CSCE 230L – Lab 2: Loops and Arrays

January, 2014

Objectives

- A better understanding and familiarity with assembly languages
- Learn how data and arrays are structured and accessed
- Learn how to use loops

Useful References

Appendix B section 1 to 6 in the textbook (some parts are not necessary to read at this point, such as B 4.7)

List of instructions for the NIOS II Processor:
http://www.altera.com/literature/hb/nios2/n2cpu_nii51017.pdf

What to Turn In

- Pre-lab questions (due at the beginning of lab; in paper)
  - All questions total to 20 points
- Code from tasks 1, and 2 (submit on handin)
  - Task 1 – 30 points
  - Task 2 – 40 points
- Questions asked at part2 (submit on handin)
  - Task 2 – 10 points

For the code, all you need to submit are the assembly files (**.s files**). Other files from the Altera Monitor Program project are not needed. For in-lab questions, put your answers in a text file.

Name (1 point):

Pre-lab (19 points)

1. Make sure you read through the entire lab! It will help you work quicker in lab and can also help you get through the pre-lab.

2. Assembler directives are in every program and have some useful functions. You may have noticed some already used like *.text* and *.global*. The *.text* signifies where the code section starts and the *.global* gives your code a label so that it can be used by other programs. Other common ones can define variables, or place the code at a certain location in memory. After the code in a program, there can also be a data section where variables can be defined. This is indicated with the *.data* directive, which comes after the code, but before the *.end* directive. Section B.6 in the textbook lists some common directives and their use.
    a. Refer to page 549-550 in your textbook, how would you define the following in a .data section?
        - An 8-bit value of 42 in memory at location N1
        - An array of seven 32 bit integers in memory at location ARR
        - The letters A-F (8-bit each) in memory at location ALPHA

3. Look at example problem B.12 and figure B.17 in the textbook on page 565. The following questions refer to it.

a. At the start of the LOOP section, why is the instruction *ldb* used instead of *ldw*?

b. At the end of the LOOP section the comments say to loop back if not done, but the instruction is an unconditional branch. How does the code finish?

c. If n=0 and there are no ASCII digits to convert, how many times would the line *br LOOP* be executed?

Lab

Part 1

In the last lab, you learned some basics of how assembly programs are structured and had practice writing code and using conditionals. If we go one step further, and use a register as a counter, we can implement a loop. For example, the following code segment will add r3 to r4 10 times and then continue.

```
        addi    r2, r0, 0       /* counter = 0 */
        addi    r1, r0, 10      /* end = 10 */
LOOP:   add     r4, r4, r3      /* r4 = r4 + r3 */
        addi    r2, r2, 1       /* increment counter */
        blt     r2, r1, LOOP    /* keep looping as long as counter < end */
        …       /* code continues after loop */
```

Task 1

To get some practice making your own loops, write an assembly program that will sum the numbers from 20 to 40 (including 40). Look at the debugging window after your loop, what is the value of the register in which you are storing the sum?

Part 2

Now that you are an expert on loops, we will take a look at one of the most basic data structures: arrays. Arrays are referenced with a pointer (an address location in memory) to where the values are stored. To read and write to the array, we use the load and store instructions, with the memory location being the register that is being used as the pointer.

Use the following as a format:

ldw destReg, offset(srcReg),

Here destReg is the register in which you want to store the array value, srcReg is the address of the array in memory, and offset (**must be a numerical value**) is used as an index. For example, ldw r3, 8(r5) will read from memory location [r5 + 8] and store that value in r3. One thing to keep in mind is that offset is in terms of bytes, so if we had an array of 32-bit integers, the offset would need to increase by 4 to get to the next integer in the array.

Suppose we have an array of integers [2, 8, 13, 1, 65], with the address of the array at LOC. The following code demonstrates how to use the array by adding up all 5 values and storing the result back to the first location in the array.

```
addi    r1, r0, 0       /* set sum = 0 */
movia   r2, LOC         /* move the address of the array into r2 */
ldw     r3, 0(r2)       /* load the first element in the array into r3 */
add     r1, r1, r3      /* sum += first element (2) */
ldw     r3, 4(r2)       /* load the second element in the array into r3 */
add     r1, r1, r3      /* sum += second element (8) */
ldw     r3, 8(r2)       /* load the third element in the array into r3 */
add     r1, r1, r3      /* sum += third element (13) */
ldw     r3, 12(r2)      /* load the fourth element in the array into r3 */
add     r1, r1, r3      /* sum += fourth element (1) */
ldw     r3, 16(r2)      /* load the fifth element in the array into r3 */
add     r1, r1, r3      /* sum += fifth element (65) */
stw     r1, 0(r2)       /* store the sum in the first location in the array */
```

Part 2 Question

1. Since the offset has to be a numerical value when we compile, what can we do if we want to have a loop which traverses an array of arbitrary length? Hint: the memory location accessed by the load instruction is equal to srcReg + offset. The following two ways are the same:

```
ldw     r3, 8(r2)
```

```
addi    r2, r2, 8;
ldw     r3, (r2)
```

2. What should the offset be for loading from an array of the following data types:
   - Byte
   - HalfWord
   - Word

Task 2 Exercise

Define in your data section an array of 15 arbitrary integers. Remember to use the '.data' directive and put it toward the bottom of your code. Now write a program that uses a loop to sum all the values in the array.

Deadline for in-lab tasks:

Submit the answers and codes at

Tuesday 01/28/2014 11:59 PM if you are at Wednesday's section

Wednesday 01/29/2014 11:59 PM if you are at Thursday's section