13/05/2019

EE446 LABORATORY

EXPERIMENT 5

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, [Rm]
  - SUBI Rdi Rn, [Rm]
  - SUBI Rd, Rn, [Rm]
- 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 [Rm]
  - 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					

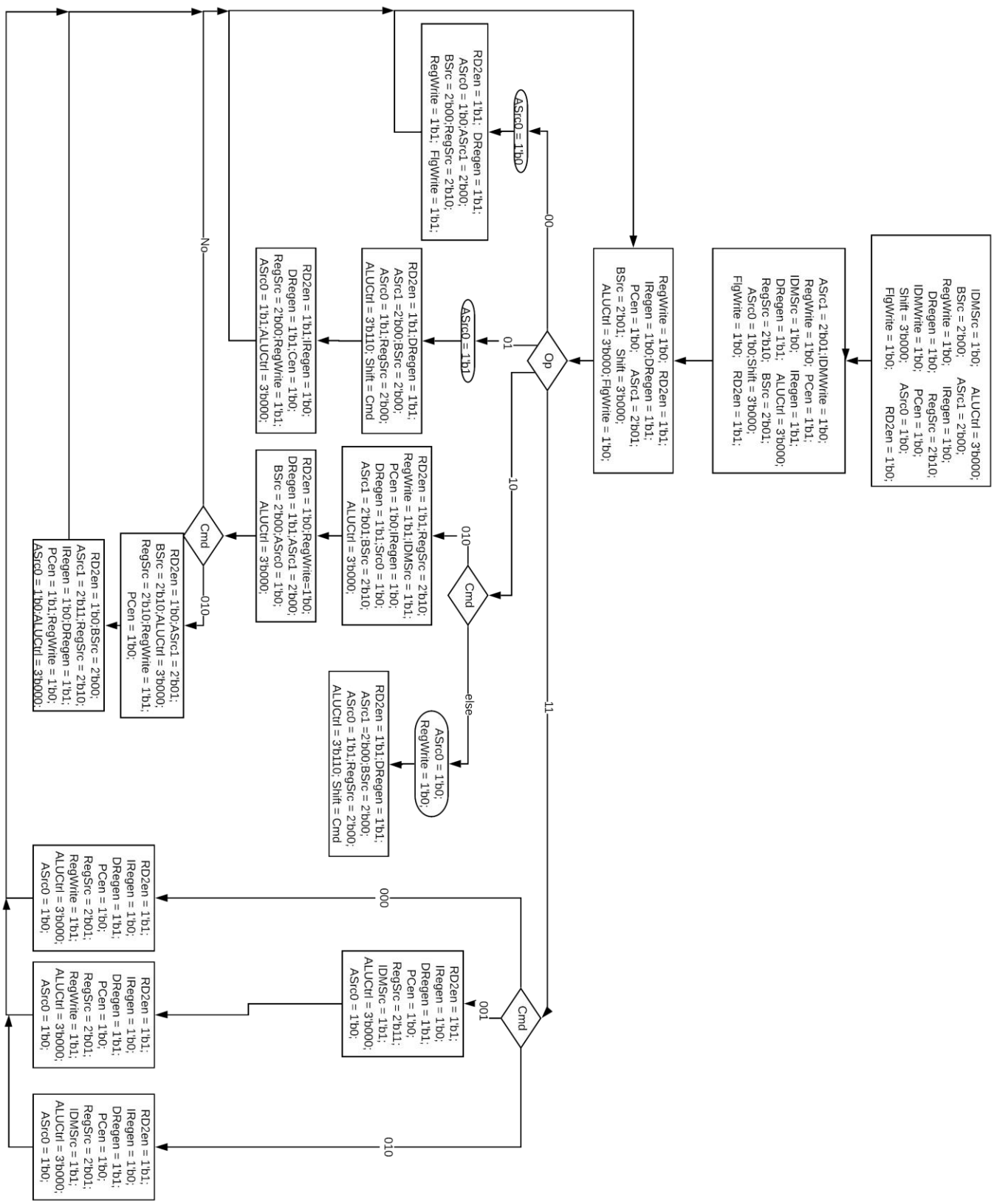
Arithmetic and logic instructions use Register file, ALU and IDM (Instruction / Data Memory).

Shift instructions use Shifter, Register File and IDM.

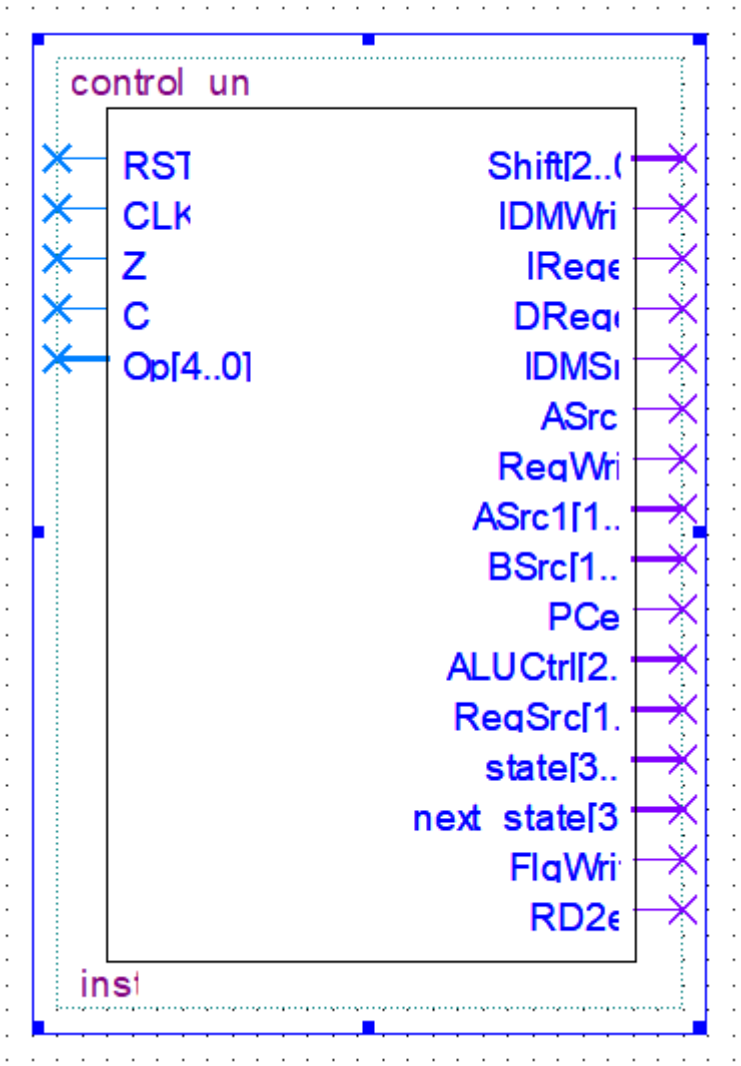
Load/Store type of instructions use Register file and IDM.

	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	

ASM Chart


IDMSrc = 1'b0; ALUCtrl = 3'b000;
BSrc = 2'b00; ASrc1 = 2'b00;
RegWrite = 1'b0; IRegen = 1'b0;
DRegen = 1'b0; RegSrc = 2'b10;
IDMWrite = 1'b0; PCen = 1'b0;
Shift = 3'b000; ASrc0 = 1'b0;
FlgWrite = 1'b0; RD2en = 1'b0;
ASrc1 = 2'b01;IDMWrite = 1'b0;
RegWrite = 1'b0; PCen = 1'b1;
IDMSrc = 1'b0; IRegen = 1'b1;
DRegen = 1'b1; ALUCtrl = 3'b000;
RegSrc = 2'b10; BSrc = 2'b01;
ASrc0 = 1'b0;Shift = 3'b000;
FlgWrite = 1'b0; RD2en = 1'b1;
RegWrite = 1'b0; RD2en = 1'b1;
IRegen = 1'b0;DRegen = 1'b1;
PCen = 1'b0; ASrc1 = 2'b01;
BSrc = 2'b01; Shift = 3'b000;
ALUCtrl = 3'b000;FlgWrite = 1'b0;
Op
00
01
10
11
ASrc0 = 1'b0
ASrc0 = 1'b1
RD2en = 1'b1; DRegen = 1'b1;
ASrc0 = 1'b0;ASrc1 = 2'b00;
BSrc = 2'b00;RegSrc = 2'b10;
RegWrite = 1'b1; FlgWrite = 1'b1;
RD2en = 1'b1;DRegen = 1'b1;
ASrc1 =2'b00;BSrc = 2'b00;
ASrc0 = 1'b1;RegSrc = 2'b00;
ALUCtrl = 3'b110; Shift = Cmd
RD2en = 1'b1;IRegen = 1'b0;
DRegen = 1'b1;Cen = 1'b0;
RegSrc = 2'b00;RegWrite = 1'b1;
ASrc0 = 1'b1;ALUCtrl = 3'b000;
RD2en = 1'b1;RegSrc = 2'b10;
RegWrite = 1'b1;IDMSrc = 1'b1;
PCen = 1'b0;IRegen = 1'b0;
DRegen = 1'b1;Src0 = 1'b0;
ASrc1 = 2'b01;BSrc = 2'b10;
ALUCtrl = 3'b000;
RD2en = 1'b0;RegWrite=1'b0;
DRegen = 1'b1;ASrc1 = 2'b00;
BSrc = 2'b00;ASrc0 = 1'b0;
ALUCtrl = 3'b000;
RD2en = 1'b0;ASrc1 = 2'b01;
BSrc = 2'b10;ALUCtrl = 3'b000;
RegSrc = 2'b10;RegWrite = 1'b1;
PCen = 1'b0;
RD2en = 1'b0;BSrc = 2'b00;
ASrc1 = 2'b11;RegSrc = 2'b10;
IRegen = 1'b0;DRegen = 1'b1;
PCen = 1'b1;RegWrite = 1'b0;
ASrc0 = 1'b0;ALUCtrl = 3'b000;
RD2en = 1'b1;DRegen = 1'b1;
ASrc1 =2'b00;BSrc = 2'b00;
ASrc0 = 1'b1;RegSrc = 2'b00;
ALUCtrl = 3'b110; Shift = Cmd
ASrc0 = 1'b0;
RegWrite = 1'b0;
Cmd
else
010
No
Cmd
010
Cmd
000
001
010
RD2en = 1'b1;
IRegen = 1'b0;
DRegen = 1'b1;
PCen = 1'b0;
RegSrc = 2'b01;
RegWrite = 1'b1;
ALUCtrl = 3'b000;
ASrc0 = 1'b0;
RD2en = 1'b1;
IRegen = 1'b0;
DRegen = 1'b1;
PCen = 1'b0;
RegSrc = 2'b11;
IDMSrc = 1'b1;
ALUCtrl = 3'b000;
ASrc0 = 1'b0;
RD2en = 1'b1;
IRegen = 1'b0;
DRegen = 1'b1;
PCen = 1'b0;
RegSrc = 2'b01;
IDMSrc = 1'b1;
ALUCtrl = 3'b000;
ASrc0 = 1'b0;

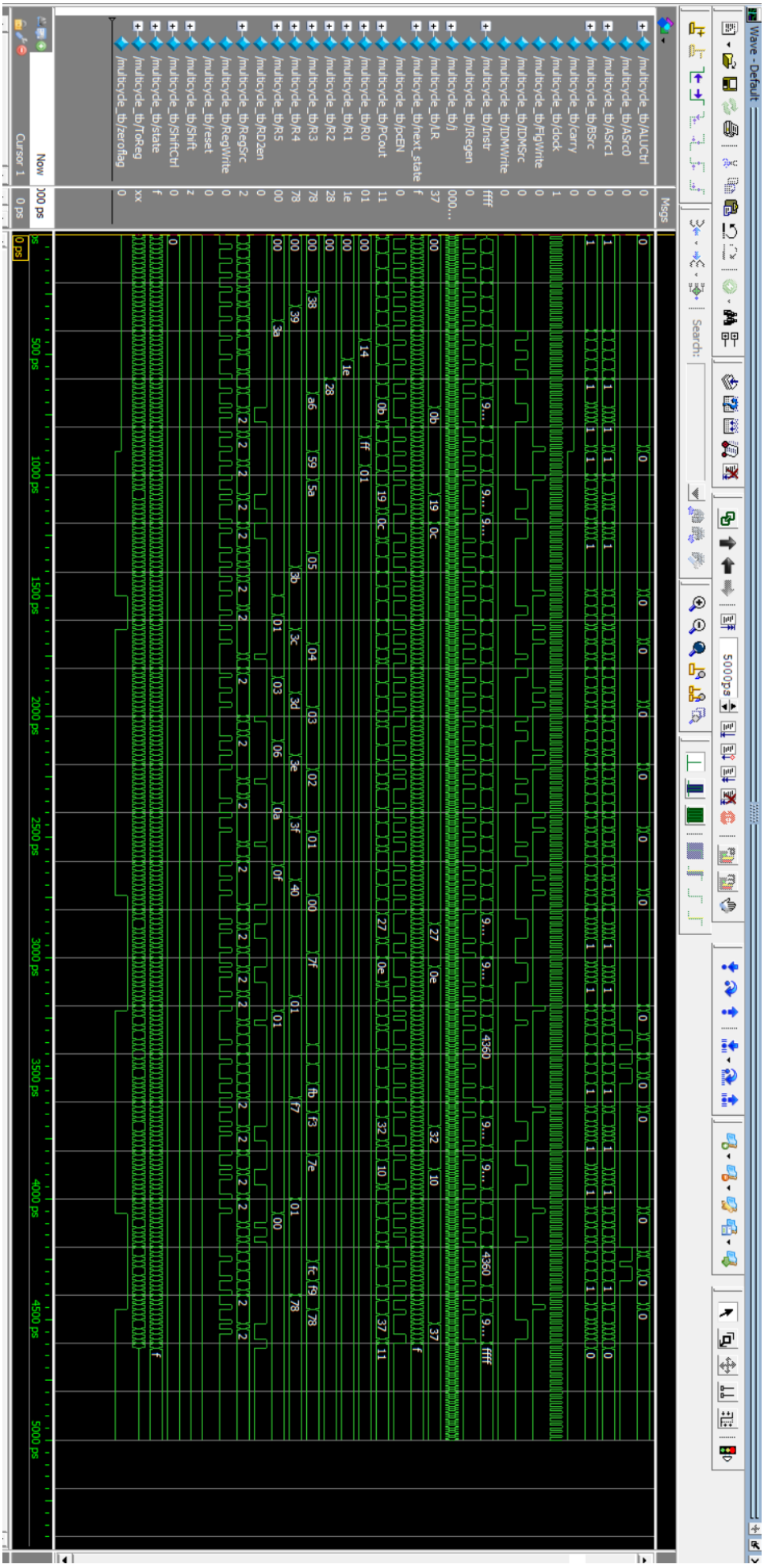
Control Unit



Testbench


```
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
```

Simulation Result



```
module IDM(A, WD, clk, WE, out);
 input [7:0] WD;
 input[7:0] A;
 input clk, WE;
 output [15:0] out;
 reg [15:0] data[63:0]; // 2^6 memory slots

 initial begin
 // INSTRUCTION PART ////////////////////////////////////
   data[0] = 16'b11_000_000_00000000;   // LDD R0, #0
   data[1] = 16'b11_000_001_00000000;   // LDD R1, #0
   data[2] = 16'b11_000_010_00000000;   // LDD R2, #0
   data[3] = 16'b11_000_011_00111000;   // LDD R3, #56
   data[4] = 16'b11_000_100_00111001;   // LDD R4, #57
   data[5] = 16'b11_000_101_00111010;   // LDD R5, #58
   data[6] = 16'b00_010_000_000_011_00; // ADDI R0, R0, [R3]
   data[7] = 16'b00_010_001_001_100_00; // ADDI R1, R1, [R4]
   data[8] = 16'b00_010_010_010_101_00; // ADDI R2, R2, [R5]
   data[9] = 16'b11_000_011_10100110;   // LDD R3, #8'b10100110; R3 = 0xA6

   data[10] = 16'b10_010_110_000_000_00;// BLI [R0];  2s Complement

   data[11] = 16'b10_010_110_000_001_00;// BLI [R1];  Sum of an array

   data[12] = 16'b11_000_011_01111111;  // LDD R3, #0x7F
   data[13] = 16'b10_010_110_000_010_00;// BLI [R2];  Odd?

   data[14] = 16'b11_000_011_01111110;  // LDD R3, #0x7E
   data[15] = 16'b10_010_110_000_010_00;// BLI [R2];  Even?
   data[16] = 16'b11_111_111_11111111;  // END

// Subroutine 1 , 2s Complement
   data[20] = 16'b11_000_000_11111111;  // LDD R0, #0xFF
   data[21] = 16'b00_110_011_011_000_00;// XOR R3, R3, R0
   data[22] = 16'b11_000_000_00000001;  // LDD R0, #1
   data[23] = 16'b00_000_011_011_000_00;// ADD R3, R3, R0
   data[24] = 16'b10_010_110_000_110_00;// BLI LR

// Subroutine 2, Sum of an array
   data[30] = 16'b11_000_011_00000101;  // LDD R5, #5
   data[31] = 16'b11_000_100_00111011;  // LDD R4, #59
   data[32] = 16'b11_000_000_00000001;  // LDD R0, #1
   data[33] = 16'b00_111_101_00000000;  // CLR R3
   data[34] = 16'b00_010_101_101_100_00;// ADDI R3, R3, R4
   data[35] = 16'b00_000_100_100_000_00;// ADD R4, R4, R0
   data[36] = 16'b00_001_011_011_000_00;// SUB R5, R5, R0
   data[37] = 16'b10_100_00000100010;   // BNE #34
   data[38] = 16'b10_010_110_000_110_00;// BLI LR

// Subroutine 3, Even/Odd?
   data[40] = 16'b11_000_100_00000001;  // LDD R4, #0x1
   data[41] = 16'b00_100_101_100_011_00;// AND R5, R4, R3
   data[42] = 16'b10_100_00000101100;   // BNE #44
   data[43] = 16'b10_011_00000110010;   // BEQ #50

    //MEM1
   data[44] = 16'b01_000_011_011_00000; // ROL R3
   data[45] = 16'b01_000_011_011_00000; // ROL R3
   data[46] = 16'b01_000_011_011_00000; // ROL R3
   data[47] = 16'b11_000_100_11110111;  // LDD R4, #0xF7
   data[48] = 16'b00_100_011_011_100_00;// AND R3, R4
   data[49] = 16'b10_010_110_000_110_00;// BLI LR

    //MEM2
   data[50] = 16'b01_000_011_011_00000; // ROL R3
   data[51] = 16'b01_000_011_011_00000; // ROL R3
   data[52] = 16'b11_000_100_01111000;  // LDD R4, #0x78
   data[53] = 16'b00_100_011_011_100_00;// AND R3, R4
   data[54] = 16'b10_010_110_000_110_00;// BLI LR

 // DATA PART ///////////////////////////////////////
   data[56] = 20;
   data[57] = 30;
   data[58] = 40;

   data[59] = 4'h0001;
   data[60] = 4'h0002;
```

```
78        data[61] = 4'h0003;
79        data[62] = 4'h0004;
80        data[63] = 4'h0005;
81
82      end
83
84      always @(posedge clk)begin
85       if(WE) data[A[7:0]]<= WD;
86      end
87
88      assign out = data[A[7:0]];
89    endmodule
90
```

```
module control_unit(
   Shift,
   IDMWrite,
   IRegen,
   DRegen,
   IDMSrc,
   ASrc0,
   RegWrite,
   ASrc1,
   BSrc,
   PCen,
   ALUCtrl,
   RegSrc,
   RST,
   CLK,
   Z,
   C,
   Op,
   state, next_state,
   FlgWrite, RD2en
    );

output   reg   [2:0] Shift,ALUCtrl;
output   reg   [1:0] RegSrc, BSrc, ASrc1;
output   reg   PCen,IDMSrc,IDMWrite,IRegen,DRegen,RegWrite,ASrc0, FlgWrite, RD2en;

input [4:0] Op;
input CLK,RST;
input Z,C;

output reg  [3:0] state, next_state;

//ALUOp Values:
// 00 -> addition
// 01 -> subtraction
// 10 -> Function (R-Type)
// 11 -> Op (I-Type)


//////////////////////
//STATES/////////////
//////////////////////

   parameter S0 = 4'b0000; //Instruction Fetch
   parameter S1 = 4'b0001; //Instruction Decode
   parameter S2 = 4'b0010;
   parameter S3 = 4'b0011;
   parameter S4 = 4'b0100;
   parameter S5 = 4'b0101;
   parameter S6 = 4'b0110;
   parameter S7 = 4'b0111;
   parameter S8 = 4'b1000;
   parameter S9 = 4'b1001;  //Memory Write
   parameter S10 = 4'b1010; //Register Write
   parameter S11 = 4'b1011;
   parameter S12 = 4'b1100;
   parameter S13 = 4'b1101;
   parameter S14 = 4'b1110;
   parameter S15 = 4'b1111;

initial
begin
      IDMSrc = 1'b0;
      ALUCtrl = 3'b000;
      BSrc = 2'b00;
      ASrc1 = 2'b00;
      RegWrite = 1'b0;
      IRegen = 1'b0;
      DRegen = 1'b0;
      RegSrc = 2'b10;
      IDMWrite = 1'b0;
      PCen = 1'b0;
      Shift = 3'b000;
      ASrc0 = 1'b0;
      FlgWrite = 1'b0;
      RD2en = 1'b0;
      state = 4'b1111;
```

```
78          next_state = 4'b0000;
79      end
80      //Sequential Logic Synced to Clock
81
82      always @(posedge CLK)
83      begin
84         state <= next_state;
85      end
86
87
88      always @ (state or RST)
89      begin
90         if(RST == 1'b1)
91            begin
92            next_state = S0;
93            IDMSrc = 1'b0;
94            ALUCtrl = 3'b000;
95            BSrc = 2'b00;
96            ASrc1 = 2'b00;
97            RegWrite = 1'b0;
98            IRegen = 1'b0;
99            DRegen = 1'b0;
100           RegSrc = 2'b10;
101           IDMWrite = 1'b0;
102           PCen = 1'b0;
103           Shift = 3'b000;
104           ASrc0 = 1'b0;
105           FlgWrite = 1'b0;
106           RD2en = 1'b0;
107           end
108
109        else
110        begin
111
112        case(state)
113
114           //Instruction Fetch
115           S0:
116           begin
117              //RST Controller Values
118              ASrc1 = 2'b01;
119              IDMWrite = 1'b0;
120              RegWrite = 1'b0;
121              PCen = 1'b1;
122              IDMSrc = 1'b0;
123              IRegen = 1'b1;
124              DRegen = 1'b1;
125              ALUCtrl = 3'b000;
126              RegSrc = 2'b10;
127              BSrc = 2'b01;
128              ASrc0 = 1'b0;
129              Shift = 3'b000;
130              FlgWrite = 1'b0;
131              RD2en = 1'b1;
132              next_state = S1;
133           end
134
135           //Instruction Decode
136           S1:
137           begin
138              RegWrite = 1'b0;
139              RD2en = 1'b1;
140              IRegen = 1'b0;
141              DRegen = 1'b1;
142              PCen = 1'b0;
143              ASrc1 = 2'b01;
144              BSrc = 2'b01;
145              Shift = 3'b000;
146              ALUCtrl = 3'b000;
147              FlgWrite = 1'b0;
148              //shift
149              if(Op[4:3] == 2'b01)
150                 begin
151                    ASrc0 = 1'b1;
152                    next_state = S11;
153                 end
154
```

```
155             //R-Type
156             else if(Op[4:3] == 2'b00 && Op[2:0] != 3'b010 && Op[2:0] != 3'b011)
157                begin
158                ASrc0 = 1'b0;
159                   next_state = S6;
160                end
161
162
163             //Branch Type
164             else if(Op[4:3] == 2'b10)
165                begin
166                   if(Op[2:0] == 3'b010)
167                      next_state = S4;
168                   else
169                      begin
170                      ASrc0 = 1'b0;
171                      RegWrite = 1'b0;
172                      next_state = S10;
173                      end
174                end
175
176             //I-Type
177             else if(Op[4:3] == 2'b11)
178                begin
179                ASrc0 = 1'b0;
180                   if(Op[2:0] == 3'b000) //loadimm
181                   begin
182                         next_state = S12;
183                   end
184
185                   else if(Op[2:0] == 3'b001)
186                   begin
187                         next_state = S2;
188                   end
189
190                   else if(Op[2:0] == 3'b010) //store
191                   begin
192                         next_state = S14;
193                   end
194
195                   else if(Op[2:0] == 3'b111) // end
196                   begin
197                         next_state = S15;
198                   end
199
200                end
201                else if(Op[4:3] == 2'b00 ) //ADDI or SUBI
202                   begin
203                      next_state = S5;
204                   end
205             end
206
207          //load
208          S2:
209          begin
210                RD2en = 1'b1;
211                IRegen = 1'b0;
212                DRegen = 1'b1;
213                PCen = 1'b0;
214                RegSrc = 2'b11;
215                IDMSrc = 1'b1;
216                ASrc0 = 1'b0;
217                ALUCtrl = 3'b000;
218                next_state = S3;
219
220          end
221
222          //load comp
223          S3:
224          begin
225             RD2en = 1'b1;
226             IRegen = 1'b0;
227             DRegen = 1'b1;
228             PCen = 1'b0;
229             RegSrc= 2'b01;
230             RegWrite=1'b1;
231             ASrc0 = 1'b0;
```

```
232             ALUCtrl = 3'b000;
233             next_state= S0;
234         end
235
236         // BLI
237         S4:
238         begin
239             RD2en = 1'b1;
240             RegSrc = 2'b10;
241             RegWrite = 1'b1;
242             IDMSrc = 1'b1;
243             PCen = 1'b0;
244             IRegen = 1'b0;
245             DRegen = 1'b1;
246             ASrc0 = 1'b0;
247             ASrc1 = 2'b01;
248             BSrc = 2'b10;
249             ALUCtrl = 3'b000;
250             next_state = S10;
251         end
252
253         // ADDI SUBI
254         S5:
255         begin
256             RD2en = 1'b1;
257             RegSrc = 2'b11;
258             IDMSrc = 1'b1;
259             PCen = 1'b0;
260             IRegen = 1'b0;
261             DRegen = 1'b1;
262             ASrc0 = 1'b0;
263             ASrc1 = 2'b00;
264             BSrc = 2'b11;
265
266             //ADD ind
267             if(Op[2:0] == 3'b010)
268                 ALUCtrl = 3'b000;
269
270             //SUB ind
271             else if(Op[2:0] == 3'b011)
272                 ALUCtrl = 3'b001;
273             next_state = S7;
274         end
275
276         //Execution: R-Type
277         S6:
278         begin
279             RD2en = 1'b1;
280             DRegen = 1'b1;
281             ASrc0 = 1'b0;
282             ASrc1 = 2'b00;
283             BSrc = 2'b00;
284             RegSrc = 2'b10;
285             RegWrite = 1'b1;
286             FlgWrite = 1'b1;
287             next_state = S0;
288
289             //add
290             if(Op[2:0] == 3'b000)
291                 ALUCtrl = 3'b000;
292
293             //sub
294             else if(Op[2:0] == 3'b001)
295                 ALUCtrl = 3'b001;
296
297             //ADD ind
298             else if(Op[2:0] == 3'b010)
299                 ALUCtrl = 3'b000;
300
301             //SUB ind
302             else if(Op[2:0] == 3'b011)
303                 ALUCtrl = 3'b001;
304
305             //and
306             else if(Op[2:0] == 3'b100)
307                 ALUCtrl = 3'b010;
308
```

```
309             //or
310             else if(Op[2:0] == 3'b101)
311                 ALUCtrl = 3'b011;
312 
313             //XOR
314             else if(Op[2:0] == 3'b110)
315                 ALUCtrl = 3'b100;
316 
317             //clear
318             else if(Op[2:0] == 3'b111)
319                 ALUCtrl = 3'b101;
320         end
321 
322         //R-Type Completion
323         S7:
324             begin
325             RD2en = 1'b1;
326             IRegen = 1'b0;
327             DRegen = 1'b0;
328             PCen = 1'b0;
329             RegSrc = 2'b10;
330             RegWrite = 1'b1;
331             ASrc0 = 1'b0;
332             next_state = S0;
333             end
334 
335         //shift comp
336         S8:
337         begin
338             RD2en = 1'b1;
339             IRegen = 1'b0;
340             DRegen = 1'b1;
341             PCen = 1'b0;
342             RegSrc = 2'b00;
343             RegWrite = 1'b1;
344             ASrc0 = 1'b1;
345             ALUCtrl = 3'b000;
346             next_state = S0;
347         end
348 
349         //I-Type Completion
350         S9:
351         begin
352             RD2en = 1'b0;
353             ASrc1 = 2'b01;
354             BSrc = 2'b10;
355             ALUCtrl = 3'b000;
356             RegSrc = 2'b10;
357             RegWrite = 1'b1;
358             PCen = 1'b0;
359             next_state = S13;
360         end
361 
362         //Write Back - I-Type
363         S10:
364         begin
365             RD2en = 1'b0;
366             RegWrite=1'b0;
367             DRegen = 1'b1;
368             ASrc1 = 2'b00;
369             BSrc = 2'b00;
370             ASrc0 = 1'b0;
371             ALUCtrl = 3'b000;
372             next_state = S0;
373 
374             //B
375             if(Op[2:0] == 3'b000)
376                 begin
377                 PCen = 1'b1;
378                 RegSrc = 2'b01;
379                 IDMSrc = 1'b1;
380                 end
381 
382             //Bwlink
383             else if(Op[2:0] == 3'b001)
384                 begin
385                 PCen = 1'b1;
```

```
386                RegSrc = 2'b01;
387                IDMSrc = 1'b1;
388                end
389
390             //Bindwlink
391             else if(Op[2:0] == 3'b010)
392                begin
393                PCen = 1'b0;
394                RegSrc = 2'b11;
395                IDMSrc = 1'b1;
396                DRegen = 1'b1;
397                ASrc1 = 2'b11;
398                BSrc = 2'b11;
399                next_state = S9;
400                end
401
402             //BifZ
403             else if(Op[2:0] == 3'b011)
404                begin
405                   if(Z==0)
406                      begin
407                      PCen = 1'b0;
408                      RegSrc = 2'b01;
409                      IDMSrc = 1'b0;
410                      end
411                   else
412                      begin
413                      PCen = 1'b1;
414                      RegSrc = 2'b01;
415                      IDMSrc = 1'b1;
416                      end
417
418                end
419
420             //BifnotZ
421             else if(Op[2:0] == 3'b100)
422                begin
423                if(Z==1)
424                      begin
425                      PCen = 1'b0;
426                      RegSrc = 2'b01;
427                      IDMSrc = 1'b0;
428                      end
429                   else
430                      begin
431                      PCen = 1'b1;
432                      RegSrc = 2'b01;
433                      IDMSrc = 1'b1;
434                      end
435                end
436
437              //Bif C
438             else if(Op[2:0] == 3'b101)
439                begin
440                if(C==0)
441                      begin
442                      PCen = 1'b0;
443                      RegSrc = 2'b01;
444                      IDMSrc = 1'b0;
445                      end
446                   else
447                      begin
448                      PCen = 1'b1;
449                      RegSrc = 2'b01;
450                      IDMSrc = 1'b1;
451                      end
452                end
453
454             //Bif not C
455             else if(Op[2:0] == 3'b111)
456                begin
457                if(C==1)
458                      begin
459                      PCen = 1'b0;
460                      RegSrc = 2'b01;
461                      IDMSrc = 1'b0;
462                      end
```

```
463                     else
464                        begin
465                        PCen = 1'b1;
466                        RegSrc = 2'b01;
467                        IDMSrc = 1'b1;
468                        end
469                end
470            end
471
472            //shift
473            S11:
474            begin
475               RD2en = 1'b1;
476               DRegen = 1'b1;
477               ASrc1 =2'b00;
478               BSrc = 2'b00;
479               ASrc0 = 1'b1;
480               RegSrc = 2'b00;
481               ALUCtrl = 3'b110;
482               next_state = S8;
483
484               //rl
485               if(Op[2:0] == 3'b000)
486               begin
487                  Shift = 3'b000;
488               end
489
490               //rr
491               else if(Op[2:0] == 3'b001)
492               begin
493                  Shift = 3'b001;
494               end
495
496               //sl
497               else if(Op[2:0] == 3'b010)
498               begin
499                   Shift = 3'b010;
500               end
501
502               //asr
503               else if(Op[2:0] == 3'b011)
504               begin
505                   Shift = 3'b011;
506               end
507
508               //lsr
509               else if(Op[2:0] == 3'b100)
510               begin
511                   Shift = 3'b100;
512               end
513
514            end
515
516            S12:
517            begin
518               RD2en = 1'b1;
519               IRegen = 1'b0;
520               DRegen = 1'b1;
521               PCen = 1'b0;
522               RegSrc = 2'b01;
523               RegWrite = 1'b1;
524               ALUCtrl = 3'b000;
525               ASrc0 = 1'b0;
526               next_state = S0;
527            end
528
529            S13:
530            begin
531               RD2en = 1'b0;
532               BSrc = 2'b00;
533               ASrc1 = 2'b11;
534               RegSrc = 2'b10;
535               IRegen = 1'b0;
536               DRegen = 1'b1;
537               PCen = 1'b1;
538               RegWrite = 1'b0;
539               ASrc0 = 1'b0;
```

```
540             ALUCtrl = 3'b000;
541             next_state = S0;
542          end
543
544          S14:
545          begin
546             RD2en = 1'b1;
547             IRegen = 1'b0;
548             DRegen = 1'b1;
549             PCen = 1'b0;
550             IDMSrc = 1'b1;
551             RegSrc = 2'b01;
552             IDMWrite= 1'b1;
553             ALUCtrl = 3'b000;
554             ASrc0 = 1'b0;
555             next_state = S0;
556          end
557
558          S15:
559          begin
560             IDMSrc = 1'b0;
561             ALUCtrl = 3'b000;
562             BSrc = 2'b00;
563             ASrc1 = 2'b00;
564             RegWrite = 1'b0;
565             IRegen = 1'b0;
566             DRegen = 1'b0;
567             RegSrc = 2'b10;
568             IDMWrite = 1'b0;
569             PCen = 1'b0;
570             Shift = 3'b000;
571             ASrc0 = 1'b0;
572             FlgWrite = 1'b0;
573             RD2en = 1'b0;
574             next_state = S15;
575          end
576
577       endcase
578       end
579    end
580
581    endmodule
```