Michael Santoro

Bradley Morales

Julio Sarzosa

Progress Report 2

**Description:**

For this progress report, we were tasked to finish our single-cycle implementation of the MIPS CPU by adding 4 instructions to complete our instruction set. These new instructions are lw, sw, beq and bne. For the implementation of these instructions, some modifications had to be made, including increasing the outputs of the Main Control to include MemToReg, MemWrite, BNE, and BEQ. An updated Truth Table for the Main Control including our Instruction Set is included below.

**Instruction Set:**

| Instruction | OpCode | RegDst | ALUsrc | MemToReg | RegWrite | MemWrite | BNE | BEQ | ALUCtl |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| add | 0000 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0010 |
| sub | 0001 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0110 |
| and | 0010 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0000 |
| or | 0011 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0001 |
| nor | 0100 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1100 |
| nand | 0101 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1101 |
| slt | 0110 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0111 |
| addi | 0111 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0010 |
| lw | 1000 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0010 |
| sw | 1001 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0010 |
| beq | 1010 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0110 |
| bne | 1011 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0110 |

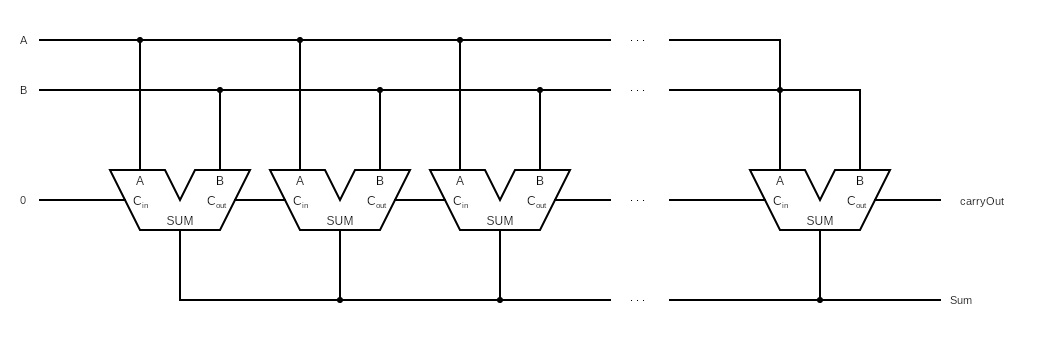


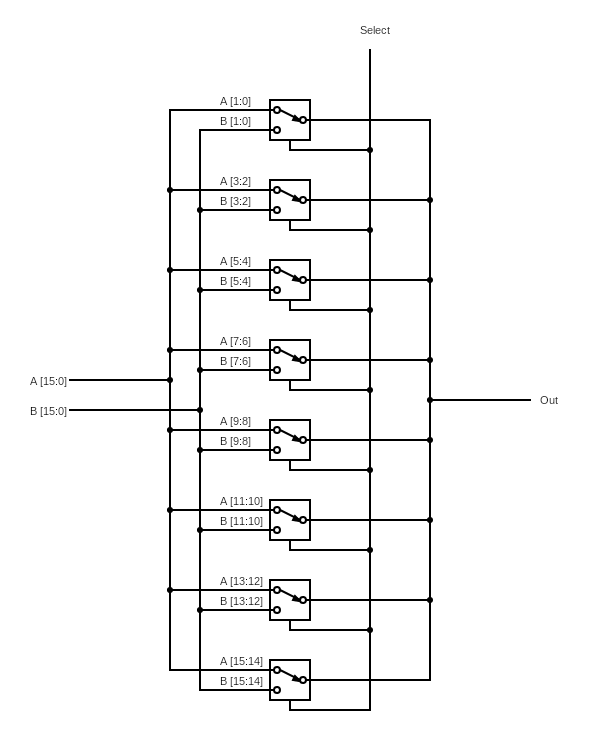
**Logic Diagrams:**

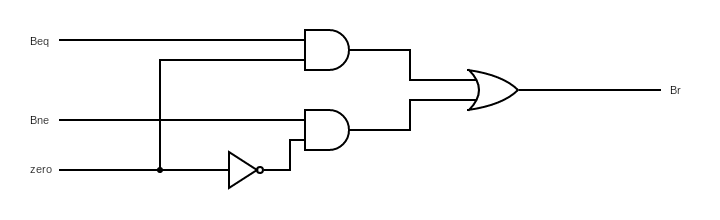
**ALU Diagram:**

**CPU Diagram:**

**16-Bit Full Adder Diagram:**

Note: The software used to create this diagram represents a full adder as the shape of an ALU. In this diagram these are supposed to be 16 Full-Adders strung together.

**16-Bit 2x1 Multiplexor:**

**Branch Control Diagram:**

**Verilog Code:**

// Michael Santoro

// Julio Sarzosa

// Bradley Morales

// Progress Report 2

module reg\_file (RR1,RR2,WR,WD,RegWrite,RD1,RD2,clock);

input [1:0] RR1,RR2,WR;

input [15:0] WD;

input RegWrite,clock;

output [15:0] RD1,RD2;

reg [15:0] Regs[0:3];

assign RD1 = Regs[RR1];

assign RD2 = Regs[RR2];

initial Regs[0] = 0;

always @(negedge clock)

if (RegWrite==1 & WR!=0)

Regs[WR] <= WD;

endmodule

// 16-bit MIPS ALU in Verilog using modified template of 4-bit ALU

module ALU (op,a,b,result,zero);

input [15:0] a;

input [15:0] b;

input [3:0] op;

output [15:0] result;

output zero;

wire c1,c2,c3,c4,c5,c6,c7,c8,c9,c10,c11,c12,c13,c14,c15,c16;

//16 single bit alu's

ALU1 alu0 (a[0], b[0], op[3], op[2], op[1:0],set,op[2],c1,result[0]);

ALU1 alu1 (a[1], b[1], op[3], op[2], op[1:0],1'b0, c1, c2,result[1]);

ALU1 alu2 (a[2], b[2], op[3], op[2], op[1:0],1'b0, c2, c3,result[2]);

ALU1 alu3 (a[3], b[3], op[3], op[2], op[1:0],1'b0, c3, c4,result[3]);

ALU1 alu4 (a[4], b[4], op[3], op[2], op[1:0],1'b0, c4, c5,result[4]);

ALU1 alu5 (a[5], b[5], op[3], op[2], op[1:0],1'b0, c5, c6,result[5]);

ALU1 alu6 (a[6], b[6], op[3], op[2], op[1:0],1'b0, c6, c7,result[6]);

ALU1 alu7 (a[7], b[7], op[3], op[2], op[1:0],1'b0, c7, c8,result[7]);

ALU1 alu8 (a[8], b[8], op[3], op[2], op[1:0],1'b0, c8, c9,result[8]);

ALU1 alu9 (a[9], b[9], op[3], op[2], op[1:0],1'b0, c9, c10,result[9]);

ALU1 alu10 (a[10],b[10],op[3], op[2], op[1:0],1'b0, c10, c11,result[10]);

ALU1 alu11 (a[11],b[11],op[3], op[2], op[1:0],1'b0, c11, c12,result[11]);

ALU1 alu12 (a[12],b[12],op[3], op[2], op[1:0],1'b0, c12, c13,result[12]);

ALU1 alu13 (a[13],b[13],op[3], op[2], op[1:0],1'b0, c13, c14,result[13]);

ALU1 alu14 (a[14],b[14],op[3], op[2], op[1:0],1'b0, c14, c15,result[14]);

ALUmsb alu15 (a[15],b[15],op[3], op[2], op[1:0],1'b0, c15, c16,result[15],set);

nor nor1(zero,result [15:0]);

endmodule

// 1-bit ALU for bits 0-14

module ALU1 (a,b,ainvert,binvert,op,less,carryin,carryout,result);

input a,b,less,carryin,ainvert, binvert;

input [1:0] op;

output carryout,result;

wire sum, a\_and\_b, a\_or\_b, b\_inv, a\_inv;

// Choose if we use a or a invert

not not1(a\_inv, a);

mux2x1 mux1(a,a\_inv,ainvert,a1);

// Chose if we use b or b invert

not not2(b\_inv, b);

mux2x1 mux2(b,b\_inv,binvert,b1);

and and1(a\_and\_b, a1, b1);

or or1(a\_or\_b, a1, b1);

fulladder adder1(sum,carryout,a1,b1,carryin);

mux4x1 mux3(a\_and\_b,a\_or\_b,sum,less,op[1:0],result);

endmodule

// 1-bit ALU for smb

module ALUmsb (a,b,ainvert, binvert,op,less,carryin,carryout,result,sum);

input a,b,less,carryin,ainvert,binvert;

input [1:0] op;

output carryout,result,sum;

wire sum, a\_and\_b, a\_or\_b, b\_inv, a\_inv;

// Choose if we use a or a invert

not not1(a\_inv, a);

mux2x1 mux1(a,a\_inv,ainvert,a1);

// Choose if we use b or b invert

not not2(b\_inv, b);

mux2x1 mux2(b,b\_inv,binvert,b1);

and and1(a\_and\_b, a1, b1);

or or1(a\_or\_b, a1, b1);

fulladder adder1(sum,carryout,a1,b1,carryin);

mux4x1 mux3(a\_and\_b,a\_or\_b,sum,less,op[1:0],result);

endmodule

// Adders

module halfadder (S,carryOut,A,B);

input A,B;

output S,carryOut;

xor (S,A,B);

and (carryOut,A,B);

endmodule

module fulladder (S,carryOut,A,B,carryIn);

input A,B,carryIn;

output S,carryOut;

wire S1,D1,D2;

halfadder HA1 (S1,D1,A,B),

HA2 (S,D2,S1,carryIn);

or g1(carryOut,D2,D1);

endmodule

// 16-bit Full Adder

// Used for Extra Work to increment PC by 2

module adder\_16bit(sum,carryOut,A,B);

input [15:0] A;

input [15:0] B;

output [15:0] sum;

output carryOut;

wire [15:0] carry;

fulladder FA0(sum[0],carry[0],A[0],B[0],1'b0);

fulladder FA1(sum[1],carry[1],A[1],B[1],carry[0]);

fulladder FA2(sum[2],carry[2],A[2],B[2],carry[1]);

fulladder FA3(sum[3],carry[3],A[3],B[3],carry[2]);

fulladder FA4(sum[4],carry[4],A[4],B[4],carry[3]);

fulladder FA5(sum[5],carry[5],A[5],B[5],carry[4]);

fulladder FA6(sum[6],carry[6],A[6],B[6],carry[5]);

fulladder FA7(sum[7],carry[7],A[7],B[7],carry[6]);

fulladder FA8(sum[8],carry[8],A[8],B[8],carry[7]);

fulladder FA9(sum[9],carry[9],A[9],B[9],carry[8]);

fulladder FA10(sum[10],carry[10],A[10],B[10],carry[9]);

fulladder FA11(sum[11],carry[11],A[11],B[11],carry[10]);

fulladder FA12(sum[12],carry[12],A[12],B[12],carry[11]);

fulladder FA13(sum[13],carry[13],A[13],B[13],carry[12]);

fulladder FA14(sum[14],carry[14],A[14],B[14],carry[13]);

fulladder FA15(sum[15],carryOut,A[15],B[15],carry[14]);

endmodule

// Multiplexors

// 16-bit 4x1 Mux

module mux16bit(a, b, c, d, select, OUT);

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

input [1:0] select;

output [15:0] OUT;

//16 4x1 mux

mux4x1 mux0(a[0], b[0], c[0], d[0], select, OUT[0]);

mux4x1 mux1(a[1], b[1], c[1], d[1], select, OUT[1]);

mux4x1 mux2(a[2], b[2], c[2], d[2], select, OUT[2]);

mux4x1 mux3(a[3], b[3], c[3], d[3], select, OUT[3]);

mux4x1 mux4(a[4], b[4], c[4], d[4], select, OUT[4]);

mux4x1 mux5(a[5], b[5], c[5], d[5], select, OUT[5]);

mux4x1 mux6(a[6], b[6], c[6], d[6], select, OUT[6]);

mux4x1 mux7(a[7], b[7], c[7], d[7], select, OUT[7]);

mux4x1 mux8(a[8], b[8], c[8], d[8], select, OUT[8]);

mux4x1 mux9(a[9], b[9], c[9], d[9], select, OUT[9]);

mux4x1 mux10(a[10], b[10], c[10], d[10], select, OUT[10]);

mux4x1 mux11(a[11], b[11], c[11], d[11], select, OUT[11]);

mux4x1 mux12(a[12], b[12], c[12], d[12], select, OUT[12]);

mux4x1 mux13(a[13], b[13], c[13], d[13], select, OUT[13]);

mux4x1 mux14(a[14], b[14], c[14], d[14], select, OUT[14]);

mux4x1 mux15(a[15], b[15], c[15], d[15], select, OUT[15]);

endmodule

// 1-bit 4x1 Mux

module mux4x1(a,b,c,d,select,OUT);

input a,b,c,d;

input [1:0] select;

output OUT;

mux2x1 mux1(a, b, select[0], m1);

mux2x1 mux2(c, d, select[0], m2);

mux2x1 mux3(m1, m2, select[1], OUT);

endmodule

// 16-bit 2x1 Mux

// Rewritten For Extra Work to use 8 2-bit 2x1 Mux's instead of 16 1-bit 2x1 Mux's

module mux2x1\_16bit(A, B, select, OUT);

input [15:0] A,B;

input select;

output [15:0] OUT;

//8 2-bit 2x1 mux's

mux2x1\_2bit mux1(A[1:0],B[1:0],select,OUT[1:0]);

mux2x1\_2bit mux2(A[3:2],B[3:2],select,OUT[3:2]);

mux2x1\_2bit mux3(A[5:4],B[5:4],select,OUT[5:4]);

mux2x1\_2bit mux4(A[7:6],B[7:6],select,OUT[7:6]);

mux2x1\_2bit mux5(A[9:8],B[9:8],select,OUT[9:8]);

mux2x1\_2bit mux6(A[11:10],B[11:10],select,OUT[11:10]);

mux2x1\_2bit mux7(A[13:12],B[13:12],select,OUT[13:12]);

mux2x1\_2bit mux8(A[15:14],B[15:14],select,OUT[15:14]);

endmodule

// 2-bit 2x1 Mux

module mux2x1\_2bit(A,B,select,OUT);

input [1:0] A,B;

input select;

output [1:0] OUT;

//2 2x1 muxs

mux2x1 mux1(A[0], B[0], select, OUT[0]),

mux2(A[1], B[1], select, OUT[1]);

endmodule

// 1-bit 2x1 Mux

module mux2x1(A,B,select,OUT);

input A,B,select;

output OUT;

not not1(i0, select);

and and1(i1, A, i0);

and and2(i2, B, select);

or or1(OUT, i1, i2);

endmodule

module MainControl (Op,Control);

input [3:0] Op;

output reg [10:0] Control;

// RegDst,ALUSrc,MemtoReg,RegWrite,MemWrite,BEQ,BNE,ALUOp

always @(Op) case (Op)

4'b0000: Control <= 11'b10010\_0\_0\_0010; // add

4'b0001: Control <= 11'b10010\_0\_0\_0110; // sub

4'b0010: Control <= 11'b10010\_0\_0\_0000; // and

4'b0011: Control <= 11'b10010\_0\_0\_0001; // or

4'b0100: Control <= 11'b10010\_0\_0\_1101; // nor

4'b0101: Control <= 11'b10010\_0\_0\_1100; // nand

4'b0110: Control <= 11'b10010\_0\_0\_0111; // slt

4'b0111: Control <= 11'b01010\_0\_0\_0010; // addi

4'b1000: Control <= 11'b01110\_0\_0\_0010; // lw

4'b1001: Control <= 11'b01001\_0\_0\_0010; // sw

4'b1010: Control <= 11'b00000\_1\_0\_0110; // beq

4'b1011: Control <= 11'b00000\_0\_1\_0110; // bne

endcase

endmodule

module CPU (clock,PC,WD,IR);

input clock;

output [15:0] WD,IR,PC;

reg[15:0] PC;

reg[15:0] IMemory[0:1023], DMemory[0:1023];

wire [15:0] IR,NextPC,A,B,ALUOut,RD2,SignExtend, Target, PCplus4;

wire [3:0] ALUctl;

wire [1:0] WR;

// Test Program

initial begin

IMemory[0] = 16'b1000\_00\_01\_00000000; // lw $t1, 0($0)

IMemory[1] = 16'b1000\_00\_10\_00000010; // lw $t2, 2($0)

IMemory[2] = 16'b0110\_01\_10\_11\_000000; // slt $t3, $t1, $t2

IMemory[3] = 16'b1010\_11\_00\_00000010; // beq $t3, $0, IMemory[6]

IMemory[4] = 16'b1001\_00\_01\_00000010; // sw $t1, 2($0)

IMemory[5] = 16'b1001\_00\_10\_00000000; // sw $t2, 0($0)

IMemory[6] = 16'b1000\_00\_01\_00000000; // lw $t1, 0($0)

IMemory[7] = 16'b1000\_00\_10\_00000010; // lw $t2, 2($0)

IMemory[8] = 16'b0100\_10\_10\_10\_000000; // nor $t2, $t2, $t2 (invert all bits)

IMemory[9] = 16'b0111\_10\_10\_00000001; // addi $t2, $t2, 1 (add 1, result is 2s complement)

IMemory[10] =16'b0000\_01\_10\_11\_000000; // add $t3, $t1, $t2

// Data

DMemory[0] = 16'd5;

DMemory[1] = 16'd7;

end

initial PC = 0;

assign IR = IMemory[PC>>1];

assign SignExtend = {{16{IR[7]}},IR[7:0]}; // Sign extends Instruction bits [7:0] to 16 bits by copying the 7th bit 8 times

reg\_file rf (IR[11:10],IR[9:8],WR,WD,RegWrite,A,RD2,clock);

// For Extra Work

// Use 16-bit Full Adder instead of ALU to increment PC

adder\_16bit Adder (PCplus4,\_,PC,16'd2);

ALU ex (ALUctl, A, B, ALUOut, Zero);

ALU branch (4'b0010,SignExtend<<1,PCplus4,Target,Unused2); // ALU branch

MainControl MainCtr (IR[15:12],{RegDst,ALUSrc,MemtoReg,RegWrite,MemWrite,Beq,Bne,ALUctl});

mux2x1\_2bit m1 (IR[9:8],IR[7:6],RegDst,WR); // Mux for RegDst

mux2x1\_16bit m2(RD2,SignExtend,ALUSrc,B); // Mux for ALUsrc

mux2x1\_16bit m3(ALUOut,DMemory[ALUOut>>1],MemtoReg,WD); // Mux for MemtoReg

// Branch Control

and (Bz,Beq,Zero);

and (Bnz,Bne,~Zero);

or (Br,Bz,Bnz);

mux2x1\_16bit m4(PCplus4,Target,Br,NextPC);

always @(negedge clock) begin //$display("%b %b %b %b",ALUctl, A, B, ALUOut);

PC <= NextPC;

if (MemWrite) DMemory[ALUOut>>1] <= RD2;

end

endmodule

// Test module

module test ();

reg clock;

wire signed [15:0] WD,IR,PC;

CPU test\_cpu(clock,PC,WD,IR);

always #1 clock = ~clock;

initial begin

$display ("PC IR WD");

$monitor ("%2d %b %3d (%b)",PC,IR,WD,WD);

clock = 1;

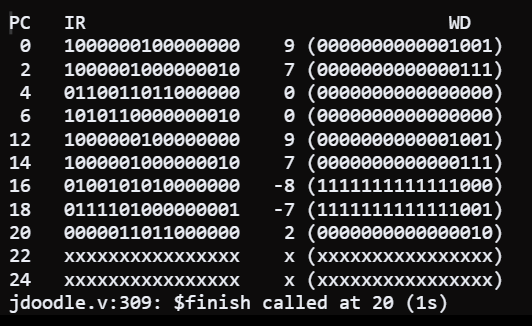
#20 $finish;

end

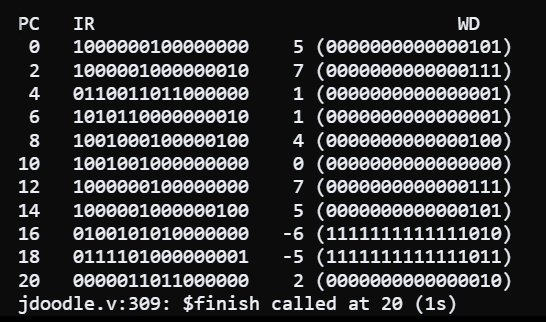
endmodule

**Output:**

$t1 = 7, $t2 = 5

****

$t1 = 7, $t2 = 5



$t1 = 9, $t2 = 7

