Notes for cs452
Compiling & Running a Program

Bill Cowan
University of Waterloo

All the action in the first assignment takes place inside the small grey box, which should have two RS-232 cables attached to the DB-9 connectors labelled `COM1` and `COM2`. One cable is connected to the train controller; the other is connected to a monitor.

To compile and run your first program do the following.

1. Log on to your account in the student Linux environment.
2. Look at the files in /u/wbcowan/cs452/io, which comprise the source code and make files for a small busy-wait IO library which you can use for diagnostic output.
3. Create a program that uses the busy wait functions to put output on the terminal attached to the TS-7200. The first argument of each busy wait function is a channel. You should be including `ts7200.h` in your program because it contains the definitions you need. Output to appear on the monitor is directed to `COM2`.
4. Add `/u/wbcowan/gnuarm-4.0.2/libexec/gcc/arm-elf/4.0.2` and `/u/wbcowan/gnuarm-4.0.2/arm-elf/bin` to your `PATH` variable.
5. Create a `Makefile` to compile and link your program.
   i. I like to keep my `Makefile`s simple, as you can see.
   ii. I like the compilation to leave around assembly language versions of the compiled program (`.s` files) because I can later, if necessary, edit the assembly file.
   iii. The link-loader generates two important files, a .map file, which is a load map and a `.elf` file. The `.elf` file is the executable, which you download onto the ARM box. It's not necessary to understand the `.elf` file.
   The load file shows you where each component is in the executable. The components of the executable are functions, listed by name. Beside each function is its offset, the difference between the base address at which the executable is loaded, and the address of the first instruction to be called when the function is called. This information can be very useful when debugging.
   iv. You can create a linker script, for the loader asking it to put things in non-standard places. (If you want to see the default linker script used by the loader when you do not supply one, `ld --verbose` will print it on the terminal. I hope you're disgusted.) The target file that I used, `orex.ld`, is in the directory with the `Makefile`. I

do not claim that this is a model link file; it's something crude to show you how simple a link file can be.

v. Disregard the warning from the loader that it is placing the entry point at `0x00080000`. The RedBoot loader will find your entry point from the ELF-header and position the file appropriately in memory.

6. Place the `.elf` file in the tftp uploading directory, `/u/cs452/tftp/ARM`. Coordinate with whoever else is using the server because you want to make certain that you are not overwriting somebody else's file, and that they are not overwriting yours.
7. Go to the terminal attached to the TS-7200 board you are using. It should be displaying the `RedBoot>` prompt. If not, press the reset button: you will see the processor booting into RedBoot. At the `RedBooot>` prompt type
   ```
   load -b 0x00218000 -h 10.15.167.5 "ARM<your program name>.elf"
   ```
   The IP number `10.15.167.4` should also work. These IP numbers occasionally change. If they don't work try typing host `tftp[12].student.cs.uwaterloo.ca` to a shell prompt.
8. The address 0x00218000 is chosen to be enough above RedBoot that you won't run into any problems. The `-b` argument tells RedBoot where in memory to place your executable, and should be unnecessary. The `-h` argument is the IP number of the TFTP server in the undergrad environment. (Sorry, no dns service.) RedBoot will tell you where it assumes that starting point of execution is, which is usually the base address, `0x00218000` in this example.
9. Type `go`, or `go 0x00218000` at the RedBoot prompt and your program will execute. If it terminates it will return to the RedBoot prompt. If it goes into an infinite loop the prompt will not return and you will have to push the reset button, the black button among the connectors.
10. At any time you can look at the memory using RedBoot. The memory dump command, for example,
    ```
    x -b 0x00218000 -l 0x40
    ```
    shows you `0x40 = 64` bytes starting at `0x00218000`. The first instruction, where execution starts, is
    ```
    mov     ip, sp
    ```
    which is `0xe1a0c00d`. Note. The linker asked for little-endian.