Pipeline Data Hazards

Warning, warning, warning!

Dealing With Data Hazards

• In Software
  – inserting _____________ instructions

• In Hardware
  – inserting ____________ (stalling the pipeline)
  – data forwarding

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

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

Dealing with Data Hazards in Software

How Many No-ops?

```plaintext
sub $2, $1,$3
and $1, $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

Pipeline Stalls

sub $2, $1, $3
add $12, $3, $5
or $13, $6, $2
add $14, $2, $2
sw $15, 100($2)

Dealing with Data Hazards in Hardware
Part II-Pipeline Stalls

Pipeline Stalls

• To insure proper pipeline execution in light of register dependences, we must:
  --
Knowing When to Stall

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

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

The Pipeline

- What comparisons tell us when to stall?

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
Reducing Data Hazards Through Forwarding

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
Forwarding in Action

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

Memory Access
Write Back

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)
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)

Datapath with Hazard-Detection

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

Hazard Detection

Show stalls and forwarding for this code

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

Show stalls and forwarding for this code

try this one...

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