arena, use any of the functions **malloc()**, **calloc()**, or **realloc()**. To return an allocated portion of memory to the free-memory pool, use the function **free()**. To check whether a given portion of the arena is already allocated, use the function **notmem()**. To check whether the arena has been corrupted, use the function **memok()**.

See Also

calloc(), free(), malloc(), memok(), notmem(), Programming COHERENT, realloc()

argc — C Language

Argument passed to main()
int argc;

argc is an abbreviation for “argument count”. It is the traditional name for the first argument to a C program's **main** routine. By convention, it holds the number of arguments that are passed to **main** in the argument vector **argv**. Because **argv[0]** is always the name of the command, the value of *argc* is always one greater than the number of command-line arguments that the user enters.

Example

For an example of how to use **argc**, see the entry for **argv**.

See Also

argv, C language, envp, main()
ANSI Standard, §5.1.2.2.1

argv — C Language

Argument passed to main()
char *argv[];

argv is an abbreviation for "argument vector". It is the traditional name for a pointer to an array of string pointers passed to a C program's **main** function; by convention, it is the second argument passed to **main**. By convention, **argv[0]** always points to the name of the command itself.

Example

This example demonstrates both **argc** and **argv[]**, to recreate the command **echo**.

```
main(argc, argv)
int argc; char *argv[];
{
      int i;

      for (i = 1; i < argc; ) {
            printf("%s", argv[i]);
            if (++i < argc)
                  putchar(' ');
      }

      putchar('\n');
      exit(0);
}
```

See Also

argc, C language, envp, main()
ANSI Standard, §5.1.2.2.1

ARHEAD — Environmental Variable

Append options to beginning of ar command line
export ARHEAD=*options*

The COHERENT archiver **ar** reads the environmental variables **ARHEAD** and **ARTAIL** before it begins its work. You can set these variables to hold the default options that you want the archiver always to use. **ar** appends the options in **ARHEAD** to the beginning of its command line.

See Also

ar, ARTAIL, environmental variables

Notes

This environmental variable is included only to support existing code. Its use is deprecated, and it may not be supported in future releases of COHERENT.

array — Definition

An **array** is a concatenation of data elements, all of which are of the same type. All the elements of an array are stored consecutively in memory, and each element within the array can be addressed by the array name plus a subscript.

For example, the array **int foo[3]** has three elements, each of which is an **int**. The three **int**s are stored consecutively in memory, and each can be addressed by the array name **foo** plus a subscript that indicates its place within the array, as follows: **foo[0]**, **foo[1]**, and **foo[2]**. Note that the numbering of elements within an array always begins with '0'.

Arrays, like other data elements, may be automatic (**auto**), **static**, or external (**extern**).

Arrays can be multi-dimensional; that is to say, each element in an array can itself be an array. To declare a multi-dimensional array, use more than one set of square brackets. For example, the multi-dimensional array **foo[3][10]** is a two-dimensional array that has three elements, each of which is an array of ten elements. The second sub-script is always necessary in a multi-dimensional array, whereas the first is not. For example, the form **foo[][10]** is acceptable, whereas **foo[10][]** is not. The first form is an indefinite number of ten-element arrays, which is correct C, whereas the second form is ten copies of an indefinite number of elements, which is illegal.

You can initialize automatic arrays and structures, provided that you know the size of the array, or of any array contained within a structure. An automatic array is initialized in the same manner as aggregate, but initialization is performed on entry to the routine at run time, instead of at compile time.

Flexible Arrays

A **flexible array** is one whose length is not declared explicitly. Each has exactly one empty '[ ]' array-bound declaration. If the array is multidimensional, the flexible dimension of the array must be the *first* array bound in the declaration; for example:

```
      int example1[][20];     /* RIGHT */
      int example2[20][];     /* WRONG */
```

The C language allows you to declare an indefinite number of array elements of a set length, but not a set number of array elements of an indefinite length.

Flexible arrays occur in only a few contexts; for example, as parameters:

```
      char *argv[];
      char p[][8];
```

as **extern** declarations:

```
      extern int end[];
```

or as a member of a structure — usually, though not necessarily, the last:

```
      struct nlist {
            struct nlist *next;
            char name[];
      };
```

Example

The following program initializes an automatic array, and prints its contents.

```
main()
{
      int foo[3] = { 1, 2, 3 };

      printf("Here's foo's contents: %d %d %d\n",
             foo[0], foo[1], foo[2]);
}
```

See Also

initialization, Programming COHERENT, struct

ARTAIL — Environmental Variable

Append options to end of ar command line
export ARTAIL=*options*

The COHERENT archiver **ar** reads the environmental variables **ARHEAD** and **ARTAIL** before it begins its work. You can set these variables to hold the default options that you want the archiver always to use.

ar appends the options in **ARTAIL** to the end of its command line.

See Also

ar, ARHEAD, environmental variables,

Notes

This environmental variable is included only to support existing code. Its use is deprecated, and it may not be supported in future releases of COHERENT.

as — Command

i80386 assembler
as [-o *outfile***] [-bfglnpwxX]** *infile*

The 80386 version of **as**, the COHERENT assembler, assembles programs written in any of several different dialects of assembly language into object modules in COFF format, which can be linked with objects written by the COHERENT C compiler. This version of **as** contains numerous features not available with the COHERENT 286 assembler:

- It serves as a flexible base for writing programs in native 80386 assembly language.
- It assembles programs written in older flavors of COHERENT assembly language.
- It assembles programs written in UNIX assembly language.
- Unlike the old COHERENT assembler and the UNIX assembler, 80386 **as** comes with full macro faculities.
- It is also designed to detect many of the common errors made by assembly-language programmers.

The COHERENT system also includes the command **asfix**, which updates files written in the COHERENT 286 assembler. **asfix** changes local and character symbols to the new format.

Invoking the Assembler

as permits file names and options to be interspersed upon the command line. It recognizes the following command-line options:

-D*name***=***string*
Initialize string variable *name* to *string*. For example, the option

```
-Dname=some_string
```

is equivalent to:

```
name  .define     some_string
```

-E*name***=***value*
Initialize variable *name* to *value*. For example, the option

```
-Ename=17
```

is equivalent to:

```
name  .equ  17
```

-a Set alignment for data objects. For example, when this option is used the express

```
.long 5
```

is automatically aligned to a four-byte boundary, but is left unaligned without it.

-b Reverse bracket sense; that is, use () for expressions and [] for code. For example:

```
movl   $[2 * 5], (%eax)       / without -b
movl   $(2 * 5), [%eax]       / with -b
```

-f Reverse the order of the operands, from UNIX-assembler form to that of the Intel documentation or the 80286 version of **as**.

-g Make undefined symbols **.globl**.

-l Generate an output listing.

-n This option turns off the **as** mechanism for handling bugs in the 80386 chip. **as** tries to cope with known 80386 bugs by changing code at appropriate points in its output. If these changes create problems with your code, you can turn off the **as** bug-handler mechanism by using the **-n** option to **as**.

-o *outfile***.o**
Write the output into *outfile***.o**. Note that the suffix **.o** must appear in the output file's name, or the assembler will exit with an error message. The default output file is *infile***.o**.

-p Don't use '%' on register names; e.g., use **ax**, not **%ax**.

-Q Quiet: Suppress all error messages, no matter how awful an error they indicate.

-w Disable warning messages.

-x Remove all non-global symbols from the common symbol output.

-X Remove all non-global symbols starting with **.L** from the common symbol output.

as reads the environmental variables **ASHEAD** and **ASTAIL** and appends them to, respectively, the beginning and the end of its command line. By setting these variables, you can ensure that **as** always executes with the switches that you want. For example, to ensure that **as** always executes with the **-g** switch set, insert the following into your **.profile**:

```
     export ASHEAD=-g
```

Lexography

A symbol consists of from one to 256 characters. The assembler defines a *character* as being an alphabetic character, question mark, period, percent sign, or underscore. **Xyz**, **.20**, and **hi_there** are legal symbols; whereas **85i** is not.

Like C, the **as** assembly language is case sensitive.

Local symbols begin with a question mark. These are recognizable (or *visible*) only between nonlocal symbols. For example:

```
/       ?loop invisible here
abc     mov     $10, %cx
?loop   add     $1, %bx             / ?loop visible here
        jcxz    xyz
        jmp     ?loop
xyz:
/       ?loop invisible here
```

An octal number is defined just as in the C language: it consists of an initial **0** plus two other numerals between 0 and 7. For example, **077** is a legal octal number.

A hexadecimal number consists of an initial **0x** or **0X** plus two other numerals: 0 through 9, a through f, or A through F. For example, **0x0F** and **0Xa3** are legal hexadecimal numbers.

A binary number consists of an intial **0b** or **0B** followed by an indefinite number 0's and 1's. For example, **0b01001010** is a legal binary number.

A decimal number begins with a numeral other than 0, followed by an indefinite number of numerals between 0 and 9. For example, **109** is a legal decimal number.

A floating-point number begins is a string of numerals, 0 through 9, with a period or **e** within or at the end of it. It is like a C floating-point number, except that it cannot begin with a period because a symbol may begin with a period. For example, **123.456**, **123456.**, and **17e26** are legal floating-point numbers, but **.123456** is not.

A character constant is enclosed between apostrophes, as in C. **as** recognizes the same escape sequences as C. See the Lexicon article **C language** for a table of these constants.

String constants are enclosed between quotation marks, as in the C language, and use the same escape sequences as C. See the Lexicon article **C language** for a table of these sequences.

Pseudo-Opcodes

as recognizes a rich set of pseudo-opcodes. These are not true assembly-language opcodes, but are interpreted by the assembler; they are designed to help make your life easier. The following briefly summarizes the pseudo-opcodes.

.16	16-bit mode
.2byte	Make unaligned short variables
.32	32-bit mode
.4byte	Make unaligned long variables
.align	Increment location counter to two- or four-byte aligned spot
.alignoff	Turn alignment off
.alignon	Turn alignment on
.blkb	Set tag in **.data**
.bracketnorm	Normal bracket sense — see **-b** option
.brcketrev	Reverse bracket sense — see **-b** option
.bss	Set tag in **.bss**
.bssd	Set tag in **.bss**
.byte	Make byte variables
.comm	Set label as common
.data	Change segment to **.data**
.def	Reserved to set auxiliary symbol entries in a later release
.define	Define string constant

.dim	Reserved to set auxiliary symbol entries in a later release
.double	Make double variables
.eject	Force a page break
.else	Connected to **.if**
.endef	Reserved to set auxiliary symbol entries in a later release
.endi	End **.if**
.endm	End **.macro** definition
.endw	End **.while**
.equ	Define numeric constant
.errataoff	Turn off chip errata fixes
.errataon	Turn on chip errata fixes
.even	Increment location counter to byte-aligned spot
.fail	Print error message
.file	Reserved to set auxiliary symbol entries in a later release
.float	Make **float** variables
.globl	Declare names as visible to linker
.ident	**.ident** string
.if	Compile-time conditional
.include	Include a file
.intelorder	Intel operand order — see **-f** option
.lcomm	Set name up as common
.line	Reserved to set auxiliary symbol entries in a later release
.list	Turn on listing (assumes **-l** option)
.llen	Set print line length
.ln	Reserved to set auxiliary symbol entries in a later release
.long	Make long variables
.macro	Define a macro name
.mexit	Exit current macro expansion
.mlist	Toggle listing of macro expansion
.nolist	Turn off listing (assumes **-l** option)
.nopage	Turn off page breaks and titles
.number	Convert a string to a number.
.org	Change location counter
.page	Turn on page breaks and titles
.plen	Set page length
.prvd	Change segment to **.data**
.prvi	Change segment to **.text**
.scl	Reserved to set auxiliary symbol entries in a later release
.set	Makes name equal to expr
.shift	Shift macro parameters
.shrd	Change segment to **.data**
.shri	Change segment to **.text**
.size	Reserved to set auxiliary symbol entries in a later release
.string	Convert a floating-point expression to a string
.strn	Change segment to **.data**
.tag	Reserved to set auxiliary symbol entries in a later release
.text	Change segment to **.text**
.ttl	Set page titles
.type	Reserved to set auxiliary symbol entries in a later release
.undef	Free string, numeric constant, or opcode
.unixorder	Return normal order of operands; undoes **.intelorder**
.val	Reserved to set auxiliary symbol entries in a later release
.value	Make short variables
.version	Comment string
.warn	Print a warning message
.warnoff	Turn off warning messages
.warnon	Turn on warning messages
.while	Compile-time loop control
.word	Make short variables
.zero	Create zero-filled memory

Each pseudo-opcode is described in the following sections.

Input Format

An assembly-language program consists of a series of lines with the following format:

```
[#][label]  [opcode]    [operands]  [/ comment]
```

The optional '#' at the beginning of the line tells **as** not to replace any **.define** symbols within the line. (These are described below.) Normally, the assembler replaces all **.define** symbols in a line before it parses that line. Without this option, a series of **.define**s could lead to awkward results.

For example, the code

```
#%ecx .define     xx
#xx   .define     (%ecx)
      mov   $3, %ecx
```

results in:

```
      mov   $3, (%ecx)
```

Like the C compiler, **as** will not go into an infinite loop if two **.define** statements mirror each other.

A comment begins with a slash '/' and may include the entire line. Blank lines are also legal.

Extra operands are not assumed to be comments. This is to tighten up error checking for the convenience of new and part-time assembly-language programmers.

Expression Format

The **as** macro assembler has mostly the same operators and precedence as the C preprocessor. The exceptions are **?:**, **&&**, **||**, **:**, and ',' (which are missing), '/' (which is spelled **.div**), and '%' (which is spelled **.rem**).

In addition, the macro assembler includes the following directives: **.defined**, **.sizeof**, **.segment**, **.parmct**, **.location**, **.string**, **.number**, and **.float**.

Expression bracketing is normally done by **[]**, because **()** is used by the operand format. This may be reversed by the **-b** option, described above.

The unary operators have the following priority:

.float .number .string	Conversion
.defined .sizeof	
.location .segment	Inquiry
-	Negation
!	Logical negation

The binary operators have the following priority:

[]	
*** .div .rem**	Multiply, divide, remainder
+ -	Add, subtract.
>> <<	Left shift , right shift
< > <= >= == !=	Comparison
&	AND
^	Exclusive OR
|	OR
#	Repeat

Most binary operators should be familiar to C programmers; the exception is the **#**, which repeats an instruction *N* times. For example, the expression

```
      .byte 5 # 3
```

produces five copies of byte 3, whereas the expression

```
      .long 7 # 4
```

produces seven copies of the long '4'. Note that this operator has the lowest precedence of all binary operators.

You can use an expression wherever you can use a number. This includes address displacements, constants, and

.if and **.while** statements. Integers are internally 32 bits, floats are internally C doubles.

Like C, comparison operators return one for true and zero for false.

In addition, **as** provides string operators. Like C, the first element of a string is indexed as zero. Unlike C, however, attempts to access past the end of a string gives all zeroes. The following summarizes the **as** suite of string operators:

string **+** *string*
: Concatenate two strings. For example, **"12" + "34"** yields **"1234"**.

string **[** *expr1***,** *expr2* **]**
: Address a substring from *expr1* to *expr2*. For example, **"1234567"[1,3]** yields **"234"**; and **"123"[1,10]** yields **"23"**.

string **[** *expr* **]**
: Address a substring from *expr* to the end of the string. For example **"1234567"[5]** yields **"67"**.

.string *expr*
: Convert a numeric expression to a string. For example, **.string 123** gives **"123"**.

.string *float*
: Convert a floating-point expression to a string. For example, **.string 0.5 * 3** gives **"1.5"**

.float *string*
: Convert a string to a floating-point number.

.float *expr*
: Convert a numeric expression to a floating-point number.

.number *string*
: Convert a string to a number.

.number *float*
: Convert a floating-point number to a number.

.string (*expr* **)**
: Return character at position *expr* as a number. For example, **"123"(1)** gives two.

string1 @ *string2*
: Return the position at which *string2* begins within *string1*. For example, **"12345" @ "23"** returns one; and **"123" @ "jj"** gives -1 (because “jj” does not appear within “123”).

The unary operator **:** creates a label equal to the current location. It is generally not needed. For example, the expression

```
connected   .long 5
```

builds an aligned **long**, initializes it to five, and gives it the label **connected**. However, the expression

```
unconnected:      .long 5
```

builds the label **unconnected** at the current location, then builds an aligned **long** with a value of five. Note that the label **connected** will be on the five, whereas the label **unconnected** may be somewhere else if there was alignment. For example, the expression

```
      .align      4
lab1: lab2: lab3: .long 5
```

puts **lab1**, **lab2**, and **lab3** on the **long** because it is already aligned.

Macros and Conditional Compilation

The **as** directive **.macro** lets you declare a macro that you can use through a program. The directive **.endm** marks the end of a macro declaration.

A macro has the following form:

name **.macro** *params*
body of macro
.endm

The following example creates and uses the macro **store**:

```
store   .macro  xy,xz                   / declare "store" with two parms: xy and xz
        movl    xy,%ecx
        movl    %ecx,(%eax)
        movl    xz,%ecx
        movl    %ecx,4(%eax)
        .endm                           / end of macro

        store   5,10                    / moves 5 and 10 to where %eax points.
```

Macros can contain **.if** statements, and can even define other macros. For example:

```
def     .macro  .name, to               / macro for defining other macros
name    .macro
        movl    from, to
        .endm
        .endm

        def     frog, %eax, %ebx        / define the macro frog
        frog                            / movel%eax, %ebx
```

as increments a count every time you expand any macro, and associates that number with the macro. When the keyword **.macno** is used within a macro, **as** translates it into that number. Thus, **.macno** is a unique number within each macro expansion. This allows the generation of unique labels internal to macros. For example:

```
stradd  .macro  str
        .data
L\.macno        .byte str, 0            / create a data item
        .text
        movl    L\.macno, %eax          / put its address into %eax
        .endm
```

L\.macno becomes something like L51. Note that a '\' before any defined symbol or macro name vanishes in the expansion pass.

To permit macros with indefinite parameter counts, the assembler offers the reserved word **.parmct** and the command **.shift**. The former holds the number of parameters passed to a macro, and the latter shifts the parameters one position to the left. For example:

```
kall    .macro  fun, parm
        .while .parmct > 1              / while more than one parm remains
        push parm
        .shift                          / parm 3 becomes parm 2, parm 4 parm 3 etc
        call fun
        .endm
```

The operators **.if**, **.else**, and **.endi** allow a program to implement compile-time decisions. These may be inside or outside of macros. When a macro exits, the assembler automatically closes all **.if** statments that had been started within it. For example:

```
defy    .macro
        .if .defined y                  / if y has been defined true
        .mexit                          / exits closing any if statements
        .else
y       .equ 1  / define y as 1
/ For UNIX compatibility
/       .set    y, 1
/ produces the same result
        .endm
```

When used with a label, the operator **.defined** is true if that label had been defined in this pass. If the label is defined later, **.defined** can still be used with it, but causes a phase error, as occurs in some assemblers.

The operator **.fail** permits the flagging of errors. For example:

```
      .if ! .defined y
      .fail y is not defined
      .endi
```

The operator **.include** permits the inclusion of files. For example:

```
      .include    somefile.h
```

Undefining Symbols or Opcodes

as Some software (e.g., the GNU C compiler) requires that opcodes be recognized on column one and that opcodes be replacable by macros. The command **.undef** un-defines all macros and opcodes. Once you have un-defined an identifier, you can re-use it to name a macro or other data item. For example, to use **mov** (which names an opcode) to name a macro, do the following:

```
      .undef      mov
mov   .macro      foo, bar
      movl  foo, bar
      .endm
```

Data-Definition Operators

The following describes the data-definition operators that **as** supports.

.byte *expr*
Define *expr* as an array of single bytes. *expr* can take any number of forms, as shown by the following examples:

```
        .byte   5, 2                / defines 2 bytes 0x05 and 0x02
        .byte   "Hello World", 0    / a zero-terminated Hello World
        .byte   10 # 1              / 10 repetitions of 0x01
```

.word *expr*
Define *expr* as a word, that is, as a two-byte integer. For example:

```
        .word   .sizeof xx          / a short the size of xx
        .word   50 * 50             / a short of 100
/ For UNIX compatability
/       .value  50 * 10
/ produces the same result.
```

.long *expr*
Define *expr* as a long (four-byte) integer. For example:

```
        .long   10                  / a long of 10
```

.comm *name***,** *length*
Define a common variable named *name*, that is *length* bytes long. (See the entry for **.lcomm**, below, for a discussion of what segment the variable is stored.) If *name* is linked with another module that also declares *name* but sets it to another length, the linker creates one such variable and gives it the greater length of the two.

The linker deduces the alignment of a common variable from its length: if the length of a common is divisible by four, it is aligned on a four-byte boundary; if it is divisible by two, it is aligned on a two-byte boundary. Otherwise, it is assumed to be unaligned. The linker supports only three classes of alignment: four-byte, two-byte, and unaligned.

A common variable is aligned according to its most strongly aligned contributor. For example, if one module contributes a common variable named **xyz** whose length is four bytes, and another contributes an **xyz** whose length is five bytes, the resulting **xyz** is given a length of eight bytes to satisfy the length requirement (at least five) and the alignment requirement (four-byte boundary).

After the first linker pass, all common variables are placed at the end of the **.bss** segment: first the four-byte-aligned variables, then the two-byte-aligned, then the unaligned.

By default, **as** does not align its data objects. The command-line option **-a** instructs **as** to align all data objects automatically.

.lcomm *label***,** *length*
Same as **comm**, described above.

Please note that on a COFF-based system, it is not possible to put common data into the **.data** section, even though the UNIX assembler documentation claims that **.comm** does this. Both **.comm** and **.lcomm** place data into the **.bss**.

The problem is that COFF format for common variables leaves no place for information about alignment or segment. This creates two problems. First, the lack of information about alignment forces COFF to adopt the complex strategy of deducing alignment from length. Second, the lack of information about segment compels COFF to store all common variables in one segment, **.bss** being chosen.

.float *expr*
Define *expr* as a single-precision floating-point number. For example:

```
        .float  1.5                   / a float of 1.5
```

.double *expr*
Define *expr* as a double-precision floating-point number. For example:

```
        .double 3.0 * 0.5             / a double of 1.5
```

Resetting the Location Counter

The instructions **.org** and **.align** reset the location counter. For example:

```
        .org    .+5                 / Location counter to here plus 5
        .org                        / Location counter to top of current section
        .align  2                   / Up to nearest two-byte boundary
```

The pseudo-opcodes **.alignon** and **.alignoff** respectively turn aligning on and off.

As noted above, the command-line option **-a** instructs **as** to align all data objects automatically.

The instructions **.text**, **.data**, and **.bss** reset the location counter to the corresponding sections. Instructions are placed in the **.text** section, initialized data in the **.data** section, and the **.bss** is reserved for unitialized data. Placing information into the **.bss** results in an error.

Dynamic Linking

The Intel Binary Compatibility Standard dictates the way that **as** computes addresses, to permit dynamic linking of objects.

In object files, all **.data** addresses must follow all **.text** addresses, and all **.bss** address must follow all **.data** addresses. This allows dynamic linking of object files, in which the object file is mapped, not read in pieces.

In the **as** assembly language, **.data** and **.text** addresses are started from 0 for each module. At the end of assembly, during the output phase, **as** fixes these addresses to make **.data** follow **.text**, and so on.

For example, if you have a conditional like

```
      .if   some_data_address > 0x300
```

as calculates the address for the **.if** statement from the beginning of its segment; and the address is only corrected in the final output. Such statements may appear to be working incorrectly.

Listing Commands

as prints a listing if you use its **-l** option. The following commands modify the form of this listing.

.ttl *string*
Print *string* as the title to the command page. For example:

```
        .ttl  This is a page title
```

If you do not use this command, the assembler uses the file name for the title. The first **.ttl** encountered in the assembly pass 0 is used to set the first title. Subsequent **.ttl** commands reset the title before printing.

.nopage
Turn off page breaks and titles.

.page Turn on page breaks and titles.

.eject Force a page break.

.nolist Turn off the listing.

.list Turn the listing back on.

.mlist off
Turn off the listing of macro expansions.

.mlist on
Turn on the listing of macro expansions.

Addressing Modes

as recognizes two modes of addressing: *16-bit mode* and *32-bit mode*. In 16-bit mode, the address type and operand mode default to 16 bits; in 32-bit mode they default to 32 bits. For example:

```
.16
movw   %ax, (%si)     # Is generated without escapes.
movl   %eax, (%esi)   # Has two escapes, address and operand
.32
movw   %ax, (%si)     # Has two escapes, address and operand
movl   %eax, (%esi)   # Is generated without escapes.
```

In 16-bit mode, the 16-bit addressing forms in table 17-2 of the *Intel 386 Programmer's Manual* are generated where they fit; otherwise, an address escape is built and the 32-bit forms in tables 17-3 and 17-4 are used. In 32 bit mode, this is reversed.

as uses the following grammar in its addressing modes:

Eight-bit registers

```
r8  : %al | %cl | %dl | %bl | %ah | %ch | %dh | %bh;
```

16-bit registers

```
r16 : %ax | %cx | %dx | %bx | %sp | %bp | %si | %di;
```

32-bit registers

```
r32 : %eax | %ecx | %edx | %ebx | %esp | %ebp | %esi | %edi;
```

Segment registers

```
sreg : %es | %cs | %ss | %ds | %fs | %gs;
```

Control registers

```
ctlreg : %cr0 | %cr2 | %cr3;
```

Debug registers

```
dbreg : %dr0 | %dr1 | %dr2 | %dr3 | %dr6 | %dr7;
```

Test registers

```
testreg : %tr6 | %tr7;
```

m16 These addresses can have a segment prefix:

```
m16 : m16b | sreg ':' m16b;
```

m32 These addresses can have a segment prefix:

```
m32 : m32b | sreg ':' m32b;
```

rm16 These addresses can have a segment prefix or may be **r16**:

```
rm16 : rm16b | sreg ':' rm16b;
```

rm32 These addresses can have a segment prefix or may be **r32**:

```
rm32 : r32 | rm32b | sreg ':' rm32b;
```

rm8 These addresses can be **rm32**, **rm16**, or **r8**:

```
rm8  : r8 | rm16b | sreg ':' rm16b | rm32b | sreg ':' rm32b;
```

rm16b

```
displacement | (vx, vy) | displacement(vx, vy) |
      displacement(vw) | (vz);
vx : %bx | %si;
vy : %si | %di;
vz : %si | %di | %bx;
```

rm32b

```
(va) | displacement(vb) | (, vb, scale) | (vb, scale)
     | displacement(vb, scale) | (vb, vb, scale)
     | displacement(vb, vb, scale);
va : %eax | %ecx | %edx | %ebx | %esi | %edi;
vb : %eax | %ecx | %edx | %ebx | %ebp | %esi | %edi;
vb : %eax | %ecx | %edx | %ebx | %ebp | %esp | %esi | %edi;
scale : 0 | 1 | 2 | 4 | 8;
```

mem32
A 32-bit memory address.

mem16
A 16-bit memory address.

reli Expand to eight-, 16-, or 32-bit relative addresses.

rel8 Eight-bit relative addresses.

rel16 Sixteen- or 32-bit relative addresses.

Using as To Create Debug Information

Some UNIX languages, such as **gcc** and **gc++**, produce assembly language rather than object code. The following documents how to use **as** with such compilers. Note that error checking is minimal, and that bad debug information can corrupt the generated COFF output. This section must be read with a listing of the header file **coff.h** for reference; or see the Lexicon article **coff.h**.

The compiler starts with type and line information in a format much like that of the desired COFF output files. It must break this down into lines to ship through the assembler, and the assembler then must rebuild the information into COFF format for output.

.file *filename*
This connects the object file to the original source file. If used, this should be the first statement in the file. It produces a **SYMENT** of *n_sclass* = *C_FILE* and an **AUXENT** with *ae_fname* = *filename*.

.def *symbolName*
This instruction initializes **SYMENT** with *n_name* = *symbolName*. If there is a symbol by that name on the assembler's internal symbol table, it is marked to prevent output to the symbol table. Any **RELOC** references point to this table entry, so its *n_value* must be correct. Because we assume that code of this kind is result of a compiler, we assume it is correct. The following commands up to and including **.endef** refer to this **SYMENT**.

.type *number*
This sets this **SYMENT**'s *n_type number*. If *number* indicates a function, **DT_FCN**, a **LINENO** record is built pointing at this **SYMENT**.

.val [*symbol***] [***number***]**
This sets this **SYMENT**'s *n_value*. If it is a *symbol*, it sets *n_scnum* to the symbol's section number.

.scl *number*
This sets this **SYMENT**'s *n_sclass* to *number*.

.dim *number* **[,** *number* **[,** *number* **[,** *number***]]]**
This sets up to four entries in an **AUXENT**'s *ae_dimen*. It describes multidimensioned arrays to COFF. This command supports only four dimensions because the COFF specifications are reliable only though four dimensions.

.size *n* This sets this **AUXENT** *ae_size* to *n*.

.tag *name*
This scans backwards on the **SYMENT**s for a matching *n_name*. It points this *ae_tagndx* to that name's symbol number and that *ae_endndx* to the next symbol *number*.

A good example is a **struct**: It would start with a **SYMENT** of type **T_STRUCT**, then then have **SYMENT**s for its members. At the end, there would be a **C_EOS** (end of structure) with a tag that gets us back to the symbol's name. **.tag** connects the forward and backward pointers.

.line *n* This sets the **AUXENT**'s *ae_lnno* to *n*.

.endef This marks the end of a **SYMENT** started by **.def**. If the *n_sclass* **== C_EFCN** (end of function), it builds the functions *ae_fsize* and *ae_endndx* and does not output this **SYMENT**. If any **AUXENT** fields were set, an **AUXENT** record follows this **SYMENT**.

.ln *number*
This builds a **LINENO** record with **l_lnno =** *n* and **l_addr.l_paddr =** the current location.

Instructions

In matching instructions, **as** first looks up the name of the instruction. A number of actual instructions will match that name. For example, **btsw** matches 0xab and 0x0fab /5, and **bts** matches anything that matches **btsw** and **btsl**.

as attempts to match operands to the instruction until a form is found that will accept all the operands. If no form matches all the operands, **as** prints the error message

```
Illegal combination of opcode and operands
```

The assembler at that point cannot say which operand is wrong because of the nature of the 80386 instruction set.

If you see a great number of these messages, **as**'s command-line option **-f** may be in the wrong sense: although the opcode is valid and the operands are valid, there is no form of this opcode that takes these operands in this order.

as first attempts to match opcodes that do not require an operand-mode escape: that is, in 80386 mode it attempts to match long-mode instructions first, then short-mode instructions.

Register Usage

The COHERENT C compiler uses the following save/restore sequence for a function, to set the frame pointer when the function contains no automatic variables:

```
push  %ebp
movl  %ebp, %esp
```

If *n* bytes of autos are required, then it uses the following sequence:

```
enter $n, $0
```

It then executes the code

```
push  %esi
push  %edi
push  %ebx
```

to preserve register variables *as required:* they are not saved/restored if the function does not touch them. (This is why they are saved after the frame adjust, not before). To restore register variables, it executes

```
pop   %ebx
pop   %edi
pop   %esi
```

as required, followed by

```
leave
ret
```

Routines written in assembly language must preserve registers **ebp**, **esi**, **edi**, and **ebx**; they may overwrite **eax**, **ecx**, and **edx**.

Absolute Symbols

as can create what COFF calls "absolute symbols." For example

```
      .globl      x
x     .equ  10
x     .equ  x * x / The last value of x in the module
```

leaves on the symbol table an absolute symbol for **x** of 100. For internal reason, the **.globl** must preceed any **.equ**.

Opcodes

The following gives a table of the opcodes recognized by **as**. Note that the opcode is sometimes followed by a slash and a number, or a letter. For example,

```
      D0 /4 salb  con1, rm8
```

means opcode is 0xD0 place 4 in the register/opcode field of the **modr/m** byte.

```
      58 +r popl  r32
```

means add the register number to 0x58.

Opcode	*Instruction*	*Operands*	*Description*
37	**aaa**		Adjust after addition
D5 0A	**aad**		Adjust AX before division
D4 0A	**aam**		Adjust AX after multiply
3F	**aas**		Adjust AL after subtraction
	adc		Add with carry
83 /2	**adcl**	*imm8s,rm32*	
83 /2	**adcw**	*imm8s,rm16*	
14	**adcb**	*imm8,al*	
15	**adcw**	*imm16,ax*	
15	**adcl**	*imm32,eax*	
15	**adcl**	*imm32*	
80 /2	**adcb**	*imm8,rm8*	
81 /2	**adcw**	*imm16,rm16*	
81 /2	**adcl**	*imm32,rm32*	
12 /r	**adcb**	*rm8,r8*	
13 /r	**adcw**	*rm16,r16*	
13 /r	**adcl**	*rm32,r32*	
10 /r	**adcb**	*r8,rm8*	
11 /r	**adcw**	*r16,rm16*	
11 /r	**adcl**	*r32,rm32*	
	add		Add
83 /0	**addl**	*imm8s,rm32*	
83 /0	**addw**	*imm8s,rm16*	
04	**addb**	*imm8,al*	
05	**addw**	*imm16,ax*	
05	**addl**	*imm32,eax*	
05	**addl**	*imm32*	
80 /0	**addb**	*imm8,rm8*	
81 /0	**addw**	*imm16,rm16*	
81 /0	**addl**	*imm32,rm32*	
02 /r	**addb**	*rm8,r8*	
03 /r	**addw**	*rm16,r16*	
03 /r	**addl**	*rm32,r32*	
00 /r	**addb**	*r8,rm8*	
01 /r	**addw**	*r16,rm16*	
01 /r	**addl**	*r32,rm32*	
	and		Logical AND
83 /4	**andl**	*imm8s,rm32*	
83 /4	**andw**	*imm8s,rm16*	
24	**andb**	*imm8,al*	
25	**andw**	*imm16,ax*	
25	**andl**	*imm32,eax*	
25	**andl**	*imm32*	

80 /4	**andb**	*imm8,rm8*	
81 /4	**andw**	*imm16,rm16*	
81 /4	**andl**	*imm32,rm32*	
22 /r	**andb**	*rm8,r8*	
23 /r	**andw**	*rm16,r16*	
23 /r	**andl**	*rm32,r32*	
20 /r	**andb**	*r8,rm8*	
21 /r	**andw**	*r16,rm16*	
21 /r	**andl**	*r32,rm32*	
63 /r	**arpl**	*r16,rm16*	Adjust RPL field of selector
	bound		Check if register is within bounds
62 /r	**boundw**	*m16,r16*	
62 /r	**boundl**	*m32,r32*	
	bsf		Bit scan forward
0F BC	**bsfw**	*rm16,r16*	
0F BC	**bsfl**	*rm32,r32*	
	bsr		Bit scan reverse
0F BD	**bsrw**	*rm16,r16*	
0F BD	**bsrl**	*rm32,r32*	
	bt		Save bit in carry flag
0F A3	**btw**	*r16,rm16*	
0F A3	**btl**	*r32,rm32*	
0F BA /4	**btw**	*imm8,rm16*	
0F BA /4	**btl**	*imm8,rm32*	
	btc		Bit test and complement
0F BB	**btcw**	*r16,rm16*	
0F BB	**btcl**	*r32,rm32*	
0F BA /7	**btcw**	*imm8,rm16*	
0F BA /7	**btcl**	*imm8,rm32*	
	btr		Bit test and reset
0F B3	**btrw**	*r16,rm16*	
0F B3	**btrl**	*r32,rm32*	
0F BA /6	**btrw**	*imm8,rm16*	
0F BA /6	**btrl**	*imm8,rm32*	
	bts		Bit test and set
0F AB	**btsw**	*r16,rm16*	
0F AB	**btsl**	*r32,rm32*	
0F BA /5	**btsw**	*imm8,rm16*	
0F BA /5	**btsl**	*imm8,rm32*	
E8	**call**	*reli*	Call Procedure
98	**cbtw**		Sign extend AL
98	**cbw**		Sign extend AL
99	**cdq**		Double word to quad word
F8	**clc**		Clear carry
FC	**cld**		Clear direction Flag
FA	**cli**		Clear interrupt Flag
99	**cltd**		Double word to quad word
0F 06	**clts**		Clear task-switched flag in CR0
F5	**cmc**		Complement carry flag
	cmp		Compare
83 /7	**cmpl**	*imm8s,rm32*	
83 /7	**cmpw**	*imm8s,rm16*	
3C	**cmpb**	*imm8,al*	
3D	**cmpw**	*imm16,ax*	
3D	**cmpl**	*imm32,eax*	
3D	**cmpl**	*imm32*	

80 /7	**cmpb**	*imm8,rm8*	
81 /7	**cmpw**	*imm16,rm16*	
81 /7	**cmpl**	*imm32,rm32*	
3A /r	**cmpb**	*rm8,r8*	
3B /r	**cmpw**	*rm16,r16*	
3B /r	**cmpl**	*rm32,r32*	
38 /r	**cmpb**	*r8,rm8*	
39 /r	**cmpw**	*r16,rm16*	
39 /r	**cmpl**	*r32,rm32*	
A6	**cmpsb**		Compare bytes
A7	**cmpsl**		Compare long
A7	**cmpsw**		Compare words
99	**cwd**		Word to double word
98	**cwde**		Sign extend AX
99	**cwtd**		Word to double word
98	**cwtl**		Sign extend AX
27	**daa**		Decimal adjust after addition
2F	**das**		Decimal adjust after subtraction
	dec		Decrement by 1
48 +r	**decw**	*r16*	
48 +r	**decl**	*r32*	
FE /1	**decb**	*rm8*	
FF /1	**decw**	*rm16*	
FF /1	**decl**	*rm32*	
	div		Unsigned divide
F6 /6	**divb**	*rm8,al*	
F6 /6	**divb**	*rm8*	
F7 /6	**divw**	*rm16,ax*	
F7 /6	**divw**	*rm16*	
F7 /6	**divl**	*rm32,eax*	
F7 /6	**divl**	*rm32*	
C8	**enter**	*imm8,imm16*	Make stack frame for procedure
D9 F0	**f2xm1**		ST = 2 ** ST - 1
D9 E1	**fabs**		ST = abs(ST)
	fadd		Floating add
D8 /0	**fadds**	*m32*	
DC /0	**faddl**	*m64*	
D8 C0 +r	**fadd**	*fpreg,st0*	
D8 C0 +r	**fadd**	*fpreg*	
DE C1	**fadd**		
DC C0 +r	**fadd**	*st0,fpreg*	
	faddp		Floating add and pop
DE C0 +r	**faddp**	*st0,fpreg*	
DE C0 +r	**faddp**	*fpreg*	
DE C1	**faddp**		
DF /4	**fbld**	*m80*	Load Binary Coded Decimal
DF /6	**fbstp**	*m80*	Store Binary Coded Decimal and Pop
D9 E0	**fchs**		Change Floating Sign
9B DB E2	**fclex**		Clear floating point exception flags
	fcom		Floating Compare
D8 /2	**fcoms**	*m32*	
DC /2	**fcoml**	*m64*	
D8 D0 +r	**fcom**	*fpreg,st0*	
D8 D0 +r	**fcom**	*fpreg*	
D8 D1	**fcom**		
	fcomp		Floating Compare and Pop

D8 /3	**fcomps**	*m32*	
DC /3	**fcompl**	*m64*	
D8 D8 +r	**fcomp**	*fpreg*	
D8 D9	**fcomp**		
DE D9	**fcompp**		Floating Compare and pop twice
D9 FF	**fcos**		Cosine
D9 F6	**fdecstp**		Decrement Stack Top Pointer
	fdiv		Floating divide
D8 /6	**fdivs**	*m32*	
DC /6	**fdivl**	*m64*	
D8 F0 +r	**fdiv**	*fpreg,st0*	
D8 F0 +r	**fdiv**	*fpreg*	
DE F1	**fdiv**		
DC F0 +r	**fdiv**	*st0,fpreg*	
	fdivp		Floating divide and pop
DE F0 +r	**fdivp**	*st0,fpreg*	
DE F0 +r	**fdivp**	*fpreg*	
DE F1	**fdivp**		
	fdivr		Reverse floating divide
D8 /7	**fdivrs**	*m32*	
DC /7	**fdivrl**	*m64*	
D8 F8 +r	**fdivr**	*fpreg,st0*	
D8 F8 +r	**fdivr**	*fpreg*	
DE F9	**fdivr**		
DC F8 +r	**fdivr**	*st0,fpreg*	
	fdivrp		Reverse floating divide and pop
DE F8 +r	**fdivrp**	*st0,fpreg*	
DE F8 +r	**fdivrp**	*fpreg*	
DE F9	**fdivrp**		
DD C0 +r	**ffree**	*fpreg*	Free Floating Point Register
	fiadd		Add integer to float
DA /0	**fiaddl**	*m32*	
DE /0	**fiadds**	*m16*	
	ficom		Compare float to integer
DA /2	**ficoml**	*m32*	
DE /2	**ficoms**	*m16*	
	ficomp		Compare float to integer and pop
DA /3	**ficompl**	*m32*	
DE /3	**ficomps**	*m16*	
	fidiv		Divide float by integer
DA /6	**fidivl**	*m32*	
DE /6	**fidivs**	*m16*	
	fidivr		Reverse divide integer by float
DA /7	**fidivrl**	*m32*	
DE /7	**fidivrs**	*m16*	
	fild		Load integer
DB /0	**fildl**	*m32*	
DF /0	**filds**	*m16*	
DF /5	**fildll**	*m64*	
	fimul		Multiply integer to float
DA /1	**fimull**	*m32*	
DE /1	**fimuls**	*m16*	
D9 F7	**fincstp**		Increment Stack Top Pointer

9B DB E3	**finit**		Initialize Floating Point Unit
	fist		Store integer
DB /2	**fistl**	*m32*	
DF /2	**fists**	*m16*	
	fistp		Store integer and pop
DB /3	**fistpl**	*m32*	
DF /3	**fistps**	*m16*	
DF /7	**fistpll**	*m32*	
	fisub		Subtract integer from float
DA /4	**fisubl**	*m32*	
DE /4	**fisubs**	*m16*	
	fisubr		Reverse subtract integer from float
DA /5	**fisubrl**	*m32*	
DE /5	**fisubrs**	*m16*	
	fld		Load Real
D9 C0 +r	**fld**	*fpreg*	
D9 /0	**flds**	*m32*	
DD /0	**fldl**	*m32*	
DB /5	**fldt**	*m64*	
D9 E8	**fld1**		Load Constant 1
D9 /5	**fldcw**	*m32*	Load Floating Point Control Word
D9 /4	**fldenv**	*m32*	Load FPU Environment
D9 EA	**fldl2e**		Load Constant log(e) base 2
D9 E9	**fldl2t**		Load Constant log(10) base 2
D9 EC	**fldlg2**		Load Constant log(2) base 10
D9 ED	**fldln2**		Load Constant log(2) base e
D9 EB	**fldpi**		Load Constant pi
D9 EE	**fldz**		Load Constant 0.0
	fmul		Floating multiply
D8 /1	**fmuls**	*m32*	
DC /1	**fmull**	*m64*	
D8 C8 +r	**fmul**	*fpreg,st0*	
D8 C8 +r	**fmul**	*fpreg*	
DE C9	**fmul**		
DC C8 +r	**fmul**	*st0,fpreg*	
	fmulp		Floating multiply and pop
DE C8 +r	**fmulp**	*st0,fpreg*	
DE C8 +r	**fmulp**	*fpreg*	
DE C9	**fmulp**		
DB E2	**fnclex**		Clear floating point exception flags no wait
DB E3	**fninit**		Initialize Floating Point Unit no wait
D9 D0	**fnop**		No Operation
DD /6	**fnsave**	*m32*	Store FPU State no wait
D9 /7	**fnstcw**	*m32*	Store Control Word no wait
D9 /6	**fnstenv**	*m32*	Store FPU Environment no wait
	fnstsw		Store Status Word no wait
DD /7	**fnstsw**	*m16*	
DF E0	**fnstsw**	*ax*	
D9 F3	**fpatan**		Partial Arctangent
D9 F8	**fprem**		Partial Remainder toward 0
D9 F5	**fprem1**		Partial Remainder < 1/2 modulus
D9 F2	**fptan**		Partial Tangent
D9 FC	**frndint**		Round To Integer
DD /4	**frstor**	*m32*	Resore FPU State
DB F4	**frstpm**		set 287XL real mode (nop for 387/486)

9B DD /6	**fsave**	*m32*	Store FPU State
D9 FD	**fscale**		Scale
DB E4	**fsetpm**		set 287 protected mode (nop for 387/486)
D9 FE	**fsin**		Sine
D9 FB	**fsincos**		Sine and Cosine
D9 FA	**fsqrt**		Square Root
	fst		Store Real
DD D0 +r	**fst**	*fpreg*	
D9 /2	**fsts**	*m32*	
DD /2	**fstl**	*m64*	
9B D9 /7	**fstcw**	*m32*	Store Control Word
9B D9 /6	**fstenv**	*m32*	Store FPU Environment
	fstp		Store Real and pop
DD D8 +r	**fstp**	*fpreg*	
D9 /3	**fstps**	*m32*	
DD /3	**fstpl**	*m64*	
DB /7	**fstpt**	*m80*	
	fstsw		Store Status Word
9B DD /7	**fstsw**	*m16*	
9B DF E0	**fstsw**	*ax*	
	fsub		Floating subtract
D8 /4	**fsubs**	*m32*	
DC /4	**fsubl**	*m64*	
D8 E0 +r	**fsub**	*fpreg,st0*	
D8 E0 +r	**fsub**	*fpreg*	
DE E1	**fsub**		
DC E0 +r	**fsub**	*st0,fpreg*	
	fsubp		Floating subtract and pop
DE E0 +r	**fsubp**	*st0,fpreg*	
DE E0 +r	**fsubp**	*fpreg*	
DE E1	**fsubp**		
	fsubr		Reverse floating subtract
D8 /5	**fsubrs**	*m32*	
DC /5	**fsubrl**	*m64*	
D8 E8 +r	**fsubr**	*fpreg,st0*	
D8 E8 +r	**fsubr**	*fpreg*	
DE E9	**fsubr**		
DC E8 +r	**fsubr**	*st0,fpreg*	
	fsubrp		Reverse floating subtract and pop
DE E8 +r	**fsubrp**	*st0,fpreg*	
DE E8 +r	**fsubrp**	*fpreg*	
DE E9	**fsubrp**		
D9 E4	**ftst**		Test
	fucom		Unordered compare real
DD E0 +r	**fucom**	*st0,fpreg*	
DD E0 +r	**fucom**	*fpreg*	
DD E1	**fucom**		
	fucomp		Unordered compare real and pop
DD E8 +r	**fucomp**	*st0,fpreg*	
DD E8 +r	**fucomp**	*fpreg*	
DD E9	**fucomp**		
DA E9	**fucompp**		Unordered compare %st %st1 and pop twice
9B	**fwait**		Wait for coprocessor
D9 E5	**fxam**		Examine

	fxch		Floating exchange
D9 C8 +r	**fxch**	*st0,fpreg*	
D9 C8 +r	**fxch**	*fpreg,st0*	
D9 C8 +r	**fxch**	*fpreg*	
D9 C9	**fxch**		
D9 F4	**fxtract**		Extract Exponent and Significand
D9 F1	**fyl2x**		%st1 * log(%st) base 2
D9 F9	**fyl2xp1**		%st1 * log(%st + 1.0) base 2
F4	**hlt**		Halt
FF /2	**icall**	*rm32*	Call indirect
	idiv		Signed divide
F6 /7	**idivb**	*rm8,al*	
F6 /7	**idivb**	*rm8*	
F7 /7	**idivw**	*rm16,ax*	
F7 /7	**idivw**	*rm16*	
F7 /7	**idivl**	*rm32,eax*	
F7 /7	**idivl**	*rm32*	
FF /4	**ijmp**	*rm32*	Jump indirect
FF /3	**ilcall**	*m32*	Long call indirect
FF /5	**iljmp**	*m32*	Long jump indirect
	imul		Signed multiply
F6 /5	**imulb**	*rm8,al*	
F6 /5	**imulb**	*rm8*	
F7 /5	**imulw**	*rm16,ax*	
F7 /5	**imulw**	*rm16*	
F7 /5	**imull**	*rm32,eax*	
F7 /5	**imull**	*rm32*	
0F AF /r	**imulw**	*rm16,r16*	
0F AF /r	**imull**	*rm32,r32*	
6B	**imulw**	*imm8s,rm16,r16*	
6B	**imull**	*imm8s,rm32,r32*	
6B /r	**imulw**	*imm8s,r16*	
6B /r	**imull**	*imm8s,r32*	
69	**imulw**	*imm16,rm16,r16*	
69	**imull**	*imm32,rm32,r32*	
69 /r	**imulw**	*imm16,r16*	
69 /r	**imull**	*imm32,r32*	
	in		Input from port
E4	**inb**	*imm8*	
E5	**inw**	*imm8*	
E5	**inl**	*imm8*	
EC	**inb**	*(dx)*	
ED	**inw**	*(dx)*	
ED	**inl**	*(dx)*	
	inc		Increment by one
40 +r	**incw**	*r16*	
40 +r	**incl**	*r32*	
FE /0	**incb**	*rm8*	
FF /0	**incw**	*rm16*	
FF /0	**incl**	*rm32*	
6C	**insb**		Input byte from port into ES:(E)DI
6C	**insb**	*(dx)*	Input byte from port into ES:(E)DI
6D	**insl**		Input long from port into ES:(E)DI
6D	**insl**	*(dx)*	Input long from port into ES:(E)DI
6D	**insw**		Input word from port into ES:(E)DI
6D	**insw**	*(dx)*	Input word from port into ES:(E)DI
CC	**int**	*con3*	Interrupt 3

CD	**int**	*imm8*	Interrupt
CE	**into**		Int 4 if overflow is 1
CF	**iret**		Interrupt return
CF	**iretd**		Different mode different opcode ?
07	**ja**	*reli*	Jump if above
03	**jae**	*reli*	Jump if above or equal
02	**jb**	*reli*	Jump if below
06	**jbe**	*reli*	Jump if below or equal
02	**jc**	*reli*	Jump if carry
E3	**jcxz**	*rel8*	Jump if CX is zero
04	**je**	*reli*	Jump if equal
E3	**jecxz**	*rel8*	Jump if CX is zero
0F	**jg**	*reli*	Jump if greater
0D	**jge**	*reli*	Jump if greater or equal
0C	**jl**	*reli*	Jump if less
0E	**jle**	*reli*	Jump if less or equal
E9	**jmp**	*reli*	Jump absolute
06	**jna**	*reli*	Jump if not above
02	**jnae**	*reli*	Jump if not above or equal
03	**jnb**	*reli*	Jump if not below
07	**jnbe**	*reli*	Jump if not below or equal
03	**jnc**	*reli*	Jump if no carry
05	**jne**	*reli*	Jump if not equal
0E	**jng**	*reli*	Jump if not greater
0C	**jnge**	*reli*	Jump if not greater or equal
0D	**jnl**	*reli*	Jump if not less
0F	**jnle**	*reli*	Jump if not less or equal
01	**jno**	*reli*	Jump if not overflow
0B	**jnp**	*reli*	Jump if not parity
09	**jns**	*reli*	Jump if not sign
05	**jnz**	*reli*	Jump if not zero
00	**jo**	*reli*	Jump if overflow
0A	**jp**	*reli*	Jump if parity
0A	**jpe**	*reli*	Jump if parity even
0B	**jpo**	*reli*	Jump if parity odd
08	**js**	*reli*	Jump if sign
04	**jz**	*reli*	Jump if zero
04	**jz**	*reli*	Jump if zero
9F	**lahf**		Load flags into AH register
	lar		Load access rights byte
0F 02 /r	**larw**	*rm16,r16*	
0F 02 /r	**larl**	*rm32,r32*	
9A	**lcall**	*imm16x,imm32*	Long call
	lds		load full pointer DS:r16
C5 /r	**ldsw**	*m16,r16*	
C5 /r	**ldsl**	*m32,r32*	
	lea		Load effective address
8D /r	**leaw**	*m16,r16*	
8D /r	**leal**	*m32,r32*	
C9	**leave**		High level procedure exit
	les		Load full pointer ES:r16
C4 /r	**lesw**	*m16,r16*	
C4 /r	**lesl**	*m32,r32*	
	lfs		Load full pointer FS:r16
0F B4 /r	**lfsw**	*m16,r16*	
0F B4 /r	**lfsl**	*m32,r32*	
	lgdt		Load m into DGTR

0F 01 /2	**lgdtw**	*m16*	
0F 01 /2	**lgdtl**	*m32*	
	lgs		Load full pointer GS:r16
0F B5 /r	**lgsw**	*m16,r16*	
0F B5 /r	**lgsl**	*m32,r32*	
	lidt		Load m into IDTR
0F 01 /3	**lidtw**	*m16*	
0F 01 /3	**lidtl**	*m32*	
EA	**ljmp**	*imm16x,imm32*	Long jump
0F 00 /2	**lldt**	*rm16*	Load local descriptor table register
0F 01 /6	**lmsw**	*rm16*	Load machine status word
F0	**lock**		Assert lock signal for next instruction
AC	**lodsb**		Load string operand byte
AD	**lodsl**		Load string operand long
AD	**lodsw**		Load string operand word
E2	**loop**	*rel8*	Dec count jmp if count <> 0
E1	**loope**	*rel8*	Dec count jmp if count <> 0 and ZF = 1
E0	**loopne**	*rel8*	Dec count jmp if count <> 0 and ZF = 0
E0	**loopnz**	*rel8*	Dec count jmp if count <> 0 and ZF = 0
E1	**loopz**	*rel8*	Dec count jmp if count <> 0 and ZF = 1
CB	**lret**		Far return
CA	**lret**	*imm16*	Far return pop imm16 bytes of parms
	lsl		Load segment limit
0F 03 /r	**lslw**	*rm16,r16*	
0F 03 /r	**lsll**	*rm32,r32*	
	lss		Load full pointer SS:r16
0F B2 /r	**lssw**	*m16,r16*	
0F B2 /r	**lssl**	*m32,r32*	
0F 00 /3	**ltr**	*rm16*	Load task register
	mov		Move data
A0	**movb**	*moffs,al*	
A1	**movw**	*moffs,ax*	
A1	**movl**	*moffs,eax*	
A2	**movb**	*al,moffs*	
A3	**movw**	*ax,moffs*	
A3	**movl**	*eax,moffs*	
8A /r	**movb**	*rm8,r8*	
8B /r	**movw**	*rm16,r16*	
8B /r	**movl**	*rm32,r32*	
88 /r	**movb**	*r8,rm8*	
89 /r	**movw**	*r16,rm16*	
89 /r	**movl**	*r32,rm32*	
8C /r	**movw**	*sreg,rm16*	
8E /r	**movw**	*rm16,sreg*	
B0 +r	**movb**	*imm8,r8*	
B8 +r	**movw**	*imm16,r16*	
B8 +r	**movl**	*imm32,r32*	
C6	**movb**	*imm8,rm8*	
C7	**movw**	*imm16,rm16*	
C7	**movl**	*imm32,rm32*	
0F 20 /r	**movl**	*ctlreg,r32*	
0F 22 /r	**movl**	*r32,ctlreg*	
0F 21 /r	**movl**	*dbreg,r32*	
0F 23 /r	**movl**	*r32,dbreg*	
0F 24 /r	**movl**	*treg,r32*	
0F 26 /r	**movl**	*r32,treg*	
A4	**movsb**		Move bytes

A5	**movsl**		Move longs
A5	**movsw**		Move words
	movsx		Move with sign extend
0F BE /r	**movsxb**	*rm8,r16*	
0F BE /r	**movsxb**	*rm8,r32*	
0F BF /r	**movsxw**	*rm16,r32*	
0F BE /r	**movsbw**	*rm8,r16*	
0F BE /r	**movsbl**	*rm8,r32*	
0F BF /r	**movswl**	*rm16,r32*	
	movzx		Move with zero extend
0F B6 /r	**movzxb**	*rm8,r16*	
0F B6 /r	**movzxb**	*rm8,r32*	
0F B7 /r	**movzxw**	*rm16,r32*	
0F B6 /r	**movzbw**	*rm8,r16*	
0F B6 /r	**movzbl**	*rm8,r32*	
0F B7 /r	**movzwl**	*rm16,r32*	
	mul		Unsigned multiply
F6 /4	**mulb**	*rm8,al*	
F6 /4	**mulb**	*rm8*	
F7 /4	**mulw**	*rm16,ax*	
F7 /4	**mulw**	*rm16*	
F7 /4	**mull**	*rm32,eax*	
F7 /4	**mull**	*rm32*	
	neg		Negate
F6 /3	**negb**	*rm8*	
F7 /3	**negw**	*rm16*	
F7 /3	**negl**	*rm32*	
90	**nop**		No operation
	not		Invert bits
F6 /2	**notb**	*rm8*	
F7 /2	**notw**	*rm16*	
F7 /2	**notl**	*rm32*	
	or		Logical inclusive OR
83 /1	**orl**	*imm8s,rm32*	
83 /1	**orw**	*imm8s,rm16*	
0C	**orb**	*imm8,al*	
0D	**orw**	*imm16,ax*	
0D	**orl**	*imm32,eax*	
0D	**orl**	*imm32*	
80 /1	**orb**	*imm8,rm8*	
81 /1	**orw**	*imm16,rm16*	
81 /1	**orl**	*imm32,rm32*	
0A /r	**orb**	*rm8,r8*	
0B /r	**orw**	*rm16,r16*	
0B /r	**orl**	*rm32,r32*	
08 /r	**orb**	*r8,rm8*	
09 /r	**orw**	*r16,rm16*	
09 /r	**orl**	*r32,rm32*	
	out		Output from port
E6	**outb**	*imm8*	
E7	**outw**	*imm8*	
E7	**outl**	*imm8*	
EE	**outb**	*(dx)*	
EF	**outw**	*(dx)*	
EF	**outl**	*(dx)*	
6E	**outsb**		Output byte to port into ES:(E)DI

6F	**outsl**		Output long to port into ES:(E)DI
6F	**outsw**		Output word to port into ES:(E)DI
	pop		Pop a word from the stack
58 +r	**popw**	*r16*	
58 +r	**popl**	*r32*	
1F	**popw**	*ds*	
07	**popw**	*es*	
17	**popw**	*ss*	
0F A1	**popw**	*fs*	
0F A9	**popw**	*gs*	
8F /0	**popw**	*m16*	
8F /0	**popl**	*m32*	
	popa		Pop all
61	**popaw**		
61	**popal**		
	popf		Pop stack into flags
9D	**popfw**		
9D	**popfl**		
	push		Push a word on the stack
50 +r	**pushw**	*r16*	
50 +r	**pushl**	*r32*	
6A	**pushb**	*imm8s*	
68	**pushw**	*imm16*	
68	**pushl**	*imm32*	
0E	**pushw**	*cs*	
1E	**pushw**	*ds*	
06	**pushw**	*es*	
16	**pushw**	*ss*	
0F A0	**pushw**	*fs*	
0F A8	**pushw**	*gs*	
FF /6	**pushw**	*m16*	
FF /6	**pushl**	*m32*	
	pusha		Push all
60	**pushaw**		
60	**pushal**		
	pushf		Push flags
9C	**pushfw**		
9C	**pushfl**		
	rcl		Rotate carry left
D0 /2	**rclb**	*con1,rm8*	
D0 /2	**rclb**	*rm8*	
D2 /2	**rclb**	*cl,rm8*	
C0 /2	**rclb**	*imm8,rm8*	
D1 /2	**rclw**	*con1,rm16*	
D1 /2	**rclw**	*rm16*	
D3 /2	**rclw**	*cl,rm16*	
C1 /2	**rclw**	*imm8,rm16*	
D1 /2	**rcll**	*con1,rm32*	
D1 /2	**rcll**	*rm32*	
D3 /2	**rcll**	*cl,rm32*	
C1 /2	**rcll**	*imm8,rm32*	
	rcr		Rotate carry right
D0 /3	**rcrb**	*con1,rm8*	
D0 /3	**rcrb**	*rm8*	
D2 /3	**rcrb**	*cl,rm8*	
C0 /3	**rcrb**	*imm8,rm8*	
D1 /3	**rcrw**	*con1,rm16*	

D1 /3	**rcrw**	*rm16*	
D3 /3	**rcrw**	*cl,rm16*	
C1 /3	**rcrw**	*imm8,rm16*	
D1 /3	**rcrl**	*con1,rm32*	
D1 /3	**rcrl**	*rm32*	
D3 /3	**rcrl**	*cl,rm32*	
C1 /3	**rcrl**	*imm8,rm32*	
F3	**rep**		rep following instruction CX times
F3	**repe**		repe following instruction CX times or eq
F2	**repne**		repne following instruction CX times or ne
F2	**repnz**		alternate name for repnz
F3	**repz**		alternate name for repe
C3	**ret**		Return
C2	**ret**	*imm16*	Return pop imm16 bytes of parms
	rol		Rotate left
D0 /0	**rolb**	*con1,rm8*	
D0 /0	**rolb**	*rm8*	
D2 /0	**rolb**	*cl,rm8*	
C0 /0	**rolb**	*imm8,rm8*	
D1 /0	**rolw**	*con1,rm16*	
D1 /0	**rolw**	*rm16*	
D3 /0	**rolw**	*cl,rm16*	
C1 /0	**rolw**	*imm8,rm16*	
D1 /0	**roll**	*con1,rm32*	
D1 /0	**roll**	*rm32*	
D3 /0	**roll**	*cl,rm32*	
C1 /0	**roll**	*imm8,rm32*	
	ror		Rotate right
D0 /1	**rorb**	*con1,rm8*	
D0 /1	**rorb**	*rm8*	
D2 /1	**rorb**	*cl,rm8*	
C0 /1	**rorb**	*imm8,rm8*	
D1 /1	**rorw**	*con1,rm16*	
D1 /1	**rorw**	*rm16*	
D3 /1	**rorw**	*cl,rm16*	
C1 /1	**rorw**	*imm8,rm16*	
D1 /1	**rorl**	*con1,rm32*	
D1 /1	**rorl**	*rm32*	
D3 /1	**rorl**	*cl,rm32*	
C1 /1	**rorl**	*imm8,rm32*	
9E	**sahf**		Store AH into flags
	sal		Shift arithmetic left
D0 /4	**salb**	*con1,rm8*	
D0 /4	**salb**	*rm8*	
D2 /4	**salb**	*cl,rm8*	
C0 /4	**salb**	*imm8,rm8*	
D1 /4	**salw**	*con1,rm16*	
D1 /4	**salw**	*rm16*	
D3 /4	**salw**	*cl,rm16*	
C1 /4	**salw**	*imm8,rm16*	
D1 /4	**sall**	*con1,rm32*	
D1 /4	**sall**	*rm32*	
D3 /4	**sall**	*cl,rm32*	
C1 /4	**sall**	*imm8,rm32*	
	sar		Shift arithmetic right
D0 /7	**sarb**	*con1,rm8*	
D0 /7	**sarb**	*rm8*	
D2 /7	**sarb**	*cl,rm8*	

C0 /7	**sarb**	*imm8,rm8*	
D1 /7	**sarw**	*con1,rm16*	
D1 /7	**sarw**	*rm16*	
D3 /7	**sarw**	*cl,rm16*	
C1 /7	**sarw**	*imm8,rm16*	
D1 /7	**sarl**	*con1,rm32*	
D1 /7	**sarl**	*rm32*	
D3 /7	**sarl**	*cl,rm32*	
C1 /7	**sarl**	*imm8,rm32*	
	sbb		Subtract with borrow
83 /3	**sbbl**	*imm8s,rm32*	
83 /3	**sbbw**	*imm8s,rm16*	
1C	**sbbb**	*imm8,al*	
1D	**sbbw**	*imm16,ax*	
1D	**sbbl**	*imm32,eax*	
1D	**sbbl**	*imm32*	
80 /3	**sbbb**	*imm8,rm8*	
81 /3	**sbbw**	*imm16,rm16*	
81 /3	**sbbl**	*imm32,rm32*	
1A /r	**sbbb**	*rm8,r8*	
1B /r	**sbbw**	*rm16,r16*	
1B /r	**sbbl**	*rm32,r32*	
18 /r	**sbbb**	*r8,rm8*	
19 /r	**sbbw**	*r16,rm16*	
19 /r	**sbbl**	*r32,rm32*	
AE	**scasb**		Compare string bytes
AF	**scasl**		Compare string longs
AF	**scasw**		Compare string words
0F 97	**seta**	*rm8*	Set byte if above
0F 93	**setae**	*rm8*	Set byte if above or equal
0F 92	**setb**	*rm8*	Set byte if below
0F 96	**setbe**	*rm8*	Set byte if below or equal
0F 92	**setc**	*rm8*	Set byte if carry
0F 94	**sete**	*rm8*	Set byte if equal
0F 9F	**setg**	*rm8*	Set byte if greater
0F 9D	**setge**	*rm8*	Set byte if greater or equal
0F 9C	**setl**	*rm8*	Set byte if less
0F 9E	**setle**	*rm8*	Set byte if less or equal
0F 96	**setna**	*rm8*	Set byte if not above
0F 92	**setnae**	*rm8*	Set byte if not above or equal
0F 93	**setnb**	*rm8*	Set byte if not below
0F 97	**setnbe**	*rm8*	Set byte if not below or equal
0F 93	**setnc**	*rm8*	Set byte if no carry
0F 95	**setne**	*rm8*	Set byte if not equal
0F 9E	**setng**	*rm8*	Set byte if not greater
0F 9C	**setnge**	*rm8*	Set byte if not greater or equal
0F 9D	**setnl**	*rm8*	Set byte if not less
0F 9F	**setnle**	*rm8*	Set byte if not less or equal
0F 91	**setno**	*rm8*	Set byte if not overflow
0F 9B	**setnp**	*rm8*	Set byte if not parity
0F 99	**setns**	*rm8*	Set byte if not sign
0F 95	**setnz**	*rm8*	Set byte if not zero
0F 90	**seto**	*rm8*	Set byte if overflow
0F 9A	**setp**	*rm8*	Set byte if parity
0F 9A	**setpe**	*rm8*	Set byte if parity even
0F 9B	**setpo**	*rm8*	Set byte if parity odd
0F 98	**sets**	*rm8*	Set byte if sign
0F 94	**setz**	*rm8*	Set byte if zero
0F 94	**setz**	*rm8*	Set byte if zero
0F 01 /0	**sgdt**	*mem32*	Store gdtr

	shl		Shift arithmetic left
D0 /4	**shlb**	*con1,rm8*	
D0 /4	**shlb**	*rm8*	
D2 /4	**shlb**	*cl,rm8*	
C0 /4	**shlb**	*imm8,rm8*	
D1 /4	**shlw**	*con1,rm16*	
D1 /4	**shlw**	*rm16*	
D3 /4	**shlw**	*cl,rm16*	
C1 /4	**shlw**	*imm8,rm16*	
D1 /4	**shll**	*con1,rm32*	
D1 /4	**shll**	*rm32*	
D3 /4	**shll**	*cl,rm32*	
C1 /4	**shll**	*imm8,rm32*	
	shld		Shift double precision left
0F A4	**shldw**	*imm8,r16,rm16*	
0F A4	**shldl**	*imm8,r32,rm32*	
0F A5	**shldw**	*cl,r16,rm16*	
0F A5	**shldl**	*cl,r32,rm32*	
	shr		Shift right
D0 /5	**shrb**	*con1,rm8*	
D0 /5	**shrb**	*rm8*	
D2 /5	**shrb**	*cl,rm8*	
C0 /5	**shrb**	*imm8,rm8*	
D1 /5	**shrw**	*con1,rm16*	
D1 /5	**shrw**	*rm16*	
D3 /5	**shrw**	*cl,rm16*	
C1 /5	**shrw**	*imm8,rm16*	
D1 /5	**shrl**	*con1,rm32*	
D1 /5	**shrl**	*rm32*	
D3 /5	**shrl**	*cl,rm32*	
C1 /5	**shrl**	*imm8,rm32*	
	shrd		Shift double precision right
0F AC	**shrdw**	*imm8,r16,rm16*	
0F AC	**shrdl**	*imm8,r32,rm32*	
0F AD	**shrdw**	*cl,r16,rm16*	
0F AD	**shrdl**	*cl,r32,rm32*	
0F AD	**shrdw**	*r16,rm16*	
0F AD	**shrdl**	*r32,rm32*	
0F 01 /1	**sidt**	*mem32*	Store idtr
0F 00 /0	**sldt**	*rm16*	Store ldtr to EA word
0F 01 /4	**smsw**	*rm16*	Store machine status to EA word
F9	**stc**		Set carry flag
FD	**std**		Clear direction flag
FB	**sti**		Set interrupt flag
AA	**stosb**		Store string byte
AB	**stosl**		Store string long
AB	**stosw**		Store string word
0F 00 /1	**str**		Store task register
	sub		Subtract
83 /5	**subl**	*imm8s,rm32*	
83 /5	**subw**	*imm8s,rm16*	
2C	**subb**	*imm8,al*	
2D	**subw**	*imm16,ax*	
2D	**subl**	*imm32,eax*	
2D	**subl**	*imm32*	
80 /5	**subb**	*imm8,rm8*	
81 /5	**subw**	*imm16,rm16*	
81 /5	**subl**	*imm32,rm32*	

2A /r	**subb**	*rm8,r8*	
2B /r	**subw**	*rm16,r16*	
2B /r	**subl**	*rm32,r32*	
28 /r	**subb**	*r8,rm8*	
29 /r	**subw**	*r16,rm16*	
29 /r	**subl**	*r32,rm32*	
	test		Logical compare
A8	**testb**	*imm8,al*	
A9	**testw**	*imm16,ax*	
A9	**testl**	*imm32,eax*	
A9	**testl**	*imm32*	
F6 /0	**testb**	*imm8,rm8*	
F7 /0	**testw**	*imm16,rm16*	
F7 /0	**testl**	*imm32,rm32*	
84 /r	**testb**	*r8,rm8*	
85 /r	**testw**	*r16,rm16*	
85 /r	**testl**	*r32,rm32*	
0F 00 /4	**verr**	*rm16*	Verify segment for read
0F 00 /5	**verw**	*rm16*	Verify segment for write
9B	**wait**		Wait for coprocessor
	xchg		Exchange register
90 +r	**xchgw**	*r16,ax*	
90 +r	**xchgw**	*ax,r16*	
90 +r	**xchgl**	*r32,eax*	
90 +r	**xchgl**	*r32*	
90 +r	**xchgl**	*eax,r32*	
86 /r	**xchgb**	*rm8,r8*	
87 /r	**xchgw**	*rm16,r16*	
87 /r	**xchgl**	*rm32,r32*	
86 /r	**xchgb**	*r8,rm8*	
87 /r	**xchgw**	*r16,rm16*	
87 /r	**xchgl**	*r32,rm32*	
D7	**xlat**		Table lookup translation
D7	**xlatb**		Table lookup translation
	xor		Logical exclusive OR
83 /6	**xorl**	*imm8s,rm32*	
83 /6	**xorw**	*imm8s,rm16*	
34	**xorb**	*imm8,al*	
35	**xorw**	*imm16,ax*	
35	**xorl**	*imm32,eax*	
35	**xorl**	*imm32*	
80 /6	**xorb**	*imm8,rm8*	
81 /6	**xorw**	*imm16,rm16*	
81 /6	**xorl**	*imm32,rm32*	
32 /r	**xorb**	*rm8,r8*	
33 /r	**xorw**	*rm16,r16*	
33 /r	**xorl**	*rm32,r32*	
30 /r	**xorb**	*r8,rm8*	
31 /r	**xorw**	*r16,rm16*	
31 /r	**xorl**	*r32,rm32*	

Using C to Prototype Assembly Language

The COHERENT C compiler includes a switch, **-S**, that translates C code into COHERENT assembly language. The assembly language it produces cannot be directly assembled, but you can examine it to see just what the compiler does under given circumstances; and you can use it to prototype a routine in assembly language.

Suppose, for example, that you wish to write a function that takes two parameters: an integer, which gives a port number to read from; and an address where the data should go. Start by writing a C function with the correct calling sequence. For example, the following function is in a file called **proto.c**:

```
readstuff(addr, port)
register char *addr;
int port;
{
      register int dx = port;
      char *foo = addr;
}
```

Compile it with the following command line:

```
      cc -S proto.c
```

This produces file **proto.s**, which contains the following:

```
/     module name foo
      .alignoff

      .text
      .globl readstuff
readstuff:
      push  %ebp
      movl  %ebp, %esp
      push  %esi
      push  %edi
      push  %ebx
      movl  %ebx, 8(%ebp)
      movl  %esi, 12(%ebp)
      movl  %edi, %ebx
      pop   %ebx
      pop   %edi
      pop   %esi
      leave
      ret
      .align      4
```

This is your prototype. You can easily modify it into what you want; for example:

```
/     This will only work if you install it as a driver.
/     As the operating system will protect itself if
/     Ordinary users try to access ports. Ask about our
/     Device driver kits.

       .text
       .globl readstuff
readstuff:
       push       %ebp
       movl       %ebp, %esp
       push       %edi                       / Save the edi for the caller
       movl       %edx, 8(%ebp)              / Get the port number
       movl       %edi, 12(%ebp)             / Get the user address

      insb        / Read port (%dx) to %es:%edi

      pop   %edi  / See 386 calling conventions
      leave
      ret
```

Example

The following example echoes strings onto your screen.

```
/ sstatic void foo(i) { printf("Parm is %d\n", i); }

      .text
.L2:  .byte "Parm is %d0, 0
```

```
foo:
      push  %ebp        / set up stack frame
      movl  %ebp, %esp
      push  8(%ebp)           / push parms from right to left
      push  $.L2
      call  printf
      leave             / %esp <- %ebp; pop %ebp
      ret

/ main() { foo(5); }

      .globl main
main:
      push  %ebp
      movl  %ebp, %esp
      push  $5
      call  foo
      leave
      ret
```

See Also

asfix, calling conventions, cc, cdmp, commands
Intel Corporation: *386 DX Programmer's Reference Manual.* Santa Clara, CA: Intel Corporation, 1990. *Highly recommended.*

Diagnostics

The following gives the error messages returned by **as**. The messages are in alphabetical order. Each message is marked as to its degree of severity: A *fatal* message usually indicates a condition that caused the assembler to terminate execution. Often, they indicate internal problems in the assembler. An *error* message points to a condition in the source code that the assembler cannot resolve. This almost always occurs when the program does something illegal. A *warning* message points out code that is compilable, but may produce trouble when the program is executed.

.align must be 1, 2 or 4 *(error)*
  .align must work after the link. These are the only values for which this can be true.

Ambiguous operand length, *n* bytes selected *(warning)*
  The assembler cannot tell the operand length by looking at the opcode and the operands. You may want to do something like change **mov** to **movl**.

Arithmetic between addresses on different segments *(error)*
  You may only add or subtract addresses if they are in the same segment.

Bad scale *(error)*
  Address scale must be 0, 1, 2, 4, or 8.

16 bit addressing mode used in 32 bit code *(warning)*
  You probably don't want to do this. For example, you may want to say **(%esi)**, not **(%si)**.

32 bit addressing mode used in 16 bit code *(warning)*
  You probably don't want to do this. For example, you may want to say **(%si)**, not **(%esi)**.

Cannot fopen(*string*, *string*) *(fatal)*
Cannot insert \0 in string *(error)*
  NUL (\0) terminates strings. Instead of

```
      .byte "hello\n\0"
```

  use:

```
      .byte "hello\n", 0
```

Character constant *n* long *(error)*
  Character constants must be one byte long.

.comm must have tag *(error)*
  The format of **.comm** is **.comm name, size**.

Command option '*c*' missing its argument *(fatal)*
Data defined in .bss *(error)*
The **.bss** segment is uninitialized data. You cannot place actual values there.

.define must have a label *(error)*
Duplicate symbol '*string*' *(error)*
symbol is defined on two different lines.

.else detected logic type *n* *(fatal)*
Logic error in assembler. Please report this problem to Mark Williams technical support.

End of line after backslash reading parm *(error)*
Macro parmeters may not be broken up with backslash.

End of line after backslash *(error)*
End of line detected in character constant *(error)*
End of line detected in string *(error)*
End of macro building .while *(error)*
A **.macro** ended while reading in a **.while** loop.

.endi detected logic type *n* *(fatal)*
Logic error in assembler. Please report this problem to Mark Williams technical support.

Error in binary number *(error)*
Error in octal number *(error)*
Found *n* parms expected *n* *(error)*
Illegal combination of opcode and operands *(error)*
Although the opcode is valid and the operands are valid, there is no form of this opcode which takes this combination of operands in this order.

Illegal use of local symbol *(error)*
Illegal use of of predefined symbol *string*. *(error)*
Improper instruction following lock *(warning)*
Only a few instructions are valid after a lock instruction. See your machine documentation for details.

Improper instruction following rep *(warning)*
Only a few instructions are valid after a rep instruction. See your machine documentation for details.

Indirect mode on invalid instruction *(error)*
Indirection is only allowed on call and jump near instructions.

Internal error relative branch logic *(fatal)*
Logic error in assembler. Please report this problem to Mark Williams technical support.

Invalid .mlist option must be on or off *(error)*
Invalid character '*c*' *string* at position *n* *(error)*
Invalid character 0x*0xn* *string* at position *n* *(error)*
Invalid data type, must be symbol *(error)*
Invalid floating point register number *(error)*
Invalid opcode: '*string*' *(error)*
The string in the opcode position is not one of our opcodes or one of your macros.

Invalid operand type *(error)*
string is an improper register in this context *(error)*
Label ignored *(error)*
This statement cannot take a label.

Label on invalid operator *(error)*
Label required *(error)*
Length *n* string range exceeded *(error)*
Strings may not exceed 32 kilobytes.

Logic error in macro def '*string*' *n* *(fatal)*
Logic error in assembler. Please report this problem to Mark Williams technical support.

Logic error in umark *(fatal)*
Logic error in assembler. Please report this problem to Mark Williams technical support.

Macro definition must have a label *(error)*
.mexit not in macro *(error)*
Missing .endi *(error)*
 Input ended leaving **.if** open.

Missing .endm *(error)*
 Input ended leaving **.macro** open.

Missing .endw *(error)*
 Input ended leaving **.while** open.

Mixed 386/286 addressing modes *(error)*
 No opcode allows mixed 286 and 386 addressing modes.

Mixed 386/286 data modes *(error)*
 No 386 opcode allows mixed 286 and 386 data modes.

Mixed length addressing registers *(error)*
 Addressing registers must both be the same length.

more than one file to process *(fatal)*
 The assembler will only process one file at a time.

Name required *(error)*
 The format of **set** is **.set name, value**

no work *(fatal)*
 There were no files listed on the command line.

NULL address in relative branch *(fatal)*
 Logic error in assembler. Please report this problem to Mark Williams technical support.

Octal number *n* truncated to char *(error)*
 An octal number in a string was too big.

Optype *n* in lex *(fatal)*
 Logic error in assembler. Please report this problem to Mark Williams technical support.

Org to invalid value *(error)*
 You may not **.org** to doubles or strings.

Org to wrong segment *(error)*
 You must **.org** to the current segment.

Out of space *(fatal)*
 A call to **malloc()** failed. The typical large consumers of RAM are macros and **.defines**; symbols consume less. Can you break your assembly into smaller pieces? Could you be in some sort of endless recursion or loop?

Parm *n* not found *(error)*
 An attempt to **.shift** too far has been made.

.parmct not in macro *(error)*
 .parmct returns the number of parameters in the current macro.

Phase error '*string*' *(error)*
 A symbol is defined one way in one phase of the assembly and another way in the next phase.

Redefinition of '*string*' *(error)*
 An assembler internal symbol is being redefined.

Seek error on object file *(fatal)*
Seek error on object file *(fatal)*
.shift not in macro *(error)*
 .shift shifts macro parameters. It has no meaning outside a macro.

String must be on .byte *(error)*
 For example:

```
            .byte "This is how we place a string", 0
```

Symbol may not be double *(error)*
    You may not convert a symbol to a floating-point value.

Symbol may not be float *(error)*
    You may not convert a symbol to a floating-point value.

Syntax error *(error)*
    The syntax of this statement makes no sense to the parser. This can be a variety of problems.

Table error kind *0xn* detected *(fatal)*
    Logic error in assembler. Please report this problem to Mark Williams technical support.

This code may not work the same way on all chips *(warning)*
    Some chips may not execute this code as expected.

Too many operands *(error)*
    No 386 opcode has more than three operands.

Undefined symbol '*string*' *(error)*
    A symbol was used without defining it or using a **-g** option. You must define local symbols.

Unexpected .else statement *(error)*
Unexpected .endi statement *(error)*
Unexpected .endm ignored *(error)*
Unexpected .endw *(error)*
Unexpected return from parser *(fatal)*
    Logic error in assembler. Please report this problem to Mark Williams technical support.

Unknown command option *c* *(fatal)*
Unlikely output file '*string*' *(fatal)*
    Output file-names should have **.o** suffixes. Because this is generally a typographical error, **as** aborts to avoid overwriting an important file.

Unmatched '*c*' *(error)*
    A delimeter, [, (,), or] is unmatched in this command.

Unmatched bracket in parmeter *(error)*
    Line ended leaving an open bracket or parenthesis.

Write error on object file *(fatal)*
    as could not write the object module. This error can have any of several causes; the most common is that you lack permission to write into the current directory, or you lack permission to overwrite an existing file of the same name.

Notes

We have designed **as** to ease porting of programs written in other dialects of UNIX 386 assembly language, as well as to be a powerful tool for development of new programs. We think you will find the features and documentation of our assembler considerably more complete than are available anywhere else. However, we have chosen *not* to duplicate behavior of other assemblers that leads to inefficient or incorrect output, or that generates code without warning when given questionable input. We have also chosen to support operator precedence rather than perpetuating antiquated left-to-right evaluation schemes seen elsewhere. *Caveat utilitor.*

In the course of writing this assembler, we have discovered that the UNIX implementation of **fdiv**, **fdivr**, **fsub**, and **fsubr** differs from that described in the Intel documents. The COHERENT assembler conforms to the UNIX standard by default. You should be very careful with the order of operands to these instructions. Once again, *caveat utilitor.*

ASCII — Definition

ASCII is an acronym for the American Standard Code for Information Interchange, as defined by the American National Standards Institute standard X3.4-1977. It is a table of seven-bit binary numbers that encode the letters of the alphabet, numerals, punctuation, and the most commonly used control sequences for printers and terminals. ASCII codes are used on all microcomputers sold in the United States.

The following table gives the ASCII characters in octal, decimal, and hexadecimal numbers, their definitions, and expands abbreviations where necessary.

000	0	0x00	NUL	**<ctrl-@>**	Null character
001	1	0x01	SOH	**<ctrl-A>**	Start of header
002	2	0x02	STX	**<ctrl-B>**	Start of text
003	3	0x03	ETX	**<ctrl-C>**	End of text
004	4	0x04	EOT	**<ctrl-D>**	End of transmission
005	5	0x05	ENQ	**<ctrl-E>**	Enquiry
006	6	0x06	ACK	**<ctrl-F>**	Positive acknowledgement
007	7	0x07	BEL	**<ctrl-G>**	Bell
010	8	0x08	BS	**<ctrl-H>**	Backspace
011	9	0x09	HT	**<ctrl-I>**	Horizontal tab
012	10	0x0A	LF	**<ctrl-J>**	Line feed
013	11	0x0B	VT	**<ctrl-K>**	Vertical tab
014	12	0x0C	FF	**<ctrl-L>**	Form feed
015	13	0x0D	CR	**<ctrl-M>**	Carriage return
016	14	0x0E	SO	**<ctrl-N>**	Shift out
017	15	0x0F	SI	**<ctrl-O>**	Shift in
020	16	0x10	DLE	**<ctrl-P>**	Data link escape
021	17	0x11	DC1	**<ctrl-Q>**	Device control 1 (XON)
022	18	0x12	DC2	**<ctrl-R>**	Device control 2 (tape on)
023	19	0x13	DC3	**<ctrl-S>**	Device control 3 (XOFF)
024	20	0x14	DC4	**<ctrl-T>**	Device control 4 (tape off)
025	21	0x15	NAK	**<ctrl-U>**	Negative acknowledgement
026	22	0x16	SYN	**<ctrl-V>**	Synchronize
027	23	0x17	ETB	**<ctrl-W>**	End of transmission block
030	24	0x18	CAN	**<ctrl-X>**	Cancel
031	25	0x19	EM	**<ctrl-Y>**	End of medium
032	26	0x1A	SUB	**<ctrl-Z>**	Substitute
033	27	0x1B	ESC	**<ctrl-[>**	Escape
034	28	0x1C	FS	**<ctrl-\>**	Form separator
035	29	0x1D	GS	**<ctrl-]>**	Group separator
036	30	0x1E	RS	**<ctrl-^>**	Record separator
037	31	0x1F	US	**<ctrl-_>**	Unit separator
040	32	0x20	SP		Space
041	33	0x21	!		Exclamation point
042	34	0x22	"		Quotation mark
043	35	0x23	#		Pound sign (sharp)
044	36	0x24	$		Dollar sign
045	37	0x25	%		Percent sign
046	38	0x26	&		Ampersand
047	39	0x27	’		Apostrophe
050	40	0x28	(		Left parenthesis
051	41	0x29	)		Right parenthesis
052	42	0x2A	*		Asterisk
053	43	0x2B	+		Plus sign
054	44	0x2C	,		Comma
055	45	0x2D	-		Hyphen (minus sign)
056	46	0x2E	.		Period
057	47	0x2F	/		Virgule (slash)
060	48	0x30	0		
061	49	0x31	1		
062	50	0x32	2		
063	51	0x33	3		
064	52	0x34	4		
065	53	0x35	5		
066	54	0x36	6		
067	55	0x37	7		
070	56	0x38	8		
071	57	0x39	9		
072	58	0x3A	:		Colon
073	59	0x3B	;		Semicolon
074	60	0x3C	<		Less-than symbol (left angle bracket)

075	61	0x3D	=	Equal sign
076	62	0x3E	>	Greater-than symbol (right angle bracket)
077	63	0x3F	?	Question mark
0100	64	0x40	@	At sign
0101	65	0x41	A	
0102	66	0x42	B	
0103	67	0x43	C	
0104	68	0x44	D	
0105	69	0x45	E	
0106	70	0x46	F	
0107	71	0x47	G	
0110	72	0x48	H	
0111	73	0x49	I	
0112	74	0x4A	J	
0113	75	0x4B	K	
0114	76	0x4C	L	
0115	77	0x4D	M	
0116	78	0x4E	N	
0117	79	0x4F	O	
0120	80	0x50	P	
0121	81	0x51	Q	
0122	82	0x52	R	
0123	83	0x53	S	
0124	84	0x54	T	
0125	85	0x55	U	
0126	86	0x56	V	
0127	87	0x57	W	
0130	88	0x58	X	
0131	89	0x59	Y	
0132	90	0x5A	Z	
0133	91	0x5B	[	Left bracket (left square bracket)
0134	92	0x5C	\	Backslash
0135	93	0x5D	]	Right bracket (right square bracket)
0136	94	0x5E	^	Circumflex
0137	95	0x5F	_	Underscore
0140	96	0x60	‘	Grave
0141	97	0x61	a	
0142	98	0x62	b	
0143	99	0x63	c	
0144	100	0x64	d	
0145	101	0x65	e	
0146	102	0x66	f	
0147	103	0x67	g	
0150	104	0x68	h	
0151	105	0x69	i	
0152	106	0x6A	j	
0153	107	0x6B	k	
0154	108	0x6C	l	
0155	109	0x6D	m	
0156	110	0x6E	n	
0157	111	0x6F	o	
0160	112	0x70	p	
0161	113	0x71	q	
0162	114	0x72	r	
0163	115	0x73	s	
0164	116	0x74	t	
0165	117	0x75	u	
0166	118	0x76	v	
0167	119	0x77	w	
0170	120	0x78	x	
0171	121	0x79	y	
0172	122	0x7A	z	

0173	123	0x7B	{	Left brace (left curly bracket)
0174	124	0x7C	|	Vertical bar
0175	125	0x7D	}	Right brace (right curly bracket)
0176	126	0x7E	~	Tilde
0177	127	0x7F	DEL	**<ctrl->** Delete

Files

/usr/pub/ascii

See Also

Latin 1, Programming COHERENT

asctime() — Time Function (libc)

Convert time structure to ASCII string
#include <time.h>
#include <sys/types.h>
char *asctime(*tmp***)**
struct tm ****tmp*;**

asctime() takes the data found in *tmp*, and turns it into an ASCII string. *tmp* is of the type **tm**, which is a structure defined in the header file **time.h**. This structure must first be initialized by either **gmtime()** or **localtime()** before it can be used by **asctime()**. For a further discussion of **tm**, see the entry for **time**.

asctime() returns a pointer to where it writes the text string it creates.

Example

The following example demonstrates the functions **asctime()**, **ctime()**, **gmtime()**, **localtime()**, and **time()**, and shows the effect of the environmental variable **TIMEZONE**. For a discussion of the variable **time_t**, see the entry for **time()**.

```
#include <time.h>
#include <sys/types.h>
main()
{
      time_t timenumber;
      struct tm *timestruct;

      /* read system time, print using ctime */
      time(&timenumber);
      printf("%s", ctime(&timenumber));

      /* use gmtime to fill tm, print with asctime */
      timestruct = gmtime(&timenumber);
      printf("%s", asctime(timestruct));

      /* use localtime to fill tm, print with asctime */
      timestruct = localtime(&timenumber);
      printf("%s", asctime(timestruct));
}
```

See Also

libc, time(), time [overview]
ANSI Standard, §7.12.3.1
POSIX Standard, §8.1.1

Notes

asctime() returns a pointer to a statically allocated data area that is overwritten by successive calls.

asfix — Command

Convert assembly-language programs into 80386 format
asfix < *oldfile* **>** *newfile*

The command **asfix** converts programs written in the COHERENT-286 assembly language into a form that can be assembled by the COHERENT 386 edition of **as**, the COHERENT assembler.

asfix reads the standard input and writes to the standard output. It changes DEC-form local symbols to the form recognized by COHERENT-386 **as**, changes character constants from the form **'x** to the form **'x'**, and changes local symbols from the COHERENT-286 form to the COHERENT-386 form.

See Also

as, commands

ASHEAD — Environmental Variable

Append options to beginning of as command line
export ASHEAD=*options*

The COHERENT assembler **as** reads the environmental variables **ASHEAD** and **ASTAIL** before it begins its work. You can set these variables to hold the default options that you want the assembler always to use.

as appends the options in **ASHEAD** to the beginning of its command line.

See Also

as, ASTAIL, environmental variables

asin() — Mathematics Function (libm)

Calculate inverse sine
#include <math.h>
double asin(*arg***) double** *arg***;**

asin() calculates the inverse sine of *arg*, which must be in the range [-1., 1.]. The result will be in the range [-π/2, π/2].

If all goes well, **asin()** returns the inverse sine. However, if *arg* is out of range, **asin()** sets **errno** to **EDOM** and returns zero.

Example

For an example of this function, see the entry for **tan()**.

See Also

libm, sin()
ANSI Standard, §7.5.2.2
POSIX Standard, §8.1

ASKCC — Environmental Variable

Force prompting for CC names
ASKCC=*YES/NO*

The environmental variable **ASKCC** directs the mailer program **mail** to prompt for carbon-copy names. A carbon-copy (or CC) name gives another person to whom a mail message should be sent. To turn on prompting, use the command:

```
	export ASKCC=YES
```

See Also

environmental variables, mail

assert() — Macro Diagnostics (assert.h)

Check assertion at run time
#include <assert.h>
void assert(*outcome***)**
int *outcome***;**

assert() checks the value of *outcome*, which usually is the product of an expression. If *outcome* is false (zero), **assert()** sends a message into the standard-error stream and calls **exit()**. It is useful for verifying that a necessary condition is true.

The error message includes the text of the assertion that failed, the name of the source file, and the line within the source file that holds the expression in question. These last two elements consist, respectively, of the values of the preprocessor macros _ _**FILE**_ _ and _ _**LINE**_ _.

assert() calls **exit()**, which never returns.

To turn off **assert()**, define the macro **NDEBUG** prior to including the header **assert.h**. This forces **assert()** to be redefined as

```
	#define assert(ignore)
```

See Also

exit(), assert.h, C preprocessor,
ANSI Standard, §7.2.1.1
POSIX Standard, §8.1

Notes

The ANSI Standard requires that **assert()** be implemented as a macro, not a library function. If a program suppresses the macro definition in favor of a function call, its behavior is undefined.

Turning off **assert()** with the macro **NDEBUG** will affect the behavior of a program if the expression being evaluated normally generates side effects.

assert() is useful for debugging, and for testing boundary conditions for which more graceful error recovery has not yet been implemented.

assert.h — Header File

Define assert()
#include <assert.h>

assert.h is the header file that defines the macro **assert()**.

See Also

assert(), header files,
ANSI Standard, §7.2

ASTAIL — Environmental Variable

Append options to end of as command line
export ASTAIL=*options*

The COHERENT assembler **as** reads the environmental variables **ASHEAD** and **ASTAIL** before it begins its work. You can set these variables to hold the default options that you want the assembler always to use.

as appends the options in **ASTAIL** to the end of its command line.

See Also

as, ASHEAD, environmental variables,

asy — Device Driver

Device driver for asynchronous serial lines

The device driver **asy** supports serial ports. It uses major number 5.

asy can handle from one to 32 serial ports. The ports can be any mixture of 8250, 8250B, 16550, 16550A, and equivalent devices, including nearly all conventional COM1 through COM4 serial cards, and most non-intelligent multiport add-in cards. It automatically recognizes, and uses, on-chip FIFO, and it can specify groups of ports that share a single interrupt status.

Types of Port Configuration

Each port that **asy** serves has a base name, e.g., **/dev/com1r**. Each has its own minor device number. Different configurations of the port are selected by using different suffixes, as follows:

l (Local) "Local mode" means that the line will have a terminal plugged into it, or is connected to a modem running in command mode. Local mode uses the minor device with the modem-control bit (bit 7) set.

r (Remote) "Modem control" means that the line will have a modem plugged into it. Modem control is enabled on a serial line by resetting the modem control bit (bit 7) in the minor number for the device. This allows the system to generate a hangup signal when the modem indicates loss of carrier by dropping DCD (Data Carrier Detect). A modem line should always have its DSR, DCD and CTS pins connected. If left hanging, spurious transitions can cause severe system thrashing. An **open()** to a modem-control line will block until a carrier is detected (i.e., until DCD goes true).

p (Polled mode) "Polled mode" means that the port cannot generate an interrupt, but must be checked (or polled) constantly by the COHERENT system to see if activity has occurred on it. Such polling takes a significant toll on system performance. The main reason for supporting polled devices is that older style COM equipment will not allow both **com1** and **com3** to use interrupts at the same time, nor will it allow both **com2** and **com4** to use interrupts at the same time. If you use a port in polled mode, you will get better performance using one of the newer FIFO parts, such as the 16550A.

To convert from using a polled to an interrupt driven device, edit file **/etc/ttys** and then type the command:

```
kill quit 1
```

For details, see the Lexicon entry for **ttys**.

f (Flow control) A device with hardware flow control. Here, signal CTS must be active for the driver to send data out the port, and signal RTS will be set active by the driver whenever it is ready for input. Some high-speed modems, and some serial printers, are capable of using these conventions. If your equipment does not support RTS/CTS handshaking, there is no benefit to using this option.

Due to limitations in the design of the ports, you can enable interrupts on either COM1 or COM3 (or on COM2 or COM4), but not both. If you wish to use both ports simultaneously, one must be run in polled mode. For example, if you wish to open all four serial lines, you can open two of the lines in interrupt mode: you can open either COM1 or COM3 in interrupt mode, and you can open either COM2 or COM4 in interrupt mode. The other two lines must be opened in polled mode.

Opening a device in polled mode consumes many CPU cycles, based upon the speed of the highest baud rate requested. For example, on a 20 MHz 80386-based machine, polling at 9600-baud was found to consume about 15% of the CPU time. As only one device can use the interrupt line at any given time, the best approach is to make the high-speed line of the pair interrupt driven and open the low-speed or less-frequently used line in polled mode. However, if you enable a polled line for logins, the port is open and will be polled as long as the port remains open (enabled). Thus, even if a port is not in use, the fact that it has a **getty** on it consumes CPU cycles. As a rule of thumb, try to open a port in interrupt mode. If you cannot, use the polled version.

If you intend to use a modem on your serial port, you must insure that the DCD signal from the modem actually *follows* the state of carrier detect. Some modems allow the user to "strap" or set the DCD signal so that it is always asserted (true). This incorrect setup will cause COHERENT to think that the modem is "connected" to a remote modem, even when there is no such connection.

There are eight possible configurations, and eight valid suffixes. In the example of the port whose base name is **com1**, the configurations would be found in the directory **/dev** as **/dev/com1l**, **/dev/com1r**, **/dev/com1pl**, **/dev/com1pr**, **/dev/com1fl**, **/dev/com1fr**, **/dev/com1fpl**, and **/dev/com1fpr**.

Driver Configuration

asy is usually configured — and proper names are created in directory **/dev** — when you install COHERENT. The following explains how to configure **asy**, in case you must modify the original installation.

To configure **asy**, do the following:

1. Type the following command to become the superuser **root**:

```
su root
```

2. Change to directory **/etc/conf**.

3. Execute script **asy/mkdev**. This script walks you through the process of describing your serial ports to COHERENT.

4. When you have successfully completed **asy/mkdev**, type the command:

```
bin/idmkcoh -o cohtest
```

This generates a new kernel, called **cohtest**, which incorporates the changes you described when you ran **asy/mkdev**.

5. Boot your new kernel. If you do not know how to do this, read the Lexicon entry **booting**.

Editing /etc/default/async

The first step in reconfiguring **asy** is to edit **/etc/default/async**. This file holds the description of how the **asy** driver is to be configured.

asy ignores blank lines and lines that begin with a pound sign '#'; you can use them as comments if you wish. Each port that is not in a group must have a line beginning with the letter 'P', followed by seven numbers:

- The hexadecimal base address for the port.
- The IRQ number, in decimal, used by the port (use zero if no interrupt line is needed).
- The hexadecimal value used for control lines OUT1 and OUT2 when the port is open. Permissible values are 0, 4, 8, and C. Use 4 if OUT1 must be asserted, 8 if OUT2 must be asserted, and C if both signals are needed. The most common value needed in this field is 8.
- One if the port needs exclusive use of its interrupt line (true for conventional COM1/COM4 equipment), zero otherwise.
- Default baud rate for the port.
- Channel number for the port (0-31).
- A flag to indicate if modem-status interrupts are to be disabled for this board: one if they are to be disabled, zero if they are not.

The last field is required because some chips are defective and lock up the system if modem status interrupts are enabled. This flag protects you against such problems, but at the price of disabling hardware flow control.

Many multiport boards support a separate I/O address that can be read to determine which port requires service. Each group of up to 16 ports must have a line beginning with the letter 'G', followed by a separate line describing each port in the group. There are four different group types:

1. Bits in the status port are one when the corresponding port needs service, zero otherwise. (Sealevel, Comtrol, Star Gate, Connect Tech, Boca Research.)

2. Bits in the status port are zero when the corresponding port needs service, one otherwise. (Arnet.)

3. The low three bits in the status port give the slot number on the card for the port needing service. (GTEK.)

4. The low four bits in the status port give the slot number on the card for the port reading service. If no port needs service, the status port contains hexadecimal value FF. (Digiboard.)

The 'G' line requires the following fields. All are in decimal, except as noted:

- The hexadecimal address for the group-status port.
- The IRQ number used by the group. Use zero if no interrupt line is needed.
- The hexadecimal value used for control lines OUT1 and OUT2 when the port is open (usually eight).
- The type number of the group — one, two, or three, as described above.
- The number of ports in the group, 1 through 16.
- A flag to indicate if modem-status interrupts are to be disabled for this board: one if they are to be disabled, zero if they are not.

Each group line is followed by a separate 'M' line for each member of the group. Fields required on the 'M' line (in decimal, except as noted) are:

- The hexadecimal base address for the port.
- Default baud rate for the port.
- The slot number of the port within the group 0 through 7. For group types 1 and 2, slot 0 corresponds to the least-order bit in the status port, slot 7 to the highest order bit.
- Channel number for the port (0-31).

The following gives the **async** file for a system with standard **COM1** through **COM4** ports as channels 0 through 3, a Comtrol Hostess 550/16 as channels 4 through 19, and finally an Arnet Multiport as channels 20 through 27.

```
# /etc/default/async spec for standard com1-com4
#Record formats:
#P        Port      Irq       OUT[12]   Excl      Speed     Channel     No MS int
#G        Port      Irq       OUT[12]   Type      Number-of-Slots       No MS int
#M        Port      Speed     Slot      Channel

# com1/2/3/4
P         3f8       4         8         1         9600      0    0
P         2f8       3         8         1         9600      1    0
P         3e8       4         8         1         9600      2    0
P         2e8       3         8         1         9600      3    0

# Hostess 550 16 - two groups of 8 ports, using irq 12
G         507       12        8         1         8         0
M         500       9600      0         4
M         508       9600      1         5
M         510       9600      2         6
M         518       9600      3         7
M         520       9600      4         8
M         528       9600      5         9
M         530       9600      6         10
M         538       9600      7         11

G         547       12        8         1         8         0
M         540       9600      0         12
M         548       9600      1         13
M         550       9600      2         14
M         558       9600      3         15
M         560       9600      4         16
M         568       9600      5         17
M         570       9600      6         18
M         578       9600      7         19

# Arnet Multiport - one group of 8 ports, using irq 7
G         272       7         0         2         8         0
M         280       9600      0         20
M         288       9600      1         21
M         290       9600      2         22
M         298       9600      3         23
M         2A0       9600      4         24
M         2A8       9600      5         25
M         2B0       9600      6         26
M         2B8       9600      7         27
```

You should look at the version of **/etc/default/async** that is shipped with COHERENT for examples of all **async** features, including those described above. This file includes sample configurations for every board that Mark Williams Company had available for testing.

Building a New Kernel

Now that you have described how you want **asy** to be configured, the next step is to build a new kernel. Log in as the superuser **root** and execute the following commands:

```
cd /etc/conf
asy/mkdev
bin/idmkcoh -o /kernel_name
```

where *kernel_name* is the new kernel that includes the **asy** driver. To run this new kernel, simply reboot your machine.

See Also

asymkdev, device drivers, RS-232

Notes

If your system loses characters while transferring files on 4800-bps or higher-speed lines, *we strongly urge you to replace your existing 8250- or 16450-based UARTs with those based upon the 16550A design, such as the National Semiconductor NS16550AFN.* These newer UARTs are pin-compatible with the older UARTs. COHERENT automatically senses and enables them when it boots.

asymkdev — Command

Create nodes for asynchronous devices
/conf/asymkdev [-u] [*async_file* **[***outfile***]]**

The command **asymkdev** reads *async_file*, questions the user about her system, and writes a shell script into *outfile*. When run, the script creates the proper nodes (up to 256 of them) for the asynchronous devices in **/dev**.

If you name no *async_file*, **asymkdev /dev/default/async**. If you name no *outfile*, it writes its script into **asy_mknod**.

asymkdev asks about each asynchronous channel for which a port is configured. It asks for the basic device name (e.g., **asy00** or **com0**), and then asks which of the eight possible port configurations will be used. The options are:

l or **r**	Local or remote.
i or **p**	Interrupt-driven or polled.
f or **n**	RTS-CTS flow control or no hardware flow control.

For details of what the options mean, see the Lexicon article for the device driver **asy**.

Suffix letters “rlipnf” respectively indicate remote, local, interrupt, polled, no-flow, and flow-control configurations, as explained in the Lexicon article for **asy**.

For each question, type the value that applies or press **<Enter>**, to select the default displayed in brackets. The option **-u** suppresses prompts.

See Also

asy, asypatch, commands

Notes

Only the superuser **root** can run this command.

asypatch — Command

Patch a kernel file for an asynchronous configuration
/conf/asypatch [-v] <*kernel_name***> <***async_file*

The command **asypatch** patches a kernel file for the asynchronous configuration specified in *async_file*. The format of *async_file* is described in the Lexicon article for the device driver **asy**.

See Also

asy, asymkdev, commands

at — Device Driver

Drivers for hard-disk partitions

/dev/at* are the COHERENT system’s AT devices for the hard-disk’s partitions. Each device is assigned major-device number 11, and may be accessed as a block- or character-special device.

at handles two drives with up to four partitions each:

- Minor devices 0 through 3 identify the partitions on drive 0.
- Minor devices 4 through 7 identify the partitions on drive 1.
- Minor device 128 allows access to all of drive 0.
- Minor device 129 allows access to all of drive 1.

To modify the offsets and sizes of the partitions, use the command **fdisk** on the special device for each drive (minor devices 128 and 129).

To access a disk partition through COHERENT, directory **/dev** must contain a device file that has the appropriate type, major and minor device numbers, and permissions. To create a special file for this device, invoke the command **mknod** as follows:

```
/etc/mknod /dev/at0a b 11   0 ; : drive 0, partition 0
/etc/mknod /dev/at0b b 11   1 ; : drive 0, partition 1
/etc/mknod /dev/at0c b 11   2 ; : drive 0, partition 2
/etc/mknod /dev/at0d b 11   3 ; : drive 0, partition 3
/etc/mknod /dev/at0x b 11 128 ; : drive 0, partition table
```

Drive Characteristics

When processing BIOS I/O requests prior to booting COHERENT, many IDE drives use translation-mode drive parameters: number of heads, cylinders, and sectors per track. These numbers are called “translation-mode” parameters because they do not reflect true physical drive geometry. The translation-mode parameters used by the BIOS code present on your host adapter can be obtained using the command **info** from within the tertiary-boot routine **tboot**. (For details on **info**, see the Lexicon entry for **tboot**.) It is often necessary to patch the **at** driver with BIOS values of translation-mode parameters in order to boot COHERENT on IDE hard drives. In COHERENT versions 3.1.0 and later, drive parameters are stored in table **atparm** in the driver. For the first hard drive, number of cylinders is a short (two-byte) value at **atparm+0**, number of heads is a single byte at **atparm+2**, and number of sectors per track is a single byte at **atparm+14**. For the second hard drive, number of cylinders is a short value at **atparm+16**, number of heads is a single byte at **atparm+18**, and number of sectors per track is a single byte at **atparm+30**. For example, if **testcoh** is a kernel linked with the **at** driver and you want to patch it for a second hard drive with 829 cylinders, 10 heads, and 26 sectors per track, you can do:

```
/conf/patch testcoh atparm+16=829:s atparm+18=10:c atparm+30=26:c
```

To read the characteristics of a hard disk once the **at** driver is running, use the call to **ioctl** of the following form:

```
#include <sys/hdioctl.h>
hdparm_t hdparms;
      . . .
ioctl(fd, HDGETA, (char *)&hdparms);
```

where *fd* is a file descriptor for the hard-disk device and *hdparms* receives the disk characteristics.

Non-Standard and Unsupported Types of Drives

Prior releases of the the COHERENT **at** hard-disk driver would not support disk drives whose geometry was not supported by the BIOS disk parameter tables. COHERENT adds support for these drives during installation by “patching” the disk parameters into the bootstrap and the **/coherent** image on the hard disk.

Files

/dev/at* — Block-special files
/dev/rat* — Character-special files

See Also

device drivers, fdisk, hai, ideinfo

Notes

The driver **at** offers two varieties of polling: normal and alternate. Normal, as its name implies, is used with most varieties of AT controllers. Alternate polling is for Perstor controllers and some other older equipment. Using the wrong type of polling causes frequent controller timeouts and bad-track messages.

at also lets you specify the number of seconds to wait for a response from the drive after an I/O request. The default value is six. Some IDE drives occasionally become unresponsive for long intervals (several seconds) while control firmware makes adjustments to drive operation.

To set either the type of polling or the default waiting period, **su** to the superuser **root**; then **cd** to directory **/etc/conf** and run the script **at/mkdev**. This script will walk you through describing your AT controller to COHERENT. Once you have run this script, execute the command

```
/etc/conf/bin/idmkcoh -o cohtest
```

to create a test kernel that incorporates your changes; then reboot your system and invoke the new kernel, as described in the Lexicon entry **booting**. Note that the changes you make to the driver will not be seen by your COHERENT system until you boot the new kernel.

The **at** driver lets you have up to two AT hard disks on your system. Note, however, that in our experience, it is very difficult to combine different brands of AT hard disks and have both run successfully. This is especially true with Conner drives, which apparently do not cooperate with other IDE drives as master and slave. *Caveat utilitor.*

at — Command

Execute commands at given time
at [**-v**] [**-c** *command*] *time* [[*day*] *week*] [*file*]
at [**-v**] [**-c** *command*] *time month day* [*file*]

at executes commands at a given time in the future.

If the **-c** option is used, **at** executes the following *command*. If *file* is named, **at** reads the commands from it. If neither is given, **at** reads the standard input for commands.

If *time* is a one-digit or two-digit number, **at** interprets it as specifying an hour. If *time* is a three-digit or four-digit number, **at** interprets it as specifying an hour and minutes. If *time* is followed by **a**, **p**, **n**, or **m**, **at** assumes **AM**, **PM**, **noon**, or **midnight**, respectively; otherwise, it assumes that *time* indicates a 24-hour clock. Note that you should *not* type a colon ':' in the time string.

For example, the command

```
at -c "time | msg henry" 1450
```

set the **time** command to be executed at 2:50 PM, and pipe **time**'s output to the **msg** command, which will pass it to the terminal of user **henry**. The argument to the **-c** option had to be enclosed in quotation marks because it contains spaces and special characters; if this were not done, **at** would not be able to tell when the argument ended, and so would generate an error message. If you wish to pass information to a user's terminal with the **at** command, you must tell **at** to whom to send the information. The command

```
at 250p commandfile
```

sets the file **commandfile** to be read and executed at 2:50 PM. It is *not* necessary to use the file's full path name. Also, if the suffix **p** were not appended to the time, the file would be set to be read at 2:50 AM.

The time set in **at**'s command line is *not* the exact time that the command is executed. Rather, the daemon **cron** periodically executes the command **/usr/lib/atrun** to see if any commands have been scheduled commands to be executed at or before the present time. The frequency with which **cron** executes **atrun** determines the "granularity" of **at** execution times. To change when **cron** executes **atrun**, edit file **/usr/spool/cron/crontabs/root**. For example, the entry

```
0,5,10,15,20,25,30,35,40,45,50,55 * * * *   /usr/lib/atrun
```

sets **/usr/lib/atrun** to be executed every five minutes. Thus, the **at** command that is set, for example, to 2:53 PM will actually be executed at 2:55 PM. **atrun** executes specified commands when it discovers that the given time is past; therefore, **at** commands are executed even if the system is down at the specified time or if the system's time is changed.

The **at** command has two forms, as shown above. In the first form, the option *day* names a day of the week (lower case, spelled out). If **week** is specified, **at** interprets the given *time* and *day* as meaning that time and day the following week. For example, the command

```
at -c "time | msg henry" 1450 friday week
```

executes **time** and sends its output to **henry**'s terminal one week from Friday at 2:50 PM.

In the second form given above, *month* specifies a month name (lower case, spelled out) and the number *day* specifies a day of the month. For example, the command

```
at 1450 july 4 commandfile
```

set the file **commandfile** to be read at 2:50 PM on July 4.

If the **-v** flag is given, **at** prints the time when the commands will be executed, giving you enough information to plan for the execution of the command. For example, if it is now August 13, 1990, at 2:30 PM, and you type the command

```
at -v -c "/usr/games/fortune | msg henry" 1435
```

at will reply:

```
Tue Aug 13 14:35:00
```

indicating that the command will be executed five minutes from now. However, if you type

```
	at -v -c "/usr/games/fortune | msg henry" 1435 august 10
```

at will reply

```
	Sun Aug 10 14:35:00 1991
```

which indicates that on Sunday, August 10 of next year, at 2:35 PM, the COHERENT system will print a **fortune** onto your terminal.

Should you create such a long-distance **at** file by accident, you can correct the error by simply deleting the file that encodes it from the directory **/usr/spool/at**. The file will be named after the time that it is set to execute, plus a unique two-character suffix, should more than one command be scheduled to run at the same time. For example, the file for the above command would be named **9108101435.aa**.

Finally, note that the current working directory, exported shell variables, file creation mask, user id, and group id are restored when the given command is executed.

Example

The following example invokes the command **wall** at 11 P.M. to confirm that the **at** command is working properly:

```
	at -c "echo 'testing to see if cron is working' | /etc/wall" 2300
```

Files

/bin/pwd — To find current directory
/usr/lib/atrun — Execute scheduled commands
/usr/spool/at — Scheduled activity directory
/usr/spool/at/ *yymmddhhmm.xx* — Commands scheduled at given time

See Also

at, commands, cron

atan() — Mathematics Function (libm)

Calculate inverse tangent
#include <math.h>
double atan(*arg***) double** *arg***;**

atan() calculates the inverse tangent of *arg*, which may be any real number. The result will be in the range [-π/2, π/2].

Example

For an example of this function, see the entry for **acos()**.

See Also

errno, libm, tan(),
ANSI Standard, §7.5.2.3
POSIX Standard, §8.1

atan2() — Mathematics Function (libm)

Calculate inverse tangent
#include <math.h>
double atan2(*num, den***) double** *num, den***;**

atan2() calculates the inverse tangent of the quotient of its arguments *num*/*den*. *num* and *den* may be any real numbers. The result will be in the range [-π, π]. The sign of the result will have the same sign as *num*, and the cosine will have the same sign as *den*.

Example

For an example of this function, see the entry for **hypot()**.

See Also

errno, libm
ANSI Standard, §7.5.2.4
POSIX Standard, §8.1

ATclock — Command

Read or set the AT realtime clock
/etc/ATclock *[yy[mm[dd[hh[mm[.ss]]]]]]*

ATclock reads or sets your system's "hardware" time, which is stored in your system's CMOS. This clock should contain the current standard time for your locale.

With no argument, the command **ATclock** reads the hardware clock and returns a string in the format expected by the command **date**. With an argument, it sets the hardware clock to the given date. For example, to set your hardware clock to October 24, 1994, at 9:30 PM, use the command:

```
	/etc/ATclock 9410242130
```

ATclock also lets you reset the time incrementally: that is, you can reset only the year; the year and month; the year, month, and day; and so on down to the second.

Note that if you use **ATclock** to reset your hardware clock, you *must* reset it to the standard time in your locale, even if daylight-savings time happens to be in effect when you reset the clock. If you do not, COHERENT's commands that set the local time on your system (e.g., the command **date**) will be off by one hour when daylight-savings time is in effect.

The system startup file **/etc/brc** typically contains a command of the form

```
	date -s `/etc/ATclock`
```

to reset the time properly when the COHERENT system starts up.

See Also

brc, clock, CMOS, commands, date

atexit() — General Function (libc)

Register a function to be called when the program exits
#include <stdlib.h>
int atexit(void (*function***)**
void (**function***)();**

atexit() registers one or more functions to be called when the program exits. These registered functions can, for example, perform clean-up beyond what is ordinarily performed when a program exits. **atexit()** can register up to 32 functions.

function points to the function to be called. A registered function takes no arguments and returns nothing.

The functions that **atexit()** registers are called when the program exits normally, i.e., when the function **exit()** is called or when **main()** returns. They are called in *reverse* order of registration.

atexit() returns zero if *function* could be registered, and a value other than zero if it could not.

Example

This example registers two functions to be executed upon exiting: one displays a message, and the other waits for the user to press a key before terminating.

```
#include <stdlib.h>
#include <stdio.h>

void
lastgasp()
{
	fprintf(stderr, "Type return to continue");
}

void
get1()
{
	getchar();
}
```

```
main()
{
      /* set up get1() as last exit routine */
      atexit(get1);
      /* set up lastgasp() as exit routine */
      atexit(lastgasp);

      /* exit, which invokes exit routines */
      exit(EXIT_SUCCESS);
}
```

See Also

exit(), libc
ANSI Standard, §7.10.4.2

atof() — General Function (libc)

Convert ASCII strings to floating point
#include <stdlib.h>
double atof(*string***) char *** *string***;**

atof converts *string* into the binary representation of a double-precision floating point number. *string* must be the ASCII representation of a floating-point number. It can contain a leading sign, any number of decimal digits, and a decimal point. It can be terminated with an exponent, which consists of the letter **'e'** or **'E'** followed by an optional leading sign and any number of decimal digits. For example,

```
      123e-2
```

is a string that can be converted by **atof()**.

atof() ignores leading blanks and tabs; it stops scanning when it encounters any unrecognized character.

Example

For an example of this function, see the entry for **acos()**.

See Also

atoi(), atol(), float, libc, long, printf(), scanf(), stdlib.h
ANSI Standard, §7.10.1.1
POSIX Standard, §8.1

Notes

atof does not check to see if the value represented by *string* fits into a **double**. It returns zero if you hand it a string that it cannot interpret.

atoi() — General Function (libc)

Convert ASCII strings to integers
#include <stdlib.h>
int atoi(*string***) char ***string***;**

atoi() converts *string* into the binary representation of an integer. *string* may contain a leading sign and any number of decimal digits. **atoi()** ignores leading blanks and tabs; it stops scanning when it encounters any non-numeral other than the leading sign, and returns the resulting **int**.

Example

The following demonstrates **atoi()**. It takes a string typed at the terminal, turns it into an integer, then prints that integer on the screen. To exit, type **<ctrl-C>**.

```
#include <stdlib.h>

main()
{
      extern char *gets();
      extern int atoi();
      char string[64];
      for(;;) {
            printf("Enter numeric string: ");
            if(gets(string))
                  printf("%d\n", atoi(string));
            else
                  break;
      }
}
```

See Also

libc
ANSI Standard, §7.10.1.2
POSIX Standard, §8.1

Notes

atoi does not check to see if the number represented by *string* fits into an **int**. It returns zero if you hand it a string that it cannot interpret.

atol() — General Function (libc)

Convert ASCII strings to long integers
#include <stdlib.h>
long atol(*string***) char ****string***;**

atol() converts the argument *string* to a binary representation of a **long**. *string* may contain a leading sign (but no trailing sign) and any number of decimal digits. **atol()** ignores leading blanks and tabs; it stops scanning when it encounters any non-numeral other than the leading sign, and returns the resulting **long**.

Example

```
#include <stdlib.h>

main()
{
      extern char *gets();
      extern long atol();
      char string[64];

      for(;;) {
            printf("Enter numeric string: ");
            if(gets(string))
                  printf("%ld\n", atol(string));
            else
                  break;
      }
}
```

See Also

atof(), atoi(), float, libc, long, printf(), scanf(), stdlib.h
ANSI Standard, §7.10.1.3
POSIX Standard, §8.1

Notes

No overflow checks are performed. **atol()** returns zero if it receives a string it cannot interpret.

atrun — System Administration

Execute commands at a preset time

atrun is a program that executes programs at a time set by the command **at**.

When user **steve** types

```
      at 1230 /v/steve/lunchtime
```

the command **at** creates a shell script in directory **/usr/spool/at** that contains the information needed to execute command **/v/steve/lunchtime** at a later time — in this instance, 12:30 PM. The spooled file sits in **/usr/spool/at** until **/usr/lib/atrun** sees that the specified time has been reached. **atrun** then executes the spooled command and removes it from **/usr/spool/at**.

atrun is not a daemon; that is, it is invoked by another program, does its work, and exits. Thus, it is typically run periodically from an entry in the **cron** file owned by the superuser **root**.

See Also

Administering COHERENT, at

Notes

Although **atrun** technically is a command, it is never invoked by a user.

auto — C Keyword

Note an automatic variable

auto is an abbreviation for an *automatic variable*. This is a variable that applies only to the function that invokes it, and vanishes when the functions exits. The word **auto** is a C keyword, and must not be used to name any function, macro, or variable.

See Also

C keywords, extern, static, storage class,
ANSI Standard, §6.5.1

awk — Command

Pattern-scanning language
awk [*POSIX or GNU style options* **] -f** *program-file* **[--]** *file ...*
awk [*POSIX or GNU style options* **] [--]** *program-text file ...*

awk is a general-purpose language designed for processing input data. Its features allow you to write programs that scan for patterns, produce reports, and filter relevant information from a mass of input data. It acts upon the contents of each *program-file*, or the standard input if no *-program-file* is specified.

You can specify the program either as an argument (usually enclosed in quotation marks to prevent interpretation by the shell **sh**) or in the form **-f** *program-file*. If no **-f** option appears, the first non-option argument is the **awk** program.

awk views its input as a sequence of records, each consisting of zero or more fields. By default, newlines separate records and white space (spaces or tabs) separates fields. The option **-F***c* changes the input field separator characters to the characters in the string *c*. An **awk** program can also change the field and record separators. The program can access the values of each field and the entire record through built-in variables.

For details on the construction of **awk** programs, consult the tutorial to **awk** that appears in this manual. Briefly, an **awk** program consists of one or more lines, each containing a *pattern* or an *action*, or both. A *pattern* determines whether **awk** performs the associated *action*. It may consist of regular expressions, line ranges, boolean combinations of variables, and beginning and end of input-text predicates. If no *pattern* is specified, **awk** executes the *action* (the pattern matches by default).

An *action* is enclosed in braces. The syntax of actions is C-like, and consists of simple and compound statements constructed from constants (numbers, strings), input fields, built-in and user-defined variables, and built-in functions. If an *action* is missing, **awk** prints the entire input record (line).

Unlike **lex** or **yacc**, **awk** does not compile programs into an executable image, but interprets them directly. Thus, **awk** is ideal for quickly-implemented, one-shot efforts.

Examples

The following examples illustrate the economy of expression of **awk** programs.

The first example reads the standad input, and echoes all lines containing the string “COHERENT”:

```
awk '/COHERENT/'
```

To exit, type **<ctrl-D>/**

The built-in variable **NR** is the number of the current input record. The next example reads the standard input, and prints the number of records you typed after you exit (again, by typing **<ctrl-D>**):

```
awk 'END { print NR }'
```

The built-in variable **$3** gives the value of the third field of the current record. The last example sums the third field from each record you type on the standard input, and prints the total when you exit:

```
awk '{ sum += $3 }
      END   { print sum }'
```

See Also

commands, gawk, lex, sed, yacc
Introduction to the awk Language, tutorial.

Notes

Beginning with release 4.2.14 of COHERENT, **awk** has been replaced by **gawk**, the GNU implementation of this language. For details on this implementation of the **awk** language, see the Lexicon entry for **gawk**.