Source File

Design Sources:

1. Instruction Memory
2. `timescale 1ns / 1ps
3. /*
4. * Module: InstructionMemory
5. *
6. * Implements read-only instruction memory
7. * Memory contents are initialized from the file "ImemInit.v"
8. */
9. module InstructionMemory(Data, Address);
10. parameter T_rd = 20;
11. parameter MemSize = 40;
12.
13. output [31:0] Data;
14. input [31:0] Address;
15. reg [31:0] Data;
16.
17. /*
18. * ECEN 651 Processor Test Functions
19. * Texas A&M University
20. */
21.
22. always @ (Address) begin
23. case(Address)
24. /*
25. * Test Program 1:
26. * Sums $a0 words starting at $a1. Stores the sum at the end of the array
27. * Tests add, addi, lw, sw, beq
28. */
29.
30. /*
31. main:
32. li $t0, 50 # Initialize the array to (50, 40, 30)
33. sw $t0, 0($0) # Store first value
34. li $t0, 40
35. sw $t0, 4($0) # Store Second Value
36. li $t0, 30
37. sw $t0, 8($0) # Store Third Value
38. li $a0, 0 # address of array

```
                    li $a1, 3                   # 3 values to sum
TestProg1:
                    add $t0, $0, $0             # This is the sum
                    add $t1, $0, $a0            # This is our array pointer
                    add $t2, $0, $0             # This is our index counter
P1Loop:             beq $t2, $a1, P1Done # Our loop
                    lw      $t3, 0($t1)         # Load Array[i]
                    add $t0, $t0, $t3           # Add it into the sum
                    add $t1, $t1, 4             # Next address
                    add $t2, $t2, 1             # Next index
                    j P1Loop                    # Jump to loop
P1Done:             sw $t0, 0($t1)              # Store the sum at end of array
                    lw $t0, 12($0)              # Load Final Value
                    nop                         # Complete
                    add $0, $s0, $s0        # do nothing
                */
                32'h00: Data = 32'h34080032;
                32'h04: Data = 32'hac080000;
                32'h08: Data = 32'h34080028;
                32'h0C: Data = 32'hac080004;
                32'h10: Data = 32'h3408001e;
                32'h14: Data = 32'hac080008;
                32'h18: Data = 32'h34040000;
                32'h1C: Data = 32'h34050003;
                32'h20: Data = 32'h00004020;
                32'h24: Data = 32'h00044820;
                32'h28: Data = 32'h00005020;
                32'h2C: Data = 32'h11450005;
                32'h30: Data = 32'h8d2b0000;
                32'h34: Data = 32'h010b4020;
                32'h38: Data = 32'h21290004;
                32'h3C: Data = 32'h214a0001;
                32'h40: Data = 32'h0800000b;
                32'h44: Data = 32'had280000;
                32'h48: Data = 32'h8c08000c;
                32'h4C: Data = 32'h00000000;
                32'h50: Data = 32'h02100020;

```

```
77.             /*
78.              * Test Program 2:
79.              * Does some arithmetic computations and stores result in memory
80.              */
81.
82.             /*
83.             main2:
84.                             li      $a0, 32                         # Address
    of memory to store result
85.             TestProg2:
86.                             addi $2, $0, 1                  # $2 = 1
87.                             sub     $3, $0, $2              # $3 = -1
88.                             slt     $5, $3, $0              # $5 = 1
89.                             add     $6, $2, $5      # $6 = 2
90.                             or      $7, $5, $6                      # $7 = 3
91.                             sub     $8, $5, $7              # $8 = -2
92.                             and     $9, $8, $7              # $9 = 2
93.                             sw      $9, 0($a0)                      # Store $9
    in DMem[8]
94.                             lw  $9, 32($0)              # Load Final Value
95.                             nop                                     #
    Complete
96.             */
97.                     32'h60: Data = 32'h34040020;
98.                     32'h64: Data = 32'h20020001;
99.                     32'h68: Data = 32'h00021822;
100.                            32'h6C: Data = 32'h0060282a;
101.                            32'h70: Data = 32'h00453020;
102.                            32'h74: Data = 32'h00a63825;
103.                            32'h78: Data = 32'h00a74022;
104.                            32'h7C: Data = 32'h01074824;
105.                            32'h80: Data = 32'hac890000;
106.                            32'h84: Data = 32'h8c090020;
107.                            32'h88: Data = 32'h00000000;
108.
109.                    /*
110.                     * Test Program 3
111.                     * Test Immediate Function
112.                     */
113.
114.                    /*
115.                                    TestProg3:
116.                                    li $a0, 0xfeedbeef              # $a0 =
    0xfeedbeef
117.                                    sw $a0, 36($0)                  # Store $a0
    in DMem[9]
```

```
addi $a1, $a0, -2656        # $a1 =
    0xfeedb48f
sw $a1, 40($0)                  # Store $a1
    in DMem[10]
addiu $a1, $a0, -2656       # $a1 =
    0xfeeeb48f
sw $a1, 44($0)                  # Store $a1
    in DMem[11]
andi $a1, $a0, 0xf5a0       # $a1 = 0xb4a0
sw $a1, 48($0)                  # Store $a1
    in DMem[12]
sll $a1, $a0, 5             # $a1 =
    0xddb7dde0
sw $a1, 52($0)                  # Store $a1
    in DMem[13]
srl $a1, $a0, 5             # $a1 =
    0x07f76df7
sw $a1, 56($0)                  # Store $a1
    in DMem[14]
sra $a1, $a0, 5             # $a1 = 0xfff76df7
sw $a1, 60($0)                  # Store $a1
    in DMem[15]
slti $a1, $a0, 1            # $a1 = 1
sw $a1, 64($0)                  # Store $a1
    in DMem[16]
slti $a1, $a1, -1               # $a1 = 0
sw $a1, 68($0)                  # Store $a1
    in DMem[17]
sltiu $a1, $a0, 1               # $a1 = 0
sw $a1, 72($0)                  # Store $a1
    in DMem[18]
sltiu $a1, $a1, -1          # $a1 = 1
sw $a1, 76($0)                  # Store $a1
    in DMem[19]
xori $a1, $a0, 0xf5a0  # $a1 = 0xfeed4b4f
sw $a1, 80($0)                  # Store $a1
    in DMem[20]
lw $a0, 36($0)                  # Load
    Value to test
lw $a1, 40($0)                  # Load
    Value to test
lw $a1, 44($0)                  # Load
    Value to test
lw $a1, 48($0)                  # Load
    Value to test
```

```
144.                                    lw $a1, 52($0)          # Load
    Value to test
145.                                    lw $a1, 56($0)          # Load
    Value to test
146.                                    lw $a1, 60($0)          # Load
    Value to test
147.                                    lw $a1, 64($0)          # Load
    Value to test
148.                                    lw $a1, 68($0)          # Load
    Value to test
149.                                    lw $a1, 72($0)          # Load
    Value to test
150.                                    lw $a1, 76($0)          # Load
    Value to test
151.                                    lw $a1, 80($0)          # Load
    Value to test
152.                                    nop
            # Complete
153.                        */
154.                                32'hA0: Data = 32'h3c01feed;
155.                                32'hA4: Data = 32'h3424beef;
156.                                32'hA8: Data = 32'hac040024;
157.                                32'hAC: Data = 32'h2085f5a0;
158.                                32'hB0: Data = 32'hac050028;
159.                                32'hB4: Data = 32'h2485f5a0;
160.                                32'hB8: Data = 32'hac05002c;
161.                                32'hBC: Data = 32'h3085f5a0;
162.                                32'hC0: Data = 32'hac050030;
163.                                32'hC4: Data = 32'h00042940;
164.                                32'hC8: Data = 32'hac050034;
165.                                32'hCC: Data = 32'h00042942;
166.                                32'hD0: Data = 32'hac050038;
167.                                32'hD4: Data = 32'h00042943;
168.                                32'hD8: Data = 32'hac05003c;
169.                                32'hDC: Data = 32'h28850001;
170.                                32'hE0: Data = 32'hac050040;
171.                                32'hE4: Data = 32'h28a5ffff;
172.                                32'hE8: Data = 32'hac050044;
173.                                32'hEC: Data = 32'h2c850001;
174.                                32'hF0: Data = 32'hac050048;
175.                                32'hF4: Data = 32'h2ca5ffff;
176.                                32'hF8: Data = 32'hac05004c;
177.                                32'hFC: Data = 32'h3885f5a0;
178.                                32'h100: Data = 32'hac050050;
179.                                32'h104: Data = 32'h8c040024;
180.                                32'h108: Data = 32'h8c050028;
```

```
                32'h10C: Data = 32'h8c05002c;
                32'h110: Data = 32'h8c050030;
                32'h114: Data = 32'h8c050034;
                32'h118: Data = 32'h8c050038;
                32'h11C: Data = 32'h8c05003c;
                32'h120: Data = 32'h8c050040;
                32'h124: Data = 32'h8c050044;
                32'h128: Data = 32'h8c050048;
                32'h12C: Data = 32'h8c05004c;
                32'h130: Data = 32'h8c050050;
                32'h134: Data = 32'h00000000;

        /*
         * Test Program 4
         * Test jal and jr
         */
        /*
        TestProg4:
                li $t1, 0xfeed          # $t1 = 0xfeed
                li $t0, 0x190           # Load address of P4jr
                jr $t0                  # Jump to P4jr
                li $t1, 0               # Check for failure to jump
P4jr:   sw $t1, 84($0)          # $t1 should be 0xfeed if successful
                li $t0, 0xcafe          # $t0 = 0xcafe
                jal P4Jal               # Jump to P4Jal
                li $t0, 0xbabe          # Check for failure to jump
P4Jal:  sw $t0, 88($0)          # $t0 should be 0xcafe if successful
                li $t2, 0xface          # $t2 = 0xface
                j P4Skip
                        # Jump to P4Skip
                li $t2, 0
P4Skip:         sw $t2, 92($0)          # $t2 should be 0xface if successful
                sw $ra, 96($0)          # Store $ra
                lw $t0, 84($0)          # Load value for check
```

```
                lw $t1, 88($0)          # Load value for check
                lw $t2, 92($0)          # Load value for check
                lw $ra, 96($0)          # Load value for check

        */
                32'h180: Data = 32'h3409feed;
                32'h184: Data = 32'h34080190;
                32'h188: Data = 32'h01000008;
                32'h18C: Data = 32'h34090000;
                32'h190: Data = 32'hac090054;
                32'h194: Data = 32'h3408cafe;
                32'h198: Data = 32'h0c000068;
                32'h19C: Data = 32'h3408babe;
                32'h1A0: Data = 32'hac080058;
                32'h1A4: Data = 32'h340aface;
                32'h1A8: Data = 32'h0800006c;
                32'h1AC: Data = 32'h340a0000;
                32'h1B0: Data = 32'hac0a005c;
                32'h1B4: Data = 32'hac1f0060;
                32'h1B8: Data = 32'h8c080054;
                32'h1BC: Data = 32'h8c090058;
                32'h1C0: Data = 32'h8c0a005c;
                32'h1C4: Data = 32'h8c1f0060;
                32'h1C8: Data = 32'h00000000;


        /*
         * Test Program 5
         * Tests Overflow Exceptions
         */

        /*
        Test5-1:
                li $t0, -2147450880
                add $t0, $t0, $t0
                lw $t0, 4($0)       #incorrect if this instruction completes

        Test5-2:
                li $t0, 2147450879
                add $t0, $t0, $t0
                lw $t0, 4($0)       #incorrect if this instruction completes
```

```
255.
256.                    Test 5-3:
257.                                lw $t0, 4($0)
258.                                li $t0, -2147483648
259.                                li $t1, 1
260.                                sub $t0, $t0, $t1
261.                                lw $t0, 4($0)
262.
263.                    Test 5-4:
264.                                li $t0, 2147483647
265.                                mula $t0, $t0, $t0
266.                                lw $t0, 4($0)
267.                    */
268.                        32'h300: Data = 32'h3c018000;
269.                        32'h304: Data = 32'h34288000;
270.                        32'h308: Data = 32'h01084020;
271.                        32'h30C: Data = 32'h8c080004;
272.
273.                        32'h310: Data = 32'h3c017fff;
274.                        32'h314: Data = 32'h34287fff;
275.                        32'h318: Data = 32'h01084020;
276.                        32'h31C: Data = 32'h8c080004;
277.
278.                        32'h320: Data = 32'h8c080004;
279.                        32'h324: Data = 32'h3c088000;
280.                        32'h328: Data = 32'h34090001;
281.                        32'h32C: Data = 32'h01094022;
282.                        32'h330: Data = 32'h8c080004;
283.
284.                        32'h334: Data = 32'h3c017FFF;
285.                        32'h338: Data = 32'h3428FFFF;
286.                        32'h33C: Data = 32'h01084038;
287.                        32'h340: Data = 32'h8c080004;
288.
289.                    /*
290.                     * Overflow Exception
291.                     */
292.                    /*
293.                                lw $t0, 0($0)
294.                    */
295.                        32'hF0000000: Data = 32'h8c080000;
296.
297.            /*
298.                     * Test Program 6
299.                     * Test Branch Prediction performance
300.                     */
```

```
                                        /*
                                li $t5, 0           # initialize data to 0
                                li $t0, 100         # initialize exit value
                                li $t1, 0           # initialize outer loop index to
    0
                        outer_loop:
                                addi $t1, $t1, 1 #increment outer loop index
                                li $t2, 0       #initialize inner loop index to 0
                        inner_loop:
                                addi $t2, $t2, 1 #increment inner loop index
                                addi $t5, $t5, 1 #increment data
                                bne $t2, $t0, inner_loop #go back to top of
    inner loop
                                bne $t1, $t0, outer_loop #go back to top of
    outer loop
                                sw $t5, 12($0) #store data into memory
                                lw $t5, 12($0) #load data back out of memory
                        */
                32'h500: Data = 32'h240d0000;
                32'h504: Data = 32'h24080064;
                32'h508: Data = 32'h24090000;
                32'h50C: Data = 32'h21290001;
                32'h510: Data = 32'h240a0000;
                32'h514: Data = 32'h214a0001;
                32'h518: Data = 32'h21ad0001;
                32'h51C: Data = 32'h1548fffd;
                32'h520: Data = 32'h1528fffa;
                32'h524: Data = 32'hac0d000c;
                32'h528: Data = 32'h8c0d000c;


                        /*
                                * Test Program 7
                                * Test Branch Prediction performance again
                                */
                                /*
                                li $t5, 0           # initialize data to 0
                                li $t0, 100         # initialize exit value
                                li $t1, 0           # initialize outer loop index to
    0
                        outer_loop:
                                addi $t1, $t1, 1 #increment outer loop index
                                li $t2, 0       #initialize inner loop index to 0
                        inner_loop:
                                addi $t2, $t2, 1 #increment inner loop index
                                andi $t3, $t2, 2 #mask inner loop index
```

```
343.                                    li $t4, 1       #set $t4 to 1
344.                                    beq $t3, $0, skip1
345.                                    li $t4, 0       #set $t4 to 0
346.                            skip1:
347.                                    beq $t4, $0, skip2
348.                                    addi $t5, $t5, 1 #increment data
349.                            skip2:
350.                                    beq $t2, $t1, exit_inner
351.                                    j inner_loop #go back to top of loop
352.                            exit_inner:
353.                                    beq $t1, $t0, exit_outer
354.                                    j outer_loop
355.                            exit_outer:
356.                                    sw $t5, 12($0) #store data into memory
357.                                    lw $t5, 12($0) #load data back out of memory
358.                             */
359.
360.             32'h400: Data = 32'h240d0000;
361.             32'h404: Data = 32'h24080064;
362.             32'h408: Data = 32'h24090000;
363.             32'h40C: Data = 32'h21290001;
364.             32'h410: Data = 32'h240a0000;
365.             32'h414: Data = 32'h214a0001;
366.             32'h418: Data = 32'h314b0002;
367.             32'h41C: Data = 32'h240c0001;
368.             32'h420: Data = 32'h11600001;
369.             32'h424: Data = 32'h240c0000;
370.             32'h428: Data = 32'h11800001;
371.             32'h42C: Data = 32'h21ad0001;
372.             32'h430: Data = 32'h11490001;
373.             32'h434: Data = 32'h08000105;
374.             32'h438: Data = 32'h11280001;
375.             32'h43C: Data = 32'h08000103;
376.             32'h440: Data = 32'hac0d000c;
377.             32'h444: Data = 32'h8c0d000c;
378.
379.
380.                                     default: Data = 32'hXXXXXXXX;
381.                             endcase
382.                     end
383.             Endmodule
```

2. Control unit

```
`timescale 1ns / 1ps
```

```
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    12:23:34 03/10/2009
// Design Name:
// Module Name:    SingleCycleControl
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
`define RTYPEOPCODE 6'b000000
`define LWOPCODE        6'b100011
`define SWOPCODE        6'b101011
`define BEQOPCODE       6'b000100
`define JOPCODE     6'b000010
`define ORIOPCODE       6'b001101
`define ADDIOPCODE  6'b001000
`define ADDIUOPCODE 6'b001001
`define ANDIOPCODE  6'b001100
`define LUIOPCODE       6'b001111
`define SLTIOPCODE  6'b001010
```

```
`define SLTIUOPCODE 6'b001011
`define XORIOPCODE  6'b001110
`define SLLFunc 6'b000000
`define SRLFunc 6'b000010
`define SRAFunc 6'b000011

`define AND     4'b0000
`define OR      4'b0001
`define ADD     4'b0010
`define SLL     4'b0011
`define SRL     4'b0100
`define SUB     4'b0110
`define SLT     4'b0111
`define ADDU    4'b1000
`define SUBU    4'b1001
`define XOR     4'b1010
`define SLTU    4'b1011
`define NOR     4'b1100
`define SRA     4'b1101
`define LUI     4'b1110
`define FUNC    4'b1111

module PipelinedControl(RegDst, ALUSrc_shmt,ALUSrc_immd, MemToReg, RegWrite,
MemRead, MemWrite, Branch, Jump, SignExtend, ALUOp, Opcode,Func);
   input wire [5:0] Opcode;
   output RegDst;
   output ALUSrc_shmt;  //Takes shamt value when high ,else register file value
   output ALUSrc_immd;  //Takes immediate value when high ,else register file value
   output MemToReg;    //To select datamemory/ALU output
   output RegWrite;
   output MemRead;
```

```
  output MemWrite;
  output Branch;
  output Jump;
  output SignExtend;
  output [3:0] ALUOp;
  input wire [5:0] Func;//input functional bits from instruction.


   reg RegDst, ALUSrc_shmt,ALUSrc_immd,MemToReg, RegWrite, MemRead, MemWrite,
Branch, Jump, SignExtend;
   reg  [3:0] ALUOp;


   always @ (Opcode or Func) begin
      case(Opcode)
         `RTYPEOPCODE: begin
          if ((Func == `SLLFunc)| (Func == `SRLFunc)| (Func == `SRAFunc))begin
            RegDst <=  1'b1;
            ALUSrc_shmt <=  1'b1;   //FOr shift functions, use it as shamt instead of register
            ALUSrc_immd <=  1'b0;
            MemToReg <= 1'b0;
            RegWrite <=  1'b1;
            MemRead <=  1'b0;
            MemWrite <= 1'b0;
            Branch <= 1'b0;
            Jump <= 1'b0;
            SignExtend <= 1'b0;
            ALUOp <= `FUNC;
         end
         else begin
            RegDst <= 1'b1;
            ALUSrc_shmt <= 1'b0;
            ALUSrc_immd <= 1'b0;
```

```
    MemToReg <= 1'b0;
    RegWrite <= 1'b1;
    MemRead <= 1'b0;
    MemWrite <= 1'b0;
    Branch <= 1'b0;
    Jump <= 1'b0;
    SignExtend <= 1'b0;
    ALUOp <= `FUNC;
    end
    end
`LWOPCODE: begin
    RegDst <=  1'b0;
    ALUSrc_shmt <=  1'b0;
    ALUSrc_immd <=  1'b1;    // Use immediate address
    MemToReg <= 1'b1;
    RegWrite <=  1'b1;
    MemRead <=  1'b1;
    MemWrite <= 1'b0;
    Branch <= 1'b0;
    Jump <= 1'b0;
    SignExtend <= 1'b1;
    ALUOp <= `ADD;
end
`SWOPCODE: begin
    RegDst <=  1'b0;
    ALUSrc_shmt <=  1'b0;  //NO need to shift this
    ALUSrc_immd <=  1'b1;  //Should take as immediate value
    MemToReg <= 1'b1;
    RegWrite <=  1'b0;
    MemRead <=  1'b0;
```

```
    MemWrite <= 1'b1;
    Branch <= 1'b0;
    Jump <= 1'b0;
    SignExtend <= 1'b1;
    ALUOp <= `ADD;
end
`BEQOPCODE: begin
    RegDst <=  1'b0;
    ALUSrc_shmt <=  1'b0;  //No need of shamt in branch
    ALUSrc_immd <=  1'b0;  //Equality comparison for branch is done in register file
    MemToReg <= 1'b0;
    RegWrite <=  1'b0;
    MemRead <=  1'b0;
    MemWrite <= 1'b0;
    Branch <= 1'b1;
    Jump <= 1'b0;
    SignExtend <= 1'b1;
    ALUOp <= `SUB;
end
`JOPCODE: begin
    RegDst <=  1'b0;
    ALUSrc_shmt <=  1'b0;//No need of shamt in jump
    ALUSrc_immd <=  1'b0;
    MemToReg <= 1'b0;
    RegWrite <=  1'b0;
    MemRead <=  1'b0;
    MemWrite <= 1'b0;
    Branch <= 1'b0;
    Jump <= 1'b1;
    SignExtend <= 1'b1;
```

```
    ALUOp <= `AND;
  end
  `ORIOPCODE: begin
    RegDst <=  1'b0;
    ALUSrc_shmt <=  1'b0;
    ALUSrc_immd <=  1'b1;  //immediate required here
    MemToReg <= 1'b0;
    RegWrite <=  1'b1;
    MemRead <=  1'b0;
    MemWrite <= 1'b0;
    Branch <= 1'b0;
    Jump <= 1'b0;
    SignExtend <= 1'b0;
    ALUOp <= `OR;
  end
  `ADDIOPCODE: begin
    RegDst <=  1'b0;
    ALUSrc_shmt <=  1'b0;
    ALUSrc_immd <=  1'b1;  //immediate taken here
    MemToReg <= 1'b0;
    RegWrite <=  1'b1;
    MemRead <=  1'b0;
    MemWrite <= 1'b0;
    Branch <= 1'b0;
    Jump <= 1'b0;
    SignExtend <= 1'b1;
    ALUOp <= `ADD;
  end
  `ADDIUOPCODE: begin
    RegDst <=  1'b0;
```

```
    ALUSrc_shmt <=  1'b0;
    ALUSrc_immd <=  1'b1;
    MemToReg <= 1'b0;
    RegWrite <=  1'b1;
    MemRead <=  1'b0;
    MemWrite <= 1'b0;
    Branch <= 1'b0;
    Jump <= 1'b0;
    SignExtend <= 1'b0;
    ALUOp <= `ADDU;
end
`ANDIOPCODE: begin
    RegDst <=  1'b0;
    ALUSrc_shmt <=  1'b0;
    ALUSrc_immd <=  1'b1;
    MemToReg <= 1'b0;
    RegWrite <=  1'b1;
    MemRead <=  1'b0;
    MemWrite <= 1'b0;
    Branch <= 1'b0;
    Jump <= 1'b0;
    SignExtend <= 1'b0;
    ALUOp <= `AND;
end
`LUIOPCODE: begin
    RegDst <=  1'b0;
    ALUSrc_shmt <=  1'b0;
    ALUSrc_immd <=  1'b1;
    MemToReg <= 1'b0;
    RegWrite <=  1'b1;
```

```
    MemRead <=  1'b0;
    MemWrite <= 1'b0;
    Branch <= 1'b0;
    Jump <= 1'b0;
    SignExtend <= 1'b0;
    ALUOp <= `LUI;
end
`SLTIOPCODE: begin
    RegDst <=  1'b0;
    ALUSrc_shmt <=  1'b0;
    ALUSrc_immd <=  1'b1;
    MemToReg <= 1'b0;
    RegWrite <=  1'b1;
    MemRead <=  1'b0;
    MemWrite <= 1'b0;
    Branch <= 1'b0;
    Jump <= 1'b0;
    SignExtend <= 1'b1;
    ALUOp <= `SLT;
end
`SLTIUOPCODE: begin
    RegDst <=  1'b0;
    ALUSrc_shmt <=  1'b0;
    ALUSrc_immd <=  1'b1;
    MemToReg <= 1'b0;
    RegWrite <=  1'b1;
    MemRead <=  1'b0;
    MemWrite <= 1'b0;
    Branch <= 1'b0;
    Jump <= 1'b0;
```

```
            SignExtend <= 1'b1;
            ALUOp <= `SLTU;
        end
        `XORIOPCODE: begin
            RegDst <=  1'b0;
            ALUSrc_shmt <=  1'b0;
            ALUSrc_immd <=  1'b1;
            MemToReg <= 1'b0;
            RegWrite <=  1'b1;
            MemRead <=  1'b0;
            MemWrite <= 1'b0;
            Branch <= 1'b0;
            Jump <= 1'b0;
            SignExtend <= 1'b0;
            ALUOp <= `XOR;
        end
        default: begin
            RegDst <= 1'bx;
            ALUSrc_shmt <= 1'bx;
            ALUSrc_immd <=  1'b1;
            MemToReg <= 1'bx;
            RegWrite <= 1'bx;
            MemRead <= 1'bx;
            MemWrite <= 1'bx;
            Branch <= 1'bx;
            Jump <= 1'bx;
            SignExtend <= 1'bx;
            ALUOp <= 4'bxxxx;
        end
    endcase
```

end

endmodule

3.Register File

```
3.  `timescale 1ns / 1ps
4.  //////////////////////////////////////////////////////////////////////////////////
5.  // Company:
6.  // Engineer:
7.  //
8.  // Create Date: 09/18/2019 08:40:41 AM
9.  // Design Name:
10. // Module Name: RegisterFile
11. // Project Name:
12. // Target Devices:
13. // Tool Versions:
14. // Description:
15. //
16. // Dependencies:
17. //
18. // Revision:
19. // Revision 0.01 - File Created
20. // Additional Comments:
21. //
22. //////////////////////////////////////////////////////////////////////////////////
23.
24.
25. module RegisterFile(
26.     output  [31:0] BusA,
27.     output  [31:0] BusB,
28.     input [31:0] BusW,
29.     input [4:0] RA,
30.     input [4:0] RB,
31.     input [4:0] RW,
32.     input RegWr,
33.     input Clk
34.     );
35.     reg [31:0]registers[31:0];
36.
37.
38.    assign BusA = (RA==0)? 32'd0 :registers[RA] ;     //Asynchronous read of RA.
39.    assign BusB = (RB==0)? 32'd0 :registers[RB];      //Asynchronous read of RB.
40.
41.     always@(posedge Clk)              //Storage at pos egde of clk to allow bypassing
42.     begin
43.         if (RegWr && RW!=5'd0)
44.         begin
```

```
45.             registers[RW] <= BusW;
46.
47.         end
48.     end
49.
50. Endmodule
```

4. Sign Extension

```
51. `timescale 1ns / 1ps
52. //////////////////////////////////////////////////////////////////////////////////
53. // Company:
54. // Engineer:
55. //
56. // Create Date: 10/04/2019 07:33:25 PM
57. // Design Name:
58. // Module Name: SignExtender
59. // Project Name:
60. // Target Devices:
61. // Tool Versions:
62. // Description:
63. //
64. // Dependencies:
65. //
66. // Revision:
67. // Revision 0.01 - File Created
68. // Additional Comments:
69. //
70. //////////////////////////////////////////////////////////////////////////////////
71.
72.
73. module SignExtender(
74.     output [31:0] signex,
75.     input SignExtend,
76.     input [15:0] Instruction
77.     );
78. wire signbit;
79. assign signbit = SignExtend==1'b1 ? Instruction[15] : 0;
80. assign signex={{16{signbit}},Instruction[15:0]};
81.
82. Endmodule
```

5.ALU Control

```
83. `timescale 1ns / 1ps
84. //////////////////////////////////////////////////////////////////////////////////
85. // Company:
86. // Engineer:
```

```
87. //
88. // Create Date: 10/02/2019 08:12:01 AM
89. // Design Name:
90. // Module Name: ALUControl
91. // Project Name:
92. // Target Devices:
93. // Tool Versions:
94. // Description:
95. //
96. // Dependencies:
97. //
98. // Revision:
99. // Revision 0.01 - File Created
100.        // Additional Comments:
101.        //
102.        //////////////////////////////////////////////////////////////////////////////////
103.
104.        `define SLLFunc  6'b000000
105.        `define SRLFunc  6'b000010
106.        `define SRAFunc  6'b000011
107.        `define ADDFunc  6'b100000
108.        `define ADDUFunc 6'b100001
109.        `define SUBFunc  6'b100010
110.        `define SUBUFunc 6'b100011
111.        `define ANDFunc  6'b100100
112.        `define ORFunc   6'b100101
113.        `define XORFunc  6'b100110
114.        `define NORFunc  6'b100111
115.        `define SLTFunc  6'b101010
116.        `define SLTUFunc 6'b101011
117.
118.        `define AND 4'b0000
119.        `define OR 4'b0001
120.        `define ADD 4'b0010
121.        `define SLL 4'b0011
122.        `define SRL 4'b0100
123.        `define SUB 4'b0110
124.        `define SLT 4'b0111
125.        `define ADDU 4'b1000
126.        `define SUBU 4'b1001
127.        `define XOR 4'b1010
128.        `define SLTU 4'b1011
129.        `define NOR 4'b1100
130.        `define SRA 4'b1101
131.        `define LUI 4'b1110
132.
```

```
133.        module ALUControl(ALUCtrl, ALUop, FuncCode);
134.        output reg [3:0]ALUCtrl;
135.        input [3:0]ALUop;
136.        input [5:0]FuncCode;
137.
138.        always@(*)
139.        begin
140.           case({ALUop,FuncCode})
141.           {4'b1111,`SLLFunc}: ALUCtrl<= `SLL;
142.           {4'b1111,`SRLFunc}: ALUCtrl<= `SRL;
143.           {4'b1111,`SRAFunc}: ALUCtrl<= `SRA;
144.           {4'b1111,`ADDFunc}: ALUCtrl<= `ADD;
145.           {4'b1111,`ADDUFunc}: ALUCtrl<= `ADDU;
146.           {4'b1111,`SUBFunc}: ALUCtrl<= `SUB;
147.           {4'b1111,`SUBUFunc}: ALUCtrl<= `SUBU;
148.           {4'b1111,`ANDFunc}: ALUCtrl<= `AND;
149.           {4'b1111,`ORFunc}: ALUCtrl<= `OR;
150.           {4'b1111,`XORFunc}: ALUCtrl<= `XOR;
151.           {4'b1111,`NORFunc}: ALUCtrl<= `NOR;
152.           {4'b1111,`SLTFunc}: ALUCtrl<= `SLT;
153.           {4'b1111,`SLTUFunc}: ALUCtrl<= `SLTU;
154.           default: ALUCtrl <= ALUop;
155.           endcase
156.        end
157.
158.        Endmodule
159.
```

6.ALU

```
`timescale 1ns / 1ps



`define AND 4'b0000
`define OR 4'b0001
`define ADD 4'b0010
`define SLL 4'b0011
`define SRL 4'b0100
`define SUB 4'b0110
`define SLT 4'b0111
`define ADDU 4'b1000
`define SUBU 4'b1001
`define XOR 4'b1010
`define SLTU 4'b1011
`define NOR 4'b1100
`define SRA 4'b1101
```

```
`define LUI 4'b1110

module ALU(BusW, Zero, BusA, BusB, ALUCtrl);

input wire [31:0] BusA, BusB;
output reg [31:0] BusW;
input wire [3:0] ALUCtrl ;
output wire Zero ;

wire less;
wire [63:0] Bus64;
assign less = ({1'b0,BusA} < {1'b0,BusB}  ? 1'b1 : 1'b0); //checks which value is
greater.

assign Zero = ((BusW==0)  ? 1'b1 : 1'b0); //checks which value is greater.
//assign Bus64 =


always@(*)begin

        case (ALUCtrl)
        `AND:   BusW <= BusA & BusB;
   `OR:    BusW <= BusA | BusB;
   `ADD:   BusW <= BusA + BusB;
   `ADDU:  BusW <= BusA + BusB;
   `SLL:   BusW <= BusB << BusA;
   `SRL:   BusW <= BusB >> BusA;
   `SUB:   BusW <= BusA - BusB;
   `SUBU:  BusW <= BusA - BusB;
   `XOR:   BusW <= BusA ^ BusB;
   `NOR:   BusW <= ~(BusA|BusB);

        `SLT:   if(BusA[31] != BusB[31])
                begin
                  if(BusA[31] > BusB[31])begin
                   BusW <= 1;
                   end else begin
                   BusW <= 0;
                   end
                   end else begin
                                if (BusA < BusB)
                                begin
                                   BusW <= 1;
                                end
                                else
                                begin
```

```
                    BusW <= 0;
                end
            end


        `SLTU: if (BusA < BusB)
              begin
                BusW <= 1;
               end
                else
              begin
              BusW <= 0;
               end

        `SRA:   BusW <= $signed(BusB)>>>$signed(BusA) ;
        `LUI:   BusW <= BusB<<16;
    default:BusW <= 32'd0;
      endcase
end
endmodule
```

7. Data Memory

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 09/18/2019 09:08:30 AM
// Design Name:
// Module Name: DataMemory
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module DataMemory(ReadData, Address, WriteData, MemoryRead,
MemoryWrite, Clock);
```

```
output reg [31:0] ReadData;
input wire [31:0] WriteData;
input wire [5:0] Address ;

input wire MemoryRead, MemoryWrite ;
input wire Clock;

reg [31:0] regs [63:0];

integer i;
  initial begin
  for (i = 0; i < 32; i = i + 1) begin
              regs[i] <= 0;
            end
            end

always @ (negedge Clock) begin
 if(MemoryWrite) begin
   regs[Address] <= WriteData;
   end
end

 always @ (posedge Clock) begin
   if(MemoryRead) begin
   ReadData = regs[Address];
     //regs[Address] <= WriteData;
     end
  end


endmodule
```

8. Forwarding Unit

```
module ForwardingUnit(UseShamt, UseImmed, ID_Rs, ID_Rt, EX_Rw,
MEM_Rw, EX_RegWrite, MEM_RegWrite, AluOpCtrlA, AluOpCtrlB,
                DataMemForwardCtrl_EX, DataMemForwardCtrl_MEM);
input  UseShamt, UseImmed;
input [4:0] ID_Rs, ID_Rt, EX_Rw, MEM_Rw;
input  EX_RegWrite, MEM_RegWrite;
output reg [1:0] AluOpCtrlA, AluOpCtrlB;
output reg DataMemForwardCtrl_EX, DataMemForwardCtrl_MEM;

//wire [1:0] AluOpCtrlA, AluOpCtrlB;
```

```
always @ ( * ) begin

   // determining forwarding control signal for ALU forwardA
   if (UseShamt)
   AluOpCtrlA <= 2'b0;    //OUTPUT A = 00

   else if (EX_RegWrite && (EX_Rw != 0) && (EX_Rw == ID_Rs))
   AluOpCtrlA <= 2'd2; // OUTPUR A = 10

   else if (MEM_RegWrite && (MEM_Rw!=0) && (MEM_Rw == ID_Rs))
   AluOpCtrlA <= 2'd1; // 01

   else
   AluOpCtrlA=2'd3;  //OUTPUT A = 11



   // determining forwarding control signal for ALU forwardB
   if(UseImmed)
   AluOpCtrlB <= 2'd0;

   else if (EX_RegWrite && (ID_Rt==EX_Rw) && (EX_Rw!=0) )
   AluOpCtrlB <= 2'd2;

   else if ( (MEM_RegWrite) && (ID_Rt == MEM_Rw) && (MEM_Rw!=0) )
   AluOpCtrlB <= 2'd1;

   else
   AluOpCtrlB <= 2'd3;
 end

     //FOR DATA MEMORY

     always @(*) begin
           if(EX_RegWrite && (EX_Rw == ID_Rt) && (EX_Rw!=0)) begin
              DataMemForwardCtrl_MEM = 1;
              DataMemForwardCtrl_EX = 0;
           end
           else if(MEM_RegWrite && (MEM_Rw == ID_Rt ) &&
(MEM_Rw!=0))begin
              DataMemForwardCtrl_MEM = 0; //default
              DataMemForwardCtrl_EX = 1;
           end

           else begin
```

```
                DataMemForwardCtrl_EX = 0; //default
                DataMemForwardCtrl_MEM = 0;
            end

        end


    endmodule
```

9. Hazard Unit

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/10/2019 11:59:40 PM
// Design Name:
// Module Name: HazardUnit
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module HazardUnit (IF_write, PC_write, bubble, addrSel, Jump, Branch, ALUZero,
                   memReadEX, currRs, currRt, prevRt, UseShamt, UseImmed, Clk, Rst);
```

```
output reg IF_write, PC_write, bubble;

output reg [1:0] addrSel;

input Jump, Branch, ALUZero, memReadEX, Clk, Rst;

input UseShamt, UseImmed;

input [4:0] currRs, currRt, prevRt;


reg  LdHazard;  // TO INDICATE A HAZARD CONDITON THAT REQUIRES STALLING

reg [1:0] FSM, FSM_next; // States for FSM


parameter NoHazard_state = 2'b00;

parameter Jump_state = 2'b01;

parameter Branch_0_state = 2'b10; //branch taken

parameter Branch_1_state = 2'b11; //branch not taken


always @(*) begin

// Logic to detect if hazard is there or not

if(memReadEX && prevRt!= 5'd0) begin // rt from execute should be there along with mem_read signal High


   if(prevRt == currRs) begin   //if output of execute stage equals either of source registers from ID stage, hazard is present

     LdHazard <=1;

   end


    else begin


      if ((UseShamt==0) && (UseImmed==0)) begin // When these are high we dont forward data. When low ,we need to handle hazard comdition

            if (currRt==prevRt)            // If UseImmediate is high, then register is not used hence hazard wont be there
```

```
                                    //If Useshamt is used then rs is not used, rt will be used
                LdHazard<=1;
            else
                LdHazard<=0;
        end

      else begin
            LdHazard <=0;
      end

   end     // end of else
end      // end of 1st If
  else begin
     LdHazard<=0;
   end
end     // end of always


always @(negedge Clk) begin
    if (Rst==0)
      FSM <=0;              // Reset FSM
    else
      FSM <=FSM_next;      //Point to next state
end


always @(*) begin
case (FSM)

NoHazard_state: begin
```

```
                    if (Jump) begin // ID STALL // Make IF WRITE LOW to stall ID
                    {IF_write, PC_write, bubble, addrSel} = 5'b01001; // addrsel==01
                    FSM_next = 2'b01; //This is the jump state
                    end

                    else if (LdHazard) begin//if hazard detected, stop fetching new IF and stop incrementing the PC and writing into pC
                    {IF_write, PC_write, bubble, addrSel} = 5'b00100; //addrsel == 00
                    FSM_next = 2'b00; // No Hazard State
                    end

                    else if (Branch) begin //IF STALL
                    {IF_write, PC_write, bubble, addrSel} = 5'b00000;//as per the flow chart, go to branch zero and check for alu zero flag. Later resolve it back to normal
                    FSM_next = 2'b10;  //Branch Taken
                    end

                    else begin
                    {IF_write, PC_write, bubble, addrSel} = 5'b11000;//normal state
                    FSM_next = 2'b00;  // No Hazard State
                    end

                    end

Jump_state: begin
                    {IF_write, PC_write, bubble, addrSel} = 5'b11100;//stopping execution until jump is resolved completely in nohazard unit
```

```
                    FSM_next = 2'b00; //No Hazard state
                    end

Branch_0_state: begin
                        if (!ALUZero)begin //branch not equal result
                        {IF_write, PC_write, bubble, addrSel} = 5'b11100;
                        FSM_next = 2'b00;  // No Hazard State
                        end
                        else if (ALUZero) begin // branch equal result.
                        {IF_write, PC_write, bubble, addrSel} =
5'b01110;//stopping the IF stage incase branch gets detected. Assuming it to be BEQ
and not BNEQ
                        FSM_next = 2'b11; //branch not taken
                        end
                        end

Branch_1_state: begin
                        {IF_write, PC_write, bubble, addrSel} =
5'b11100;//stopping execution until branch in completely resolved in no hazard unit.
                        FSM_next = 2'b00;  // No Hazard State
                        end

default :       begin
                    FSM_next = 2'b00;  // No Hazard State
                    PC_write = 1'bx;
                    IF_write = 1'bx;
                    bubble = 1'bx;
                    addrSel = 2'bxx;
                    end
endcase
end
```

endmodule

10. Pipelined Proc

```
`timescale 1ns / 1ps



module PipelinedProc(CLK,Reset_L,startPC,dMemOut);

    input CLK;
                input Reset_L;
                input [31:0] startPC;
                output [31:0] dMemOut;



//Hazard
                wire Bubble;
                wire PCWrite;
                wire IFWrite;

                //Stage 1
                wire    [31:0]  currentPCPlus4;
                wire    [31:0]  jumpDescisionTarget;
                wire    [31:0]  nextPC;

                reg     [31:0]  currentPC;
                wire    [31:0]  currentInstruction;

                //Stage 2
                reg     [31:0]  currentInstruction2;
                wire    [5:0]   opcode;
```

```
wire    [4:0]   rs, rt, rd;
wire    [15:0]  imm16;
wire    [4:0]   shamt;
wire    [5:0]   func;
wire    [31:0]  busA, busB, ALUImmRegChoice, signExtImm;
wire    [31:0]  jumpTarget;

//Stage 2 Control Wires
wire    regDst, aluSrc, memToReg, regWrite, memRead, memWrite,
branch, jump, signExtend;
wire    UseShiftField;
wire    rsUsed, rtUsed;
wire    [4:0]   rw;
wire    [3:0]   aluOp;
wire    [31:0]  ALUBIn;
wire    [31:0]  ALUAIn;

//Stage 3
reg     [31:0]  ALUAIn3, ALUBIn3, busB3, signExtImm3;
reg     [4:0]   rw3;
wire    [5:0]   func3;
wire    [31:0]  shiftedSignExtImm;
wire    [31:0]  branchDst;
wire    [3:0]   aluCtrl;
wire    aluZero;
wire    [31:0]  aluOut;

//Stage 3 Control
reg     regDst3, memToReg3, regWrite3, memRead3, memWrite3,
branch3;
reg     [3:0]   aluOp3;
```

```
//Stage 4
reg     aluZero4;
reg     [31:0]  branchDst4, aluOut4, busB4;
reg     [4:0]   rw4;
wire    [31:0]  memOut;

assign dMemOut = memOut;

//Stage 4 Control
reg memToReg4, regWrite4, memRead4, memWrite4, branch4;

//Stage 5
reg     [31:0]  memOut5, aluOut5;
reg     [4:0]   rw5;
wire    [31:0]  regWriteData;

//Stage 5 Control
reg memToReg5, regWrite5;



wire [31:0] Data;
wire [31:0] Address;
//reg [31:0] currentPC;

//output of Hazard Unit
wire [1:0] addrSel; // Mux select for PC
wire IF_write;
wire PC_write;
```

```
wire bubble;


//Control Unit wires
wire RegDst; //selecting the actual bits of register file. If ==1 register file = 15 to 11 else  20 to 16.
wire MemToReg;// data of memory is written if =1 or data of alu is written if =0
wire RegWrite;//used to write to the registers in the register file
wire MemRead;//used to read from the memory
wire MemWrite;//used to write into the memory unit. Data to be written comes from bus B
wire Branch;//used to activate branch control
wire Jump;//used to activate jump controll
wire SignExtend;//used to extend the sign of the immedieate value
wire UseShamt;//used to change the source registers in case of shift operations
wire UseImmed;// used to indicate a immedieate type of instruction.
wire [3:0]ALUOp_IF_ID;//defines the type of operation to tbe performed.



//Forwarding Unit
wire [1:0] AluOpCtrlA_ID;
wire [1:0] AluOpCtrlB_ID;
wire DataMemForwardCtrl_EX_IF_ID;
wire DataMemForwardCtrl_MEM_IF_ID;


//sign extended data
wire[31:0] sign_extender;

//register RF1 wires IF_ID  STAGE
wire [31:0] Register_file_A_IF_ID, Register_file_B_IF_ID;  //outputs of register file
```

```
wire [31:0] BusW_WB;                //outputs of register file
wire [4:0] RA_IF_ID, RB_IF_ID, RW_WB;    //inputs of register file
wire RegWr_WB;              //Regwrite signal from wb stage to write data to register file
wire Clk;



//////////////////////////////
wire[4:0] write_register_select_IF_ID;
wire [31:0] write_register_data_IF_ID;
wire [5:0] Opcode_IF_ID;
wire [4:0] RS_IF_ID;
wire [4:0] RT_IF_ID;
wire [4:0] RD_IF_ID;
wire [5:0] FUNC_IF_ID;
reg [31:0] IM_IF_ID;
wire [25:0] Jump_IF_ID;
wire [15:0] Immedi_IF_ID;
reg  [31:0] PC_Address;
//////////////////////////////////////

// Stage 3 ID -> Execute
reg RegDst_ID_EX;
reg MemToReg_ID_EX;
reg RegWrite_ID_EX;
reg MemRead_ID_EX;
reg MemWrite_ID_EX;
reg [3:0]ALUOp_ID_EX;
reg [1:0] AluOpCtrlA_ID_EX;
reg [1:0] AluOpCtrlB_ID_EX;
reg DataMemForwardCtrl_EX_ID_EX;
```

```
reg DataMemForwardCtrl_MEM_ID_EX;
reg [20:0] IM_20_0_ID_EX;

reg [31:0] Sign_Extended_ID_EX;
reg [31:0] Registers_A_ID_EX;
reg [31:0] Registers_B_ID_EX;

wire [5:0] Funccode_ID_EX;
wire [4:0] RT_ID_EX;
wire [4:0] RD_ID_EX;
wire [4:0] Shamt_ID_EX;
wire [4:0] RW_ID_EX;
wire [31:0] Data_Memory_Input_ID_EX;



//ALU CONTROL
wire [31:0] ALU_OUT;
reg [31:0] ALU_IN1;
reg [31:0] ALU_IN2;
wire ALU_Zero;
wire [3:0] ALU_control;


// Stage 4_ DATA Memory
wire [31:0] Data_memory_out;
reg [31:0] Data_Memory_Input_EX_MEM;
reg [31:0] ALU_OUT_EX_MEM;
reg [4:0] RW_EX_MEM;
```

```
wire [31:0] Data_Memory_actual_in;

reg MemToReg_EX_MEM;
reg RegWrite_EX_MEM;
reg MemRead_EX_MEM;
reg MemWrite_EX_MEM;

reg DataMemForwardCtrl_MEM_EX_MEM;


// stage 5 - WB
reg MemToReg_MEM_WB;
reg RegWrite_MEM_WB;
reg [4:0] RW_MEM_WB;
reg [31:0] DataOut_MEM_WB;
reg [31:0] ALU_OUT_MEM_WB;
wire [31:0] Register_W_MEM_WB;


//Stage 1 - INSTRUCTION FETCH STAGE
////////////////////////////////////////////////////////////////////////////////////////////////

assign nextPC = (addrSel==2'b00) ? currentPCPlus4 :(addrSel == 2'b01) ?
jumpTarget : PC_Address + (Sign_Extended_ID_EX[30:0] << 2);
assign jumpTarget ={PC_Address [31:28], Jump_IF_ID, 2'b0};   // Calculate jump
address PC_Address
assign #1 currentPCPlus4 = currentPC + 4;
// PC COntrol Logic
always @ (negedge CLK or negedge Reset_L)begin
if (~Reset_L)
```

```
currentPC <= startPC;
else if (PC_write)
currentPC <= nextPC;
end

InstructionMemory instrmem(.Data(Data), .Address(currentPC));
//CURRENT PC IS THE INPUT TO INSTRUCTION MEMORY
//Data is the 32 bit instuction fetched

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////

//Stage 2 - REGISTER FILE
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
///////

//SET ADDRESS VALUES FROM INSTRUCTION MEMORY AND UPDATE PC
always @ (negedge CLK or negedge Reset_L) begin
if(~Reset_L) begin
IM_IF_ID <= 32'b0;     //32 BIT ADDRESS FROM INSTRUCTION MEMORY
PC_Address <= 32'b0;   // 32 BIT PC ADDRESS
end
else if(IF_write) begin          // IF WE WANT TO GO TO DECODE STAGE FROM
IF
IM_IF_ID <= Data;                //MAP ADDRESS TO CURRENT INSTRUCTION
PC_Address <= currentPC + 4;      // Increment PC ADDRESS
end
end

// IM_IF_ID HAS THE CURRENT INSTRUCTION ADDRESS
```

```
//ASSIGN VALUES ACCORDING TO INSTRUCTION SET FORMAT

assign Opcode_IF_ID = IM_IF_ID[31:26];  //6BIT OPCPDE

assign FUNC_IF_ID = IM_IF_ID [5:0];     //6 BIT FUNCT

assign RS_IF_ID = IM_IF_ID[25:21];     // 5BIT SOURCE REGISTER RS (RA OF REGISTER FILE) from ID STAGE

assign              RT_IF_ID =   IM_IF_ID[20:16];     // 5BIT SOURCE REGISTER RT (RB OF REGISTER FILE) FROM ID STAGE

assign Jump_IF_ID = IM_IF_ID[25:0];     //DIRECTLY TO ID/EX LATCH

assign Immedi_IF_ID =          IM_IF_ID[15:0];   //INPUT TO SIGN EXTENDER



PipelinedControl controller (.RegDst(RegDst),
.ALUSrc_shmt(UseShamt),.ALUSrc_immd(UseImmed), .MemToReg(MemToReg),
.RegWrite(RegWrite),

.MemRead(MemRead), .MemWrite(MemWrite), .Branch(Branch), .Jump(Jump),
.SignExtend(SignExtend),

.ALUOp(ALUOp_IF_ID), .Opcode(Opcode_IF_ID), .Func(FUNC_IF_ID));


//sign extension

assign sign_extender = SignExtend ? {{16{IM_IF_ID[15]}},IM_IF_ID[15:0]} :
{{16{1'b0}},IM_IF_ID[15:0]}  ;



//hazard unit

HazardUnit hazard (.IF_write(IF_write), .PC_write(PC_write), .bubble(bubble),
.addrSel(addrSel), .Jump(Jump), .Branch(Branch), .ALUZero(ALU_Zero),

.memReadEX(MemRead_ID_EX), .currRs(RS_IF_ID), .currRt(RT_IF_ID),
.prevRt(RT_ID_EX), .UseShamt(UseShamt), .UseImmed(UseImmed), .Clk(CLK),
.Rst(Reset_L));


//Register file

RegisterFile RF (.BusA(Register_file_A_IF_ID), .BusB(Register_file_B_IF_ID),
.BusW(Register_W_MEM_WB),
```

```
.RA(RS_IF_ID), .RB(RT_IF_ID), .RW( RW_MEM_WB ),
.RegWr(RegWrite_MEM_WB), .Clk(CLK));


//Forwarding Unit

ForwardingUnit ForwUt( .UseShamt(UseShamt), .UseImmed(UseImmed),
.ID_Rs(RS_IF_ID),.ID_Rt(RT_IF_ID),

.EX_Rw(RW_ID_EX),.MEM_Rw(RW_EX_MEM),
.EX_RegWrite(RegWrite_ID_EX), .MEM_RegWrite(RegWrite_EX_MEM),

.AluOpCtrlA(AluOpCtrlA_ID), .AluOpCtrlB(AluOpCtrlB_ID),

.DataMemForwardCtrl_EX(DataMemForwardCtrl_EX_IF_ID),
.DataMemForwardCtrl_MEM(DataMemForwardCtrl_MEM_IF_ID) );



////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////////

// STAGE 3 Execute



///DEclaring control unit and REgister File outputs

always @ (negedge CLK or negedge Reset_L) begin

   if(~Reset_L) begin

      RegDst_ID_EX<=1'b0;

      MemToReg_ID_EX<=1'b0;

      RegWrite_ID_EX<=1'b0;

      MemRead_ID_EX<=1'b0;

      MemWrite_ID_EX<=1'b0;

      ALUOp_ID_EX<=4'b0;

      AluOpCtrlA_ID_EX<=2'b0;

      AluOpCtrlB_ID_EX<=2'b0;

      DataMemForwardCtrl_EX_ID_EX<=1'b0;

      DataMemForwardCtrl_MEM_ID_EX<=1'b0;

      IM_20_0_ID_EX<=21'b0;
```

```
    Sign_Extended_ID_EX<=32'b0;
    Registers_A_ID_EX<=32'b0;
    Registers_B_ID_EX<=32'b0;
  end
 else if(bubble) begin                    // Stall the execution
    RegDst_ID_EX<=1'b0;
    MemToReg_ID_EX<=1'b0;
    RegWrite_ID_EX<=1'b0;
    MemRead_ID_EX<=1'b0;
    MemWrite_ID_EX<=1'b0;
    ALUOp_ID_EX<=4'b0;
    AluOpCtrlA_ID_EX<=2'b0;
    AluOpCtrlB_ID_EX<=2'b0;
    DataMemForwardCtrl_EX_ID_EX<=1'b0;
    DataMemForwardCtrl_MEM_ID_EX<=1'b0;
    IM_20_0_ID_EX<=21'b0;
    Sign_Extended_ID_EX<=32'b0;
    Registers_A_ID_EX<=32'b0;
    Registers_B_ID_EX<=32'b0;
   end
 else begin
    RegDst_ID_EX<=RegDst;
    MemToReg_ID_EX<=MemToReg;
    RegWrite_ID_EX<=RegWrite;
    MemRead_ID_EX<=MemRead;
    MemWrite_ID_EX<=MemWrite;
    ALUOp_ID_EX<=ALUOp_IF_ID;
    AluOpCtrlA_ID_EX<=AluOpCtrlA_ID;
    AluOpCtrlB_ID_EX<=AluOpCtrlB_ID;
    DataMemForwardCtrl_EX_ID_EX<=DataMemForwardCtrl_EX_IF_ID;
```

```
        DataMemForwardCtrl_MEM_ID_EX<=DataMemForwardCtrl_MEM_IF_ID;
        IM_20_0_ID_EX<=IM_IF_ID[20:0];
        Sign_Extended_ID_EX<=sign_extender;
        Registers_A_ID_EX<=Register_file_A_IF_ID;
        Registers_B_ID_EX<=Register_file_B_IF_ID;
    end
end

////////////////////////////////////////////////////////////////////////////////////////////////////////////
//Stage 3

assign RT_ID_EX = IM_20_0_ID_EX[20:16];  //Destination register from execute stage
assign RD_ID_EX = IM_20_0_ID_EX[15:11];
assign Shamt_ID_EX = IM_20_0_ID_EX[10:6];
assign Funccode_ID_EX = IM_20_0_ID_EX [5:0];

// input declaration to 2 muxes before ALU
// input 1
always @(*)begin
case (AluOpCtrlA_ID_EX)
2'b00: ALU_IN1 = {27'b0, Shamt_ID_EX};
2'b01: ALU_IN1 = Register_W_MEM_WB;
2'b10: ALU_IN1 = ALU_OUT_EX_MEM;
2'b11: ALU_IN1 = Registers_A_ID_EX;
endcase
end
//input 2
always @(*)begin
case (AluOpCtrlB_ID_EX)
2'b00: ALU_IN2 = Sign_Extended_ID_EX;
```

```
2'b01: ALU_IN2 = Register_W_MEM_WB;

2'b10: ALU_IN2 = ALU_OUT_EX_MEM;

2'b11: ALU_IN2 = Registers_B_ID_EX;

endcase

end



//ALU

ALU MainAlu (.BusW(ALU_OUT), .Zero(ALU_Zero), .BusA(ALU_IN1),
.BusB(ALU_IN2), .ALUCtrl(ALU_control));


//ALU Control Unit.

ALUControl ALUCntrl (.ALUCtrl(ALU_control), .ALUop(ALUOp_ID_EX),
.FuncCode(Funccode_ID_EX));




assign RW_ID_EX = RegDst_ID_EX ? RD_ID_EX : RT_ID_EX;

assign Data_Memory_Input_ID_EX = DataMemForwardCtrl_EX_ID_EX ?
Register_W_MEM_WB : Registers_B_ID_EX;


////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
/////////////

//STAGE 4 MEMORY TO WB


always @ (negedge CLK or negedge Reset_L) begin

if (~Reset_L)begin


Data_Memory_Input_EX_MEM <= 32'b0;

ALU_OUT_EX_MEM      <=32'b0;

RW_EX_MEM           <=5'b0;

MemToReg_EX_MEM     <=1'b0;
```

```
RegWrite_EX_MEM <=1'b0;
MemRead_EX_MEM<=1'b0;
MemWrite_EX_MEM <=1'b0;
DataMemForwardCtrl_MEM_EX_MEM<=1'b0;
end

else
begin
Data_Memory_Input_EX_MEM <= Data_Memory_Input_ID_EX;
ALU_OUT_EX_MEM <= ALU_OUT;
RW_EX_MEM <= RW_ID_EX;
MemToReg_EX_MEM <= MemToReg_ID_EX;
RegWrite_EX_MEM <= RegWrite_ID_EX;
MemRead_EX_MEM<= MemRead_ID_EX;
MemWrite_EX_MEM <= MemWrite_ID_EX;
DataMemForwardCtrl_MEM_EX_MEM <=DataMemForwardCtrl_MEM_ID_EX;
end
end

// Stage 4 - MEMORY
//MUX BEFORE DATA MEMORY
assign Data_Memory_actual_in = DataMemForwardCtrl_MEM_EX_MEM ?
Register_W_MEM_WB :Data_Memory_Input_EX_MEM;

// DATA MEMORY MODULE Instantiation:
DataMemory dmem (.ReadData(Data_memory_out),
.Address(ALU_OUT_EX_MEM), .WriteData(Data_Memory_actual_in),
.MemoryRead(MemRead_EX_MEM), .MemoryWrite(MemWrite_EX_MEM),
.Clock(CLK));


//MEMORY TO WB STAGE
```

```
always @(negedge CLK or negedge Reset_L)begin
if (~Reset_L)begin
MemToReg_MEM_WB<=1'b0;
RegWrite_MEM_WB<=1'b0;
RW_MEM_WB<=5'b0;
DataOut_MEM_WB<=32'b0;
ALU_OUT_MEM_WB<=32'b0;
end

else begin
MemToReg_MEM_WB<=MemToReg_EX_MEM;
RegWrite_MEM_WB<=RegWrite_EX_MEM;
RW_MEM_WB<=RW_EX_MEM;
DataOut_MEM_WB<=Data_memory_out;
ALU_OUT_MEM_WB<=ALU_OUT_EX_MEM;
end

end



// stage 5 Write back

assign Register_W_MEM_WB = MemToReg_MEM_WB ? DataOut_MEM_WB :
ALU_OUT_MEM_WB;



// actual output
assign dMemOut = DataOut_MEM_WB;
```

```
endmodule
```

TESTBENCH:

```
`timescale 1ns / 1ps

`define STRLEN 32
`define HalfClockPeriod 60
`define ClockPeriod `HalfClockPeriod * 2
module PipelinedProcTest_v;

	task passTest;
		input [31:0] actualOut, expectedOut;
		input [`STRLEN*8:0] testType;
		inout [7:0] passed;

		if(actualOut == expectedOut) begin $display ("%s passed", testType); passed = passed + 1; end
		else $display ("%s failed: 0x%x should be 0x%x", testType, actualOut, expectedOut);
	endtask

	task allPassed;
		input [7:0] passed;
		input [7:0] numTests;

		if(passed == numTests) $display ("All tests passed");
		else $display("Some tests failed: %d of %d passed", passed, numTests);
	endtask

	// Inputs
	reg CLK;
```

```
	reg Reset_L;
	reg [31:0] startPC;
	reg [7:0] passed;


	// Outputs
	wire [31:0] dMemOut;

	//book keeping
	reg [31:0] cc_counter;

	always@(negedge CLK)
		if(~Reset_L)
			cc_counter <= 0;
		else
			cc_counter <= cc_counter+1;

	initial begin
		CLK= 1'b0;
	end

	 /*generate clock signal*/
always begin
  #`HalfClockPeriod CLK = ~CLK;
  #`HalfClockPeriod CLK = ~CLK;
end

	// Instantiate the Unit Under Test (UUT)
	PipelinedProc uut (
		.CLK(CLK),
```

```
        .Reset_L(Reset_L),
        .startPC(startPC),
        .dMemOut(dMemOut)
);

initial begin
        // Initialize Inputs
        Reset_L = 1;
        startPC = 0;
        passed = 0;

        // Wait for global reset
        #(1 * `ClockPeriod);

        // Program 1
        #1;
        Reset_L = 0; startPC = 32'h0;
        #(1 * `ClockPeriod);
        Reset_L = 1;
        #(46 * `ClockPeriod);
        passTest(dMemOut, 120, "Results of Program 1", passed);

        // Program 2
        #(1 * `ClockPeriod);
        Reset_L = 0; startPC = 32'h60;
        #(1 * `ClockPeriod);
        Reset_L = 1;
        #(34 * `ClockPeriod);
        passTest(dMemOut, 2, "Results of Program 2", passed);
```

```
// Program 3
#(1 * `ClockPeriod);
Reset_L = 0; startPC = 32'hA0;
#(1 * `ClockPeriod);
Reset_L = 1;
#(29 * `ClockPeriod);
passTest(dMemOut, 32'hfeedbeef, "Result 1 of Program 3", passed);
#(1 * `ClockPeriod);
passTest(dMemOut, 32'hfeedb48f, "Result 2 of Program 3", passed);
#(1 * `ClockPeriod);
passTest(dMemOut, 32'hfeeeb48f, "Result 3 of Program 3", passed);
#(1 * `ClockPeriod);
passTest(dMemOut, 32'h0000b4a0, "Result 4 of Program 3", passed);
#(1 * `ClockPeriod);
passTest(dMemOut, 32'hddb7dde0, "Result 5 of Program 3", passed);
#(1 * `ClockPeriod);
passTest(dMemOut, 32'h07f76df7, "Result 6 of Program 3", passed);
#(1 * `ClockPeriod);
passTest(dMemOut, 32'hfff76df7, "Result 7 of Program 3", passed);
#(1 * `ClockPeriod);
passTest(dMemOut, 1, "Result 8 of Program 3", passed);
#(1 * `ClockPeriod);
passTest(dMemOut, 0, "Result 9 of Program 3", passed);
#(1 * `ClockPeriod);
passTest(dMemOut, 0, "Result 10 of Program 3", passed);
#(1 * `ClockPeriod);
passTest(dMemOut, 1, "Result 11 of Program 3", passed);
#(1 * `ClockPeriod);
passTest(dMemOut, 32'hfeed4b4f, "Result 12 of Program 3", passed);
```

```
        // Done
        allPassed(passed, 14);
        $finish;
    end


endmodule
```