CSCE 230 – Lab 2: Loops and Arrays

August 28th/29th 2015

Objectives

- A better understanding and familiarity with assembly language
- 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 (Due at beginning of lab in paper)
    - Totals 20 points

Files and questions do not need to be handed in. Checkoff for this lab is due on September 1st

Pre-lab (20 points)

Name: **Joshua Behlen**

1. 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. Below 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.

   Refer to page 549-550 in your textbook, how would you define the following in a .data section? Each answer should be of the format

   LABEL: TYPE VALUE1….VALUEN

   A. An 8-bit value of 42 in memory at location N1

      N1: .byte 42

   B. An array of seven 32 bit integers in memory at location ARR

      ARR: .word 0…6

   C. The letters A-F (8-bit each) in memory at location ALPHA

      ALPHA: .ascii A…F

2. 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*?

      "ldb" is used because they need to load a single byte (the next ASCII digit) and not a word.

   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?

      It will always loop back if it's not done because it's an unconditional branch (it always occurs).

   C. If n=0 and there are no ASCII digits to convert, how many times would the line *br LOOP* be executed? (This is not zero and will require some thought)

      If n=0, then when the instruction 'subi r2, r2, 1' is completed it will be some random value and when the next instruction, 'beq r2, r0, DONE' is up an error will be thrown for an undefined value.

Lab (80 points)

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 11 to 41 (inclusive), and then store that value into R7. Make sure to test this using the monitor program and make sure R7 equals 806 at the end of your program. Make sure to set a breakpoint after the loop and read what R7 is to see if it worked.

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)        OR        ldw    RA, #(RB),
```

Here destReg(RA) is the register in which you want to store the array value, srcReg(RB) is the address of the array in memory, and offset (**must be a numerical value, you cannot put a Register here or it will not compile**) 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.** For example, the instruction used right above would load the 3rd element in an integer 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 Questions

1. Given the instruction below, how many elements am I loading from the array, and what should be the unknown number in the *addi R4, R4, #* instruction that would allow me to move from one integer value to the next in the array? (Think of the size of an integer)

```
            movia   R4, ARRAY
            addi    R5, R0, 10
LOOP:       beq     R0, R5, DONE
            ldw     r2, 0 (R4)
            subi    R5, R5, 1
            addi    R4, R4, #
            br      LOOP
DONE:       nop
```

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 7 arbitrary integers. Remember to use the '.data' directive and put it toward the bottom of your code **but before the .end directive or you will not have the data**. Now write a program that uses a loop to sum all the values in the array. **This must be done in a loop, hard coding it to go through each value will not receive any credit.**

Checkoff due September 1st