BIRZEIT UNIVERSITY

Electrical and Computer Engineering

ENCS2380 – Computer Organization and Microprocessors- Spring 2023

Course Project (I)

Submission is only accepted on Moodle (itc.birzeit.edu)

Submission deadline: Thursday 15/6/2023 11:55PM.

Works in groups: you can work this project in groups of maximum 3 students together. The contribution of each student ***must be clarified*** in the submitted report.

Project report: Each group should submit a short report [3-4 pages], which includes the design details and the simulation of each required task. The simulation waveform can be added as a snapshot image in the report document.

Project specifications:

In this project, we need to design an accumulator computer. The memory in this computer system is 16-bit cell memory. Assume that 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).

In addition to the accumulator register (AC), the CPU has a program counter (PC) register and an instruction register (IR).

The 16-bit instruction format is as follow:

Opcode(4bits)	M (1bit)	Memory address/constant (11bits)

Mode bit : M=1 ➔ Memory address

M = 0 ➔ Immediate constant (signed integer 2's complement)

This machine should support at least the following instructions with the following opcodes:

Instruction	Opcode
Load	1
Store	2
ADD	3
Sub	4
Mul	5
Div	6
Branch	7
BRZ*	8

(*) Branch if Z flag is set, i.e. ZF=1. With Branch and BRZ instructions, use M=1 and jump to the specified memory address.

Assume data is singed 16-bit integers in the 2's complement format.

Example:

To add memory cell of address 10 with memory cell of address 11, and store the result in memory cell of address 12, we use the following program:

Assembly	Machine code
Load [10]	0001 1 00000001010
ADD [11]	0011 1 00000001011
Store [12]	0010 1 00000001100

Computer design and implementation:

1) Design and implement main memory as a Verilog module:

The memory can be implemented as an array of registers in Verilog with each register represents a memory cell. The array index works as the memory address. This module should takes address, data_in and control signals read and write as input and the data_out as output.

Example, `my_memory module (address, data_in, rd, wr, data_out);`

For your project, you need to determine the maximum memory size (in cells and in bytes) that this machine can access directly. Remember, one cell is 16 bits wide.

The memory can be implemented as an array of registers in Verilog. For example, the following line declares a 16-bit cells memory of size 64 (i.e. memory size 128 Bytes).

```
Reg [15:0] memory[0:63];
```

2) Design and implement CPU as a Verilog module:

The CPU module consists of the registers; AC, PC, IR, etc, and the arithmetic and logical unit (ALU) and the control unit which controls the data flow by providing control signals at specific timing steps for each specific instruction. In addition to the arithmetic and logical operation result, the ALU provides four flags (Carry, Zero, Overflow, and Negative) with each arithmetic and logical operation.

The CPU can be implemented as one module that takes clock, memory data as input, and memory address, data out, read, and write as output.

3) Design and implement the whole computer:

This is the top-level design of this basic computer. This can be implemented as a schematic file, which include the memory module and the CPU module as blocks and connect them together.

Simulation:

1) Initialize the memory with the following four instructions at memory address 0-3, and data at memory address 10-12, as shown in the following table.

a) Interpret each instruction into assembly instruction and add it to its corresponding instruction in the table. Also, interpret the integer's data into decimal and add them into the table.

b) Simulate the four instructions at address 0 by initializing PC =0. Provide a snapshot of your resulted waveform. Verify that it works correctly and the also verify that the result stored at address 12 is correct. Attach simulation waveform and the Verilog source file.

Memory Address	**Content**	**Content interpretation: assembly instruction + data in decimal**
0	0x180A	
1	0x580B	
2	0x3005	
3	0x280C	
10	0x0009	
11	0xFFFC	
12	0x0000	

2) Assume A,B,C,D,E and Y are memory cells with addresses 20,21,22,23,24, and 25, respectively.

Given , $Y = \frac{A+B*C-5}{D+E+1}$,

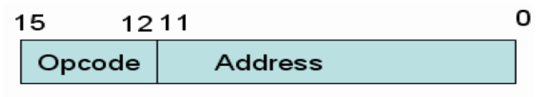
a) Write assembly code for implementing the above arithmetic expression?

b) Convert the above assembly instructions into machine code and store them in the memory starting at address 0.

address	**content**
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

c) Set PC=0 and simulate the above program. Verify that it works correctly and the result stored at memory variable Y is correct. Attach simulation waveform and the Verilog source file. **Assume A, B, C, D and E have the values 2, 3, 5, 8, and -5, respectively.**

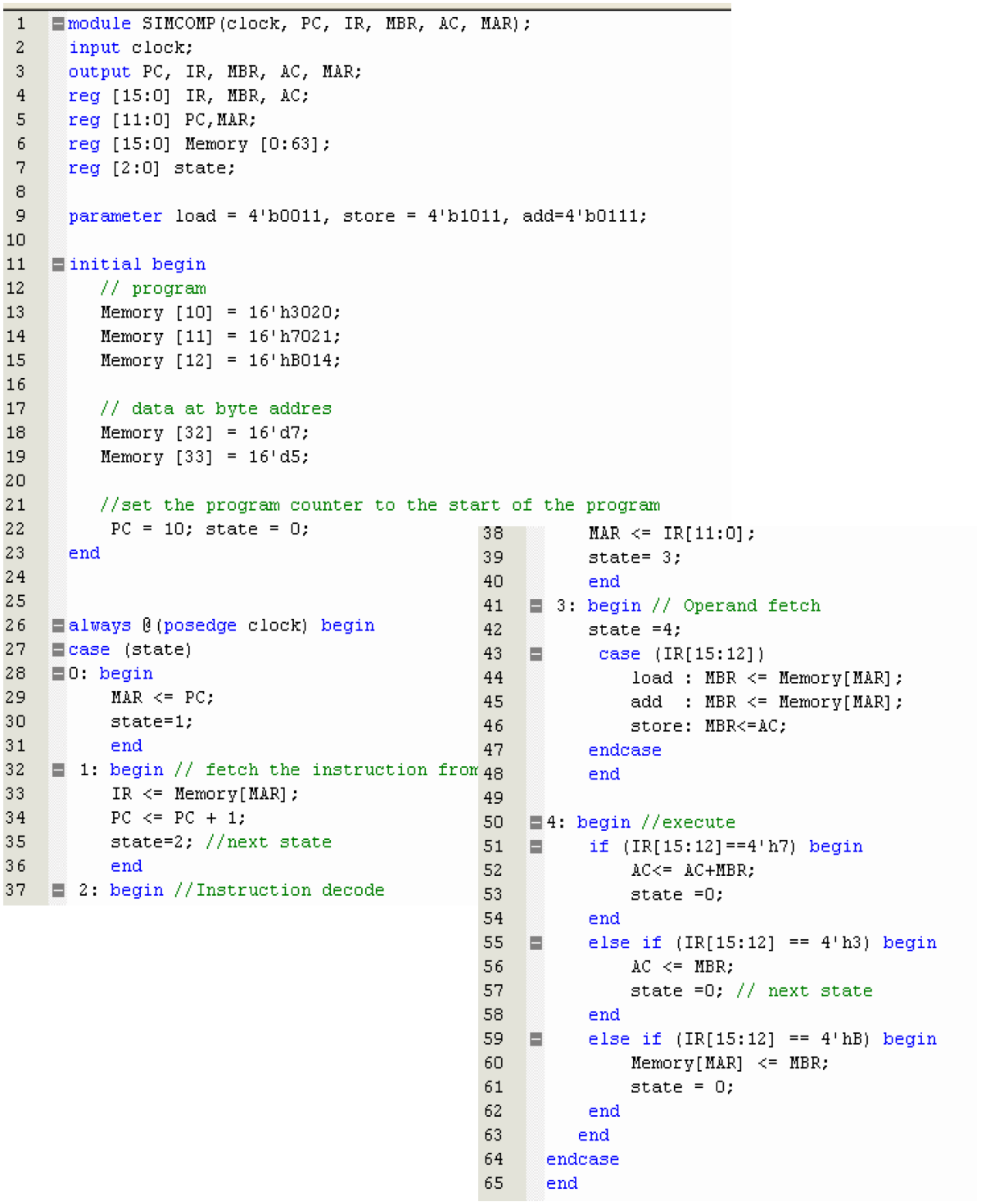
Important Note: you can start with the Verilog implementation of a simple accumulator machine, as shown in the following:



Instruction Format

__The opcodes are:__

- **0011 LOAD M** // loads the contents of memory location **M** into the accumulator.
- **1011 STORE M** // stores the contents of the accumulator in memory location **M**.
- **0111 ADD M** // adds the contents of memory location **M** to the contents of the accumulator.



```
module SIMCOMP(clock, PC, IR, MBR, AC, MAR);
input clock;
output PC, IR, MBR, AC, MAR;
reg [15:0] IR, MBR, AC;
reg [11:0] PC,MAR;
reg [15:0] Memory [0:63];
reg [2:0] state;

parameter load = 4'b0011, store = 4'b1011, add=4'b0111;

initial begin
   // program
   Memory [10] = 16'h3020;
   Memory [11] = 16'h7021;
   Memory [12] = 16'hB014;

   // data at byte addres
   Memory [32] = 16'd7;
   Memory [33] = 16'd5;

   //set the program counter to the start of the program
    PC = 10; state = 0;
end


always @(posedge clock) begin
case (state)
0: begin
    MAR <= PC;
    state=1;
    end
 1: begin // fetch the instruction from
    IR <= Memory[MAR];
    PC <= PC + 1;
    state=2; //next state
    end
 2: begin //Instruction decode
    MAR <= IR[11:0];
    state= 3;
    end
 3: begin // Operand fetch
    state =4;
     case (IR[15:12])
       load : MBR <= Memory[MAR];
       add  : MBR <= Memory[MAR];
       store: MBR<=AC;
    endcase
    end

4: begin //execute
    if (IR[15:12]==4'h7) begin
        AC<= AC+MBR;
        state =0;
    end
    else if (IR[15:12] == 4'h3) begin
        AC <= MBR;
        state =0; // next state
    end
    else if (IR[15:12] == 4'hB) begin
        Memory[MAR] <= MBR;
        state = 0;
    end
   end
endcase
end
```