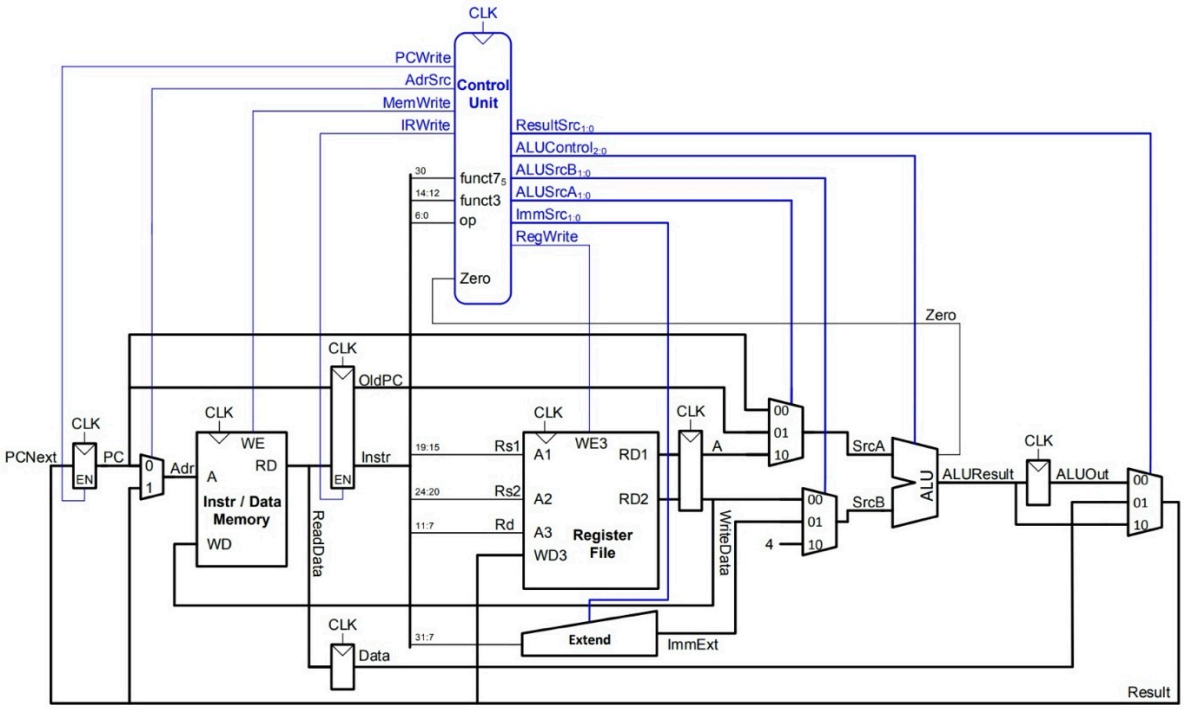
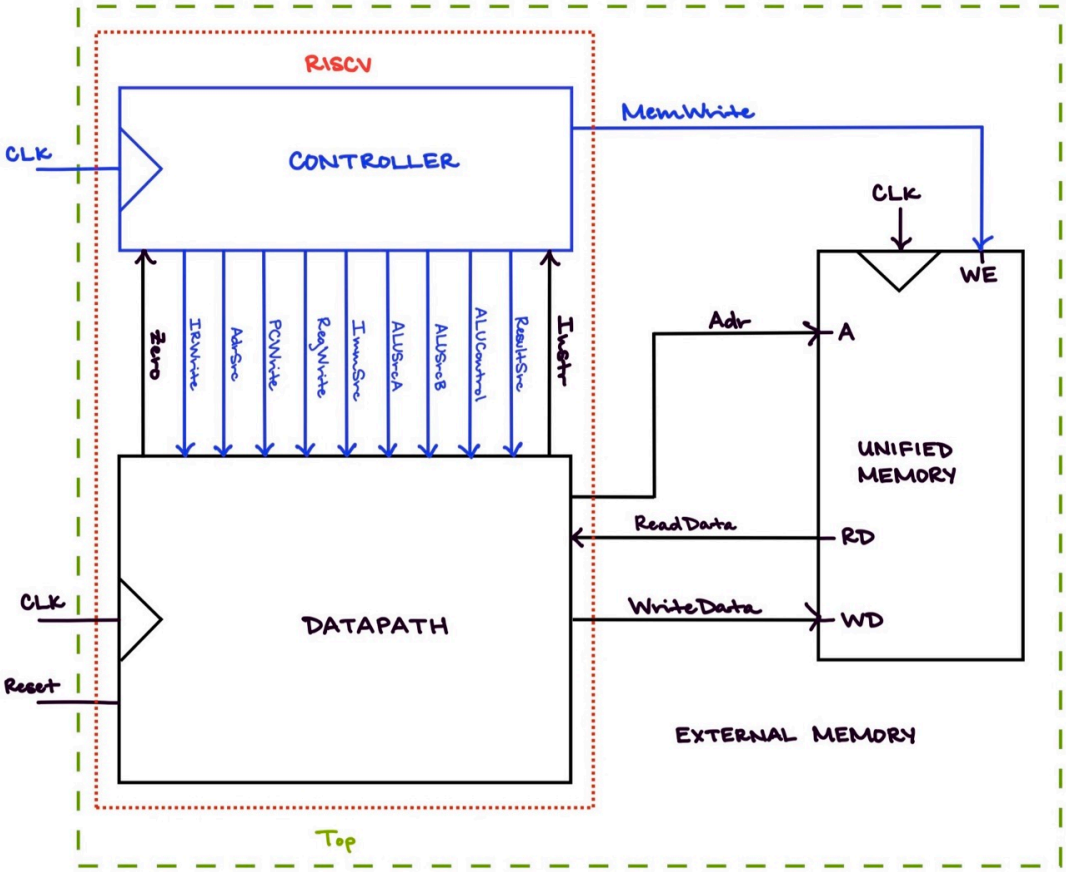
Complete Multicycle Processor



Digital Design and Computer Architecture: RISC-V Edition Harris & Harris © 2022 Elsevier

Multicycle processor high-level hierarchy



```
1   // Lab 11: Multicycle Processor
2   // Sebastian Heredia, dheredia@g.hmc.edu, 11/22/2024
3
4    module Top(input  logic  clk,
5               input  logic  reset,
6               output logic  [31:0]WriteData,
7               output logic  [31:0]Adr,
8               output logic  MemWrite);
9
10    logic [31:0]ReadData;
11
12     // Instantiate RISCVmulti
13     RISCVmulti RISCVmulti(clk, reset, ReadData, MemWrite, Adr, WriteData);
14
15     // Instantiate UnifiedMemory
16     UnifiedMemory UnifiedMemory(clk, MemWrite, Adr, WriteData, ReadData);
17
18  endmodule
19
20
```

```
1   // RISCV Multicycle Processor
2
3   module RISCVmulti (input  logic        clk,
4                      input  logic        reset,
5                      input  logic  [31:0]ReadData,
6                      output logic        MemWrite,
7                      output logic  [31:0]Adr,
8                      output logic  [31:0]WriteData);
9
10     // Internal Logic
11     logic [31:0]Instr;
12     logic [1:0]ALUSrcA, ALUSrcB, ImmSrc, ResultSrc;
13     logic [2:0]ALUControl;
14     logic zero, IRWrite, AdrSrc, RegWrite;
15     logic PCWrite;
16
17     // Instantiate Controller (mainfsm, instrdecoder, aludecoder)
18     controller controller(clk, reset, Instr[6:0], Instr[14:12], Instr[30], zero, ImmSrc,
       ALUSrcA, ALUSrcB, ResultSrc, AdrSrc, ALUControl, IRWrite, PCWrite, RegWrite, MemWrite);
19
20     // Instantiate DataPath
21     DataPath DataPath(clk, reset, ResultSrc, ALUSrcA, ALUSrcB, RegWrite, ImmSrc, ALUControl,
       ReadData, PCWrite, AdrSrc, IRWrite, zero, Instr, WriteData, Adr);
22
23  endmodule
24
```

```
1   // Controller: Top level hierarchy
2
3
4   module controller(input logic clk,
5    input logic reset,
6    input logic [6:0] op,
7    input logic [2:0] funct3,
8    input logic funct7b5,
9    input logic zero,
10   output logic [1:0] immsrc,
11   output logic [1:0] alusrca, alusrcb,
12   output logic [1:0] resultsrc,
13   output logic adrsrc,
14   output logic [2:0] alucontrol,
15   output logic irwrite, pcwrite,
16   output logic regwrite, memwrite);
17
18      // Internal node
19      logic branch, pcupdate;
20      logic [1:0]ALUOp;
21
22      assign pcwrite = branch & zero | pcupdate;
23
24    // Instantiate aludecoder, instrdecoder, mainfsm
25  aludecoder a2(ALUOp, funct3, op[5], funct7b5, alucontrol);
26
27  instrdecoder i1(op, immsrc);
28
29  mainfsm m1(clk, reset, op, branch, pcupdate, regwrite, memwrite, irwrite, resultsrc,
    alusrcb, alusrca, adrsrc, ALUOp);
30
31  endmodule
32
```

```
1   // ALU Decoder
2
3   module aludecoder(input logic [1:0] ALUOp,
4                     input logic [2:0] Funct3,
5                     input logic Op5,
6                     input logic Funct7b5,
7                     output logic [2:0] ALUControl);
8
9   always_comb
10     casez ({ALUOp, Op5, Funct7b5, Funct3})
11        7'b00?????: ALUControl = 3'b000; // add
12        7'b01?????: ALUControl = 3'b001; // subtract
13        7'b1000000: ALUControl = 3'b000; // add
14        7'b1001000: ALUControl = 3'b000; // add
15        7'b1010000: ALUControl = 3'b000; // add
16        7'b1011000: ALUControl = 3'b001; // subtract
17        7'b10?0010: ALUControl = 3'b101; // set less than
18        7'b10?0110: ALUControl = 3'b011; // or
19        7'b10?0111: ALUControl = 3'b010; // and
20        default: ALUControl = 3'b000;
21     endcase
22  endmodule
23
24
```

```
1   // Instruction Decoder
2
3   module instrdecoder( input logic [6:0] Op,
4                        output logic [1:0] ImmSrc);
5
6   always_comb
7      case (Op)
8         7'b0110011: ImmSrc = 2'b00;   // R-Type
9         7'b0010011: ImmSrc = 2'b00;   // I-Type
10        7'b0000011: ImmSrc = 2'b00;   // lw
11        7'b0100011: ImmSrc = 2'b01;   // sw
12        7'b1100011: ImmSrc = 2'b10;   // beq
13        7'b1101111: ImmSrc = 2'b11;   // jal
14
15        default: ImmSrc = 2'bxx;
16     endcase
17  endmodule
```

```
1   // Main FSM
2
3   module mainfsm(input logic clk,
4                  input logic reset,
5                  input logic [6:0] Op,
6                  output logic Branch,
7                  output logic PCUpdate,
8                  output logic RegWrite,
9                  output logic MemWrite,
10                 output logic IRWrite,
11                 output logic [1:0] ResultSrc,
12                 output logic [1:0] ALUSrcB,
13                 output logic [1:0] ALUSrcA,
14                 output logic AdrSrc,
15                 output logic [1:0] ALUOp);
16
17     typedef enum logic [3:0] {s0, s1, s2, s3, s4, s5, s6 ,s7, s8, s9, s10} statetype;
18     statetype state, nextstate;
19
20     // State register w/ asychronous reset
21     always_ff @ (posedge clk, posedge reset)
22        if (reset)  state <= s0;
23        else        state <= nextstate;
24
25     // Next state logic
26     always_comb
27        case(state)
28           s0: nextstate = s1;
29
30           s1: if (Op == 7'b0000011 || Op == 7'b0100011) nextstate = s2;
31               else if (Op == 7'b0110011) nextstate = s6;
32               else if (Op == 7'b0010011) nextstate = s8;
33               else if (Op == 7'b1101111) nextstate = s9;
34               else if (Op == 7'b1100011) nextstate = s10;
35               else nextstate = s1;
36
37           s2: if (Op == 7'b0000011) nextstate = s3;
38               else if (Op == 7'b0100011) nextstate = s5;
39               else nextstate = s2;
40
41           s3:   nextstate = s4;
42
43           s4:   nextstate = s0;
44
45           s5:   nextstate = s0;
46
47           s6:   nextstate = s7;
48
49           s7:   nextstate = s0;
50
51           s8:   nextstate = s7;
52
53           s9:   nextstate = s7;
54
55           s10:  nextstate = s0;
56        default: nextstate = s0;
57     endcase
58
59  // Output logic
60  always_comb begin
61     case(state)
62
63  s0: begin // Fetch
64     AdrSrc = 1'b0;
65     IRWrite = 1'b1;
66     ALUSrcA = 2'b00;
67     ALUSrcB = 2'b10;
68     ALUOp = 2'b00;
69     ResultSrc = 2'b10;
70     PCUpdate = 1'b1;
```

```
71      Branch = 1'b0;
72      RegWrite = 1'b0;
73      MemWrite = 1'b0;
74    end
75
76    s1: begin // Decode
77      ALUSrcA = 2'b01;
78      ALUSrcB = 2'b01;
79      ALUOp = 2'b00;
80      AdrSrc = 1'b0;
81      Branch = 1'b0;
82      PCUpdate = 1'b0;
83      RegWrite = 1'b0;
84      MemWrite = 1'b0;
85      IRWrite = 1'b0;
86      ResultSrc = 2'b00;
87    end
88
89    s2: begin // MemAdr
90      ALUSrcA = 2'b10;
91      ALUSrcB = 2'b01;
92      ALUOp = 2'b00;
93      AdrSrc = 1'b0;
94      Branch = 1'b0;
95      PCUpdate = 1'b0;
96      RegWrite = 1'b0;
97      MemWrite = 1'b0;
98      IRWrite = 1'b0;
99      ResultSrc = 2'b00;
100   end
101
102   s3: begin // MemRead
103     ResultSrc = 2'b00;
104     AdrSrc = 1'b1;
105     Branch = 1'b0;
106     PCUpdate = 1'b0;
107     RegWrite = 1'b0;
108     MemWrite = 1'b0;
109     IRWrite = 1'b0;
110     ALUSrcB = 2'b00;
111     ALUSrcA = 2'b00;
112     ALUOp = 2'b00;
113   end
114
115   s4: begin // MemWB
116     ResultSrc = 2'b01;
117     RegWrite = 1'b1;
118     Branch = 1'b0;
119     PCUpdate = 1'b0;
120     MemWrite = 1'b0;
121     IRWrite = 1'b0;
122     ALUSrcB = 2'b00;
123     ALUSrcA = 2'b00;
124     AdrSrc = 1'b0;
125     ALUOp = 2'b00;
126   end
127
128   s5: begin // MemWrite
129     ResultSrc = 2'b00;
130     AdrSrc = 1'b1;
131     MemWrite = 1'b1;
132     Branch = 1'b0;
133     PCUpdate = 1'b0;
134     RegWrite = 1'b0;
135     IRWrite = 1'b0;
136     ALUSrcB = 2'b00;
137     ALUSrcA = 2'b00;
138     AdrSrc = 1'b1;
139     ALUOp = 2'b00;
140   end
```

```
141
142     s6: begin // ExecuteR
143        ALUSrcA = 2'b10;
144        ALUSrcB = 2'b00;
145        ALUOp = 2'b10;
146        Branch = 1'b0;
147        PCUpdate = 1'b0;
148        RegWrite = 1'b0;
149        MemWrite = 1'b0;
150        IRWrite = 1'b0;
151        ResultSrc = 2'b00;
152        AdrSrc = 1'b0;
153     end
154
155     s7: begin // ALUWB
156        ResultSrc = 2'b00;
157        RegWrite = 1'b1;
158        Branch = 1'b0;
159        PCUpdate = 1'b0;
160        MemWrite = 1'b0;
161        IRWrite = 1'b0;
162        ALUSrcB = 2'b00;
163        ALUSrcA = 2'b00;
164        AdrSrc = 1'b0;
165        ALUOp = 2'b00;
166     end
167
168     s8: begin // ExecuteI
169        ALUSrcA = 2'b10;
170        ALUSrcB = 2'b01;
171        ALUOp = 2'b10;
172        Branch = 1'b0;
173        PCUpdate = 1'b0;
174        RegWrite = 1'b0;
175        MemWrite = 1'b0;
176        IRWrite = 1'b0;
177        ResultSrc = 2'b00;
178        AdrSrc = 1'b0;
179     end
180
181     s9: begin // JAL
182        ALUSrcA = 2'b01;
183        ALUSrcB = 2'b10;
184        ALUOp = 2'b00;
185        ResultSrc = 2'b00;
186        PCUpdate = 1'b1;
187        Branch = 1'b0;
188        RegWrite = 1'b0;
189        MemWrite = 1'b0;
190        IRWrite = 1'b0;
191        AdrSrc = 1'b0;
192     end
193
194     s10: begin // BEQ
195        ALUSrcA = 2'b10;
196        ALUSrcB = 2'b00;
197        ALUOp = 2'b01;
198        ResultSrc = 2'b00;
199        Branch = 1'b1;
200        PCUpdate = 1'b0;
201        RegWrite = 1'b0;
202        MemWrite = 1'b0;
203        IRWrite = 1'b0;
204        AdrSrc = 1'b0;
205     end
206
207     default: begin
208        ALUSrcA = 2'b00;
209        ALUSrcB = 2'b00;
210        ALUOp = 2'b00;
```

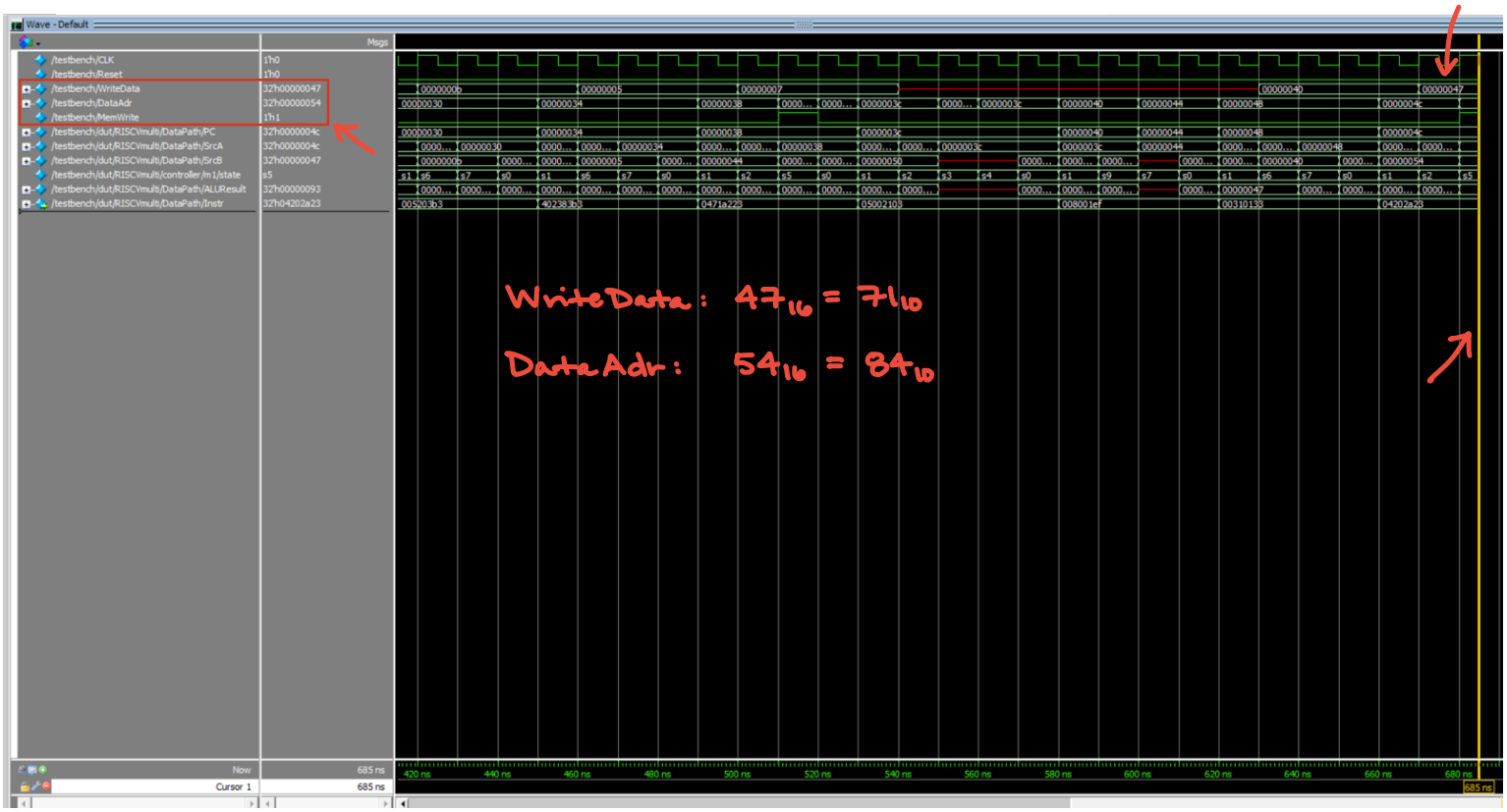
```
211       ResultSrc = 2'b00;
212       Branch = 1'b0;
213       PCUpdate = 1'b0;
214       RegWrite = 1'b0;
215       MemWrite = 1'b0;
216       IRWrite = 1'b0;
217       AdrSrc = 1'b0;
218    end
219    endcase
220    end
221    endmodule
222
```

```
// DataPath (+Register File)

module DataPath(input  logic        clk, reset,
                input  logic [1:0]  ResultSrc,
                input  logic [1:0]  ALUSrcA, ALUSrcB,
                input  logic        RegWrite,
                input  logic [1:0]  ImmSrc,
                input  logic [2:0]  ALUControl,
                input  logic [31:0] ReadData,
                input  logic        PCWrite,
                input  logic        AdrSrc,
                input  logic        IRWrite,
                output logic        zero,
                output logic [31:0] Instr,
                output logic [31:0] WriteData,
                output logic [31:0] Adr);

   // Internal Logic
   logic [31:0] rd1, rd2, A, PC, PCOld, Data, ImmExt, ALUResult, ALUOut, Result, SrcA, SrcB;

   // Section 1: PCNext Logic
   mux3 #(32) pcnextmux(ALUOut, Data, ALUResult, ResultSrc, Result); // d0, d1, d2, s, y
   flopren #(32) pcregen1(clk, reset, PCWrite, Result, PC); // Order is important here. From Module: clk, reset, EN, d (input), q (output).
   mux2 #(32) pcmux(PC, Result, AdrSrc, Adr);

   // Section 2: PCOld Logic
   flopren #(32) pcregen2a(clk, reset, IRWrite, PC, PCOld); // 2a takes care of 1st output: PC --> PCOld
   flopren #(32) pcregen2b(clk, reset, IRWrite, ReadData, Instr); // 2b takes care of 2nd output: ReadData --> Instr
   flopr #(32) rdreg(clk, reset, ReadData, Data);

   // Section 3: Register File Logic
   RegFile RF(clk, RegWrite, Instr[19:15], Instr[24:20], Instr[11:7], Result, rd1, rd2); // Order: clk, WE3, A1, A2, A3, WD3, RD1, RD2
   Extend Ext(Instr[31:7], ImmSrc, ImmExt);
   flopr #(32) rfregA(clk, reset, rd1, A);
   flopr #(32) rfregB(clk, reset, rd2, WriteData);

   // Section 4: ALU Logic
   mux3 #(32) SrcAmux(PC, PCOld, A, ALUSrcA, SrcA);
   mux3 #(32) SrcBmux(WriteData, ImmExt, 'd4, ALUSrcB, SrcB); // +4 for signal ALUSrcB: 10.
   alu alu(SrcA, SrcB, ALUControl, ALUResult, zero);
   flopr #(32) alureg(clk, reset, ALUResult, ALUOut); // Followed by mux3 in Section 1.

endmodule

// MODULES DECLARATIONS (Building Blocks)
// Register File Module [1]
module RegFile(input  logic        clk,
               input  logic        WE3,
               input  logic [ 4:0] A1, A2, A3,
               input  logic [31:0] WD3,
               output logic [31:0] RD1, RD2);

  logic [31:0] rf[31:0];

  // three ported register file
  // read two ports combinationally (A1/RD1, A2/RD2)
  // write third port on rising edge of clock (A3/WD3/WE3)
  // register 0 hardwired to 0

  always_ff @(posedge clk)
    if (WE3) rf[A3] <= WD3;

  assign RD1 = (A1 != 0) ? rf[A1] : 0;
  assign RD2 = (A2 != 0) ? rf[A2] : 0;
endmodule

```

```
67  // Sign Extender Module [1]
68  module Extend(input  logic [31:7] Instr,
69                input  logic [1:0]  ImmSrc,
70                output logic [31:0] ImmExt);
71
72    always_comb
73      case(ImmSrc)
74                  // I-type
75        2'b00:    ImmExt = {{20{Instr[31]}}, Instr[31:20]};
76                  // S-type (Stores)
77        2'b01:    ImmExt = {{20{Instr[31]}}, Instr[31:25], Instr[11:7]};
78                  // B-type (Branches)
79        2'b10:    ImmExt = {{20{Instr[31]}}, Instr[7], Instr[30:25], Instr[11:8], 1'b0};
80                  // J-type (Jumps)
81        2'b11:    ImmExt = {{12{Instr[31]}}, Instr[19:12], Instr[20], Instr[30:21], 1'b0};
82        default: ImmExt = 32'bx; // undefined
83      endcase
84  endmodule
85
86  // Regular Flopr Module [3]
87  module flopr #(parameter WIDTH = 8)
88                (input  logic             clk, reset,
89                 input  logic [WIDTH-1:0] d,
90                 output logic [WIDTH-1:0] q);
91
92    always_ff @(posedge clk, posedge reset)
93      if (reset) q <= 0;
94      else       q <= d;
95
96   endmodule
97
98  // Enable Flopr Module [2]
99  module flopren #(parameter WIDTH = 8)
100               (input  logic             clk, reset, enable,
101                input  logic [WIDTH-1:0] d,
102                output logic [WIDTH-1:0] q);
103
104   always_ff @(posedge clk, posedge reset)
105     if (reset) q <= 0;
106     else if (enable) q <= d;
107
108 endmodule
109
110 // 2-Input Mux [1]
111 module mux2 #(parameter WIDTH = 8)
112              (input  logic [WIDTH-1:0] d0, d1,
113               input  logic             s,
114               output logic [WIDTH-1:0] y);
115
116   assign y = s ? d1 : d0;
117 endmodule
118
119 // 3-Input Mux [3]
120 module mux3 #(parameter WIDTH = 8)
121              (input  logic [WIDTH-1:0] d0, d1, d2,
122               input  logic [1:0]       s,
123               output logic [WIDTH-1:0] y);
124
125   assign y = s[1] ? d2 : (s[0] ? d1 : d0);
126 endmodule
127
128 // ALU [1]
129 module alu(input  logic [31:0] a, b,
130            input  logic [2:0]  alucontrol,
131            output logic [31:0] result,
132            output logic        zero);
133
134   logic [31:0] condinvb, sum;
135   logic        sub;
136
```

```
137     assign sub = (alucontrol[1:0] == 2'b01);
138     assign condinvb = sub ? ~b : b; // for subtraction or slt
139     assign sum = a + condinvb + sub;
140
141     always_comb
142       case (alucontrol)
143         3'b000: result = sum;          // addition
144         3'b001: result = sum;          // subtraction
145         3'b010: result = a & b;        // and
146         3'b011: result = a | b;        // or
147         3'b101: result = sum[31];      // slt
148         default: result = 0;
149     endcase
150
151     assign zero = (result == 32'b0);
152   endmodule
```

```
1   // UnifiedMemory (Instruction/Data Memory)
2
3   module UnifiedMemory(input    logic    clk,
4                        input    logic    MemWrite,
5                        input    logic    [31:0] Adr,
6                        input    logic    [31:0] WriteData,
7                        output   logic    [31:0] ReadData);
8
9     logic [31:0] RAM[63:0];
10
11    initial
12      $readmemh("memfile.dat",RAM);
13
14    assign ReadData = RAM[Adr[31:2]]; // word aligned
15
16    always_ff @(posedge clk)
17      if (MemWrite) RAM[Adr[31:2]] <= WriteData;
18
19  endmodule
20
21
22
```



Transcript

```
# Loading work.instrdecoder(fast)
# Loading work.mainfsm(fast)
# Loading work.DataPath(fast)
# Loading work.mux3(fast)
# Loading work.flopren(fast)
# Loading work.mux2(fast)
# Loading work.flopr(fast)
# Loading work.RegFile(fast)
# Loading work.Extend(fast)
# Loading work.alu(fast)
# Loading work.UnifiedMemory(fast)
VSIM 52> run 1000
# Simulation succeeded
# ** Note: $stop    : C:/Users/dheredia/Desktop/Lab11_DSH/testbench.sv(30)
#    Time: 685 ns  Iteration: 1  Instance: /testbench
# Break in Module testbench at C:/Users/dheredia/Desktop/Lab11_DSH/testbench.sv line 30

VSIM 53>
```