Computer Organisation (CSE/ECE-112), 2024
END-SEM Questions and Rubric

ISA to be used for this question paper:

- This ISA supports 16 registers (R0-R15).

(a) Basic instructions:

S. No.	Instruction (Mnemonic, Operands)	Operation Performed
1.	MUL R1, R2, R3	Multiply the contents of registers R2 and R3, and store the result in the specified register, R1.
2.	SUBI R1, #Imm	Subtract the content of the specified register(R1) by immediate value (imm).
3.	BEQ R1, R2, Address/Label	Branch to address/label if the content of specified register (R1) is equal to content of register R2
4.	ADD R1, R2, R3	Add contents of registers R2 and R3 and store the result in R1.
5.	MVI R1, #Imm	Copy the Immediate value into the specified register (R1).
6.	ST R1, Address/Label	Store the content of register R1 to the specified address/label.
7.	MOV R1, R2	Copy the content of the register R2 into the register R1.
8.	LD R1, Address/Label	Load the content specified at the mentioned address/label into the register (R1).
9.	BL Address	Branch to the mentioned address and store the return address (address of next instruction) in the Link register.
10.	B Address/Label	Unconditional branch to specified address/label

(b) Some additional instructions/subroutines:

S. No.	Subroutine (Mnemonic, Operand)	Operation Performed
1.	POP R1	Pops the data stored on the top of the stack into register R1
2.	PUSH R1	Pushes the data stored in register R1 onto the stack
3.	IN R1	Reads immediate data from the user into register R1
4.	OUT “**str**”	Prints string “**str**” on the console

Q1. [8+12 = 20 Marks]

For this question, register "**R7**" is link register and "**R10**" stores the program counter (PC) value.

(a) Using the ISA provided, write a function called "**factorial**" in assembly code that takes a number as an input from the user and calculates its factorial value. Final result should be stored in the register "**R3**". You have to ensure that as soon as the calculation of factorial value is finished, the code automatically returns to the callee function.

(b) Write assembly code for the "**main**" function which calls the function "**factorial**" defined by you **part (a)** to use the calculated value of factorial to successively print the string " * " on the console in the following manner, example shown for calculated factorial value = 6:

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

⇒ Note that printing " **\n** " moves the cursor to the next line.

Solution: (line numbers are added just for visual clarity, and do not convey any other information)

Note to TAs: For both parts, students may have written different solutions (using stack, etc.). But if the answers are **logically correct**, and in line with the **conditions** specified in the **Question**, then you can treat them as the correct answer.

(a) Code (multiple answers possible, mark according to the conditions specified):

```
1.  FACTORIAL:
2.     MVI R3, #1
3.     MVI R0, #0
4.     IN  R1
5.  LOOP:
6.     MUL R3, R1, R3
7.     SUBI R1, #1
8.     BEQ R0, R1, RETURN
9.     B LOOP
10. RETURN:
11a.   B FACTORIAL // Returning to callee
// (Alternative code possible for line #11, if part (b) is also taken into account)
11b.   MOV  R10,  R7  //  updating  PC  to  move  back  to  caller's  next  instruction's  address  in  link
register
```

- Conditions and marks for partial marking: **[8 Marks]**
  1. **1 Mark: Final result in R3?**
  2. **1 Mark: Input taken from user?**
  3. **0.5 Marks: Multiplication done correctly?**
  4. **1 Mark: Branching used correctly?**
  5. **0.5 Marks: Iteration done correctly?**
  6. **1 Mark: Either the function branches to the callee (like line #11a) or updates link register (alternative line #11b).**
  7. **3 Marks: If R1 = 4, then check if R3 = 24 at the end of execution or not?**

(b) [1.5+10.5 = 12 Marks]

- Depending on how part (a) is attempted, some changes might be required in the previous code to suit the caller-callee convention. It has to be awarded **1.5 Marks** in either of the following **two cases**:
  - **If the student has used the alternative code similar to line #11b in part (a):**
    No need to mention any change required in part (a).
  - **If the student has used the code similar to line #11a in part (a):**
    Need to mention in part (b) that a change is required to update the PC to return to the "main" function after "factorial" has finished its calculation. At the end of the factorial calculation, PC (R10) should be updated with the link register (R7) value, using the following code:

```
11.   MOV R10, R7 // updating PC to move back to caller's (main) next instruction's address in link register
```

- **Code for Main Function (multiple answers possible, mark according to the conditions specified):**

```
0.    MAIN:
1.          BL FACTORIAL        // RESULT (COUNT) IN R3
2.          MVI R5, #0
3.    CHANGE:
4.          ADD R4, R3, R5      // Making R4 = R5. Inner loop iterator, i.e., i = R4
5.    LINE1:
6.          OUT " * "           // Prints *
7.          SUBI R4, #1         // i = i - 1
8.          BEQ R4, R5, LINE2   // To check if its end of line (EOL).
9.          B LINE1             // unconditional branch to print " * " if not EOL.
10.   LINE2:
11.         OUT " \n "          // If EOL, then go to the next line.
12.         SUBI R3, #1         // second iterator j = j - 1
13.         BEQ R3, R5, END     // if j == 0, then end the code
14.         B CHANGE            // unconditional branch if j != 0 to start printing next line
15.   END
```

  - Conditions and marks for partial marking for the above code: **[10.5 Marks]**
    1. **1 Mark: BL is used to call the function "factorial".**
    2. **1.5 Marks: Correct use of R3 to decide the number of iterations?**
    3. **1 Mark: OUT " * " done at the correct place?**
    4. **1 Mark: Used " \n " correctly?**
    5. **3 Marks: Use of two loops or two iterations; one for printing " * " in the same line and the other to decide the number of lines to be printed and change lines?**
    6. **3 Marks: Correct pattern printed at the end for any calculated factorial value?**

Q2. [12+3 = 15 Marks]

Consider a **system A with a cache** and another **system B without a cache**. Both systems have main memory. Details regarding both systems are provided in the table below.

Specifications	System A	System B
Clock	100 MHz	200MHz
Cache Present (access time)	Yes (20ns)	No
Main memory access time	80 ns	80ns

Executing each instruction on System A takes 5 cycles in addition to memory access time. For System B, each instruction takes 2 cycles in addition to its memory access time. None of the systems have pipeline.

(a) An assembly code snippet for a certain application is provided below. The code is executed on systems A and B. For System A, all the memory reads and writes are limited to the cache memory. Calculate the total time it takes for systems A and B to execute this application. Provide the time for both systems in the units of the number of cycles and in nanoseconds (ns).
***Note**: Line numbers are added just for visual clarity and do not convey any other information.*

```
1. ADD R1, R2, R3
2. SUBI R3, #1
3. LD R5, memaddr1
4. SUBI R8, #5
5. ST R8, memaddr2
```

(b) Consider an image processing application with significant temporal locality (requiring frequent reuse of data). Which System (A or B) would you choose to execute such an application, and why?

Solution:

(a) **System A:** **[6 Marks]**
Period = 1/100MHz = 10ns
Inst.1: 5 cycles (50ns) **[1 Mark]**
Inst.2: 5 cycles (50ns) **[1 Mark]**
Inst.3: 7 cycles (70ns=50ns+20ns) (since cache will shadow the high access time of main memory) **[1.5 Marks]**
Inst.4: 5 cycles (50ns) **[1 Marks]**
Inst.5: 7 cycles (70ns=50ns+20ns) (since cache will shadow the high access time of main memory) **[1.5 Marks]**
Total: 29 cycles (290ns)
System B: **[6 Marks]**
Period = 1/200MHz = 5ns
Inst.1: 2 cycles (10ns) **[1 Mark]**
Inst.2: 2 cycles (10ns) **[1 Mark]**
Inst.3: 18 cycles (90ns = 10ns + 80ns) (High access time for main memory) **[1.5 Marks]**
Inst.4: 2 cycles (10ns) **[1 Mark]**
Inst.5: 18 cycles (90ns = 10ns + 80ns) (High access time for main memory) **[1.5 Marks]**
Total: 42 cycles (210ns)

(b) ***Note:*** *The keywords are marked in bold.*
We would choose **System A** because, even though its frequency is low, high temporal locality applications will benefit with the **presence of a cache**. By storing frequently accessed data closer to the processor, the cache will significantly reduce **the average memory access time** in the **long run** for the frequently required data and can outweigh any performance limitations due to the **low frequency of operation as compared to System B with double the frequency**. **[3 marks]**
[1.5 marks for writing the presence of cache]
[1.5 marks for comparing the low frequency of System A with System B and/or for mentioning "average" memory access time]

Q3. [15+5 = 20 Marks]

Consider the following program with five pipeline stages: Fetch(F), Decode(D), Execute (E), Memory (M), and Writeback (W). Each pipeline stage is of one clock cycle each. The stages are described below:

1. Fetch - Fetches the instructions from instruction memory
2. Decode - Decodes the instructions and type of operations. It also reads the input operands. If the input operands are not present, the decode stage is stalled for one cycle to read the operands after they are made available.
3. Execute - Executes the instruction in the ALU
4. Memory- For load operation, takes the data from memory and copies it to the register.
5. Writeback - Updates the register value at the end of the stage.

```
1.  LD R1,12
2.  LD R2,16
3.  ADD R11,R2,R3
4.  LD R1,4
5.  ADD R2,R3,R4
6.  MUL R5,R1,R5
7.  LD R6,8
8.  LD R7,9
9.  LD R9,9
10. SUBI R6,#3
11. ADD R10,R7,R6
12. ADD R11,R1,R2
```

(a) Draw the pipeline diagram and determine the number of clock cycles needed to execute the above program.
(b) We now add a direct mapped data cache with 8 locations. The structure with locations of the data cache is given below. When the load instruction is executed, the data copied to the register is also copied to the data cache. For the above program, determine the memory addresses present in the data cache after each load instruction. Indicate replacements by striking out the old values and entering new ones. (**Hint**: In a direct mapped cache, each main memory address maps to exactly one unique cache location.)

Solution:

(a)

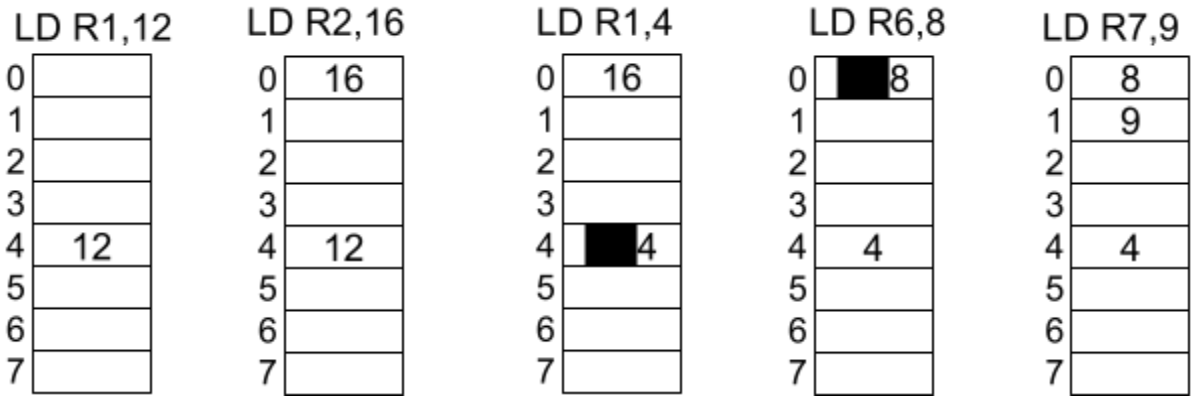
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
LD R1,12	F	D	E	M	W																				
LD R2,16		F	D	E	M	W																			
ADD R11,R2,R3			F	D	D	D	D	E	M	W															
LD R1,4				F	F	F	F	D	E	M	W														
ADD R2,R3,R4								F	D	E	M	W													
MUL R5,R1,R5									F	D	D	D	E	M	W										
LD R6,8										F	F	F	D	E	M	W									
LD R7,9													F	D	E	M	W								
LD R9,9														F	D	E	M	W							
SUBI R6,#3															F	D	D	E	M	W					
ADD R10,R7,R6																F	F	D	D	D	D	E	M	W	
ADD R11,R1,R2																		F	F	F	F	D	E	M	W

Alternate Answer (Only if written “attempted before announcement”)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
LD R1,12	F	D	E	M	W																	
LD R2,16		F	D	E	M	W																
ADD R11,R2,R3			F	D	D	D	E	M	W													
LD R1,4				F	F	F	D	E	M	W												
ADD R2,R3,R4							F	D	E	M	W											
MUL R5,R1,R5								F	D	D	E	M	W									
LD R6,8									F	F	D	E	M	W								
LD R7,9											F	D	E	M	W							
LD R9,9												F	D	E	M	W						
SUBI R6,#3													F	D	E	M	W					
ADD R10,R7,R6														F	D	D	D	D	E	M	W	
ADD R11,R1,R2															F	F	F	F	D	E	M	W

[0.5+0.5+2+1+1+2+1+1+1+2+2+1 = 15 Marks]

(b) For a direct mapped cache, the location of data is given by: (Address % No of locations) = (Address %8)



[1*5 = 5 Marks]

In R9,9, there is a cache hit and hence no changes occur in cache.

Q4. [10+10+5 = 25 Marks]

Consider a stack that has the initial value of stack pointer (SP) equal to the memory location **0x45** (hexadecimal). Whenever a value is put into the stack via **PUSH** instruction, the value of the Stack Pointer (SP) **decrements** by one (binary). Whenever a value is moved out of the stack via **POP** instruction, the value of SP **increments** by one (binary). Study the code given below and answer the questions that follow (line numbers added just for visual clarity only):

```
1.  PUSH R2
2.  POP R3
3.  PUSH R5
4.  PUSH R6
5.  PUSH R7
6.  PUSH R8
7.  POP R9
8.  PUSH R1
9.  PUSH R11
10. POP R2
11. POP R3
12. PUSH R14
13. POP R4
14. POP R9
15. PUSH R4
16. POP R3
17. POP R4
18. PUSH R9
19. POP R6
20. POP R1
```

(a) For the given code, determine the value of updated SP after every instruction is executed.

(b) For the same code, write the updated value of the register modified after every 'POP' instruction, with initial values of R1=1, R2=2, R3=3, R4=4, R5=5, R6=6, R7=7, R8=8, R9=9, R10=10, R11=11 and R14=14.

(c) Also report just the value pushed in the stack after every PUSH instruction.

***Note**: You may answer all the above parts (a), (b), and (c), using a common table as shown:*

Instructions	PUSHed Value	Register and its value after 'POP'	Updated SP Value

Solution:

PUSH (1st Column) : 0.5 Marks * 10 = 5 Marks
POP (2nd Column) : 1 Mark * 10 = 10 Marks
SP (3rd Column) : 0.5 Marks * 20 = 10 Marks

Instructions	Stack Status (optional, no marks)	PUSHed Value	Register and its value after 'POP'	Updated SP Value
1. PUSH R2	2	2	-	0x44
2. POP R3	-	-	R3=2	0x45
3. PUSH R5	5	5	-	0x44
4. PUSH R6	5,6	6	-	0x43
5. PUSH R7	5,6,7	7	-	0x42
6. PUSH R8	5,6,7,8	8	-	0x41
7. POP R9	5,6,7	-	R9=8	0x42
8. PUSH R1	5,6,7,1	1	-	0x41
9. PUSH R11	5,6,7,1,11	11	-	0x40
10. POP R2	5,6,7,1	-	R2=11	0x41
11. POP R3	5,6,7	-	R3=1	0x42
12. PUSH R14	5,6,7,14	14	-	0x41
13. POP R4	5,6,7	-	R4=14	0x42
14. POP R9	5,6	-	R9=7	0x43
15. PUSH R4	5,6,14	14	-	0x42
16. POP R3	5,6	-	R3=14	0x43
17. POP R4	5	-	R4=6	0x44
18. PUSH R9	5,7	7	-	0x43
19. POP R6	5	-	R6=7	0x44
20. POP R1	-	-	R1=5	0x45