Homework Assignment 3

CS/ECE 3810: Computer Organization
Sept 16, 2020

MIPS Assembly

Due Date: Sept 23, 2020.
(100 points)

Important Notes:

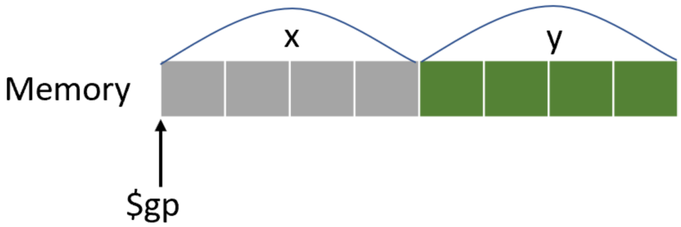
- Solutions turned in must be your own. Please, mention references (if any) at the end of each question. *Please refrain from cheating.*
- All solutions must be accompanied by the equations used/logic/intermediate steps. Writing only the final answer will receive **zero** credits.
- Partial score of every question is dedicated to each correct final answer provided by you. Please ensure both your equation/logic and final answer are correct. Moreover, you are expected to provide explanation for your solutions.
- All units must be mentioned wherever required.
- Late submissions **(after 11:59PM on 09/23/2020)** will not be accepted.
- We encourage all solutions to be typed in for which you could use software programs like LaTeX, Microsoft Word etc. If you submit handwritten solutions, they must be readable by the TAs to receive credits.
- All submitted solutions must be in the PDF format unless otherwise mentioned.

Set on less than (SLT). The SLT instruction performs a less-than comparison in MIPS. Here is an example.

$$slt\ \ \$s0,\ \ \$s3,\ \ \$s4$$

The above instruction sets $s0 to 1 if the value in $s3 is less than the value in $s4; otherwise, it sets $s0 to 0. [Refer to textbook chapter 2.7 page 93]

Question 1. Convert the following pseudo-code to MIPS assembly using the SLT instruction when necessary. Do not use MIPS pseudo instructions for this conversion. Below is the memory layout of variables x and y, each of them is of size 4 bytes. The grey-colored bytes represent variable x and green bytes represent variable y. The base address is stored in $gp. **(20 points)**



Code snippet

```
if (x ≥ 0) {
        $x = y$;
} else {
        $x = -y$ ;
}
```

```
lw $s0, 0($gp)     #Loads x
lw $s1, 4($gp)     #Loads y

        sge $s1, $s0, $zero
        bne $s1, 1, Else
        sw $s0, 4($gp)
        j End
Else:
        sub $s2, $zero, $s1
        sw $s0, 12($gp)
End:
```

Question 2. The aim of this question is to help students familiarize themselves with different MIPS instructions, and understand what each instruction does. Below is a set of MIPS assembly instructions to perform a specific task. Fill up the given table. Some rows are already filled. Use a similar format to fill up the rest of the rows. **(20 points)**

Instruction	**Opcode**	**Source Data Operands**	**Destination**	**Immediate Value**
lw $s0, 0($gp)	lw	Memory[$gp]	$s0	0
lw $s1, 4($gp)	lw	Memory[$gp+4]	$s1	4
lw $s2, 8($gp)	lw	Memory[$gp+8]	$s2	8
lw $s3, 12($gp)	lw	Memory[$gp+12]	$s3	12
addi $s5, $zero ,0	addi	$zero	$s5	0
LOOP: slt $s6, $s5, $s2	slt	$s5, $s2	$s6	1 if $s5 < $s2
beq $s6, $zero, **EXIT**	beq	$s6, $zero	EXIT	EXIT if $s6 = $zero
mul $s7, $s3, $s1	mul	$s3, $s1	$s7	$s3 * $s1
add $s4, $s0, $s7	add	$so, $s7	$s4	$s0 + $s7
div $s8, $s4, $s3	div	$s4, $s3	$s8	$s4 / $s3
bne $s8, $zero **INNER**	bne	$s8, $zero	INNER	INNER if $s8 != $zero
addi $s0, $s0, 5	addi	$s0, 5	$s0	$s0 + 5
INNER: addi $s0, $s0, 1	addi	$s0, 1	$s0	$s0 + 1
addi $s1, $s1, 1	addi	$s1, 1	$s1	$s1 + 1
addi $s5, $s5, 1	addi	$s5, 1	$s5	$s5 + 1
j **LOOP**	j	None	LOOP	None
EXIT:	None	None	None	None

Variable assignment. Below is a variable assignment in a high-level language.

$$x = 35;$$

Assume the memory location for x is at $gp+64. To translate the above code to assembly, we need to create a constant value 35, then store the constant value in the memory location of x. Below is the MIPS translation for this assignment.

```
addi $s0, $zero, 35      # put the constant value 35 in $s0
sw $s0, 64($gp)          # store the constant 35 at the memory location $gp+64, which is x.
```

Question 3. i) Convert the following pseudo-code to MIPS assembly. As shown above, for every assignment operation create constant, load temporary registers, and store the register value in the memory location of the variable.

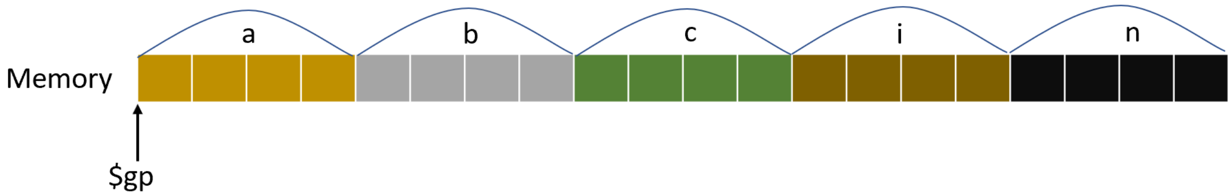
Code snippet

```
$a = 0;$
$b = 1;$
$c = 0;$
$n = 100;$
for $(i = 2; i < n; i = i + 1)$ {
        $c = a + b;$
        $a = b;$
        $b = c;$
}
```

```
addi $s0, $zero, 0     #Add a
sw $s0, 0($gp)
addi $s1, $zero, 1     #Add b
sw $s1, 4($gp)
addi $s2, $zero, 0     #Add c
sw $s2, 8($gp)
addi $s3, $zero, 2     #Add i
sw $s3, 12($gp)
addi $s4, $zero, 100     #Add n
sw $s4, 16($gp)

For: slt $t0, $s3, $s4
       beq $t0, $zero, Exit
       add $s2, $s2, $s0
       add $s2, $s2, $s1
       sw $s0, 4($gp)
       sw $s1, 8($gp)
       j For
Exit:
```

Below is the memory layout of the variables a, b, c, i, and n respectively. Each variable is of size 4 bytes and are stored sequentially in memory. Each color in the below diagram represents a variable. As shown below the base address is stored in global register \$gp. **(15 points)**



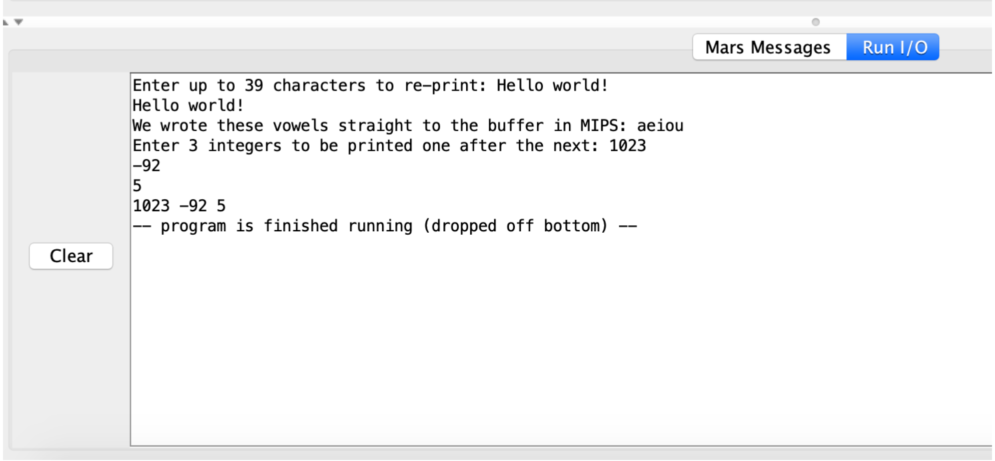
ii) Translate the above code to MIPS assembly without any load and store instructions assuming that variables a, b, c, i, and n are already stored in registers \$s0, \$s1, \$s2, \$s3, and \$s4 respectively. [Refer to Control Instructions Lecture slide 21] **(15 points)**

```
For: slt $t0, $s3, $s4
     beq $t0, $zero, Exit
     add $s2, $s2, $s0
     add $s2, $s2, $s1
     sw $s0, 4($gp)
     sw $s1, 8($gp)
     j For
Exit:
```

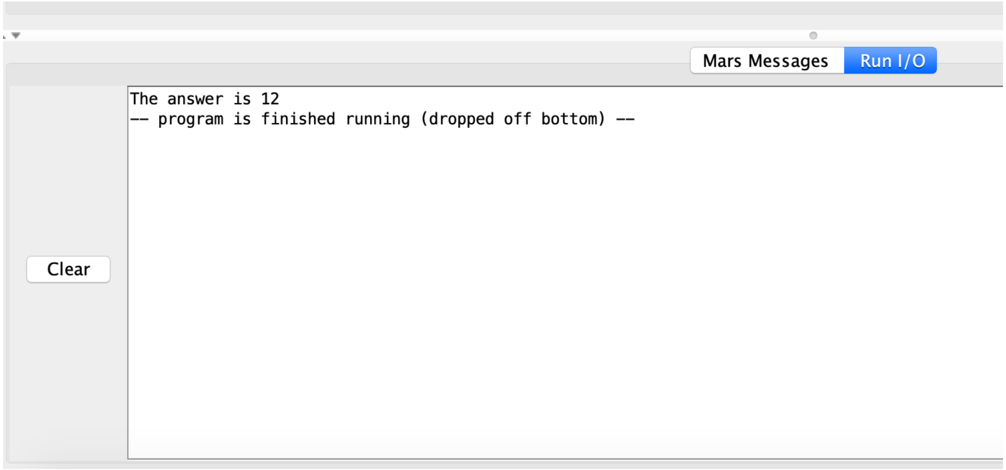
Question 4. The aim of this question is for you to get familiarized with the simulator. Future assignments would require you to execute all your assembly code in MARS and sending us the .asm files with comments.

Here is the document that provides overview of the MARS simulator: Click Here. Use this document to install MARS simulator in your computer. Implement (assemble and run) the examples shown in Section 6 and Section 7 of the given document. Attach the screenshots of the results shown in RunI/O. **(30 points)**

(NOTE: You are asked to copy paste the code snippets given as part of the doc, and run them in the simulator. The question does not expect you to add your own code to any of the code snippets)



```
Enter up to 39 characters to re-print: Hello world!
Hello world!
We wrote these vowels straight to the buffer in MIPS: aeiou
Enter 3 integers to be printed one after the next: 1023
-92
5
1023 -92 5
-- program is finished running (dropped off bottom) --
```



```
The answer is 12
-- program is finished running (dropped off bottom) --
```