Computer Architecture Project1

Pipelined CPU using Verilog

1. 工作分配

   B05902053　　陳奕均　維護 spec，寫 report

   B05902071　　陳郁文　解釋架構，寫小部分 code

   B05902083　　余柏序　report 顧問，寫大部分 code

2. 實作方式

   (1) instruction -> ALU-control

   由原始的 opcode 可以決定各種 signal 和 ALU_Op

   R-type ALU_Op 都是 10，要看 function code

   其他類 instruction ALU_Op 可能是 00 或 01 (對應到加和減)

R-type					
opcode	instruction	funct7	funct3	ALU-action	ALU-control
0110011	and	0000000	111	AND	001
0110011	or	0000000	110	OR	010
0110011	add	0000000	000	ADD	011
0110011	sub	0100000	000	SUB	100
0110011	mul	0000001	000	MUL	101
I-type					
0010011	addi	X	000	ADD	011
0000011	lw	X	010	ADD	011
S-type					
0100011	sw	X	010	ADD	011
SB-type					
1100011	beq	X	000	SUB	100

欲決定 ALU_Ctrl

如果是 R-type，即 ALU_Op = 10

這時看 function code: funct_i = {inst[30], inst[25], inst[14], inst[13], inst[12]};

由下表得 ALU_Control

funct_i	ALU-action	ALU-control
00111	AND	001
00110	OR	010
00000	ADD	011
10000	SUB	100
01000	MUL	101

如果不是 R-type，看 ALU_Op 即可

(2) instruction -> Control Signal (with pipeline)

必須實作出下方這張表

		R-type	addi	lw	sw	beq
ID	Branch	0	0	0	0	1
EX	ALUOp	10	00	00	00	01
	ALUSrc	0	1	1	1	0
M	MemRead	0	0	1	0	0
	MemWrite	0	0	0	1	0
WB	RegWrite	1	1	1	0	0
	MemtoReg	0	0	1	X	X
opcode		0110011	0010011	0000011	0100011	1100011

原先 single cycle processor 可以直接 decode，直接傳 signal，但現在還要考慮 pipeline

一種做法是先把 signal decode 出來之後全部存在 pipeline register 中，逐步傳遞，不過我們採取作法的是從 Control 開始，每次都傳 opcode，一步一步往下走，當需要用到 Signal 在當場 decode 即可

比對 opcode 會發現只需要傳前 3 個 bit 就可以了

在 project 中實現如下:

opcode_ID => opcode_EX => opcode_MEM => opcode_WB (傳遞)

產生 Signal:

opcode_ID -> Branch Signal (Beq_Op in our code)

opcode_EX-> ALUOp, ALUSrc Signal

opcode_MEM -> MemWrite Signal

opcode_WB-> RegWrite Signal (regwrite_MEM, regwrite_WB in our code)

opcode_WB-> MemtoReg Signal (reg_src_WB in our code)

(3) Data Forwarding

考慮將前 2 個 stage 的結果預先傳回來

在 source code 中, RegWrite_p, RegWrite_pp 是前一個，前兩個 stage 的 RegWrite Signal

Rd_p, Rd_pp 是前一個，前兩個 stage 的 Rd 位址

Rs1_i, Rs2_i 是當前指令需要的 rs1 和 rs2 位址

比對位址後可以知道要不要以及如何做 forwarding，送給兩個 MUX32_3i 適當的選擇訊號，讓 ALU 拿到兩個正確的 input

(4) Data Hazard

由 Hazard_Detect 來做檢驗，當指令是 lw instruction，且後面的指令需要用到前面指令應寫入而尚未寫入的 register，讓 PC 無法寫入來 stall 一個 cycle

(5) Control Hazard

beq instruction 在 decode 的時候就檢查有沒有 equal (提早檢查可以少 flush 一個 instruction) 如果有, Beq 會被 set，送到 Control 模組，清空 IF/ID pipeline register，也會送給 PC_Mux 對應的選擇訊號，使 PC 能跳到該跳的位置

3. 其他細節

(1) ALU 的 Zero_o 原本是用來判斷 beq，但現在 beq 在 decode 的時候已經提早判斷，這個 port 已經用不到，所以省去

(2) Control 的 output port valid bit 用來判斷指令是否有效，超出 project1 範圍無效。或者當 beq 成立時會 flush IF/ID pipeline register，這時讀出的 instruction 會是 32'b0，也是無效的

(3) 0 號 register 不可寫入(Register.v)