Brandon Amisael Garrido Ramírez
Carné: 19421
Sección: 20

Ing. Mecatrónica
Electrónica Digital 1

Laboratorio #10

Enlace al repositorio de GitHub: https://github.com/gar19421/Lab-10.git

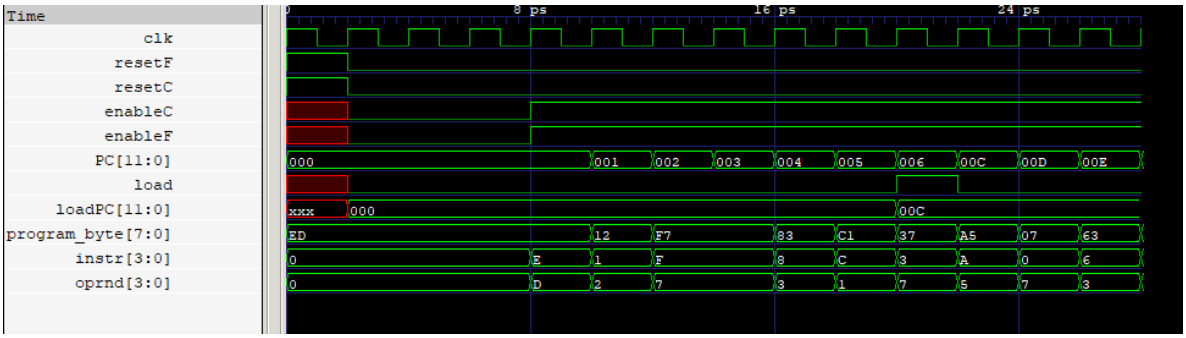
---

Ejercicio 1:

En este ejercicio se realizó un fragmento del microprocesador a trabajar en el laboratorio, en este ejercicio como se puede observar se implemento el program counter utilizando el contador codificado en laboratorios previos, el fetch el cual es un flip flop de 8bits que se implemento con 8 flipflops de 1bit codificados anteriormente. Por aparte tambien se implemento una program ROM junto con una lista de memoria de 32 direcciones, todo esto se implemento en un modulo control para manejar el fragmento del microprocesador.

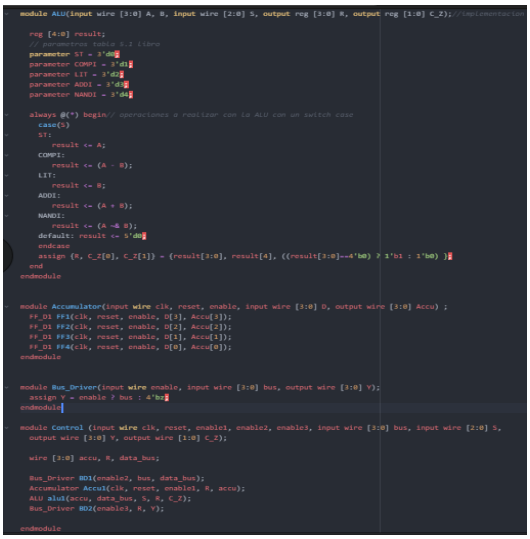
```
module FF_1BIT(input wire clk, reset, enable, D, output reg Y) ;
  always @(posedge clk or posedge enable or posedge reset) begin
    if (reset)  Y <= 1'b0;
    else if(enable) Y <= D;
  end
endmodule

module Program_Couter(input wire clk, reset, enable, load, input [11:0] loadPC, output reg[11:0] count);
//contador de 12 bits con entradas de clock, reset, enable y load

  always @ (posedge clk or posedge reset or posedge enable or posedge load) begin
    if(reset) begin
     count <= 12'd0; // si se resetea empieza en cero
    end
    else if(clk & enable & ~load) begin
        #2 count <= count + 12'd1; // si esta en enable cuenta el contador sino no cuenta
    end
    else if(clk & load & enable) begin
        #2 count <= loadPC; //precargando el valor
    end
  end

endmodule

module Fetch (input wire clk, reset, enable, input wire [7:0] program_byte, output wire [3:0] instr, oprnd) ;
  FF_1BIT FF1(clk, reset, enable, program_byte[7], instr[3]);
  FF_1BIT FF2(clk, reset, enable, program_byte[6], instr[2]);
  FF_1BIT FF3(clk, reset, enable, program_byte[5], instr[1]);
  FF_1BIT FF4(clk, reset, enable, program_byte[4], instr[0]);
  FF_1BIT FF5(clk, reset, enable, program_byte[3], oprnd[3]);
  FF_1BIT FF6(clk, reset, enable, program_byte[2], oprnd[2]);
  FF_1BIT FF7(clk, reset, enable, program_byte[1], oprnd[1]);
  FF_1BIT FF8(clk, reset, enable, program_byte[0], oprnd[0]);
endmodule

module Program_ROM (input wire [11:0] address, output wire [7:0] word);//memoria rom
  reg [7:0] mem [0:4095];// creacion de arreglo de memoria
  initial begin
      $readmemb("memory.list", mem);//Lectura del archivo de datos
  end
  assign word = mem[address];// resultado leido de la memoria
endmodule

module Control (input clk, resetC, resetF, enableC, enableF, load, input [11:0] loadPC, output [3:0] instr, oprnd);

wire [11:0] PC;
wire [7:0] program_byte;

Program_Couter PC1(clk, resetC, enableC, load, loadPC, PC);
Program_ROM PR1(PC, program_byte);
Fetch F1(clk, resetF, enableF, program_byte, instr, oprnd);
```

Como se puede observar en el diagrama de timing, en cada flanco de reloj el contador aumenta y se obtiene el valor de memoria en dicha posición y cuando se activa el load en 1 se carga el valor de memoria en la posición precargada identificada como loadPC.



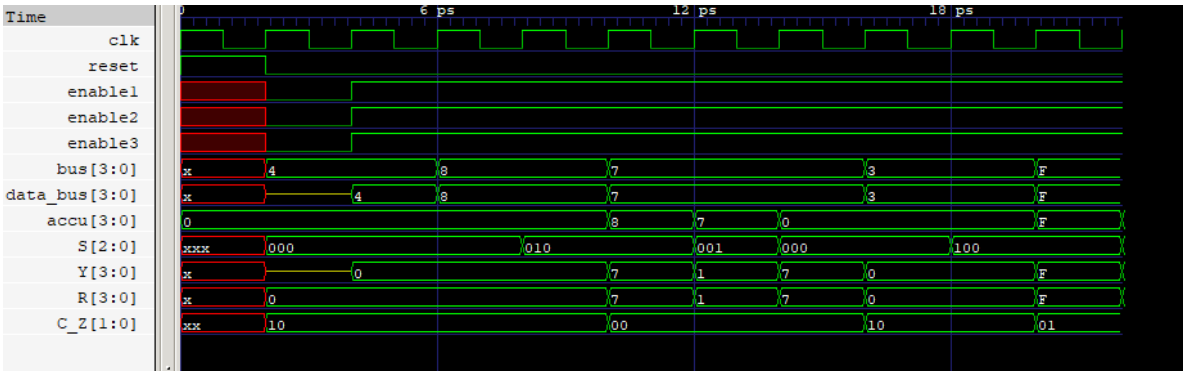
Ejercicio 2:

En este fragmento del microprocesador, como se puede observar, se implemento la ALU del microprocesador, de una ALU codificada en laboratorios previos, con la diferencia que en este se realizaban las 5 instrucciones propias del ALU del proyecto. Así mismo se le agregó un control de 2 bits que correspondía al bit de cero y carry on de las operaciones del ALU identificado como C_Z. También se implemento el acumulador como un flip flop de 4 bits y un bus driver que se implemento 2 veces el cual consistía en un buffer triestado de 4 bits codificado en anteriores laboratorios. Todo esto unido en el modulo de control del fragmento del microprocesador.



```
module ALU(input wire [3:0] A, B, input wire [2:0] S, output reg [3:0] R, output reg [1:0] C_Z);

  reg [4:0] result;
  // parametros Table 8.1 Libro
  parameter ST = 3'd0;
  parameter COMP1 = 3'd1;
  parameter LIT = 3'd2;
  parameter ADD1 = 3'd3;
  parameter NAND1 = 3'd4;

  always @(*) begin// operaciones a realizar con la ALU con un switch case
    case(S)
    ST:
      result <= A;
    COMP1:
      result <= (A - B);
    LIT:
      result <= B;
    ADD1:
      result <= (A + B);
    NAND1:
      result <= (A ~& B);
    default: result <= 5'd0;
    endcase
    assign {R, C_Z[0], C_Z[1]} = {result[3:0], result[4], ((result[3:0]==4'b0) ? 1'b1 : 1'b0) };
  end
endmodule


module Accumulator(input wire clk, reset, enable, input wire [3:0] D, output wire [3:0] Accu) ;
  FF_D1 FF1(clk, reset, enable, D[3], Accu[3]);
  FF_D1 FF2(clk, reset, enable, D[2], Accu[2]);
  FF_D1 FF3(clk, reset, enable, D[1], Accu[1]);
  FF_D1 FF4(clk, reset, enable, D[0], Accu[0]);
endmodule


module Bus_Driver(input wire enable, input wire [3:0] bus, output wire [3:0] Y);
  assign Y = enable ? bus : 4'bz;
endmodule

module Control (input wire clk, reset, enable1, enable2, enable3, input wire [3:0] bus, input wire [2:0] S,
  output wire [3:0] Y, output wire [1:0] C_Z);

  wire [3:0] accu, R, data_bus;

  Bus_Driver BD1(enable2, bus, data_bus);
  Accumulator Accu1(clk, reset, enable1, R, accu);
  ALU alu1(accu, data_bus, S, R, C_Z);
  Bus_Driver BD2(enable3, R, Y);

endmodule
```

Como se puede observar en el diagrama de timing, se muestra que mientras los enables estén en cero la salida de la data_bus se encuentra en alta impedancia al igual que Y. Por otro lado, se puede observar que respecto a la entrada del bus que representa la entrada del bus driver al ALU, dependiendo de la operación S que se realice la salida Y y R se actualiza. Así mismo la señal del C_Z es 10 cuando la salida R, Y es cero y es 01, cuando existe un overflow en la salida.



Otras evidencias (Screenshots):

- Pruebas en consola:

```
Ejercicio 1

CLK ResetC ResetF| EnableC  EnableF  Load | instr oprnd
-----------------|-----------------------|-------------
VCD info: dumpfile ej01_tb.vcd opened for output.
1     1      1   |  x         x        x   | 0000   0000
0     1      1   |  x         x        x   | 0000   0000
1     0      0   |  0         0        0   | 0000   0000
0     0      0   |  0         0        0   | 0000   0000
1     0      0   |  0         0        0   | 0000   0000
0     0      0   |  0         0        0   | 0000   0000
1     0      0   |  0         0        0   | 0000   0000
0     0      0   |  0         0        0   | 0000   0000
1     0      0   |  1         1        0   | 1110   1101
0     0      0   |  1         1        0   | 1110   1101
1     0      0   |  1         1        0   | 0001   0010
0     0      0   |  1         1        0   | 0001   0010
1     0      0   |  1         1        0   | 1111   0111
0     0      0   |  1         1        0   | 1111   0111
1     0      0   |  1         1        0   | 1111   0111
0     0      0   |  1         1        0   | 1111   0111
1     0      0   |  1         1        0   | 1000   0011
0     0      0   |  1         1        0   | 1000   0011
1     0      0   |  1         1        0   | 1100   0001
0     0      0   |  1         1        0   | 1100   0001
1     0      0   |  1         1        1   | 0011   0111
0     0      0   |  1         1        1   | 0011   0111
1     0      0   |  1         1        0   | 1010   0101
0     0      0   |  1         1        0   | 1010   0101
1     0      0   |  1         1        0   | 0000   0111
0     0      0   |  1         1        0   | 0000   0111
1     0      0   |  1         1        0   | 0110   0011
0     0      0   |  1         1        0   | 0110   0011
1     0      0   |  1         1        0   | 1110   1001
gtkwave ej01_tb.vcd ej01_tb.gtkw
```

```
Ejercicio 2

CLK Reset | Enable1  Enable2  Enable3 | Bus      S    | Y    C_Z
----------|--------------------------|--------------|----------
VCD info: dumpfile ej02_tb.vcd opened for output.
1     1   |   x         x         x   | xxxx    xxx   | xxxx  xx
0     1   |   x         x         x   | xxxx    xxx   | xxxx  xx
1     0   |   0         0         0   | 0100    000   | zzzz  10
0     0   |   0         0         0   | 0100    000   | zzzz  10
1     0   |   1         1         1   | 0100    000   | 0000  10
0     0   |   1         1         1   | 0100    000   | 0000  10
1     0   |   1         1         1   | 1000    000   | 0000  10
0     0   |   1         1         1   | 1000    000   | 0000  10
1     0   |   1         1         1   | 1000    010   | 0000  10
0     0   |   1         1         1   | 1000    010   | 0000  10
1     0   |   1         1         1   | 0111    010   | 0111  00
0     0   |   1         1         1   | 0111    010   | 0111  00
1     0   |   1         1         1   | 0111    001   | 0001  00
0     0   |   1         1         1   | 0111    001   | 0001  00
1     0   |   1         1         1   | 0111    000   | 0111  00
0     0   |   1         1         1   | 0111    000   | 0111  00
1     0   |   1         1         1   | 0011    000   | 0000  10
0     0   |   1         1         1   | 0011    000   | 0000  10
1     0   |   1         1         1   | 0011    100   | 0000  10
0     0   |   1         1         1   | 0011    100   | 0000  10
1     0   |   1         1         1   | 1111    100   | 1111  01
0     0   |   1         1         1   | 1111    100   | 1111  01
1     0   |   1         1         1   | 1111    011   | 1110  01
gtkwave ej02_tb.vcd ej02_tb.gtkw
```

- Testbech:

```
module testbench();
reg clock, resetC, resetF, enableC, enableF, load;//variables para el contador
reg [11:0] loadPC;
wire [3:0] instr;
wire [3:0] oprnd;


always//instanciacion del clock
  begin
    clock <= 1;
    #1 clock <= ~clock;// se realiza el cambio del reloj
    #1;
end


Control ctrl1(clock, resetC, resetF, enableC, enableF, load, loadPC, instr, oprnd);


initial begin// test del programa
  $display("Ejercicio 1\n");
  $display("CLK ResetC ResetF| EnableC  EnableF  Load | instr oprnd");
  $display("-----------------|-----------------------|-------------");
  $monitor("%b     %b      %b    |  %b         %b        %b  | %b   %b", clock,resetC,resetF,e
  resetC = 1; resetF = 1;
  #2 enableF = 0;  enableC = 0; resetC = 0; resetF = 0; load = 0; loadPC = 12'd0;
  #2; #2;
  #2 enableC = 1;  enableF = 1;
  #2; #2; #2; #2; #2;
  #2 load = 1; loadPC = 12'd12;
  #2 load = 0;
  #2; #2; #2;
  $finish;
end


initial begin//ejecutar GTKWave para diagramas de timing.
  $dumpfile("ej01_tb.vcd");
  $dumpvars(0, testbench);
end

endmodule
```

```
module testbench();
reg clock, reset, enable1, enable2, enable3;
reg [3:0] bus;
reg [2:0] s;
wire [3:0] Y;
wire [1:0] C_Z;


always//instanciacion del clock
  begin
    clock <= 1;
    #1 clock <= ~clock;// se realiza el cambio del reloj
    #1;
end


Control ctrl1(clock, reset, enable1, enable2, enable3, bus, s, Y, C_Z);


initial begin// test del programa
  $display("Ejercicio 2\n");
  $display("CLK Reset | Enable1  Enable2  Enable3 | Bus      S    | Y    C_Z");
  $display("----------|--------------------------|--------------|----------");
  $monitor("%b     %b   |   %b         %b         %b   | %b    %b  | %b  %b", clock,reset,enable1,enable
  reset = 1;
  #2 enable1 = 0;  enable2 = 0; enable3 = 0; reset = 0; bus = 4'd4; s = 3'd0;
  #2 enable1 = 1;  enable2 = 1; enable3 = 1;
  #2 bus = 4'd8; #2 s = 3'd2;
  #2 bus = 4'd7; #2 s = 3'd1;
  #2 s = 3'd0;
  #2 bus = 4'd3; #2 s = 3'd4;
  #2 bus = 4'd15; #2 s = 3'd3;

  $finish;
end

initial begin//ejecutar GTKWave para diagramas de timing.
  $dumpfile("ej02_tb.vcd");
  $dumpvars(0, testbench);
end
```