Lecture 19: Pipelining

• Today’s topics:
  - Hazards and instruction scheduling
  - Branch prediction
  - Out-of-order execution
Problem 1

add $1, $2, $3

lw $4, 8($1)
Problem 2

```
lw $1, 8($2)
lw $4, 8($1)
```
Problem 3

```
lw    $1, 8($2)
sw    $1, 8($3)
```
Problem 4

A 7 or 9 stage pipeline

lw  $1, 8($2)

add  $4, $1, $3
Problem 4

Without bypassing: 4 stalls

With bypassing: 2 stalls

lw  $1, 8($2)
add  $4, $1, $3
Control Hazards

- Simple techniques to handle control hazard stalls:
  - for every branch, introduce a stall cycle (note: every 6th instruction is a branch!)
  - assume the branch is not taken and start fetching the next instruction – if the branch is taken, need hardware to cancel the effect of the wrong-path instruction
  - fetch the next instruction (branch delay slot) and execute it anyway – if the instruction turns out to be on the correct path, useful work was done – if the instruction turns out to be on the wrong path, hopefully program state is not lost
  - make a smarter guess and fetch instructions from the expected target
Branch Delay Slots

a. From before

```
add $s1, $s2, $s3
if $s2 = 0 then
   Delay slot
```

Becomes

```
if $s2 = 0 then
   add $s1, $s2, $s3
```

b. From target

```
sub $t4, $t5, $t6
   ...
add $s1, $s2, $s3
if $s1 = 0 then
   Delay slot
```

Becomes

```
add $s1, $s2, $s3
if $s1 = 0 then
   sub $t4, $t5, $t6
```
Pipeline without Branch Predictor

IF (br)

PC

PC + 4

Reg Read
Compare
Br-target
Pipeline with Branch Predictor

IF (br) → Branch Predictor → PC → Reg Read Compare Br-target
2-Bit Prediction

- For each branch, maintain a 2-bit saturating counter:
  - if the branch is taken: counter = min(3,counter+1)
  - if the branch is not taken: counter = max(0,counter-1)
  … sound familiar?

- If (counter >= 2), predict taken, else predict not taken

- The counter attempts to capture the common case for each branch
Bimodal Predictor

14 bits

Branch PC

Table of 16K entries of 2-bit saturating counters
Slowdowns from Stalls

- Perfect pipelining with no hazards → an instruction completes every cycle (total cycles ~ num instructions) → speedup = increase in clock speed = num pipeline stages

- With hazards and stalls, some cycles (= stall time) go by during which no instruction completes, and then the stalled instruction completes

- Total cycles = number of instructions + stall cycles
Multicycle Instructions

- Multiple parallel pipelines – each pipeline can have a different number of stages

- Instructions can now complete out of order – must make sure that writes to a register happen in the correct order
An Out-of-Order Processor Implementation

Branch prediction and instr fetch

Instr Fetch Queue

R1 ← R1+R2
R2 ← R1+R3
BEQZ R2
R3 ← R1+R2
R1 ← R3+R2

Decode & Rename

Instr 1
Instr 2
Instr 3
Instr 4
Instr 5
Instr 6

Reorder Buffer (ROB)

Instr 1  T1
Instr 2  T2
Instr 3  T3
Instr 4  T4
Instr 5  T5
Instr 6  T6

Register File R1-R32

ALU

Issue Queue (IQ)

T1 ← R1+R2
T2 ← T1+R3
BEQZ T2
T4 ← T1+T2
T5 ← T4+T2

Results written to ROB and tags broadcast to IQ

Results written to ROB and tags broadcast to IQ