NAME

cc − C compiler (nmcc for cross-support)

SYNOPSIS

cc [option] ... *file* ...

nmcc [option] ... *file* ...

DESCRIPTION

Cc is the GENIX C compiler. It accepts several types of arguments:

Arguments whose names end with '.c' are taken to be C source programs; they are compiled, and object programs are left in files whose names are those of the source with '.o' substituted for '.c'. The '.o' file is normally deleted, however, if a single C program is compiled and loaded all at the same time.

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

The following options are interpreted by *cc*. See *ld*(1) for load-time options.

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

−w Suppress warning diagnostics.

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

−O Invoke an object code improver. Do not use on programs that contain **asm** statements as the result may not be correct.

−i Suppress optimizations that are incorrect when memory locations might spontaneously change value. C Memory mapped device register registers have this property. The **−i** option is ignored when **−O** is not present.

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

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

−C Prevent the macro preprocessor from removing comments.

−o *output*
Name the final output file *output*. When this option is used, *cc* leaves any existing file named *a.out* undisturbed.

−v List the utilities *cc* calls and their arguments on standard output. Information produced by the **−v** verbose flag is useful for debugging.

−vn List the utilities *cc* would call and their arguments. *Cc* does not actually call the utilities. The only result is the list -- *file* is not compiled.

−D*name***=***def*
−D*name* Define the *name* to the preprocessor, as if by '.sp 4800u
define'. If no definition is given, the name is defined as '1'.

−U*name*
Remove any initial definition of *name*. (The C preprocessor supplies initial definitions of '1' for the symbols "ns16000" and "unix".)

−I*dir* '#include' files whose names do not begin with '/' are always sought first in the directory of the *file* argument, then in directories named in **−I** options, then in directories on a standard list. Standard list is /usr/include or /usr/NSC/include for cross-support.

−B*string*
: Find substitute compiler passes in the file or path named *string*. **−B** appends cpp (for preprocessor), ccom (for the compiler proper), or c2 (for the optimizer) to *string.* If *string* is not specified, the default is /usr/c/o.

−t[**p02**]
: Find only the designated compiler passes in the file or path whose names are constructed by a **−B** option. **p** specifies the preprocessor, **0** specifies the compiler, and **2** specifies the optimizer. In the absence of a **−B** option, the *string* is taken to be /usr/c/n.

−g Print line numbers as comments in the assembly language source when the -S flag is also given. Otherwise, it calls the assembler with the **−g** flag, which causes the assembler to emit more symbol table information. See *as*(1) for details.

−F Gives fast access to global variables. It locates all non-extern declarations of variables in the local static base area; there is no common area. This results in a considerable increase in speed as all variable accesses in the module that declared the variable are SB relative, not external. Note that only one non-extern declaration of each variable in the whole program is possible. This option increases the object file size and the start-up time for a program, so is most useful for programs that execute for a long time and do not declare large uninitialized arrays.

−q Pass this switch to the assembler which then puts the link table relative to the SB register. This speeds up execution time of external calls and access time of external variables. See *as(1).*

Cc assumes that other arguments are loader option arguments, C-compatible object programs, typically produced by an earlier *cc* run, or libraries of C-compatible routines. These programs are loaded (in the order given) to produce an executable program with name *a.out.*

FILES

file.c	input file
file.o	object file
file.s	assembly file
a.out	loaded output
/usr/tmp/ctm? /tmp	temporary
/lib/cpp	preprocessor
/lib/ccom	compiler
/usr/c/occom	backup compiler
/lib/c2	opttional optimizer
/lib/crt0.o	run-time startoff
/lib/mcrt0.o	startoff for profiling
/lib/libc.a	standard library, see (3)
/usr/include	standard directory for ‘#include’ files

SEE ALSO

B. W. Kernighan and D. M. Ritchie, *The C Programming Language,* Prentice-Hall, 1978
B. W. Kernighan, *Programming in C—a tutorial*
D. M. Ritchie, *C Reference Manual*
as(1), monitor(3), prof(1), ddt(1), ld(1)

DIAGNOSTICS

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

CROSS-SUPPORT

In a cross-support environment *cc* is called *nmcc.* Temporary files are stored on /tmp instead of /usr/tmp. The default substitute compiler passes are in /usr/NSC/lib/o for **−B** and in /usr/NSC/lib/n for **−t**.

BUGS

The compiler currently ignores instructions to put **char**, **unsigned char**, **short** or **unsigned short** variables in registers. It previously produced poor, and in some cases incorrect, code for such declarations.

The NS16000 does not support signed bitfields so all bitfields are unsigned. However if a bitfield is declared as int or long instead of unsigned the compiler will generate int operations and comparisons on it rather than unsigned. The bitfield will always be a positive number.

Nested assignment statements do not always preserve the types of the variables being assigned. For example:

```
int a,c;
short b;
a=b =c;
```

will not always result in a only getting the low order 16 bits of c . It will often get the whole 32 bits. To be safe use:

```
b=c;
a=b;
```

in this situation.

Post-incrementing or post-decrementing a bit field (for example, “d*a.b++”) in an expression does not work. *Cc* only allows identifiers up to 31 significant characters because of an assembler restriction.