Computer Architecture Project 1

Team 18

1. Members and Teamworks

School ID.	Name	Works	*.v
B03902059	紀典佑	Hazard Unit Forward Unit Memory Unit Report	Hazard_Unit.v Forward_Unit.v Data_Memory.v
B03902089	林良翰	Single Cycle CPU Pipelined CPU Control Unit Multiplexers ID/WB hazard fix Debugging	CPU.v IFID.v IDEX.v EXMEM.v MEMWB.v Control.v MUX5.v MUX8.v MUX32.v MUX32_3.v Equal5.v
B03902101	楊力權	Instruction Fetch Register Unit ALU Unit Adder Unit Sign Extender Shift, AND, OR Equal Unit Debugging	Instruction_Memory.v Registers.v ALU.v ALU_Control.v Adder.v Sign_Extend.v Shift_Left_Imm.v Shift_Left_Jump.v Equal32.v

2. We implement this pipelined CPU with 4 phase:

a. Determine what modules we need and connect them in CPU.v.

b. Write modules except for control unit, hazard detection unit and forward unit.

c. Write the control unit, hazard detection unit and forward unit.

d. Debug with printing out values of all units, wires and registers.

3. Implementation of each module:

a. Instruction_memory.v

Let instruction output=memory[addr_input >> 2] (multiply 4).

b. Registers.v

Let output register data = register[addr_input(5 bits)] and use always to see if RegWrite bit is set, if set let register [write_addr] = input data.

c. ALU.v

Use always @(*) to determine what ALU control bit is and let result equals corresponding operation.

d. Data_Memory.v

Determine whether MemRead bit is set, if set let output data = memory[addr_input]. And use always @(posedge clock) to see whether MemWrite bit is set, if set let memory [addr _input] = data.

e. PC.v:

Use always @(posedge clock or negedge restart) to determine which PC output we need. If restart is set, let output = 32'b0, if hazard bit is set let output = last pc (use register to store). Else if start bit is set, let output = pc_input.

f. Adder.v

Use assign to let output = data1 + data2.

g. Sign_Extend.v

Use trivial assign to let output = {16'h0000, data_input} or {16'hffff, data_input}.

h. Shift_Left_Imm.v

Use assign to let output = { input[29:0], 2'b00}.

i. Shift_Left_Jump.v

Use assign to let output = {input, 2'b00}.

j. Equal32.v, Equal5.v

Use always to see if data input1 is equal to input2. (32 and 5 bits)

k. And.v

use trivial assign to let ouput = (in1 && in2)? 1 : 0.

l. Merge.v

Use assign to let ouput = { input2(4-bits), input1(28-bits)}.

m. Mux32.v, Mux8.v, Mux5.v

Use always @(*) and determine if select bit is set or not, if set let

output = input1 else output = input0.

n. Mux32_3.v

Use always @(*) and determine what select bits(2 bits) are, if select = 00 -> output = input0, else if select = 01 -> output = input1, else if select = 10 -> output = input2.

o. OR.v

Use trivial assign to let output = input1 | input2.

p. ALU_Control.v: assign output bits correspond to fuction_i and ALUOp_i.

q. IFID.v

Use always @(posedge clock) and determine whether flush bit is set, if set let inst and pc = 0 to flush the instruction. Else if hazard bit is set let output = last one (by register) to stall. Else just let output = input.

r. IDEX.v, EXMEM.v, MEMWB.v

Use always @(posedge clock) and set output correspond to each input.

s. CPU.v:

Connect according to page12 in project pdf and add forward ID stage.

4. Problem and Solution:

a. 連接 module 時，因為需要大量的 wire，每個 module 又有很多 input 與 output，會出現接錯的情況。

b. 因為判斷 equal 是在 ID stage 執行，所以我們在執行 branch 判斷時，會因為來不及 write back 而拿到錯誤的值。因此我們的解決方式為把要 write back 的值 forward 到 equal module 前。

c. 要控制 pipeline，所以必須在要控制 data flow 的 module 使用 posedge。

d. 在 testbench.v 印出所有 Unit 的 wire、register 的值來找出出錯的位置，並觀察 Pipeline 狀況。