Lecture 20: Branches, OOO

Today’s topics:

- Branch prediction
- Out-of-order execution
- (Also see class notes on pipelining, hazards, etc.)
Problem 4 – no Byp

A 7 or 9 stage pipeline, RR and RW take an entire stage

lw $1, 8($2)

add $4, $1, $3
Problem 4 – with Byp

A 7 or 9 stage pipeline, RR and RW take an entire stage

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
Pipelining Example (Recap)

• Unpipelined design: the entire circuit takes 10ns to finish
  Cycle time = 10ns;  Clock speed = 1/10ns = 100 MHz
  CPI = 1 (assuming no stalls)
  Throughput in instructions per second =
    #cycles in a second x instructions-per-cycle =
    \[ 100 \text{ M} \times 1 = 100 \text{ M instrs per second} = 0.1 \text{ BIPS} \] (billion instrs per sec)

• 5-stage pipeline: under ideal conditions, each stage takes 2ns
  Cycle time = 2ns;  Clock speed = 1/2ns = 500 MHz (5x higher)
  CPI = 1 (continuing to assume no stalls)
  Throughput = # cycles in a second x instrs-per-cycle
    \[ = 500 \text{ M} \times 1 = 500 \text{ MIPS} = 0.5 \text{ BIPS} \]
  Under ideal conditions, a 5-stage pipeline gives a 5x speedup.
• 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
Control Hazards
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

Source: H&P textbook
Pipeline without Branch Predictor

- **IF (br)**
- **Reg Read Compare Br-target**
- **PC + 4**

PC → IF (br) → Reg Read Compare Br-target → PC
Pipeline with Branch Predictor

IF (br) → Branch Predictor → Reg Read Compare Br-target →...
Bimodal Predictor

14 bits

Branch PC

Table of 16K entries of 2-bit saturating counters
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

Indexing functions
Multiple branch predictors
History, trade-offs
Slowdowns from Stalls

• Perfect pipelining with no hazards \( \rightarrow \) an instruction completes every cycle (total cycles \( \sim \) num instructions) \( \rightarrow \) 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