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:**

1. 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。
2. 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。
3. 要控制pipeline，所以必須在要控制data flow的module使用posedge。
4. 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。