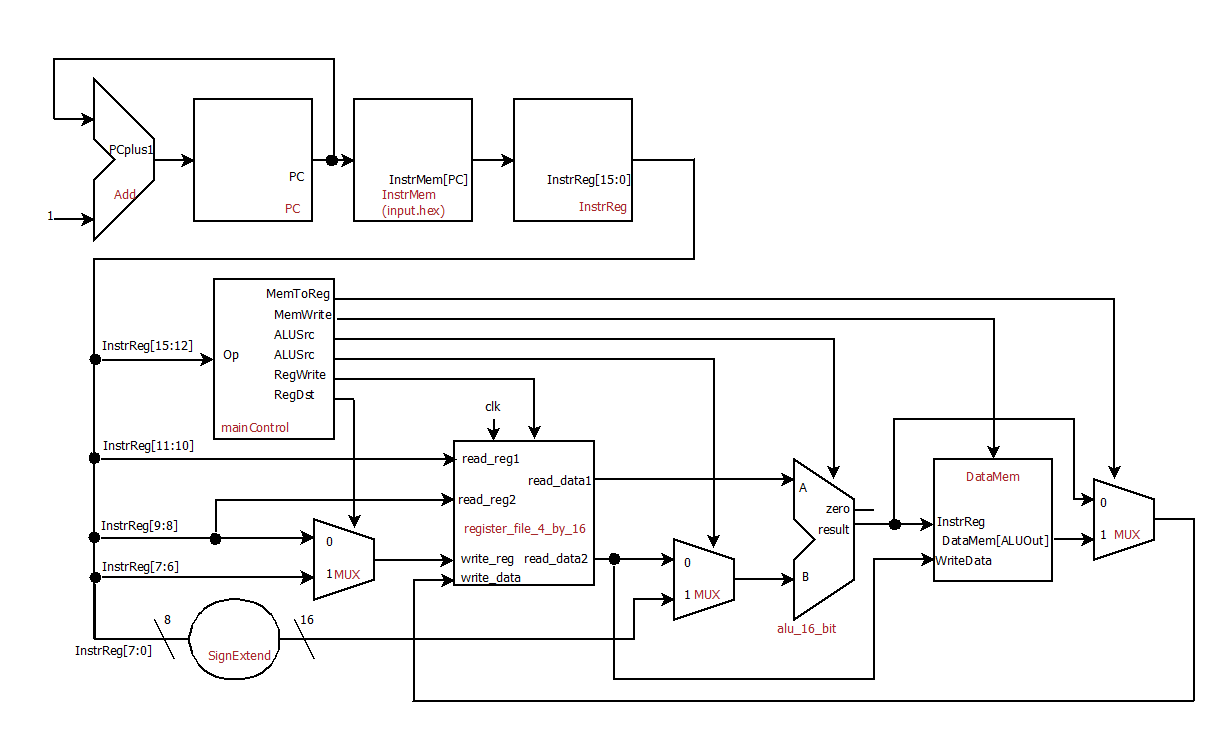
CS 385 – Progress Report I

Rachael Grabowski: Diagrams, progress report  
Michael Lang: Code

# I. Diagrams

## Single Cycle Simplified Datapath for R-type

Diagram is viewed best when document size is 150% or larger.



# II. 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 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 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 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 MainControl(Op,Control);

input [3:0] Op;

output reg [7:0] Control;

always @(Op) case (Op)

// {RegDst, MemtoReg, MemWrite, ALUSrc, RegWrite, ALUctl[2], ALUctl[1], ALUctl[0]}

4'b0000: Control <= 8'b10001010; // add

4'b0001: Control <= 8'b10001110; // sub

4'b0010: Control <= 8'b10001000; // and

4'b0011: Control <= 8'b10001001; // or

4'b0100: Control <= 8'b00011010; // addi

4'b0101: Control <= 8'b01110010; // lw

4'b0110: Control <= 8'b00110010; // sw

4'b0111: Control <= 8'b10001111; // slt

endcase

endmodule

## module CPU (clock, WriteData, InstrReg);

input clock;

output [15:0] WriteData, InstrReg;

reg [15:0] PC;

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

wire [15:0] InstrReg, SignExtend, NextPC, ReadData2, A, B, ALUOut, PCplus1, AluOrMem;

wire [1:0] WriteReg;

wire [3:0] op;

wire [2:0] ALUctl;

// Read hex data from text file into InstrMem

initial

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

initial PC = 0;

assign InstrReg = InstrMem[PC];

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

MainControl mainCtrl(InstrReg[15:12], {RegDst, MemtoReg, MemWrite, ALUSrc, RegWrite, ALUctl[2], ALUctl[1], ALUctl[0]});

register\_file\_4\_by\_16 regFile(A, ReadData2, InstrReg[11:10], InstrReg[9:8], WriteReg, WriteData, RegWrite, clock);

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

ex(ALUOut, unused4, Zero, unused5, A, B, ALUctl);

mux\_4\_to\_2 mux1(WriteReg, InstrReg[9:8], InstrReg[7:6], RegDst);

mux\_32\_to\_16 mux2(WriteData, ALUOut, DataMem[ALUOut], MemtoReg);

mux\_32\_to\_16 mux3(B, ReadData2, SignExtend, ALUSrc);

assign NextPC = PCplus1;

always @(negedge clock) begin

PC <= NextPC;

if(MemWrite)

DataMem[ALUOut] <= ReadData2;

end

endmodule

// Test module

module test ();

reg clock;

wire [15:0] WriteData, InstrReg;

CPU test\_cpu(clock, WriteData, InstrReg);

always #1 clock = ~clock;

initial begin

$display ("time clock InstrReg WriteData");

$monitor ("%2d %b %h %h", $time, clock, InstrReg, WriteData);

clock = 1;

#15 $finish;

end

endmodule

# III. Testing & Results

## Testing Code

### input.hex

410f // addi $1, $0, 15 ==> 0100000100001111

4207 // addi $2, $0, 7 ==> 0100001000000111

26c0 // and $3, $1, $2 ==> 0010011011000000

1780 // sub $2, $1, $3 ==> 0001011110000000

3b80 // or $2, $2, $3 ==> 0011101110000000

7e40 // slt $1, $3, $2 ==> 0111111001000000

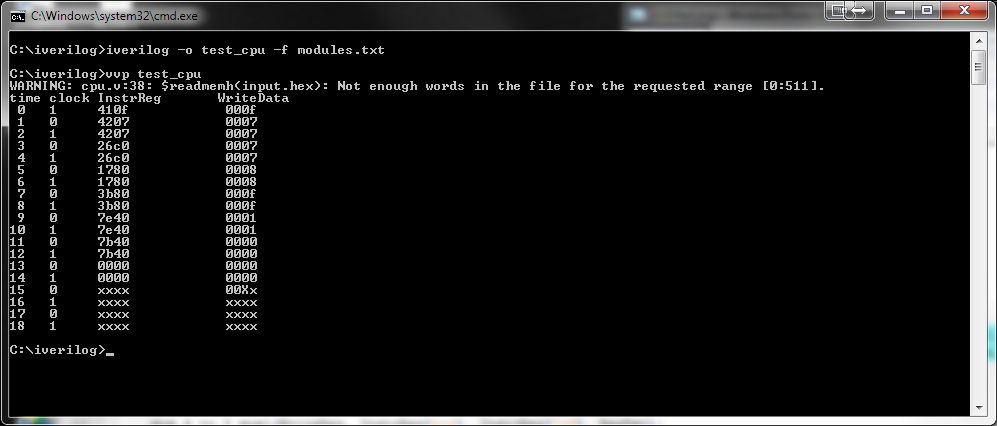
7b40 // slt $1, $2, $3 ==> 0111101101000000

0000 // nop ==> 0000000000000000

## Testing Results

Please note that the warning is intentional. Since the code is reading the instructions from a file directly into a reg vector, it's leaving most of the vector uninitialized, thus causing the warning.

### Screenshot



### Plaintext

C:\iverilog>vvp test\_cpu

WARNING: cpu.v:38: $readmemh(input.hex): Not enough words in the file for the requested range [0:511].

time clock InstrReg WriteData

0 1 410f 000f

1 0 4207 0007

2 1 4207 0007

3 0 26c0 0007

4 1 26c0 0007

5 0 1780 0008

6 1 1780 0008

7 0 3b80 000f

8 1 3b80 000f

9 0 7e40 0001

10 1 7e40 0001

11 0 7b40 0000

12 1 7b40 0000

13 0 0000 0000

14 1 0000 0000

15 0 xxxx 00Xx

16 1 xxxx xxxx

17 0 xxxx xxxx

18 1 xxxx xxxx