Lec-04: Data and Control Hazards for Simple Pipelines

ECE-720-topic4: Innovations in Processor Design

2008t1

Mark Aagaard

Data Hazards

When are hazards detected?

- the process of letting an instruction move from the instruction decode stage (ID) to the execution stage (EX) is called *instruction issue*
- an instruction in the EX stage is said to have been *issued*
- hazards are usually checked during the ID stage
- when a hazard is detected, the instruction will not be issued until the hazard has been resolved

Scheduling: Software vs Hardware

objective: rearrange instructions so that data hazards caused by loads are eliminated or reduced

needed regardless of whether a hardware interlock is provided

- hardware interlocks lead to stall cycles
- in the absence of a hardware interlock, NOP instructions must be put in unschedulable load delay slots

Effectiveness of Scheduling

- without scheduling 42-65% of loads cause a stall
- with scheduling 14-31% of loads cause a stall
- *Note*: other optimizations affect these results. E.g., optimizations which reduce the total number of loads and stores will cause the fraction of loads causing stalls to increase

Data Hazard Hardware

- Detection
  - comparators to check if either source register index is the same as the destination index
- Resolution
  - additional inputs to bypass MUXes on the ALU
  - extra paths from the data output to bypass MUXes

Control Hazards

Reducing Control Hazard Stalls

determine the new PC value ASAP

- add PC to offset field as soon as IR is loaded

determine whether the branch is taken ASAP

- Simple RISC ISAs have test only for zero/non-zero

a pipeline with a one cycle stall

Branch Penalties

the *branch delay* is the length of the duration of the control hazard

an unresolved branch delay leads to a *branch penalty*

typically, the deeper the pipeline, the higher the branch penalty

Branch Instruction Stats

branch/jump instruction frequencies

	unconditional	conditional
DLX	2%	11%
x86	7%	14%
VAX	8%	17%

approx 53% of conditional branches executed are taken

approx 75% of branches executed go forward

Reducing Pipeline Branch Penalties

simple option--do nothing

- stall the pipeline until the target address has been computed and the condition tested

make a prediction

- static
  - architecture implementation
  - compile time
- dynamic

Static Predict-Not-Taken

assume branches are not taken

continue filling pipeline from sequential instructions until branch outcome is determined

if branch is not taken

- a win!

if branch is taken

- *squash* instructions in the pipeline
  - a pipeline bubble

Predict-Not-Taken--Success

	1	2	3	4	5	6	7	8	9
IF									
ID									
EX									
MEM									
WB									

13

Predict-Not-Taken--Fail

	1	2	3	4	5	6	7	8	9
IF									
ID									
EX									
MEM									
WB									

14

Static Predict-Taken (continued)

Problem**: If branch target requires register read, then by the time the target address is computed the branch outcome is also known**

- no benefit

15

Delayed Branch

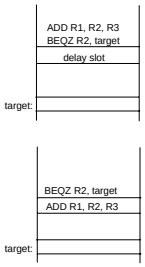
- instead of adding hardware, the architect can simply decree that it is a *feature* of the architecture that the PC will not be modified until, say, one instruction after the branch has completed
  - *delayed branch*
- the slots in the pipeline following a branch during which time the new PC has not been computed are called *branch delay slots*
- it is the responsibility of the programmer or compiler to ensure that branch delay slots are filled with useful work

16

Delayed-Branch Scheduling Strategies

From before branch

- requirements: branch does not depend on the rescheduled instructions
- improves performance always

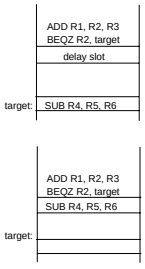


17

Delayed-Branch Scheduling Strategies

From target

- requirements: must be ok to execute scheduled instructions even if branch is not taken
- improves performance when branch is taken

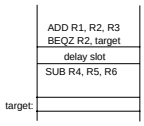


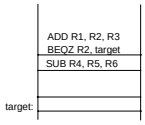
18

Delayed-Branch Scheduling Strategies

From fall through

- requirements: must be ok to execute scheduled instructions when branch is taken
- improves performance when branch not taken





Speedup Statistics

scheduling Scheme	branch penalty	effective CPI	pipeline speedup over unpipelined machine	pipeline speedup over stall pipeline on branch
stall pipeline	3	1.42	3.5	1.0
predict taken	1	1.14	4.4	1.26
predict not taken	1	1.09	4.5	1.29
delayed branch	1.0	1.07	4.6	1.31

Effectiveness of Scheduling

with one delay slot

- delay slots filled by scheduling: 53-60%
- delay slots usefully filled by scheduling: 45-50%
- unfilled slots contain NOPs

with two or more delay slots

- fraction of slots filled decreases dramatically

Performance Implications

Recall

$$\text{pipeline speedup} = \frac{\text{ideal CPI} \times \text{pipeline length}}{\text{ideal CPI} + \#\text{of stall cycles}}$$

Assume CPI = 1

$$\#\text{of stall cycles} = \text{branch frequency} \times \text{branch penality}$$

Assume stalls are due to branches

$$\text{pipeline speedup} = \frac{\text{pipelength}}{1 + \text{branch frequency} \times \text{branch penalty}}$$