Compiling C programs for the 68040 board

Alastair Scobie (AJS)

April 16, 2000

1 Introduction

This note describes how to cross-compile C programs and download them to the 68040 board. It also describes how to assemble 68040 code and link code files together.

2 How to speak to the 68040 board

The Linux PC has two ports, `ttyS0` and `ttyS1`. The P7 port of the 68040 board is normally connected to `ttyS0`. The Linux `minicom` command is used to speak to the board.

```
[pclab1]ajs: minicom ttyS0
```

< ... the screen clears ... >

```
Welcome to minicom 1.75

Press CTRL-A Z for help on special keys
```

< ... turn on the 68040 board power or hit the reset key ...>

```
Copyright 1993 Motorola Inc. All rights reserved
Copyright 1993 Software Components Group. All rights reserved
IDP ROM Version 3.0
Processor Type is: M68040
DRAM Size: 2 Megs

ROM68K :->
```

You can exit `minicom` by typing `CTRL-A X`. Type `CTRL-A Z` for the complete list of `minicom` commands.

3 Cross-Compilation

For this exercise, versions of the GNU C compiler (`gcc`), GNU assembler (`gas`) and GNU loader (`gld`) have been configured to cross-compile on a Linux PC producing 68000 code.

3.1 Standard gcc

The `gcc` compiler, like most UNIX C compilers, consists of one compiler driver program (`gcc`) and four sub-programs called by the driver program in the following sequence :

		Input		Output	
`cpp`	the C pre-processor	C code	`(.c)`	expanded C code	`(.c)`
`cc1`	the compiler	raw C code	`(.c)`	assembler	`(.s)`
`as`	the assembler	assembler	`(.s)`	object, non resolved, non relocated	`(.o)`
`ld`	the loader	object	`(.o)`	executable, resolved, relocated	`(a.out)`

By setting certain flags, you can stop the compilation at certain stages :-

`gcc -S` compiles C code producing assembler output (`cpp -> cc1`)
`gcc -c` compiles C code producing an object file (`cpp -> cc1 -> as`)

Normally only the compiler driver program `gcc` lives in `/usr/bin` with the sub-programs residing in a subtree of `/usr/lib/gcc-lib`.

3.2 Hacked gcc

The 68040 cross compiler driver program is called `gcc68` to avoid clashing with the standard driver program `gcc`. The compiler sub-programs live in `/usr/lib/gcc-lib/m68k-68board-blob/2.7.2.1`. The assembler and loader are softlinked into `/usr/bin` as `gas68` and `gld68` respectively.

3.3 C environment

There is NO library support. STDIO support does NOT exist. Simple floating point operations are supported at both single and double resolution. There are two library routines `mon_putc` and `mon_getc` which put and get characters to and from the terminal.

3.4 Structure definitions

The file `/usr/lib/gcc-lib/m68k-68board-blob/include/m68kboard.h` includes structure definitions for the registers of the on-board 68681 and 68230 devices. The `volatile` modifier ensures that the `gcc` optimizer does not optimise out access to these registers - this could cause problems when busy waiting on the registers. This include file can be referenced from your C source with :-

```
#include <m68kboard.h>
```

3.5 Example session

This simple program, `frog.c`, repeatedly reads a character using the `mon_getc` function and prints it using the `mon_putc` function.

```
main()
{
      char c;

      while (( c = mon_getc()) != 4) {
            mon_putc(c);
      }
}
```

The following command would be used to compile, assemble and link `frog.c`, placing code that can be run on the 68040 board in the file `frog`. Note that the `-v` option is only used to direct the compiler to produce verbose messages.

```
[pclab1]ajs: gcc68 -v -O -o frog frog.c
Reading specs from /usr/lib/gcc-lib/m68k-68board-blob/2.7.2.1/specs
gcc version 2.7.2.1
 /usr/lib/gcc-lib/m68k-68board-blob/2.7.2.1/cpp -lang-c -v -undef
-D__GNUC__=2 -
D__GNUC_MINOR__=7 -Dmc68000 -D__mc68000__ -D__mc68000 -D__OPTIMIZE__
-Dmc68020 -
D__mc68020__ -D__mc68020 frog.c /tmp/cca21541.i
GNU CPP version 2.7.2.1 (68k, MIT syntax)
#include "..." search starts here:
#include <...> search starts here:
 /usr/lib/gcc-lib/m68k-68board-blob/2.7.2.1/include
 /usr/lib/gcc-lib/m68k-68board-blob/2.7.2.1/sys-include
 /usr/lib/gcc-lib/m68k-68board-blob/include
End of search list.
 /usr/lib/gcc-lib/m68k-68board-blob/2.7.2.1/cc1 /tmp/cca21541.i
-msoft-float -m6
8020 -quiet -dumpbase frog.c -O -version -o /tmp/cca21541.s
GNU C version 2.7.2.1 (68k, MIT syntax) compiled by GNU C version 2.7.2.1.
 /usr/lib/gcc-lib/m68k-68board-blob/2.7.2.1/as -mc68020 -o /tmp/cca215411.o
/tmp
/cca21541.s
 /usr/lib/gcc-lib/m68k-68board-blob/2.7.2.1/ld -N -e _start -Ttext 40000 -o
frog
 /usr/lib/gcc-lib/m68k-68board-blob/2.7.2.1/startfile.o
-L/usr/lib/gcc-lib/m68k-
68board-blob/2.7.2.1 /tmp/cca215411.o -lgcc -lgcc
[pclab1]ajs:
```

If instead we had wanted to perform each individual step separately we would go through the following stages.

We would use the following to produce assembler code from the C code (the `-O` flag instructs the compiler to produce optimized code).

```
[pclab1]ajs: gcc68 -O -S frog.c
```

The resulting assembler code would be placed in `frog.s`. The contents of that file are reproduced below :-

```
#NO_APP
gcc2_compiled.:
___gnu_compiled_c:
.text
      .even
.globl _main
_main:
      link a6,#0
      jbsr ___main
L2:
      jbsr _mon_getc
      cmpb #4,d0
      jeq L3
      extbl d0
      movel d0,sp@-
      jbsr _mon_putc
      addqw #4,sp
      jra L2
L3:
      unlk a6
      rts
```

`.text`	declares the following assembler code as being code text.
`.even`	directs the assembler to start on a word (16bit) boundary
`.globl _main`	declares the symbol `_main` as a global symbol, to be seen by the loader at link time. Normal symbols (eg `L3`) are local to this file.
`_main`	declares the start of the C function `main()`

The routine `__main` which is called almost immediately, is a routine which initialises the C environment.

Note that `gcc` prefixes all external names (eg for external functions) with an underscore character in its output, eg `mon_putc` becomes `_mon_putc`.

The routine calls the two external routines `_mon_putc` and `_mon_getc`.

We now use the following to assemble `frog.s`, placing the object code in `frog.o`.

```
[pclab1]ajs: as68 -o frog.o frog.s
```

We can now use the `nm68` command to gleam some information about `frog.o` :-

```
[pclab1]ajs: nm68 -g frog.o
           U ___main
00000000   T _main
           U _mon_getc
           U _mon_putc
[pclab1]ajs:
```

This shows that `frog.o` references `_mon_getc`, `_mon_putc` and `__main` and defines `_main`.

The code file `frog.o` now needs to be linked with the loader (linker) `gld68` to produce and executable file. It needs to be linked with the object file wrapper `startfile` that defines `_mon_getc` and `_mon_putc` and calls your code, and the object library `libgcc` that supports floating point operations and certain maths operations.

```
[pclab1]ajs: ld68 -o frog -N -e _start -Ttext 40000 /usr/lib/
gcc-lib/m68k-68board-blob/2.7.2.1/startfile.o frog.o -L /usr/lib/
gcc-lib/m68k-68board-blob/2.7.2.1 -l gcc -lgcc
```

`-e _start` sets the start execution address of the program to be the address of the function `_start`. This function is defined in `startfile.o` and calls the `main()` function in your program.

`-Ttext 40000` sets the base address of the `text` (code) segment. This is the address at which you should load your code. The function `_start` is planted at this address.

Another `gld` option that you may want to use is `-Tdata`. Normally any `static` data you declare will be located intermingled with your code. This is fine when you download your code into RAM, but not very great when your code is in EPROM. The option `-Tdata` specifies that any `static` data that you declare will be located at the given address.

Fortunately, you can use the compiler driver (`gcc`) to call the linker if you are prepared to accept the default values for the switches described above.

```
[pclab1]ajs: gcc68 -o frog frog.o
```

4 Downloading a program

To download a program to the 68040 board you use the `CTRL-A G` command of `minicom` in conjunction with the `dc` command of the 68040 ROM-68K monitor.

```
ROM68K :->  dc
Waiting for S-Records from host...
```

<... now type CTRL-A G to invoke the download menu ...>

```
+==================[Run a script]==========================+
|                                                          |
| A - Username          :                                  |
| B - Password          :                                  |
| C - Name of script    :                                  |
|                                                          |
|    Change which setting?   (Return to run, ESC to stop)  |
+==========================================================+
```

<... now type C to set the `Name of script` field to the executable you wish to ...>
<... download to the board. The executable from our example in section 3 is ...>
<... `frog`. Hit return and `minicom` will convert your executable into S-records ...>
<... that the board is expecting ...>

```
9 records read
ROM68K :-> go 40000
The quick brown fox jumped over the lazy moon...
```

Unfortunately the ROM-68K monitor does not exit gracefully when a program completes - it generates an illegal instruction to return to the monitor prompt. The only way that you can be sure that your program has completed and not crashed is to check that the PC register is at 40006 at the time of the exception.