Pipeline Data Hazards
Dealing With Data Hazards

• In Software
  –

• In Hardware
  –
  –

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

sub $2, $1, $3

nop

and $12, $2, $5

CSE 141, S2'06

Jeff Brown
How Many No-ops?

sub $2, $1,$3
and $4, $2,$5
or $8, $2,$6
add $9, $4,$2
slt $1, $6,$7
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
Dealing with Data Hazards in Hardware
Part II-Pipeline Stalls

sub $2, $1, $3

and $12, $2, $5

or $13, $6, $2

add $14, $2, $2

sw $15, 100($2)
Pipeline Stalls

sub $2, $1, $3
add $12, $3, $5
or $13, $6, $2
add $14, $12, $2
sw $14, 100($2)
Pipeline Stalls

- To ensure proper pipelined execution in light of register dependences, we must:
  - detect the hazard
  - stall the pipeline
Knowing When to Stall

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

- What comparisons tell us when to stall?
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) essentially, 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

How else can we deal with data hazards?
Reducing Data Hazards Through Forwarding

add $2, $3, $4

add $5, $3, $2
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

(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
Eliminating Data Hazards via Forwarding

sub $2, $1, $3
and $6, $2, $5
or $13, $6, $2
add $14, $2, $2
sw $15, 100($2)

CSE 141, S2'06

Jeff Brown
Forwarding in Action

- `add $1, $12, $3`
- `sub $12, $3, $4`
- `add $3, $10, $11`

Memory Access
Write Back

Diagram of processor pipeline stages with forwarding unit and control flow.
Forwarding in Action

Instruction Fetch

\[ \text{add } \$1, \$12, \$3 \quad \text{sub } \$12, \$3, \$4 \quad \text{add } \$3, \$10, \$11 \quad \text{Write Back} \]
Forwarding in Action

Instruction Fetch  Instruction Decode

add $1, $12, $3  sub $12, $3, $4  add $3, $10, $11
Eliminating Data Hazards via Forwarding??

lw $2, 10($1)  
and $12, $2, $5  
or $13, $6, $2  
add $14, $2, $2  
sw $15, 100($2)

CSE 141, S2'06
Eliminating Data Hazards via Forwarding and stalling

lw $2, 10($1)

and $12, $2, $5

or $13, $6, $2

add $14, $2, $2

sw $15, 100($2)

CSE 141, S2'06

Jeff Brown
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
add $9, $3, $6
if (ID/EX.MemRead and
    ((ID/EX.RegisterRt = IF/ID.RegisterRs) or
     (ID/EX.RegisterRt = IF/ID.RegisterRt)))
then stall the pipeline
Hazard Detection

and $4, $2, $5

$Iw$ $2, 20($1)
Hazard Detection

and $4, $2, $5
nop (bubble)

lw $2, 20($1)
Data Hazard Key Points

• Pipelining provides high throughput, but does not handle data dependences easily.
• Data dependences cause *data hazards*.
• Data hazards can be solved by:
  – software (nops)
  – hardware stalling
  – hardware forwarding
• Our processor, and indeed all modern processors, use a combination of forwarding and stalling.
• \[ ET = IC \times CPI \times CT \]