CSE 2441 Term Project: TRISC Design and Realization

Rolando Rosales

University of Texas at Arlington

CSE 2441: Digital Logic Design

Dr. Bill D. Carroll

May 2, 2022

Table of Contents

Table of Figures

Index of Tables

Introduction

Overview

The Tiny Reduced Instruction Set Computer is a simple 4-bit Processor that features an Arithmetic Logic Unit (ALU), Control Unit (CU), Accumulator (ACC), Program Counter (PC), and Random Access Memory (RAM) that can perform very basic operations between two four bit numbers (Figure 1). These operations include loading the accumulator (LDA), storing into the accumulator (STA), adding numbers using the ALU and storing them into the accumulator (ADD), incrementing the accumulator (INC), clearing the accumulator (CLR), and jumping (JMP) (Table 2).

This TRISC realization was created using *Intel Quartus Prime Software* with the Verilog HDL, and tested on a *Terasic DE10-Lite* FPGA using its various switches and keys. In order to test if all of the TRISC instructions were implemented correctly, a program to test all of the functions was loaded into the TRISC's RAM using the *DE-10's* switches and keys in order to execute a specific sequence of operations (Table 3) in hopes of receiving the required output sequence as specified by the project addendum (Table 4), displayed on the *DE10-Lite's* HEX display. The control signals from the CU were also displayed on the red and green LED's on the *DE10-Lite*.

Status

The TRISC realization in this report is fully operational and can perform all of the necessary operations required, as the program used to test the TRISC realization outputs the correct output sequence as specified in the project addendum (see Test Results). It implements all the necessary components and operations as outlined in the overview.

System Design

System-level Description and Diagrams

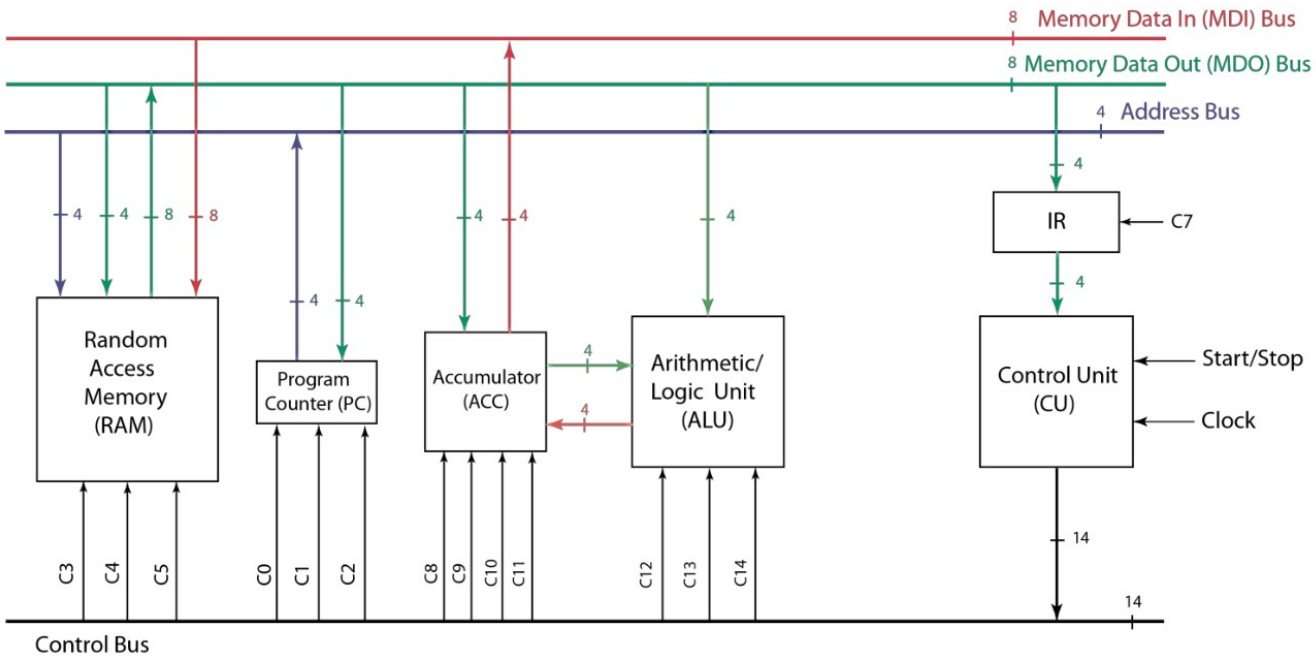

Figure 1: TRISC Design Overview

Table 1: Control Signal Definitions (Active High)

Signal	Definition
C0	*Clear* PC
C1	*Load* PC
C2	*Increment* PC
C3	*Select* MAR source (0: PC, 1: MDO)
C4	*Execute* RAM RW cycle
C5	*Enable* RAM W cycle
C7	*Load* IR
C8	*Clear* ACC
C9	*Increment* ACC
C10	*Select* ACC source (0: ALU, 1: MDO)
C11	*Load* ACC
{C12, C13}	ALU function code (00: ADD, 10: SUB, 01: AND, 11: XOR)
C14	*Load* BR

Table 2: TRISC Instructions and OP Codes

Instruction	Function	Register Transfer	OP Code
LDA	Load ACC	ACC ← (MDR)	0000
STA	Store ACC	MDR ← (ACC)	0001
ADD	Add ACC	ACC ← (ACC) + (MDR)	0010
INC	Increment ACC	ACC ← (ACC) + 1	0110
CLR	Clear ACC	ACC ← 0	0111
JMP	Jump	PC ← (MDR)	1000

TRISC is composed of five main components: the RAM, PC, ACC, ALU, and CU. Connecting all of these components are the various input signals (Table 1) that tell each component how to behave (See appendix for each signals' function). The OP Codes enable a combination of signals in order to perform instructions on the processor (Table 2). The four busses (color coded in Figure 1) allow data to move throughout the processor, and let each component communicate with each other.

Random Access Memory (RAM)

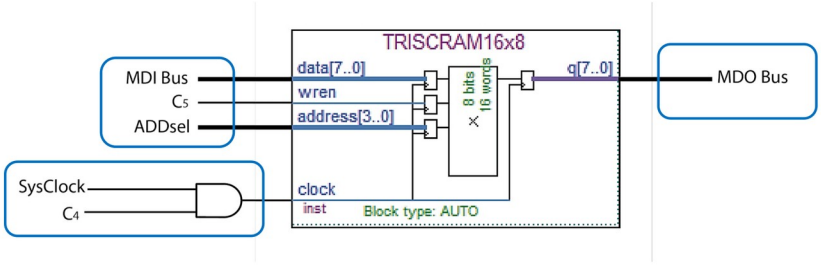

Figure 2: TRISC RAM Interface

TRISC RAM has the ability to store 8-bit values into 16 different addresses. It has inputs from the ACC through the MDI bus in order to store values into the RAM, and various other input signals that control the read or write behavior of the RAM (C4 and C5), along with one signal (C3) that selects the memory source, the PC or MDO. TRISC RAM has the MDO bus that allows data to be sent to the PC, ACC, and/or ALU.

Program Counter (PC)

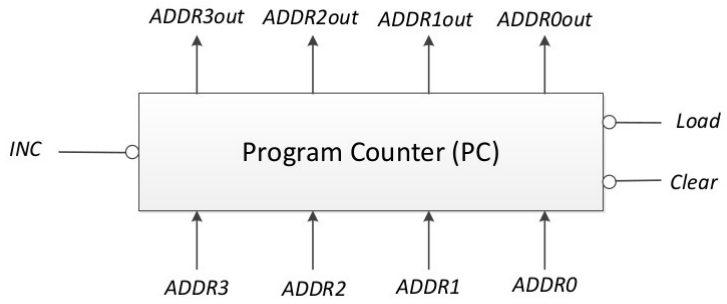

Figure 3: Program Counter Design

The program counter holds the memory address of the next TRISC instruction that is to be executed. It has a four-bit input and four-bit output, which specify addresses in memory. INC, Load, and Clear are controlled by the various signals, detailed in Table 1.

Accumulator (ACC)

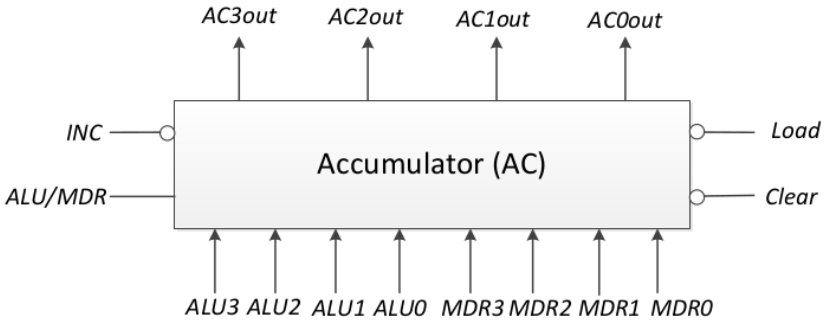

Figure 4: Accumulator Design

The accumulator holds results from operations performed by the ALU in order to either be used in another operation or to be written into the RAM. There are two inputs, one from the MDO bus which comes from the RAM, and the second being the ALU results, sent from a buffer register. The selector, INC, load, and clear signals are detailed in Table 1.

Arithmetic Logic Unit (ALU)

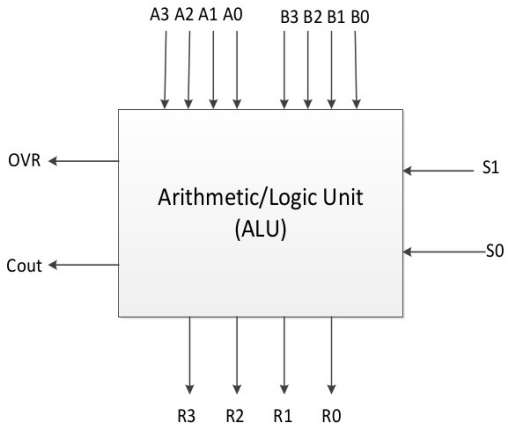

Figure 5: Arithmetic Logic Unit Design

The Arithmetic Logic Unit in this TRISC design can only perform ADD operations. It has two inputs, A and B. A comes from the MDI bus, and B from the MDO bus. The R output sends the result of the ALU's operation to the buffer register, in order to be sent to the ACC. OVR and Cout are the overflow and carry out bits, respectively. S1 and S0 select which operation is to be performed, and are controlled by the signals C12 and C13 respectively.

Buffer Register (BR)

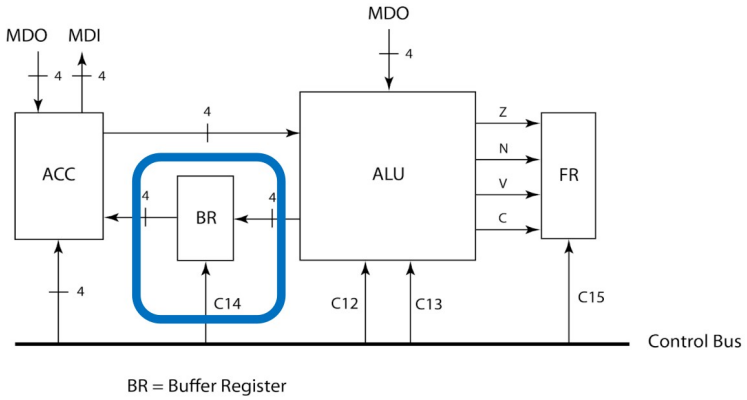

Figure 6: Buffer Register TRISC Location

Although not displayed in the TRISC design overview (Figure 1), the buffer register (BR) is a simple PI/PO register that sends the result of the ALU into the ACC. Signal C14 enables this behavior.

Input Register (IR)

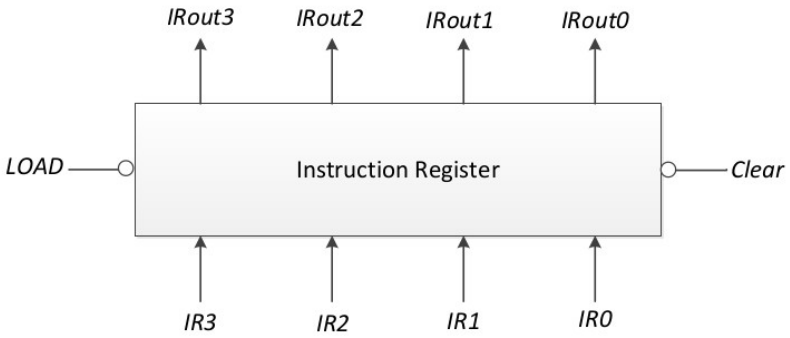

Figure 7: Instruction Register Design

The input register, similar to the BR, is a simple PI/PO register that sends the inputs - which are the OP codes for the various TRISC instructions - to the CU. The signals for LOAD and Clear are detailed in Table 2.

Verilog Code

The Verilog HDL code for each of the individual components (excluding the RAM as its given) in the TRISC processor is detailed in this section, in the same order as the last section. But first will be the code for the highest level of the TRISC processor, which instantiates all of the components.

TRISC2.v + (~/Documents/Quartus/MyCSE2441Labs/Term Project/TRISC2) - VIM

```
1  //Rolando Rosales 1001850424 - TRISC2
2
3  module TRISC2(
4     input SysClock, StartStop, Mode, ClockIn, ClearAddGen, RW, //Mode = SW9, ClockIn = K
   ey2, ClearAddGen = Key3, RW = Key5
5     input [7:0] DataIn, //DataIn = {SW7,SW6,SW5,SW4,SW3,SW2,SW1,SW0}
6     output [14:0] Cled,
7     output [6:0] hex5out, hex4out, hex3out, hex2out, hex1out, hex0out);
8
9     wire [3:0] ADDRbus, RAMadd, IRout, ALUout, BRout, AddIn, AddGen, hex5in, hex4in, hex
   3in, hex2in, hex1in, hex0in;
10    wire RAMin, RAMwrite, toggle;
11    wire [7:0] RAMdata, MDI, MDO;
12    wire C0, C1, C2, C3, C4, C7, C8, C9, C5, C10, C11, C12, C13, C14, OVR, Cout;
13
14    assign RAMadd = C3 == 0 ? MDO[3:0] : ADDRbus;
15    assign AddIn = Mode == 1'b0 ? RAMadd : AddGen;
16    assign RAMin = Mode == 1'b0 ? SysClock*C4 : ClockIn;
17    assign RAMdata = Mode == 1'b0 ? MDI : DataIn;
18    assign RAMwrite = Mode == 1'b0 ? C5 : ~RW;
```

Figure 8: TRISC Top-Level Verilog Code, Part 1

TRISC2.v + (~/Documents/Quartus/MyCSE2441Labs/Term Project/TRISC2) - VIM

```
20    assign Cled[0] = C0, Cled[1] = C1, Cled[2] = C2, Cled[3] = C3, Cled[4] = C4, Cled[5]
 = C5, Cled[7] = C7;
21    assign Cled[8] = C8, Cled[9] = C9, Cled[10] = C10, Cled[11] = C11, Cled[12] = C12, C
led[13] = C13, Cled[14] = C14;
22    assign hex5in = ADDRbus, hex4in = AddIn, hex3in = MDO[7:4], hex2in = MDO[3:0], hex1i
n = RAMdata[7:4], hex0in = RAMdata[3:0];
23
24    OnOffToggle DivideX2
25    (
26       .OnOff(ClockIn) , // input OnOff_si
27       .IN(1'b1) , // input IN_sig
28       .OUT(toggle) // output OUT_sig
29    );
30    BinUp AddressGen
31    (
32       .inc(toggle) , // input inc_sig
33       .clear(ClearAddGen) , // input clear_sig
34       .load(1'b1) , // input load_sig
35       .D(4'b0) , // input [N-1:0] D_sig
36       .Q(AddGen) // output [N-1:0] Q_sig
37    );
38    Lab11RAM RAM
39    (
40       .address ( AddIn ),
41       .clock ( ~RAMin ),
42       .data ( RAMdata ),
43       .wren ( RAMwrite ),
44       .q ( MDO )
45    );
46    BinUp PC
47    (
48       .inc(~C2) , // input  inc_sig
49       .clear(~C0) ,  // input  clear_sig
50       .load(~C1) ,   // input  load_sig
51       .D(MDO[3:0]) , // input [N-1:0] D_sig
52       .Q(ADDRbus)    // output [N-1:0] Q_sig
53    );
54    piporeg IR
55    (
56       .x(MDO[7:4]) , // input [N-1:0] x_sig
57       .CLK(~C7) , // input  CLK_sig
58       .CLR(StartStop) , // input  CLR_sig
59       .y(IRout)   // output [N-1:0] y_sig
60    );
61    accumulator ACC
62    (
63       .A(MDO[3:0]) , // input [3:0] A_sig
64       .B(BRout) , // input [3:0] B_sig
65       .INC(~C9) , // input  INC_sig
66       .LDR(~C11) ,   // input  LDR_sig
67       .CLR(~C8) , // input  CLR_sig
68       .AB(C10) ,  // input  AB_sig
69       .Z(MDI[3:0])   // output [3:0] Z_sig
                                                    64,34-38      25%
```

Figure 9: TRISC Top-Level Verilog Code, Part 2

TRISC2.v + (~/Documents/Quartus/MyCSE2441Labs/Term Project/TRISC2) - VIM

```
69       .Z(MDI[3:0])   // output [3:0] Z_sig
70    );
71    piporeg BR
72    (
73       .x(ALUout) ,   // input [N-1:0] x_sig
74       .CLK(~C14) ,   // input  CLK_sig
75       .CLR(StartStop) , // input  CLR_sig
76       .y(BRout)   // output [N-1:0] y_sig
77    );
78    alu ALU
79    (
80       .A(MDI[3:0]) , // input [3:0] A_sig
81       .B(MDO[3:0]) , // input [3:0] B_sig
82       .S0(C12) ,  // input  S0_sig
83       .S1(C13) ,  // input  S1_sig
84       .R(ALUout) ,   // output [3:0] R_sig
85       .OVR(OVR) , // output  OVR_sig
86       .Cout(Cout)    // output  Cout_sig
87    );
88    controlunit CU
89    (
90       .SysClock(~SysClock) ,  // input  SysClock_sig
91       .StartStop(StartStop) , // input  StartStop_sig
92       .SW(IRout) ,   // input [3:0] SW_sig
93       .C0(C0) ,   // output  C0_sig
94       .C1(C1) ,   // output  C1_sig
95       .C2(C2) ,   // output  C2_sig
96       .C3(C3) ,   // output  C3_sig
97       .C4(C4) ,   // output  C4_sig
98       .C7(C7) ,   // output  C7_sig
99       .C8(C8) ,   // output  C8_sig
100      .C9(C9) ,   // output  C9_sig
101      .C5(C5) ,   // output  C5_sig
102      .C10(C10) , // output  C10_sig
103      .C11(C11) , // output  C11_sig
104      .C12(C12) , // output  C12_sig
105      .C13(C13) , // output  C13_sig
106      .C14(C14) , // output  C14_sig
107   );
108   sevenseghex hex0(hex0in[3], hex0in[2], hex0in[1], hex0in[0], hex0out[0], hex0out[1],
    hex0out[2], hex0out[3], hex0out[4], hex0out[5], hex0out[6]);
109   sevenseghex hex1(hex1in[3], hex1in[2], hex1in[1], hex1in[0], hex1out[0], hex1out[1],
    hex1out[2], hex1out[3], hex1out[4], hex1out[5], hex1out[6]);
110   sevenseghex hex2(hex2in[3], hex2in[2], hex2in[1], hex2in[0], hex2out[0], hex2out[1],
    hex2out[2], hex2out[3], hex2out[4], hex2out[5], hex2out[6]);
111   sevenseghex hex3(hex3in[3], hex3in[2], hex3in[1], hex3in[0], hex3out[0], hex3out[1],
    hex3out[2], hex3out[3], hex3out[4], hex3out[5], hex3out[6]);
112   sevenseghex hex4(hex4in[3], hex4in[2], hex4in[1], hex4in[0], hex4out[0], hex4out[1],
    hex4out[2], hex4out[3], hex4out[4], hex4out[5], hex4out[6]);
113   sevenseghex hex5(hex5in[3], hex5in[2], hex5in[1], hex5in[0], hex5out[0], hex5out[1],
    hex5out[2], hex5out[3], hex5out[4], hex5out[5], hex5out[6]);
114 endmodule
~
-- VISUAL --                                    1          114,10         Bot
```

Figure 10: TRISC Top-Level Verilog Code, Part 3

BinUp.v + (~/Documents/Quartus/MyCSE2441Labs/Term Project/TRISC2) - VIM

```
 1 // N-bit binary up counter with load
 2 module BinUp #(parameter N = 4) (
 3      input inc, clear, load,
 4      input [N-1:0] D,
 5      output reg [N-1:0] Q                    // Q is defined as an N-bit register
 6 );
 7      always @ (negedge inc, negedge clear, negedge load)
 8           if (clear==1'b0) Q = 0;            // Q is loaded with all 0's
 9                else if (load==1'b0) Q = D;   // Q is loaded with D
10                   else Q = Q + 1'b1;         // Q is incremented
11 endmodule
-- INSERT --                                                    11,10         All
```

Figure 11: Program Counter Verilog Code

accumulator.v (~/Documents/Quartus/MyCSE2441Labs/Term Project/TRISC2) -

```
 1 //Rolando Rosales 1001850424 - HW8 P4 4-bit accumulator
 2
 3 module accumulator (
 4    input [3:0] A, B,
 5    input INC, LDR, CLR, AB,
 6    output [3:0] Z);
 7    wire [3:0] W;
 8    twoto1mux s0 (A, B, AB, W);
 9    upcounter s1 (INC, CLR, LDR, W, Z);
10 endmodule
-- VISUAL --                                    1         10,10         All
```

twoto1mux.v (~/Documents/Quartus/MyCSE2441Labs/Term Project/TRISC2) - V

```
 1 //Rolando Rosales 1001850424 - HW8 P1 parametrized 4-bit two-to-one mux
 2
 3 module twoto1mux #(parameter N = 4)
 4 (
 5    input [N-1:0] A, B,
 6    input AB,
 7    output [N-1:0] Y
 8 );
 9    assign Y = AB == 0 ? A : B;
10 endmodule
-- VISUAL --                                    1         10,10         All
```

upcounter.v (~/Documents/Quartus/MyCSE2441Labs/Term Project/TRISC2) - VI

```
 1 //Rolando Rosales 1001850424 - HW8 P3 parameterized 4-bit binary up counter
 2
 3 module upcounter #(parameter N = 4) (
 4    input INC, CLR, LDR,
 5    input [N-1:0] x,
 6    output reg [N-1:0] y);
 7    always @ (negedge INC, negedge CLR, negedge LDR)
 8       if (CLR == 1'b0) y <= 0;
 9          else if (LDR == 1'b0) y <= x;
10          else if (INC == 1'b0) y <= y + 1'b1;
11 endmodule
-- VISUAL --                                    1         11,10         All
```

Figure 12: Accumulator Verilog Code

alu.v (~/Documents/Quartus/MyCSE2441Labs/Term Project/TRISC2) - VIM

```
1  //Rolando Rosales 1001850424 - 4-bit ALU with two's complement ADD/SUB w/ OVR detection
   and logical AND/XOR
2
3  module alu (
4     input [3:0] A, B,
5     input S0, S1,
6     output [3:0] R, //Aout, Bout,
7     output OVR, Cout/*, S0out, S1out, a1, b1, c1, d1, e1, f1, g1, a2, b2, c2, d2, e2, f2
   , g2*/);
8     wire [3:0] X, N, S, XN, out;
9     ripplecarryadder r0 (A, B, S0, S, out[0], out[1]);
10    assign N = A & B;
11    assign X = A ^ B;
12    assign XN = S0 == 0 ? N : X;
13    assign R = S1 == 0 ? S : XN;
14    //assign Aout = A, Bout = B, S0out = S0, S1out = S1;
15    assign OVR = S1 == 1 ? 0 : out[0], Cout = S1 == 1 ? 0 : out[1];
16    //sevensegdec r1 (R[3], R[2], R[1], R[0], a1, b1, c1, d1, e1, f1, g1, a2, b2, c2, d2
   , e2, f2, g2);
17 endmodule
-- VISUAL --                                          1          17,10          All
```

ripplecarryadder.v (~/Documents/Quart...SE2441Labs/Term Project/TRISC2) - VI

```
1  //Rolando Rosales 1001850424 - Two's Complement Ripple Carry Adder/Subtractor w/ Overfl
   ow Detection
2  module ripplecarryadder (
3     input [3:0] A, B,
4     input C0,
5     output [3:0] S,
6     output OVR, Cout);
7     wire [4:1] C;
8     fulladder s0 (A[0], C0^B[0], C0, S[0], C[1]);
9     fulladder s1 (A[1], C0^B[1], C[1], S[1], C[2]);
10    fulladder s2 (A[2], C0^B[2], C[2], S[2], C[3]);
11    fulladder s3 (A[3], C0^B[3], C[3], S[3], C[4]);
12       assign Cout = C[4];
13       assign OVR = C[3]^C[4];
14 endmodule
-- VISUAL --                                          1          14,10          All
```

fulladder.v (~/Documents/Quartus/MyCSE2441Labs/Term Project/TRISC2) - VIM

```
1  //Rolando Rosales 1001850424 - Full Adder
2  module fulladder (
3     input ai, bi, cini,
4     output reg si, couti);
5        always @ (ai,bi,cini)
6           case ({ai,bi,cini})
7              3'b000: begin couti = 1'b0; si = 1'b0; end
8              3'b001: begin couti = 1'b0; si = 1'b1; end
9              3'b010: begin couti = 1'b0; si = 1'b1; end
10             3'b011: begin couti = 1'b1; si = 1'b0; end
11             3'b100: begin couti = 1'b0; si = 1'b1; end
12             3'b101: begin couti = 1'b1; si = 1'b0; end
13             3'b110: begin couti = 1'b1; si = 1'b0; end
14             3'b111: begin couti = 1'b1; si = 1'b1; end
15          endcase
16 endmodule
-- VISUAL --                                          1          16,9           All
```

Figure 13: Arithmetic Logic Unit Verilog Code

piporeg.v + (~/Documents/Quartus/MyCSE2441Labs/Term Project/TRISC2) - VIM

```
1  //Rolando Rosales 1001850424 - HW8 P2 parameterized 4-bit PI/PO reg
2
3  module piporeg #(parameter N = 4)(
4     input [N-1:0] x,
5     input CLK, CLR,
6     output reg [N-1:0] y);
7        always @ (negedge CLK, negedge CLR)
8           begin
9              if (CLR == 1'b0) y <= 0;
10                else if (CLK == 1'b0) y <= x;
11          end
12 endmodule
```

-- VISUAL -- 1 12,9 All

Figure 14: Buffer Register and Input Register Verilog Code

Test Results

The Verilog code from the previous section was all compiled using *Intel Quartus Prime* and tested on a *Terasic DE10-Lite* FPGA, and the test program correctly (see upcoming Test Results section), with all of the components, signals, and busses from Figure 1 being correctly realized in the Verilog code.

Controller Design Details

Functional Description and Diagram

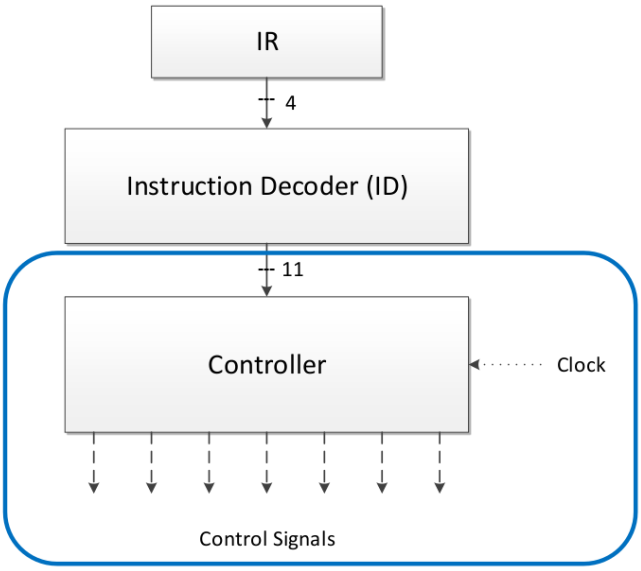

Figure 15: Control Unit Design

The control unit is made up of two components; the controller and instruction decoder (ID). The controller is what makes up most of the control unit, as it is a finite state machine (FSM) that controls what control signals are sent throughout the processor, and in what order; effectively controlling the TRISC behavior. It takes an 11-bit value from the ID, which takes 4-bit values from the IR being the OP codes for these instructions (Table 2). These OP codes tell the controller what control signals to send to the rest of the processor in order to execute instructions. The control signals are the same ones as seen in Table 1.

State Diagram

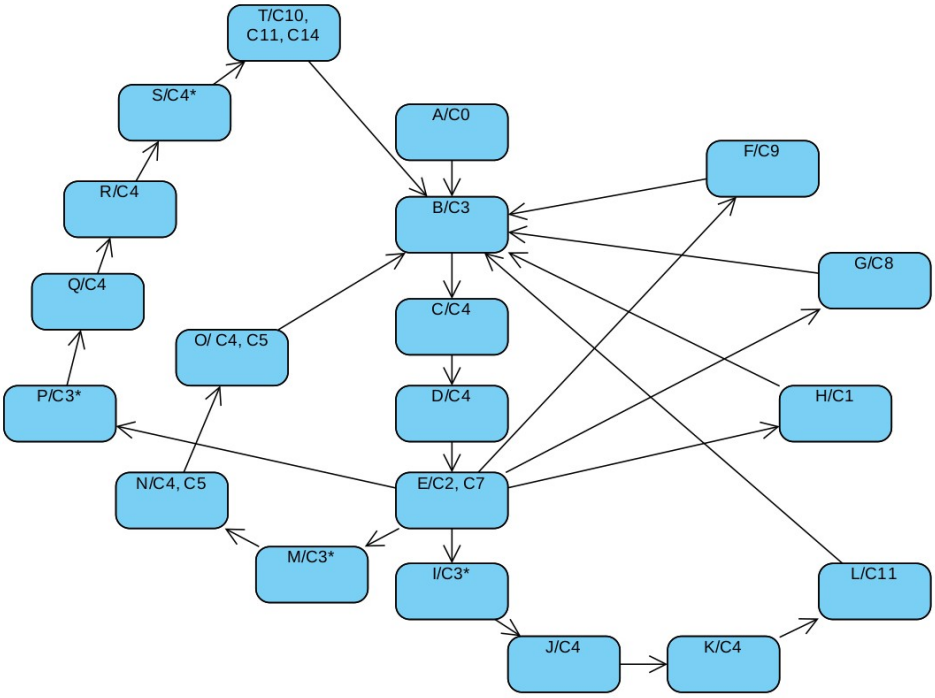

Figure 16: Controller State Diagram

Verilog Code

The Verilog code for the entire Control Unit is detailed here; starting with the controller, decoder, and then the top-level control unit.

controller.v (~/Documents/Quartus/MyCSE2441Labs/Term Project/TRISC2) - VIM

```
1  //Rolando Rosales 1001850424 - TRISC Processor Controller
2
3  module controller
4  (
5     input SysClock,StartStop,LDA,STA,ADD,SUB,XOR,INC,CLR,JMP,JPZ,JPN,HLT,
6     output reg C0,C1,C2,C3,C4,C7,C8,C9,C5,C10,C11,C12,C13,C14
7  );
8     reg [4:0] state,nextstate;
9     parameter A=5'b00000,B=5'b00001,C=5'b00010,D=5'b00011,E=5'b00100,F=5'b00101,G=5'b0011
   0,H=5'b00111,I=5'b01000,J=5'b01001,K=5'b01010,L=5'b01011,
10    M=5'b01100,N=5'b01101, O=5'b01110,P=5'b01111,Q=5'b10000,R=5'b10001,S=5'b10010,T=5'b10
   011,U=5'b10100;
11    always @ (negedge SysClock,negedge StartStop)
12       if (StartStop==1'b0) state <= A; else state <= nextstate;
13    always @ (state,INC,CLR,JMP,LDA,STA,ADD)
```

Figure 17: Controller Verilog Code, Part 1

controller.v + (~/Documents/Quartus/MyCSE2441Labs/Term Project/TRISC2) - VIM

```
13    always @ (state,INC,CLR,JMP,LDA,STA,ADD)
14       case (state)
15          A: begin {C0,C1,C2,C3,C4,C7,C8,C9,C5,C10,C11,C12,C13,C14} = 14'b10000000000000;
    nextstate = B; end //INITIALIZE
16          B: begin {C0,C1,C2,C3,C4,C7,C8,C9,C5,C10,C11,C12,C13,C14} = 14'b00010000000000;
    nextstate = C; end //FETCH
17          C: begin {C0,C1,C2,C3,C4,C7,C8,C9,C5,C10,C11,C12,C13,C14} = 14'b00011000000000;
    nextstate = D; end //FETCH
18          D: begin {C0,C1,C2,C3,C4,C7,C8,C9,C5,C10,C11,C12,C13,C14} = 14'b00011000000000;
    nextstate = E; end //FETCH
19          E: begin {C0,C1,C2,C3,C4,C7,C8,C9,C5,C10,C11,C12,C13,C14} = 14'b00110100000000;
    //DECODE
20             if (INC) nextstate = F;
21                else if (CLR) nextstate = G;
22                   else if (JMP) nextstate = H;
23                      else if (LDA) nextstate = I;
24                         else if (STA) nextstate = M;
25                            else if (ADD) nextstate = P; end
26          F: begin {C0,C1,C2,C3,C4,C7,C8,C9,C5,C10,C11,C12,C13,C14} = 14'b00000001000000;
    nextstate = B; end //INC=0110
27          G: begin {C0,C1,C2,C3,C4,C7,C8,C9,C5,C10,C11,C12,C13,C14} = 14'b00000010000000;
    nextstate = B; end //CLR=0111
28          H: begin {C0,C1,C2,C3,C4,C7,C8,C9,C5,C10,C11,C12,C13,C14} = 14'b01000000000000;
    nextstate = B; end //JMP=1000
29          I: begin {C0,C1,C2,C3,C4,C7,C8,C9,C5,C10,C11,C12,C13,C14} = 14'b00000000000000;
    nextstate = J; end //LDA=0000
30          J: begin {C0,C1,C2,C3,C4,C7,C8,C9,C5,C10,C11,C12,C13,C14} = 14'b00001000000000;
    nextstate = K; end //LDA
31          K: begin {C0,C1,C2,C3,C4,C7,C8,C9,C5,C10,C11,C12,C13,C14} = 14'b00001000000000;
    nextstate = L; end //LDA
32          L: begin {C0,C1,C2,C3,C4,C7,C8,C9,C5,C10,C11,C12,C13,C14} = 14'b00000000001000;
    nextstate = B; end //LDA
33          M: begin {C0,C1,C2,C3,C4,C7,C8,C9,C5,C10,C11,C12,C13,C14} = 14'b00000000000000;
    nextstate = N; end //STA (START)
34          N: begin {C0,C1,C2,C3,C4,C7,C8,C9,C5,C10,C11,C12,C13,C14} = 14'b00001000100000;
    nextstate = O; end //STA
35          O: begin {C0,C1,C2,C3,C4,C7,C8,C9,C5,C10,C11,C12,C13,C14} = 14'b00001000100000;
    nextstate = B; end //STA
36          P: begin {C0,C1,C2,C3,C4,C7,C8,C9,C5,C10,C11,C12,C13,C14} = 14'b00000000000000;
    nextstate = Q; end //ADD
37          Q: begin {C0,C1,C2,C3,C4,C7,C8,C9,C5,C10,C11,C12,C13,C14} = 14'b00001000000000;
    nextstate = R; end //LDR
38          R: begin {C0,C1,C2,C3,C4,C7,C8,C9,C5,C10,C11,C12,C13,C14} = 14'b00001000000000;
    nextstate = S; end //LDR
39          S: begin {C0,C1,C2,C3,C4,C7,C8,C9,C5,C10,C11,C12,C13,C14} = 14'b00000000000000;
    nextstate = T; end //ADD
40          T: begin {C0,C1,C2,C3,C4,C7,C8,C9,C5,C10,C11,C12,C13,C14} = 14'b00000000011001;
    nextstate = B; end //BUFF
41          //U: begin {C0,C1,C2,C3,C4,C7,C8,C9,C5,C10,C11,C12,C13,C14} = 14'b0000000000100
    0; nextstate = B; end //ACC
42       endcase
43 endmodule
~
-- VISUAL --                                          1          43,9          Bot
```

Figure 18: Controller Verilog Code, Part 2

fourto11decoder.v (~/Documents/Quartus/MyCSE2441Labs/Term Project/TRISC2)

```
1  //Rolando Rosales 1001850424 - 4-bit Four to Eleven Decoder
2
3  module fourto11decoder (
4     input [3:0] x,
5     output reg LDA, STA, ADD, SUB, XOR, INC, CLR, JMP, JPZ, JPN, HLT);
6     always @ (x)
7        case ({x})
8        4'b0000: {LDA, STA, ADD, SUB, XOR, INC, CLR, JMP, JPZ, JPN, HLT} = 11'b10000000000;
9        4'b0001: {LDA, STA, ADD, SUB, XOR, INC, CLR, JMP, JPZ, JPN, HLT} = 11'b01000000000;
10       4'b0010: {LDA, STA, ADD, SUB, XOR, INC, CLR, JMP, JPZ, JPN, HLT} = 11'b00100000000;
11       4'b0011: {LDA, STA, ADD, SUB, XOR, INC, CLR, JMP, JPZ, JPN, HLT} = 11'b00010000000;
12       4'b0100: {LDA, STA, ADD, SUB, XOR, INC, CLR, JMP, JPZ, JPN, HLT} = 11'b00001000000;
13       4'b0110: {LDA, STA, ADD, SUB, XOR, INC, CLR, JMP, JPZ, JPN, HLT} = 11'b00000100000;
14       4'b0111: {LDA, STA, ADD, SUB, XOR, INC, CLR, JMP, JPZ, JPN, HLT} = 11'b00000010000;
15       4'b1000: {LDA, STA, ADD, SUB, XOR, INC, CLR, JMP, JPZ, JPN, HLT} = 11'b00000001000;
16       4'b1100: {LDA, STA, ADD, SUB, XOR, INC, CLR, JMP, JPZ, JPN, HLT} = 11'b00000000100;
17       4'b1001: {LDA, STA, ADD, SUB, XOR, INC, CLR, JMP, JPZ, JPN, HLT} = 11'b00000000010;
18       4'b1111: {LDA, STA, ADD, SUB, XOR, INC, CLR, JMP, JPZ, JPN, HLT} = 11'b00000000001;
19    endcase
20 endmodule
                                                                    1,1           All
```

Figure 19: Instruction Decoder Verilog Code

controlunit.v (~/Documents/Quartus/MyCSE2441Labs/Term Project/TRISC2) - VIM

```
1  //Rolando Rosales 1001850424 - TRISC Control Unit
2
3  module controlunit(
4     input SysClock, StartStop,
5     input [3:0] SW,
6     output C0, C1, C2, C3, C4, C7, C8, C9, C5, C10, C11, C12, C13, C14/*,
7     output [3:0] SWled,
8     output [1:0] Bled*/);
9     wire LDA, STA, ADD, SUB, XOR, INC, CLR, JMP, JPZ, JPN, HLT;
10    fourto11decoder p0 (SW, LDA, STA, ADD, SUB, XOR, INC, CLR, JMP, JPZ, JPN, HLT);
11    controller p1 (SysClock, StartStop, LDA, STA, ADD, SUB, XOR, INC, CLR, JMP, JPZ, JPN,
   HLT, C0, C1, C2, C3, C4, C7, C8, C9, C5, C10, C11, C12, C13, C14);
12    //assign SWled = SW;
13    //assign Bled[0] = ~StartStop, Bled[1] = ~SysClock;
14 endmodule
                                                                   14,10          All
```

Figure 20: Control Unit Verilog Code

Test Results

Summary

Table 3: Program Load

ADDR	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Input	0F	61	62	1E	74	0E	66	89	88	69	2E	7B	6C	88	EE	FF

Table 4: Program Results

Position	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Output	F	0	1	1	0	1	2	2	1	3	4	0	1	1	NA	NA

In order to test the TRISC realization, the RAM was loaded with the program in Table 3. This program would test the various instructions of the processor. Once loaded, the system clock should cycle repeatedly to run the processor's various instructions from the loaded program, and keep incrementing the program counter until the program ended. The results from the program should be displayed on the *DE10-Lite's* HEX display, and should match that of Table 4's.

After loading the program onto the TRISC RAM it by repeatedly cycling the system clock, the HEX display displayed the correct values for each position Table 4, nearly in the same sequence however skipping past position 8 and returning to it at the end, as was mentioned in the project addendum. The correct control signals were also displayed for each instruction on the red and green LED's on the *DE10-Lite*, matching that of the controller's FSM (Figure 16).

Photos or Video of Program Execution

The TRISC realization was demonstrated to Dr. Caroll in-person in the lab, and so neither a photo or video of the program execution was necessary.

Conclusion

Resolution of design

All of the components of the TRISC realization in this project report work as they should, as the program to test the instructions (detailed in the Test Results section) ran correctly showing that every instruction seems to execute correctly when running the program. It can be said that this TRISC realization is completely functional, and that its design presents no issues.

Lessons learned

This lab taught me a lot about how the various different combinational and sequential circuits that we have been making throughout this semester can be used to build a CPU. Although the design and functionality of the TRISC was quite simple, it was nevertheless a huge learning experience as it showed me how the different components worked both individually and together. Along with designing TRISC, I also learned a lot about how to use the Verilog HDL, and saw that it is much easier to create these designs in Verilog than a solderless breadboard or wiring up the diagrams using Quartus Prime. And with a project that spanned almost a third the semester, it reinforced how helpful it is to have all of my documents organized and ready to go for projects such as these. I would not have been able to submit this report on the early completion date had I needed to scramble together all of my files to make them work in Quartus, it was effectively drag and drop for me. I very much enjoyed this project.