CmpEn 431
Computer Architecture
Fall 2015

Chapter 4B: The Pipelined Processor: Dealing with Data Hazards

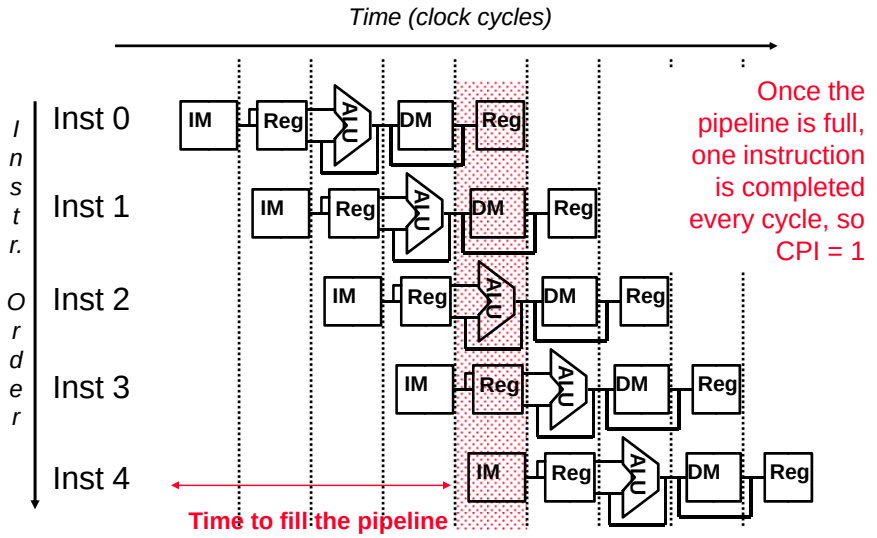
Mary Jane Irwin (www.cse.psu.edu/~mji)

[Adapted from *Computer Organization and Design, 5th Edition*, Patterson & Hennessy, © 2014, Morgan Kaufmann]

Reminders

- This week
  - MIPS pipelined datapath, handling data hazards – PH, Chapter 4.5-4.7
- Next week
  - Reducing branch hazard costs – PH 4.8
  - Branch prediction; dealing with exceptions – PH 4.9
- Reminders
  - HW2 now online, dropbox closes midnight Sep 17th
  - HW3 will come out Sep 18th (and will be due on Sep Oct 1st)
  - Quiz 2 will open Sept 11th and will close midnight Sept 22nd
  - First evening midterm exam scheduled
    - Tuesday, October 6th, 20:15 to 22:15, Location 22 Deike

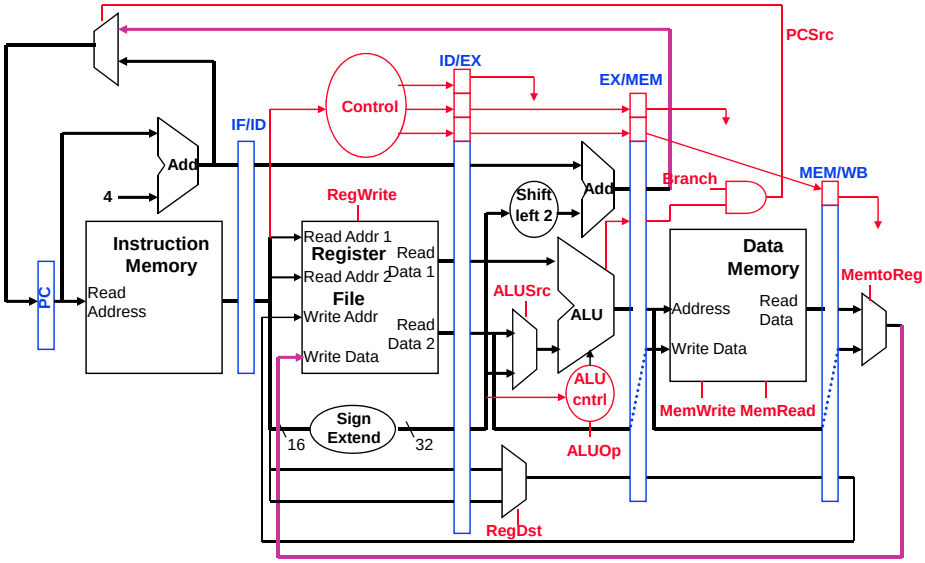
Review: Why Pipeline? For Performance!



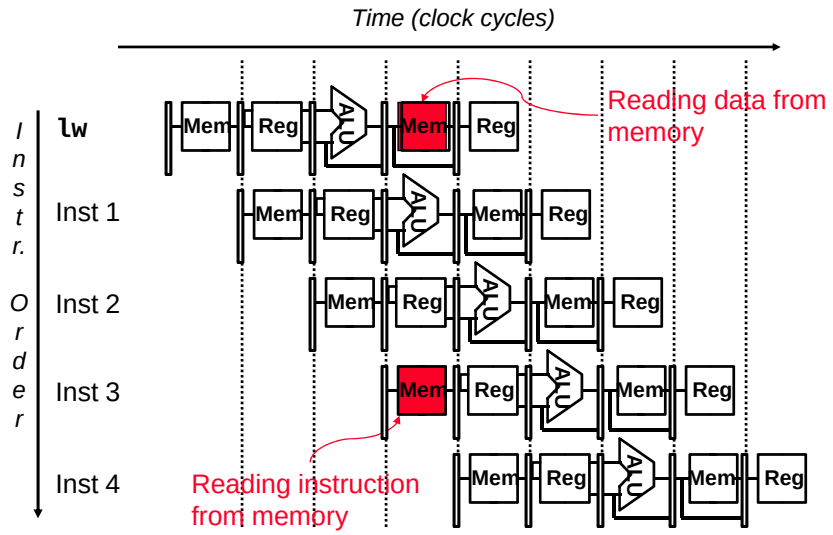
Review: Pipelining - What Makes it Hard ?

- Pipeline Hazards
  - structural hazards: attempt to use the same resource by two different instructions at the same time
  - data hazards: attempt to use data before it is ready
    - An instruction's source operand(s) are produced by a prior instruction still in the pipeline
  - control hazards: attempt to make a decision about program control flow before the condition has been evaluated and the new PC target address calculated
    - branch and jump instructions, exceptions
- Pipeline hardware control must detect the hazard and then take action to resolve hazard

Review: MIPS Pipeline Data and Control Paths

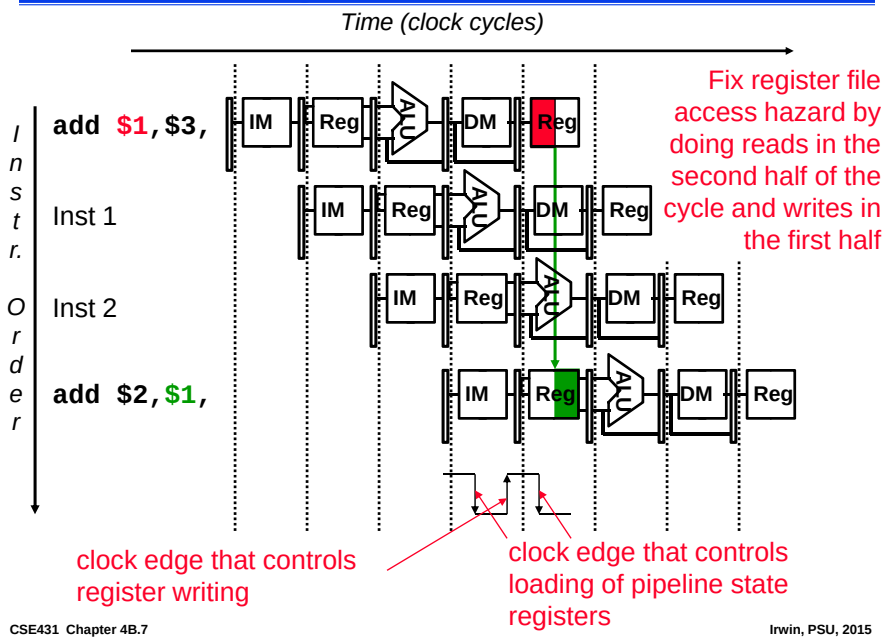


A Single Memory Would Be a Structural Hazard



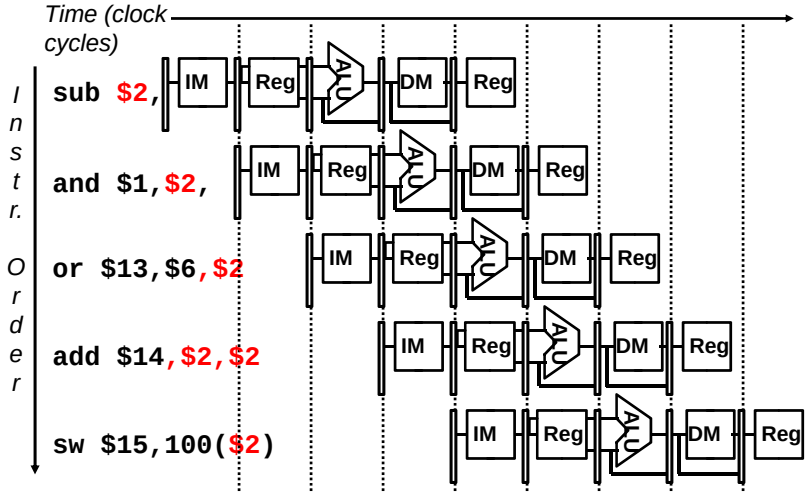
- Fix with separate instr and data memories (I$ and D$)

How About Register File Access?

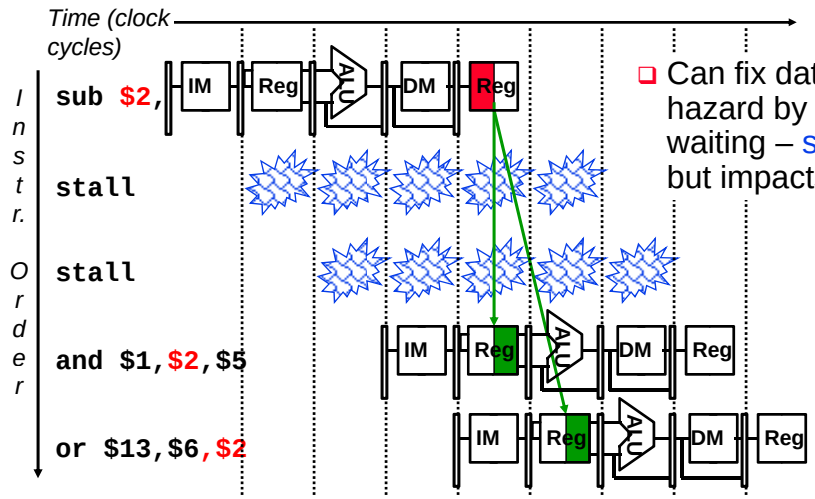


Register Usage Can Cause Data Hazards

- Dependencies backward in time cause hazards

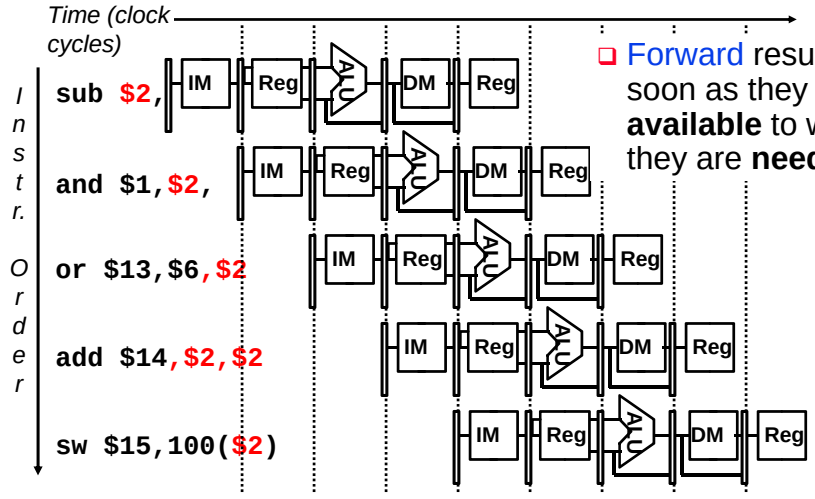


One Way to "Fix" a Data Hazard



- Can fix data hazard by waiting – stall – but impacts CPI

Another Way to "Fix" a Data Hazard

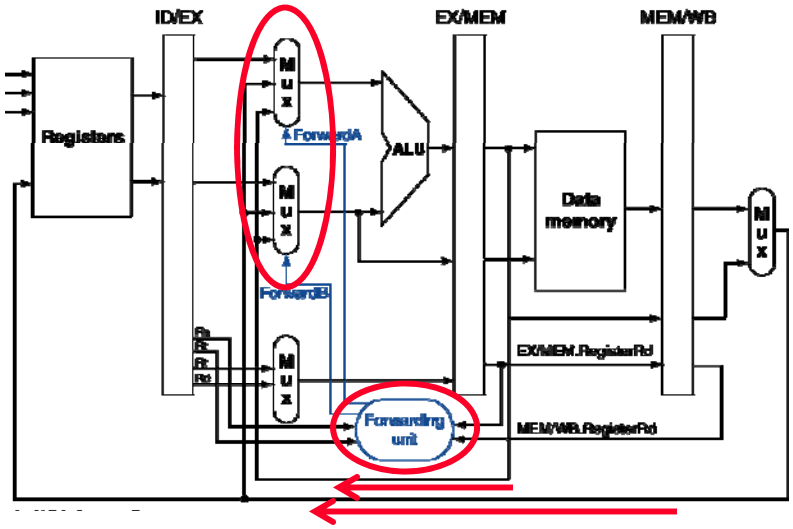


- Forward results as soon as they are **available** to where they are **needed**

Data Forwarding (aka Bypassing)

- Take the result from a downstream pipeline state register that holds the needed data that cycle and forward it to the functional units (e.g., the ALU) that need that data that cycle
- For ALU functional unit: the inputs can come from other pipeline registers than just ID/EX by
  - adding multiplexors to the inputs of the ALU
  - connecting the Rd write data in EX/MEM or MEM/WB to either (or both) of the EX's stage Rs and Rt ALU mux inputs
  - adding the proper control hardware to control the new muxes
- Other functional units may need similar forwarding logic (e.g., the DM)
- With forwarding can achieve a CPI of 1 even in the presence of data dependencies

Forwarding Logic



Data Forwarding Control Conditions

1. EX/MEM Forward Unit:

```
if (EX/MEM.RegWrite
and (EX/MEM.RegisterRd != 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRs))
      ForwardA = 10
if (EX/MEM.RegWrite
and (EX/MEM.RegisterRd != 0)
and (EX/MEM.RegisterRd = ID/EX.RegisterRt))
      ForwardB = 10
```

Forwards the result from the previous instr. to either input of the ALU

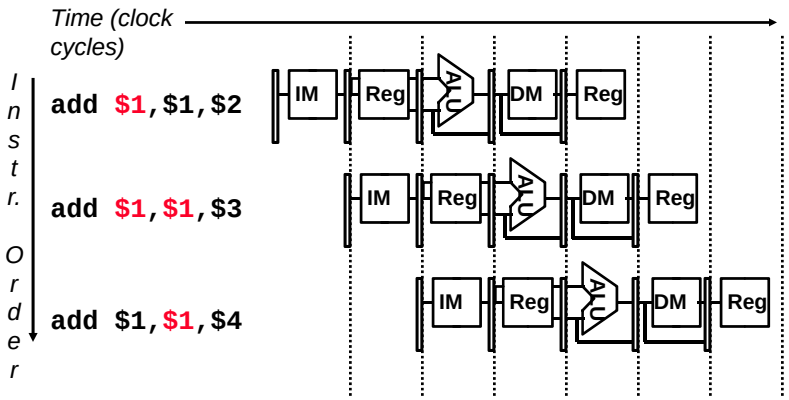
2. MEM/WB Forward Unit:

```
if (MEM/WB.RegWrite
and (MEM/WB.RegisterRd != 0)
and (MEM/WB.RegisterRd = ID/EX.RegisterRs))
      ForwardA = 01
if (MEM/WB.RegWrite
and (MEM/WB.RegisterRd != 0)
and (MEM/WB.RegisterRd = ID/EX.RegisterRt))
      ForwardB = 01
```

Forwards the result from the second previous instr. to either input of the ALU

Yet Another Complication!

- Another potential data hazard can occur when there is a conflict between the result of the WB stage instruction and the MEM stage instruction – which should be forwarded?



Corrected Data Forwarding Control Conditions

1. EX/MEM Forward Unit:

```
if (EX/MEM.RegWrite
...
```

Forwards the result from the previous instr. to either input of the ALU

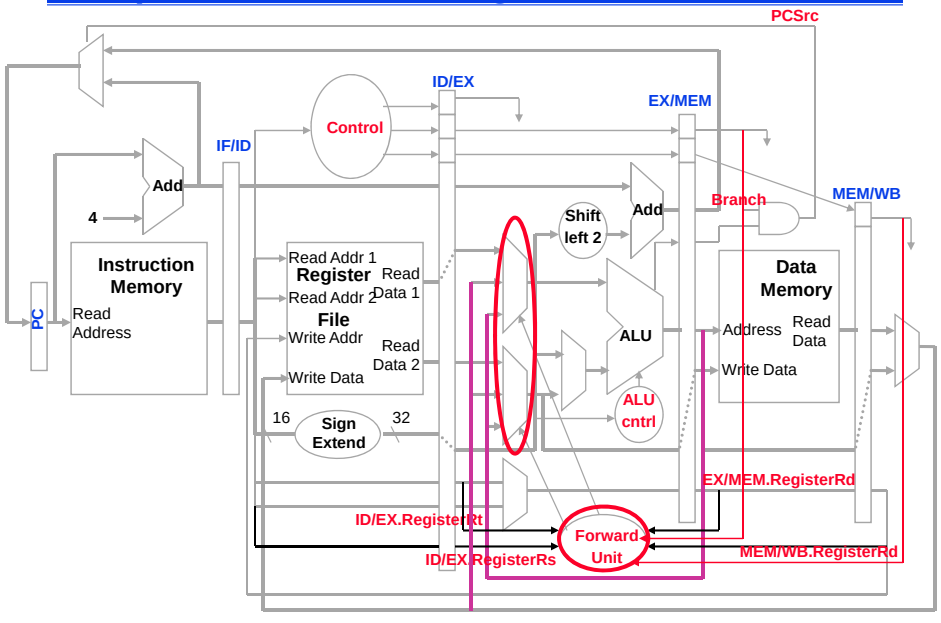
2. MEM/WB Forward Unit:

```
if (MEM/WB.RegWrite
and (MEM/WB.RegisterRd != 0)
and !(EX/MEM.RegWrite and (EX/MEM RegisterRD != 0)
   and (EX/MEM.RegisterRd = ID/EX.RegisterRs))
and (MEM/WB.RegisterRd = ID/EX.RegisterRs))
      ForwardA = 01
```

Forwards the result from the previous or second previous instr. to either input of the ALU

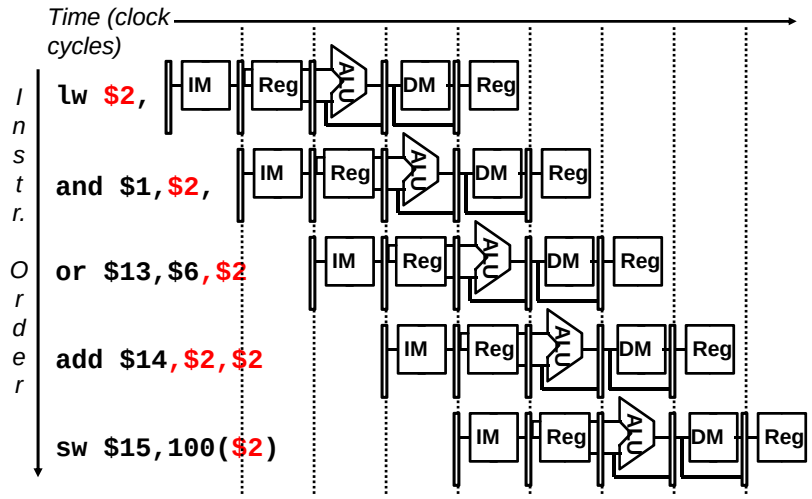
```
if (MEM/WB.RegWrite
and (MEM/WB.RegisterRd != 0)
and !(EX/MEM.RegWrite and (EX/MEM.RegisterRd != 0)
   and (EX/MEM.RegisterRd = ID/EX.RegisterRt)
and (MEM/WB.RegisterRd = ID/EX.RegisterRt))
      ForwardB = 01
```


Datapath with Forwarding Hardware



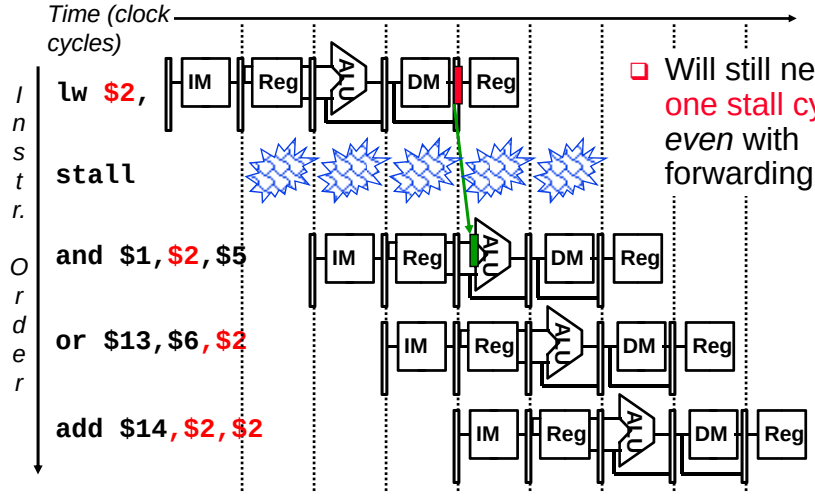
Loads Can Cause Data Hazards

- Dependencies backward in time cause hazards



Load-Use Can Cause Data Hazards

- Dependencies backward in time cause hazards



- Will still need one stall cycle *even* with forwarding

Load-use Data Hazard Detection Unit

- Need a Hazard detection Unit in the ID stage that inserts a `stall` between the load and its use

1. ID Hazard detection Unit:

```
if (ID/EX.MemRead
and ((ID/EX.RegisterRt = IF/ID.RegisterRs)
or  (ID/EX.RegisterRt = IF/ID.RegisterRt)))
stall the pipeline
```

- The first line tests to see if the instruction now in the EX stage is a `lw`; the next two lines check to see if the destination register of the `lw` matches either source register of the instruction in the ID stage (the load-use instruction)
- After this one cycle stall, the forwarding logic can handle the remaining data hazards

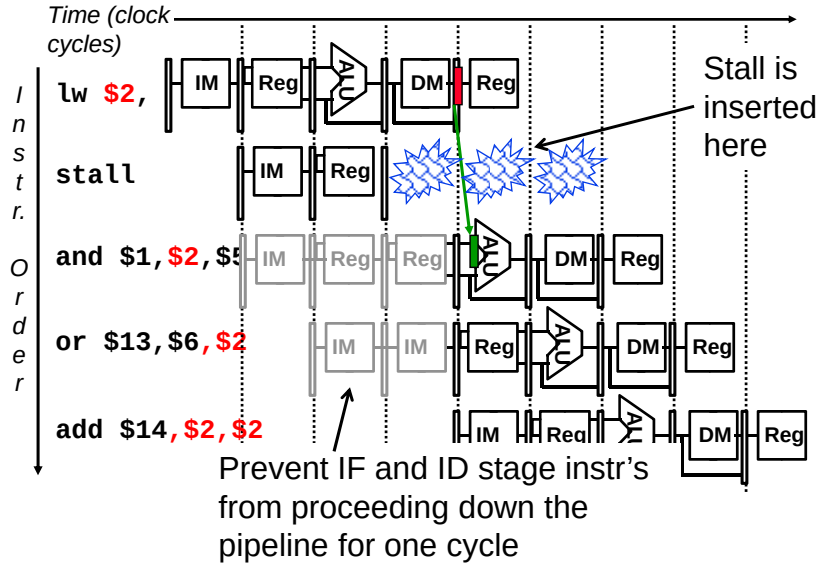
Data Hazard/Stall Hardware

Along with the Hazard Unit, we have to *implement* the stall

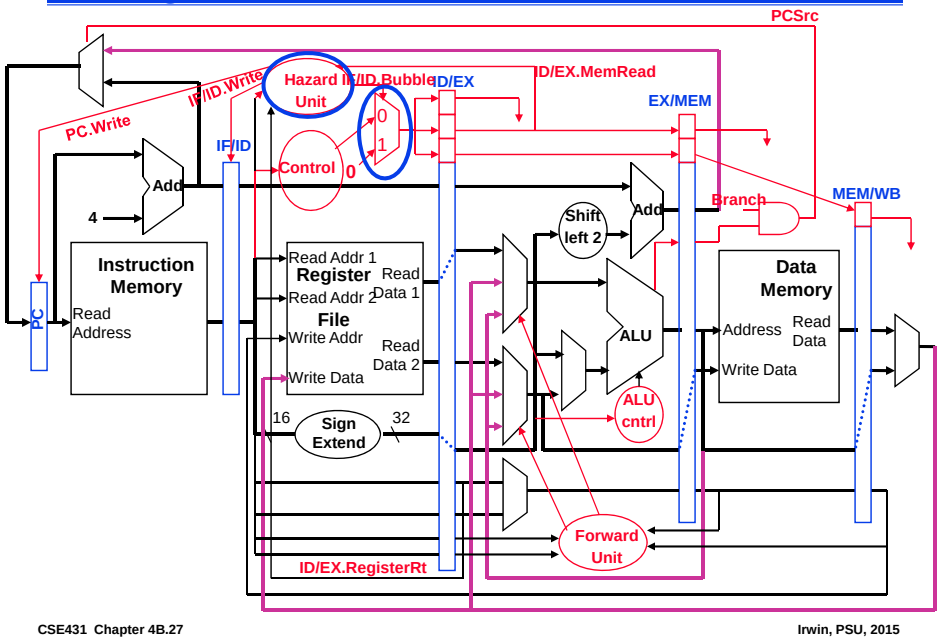
1. Insert a "bubble" between the `lw` instruction (in the EX stage) and the load-use instruction (in the ID stage) (i.e., insert a `noop` with `IF/ID.Bubble`)
   - Set the control bits in the EX, MEM, and WB control fields of the ID/EX pipeline register to 0 (`noop`). The Hazard Unit controls the mux that chooses between the real control values and the 0's.
2. Prevent the instr's in the IF and ID stages from proceeding down the pipeline – by preventing the PC register and the IF/ID pipeline register from changing
   - Hazard detection Unit controls the writing of the PC (`PC.write`) and IF/ID (`IF/ID.Write`) registers
3. Let the `lw` instruction and the instructions after it in the pipeline (before it in the code) proceed normally down the pipeline

Load-Use Can Cause Data Hazards

- Dependencies backward in time cause hazards



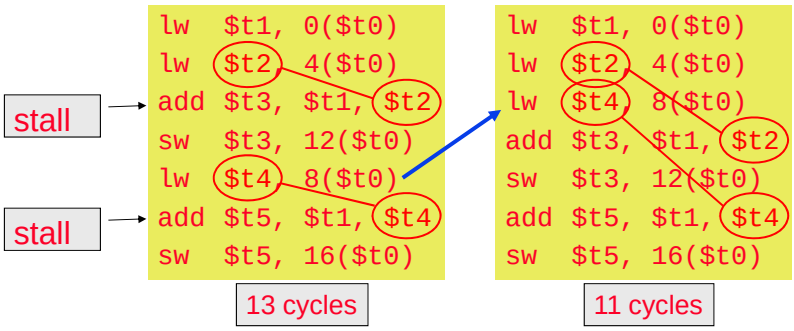
Adding the Data Hazard/Stall Hardware



Use Code Scheduling to Avoid L-U Stalls

- Reorder code to avoid use of load result in the next instruction

  C code for `A = B + E; C = B + F;`



Summary

- All modern day processors use pipelining for performance (a CPI of 1 and fast a CC)
- Pipeline clock rate limited by slowest pipeline stage – so designing a balanced pipeline is important
- Must detect and resolve hazards
  - Structural hazards – resolved by designing the pipeline correctly
  - Data hazards
    - Stall (impacts CPI)
    - Forward (requires hardware support)
  - Control hazards – put the branch decision hardware in as early a stage in the pipeline as possible
    - Stall (impacts CPI)
    - Delay decision (requires compiler support)
    - Static and dynamic prediction (requires hardware support)
- Pipelining complicates exception handling

Reminders

- Next week
  - Reducing branch hazard costs – PH 4.8
  - Branch prediction; dealing with exceptions – PH 4.9

- Reminders
  - HW2 now online, dropbox closes midnight Sep 17th
  - HW3 will come out Sep 18th (and will be due on Sep Oct 1st)
  - Quiz 2 will open Sept 11th and will close midnight Sept 22nd
  - First evening midterm exam scheduled
    - Tuesday, October 6th, 20:15 to 22:15, Location 22 Deike
    - NO conflict exam has been schedule