shan@iha.dk
V1.0

Building C++ programs for target

Introduction

In this exercise you will use direct compiler invocation, `make` and Eclipse to rebuild the simple programs from Lab Exercise *Building C++ programs for host* for execution on the DevKit8000.

Prerequisites

In order to complete this exercise, you must:

- Have completed the exercises *Building C++ programs for host* and *Connecting to target*
- Have access to a *target* - DevKit8000.

Goal

When you have completed this exercise, you will:

- have learned the difference between compiling for *host* and *target*
- have learned how to use direct compiler invocation to compile a program for *target*
- have written a makefile and used make for compilation for both *host* and *target*
- have used Eclipse to create and compile a program for *target*
- have run programs that are built by direct compiler invocation, make and Eclipse

---

Exercise 1 Cross compiling the program Hello World and running it on target

Copy the source code for the "Hello world"-program from Lab Exercise *Building C++ programs for host* to a new directory. Use direct invocation of the Angstrom C++ compiler `arm-angstrom-linux-gnueabi-g++`[1] to create an executable `hello.target` for *target*. You can use the command `file hello.target` to verify that the executable file `hello.target` is indeed built for the ARM *target* architecture. Transfer the executable to *target* and execute it.

Exercise 2 Using makefiles to build host and target programs

Copy the source code and the `makefile` from Lab Exercise *Building C++ programs for host* for the program "Hello Program" to a new directory on *host*. Then copy `makefile` to `makefile.target`, so that you have two makefiles. Change `makefile.target` to compile `hello` for *target* - if you have made good use of variables in the makefile for Lab Exercise *Building C++ programs for host*, the changes are minimal.

Build your program for both *host* and *target* using `make`. You can specify that make should use `makefile.target` instead of the default `makefile` by using the command:

```
make -f makefile.target
```

Then execute the *host* executable on the *host* and the *target* executable on the *target*.

[1]The compiler is in your *PATH* variable meaning that you have direct access to it

Exercise 3 Cross compiling with more extended makefiles

Repeat Exercise 2, this time for the program `parts` from Lab Exercise *Building C++ programs for host*.

Exercise 4 Improving cross compilation handling in makefiles

In Exercises 2 and 3 above, there are two flaws: When you have made an executable, say for `host`, and then re-make it for `target`, you actually overwrite the host executable. In many cases, this is unacceptable. Furthermore unless you have placed the intermediate created object files in separate directories, then the build process will fail since the linker will try to link object of the wrong type.

Rewrite `makefile` and `makefile.target` so that `make` places the executable(& objects) for `host` in the sub-directory `host/` and the executable(& objects) for target in the subdirectory `target/`. Also make sure that `make clean` and `make -f makefile.target clean` will not remove the opposite executable(& objects) - in other words, `make clean` must not remove the target executable(& objects), and `make -f makefile.target clean` must not remove the `host` executable(& objects).

Make no mistake this exercise will take time and it would be very prudent to determine ahead of time, how the different challenges should be dealt with.

Exercise 5 Using Eclipse instead of makefiles

Open your solution to Lab Exercise *Building C++ programs for host* in Eclipse and add a configuration “Target” to your project. Set it up for target build and use it to build the program for target. Transfer the executable to target and execute it.

NOTE: Be sure to complete this exercise. At a later lecture we will be cross debugging in eclipse. Having at least a fundamental understanding of eclipse will therefore be of huge help.

Exercise 6 Merging makefile and makefile.target (OPTIONAL)

The solution found in exercise 4 has but one flaw and that is that we now have to maintain two different makefiles. A huge improvement would therefore be to merge the two makefiles.

If you choose to do so, then the following should yield an executable viable for the *target*:

```
make ARCH=target
```

The following should produce an executable for *host*:

```
make ARCH=host
```

If nothing is specified it should produce an executable for *host*.

This feature is obviously done by using conditionals in `make`, how this is done is something *google* can help with...

While you are at it, what about handling debug build as well? :-) A debug build for target could be something like:

```
make ARCH=target DEBUG=y
```

Enjoy!!!