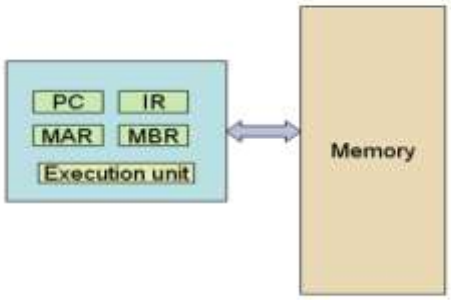
Part 1: Simple implementation of Load, Store, and Add instructions.

A simple computer has 16-bit cell memory with size of 128 Bytes (i.e. cell width = 16 bits). The memory is synchronous to the CPU, and the CPU can read/write one cell in a single clock cycle. The memory can only be accessed through the memory address register (MAR) and the memory buffer register (MBR).

To read from memory, you use: MBR <= Mem[MAR];

To write to memory, you use :Mem[MAR] <= MBR;

The CPU has a program counter (PC) register and an instruction register (IR). This CPU has eight 16-bit general-purpose registers. You have to implement the register file as two-dimensional array each entry has size of 16 bits. For example to access register 0 you can use R[0] notation.



This simple computer has only four instructions -- Load, Store, Add and Subtract. The size of all instructions is 16 bits with the following format:

Opcode(3bits)	Register (3bits)	Memory Address (6bits)	Not used (zeros) (4bits)

The opcodes are:

000 LOAD Ri, M: loads the contents of memory location **M** into register **Ri**, where **Ri** is the number of the register

011 STORE Ri, M: Stores the contents of **Ri** in memory location **M**

100 ADD Ri, M: Adds the contents of memory location **M** to the contents of **Ri,** and store the result in **Ri.**

111 SUB Ri, M: subtract the contents of memory location **M** from **Ri,** and store the result in **Ri.**

Testing:

Simulate the following program by converting each instruction to corresponding machine code. Then store the machine code in memory starting from location 10:

Memory Address	**Contents**
10	Load R1, [20] (instruction)
11	Add R1, [21] (instruction)
12	Sub R1, [22] (instruction)
13	Store R1, [23] (instruction)
20	9 (data)
21	4 (data)
22	3 (data)
23	0 (data)

Part 2: Add More Instructions and More Addressing Modes:

Objective: to be familiar with different addressing modes.

Update the instruction format by using least significant 4-bit as addressing mode to specify the addressing mode of the operand field. The instruction format is as follow:

Opcode (3bits)	Register (3bits)	Operand (6bits)	Addressing Mode (4bits)

Addressing mode field consists of 4 bits that determine the addressing mode of the operand field, as follow:
1-Direct Addressing (0000): Operand field is a direct memory address.

2-Indirect Addressing (0001): in this mode, memory cell pointed to address field contains the address of (pointer to) the operand.

3-Immediate Addressing (0010): in this mode, the operand field is a constant (integer).

4-Register Addressing (0011): in this mode, the operand field is a register. Least 3 bits can be used to specify the register and the most 3 bits can be set to zeros.

5-Register Indirect Addressing (0100): in this mode, the operand is in the memory cell pointed by the register specified in the operand filed.

Task:

Modify your simple computer to add the following new instructions:

000 LOAD Ri, M: loads the contents of memory location M into Ri, where Ri is the number of the register
001 LOAD Ri,5 set Ri to 5 (Immediate Addressing)
010 LOAD Ri, [[M]]: use the contents of memory location M as a pointer to the operand then load it to Ri (i.e. memory indirect address mode).

011 STORE Ri, M: stores the contents of Ri in memory location M.
100 ADD Ri,M: adds the contents of memory location M to the contents of Ri.
101 ADD Ri,5: adds constant 5 to the contents of Ri.

110 JUMP M: jumps to location M in memory.
111 SUB Ri, M: subtracts the contents of memory location M from the contents of Ri.

When you convert each form to corresponding machine code, you can construct the instruction format as following:

1-Opcode (3): Opcode of each instruction, for example
LOAD Ri,8 (001)

2-Register and Address/Immediate fields: if the form has two operands such as LOAD Ri, M , use Register field for Ri and operand for M. for example
Load R5,[3] -> Register filed=101 and address =000011

If the form has one operand only such as JUMP 7
Address field =000111 and don't care to Register filed.

Testing:

1-Simulate the following program by converting each instruction to corresponding machine code. Then store the machine code in memory starting from location 10:

Memory Address	Contents
10	Load R1,[20]
11	Load R2,21
12	Add R1,R2
13	Sub R1, [22]
14	Store R1, [23]
20	9 (Data)
21	4
22	3
23	0

2-Simulate the following program by converting each instruction to corresponding machine code. Then store the machine code in memory starting from location 10:

Memory Address	Contents
10	Load R1,20
11	load R2,2
12	Add R1,R2
	Add R1, [22]
13	
20	9 (data)
21	4
22	3
23	0

3-Simulate the following program by converting each instruction to corresponding machine code. Then store the machine code in memory starting from location 10:

Address	**Contents**
10	Load R2,21
11	load R1,[20]
12	Sub R1, [R2]
13	Store R1, [23]
14	Jump 10
20	9 (data)
21	4
22	3
23	0