

Pipeline Data Hazards

Warning, warning, warning!

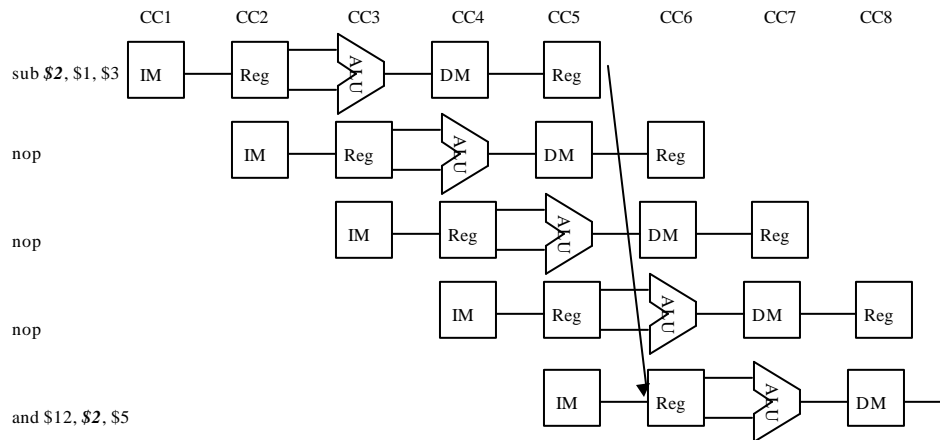
Dealing With Data Hazards

- In Software
 - inserting independent instructions
- In Hardware
 - inserting bubbles (stalling the pipeline)
 - data forwarding

Data Hazards are caused by *instruction dependences*. For example, the add is data-dependent on the subtract:

```
subi $5, $4, #45  
add  $8, $5, $2
```

Dealing with Data Hazards in Software



CSE 141

Allan Snively

How Many No-ops?

sub \$2, \$1,\$3
 and \$4, \$2,\$5
 or \$8, \$2,\$6
 add \$9, \$4,\$2
 slt \$1, \$6,\$7

CSE 141

Allan Snively

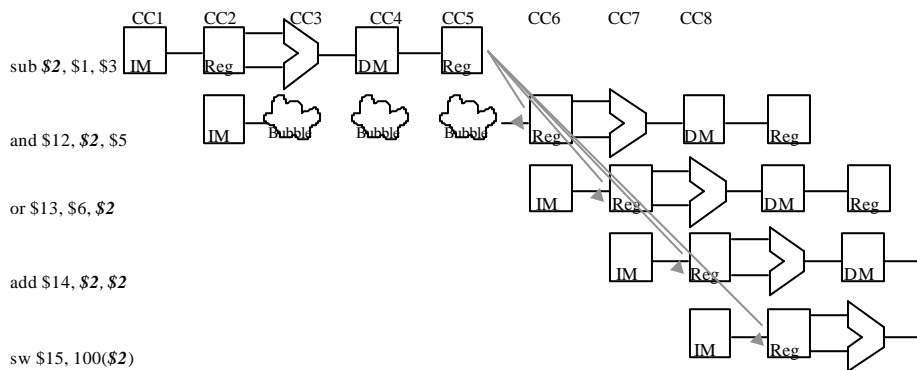
Are No-ops Really Necessary?

sub \$2, \$1,\$3
and \$4, \$2,\$5
or \$8, \$3,\$6
add \$9, \$2,\$8
slt \$1, \$6,\$7

CSE 141

Allan Snively

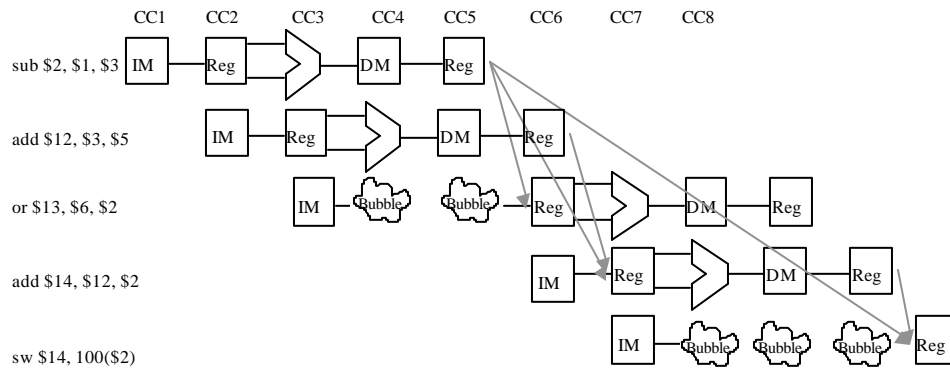
Dealing with Data Hazards in Hardware Part II-Pipeline Stalls



CSE 141

Allan Snively

Pipeline Stalls



CSE 141

Allan Snively

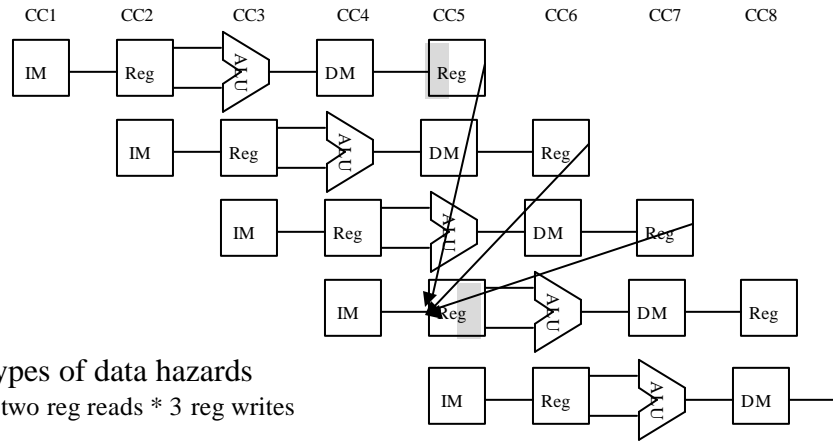
Pipeline Stalls

- We must:
 - Detect the hazard
 - Stall the pipeline

CSE 141

Allan Snively

Knowing When to Stall

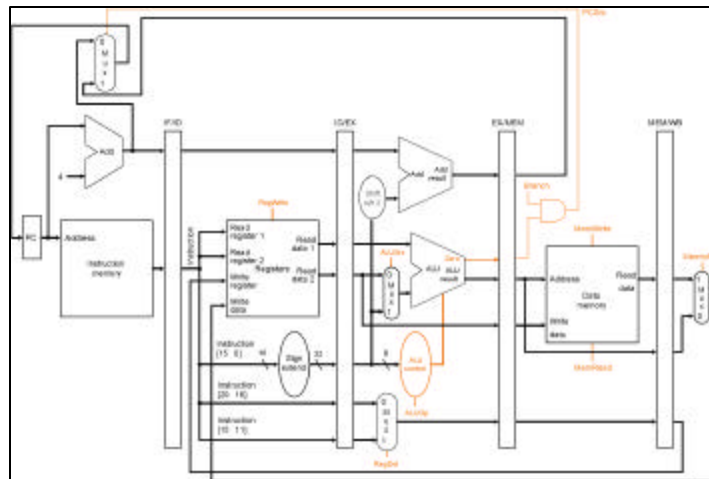


- 6 types of data hazards
 - two reg reads * 3 reg writes

CSE 141

Allan Snively

The Pipeline



- What comparisons tell us when to stall?

CSE 141

Allan Snively

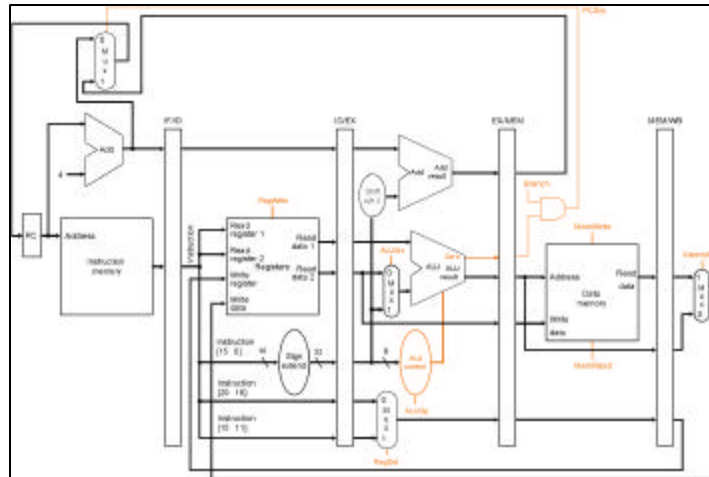
Stalling the Pipeline

- Once we detect a hazard, then we have to be able to stall the pipeline (insert a bubble).
- Stalling the pipeline is accomplished by
 - (1) preventing the IF and ID stages from making progress
 - the ID stage because it cannot proceed until the dependent instruction completes
 - the IF stage because we do not want to lose any instructions.
 - (2) inserting “nops” in hardware

Stalling the Pipeline

- Preventing the IF and ID stages from proceeding
 - don't write the PC ($PCWrite = 0$)
 - don't rewrite IF/ID register ($IF/IDWrite = 0$)
- Inserting “nops”
 - set all control signals propagating to EX/MEM/WB to zero

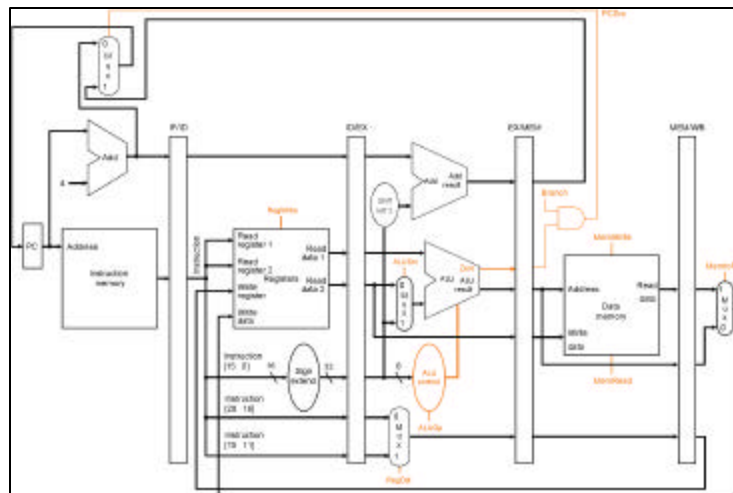
The Pipeline



CSE 141

Allan Snively

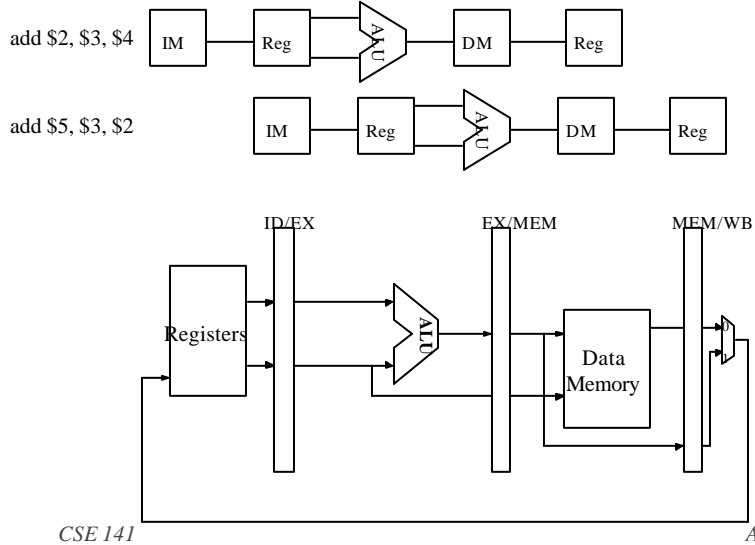
The Pipeline



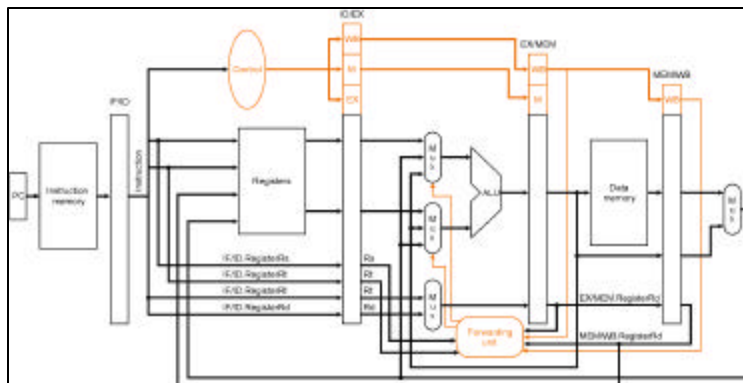
CSE 141

Allan Snively

Reducing Data Hazards Through Forwarding



Reducing Data Hazards Through Forwarding



EX 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

CSE 141

Allan Snively

(similar for the MEM stage)

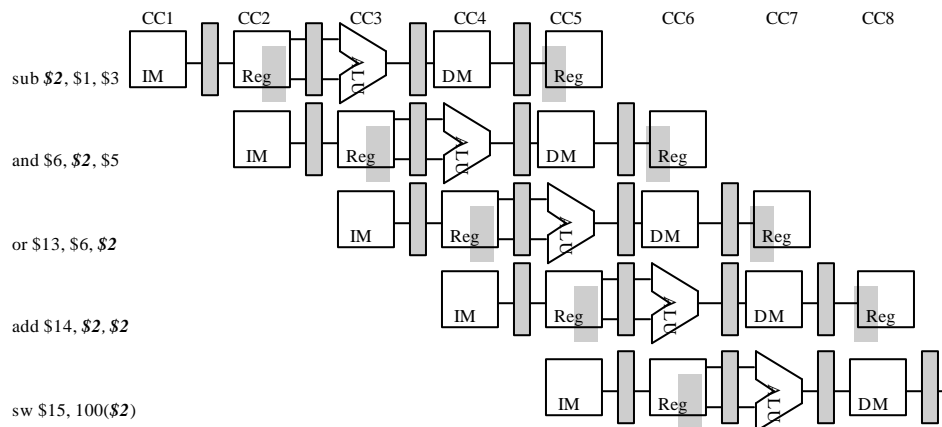
Data Forwarding

- The Previous Data Path handles two types of data hazards
 - EX hazard
 - MEM hazard
- We assume the register file handles the third (WB hazard)
 - if the register file is asked to read and write the same register in the same cycle, we assume that the reg file allows the write data to be forwarded to the output

CSE 141

Allan Snively

Eliminating Data Hazards via Forwarding



CSE 141

Allan Snively

Forwarding in Action

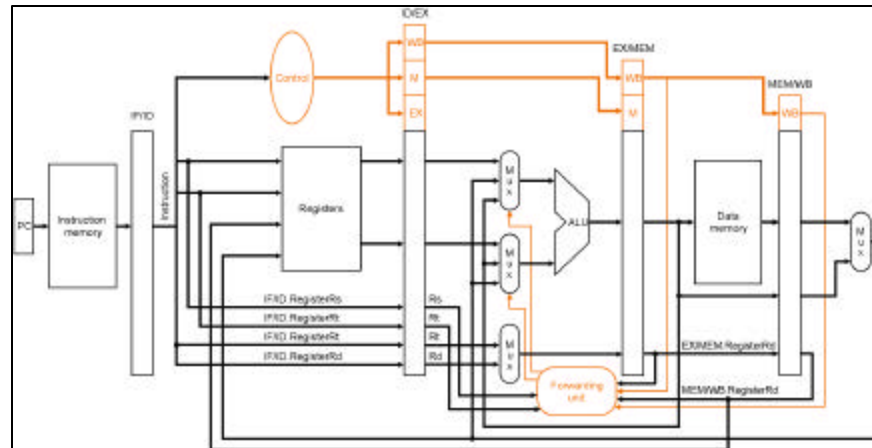
add \$1, \$12, \$3

sub \$12, \$3, \$4

add \$3, \$10, \$11

Memory Access

Write Back



CSE 141

Allan Snively

Forwarding in Action

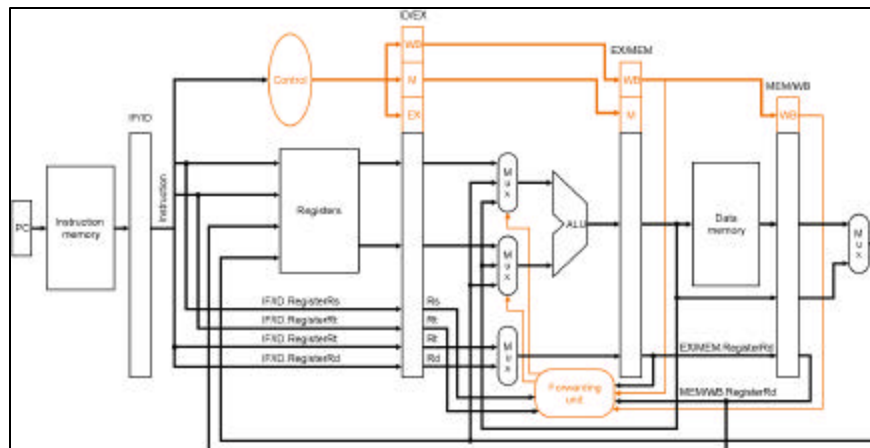
Instruction Fetch

add \$1, \$12, \$3

sub \$12, \$3, \$4

add \$3, \$10, \$11

Write Back

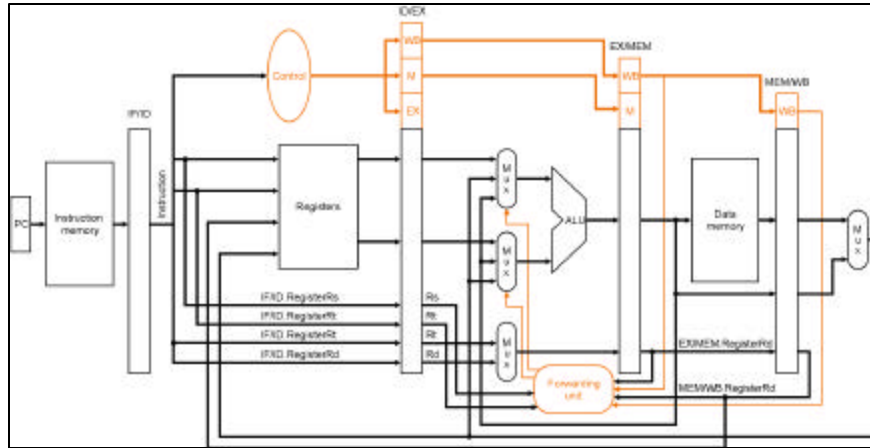


CSE 141

Allan Snively

Forwarding in Action

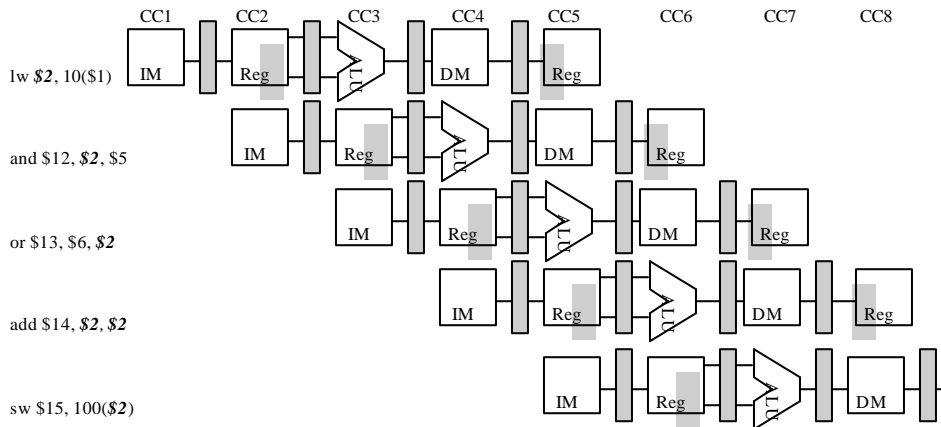
Instruction Fetch Instruction Decode **add \$1, \$12, \$3** **sub \$12, \$3, \$4** **add \$3, \$10, \$11**



CSE 141

Allan Snively

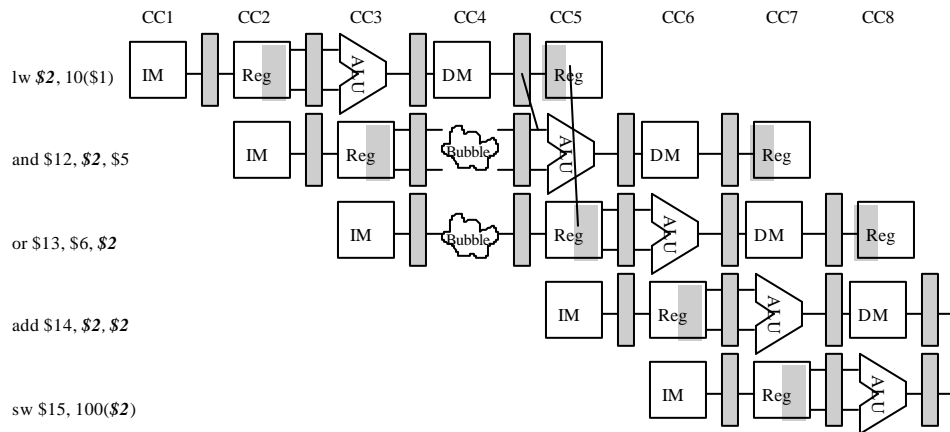
Eliminating Data Hazards via Forwarding??



CSE 141

Allan Snively

Eliminating Data Hazards via Forwarding and stalling



CSE 141

Allan Snively

Try this one...

Show stalls and forwarding for this code

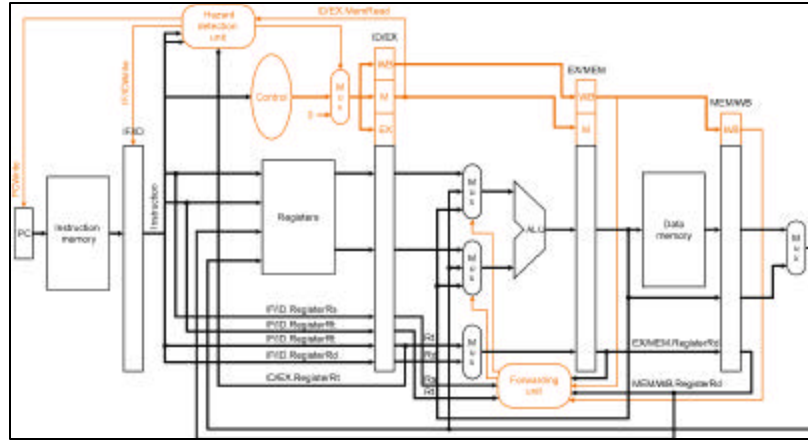
```

add $3, $2, $1
lw $4, 100($3)
and $6, $4, $3
sub $7, $6, $2
    
```

CSE 141

Allan Snively

Datapath with Hazard-Detection



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

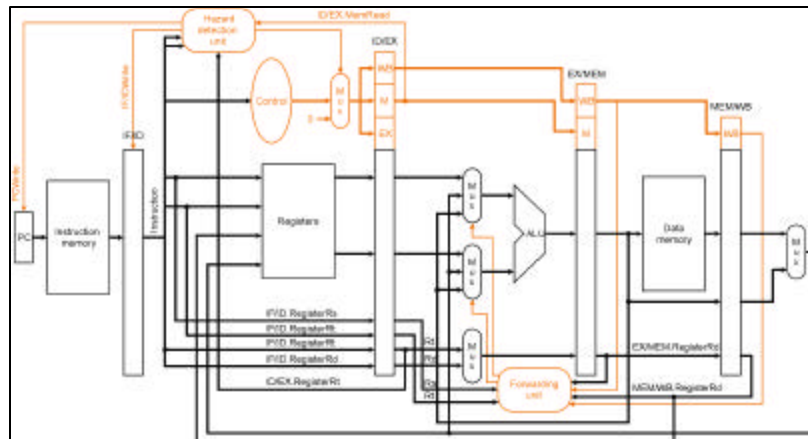
CSE 141

Allan Snively

Hazard Detection

and \$4, \$2, \$5

lw \$2, 20(\$1)



CSE 141

Allan Snively

