Practicum No.2 **Date:……….**

Develop solutions that perform logic operations based on ALU Status Flags

I. Practical Significance

Modern processors use ALU status flags to summarize the outcome of arithmetic and logical operations. Instead of re-evaluating numeric results, CPUs apply logic operations on these flags to decide whether a result is valid, erroneous, safe for further use, or requires special handling. This experiment helps students understand how logic operations on status flags drive processor decision-making, which is a fundamental concept in computer organization, embedded systems, and control-unit design.

II. Competency and Industry-Specific Practical Skills

Use the principles of computer architecture to optimize system performance

a) ***Construct*** logical expressions using Boolean operators. (PDO)
b) ***Implement*** logic-based solutions using MATLAB logical operators. (PDO)
c) ***Interpret*** logical outputs derived from processor status data. (CDO)

III. Relevant CO(s) *(As stated in the Course Structure)*

- **CO1.** ***Use*** the relevant instruction cycle to efficiently manage the processor's functionality.

IV. Practical Outcome (PrO) *(As stated in the Course Structure)*

- ***Develop*** solutions that perform logical operations for ***the given*** ALU status flags.

V. Related ADO(s) *(As stated in the Course Structure)*

a) ***Follow*** ethical practices in designing or evaluating the system design.
b) ***Apply*** systematic design principles and debugging techniques.
c) ***Function*** effectively as a team member.

VI. Minimum Underpinning Theory *(Include relevant content & images for this practicum.)*

Each ALU operation updates a set of status flags such as Carry Flag (CF), Overflow Flag (OF), Sign Flag (SF), and Parity Flag (PF). These flags provide compact information about the operation result. The CPU applies logic operations on these flags using basic gates such as AND, NOT, and NAND to make decisions related to result validity, error detection, and signed interpretation. This hardware-level decision approach ensures fast execution and minimal overhead in processor control logic. Such logic-based interpretation is widely used in processor control units and exception handling.

Flag	**0**	**1**
SF (Sign Flag)	Positive result	Negative result
OF (Overflow Flag)	No signed overflow	Signed overflow
CF (Carry Flag)	No carry generated	Carry generated
PF (Parity Flag)	Odd parity	Even parity

VII. Practicum Set-up/Circuit Diagram/Algorithm/Flowchart *(Include probable photos of experimental set-up.)*

Algorithm

1. Perform an arithmetic operation using signed 8-bit data.
2. Record the ALU status flags (CF, OF, SF, PF).
3. Apply logic expressions for the given Scenario.
4. Observe logic output and classify the result.
5. Validate decisions using multiple test cases.

VIII. Resources Required

S. No.	Resource	Suggested Broad Specification	Quantity
1	MATLAB	(R2016b or newer) or GNU Octave (with mostly compatible syntax).	1 No.
2	PC	PC with MATLAB installed.	1 No.
3	Text editor / MATLAB Editor	-	1 No.

IX. Safety Precautions

a) ***Save*** work frequently.
b) ***Comment*** code and include header describing author, date, input format and usage.
c) For shared code, ***remove*** hard-coded absolute paths.
d) ***Validate*** user inputs to avoid invalid logical values and runtime errors.

X. Procedure *(Self-Instructional)*

1. ***Select*** signed input values.
2. ***Perform*** arithmetic and note status flags.
3. ***Implement*** the Scenario using gates.
4. ***Record*** logic output and CPU decision.

Scenario: Safe and Clean ALU Output using AND (**AND gate output should be 1 – for safe and clean**)
Condition after arithmetic operation:
CF = 0, OF = 0, PF = 1
Logic Expression:
SAFE_OUTPUT = $\overline{\mathrm{CF}}$ AND $\overline{\mathrm{OF}}$ AND PF
CPU Interpretation:
No carry, no signed overflow, and even parity (to avoid error) — the CPU treats the result as safe and clean.

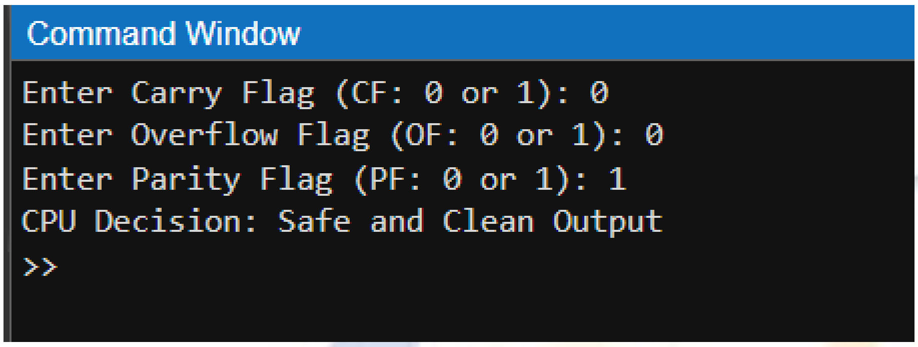
% Scenario: Safe and Clean Output using **AND and NOT gates**

```
clc; clear;

% Input ALU flags (0 or 1)
CF = input('Enter Carry Flag (CF: 0 or 1): ');
OF = input('Enter Overflow Flag (OF: 0 or 1): ');
PF = input('Enter Parity Flag (PF: 0 or 1): ');

% NOT operations
CF_bar = ~CF;
OF_bar = ~OF;

% AND operation
SAFE_OUTPUT = CF_bar & OF_bar & PF;

% Display result
if SAFE_OUTPUT == 1
   disp('CPU Decision: Safe and Clean Output');
else
```

```
    disp('CPU Decision: Output is NOT Safe');
end
```

Sample Input & Sample Output:

OUTPUT:



```
Command Window
Enter Carry Flag (CF: 0 or 1): 0
Enter Overflow Flag (OF: 0 or 1): 0
Enter Parity Flag (PF: 0 or 1): 1
CPU Decision: Safe and Clean Output
>>
```

XI. Actual Resources Used (*To be filled by the students*)

S. No.	Name of Resource	Broad Specifications		Quantity	Remarks (If any)
		Make	Details		
1.	MATLAB Software	MathWorks	MATLAB R2025b	1	Used for logical operations on ALU flags
2.	Personal Computer (PC)	HP / Dell / Lenovo	Intel i3 / i5 Processor, 8 GB RAM, Windows OS	1	Laboratory system
3.	MATLAB Editor	MATLAB Integrated	Built-in MATLAB Script Editor	1	Used for writing and executing code

XII. Actual Procedure Followed (*To be filled by the students*)

1. The system was powered on and MATLAB software was launched on the laboratory PC.
2. A new MATLAB script file was created and appropriately named for the experiment on ALU status flag logic operations.
3. Signed input values were selected conceptually, and the corresponding **ALU status flags** (Carry Flag – CF, Overflow Flag – OF, Sign Flag – SF, and Parity Flag – PF) were identified for each arithmetic operation.
4. The required logic scenarios were analyzed based on the given conditions:
    a. Safe and Clean Output using AND and NOT gates
    b. Valid Negative Result Detection using NAND gate
    c. Abnormal Result Detection using OR gate
    d. Clean Result Detection using NOR gate
5. MATLAB code was written to accept ALU status flags as user inputs (0 or 1) and to implement the respective logic expressions using logical operators (~, &, |).
6. The program was executed for different combinations of flag values, and the logic output was observed for each test case.
7. Based on the logic output (0 or 1), the CPU decision such as **Safe Output**, **Valid Negative Result**, **Abnormal Result**, or **Clean Arithmetic Result** was displayed.
8. Multiple test cases were executed to verify the correctness of the logic expressions under different arithmetic conditions.

9. The observed outputs and corresponding CPU decisions were recorded systematically in the observation table.
10. The results were analyzed to confirm that logical operations on ALU status flags correctly determine the validity and condition of arithmetic results

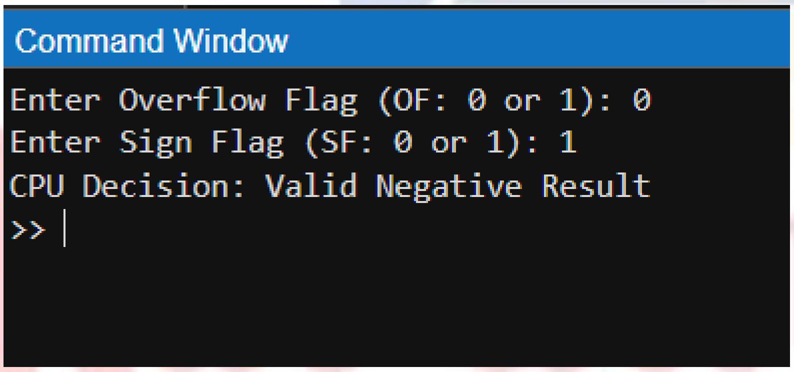
XIII. Additional Questions

1. Implement a Valid Negative Result Detection using **NAND** Gate (**NAND output = 0 – Valid**) - VALID_NEGATIVE = NAND ($\overline{OF}$, SF). **Note:** SF = 1 → Result is negative, SF = 0 → Result is positive.

PROGRAM:

```
clc; clear;

% Input ALU flags
OF = input('Enter Overflow Flag (OF: 0 or 1): ');
SF = input('Enter Sign Flag (SF: 0 or 1): ');

% NOT operation on Overflow Flag
OF_bar = ~OF;

% NAND operation
VALID_NEGATIVE = ~(OF_bar & SF);

% CPU Decision
if VALID_NEGATIVE == 0
   disp('CPU Decision: Valid Negative Result');
else
   disp('CPU Decision: Result is NOT a Valid Negative');
end
```

OUTPUT:



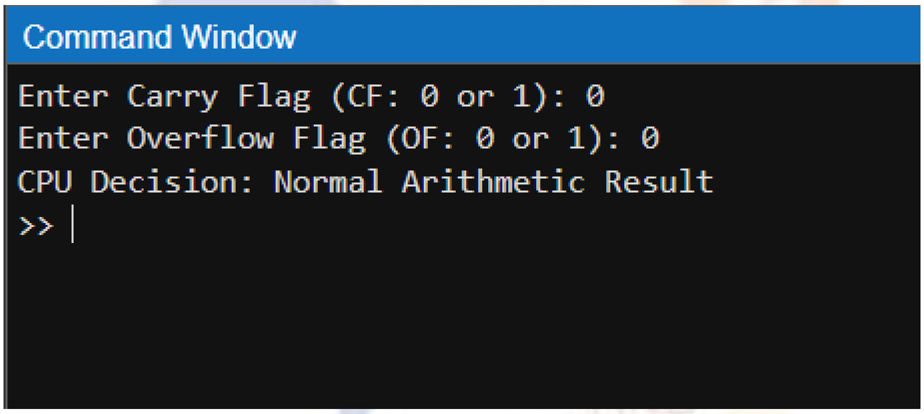
```
Enter Overflow Flag (OF: 0 or 1): 0
Enter Sign Flag (SF: 0 or 1): 1
CPU Decision: Valid Negative Result
>> 
```

2. Implement an Abnormal Result Detection using **OR** Gate (O**R output = 1**) - ABNORMAL = CF OR OF. **Note**: Only when both CF and OF are 0 is the result normal i.e. 0.

PROGRAM:

```
clc; clear;

% Input ALU flags
CF = input('Enter Carry Flag (CF: 0 or 1): ');
OF = input('Enter Overflow Flag (OF: 0 or 1): ');

% OR operation
ABNORMAL = CF | OF;

% CPU Decision
if ABNORMAL == 1
   disp('CPU Decision: Abnormal Arithmetic Result');
else
   disp('CPU Decision: Normal Arithmetic Result');
end
```

OUTPUT:

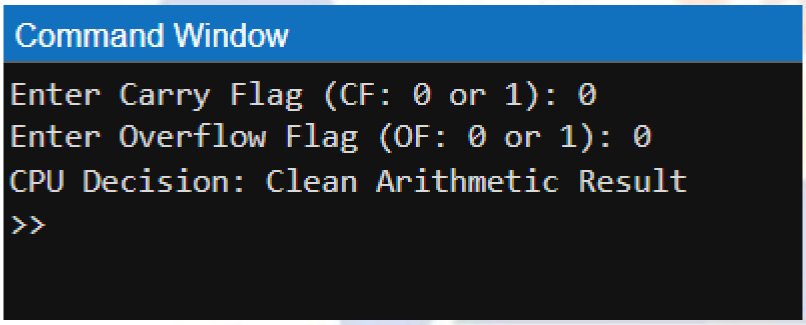


```
Enter Carry Flag (CF: 0 or 1): 0
Enter Overflow Flag (OF: 0 or 1): 0
CPU Decision: Normal Arithmetic Result
>> 
```

3. Implement a Clean Result Detection using **NOR** Gate (**NOR output = 1**)- CLEAN = NOR (CF, OF). **Note:** When neither carry nor overflow is detected, the CPU confirms a clean arithmetic result and output of NOR will be 1.
PROGRAM:

```
clc; clear;

% Input ALU flags
CF = input('Enter Carry Flag (CF: 0 or 1): ');
OF = input('Enter Overflow Flag (OF: 0 or 1): ');

% NOR operation
CLEAN = ~(CF | OF);

% CPU Decision
if CLEAN == 1
   disp('CPU Decision: Clean Arithmetic Result');
else
   disp('CPU Decision: Arithmetic Error Detected');
end
```

OUTPUT:



```
Enter Carry Flag (CF: 0 or 1): 0
Enter Overflow Flag (OF: 0 or 1): 0
CPU Decision: Clean Arithmetic Result
>>
```

XIV. Observations and Dimensional Measurement *(To be filled by the students)*

S. No.	Test Case ID	Input Value A	Input Value B	Operation Performed	ALU Status Flags (CF, OF, SF, PF)	Logic Expression Applied	Logic Output (0/1)	CPU Decision / Remarks
1	CASE-1	25	10	ADD	0 0 0 1	$\overline{CF}\cdot\overline{OF}\cdot PF$	1	Safe Output
2	CASE-2	-120	-20	ADD	0 1 1 0	CF OR OF	1	Overflow
3	CASE-3	-15	5	SUB	0 0 1 1	NAND($\overline{OF}$,SF)	0	Valid Negative
4	CASE-4	100	50	ADD	1 0 0 1	NOR(CF,OF)	0	Valid Negative

Note: Instructions for Filling the Table:

- Test Case ID:
  - Assign a unique identifier (Case–1, Case–2, etc.)
- ALU Status Flags:
  - Record CF, OF, SF, PF obtained from MATLAB program.
- Logic Expression Applied:
  - Example: $\overline{CF}$ AND $\overline{OF}$ AND PF or NAND($\overline{SF}$, OF)
- Logic Output:
  - Record 1 for TRUE, 0 for FALSE.
- CPU Decision / Remarks:
  - Example: Safe output, Valid negative result, Rejected due to overflow.

XV. Results/Observations *(To be filled by the students)*

- Logical operations were successfully implemented on ALU status flags using **AND, OR, NAND, and NOR gates** in MATLAB.
- The **Safe and Clean Output** condition was correctly identified when **Carry Flag (CF) = 0**, **Overflow Flag (OF) = 0**, and **Parity Flag (PF) = 1**, producing a logic output of **1** using AND and NOT operations.
- The **Valid Negative Result** was accurately detected using a **NAND gate**, where the logic output became **0** only when the result was negative (**SF = 1**) and no signed overflow occurred (**OF = 0**).
- The **Abnormal Result Detection** logic using an **OR gate** correctly flagged abnormal arithmetic results whenever **either carry or overflow** was present (CF = 1 or OF = 1).
- The **Clean Result Detection** using a **NOR gate** produced an output of **1** only when **both carry and overflow flags were zero**, confirming error-free arithmetic execution.
- Multiple test cases were executed, and in all cases, the **logic outputs matched the expected theoretical behavior** of ALU status flags and logic gates.
- The experiment demonstrated that **CPU decisions can be made efficiently using Boolean logic on status flags** without reprocessing arithmetic data.

XVI. Interpretation of Results *(Students to state the meaning of the above-obtained results/observations)*

- The obtained results show that **ALU status flags (CF, OF, SF, PF)** can be effectively interpreted using **basic Boolean logic operations**.
- The AND, OR, NAND, and NOR gate–based expressions correctly classified arithmetic results without requiring re-evaluation of numerical data.
- The **Valid Negative Result Detection** confirmed that a negative result is meaningful only when **no signed overflow occurs**, demonstrating correct signed arithmetic interpretation.
- The **Abnormal Result Detection** logic identified arithmetic errors whenever carry or overflow flags were set, indicating unsafe results for further processing.
- The **Clean Result Detection** verified that the absence of both carry and overflow flags corresponds to a reliable and error-free arithmetic outcome.
- The logical outputs observed during the experiment matched theoretical expectations, validating the correctness of the implemented control logic.

XVII. Conclusions *(Students to draw conclusions (or take decisions) based on the interpretation of results)*

This experiment successfully demonstrated how **logical operations on ALU status flags** are used to make reliable CPU decisions. By applying Boolean logic using AND, OR, NAND, and NOR gates, arithmetic results were efficiently classified as **safe, abnormal, valid negative, or clean**. The results highlight the importance of **flag-based decision-making** in processor control units, embedded systems, and exception handling. The experiment reinforces fundamental concepts of **computer organization, digital logic design, and efficient processor functionality**.

XVIII. Suggested Practicum Related Questions:

Note: *Below are a few questions related to industry-specific higher-order thinking skills (HOT) that teachers must use to ensure students achieve the predetermined course outcomes.*

1. ***Justify*** how XOR(SF, OF) can be used to detect signed overflow and justify its correctness..
2. ***Determine*** how XNOR(SF, OF) confirms a valid signed result.
3. ***Recommend*** a logic expression using OR(CF, OF) to identify any arithmetic error.
4. ***Generate*** a condition using NOR(CF, OF) to detect a completely clean arithmetic result.
5. ***Justify*** why NAND is preferred over AND in certain CPU control circuits.

XIX References / Suggestions for further reading

1. Computer Organization and Architecture – ALU and Status Flags
2. Digital Logic Design – Logic Gates and Applications
3. Processor Architecture Textbooks

XX Suggested Assessment Scheme

The performance indicators given serve as a guideline for assessing the *'process-related skills/LOs'* (marks to be awarded in real-time in the laboratory by the faculty member, as these cannot be measured after the practicum is over) and *'product-related skills/LOs'*

Note: The weightage for the ***process-related* skills** and ***product-related* skills** will change depending on the PrO, COs, and the competency related to the concerned course.

	Performance Indicators	Maximum Marks	Marks Obtained
Process-related Skills: 18 Marks - 60%			
1	Correctly creating and running MATLAB scripts (coding hygiene, commenting, workspace handling).	6	
2	Selecting appropriate arithmetic operations for the given data.	5	
3	Following safety, coding ethics, and good laboratory practices (file management, backups).	4	
4	Contribution as a team member (peer collaboration, sharing datasets, debugging).	3	
Product-related Skills: 12 Marks - 40%			
1	Producing accurate computation results and well-formatted MATLAB Table output.	3	
2	Correct interpretation of logical comparisons.	2	
3	Interpretation of results (clarity, mathematical validity).	3	
4	Drawing meaningful conclusions using NEP-aligned verbs (justify, determine, generate).	2	
5	Ability to answer practicum-related higher-order questions.	1	
6	Submitting the report on time with screenshots and code	1	
	Total	**30**	

To be filled by Student:

S.No.	Name of the Student	Register No.
1	**SILAS JOHN**	**URK25CS6011**

To be filled by the Faculty Member:

Marks Obtained			Name and Designation of the Faculty Member	Date of Evaluation
Process-Related Skills (18)	Product-Related Skills (12)	Total (30)		

****