EN164: Design of Computing Systems
Lecture 18: Processor / Pipeline Design

Professor Sherief Reda
http://scale.engin.brown.edu
Electrical Sciences and Computer Engineering
School of Engineering
Brown University
Spring 2011

[ material from Patterson & Hennessy, 4th ed and Harris 1st ed ]
Pipelining hazards

✓ Structural hazards

2. Data hazard
3. Control hazard

✓ Compile-time techniques
✓ Forward data at run time
C. Stall the processor at run time
Data hazards from `lw`

Program execution order (in instructions):

- `lw $2, 20($1)`
- `and $4, $2, $5`
- `or $8, $2, $6`
- `add $9, $4, $2`
- `slt $1, $6, $7`
Forwarding is not going to eliminate all hazards

Program execution order (in instructions)

- $lw \$2, 20(\$1)$
- $and \$4, \$2, \$5$
- $or \$8, \$2, \$6$
- $add \$9, \$4, \$2$
- $sli \$1, \$6, \$7$
C. Data hazard elimination by stalling

Program execution order (in instructions)

- lw $2, 20($1)
- and becomes nop
- and $4, $2, $5
- or $8, $2, $6
- add $9, $4, $2

Time (in clock cycles)
CC 1  CC 2  CC 3  CC 4  CC 5  CC 6  CC 7  CC 8  CC 9  CC 10

Clock cycle wasted – necessary for correctness

Stall inserted here
When to stall? How to stall?

When?

• ID/EX.MemRead is true
• ID/EX.RegisterRt = IF/ID.RegisterRs
  or ID/EX.RegisterRt = IF/ID.RegisterRt

How to insert a bubble?

• Do not update PC or IF/ID (instruction in ID stage is decoded again, instruction in IF stage is fetched again)
• Force control values in ID/EX register to 0  
  1-cycle stall allows MEM to read data for lw before being used
Pipeline with bubble insertion for hazards

S. Reda EN164 Sp ’11

lw  $2, 20($1)
and $4, $2,$5
or  $4, $4,$2
add $9, $4,$2
Cycle 2

and $4, 2, 5$

lw $2, 20($1)

1w $2, 20($1)

before<1>

before<2>

before<3>

lw $2, 20($1)

and $4, 2, 5$

or $4, 4, 2$

add $9, 4, 2$
Cycle 3

lw $2, 20($1)
and $4, $2, $5
or $4, $2, $5
add $9, $4, $2

or $4, $4, $2
and $4, $2, $5

Hazard detection unit

Clock 3
Cycle 4

lw $2, 20($1)
and $4, $2, $5
or $4, $4, $2
add $9, $4, $2
Cycle 5

lw $2, 20($1)
and $4, $2, $5
or $4, $4, $2
add $9, $4, $2

Bubble

Clock 5
Cycle 6

lw $2, 20($1)
and $4, $2,$5
or $4, $4,$2
add $9, $4,$2
Cycle 7

```
lw   $2, 20($1)
and  $4,  $2,$5
or   $4,  $4,$2
add  $9,  $4,$2
```
Data hazard summary

- Compiler can arrange code to avoid hazards and stalls. Requires knowledge of the pipeline structure.

- Forwarding can sometimes avoid stalls at the expense of extra hardware complexity.

- Stalls reduce performance by increasing the average cycles per instruction (CPI). But sometimes are absolutely necessary to get correct results.
Pipeline design reminder
3. Control hazards

If branch evaluation does not occur until the MEM pipeline stage then 3 cycles could be wasted. Either stall be default or predict branch target and flush if necessary.
Reduced branch penalty

- If we move hardware (target address adder and register comparator) that determine branch outcome to ID stage (instead of MEM) then only 1 instruction needs to be flushed or stalled.
Pipeline modification for earlier branch evaluation

- MIPS architecture (beq and bne) – and RISC architectures in general were designed for fast single-cycle branches with a small branch penalty. Hypothetical instructions like bgeq or bseq require bulkier comparators with greater propagation delay.