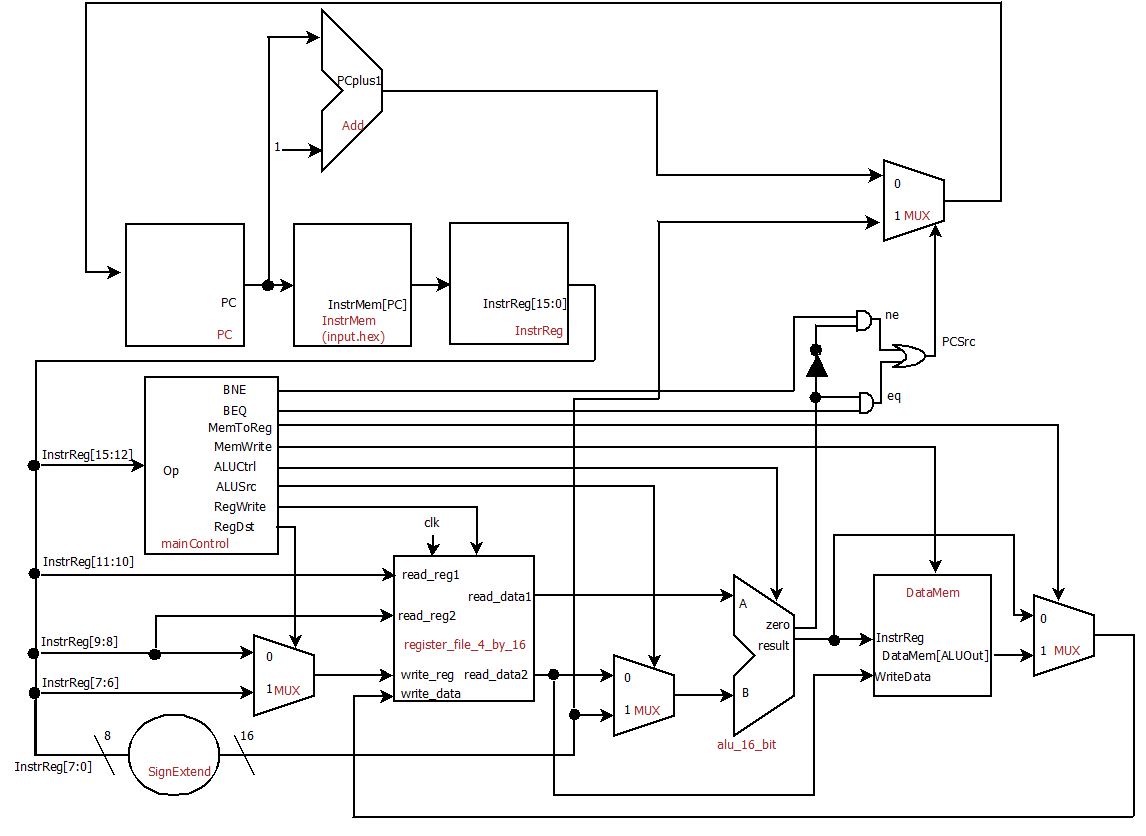
CS 385 – Progress Report II

Rachael Grabowski: Diagrams, progress report  
Michael Lang: Code

# I. Diagrams

## Single Cycle Complete 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 [9:0] Control;

always @(Op) case (Op)  
 // {RegDst, BEQ, BNE, MemtoReg, MemWrite, ALUSrc, RegWrite, ALUctl[2], ALUctl[1], ALUctl[0]}  
 4'b0000: Control <= 10'b1000001010; // add  
 4'b0001: Control <= 10'b1000001110; // sub   
 4'b0010: Control <= 10'b1000001000; // and   
 4'b0011: Control <= 10'b1000001001; // or   
 4'b0100: Control <= 10'b0000011010; // addi   
 4'b0101: Control <= 10'b0001111010; // lw   
 4'b0110: Control <= 10'b0000110010; // sw   
 4'b0111: Control <= 10'b1000001111; // slt   
 4'b1000: Control <= 10'b0100000110; // beq  
 4'b1001: Control <= 10'b0010000110; // bne   
 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 begin  
 $readmemh("input2.hex", InstrMem);  
  
 DataMem [0] = 16'h8;  
 DataMem [1] = 16'ha;  
 end  
  
 initial PC = 0;  
  
 assign InstrReg = InstrMem[PC];  
 assign SignExtend = {{8{InstrReg[7]}},InstrReg[7:0]}; // sign extension  
 MainControl mainCtrl(InstrReg[15:12], {RegDst, BEQ, BNE, 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);  
  
 not g1(NotZero, Zero);  
 and g2(ne, BNE, NotZero),  
 g3(eq, BEQ, Zero);  
 or g4(PCsrc, ne, eq);  
  
 mux\_4\_to\_2 mux1(WriteReg, InstrReg[9:8], InstrReg[7:6], RegDst);  
 mux\_32\_to\_16 mux2(WriteData, ALUOut, DataMem[ALUOut], MemtoReg),  
 mux3(B, ReadData2, SignExtend, ALUSrc),  
 mux4(NextPC, PCplus1, SignExtend, PCsrc);  
  
 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;  
 #32 $finish;  
end  
endmodule

# III. Testing & Results

## Testing Code

### input2.hex

5100 // lw $1, 0($0) ==> 0101000100000000

5201 // lw $2, 1($0) ==> 0101001000000001

76c0 // slt $3, $1, $2 ==> 0111011011000000

8c08 // beq $3, $0, 8 ==> 1000110000001000

6101 // sw $1, 1($0) ==> 0110000100000001

6200 // sw $2, 0($0) ==> 0110001000000000

5100 // lw $1, 0($0) ==> 0101000100000000

5201 // lw $2, 1($0) ==> 0101001000000001

1640 // sub $1, $1, $2 ==> 0001011001000000

43ff // addi $3, $0, -1 ==> 0100001111111111

4f01 // addi $3, $3, 1 ==> 0100111100000001

9d0a // bne $3, $1, 10 ==> 1001110100001010

0000 // nop ==> 0000000000000000

## Testing Results

The test program calculates the absolute value of the difference between two values stored in data memory, then enters a loop which increments a counter value until the counter is equal to the computed difference. The simulation is executed twice. For the first run, the first two words of the data memory are initialized with the values 0x08 and 0x0a respectively. As such, the first branch is not taken, and the two values are swapped in data memory prior to the difference being computed and entering the loop. In the second run, the initial values of the two words in data memory are manually swapped, and the simulation recompiled, such that the first branch is taken and the difference is computed without first performing a swap in memory.  
  
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.

### Execution1

#### Screenshot

#### Plaintext

michael@berlioz ~/school/Arch\_CS385/cs385-finalProject (master\*) $ iverilog -o test\_cpu\_a -f modules2a.txt

michael@berlioz ~/school/Arch\_CS385/cs385-finalProject (master\*) $ vvp test\_cpu\_a

WARNING: cpu2a.v:38: $readmemh(input2.hex): Not enough words in the file for the requested range [0:511].

time clock InstrReg WriteData

0 1 5100 0008

1 0 5201 000a

2 1 5201 000a

3 0 76c0 0001

4 1 76c0 0001

5 0 8c08 0001

6 1 8c08 0001

7 0 6101 0001

8 1 6101 0001

9 0 6200 0000

10 1 6200 0000

11 0 5100 000a

12 1 5100 000a

13 0 5201 0008

14 1 5201 0008

15 0 1640 0002

16 1 1640 0002

17 0 43ff ffff

18 1 43ff ffff

19 0 4f01 0000

20 1 4f01 0000

21 0 9d0a fffe

22 1 9d0a fffe

23 0 4f01 0001

24 1 4f01 0001

25 0 9d0a ffff

26 1 9d0a ffff

27 0 4f01 0002

28 1 4f01 0002

29 0 9d0a 0000

30 1 9d0a 0000

31 0 0000 0000

32 1 0000 0000

### Execution 2

#### Screenshot

#### Plaintext

michael@berlioz ~/school/Arch\_CS385/cs385-finalProject (master\*) $ iverilog -o test\_cpu\_b -f modules2b.txt

michael@berlioz ~/school/Arch\_CS385/cs385-finalProject (master\*) $ vvp test\_cpu\_b

WARNING: cpu2b.v:38: $readmemh(input2.hex): Not enough words in the file for the requested range [0:511].

time clock InstrReg WriteData

0 1 5100 000a

1 0 5201 0008

2 1 5201 0008

3 0 76c0 0000

4 1 76c0 0000

5 0 8c08 0000

6 1 8c08 0000

7 0 1640 0002

8 1 1640 0002

9 0 43ff ffff

10 1 43ff ffff

11 0 4f01 0000

12 1 4f01 0000

13 0 9d0a fffe

14 1 9d0a fffe

15 0 4f01 0001

16 1 4f01 0001

17 0 9d0a ffff

18 1 9d0a ffff

19 0 4f01 0002

20 1 4f01 0002

21 0 9d0a 0000

22 1 9d0a 0000

23 0 0000 0000

24 1 0000 0000

25 0 xxxx 00XX

26 1 xxxx xxxX

27 0 xxxx xxxX

28 1 xxxx xxxX

29 0 xxxx xxxX

30 1 xxxx xxxX

31 0 xxxx xxxX

32 1 xxxx xxxX