CMSC 611: Advanced Computer Architecture

Pipelining

Some material adapted from Mohamed Younis, UMBC CMSC 611 Spr 2003 course slides
Some material adapted from Hennessy & Patterson / © 2003 Elsevier Science
Pipeline Hazards

• Cases that affect instruction execution semantics and thus need to be detected and corrected

• Hazards types
  – **Structural hazard**: attempt to use a resource two different ways at same time
    • Single memory for instruction and data
  – **Data hazard**: attempt to use item before it is ready
    • Instruction depends on result of prior instruction still in the pipeline
  – **Control hazard**: attempt to make a decision before condition is evaluated
    • branch instructions

• Hazards can always be resolved by waiting
Detecting and Resolving Structural Hazard

Time (clock cycles)

Cycle 1 | Cycle 2 | Cycle 3 | Cycle 4 | Cycle 5 | Cycle 6 | Cycle 7

Load

Instr 1

Instr 2

Stall

Instr 3

Slide: David Culler
Stalls & Pipeline Performance

Pipelining Speedup = \frac{\text{Average instruction time unpipelined}}{\text{Average instruction time pipelined}}

= \frac{\text{CPI unpipelined}}{\text{CPI pipelined}} \times \frac{\text{Clock cycle unpipelined}}{\text{Clock cycle pipelined}}

Ideal CPI pipelined = 1

CPI pipelined = \text{Ideal CPI} + \text{Pipeline stall cycles per instruction}

= 1 + \text{Pipeline stall cycles per instruction}

Speedup = \frac{\text{CPI unpipelined}}{1 + \text{Pipeline stall cycles per instruction}} \times \frac{\text{Clock cycle unpipelined}}{\text{Clock cycle pipelined}}

Assuming all pipeline stages are balanced

Speedup = \frac{\text{Pipeline depth}}{1 + \text{Pipeline stall cycles per instruction}}
Data Hazards

Time (clock cycles)

Instr. Order

add r1, r2, r3
sub r4, r1, r3
and r6, r1, r7
or r8, r1, r9
xor r10, r1, r11
Three Generic Data Hazards

• Read After Write (RAW)
  Instr\(_J\) tries to read operand before Instr\(_I\) writes it

  I: add \(r_1, r_2, r_3\)
  J: sub \(r_4, r_1, r_3\)

• Caused by a “Data Dependence” (in compiler nomenclature). This hazard results from an actual need for communication.

Slide: David Culler
Three Generic Data Hazards

- Write After Read (WAR)
  Instr\textsubscript{J} writes operand before Instr\textsubscript{I} reads it

  \begin{align*}
  I: & \text{ sub } r4, r1, r3 \\
  J: & \text{ add } r1, r2, r3 \\
  K: & \text{ mul } r6, r1, r7
  \end{align*}

- Called an “anti-dependence” in compilers.
  - This results from reuse of the name “r1”.

- Can’t happen in MIPS 5 stage pipeline because:
  - All instructions take 5 stages, and
  - Reads are always in stage 2, and
  - Writes are always in stage 5
Three Generic Data Hazards

• Write After Write (WAW)
  Instr\textsubscript{j} writes operand before Instr\textsubscript{i} writes it.

\[
\begin{align*}
  I: & \quad \text{mul } r1,r4,r3 \\
  J: & \quad \text{add } r1,r2,r3 \\
  K: & \quad \text{sub } r6,r1,r7
\end{align*}
\]

• Called an “output dependence” in compilers
  – This also results from the reuse of name “r1”.

• Can’t happen in MIPS 5 stage pipeline:
  – All instructions take 5 stages, and
  – Writes are always in stage 5

• Do see WAR and WAW in more complicated pipes
Forwarding to Avoid Data Hazard

Time (clock cycles)

Instr. Order

add r1, r2, r3
sub r4, r1, r3
and r6, r1, r7
or  r8, r1, r9
xor r10, r1, r11
HW Change for Forwarding
Data Hazard Even with Forwarding

Time (clock cycles)

I n s t r. O r d e r

lw r1, 0(r2)

sub r4, r1, r6

and r6, r1, r7

or r8, r1, r9

Slide: David Culler
Resolving Load Hazards

- Adding hardware? How? Where?
- Detection?
- Compilation techniques?
- What is the cost of load delays?
Resolving the Load Data Hazard

How is this different from the instruction issue stall?

lw r1, 0(r2)
sub r4, r1, r6
and r6, r1, r7
or r8, r1, r9

Slide: David Culler
Software Scheduling to Avoid Load Hazards

Try producing fast code for

\[
\begin{align*}
a &= b + c; \\
d &= e - f;
\end{align*}
\]

assuming \(a\), \(b\), \(c\), \(d\), \(e\), and \(f\) in memory.

**Slow code:**

- \(LW\) \(Rb\), \(b\)
- \(LW\) \(Rc\), \(c\)
- \(ADD\) \(Ra\), \(Rb\), \(Rc\)
- \(SW\) \(a\), \(Ra\)
- \(LW\) \(Re\), \(e\)
- \(LW\) \(Rf\), \(f\)
- \(SUB\) \(Rd\), \(Re\), \(Rf\)
- \(SW\) \(d\), \(Rd\)

**Fast code:**

- \(LW\) \(Rb\), \(b\)
- \(LW\) \(Rc\), \(c\)
- \(ADD\) \(Ra\), \(Rb\), \(Rc\)
- \(SW\) \(a\), \(Ra\)
- \(LW\) \(Re\), \(e\)
- \(LW\) \(Rf\), \(f\)
- \(SUB\) \(Rd\), \(Re\), \(Rf\)
- \(SW\) \(d\), \(Rd\)
Instruction Set Connection

- What is exposed about this organizational hazard in the instruction set?
- k cycle delay?
  - bad, CPI is not part of ISA
- k instruction slot delay
  - load should not be followed by use of the value in the next k instructions
- Nothing, but code can reduce run-time delays
- MIPS did the transformation in the assembler