CS 385 – FINAL REPORT

RACHAEL GRABOWSKI & MICHAEL LANG

# INSTRUCTION SET

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Syntax** | **Meaning** | **Format** | **Opcode** | **Encoding** |
| **Add** | add $d, $s, $t | $d 🡨 $s + $t | R | 0000 | 0000 sstt dd-- ---- |
| **Subtract** | sub $d, $s, $t | $d 🡨 $s - $t | R | 0001 | 0001 sstt dd-- ---- |
| **And** | and $d, $s, $t | $d 🡨 $s & $t | R | 0010 | 0010 sstt dd-- ---- |
| **Or** | or $d, $s, $t | $d 🡨 $s | $t | R | 0011 | 0011 sstt dd-- ---- |
| **Add Immediate** | addi $t, $s, C | $t 🡨 $s + C | I | 0100 | 0100 sstt CCCC CCCC |
| **Load word** | lw $t, C($s) | $t 🡨 DataMem[$s+C] | I | 0101 | 0101 sstt CCCC CCCC |
| **Store word** | sw $t, C($s) | DataMem[$s+C] 🡨 $t | I | 0110 | 0110 sstt CCCC CCCC |
| **Set on less than** | slt $d, $s, $t | $d = ($s < $t) | R | 0111 | 0111 sstt dd-- ---- |
| **Branch on equal** | beq $s, $t, C | if ($s == $t) goto C | I | 1000 | 1000 sstt CCCC CCCC |
| **Branch on not equal** | bne $s, $t, C | if ($s != $t) goto C | I | 1001 | 1001 sstt CCCC CCCC |
| **Shift left logical** | sll $d, $t, shamt | $d 🡨 $t << shamt | R | 1010 | 1010 --tt ddaa aa-- |
| **Shift right logical** | srl $d, $t, shamt | $d 🡨 $t >> shamt | R | 1011 | 1011 --tt ddaa aa-- |
| **Jump** | j C | PC 🡨 C | I | 1100 | 1100 ---- CCCC CCCC |

Our implementation of a 16-bit mini pipelined MIPS machine (using gate modeling and behavioral modeling in Verilog) includes the following instructions: add, sub, and, or, addi, lw, sw, slt, beq, bne, sll, srl, j (see the table above for corresponding OP codes).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **R-Format** | | | | | |
| **OP** | RS | RT | RD | shamt | Unused |
| **4** | 2 | 2 | 2 | 4 | 2 |

|  |  |  |  |
| --- | --- | --- | --- |
| I-Format | | | |
| OP | RS | RT | Address/Value |
| 4 | 2 | 2 | 8 |

# Major Component Descriptions

## Instruction Memory

The instruction memory is represented by our input.hex file (see “input.hex” below)

## ALU

The ALU is a 16-bit ALU comprised of four 4-bit ALUs. It is capable of detecting overflow and zero.

## Register File

Register r0 is static 0 (it cannot change or be changed). The register file is 4-word by 16-bits-per-word.

## Control Unit

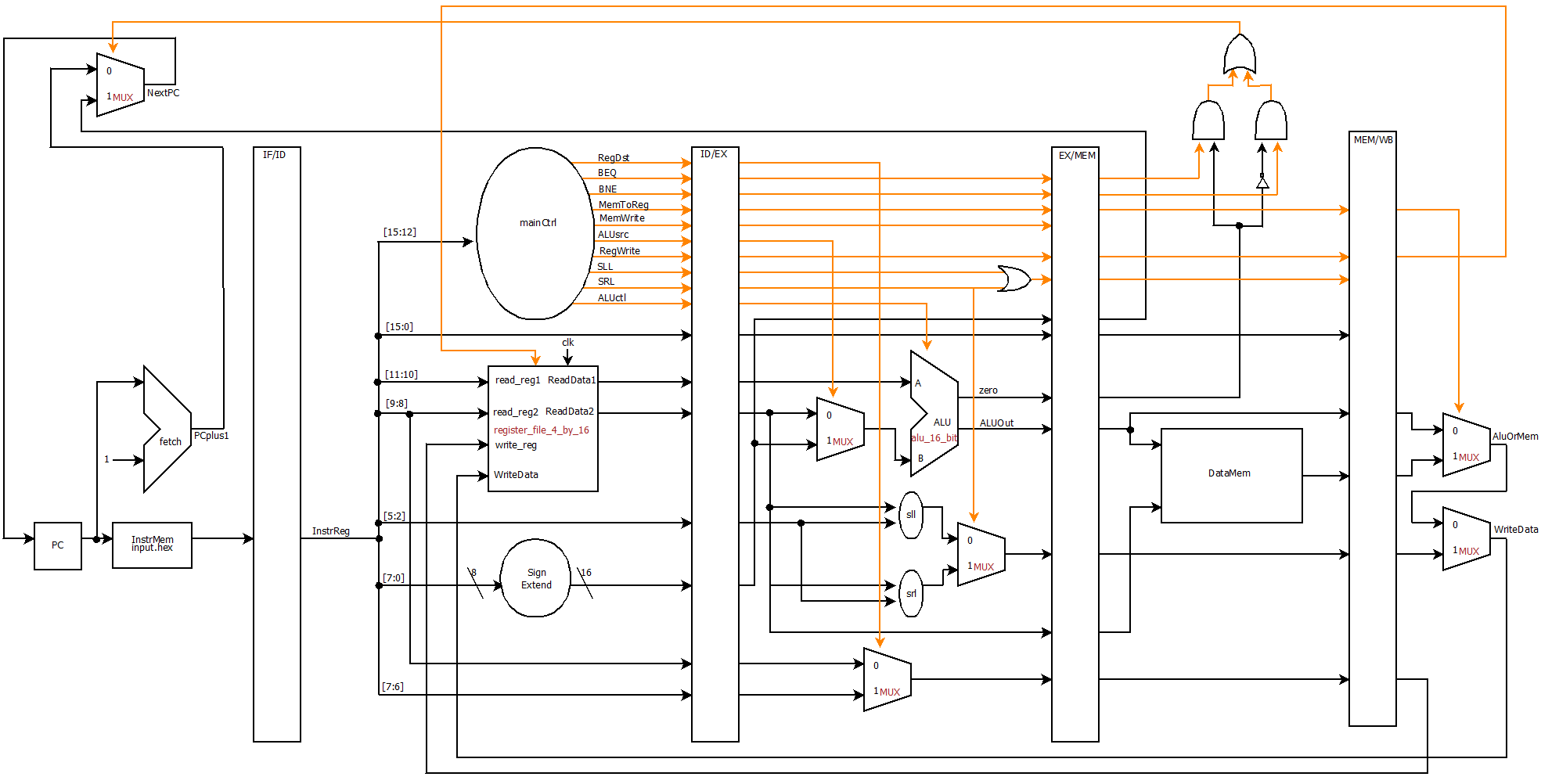
The mainCtrl unit takes IFID\_InstrMem[15:12] as input, and outputs a control signal to the IDEX register. The control options are: RegDst, BEQ, BNE, MemToReg, MemWrite, ALUsrc, RegWrite, SLL, SRL, and ALUctl.

## input.hex

The input.hex file is how we loaded our code into our program, instead of hard-coding our testing instructions. This made it very easy to swap between different sets of instructions for testing at various points in the project. MIPS ASM instructions were converted into binary, and those binary instructions were then converted into hex instructions. Those hex instructions were then read by the CPU module into instruction memory.

# DIAGRAM

Diagram is best viewed at 250% or larger



# VERILOG SOURCE CODE

## module alu\_1\_bit(result, c\_out, a, b, less, c\_in, op);

input a, b;

input less;

input c\_in;

input [2:0] op;

output result, c\_out;

wire m, n, p, q, r;

not g1(m, b);

mux\_2\_to\_1 mux1(n, b, m, op[2]);

and g2(p, a, n);

or g3(q, a, n);

full\_adder fa(r, c\_out, a, n, c\_in);

mux\_4\_to\_1 mux2(result, p, q, r, less, op[1:0]);

endmodule

## module alu\_1\_bit\_msb(result, overflow, set, c\_out, a, b, less, c\_in, op);

input a, b;

input less;

input c\_in;

input [2:0] op;

output result, overflow, set, c\_out;

wire m, n, p, q;

not g1(m, b);

mux\_2\_to\_1 mux1(n, b, m, op[2]);

and g2(p, a, n);

or g3(q, a, n);

full\_adder fa(set, c\_out, a, n, c\_in);

mux\_4\_to\_1 mux2(result, p, q, set, less, op[1:0]);

xor g4(overflow, c\_out, c\_in);

endmodule

## module alu\_4\_bit(result, zero, c\_out, a, b, less, c\_in, op);

input [3:0] a;

input [3:0] b;

input less;

input c\_in;

input [2:0] op;

output [3:0] result;

output zero, c\_out;

wire p, q, r, s, t;

alu\_1\_bit alu\_0(result[0], p, a[0], b[0], less, c\_in, op),

alu\_1(result[1], q, a[1], b[1], 0, p, op),

alu\_2(result[2], r, a[2], b[2], 0, q, op),

alu\_3(result[3], c\_out, a[3], b[3], 0, r, op);

or g1(s, result[0], result[1]),

g2(t, result[2], result[3]);

nor g3(zero, s, t);

endmodule

## module alu\_4\_bit\_last(result, overflow, set, zero, c\_out, a, b, c\_in, op);

input [3:0] a;

input [3:0] b;

input c\_in;

input [2:0] op;

output [3:0] result;

output overflow, set, zero, c\_out;

wire p, q, r, s, t;

alu\_1\_bit alu\_0(result[0], p, a[0], b[0], 0, c\_in, op),

alu\_1(result[1], q, a[1], b[1], 0, p, op),

alu\_2(result[2], r, a[2], b[2], 0, q, op);

alu\_1\_bit\_msb alu\_3(result[3], overflow, set, c\_out, a[3], b[3], 0, r, op);

or g1(s, result[0], result[1]),

g2(t, result[2], result[3]);

nor g3(zero, s, t);

endmodule

## module alu\_16\_bit(result, overflow, zero, c\_out, a, b, op);

input [15:0] a;

input [15:0] b;

input [2:0] op;

output [15:0] result;

output overflow, zero, c\_out;

wire p, q, r, s, t, u, v, w, x;

wire less;

alu\_4\_bit alu\_0(result[3:0], s, p, a[3:0], b[3:0], less, op[2], op),

alu\_1(result[7:4], t, q, a[7:4], b[7:4], 0, p, op),

alu\_2(result[11:8], u, r, a[11:8], b[11:8], 0, q, op);

alu\_4\_bit\_last alu\_3(result[15:12], overflow, less, v, c\_out, a[15:12], b[15:12], r, op);

and g1(w, s, t),

g2(x, u, v),

g3(zero, w, x);

endmodule

## module d\_flip\_flop(D,CLK,Q);

input D,CLK;

output Q;

wire CLK1, Y;

not not1 (CLK1,CLK);

d\_latch D1(D,CLK, Y),

D2(Y,CLK1,Q);

endmodule

## module d\_latch(D,C,Q);

input D,C;

output Q;

wire x,y,D1,Q1;

nand nand1 (x,D, C),

nand2 (y,D1,C),

nand3 (Q,x,Q1),

nand4 (Q1,y,Q);

not not1 (D1,D);

endmodule

## module decoder\_2\_to\_4(out, sel);

input [1:0] sel;

output [3:0] out;

wire not\_sel0, not\_sel1;

not n0(not\_sel0, sel[0]),

n1(not\_sel1, sel[1]);

and a0(out[0], not\_sel0, not\_sel1),

a1(out[1], sel[0], not\_sel1),

a2(out[2], not\_sel0, sel[1]),

a3(out[3], sel[0], sel[1]);

endmodule

## module half\_adder(sum, c\_out, a, b);

input a, b;

output sum, c\_out;

xor g1 (sum, a, b);

and g2 (c\_out, a, b);

endmodule

## module full\_adder(sum, c\_out, a, b, c\_in);

input a, b, c\_in;

output sum, c\_out;

wire p, q, r;

half\_adder ha1(p, q, a, b),

ha2(sum, r, p, c\_in);

or g1(c\_out, r, q);

endmodule

## module mux\_2\_to\_1(x, a, b, sel);

input a, b, sel;

output x;

wire p, q, r;

not g1 (p, sel);

and g2 (q, a, p),

g3 (r, b, sel);

or g4 (x, q, r);

endmodule

## module mux\_4\_to\_1(x, a, b, c, d, sel);

input a, b, c, d;

input [1:0] sel;

output x;

wire p, q;

mux\_2\_to\_1 mux1(p, a, b, sel[0]),

mux2(q, c, d, sel[0]),

mux3(x, p, q, sel[1]);

endmodule

## module mux\_4\_to\_2(x, a, b, sel);

input [1:0] a, b;

input sel;

output [1:0] x;

mux\_2\_to\_1 mux1(x[0], a[0], b[0], sel),

mux2(x[1], a[1], b[1], sel);

endmodule

## module mux\_32\_to\_16(x, a, b, sel);

input [15:0] a, b;

input sel;

output [15:0] x;

mux\_2\_to\_1 m0(x[0], a[0], b[0], sel),

m1(x[1], a[1], b[1], sel),

m2(x[2], a[2], b[2], sel),

m3(x[3], a[3], b[3], sel),

m4(x[4], a[4], b[4], sel),

m5(x[5], a[5], b[5], sel),

m6(x[6], a[6], b[6], sel),

m7(x[7], a[7], b[7], sel),

m8(x[8], a[8], b[8], sel),

m9(x[9], a[9], b[9], sel),

m10(x[10], a[10], b[10], sel),

m11(x[11], a[11], b[11], sel),

m12(x[12], a[12], b[12], sel),

m13(x[13], a[13], b[13], sel),

m14(x[14], a[14], b[14], sel),

m15(x[15], a[15], b[15], sel);

endmodule

## module mux\_64\_to\_16(x, a, b, c, d, sel);

input [15:0] a, b, c, d;

input [1:0] sel;

output [15:0] x;

mux\_4\_to\_1 m0(x[0], a[0], b[0], c[0], d[0], sel),

m1(x[1], a[1], b[1], c[1], d[1], sel),

m2(x[2], a[2], b[2], c[2], d[2], sel),

m3(x[3], a[3], b[3], c[3], d[3], sel),

m4(x[4], a[4], b[4], c[4], d[4], sel),

m5(x[5], a[5], b[5], c[5], d[5], sel),

m6(x[6], a[6], b[6], c[6], d[6], sel),

m7(x[7], a[7], b[7], c[7], d[7], sel),

m8(x[8], a[8], b[8], c[8], d[8], sel),

m9(x[9], a[9], b[9], c[9], d[9], sel),

m10(x[10], a[10], b[10], c[10], d[10], sel),

m11(x[11], a[11], b[11], c[11], d[11], sel),

m12(x[12], a[12], b[12], c[12], d[12], sel),

m13(x[13], a[13], b[13], c[13], d[13], sel),

m14(x[14], a[14], b[14], c[14], d[14], sel),

m15(x[15], a[15], b[15], c[15], d[15], sel);

endmodule

module register\_16\_bit(out, in, clk);

input [15:0] in;

input clk;

output [15:0] out;

d\_flip\_flop d0(in[0], clk, out[0]),

d1(in[1], clk, out[1]),

d2(in[2], clk, out[2]),

d3(in[3], clk, out[3]),

d4(in[4], clk, out[4]),

d5(in[5], clk, out[5]),

d6(in[6], clk, out[6]),

d7(in[7], clk, out[7]),

d8(in[8], clk, out[8]),

d9(in[9], clk, out[9]),

d10(in[10], clk, out[10]),

d11(in[11], clk, out[11]),

d12(in[12], clk, out[12]),

d13(in[13], clk, out[13]),

d14(in[14], clk, out[14]),

d15(in[15], clk, out[15]);

endmodule

module register\_file\_4\_by\_16(read\_data1, read\_data2, read\_reg1, read\_reg2, write\_reg, write\_data, reg\_write, clk);

input [1:0] read\_reg1, read\_reg2, write\_reg;

input [15:0] write\_data;

input reg\_write, clk;

output [15:0] read\_data1, read\_data2;

wire [3:0] w, c;

wire p;

wire [15:0] q1, q2, q3;

decoder\_2\_to\_4 dec(w, write\_reg);

and g1(p, reg\_write, clk),

g2(c[1], p, w[1]),

g3(c[2], p, w[2]),

g4(c[3], p, w[3]);

register\_16\_bit r1(q1, write\_data, c[1]),

r2(q2, write\_data, c[2]),

r3(q3, write\_data, c[3]);

mux\_64\_to\_16 m1(read\_data1, 0, q1, q2, q3, read\_reg1),

m2(read\_data2, 0, q1, q2, q3, read\_reg2);

endmodule

## module shift\_left\_logical(out, in, shift\_amount);

input [15:0] in;

input [3:0] shift\_amount;

output [15:0] out;

wire [15:0] p, q, r;

mux\_32\_to\_16 mux1(p, in[15:0], {in[7:0],8'b0}, shift\_amount[3]),

mux2(q, p, {p[11:0],4'b0}, shift\_amount[2]),

mux3(r, q, {q[13:0],2'b0}, shift\_amount[1]),

mux4(out, r, {r[14:0],1'b0}, shift\_amount[0]);

endmodule

## module shift\_right\_logical(out, in, shift\_amount);

input [15:0] in;

input [3:0] shift\_amount;

output [15:0] out;

wire [15:0] p, q, r;

mux\_32\_to\_16 mux1(p, in[15:0], {8'b0,in[15:8]}, shift\_amount[3]),

mux2(q, p, {4'b0,p[15:4]}, shift\_amount[2]),

mux3(r, q, {2'b0,q[15:2]}, shift\_amount[1]),

mux4(out, r, {1'b0,r[15:1]}, shift\_amount[0]);

endmodule

## module CPU (clock, PC, IFID\_InstrReg, IDEX\_InstrReg, EXMEM\_InstrReg, MEMWB\_InstrReg, WriteData);

input clock;

output [15:0] PC, WriteData, IFID\_InstrReg, IDEX\_InstrReg, EXMEM\_InstrReg, MEMWB\_InstrReg;

// IF

wire [15:0] NextPC, PCplus1;

reg [15:0] PC, IFID\_InstrReg;

reg [15:0] InstrMem[0:511];

alu\_16\_bit fetch(PCplus1, unused1, unused2, unused3, PC, 1, 3'b010);

mux\_32\_to\_16 mux1(NextPC, PCplus1, EXMEM\_SignExtend, PCsrc);

// Read hex data from text file into InstrMem

initial begin

$readmemh("input4.hex", InstrMem);

DataMem[0] = 16'h400;

DataMem[1] = 16'hfad;

end

// ID

reg [15:0] IDEX\_InstrReg;

wire [11:0] control;

reg IDEX\_RegDst, IDEX\_BEQ, IDEX\_BNE, IDEX\_MemToReg, IDEX\_MemWrite, IDEX\_ALUsrc, IDEX\_RegWrite, IDEX\_SLL, IDEX\_SRL;

reg [2:0] IDEX\_ALUctl;

wire [15:0] ReadData1, ReadData2, SignExtend, WriteData;

reg [15:0] IDEX\_ReadData1, IDEX\_ReadData2, IDEX\_SignExtend;

reg [1:0] IDEX\_rt, IDEX\_rd;

reg [3:0] IDEX\_shamt;

register\_file\_4\_by\_16 regFile(ReadData1, ReadData2, IFID\_InstrReg[11:10], IFID\_InstrReg[9:8], MEMWB\_WriteReg, WriteData, MEMWB\_RegWrite, clock);

main\_control mainCtrl(IFID\_InstrReg[15:12], control);

assign SignExtend = {{8{IFID\_InstrReg[7]}},IFID\_InstrReg[7:0]}; // sign extension

// EX

wire [15:0] B, ALUout, ShiftLOut, ShiftROut, Shifted;

wire [1:0] WriteReg;

wire Zero, ShiftToReg;

reg EXMEM\_RegWrite, EXMEM\_MemToReg;

reg EXMEM\_MemWrite, EXMEM\_BEQ, EXMEM\_BNE, EXMEM\_ShiftToReg;

reg EXMEM\_Zero;

reg [15:0] EXMEM\_SignExtend, EXMEM\_ALUout, EXMEM\_ReadData2, EXMEM\_Shifted;

reg [15:0] EXMEM\_InstrReg;

reg [1:0] EXMEM\_WriteReg;

alu\_16\_bit ex(ALUout, unused4, Zero, unused5, IDEX\_ReadData1, B, IDEX\_ALUctl);

mux\_4\_to\_2 mux2(WriteReg, IDEX\_rt, IDEX\_rd, IDEX\_RegDst);

mux\_32\_to\_16 mux3(B, IDEX\_ReadData2, IDEX\_SignExtend, IDEX\_ALUsrc);

shift\_left\_logical sll(ShiftLOut, IDEX\_ReadData2, IDEX\_shamt);

shift\_right\_logical srl(ShiftROut, IDEX\_ReadData2, IDEX\_shamt);

mux\_32\_to\_16 mux4(Shifted, ShiftLOut, ShiftROut, IDEX\_SRL);

or g1(ShiftToReg, IDEX\_SLL, IDEX\_SRL);

// MEM

wire [15:0] MemOut, AluOrMem;

wire NotZero, ne, eq, PCsrc;

reg MEMWB\_RegWrite, MEMWB\_MemToReg, MEMWB\_ShiftToReg;

reg [15:0] DataMem[0:511];

reg [15:0] MEMWB\_MemOut, MEMWB\_ALUout, MEMWB\_Shifted;

reg [15:0] MEMWB\_InstrReg;

reg [1:0] MEMWB\_WriteReg;

not g2(NotZero, EXMEM\_Zero);

and g3(ne, EXMEM\_BNE, NotZero),

g4(eq, EXMEM\_BEQ, EXMEM\_Zero);

or g5(PCsrc, ne, eq);

assign MemOut = DataMem[EXMEM\_ALUout];

always @(negedge clock) begin

if(EXMEM\_MemWrite)

DataMem[EXMEM\_ALUout] <= EXMEM\_ReadData2;

end

// WB

mux\_32\_to\_16 mux5(AluOrMem, MEMWB\_ALUout, MEMWB\_MemOut, MEMWB\_MemToReg);

mux\_32\_to\_16 mux6(WriteData, AluOrMem, MEMWB\_Shifted, MEMWB\_ShiftToReg);

initial begin

// Initialize registers

PC = 0;

IFID\_InstrReg = 0;

IDEX\_InstrReg = 0;

IDEX\_RegDst = 0;

IDEX\_BEQ = 0;

IDEX\_BNE = 0;

IDEX\_MemToReg = 0;

IDEX\_MemWrite = 0;

IDEX\_ALUsrc = 0;

IDEX\_RegWrite = 0;

IDEX\_SLL = 0;

IDEX\_SRL = 0;

IDEX\_shamt = 0;

IDEX\_ALUctl = 3'b000;

IDEX\_ReadData1 = 0;

IDEX\_ReadData2 = 0;

IDEX\_SignExtend = 0;

IDEX\_rd = 0;

IDEX\_rt = 0;

EXMEM\_InstrReg = 0;

EXMEM\_BEQ = 0;

EXMEM\_BNE = 0;

EXMEM\_MemToReg = 0;

EXMEM\_MemWrite = 0;

EXMEM\_RegWrite = 0;

EXMEM\_ShiftToReg = 0;

EXMEM\_ALUout = 0;

EXMEM\_ReadData2 = 0;

EXMEM\_SignExtend = 0;

EXMEM\_Shifted = 0;

EXMEM\_WriteReg = 0;

EXMEM\_Zero = 0;

MEMWB\_MemToReg = 0;

MEMWB\_RegWrite = 0;

MEMWB\_ShiftToReg = 0;

MEMWB\_ALUout = 0;

MEMWB\_InstrReg = 0;

MEMWB\_MemOut = 0;

MEMWB\_Shifted = 0;

MEMWB\_WriteReg = 0;

end

always @(negedge clock) begin

// Stage 1 -- IF

IFID\_InstrReg <= InstrMem[PC];

PC <= NextPC;

// Stage 2 -- ID

IDEX\_InstrReg <= IFID\_InstrReg;

{IDEX\_RegDst, IDEX\_BEQ, IDEX\_BNE, IDEX\_MemToReg, IDEX\_MemWrite, IDEX\_ALUsrc, IDEX\_RegWrite, IDEX\_SLL, IDEX\_SRL, IDEX\_ALUctl} <= control;

IDEX\_ReadData1 <= ReadData1;

IDEX\_ReadData2 <= ReadData2;

IDEX\_SignExtend <= SignExtend;

IDEX\_rt <= IFID\_InstrReg[9:8];

IDEX\_rd <= IFID\_InstrReg[7:6];

IDEX\_shamt <= IFID\_InstrReg[5:2];

// Stage 3 -- EX

EXMEM\_InstrReg <= IDEX\_InstrReg;

EXMEM\_RegWrite <= IDEX\_RegWrite;

EXMEM\_MemToReg <= IDEX\_MemToReg;

EXMEM\_ShiftToReg <= ShiftToReg;

EXMEM\_BEQ <= IDEX\_BEQ;

EXMEM\_BNE <= IDEX\_BNE;

EXMEM\_MemWrite <= IDEX\_MemWrite;

EXMEM\_Zero <= Zero;

EXMEM\_ALUout <= ALUout;

EXMEM\_ReadData2 <= IDEX\_ReadData2;

EXMEM\_WriteReg <= WriteReg;

EXMEM\_SignExtend <= IDEX\_SignExtend;

EXMEM\_Shifted <= Shifted;

// Stage 4 -- MEM

MEMWB\_InstrReg <= EXMEM\_InstrReg;

MEMWB\_RegWrite <= EXMEM\_RegWrite;

MEMWB\_MemToReg <= EXMEM\_MemToReg;

MEMWB\_ShiftToReg <= EXMEM\_ShiftToReg;

MEMWB\_MemOut <= MemOut;

MEMWB\_ALUout <= EXMEM\_ALUout;

MEMWB\_WriteReg <= EXMEM\_WriteReg;

MEMWB\_Shifted <= EXMEM\_Shifted;

end

endmodule

## module main\_control(Op,Control);

input [3:0] Op;

output reg [11:0] Control;

always @(Op) case (Op)

// {RegDst, BEQ, BNE, MemtoReg, MemWrite, ALUSrc, RegWrite, SLL, SRL, ALUctl[2], ALUctl[1], ALUctl[0]}

4'b0000: Control <= 12'b100000100010; // add

4'b0001: Control <= 12'b100000100110; // sub

4'b0010: Control <= 12'b100000100000; // and

4'b0011: Control <= 12'b100000100001; // or

4'b0100: Control <= 12'b000001100010; // addi

4'b0101: Control <= 12'b000111100010; // lw

4'b0110: Control <= 12'b000011000010; // sw

4'b0111: Control <= 12'b100000100111; // slt

4'b1000: Control <= 12'b010000000110; // beq

4'b1001: Control <= 12'b001000000110; // bne

4'b1010: Control <= 12'b100000110000; // sll

4'b1011: Control <= 12'b100000101000; // srl

4'b1100: Control <= 12'b011000000110; // j

endcase

endmodule

## module test ();

reg clock;

wire [15:0] PC, IFID\_InstrReg, IDEX\_InstrReg, EXMEM\_InstrReg, MEMWB\_InstrReg, WriteData;

CPU test\_cpu(clock, PC, IFID\_InstrReg, IDEX\_InstrReg, EXMEM\_InstrReg, MEMWB\_InstrReg, WriteData);

always #1 clock = ~clock;

initial begin

$display ("time PC IFID\_IR IDEX\_IR EXMEM\_IR MEMWB\_IR WD");

$monitor ("%2d %3d %h %h %h %h %h", $time, PC, IFID\_InstrReg, IDEX\_InstrReg, EXMEM\_InstrReg, MEMWB\_InstrReg, WriteData);

clock = 1;

#163 $finish;

end

endmodule

# Testing & Results

## Testing Code

### input4.hex

5100 // lw $1, 0($0) ==> 0101 0001 0000 0000 // $1 <- x

5201 // lw $2, 1($0) ==> 0101 0010 0000 0001 // $2 <- y

4300 // addi $3, $0, 0 ==> 0100 0011 0000 0000 // $3 <- 0

0000 // nop ==> 0000 0000 0000 0000 // nop

0000 // nop ==> 0000 0000 0000 0000 // nop

8827 // beq $2, $0, 39 ==> 1000 1000 0010 0111 // if y == 0 goto 39

0000 // nop ==> 0000 0000 0000 0000 // nop

0000 // nop ==> 0000 0000 0000 0000 // nop

0000 // nop ==> 0000 0000 0000 0000 // nop

6302 // sw $3, 2($0) ==> 0110 0011 0000 0010 // a <- $3

0000 // nop ==> 0000 0000 0000 0000 // nop

0000 // nop ==> 0000 0000 0000 0000 // nop

0000 // nop ==> 0000 0000 0000 0000 // nop

4301 // addi $3, $0, 1 ==> 0100 0011 0000 0001 // $3 <- 1

0000 // nop ==> 0000 0000 0000 0000 // nop

0000 // nop ==> 0000 0000 0000 0000 // nop

0000 // nop ==> 0000 0000 0000 0000 // nop

2bc0 // and $3, $2, $3 ==> 0010 1011 1100 0000 // $3 <- y && 1

0000 // nop ==> 0000 0000 0000 0000 // nop

0000 // nop ==> 0000 0000 0000 0000 // nop

0000 // nop ==> 0000 0000 0000 0000 // nop

8c1e // beq $3, $0, 30 ==> 1000 1100 0001 1110 // if $3 == 0 goto 30

0000 // nop ==> 0000 0000 0000 0000 // nop

0000 // nop ==> 0000 0000 0000 0000 // nop

0000 // nop ==> 0000 0000 0000 0000 // nop

5302 // lw $3, 2($0) ==> 0101 0011 0000 0010 // $3 <- a

0000 // nop ==> 0000 0000 0000 0000 // nop

0000 // nop ==> 0000 0000 0000 0000 // nop

0000 // nop ==> 0000 0000 0000 0000 // nop

0dc0 // add $3, $3, $1 ==> 0000 1101 1100 0000 // $3 <- a + x

a144 // sll $1, $1, 1 ==> 1010 0001 0100 0100 // x \*= 2

b286 // srl $2, $2, 1 ==> 1011 0010 1000 0100 // y //= 2

0000 // nop ==> 0000 0000 0000 0000 // nop

0000 // nop ==> 0000 0000 0000 0000 // nop

0000 // nop ==> 0000 0000 0000 0000 // nop

c005 // j 5 ==> 1100 0000 0000 0101 // goto 5

0000 // nop ==> 0000 0000 0000 0000 // nop

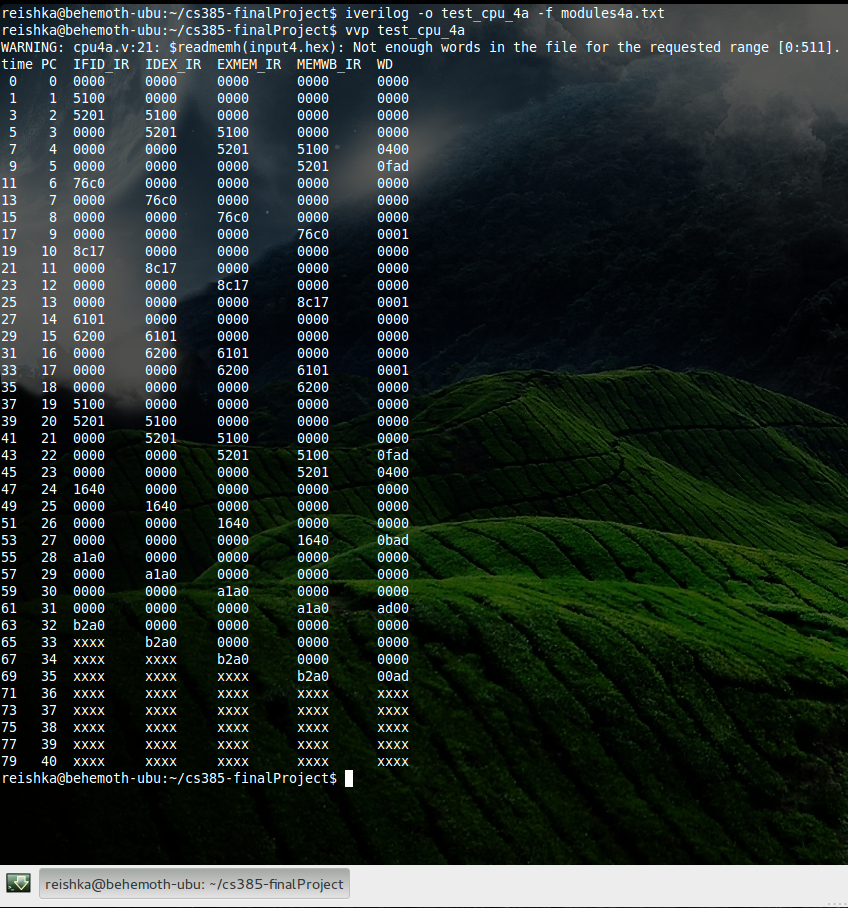
0000 // nop ==> 0000 0000 0000 0000 // nop

0000 // nop ==> 0000 0000 0000 0000 // nop

4f00 // addi $3, $3, 0 ==> 0100 1111 0000 0000 // a += 0

## Testing Results

### Screenshot



### Plaintext

reishka@behemoth-ubu:~/cs385-finalProject$ iverilog -o test\_cpu\_4a -f modules4a.txt

reishka@behemoth-ubu:~/cs385-finalProject$ vvp test\_cpu\_4a

WARNING: cpu4a.v:21: $readmemh(input4.hex): Not enough words in the file for the requested range [0:511].

time PC IFID\_IR IDEX\_IR EXMEM\_IR MEMWB\_IR WD

0 0 0000 0000 0000 0000 0000

1 1 5100 0000 0000 0000 0000

3 2 5201 5100 0000 0000 0000

5 3 0000 5201 5100 0000 0000

7 4 0000 0000 5201 5100 0400

9 5 0000 0000 0000 5201 0fad

11 6 76c0 0000 0000 0000 0000

13 7 0000 76c0 0000 0000 0000

15 8 0000 0000 76c0 0000 0000

17 9 0000 0000 0000 76c0 0001

19 10 8c17 0000 0000 0000 0000

21 11 0000 8c17 0000 0000 0000

23 12 0000 0000 8c17 0000 0000

25 13 0000 0000 0000 8c17 0001

27 14 6101 0000 0000 0000 0000

29 15 6200 6101 0000 0000 0000

31 16 0000 6200 6101 0000 0000

33 17 0000 0000 6200 6101 0001

35 18 0000 0000 0000 6200 0000

37 19 5100 0000 0000 0000 0000

39 20 5201 5100 0000 0000 0000

41 21 0000 5201 5100 0000 0000

43 22 0000 0000 5201 5100 0fad

45 23 0000 0000 0000 5201 0400

47 24 1640 0000 0000 0000 0000

49 25 0000 1640 0000 0000 0000

51 26 0000 0000 1640 0000 0000

53 27 0000 0000 0000 1640 0bad

55 28 a1a0 0000 0000 0000 0000

57 29 0000 a1a0 0000 0000 0000

59 30 0000 0000 a1a0 0000 0000

61 31 0000 0000 0000 a1a0 ad00

63 32 b2a0 0000 0000 0000 0000

65 33 xxxx b2a0 0000 0000 0000

67 34 xxxx xxxx b2a0 0000 0000

69 35 xxxx xxxx xxxx b2a0 00ad

71 36 xxxx xxxx xxxx xxxx xxxx

73 37 xxxx xxxx xxxx xxxx xxxx

75 38 xxxx xxxx xxxx xxxx xxxx

77 39 xxxx xxxx xxxx xxxx xxxx

79 40 xxxx xxxx xxxx xxxx xxxx

reishka@behemoth-ubu:~/cs385-finalProject$