ACOL 216: Introduction to Computer Architecture

IIT Delhi – Abu Dhabi
Semester II, 2025–26

Assignment 1: RISC-Tac-Toe

Due Date: **April 8, 2026**

Total Marks: 50

Overview

In this assignment, you will design and implement a complete **Tic-Tac-Toe State Verifier** for a generalized 6×6 board using the **SimpleRISC assembly language**. Your program will be given a **snapshot of a game state** via memory and must determine whether the state is **legal or illegal**. If the state is legal, the program must further determine whether the game outcome is a **win for Player 1**, **win for Player 2**, or a **draw**.

All inputs are provided before execution begins, and the program must produce a single deterministic output value. The assignment emphasizes low-level algorithm design, correctness reasoning, memory discipline, and exhaustive handling of edge cases under architectural constraints.

This assignment contributes **10%** to the course total.

Game Description

The game board is a fixed-size 6×6 grid. Each cell of the grid may contain one of the following values:

- **0** $\rightarrow$ empty cell,
- **1** $\rightarrow$ marker placed by Player 1,
- **2** $\rightarrow$ marker placed by Player 2.

Player 1 always moves first. Players alternate turns thereafter.

The provided game state represents a snapshot after an arbitrary number of moves. The program must infer legality and outcome solely from this snapshot.

Board Representation

The board is stored in memory in **row-major order**. Each cell occupies one memory word.

Let (i, j) denote the cell at row i and column j, where $0 \leq i, j < 6$. The memory address of cell (i, j) is computed as:

$$\texttt{base} + 4 \times (6i + j)$$

You must compute all indices algorithmically. Hardcoding cell addresses or winning patterns is not permitted.

Input Specification

All input is provided **before execution begins** through predefined memory locations. Input memory must not be modified.

Board State

A contiguous memory region contains exactly 36 words representing the 6×6 board snapshot. Each word stores one of the values $\{0, 1, 2\}$. No additional input will be provided.

Legality Rules

A board state is considered **legal if and only if all** of the following conditions hold:

1. Every cell contains only one of the values $\{0, 1, 2\}$.
2. The number of Player 1 markers is either equal to or exactly one greater than the number of Player 2 markers.
3. Player 2 cannot have more markers than Player 1.
4. Both players cannot simultaneously satisfy a winning condition.

If any of the above rules is violated, the state is **illegal**.

Winning Conditions

A player is declared a winner if the board contains **any contiguous sequence** of that player's markers of length k, where k can be $3, 4, 5,$ or 6. **The value of k will be given as input.**
in any of the following directions:

- horizontal,
- vertical,
- main diagonal,
- anti-diagonal.

All win detection must be performed algorithmically using loops and conditional branching. Hardcoding specific winning positions or patterns is not allowed. If both players satisfy a winning condition, the state is illegal.

Draw Condition

A board state is a **draw** if:

- the state is legal,
- neither player satisfies a winning condition.

There is no requirement that the board be completely filled for a draw.

Output Specification

At termination, your program must write exactly one integer value to the designated output memory location:

- **-1** : illegal state,
- **0** : legal state and draw,
- **1** : Player 1 wins,
- **2** : Player 2 wins.

Any other output value is considered incorrect.

Memory Interface Specification

This assignment will be **automatically graded**. All memory locations described below will be initialized by the grader before execution.

Input Memory

- **Board Base Address:** A fixed memory address marks the start of the 6×6 board state.

Input memory must not be overwritten.

Output Memory

- **Result Location:** A single designated memory location must be written exactly once with the final outcome.

Submission Instructions

This assignment must be completed **individually**.

Submit a single compressed file named:

`<entry_number>.zip`

Upon extraction, the archive must create a directory named `<entry_number>/` containing:

- `solution.asm` — the complete SimpleRISC implementation,
- `report.pdf` — a technical report (maximum **2 pages**) describing the design, memory layout, and correctness arguments.

No additional files or directories are permitted.

Evaluation Rubric

The assignment will be evaluated out of **50 marks**, based on how many test cases your program passes successfully during the evaluation. The exact number of test cases and marks for each will be disclosed later. **5 marks** are also reserved for the clarity of the report; however, the report will not compensate for incorrect program behaviour.

Academic Integrity

This assignment must be completed individually. Plagiarism in any form will result in **zero marks** for the assignment. We will use automated tools such as **MOSS** for plagiarism detection and **GPTZero** (or similar tools) for detecting inappropriate use of AI-based systems. Severe cases may attract additional penalties, including disciplinary action as per institute norms. If you are unsure whether a particular form of assistance is permitted, seek clarification before submission.

Important Notes

- **The process to install the SimpleRISC emulator and run the assembly programs on it will be demonstrated in the Tutorial sessions.**
- The assignment is strictly auto-graded.
- Correctness must hold for *all* legal board configurations.
- Incremental or ad-hoc coding approaches are unlikely to succeed.