AMERICAN
INTERACTIVE
MEDIA
A Philips/PolyGram Corporation

AIM Technical Note #64

Using the GNU Cross Compiler

Charles Golvin

March 21, 1991

Capitol Disc recently merged two implementations of GNU to produce a cross compiler that runs under the SunOS and produces OS-9 assembly code. The GNU cross compiler employs the Microware assembler (o68) and linker (l68) to produce code executable on the CD-I player.

Using the GNU Cross Compiler

Background

The GNU compiler is widely distributed and has been ported to many machines, including Suns and 680x0 CPUs running OS-9. Capitol Disc has recently merged these two implementations to produce a cross compiler running under SunOS that produces OS-9 assembly code. While GNU also distributes an assembler and linker with this package, the development on these pieces has been slower. Consequently, the compilation path using the GNU cross compiler employs the Microware assembler (o68) and linker (l68) to produce code executable on the CD-I player.

The most persuasive arguments as to why one should employ this path in preference over the Microware cross compiler path are that it offers smaller and more optimized code and ANSI compatibility. In general, I have observed that code size is reduced between five and ten percent using the GNU cross compiler, but I am unable to verify a concomitant increase in performance of the code. ANSI compatibility provides you with better information during compilation and helps prevent parameter mismatches, use of uninitialized variables, and errors that the Microware compiler will happily let you make. The down side of this argument is that you may not use the GNU compiler to produce code that can be debugged using **srcdbg** or **unibug**, and a nontrivial amount of work is required to generate proper include files for full ANSI compatibility. The first problem forces you to write code that has preprocessor directives to control whether output will be produced by the GNU compiler or the Microware compiler. Fortunately for all of you, Blake Senftner has already done most of the work to alleviate the latter problem.

My intent in this document is only to tell you how to use the GNU compiler in its non-ANSI mode (the default). Blake has written a document describing how to convert over to the ANSI version; you may wish to consult his document, as well.

Getting Started

Currently, the compiler is located in

```
/eng/coredisc/gcc-1.37.1/bin{3,4}
```

You will need to place one of these directories in your path: bin3, if you are using a Sun3; bin4, if you are using a Sun4 (Sparc) machine. It is not sufficient to simply invoke the compiler with the full path name, because the compiler executive (gc68) forks the preprocessor (gcc) and the compiler itself (gxx). Once you have added this entry to your path, you will be able to simply invoke the compiler as gc68.

Compiling

Based on the description of the cross compiler above, you might think that the output of the GNU cross compiler would be an assembly language file. Luckily, the cross compiler is kind enough to give you a direct pipeline into *o68*, the Microware assembler so that your makefile will look essentially the same as it does with the Microware compiler: one compile line for each source file and one link line for the output program.

Your compile lines must contain these flags (meaning follows):

`-DOSK`	This defines the symbol OSK to the preprocessor. On os9 systems OSK is always defined.
`-c`	Do not run the linker.
`-o68`	Run the OS-9 assembler following compilation.
`-o` *filename*	Place the assembler's output in *filename*.

The other flags I always use are:

`-O`	Optimize code.
`-W`	Generate full warnings.

or

`-Wunused`	Only issue warnings about unused variables.

If you are compiling code that has ANSI type function declarations:

`-ANSI`	Support ANSI standard.

The following is a summary of all the options recognized by gc68. On the Sun, after you have added the correct entry to your path, you may regenerate this summary using one of the following commands:

```
unix% set noglob;gc68 -? >& more;unset noglob
unix% gc68 -\? >& more
```

Control compile stages

`-c`	Do not run linker
`-o68`	Use o68 before assemble
`-S`	Only compile source files
`-E`	Only preprocess source files
`-M`	Only print dependencies
`-MM`	Only print dependencies sans system header files
`-pipe`	Connect preprocessor and compiler via pipe

`-debug`	Do not execute but only print command to execute
`-v`	Print version and commands to execute
`-Q`	Not quiet mode
`-BPREFIX`	Use PREFIX for commands
`-bPREFIX`	Use machine dependent PREFIX for commands
`-LDIR`	Search DIR for library files
`-TDIR`	Specify DIR of temp files
`-FPROGsSIZEpPRIO`	Fork PROGram (one of cccp cc1 o68 r68(020) l68) with additional stack SIZE and PRIOrity
Output	
`-o FILE`	Output to FILE
`-x`	Put executable file into default execution directory
Warning	
`-W`	Issue warnings
`-WCASE`	CASE is one of implicit return-type unused switch comment trigraphs all shadow id-clash-LEN pointer-arith cast-qual write-strings
`-w`	Inhibit all warnings
Portability	
`-ansi`	Support ANSI standard
`-traditional`	Try to imitate traditional C compiler
`-pedantic`	Support pedantic ANSI standard
Preprocessor	
`-DMACRO[=DEF]`	Define MACRO (by DEF)
`-UMACRO`	Undefine MACRO
`-IDIR`	Search include file in DIR
`-I-`	Dir specified by the -Idir before -I- applies to only user include files
`-iFILE`	Get only #define's in FILE
`-nostdinc`	Do not use standard directory of include files
`-C`	Include comments in preprocessor output
`-trigraphs`	Support ANSI trigraphs
Compiler	
`-O`	Optimize
`-p`	Generate profile code too
`-a`	Generate block profile code too
`-mMACHINESPEC`	MACHINESPEC is one of (c)68000 (c)68020 68881 soft-float short (no)bitfield rtd (no)remote (no)stack-check (no)gss

-fFLAG	FLAG is one of caller-saves combine-regs cond-mismatch float-store force-addr force-mem inline-functions keep inline-functions omit-frame-pointer pcc-struct-return strength-reduce unsigned-char volatile writable-strings asm defer-pop function-cse signed-char delayed-branch and the above with prefix no- and fixed-REG call-used REG call-saved-REG
-dLETTERS	dump debug info in various stages. LETTER is one of y r j s L f c l g J d

Linking

As stated above, you must use the Microware linker to create the executable. You may use the same command line to cc68 as you previously used to do your link, with one crucial addition: you must link to the library

```
/eng/coredisc/gcc-1.37.1/gnulib.l.
```

Careful!

There are some implementation differences between the two compilers. The most glaring difference concerns the use of pre- and post-increment inside an expression. The GNU compiler does not apply the increment until the entire expression has been evaluated; whereas the Microware compiler applies the increment following the evaluation of the individual expression where the increment is used. The following code illustrates this point:

```
unsigned char red = 0;
int color, idx = 0;

color = cp_clut( idx++, red++, red++, red++ );
```

Under the Microware compiler, you end up with clut 0 = (0,1,2); however, under the GNU compiler you get clut 0 = (0,0,0), which is probably not the result you want.

An Example

Let's suppose that you have been working on this program *brain_hurts*, which is a self contained program. Using the Microware compiler, you do the following in your makefile:

```
DEFS = /usr/local/os9/DEFS
LIBS = /usr/local/os9/LIB
CFLAGS = -v=${DEFS} -dlint -dNO_BVC
LFLAGS = -l=${LIBS}/os9.l -l=${LIBS}/sys.l
         -l=${LIBS}/iff.l -l=${LIBS}/cdi.l \
         -l=${LIBS}/cdisys.l

brain_hurts: ../rels/brain_hurts.r
     cc68 ../rels/brain_hurts.r ${LFLAGS}
     -f=../bin/brain_hurts
../rels/brain_hurts.r: ../src/brain_hurts.c
     cc68 ${CFLAGS} -r=../rels ../src/brain_hurts.c
```

Assuming the above defines, you could adapt to using the GNU compiler with the following lines in your makefile:

```
GCFLAGS = -Dlint -DNO_BVC -DOSK -c -O -o68 -I${DEFS}
GLFLAGS = -l=/eng/coredisc/gcc-1.37.1/lib/gnulib.l

gbrain_hurts: ../rels/gbrain_hurts.r
     cc68 ../rels/brain_hurts.r ${GLFLAGS} ${LFLAGS}
     -f=../bin/brain_hurts
../rels/gbrain_hurts.r: ../src/brain_hurts.c
     gc68 ${GCFLAGS} ../src/brain_hurts.c -o
     ../rels/gbrain_hurts.r
```

Given this makefile, the commands

```
unix% make brain_hurts
unix% make gbrain_hurts
```

would produce Microware compiler-generated code and GNU compiler generated code, respectively.