Anuar Zhanat, 201629848

Cute processor

Demo video link: https://youtu.be/YIaDovYV480

Description

In this lab, we established the simple processor by using Verilog and CTKWave programs, designing simple architecture of the processor. The CPU contains the 16-bit registers, MUX, ALU (add/subtract unit), counter (clock), and control unit with DIN input. The input loads 16-bit MUX into registers named as R0, R1, R2, …, R7, transferring the data from one to another by the output wires, buswires. In addition, bus wires will help to transfer the data to A register and perform add/subtraction operations resulting in the G registrar. It will perform required operations based on clock cycle and control unit with the proper instructions established in the table below:

Operation	Function performed
mv R_X, R_Y	$R_X \leftarrow [R_Y]$
mvi $R_X, \#D$	$R_X \leftarrow D$
add R_X, R_Y	$R_X \leftarrow [R_X] + [R_Y]$
sub R_X, R_Y	$R_X \leftarrow [R_X] - [R_Y]$

Each instruction such as mv, mvi, add, and sub will be encoded and stored in the IR register in 9-bit format with instruction and two source registers. However, due to the time-complexity of the AddSub module it will take more than one clock cycle to complete the performance. Therefore, it was necessary to use a 2-bit counter by controller, “step-by-step” instruction (see in the table below).

	T_1	T_2	T_3
(mv): I_o	$RY_{out}, RX_{in}, Done$		
(mvi): I_1	$DIN_{out}, RX_{in}, Done$		
(add): I_2	$RX_{out}. A_{in}$	RY_{out}, G_{in}	$G_{out}, RX_{in}, Done$
(sub): I_3	RX_{out}, A_{in}	$RY_{out}, G_{in}, AddSu$	$G_{out}, RX_{in}, Done$

Results

cute_processor.v verilog code:

```
/*
Anuar Zhanat
Cute Processor
Laboratory work 3
Introduction to Computer Architecture
*/

module dec3to8(   input [2:0] W,
            input En,
            output reg [0:7] Y);

            always @(W or En)
            begin
                  if (En == 1)
                        case (W) // 8 number of registers R0-R7
                              3'b000: Y = 8'b10000000;
                              3'b001: Y = 8'b01000000;
                              3'b010: Y = 8'b00100000;
                              3'b011: Y = 8'b00010000;
                              3'b100: Y = 8'b00001000;
                              3'b101: Y = 8'b00000100;
                              3'b110: Y = 8'b00000010;
                              3'b111: Y = 8'b00000001;
                        endcase
                  else
                        Y = 8'b00000000;
            end
endmodule

module regn(R, Rin, Clock, Q);
      parameter n = 16;
      input [n-1:0] R;
      input Rin, Clock;
      output [n-1:0] Q;
      reg [n-1:0] Q;
      always @(posedge Clock)
            if (Rin)
                  Q <=R;
endmodule

// Main process module
module cute_processor(
      input [15:0] DIN,
```

```
	input Resetn, Clock, Run,
	output reg Done,
	output reg [15:0] BusWires);

	parameter T0 = 2'b00, T1 = 2'b01, T2 = 2'b10, T3 = 2'b11;
	reg [0:7] Rin, Rout;
	reg [15:0] Sum;
	reg IRin, DINout, Ain, Gin, Gout, AddSub;
	reg [1:0] Tstep_Q, Tstep_D;
	wire [2:0] I;
	wire [0:7] Xreg, Yreg;
	wire [15:0] R0, R1, R2, R3, R4, R5, R6, R7, A, G;
	wire [1:9] IR;
	wire [1:10] Sel;
	wire clear=Resetn|Done|(~Run&~Tstep_Q[0]&~Tstep_Q[1]);
	assign I = IR[1:3];
	dec3to8 decX (IR[4:6], 1'b1, Xreg);
	dec3to8 decY (IR[7:9], 1'b1, Yreg);

	always @(Tstep_Q, Run, Done)
	begin
		case (Tstep_Q)
			T0:
				if (~Run) Tstep_D = T0;
				else Tstep_D = T1;
			T1:
				if (Done) Tstep_D = T0;
				else Tstep_D = T2;
			T2:
				Tstep_D = T3;
			T3:
				Tstep_D = T0;
		endcase
	end

	parameter mv = 3'b000, mvi = 3'b001, add = 3'b010, sub = 3'b011;
	always @(Tstep_Q or I or Xreg or Yreg)
	begin
		Done = 1'b0; Ain = 1'b0; Gin = 1'b0; Gout = 1'b0; AddSub = 1'b0;
		IRin = 1'b0; DINout = 1'b0; Rin = 8'b0; Rout = 8'b0;
		case (Tstep_Q)
			T0:
				begin
					IRin = 1'b1;
				end
			T1:
				case (I)
					mv:
					begin // Copying the data from R registers
```

```
                                    Rout = Yreg;
                                    Rin = Xreg;
                                    Done = 1'b1;
                              end
                              mvi:
                              begin
                                    DINout = 1'b1;
                                    Rin = Xreg;
                                    Done = 1'b1;
                              end
                              add, sub:
                              begin
                                    Rout = Xreg;
                                    Ain = 1'b1;
                              end
                              default: ;
                        endcase
                  T2:
                        case (I)
                              add:
                              begin
                                    Rout = Yreg;
                                    Gin = 1'b1;
                              end
                              sub:
                              begin
                                    Rout = Yreg;
                                    AddSub = 1'b1;
                                    Gin = 1'b1;
                              end
                              default: ;
                        endcase
                  T3:
                        case (I)
                              add, sub:
                              begin
                                    Gout = 1'b1;
                                    Rin = Xreg;
                                    Done = 1'b1;
                              end
                              default: ;
                        endcase
            endcase
      end

      always @(posedge Clock, negedge Resetn)
            if (!Resetn)
                  Tstep_Q <= T0;
            else
```

```
                  Tstep_Q <= Tstep_D;
      regn reg_0 (BusWires, Rin[0], Clock, R0);
      regn reg_1 (BusWires, Rin[1], Clock, R1);
      regn reg_2 (BusWires, Rin[2], Clock, R2);
      regn reg_3 (BusWires, Rin[3], Clock, R3);
      regn reg_4 (BusWires, Rin[4], Clock, R4);
      regn reg_5 (BusWires, Rin[5], Clock, R5);
      regn reg_6 (BusWires, Rin[6], Clock, R6);
      regn reg_7 (BusWires, Rin[7], Clock, R7);
      regn reg_A (BusWires, Ain, Clock, A);
      regn #(.n(9)) reg_IR (DIN[15:7], IRin, Clock, IR);

      // ALU
      always @(AddSub or A or BusWires) // Instruction add/sub
            begin
            if (!AddSub)
                  Sum = A + BusWires;
            else
                  Sum = A - BusWires;
                  end
            regn reg_G (Sum, Gin, Clock, G);
            assign Sel = {Rout, Gout, DINout}; // Defining the bus

            always @(*)
            begin
                  if (Sel == 10'b1000000000)
                        BusWires = R0;
                  else if (Sel == 10'b0100000000)
                        BusWires = R1;
                  else if (Sel == 10'b0010000000)
                        BusWires = R2;
                  else if (Sel == 10'b0001000000)
                        BusWires = R3;
                  else if (Sel == 10'b0000100000)
                        BusWires = R4;
                  else if (Sel == 10'b0000010000)
                        BusWires = R5;
                  else if (Sel == 10'b0000001000)
                        BusWires = R6;
                  else if (Sel == 10'b0000000100)
                        BusWires = R7;
                  else if (Sel == 10'b0000000010)
                        BusWires = G;
                  else BusWires = DIN;
            end
endmodule
```

proc_tb.v testbench verilog code:

```
module processor_tb();
	reg [15:0]DIN;
	reg Resetn;
	reg Clock;
	reg Run;
	wire Done;
	wire [15:0]BusWires;

		cute_processor dut (
		.DIN(DIN),
		.Resetn(Resetn),
		.Clock(Clock),
		.Run(Run),
		.Done(Done),
		.BusWires(BusWires));

		initial
		begin
			Resetn = 0;
			#20 Resetn = 1;
		end

		initial
		begin
			Clock = 0;
			#10 Clock = 1;
			forever #(10) Clock = ~Clock;
		end
		initial
		begin
			#0 Run = 0; DIN = 16'h0000;
			#20 Run = 1; DIN = 16'h2000;
			#20 Run = 0; DIN = 16'h0005;
		end
		initial
		begin
			#60 Run = 1; DIN = 16'h0400;
			#20 Run = 0; DIN = 16'h0400;
		end
		initial
		begin
			#100 Run = 1; DIN=16'h4080;
			#20 Run =0 ; DIN=16'h4080;
		end
		initial
		begin
			#180 Run = 1; DIN=16'h6000;
```

```
                    #20 Run = 0; DIN=16'h0000;
                end
                initial
                begin
                    $dumpfile("cute_proc.vcd"); $dumpvars(0, processor_tb);

                end
endmodule
```

To perform the process type in the terminal:

```
iverilog -o cpu processor_tb.v cute_processor1.v
vvp cpu
```

Experiment results by GTKWave: