UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

Disciplina: ELE0517 - Sistemas Digitais

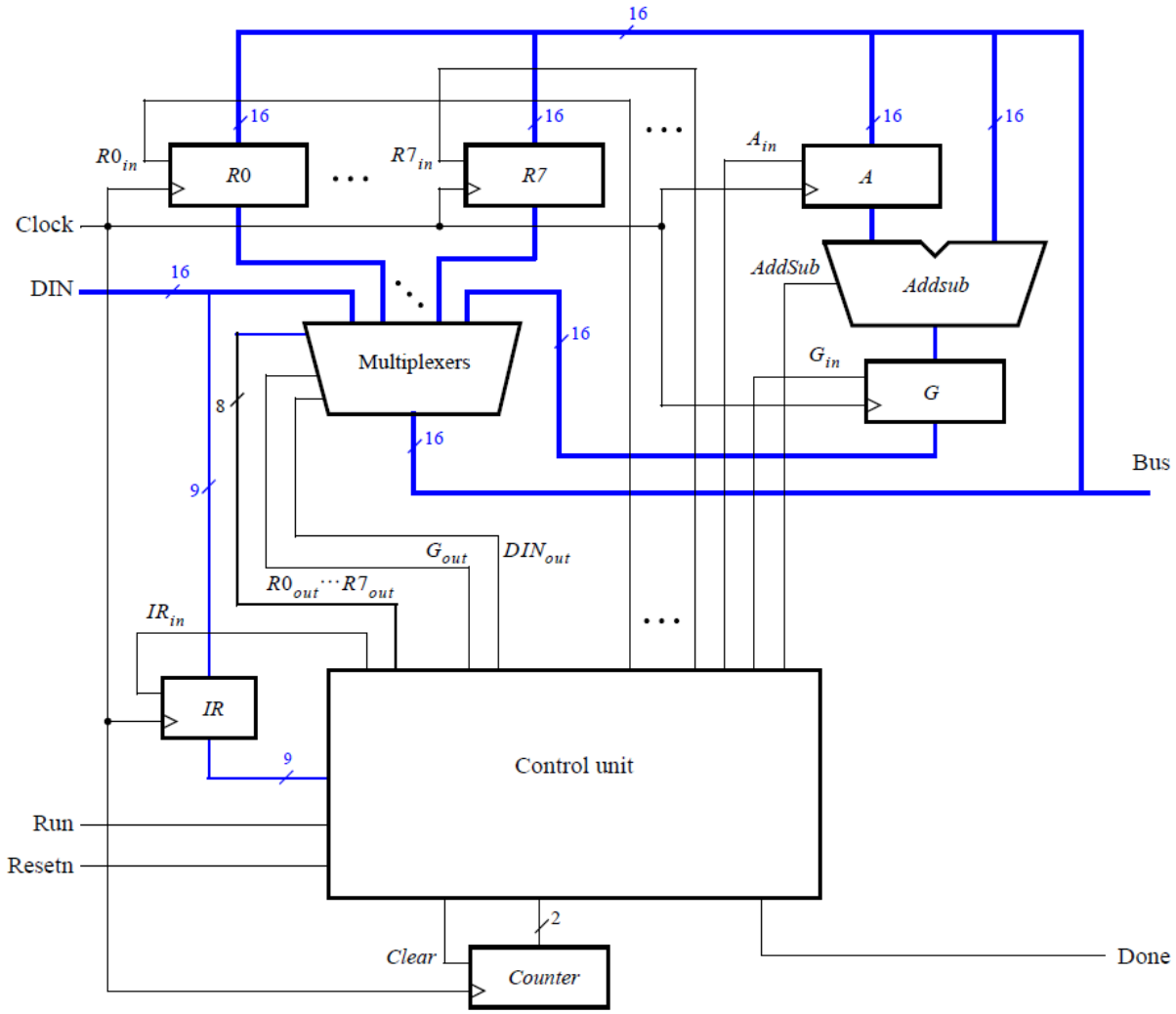
Professor: Victor Araujo Ferraz

Atividade 2

Projeto – Um processador simples

A Figura 1 mostra um sistema digital que contém vários registradores de 16 bits, um multiplexador, uma unidade somadora/subtratora, um contador e uma unidade de controle. Os dados são introduzidos neste sistema através da entrada DIN de 16 bits. Esses dados podem ser carregados através do multiplexador de 16 bits em vários registradores, como R0, . . .,R7 e A. O multiplexador também permite que os dados sejam transferidos de um registrador para outro. Os sinais de saída do multiplexador são chamados de barramento (*Bus*) na figura porque esse termo é frequentemente usado para as conexões que permitem que os dados sejam transferidos de um local para outro em um sistema.

Figura 1 - Um sistema digital.



Adição ou subtração são realizadas usando o multiplexador para primeiro colocar um número de 16 bits nos sinais do barramento e carregar esse número no registrador A. Feito isso, um segundo número de 16 bits é colocado no barramento, a unidade somadora/subtratora executa a operação necessária e o resultado é carregado no registrador G. Os dados em G podem então ser transferidos para um dos outros registros conforme necessário.

O sistema pode realizar diferentes operações em cada ciclo de clock, conforme arbitrado pela unidade de controle. Esta unidade controla quando determinados dados são colocados nos sinais do barramento e controla também qual dos registradores deve ser carregado com esses dados. Por exemplo, se a unidade de controle ativa os sinais $R0_{out}$ e A_{in}, o multiplexador colocará o conteúdo do registrador R0 no barramento e esses dados serão carregados na próxima borda de clock ativa no registrador A. Um sistema digital como esse é também chamado de processador. Ele executa operações especificadas na forma de instruções.

A Tabela 1 lista as instruções que o processador deve suportar para este exercício. A coluna da esquerda mostra o nome de uma instrução e seu operando. O significado da sintaxe RX ← [RY] é que o conteúdo do registrador RY é carregado no registrador RX. A instrução mv (move) permite que os dados sejam copiados de um registrador para outro. Para a instrução mvi (move imediatamente) a expressão RX ← D indica que a constante de 16 bits D será carregada no registrador RX.

Tabela 1 - Instruções do processador.

Operação	Função realizada
mv Rx, Ry	Rx ← [Ry]
mvi Rx, #D	Rx ← D
add Rx, Ry	Rx ← [Rx] + [Ry]
sub Rx, Ry	Rx ← [Rx] - [Ry]

Cada instrução pode ser codificada e armazenada no registrador IR usando o formato de 9 bits IIIXXXYYY, onde III representa a instrução, XXX é o registrador RX e YYY é o registrador RY. Embora apenas dois bits sejam necessários para codificar nossas quatro instruções, estamos usando três bits porque outras instruções poderão ser adicionadas ao processador. Portanto, IR deve ser conectado a nove bits da entrada DIN de 16 bits, conforme indicado na Figura 1. Para a instrução **mvi**, o campo YYY não tem significado, e os dados imediatos #D devem ser fornecido na entrada DIN de 16 bits após a palavra de instrução **mvi** ser armazenada em IR.

Algumas instruções, como uma adição ou subtração, levam mais de um ciclo de clock para serem concluídas, porque várias transferências precisam ser executadas no barramento. A unidade de controle usa o contador de dois bits mostrado na Figura 1 para permitir que ela "percorra" essas instruções. O processador inicia a execução da instrução na entrada DIN quando o sinal Run é ativado e o processador ativa a saída Done quando a instrução é concluída. A Tabela 2 indica os sinais de controle que podem ser ativados em cada etapa de tempo para implementar as instruções da Tabela 1. Observe que o único sinal de controle ativado na etapa de tempo 0 é IR_{in}, portanto, esta etapa de tempo não é mostrada na tabela.

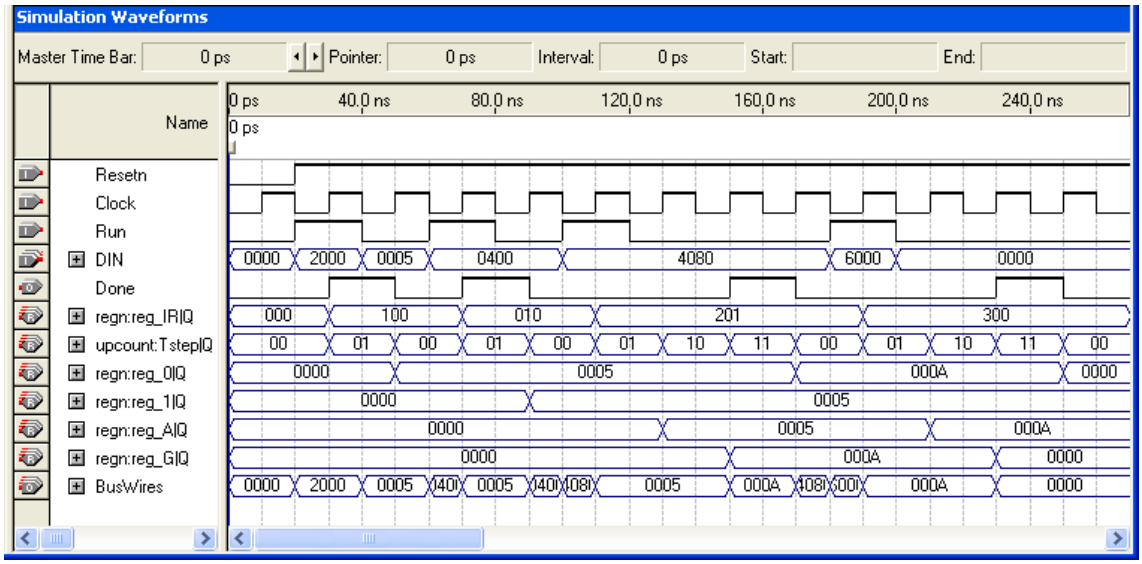
Tabela 2 - Sinais de controle ativados em cada etapa de tempo.

	T_1	T_2	T_3
(**mv**): I_0	$RY_{out}, RX_{in},$ *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}	$RY_{out}, G_{in},$ *AddSub*	$G_{out}, RX_{in},$ *Done*

Projete e implemente o processador da Figura 1 usando VHDL/Verilog da seguinte forma:

1. Crie um novo projeto Quartus II para esse exercício. Selecione como chip de destino o Cyclone II EP2C35F672C6, que é o chip FPGA na placa Altera DE2.
2. Escreva um arquivo VHDL/Verilog que implemente o processador da FIgura 1. Um esqueleto em VHDL para o código pode ser visto nos Anexos 1, 2, 3 e 4. Em Verilog pode ser visto nos Anexos 5, 6, 7 e 8.
3. Use a simulação funcional para verificar se seu código está correto. Um exemplo da saída produzida por uma simulação funcional para um circuito projetado corretamente é dado na Figura 2. Ele mostra o valor $(2000)_{16}$ sendo carregado no IR a partir do DIN no tempo de 30 ns. Este padrão representa a instrução **mvi** R0,#D, onde o valor D = 5 é carregado em R0 na borda do clock em 50 ns. A simulação mostra então a instrução **mv** R1,R0 em 90 ns, adiciona R0,R1 em 110 ns e sub R0,R0 em 190 ns. Observe que a saída da simulação mostra DIN como um número hexadecimal de 4 dígitos e mostra o conteúdo de IR como um número octal de 3 dígitos.
4. No seu projeto para a placa DE2, use as chaves SW_{15-0} para acionar a porta de entrada DIN do processador e use a chave SW_{17} para acionar a entrada Run. Além disso, use o botão KEY_0 para Resetn e KEY_1 para Clock. Conecte os sinais do barramento do processador ao $LEDR_{15-0}$ e conecte o sinal Done ao $LEDR_{17}$.
5. Faça os devidos “pin assignments”, compile o projeto e programe o FPGA.
6. Teste a funcionalidade do seu projeto alternando os interruptores e observando os LEDs. Como a entrada do clock do processador é controlada por um botão de pressão, é fácil percorrer a execução das instruções e observar o comportamento do circuito.

Figura 2 -Simulação do processador.



Anexo 1 - Esqueleto de código em VHDL para o processador.

```
LIBRARY ieee; USE ieee.std_logic_1164.all;
USE ieee.std_logic_signed.all;
ENTITY proc IS
      PORT (DIN : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
            Resetn, Clock, Run : IN STD_LOGIC;
            Done : BUFFER STD_LOGIC;
            BusWires : BUFFER STD_LOGIC_VECTOR(15 DOWNTO 0));
END proc;
ARCHITECTURE Behavior OF proc IS
. . . declare components
. . . declare signals
BEGIN
      High <= '1';
      Clear <= . . .
      Tstep: upcount PORT MAP (Clear, Clock, Tstep_Q);
      I <= IR(1 TO 3);
      decX: dec3to8 PORT MAP (IR(4 TO 6), High, Xreg);
      decY: dec3to8 PORT MAP (IR(7 TO 9), High, Yreg);
      controlsignals: PROCESS (Tstep_Q, I, Xreg, Yreg)
      BEGIN
```

```
      . . . specify initial values
      CASE Tstep_Q IS
            WHEN "00" => --store DIN in IR as long as Tstep_Q = 0
                  IRin <= '1';
            WHEN "01" => -- define signals in time step T1
                  CASE I IS
                        . . .
                  END CASE;
            WHEN "10" => -- define signals in time step T2
                  CASE I IS
                        . . .
                  END CASE;
            WHEN "11" => -- define signals in time step T3
                  CASE I IS
                        . . .
                  END CASE;
            END CASE;
END PROCESS;
reg_0: regn PORT MAP (BusWires, Rin(0), Clock, R0);
. . . instantiate other registers and the adder/subtracter unit
. . . define the bus
END Behavior;
```

Anexo 2 - Código em VHDL para o componente upcount.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_signed.all;
ENTITY upcount IS
      PORT (     Clear, Clock : IN STD_LOGIC;
                  Q : OUT STD_LOGIC_VECTOR(1 DOWNTO 0));
      END upcount;
ARCHITECTURE Behavior OF upcount IS
```

```
      SIGNAL Count : STD_LOGIC_VECTOR(1 DOWNTO 0);
BEGIN
      PROCESS (Clock)
      BEGIN
            IF (Clock'EVENT AND Clock = '1') THEN
                  IF Clear = '1' THEN
                        Count <= "00";
                  ELSE
                        Count <= Count + 1;
                  END IF;
            END IF;
END PROCESS;
Q <= Count;
END Behavior;
```

Anexo 3 - Código em VHDL para o componente dec3to8.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY dec3to8 IS
      PORT (     W : IN STD_LOGIC_VECTOR(2 DOWNTO 0);
                        En : IN STD_LOGIC;
                        Y : OUT STD_LOGIC_VECTOR(0 TO 7));
END dec3to8;
ARCHITECTURE Behavior OF dec3to8 IS
BEGIN
      PROCESS (W, En)
      BEGIN
            IF En = '1' THEN
                  CASE W IS
                        WHEN "000" => Y <= "10000000";
                        WHEN "001" => Y <= "01000000";
                        WHEN "010" => Y <= "00100000";
```

```
                    WHEN "011" => Y <= "00010000";
                    WHEN "100" => Y <= "00001000";
                    WHEN "101" => Y <= "00000100";
                    WHEN "110" => Y <= "00000010";
                    WHEN "111" => Y <= "00000001";
               END CASE;
          ELSE
               Y <= "00000000";
          END IF;
     END PROCESS;
END Behavior;
```

Anexo 4 - Código em VHDL para o componente regn.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY regn IS
     GENERIC (n : INTEGER := 16);
     PORT (     R : IN STD_LOGIC_VECTOR(n-1 DOWNTO 0);
               Rin, Clock : IN STD_LOGIC;
               Q : BUFFER STD_LOGIC_VECTOR(n-1 DOWNTO 0));
END regn;
ARCHITECTURE Behavior OF regn IS
BEGIN
     PROCESS (Clock)
     BEGIN
          IF Clock'EVENT AND Clock = '1' THEN
               IF Rin = '1' THEN
                    Q <= R;
               END IF;
          END IF;
     END PROCESS;
END Behavior;
```

Anexo 5 - Esqueleto do código em Verilog HDL para o processador.

```
module proc (DIN, Resetn, Clock, Run, Done, BusWires);
      input [15:0] DIN;
      input Resetn, Clock, Run;
      output Done;
      output [15:0] BusWires;

      . . . declare variables

      wire Clear = . . .
      upcount Tstep (Clear, Clock, Tstep_Q);
      assign I = IR[1:3];
      dec3to8 decX (IR[4:6], 1'b1, Xreg);
      dec3to8 decY (IR[7:9], 1'b1, Yreg);
      always @(Tstep_Q or I or Xreg or Yreg)
      begin
            . . . specify initial values
            case (Tstep_Q)
                  2'b00: // store DIN in IR in time step 0
                  begin
                        IRin = 1'b1;
                  end
                  2'b01: //define signals in time step 1
                        case (I)
                              . . .
                        endcase
                  2'b10: //define signals in time step 2
                        case (I)
                              . . .
                        endcase
                  2'b11: //define signals in time step 3
                        case (I)
```

```
                              . . .
                        endcase
            endcase
      end
      regn reg_0 (BusWires, Rin[0], Clock, R0);
      . . . instantiate other registers and the adder/subtracter
unit
      . . . define the bus
endmodule
```

Anexo 6 - Código em Verilog HDL para o componente upcount.

```
module upcount(Clear, Clock, Q);
      input Clear, Clock;
      output [1:0] Q;
      reg [1:0] Q;

      always @(posedge Clock)
            if (Clear)
                  Q <= 2'b0;
            else
                  Q <= Q + 1'b1;
endmodule
```

Anexo 7 - Código em Verilog HDL para o componente dec3to8.

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

```
      always @(W or En)
      begin
            if (En == 1)
                  case (W)
                        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
```

Anexo 8 - Código em Verilog HDL para o componente regn.

```
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
```