**Akdeniz University Computer Engineering**
**CSE206 Computer Organization**
**2024-2025 Assignment#1 - The CPU Emulator, Due Date: May 7th 2025**

## 1. Introduction

This assignment aims to help students understand the architecture and behavior of a simple computer system. You are to implement a CPU emulator that supports binary-encoded 16-bit instructions and simulates a small memory and cache system. Write a CPU emulator software that supports a basic instruction set (15 instructions) given below.

## 2. System Specifications

• Memory: 65,536 bytes (64 KB) of byte-addressable main memory
• Cache: Direct-mapped, 16 bytes total (8 blocks × 2 bytes)
• Instruction Format:
- 4-bit opcode + 12-bit operand
- Memory address range: 0-4095 (12 bits)
• Two addressing modes:
  • Absolute (for PC and jumps)
  • Relative (for memory operands)
• Files:
- program.txt: Contains binary instructions (16-bit strings, one per line)
- config.txt:
  Line 1: Program load address (**hexadecimal**, e.g., 0x1000)
  Line 2: Initial PC value (**hexadecimal**, absolute address)

## 3. Complete Instruction Set

| Opcode | Mnemonic | Description |
|---|---|---|
| 0000 | START | Begin execution |
| 0001 | LOAD | AC = Immediate value |
| 0010 | LOADM | AC = Memory[address] (cached) |
| 0011 | STORE | Memory[address] = AC (cached) |
| 0100 | CMPM | Compare AC with Memory[address]; set flag |
| 0101 | CJMP | Jump if comparison flag > 0 |
| 0110 | JMP | Unconditional jump |
| 0111 | ADD | AC += Immediate value |
| 1000 | ADDM | AC += Memory[address] (cached) |
| 1001 | SUB | AC -= Immediate value |
| 1010 | SUBM | AC -= Memory[address] (cached) |
| 1011 | MUL | AC *= Immediate value |
| 1100 | MULM | AC *= Memory[address] (cached) |
| 1101 | DISP | Print AC value |
| 1110 | HALT | Stop execution |

## 4. Full Sample Program

The following example code is an app that can compute the sum of the numbers between 0 and 20. Your emulator must execute this sample code at minimum. Note that I may test your emulator with any code that is supported by the instruction set. You can implement your code in **Java**. Assume that initially all flags are set to zero. Only the 16-bit binary string will be in the file. Every line will have one single instruction.

```
Address   Instruction        Assembly
-------   ----------------   ------------
0000      0000000000000000   ; START
0001      0001000000010100   ; LOAD 20
0002      0011000011001000   ; STORE 200
0003      0001000000000000   ; LOAD 0
0004      0011000011001001   ; STORE 201
0005      0011000011001010   ; STORE 202
0006      0100000011001000   ; CMPM 200
0007      0101000000001111   ; CJMP 15
```

```
0008     0010000011001010    ; LOADM 202
0009     1000000011001001    ; ADDM 201
000A     0011000011001010    ; STORE 202
000B     0010000011001001    ; LOADM 201
000C     0111000000000001    ; ADD 1
000D     0011000011001001    ; STORE 201
000E     0110000000000110    ; JMP 6
000F     0010000011001010    ; LOADM 202
0010     1101000000000000    ; DISP
0011     1110000000000000    ; HALT
```

**5. Addressing Examples**

Given config.txt:

0x2000 (load address)
0x2000 (initial PC)

Note that only the hex numbers present in the config.txt file. It always consists of 2 lines

- "STORE 300" → Writes to 0x2000 + 300 = 0x212C
- "JMP 0x2050" → Sets PC to 0x2050 (absolute)
- "LOADM 100" → Reads from 0x2000 + 100 = 0x2064

**6. Execution and Expected Output**

When executing the sample program, report the last value only:

```
# java a1YourStudentId program.txt config.txt
 Value in AC: 210
 Cache hit ratio: 42.86%
```

**7. Implementation Requirements**

- Cache must handle:
  - Operate on physical addresses
  - Block replacement
  - Implement write-through policy
  - Track hit/miss statistics

- Required components:
  - Instruction decoder
  - Memory subsystem
  - Cache controller
  - CPU registers (PC, AC, Flag)

**8. Submission Format**

a1[StudentID].zip containing:

✓ Main.java (entry point)
✓ CPUEmulator.java
✓ Cache.java
✓ Memory.java
✓ program.txt
✓ config.txt
✓ README.txt

**9. Grading Breakdown**

- Functionality (60%)
- Cache Implementation (15%)
- Code Quality (10%)
- Report (15%)

Notes: All memory accesses MUST go through cache logic. Initialize all registers to 0 . Assume little-endian byte ordering. Please provide the java version that you use in the assignment in your report.