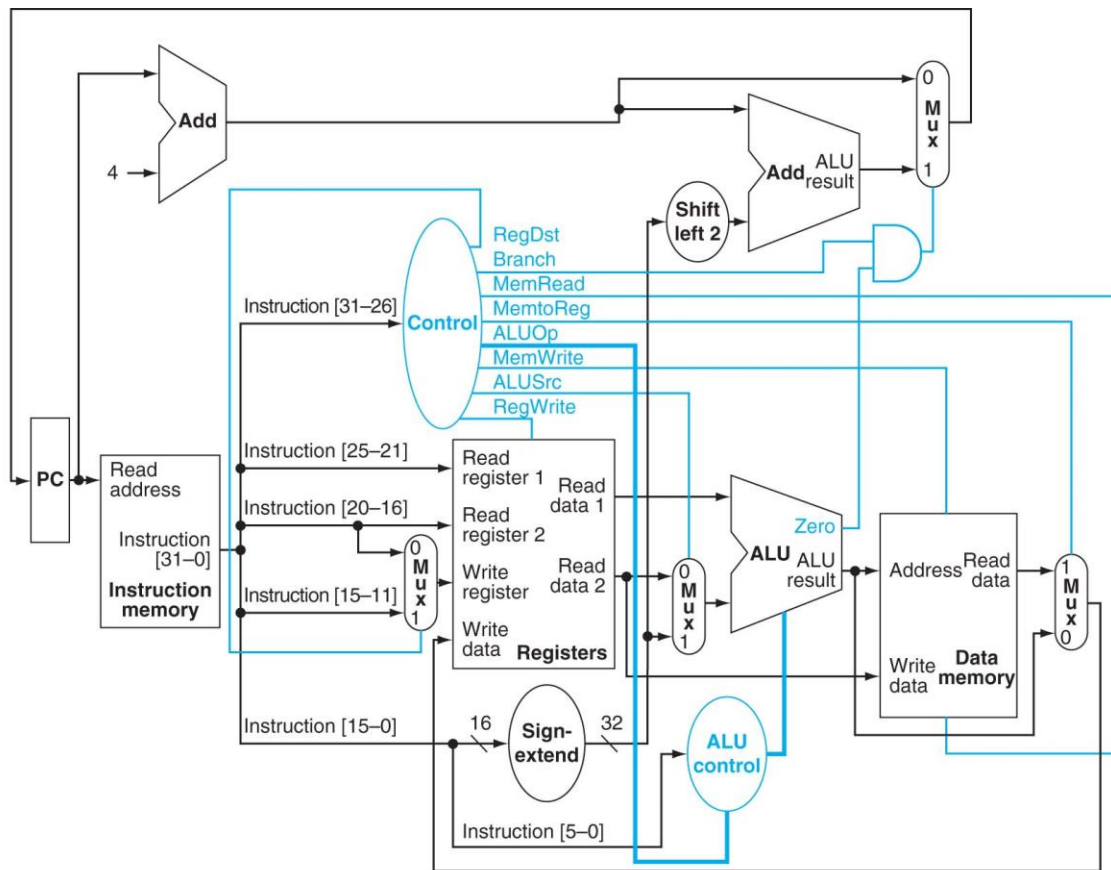


Design Project #3: Single-Cycle Datapath Design

(Team project – 2 person teams)

Your task is to create the simple datapath from Chapter 4. The specific implementation you will be using is shown below:

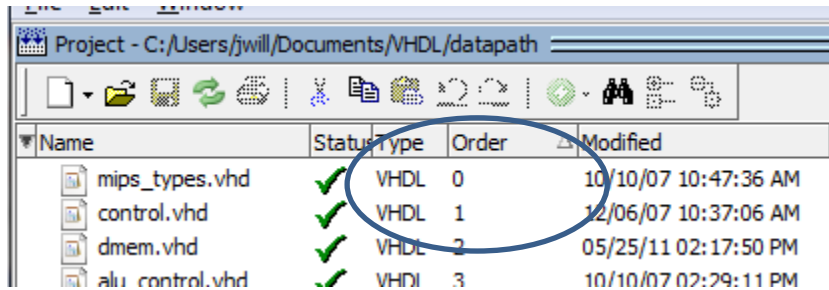


You will be using your own ALU from the previous assignment for the ALUs in the datapath. The other functional units will be given to you. Each functional block will be in a separate .vhd file. They are as follows:

- alu_control.vhd : combinational logic that creates the control lines to the ALU
- control.vhd : combinational logic that creates the control signals for the datapath
- dmem.vhd : data memory
- imem.vhd : instruction memory
- mips_types.vhd : package of types that will be used for this design
- mux2.vhd : 2x1 mux for two 32-bit buses

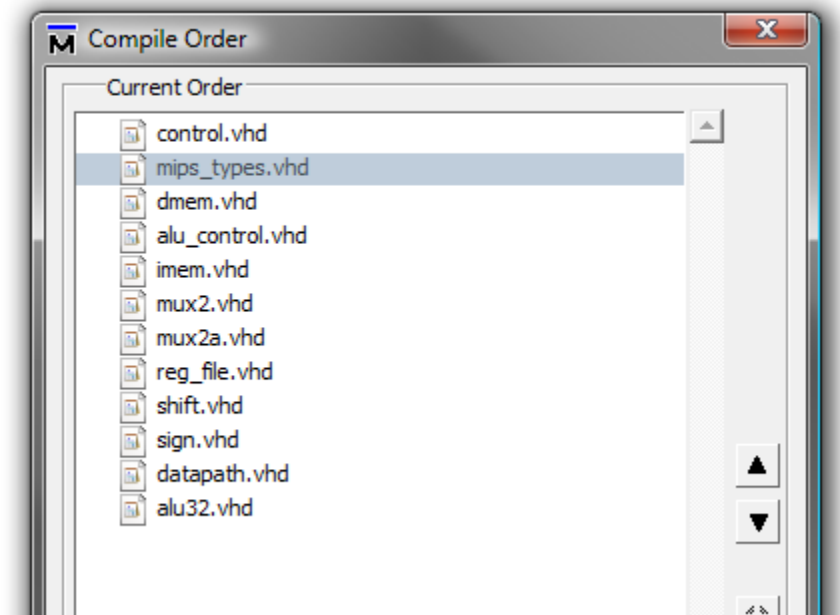
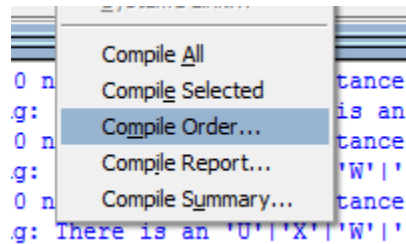
- mux2a.vhd: 2x1 mux for two 5-bit buses
- reg_file.vhd : register file
- shift.vhd : combinational logic that shifts a 32-bit word 2 places to the left
- sign.vhd : combinational logic to sign extend a 16-bit number to a 32-bit number

Add each of the files to your project. Make sure that the mips_types.vhd is compiled first:



Name	Status	Type	Order	Modified
mips_types.vhd	✓	VHDL	0	10/10/07 10:47:36 AM
control.vhd	✓	VHDL	1	12/06/07 10:37:06 AM
dmem.vhd	✓	VHDL	2	05/25/11 02:17:50 PM
alu_control.vhd	✓	VHDL	3	10/10/07 02:29:11 PM

To change the order...



Though these functional blocks are given to you, a few of them will require your modification. Look for the comments sections in each of these to see what you must modify. The files to modify are:

CONTROL.VHD: Modify the output signals for the different types of instructions.

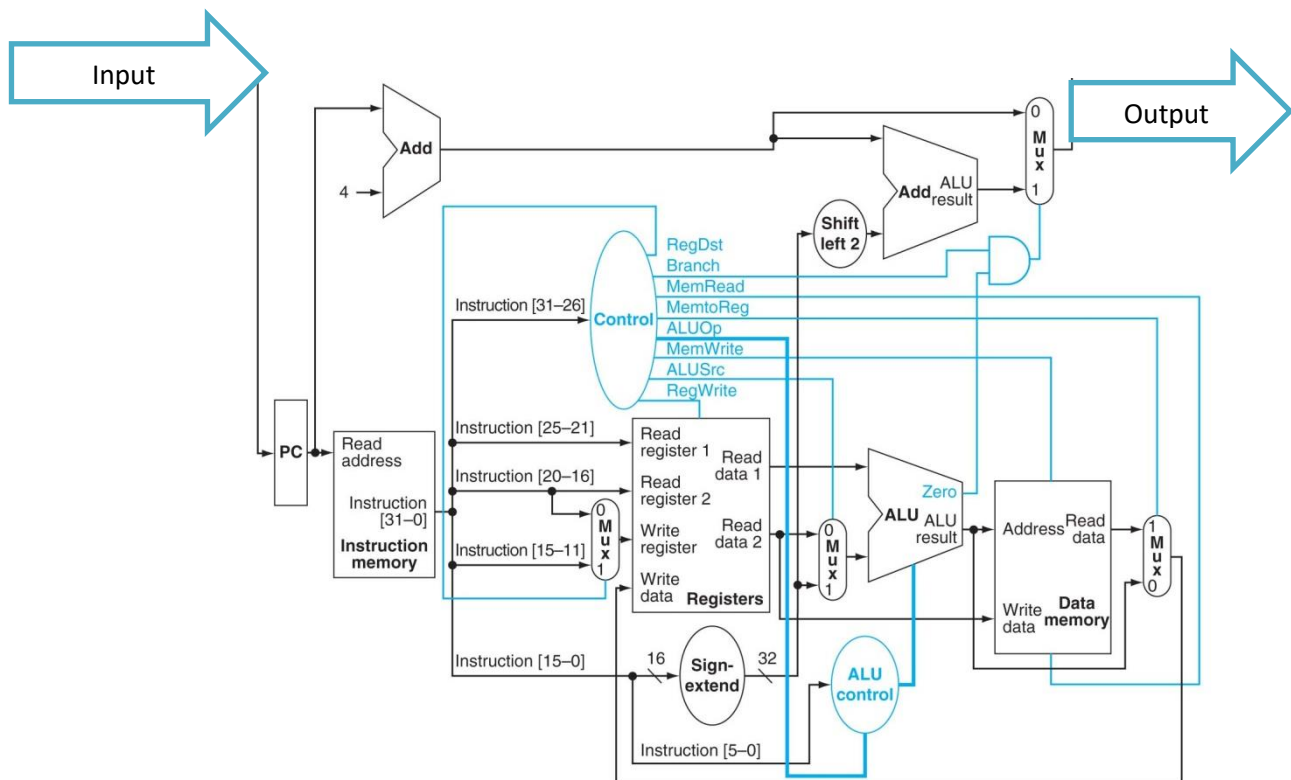
DMEM.VHD: You will need to initialize the contents of the data memory. The contents for the first part of the assignment are already initialized, but you will later change these.

ALU_CONTROL.VHD: Fill in the output values for the conditions of branch instruction, mem-type instruction, and R-type instruction.

IMEM.VHD: This is where your hand-assembled code will go. You will need to fill in the values of instruction memory with the appropriate 32-bit instructions.

In creating the datapath for this assignment, a key task will be for you to read the input/output specification for each of the functional units, but you do not need to fully understand how they work. Create your own “datapath.vhd” file. This file will contain lots of component declarations. It will also have to have quite a few signal declarations. In fact, you will probably have a signal for each of the internal wires shown in the datapath.

To be able to simulate this datapath, we will need to have at least one input and one output to our VHDL system. We will thus think of the datapath as follows:



For instance, your declaration of the datapath entry may look something like this:

```
ENTITY datapath IS

    PORT ( pc : IN STD_LOGIC_VECTOR ( 31 downto 0 );
          nextPC: OUT STD_LOGIC_VECTOR (31 downto 0) );

END datapath;
```

In defining the architecture of your datapath, you will have many component declarations. You will have a component declaration for every block of the datapath shown above. For example, you will have a large block declaring components such as:

```
COMPONENT alu_control
    port( INSTR: in std_logic_vector(5 DOWNT0 0);
          ALUOP: in std_logic_vector(1 DOWNT0 0);
          ALUCTRL : OUT std_logic_vector(3 DOWNT0 0));
END COMPONENT;

COMPONENT control
    port( INSTR: in std_logic_vector(5 DOWNT0 0);
          REGDST: OUT std_logic;
          BRANCH: OUT std_logic;
          MEMREAD: OUT std_logic;
          MEMTOREG: OUT std_logic;
          ALUOP: OUT std_logic_vector(1 DOWNT0 0);
          MEMWRITE: OUT std_logic;
          ALUSRC: OUT std_logic;
          REGWRITE: OUT std_logic);
END COMPONENT;

COMPONENT dmemory
    port(
        DIN: in mips_data;
        ADDR: in mips_address;
        DOUT: out mips_data;
        WE, RE: in std_logic;
        CLK: in std_logic
    );
END COMPONENT;

COMPONENT imemory
    port(
        ADDR: in mips_address;
        DOUT: out mips_data
    );
END COMPONENT;
```

Since the data memory and the register file need a clock signal, you must add an input signal, **sys_clock**, to your model. Use the following code for this:

```
SIGNAL sys_clock : std_logic := '0';
CONSTANT Tcycle : time := 100 ns;

BEGIN

    -- create clock process
    clk_gen: process
    begin
        sys_clock <= '1' after Tcycle/3, '0' after Tcycle;
        wait until sys_clock = '0';
    end process clk_gen;
```

When you do your PORT MAP of the **register file** and **data memory**, use “**sys_clock**” as the final argument (this is what the component expects for the “CLK” input).

Use the following program to test the operation of your datapath:

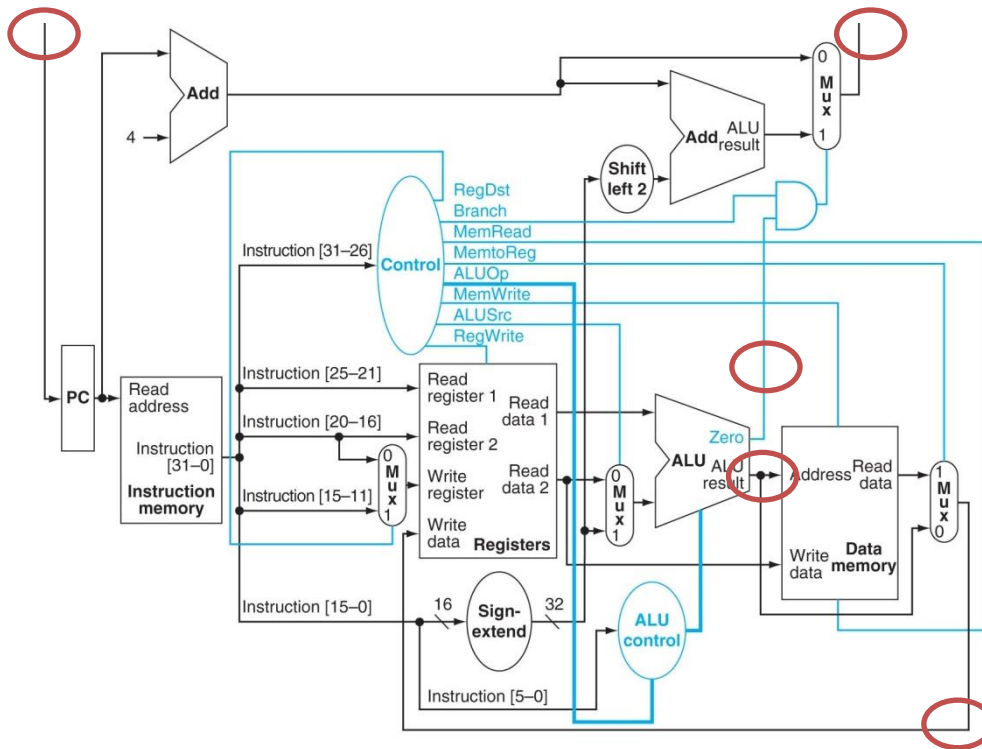
```
add $1,$0,$0      #$1=0
lw  $2,0($1)      #$2=M[0]
lw  $3,4($1)      #$3=M[1]
and $4,$2,$3      #$4=M[0]&M[1]
beq $4,$0,SKIP    #branch to SKIP if M[0]&M[1]==0
sw  $4,8($1)      #store M[0]&M[1] in M[2]
SKIP sw $4,12($1)  #store M[0]&M[1] in M[3]
```

To verify the program you will need to hand assemble the above code and place the resulting machine code in the instruction memory (imem.vhd) starting at address 0. Test your program and datapath on 2 sets of data in memory: when the first four memory locations have the following values: 5, 6, 7, 8 and then when they have the values: 1, 2, 3, 4.

When you create your force file for this program, you will have to manually set the values for the PC given the correct sequencing. Note that when the data memory has the values 5, 6, 7, 8, then the branch is not taken, but it will be when data memory has the values 1, 2, 3, 4. Therefore, the PC input to your simulation would be 0, 4, 8, 12, 16, 20, 24 in the case where the data memory has values 5, 6, 7, 8. Your input to the simulation would be 0, 4, 8, 12, 16, 24 in the latter case, since the `sw $4, 8($1)` instruction is skipped. Note that the value for the nextPC should match the values that you’ve hard coded for the following instruction.

To turn in:

Show the waveforms for the following signals



On your waveforms, show only the values of the five lines circled above. Their names should be:

PC
ALUResult
Zero
NextPC
WriteData

*****Use the exact names for the signals above *****

Show the waveforms for the two different cases. You will need to *write these values into the data memory* for each trial. Note the flow of the code for the given values. In one case the branch should be taken. In the other case, it should not. Your PC input should represent the values the program counter would have as it executes the code.

Case1: Data memory contains 5,6,7,8

Case2: Data memory contains 1,2,3,4

Since your incremented PC+4 isn't connected (nor is the branch target) back, you will have to figure out what the PC sequence *should* be and then use this in your force file. In one of the Cases the branch is taken, and in the other the branch fails and the PC just goes to the next instruction. You need to program this behavior into your force file.

REMINDER: You have a NextPC value (which would normally loop back). This value should always match the value you programmed in the force file for the following instruction, as if the NextPC signal had looped back around to determine the next value in the PC register.

Print them each out on a separate page. Attach the cover sheet to the first page.

ECE 424

DUE: Friday, Dec 9, 11:30 pm

Design Project #3: Single-Cycle Datapath Design (100 points)

COVER SHEET

Honor Code: _____

Name: _____ Signature: _____

Honor Code: _____

Name: _____ Signature: _____

A complete assignment will contain:

1. This cover sheet
2. A printout of your VHDL code including your main file, modified imem.vhd, modified alu-control.vhd, and modified dmem.vhd.
3. Waveforms for the two different cases, showing the five (and only the five) signals in hex format.