**HW 3**

**The Problem**

Using C programming language write a program that simulates a variant of the Tiny Machine Architecture. In this implementation memory (RAM) is split into Instruction Memory (IM) and Data Memory (DM). Your code must implement the basic instruction set architecture (ISA) of the Tiny Machine Architecture:

1 🡪 LOAD

2 🡪 ADD

3 🡪 STORE

4 🡪 SUB

5 🡪 IN

6 🡪 OUT

7 🡪 END

8 🡪 JMP

9 🡪 SKIPZ

Each piece of the architecture must be accurately represented in your code (Instruction Register, Program Counter, Memory Address Registers, Instruction Memory, Data Memory, Memory Data Registers, and Accumulator). Data Memory will be represented by an integer array. Your Program Counter will begin pointing to the first instruction of the program.

**For the sake of simplicity Instruction Memory (IM) and Data Memory (DM) may be implemented as separate arrays.**

Hint: Implementing a struct for your Instructions and an array of these **structs** as your Instruction Memory greatly simplifies this program.

Example:

typedef struct {

int opCode, device Or Address;

} **Instruction**;

Instruction IM[MAXPROGRAMSIZE];

Note: IM, MDR1, and IR are of type **Instruction**. All other CPU registers and Data Memory (DM) are of type int.

**Input Specifications**

Your simulator 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 simulator to store in “Instruction Memory” and then run via the fetch/execute cycle. In the input file each instruction is represented with two integers: the first one represents the opcode and the second one a memory address or a device number depending on the instruction.

Example:

Input File

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

**Output Specifications**

Your simulator 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 simulator. 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 simulator. 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:

Assembling Program…

Program Assembled.

Run.

PC = 10 | A = NULL | DM = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

/\* input value \*/

*X*

PC = 11 | A = *X* | DM = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

/\* outputting accumulator to screen \*/

*X*

PC = 12 | A = *X* | DM = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

/\* storing accumulator to memory location *0* \*/

PC = 13 | A = *X* | DM = [*X*, 0, 0, 0, 0, 0, 0, 0, 0, 0]

… etc

Program complete.

**Grading**

Your simulator will be graded on the above criteria. Your program should compile and run from the command line with one input file parameter. Please note that your program will not just be graded on whether or not it runs successfully; accurate simulation and a thorough demonstration of your understanding on the workings of this architecture will constitute a large portion of this grade. As that is the case it is in your best interest to comment your program in a concise and readable way. However, if your program does not run or compile the maximum points possible will be 30.

For instance, to implement FETCH and instruction LOAD you must implement each step:

**FETCH**

MAR 🡨 PC

PC 🡨 PC + 1

MDR 🡨 IM [MAR] // IM stands for Instruction Memory (program memory)

IR 🡨 MDR

Case IR.OP = 1 Load is executed.

**LOAD (Execute cycle)**

MAR2 🡨 IR.ADDR

MDR2 🡨 DM [MAR2] //DM stands for Data Memory

A 🡨 MDR2

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

**Submission**

Your program must be submitted as a C file. For example: NameMyProgram.c

Please check and double check your submission.

**Note: you can use one or two MARs, MAR for IM and MAR2 for DM.**

**Tiny Machine ISA:**

**FETCH**

MAR 🡨 PC

PC 🡨 PC + 1

MDR 🡨 IM [MAR] // IM stands for Instruction Memory (program memory)

IR 🡨 MDR

**Depending on IR.OP one of the following instructions will be executed:**

**(Execute cycle)**

**LOAD**

MAR2 🡨 IR.ADDR

MDR2 🡨 DM[MAR2]

A 🡨 MDR2

**ADD**

MAR2 🡨 IR.ADDR

MDR2 🡨 DM[MAR2]

A 🡨 A + MDR2

**STORE**

MAR2 🡨 IR.ADDR

MDR2 🡨 A

DM[MAR2] 🡨 MDR2

**SUB**

MAR2 🡨 IR.ADDR

MDR2 🡨 DM[MAR2]

A 🡨 A - MDR2

**IN**

A 🡨 Input value from keyboard

**OUT**

Screen 🡨 A

**END**

Run 🡨 0 // In your program Run must be initialized to 1 to control the instruction cycle.

**JMP**

PC 🡨 IR.ADDR

**SKIP**

IF (A == 0) PC 🡨 PC + 1