

29/04/2019

EE446 LABORATORY

EXPERIMENT 4

PRELIMINARY REPORT

Muttalip Caner TOL

2031466

Tuesday Afternoon

## 1.2. ISA Configuration

- Memory Instructions:
  - LDM Rd, [imm]
  - LDD Rd, imm
  - STD Rd, [imm]
- Arithmetic Instructions:
  - ADD Rd, Rn, Rm
  - SUB Rd, Rn
  - ADDI Rd, Rn, [imm]
  - SUBI Rd, Rn, [imm]
  - SUBI Rd, Rn, [imm]
- Logic Instructions:
  - AND Rd, Rn, Rm
  - XOR Rd, Rn, Rm
  - CLR Rd
  - ORR Rd, Rn, Rm
- Shift Instructions:
  - LSL Rd, Rn
  - LSR Rd, Rn
  - ROR Rd, Rn
  - ROL Rd, Rn
  - ASR Rd, Rn
- Branch Instructions:
  - BUN [imm]
  - BLD [imm]
  - BLI [imm]
  - BEQ [imm]
  - BNE [imm]
  - BCS [imm]
  - BCC [imm]

We have 8 registers in total which are R0, R1, R2, R3, R4, R5, LR and PC.

ADD, SUB, XOR,CLR	Op	Cmd	Rd	Rn	Rm	00
ADDI, SUBI	Op	Cmd	Rd	Rn	Memory Address	
SHIFT	Op	Cmd	Rd	Rd	00000	
LDD	Op	Cmd	Rd	Data		
LDM/ STD	Op	Cmd	Rd	Memory Address		
Bxx	Op	Cmd	Branch Address			
	2 bit	3 bit	3 bit	3 bit	3 bit	2 bit
				Data length: 8 bit		
	Total instruction length: 16 bit					

Arithmetic and logic instructions use Register file, ALU and IDM (Instruction / Data Memory).

Shift instructions use Shifter, Register File and IDM.

Load/Store type of instructions use Register file and IDM.

	Inst.	Cmd	Op
Arithmetic & Logic Inst.	ADD	000	00
	SUB	001	
	ADDI	010	
	SUBI	011	
	AND	100	
	OR	101	
	XOR	110	
	CLR	111	
Shift Inst.	ROL	000	01
	ROR	001	
	LSL	010	
	ASR	011	
	LSR	100	
Branch Inst.	BUN	000	10
	BLD	001	
	BLI	010	
	BEQ	011	
	BNE	100	
	BCS	101	
	BCC	110	
Memory Inst.	LDD	000	11
	LDM	001	
	STD	010	

Shifter module:

```

1  module SHIFT (data, control,out);
2      input [2:0]control;
3      input signed [7:0]data;
4      output [7:0]out;
5      reg [7:0]out;
6
7      always @(*)
8      begin
9          if(control[2:0]==3'b000)
10             out = {data[6:0],data[7]} ;
11
12             else if(control[2:0]==3'b001)
13                 out = {data[0],data[7:1]} ;
14
15             else if(control[2:0]==3'b010)
16                 out = data << 1 ;
17
18             else if(control[2:0]==3'b011)
19                 out = data >>> 1 ;
20
21             else if (control[2:0]==3'b100)
22                 out = data >> 1;
23         end
24     endmodule

```

## Arithmetic Logic Unit

Date: April 29, 2019

ALU.v

Project: de0nano\_embedding

```
1  module ALU #(parameter W=8)(ALUCtrl,A,B,out,co,ovf,z,n);
2      input [(W-1):0] A,B;
3      input [2:0] ALUCtrl;
4      output [(W-1):0] out;
5      output reg co,ovf,z,n;
6      reg [(W-1):0] out;
7      reg [15:0]temp;
8      reg [7:0] multiplier_copy;
9      reg [15:0] multiplicand_copy;
10     reg negative_output;
11     reg [15:0] product_temp;
12
13     reg [5:0] bit;
14
15     initial bit = 0;
16     initial negative_output = 0;
17
18
19     always @(*)
20     begin
21         case(ALUCtrl)
22             0: // Add
23             begin
24                 {co,out} = A + B ;
25                 z=(out == 0) ? 1 : 0;
26                 if ((A[W-1] == 0 && B[W-1] == 0 && out[W-1] == 1) || (A[W-1] == 1 && B[W-1] == 1
&& out[W-1] == 0)) begin
27                     ovf = 1;
28                     n=0;
29                 end else begin
30                     ovf = 0;
31                     if (out[W-1]==1) begin
32                         n=1;
33                     end else begin
34                         n=0;
35                     end
36                 end
37             end
38             1: // SubAB
39             begin
40                 {co,out} = A + (~B + 1) ; // (~B + 1)
41                 z=(out == 0) ? 1 : 0;
42                 if ((A[W-1] == 1 && B[W-1] == 0 && out[W-1] == 0) || (A[W-1] == 0 && B[W-1] ==
1 && out[W-1] == 1)) begin
43                     ovf = 1;
44                     n=0;
45                 end else begin
46                     ovf = 0;
47                     if (out[W-1]==1) begin
48                         n=1;
49                     end else begin
50                         n=0;
51                     end
52                 end
53             end
54             2: //and
55             begin
56                 out = A & B;
57                 co=0;
58                 ovf=0;
59                 z=(out == 0) ? 1 : 0;
60                 if (out[W-1]==1) begin
61                     n=1;
62                 end else begin
63                     n=0;
64                 end
65             end
66             3: // or
67             begin
68                 out = A | B;
69                 co=0;
70                 ovf=0;
71                 z=(out == 0) ? 1 : 0;
72                 if (out[W-1]==1) begin
73                     n=1;
74                 end else begin
75                     n=0;
76                 end
77             end
78         end case
79     end
```

```

76         n=0;
77     end
78 end
79
80 4: // xor
81 begin
82     out = A ^ B;
83     co=0;
84     ovf=0;
85     z=(out == 0) ? 1 : 0;
86     if (out[W-1]==1) begin
87         n=1;
88     end else begin
89         n=0;
90     end
91 end
92
93 5: // clear
94 begin
95     out = 0;
96     co=0;
97     ovf=0;
98     z=(out == 0) ? 1 : 0;
99     if (out[W-1]==1) begin
100         n=1;
101     end else begin
102         n=0;
103     end
104 end
105
106 6: // shift
107 begin
108     out = A;
109     co=0;
110     ovf=0;
111     z=(out == 0) ? 1 : 0;
112     if (out[W-1]==1) begin
113         n=1;
114     end else begin
115         n=0;
116     end
117 end
118
119
120     default: out = A + B ;
121 endcase
122 end
123
124 endmodule

```

## Register File

```

1 module REG_FILE(input RST,CLK,
2                 input WE3,
3                 input [2:0] RA1, RA2, WA3,
4                 input [7:0] WD3, R15,
5                 output wire [7:0] RD1, RD2, RDstr, R2, R3);
6
7     reg [7:0] RF[7:0];
8     integer i;
9     initial
10 begin
11     for (i=0; i<7; i=i+1) RF[i] <= 8'b0;
12 end
13
14     always @(posedge CLK)
15         // Sequential Write
16         begin
17             if (WE3) RF[WA3] <= WD3;
18             if(RST==1) for (i=0; i<7; i=i+1) RF[i] <= 8'b0;
19         end
20
21     assign RD1 = RF[RA1]; // Combinational Read
22     assign RD2 = RF[RA2];
23     assign RDstr = RF[WA3];
24     assign R0 = RF[0];
25     assign R1 = RF[1];
26     assign R2 = RF[2];
27     assign R3 = RF[3];
28     assign R4 = RF[4];
29     assign R5 = RF[5];
30     assign R6 = RF[6]; // LR
31     assign R_15 = R15; // PC
32
33 endmodule

```

## Instruction Data Memory

Date: April 29, 2019

IDM.v

Project: deOnano\_embedding

```
1  module IDM(A, WD, clk, WE, out);
2      input [7:0] WD;
3      input [7:0] A;
4      input clk, WE;
5      output [15:0] out;
6      reg [15:0] data[63:0]; // 2^6 memory slots
7
8      initial begin
9          data[0] <= 16'b10_001_01100001010;
10         data[1] <= 16'b10_001_01100010101;
11         data[2] <= 16'b10_001_01100011000;
12         data[3] <= 16'b11_000_01000000000;
13
14
15         data[10] <= 16'b11_000_010_00000001; // LDD R2, #1 ; load immediate numbers to the
registers
16         data[11] <= 16'b11_000_011_00000100; // LDD R3, #4
17         data[12] <= 16'b11_000_100_00000011; // LDD R4, #3
18         data[13] <= 16'b11_000_101_00000011; // LDD R5, #3
19         data[14] <= 16'b11_000_111_00000011; // LDD R7, #3
20         data[15] <= 16'b00_000_010_010_011_00; // ADD R2, R3
21         data[16] <= 16'b00_000_010_010_100_00;
22         data[17] <= 16'b00_000_010_010_101_00;
23         data[18] <= 16'b00_000_010_010_111_00;
24         data[19] <= 16'b10_001_01100000001;
25
26         data[21] <= 16'b11_00001000101010;
27         data[22] <= 16'b00_00101000101000;
28         data[23] <= 16'b10_00101100000010;
29
30         data[24] <= 16'b11_00001000000000;
31         data[25] <= 16'b11_00001100000000;
32         data[26] <= 16'b11_00001000000111;
33         data[27] <= 16'b11_00001100000111;
34         data[28] <= 16'b01_01001001000000;
35         data[29] <= 16'b01_01001001000000;
36         data[30] <= 16'b01_01001001000000;
37         // to be continued
38
39     end
40
41     always @(posedge clk) begin
42         if(WE) data[A[7:0]] <= WD;
43     end
44
45     assign out = data[A[7:0]];
46 endmodule
47
```

## Testbench

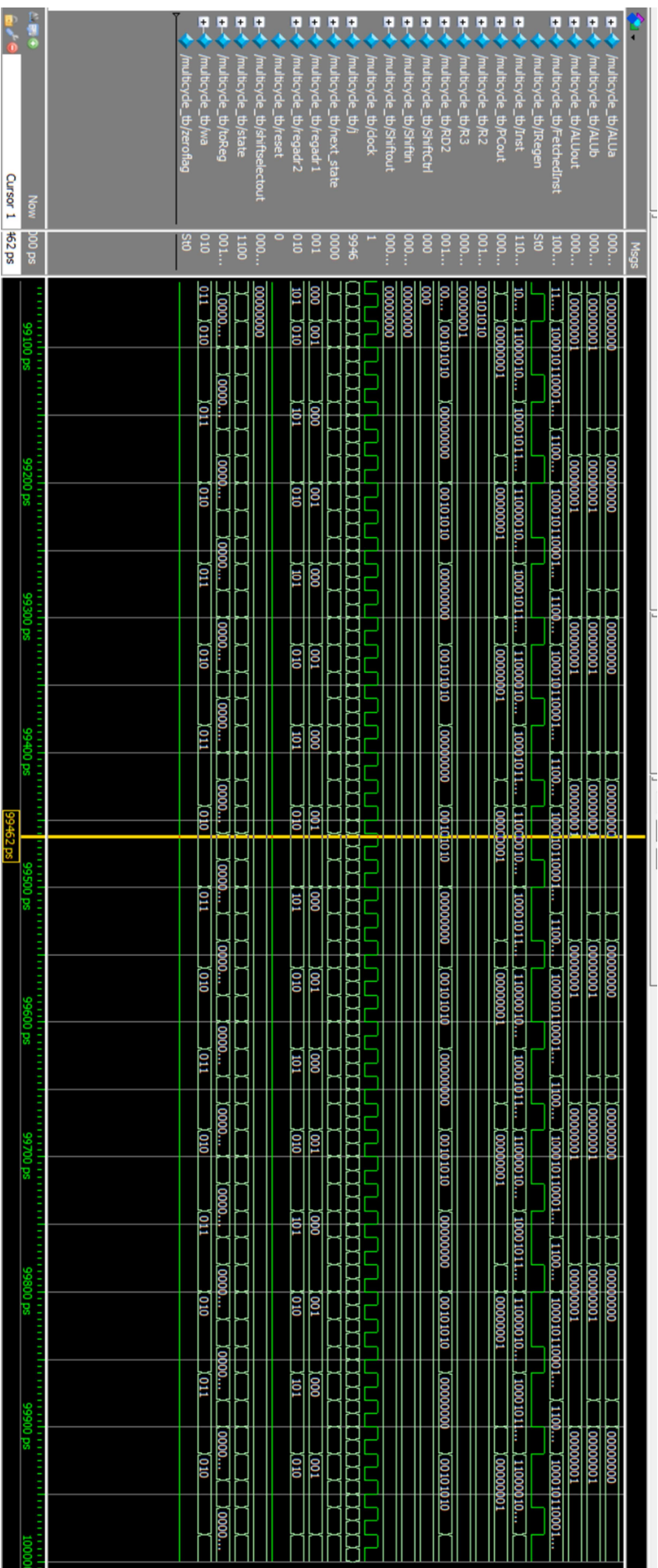
Date: April 29, 2019

multicycle\_tb.v

Project: de0nano\_embedding

```
1  module multicycle_tb();
2
3  reg clock;
4  reg reset;
5  wire zeroflag;
6  wire IRegen;
7  wire [7:0] ALUa;
8  wire [7:0] ALUb;
9  wire [7:0] ALUout;
10 wire [15:0] FetchedInst;
11 wire [15:0] Inst;
12 wire [3:0] next_state;
13 wire [7:0] PCout;
14 wire [7:0] R2;
15 wire [7:0] R3;
16 wire [7:0] RD2;
17 wire [2:0] regadr1;
18 wire [2:0] regadr2;
19 wire [2:0] ShiftCtrl;
20 wire [7:0] Shiftin;
21 wire [7:0] Shiftout;
22 wire [7:0] shiftselectout;
23 wire [3:0] state;
24 wire [7:0] toReg;
25 wire [2:0] wa;
26
27
28 multicycle DUT(
29     reset,
30     clock,
31     zeroflag,
32     IRegen,
33     ALUa,
34     ALUb,
35     ALUout,
36     FetchedInst,
37     Inst,
38     next_state,
39     PCout,
40     R2,
41     R3,
42     RD2,
43     regadr1,
44     regadr2,
45     ShiftCtrl,
46     Shiftin,
47     Shiftout,
48     shiftselectout,
49     state,
50     toReg,
51     wa
52 );
53
54 integer j;
55
56 initial begin
57     clock=0;
58     reset = 0;
59 end
60
61 always @(*)
62 begin
63     for(j=0; j<10000; j=j+1) begin
64         clock = ~clock;
65         if(R2 != 8'b00000000)
66             $display("Error in %1d case",R2);
67         else
68             $display("No error in %1d case",R2);
69         #10;
70     end
71     #1 $display("t=%t", $time, " r2=%1d", R2);
72 end
73
74
75 endmodule
```







Date: April 29, 2019

