University of Central Florida

Department of Computer Science

CDA 3103C Computer Logic and Organization

Spring 2025

Programming Project (Tiny Architecture)

DUE 04/07/2025 by 11:59 p.m.

(This is a team Project: Max numbers of student by team = 2)

The Problem

Using C programming language, in this assignment you will implement a virtual tiny machine (VM) known as the Tiny-Harvard Machine Architecture. Your code must implement the basic instruction set architecture that the Tiny-Harvard Machine Architecture adheres to:

Opcode	mnemonic	meaning
01 →	LOAD <x>	A <— DM[x]
02 →	ADD <x>	A <— A + DM[x]
03 →	STORE <x>	DM[x] <— A
04 →	SUB <x>	A <— A - DM[x]
05 →	IN <5>	A <— Read from Device (5 represents keyboard)
06 →	OUT <7>	A —> Write to device (7 represents Screen)
07 →	HALT	End of program (stop running)
08 →	JMP <x>	PC <— x
09 →	SKIPZ	If A is equal zero, PC = PC + 4
10 →	SKIPG	If A > zero, PC = PC + 4
11 →	SKIPL	If A < zero, PC = PC + 4

Note: "x" represents a memory address. The numbers "5" and "7" represent device numbers

Each piece of the architecture must be accurately represented in your code (Instruction Register, Program Counter, Memory Address Register (MAR), Data Memory (DM), Memory Data Register (MDR), Instruction Memory (IM), and Accumulator). Instruction Memory will be implemented by a 0-128 array, and Data Memory will be represented by a 0-9 array. Your Program Counter will begin at address "0" in IM.

Program Memory (IM) and Data Memory (DM) may be implemented as separate arrays.

Hint: Each instruction will require 2 elements of the IM array: one for the opcode and one for the address.

CPU registers: PC, Accumulator (A), and IR

All other CPU registers are of type int except IR with is a struct (op, addr).

Instruction Cycle:

FETCH STEP:
In Fetch step the PC must be incremented by 2 because each instruction requires 2 memory words.

EXECUTE STEP
In the Execute step, the instruction placed in IR, is executed by the VM-CPU. The op-code (OP) component that is stored in the IR register (IR.OP) indicates the operation to be executed.

For instance, to implement FETCH and instruction LOAD you must implement each step:
(Do not use functions to implement each instruction)

FETCH
IR.op <— IM[PC]
IR.addr <— IM[PC+1]
PC <— PC + 2

LOAD (Execute step)
MAR <— IR.addr
MDR <— DM[MAR]
A <— MDR

ADD
MAR <— IR.addr
MDR <— DM[MAR]
A <— A + MDR

STORE
MAR <— IR.addr
MDR <— A
DM[MAR]. <— MDR

IN 5
A <— Keybord

OUT 7
Scree <— A

SKIPZ
If A == 0 then PC = PC + 4

JMP
PC <— IR.addr

HLT
Make the variable that controls the CPU execution loop equal zero.

Note: Lecture 1 describes the instruction set architecture of the Tiny Machine.

SUB is similar to **ADD. SKIPG and SKIPL** are implemented as **SKIPZ**.

Input Specifications

Your VM must run from the command line with a single input file as a parameter to main. This file will contain a sequence of instructions for your VM. These instructions must be store in "Instruction memory", and then run via the fetch/execute cycle.

VM is a one address machine, whose instruction format is: [OP| ADDRESS]. Example:

```
elf.text
5 5          //IN 5
6 7          //OUT 7
3 0          //STORE 0
5 5          //IN 5
6 7          //OUT 7
3 1          //STORE 1
1 0          //LOAD 0
4 1          //SUB 1
3 0          //STORE 0
6 7          //OUT 7
1 1          //LOAD 1
6 7          //OUT 7
7 0          //END
```

The elf.text file is the Executable Linkable File (object code - we reusing decimals instead of binary)

Output Specifications

Your VM should provide output according to the input file. Along with this output your program should provide status messages identifying details on the workings of your VM. Output text does not have to reflect my example word-for-word, but please provide detail on the program as it runs in a readable format that does not conflict with the actual output of your VM. After each instruction print the current state of the Program Counter, Accumulator, and Data Memory. The INPUT instruction is the only one that should prompt an interaction from the user.

Example:

Initial values

PC = 0 | A = NULL | DM = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

IN <5>
"Input a value:" 777

PC = 2 | A = *777*| DM = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

OUT <7>
Result is: 777

PC = 4 | A = 777 | DM = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

STORE <0>
PC = 6 | A = 777 | DM = [*777*, 0, 0, 0, 0, 0, 0, 0, 0, 0]

… etc

End of Program.

Rubric:

10 – Compiles
20 – Produces lines of meaningful execution before segfaulting or looping infinitely
5 – Follows IO specifications (takes command line argument for input file name and prints output to console)
5 – README.txt containing author names
5 – Fetch cycle is implemented correctly
15 – Instructions are not implemented with individual functions
5 – Well commented source code
5 – Arithmetic instructions are implemented correctly
5 – Read and write instructions are implemented correctly
10 – Load and store instructions are implemented correctly
10 – SKIP instructions implemented correctly
5 – jump instructions are implemented correctly

Submission

Your program must be submitted as a C file. The name of the program must be tinyvm.c
Submit as well a readme document, indicting how to compile and run your program.
Submit as well, the elf.text file (executable) for a program that multiplies to numbers (2 x 3).

Submit as well:
An output file which follows the output specifications. Use multiply2 x 3 for this file.