CSCE 230L – Lab 1: Altera Monitor Program

January, 2014

Purpose

The first goal of this lab is to introduce you to the program that you will use for the first several weeks of lab. You will be using the Altera Monitor Program for the labs to learn about assembly programming. This first part will follow section 3 of the Altera Monitor Program tutorial, available on the following website. If you want to install the program on your own computer, you first need to install Quartus (web edition), and then the Altera Monitor Program (filter material-> Choose HDL: VHDL, Choose Board: DE1). You can read part 2 for more information.

http://www.altera.com/education/univ/software/monitor/unv-monitor.html

ftp://ftp.altera.com/up/pub/Altera_Material/12.0/Tutorials/Altera_Monitor_Program.pdf

The second goal is to learn how to create a project starting from scratch and to get practice writing a simple program. You can refer to the following material to learn how to use assembly instructions.

http://www.altera.com/literature/hb/nios2/n2cpu_nii51017.pdf

Objectives

- Altera Monitor Program
  - Installing the program (optional)
  - Creating, compiling, and loading a project
  - Debug basics (step by step execution and breakpoints)
  - Observe common errors
- Learn basic syntax and structure in assembly programming
- Follow execution of an assembly program
- Practice writing some simple assembly programs

Deliverables – due Wednesday (01/22)/Thursday (01/23) 11:59 PM

- All questions in a text file (50 points)
- Code from part 2 (50 points)

Lab

Part 1: Altera Monitor Program

Go through the tutorial section 3 on the website given earlier and then answer the following questions**. Choose system DE1 Basic Computer instead of DE2 Basic** Computer, and choose your own directory at corresponding step.

In some of the following steps, you are supposed to come across error messages.

1. With the "getting started" program loaded on to the board, do the following debugging.
    a. Set a breakpoint at the line: *ldwio r4, 0(r15)* (in the DO_DISPLAY section) and then run the program– *in the grey space to the left of the instruction click once, this should set a breakpoint on the instruction and a red circle should appear next to the instruction. After this click the green arrow (hotkey F3) to run the code.* On the right hand side there will be a panel called registers, what are the values of pc, r4, and r15?

    b. Now single step to execute the instruction, right after 1.a without restarting. What are the values of pc, r4, and r15?

    c. Look at the instruction containing 'HEX_bits' just before the breakpoint. Edit 'HEX_bits' in the instruction to be "HEX_bit" (open the .s file 'getting_started.s' in a text editor) and then compile. What happens?

2. **Change back** what you do in 1.c. Go to Settings->System Settings. (If these options are grey, disconnect the board first by clicking Action->Disconnect.) Change the system to one of the others, which is not DE1. Click OK. Click Actions->"Download System…" and then try to load your program on the board. Describe what happens.

3. Now open the source code of the program to edit, that would be the .s file. You have to edit with a program other than the Monitor Program (e.g. Notepad++, Visual Studio). Make the following changes ***one at a time*** and describe what happens when you compile (looking into the Info& Errors window).
    a. Change the first “movia” to “movi”.

    b. In the WAIT section, change “bne” to “bn”.

    c. In the DELAY section, change “subi r7,r7,1” to “subi r7,r7,r1”.

    d. Turn the board off and on, click ⇩ to load the program to the board.

Part 2: Assembly Basics

In the Altera Monitor Program, every assembly program we write will start with .text to tell the compiler where the code starts. Following that we need .global <label> so that there’s a starting point in your program and also allow your program to be used by other programs. By default, the <label> will be ’ _start’ when you create a new project. It’s easiest to keep it that way. Following that we put our label and a colon and then instructions afterward. The beginning of a program will look as follows:

```
        .text
        .global _start

_start:                 /*you can change this label name as you want*/

        <instructions…>
```

To end the program, you can add ‘.end’ at the end of your codes.

In the Monitor Program itself, you can only compile and view the code. To edit it, you need to open the .s file in a text editor. Also, when creating a new project, you need to define your assembly files first because when you create a new project, it will ask you which files to include in the project. When asked to specify the system, either of the DE1 systems will work. However, for the labs in this class select the DE1 Basic Computer. For right now, after you compile a program and download it onto the board, just single step through your program (as opposed to just clicking the green arrow and letting it run) so that you can see the register values change.

Conditional Example

Instructions in assembly language execute sequentially according to how they are stored, Aside from addition and subtraction, conditionals are also a basic and necessary functionality in assembly programs. If you look at the list of instructions and find the instruction *blt*. When is it used? (And no, it's not for when we're hungry). If you look at some of the nearby instructions, we can see that there are a bunch of different branching ones. Branch instructions need an address to go to if the condition is satisfied. Typically we use ***labels which will represent addresses*** when your code is compiled. For instance, you could have a branch go to '_start'.Suppose we want to convert the following code into an assembly program.

First we need to set the initial values of a, b, and c as 11, 5, 2, respectively. A good way to initialize values is to use the following instruction: addi r2, r0, 3. This adds together r0 (which is always 0) and the value 3 and stores the result in r2.

```
if( a < c) {
        b = a + c ;
} else {
        b = b - 3;
}
a++;
```

Initializing all the values gives us the following code so far:

```
        .text
        .global _start

_start:
        addi    r2, r0, 11      /* initialize a to 11 */
        addi    r3, r0, 5       /* initialize b to 5 */
        addi    r4, r0, 2       /* initialize c to 2 */
```

Now we need to create the conditional block. Since we need to check if **a** is less than **c**, we can use the instruction branch-if-less-then (blt), so "blt r2, r4, LABEL" (which means: if r2<r4, execute instruction block named after 'LABEL', otherwise execute next instruction right after current one) . You can name the label any single word, but a good name is helpful. Since the

program will branch to the label if the condition is met, a name this label might be “TRUE”. So now the code looks like the following, the label “FALSE” is just for clarity:

```
          .text
          .global _start

_start:
          addi    r2, r0, 11      /* initialize a to 11 */
          addi    r3, r0, 5       /* initialize b to 5 */
          addi    r4, r0, 2       /* initialize c to 2 */

          blt     r2, r4, TRUE    /* branch if a < c */
FALSE: …
TRUE: …
```

For the FALSE section, we want to execute b = b – 3, so an addi instruction will do. The ‘i’ in this add instruction means one of the operands is an immediate value, or constant value. For the TRUE section we’re executing b = a + c. Since these are all in registers, a normal add instruction will do. Now our code looks like the following:

```
          .text
          .global _start

_start:
          addi    r2, r0, 11      /* initialize a to 11 */
          addi    r3, r0, 5       /* initialize b to 5 */
          addi    r4, r0, 2       /* initialize c to 2 */

          blt     r2, r4, TRUE    /* branch if a < c */
FALSE:    addi    r3, r3, -3      /* b = b – 3 */
TRUE:     add     r3, r2, r4      /* b = a + c */
```

We’re almost done with the conditional block, just one more part missing. ***Assembly code executes sequentially***, so if the branch is not taken, then the FALSE section will be executed, and then the TRUE section. This last part we don’t want to happen. The solution is to make the program skip the TRUE section if the FALSE section is executed by using an ***unconditional branch***. We will have the branch go to a label after the entire conditional block, so an appropriate name could be ENDIF. Adding the a++ after the conditional is just an addi instruction. So now our completed code looks as follows:

```
            .text
            .global _start

_start:
            addi    r2, r0, 11       /* initialize a to 11 */
            addi    r3, r0, 5        /* initialize b to 5 */
            addi    r4, r0, 2        /* initialize c to 2 */

            blt     r2, r4, TRUE     /* branch if a < c */
FALSE:      addi    r3, r3, -3       /* b = b – 3 */
            br      ENDIF

TRUE:       add     r3, r2, r4       /* b = a + c */

ENDIF:
            addi    r2, r2, 1        /* a++ */
```

Task 1 (50 points)

Suppose we have three different numbers stored in r1, r2, and r3. Write an assembly program that will store the value of the largest number of the three in r5. You can choose the initial values for r1, r2, and r3.

Hint:

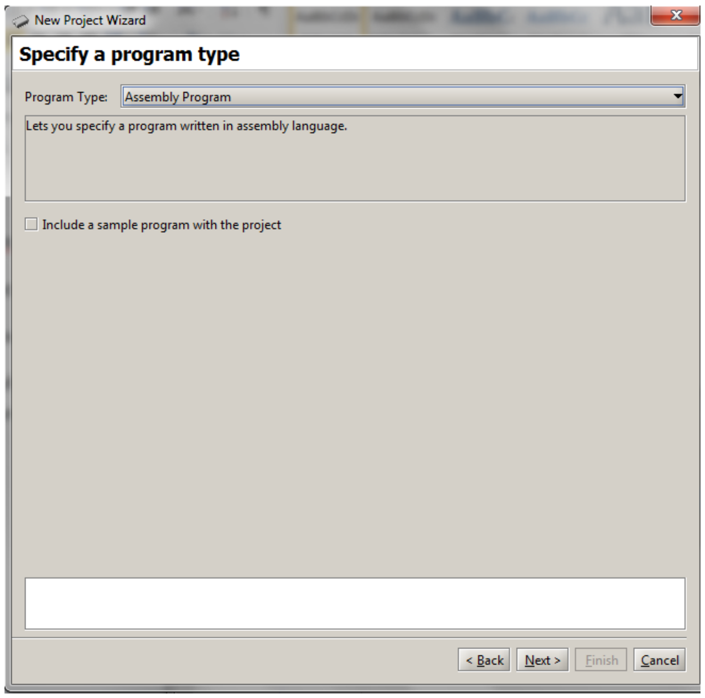
1. Write the pseudocode first

2. Label different sections

3. Convert it into assembly program

Suggestion:

You can check whether your program is right or not by building a new project, the first several steps are the same with tutorial except the following steps:



New Project Wizard

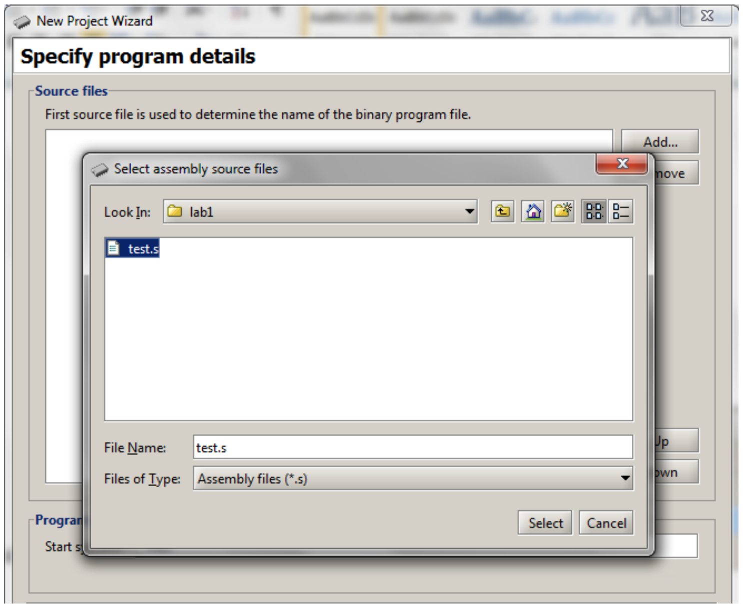
Specify a program type

Program Type: Assembly Program

Lets you specify a program written in assembly language.

☐ Include a sample program with the project

Click ‘Next’, and ‘add’, to choose your source code



New Project Wizard

Specify program details

Source files

First source file is used to determine the name of the binary program file.

Select assembly source files

Look In: lab1

test.s

File Name: test.s

Files of Type: Assembly files (*.s)

You can compile your code and download it to the board. Single step each instruction and check the changes of registers to see whether everything goes as expected, and r5 will get the right value after the execution of your instructions.

Submission

When you are all done, submit on handin the answers to all the questions as a text file, and also the source code for the assembly programs (.s files).