

Class:	CPE300L - Digital System Architecture and Design Lab	Semester:	Fall 2025
Points		Document author:	Darryll Mckoy
		Author's email:	mckoyd1@unlv.nevada.edu
		Document topic:	Postlab 7
Instructor's comments:			

1. Lab submissions

This lab took around 7 hours

2. Marked Figure 1:

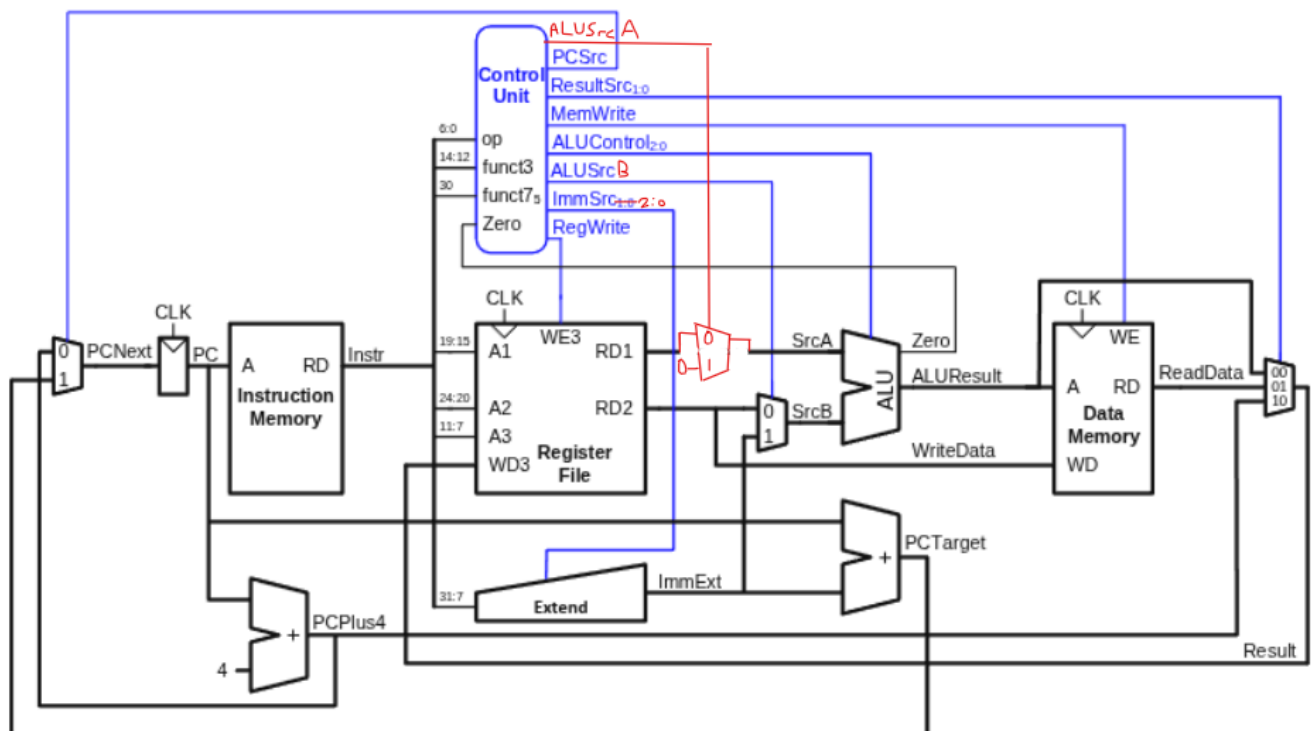


Figure 1: RISC-V single-cycle processor

3. Marked Figure 2:

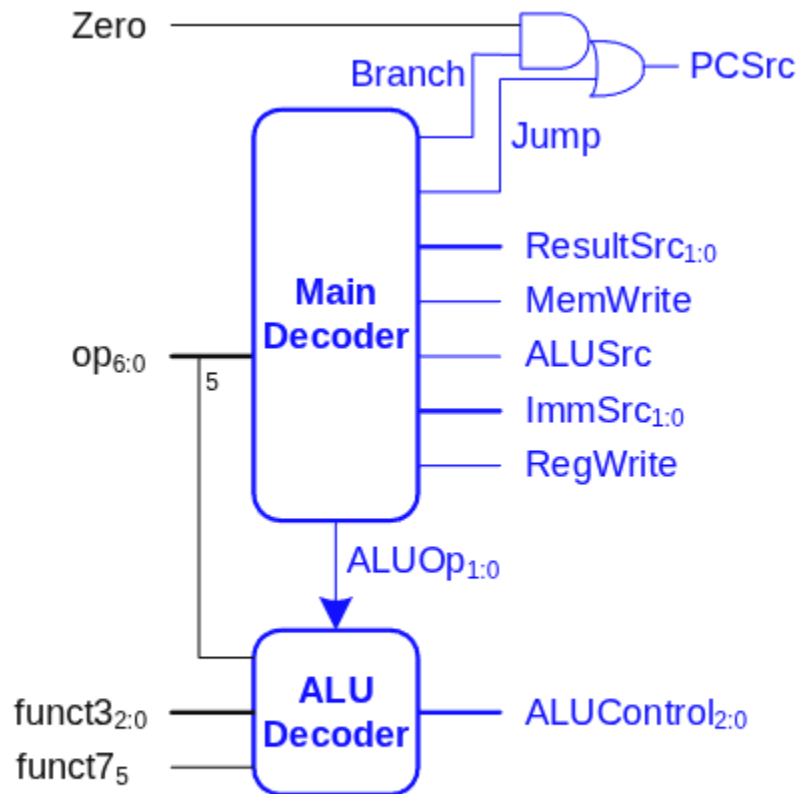


Figure 2: RISC-V single-cycle processor control unit

Marked Figure 3:

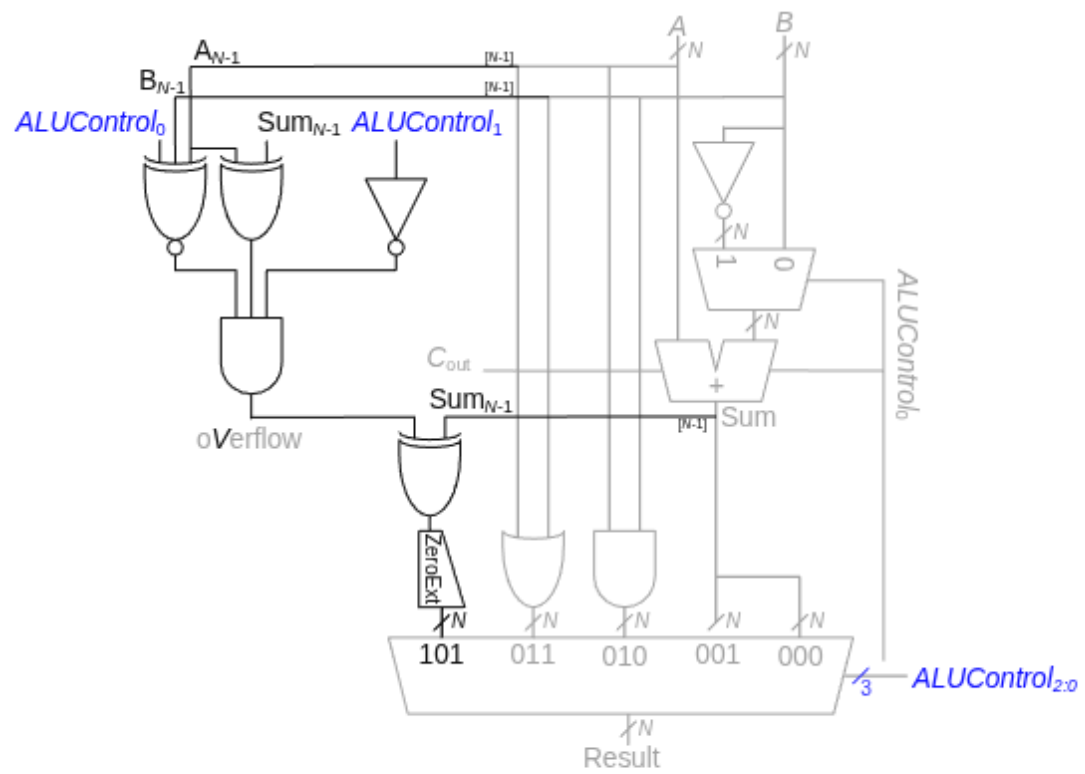


Figure 3: ALU

4. Amended Tables

Main Decoder Truth Table:

Instruction	Opcode	RegWrite	ImmSrc	ALUSrcA	ALUSrcB	MemWrite	ResultSrc	Branch	ALUOp	Jump
lw	0000011	1	000	0	1	0	01	0	00	0
sw	0100011	0	001	0	1	1	00	0	00	0
R-type	0110011	1	xxx	0	0	0	00	0	10	0
beq	1100011	0	010	0	0	0	00	1	01	0
I-type ALU	0010011	1	000	0	1	0	00	0	10	0
jal	1101111	1	011	0	0	0	10	0	00	1
lui	0110111	1	100	1	1	0	00	0	00	0

ALU Decoder Truth Table:

ALUOp _{1:0}	funct _{32:0}	{op ₅ , funct ₇₅ }	ALUControl _{2:0}	Operation
00	x	x	000	Add
01	x	x	001	Subtract
10	000	00, 01, 10	000	Add
	000	11	001	Subtract
	010	x	101	SLT
	110	x	011	OR
	111	x	010	AND

	100	x	100	XOR
--	-----	---	-----	-----

ImmSrc Truth Table:

ImmSrc	ImmExt	Type	Description
000	{{20{Instr[31]}}, Instr[31:20]}	I	12-bit signed immediate
001	{{20{Instr[31]}}, Instr[31:25], Instr[11:7]}	S	12-bit signed immediate
010	{{20{Instr[31]}}, Instr[7], Instr[30:25], Instr[11:8], 1'b0}	B	12-bit signed immediate
011	{{12{Instr[31]}}, Instr[19:12], Instr[20], Instr[30:21], 1'b0}	J	12-bit signed immediate
100	{Instr[31:12], 12'b0}	U	20-bit signed immediate

5. Amended System Verilog Code For Lui:

- Made 4 modifications to riscvsingle module, adding ALUSrcA/B, increased ImmSrc to 3 bits.

```
module riscvsingle(input  logic      clk, reset,
                  output logic [31:0] PC,
                  input  logic [31:0] Instr,
                  output logic      MemWrite,
                  output logic [31:0] ALUResult, WriteData,
                  input  logic [31:0] ReadData);

    logic      ALUSrcA, ALUSrcB, RegWrite, Jump, Zero;
    logic [1:0] ResultSrc;
    logic [2:0] ImmSrc;
    logic [2:0] ALUControl;

    controller c(Instr[6:0], Instr[14:12], Instr[30], Zero,
                ResultSrc, MemWrite, PCSrc,
                ALUSrcA, ALUSrcB, RegWrite, Jump,
                ImmSrc, ALUControl);
    datapath dp(clk, reset, ResultSrc, PCSrc,
                ALUSrcA, ALUSrcB, RegWrite,
                ImmSrc, ALUControl,
                Zero, PC, Instr,
                ALUResult, WriteData, ReadData);
endmodule
```

- Increased ImmSrc to 3 bits and added ALUsrcA/B within controller module.

```

module controller(input logic [6:0] op,
                  input logic [2:0] funct3,
                  input logic      funct7b5,
                  input logic      Zero,
                  output logic [1:0] ResultSrc,
                  output logic      MemWrite,
                  output logic      PCSrc, ALUSrcA, ALUSrcB,
                  output logic      RegWrite, Jump,
                  output logic [2:0] ImmSrc,
                  output logic [2:0] ALUControl);

    logic [1:0] ALUOp;
    logic      Branch;

    maindec md(op, ResultSrc, MemWrite, Branch,
               ALUSrcA, ALUSrcB, RegWrite, Jump, ImmSrc, ALUOp);
    aludec ad(op[5], funct3, funct7b5, ALUOp, ALUControl);

    assign PCSrc = Branch & Zero | Jump;
endmodule

```

- Increased ImmSrc to 3 bits, increased and updated controls to 13 bits, and updated all control values with addition of lui. Added ALUSrcA/B to maindec module.

```

module maindec(input  logic [6:0] op,
               output logic [1:0] ResultSrc,
               output logic      MemWrite,
               output logic      Branch, ALUSrcA, ALUSrcB,
               output logic      RegWrite, Jump,
               output logic [2:0] ImmSrc,
               output logic [1:0] ALUOp);

    logic [12:0] controls;

    assign {RegWrite, ImmSrc, ALUSrcA, ALUSrcB, MemWrite,
           ResultSrc, Branch, ALUOp, Jump} = controls;

    always_comb
    case(op)
    // RegWrite_ImmSrc_ALUSrcA_ALUSrcB_MemWrite_ResultSrc_Branch_
    7'b0000011: controls = 13'b1_000_0_1_0_01_0_00_0; // lw
    7'b0100011: controls = 13'b0_001_0_1_1_00_0_00_0; // sw
    7'b0110011: controls = 13'b1_000_0_0_0_00_0_10_0; // R-type
    7'b1100011: controls = 13'b0_010_0_0_0_00_1_01_0; // beq
    7'b0010011: controls = 13'b1_000_0_1_0_00_0_10_0; // I-type
    7'b1101111: controls = 13'b1_011_0_0_0_10_0_00_1; // jal
    7'b0110111: controls = 13'b1_100_1_1_0_00_0_00_0; // lui
    default:    controls = 13'bx_000_x_x_x_00_x_00_x; // non-im
    endcase
endmodule

```

- Modified datapath to include ALUSrcA/B, increased ImmSrc to 3 bits, added rd1 logic, added mux for lui.

```

module datapath(input logic clk, reset,
                input logic [1:0] ResultSrc,
                input logic PCSrc, ALUSrcB, ALUSrcA,
                input logic RegWrite,
                input logic [2:0] ImmSrc,
                input logic [2:0] ALUControl,
                output logic Zero,
                output logic [31:0] PC,
                input logic [31:0] Instr,
                output logic [31:0] ALUResult, WriteData,
                input logic [31:0] ReadData);

    logic [31:0] PCNext, PCPlus4, PCTarget, rd1;
    logic [31:0] ImmExt;
    logic [31:0] SrcA, SrcB;
    logic [31:0] Result;

    // next PC logic
    flopr #(32) pcreg(clk, reset, PCNext, PC);
    adder      pcadd4(PC, 32'd4, PCPlus4);
    adder      pcaddbranch(PC, ImmExt, PCTarget);
    mux2 #(32) pcmux(PCPlus4, PCTarget, PCSrc, PCNext);

    // register file logic
    regfile    rf(clk, RegWrite, Instr[19:15], Instr[24:20],
                Instr[11:7], Result, rd1, WriteData);
    extend     ext(Instr[31:7], ImmSrc, ImmExt);

    // ALU logic
    mux2 #(32) srcamux(rd1, 32'b0, ALUSrcA, SrcA);
    mux2 #(32) srcbmux(WriteData, ImmExt, ALUSrcA, SrcB);
    alu        alu(SrcA, SrcB, ALUControl, ALUResult, Zero);
    mux3 #(32) resultmux(ALUResult, ReadData, PCPlus4, ResultSrc, Result);
endmodule

```

- Added lui immext, increased immsrc to 3 bits.

```

module extend(input logic [31:7] instr,
              input logic [2:0] immsrc,
              output logic [31:0] immext);

    always_comb
    case(immsrc)
        // I-type
        3'b000: immext = {{20{instr[31]}}, instr[31:20]};
        // S-type (stores)
        3'b001: immext = {{20{instr[31]}}, instr[31:25], instr[11:7]};
        // B-type (branches)
        3'b010: immext = {{20{instr[31]}}, instr[7], instr[30:25], instr[11:8], 1'b0};
        // J-type (jal)
        3'b011: immext = {{12{instr[31]}}, instr[19:12], instr[20], instr[30:21], 1'b0};

        3'b100: immext = {instr[31:12], 12'b0};
        // U-type (lui)
        default: immext = 32'bx; // undefined
    endcase
endmodule

```

Amended System Verilog Code For XOR:

- Added XOR function to aludec module.

```
module aludec(input logic opb5,
              input logic [2:0] funct3,
              input logic funct7b5,
              input logic [1:0] ALUOp,
              output logic [2:0] ALUControl);

    logic RtypeSub;
    assign RtypeSub = funct7b5 & opb5; // TRUE for R-type subtract instruction

    always_comb
    case(ALUOp)
        2'b00: ALUControl = 3'b000; // addition
        2'b01: ALUControl = 3'b001; // subtraction
        default: case(funct3) // R-type or I-type ALU
            3'b000: if (RtypeSub)
                ALUControl = 3'b001; // sub
            else
                ALUControl = 3'b000; // add, addi
            3'b010: ALUControl = 3'b101; // slt, slti
            3'b110: ALUControl = 3'b011; // or, ori
            3'b100: ALUControl = 3'b100; // xor
            3'b111: ALUControl = 3'b010; // and, andi
            default: ALUControl = 3'bxxx; // ???
        endcase
    endcase
endmodule
```

Ammended System Verilog Code for DE2 Board

- Modified top module to include 7-seg decoder to display instruction, moved PC/Instr to output logic.

```
module top(input logic      clk, reset,
           output logic [31:0] WriteData, DataAdr, PC, Instr,
           output logic      MemWrite,
           output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7);

    logic [31:0] ReadData;

    sevensseg_decoder sevensseg_decoder0(Instr[31:28], HEX7);
    sevensseg_decoder sevensseg_decoder1(Instr[27:24], HEX6);
    sevensseg_decoder sevensseg_decoder2(Instr[23:20], HEX5);
    sevensseg_decoder sevensseg_decoder3(Instr[19:16], HEX4);
    sevensseg_decoder sevensseg_decoder4(Instr[15:12], HEX3);
    sevensseg_decoder sevensseg_decoder5(Instr[11:8], HEX2);
    sevensseg_decoder sevensseg_decoder6(Instr[7:4], HEX1);
    sevensseg_decoder sevensseg_decoder7(Instr[3:0], HEX0);

    // instantiate processor and memories
    riscvsingle rvsingle(clk, reset, PC, Instr, MemWrite, DataAdr,
        WriteData, ReadData);
    imem imem(PC, Instr);
    dmem dmem(clk, MemWrite, DataAdr, WriteData, ReadData);
endmodule
```

```

module sevenseg_decoder(input  logic [3:0] data,
                        | | | | | | | | | | output logic [6:0] segments);

    always_comb
    case (data)          //Sg - Sa
        4'h0: segments = 7'b1000000;
        4'h1: segments = 7'b1111001;
        4'h2: segments = 7'b0100100;
        4'h3: segments = 7'b0110000;
        4'h4: segments = 7'b0011001;
        4'h5: segments = 7'b0010010;
        4'h6: segments = 7'b0000010;
        4'h7: segments = 7'b1111000;
        4'h8: segments = 7'b0000000;
        4'h9: segments = 7'b0011000;
        4'hA: segments = 7'b0001000;
        4'hB: segments = 7'b0000011;
        4'hC: segments = 7'b0100111;
        4'hD: segments = 7'b0100001;
        4'hE: segments = 7'b0000110;
        4'hF: segments = 7'b0001110;
    endcase

endmodule

```

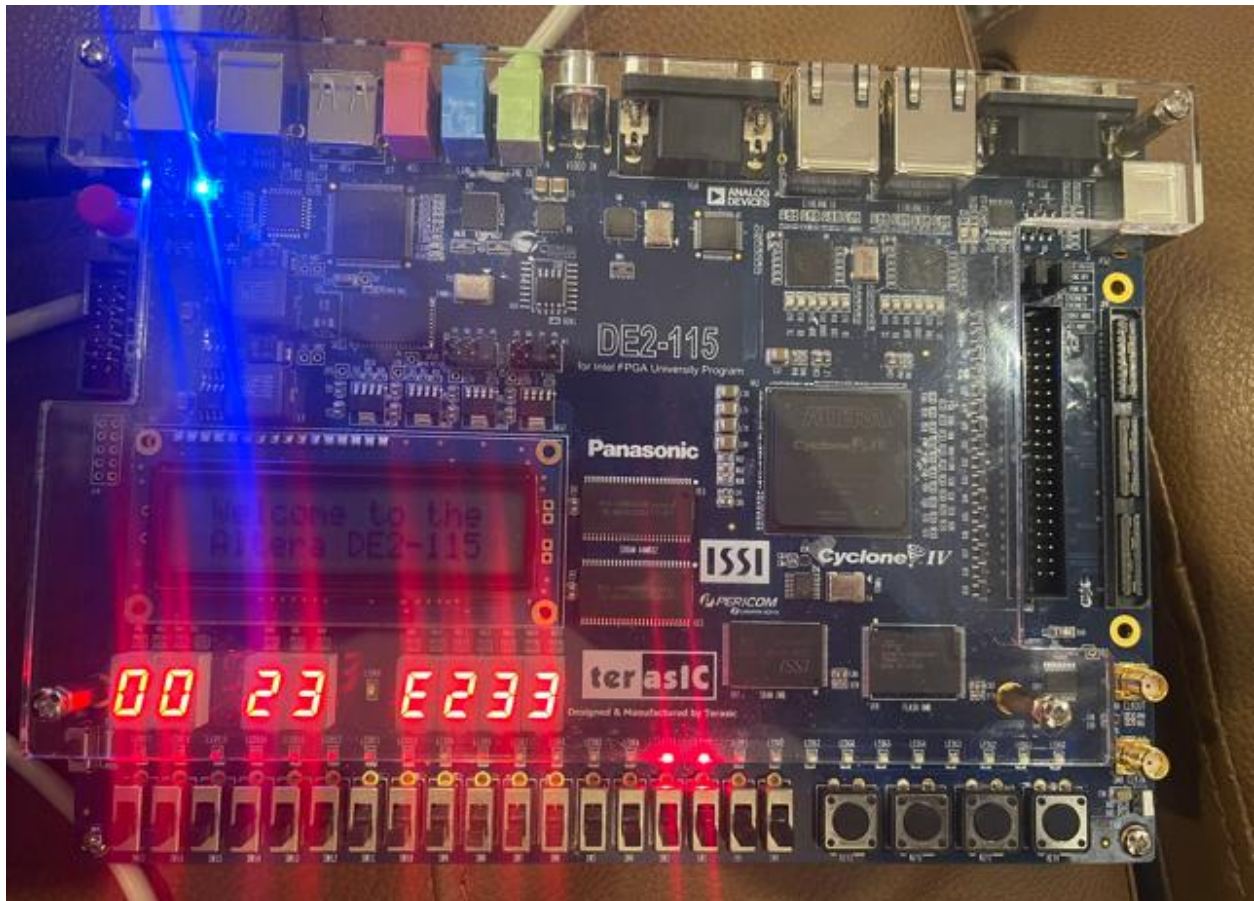
6. Modified Assembly Instruction:

#	RISC-V Assembly	Description	Address	Machine Code
main:	addi x2, x0, 5	# x2 = 5	0	00500113
	addi x3, x0, 12	# x3 = 12	4	00C00193
	addi x7, x3, -9	# x7 = (12 - 9) = 3	8	FF718393
	or x4, x7, x2	# x4 = (3 OR 5) = 7	C	0023E233
	xor x4, x3, x2	# x4 = (12 ^ 5) = 9	10	0021c233
	lui x3, 0xABCDE	# x3 = 0xABCDE	14	abcde1b7
	and x5, x3, x4	# x5 = (12 AND 7) = 4	18	0041F2B3
	add x5, x5, x4	# x5 = (4 + 7) = 11	1C	004282B3
	beq x5, x7, end	# shouldn't be taken	20	02728863
	slt x4, x3, x4	# x4 = (12 < 7) = 0	24	0041A233
	beq x4, x0, around	# should be taken	28	00020463
	addi x5, x0, 0	# shouldn't happen	2C	00000293
around:	slt x4, x7, x2	# x4 = (3 < 5) = 1	30	0023A233
	add x7, x4, x5	# x7 = (1 + 11) = 12	34	005203B3
	sub x7, x7, x2	# x7 = (12 - 5) = 7	38	402383B3
	sw x7, 84(x3)	# [96] = 7	3C	0471AA23
	lw x2, 96(x0)	# x2 = [96] = 7	40	06002103
	add x9, x2, x5	# x9 = (7 + 11) = 18	44	005104B3
	jal x3, end	# jump to end, x3 = 0x44	48	008001EF
	addi x2, x0, 1	# shouldn't happen	4C	00100113
end:	add x2, x2, x9	# x2 = (7 + 18) = 25	50	00910133
	sw x2, 0x20(x3)	# mem[100] = 25	54	0221A023
done:	beq x2, x2, done	# infinite loop	58	00210063

7. Simulation Waveform:

	Msgs							
clk	0							
reset	0							
PC	00000054	00000000		00000004	00000008	0000000c	00000010	00000014
Instr	0221a023	00500113		00c00193	ff718393	0023e233	0021c233	abcde1b7
SrcA	00000000	00000000		0000000c	00000003	0000000c	00000000	00000000
SrcB	00000020	00000005		0000000c	ffffff77	00000005		abcde000
ALUResult	00000020	00000005		0000000c	00000003	00000007	00000009	abcde000
DataAdr	00000020	00000005		0000000c	00000003	00000007	00000009	abcde000
WriteData	xxxxxxxx					00000005		
MemWrite	1							

8. DE2 Board Screenshot and Demo



Demo Link: https://youtu.be/LvzOv1TrRXI?si=6XC6et_URz_6zXoA