COMP3350 Homework 02

Important Notes:

- Solutions turned in must be your own. Please, mention references (if any) at the end of each question.
- 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.
- 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.

1. Read the assembly code below; 1) add comments to explain what each line of code is doing; 2) in one sentence, explain what this procedure is trying to accomplish. **(5 points)**

```
new-proc:
sll $a0, $a0, 24
srl $a0, $a0, 24
add $v0, $a0, $zero
jr $ra
```

2. Read the assembly code below; 1) add comments to explain what each line of code is doing; 2) provide a simple equation to express the return value $v0 as a function of input arguments $a0 and $a1. **(6 points)**

```
new-proc:
blt $a1, $zero, loop2
loop1:
beq $a1, $zero, proc-end
sll $a0, $a0, 1
addi $a1, $a1, -1
j loop1
loop2:
beq $a1, $zero, proc-end
srl $a0, $a0, 1
addi $a1, $a1, 1
j loop2
proc-end:
add $v0, $a0, $zero
jr $ra
```

3. For the (**pseudo**) assembly code below, replace X, Y, P, and Q with the smallest set of instructions to save/restore values on the stack and update the stack pointer. Assume that procA and procB were written independently by two different programmers who are following the MIPS guidelines for caller-saved and callee-saved registers. They can't see the code written by the other person. Assume that $fp isn't being used by either procA or procB. Be sure to go through the slides first. **(8 points)**

```
procA:
        X                   # please replace X with your code
        $s0 = ...
        $s1 = ...
        $s2 = ...
        $s3 = ...
        $a0 = ...
        $a1 = ...
        jal procB
        ... = $s3           # (this is the value that was originally assigned in procA)
        ... = $a0 # (this is the value that was originally passed to procA as an argument)
        $v0 = $a0 + $v0
        Y                   # please replace Y with your code
        jr $ra

procB:
        P                   # please replace P with your code
        ... = $a0
        ... = $a1
        $s1 = ...
        $s3 = ...
        $t0 = ...
        $v0 = ...
        Q                   # please replace Q with your code
        jr $ra
```

4. In the MARS simulator, create a new file and copy-paste the code below into the Edit window. Execute the program. The program should print "BOO7" at the bottom. Take a screenshot of your MARS screen. Annotate this image to show that **one of the registers and one of the memory locations has the integer 7**, and **three registers and three memory locations have the ASCII codes for the three characters "B", "O", and "O"**. Your annotation can be as simple as circles with the corresponding labels (7, B, O, O). **(5 points)**

```
.data

str: .asciiz "BOO"
myint: .word 7

.text

li $v0, 4       # load immediate; 4 is the code for print_string
la $a0, str     # the print_string syscall expects the string
# address as the argument; la is the instruction
# to load the address of the operand (str)
syscall         # MARS will now invoke syscall4.
lb $t1, ($a0)  # load the byte at address $a0 into register $t1
lb $t2, 1($a0) # load the byte at address $a0+1 into register $t2
lb $t3, 2($a0) # load the byte at address $a0+2 into register $t3
li $v0, 1       # syscall-1 corresponds to print_int
lw $a0, myint # Bring the value at label myint to register $a0.
syscall # MARS will now invoke syscall1.
```

5. Write the MIPS assembly code that corresponds to the pseudo code below. Assume that the base address is stored in $gp. Assume that the memory address for integer i is (base address + 4) and the address for a[0] is (base address + 8). To make your code efficient, i must be loaded into a register at the start, and it must be updated in memory only after you've finished the for loop.

```
for (i=19; i>=0; i-=2)
  a[i] = 8*i;
```

You may not use a multiply instruction. (**5 points**)