**CS 385 – Computer Architecture – Final Report**

Chris Mazurski, Uka Chukwuogor, Tomasz Klejmont

**OVERVIEW**

**Description:**

Our CPU is implemented using a 16-bit architecture and a 5-stage pipeline process to aid with efficiency and hazard protection. It implements the following R-type instructions: add, sub, and, or, and slt, as well as the following I-type instructions: addi, lw, sw, beq, and bne.

**Instruction Set:**

|  |  |  |
| --- | --- | --- |
| Instruction | Opcode | Type |
| add | 0000 | R-type |
| sub | 0001 | R-type |
| and | 0010 | R-type |
| or | 0011 | R-type |
| addi | 0100 | I-type |
| lw | 0101 | I-type |
| sw | 0110 | I-type |
| slt | 0111 | R-type |
| beq | 1000 | I-type |
| bne | 1001 | I-type |

**Instruction Format:**

R-format:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| op | rs | rt | rd | unused |
| 4 | 2 | 2 | 2 | 6 |

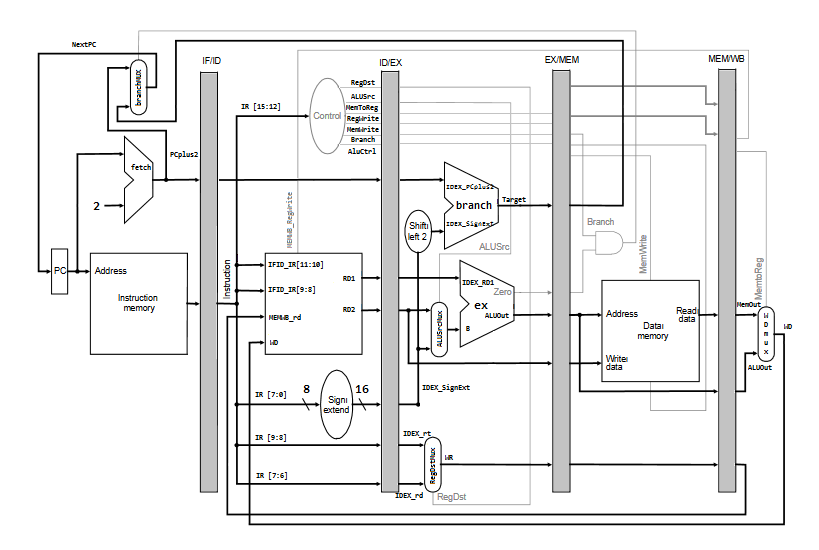
I-format:

|  |  |  |  |
| --- | --- | --- | --- |
| op | rs | rt | address/value |
| 4 | 2 | 2 | 8 |

**Components:**

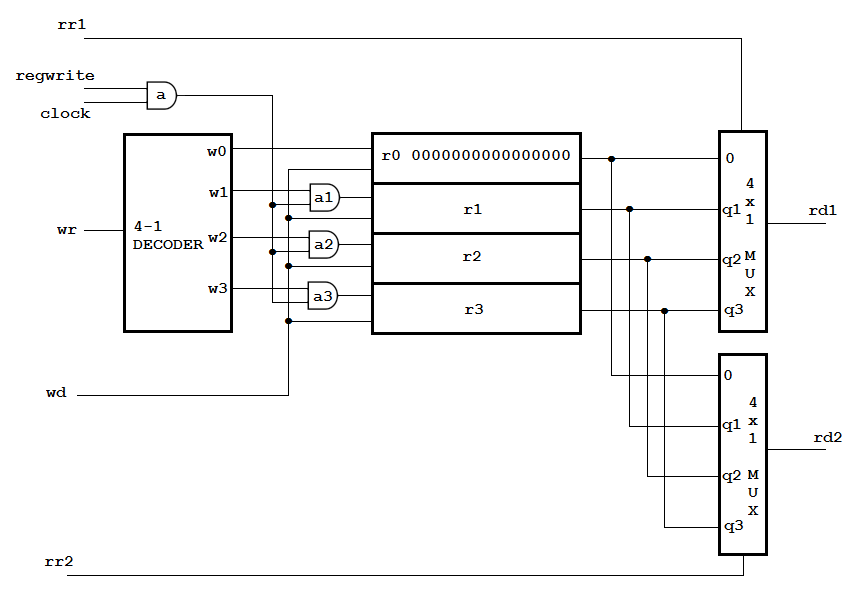
**Main Datapath:**

Follows a pipe-lined datapath.

****

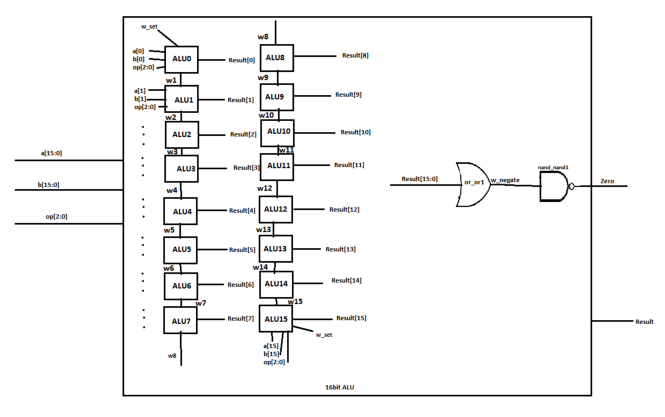
**Register File:**

Used for reading stored data and/or writing new data between cycles.

****

**ALU:**

An **arithmetic-logic unit (ALU)** is the part of a computer processor that carries out arithmetic and logic operations on the operands in computer instruction words.

****

**SOURCE CODE**

//CS385 Computer Architecture

//Chris,Tom,Uka

// ----------- REGISTER FILE ---------------

/\* INPUTS: rr1 - Read Register 1

\*\* rr2 - Read Register 2

\*\* wr - Write Register

\*\* wd - Write Data

\*\* clock - Clock

\*\* regwrite - (1) write data / (0) don't write data

\*\* OUTPUTS: rd1 - Read Data 1

\*\* rd2 - Read Data 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**;**

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

**wire** regwrite\_and\_clock**,** clk0**,**clk1**,**clk2**,**clk3**;**

// registers

// register r0 is the $0 register

register\_16bit r1 **(**wd**,**clk1**,**q1**);**

register\_16bit r2 **(**wd**,**clk2**,**q2**);**

register\_16bit r3 **(**wd**,**clk3**,**q3**);**

// output port

mux4x1\_16bit mux1 **(**16'b0**,**q1**,**q2**,**q3**,**rr1**,**rd1**);**

mux4x1\_16bit mux2 **(**16'b0**,**q1**,**q2**,**q3**,**rr2**,**rd2**);**

// input port

decoder dec**(**wr**[**1**],**wr**[**0**],**w3**,**w2**,**w1**,**w0**);**

**and** a **(**regwrite\_and\_clock**,**regwrite**,**clock**);**

**and** a1 **(**clk1**,**regwrite\_and\_clock**,**w1**),**

a2 **(**clk2**,**regwrite\_and\_clock**,**w2**),**

a3 **(**clk3**,**regwrite\_and\_clock**,**w3**);**

**endmodule**

//hierarchical design of a 16-bit register constructed from 16 D-type flip-flops

**module** register\_16bit**(**D**,**CLK**,**Q**);**

**input** **[**15**:**0**]** D**;**

**input** CLK**;**

**output** **[**15**:**0**]** Q**;**

D\_flip\_flop d1 **(**D**[**0**],** CLK**,** Q**[**0**]);**

D\_flip\_flop d2 **(**D**[**1**],** CLK**,** Q**[**1**]);**

D\_flip\_flop d3 **(**D**[**2**],** CLK**,** Q**[**2**]);**

D\_flip\_flop d4 **(**D**[**3**],** CLK**,** Q**[**3**]);**

D\_flip\_flop d5 **(**D**[**4**],** CLK**,** Q**[**4**]);**

D\_flip\_flop d6 **(**D**[**5**],** CLK**,** Q**[**5**]);**

D\_flip\_flop d7 **(**D**[**6**],** CLK**,** Q**[**6**]);**

D\_flip\_flop d8 **(**D**[**7**],** CLK**,** Q**[**7**]);**

D\_flip\_flop d9 **(**D**[**8**],** CLK**,** Q**[**8**]);**

D\_flip\_flop d10 **(**D**[**9**],** CLK**,** Q**[**9**]);**

D\_flip\_flop d11 **(**D**[**10**],** CLK**,** Q**[**10**]);**

D\_flip\_flop d12 **(**D**[**11**],** CLK**,** Q**[**11**]);**

D\_flip\_flop d13 **(**D**[**12**],** CLK**,** Q**[**12**]);**

D\_flip\_flop d14 **(**D**[**13**],** CLK**,** Q**[**13**]);**

D\_flip\_flop d15 **(**D**[**14**],** CLK**,** Q**[**14**]);**

D\_flip\_flop d16 **(**D**[**15**],** CLK**,** Q**[**15**]);**

**endmodule**

/\* hierarchical design of a D flip-flop constructed from 2 D-type latches \*/

**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**

/\* gate level design of a D latch \*/

**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**

/\* hierarchical design of a 16-bit 4x1 multiplexer constructed from 16 1-bit 4x1 multiplexers \*/

**module** mux4x1\_16bit**(**i0**,**i1**,**i2**,**i3**,**select**,**y**);**

**input** **[**15**:**0**]** i0**,**i1**,**i2**,**i3**;**

**input** **[**1**:**0**]** select**;**

**output** **[**15**:**0**]** y**;**

mux4x1 m1 **(**i0**[**0**],** i1**[**0**],** i2**[**0**],** i3**[**0**],** select**,** y**[**0**]);**

mux4x1 m2 **(**i0**[**1**],** i1**[**1**],** i2**[**1**],** i3**[**1**],** select**,** y**[**1**]);**

mux4x1 m3 **(**i0**[**2**],** i1**[**2**],** i2**[**2**],** i3**[**2**],** select**,** y**[**2**]);**

mux4x1 m4 **(**i0**[**3**],** i1**[**3**],** i2**[**3**],** i3**[**3**],** select**,** y**[**3**]);**

mux4x1 m5 **(**i0**[**4**],** i1**[**4**],** i2**[**4**],** i3**[**4**],** select**,** y**[**4**]);**

mux4x1 m6 **(**i0**[**5**],** i1**[**5**],** i2**[**5**],** i3**[**5**],** select**,** y**[**5**]);**

mux4x1 m7 **(**i0**[**6**],** i1**[**6**],** i2**[**6**],** i3**[**6**],** select**,** y**[**6**]);**

mux4x1 m8 **(**i0**[**7**],** i1**[**7**],** i2**[**7**],** i3**[**7**],** select**,** y**[**7**]);**

mux4x1 m9 **(**i0**[**8**],** i1**[**8**],** i2**[**8**],** i3**[**8**],** select**,** y**[**8**]);**

mux4x1 m10 **(**i0**[**9**],** i1**[**9**],** i2**[**9**],** i3**[**9**],** select**,** y**[**9**]);**

mux4x1 m11 **(**i0**[**10**],** i1**[**10**],** i2**[**10**],** i3**[**10**],** select**,** y**[**10**]);**

mux4x1 m12 **(**i0**[**11**],** i1**[**11**],** i2**[**11**],** i3**[**11**],** select**,** y**[**11**]);**

mux4x1 m13 **(**i0**[**12**],** i1**[**12**],** i2**[**12**],** i3**[**12**],** select**,** y**[**12**]);**

mux4x1 m14 **(**i0**[**13**],** i1**[**13**],** i2**[**13**],** i3**[**13**],** select**,** y**[**13**]);**

mux4x1 m15 **(**i0**[**14**],** i1**[**14**],** i2**[**14**],** i3**[**14**],** select**,** y**[**14**]);**

mux4x1 m16 **(**i0**[**15**],** i1**[**15**],** i2**[**15**],** i3**[**15**],** select**,** y**[**15**]);**

**endmodule**

/\* hierarchical design of a 4x1 multiplexer constructed from 3 2x1 multiplexers \*/

**module** mux4x1**(**i0**,**i1**,**i2**,**i3**,**select**,**y**);**

**input** i0**,**i1**,**i2**,**i3**;**

**input** **[**1**:**0**]** select**;**

**output** y**;**

**wire** out1**,**out2**;**

mux2x1 m1 **(**i0**,** i1**,** select**[**0**],** out1**);**

mux2x1 m2 **(**i2**,** i3**,** select**[**0**],** out2**);**

mux2x1 m3 **(**out1**,** out2**,** select**[**1**],** y**);**

**endmodule**

/\* gate level design of a 2x1 multiplexer \*/

**module** mux2x1**(**a**,**b**,**select**,**out**);**

**input** a**,**b**,**select**;**

**output** out**;**

**wire** not\_select**,**i1**,**i2**;**

**not** G1 **(**not\_select**,** select**);**

**and** G2 **(**i1**,** a**,** not\_select**),**

G3 **(**i2**,** b**,** select**);**

**or** G4 **(**out**,** i1**,** i2**);**

**endmodule**

//Hierarchical design of a 2bit 2x1 multiplexor

**module** mux2x1\_2**(**a**,**b**,**select**,**out**);**

**input** **[**1**:**0**]** a**,**b**;**

**input** select**;**

**output** **[**1**:**0**]** out**;**

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

mux2x1 m2 **(**a**[**1**],**b**[**1**],**select**,**out**[**1**]);**

**endmodule**

//Hierachical 16bit 2x1 multiplexor from 16 1 bit multiplexors

**module** mux2x1\_16**(**a**,**b**,**select**,**out**);**

**input** **[**15**:**0**]** a**,**b**;**

**input** select**;**

**output** **[**15**:**0**]** out**;**

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

mux2x1 m2 **(**a**[**1**],**b**[**1**],**select**,**out**[**1**]);**

mux2x1 m3 **(**a**[**2**],**b**[**2**],**select**,**out**[**2**]);**

mux2x1 m4 **(**a**[**3**],**b**[**3**],**select**,**out**[**3**]);**

mux2x1 m5 **(**a**[**4**],**b**[**4**],**select**,**out**[**4**]);**

mux2x1 m6 **(**a**[**5**],**b**[**5**],**select**,**out**[**5**]);**

mux2x1 m7 **(**a**[**6**],**b**[**6**],**select**,**out**[**6**]);**

mux2x1 m8 **(**a**[**7**],**b**[**7**],**select**,**out**[**7**]);**

mux2x1 m9 **(**a**[**8**],**b**[**8**],**select**,**out**[**8**]);**

mux2x1 m10 **(**a**[**9**],**b**[**9**],**select**,**out**[**9**]);**

mux2x1 m11 **(**a**[**10**],**b**[**10**],**select**,**out**[**10**]);**

mux2x1 m12 **(**a**[**11**],**b**[**11**],**select**,**out**[**11**]);**

mux2x1 m13 **(**a**[**12**],**b**[**12**],**select**,**out**[**12**]);**

mux2x1 m14 **(**a**[**13**],**b**[**13**],**select**,**out**[**13**]);**

mux2x1 m15 **(**a**[**14**],**b**[**14**],**select**,**out**[**14**]);**

mux2x1 m16 **(**a**[**15**],**b**[**15**],**select**,**out**[**15**]);**

**endmodule**

/\* gate level design of a 4 to 1 decoder \*/

**module** decoder **(**S1**,**S0**,**D3**,**D2**,**D1**,**D0**);**

**input** S0**,**S1**;**

**output** D0**,**D1**,**D2**,**D3**;**

**not** n1 **(**notS0**,**S0**),**

n2 **(**notS1**,**S1**);**

**and** a0 **(**D0**,**notS1**,**notS0**),**

a1 **(**D1**,**notS1**,** S0**),**

a2 **(**D2**,** S1**,**notS0**),**

a3 **(**D3**,** S1**,** S0**);**

**endmodule**

//Gate level Full Adder for the ALU

**module** my\_full\_adder**(**A**,**B**,**Ci**,**S**,**Co**);**

**input** A**,**B**,**Ci**;**

**output** S**,**Co**;**

**wire** x**,**y**,**z**,**a**,**b**,**d**,**e**,**f**,**g**,**h**,**i**,**j**,**k**;**

//First lets do the S output

**nand** g1 **(**x**,**B**,**B**);**

**nand** g2 **(**y**,**Ci**,**Ci**);**

**and** g3 **(**z**,**A**,**x**,**y**);**

**nand** g15**(**k**,**A**,**A**);**

**nand** g4 **(**a**,**B**,**B**);**

**and** g5 **(**b**,**k**,**a**,**Ci**);**

**and** g6 **(**d**,**A**,**B**,**Ci**);**

**nand** g7**(**e**,**A**,**A**);**

**nand** g8**(**f**,**Ci**,**Ci**);**

**and** g9**(**g**,**e**,**B**,**f**);**

**xor** g10 **(**S**,**z**,**b**,**d**,**g**);**

//Lets do the Co output now

**and** g11**(**h**,**A**,**B**);**

**and** g12**(**i**,**A**,**Ci**);**

**and** g13**(**j**,**B**,**Ci**);**

**or** g14**(**Co**,**h**,**i**,**j**);**

**endmodule**

//gate level 4x1 Multiplexor for the ALU

**module** my\_4multiplexer**(**out**,**in0**,**in1**,**in2**,**in3**,**s0**,**s1**);**

**output** out**;**

**input** in0**,**in1**,**in2**,**in3**;**

**input** s0**,**s1**;**

**wire** inv0**,**inv1**;**

**wire** a0**,**a1**,**a2**,**a3**;**

**not** not\_0 **(**inv0**,**s0**);**

**not** not\_1 **(**inv1**,**s1**);**

**and** and\_0 **(**a0**,**in0**,**inv0**,**inv1**);**

**and** and\_1 **(**a1**,**in1**,**inv0**,**s1**);**

**and** and\_2 **(**a2**,**in2**,**s0**,**inv1**);**

**and** and\_3 **(**a3**,**in3**,**s0**,**s1**);**

**or** or\_0 **(**out**,**a0**,**a1**,**a2**,**a3**);**

**endmodule**

//gate level 2x1 multiplexor for the ALU

**module** my\_2multiplexer**(**A**,**B**,**C**,**F**);**

**input** A**,**B**,**C**;**

**output** F**;**

**wire** x**,**y**,**z**;**

**and** g1 **(**x**,**B**,**C**);**

**nand** g2 **(**y**,**C**,**C**);**

**and** g3 **(**z**,**A**,**y**);**

**or** g4 **(**F**,**z**,**x**);**

**endmodule**

//1bit ALU composed of 2x1/4x1 multiplexors and a full adder

**module** my\_1bitALU**(**Result**,**carryOut**,**a**,**b**,**Less**,**Binvert**,**carryIn**,**operation1**,**operation2**);**

**input** a**,**b**,**Less**,**Binvert**,**carryIn**,**operation1**,**operation2**;**

**output** Result**,**carryOut**;**

**wire** w1**,**w2**,**w3**,**w4**,**w5**;**

**nand** g1 **(**w1**,**b**,**b**);**

my\_2multiplexer M2**(**b**,**w1**,**Binvert**,**w2**);**

**and** g2 **(**w3**,**a**,**w2**);**

**or** g3 **(**w4**,**a**,**w2**);**

my\_full\_adder FA1**(**a**,**w2**,**carryIn**,**w5**,**carryOut**);**

my\_4multiplexer M4**(**Result**,**w3**,**w4**,**w5**,**Less**,**operation1**,**operation2**);**

**endmodule**

//1bit ALU for the least significant bit

**module** my\_1bitALU2**(**Result**,**overflow**,**set**,**a**,**b**,**Less**,**Binvert**,**carryIn**,**operation1**,**operation2**);**

**input** a**,**b**,**Less**,**Binvert**,**carryIn**,**operation1**,**operation2**;**

**output** Result**,**overflow**,**set**;**

**wire** w1**,**w2**,**w3**,**w4**,**w5**;**

**nand** g1 **(**w1**,**b**,**b**);**

my\_2multiplexer M2**(**b**,**w1**,**Binvert**,**w2**);**

**and** g2 **(**w3**,**a**,**w2**);**

**or** g3 **(**w4**,**a**,**w2**);**

my\_full\_adder FA1**(**a**,**w2**,**carryIn**,**w5**,**overflow**);**

my\_4multiplexer M4**(**Result**,**w3**,**w4**,**w5**,**Less**,**operation1**,**operation2**);**

**and** g4 **(**set**,**w5**,**w5**);**

**endmodule**

//16bit ALU composed of 16 1-bit ALU's, returns a result given A,B, and the ALUop

**module** my\_16bitALU**(**op**,** a**,** b**,** result**,** zero**,** less**);**

**input** **[**15**:**0**]** a**,**b**;**

**input** **[**2**:**0**]** op**;**

**input** less**,**carryIn**;**

//output overflow;

**output** **[**15**:**0**]** result**;**

**output** zero**;**

**wire** w1**,**w2**,**w3**,**w4**,**w5**,**w6**,**w7**,**w8**,**w9**,**w10**,**w11**,**w12**,**w13**,**w14**,**w15**,**w\_set**,**w\_over**,**w\_negate**;**

//wires 15 and w\_over needed to xor for overflow

my\_1bitALU ALU0**(**result**[**0**],**w1**,**a**[**0**],**b**[**0**],**w\_set**,**op**[**2**],**op**[**2**],**op**[**1**],**op**[**0**]),**

ALU1**(**result**[**1**],**w2**,**a**[**1**],**b**[**1**],**less**,**op**[**2**],**w1**,**op**[**1**],**op**[**0**]),**

ALU2**(**result**[**2**],**w3**,**a**[**2**],**b**[**2**],**less**,**op**[**2**],**w2**,**op**[**1**],**op**[**0**]),**

ALU3**(**result**[**3**],**w4**,**a**[**3**],**b**[**3**],**less**,**op**[**2**],**w3**,**op**[**1**],**op**[**0**]),**

ALU4**(**result**[**4**],**w5**,**a**[**4**],**b**[**4**],**less**,**op**[**2**],**w4**,**op**[**1**],**op**[**0**]),**

ALU5**(**result**[**5**],**w6**,**a**[**5**],**b**[**5**],**less**,**op**[**2**],**w5**,**op**[**1**],**op**[**0**]),**

ALU6**(**result**[**6**],**w7**,**a**[**6**],**b**[**6**],**less**,**op**[**2**],**w6**,**op**[**1**],**op**[**0**]),**

ALU7**(**result**[**7**],**w8**,**a**[**7**],**b**[**7**],**less**,**op**[**2**],**w7**,**op**[**1**],**op**[**0**]),**

ALU8**(**result**[**8**],**w9**,**a**[**8**],**b**[**8**],**less**,**op**[**2**],**w8**,**op**[**1**],**op**[**0**]),**

ALU9**(**result**[**9**],**w10**,**a**[**9**],**b**[**9**],**less**,**op**[**2**],**w9**,**op**[**1**],**op**[**0**]),**

ALU10**(**result**[**10**],**w11**,**a**[**10**],**b**[**10**],**less**,**op**[**2**],**w10**,**op**[**1**],**op**[**0**]),**

ALU11**(**result**[**11**],**w12**,**a**[**11**],**b**[**11**],**less**,**op**[**2**],**w11**,**op**[**1**],**op**[**0**]),**

ALU12**(**result**[**12**],**w13**,**a**[**12**],**b**[**12**],**less**,**op**[**2**],**w12**,**op**[**1**],**op**[**0**]),**

ALU13**(**result**[**13**],**w14**,**a**[**13**],**b**[**13**],**less**,**op**[**2**],**w13**,**op**[**1**],**op**[**0**]),**

ALU14**(**result**[**14**],**w15**,**a**[**14**],**b**[**14**],**less**,**op**[**2**],**w14**,**op**[**1**],**op**[**0**]);**

my\_1bitALU2 ALU15**(**result**[**15**],**w\_over**,**w\_set**,**a**[**15**],**b**[**15**],**less**,**op**[**2**],**w15**,**op**[**1**],**op**[**0**]);**

//xor xor\_xor1(overflow,w15,w\_over);

**or** or\_or1 **(**w\_negate**,**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**],**result**[**15**]);**

**nand** nand\_nand1 **(**zero**,**w\_negate**,**w\_negate**);**

**endmodule**

/\*Combinational logic module to output the branch switch for use as an input in the branch multiplexor. (This is needed to sift through the 2 bit branch control and determine whether we need to perform BEQ or BNE or neither)\*/

//If Branch[1]==1 and Zero ==1, we are branching for a BEQ instruction

// else if Branch[0] ==1 and everything else ==0, we are branching for a BNE instruction

**module** branch\_switch**(**BranchSignal**,**Branch**,**Zero**);**

**input** **[**1**:**0**]** Branch**;**

**input** Zero**;**

**output** BranchSignal**;**

**wire** zero\_neg**,**branch\_neg**,**ne\_neg**,**beq**,**bne**;**

**nand** g1**(**zero\_neg**,**Zero**,**Zero**);**

**nand** g2 **(**branch\_neg**,**Branch**[**1**],**Branch**[**1**]);**

**nand** g3 **(**ne\_neg**,**Branch**[**0**],**Branch**[**0**]);**

**and** g4 **(**bne**,**zero\_neg**,**branch\_neg**,**Branch**[**0**]);**

**and** g5 **(**beq**,**Zero**,**Branch**[**1**],**ne\_neg**);**

**or** g6 **(**BranchSignal**,**beq**,**bne**);**

**endmodule**

//Behavioural implementation of the control unit. Generates an output of path enabling bits correlating to the opcode of the instruction.

**module** MainControl **(**Op**,**Control**);**

**input** **[**3**:**0**]** Op**;**

**output** **reg** **[**9**:**0**]** Control**;**

**always** **@(**Op**)** **case** **(**Op**)**

4'b0000**:** Control **<=** 10'b1001000010**;** //R-Type Arithmetic (add)

4'b0001**:** Control **<=** 10'b1001000110**;** //R-Type Arithmetic (sub)

4'b0010**:** Control **<=** 10'b1001000000**;** //R-Type Arithmetic (and)

4'b0011**:** Control **<=** 10'b1001000001**;** //R-Type Arithmetic (or)

4'b0111**:** Control **<=** 10'b1001000111**;** //R-Type Arithmetic (slt)

4'b0100**:** Control **<=** 10'b0101000010**;** //I-Type (addi)

4'b0101**:** Control **<=** 10'b0111000010**;** // Data Transfer (LW)

4'b0110**:** Control **<=** 10'b0100100010**;** // Data Transfer (SW)

4'b1000**:** Control **<=** 10'b0000010110**;** // Branch (BEQ)

4'b1001**:** Control **<=** 10'b0000001110**;** // Branch (BNE)

**endcase**

//From a 4 bit opcode we get 10 control bits in total: IDEX\_RegDst,ALUSrc,MemtoReg,RegWrite,MemWrite,Branch,Branch\_ne,ALUOp[2],ALUOp[1],ALUOp[0]

//Since we need BNE, we're going to need an extra bit for the branch control line. Branch\_ne == 1 if opcode calls for a BNE

//The main control unit now includes a 3bit ALUOp without the need for a seperate ALU controller. R-type function fields can be empty now since we'll be determining control bits purely from opcode

//memread unnecessary for now

**endmodule**

//CPU with the instruction memory/Data Memory implemented, and instantiates the fetch,execute,and branch ALU's to process through the instructions

**module** CPU **(**clock**,** PC**,** IFID\_IR**,** IDEX\_IR**,** EXMEM\_IR**,** MEMWB\_IR**,** WD**);**

**input** clock**;**

**reg** MEMWB\_RegWrite**,**MEMWB\_MemtoReg**;**

**reg** **[**1**:**0**]** MEMWB\_rd**;**

**output** **[**15**:**0**]** PC**,** IFID\_IR**,** IDEX\_IR**,** EXMEM\_IR**,** MEMWB\_IR**,** WD**;**

**initial** **begin**

IMemory**[**0**]** **=** 16'b0101000100000000**;** // lw $1, 0($0)

IMemory**[**1**]** **=** 16'b0101001000000010**;** // lw $2, 2($0)

IMemory**[**2**]** **=** 16'b0000000000000000**;** // nop

IMemory**[**3**]** **=** 16'b0000000000000000**;** // nop

IMemory**[**4**]** **=** 16'b0000000000000000**;** // nop

IMemory**[**5**]** **=** 16'b0111100111000000**;** // slt $3, $2, $1

IMemory**[**6**]** **=** 16'b0000000000000000**;** // nop

IMemory**[**7**]** **=** 16'b0000000000000000**;** // nop

IMemory**[**8**]** **=** 16'b0000000000000000**;** // nop

IMemory**[**9**]** **=** 16'b1001110000001000**;** // bne $3, $0, 10

IMemory**[**10**]** **=** 16'b0000000000000000**;** // nop

IMemory**[**11**]** **=** 16'b0000000000000000**;** // nop

IMemory**[**12**]** **=** 16'b0000000000000000**;** // nop

IMemory**[**13**]** **=** 16'b0110000100000010**;** // sw $1, 2($0)

IMemory**[**14**]** **=** 16'b0000000000000000**;** // nop

IMemory**[**15**]** **=** 16'b0000000000000000**;** // nop

IMemory**[**16**]** **=** 16'b0000000000000000**;** // nop

IMemory**[**17**]** **=** 16'b0110001000000000**;** // sw $2, 0($0)

IMemory**[**18**]** **=** 16'b0000000000000000**;** // nop

IMemory**[**19**]** **=** 16'b0000000000000000**;** // nop

IMemory**[**20**]** **=** 16'b0000000000000000**;** // nop

IMemory**[**21**]** **=** 16'b0101000100000000**;** // lw $1, 0($0)

IMemory**[**22**]** **=** 16'b0000000000000000**;** // nop

IMemory**[**23**]** **=** 16'b0000000000000000**;** // nop

IMemory**[**24**]** **=** 16'b0000000000000000**;** // nop

IMemory**[**25**]** **=** 16'b0101001000000010**;** // lw $2, 2($0)

IMemory**[**26**]** **=** 16'b0000000000000000**;** // nop

IMemory**[**27**]** **=** 16'b0000000000000000**;** // nop

IMemory**[**28**]** **=** 16'b0000000000000000**;** // nop

IMemory**[**29**]** **=** 16'b0001011001000000**;** // sub $1, $1, $2

IMemory**[**30**]** **=** 16'b0000000000000000**;** // nop

IMemory**[**31**]** **=** 16'b0000000000000000**;** // nop

IMemory**[**32**]** **=** 16'b0000000000000000**;** // nop

IMemory**[**33**]** **=** 16'b0100010100000110**;** //addi $1,$1, 6

DMemory **[**0**]** **=** 32'h9**;**

DMemory **[**1**]** **=** 32'h5**;**

**end**

// PIPELINE STAGES

// IF

**wire** **[**15**:**0**]** NextPC**,** PCplus2**;**

**reg[**15**:**0**]** PC**,** IMemory**[**0**:**1023**],** IFID\_IR**,** IFID\_PCplus2**,** EXMEM\_Target**;**

**reg** **[**1**:**0**]** EXMEM\_Branch**;**

**reg** EXMEM\_Zero**;**

my\_16bitALU fetch **(**3'b010**,**PC**,**16'b10**,**PCplus2**,**Unused**,**1'b0**);**

branch\_switch branchCtr **(**BranchSignal**,**EXMEM\_Branch**,** EXMEM\_Zero**);**

mux2x1\_16 branchMUX **(**PCplus2**,** EXMEM\_Target**,** BranchSignal**,** NextPC**);**

// ID

**wire** **[**9**:**0**]** Control**;**

**reg** IDEX\_RegWrite**,**IDEX\_MemtoReg**,**

IDEX\_MemWrite**,** IDEX\_ALUSrc**,** IDEX\_RegDst**;**

**reg** **[**2**:**0**]** IDEX\_AluCtrl**;**

**reg** **[**1**:**0**]** IDEX\_Branch**;**

**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\_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\_MemWrite**;**

**wire** **[**15**:**0**]** Target**;**

**reg** **[**15**:**0**]** EXMEM\_ALUOut**,**EXMEM\_RD2**;**

**reg** **[**15**:**0**]** EXMEM\_IR**;** // For monitoring the pipeline

**reg** **[**1**:**0**]** EXMEM\_rd**;**

**wire** **[**15**:**0**]** B**,**ALUOut**;**

**wire** **[**1**:**0**]** WR**;**

my\_16bitALU branch **(**3'b010**,**IDEX\_SignExt**<<**1**,**IDEX\_PCplus2**,**Target**,**Unused2**,**1'b0**);**

my\_16bitALU ex **(**IDEX\_AluCtrl**,** IDEX\_RD1**,** B**,** ALUOut**,** Zero**,** 1'b0**);**

mux2x1\_16 ALUSrcMux **(**IDEX\_RD2**,** IDEX\_SignExt**,** IDEX\_ALUSrc**,** B**);**

mux2x1\_2 RegDstMux **(**IDEX\_rt**,** IDEX\_rd**,** IDEX\_RegDst**,** WR**);**

// MEM

**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**];**

**always** **@(negedge** clock**)** **if** **(**EXMEM\_MemWrite**)** DMemory**[**EXMEM\_ALUOut**>>**1**]** **<=** EXMEM\_RD2**;**

// WB

mux2x1\_16 WDmux **(**MEMWB\_ALUOut**,** MEMWB\_MemOut**,** MEMWB\_MemtoReg**,** WD**);** // assign WD

**initial** **begin**

PC **=** 0**;**

// Initialize pipeline registers

IDEX\_RegWrite**=**0**;**IDEX\_MemtoReg**=**0**;**IDEX\_Branch**=**0**;**IDEX\_MemWrite**=**0**;**IDEX\_ALUSrc**=**0**;**IDEX\_RegDst**=**0**;**IDEX\_AluCtrl**=**0**;**

IFID\_IR**=**0**;**

EXMEM\_RegWrite**=**0**;**EXMEM\_MemtoReg**=**0**;**EXMEM\_Branch**=**0**;**EXMEM\_MemWrite**=**0**;**

EXMEM\_Target**=**0**;**

MEMWB\_RegWrite**=**0**;**MEMWB\_MemtoReg**=**0**;**

**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

**{**IDEX\_RegDst**,**IDEX\_ALUSrc**,**IDEX\_MemtoReg**,**IDEX\_RegWrite**,**IDEX\_MemWrite**,**IDEX\_Branch**,**IDEX\_AluCtrl**}** **<=** 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**;**

EXMEM\_Branch **<=** IDEX\_Branch**;**

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**

//testing module that displays output

**module** test **();**

**reg** clock**;**

**wire** **[**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 **(**"Time Clock PC IFID\_IR IDEX\_IR EXMEM\_IR MEMWB\_IR WD"**);**

$monitor **(**"%2d %b %d %b %b %b %b %d"**,** $time**,**clock**,**PC**,**IFID\_IR**,**IDEX\_IR**,**EXMEM\_IR**,**MEMWB\_IR**,**WD**);**

clock **=** 1**;**

**#**74 $finish**;**

**end**

**endmodule**

**TEST PROGRAM**

**Assembly Code:**

lw $1, 0($0) //Takes the value in memory address $0+0 (Data Memory[0]) and stores it into $1

lw $2, 2($0) //Takes the value in memory address $0+2 (DataMemory[1]) and stores it into $2

slt $3, $2, $1 //Will store 1 in $3 if the value in $2 is less than the value in $1, otherwise stores 0

bne $3, $0, 2 //Takes the immediate value 2, left shifts it by 2, and adds it to the program counter if $3 is not equal to $0

sw $1, 2($0) //Takes the value in $1 and stores it into data memory located in $0+2

sw $2, 0($0) //Takes the value in $2 and stores it into data memory located in $0+0

lw $1, 0($0) //Takes the value in memory address $0+0 (Data Memory [0]) and stores it into $1

lw $2, 2($0) //Takes the value in memory address $0+2 (Data Memory[1]) and stores it into $2

sub $1, $1, $2 //Subtracts the value in $1 with $2, and stores into $1

addi $1,$1,6 //Adds the immediate value 6 to register $1 and stores it into $1

**Machine Code:**

IMemory[0] = 16'b0101000100000000; // lw $1, 0($0)

IMemory[1] = 16'b0101001000000010; // lw $2, 2($0)

IMemory[2] = 16'b0000000000000000; // nop

IMemory[3] = 16'b0000000000000000; // nop

IMemory[4] = 16'b0000000000000000; // nop

IMemory[5] = 16'b0111100111000000; // slt $3, $2, $1

IMemory[6] = 16'b0000000000000000; // nop

IMemory[7] = 16'b0000000000000000; // nop

IMemory[8] = 16'b0000000000000000; // nop

IMemory[9] = 16'b1001110000000010; // bne $3, $0, 2

IMemory[10] = 16'b0000000000000000; // nop

IMemory[11] = 16'b0000000000000000; // nop

IMemory[12] = 16'b0000000000000000; // nop

IMemory[13] = 16'b0110000100000010; //sw $1, 2($0)

IMemory[14] = 16'b0000000000000000; // nop

IMemory[15] = 16'b0000000000000000; // nop

IMemory[16] = 16'b0000000000000000; // nop

IMemory[17] = 16'b0110001000000000; //sw $2, 0($0)

IMemory[18] = 16'b0000000000000000; // nop

IMemory[19] = 16'b0000000000000000; // nop

IMemory[20] = 16'b0000000000000000; // nop

IMemory[21] = 16'b0101000100000000; // lw $1, 0($0)

IMemory[22] = 16'b0000000000000000; // nop

IMemory[23] = 16'b0000000000000000; // nop

IMemory[24] = 16'b0000000000000000; // nop

IMemory[25] = 16'b0101001000000010; // lw $2, 2($0)

IMemory[26] = 16'b0000000000000000; // nop

IMemory[27] = 16'b0000000000000000; // nop

IMemory[28] = 16'b0000000000000000; // nop

IMemory[29] = 16'b0001011001000000; // sub $1, $1, $2

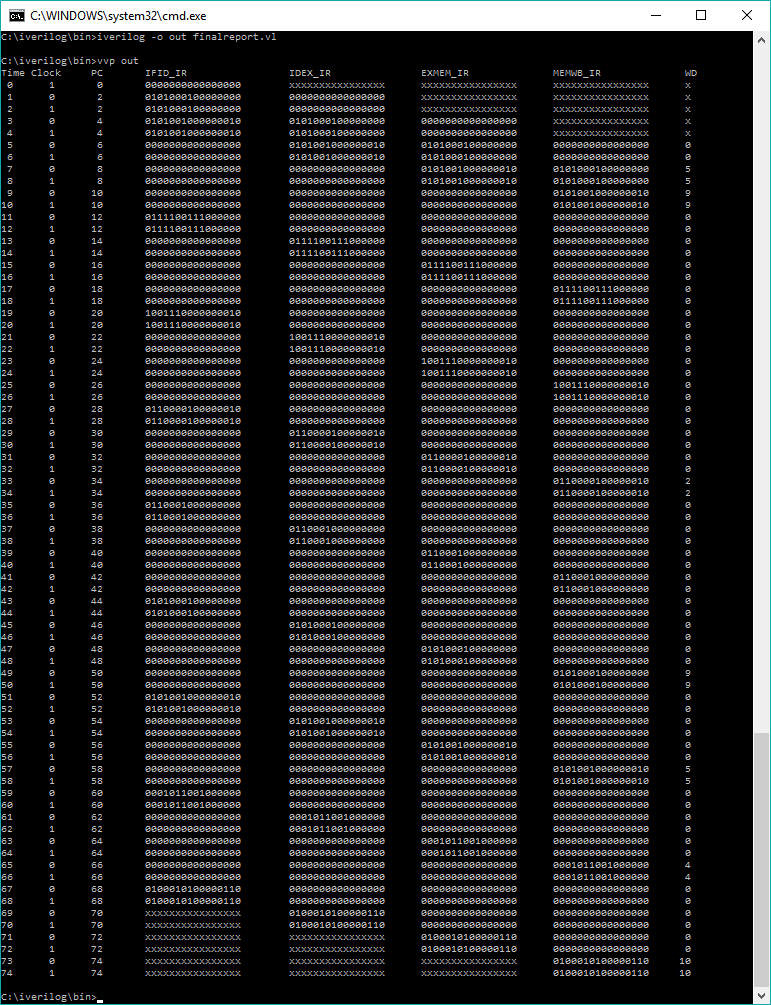
IMemory[30] = 16'b0000000000000000; // nop

IMemory[31] = 16'b0000000000000000; // nop

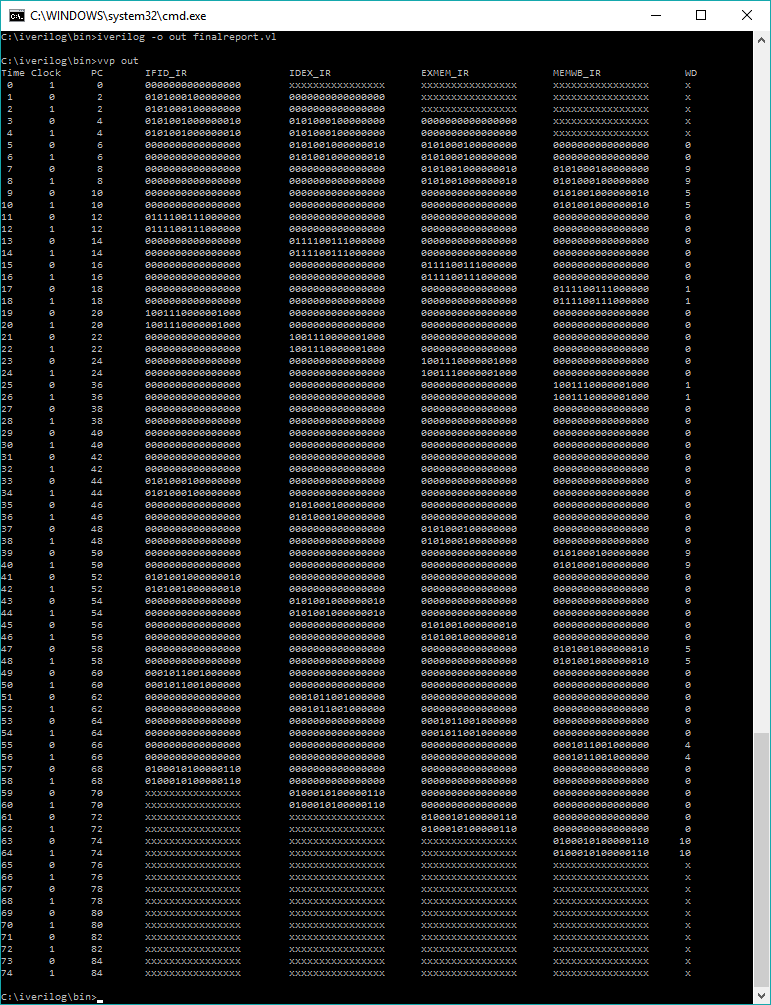
IMemory[32] = 16'b0000000000000000; // nop

IMemory[33] = 16'b0100010100000110; //addi $1,$1, 6

**Output: Branch Not Taken ($3= = $0):**



**Branch Taken ($3 = = $0):**



**Output: NOPS removed (To Demonstrate Impact of Hazards):**

