Bir zeit University

Faculty of Information Technology

Computer Systems Engineering Department

Digital Lab ENCS 211 EXP. No. 11

Simple Computer

Objective:

In this experiment we are going to design the Verilog HDL control sequence for a simple computer (SIMCOMP). The SIMCOMP is a very small computer to give you practice in the ideas of designing a simple CPU with the Verilog HDL notation.

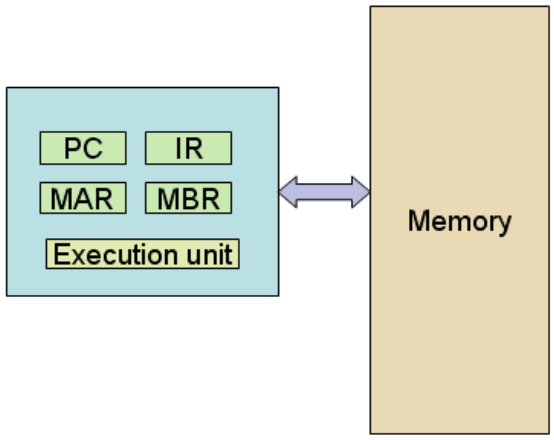
Pre-Lab:

1-read the experiment

2-Do part one and part two of the procedure

Introduction

SIMCOMP has a two byte-addressable memory with size of 128byte. The memory is synchronous to the CPU, and the CPU can read or write a word in single clock period. 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 <= Memmory[MAR];
```

To write to memory, you use

```
Memory[MA] <= MBR;
```

The CPU has three registers -- an accumulator (**AC**), a program counter (**PC**) and an instruction register (**IR**).

SIMCOMP has only three instructions -- **Load**, **Store**, and **Add**. The size of all instructions is 16 bits; all the instructions are single address instructions and access a word in memory.

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
```

Procedure:

1- Study and simulate the SIMCOMP verilog program.

2- Add extra instruction (JUMP) to SIMCOMP

JUMP M jumps to location **M** in memory.

Simulate the following program

Address	Contents
5	Load 9
6	Add 10
7	Store 11
8	Jump 6
9	Data 3
10	Data 2

3-SIMCOMP2: Add register file

Modify the instruction format so that SIMCOMP2 can handle four addressing modes and four registers,

To this end, SIMCOMP is an **accumulator** machine which you can think of as a machine with one general-purpose register. Historically, many old computers were accumulator machines.

This new SIMCOMP2 has four 16-bit general purpose registers, R[0], R[1], R[2] and R[3] which replace the AC. In Verilog, you declare R as a bank of registers much like we do Memory:

```
reg [15:0] R[0:3];
```

And, since registers are usually on the CPU chip, we have no modeling limitations as we do with Memory - *with Memory we have to use the MAR and MBR registers to access Memory*. Therefore, in a load you could use R as follows:

```
R[IR[9:8]] <= MBR;
```

where the 2 bits in the IR specify which R register to set.

Modify the four instructions of the old SIMCOMP2 to the following new form:

LOAD R[i],M loads the contents of memory location **M** into R[i].

STORE R[i],M stores the contents of R[i] in memory location **M**.

ADD R[i],R[j],R[k] adds contents of R[j] and R[k] and places result in R[i].

JUMP M jumps to location **M** in memory.

To test your SIMCOMP2 design, perform the following program where PC starts at 10.

```
 3      DATA    A
 4      DATA    6
10      Load    R1,3
 11      Load    R2,4
12      Add     R1,R1,R2
13      Store   R1,5
```

4-Add immediate addressing to the SIMCOMP2:

If bit (IR[11])is a one in a Load , the last eight bits are not an address but an operand. The operand is in the range -128 to 127.

If immediate addressing is used in an LOAD, the operand is loaded into the register.

```
LoadI R1,8          R1 <- 8
```

Simulate the following program

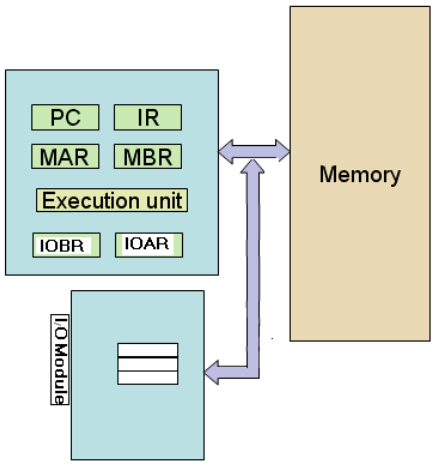
```
PC = 10
 Memory [10]   LoadI R1,3     // Load immediate
  Memory [11]   Store R1,4
  Memory [12]   LoadI R2,-4
 Memory [13]   Add   R2,R2,R1
 Memory [14]   Store R2,5
```

4-Input/output Instructions to the SIMCOMP2:

In order SIMCOMP2 to communicate with the Input and output devices we are going to add `in` and `out` instructions to SIMCOMP2

```
In N read one byte from input device at address N into lower byte of R0
Out N write the lower byte of R0 to Output device at address N
```

Add two registers (Address and Data) for I/O devices



Complete the a program below such that it reads port 1 then it adds the value to the content of memory location 4 and send the results to port 2

```
 3    DATA  A
 4    DATA  6
10    loadI R0, 0
11    IN 1
.  ..  …  …
```

simulate the program then rout it on the FPGA , use the switches and leds to show the results.

Note: your final machine should be able to correctly run the three "software"programs of all last three exercises. Be careful not to destroy the features of previous exercises. You should test this and include output in your report file to show that your **final** version of the SIMPCOMP2 works properly with the two programs.

Extra: write an assembler using c/c++ so that it convert the assembly code to machine code for SIMCOMP2
example.as is assembly file
example.hex is contains the machine code