Overcoming Data Hazards

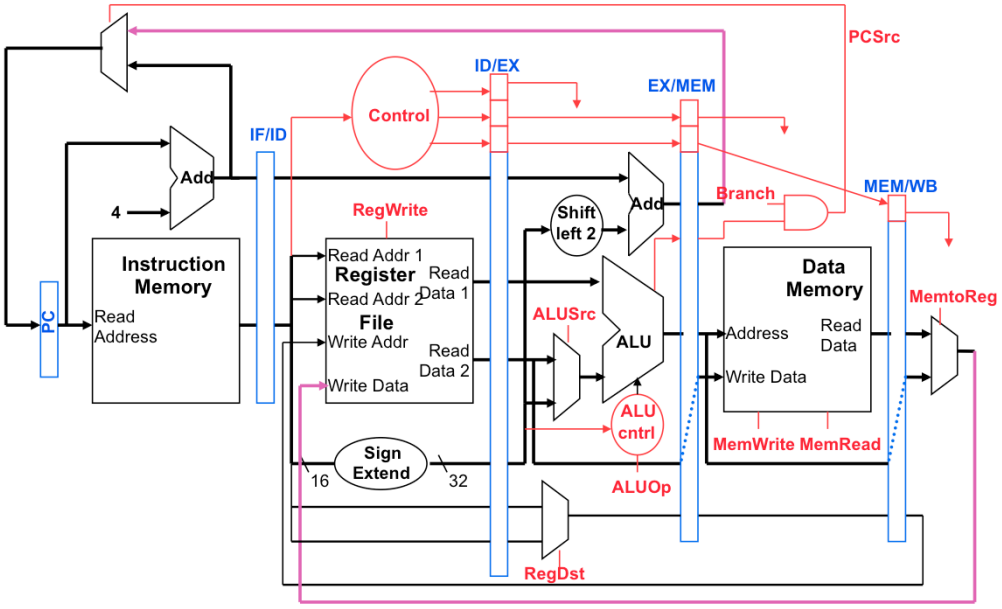
[Adapted from Mary Jane Irwin for
Computer Organization and Design,
Patterson & Hennessy, © 2005, UCB]

Other handouts

To handout next time

6.2 aus Buch

Review: MIPS Pipeline Data and Control Paths



How many bits wide is each pipeline register?

PC – 32 bits

IF/ID – 64 bits

ID/EX – 9 + 32x4 + 10 = 147

EX/MEM – 5 + 1 + 32x3 + 5 = 107

MEM/WB – 2 + 32x2 + 5 = 71

Notice that MEM/WB connects to MemtoReg and RegWrite

Signals to be set:

Instr fetch: all asserted

Instr decode/Register read: all asserted

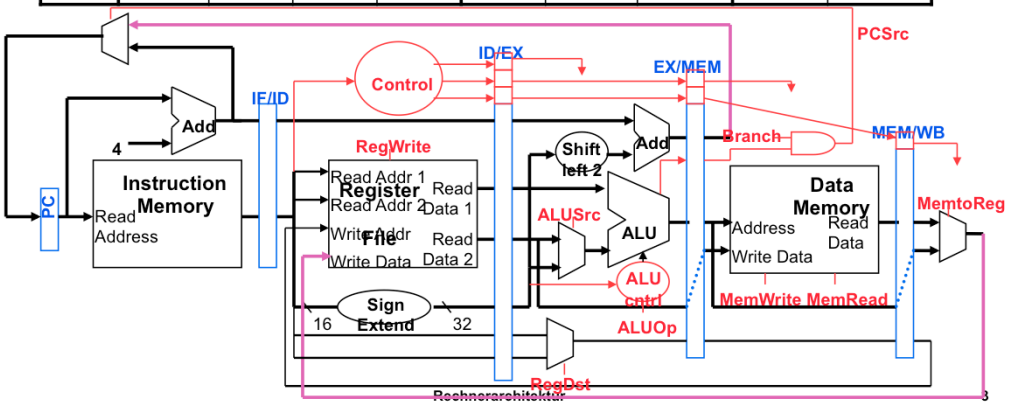
Exec/Address calculation: RegDst, ALUOp, ALUSrc, (select Result reg, ALU oper, either Read datas 2 or sign-extend immediate for ALU)

Memory access: Branch, MemRead, MemWrite (set by branch equal, load, store); PCSrc selects PC+4 unless Branch & ALU result = 0

Write Back: MemtoReg (either ALU result or memory value), RegWrite

Control Settings

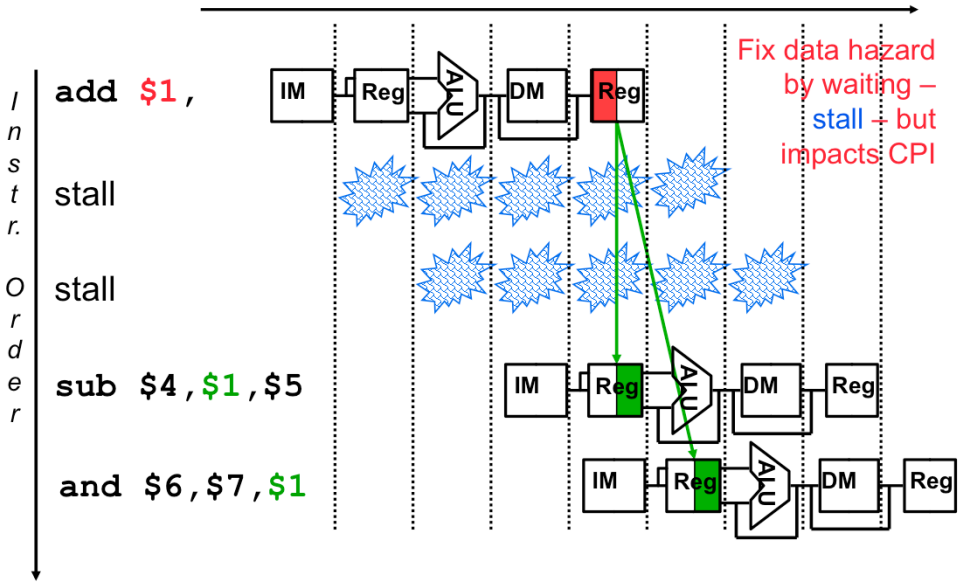
	EX Stage				MEM Stage			WB Stage	
	Reg Dst	ALU Op1	ALU Op0	ALU Src	Brch	Mem Read	Mem Write	Reg Write	Mem toReg
R	1	1	0	0	0	0	0	1	0
lw	0	0	0	1	0	1	0	1	1
sw	X	0	0	1	0	0	1	0	X
beq	X	0	1	0	1	0	0	0	X



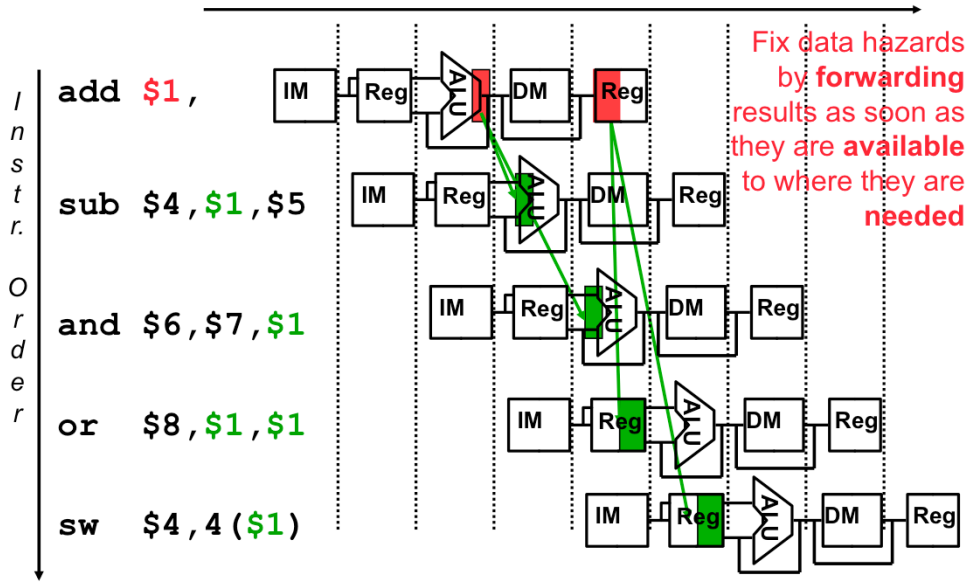
Setting of control lines was fully determined by the instruction codes

Now they are grouped according to the last three pipeline stages

Review: One Way to "Fix" a Data Hazard



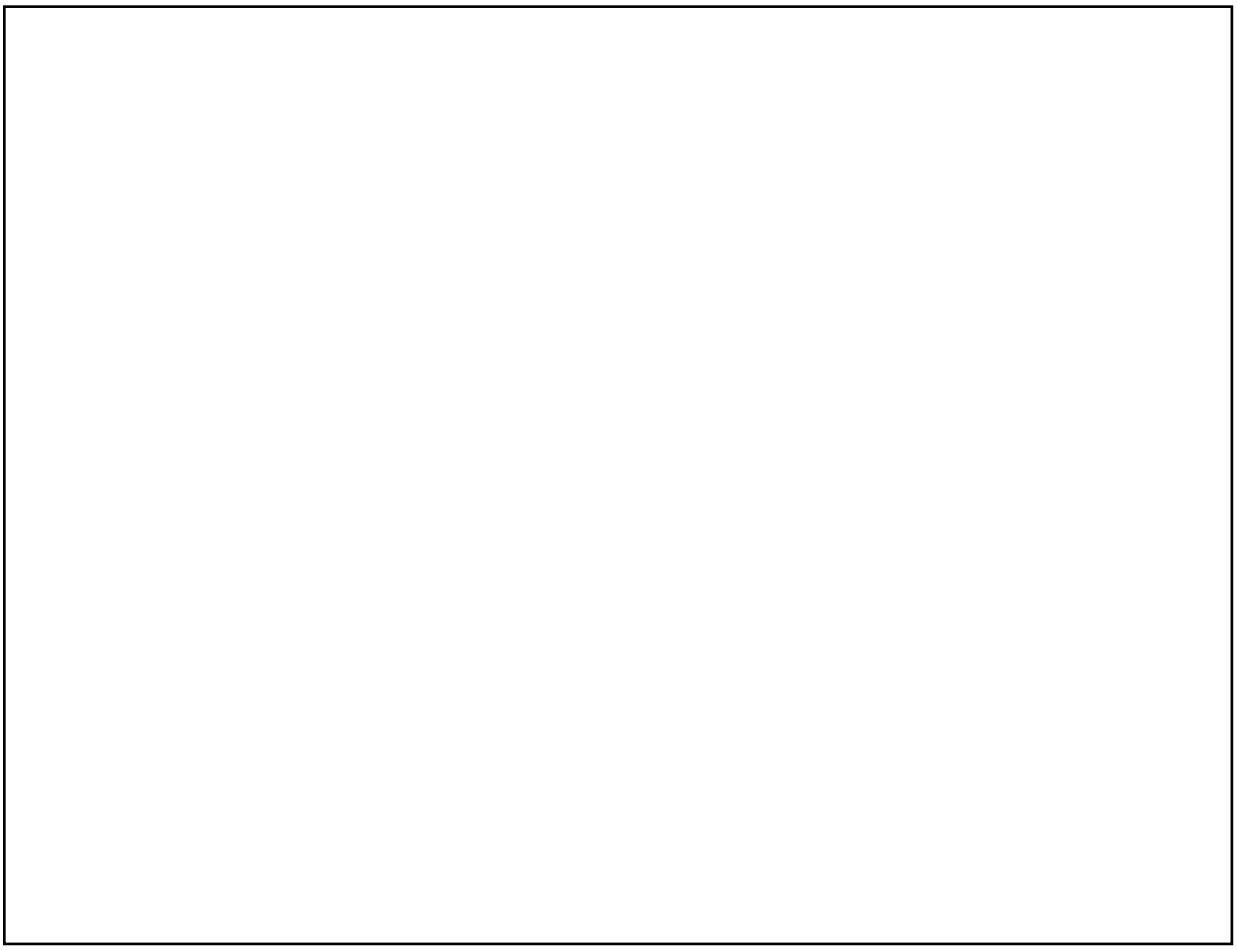
Review: Another Way to "Fix" a Data Hazard



Notice that for now we are showing the forwarded data coming out of the ALU. After looking at the problem more closely we will see that it really is supplied by the pipeline register EX/MEM and will depict it as such.

Data Forwarding (aka Bypassing)

- Take the result from the earliest point that it exists in any of the pipeline state registers and forward it to the functional units (e.g., the ALU) that need it that cycle
- For ALU functional unit: the inputs can come from any pipeline register rather than just from 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



Example

- Eliminate stalls by reordering code

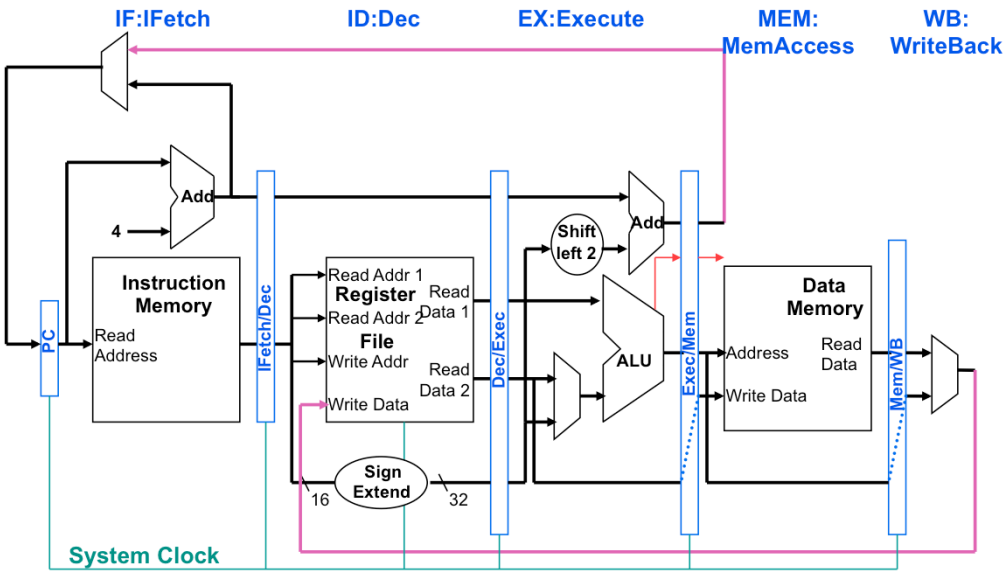
```
lw      $t1, 0($t0)
lw      $t2, 4($t0)
add     $t3, $t1,$t2
sw      $t3, 12($t0)
lw      $t4, 8($t0)
add     $t5, $t1,$t4
sw      $t5, 16($t0)
```


Solution: both add have hazard with previous lw

Move third lw to 3rd position

Will complete in 2 fewer cycles compared to original ordering

MIPS Pipeline Datapath



Recall plot to show registers and illustrate the forwarding as direct feedback

Data Forwarding Control Conditions

1. EX/MEM hazard:

```
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 hazard:

```
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

First hazard can be detected when the AND instruction is in the EX stage and the prior instruction is in the MEM stage

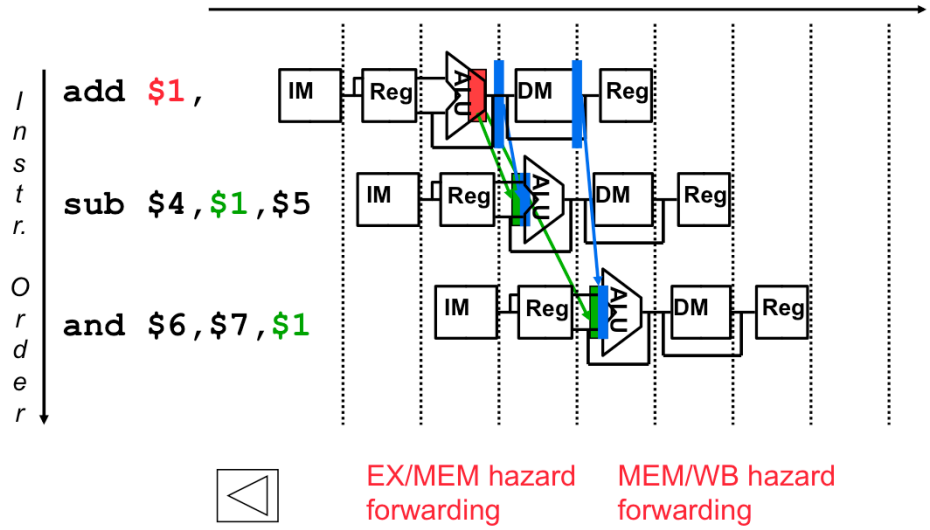
The hazard occurs when they refer to the same register.

First: we plan to write to this register,

Second: we are not using the register $0,

Third: the used register is the same as in previous stage

Forwarding Illustration



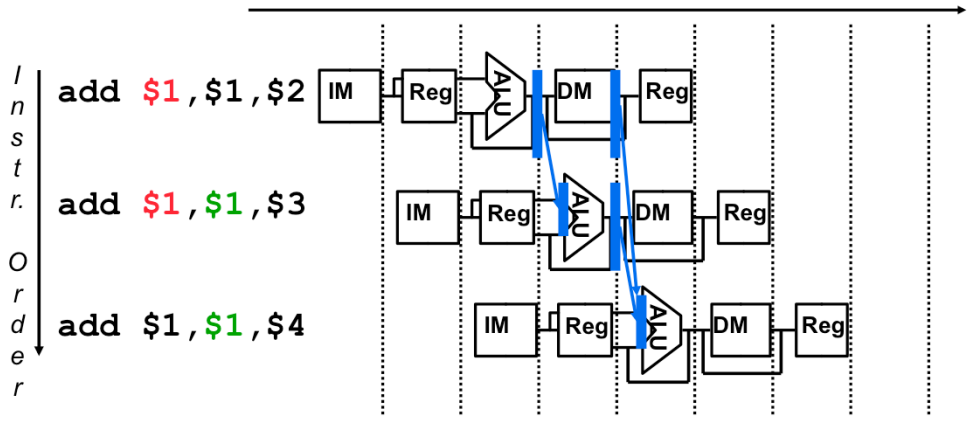
EX/MEM hazard forwarding

MEM/WB hazard forwarding

Notice that for now we are showing the forwarded data coming out of the ALU. After looking at the problem more closely we will see that it really is supplied by the pipeline register EX/MEM and will depict it as such.

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?



For lecture

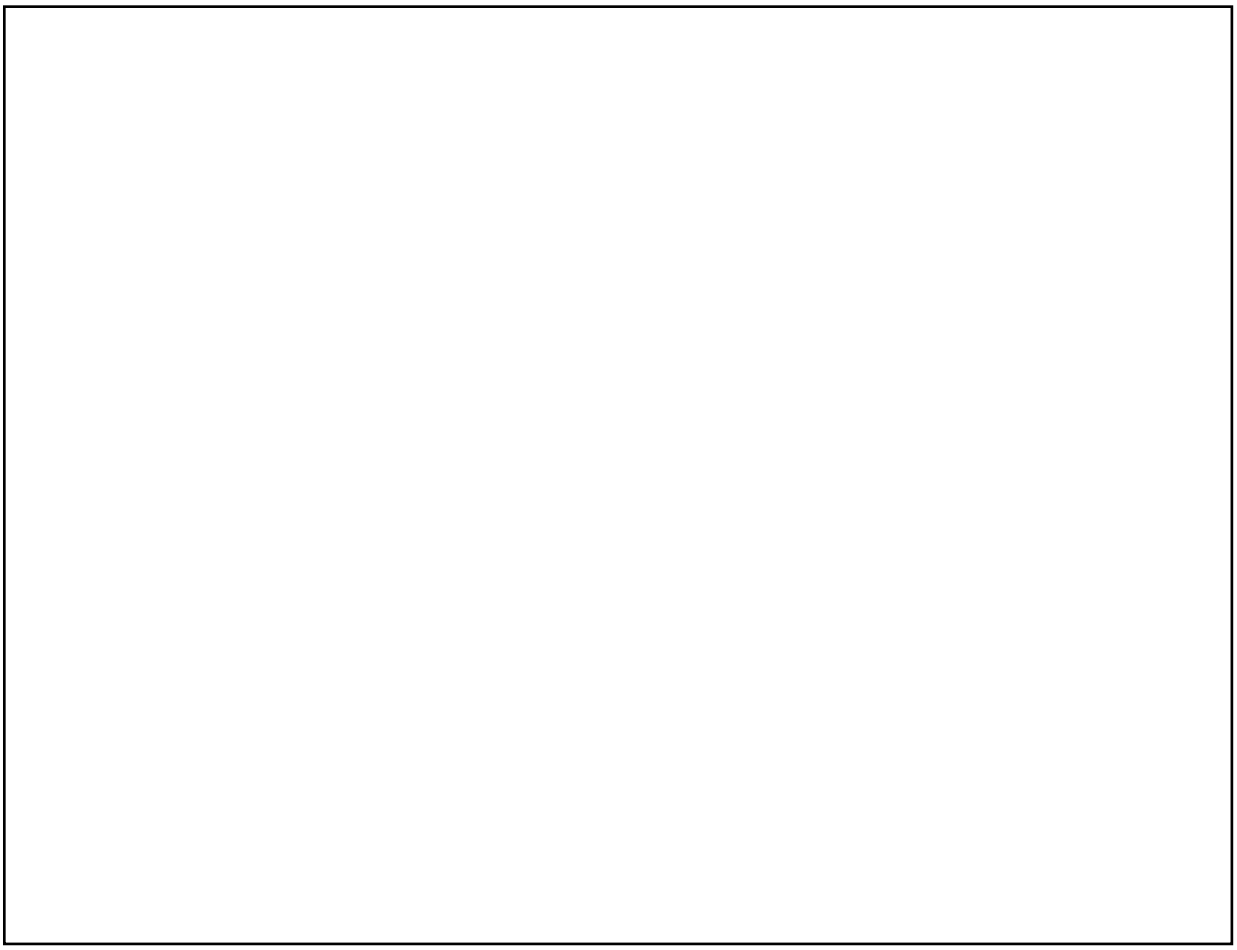
Corrected Data Forwarding Control Conditions

2. MEM/WB hazard:

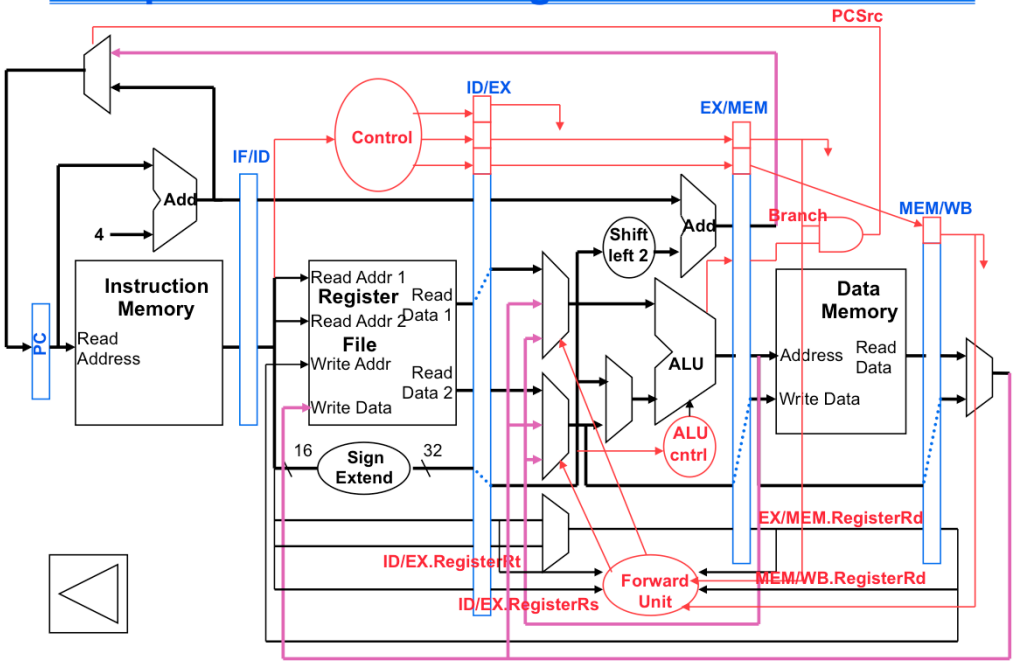
```
if (MEM/WB.RegWrite
and (MEM/WB.RegisterRd != 0)
and (EX/MEM.RegisterRd != ID/EX.RegisterRs)
and (MEM/WB.RegisterRd = ID/EX.RegisterRs))
      ForwardA = 01

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

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



Datapath with Forwarding Hardware



For lecture.

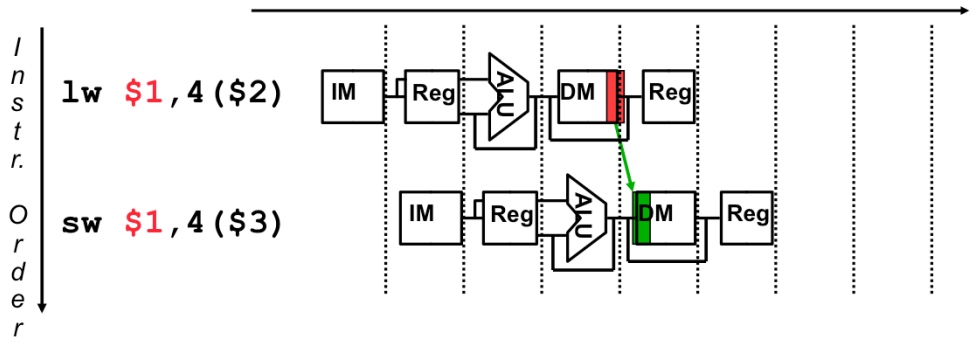
How many bits wide is each pipeline register now?

ID/EX – 9 + 32x4 + 10 = 147 + 10 = 157

Control line inputs to Forward Unit EX/MEM.RegWrite and MEM/WB.RegWrite not shown on diagram

Memory-to-Memory Copies

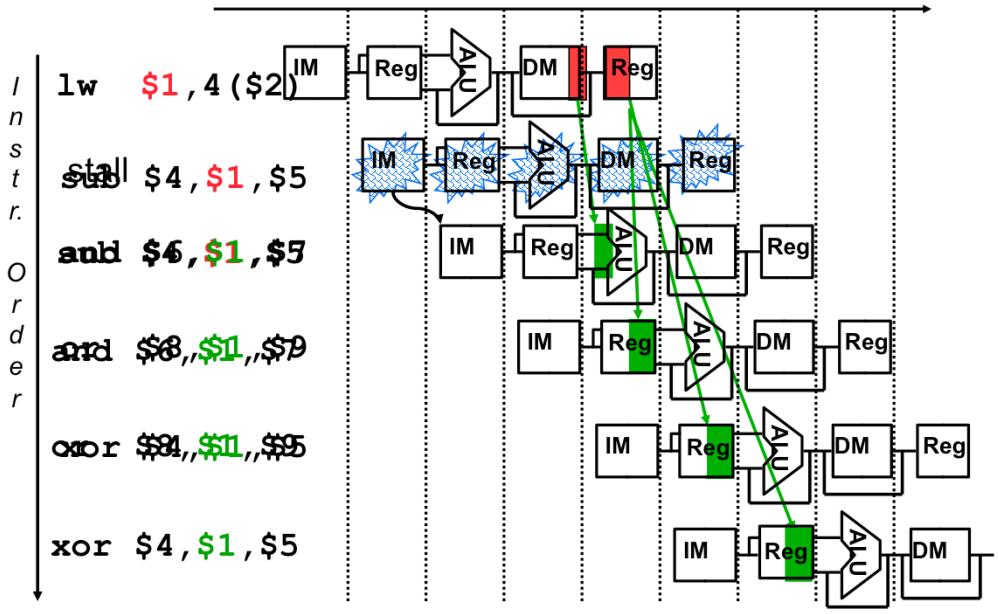
- For loads immediately followed by stores (memory-to-memory copies) can avoid a stall by adding forwarding hardware from the MEM/WB register to the data memory input.
  - Would need to add a Forward Unit and a mux to the memory access stage



What if lw was replaced with add $1, - is forwarding still needed? From where, to where? Yes, from ALUout to ALUin2

What if $1 was used to compute the effective address of the store (it would be a load-use data hazard and would require a stall insertion between the lw and sw)

Forwarding with Load-use Data Hazards



For lecture

The one case where forwarding cannot save the day is when an instruction tries to read a register following a load instruction that writes the same register.

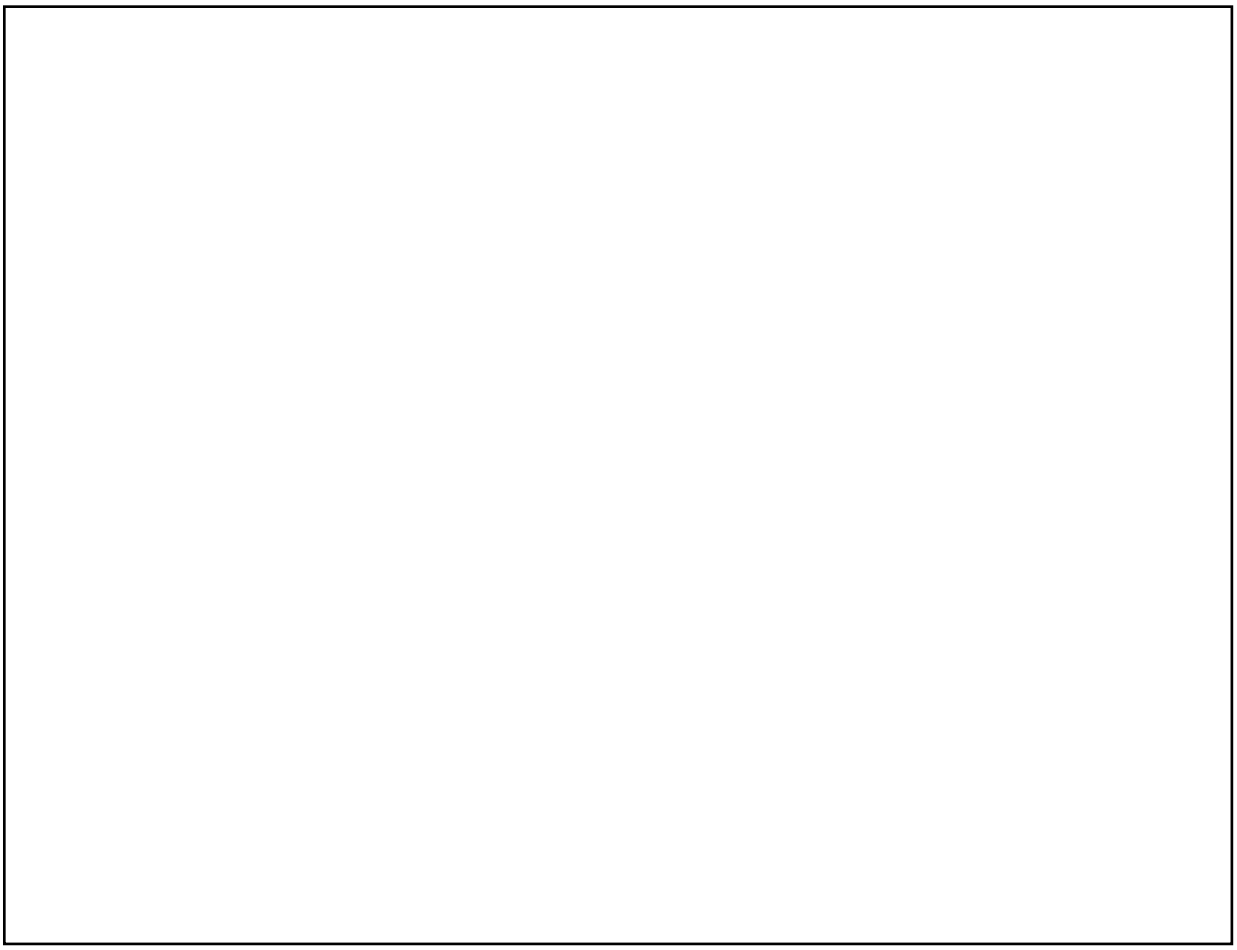
Load-use Hazard Detection Unit

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

2. ID Hazard Detection

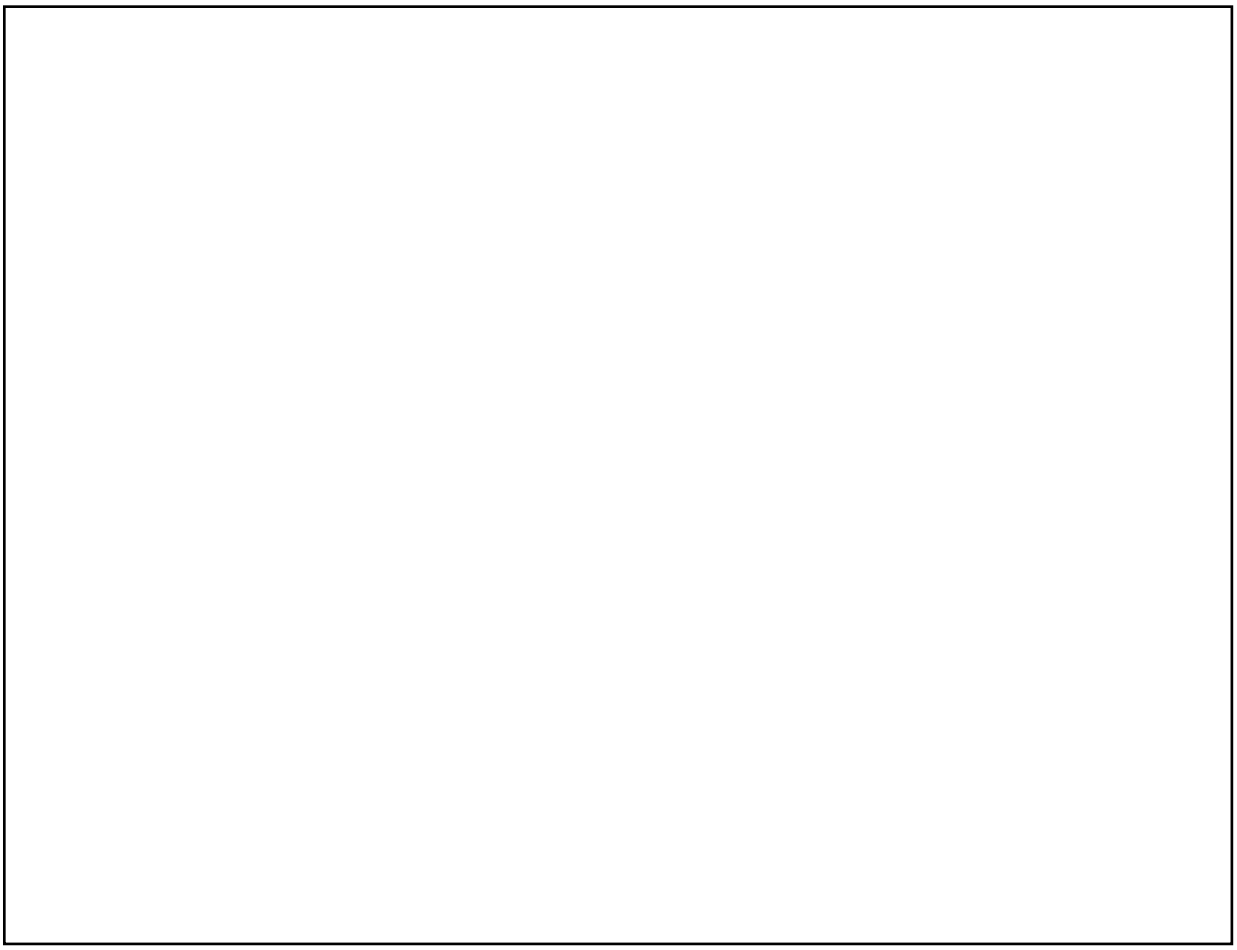
```
if (ID/EX.MemRead
and ((ID/EX.RegisterRt = IF/ID.RegisterRs)
or  (ID/EX.RegisterRt = IF/ID.RegisterRt)))
stall the pipeline
```

- 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

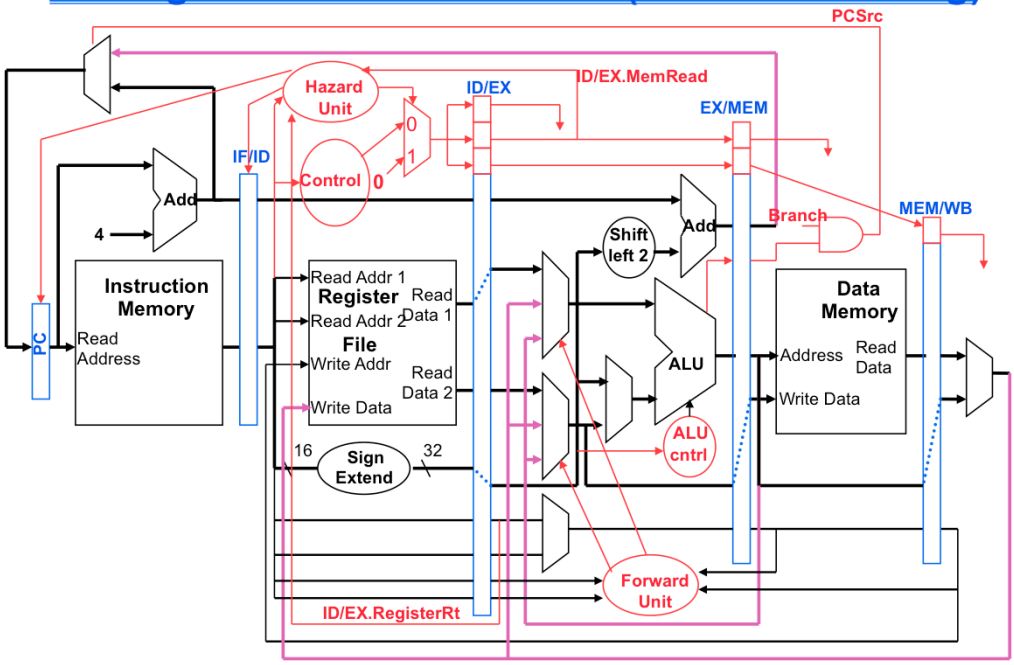


Stall Hardware

- Along with the Hazard Unit, we have to implement the stall
- Prevent the instructions in the IF and ID stages from progressing down the pipeline – done 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
- 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` in the execution stream)
  - 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.
- Let the `lw` instruction and the instructions after it in the pipeline (before it in the code) proceed normally down the pipeline



Adding the Hazard Hardware (and Clock Gating)



For lecture

In reality, only the signals RegWrite and MemWrite need to be 0, the other control signals can be don't cares.

Another consideration is energy – where clock gating is called for.

> Clock gating is one of the power-saving techniques used on the Pentium 4 processor. To save power, clock gating refers to activating the clocks in a logic block only when there is work to be done. From the earliest days of the Pentium 4 processor design, power consumption was a concern. The clock gating concept isn't a new one; however, the Pentium 4 processor used this technology to a large extent. Every unit on the chip has a power reduction plan, and almost every Functional Unit Block (FUB) contains clock gating logic.