William Horton, Andy Weekes, Tyler Martin
5/2/23
Prof. Zdravko Markov
CS 385-01

Final Report

Instruction Set Architecture:

ALU Control lines

Ainvert	Bnegate	Operation	Instruction
0	0	00	and
0	0	01	or
0	0	10	add
0	1	10	sub
0	1	11	slt
1	1	00	nor
1	1	01	nand

R Format - (ADD, SUB, AND, OR, NOR, NAND,SLT)

op	**rs**	**rt**	**rd**	**unused**
4	2	2	2	6

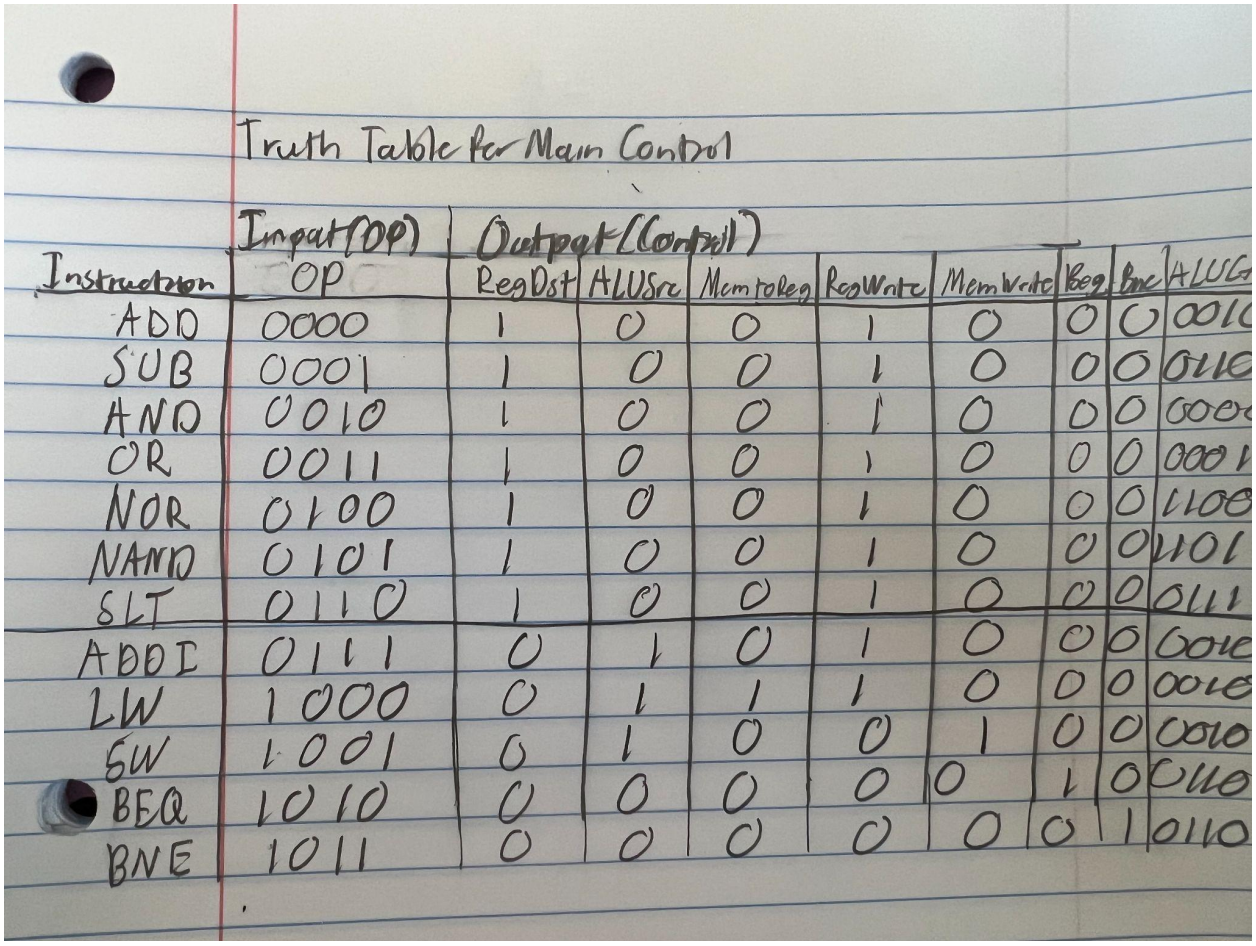
I Format - (ADDI, LW, SW, BEQ, BNE)

op	**rs**	**rt**	**address/value**
4	2	2	8

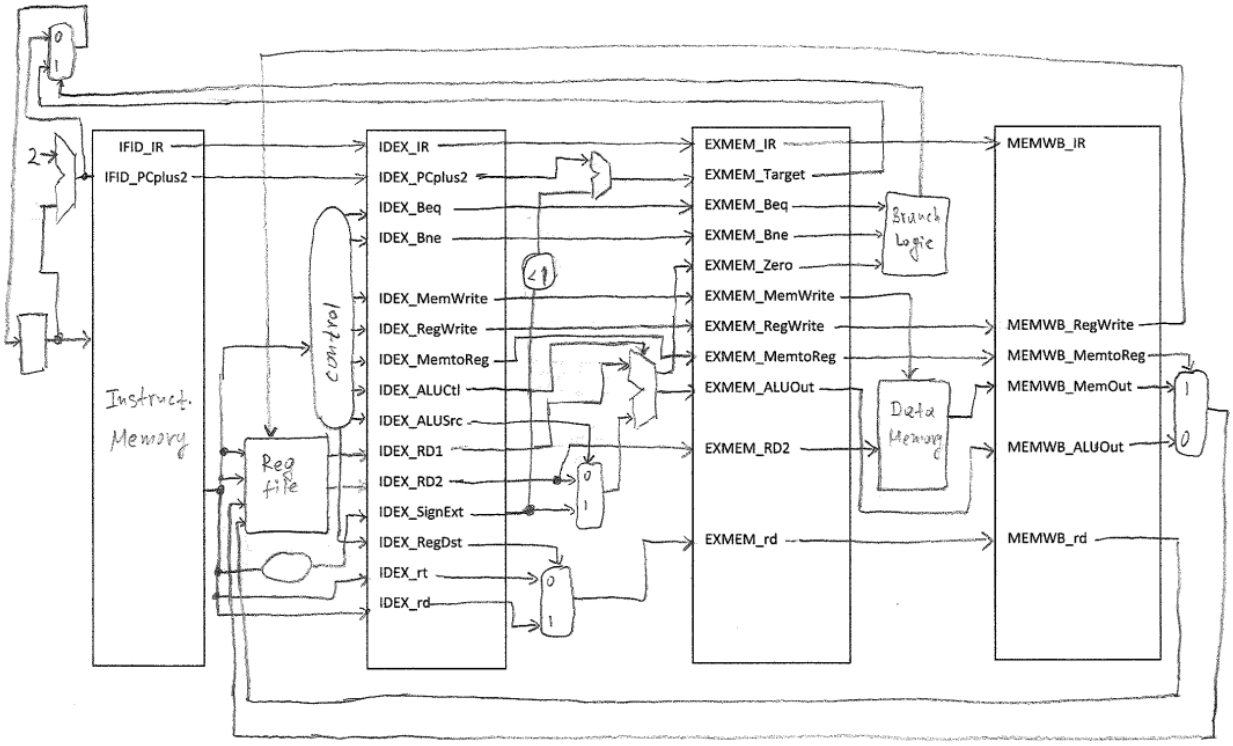
Instruction	Code	Format	Meaning
ADD	0000	R-Type	add $t1, $t2, $t3 ADD = Addition $t1 = the result of $t2 + $t3
SUB	0001	R-Type	sub $t1, $t2, $t3 SUB = Subtraction $t1 = the result of $t2 - $t3
AND	0010	R-Type	and $t1, $t2, $t3 AND = Bitwise *AND* $t1 = result of $t2 *AND* $t3
OR	0011	R-Type	or $t1, $t2, $t3 OR = Bitwise *OR* $t1 = the result of $t2 *OR* $t3
NOR	0100	R-Type	nor $t1, $t2, $t3 NOR = Bitwise *NOR* $t1 = the result of $t2 *NOR* $t3
NAND	0101	R-Type	nand $t1, $t2, $t3 NAND = Bitwise *NAND* $t1 = the result of $t2 *NAND* $t3
SLT	0110	R-Type	slt $t1, $t2, $t3 SLT = Set Less Than $t1 = 1 if $t2 < $t3 $t1 = 0 if $t2 ≥ $t3 $t2 - variable to be checked $t3 - variable to be checked with
ADDI	0111	I-Type	addi $t1, $0, value ADDI - Add immediate value $t1 = value $0 - Source Register $t1 - Destination Register value - Immediate value to be added to $t1
BEQ	1010	I-Type	beq $t1, $t2, immed BEQ = Branch if Equal Program jumps “immed” if $t1 = $t2

Instruction	Code	Format	Meaning
BNE	1011	I-Type	bne $t1, $t2, immed BNE = Branch if Not Equal Program jumps “immed” as long as $t1 does not equal $t2
LW	1000	I-Type	lw $t0, offset($t1) LW = Load Word Loads word (32 Bit) from $t1 plus the offset in bits and stores it in $t0
SW	1001	I-Type	sw $t0, offset($t1) SW = Store Word Stores word (32 Bit) from $t0 to $t1 plus the offset in bits

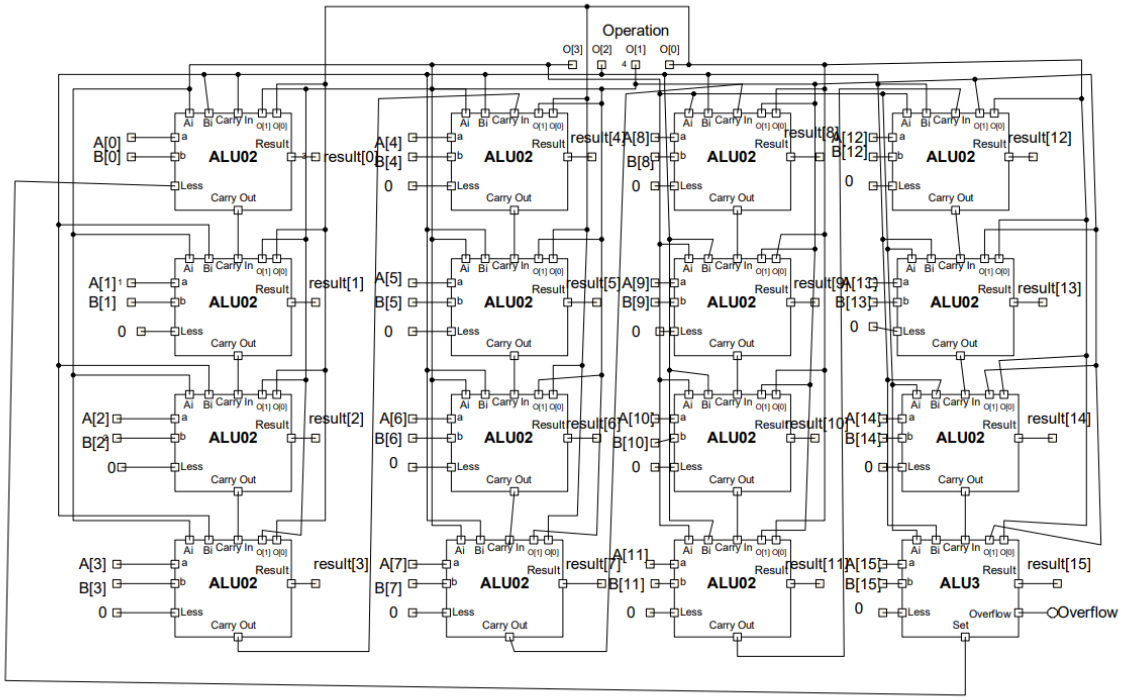
Logic Diagrams / Truth Tables:



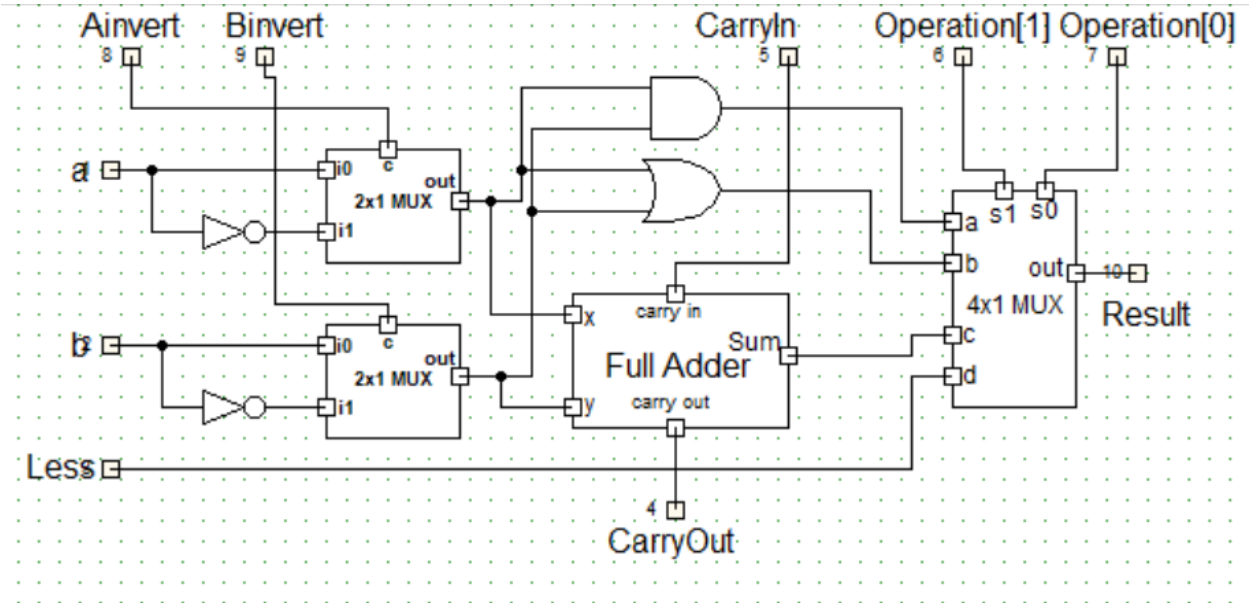
Final CPU



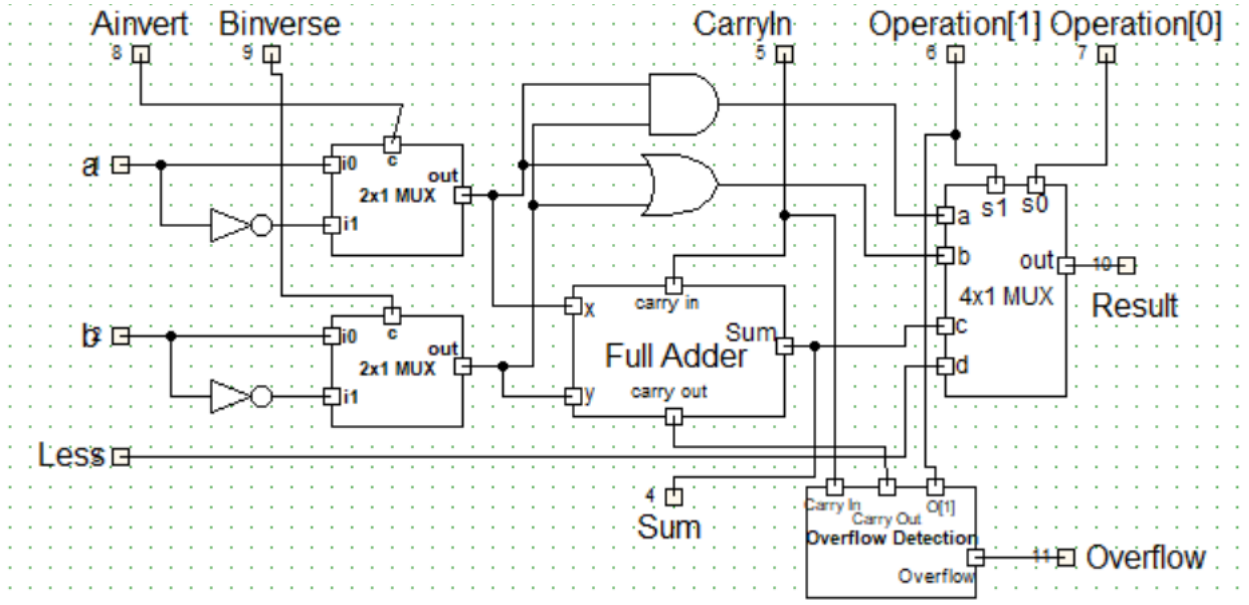
16 Bit ALU



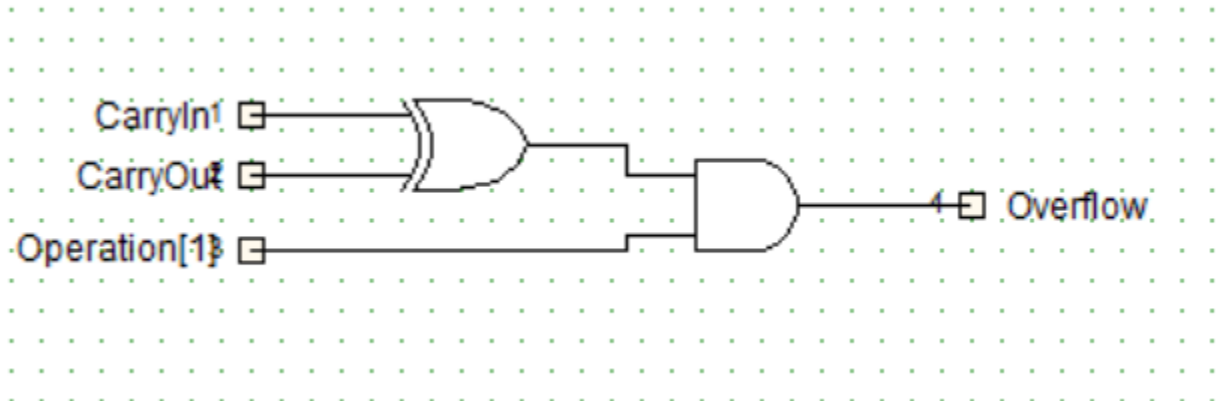
ALU 0, 1, 2, … , 14

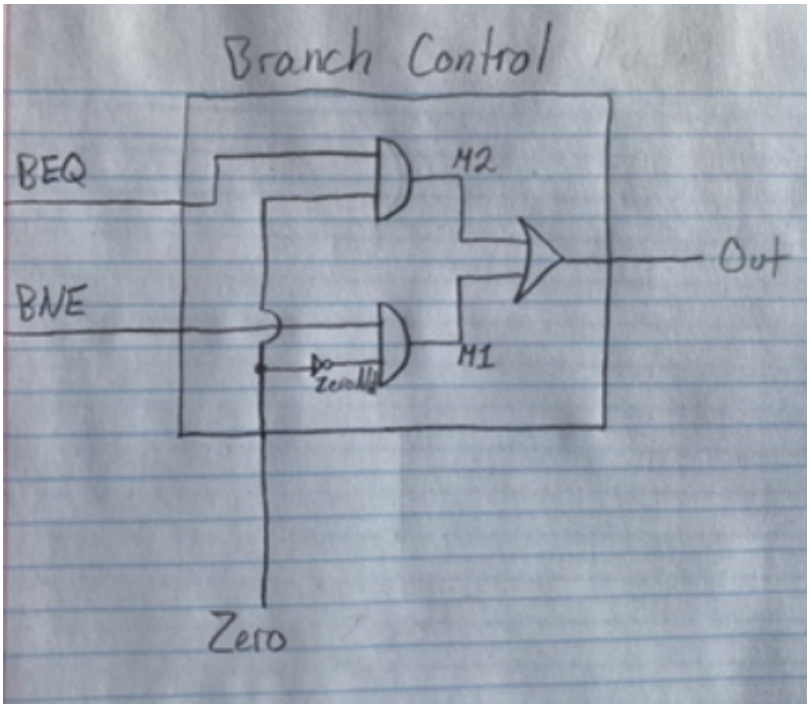


ALU 15

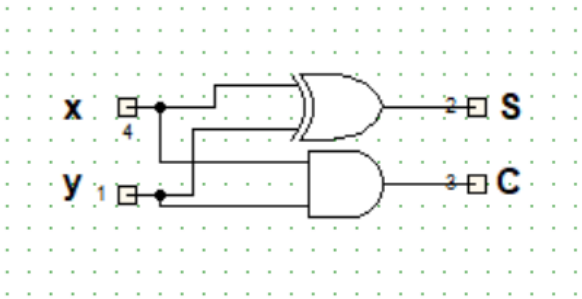


Overflow Detection

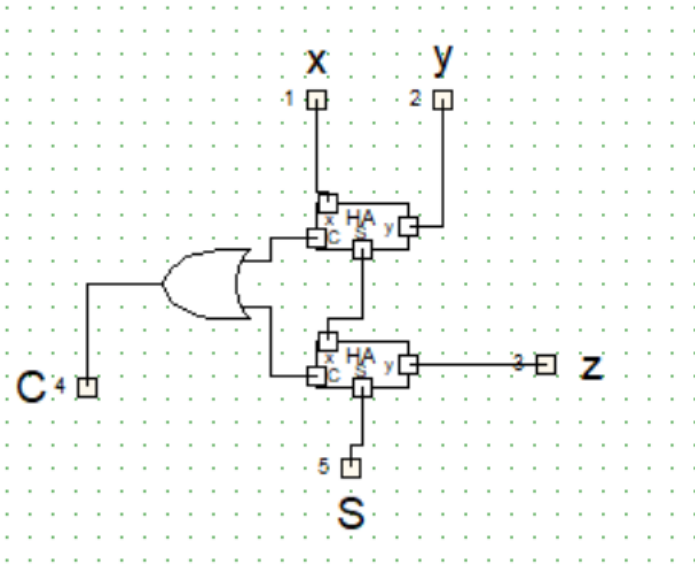




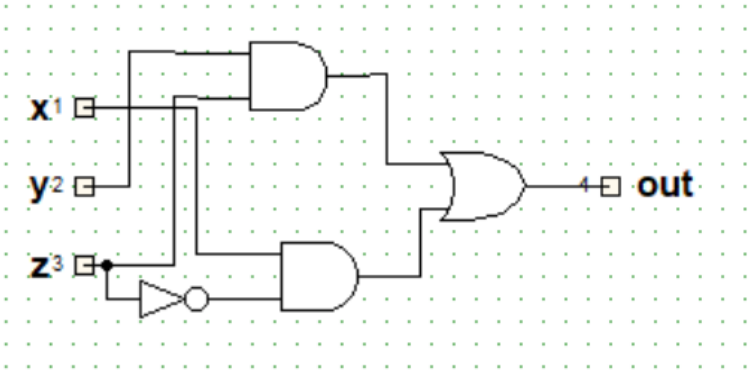
Half Adder



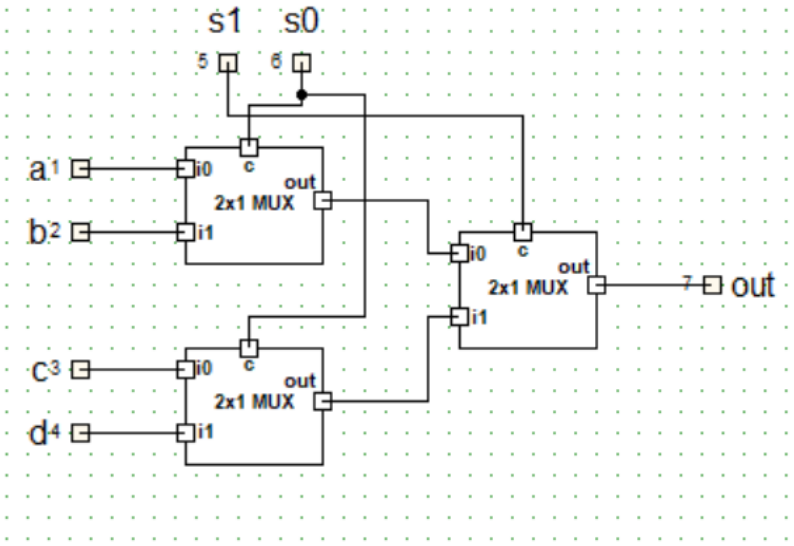
Full Adder



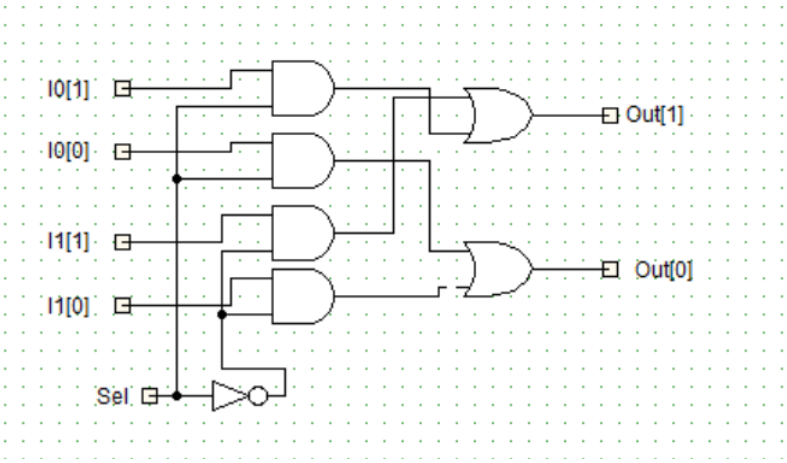
2x1 Multiplexer



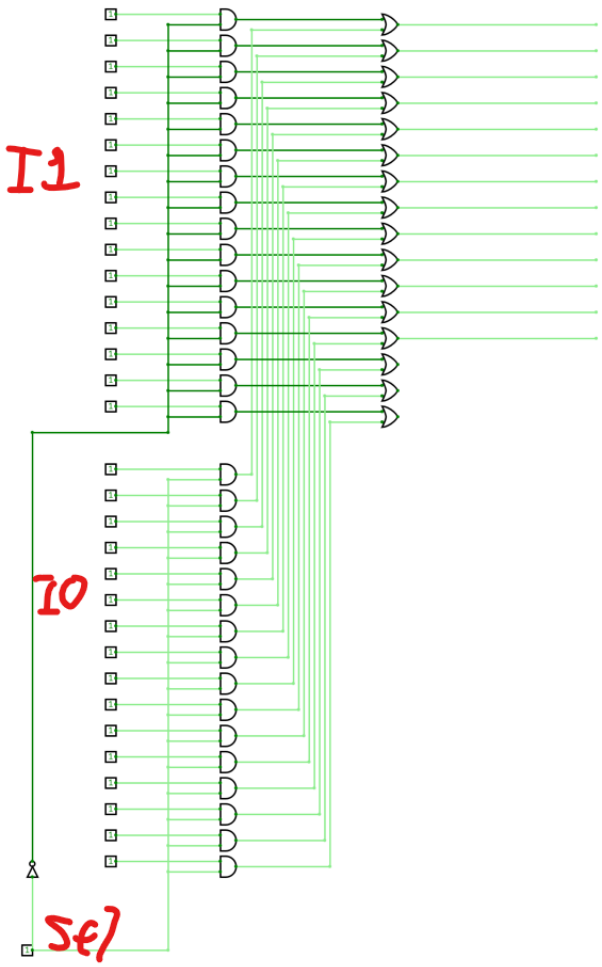
4x1 Multiplexer



2 Bit 2x1 Multiplexer



16 Bit 2x1 Multiplexer



Verilog Source Code:

```
// Behavioral model of MIPS - pipelined implementation

// CS 385-01 Progress Report 4
// by Andrew Weekes, William Horton, and Tyler Martin

// register file module
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 (1-bit ALU bihavioral implementation)
module alu (op,a,b,result,zero);

   // 16 bit A and B
   input [15:0] a;
   input [15:0] b;

   // op is same, result is 16 bits
   input [3:0] op;
   output [15:0] result;
   output zero;
   // change to 16 1 bit ALUs
   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[0],result[1],result[2],result[3],result[4],result[5],result[6],result[7],result[8],result[9],result[10],result[11],result[12],result[13],result[14],res
ult[15]);
endmodule

// 1-bit ALU for bits 0-2
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;

   // beginning gate level
   //create inverses
   not (aNot,a);
   not (bNot,b);

   // if both binvert and !Ainverse, then either NOR or NAND

   // 2x1 multiplexer and Ainverse and Binverse
   multi2x1 a1(a,aNot,ainvert,aOut);
   multi2x1 b1(b,bNot,binvert,bOut);

   // AND or NOR operation
   and (andOut,aOut,bOut);

   // OR or NAND operation
   or (orOut,aOut,bOut);

   // SUM or SUB operation
   fulladder fa(sum,carryout,aOut,bOut,carryin);

   // 4x1 multiplexer
   multi4x1 r1(andOut,orOut,sum,less,op[1],op[0],result);
endmodule

// 1-bit ALU for the most significant bit
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,sum, result;

   // beginning gate level
   //create inverses
   not (aNot,a);
   not (bNot,b);

   // if both binvert and !Ainverse, then either NOR or NAND

   // 2x1 multiplexer and Ainverse and Binverse
   multi2x1 a1(a,aNot,ainvert,aOut);
   multi2x1 b1(b,bNot,binvert,bOut);

   // AND or NOR operation
   and (andOut,aOut,bOut);

   // OR or NAND operation
   or (orOut,aOut,bOut);

   // SUM or SUB operation
   fulladder fa(sum,carryout,aOut,bOut,carryin);

   // 4x1 multiplexer
   multi4x1 r1(andOut,orOut,sum,less,op[1],op[0],result);

   // Overflow detection, not using xor as possibility of negated B when doing subtraction not causing overflow

   xor(overflow1,carryin,carryout);
   and(Overflow,overflow1,op[1]);
```

```
endmodule

// Support Modules

// Description of half adder (see Fig 4-5b)
module halfadder (S,C,x,y);
  input x,y;
  output S,C;
//Instantiate primitive gates
  xor (S,x,y);
  and (C,x,y);
endmodule

//Description of full adder (see Fig 4-8)
module fulladder (S,C,x,y,z);
  input x,y,z;
  output S,C;
  wire S1,D1,D2; //Outputs of first XOR and two AND gates
//Instantiate the halfadder
   halfadder HA1 (S1,D1,x,y),
             HA2 (S,D2,S1,z);
   or g1(C,D2,D1);
endmodule

// 2x1 multiplexer
module multi2x1(x,y,z,out);
   input x,y,z;
   output out;

// possibly other order?
   not n1(zn,z);
   and a1(m1,y,z),
       a2(m2,x,zn);
   or o1(out,m1,m2);
endmodule

// 4x1 multiplexer
module multi4x1(a,b,c,d,s1,s0,out);
   input a,b,c,d,s1,s0;
   output out;

   multi2x1 m1(a,b,s0,c0);
   multi2x1 m2(c,d,s0,c1);
   multi2x1 m3(c0,c1,s1,out);
endmodule

module branchControlMux(Bne, Beq, Zero, Out);
   input Beq, Bne, Zero;
   output Out;

   not n1(ZeroNot, Zero);
   and a1(m1, Bne, ZeroNot),
       a2(m2, Beq, Zero);
   or  o1(Out, m1, m2);
endmodule

module MainControl (Op,Control);
  input [3:0] Op;
  output reg [10:0] Control;
// RegDst,ALUSrc,MemtoReg,RegWrite,MemWrite,Beq,Bne, = 1 bit; ALUctl = 4 bits; Total = 11 bits
  always @(Op) case (Op)
```

```
    //R-type instructions
    4'b0000: Control <= 11'b10010_00_0010; // Rtype - ADD - complete
    4'b0001: Control <= 11'b10010_00_0110; // Rtype - SUB - complete
    4'b0010: Control <= 11'b10010_00_0000; // Rtype - AND - complete
    4'b0011: Control <= 11'b10010_00_0001; // Rtype - OR - complete
    4'b0100: Control <= 11'b10010_00_1100; // Rtype - NOR - complete
    4'b0101: Control <= 11'b10010_00_1101; // Rtype - NAND - complete
    4'b0110: Control <= 11'b10010_00_0111; // Rtype - SLT - complete

    //I-type instructions
    4'b0111: Control <= 11'b01010_00_0010; // Itype - ADDI - complete
    4'b1000: Control <= 11'b01110_00_0010; // LW - last 4 bits = code for add because LW adds offset to base register - complete
    4'b1001: Control <= 11'b01001_00_0010; // SW -
    4'b1010: Control <= 11'b00000_10_0110; // BEQ - last 4 bits = code for sub because BEQ subtracts address - complete
    4'b1011: Control <= 11'b00000_01_0110; // BNE - last 4 bits = code for sub because BNE subtracts address - complete
  endcase
endmodule

// Gate Level 2 bit 2x1 Multiplexer - if Sel is true - return first term, if false - return second term
module double2x1mux (I0,I1,Sel,Out);
  input [1:0] I0,I1;
  input Sel;
  output wire [1:0] Out;

  // gate level - done
  not (Selnot,Sel);

  // A and gates
  and (Aand1,I1[1],Selnot);
  and (Aand0,I1[0],Selnot);

  // B and gates

  and (Band1,I0[1],Sel);
  and (Band0,I0[0],Sel);

  // Final Or Gates for Output

  or (Out[1],Aand1,Band1);
  or (Out[0],Aand0,Band0);

endmodule

// Gate Level Quad 2x1 Multiplexer - if Sel is true - return first term, if false - return second term
module hexabit2x1mux (I0,I1,Sel,Out);
  input [15:0] I0,I1;
  input Sel;
  output wire [15:0] Out;

  // gate level - done
  not (Selnot,Sel);

  // A and gates
  and (Aand15,I1[15],Selnot);
  and (Aand14,I1[14],Selnot);
  and (Aand13,I1[13],Selnot);
  and (Aand12,I1[12],Selnot);
  and (Aand11,I1[11],Selnot);
  and (Aand10,I1[10],Selnot);
  and (Aand9,I1[9],Selnot);
  and (Aand8,I1[8],Selnot);
```

```
  and (Aand7,I1[7],Selnot);
  and (Aand6,I1[6],Selnot);
  and (Aand5,I1[5],Selnot);
  and (Aand4,I1[4],Selnot);
  and (Aand3,I1[3],Selnot);
  and (Aand2,I1[2],Selnot);
  and (Aand1,I1[1],Selnot);
  and (Aand0,I1[0],Selnot);

  // B and gates
  and (Band15,I0[15],Sel);
  and (Band14,I0[14],Sel);
  and (Band13,I0[13],Sel);
  and (Band12,I0[12],Sel);
  and (Band11,I0[11],Sel);
  and (Band10,I0[10],Sel);
  and (Band9,I0[9],Sel);
  and (Band8,I0[8],Sel);
  and (Band7,I0[7],Sel);
  and (Band6,I0[6],Sel);
  and (Band5,I0[5],Sel);
  and (Band4,I0[4],Sel);
  and (Band3,I0[3],Sel);
  and (Band2,I0[2],Sel);
  and (Band1,I0[1],Sel);
  and (Band0,I0[0],Sel);

  // Final Or Gates for Output
  or (Out[15],Aand15,Band15);
  or (Out[14],Aand14,Band14);
  or (Out[13],Aand13,Band13);
  or (Out[12],Aand12,Band12);
  or (Out[11],Aand11,Band11);
  or (Out[10],Aand10,Band10);
  or (Out[9],Aand9,Band9);
  or (Out[8],Aand8,Band8);
  or (Out[7],Aand7,Band7);
  or (Out[6],Aand6,Band6);
  or (Out[5],Aand5,Band5);
  or (Out[4],Aand4,Band4);
  or (Out[3],Aand3,Band3);
  or (Out[2],Aand2,Band2);
  or (Out[1],Aand1,Band1);
  or (Out[0],Aand0,Band0);

endmodule

// PART OF OLD MIPS - DONT NEED IT - EVERYTHING COMES FROM MAIN CONTROL

module CPU (clock,PC,IFID_IR,IDEX_IR,EXMEM_IR,MEMWB_IR,WD);
  input clock;
  output [15:0] PC,IFID_IR,IDEX_IR,EXMEM_IR,MEMWB_IR,WD;

  initial begin
// Program: swap memory cells (if needed) and compute absolute value |5-7|=2

//  ----Program with nop's----
    IMemory[0]  = 16'b1000_00_01_00000000;   // lw $1, 0($0)
    IMemory[1]  = 16'b1000_00_10_00000010;   // lw $2, 2($0)
    IMemory[2]  = 16'b0000000000000000;      // nop
    IMemory[3]  = 16'b0000000000000000;      // nop
```

```
  IMemory[4]  = 16'b0000000000000000;      // nop
  IMemory[5]  = 16'b0110_01_10_11_000000;  // slt $3, $1, $2
  IMemory[6]  = 16'b0000000000000000;      // nop
  IMemory[7]  = 16'b0000000000000000;      // nop
  IMemory[8]  = 16'b0000000000000000;      // nop
  IMemory[9]  = 16'b1010_00_11_00000101;   // beq $3, $0, IMemory[15]
  //IMemory[9]  = 16'b1011_00_11_00000101;   // bne $3, $0, IMemory[15]
  IMemory[10] = 16'b0000000000000000;      // nop
  IMemory[11] = 16'b0000000000000000;      // nop
  IMemory[12] = 16'b0000000000000000;      // nop
  IMemory[13] = 16'b1001_00_01_00000010;   // sw $1, 2($0)
  IMemory[14] = 16'b1001_00_10_00000000;   // sw $2, 0($0)
  IMemory[15] = 16'b0000000000000000;      // nop
  IMemory[16] = 16'b0000000000000000;      // nop
  IMemory[17] = 16'b0000000000000000;      // nop
  IMemory[18] = 16'b1000_00_01_00000000;   // lw $1, 0($0)
  IMemory[19] = 16'b1000_00_10_00000010;   // lw $2, 2($0)
  IMemory[20] = 16'b0000000000000000;      // nop
  IMemory[21] = 16'b0000000000000000;      // nop
  IMemory[22] = 16'b0000000000000000;      // nop
  IMemory[23] = 16'b0100_10_10_10_000000;  // nor $2, $2, $2 (sub $3, $1, $2 in two's complement)
  IMemory[24] = 16'b0000000000000000;      // nop
  IMemory[25] = 16'b0000000000000000;      // nop
  IMemory[26] = 16'b0000000000000000;      // nop
  IMemory[27] = 16'b0111_10_10_00000001;   // addi $2, $2, 1
  IMemory[28] = 16'b0000000000000000;      // nop
  IMemory[29] = 16'b0000000000000000;      // nop
  IMemory[30] = 16'b0000000000000000;      // nop
  IMemory[31] = 16'b0000_01_10_11_000000;  // add $3, $1, $2

//  ----Program without nop's
//   IMemory[0]  = 16'b1000_00_01_00000000;   // lw $1, 0($0)
//   IMemory[1]  = 16'b1000_00_10_00000010;   // lw $2, 2($0)
//   IMemory[2]  = 16'b0110_01_10_11_000000;  // slt $3, $1, $2
//   IMemory[3]  = 16'b1010_00_11_00000101;   // beq $3, $0, IMemory[15]
//   //IMemory[3]  = 16'b1011_00_11_00000101;   // bne $3, $0, IMemory[15]
//   IMemory[4] = 16'b1001_00_01_00000010;   // sw $1, 2($0)
//   IMemory[5] = 16'b1001_00_10_00000000;   // sw $2, 0($0)
//   IMemory[6] = 16'b1000_00_01_00000000;   // lw $1, 0($0)
//   IMemory[7] = 16'b1000_00_10_00000010;   // lw $2, 2($0)
//   IMemory[8] = 16'b0100_10_10_10_000000;  // nor $2, $2, $2 (sub $3, $1, $2 in two's complement)
//   IMemory[9] = 16'b0111_10_10_00000001;   // addi $2, $2, 1
//   IMemory[10] = 16'b0000_01_10_11_000000;  // add $3, $1, $2


// Data
  DMemory[0] = 16'd5;     // switch the cells and see how the simulation output changes
  DMemory[1] = 16'd7;     // (beq is taken if DMemory[0]=7; DMemory[1]=5, not taken otherwise)
 end

// VERY SIMILAR TO LAST PROJECT - ALL EXPLAINED ON DIAGRAM
// Pipeline
// IF
  wire [15:0] PCplus2, NextPC;
  reg[15:0] PC, IMemory[0:1023], IFID_IR, IFID_PCplus2;
  alu fetch (4'b0010,PC,16'd2,PCplus2,Unused1);
  // SHOULD BE BRANCHING LOGIC WHERE BRANCH IS
  //assign NextPC = (EXMEM_Beq && EXMEM_Zero || EXMEM_Bne && ~EXMEM_Zero) ? EXMEM_Target: PCplus2;
  //hexabit2x1mux BranchControl(EXMEM_Target,PCplus2,(EXMEM_Beq && EXMEM_Zero || EXMEM_Bne &&
~EXMEM_Zero),NextPC);
  hexabit2x1mux BranchControl2(EXMEM_Target,PCplus2,BranchOutput,NextPC);
```

```
// ID
  wire [10:0] Control;
  reg IDEX_RegWrite,IDEX_MemtoReg,
     IDEX_Beq, IDEX_Bne,  IDEX_MemWrite,
     IDEX_ALUSrc,  IDEX_RegDst;
  reg [3:0]  IDEX_ALUctl; // MUST BE ALU CTL - 4 bits
  wire [15:0] RD1,RD2,SignExtend, WD;
  reg [15:0] IDEX_PCplus2,IDEX_RD1,IDEX_RD2,IDEX_SignExt,IDEXE_IR;
  reg [15:0] IDEX_IR; // For monitoring the pipeline
  reg [1:0]  IDEX_rt,IDEX_rd;
  reg MEMWB_RegWrite; // part of MEM stage, but declared here before use (to avoid error)
  reg [1:0] MEMWB_rd; // part of MEM stage, but declared here before use (to avoid error)
  reg_file rf (IFID_IR[11:10],IFID_IR[9:8],MEMWB_rd,WD,MEMWB_RegWrite,RD1,RD2,clock);
  MainControl MainCtr (IFID_IR[15:12],Control);
  assign SignExtend = {{8{IFID_IR[7]}},IFID_IR[7:0]};
// EXE
  reg EXMEM_RegWrite,EXMEM_MemtoReg,
     EXMEM_Beq,EXMEM_Bne,EXMEM_MemWrite;
  wire [15:0] Target;
  reg EXMEM_Zero;
  reg [15:0] EXMEM_Target,EXMEM_ALUOut,EXMEM_RD2;
  reg [15:0] EXMEM_IR; // For monitoring the pipeline
  reg [1:0] EXMEM_rd;
  wire [15:0] B,ALUOut;
  wire [3:0] ALUctl;
  wire [1:0] WR;
  alu branch (4'b0010,IDEX_SignExt<<1,IDEX_PCplus2,Target,Unused2);
  // set out of order

  alu ex (IDEX_ALUctl, IDEX_RD1, B, ALUOut, Zero);
  // REMOVE - set ALUctl to complete control code
  assign B  = (IDEX_ALUSrc) ? IDEX_SignExt: IDEX_RD2;       // ALUSrc Mux
  assign WR = (IDEX_RegDst) ? IDEX_rd: IDEX_rt;             // RegDst Mux
// MEM
  reg MEMWB_MemtoReg;
  reg [15:0] DMemory[0:1023],MEMWB_MemOut,MEMWB_ALUOut;
  reg [15:0] MEMWB_IR; // For monitoring the pipeline
  wire [15:0] MemOut;
  assign MemOut = DMemory[EXMEM_ALUOut>>1];
  branchControlMux BranchControl1(EXMEM_Bne, EXMEM_Beq, EXMEM_Zero, BranchOutput);
  always @(negedge clock) if (EXMEM_MemWrite) DMemory[EXMEM_ALUOut>>1] <= EXMEM_RD2;
// WB
  assign WD = (MEMWB_MemtoReg) ? MEMWB_MemOut: MEMWB_ALUOut; // MemtoReg Mux

  initial begin
   PC = 0;
// Initialize pipeline registers

IDEX_RegWrite=1'b0;IDEX_MemtoReg=1'b0;IDEX_Beq=1'b0;IDEX_Bne=1'b0;IDEX_MemWrite=1'b0;IDEX_ALUSrc=1'b0;IDEX_RegDst=
1'b0;
   IFID_IR=1'b0;
   EXMEM_RegWrite=1'b0;EXMEM_MemtoReg=1'b0;EXMEM_Beq=1'b0;EXMEM_Bne=1'b0;EXMEM_MemWrite=1'b0;
   EXMEM_Target=1'b0;
   MEMWB_RegWrite=1'b0;MEMWB_MemtoReg=1'b0;
  end

// Running the pipeline
  always @(negedge clock) begin
// IF
   PC <= NextPC;
```

```
    IFID_PCplus2 <= PCplus2;
    IFID_IR <= IMemory[PC>>1];
// ID
    IDEX_IR <= IFID_IR; // For monitoring the pipeline
    // MUST BE BOTH IDEX_BNE AND BEQ
    {IDEX_RegDst,IDEX_ALUSrc,IDEX_MemtoReg,IDEX_RegWrite,IDEX_MemWrite,IDEX_Beq,IDEX_Bne,IDEX_ALUctl} <= Control;
    IDEX_PCplus2 <= IFID_PCplus2;
    IDEX_RD1 <= RD1;
    IDEX_RD2 <= RD2;
    IDEX_SignExt <= SignExtend;
    IDEX_rt <= IFID_IR[9:8];
    IDEX_rd <= IFID_IR[7:6];
// EXE
    EXMEM_IR <= IDEX_IR; // For monitoring the pipeline
    EXMEM_RegWrite <= IDEX_RegWrite;
    EXMEM_MemtoReg <= IDEX_MemtoReg;
    // BOTH BNE AND BEQ
    EXMEM_Beq <= IDEX_Beq;
    EXMEM_Bne <= IDEX_Bne;


    EXMEM_MemWrite <= IDEX_MemWrite;
    EXMEM_Target <= Target;
    EXMEM_Zero <= Zero;
    EXMEM_ALUOut <= ALUOut;
    EXMEM_RD2 <= IDEX_RD2;
    EXMEM_rd <= WR;

// MEM
    MEMWB_IR <= EXMEM_IR; // For monitoring the pipeline
    MEMWB_RegWrite <= EXMEM_RegWrite;
    MEMWB_MemtoReg <= EXMEM_MemtoReg;
    MEMWB_MemOut <= MemOut;
    MEMWB_ALUOut <= EXMEM_ALUOut;
    MEMWB_rd <= EXMEM_rd;
// WB
// Register write happens on neg edge of the clock (if MEMWB_RegWrite is asserted)
  end
endmodule

// Test module

module test ();
  reg clock;
  wire signed [15:0] PC,IFID_IR,IDEX_IR,EXMEM_IR,MEMWB_IR,WD;
  CPU test_cpu(clock,PC,IFID_IR,IDEX_IR,EXMEM_IR,MEMWB_IR,WD);
  always #1 clock = ~clock;
  initial begin
    $display ("PC   IFID_IR  IDEX_IR  EXMEM_IR MEMWB_IR  WD");
    $monitor ("%3d  %h     %h     %h     %h     %2d",PC,IFID_IR,IDEX_IR,EXMEM_IR,MEMWB_IR,WD);
    clock = 1;
    #69 $finish;
  end
endmodule
// 4 STAGES 4 PIPELINE REGISTERS - SHOWS HOW THE INSTRUCTION TRAVELS THROUGH THE PIPELINE
// EXECUTES IN LAST STAGE
```

Test Results:

BEQ Test 1

PC	IFID_IR	IDEX_IR	EXMEM_IR	MEMWB_IR	WD
0	0000	xxxx	xxxx	xxxx	x
2	8100	0000	xxxx	xxxx	x
4	8202	8100	0000	xxxx	x
6	0000	8202	8100	0000	0
8	0000	0000	8202	8100	5
10	0000	0000	0000	8202	7
12	66c0	0000	0000	0000	0
14	0000	66c0	0000	0000	0
16	0000	0000	66c0	0000	0
18	0000	0000	0000	66c0	1
20	a305	0000	0000	0000	0
22	0000	a305	0000	0000	0
24	0000	0000	a305	0000	0
26	0000	0000	0000	a305	-1
28	9102	0000	0000	0000	0
30	9200	9102	0000	0000	0
32	0000	9200	9102	0000	0
34	0000	0000	9200	9102	2
36	0000	0000	0000	9200	0
38	8100	0000	0000	0000	0
40	8202	8100	0000	0000	0
42	0000	8202	8100	0000	0
44	0000	0000	8202	8100	7
46	0000	0000	0000	8202	5
48	4a80	0000	0000	0000	0
50	0000	4a80	0000	0000	0
52	0000	0000	4a80	0000	0
54	0000	0000	0000	4a80	-6
56	7a01	0000	0000	0000	0
58	0000	7a01	0000	0000	0
60	0000	0000	7a01	0000	0
62	0000	0000	0000	7a01	-5
64	06c0	0000	0000	0000	0
66	xxxx	06c0	0000	0000	0
68	xxxx	xxxx	06c0	0000	0
70	xxxx	xxxx	xxxx	06c0	2

BEQ Test 2

PC	IFID_IR	IDEX_IR	EXMEM_IR	MEMWB_IR	WD
0	0000	xxxx	xxxx	xxxx	x
2	8100	0000	xxxx	xxxx	x
4	8202	8100	0000	xxxx	x
6	0000	8202	8100	0000	0
8	0000	0000	8202	8100	7
10	0000	0000	0000	8202	5
12	66c0	0000	0000	0000	0
14	0000	66c0	0000	0000	0
16	0000	0000	66c0	0000	0
18	0000	0000	0000	66c0	0
20	a305	0000	0000	0000	0
22	0000	a305	0000	0000	0
24	0000	0000	a305	0000	0
30	0000	0000	0000	a305	0
32	0000	0000	0000	0000	0
34	0000	0000	0000	0000	0
36	0000	0000	0000	0000	0
38	8100	0000	0000	0000	0
40	8202	8100	0000	0000	0
42	0000	8202	8100	0000	0
44	0000	0000	8202	8100	7
46	0000	0000	0000	8202	5
48	4a80	0000	0000	0000	0
50	0000	4a80	0000	0000	0
52	0000	0000	4a80	0000	0
54	0000	0000	0000	4a80	-6
56	7a01	0000	0000	0000	0
58	0000	7a01	0000	0000	0
60	0000	0000	7a01	0000	0
62	0000	0000	0000	7a01	-5
64	06c0	0000	0000	0000	0
66	xxxx	06c0	0000	0000	0
68	xxxx	xxxx	06c0	0000	0
70	xxxx	xxxx	xxxx	06c0	2
72	xxxx	xxxx	xxxx	xxxx	x
74	xxxx	xxxx	xxxx	xxxx	x

BEQ Without NOP's

PC	IFID_IR	IDEX_IR	EXMEM_IR	MEMWB_IR	WD
0	0000	xxxx	xxxx	xxxx	x
2	8100	0000	xxxx	xxxx	x
4	8202	8100	0000	xxxx	x
6	66c0	8202	8100	0000	0
8	a305	66c0	8202	8100	5
10	9102	a305	66c0	8202	7
12	9200	9102	a305	66c0	X
X	8100	9200	9102	a305	x
X	xxxx	8100	9200	9102	2
X	xxxx	xxxx	8100	9200	0
X	xxxx	xxxx	xxxx	8100	7
X	xxxx	xxxx	xxxx	xxxx	x
X	xxxx	xxxx	xxxx	xxxx	x
X	xxxx	xxxx	xxxx	xxxx	x
X	xxxx	xxxx	xxxx	xxxx	x
X	xxxx	xxxx	xxxx	xxxx	x
X	xxxx	xxxx	xxxx	xxxx	x
X	xxxx	xxxx	xxxx	xxxx	x
X	xxxx	xxxx	xxxx	xxxx	x

BNE Test 1

PC	IFID_IR	IDEX_IR	EXMEM_IR	MEMWB_IR	WD
0	0000	xxxx	xxxx	xxxx	x
2	8100	0000	xxxx	xxxx	x
4	8202	8100	0000	xxxx	x
6	0000	8202	8100	0000	0
8	0000	0000	8202	8100	5
10	0000	0000	0000	8202	7
12	66c0	0000	0000	0000	0
14	0000	66c0	0000	0000	0
16	0000	0000	66c0	0000	0
18	0000	0000	0000	66c0	1
20	b305	0000	0000	0000	0
22	0000	b305	0000	0000	0
24	0000	0000	b305	0000	0
30	0000	0000	0000	b305	-1
32	0000	0000	0000	0000	0
34	0000	0000	0000	0000	0
36	0000	0000	0000	0000	0
38	8100	0000	0000	0000	0
40	8202	8100	0000	0000	0
42	0000	8202	8100	0000	0
44	0000	0000	8202	8100	5
46	0000	0000	0000	8202	7
48	4a80	0000	0000	0000	0
50	0000	4a80	0000	0000	0
52	0000	0000	4a80	0000	0
54	0000	0000	0000	4a80	-8
56	7a01	0000	0000	0000	0
58	0000	7a01	0000	0000	0
60	0000	0000	7a01	0000	0
62	0000	0000	0000	7a01	-7
64	06c0	0000	0000	0000	0
66	xxxx	06c0	0000	0000	0
68	xxxx	xxxx	06c0	0000	0
70	xxxx	xxxx	xxxx	06c0	-2
72	xxxx	xxxx	xxxx	xxxx	x
74	xxxx	xxxx	xxxx	xxxx	x

BNE Test 2

PC	IFID_IR	IDEX_IR	EXMEM_IR	MEMWB_IR	WD
0	0000	xxxx	xxxx	xxxx	x
2	8100	0000	xxxx	xxxx	x
4	8202	8100	0000	xxxx	x
6	0000	8202	8100	0000	0
8	0000	0000	8202	8100	7
10	0000	0000	0000	8202	5
12	66c0	0000	0000	0000	0
14	0000	66c0	0000	0000	0
16	0000	0000	66c0	0000	0
18	0000	0000	0000	66c0	0
20	b305	0000	0000	0000	0
22	0000	b305	0000	0000	0
24	0000	0000	b305	0000	0
26	0000	0000	0000	b305	0
28	9102	0000	0000	0000	0
30	9200	9102	0000	0000	0
32	0000	9200	9102	0000	0
34	0000	0000	9200	9102	2
36	0000	0000	0000	9200	0
38	8100	0000	0000	0000	0
40	8202	8100	0000	0000	0
42	0000	8202	8100	0000	0
44	0000	0000	8202	8100	5
46	0000	0000	0000	8202	7
48	4a80	0000	0000	0000	0
50	0000	4a80	0000	0000	0
52	0000	0000	4a80	0000	0
54	0000	0000	0000	4a80	-8
56	7a01	0000	0000	0000	0
58	0000	7a01	0000	0000	0
60	0000	0000	7a01	0000	0
62	0000	0000	0000	7a01	-7
64	06c0	0000	0000	0000	0
66	xxxx	06c0	0000	0000	0
68	xxxx	xxxx	06c0	0000	0
70	xxxx	xxxx	xxxx	06c0	-2

BNE Without NOP's

PC	IFID_IR	IDEX_IR	EXMEM_IR	MEMWB_IR	WD
0	0000	xxxx	xxxx	xxxx	x
2	8100	0000	xxxx	xxxx	x
4	8202	8100	0000	xxxx	x
6	66c0	8202	8100	0000	0
8	b305	66c0	8202	8100	5
10	9102	b305	66c0	8202	7
12	9200	9102	b305	66c0	X
X	8100	9200	9102	b305	x
X	xxxx	8100	9200	9102	2
X	xxxx	xxxx	8100	9200	0
X	xxxx	xxxx	xxxx	8100	7
X	xxxx	xxxx	xxxx	xxxx	x
X	xxxx	xxxx	xxxx	xxxx	x
X	xxxx	xxxx	xxxx	xxxx	x
X	xxxx	xxxx	xxxx	xxxx	x
X	xxxx	xxxx	xxxx	xxxx	x
X	xxxx	xxxx	xxxx	xxxx	x
X	xxxx	xxxx	xxxx	xxxx	x