Quiz 2

CSE 112 Computer Organization

INSTRUCTIONS:

Total Marks = 40

Time Duration = 45mins(solving) + 10mins(uploading)

1. **Duration of the quiz is 45 mins, and 10 mins for scanning and uploading the solutions. No further extension of time will be given regarding this.**
2. **Question paper will be uploaded in the google classroom. Do not forget to turn in. Solutions submitted by any other means (email etc.) won't be considered for evaluation.**
3. **Students are required to switch on their cameras and mute themselves. Make sure you are sitting in a well lit room so that we are able to see your faces clearly. Please keep in mind that we'll be keeping a note of this and any violation can lead to some strict action against you.**
4. **The answers should be in your own handwriting and submission should be in PDF format only.**
5. **Write any assumption clearly, if any. Needless to say, only reasonable assumptions will be considered if any ambiguity is found in the question.**
6. **During the exam if you have any query, write it in the meet chat box. It will be taken into notice by us. Don't unnecessarily unmute your mic for it as it creates disturbance to others.**
7. **Calculators are NOT allowed during the exam time. ONLY use pen and paper for writing the exam.**

GOOD LUCK !!

ISA description for Q1 and Q2:

For All the questions, please use the given ISA. You are not allowed to use any instructions which are not the part of the ISA

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 if equal	Branch to addr if reg1 = imm	`beq reg #imm addr`
Branch and link	Jumps to label after saving the return address to r1.	`brl label`

Apart from the above instructions, the assembler and the operating system support the following subroutines:

Name	Semantics	Syntax
Input	Reads immediate data from user into reg	`in reg`
Output	Prints str on the console	`out “str”`

Q1: Write an assembly program to print the following pattern for **n** lines, where **n** is read as input from the user. Note that performing **`out “\n”`** moves the cursor to the next line. Assume that **n** will always be greater than or equal to 1.
Also write the pseudocode of your assembly program. (You may use your preferred language/syntax for the pseudocode) **[20 marks]**

```
*
* *
* * *
* * * *
* * * * *
```

(Example:- Pattern for n = 5 is given above)

ISA Extension for Q2

In the ISA mentioned in Q1, we add two more instructions along with the below caller callee convention. **[Note that these instructions are in addition to the the above instructions mentioned in ISA in Q1]**

Name	**Semantics**	**Syntax**
Push	Pushes the data stored in reg1 onto the stack	`push reg1`
Pop	Pops the data stored on the top of the stacks into reg1	`pop reg1`

Caller-callee conventions:

The following are the caller callee convention:

- There are 15 registers r0 to r15.
- r15 - program counter.
- r0 - stack pointer.
- r1 - link register and return address
- r2 - return value.
- r3 and r4 holds the first and second argument to the callee
- The stack is automatically managed by push and pop.
- All the registers from r1-r7 are caller saved. On the other hand, registers r8-r14 are callee saved.
- Whenever the branch and link instruction is used, the return address is stored in r1 and the program counter jumps to the given label.

Q2: The assembly for a high-level program is given below. Some of the instructions in the assembly are partially filled. Fill in the blanks. Some hints are given to you as comments in the assembly code.
Note that the answer for a blank can be a register, a label, an instruction mnemonic, or a complete instruction.

High-level code:

```
int my_add(int x, int y)
{
      int temp = x + y;
      return temp;
}

int foo ()
{
      int a;
      int b;
      int c;      // return value

      a = 10;
      b = 20;
      c = my_add(a, b);
      return 2*c;
}

int main()
{
      return foo();
}
```

Fill in the blanks below from (A) to (J) using the above high level assembly code and caller callee conventions. [10 X 2 = 20 marks]

```
my_add:    push (A)___      // save a caller saved register
           add r2 r3 r4
           pop r1
           (B)___ r15 r1     // return statement

foo:       mov r3 (C)___    // move an immediate value
           mov r4 #20
           push (D)___       // save a caller saved register
           (E)___ my_add    // call my_add
           pop r1
           add r2 (F)___ r2 // store the result
           mov r15 (G)___   // return statement

main:      push (H)___      //save return address of main func
           brl foo
           (I)___ r1        //restore return address of main func
           (J)___           // return statement
```