CS/COE 0447 Spring 2017
Lab 3: Memory, Endianness and Control Flow

Released: Thursday 26 January 2017, 03:00pm EST
Due: Monday 6th February 2017, 11:59pm EST

Directions: Each student must submit their own solutions via CourseWeb. If you choose to work with a neighbor or partner, you must clearly include your partner's name with your submission. **Only your last submission will be graded as a complete unit, so make sure everything's there!**

For Parts 1 and 3 of this lab, you must name your files in a specific format. Each part will tell you the name to use.

Part 1: Memory

Consider the following definition of variables in memory:

```
.data
y:.word 33
z:.word 0
x:.word 16

.text
# instructions go here
```

That above program tells MARS to set aside three words of memory. It initializes each word to a specific value. Each word can be loaded from or stored to by referring to each word's name ("x", "y", and "z"). These words of data will be set aside in an area separate from the program's instructions.
a) Write a MIPS program that subtracts x from y and stores the result in z (z = y – x). Name this program **lab03part1-xxxxxx.asm**, where xxxxxx is your Pitt ID. Your program should load x and y from memory into two different registers, perform the operation, and then store the result back into z's memory location.

Hint: consider using the "`la`" (load address) pseudoinstruction as well as the "`lw`" (load word) instruction.

Something to think about: Can you find x, y, and z in MARS's data segment view? What are their addresses and what arrangement are they in (i.e., which comes after the other).

b) Take your code from part 1a and modify it so that after z has been stored to, both x and y are overwritten with z's value. **Use constant offsets in your store instructions to do this.** An example of using a constant offset is something like this: `lw $t0, -8($t1)`. In this case, the constant offset is –8. The address stored in $t1 will have –8 added to it, and then the word stored in that location (i.e., the word at address $t1 – 8) will be loaded by the processor and put into $t0.

c) Take your part code from part 1b and modify it so that the variables are now halfwords. Use the following definition of variables in memory. Modify your program to use the proper load/store halfword instructions instead of the load/store word instruction you were using:

```
.data
y:.half 33
z:.half 0
x:.half 16
```

Something to think about: Can you find x, y, and z in MARS's data segment view? What are their addresses?

d) Take your part 1c program and modify it so that the variables are bytes. Modify your program to use the proper load/store byte instructions. Use the following definition of variables in memory:

```
.data
y:.byte 33
z:.byte 0
x:.byte 16
```

Something to think about: Can you find x, y, and z in MARS's data segment view? What are their addresses?

In **CourseWeb**, please answer these questions:

Question 1: Submit your file lab03part1-xxxxxx.asm in CourseWeb, where xxxxxx is your Pitt ID:

Submit your program from part 1d. Be sure to name the file lab03part1-xxxxxx.asm.

You do not need to submit a program for 1a, 1b, or 1c.

Part 2: Endianness

Endianness[1] describes how computer architects arrange bytes in words. Within each word, endianness specifies the byte order. It does **not** affect the bit order within each byte. Let us explore byte order by putting four bytes into memory two different ways.

The following MIPS code defines space in the data segment for four bytes and initializes those four bytes to the following values. We can refer to the start of this byte sequence by using the arbitrary label "*a*."

```
.data
a: .byte 0x7a, 0x7b, 0x7c, 0x7d
```

Copy the code above to the simulator and assemble it. Then, look in the Data Segment for those four bytes.

In **CourseWeb**, please answer these questions:

Question 2: Order the bytes above as they are shown in the Data Segment window from left to

[1] See slide 29 on page 5 of http://people.cs.pitt.edu/~childers/CS0447/lectures/mips-isa2.pdf.

right.

<u>Question 3:</u> What is the address of the byte with value 0x7b?[2]

Tip: Write a simple program to load one byte into register $t1 from memory. Ensure that the byte loaded into register $t1 is 0x4c. Remember from what effective address (the effective address is the sum of the base address plus the offset) you loaded the byte.

If our purpose is to use these 4 bytes as a word, we could have defined the previous label as follows:
four_bytes: `.word 0x7a7b7c7d # 7*16^7 + 0xa*16^6 + … + 7*16^1 + 0xd*16^0`

Replace *a*'s definition in the simulator with "`a: .word 0x7a7b7c7d`" and assemble the code again.

<u>Question 4:</u> In what order are those bytes shown in the Data Segment window from left to right now?

<u>Question 5</u>: In the hexadecimal number 0x7a7b7c7d which byte is the most significant? That is, which byte carries the most weight?

<u>Question 6:</u>

```
.data
a: .word 0x7a7b7c7d
```

After the code above is assembled, which byte has the greatest address?

<u>Question 7:</u> Is the simulator little endian or big endian? How can you tell? (Think about your answers to Questions 4 and 5. Also, see footnote 1 on the previous page.)

Part 3: Array Manipulation and Control Flow

You can declare an array of words as the following:

```
.data
Array_A:    .word 0xa1a2a3a4, 0xa5a6a7a8, 0xacabaaa9
```

This directive declares an array called '`Array_A`' which has 3 words in it. You can use memory access instructions '`lw/sw`' to access each word of this array using proper offset. For example, if you want to access the 0th word element, you can add 0 to the base address of the array to calculate the effective address of the 0th word element. If you want to access *i*th word element, you can use the offset 4**i* to the base address of the array to access the *i*th word element of the array. An array of 3 words can also be considered an array of 6 half-words, or as an array of 12 bytes.

[2] To review how Mars orders bytes in the Data Segment, please see Figure 1 in "Lab1 Instructions."

Although this array 'Array_A' is declared as a word array, you can access half-words using 'lhu/sh' and bytes using 'lbu/sb' instructions using proper offsets relative to base address. For example, the offset for the 3rd half-word will be 4 whereas the offset for the 3rd word will be 8. Notice the **unsigned form** of load and store instructions used here which is needed to make sure that the half-words or bytes are not sign-extended during load/store operation.

Write a program that asks the user for the element type (word/half/byte). For the element type, consider 'w' for a word, 'h' for a half-word, or 'b' for a byte. Notice that 'w', 'h', and 'b' are characters and you can read a character using **syscall 12 (read character)**, which represents the character using its ASCII value. A place to find an ASCII table is the MIPS "Green Sheet." The ASCII table shows, for example, that the code for 'w' is 0x77. Thus, you may determine whether a user had entered a 'w' by comparing 0x77 to the character that syscall 12 returned.

The element type tells the program how to access the array. For example, if the element type is byte, then the program will print an array of bytes.

Print the **third** element of the array using **syscall 34 (print integer hexadecimal)**, **syscall 1 (print integer)** alongside its address using **syscall 34**.

Here are sample outputs from your program:

```
Please enter element type ( 'w' -word,  'h' -half,  'b' -byte):
b <--- this is the input
Here is the output (address, value in hexadecimal, value in decimal):
0x10010002, 0x000000a2, 162
```

```
Please enter element type ( 'w' -word,  'h' -half,  'b' -byte):
h <--- this is the input
Here is the output (address, value in hexadecimal, value in decimal):
0x10010004, 0x0000a7a8, 42920
```

Please make sure your output line contains the string "Here is the output (address, value in hexadecimal, value in decimal):" as shown above in the sample output.

Question 8: Submit your file lab03part3-xxxxxx.asm in CourseWeb, where xxxxxx is your Pitt ID. Output format will be strictly checked.