Comp E 470L

Digital Systems

HW 8: Decoder/Control

Student: Davit Barbakadze

Red ID #: 822164584

19.12.2019

Contents

[Task Description 3](#_Toc22460835)

[Solution 3](#_Toc22460836)

[Simulation & Verification 10](#_Toc22460837)

[Comparison 11](#_Toc22460838)

[Conclusion 11](#_Toc22460839)

# Task Description

For this assignment students had to implement instruction decoder/controller part of CPU, following the scheme and lecture materials.(see scheme on figure 1).

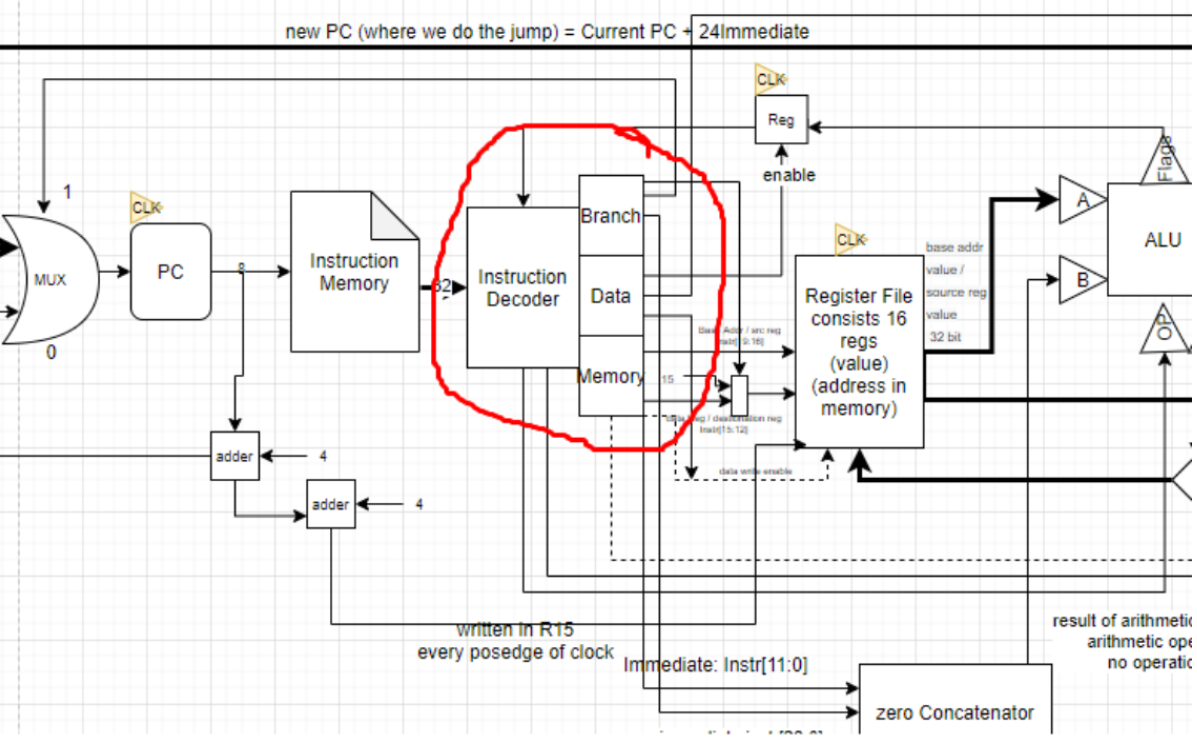


Figure 1 Decoder/Control scheme

The scheme shows whole CPU parts but we were asked to write only the part inside the red circle.

# Solution

Unlike PC or ALU arts this part is quite complicated as many inputs and outputs are being controlled through different branches. As seen on the scheme there are three inputs, one is clock for register for flags, instructions that is coming from instruction memory, which should be decoded and allocated, and flags from the ALU. We also have many outputs for different instructions, which are all shown in the Verilog files.

-------------------------------------------------VERILOG CODE-------------------------------------

module decoder\_module(

input [31:0] instr,

input clk,

input [3:0] flag,

output reg [1:0] op,

output reg [3:0] bits,

output reg [23:0] imminstr,

output reg [3:0] base,

output reg [3:0] data\_reg,

output reg [11:0] imminstr\_mem,

output reg jmp\_en,

output reg regjmp\_en,

output reg flag\_en,

output reg write\_data,

output reg memory\_data,

output reg memdata,

output reg memdata\_en

);

reg m, n, k;

always @ (posedge clk)

begin

bits <= instr [24:21];

op <= instr [27:26];

m <= instr [20];

n <= instr [20];

k <= instr [25];

case (op)

0:

begin

jmp\_en <= 0;

imminstr <= 0;

regjmp\_en <= 0;

base <= instr [19:16];

data\_reg <= instr [15:12];

memory\_data <= 0;

imminstr\_mem <= 0;

memdata <= 0;

memdata\_en <= 0;

if (m == 1) flag\_en <= 1;

else flag\_en <= 0;

if (bits == 10) write\_data <= 0;

else write\_data <= 1;

end

1:

begin

jmp\_en <= 0;

regjmp\_en <= 0;

imminstr <= 0;

flag\_en <= 0;

write\_data <= 0;

base <= instr [19:16];

data\_reg <= instr [15:12];

memdata\_en <= 1;

if (n == 1)

begin

memory\_data <= 1;

memdata <= 0;

memdata\_en <= 1;

end

else

begin

memory\_data <= 0;

memdata <= 1;

memdata\_en <= 0;

end

if (k == 1) imminstr\_mem <= instr [11:0];

else imminstr\_mem <= 0;

end

2:

begin

jmp\_en <= 1;

regjmp\_en <= 1;

imminstr <= instr [23:0];

flag\_en <= 0;

write\_data <= 0;

base <= 0;

data\_reg <= 0;

memory\_data <= 0;

imminstr\_mem <= 0;

memdata <= 0;

memdata\_en <= 0;

end

3:

begin

jmp\_en <= 0;

regjmp\_en <= 0;

imminstr <= 0;

flag\_en <= 0;

write\_data <= 0;

base <= 0;

data\_reg <= 0;

memory\_data <= 0;

imminstr\_mem <= 0;

memdata <= 0;

memdata\_en <= 0;

end

endcase

end

endmodule

-------------------------------------------------TESTBENCH CODE-----------------------------------

`timescale 1ns / 1ps

////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 16:14:32 12/19/2020

// Design Name: decoder\_module

// Module Name: D:/Desktop/Xilinx ISE Design Suite 14.5/decoder/testbench.v

// Project Name: decoder

// Target Device:

// Tool versions:

// Description:

//

// Verilog Test Fixture created by ISE for module: decoder\_module

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

////////////////////////////////////////////////////////////////////////////////

module testbench;

// Inputs

reg [31:0] instr;

reg clk;

reg [3:0] flag;

// Outputs

wire [1:0] op;

wire [3:0] bits;

wire [23:0] imminstr;

wire [3:0] base;

wire [3:0] data\_reg;

wire [11:0] imminstr\_mem;

wire jmp\_en;

wire regjmp\_en;

wire flag\_en;

wire write\_data;

wire memory\_data;

wire memdata;

wire memdata\_en;

// Instantiate the Unit Under Test (UUT)

decoder\_module uut (

.instr(instr),

.clk(clk),

.flag(flag),

.op(op),

.bits(bits),

.imminstr(imminstr),

.base(base),

.data\_reg(data\_reg),

.imminstr\_mem(imminstr\_mem),

.jmp\_en(jmp\_en),

.regjmp\_en(regjmp\_en),

.flag\_en(flag\_en),

.write\_data(write\_data),

.memory\_data(memory\_data),

.memdata(memdata),

.memdata\_en(memdata\_en)

);

always #1 clk = !clk;

initial begin

// Initialize Inputs

instr = 0;

clk = 0;

flag = 0;

// Wait 100 ns for global reset to finish

#100; instr = 32'b11010011010010010110111111111111;

#100; instr = 32'b10010011111001011010011111111111;

#100; instr = 32'b11010111111100001000111011000000;

#100; instr = 32'b10010101111011000001000011111111;

#100; instr = 32'b11101011101010101010101010101010;

#100; instr = 32'b11011011010101010101010101010101;

end

endmodule

# Simulation & Verification

Figure 6 testbench for SSD (seven segment display)

In testbench we just need to create clock signal and see if states are changed properly, remember that we had to change states so fast that eye could not see segments are blinking.

So, finally as we see in figure 6, states are being changed properly, as well as proper numbers are displayed on the seven segment display. Clock divider also works, note that I have used 2 as CEIL value for testing purposes. In case of implementation we would need much larger value.

# 

# 

# Conclusion

To conclude, decoder part turned out to be complicated, due to its amount of instruction controlling outputs. But it was essential in learning how CPU operates, so overall, it was interesting and educational part.

Git Link: https://github.com/nikanika0221/Decoder