Quiz 2
CSE 112 Computer Organization

Dt: 12/07/2022 **Total Marks = 60** **Time Duration = 80 minutes**

INSTRUCTIONS:

1. **The duration of the quiz is 80 mins. No further extension of time will be given regarding this.**
2. **Write any assumption clearly, if any. *Only reasonable assumptions will be considered if any ambiguity is found in the question*.**
3. **Do not use question paper for rough work, please show rough work (*if any*) in the answer sheet itself.**
4. **Calculators are NOT allowed during exam time. ONLY use pen and paper for writing the exam.**

GOOD LUCK !!

For All the questions, please use the following ISA. You are not allowed to use any instructions which are not part of the ISA, unless specified otherwise.
The instructions supported by the ISA are mentioned in the table below. The ISA has 16 General purpose registers: r0 to r15

Name	Semantics	Syntax
Add	Performs reg1 = reg2 + reg3	**add reg1 reg2 reg3**
Sub	Performs reg1 = reg2 - reg3	**sub reg1 reg2 reg3**
Mov imm	Performs reg1 = Imm	**mov reg1 $Imm**
Mov	Performs reg1 = reg2	**mov reg1 reg2**
Branch to label if Equal to immediate value	Branch to label if reg1 == imm	**beq reg1 $imm label**
Branch to label if Equal to register value	Branch to label if reg1 == reg2	**beq reg1 reg2 label**
Branch to address if Equal	Branch to address stored in **reg3** if reg1 == reg2	**beq reg1 reg2 reg3**
Branch unconditional	Branch to label unconditionally	**b label**
Increment by 1	Performs reg1 = reg1 + 1	**inc reg1**
Decrement by 1	Performs reg1 = reg1 – 1	**dec reg1**

Apart from the above instructions, the following subroutines are also valid and can be used in assembly code:

Name	Semantics	Syntax
Input	Reads immediate data from the user into register rg	**in rg**
Output	Prints str on the console	**out "str"**

Note: *For Questions,* ***Q1*** *and* ***Q2****, assume the initial value of all the registers r0 to r15 is equal to 0 unless specified otherwise*.

Q1.**[CO2]** Write a simple function in Assembly language named "***mid_fun***" to divide an integer number(greater than or equal to 0) by "**2**" using the ISA described above. **[10 Marks]**

There are no separate marks for Pseudo-Code. Please add relevant comments in the assembly code for a better understanding for the reader.

The following conditions should be met from your assembly code:

1. When the function finishes with the calculation of the result, it should return to the **callee** automatically, whose address is stored in register "**r7**".
2. The number to be divided by two(2) is assumed to be stored in register "**r15**".
3. The final output(**quotient**) should be stored in register "**r14**" (assuming **r14** was initially **0**)
4. The division should output the **nondecimal** part of the quotient **ONLY**, an example is:
   ⇒ If register "**r15**" contains 7 (**r15** = 7), then, on dividing by 2, the correct answer according to hand-calculation is 3.5.
   ⇒ But, the register "**r14**" should ultimately store only "3" as the final answer(without the double quotes: " "), thus ignoring the trailing decimal digits.

Note: *There is no need to store or report or write extra code for the remainder of the division. Quotient without the decimal part is stored in register* ***r14*** *and is accepted as the* ***final result*** *of the division.*

Q2.**[CO2]** Assume that the assembly function is defined in **Q1.** as "***mid_fun***" is the same as being called in the code below. Study the following assembly code and, **[4+16 = 20 Marks]**

A. Write a pseudo code in any language/syntax for the given assembly code. **[4 Marks]**
B. Write down the output of the following code if the **user input is = 5** when the code is executed. **[16 Marks]**

Note*: Assume that the result after division by 2 (in the called function,* ***mid_fun****) is stored in the register* ***r14,*** *and as soon as* ***mid_fun*** *finishes its execution, the PC returns to the* ***line number 3*** *of the following assembly code.*

Assembly Code is:

```
1.  in r15                          //input from the user
2.  b mid_fun
3.  mov r1 $0                       //iterator for loop1
4.  mov r13 $1
5.  loop1:
6.    beq r1 r14 loop1_exit         //end value for loop1 in r14
7.    mov r2 $0                     //iterator for loop2
8.    add r11 r1 r13                //end value for loop2
9.      loop2:
10.       beq r2 r11 loop2_exit
11.       out "*"
12.       inc r2                    //loop2 iterator increment by 1
13.       b loop2                   //unconditional jump
14.     loop2_exit:
15.   out "\n"                      //print new-line
16.   inc r1                        //loop1 iterator increment by 1
17.   b loop1                       //unconditional jump
18. loop1_exit:
19. mov r5 $0
20. mov r3 r1                       //iterator for loop3
21. loop3:
22.   beq r3 r5 loop3_exit
23.   mov r4 $0                     //iterator for loop4
24.   add r11 r3 r13
25.     loop4:
26.       beq r4 r11 loop4_exit
27.       out "*"
28.       inc r4                    //loop4 iterator increment by 1
29.       b loop4                   //unconditional jump
30.     loop4_exit:
31.   out "\n"
32.   dec r3                        //loop3 iterator decrement by 1
33.   b loop3                       //unconditional jump
34. loop3_exit
```

Q3.**[CO3]** Consider a hypothetical three-stage pipeline - Fetch (F), Decode (D), and Execute (E). In the fetch stage, the instructions are fetched from the instruction memory. In the decode stage, the fetched instructions are decoded and the values of the input registers are read. In the execution stage, ALU performs the computation on values read from the decode stage and updates the value of the output register.

For the instructions given below, write the pipeline diagram. Assume all register values to be zero. **[3x5 = 15 marks]**

```
1. mov  r0, r2
2. mov  r1, r3
3. add  r4, r8,r9
4. add  r5, r2, r3
5. beq  r4, r5, r7
```

Note: *Pipeline Diagram is used to tell in what stage the instructions are n. An example of a pipeline diagram is given below.* ***No marks shall be given if the answer is not in this form.***

add r0, r2, r3	F	D	E		
add r1, r2, r3		F	D	E	
beq r2,r2, r7			F	D	E

Q4.**[CO2]** Consider a stack that starts from memory location 0x400. In addition to the instructions defined in **Q1**, consider the two instructions below for stack operations.

Semantics	Syntax
Pushes the value in register reg1 into stack	PUSH reg1
Pops the last value in the stack into register reg1	POP reg1

Whenever a value is put into the stack via PUSH instruction, the value of Stack Pointer (SP) decrements by one. Whenever a value is moved out of the stack via POP instruction, the value of SP increments by one. For the given program, determine the value of SP for each instruction.

[1.5x10 = 15 marks]

1. PUSH r1
2. PUSH r2
3. POP r1
4. PUSH r1
5. PUSH r5
6. PUSH r4
7. POP r2
8. PUSH r3
9. POP r5
10. PUSH r7