Noah Niedzwiecki

ELEC 5200

PROJECT 3

March 3, 2019

The goal of this project is to build our ISA hardware with description language models. For the rest of the project the system will be described using VHDL syntax and semantics. An index is on the following page to help navigate the code. A data path reference can be located on the last page.

The following hardware is used:

- Two Sign Extension Units
- Dedicated Address Addition ALU
- Main ALU
- Register File
- Instruction Memory and Data Memory
- ALU / Data Memory Mux
- Program Counter
- Main Control Unit
- Address Mux

In the previous project report, each of the 16 instructions was explained in detail. Now, we will look at the datapath for each of these pieces of hardware during the instruction executions.

Changes that were done since last assignment:

- Calls are now called branches
- File tables have been combined into one unit
- ALU output is not demuxed, simply connected to memory and ALU/DataMemory Mux

---

Index:

Control Code 3 – 7

Control Test Bench 7 – 22

ALU Code 23- 27

PC ADDER 28 – 31

Register File 32 – 36

Data Memory 36 -37

Program Counter 38

ALU/ DataMemory Mux 39

Sign Extension1 40

Sign Extension2 41

Control Code:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;
use ieee.std_logic_unsigned.all;

entity CONTROL is
   port(
      instruction : in STD_LOGIC_VECTOR(15 downto 0);
      branchBit : in STD_LOGIC;
      regMuxCtrl : out STD_LOGIC;
      regTableA : out STD_LOGIC_VECTOR(3 downto 0);
      regTableB : out STD_LOGIC_VECTOR(3 downto 0);
      regTableD : out STD_LOGIC_VECTOR(3 downto 0);
      ALUout : out STD_LOGIC_VECTOR(3 downto 0);
      regTable : out STD_LOGIC_VECTOR(2 downto 0);

      signCtrl: out STD_LOGIC;
      dataMem : out STD_LOGIC_VECTOR(1 downto 0);
      addCtrl: out STD_LOGIC_VECTOR(1 downto 0));


end CONTROL;

architecture Behavioral of CONTROL is
signal regTableAsig : STD_LOGIC_VECTOR(3 downto 0) := "0000";
signal regTableBsig : STD_LOGIC_VECTOR(3 downto 0) := "0000";
signal regTableDsig : STD_LOGIC_VECTOR(3 downto 0) := "0000";
signal ALUOutsig : STD_LOGIC_VECTOR(3 downto 0) := "0000";
signal regTablesig : STD_LOGIC_VECTOR(2 downto 0) := "000";
signal signCtrlsig: STD_LOGIC := '0';
signal regMuxCtrlsig: STD_LOGIC := '0';
signal dataMemsig: STD_LOGIC_VECTOR(1 downto 0) := "00";
signal addCtrlsig: STD_LOGIC_VECTOR(1 downto 0) := "00";

begin

process(instruction) begin
   if(instruction(15 downto 12) = "0000") then --halt
      addCtrlsig <= "11";
   elsif(instruction(15 downto 12) = "0001")then --jump
      signCtrlsig <= '1';
      addCtrlsig <= "01"; --sign extended

   elsif(instruction(15 downto 12) = "0010")then --add
```

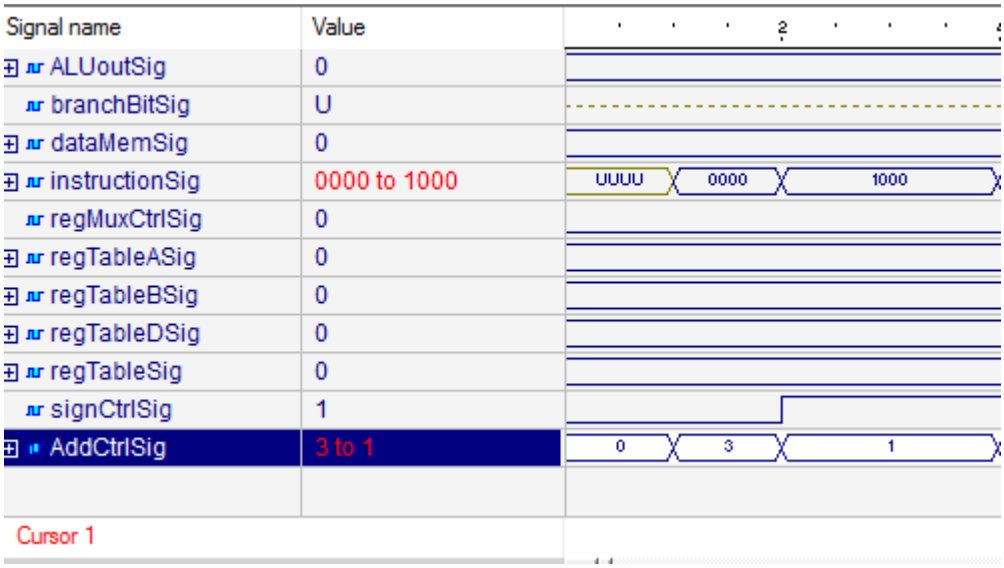
```
    regTablesig <= "000";
    regTableDsig <= instruction(11 downto 8);
    regTableAsig <= instruction(7 downto 4);
    regTableBsig <= instruction(3 downto 0);
    ALUOutsig <= "0010";
    dataMemsig <= "11"; -- do nothing
    addCtrlsig <= "00"; --PC + 1
    regMuxCtrlsig <= '0'; --ALU TO REG

elsif(instruction(15 downto 12) = "0011")then --sub
    regTablesig <= "000";
    regTableDsig <= instruction(11 downto 8);
    regTableAsig <= instruction(7 downto 4);
    regTableBsig <= instruction(3 downto 0);
    ALUOutsig <= "0011";
    dataMemsig <= "11"; --do nothing
    addCtrlsig <= "00"; --PC + 1
    regMuxCtrlsig <= '0'; --ALU TO REG

elsif(instruction(15 downto 12) = "0100")then --load
    regTablesig <= "000"; --read so regtable looks at input
    regTableDsig <= instruction(11 downto 8); --load address
    ALUOutsig <= "0000"; --do nothing
    dataMemsig <= "10"; --write pcIn address
    addCtrlsig <= "00"; --PC + 1
    regMuxCtrlsig <= '1'; --MEM TO REG

elsif(instruction(15 downto 12) = "0101")then --load C
    regTablesig <= "001"; --read sign extend 2
    regTableDsig <= instruction(11 downto 8); --load address
    ALUOutsig <= "0000"; --do nothing
    dataMemsig <= "11"; --do nothing
    addCtrlsig <= "00"; --PC + 1

elsif(instruction(15 downto 12) = "0110")then --MOV
    regTablesig <= "000"; --rtype
    regTableDsig <= instruction(11 downto 8); --destination
    regTableAsig <= instruction(7 downto 4); --value being moved
    ALUOutsig <= "0110"; --pass values from reg to reg mux
    dataMemsig <= "11"; --default
    addCtrlsig <= "00"; --PC + 1
    regMuxCtrlsig <= '0'; --ALU TO REG

elsif(instruction(15 downto 12) = "0111")then --STORE
    regTablesig <= "000"; --rtype
    regTableAsig <= instruction(7 downto 4); --value being stored
```

```
    ALUOutsig <= "0110"; --pass values from reg to reg mux
    dataMemsig <= "01"; --write to mem
    addCtrlsig <= "00"; --PC + 1
    regMuxCtrlsig <= '0'; --ALU TO REG

elsif(instruction(15 downto 12) = "1000")then --JUMP AND LINK
    regTablesig <= "010"; --save PC
    regTableAsig <= instruction(7 downto 4); --value being stored
    ALUOutsig <= "0000"; --pass values from reg to reg mux
    dataMemsig <= "11"; --do nothing
    signCtrlsig <= '0'; --put 8 bit offset to add
    addCtrlsig <= "01"; --sign extended
    --regMuxCtrlsig <= '0'; --ALU TO REG

elsif(instruction(15 downto 12) = "1001")then --RETURN
    regTablesig <= "000"; --do nothing
    dataMemsig <= "11"; --do nothing
    addCtrlsig <= "10"; -- add puts the register into PC
    ALUOutsig <= "0000"; --do nothing
    dataMemsig <= "11"; --do nothing
    regMuxCtrlsig <= '0'; --ALU TO REG

 elsif(instruction(15 downto 12) = "1010")then --BRANCH IF EQUAL
    regTablesig <= "000"; --do nothing
    dataMemsig <= "11"; --do nothing
    ALUOutsig <= "1010"; --branch if equal
    dataMemsig <= "11"; --do nothing
    regMuxCtrlsig <= '0'; --ALU TO REG
    signCtrlsig <= '0'; --put 8 bit offset to add
    if(branchBit = '1')then
       addCtrlsig <= "01"; -- add puts the sign extend into PC
    else
       addCtrlsig <= "00"; -- PC + 1
    end if;

elsif(instruction(15 downto 12) = "1011")then --BRANCH IF NOT EQUAL
    regTablesig <= "000"; --do nothing
    dataMemsig <= "11"; --do nothing
    signCtrlsig <= '0'; --put 8 bit offset to add
    ALUOutsig <= "1011"; --branch if equal
    dataMemsig <= "11"; --do nothing
    regMuxCtrlsig <= '0'; --ALU TO REG

    if(branchBit = '1')then
       addCtrlsig <= "01"; -- add puts the sign extend into PC
    else
```

```
        addCtrlsig <= "00"; -- PC + 1
    end if;

elsif(instruction(15 downto 12) = "1100")then --OR
    regTablesig <= "000";
    regTableDsig <= instruction(11 downto 8);
    regTableAsig <= instruction(7 downto 4);
    regTableBsig <= instruction(3 downto 0);
    ALUOutsig <= "1100";
    dataMemsig <= "11"; -- do nothing
    addCtrlsig <= "00"; --PC + 1
    regMuxCtrlsig <= '0'; --ALU TO REG

 elsif(instruction(15 downto 12) = "1101")then --AND
   regTablesig <= "000";
   regTableDsig <= instruction(11 downto 8);
   regTableAsig <= instruction(7 downto 4);
   regTableBsig <= instruction(3 downto 0);
   ALUOutsig <= "1101";
   dataMemsig <= "11"; -- do nothing
   addCtrlsig <= "00"; --PC + 1
   regMuxCtrlsig <= '0'; --ALU TO REG

elsif(instruction(15 downto 12) = "1110")then --BRANCH IF LESS THAN
       regTablesig <= "000"; --do nothing
       dataMemsig <= "11"; --do nothing
       signCtrlsig <= '0'; --put 8 bit offset to add
       ALUOutsig <= "1110"; --branch if equal
       dataMemsig <= "11"; --do nothing

       if(branchBit = '1')then
          addCtrlsig <= "01"; -- add puts the sign extend into PC
       else
          addCtrlsig <= "00"; -- PC + 1
       end if;

   elsif(instruction(15 downto 12) = "1111")then --BRANCH IF LESS THAN
         regTablesig <= "000"; --do nothing
         dataMemsig <= "11"; --do nothing
         signCtrlsig <= '0'; --put 8 bit offset to add
         ALUOutsig <= "1111"; --branch if equal
         dataMemsig <= "11"; --do nothing

         if(branchBit = '1')then
            addCtrlsig <= "01"; -- add puts the sign extend into PC
         else
```

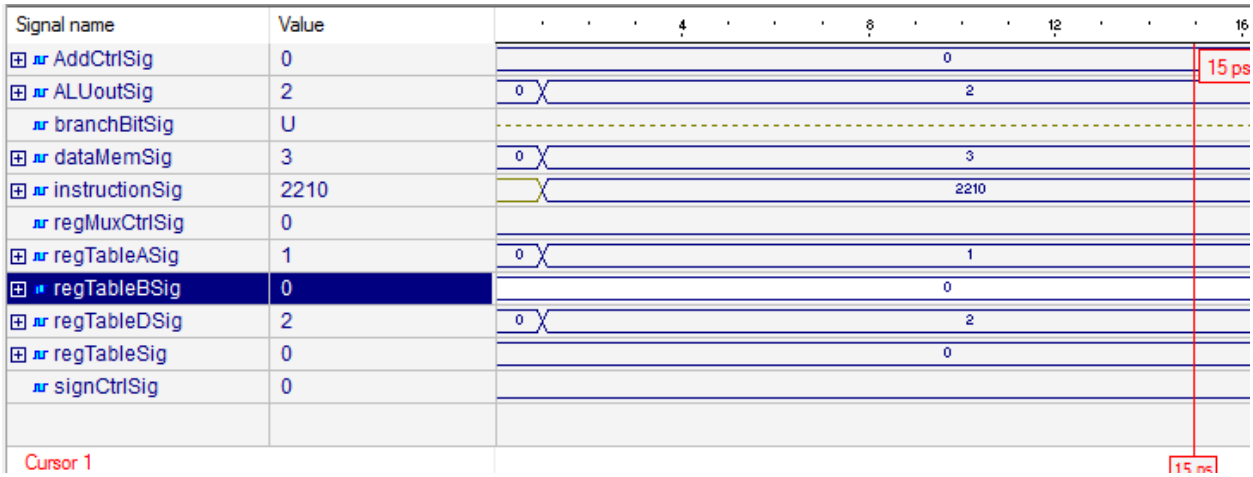
```
                addCtrlsig <= "00"; -- PC + 1
            end if;

        end if;
    end process;

    regTableA <= regTableAsig;
    regTableB <= regTableBsig;
    regTableD <= regTableDsig;
    ALUout <= ALUOutsig;
    regTable <= regTablesig;
    signCtrl <= signCtrlsig;
    dataMem <= dataMemsig;
    addCtrl <= addCtrlsig;
    regMuxCtrl <= regMuxCtrlsig;


end Behavioral;
```

Control Testing



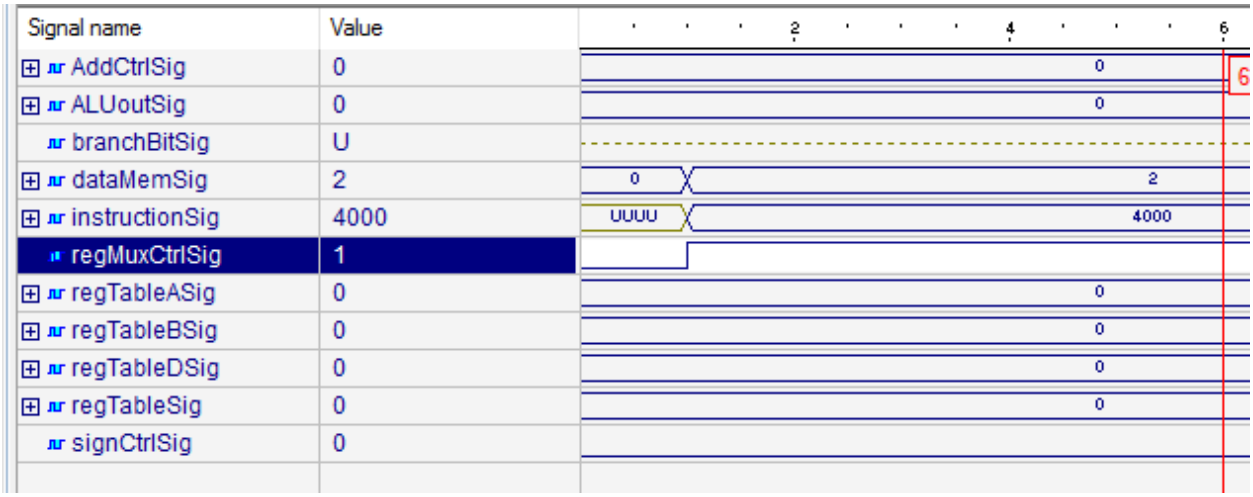
Control Halt and Jump Execution

At first, AddCtrlSig, the Adder Module signal is supplied “11” to halt all PC incrimination
Then, the Jump instruction arrives and the AddCtrlSig is changed to 1 to force the Address Mux to output an 8 bit offset from the instruction
AddCtrlSig controls the Add module which feeds the PC.



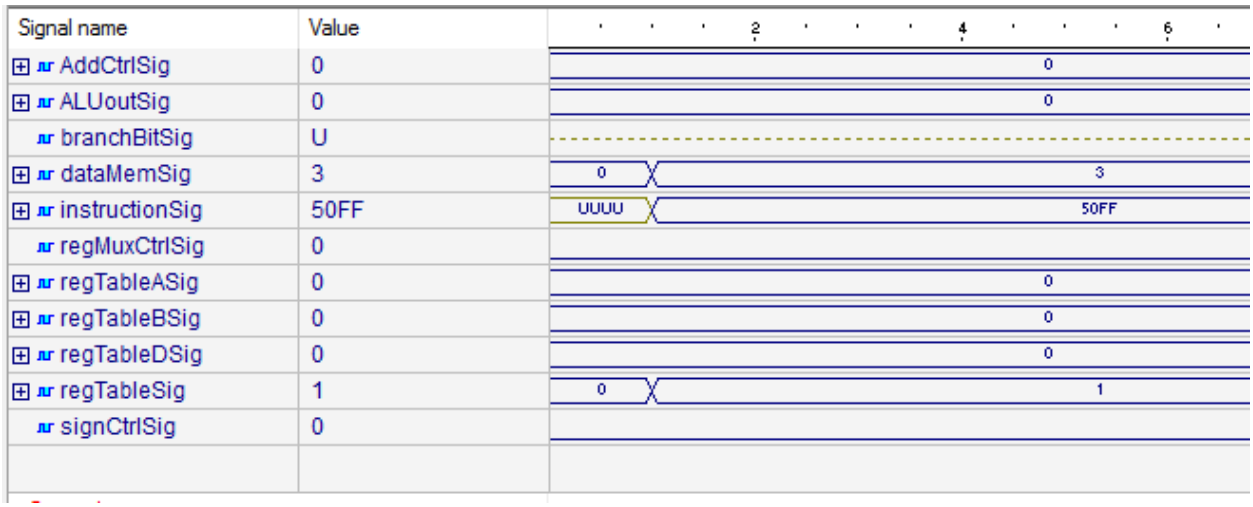
Control Add and Subtract

At one picosecond, the instruction signal arrives and the following are required to react:
The registerTableA address needs to be set to “0000”
registerTableB needs to be set to “0001”
registerTableD needs to be set to “0010”
the Add module needs to increment PC by 1 so addCtrlsig = “00”
ALUOutsig needs to be “0010” for Add
AddCtrlSig is set to 0 to increment PC by 1



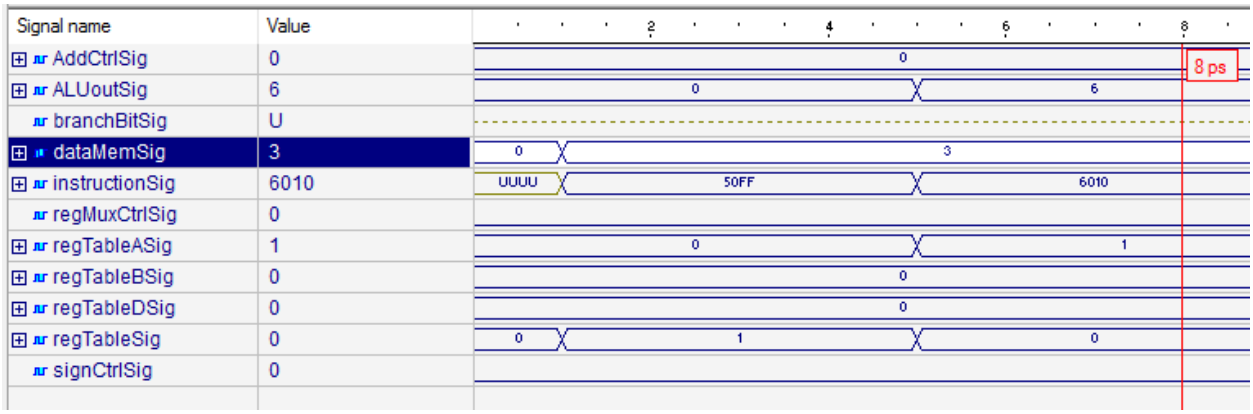
Control Load

The instruction with opcode “0100” arrives in the instruction signal
The register table regTableSig needs to be set to write, “000”
The Data Memory is set to “10” so that it is read
The regMuxCtrl which directs where the read data is going is set to ‘1’ for registers
AddCtrlSig is set to 0 to increment PC by 1



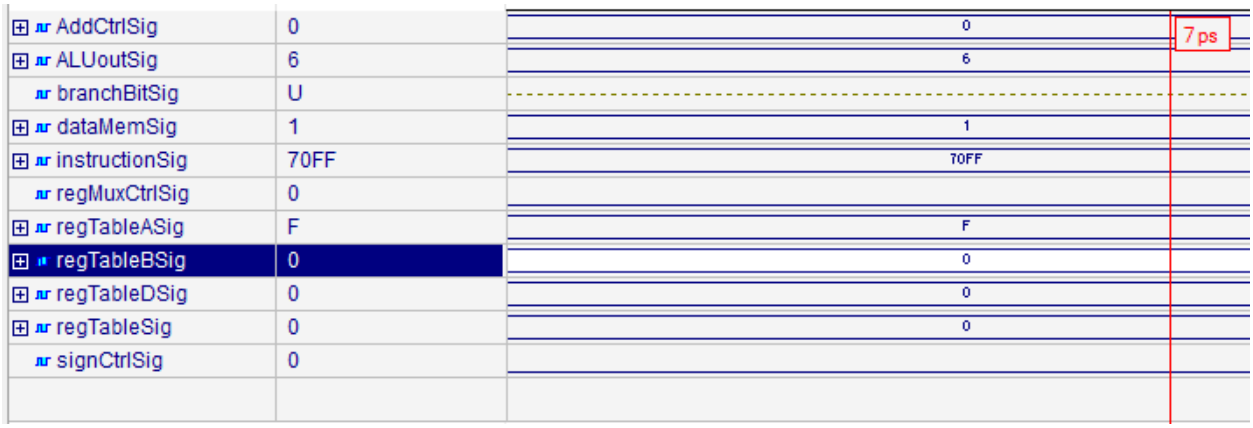
Control LoadC

LoadC takes a 8 bit constant and stores it in a specific address
In this case, the address is “0000”; the data to be written is “FF”
The data memory is set to “11” as a default read since it is not being used
The Register File is set to “000” which is a default read and or write if data changes on the input
AddCtrlSig is set to 0 to increment PC by 1



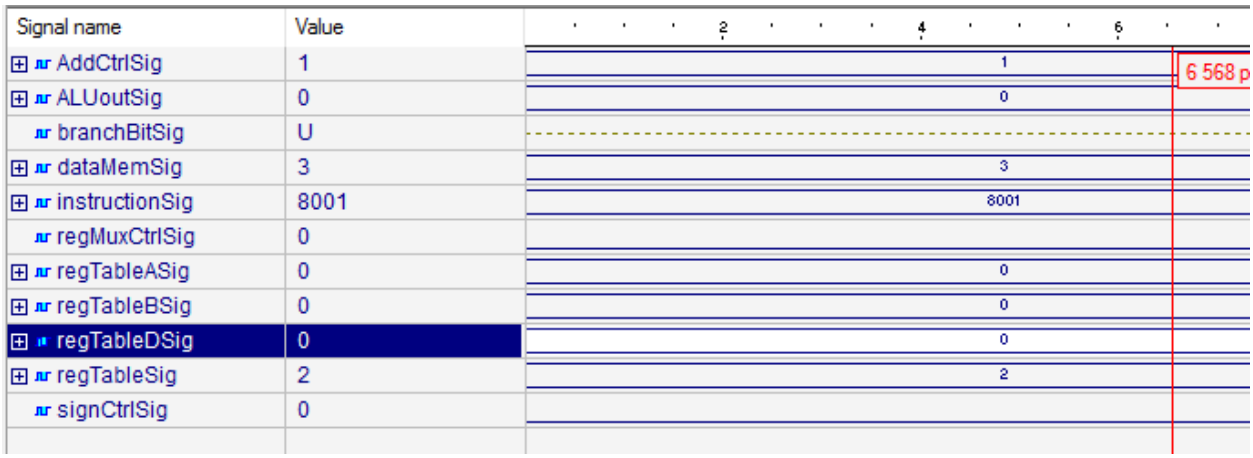
Control LoadC and Move

This includes the first part of the LoadC instruction seen above
When the instruction signal changes it can be seen that the register table signal returns to 0 to do the writing to the ALU
Since the ALU is involved, it must be set to 6 to allow the input data to flow through to the ALU/Data Memory Mux
AddCtrlSig is set to 0 to increment PC by 1



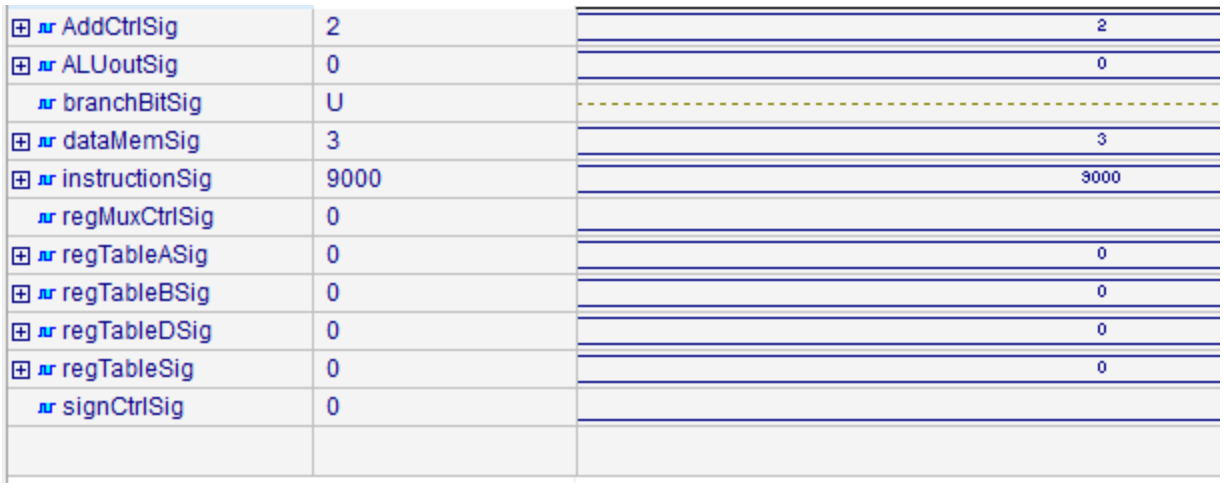
Control Store

The store instruction takes a register and writes it into the data memory.
For this to happen, the ALU must be set to 6 to allow data to flow through from the registers
The data memory must be set to 1to write data
AddCtrlSig is set to 0 to increment PC by 1



Control Jump and Link

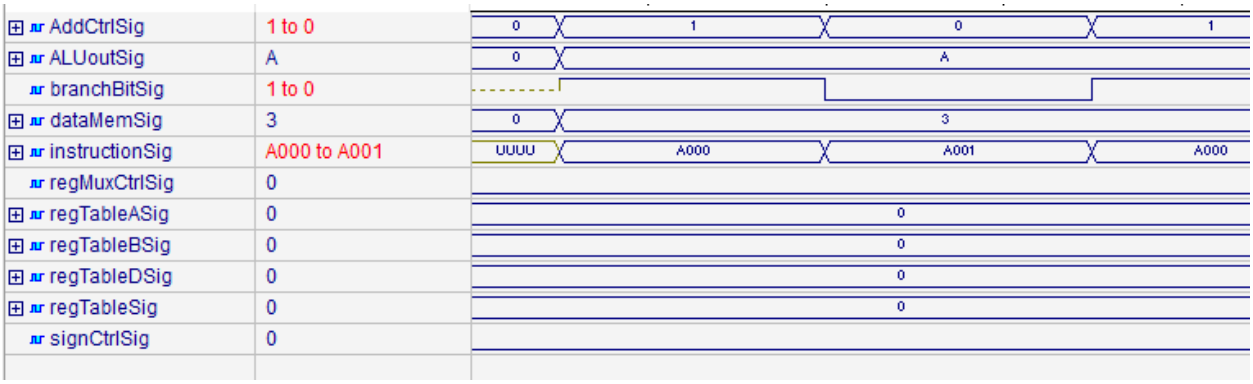
For Jump and Link, the PC is incremented by an 8 bit offset value specified by the last 8 bits of the instruction
The Sign Extension signal, signCtrlSig, is set to 0 to direct the 8 bits to the Add unit
The add unit is also supplied a signal called AddCtrlSig which at '1' adds the PC with the sign extended value
Register 15 is also written with the PC which is controlled by the regTableSig signal



Control Return

Control return takes the PC value that was written during jump and link and returns it back to the PC register.

The register file can be set to a default read state and the add module needs to be set to 2 so it takes the value from the register



Control Branch Equal/Not Equal

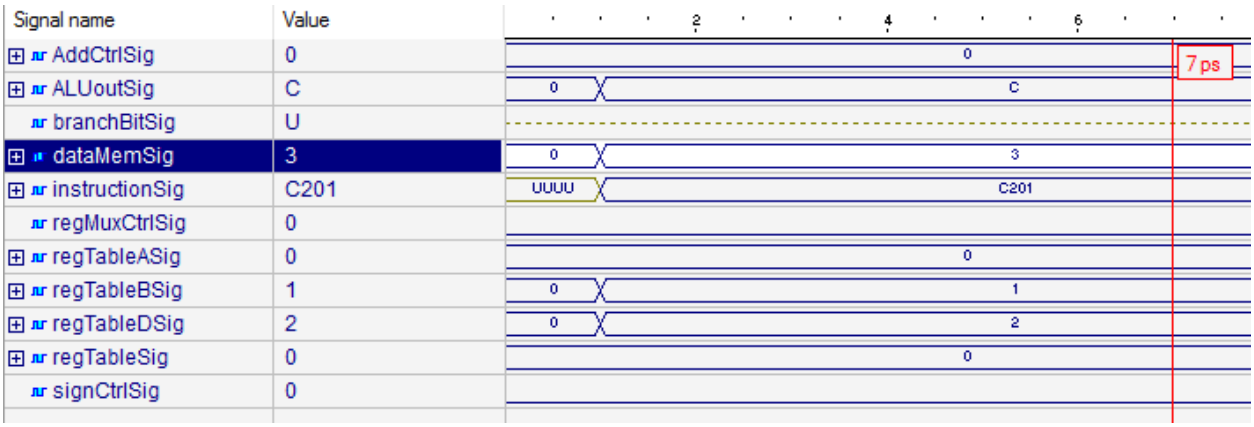
Branching does not save the address in the PC, however the adder circuit needs to take the 8 bit offset specified in the instruction.

To extend 8 bits, the signCtrlSig is set to 0

The registers need to be read so the register file is set to a default state

If the ALU determines that they are equal, then the Add circuitry needs to take the supplied offset.

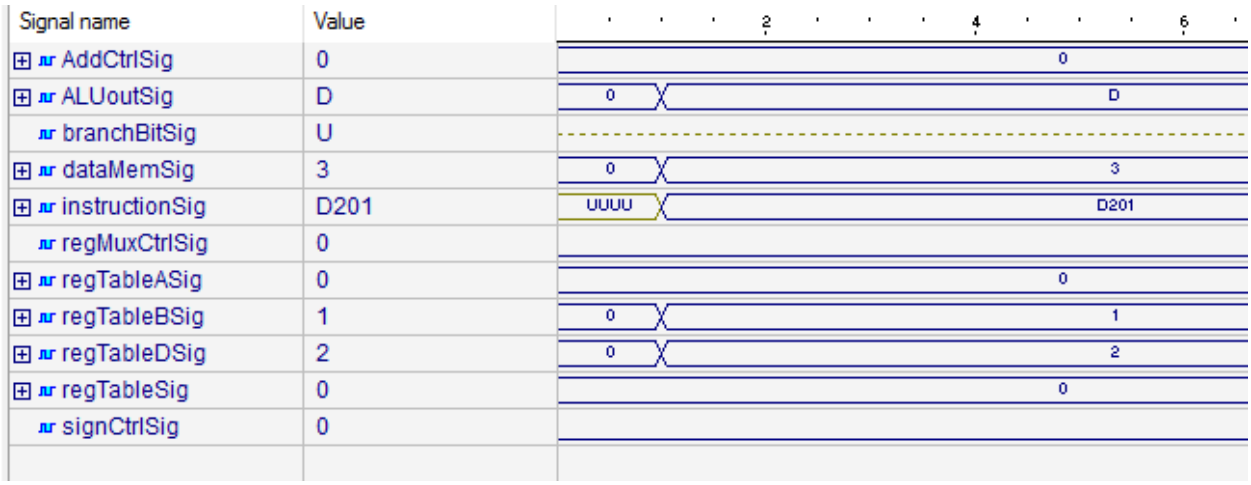
If the branch bit stays 0 then the add circuit increments the PC by one, which can be seen in the fluctuation in the AddCtrlSig as the branchBitSig is changed



Control OR

Given two registers, regTableAsig, and regTableBsig, the ALU executes a bitwise OR operation on the two and stores the result in regTableDsig.

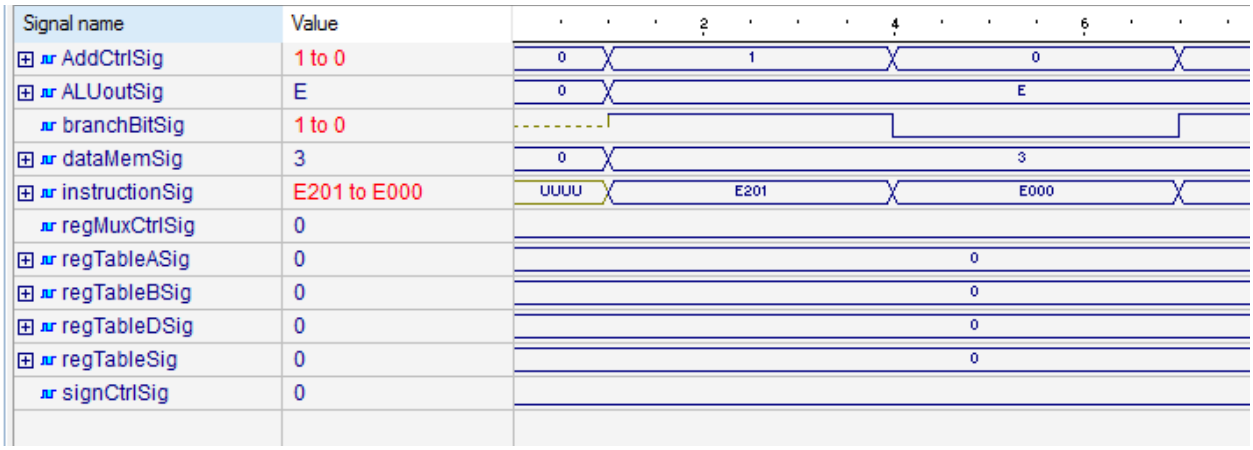
The Data Memory needs to be defaulted and the ALU/ Data Memory Mux needs to be set to 0 to allow writing to the registers



Control AND

Given two registers, regTableAsig, and regTableBsig, the ALU executes a bitwise AND operation on the two and stores the result in regTableDsig.

The Data Memory needs to be defaulted and the ALU/ Data Memory Mux needs to be set to 0 to allow writing to the registers



Control Branch if Greater/ Lesser

This is similar to the branch if equal not equal, but the ALU subtracts the two operators and determines if the first is greater or lesser than the second

As the branchBitSig signal changes the ADD circuitry reacts appropriately as defined in the other branch calls

Control Test Bench

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;



entity controlBench is
--  Port ( );
end controlBench;

architecture Behavioral of controlBench is

signal instructionSig : STD_LOGIC_VECTOR(15 downto 0);
signal branchBitSig : STD_LOGIC;
signal regMuxCtrlSig : STD_LOGIC;
signal regTableASig : STD_LOGIC_VECTOR(3 downto 0);
signal regTableBSig : STD_LOGIC_VECTOR(3 downto 0);
signal regTableDSig : STD_LOGIC_VECTOR(3 downto 0);
signal ALUoutSig: STD_LOGIC_VECTOR(3 downto 0);
signal regTableSig : STD_LOGIC_VECTOR(2 downto 0);
signal signCtrlSig: STD_LOGIC;
signal dataMemSig: STD_LOGIC_VECTOR(1 downto 0);
signal AddCtrlSig: STD_LOGIC_VECTOR(1 downto 0);

component CONTROL is
    port(
        instruction : in STD_LOGIC_VECTOR(15 downto 0);
```

```
        branchBit : in STD_LOGIC;
        regMuxCtrl : out STD_LOGIC;
        regTableA : out STD_LOGIC_VECTOR(3 downto 0);
        regTableB : out STD_LOGIC_VECTOR(3 downto 0);
        regTableD : out STD_LOGIC_VECTOR(3 downto 0);
        ALUout : out STD_LOGIC_VECTOR(3 downto 0);
        regTable : out STD_LOGIC_VECTOR(2 downto 0);
        signCtrl: out STD_LOGIC;

        dataMem : out STD_LOGIC_VECTOR(1 downto 0);
        addCtrl: out STD_LOGIC_VECTOR(1 downto 0));
end component;


begin
M1: CONTROL port map(instruction => instructionSig,  branchBit => branchBitSig,
regMuxCtrl => regMuxCtrlSig, regTableA => regTableAsig,
regTableB => regTableBsig, regTableD => regTableDsig, ALUout => ALUoutSig,
regTable => regTableSig, signCtrl => signCtrlSig, dataMem => dataMemSig,
addCtrl => addCtrlSig);

process begin
--------------------------------------------------------------------------HALT
wait for 1ps;
instructionSig <= "0000000000000000";
assert AddCtrlSig =  "11"
report "CNTRL HALT IS BAD"
severity ERROR;

--------------------------------------------------------------------------JUMP
wait for 1 ps;
instructionSig <= "0001000000000000";

assert AddCtrlSig =  "01"
report "ADDER CONTROL IS BAD"
severity ERROR;
wait for 1 ps;

assert signCtrlSig =  '1' -- 12bit extension
report "EXTEND CONTROL IS BAD"
severity ERROR;
--------------------------------------------------------------------------

--ADD reg0 to reg1 store in reg 2 / SUB WITH ALU CHANGE
wait for 1 ps;
instructionSig <= "0010001000010000";
```

```
wait for 1 ps;
assert regTableASig = "0000" -- A = 0
report "INCORRECT FIRST OP"
severity ERROR;
wait for 1 ps;
assert regTableBSig = "0001" --B = 1
report "INCORRECT SECOND OP"
severity ERROR;
wait for 1 ps;
assert regTableDSig = "0010" --D = 0010
report "INCORRECT DEST OP"
severity ERROR;
wait for 1 ps;

assert addCtrlsig = "00" -- PC + 1
report "INCORRECT DEST OP"
severity ERROR;
wait for 1 ps;

 assert  ALUOutsig = "0010"; -- ALU to ADD
 report "INCORRECT ALU ADD OP"
 severity ERROR;
 wait for 1 ps;

assert  regMuxCtrlSig = '0'; -- REG MUX to REG
 report "INCORRECT REG MUX LOCAL OP"
 severity ERROR;
 wait for 1 ps;
------------------------------------------------------------LOAD
wait for 1 ps;
instructionSig <= "0100000000000000"; --load reg 0 with add 00000000
wait for 1 ps;

assert  regTablesig = "000"; -- REG set to write
report "REG TABLE SET WRONG"
severity ERROR;
wait for 1 ps;

assert  regTableDSig = "0000" -- REGister to write
report "REG TABLE SET WRONG"
severity ERROR;
wait for 1 ps;

 assert dataMemsig = "10" -- data mem to read
 report "DATA REG SET WRONG"
 severity ERROR;
```

```
 wait for 1 ps;

assert regMuxCtrlsig = '1'
report " REG MUX SET WRONG"
severity ERROR;
wait for 1 ps;
------------------------------------------------------------LOAD C
wait for 1 ps;
instructionSig <= "0101000011111111"; --load reg 0 with 11111111
wait for 1 ps;

assert  regTablesig = "000"; -- REG set to read/write
report "REG TABLE SET WRONG"
severity ERROR;
wait for 1 ps;

assert  regTableDSig = "0000" -- REGister to write
report "REG TABLE SET WRONG"
severity ERROR;
wait for 1 ps;

assert dataMemsig = "11" -- data mem to do nothing
report "DATA REG SET WRONG"
severity ERROR;
wait for 1 ps;
------------------------------------------------------------MOV reg 1 to reg 0
instructionSig <= "0110000000010000"; --load reg 0 with 11111111
wait for 1 ps;

assert regMuxCtrlsig = '0' --ALU TO REG
report "REG MUX SET WRONG"
severity ERROR;
wait for 1 ps;

assert  regTablesig = "000"; -- REG set to read/write
report "REG TABLE SET WRONG"
severity ERROR;
wait for 1 ps;

assert ALUOutsig = "0110"
report "ALU SET WRONG"
severity ERROR;
 ----------------------------------------------------STORE reg 0 to 11111111
instructionSig <= "0111000011111111";
wait for 1 ps;
```

```
assert regMuxCtrlsig = '0' --ALU TO REG
report "REG MUX SET WRONG"
severity ERROR;
wait for 1 ps;

assert  regTablesig = "000" -- REG set to read/write
report "REG TABLE SET WRONG"
severity ERROR;
wait for 1 ps;

assert ALUOutsig = "0110"
report "ALU SET WRONG"
severity ERROR;

assert dataMemsig = "01";
report "DATA MEM SET WRONG"
severity ERROR;
-----------------------------------------------------------JUMP AND LINK
instructionSig <= "1000000000000001";

assert signCtrlsig = '0' --8 bits to PC
report "SIGN EXTEN SET WRONG"
severity ERROR;
wait for 1 ps;

assert addCtrlsig = "01"
report "ADD UNIT SET WRONG"
severity ERROR;
wait for 1 ps;

assert regTablesig = "010"
report "REGFILE SET WRONG"
severity ERROR;
wait for 1 ps;

-----------------------------------------------------------RETURN
instructionSig <= "1001000000000000";
wait for 1 ps;

assert addCtrlsig = "10"
report "ADD UNIT SET WRONG"
severity ERROR;
wait for 1 ps;

assert regTablesig = "010"
report "REGFILE SET WRONG"
```

```
severity ERROR;
wait for 1 ps;

------------------------------------------------------------BRANCH EQUAL NOT EQUAL
wait for 1 ps;
instructionSig <= "1010000000000000";
branchBitSig <= '1';
wait for 1 ps;

assert ALUOutsig <= "1010"
report "ALU SET INCORRECTLY"
severity ERROR;

assert signCtrlsig = '0' --8 bits to PC
report "SIGN EXTEN SET WRONG"
severity ERROR;
wait for 1 ps;

assert  addCtrlsig = "01"
report "BRANCH BIT RESPONSE WRONG"
severity ERROR;
wait for 1 ps;

instructionSig <= "1010000000000001";
branchBitSig <= '0';
wait for 1 ps;
assert  addCtrlsig = "00"
report "BRANCH BIT RESPONSE WRONG"
severity ERROR;
wait for 1 ps;
-------------------------------------------------OR 1 and 0 into 2

wait for 1 ps;
instructionSig <= "1100001000000001";
wait for 1 ps;
assert regTableASig = "0000" -- A = 0
report "INCORRECT FIRST OP"
severity ERROR;
wait for 1 ps;
assert regTableBSig = "0001" --B = 1
report "INCORRECT SECOND OP"
severity ERROR;
wait for 1 ps;
assert regTableDSig = "0010" --D = 0010
report "INCORRECT DEST OP"
severity ERROR;
```

```
wait for 1 ps;

assert addCtrlsig = "00" -- PC + 1
report "INCORRECT ADD OP"
severity ERROR;
wait for 1 ps;

 assert  ALUOutsig = "1100"; -- ALU to OR
 report "INCORRECT ALU OR OP"
 severity ERROR;
 wait for 1 ps;

assert  regMuxCtrlSig = '0'; -- REG MUX to REG
 report "INCORRECT REG MUX LOCAL OP"
 severity ERROR;
 wait for 1 ps;

-------------------------------------------------AND
wait for 1 ps;
instructionSig <= "1101001000000001";
wait for 1 ps;
assert regTableASig = "0000" -- A = 0
report "INCORRECT FIRST OP"
severity ERROR;
wait for 1 ps;
assert regTableBSig = "0001" --B = 1
report "INCORRECT SECOND OP"
severity ERROR;
wait for 1 ps;
assert regTableDSig = "0010" --D = 0010
report "INCORRECT DEST OP"
severity ERROR;
wait for 1 ps;

assert addCtrlsig = "00" -- PC + 1
report "INCORRECT ADD OP"
severity ERROR;
wait for 1 ps;

 assert  ALUOutsig = "1101"; -- ALU to OR
 report "INCORRECT ALU OR OP"
 severity ERROR;
 wait for 1 ps;

assert  regMuxCtrlSig = '0'; -- REG MUX to REG
 report "INCORRECT REG MUX LOCAL OP"
```

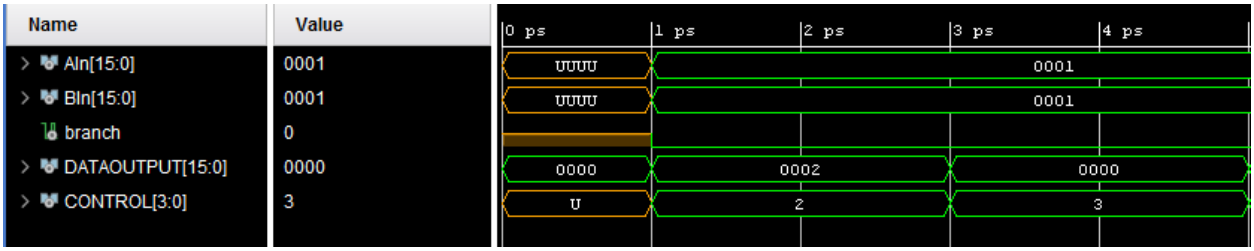
```
        severity ERROR;
        wait for 1 ps;

        ------------------------------------------------BRANCH GREATER LESS THAN

        wait for 1 ps;
        instructionSig <= "1110001000000001";
        branchBitSig <= '1';
        wait for 1 ps;

        assert ALUOutsig <= "1110"
        report "ALU SET INCORRECTLY"
        severity ERROR;

        assert signCtrlsig = '0' --8 bits to PC
        report "SIGN EXTEN SET WRONG"
        severity ERROR;
        wait for 1 ps;

        assert  addCtrlsig = "01"
        report "BRANCH BIT RESPONSE WRONG"
        severity ERROR;
        wait for 1 ps;

        instructionSig <= "1110000000000000";
        branchBitSig <= '0';
        wait for 1 ps;
        assert  addCtrlsig = "00"
        report "BRANCH BIT RESPONSE WRONG"
        severity ERROR;
        wait for 1 ps;

        end process;
end Behavioral;
```

ALU Code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;
use ieee.std_logic_unsigned.all;

entity ALU is
   port(
      bit16InA : in STD_LOGIC_VECTOR(15 downto 0);
      bit16InB : in STD_LOGIC_VECTOR(15 downto 0);
      bit16Out : out STD_LOGIC_VECTOR(15 downto 0); -- this goes to either datamem
or regtable
      branchBit : out STD_LOGIC;
      signCtrl : in STD_LOGIC_VECTOR(3 downto 0));

end ALU;

architecture Behavioral of ALU is
signal bit16Outsig : STD_LOGIC_VECTOR(15 downto 0) := "0000000000000000";
signal branchBitsig : STD_LOGIC := '0';
begin

process(signCtrl) begin
   if(signCtrl = "0010") then --add
      bit16Outsig <= bit16InA + bit16InB;
   elsif(signCtrl = "0011") then --sub
      bit16Outsig <= bit16InA - bit16InB;

   elsif(signCtrl = "0110") then --mov/store passes B out to dataMem or registers
      bit16Outsig <= bit16InA;
   elsif(signCtrl = "1010") then
      if(bit16inA - bit16inB = "0000000000000000") then --branch if equal
         branchBitsig <= '1';
      else
         branchBitsig <= '0';
      end if;
   elsif(signCtrl = "1011") then
      if(bit16inA - bit16inB = 0) then --branch if not equal
         branchBitsig <= '0';
      else
         branchBitsig <= '1';
      end if;

   elsif(signCtrl = "1100") then
      bit16Outsig <= bit16InA or bit16InB; --or
```

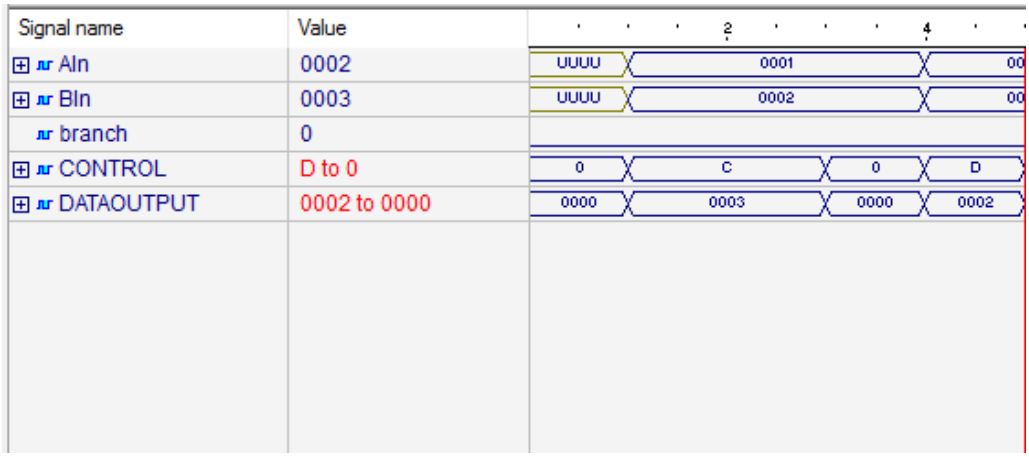
```
            elsif(signCtrl = "1101") then
               bit16Outsig <= bit16InA and bit16InB; -- and

            elsif(signCtrl = "1110") then
               if(bit16inB - bit16inA > 0) then
                  branchBitsig <= '1';  --Branch if less than
               else
                  branchBitsig <= '0';
               end if;
            elsif(signCtrl = "1111") then
               if(bit16inA - bit16inB > 0) then
                  branchBitsig <= '1';  --Branch if greater than
               else
                  branchBitsig <= '0';
               end if;
            else
               bit16Outsig <= "0000000000000000";
            end if;
         end process;


         bit16Out <= bit16Outsig;
         branchBit <= branchBitsig;
end Behavioral;
```

ALU Testing



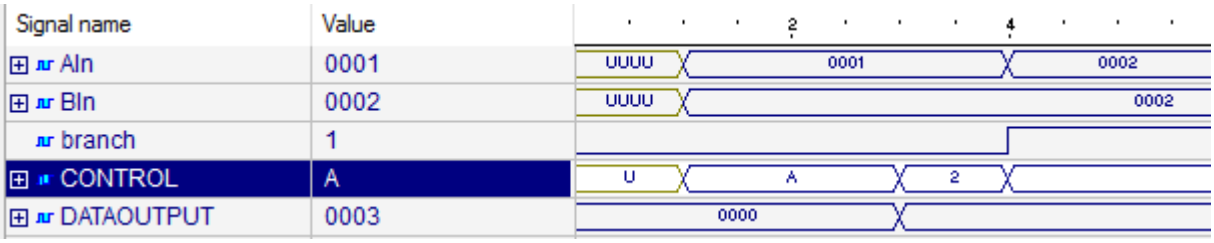
ALU Add/Sub

When the Control signal is 2, the values seen on Ain and Bin are added together
It can be seen that both Ain and Bin contain 1 so the sum is 2
After the add, the control is changed to a 3 for a subtraction to take place and the result is 0



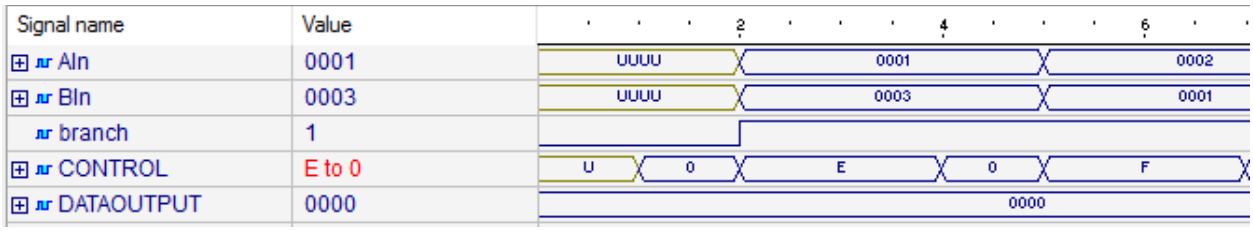
ALU AND/OR

When the CONTROL signal is C the inputs Ain and Bin are OR'd together, resulting in a 3
When the CONTROL is set to D the DATAOUTPUT is the resulting AND operation which would result in 2 with Ain being 1 and Bin being 2



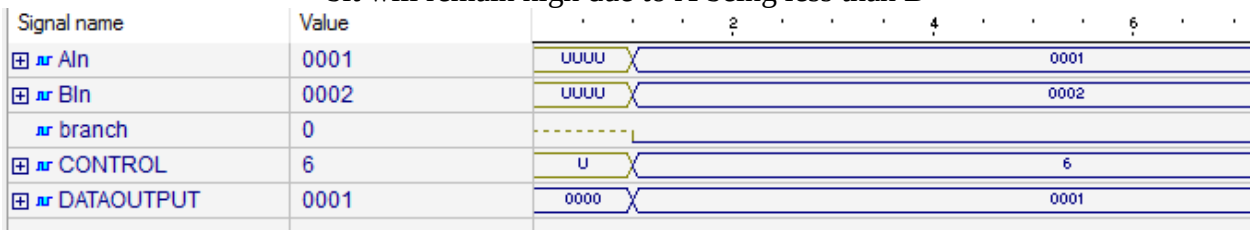
ALU Branch Equal/Not Equal

When Ain and Bin are not matching when CONTROL is A, meaning Branch Equal, the resulting branch bit output is 0.
When CONTROL is changed to B(cut off the screen), the resulting branch bit is 1



ALU Branch Less Than / Greater

This acts the exact same as equal not equal, however if A is greater than B when Control is set to E, the branch bit is set high. When branch less than is set, if the inputs are swapped the branch bit will remain high due to A being less than B



ALU Store/Move

When Storing or moving the input at AIN is set to the output.
Control must be equal to 6 for this to happen

ALU Test Bench

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;



entity aluBench is
--  Port ( );
end aluBench;

architecture Behavioral of aluBench is

signal AIn: std_logic_vector(15 downto 0);
signal BIn: std_logic_vector(15 downto 0);
signal branch: std_logic;
signal DATAOUTPUT: std_logic_vector(15 downto 0);
signal CONTROL: std_logic_vector(3 downto 0);


component ALU is
    port(
      bit16InA : in STD_LOGIC_VECTOR(15 downto 0);
      bit16InB : in STD_LOGIC_VECTOR(15 downto 0);
      bit16Out : out STD_LOGIC_VECTOR(15 downto 0); -- this goes to either datamem
or regtable
      branchBit : out STD_LOGIC;
```

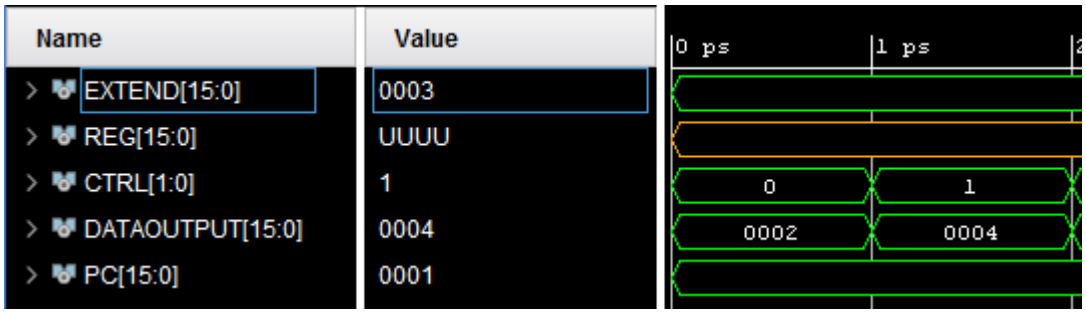
```
        signCtrl : in STD_LOGIC_VECTOR(3 downto 0));
end component;


begin
M1: ALU port map(bit16InA => AIn, bit16InB => Bin, bit16Out => DATAOUTPUT,
branchBit => branch, signCtrl => CONTROL);

process begin

----test add
wait for 1ps;
AIn <= "0000000000000001";
BIn <= "0000000000000001";
branch <= '0';
CONTROL <= "0010";
wait for 1 ps;
assert DATAOUTPUT = "0000000000000010"
report "ALU ADD IS BAD"
severity ERROR;

----test sub
wait for 1 ps;
AIn <= "0000000000000001";
BIn <= "0000000000000001";
branch <= '0';
CONTROL <= "0011";
wait for 1 ps;
assert DATAOUTPUT = "0000000000000000"
report "ALU SUB IS BAD"
severity ERROR;

--alu mov store
wait for 1 ps;
AIn <= "0000000000000001";
BIn <= "0000000000000010";
branch <= '0';
CONTROL <= "0110";
wait for 1 ps;
assert DATAOUTPUT =  "0000000000000001";
report "ALU MOVE IS BAD"
severity ERROR;

----alu equal than
wait for 1 ps;
AIn <= "0000000000000001";
```

```
BIn <= "0000000000000010";

CONTROL <= "1010";
wait for 1 ps;
assert branch =  '0'
report "ALU BRANCH EQUAL IS BAD"
severity ERROR;
wait for 1 ps;

CONTROL <= "0010";
wait for 1 ps;
CONTROL <= "1010";
AIn <= "0000000000000010";
BIn <= "0000000000000010";
wait for 1 ps;

assert branch =  '1'
report "ALU BRANCH EQUAL IS BAD"
severity ERROR;

--OR
CONTROL <= "0000";
wait for 1 ps;
CONTROL <= "1100";
AIn <= "0000000000000001";
BIn <= "0000000000000010";

wait for 1 ps;
assert DATAOUTPUT = "0000000000000011";
report "ALU OR IS BAD"
severity ERROR;

--AND
wait for 1 ps;
CONTROL <= "0000";
wait for 1 ps;
CONTROL <= "1101";
AIn <= "0000000000000010";
BIn <= "0000000000000011";
wait for 1 ps;

assert DATAOUTPUT = "0000000000000010";
report "ALU AND IS BAD"
severity ERROR;

--branch less that
```

```
        wait for 1 ps;
        CONTROL <= "0000";
        wait for 1 ps;
        CONTROL <= "1110";
        AIn <= "0000000000000001";
        BIn <= "0000000000000011";
        wait for 1 ps;

        assert BRANCH = '1'
        report "ALU BRANCH LESS IS BAD"
        severity ERROR;

        --branch greater
        wait for 1 ps;
        CONTROL <= "0000";
        wait for 1 ps;
        CONTROL <= "1111";
        AIn <= "0000000000000010";
        BIn <= "0000000000000001";
        wait for 1 ps;

        assert BRANCH = '1'
        report "ALU BRANCH GREATER IS BAD"
        severity ERROR;

        end process;
end Behavioral;
```

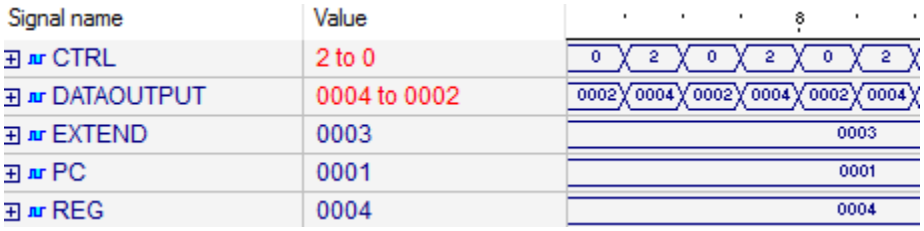
PC ADD Unit

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;
use ieee.std_logic_unsigned.all;

entity ADD is
   port(
      bit16In : in STD_LOGIC_VECTOR(15 downto 0);
      PCin : in STD_LOGIC_VECTOR(15 downto 0);
      REGin : in STD_LOGIC_VECTOR(15 downto 0);
      bit16Out : out STD_LOGIC_VECTOR(15 downto 0);
      signCtrl : in STD_LOGIC_VECTOR(1 downto 0));
end ADD;

architecture Behavioral of ADD is
signal bit16OutSig : STD_LOGIC_VECTOR(15 downto 0) := "0000000000000000";
begin

CTRL: process(signCtrl)
   begin
   if(signCtrl = "00") then
      bit16OutSig <= PCin + "0000000000000001";--PC = PC+1
   elsif(signCtrl = "01") then
      bit16OutSig <= PCin + bit16In;--from signextend1
   elsif(signCtrl = "10") then
      bit16OutSig <= REGin;--Branch value
   else
      bit16OutSig <= "0000000000000000"; --HALT
   end if;
   end process;

   bit16Out <= bit16OutSig;
end Behavioral;
```



PC + 1

When the Control signal, CTRL, is set to 1, the PC is incremented by 1. With it starting off as 1, it is immediately incremented to 2 at 0ps



PC + Extended Address

When the Control is set to 2, he offset address is added to the PC, hence 3+ 1 = 4

ADD Test Bench

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;



entity addBench is
--  Port ( );
end addBench;

architecture Behavioral of addBench is

signal EXTEND: std_logic_vector(15 downto 0);
signal REG: std_logic_vector(15 downto 0);
signal CTRL: std_logic_vector(1 downto 0);
signal DATAOUTPUT: std_logic_vector(15 downto 0);
signal PC: std_logic_vector(15 downto 0);


component ADD is
    port(
      bit16In : in STD_LOGIC_VECTOR(15 downto 0);
      PCin : in STD_LOGIC_VECTOR(15 downto 0);
      REGin : in STD_LOGIC_VECTOR(15 downto 0);
      bit16Out : out STD_LOGIC_VECTOR(15 downto 0);
      signCtrl : in STD_LOGIC_VECTOR(1 downto 0));
end component;


begin
M1: ADD port map(bit16In => EXTEND, PCin => PC, REGin => REG, bit16Out =>
DATAOUTPUT, signCtrl => CTRL);

process begin

--test control to increment PC + 1
wait for 1 ps;
--EXTEND <= "0000000000000011";
CTRL <= "00";
wait for 1 ps;
PC <=     "0000000000000001";
wait for 1 ps;

assert DATAOUTPUT = "0000000000000010"
report "PC WAS NOT INCREMENTED"
```

```
        severity ERROR;

        --test to see if extended gets added
        CTRL <= "00";
        EXTEND <= "0000000000000011";
        PC <=  "0000000000000001";
        REG <= "0000000000000100";
        wait for 2 ps;
        CTRL <= "01";
        wait for 2 ps;

        assert DATAOUTPUT = "0000000000000100"
        report "PC WAS NOT INCREMENTED"
        severity ERROR;

        ----test to see if register write works
        CTRL <= "00";
        EXTEND <= "0000000000000011";
        PC <=  "0000000000000001";
        REG <= "0000000000000100";
        wait for 2 ps;
        CTRL <= "10";
        wait for 2 ps;

        assert DATAOUTPUT = "0000000000000100"
        report "PC WAS NOT INCREMENTED"
        severity ERROR;
        end process;


end Behavioral;
```

Register File Code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;
use ieee.std_logic_unsigned.all;
--this is for reg table
entity REGTABLE is
   port(
   addressA: in STD_LOGIC_VECTOR(3 downto 0) := (others => '0');--from contrl
   addressB: in STD_LOGIC_VECTOR(3 downto 0) := (others => '0');--from contrl
   addressD: in STD_LOGIC_VECTOR(3 downto 0) := (others => '0');--from contrl
   PC: in STD_LOGIC_VECTOR(15 downto 0) := (others => '0'); --from CP
   regMuxIn: in STD_LOGIC_VECTOR(15 downto 0) := (others => '0');--from REGister
MUX
   addOUT : out STD_LOGIC_VECTOR(15 downto 0);--to ADD
   signExtend2: in STD_LOGIC_VECTOR(15 downto 0) := (others => '0'); -- for loadC
   signCTRL: in STD_LOGIC_VECTOR(2 downto 0):= (others => '0');  --from control
   bit16OutB: out STD_LOGIC_VECTOR(15 downto 0);
   bit16OutA: out STD_LOGIC_VECTOR(15 downto 0));

end REGTABLE;

architecture Behavioral of REGTABLE is
signal bit16OutSigA : STD_LOGIC_VECTOR(15 downto 0) := "0000000000000000";
signal bit16OutSigB : STD_LOGIC_VECTOR(15 downto 0) := "0000000000000000";


subtype word is std_logic_vector(15 downto 0);
type memory is array((2**4)-1 downto 0) of word;
signal RAM: memory;

begin

process(signCTRL)
begin
   if(signCTRL = "000")then
      bit16OutSigA <= RAM(conv_integer(addressA)); --basic read at addresses R TYPE
      bit16OutSigB <= RAM(conv_integer(addressB));


   elsif(signCTRL = "001")then
      RAM(conv_integer(addressA)) <= signExtend2;  --write immediate to mem
LOADC
```

```
        elsif(signCTRL = "010")then
            bit16OutSigA <= RAM(conv_integer(addressA));
            bit16OutSigB <= RAM(conv_integer(addressB));
            RAM(conv_integer("1111")) <= PC;          --write PC value to reg for Jump and
    LINK

        else
            bit16OutSigA <= RAM(conv_integer(addressA)); --basic read at addresses
            bit16OutSigB <= RAM(conv_integer(addressB)); --basic read at addresses
        end if;
    end process;

    process(regMuxIn) begin
    if(signCTRL = "000") then
        RAM(conv_integer(addressD)) <= regMuxIn;
        end if;
    end process;

    addOUT <= RAM(conv_integer("1111"));
    bit16OutA <= bit16OutSigA;
    bit16OutB <= bit16OutSigB;
end Behavioral;
```

I could not successfully simulate the register file by the due date. I will have it ready and operational by the next project.

Register File Test Bench

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.numeric_std.ALL;

use ieee.std_logic_unsigned.all;



entity regBench is

--  Port ( );

end regBench;
```

```
architecture Behavioral of regBench is

component REGTABLE is
     port(
    addressA: in STD_LOGIC_VECTOR(3 downto 0);--from contrl
    addressB: in STD_LOGIC_VECTOR(3 downto 0);--from contrl
    addressD: in STD_LOGIC_VECTOR(3 downto 0);--from contrl
    PC: in STD_LOGIC_VECTOR(15 downto 0); --from PC
    regMuxIn: in STD_LOGIC_VECTOR(15 downto 0);--from REGister MUX
    addOUT : out STD_LOGIC_VECTOR(15 downto 0);--to ADD
    signExtend2: in STD_LOGIC_VECTOR(15 downto 0); -- for loadC
    signCTRL: in STD_LOGIC_VECTOR(2 downto 0);  --from control
    bit16OutB: out STD_LOGIC_VECTOR(15 downto 0);
    bit16OutA: out STD_LOGIC_VECTOR(15 downto 0));
end component;

signal adA : STD_LOGIC_VECTOR(3 downto 0) := "0000";
signal adB : STD_LOGIC_VECTOR(3 downto 0):= "0000";
signal adD : STD_LOGIC_VECTOR(3 downto 0):= "0000";
signal PCsig: STD_LOGIC_VECTOR(15 downto 0) := "1111111111111111";
signal regMuxSig : STD_LOGIC_VECTOR(15 downto 0) := "1111111111111111";
signal signExSig: STD_LOGIC_VECTOR(15 downto 0):= "0000000000001111";
signal Ctrl : STD_LOGIC_VECTOR(2 downto 0):= "111";
signal outA : STD_LOGIC_VECTOR(15 downto 0) := "1111111111111111";
signal outB : STD_LOGIC_VECTOR(15 downto 0) := "1111111111111111";
signal addOutSig: STD_LOGIC_VECTOR(15 downto 0) := "1111111111111111";

begin
```

```
M1: REGTABLE port map(addressA => adA, addressB => adB, addressD => adD,PC => PCsig,
regMuxIn => regMuxSig, signExtend2 => signExSig, signCtrl => Ctrl, addOut =>
addOutSig,bit16OutA => outA, bit16OutB => outB);


 process begin


Ctrl <= "111"; -- read

adA <= "0000";

adB <= "0000";

adD <= "0000";

signExSig <= "0000000000001111";

regMuxSig <="0000000000000000";

PCsig <= "1111111111111111";


wait for 5 ps;

 Ctrl <= "001"; -- load imm


 wait for 5 ps;

 Ctrl <="111"; --read

 wait for 1 ps;


 assert outA = "0000000000001111"

 report "REG FILE NOT SET"

 severity ERROR;


 wait for 5 ps;

 Ctrl <= "010"; --write pc

 wait for 1 ps;


assert addOutSig = "1111111111111111"
```

```
report "REG FILE NOT SET"
severity ERROR;
 end process;
end Behavioral;
```

Data Memory Code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;
use ieee.std_logic_unsigned.all;
--this is for instruction mem
entity MEM is
    port(
    address: in STD_LOGIC_VECTOR(15 downto 0);
    bit8out: out STD_LOGIC_VECTOR(7 downto 0);
    bit12out: out STD_LOGIC_VECTOR(11 downto 0);
    bit16out: out STD_LOGIC_VECTOR(15 downto 0));
end MEM;

architecture Behavioral of MEM is
signal bit16OutSig : STD_LOGIC_VECTOR(15 downto 0) := "0000000000000000";
signal bit12OutSig : STD_LOGIC_VECTOR(15 downto 0) := "000000000000";
signal bit8OutSig : STD_LOGIC_VECTOR(15 downto 0) := "00000000";

subtype word is std_logic_vector(15 downto 0);
type memory is array((2**16)-1 downto 0) of word;
signal INSTMEM: memory;
```

```
begin
ADDR: process(address)
begin
   bit8OutSig <= INSTMEM(conv_integer(address));
   bit12OutSig <= INSTMEM(conv_integer(address));
   bit16OutSig <= INSTMEM(conv_integer(address));
end process;
bit8Out <= bit8OutSig;
bit12Out <= bit12OutSig;
bit16Out <= bit16OutSig;
end Behavioral;
```

Program Counter Register

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;
use ieee.std_logic_unsigned.all;

entity PC is
    generic ( N : integer := 16);
    port(
    CLK : in STD_LOGIC;
    PCIN : in STD_LOGIC_VECTOR(15 downto 0);
    PCOUT : out STD_LOGIC_VECTOR(15 downto 0));
end PC;

architecture Behavioral of PC is
signal PCOUTsig : STD_LOGIC_VECTOR(15 downto 0) := "0000000000000000";
begin
    CLOCK : process(CLK)
    begin
    if rising_edge(CLK) then
        PCOUTsig <= PCIN;
    end if;
    end process;

    PCOUT <= PCOUTsig;
end Behavioral;
```

ALU/ DATA MEMORY Mux

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;
use ieee.std_logic_unsigned.all;

entity REGMUX is
    port(
        signCtrl : in STD_LOGIC;
        bit16Out : out STD_LOGIC_VECTOR(15 downto 0);
        aluIn : in STD_LOGIC_VECTOR(15 downto 0);
        memIn : in STD_LOGIC_VECTOR(15 downto 0));
end REGMUX;

architecture Behavioral of REGMUX is
signal bi16Outsig : STD_LOGIC_VECTOR(15 downto 0) := "0000000000000000";
begin

process(signCtrl)begin
    if(signCtrl = '0')then
        bi16Outsig <= aluIn;
    else
        bi16Outsig <= memIn;
    end if;
end process;
bit16Out <= bi16Outsig;
end Behavioral;
```

Sign Extension 1

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;
use ieee.std_logic_unsigned.all;

entity SIGN1 is
    port(
        in8bit : in STD_LOGIC_VECTOR(7 downto 0);
        in12bit: in STD_LOGIC_VECTOR(11 downto 0);
        bit16Out: out STD_LOGIC_VECTOR(15 downto 0);
        signCtrl: in STD_LOGIC);
end SIGN1;

architecture Behavioral of SIGN1 is
signal bit16OutSig : STD_LOGIC_VECTOR(15 downto 0) := "0000000000000000";
begin
CTRL: process(signCtrl)
    begin
    if(signCtrl = '0') then
        bit16OutSig <= in8bit;
    elsif(signCtrl = '1') then
        bit16OutSig <= in12bit;
    end if;
    end process;
    bit16Out <= bit16OutSig;
end Behavioral;
```

Sign Extension 2

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;
use ieee.std_logic_unsigned.all;

entity SIGN2 is
    port(
        in8bit : in STD_LOGIC_VECTOR(7 downto 0);
        bit16Out: out STD_LOGIC_VECTOR(15 downto 0));
end SIGN2;

architecture Behavioral of SIGN2 is
signal bit16OutSig : STD_LOGIC_VECTOR(15 downto 0) := "0000000000000000";
begin
    process(in8bit)
    begin
        bit16OutSig <= in8bit;

    end process;
bit16Out <= bit16OutSig;
end Behavioral;
```

	15	14	13	12	RegisterMux	DataMemory	Mem/Reg Mux	ALU	Addr Mux	ADD	SignExtension1	RegTables
HALT	0	0	0	0	0	0	0	0	0	1	0	0
JUMP	0	0	0	1	0	0	0	0	1	1	1	0
ADD	0	0	1	0	1	0	1	1	0	0	0	1
SUB	0	0	1	1	1	0	1	1	0	0	0	1
LOAD	0	1	0	0	1	1	1	0	0	1	1	1
LOADC	0	1	0	1	0	0	0	0	0	0	0	1
MOV	0	1	1	0	1	0	1	1	0	0	0	1
STORE	0	1	1	1	0	1	0	1	1	1	1	1
CALL	1	0	0	0	0	0	0	0	1	1	1	1
RETURN	1	0	0	1	0	0	0	0	0	1	0	1
CALL if EQUAL	1	0	1	0	0	0	0	1	0	1	0	1
CALL if !EQUAL	1	0	1	1	0	0	0	1	0	1	0	1
OR	1	1	0	0	1	0	1	1	0	0	0	1
AND	1	1	0	1	1	0	1	1	0	0	0	1
CALL LESS THAN	1	1	1	0	0	0	0	1	0	1	0	1
CALL GREATER THAN	1	1	1	1	0	0	0	1	0	1	0	1

Control Unit Data Path Reference