Computer Organization, Spring, 2017

Lab 4: Pipelined CPU

Due: 2017/6/2
Demo: 2017/6/7

1. Goal:

In this Lab, we add pipeline registers to hold information produced in previous cycle. Please modify your single cycle CPU designed in lab3 to implement a simple version 5-stage pipelined CPU.

2. HW requirement:

(1) Please use Xilinx ISE as your HDL simulator.

(2) Please attach your names and student IDs as comments at the top of each file.

(3) Pipe_Reg.v, Pipe_CPU_1.v and Testbench.v are supplied.

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

Pipe_Reg #(.size(N)) ID_EX

(5) When using CO_P4_test_1 and CO_P4_test_2, these lines in Data_Memory.v should be commented as the following picture shown. However, these will be used in CO_P4_test_3 (bubble sort with only BEQ as its branch instruction).

```
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
```

(6) Basic instruction set (80%): There are only ADD, ADDI, SUB, AND, OR, SLT, SLTU, SLL, SLLV, LUI, ORI, LW, SW, and MUL in basic instruction set this time.

(7) Advanced (40%): Hazard Detection + Forwarding
Need to stall pipelined CPU if it detects load-use hazard.
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.

(8) Test cases: Please use CO_P4_test_1 to test partial basic instructions, and use CO_P4_test_2 to test hazard detection and forwarding. In addition, when you have wrong result after testing CO_P4_test_3 (bubble sort), you should translate it from machine code to MIPS instructions by yourselves (you have already practiced translating MIPS instructions to machine code in Lab3, so this time try the inverse direction)!

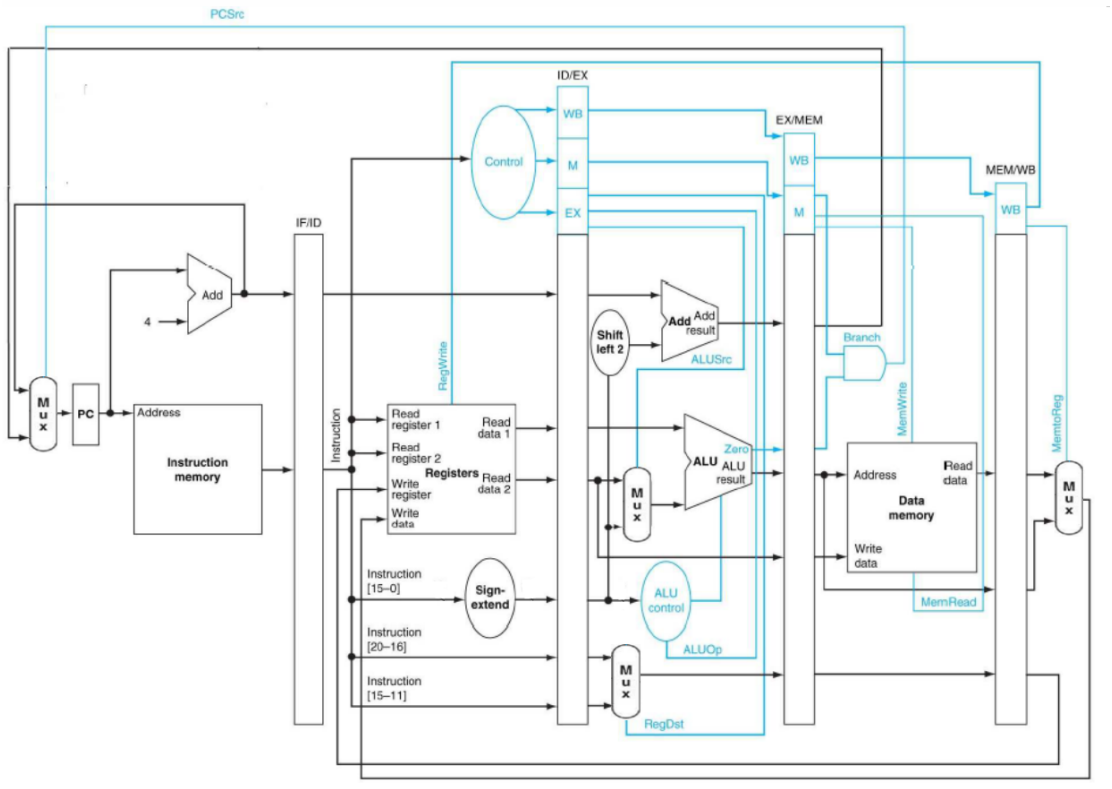
CO_P4_test_1.txt

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

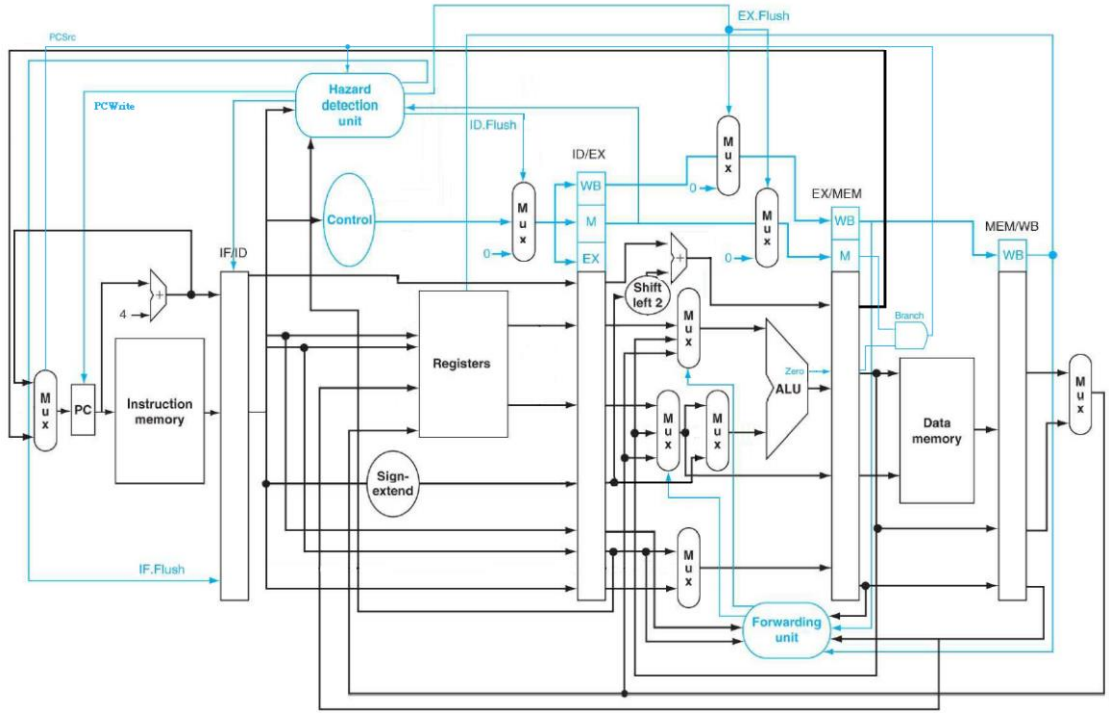
CO_P4_test_2.txt

```
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
```

3. Basic architecture (for reference only):



4. Advanced architecture (for reference only):



5. Grade

(1) Total: 140 points (basic: 80 points, advanced: 40 points and report: 20 points)

(2) Late submission: 10 points off per day

(3) No plagiarism, or you will get 0 point

6. Hand in

Please follow the following rule! Zip all of your Verilog files and report into a folder and name it as "ID1_ID2.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. Other filenames and formats such as *.rar and *.7z are NOT accepted!

7. Demo

Time: 2017/6/7

Please review your code from lab 1 to lab 4. All the questions are relative to the labs. You will get a grade after the demo, and it will be one part of your lab score.

8. Q&A

For any questions regarding Lab 4, please contact 黃甯琪 (blackitty321@gmail.com).

This lab is difficult! please work on it as soon as possible!