EE2011 Computer Organization 二乙
Final Exam Solution
Open Book, 06/15/2021

教授：林文彥
Dept. EE, Chang Gung University

109 學年 第 2 學期

學號：_____ 姓名：_____

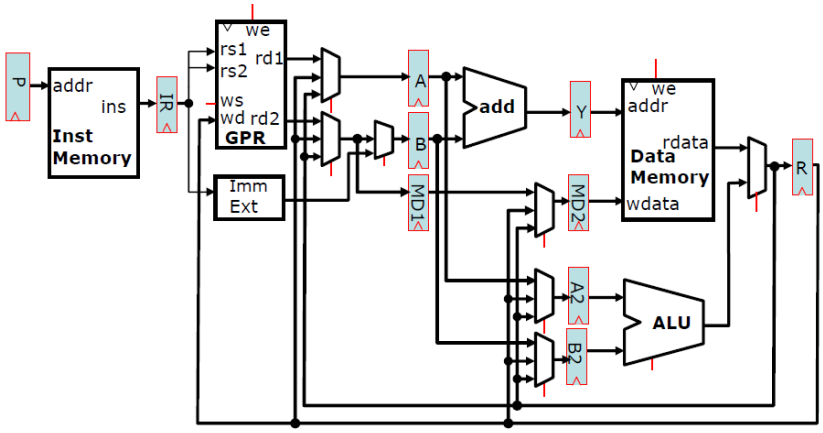
※ 若試卷答題空間不足，請另用紙張作答。作答完畢後，掃描或拍照上傳！

Ben Bitdiddle is designing a 32-bit MIPS processor that implements the ISA that has been covered in lecture. He starts off with the 5-stage datapath shown in the class handout: L10 as a baseline for his design, and then tries to optimize this pipeline.

Ben thinks that the **load-and-use** data hazard still requires one cycle of processor stall is because that when the "use" instruction in the pipeline requires the value of its operand in the Execute stage (3rd stage), the previous "load" instruction which supposes to generate the operand required by the next "use" instruction is still in the Memory stage (4th stage) and at that time, the operand value is not yet available for the "use" instruction in the Execution stage (3rd stage). Hence, Ben suggests replacing the ALU with a simple adder in the Execute stage (3rd stage) and moving the original ALU from the Execute stage to the Memory stage (4th stage) and try to make the data forwarding can work for load-and-use data hazard to prevent the processor stall in the original pipelined processor design.

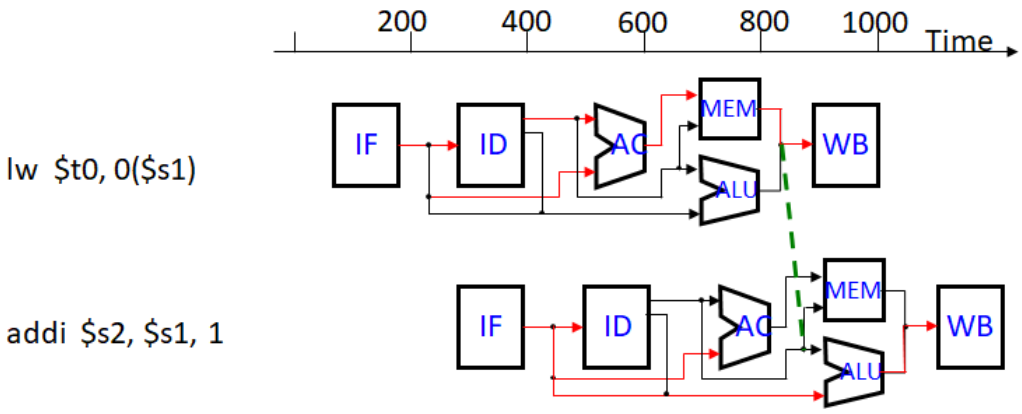
In this new datapath, the 3rd stage (formerly Execute) now contains only an adder to do address calculations if it is a load/store instruction. Otherwise, the data is simply forwarded to the 4th stage. The ALU will now run parallel to the data memory in the 4th stage of the pipeline (formerly Mem). During load/store instructions, the ALU is inactive. During ALU operations, the data memory is inactive. The datapath of the new pipeline is shown in the following figure. The pipeline control logic signals and the program-counter logic are not shown.

IF	ID	AC	EX/MEM	WB
Instruction fetch	Instruction decode and register read	Address calculation	ALU execution and memory access	Writeback to register file



(a) (15%) Please draw the diagram similar with the one shown in the p.19 of class handout L9 (i.e. p.266 of the textbook) and make an example of the “load-and-use” instruction sequences to illustrate that the load-and-use data hazard now can use data forward to prevent the pipeline stall.

[Solution:]



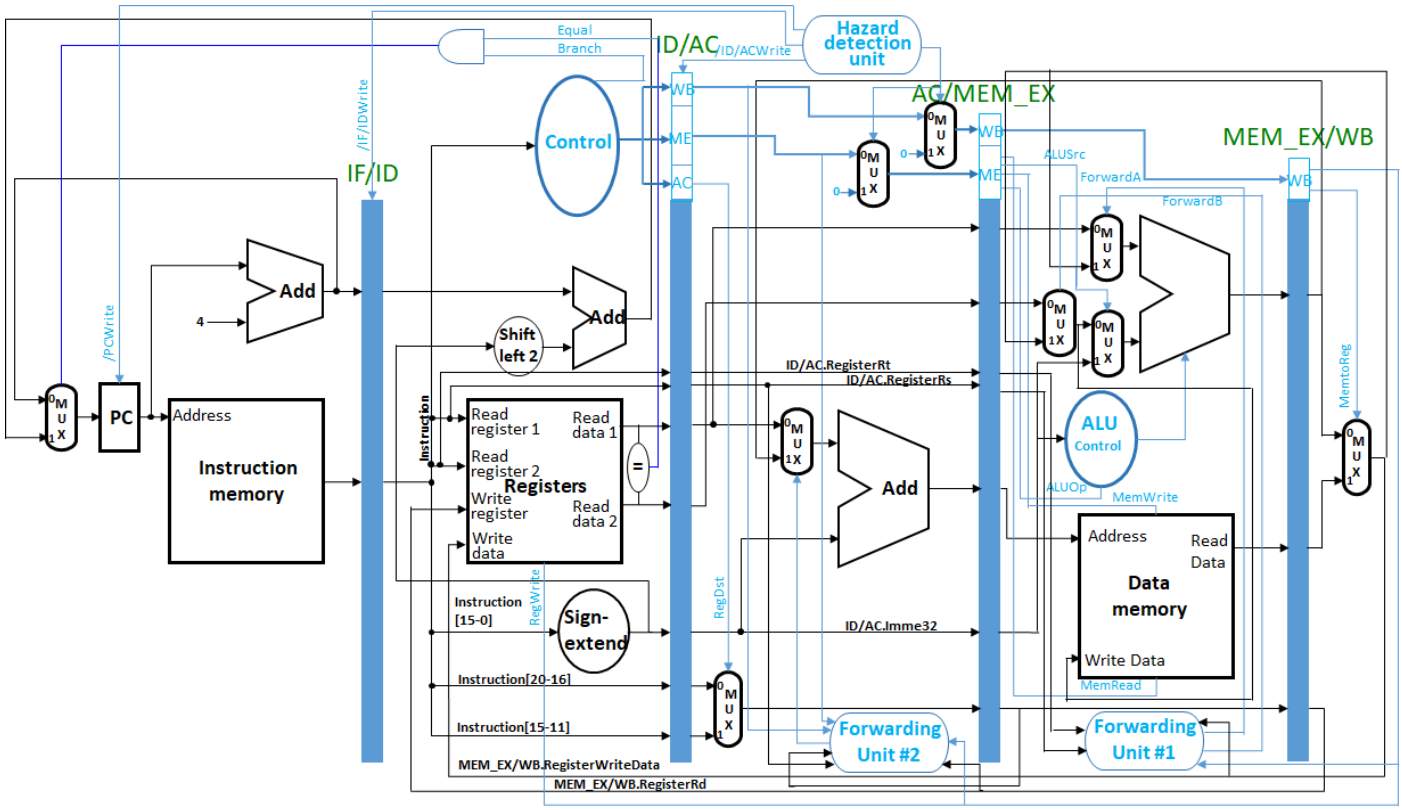
(b) (15%) Ben is extremely happy with his new implementation. He believes his new datapath is clearly better than the original. Is he right? What new hazard is introduced with Ben's modification? Do not consider jumps or branches for this part. Give an example of a set of instructions where a bubble is created in the new datapath that did not exist in the original. Please use five or fewer instructions in your example. Compare the advantages and disadvantages of Ben's modifications.

[Solution:]

```
addi    $s1, $s1, 4
lw      $t0, 0($s1)
```

(c) (40%) Draw a complete diagram of this new pipeline design with datapath, control logic signals and the program-counter logic in it. Hint: The diagram could be a combination of Fig. 4.51, Fig. 4.56, and Fig. 4.60 of the text book or the figures shown in the handout of L11, p.6., p.15, and p.21. That is, it should contain the program-counter logic, all the datapath, control logic signals, as well as the Hazard detection unit and forwarding unit.

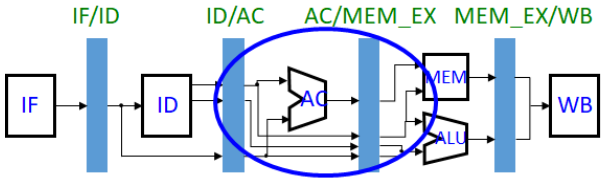
[Solution:]



(d) (30%) Now, how can you detect all the data hazards and generate the proper forwarding signals especially for resolving the load-and-use processor stall in this new pipelined processor? The data hazard detection will be similar with what we introduced for the original pipeline processor as in L11, p.14. Also, how can you detect the newly introduced hazard condition that you found in (b) to generate proper stalling signal for the processor, similar with L11, p. 17.

[Solution:]

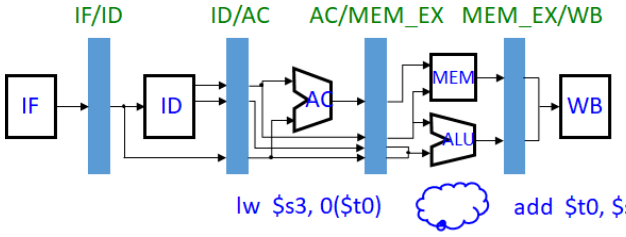
- Forwarding Unit #1



addi $s2, $t0, 4 lw $t0, 0($s1) : load-and-use data hazard
addi $s2, $t0, 4 add $t0, $s1, $s2 : regular data hazard

```
if (MEM_EX/WB.RegWrite and (MEM_EX/WB.RegisterRd ≠ 0))  { //RegWrite not to $0
    if (MEM_EX/WB.RegisterRd = AC/MEM_EX.RegisterRs))     //destination register = RegisterRS of next instruction
        ForwardA = 1                                      // Generate forwarding signal
    if (MEM_EX/WB.RegisterRd = AC/MEM_EX.RegisterRt))     //destination register = RegisterRt of next instruction
        ForwardB = 1                                      // Generate forwarding signal
}
```

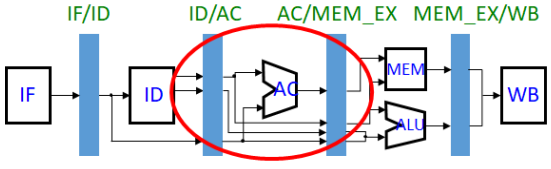
- Forwarding Unit #2



lw $s3, 0($t0) add $t0, $s1, $s2 //RegWrite-and-load after pipeline stall

```
if (MEM_EX/WB.RegWrite and (MEM_EX/WB.RegisterRd ≠ 0))  {        //RegWrite not to $0
    if (!((AC/MEM_EX.RegWrite =1) and (AC/MEM_EX.RegisterRd = ID/AC.RegisterRt)))
        if ((ID/AC.MemRead = 1) and (MEM_EX/WB.RegisterRd = ID/AC.RegisterRs)) // load-and-use
            ForwardC = 1
}
```

- Hazard Detection Unit (New Data Hazard Detection to stall pipelined processor)



lw $t0, 0($s1) addi $s2, $t0, 4 // Need to stall pipeline

This situation can only be done on 3rd stage since the load instruction can only be identified on this stage.

```
if (AC/MEM_EX.RegWrite and (AC/MEM_EX.RegisterRd ≠ 0))
    if (ID/AC.MemRead and (ID/AC.RegisterRt = AC/MEM_EX.RegisterRd))
      stall the pipeline (hold the PC and pipeline registers, IF/ID and ID/AC, and also generate 0
      for control signal in current stage (inserting bubble)
```