Lab 2: Memory Operations, Loops, and Conditionals

Description
Preparation (2 marks)
In Lab (2 marks)
Quiz (1 mark)
Further reading

Description

The purpose of this lab is to familiarize you with Nios II assembly language programming and using the debugger to observe how your program functions. It also covers concepts like sign-extension, endianess and alignment.

You will be given a list of 10 numbers in memory. Write a program to iterate through this list of numbers, determine whether each number is negative, then write negative numbers to one output list in memory, and positive numbers to another list. Count the number of negative and positive numbers and have the results in `r2` and `r3`, respectively. When the program ends, examine the two output lists of numbers for correctness, using the debugger.

The input list of signed halfwords begins at location `IN_LIST`. The two output lists of signed words start at locations `OUT_NEGATIVE` and `OUT_POSITIVE`.

There are three parts to this lab:

1. Process a 10-element list of signed halfwords, as described above
2. Do the same thing with a variable-length (zero terminated) list
3. Do the same thing with a linked list of signed words as input

In each part, the output list should consist of signed words.

Sample Input

For convenience, the sample input is also available here: lab2_sample.s.

```
.equ RED_LEDS, 0xFF200000          # (From DESL website > NIOS II > devices)

.data                              # "data" section for input and output lists

IN_LIST:                           # List of 10 signed halfwords starting at address IN_LIST
    .hword 1
    .hword -1
    .hword -2
    .hword 2
    .hword 0
    .hword -3
    .hword 100
    .hword 0xff9c
    .hword 0b1111
LAST:                               # These 2 bytes are the last halfword in IN_LIST
    .byte  0x01                     # address LAST
    .byte  0x02                     # address LAST+1

IN_LINKED_LIST:                     # Used only in Part 3
    A: .word 1
       .word B
    B: .word -1
       .word C
    C: .word -2
```

```
        .word E + 8
    D:  .word 2
        .word C
    E:  .word 0
        .word K
    F:  .word -3
        .word G
    G:  .word 100
        .word J
    H:  .word 0xffffff9c
        .word E
    I:  .word 0xff9c
        .word H
    J:  .word 0b1111
        .word IN_LINKED_LIST + 0x40
    K:  .byte 0x01                  # address K
        .byte 0x02                  # address K+1
        .byte 0x03                  # address K+2
        .byte 0x04                  # address K+3
        .word 0

OUT_NEGATIVE:
    .skip ?                         # Reserve space for 10 output words

OUT_POSITIVE:
    .skip ?                         # Reserve space for 10 output words

#---------------------------------------

.text                   # "text" section for code

    # Register allocation:
    #   r0 is zero, and r1 is "assembler temporary". Not used here.
    #   r2  Holds the number of negative numbers in the list
    #   r3  Holds the number of positive numbers in the list
    #   r_  A pointer to ___
    #   r_  loop counter for ___
    #   r16, r17 Short-lived temporary values.
    #   etc...

.global _start
_start:

    # Your program here. Pseudocode and some code done for you:

    # Begin loop to process each number

        # Process a number here:
        #    if (number is negative) {
        #        insert number in OUT_NEGATIVE list
        #        increment count of negative values (r2)
        #    } else if (number is positive) {
        #        insert number in OUT_POSITIVE list
        #        increment count of positive values (r3)
        #    }
        # Done processing.


        # (You'll learn more about I/O in Lab 4.)
        movia  r16, RED_LEDS          # r16 and r17 are temporary values
        ldwio  r17, 0(r16)
        addi   r17, r17, 1
        stwio  r17, 0(r16)
```

```
        # Finished output to LEDs.
    # End loop


LOOP_FOREVER:
    br LOOP_FOREVER                  # Loop forever.
```

Sample Output

IN_LIST has 10 numbers, 4 negative and 5 positive numbers. For part 1, when your program ends, the output should be the following:

- `r2` should be 4
- `r3` should be 5
- The first four words in memory starting at location OUT_NEGATIVE should be -1, -2, -3, and -100
- The first five words starting at location OUT_POSITIVE should be 1, 2, 100, 15 and 513.

Preparation (2 marks)

1. What is a) 1 b) -1, encoded as a signed word in hexadecimal?
2. What is a) 1 b) -1, encoded as a signed halfword in hexadecimal?
3. How do you convert a signed halfword to a signed word such that it represents the same (decoded) value?
4. What is the difference between the following instructions: ldw, ldh, ldb, ldhu, ldbu
5. What is the difference between the following instructions: stw, sth, stb
6. What is the hexidecimal value of the signed halfword at the address LAST in the array?
7. What is the hexidecimal value of the signed word at the address K in the linked list?
8. Is the signed word at address `I` positive or negative?
9. How do you check the value of the registers (`r2` and `r3`) *using the debugger* at the end of your program?
10. How would you know at what address the output lists begin?
11. How do you examine the contents of the two output lists in memory using the debugger?
12. Why is the final `br LOOP_FOREVER` necessary? (Hint: What instructions would the processor execute after finishing processing the input list if `br LOOP_FOREVER` were not there?)
13. For the linked list whose first element is at IN_LINKED_LIST, show how the linked list will appear in memory by completing the following table, assume the linked list starts at address 0x1000:

Address	**Data Word**
0x1000	0x00000001
...	...

14. Draw the linked list as a block diagram.
15. Write the sequence of numbers, in hexadecimal, seen while traversing the linked list
16. Write a few lines of assembly to perform an unaligned memory access, test it in the lab and record the result
17. Write the assembly program that separates a list of numbers into a negative and positive list, and counts the number of each, as described above.

In-lab (2 marks)

Create 3 assembly programs with the names given below. Note: you will be able to reuse most of the code from part 1 in parts 2 and 3.

1. **part1.s:** Demonstrate your working program that operates on a list of **10 signed halfwords**:
    - Single-step your program for the first few iterations to show that your code can process both negative and positive numbers correctly. Point out how the values of relevant registers change after each number is processed.
    - Set a breakpoint at a suitable location and show the rest of the numbers being added one by one into the output lists. Do the values in `r2` and `r3` match the red LEDs after all iterations are over?
    - When the program is finished (how do you know?), examine the final output lists in memory and counts in `r2` and `r3`. Verify using the debugger that all outputs are correct.
2. **part2.s:** Change part1.s to accept a zero-terminated variable-length input list (Don't erase part 1!):
    - Make your program work with an input list of **variable length** (up to 10 halfwords), where the last valid entry is followed by an element with value zero. Categorize numbers into negative and positive, and

terminate your program when a number zero is seen (but do not process the zero). In the sample input given above, there should be 2 negative and 2 positive numbers.

3. **part3.s:** Implement another variation of the program that accepts a linked list as input instead of an array:
    - Make your program accept an input linked list of **signed words** located at label IN_LINKED_LIST. Each entry in the linked list now contains *two* consecutive words. The first is the data value, which you should process in the same way as before (classify as negative or positive, write to output list, and keep counts), and the second value is a pointer to the next entry in the linked list. In the sample input given above, the first element in the linked list has the value 1, and the next entry in the linked list is located at (IN_LINKED_LIST+8). The second entry has value -1 and a pointer to (IN_LINKED_LIST+16). The linked list terminates when the next-entry pointer is zero.
4. Package your source files for all three parts into one `code.zip` file, then **submit your code** on Blackboard

Quiz (1 mark)

Be prepared to answer any questions about how to use the debugger to examine CPU and memory state using the Monitor Program, load and store operations to memory, and how to write structured programming constructs (loops, if-then-else, etc.) in assembly. Also be prepared to demonstrate the effects of an unaligned memory access (Preparation #16).

Further Reading

Nios II has 31 general-purpose registers (`r0` is always zero, `r1` through `r31` are general-purpose). For more information on how these registers are usually used by other people or C compilers (i.e., by "convention"), see Page 2 of the Nios II Application Binary Interface.