Computer Architecture Project1 Report

Team number: 19

Members and team work:

B02901065 李洺曦

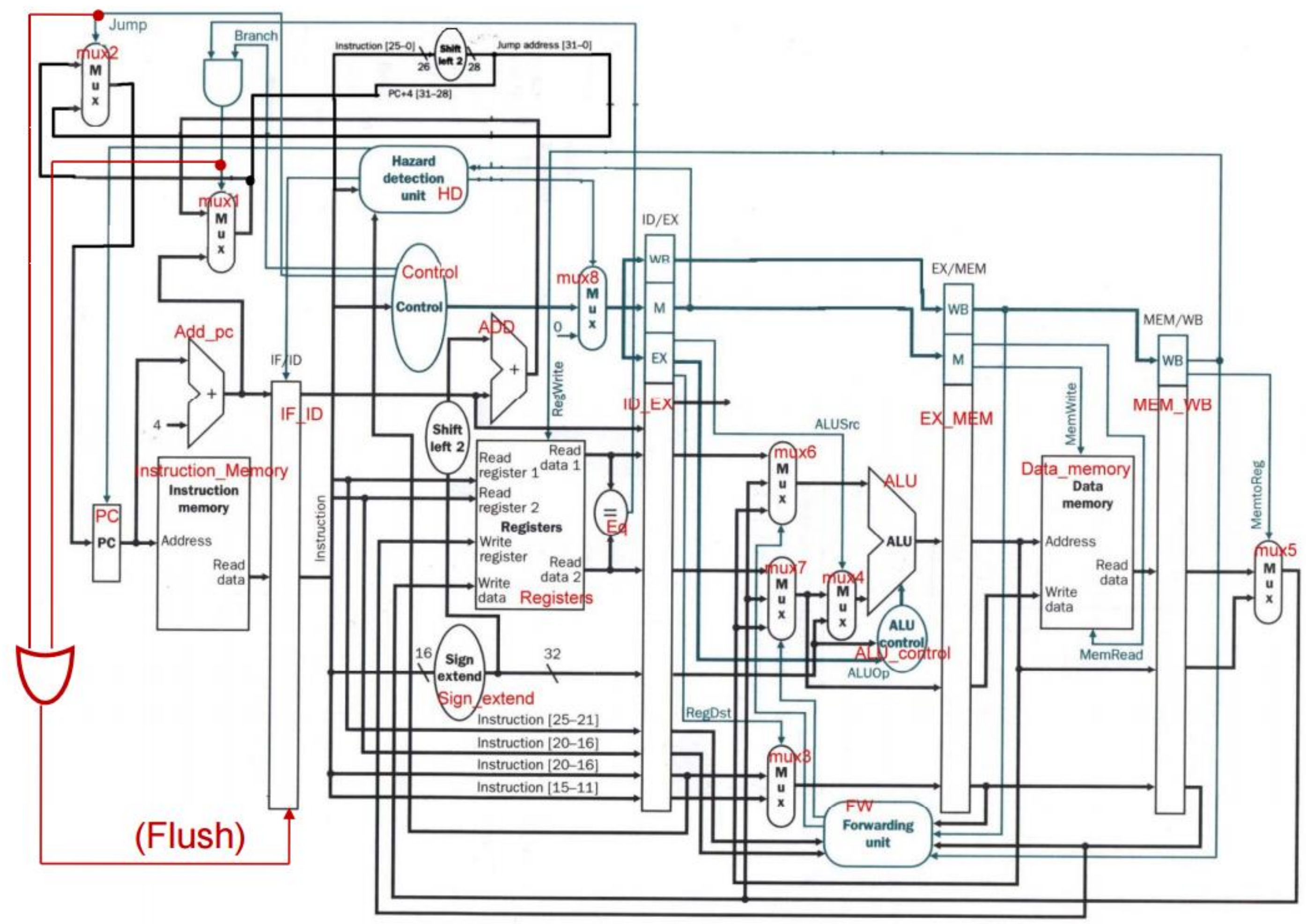
Data forwarding unit, Hazard detection unit, Debug and testing, Analyze relations between modules

B03611034 黃楷傑

Debug and testing, Write Report

How we implement this pipeline CPU:

1. Add latches between modules of different stage in order to save previous stage results and propagate the output to the next stage at rising edge of clock (taken the advantage of the characteristic that each stage takes 1 clock cycle.)
2. Modify some of the singled cycled CPU modules' input and output from and to the latches we newly created.
3. Add new modules new to hw4.
4. Build the hazard detection and data forwarding unit and set the control signal after understanding the relationships between each module.
5. For debugging, we traced through each instruction's stages, and observe value of each cycle and the waveform, then when we find the bug, we head to the original .v file of the related modules to fix the problem.



Module implementation explanation:

1. Adder.v

- Output sum of two 32-bit inputs.

2. ALU.v

- Do the calculation of the two 32-bit inputs according to the control signal from ALU_Control.v and output the result.

- The truth table of our design is showed below:

ALUCtrl_i	010	110	001	000	100	Others
Operation	add	sub	or	and	mul	32’bx

3. ALU_Control.v

- Take ALUop and funct field from Control.v as input and output ALU control signal to ALU.v according those two inputs.
- The truth table of our design is showed below:

MIPS	I-addi	R-add	R-mul	R-sub	R-and	R-or	Others
ALUOp_i	01	11	11	11	11	11	Others
funct_i	x	0000	1000	0010	0100	0101	Others
ALUCtrl_o	010	010	100	110	000	001	3'bx

4. Control.v

- Set the main control signals according to the first 6 bits of the instruction (instruction opcode).
- The truth table of our design is showed below:

MIPS	R-type	I-addi	I-lw	I-sw	I-beq	J-jump	Others
Op_i	000000	001000	100011	101011	000100	000010	Others
Jump	0	0	0	0	0	1	0
Branch	0	0	0	0	1	0	0
RegDst	1(Rd)	0(Rt)	0(Rt)	x(noRW)	x	x	x
ALUOp	11	01	01	01	xx	xx	xx
ALUSrc	0	1(SnEx)	1(SnEx)	1(SnEx)	x	x	x
MemWrite	0	0	0	1	0	0	0
MemRead	0	0	1	0	0	0	0
RegWrite	1	1	1	0	0	0	x
MemtoReg	0	0	1	x(noRW)	x	x	x

5. CPU.v

- Connect all modules together according to the design provided project1 document.

6. Multiplexors (MUX5.v, MUX8.v, MUX32.v)

- Select one 5,8,32-bit input from two 5,8,32-bit inputs.

7. Multiplexor (MUX32_3.v)

- Select one 32-bit input from three inputs.

8. Sign_Extend.v

- Extend a 16-bit input into a 32-bit output.

9. Data_Memory.v

- Read from or write to the desired address according to control signal Memwrite and MemRead generated by control unit.
- The input Address_i is shifted 2-bit rightward to get word-based address.

10. Latches (IFID.v, IDEX.v, EXMEM.v, MEMWB.v)

- Save control signal from previous stage and output to the next stage at clock rising edge.
- In IFID, the refreshment of outputs is allowed only when the IDIF_i is asserted.

11. FowardingUnit.v

- The outputs, Data1Ctrl_o and Data2Ctrl_o, which choose the ALU input from forwarding data and data from register will always be 2’b00 unless the two if statements, the hazard-occuring condition, are true:

If (EX/MEM.RegWrite and (EX/MEM.RegisterRd!=0) and (EX/MEM.RegisterRd == ID/EX.RegisterRs)) ForwardA = 10;

If (MEM/WB.RegWrite and (MEM/WB.RegisterRd!=0) and (MEM/WB.RegisterRd == ID/EX.RegisterRs) and !(ForwardA == 10)) ForwardA = 01;

12. HazardDetector.v

- The HazardDetector will stall the pipline when the destination register of lw instruction is needed by following instruction.

  If (ID/EX.MemRead and (IF/ID.RegRt == ID/EX.RegRt or IF/ID.RegRs == ID/EX.RegRt)) {IFID_o, PCWrote_o, IDEXFlush_o} = 001;

  Else {IFID_o, PCWrite_o, IDEXFlush_o} = 110;

13. L2Shifter.v

- Take 32 bit input from Sign_Extention.v and shift the bits to the left by two bits.

14. Registers.v

- Write the input to the register at rising edge, output the value of register at falling edge.

15. PC.v

- Propagate the new PC if pcWrite_i is asserted.

16. testbench.v

- Modify the testbench to count stall and flush correctly (see Problems and solutions section for more.)

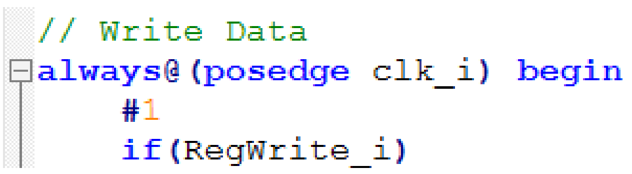
Problems and solutions:

1. **當 Simulate 開始執行時，PC = 4 始終無法在 posedge of clk 時寫入新的 PC output：**

   原因是，在第 0 個 cycle 時，Instruction Memory 的 output 尚未被寫進 IFID latch 裡，造成 IFID latch 沒有任何信號輸出，首先造成 Control unit 沒有辦法正常判斷輸出，而無法給出 jump 與 branch 兩個的信號，使得下一個 PC 路徑上的 nextPC MUX (選擇 PC+4 或者是 branch address) 以及 PC_In MUX (選擇 nextPC 的質或者是 jump address) 皆無法正常輸出 PC 的值，要解決這個方法，我們在 initial (start 信號尚未跳起) 的階段時賦予 IFID 一個 null instruction，經查閱 MIPS 指令表後，我們選擇將 Opcode 設為 111111，其餘 26 bits 為 0，如此便能使 Control 能正常判斷，並且輸出 branch 與 jump 的信號，除此之外，HazardDetector 也同樣會影響到 PC MUXs 的輸出，因此在 initial 的階段，我們使 HazardDetector 輸出 {IFID_o, PCWrite, IDEXFlush_o} = 111 的信號，如此一來可以使 IFID 以及 PC 能夠正常被刷新，並且使得 IFID 裡面的 null instruction 在 start 跳起後的下一個 posedge of clk 不被傳入 ID/EX latch 中。當然，我們也可以將 Control 以同 HazardDetector 的方式進行改寫，使一開始信號正常輸出。

2. **如何使 MEM/WB 的 input propogation 與 Register 的寫入在同一個 posedge of clk 中執行？**

   在 Register 中的寫入加上一個單位時間的 delay：



```
// Write Data
always@(posedge clk_i) begin
    #1
    if(RegWrite_i)
```

   使 Register 等到 MEM/WB 更新了 output 後再進行存入。

3. **如何更改 testbench.v 使我們的 output 能和助教的相符：**

由於我們在 initial 階段的改寫，如果只是單純做下列判斷

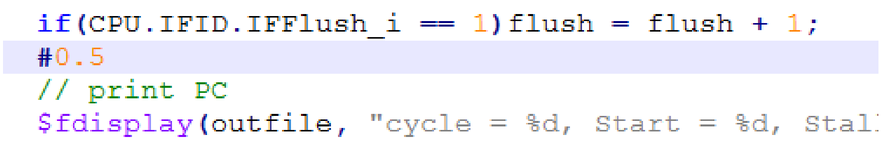
If (HazardDetector.IDFlush and Control.Jump != 0 and Control.Branch != 0)

會使得 stall 在 start 尚未跳起前就+1，因此我們在 testbench 裡新增一個變數 init 為 1，當符合上述條件但 init 為 1 時，不做 stall 的增值而是將 init 減 1，當符合上述條件且 init 不為 1 時，再將 stall 增值。

Flush 數增值的判斷為：

If (CPU.IFID.IFFlush_i == 1) flush = flush + 1;

而我們在寫入 output 時加入了 0.5 個單位時間的 delay，如下圖：



```
if(CPU.IFID.IFFlush_i == 1)flush = flush + 1;
#0.5
// print PC
$fdisplay(outfile, "cycle = %d, Start = %d, Stal
```

是因為一開始的 clk 為升起，如此會造成 output 中 PC=0 前還存在著 PC=x 的 null state，要消除此方法則是等 clk 升起後所有動作皆做完了進行記錄。而不超過 1 是因為要和助教的參考答案相符，而不將該 posedge of clk 之中的 register 寫入動作記錄起來。

4. **always statement 的誤解：**

一開始我們誤會了 always@()括號中所代表的意思，我們將其誤解為邏輯 true false 的判斷，以為將括號內輸入恆真的信號 always 便會一直執行，結果在 forwarding unit 中出現錯誤，後來知道括號中代表的意思是「信號的改變」，信號的改變會觸發 always statement 的執行，我們將所有的 input 信號 or 起來寫進括號中後，所有的程式皆正常執行了。雖然是對 verilog 的不熟悉所造成的，但這是我們 de 最久的 bug。