Computer Organization, Spring, 2017

Lab 4: Pipeline CPU

Due: 2017/06/04 11:59pm

1. Goal:

Modifying the CPU designed in lab3 and implementing a simple version pipelined CPU.

2. HW requirement:

a. Please use Xilinx as simulation platform.

b. Please attach your name and student ID as comments at the top of each file.
PLEASE FOLLOW THE FOLLOWING RULE! Zip your folder and name it as "ID.zip" (e.g., 0416001_0416002.zip) before uploading to e3. Multiple submissions are accepted, and the version with the latest time stamp will be graded.

Note: You must be uploading the ".zip" file to e3.

c. Pipe_Reg.v, Pipe_CPU_1.v and TestBench.v are supplied.

d. In the top module, based on your design, please accordingly change the following *N* to the value which is the total size (length) of input signals (including data and control) entering each set of pipeline registers.

Pipe_Reg #(.size(N)) ID_EX

e. And notice that, when using CO_P4_test_1 and CO_P4_test2, these lines in Data_Memory.v should be command (shown in the following picture). However, these will be used in CO_P4_test_data3.

```
initial begin
    for(i=0; i<128; i=i+1)
        Mem[i] = 8'b0;
    /*Mem[0] = 8'b0100;
    Mem[4] = 8'b0101;
    Mem[8] = 8'b0110;
    Mem[12] = 8'b0111;
    Mem[16] = 8'b1000;
    Mem[20] = 8'b1001;
    Mem[24] = 8'b1010;
    Mem[28] = 8'b0010;
    Mem[32] = 8'b0001;
    Mem[36] = 8'b0011;*/
end
```

3. Requirement description:

a. Code (120%):

Basic instruction set (80%): Previous Labs' instructions, such as ADD, ADDI, SUB, AND, OR, SLT, LW, SW, BEQ, and MUL. And there is no JUMP in this Lab.

Advance (40%): Hazard Detection + Forwarding

Need to stall pipelined CPU if it detects load-use.

Need to forward data if instructions have data dependency.

Basic Instructions + BEQ + BNE. Besides, neither BLE, BLT nor J will appear in any test case.

b. Testbench:

Please use the tested pattern CO_P4_test_1.txt to test basic instruction, CO_P4_test_2.txt to test hazard detection and forwarding.

```
//CO_P4_test_1.txt
Begin:
addi  $1, $0, 3;      // a = 3
addi  $2, $0, 4;      // b = 4
addi  $3, $0, 1;      // c = 1
sw    $1, 4($0);      // A[1] = 3
add   $4, $1, $1;     // $4 = 2a
or    $6, $1, $2;     // e = a | b
and   $7, $1, $3;     // f = a & c
sub   $5, $4, $2;     // d = 2a - b
slt   $8, $1, $2;     // g = a < b
beq   $1, $2, Begin;  // if b==h goto Begin
lw    $10, 4($0);     // i = A[1]
```

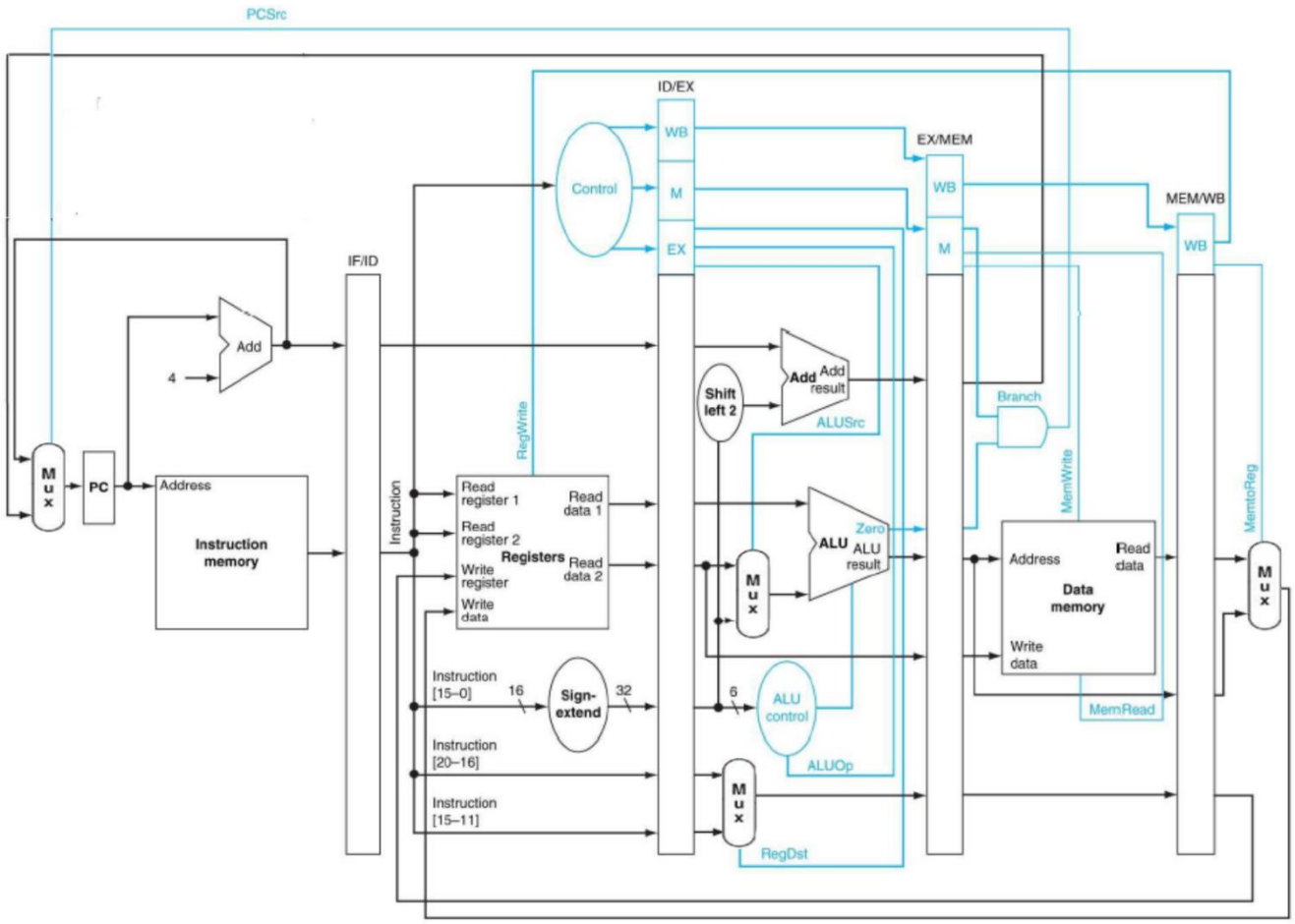
```
//CO_P4_test_2.txt
addi  $1, $0, 16
addi  $2, $1, 4
addi  $3, $0, 8
sw    $1, 4($0)
lw    $4, 4($0)
sub   $5, $4, $3
add   $6, $3, $1
addi  $7, $1, 10
and   $8, $7, $3
addi  $9, $0, 100
```

c. Report (20%):

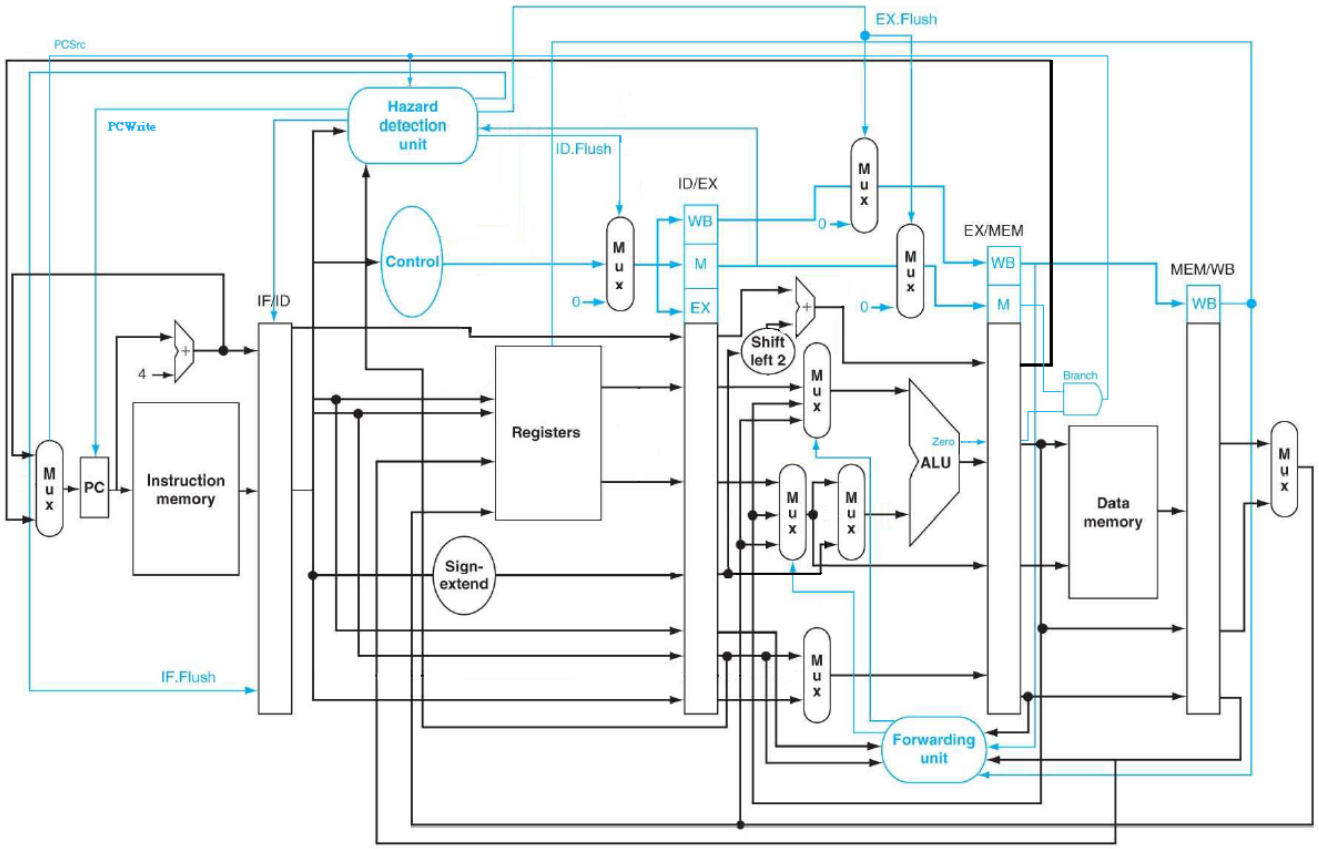
The context must include:

1. Your architecture
2. Your implementation and hardware module analysis
3. Finished part
4. Problems you met and solutions
5. Summary

4.Basic Architecture (for reference only):



5. Advance Architecture (for reference only):



6. Grade

a. Total score: 140% COPY WILL GET 0!!
b. Basic score: 80%
c. Advance score: 40%
d. Report: 20%
e. Delay: 10% off / day

7. Hand in your Assignment

Please upload the assignment to the E3.
Put all of .v source files and report into same compressed file. (Use your student ID to be the name of your compressed file and must have the form of “student IDs”. Ex. 0416001_0416002.zip)

8. Q&A

If you have any question, use E3 discussion or just send email to TAs.