CSC B58 – Winter 2015
Assignment 3 – Some Assembly Required

The due date for this assignment is: Fri. 27, at 12noon (Hand-in at the start of lecture. Submit code on mathlab)

Name (last, first):

Student No.:

I certify that I have read the UTSC policy on academic honesty and plagiarism, and that all work I am providing with this assignment is my own.

Student signature: ________________________

Note: There is a penalty of 3 marks for failing to complete the above section.

Part	Marks
1	/ 38
2	/ 50
Total	/ 88

Updated by F. Estrada, Dec. 2014

CSC B58 – Winter 2015
Assignment 3 – Some Assembly Required

This assignment is designed to help you practice your skills and improve your knowledge of MIPS 32 assembly language programming. At the same time, it is intended to help you improve your understanding of how memory is organized, how code and data are stored and manipulated by a CPU, and how variables, pointers, and data stored in memory are related.

Learning goals:

- You will strengthen your understanding of the *programmer's* memory model

- You will learn to access and manipulate data stored in memory via pointers, and understand how to use indirect addressing and indexed addressing to access information

- You will think about how simple constructions in a high-level programming language are actually implemented at the assembly level

- You will implement subroutine calls, and understand how parameters are passed back and forth between functions in a program

- *You will learn to think in terms of memory addresses and memory contents*

Skills developed:

- Working with pointers and pointer arithmetic instead of variable names

- Implementing simple routines in assembly, thinking about how to optimize Code

- Debugging code when all you can see is memory and register values

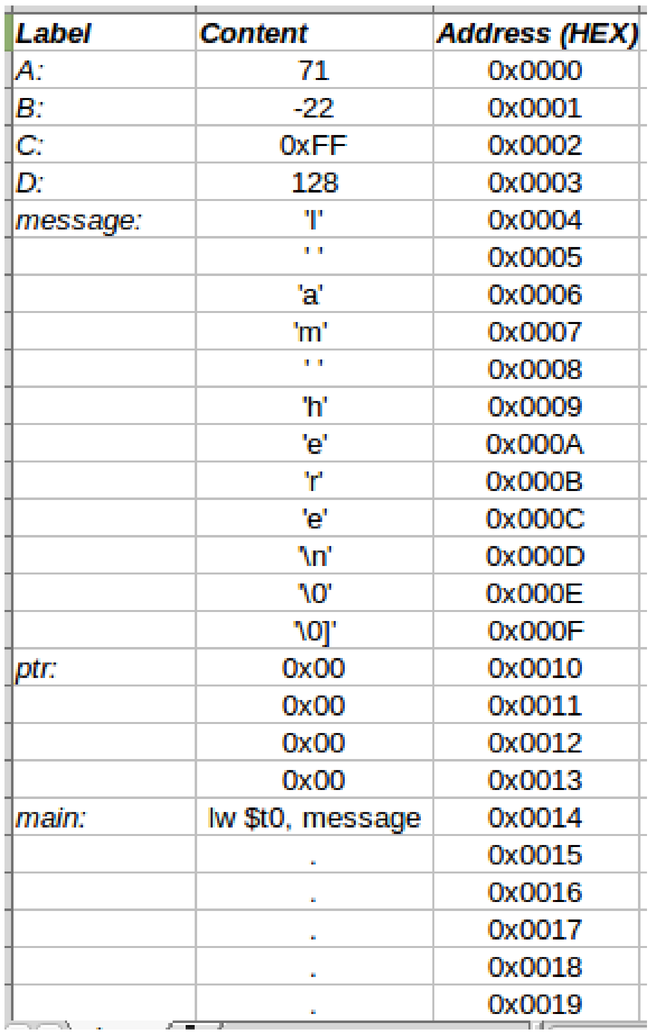
- Thinking about what is going on inside a program at the finest possible level of detail (for a programmer!)

References:

- Links to MIPS 32 and SPIM tutorials (on course web page)

Part 1 – Memory model and pointers

The diagram below shows a small segment of the ***main memory bank*** in a ***MIPS32*** machine such as the one SPIM provides. Consider the diagram carefully, and answer the corresponding questions in the space provided. When register values are requested, show ***each byte*** of the register either in ***hexadecimal***, or ***binary***. ***ASCII tables*** exist online. Assume a big-endian CPU.



Label	Content	Address (HEX)
A:	71	0x0000
B:	-22	0x0001
C:	0xFF	0x0002
D:	128	0x0003
message:	'I'	0x0004
	' '	0x0005
	'a'	0x0006
	'm'	0x0007
	' '	0x0008
	'h'	0x0009
	'e'	0x000A
	'r'	0x000B
	'e'	0x000C
	'\n'	0x000D
	'\0'	0x000E
	'\0]'	0x000F
ptr:	0x00	0x0010
	0x00	0x0011
	0x00	0x0012
	0x00	0x0013
main:	lw $t0, message	0x0014
	.	0x0015
	.	0x0016
	.	0x0017
	.	0x0018
	.	0x0019

1) (3 marks) What is the content of $t0 after
lw $t0, message

2) (3 marks) What is the content of $t1 after
lw $t1, ptr

3) (3 marks) ***(T/F plus explanation)*** $t1 now has a pointer to the start of '***A:***'

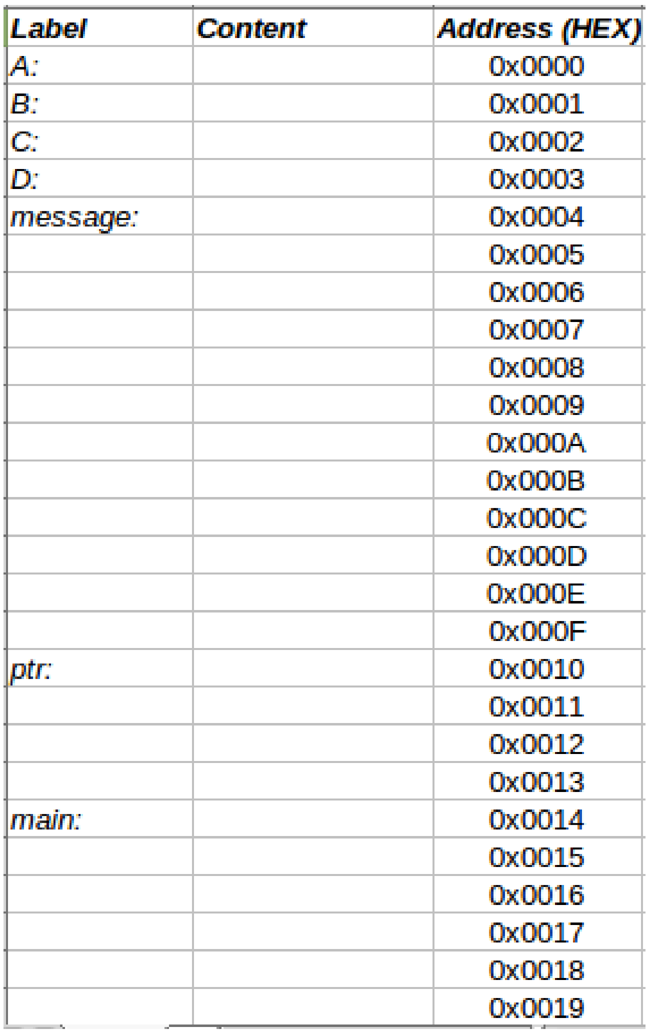
4) (5 marks) What is the content (in memory) of '***message***' after
add $t0, $t0, 11
sw $t0, message

5) (5 marks) What is in register ***$t4*** after:

li $t3, 27
lb $t2, B
add $t0, $t3, $t2
lw $t4, 4($t0)

Part 1 – Memory model and pointers

The diagram below shows the ***same memory region*** from the previous page. Show what the contents of the memory are ***after executing the small assembly program on this page.*** Assume none of the instructions in the previous page (questions 1-5) were executed so the memory contents are the original from the previous page's table.



Label	Content	Address (HEX)
A:		0x0000
B:		0x0001
C:		0x0002
D:		0x0003
message:		0x0004
		0x0005
		0x0006
		0x0007
		0x0008
		0x0009
		0x000A
		0x000B
		0x000C
		0x000D
		0x000E
		0x000F
ptr:		0x0010
		0x0011
		0x0012
		0x0013
main:		0x0014
		0x0015
		0x0016
		0x0017
		0x0018
		0x0019

1) (14 marks) Show the memory contents after the following program is executed.

```
# somewhere in the code
lb $t0, B
lb $t1, A
add $t1, $t1, $t0
sw $t1, X              # 2 marks
lw $t0, ptr
addi $t0, $t0, 4
lw $t1,($t0)
lw $t2, 4($t0)
sw $t1, 4($t0)         # 3 marks
sw $t2, ($t0)          # 3 marks
sw $t2, -8($t0)        # 3 marks
li $v0,4
la $a0, message
syscall                # see Q.2
addi $t0, 10
sw $t2, ($t0)          # 3 marks
b main
```

2) (5 marks) What is printed by the print string system call above?

3) (3 marks) What happens when 'b main' is executed?

Part 2 – MIPS 32 programming

Write the MIPS32 assembly programs that fulfil the specified tasks.

You will submit your ***.s*** files electronically via mathlab (instructions below). Make sure to comment your code appropriately, your TA will have discretion to make a global deduction of up to 10 marks for code without comments.

Also, note we're looking at the efficiency of your code. Write the simplest assembly program you can think of that performs the specified task. While performing the task correctly is the most important thing here, overly bloated, confusing, or badly organized code will be penalized with a few marks.

Be sure to test all your programs on spim. If your code does not run on spim 'as is', your TA won't be able to mark it. I can't stress this enough, make sure your code works. Partial marks are only awarded to working code that has bugs but does the correct thing at least in some of our test cases.

Problem 1 (15 marks)

Simple assembly operations and loops

Write an assembly program that

1) Prompts the user to input 10 integers between 0 and 25
   (it should check they are within these bounds)
2) Computes and prints the number of times each element is repeated
   e.g. if the user inputs: [1 5 7 5 2 5 3 4 1 9]
   Your program will print:
   1: 2
   5: 3
   7: 1
   2: 1
   3: 1
   4: 1
   9: 1
   Note that the printout does not have to be sorted by input value,
   But also note that repeated elements like 1 or 5 are reported ***once***.

Call your program "***repeats_studentNo.s***" where ***studentNo*** is your student number.
e.g. ***repeats_0123456780.s***

Part 2 – MIPS 32 programming

Problem 2 (15 marks)

Function calls, control structures, and loops

Write an assembly program called ***'divisors_studentNo.s'*** that

1) Prompts the user to input a number between 1 and 289 (why not?)
2) Finds and prints all numbers between 1 and the input number that divide the input evenly (you need to learn to use the div instruction).

   (e.g.) If the input is: 15
   The output should be:

```
15 is exactly divisible by
1
3
5
15
```

Important note:

The actual code that tests whether or not a number evenly divides the input ***must be implemented as a function call***. You must use proper ***stack management*** as discussed in lecture – ***Your TA will check you are properly using the stack***

Part 2 – MIPS 32 programming (continued)

Problem 3 (20 marks) L33T (do this and you will have learned something cool)

So far, we have been assuming that a programmer has access to all resources attached to a computer. In MIPS 32, once your code is running, it can do anything it likes with the system's memory including writing over anything else that may be stored there.

This is not how it works on a modern computer. The Operating System carefully monitors what user programs are doing and makes sure all programs are playing nice (more on this when you get to C69).

For now, let's see what we can do without an Operating System to get in the way, shall we?

Task: Write the skeleton of a computer virus in assembly

A typical virus contains code to copy itself onto a binary executable file in such a way that when the executable is loaded, the first thing that runs is the virus Code. We don't have a file system with binary executable files to play with here, but we will simulate one. Open the ***'virus_starter.s'*** file.

Notice the following:

- Some text is defined already in the data section
- There is a nice little function called '***white_sheep***'. This represents the target binary file we want to infect with the virus. If you run the code now, it will print a nice message. ***You must not modify 'white_sheep'***
- There is also a '***virus_payload***' function which contains the virus' code. ***You must not modify it***
- The main function is your virus installer. You must add code here to have ***white_sheep*** launch your virus payload. This means somehow modifyinf ***white_sheep***'s code at runtime!
  Your code must ensure that:
    * ***virus_payload*** is executed once ***main*** reaches the Last instruction: '***b white_sheep***'
    * ***white_sheep*** runs as expected (showing its usual Message despite having been modified). We would not want anyone to notice we changed it!
- You can not call ***virus_payload*** or ***white_sheep*** from main, your code must modify ***white_sheep*** to accomplish the task

  Call your completed program '***virus_studentNo.s***'

Submitting your work

Step 1:

Create a ***single*** compressed archive in ***.zip format***. The archive should contain Your 3 assembly programs, named as described above.

Name your .zip file “***assignment3_studentNo.zip***”.

Make sure the .zip file can be decompressed and restores your files in their latest version. We will not accept remark requests later because you did not submit a working .zip, or because you zipped the wrong files.

Step 2:

Submit your ***.zip*** file (and only this file!) on mathlab:

Log into ***mathlab.utsc.utoronto.ca*** using your ***utorID*** and ***password***.

Submit your file using the command:

```
submit -c cscb58w15 -a A3 -f assignment3_studentNo.zip
```

You're done!

Test that you can submit files on mathlab well before the deadline. If there are any problems, we can look into them but it will take some time, so make sure you test in advance.

If all else fails: So, you didn't test in advance, you ran out of time, or you just can't upload your file to mathlab. Submit your code by email with the subject 'B58 A3' but note this will result in a 10 mark penalty for the assembly programming part.
`

Bonus: (5 marks)
Complete this section after you have solved the rest of your assignment - there are no right answers! - this is for feedback only, and will remain confidential.

Self evaluation: This section is meant to help you find topics that are giving you trouble. Focus your studying on these topics.

I will look at this also, to see what topics are giving most people trouble, we will devote more time to these topics during tutorials, and if necessary, use tutorials for reviewing that material

1.- After solving the assignment, how well do you understand ***pointers*** and how to use them to manipulate data in memory?
very well / well / it's somewhat confusing / not at all

2.- Are you confident you can write simple MIPS32 assembly programs with loops and conditional statements?
very confident / somewhat confident / not very / not at all

3.- How confident are you that you could write more complex MIPS32 programs with subroutines, dynamic memory allocation, and more complex flow?
very confident / somewhat confident / not very / not at all

4.- What concepts from this assignment would you like to have more tutorial time devoted to? Please be specific!

5.- Do you feel you have seen enough examples of assembly programming (including this assignment's problems) to give you a reasonably good grasp of how assembly programming works?
definitely / getting there, but could use a bit more / more examples please!

6.- Do you feel that learning about assembly programming helped you understand better how programs (in C or other languages) work?
Definitely / somewhat / not really / it was more confusing than helpful