250P: Computer Systems Architecture

Lecture 5: Advanced Pipelines

Anton Burtsev
January, 2019
Hazards

• Structural hazards

• Data hazards

• Control hazards
Control Hazards

• Simple techniques to handle control hazard stalls:
  ➢ for every branch, introduce a stall cycle (note: every 6th instruction is a branch on average!)
  ➢ 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 instructions
  ➢ predict the next PC and fetch that instr – if the prediction is wrong, cancel the effect of the wrong-path instructions
  ➢ 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
Branch delay slot

(a) From before
DADD R1, R2, R3
if R2 = 0 then
Delay slot

becomes

if R2 = 0 then
DADD R1, R2, R3

(b) From target
DUSB R4, R5, R6

DADD R1, R2, R3
if R1 = 0 then
Delay slot

becomes

if R1 = 0 then
DADD R1, R2, R3
DSUB R4, R5, R6

(c) From fall-through
DADD R1, R2, R3
if R1 = 0 then
Delay slot
OR R7, R8, R9

DSUB R4, R5, R6

becomes

if R1 = 0 then
DADD R1, R2, R3
DSUB R4, R5, R6

OR R7, R8, R9

DSUB R4, R5, R6
Multicycle Instructions
Effects of Multicycle Instructions

• Potentially multiple writes to the register file in a cycle

• Frequent RAW hazards

• WAW hazards (WAR hazards not possible)

• Imprecise exceptions because of o-o-o instr completion

Note: Can also increase the “width” of the processor: handle multiple instructions at the same time: for example, fetch two instructions, read registers for both, execute both, etc.
Precise Exceptions

• On an exception:
  ➢ must save PC of instruction where program must resume
  ➢ all instructions after that PC that might be in the pipeline must be converted to NOPs (other instructions continue to execute and may raise exceptions of their own)
  ➢ temporary program state not in memory (in other words, registers) has to be stored in memory
  ➢ potential problems if a later instruction has already modified memory or registers

• A processor that fulfils all the above conditions is said to provide precise exceptions (useful for debugging and of course, correctness)
Dealing with these Effects

• Multiple writes to the register file: increase the number of ports, stall one of the writers during ID, stall one of the writers during WB (the stall will propagate)

• WAW hazards: detect the hazard during ID and stall the later instruction

• Imprecise exceptions: buffer the results if they complete early or save more pipeline state so that you can return to exactly the same state that you left at
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

• Slowdown because of stalls = 1/ (1 + stall cycles per instr)
Assume that there is a dependence where the final result of the first instruction is required before starting the second instruction.
Problem 1

- For the following code sequence, show how the instrs flow through the pipeline:
  
  ADD  R3 ← R1, R2
  LD   R7 ← 8[R6]
  ST   R9 → 4[R8]
  BEZ  R4, [R5]
Problem 1

• For the following code sequence, show how the instrs flow through the pipeline:
  
  ADD  R3 ← R1, R2
  LD  R7 ← 8[R6]
  ST  R9 → 4[R8]
  BEZ  R4, [R5]

![Pipeline Diagram]

Time (in clock cycles)
## Pipeline Summary

<table>
<thead>
<tr>
<th>RR</th>
<th>ALU</th>
<th>DM</th>
<th>RW</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD R3 ← R1, R2</td>
<td>Rd R1,R2</td>
<td>R1+R2</td>
<td>--</td>
</tr>
<tr>
<td>BEZ R1, [R5]</td>
<td>Rd R1, R5</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>LD R6 ← 8[R3]</td>
<td>Rd R3</td>
<td>R3+8</td>
<td>Get data</td>
</tr>
<tr>
<td>ST R6 → 8[R3]</td>
<td>Rd R3,R6</td>
<td>R3+8</td>
<td>Wr data</td>
</tr>
</tbody>
</table>
Problem 2

• Convert this C code into equivalent RISC assembly instructions

    a[i] = b[i] + c[i];
Problem 2

• Convert this C code into equivalent RISC assembly instructions

\[ a[i] = b[i] + c[i]; \]

LD  R2, [R1]   # R1 has the address for variable i
MUL  R3, R2, 8   # the offset from the start of the array
ADD  R7, R3, R4   # R4 has the address of a[0]
ADD  R8, R3, R5   # R5 has the address of b[0]
ADD  R9, R3, R6   # R6 has the address of c[0]
LD R10, [R8]       # Bringing b[i]
LD R11, [R9]       # Bringing c[i]
ADD R12, R11, R10  # Sum is in R12
ST  R12, [R7]      # Putting result in a[i]
Problem 3

- Show the instruction occupying each stage in each cycle (no bypassing)
  if I1 is R1+R2→R3  and I2 is  R3+R4→R5  and I3 is R7+R8→R9

<table>
<thead>
<tr>
<th>CYC-1</th>
<th>CYC-2</th>
<th>CYC-3</th>
<th>CYC-4</th>
<th>CYC-5</th>
<th>CYC-6</th>
<th>CYC-7</th>
<th>CYC-8</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td>IF</td>
<td>IF</td>
<td>IF</td>
<td>IF</td>
<td>IF</td>
<td>IF</td>
<td>IF</td>
</tr>
<tr>
<td>D/R</td>
<td>D/R</td>
<td>D/R</td>
<td>D/R</td>
<td>D/R</td>
<td>D/R</td>
<td>D/R</td>
<td>D/R</td>
</tr>
<tr>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
</tr>
<tr>
<td>DM</td>
<td>DM</td>
<td>DM</td>
<td>DM</td>
<td>DM</td>
<td>DM</td>
<td>DM</td>
<td>DM</td>
</tr>
<tr>
<td>RW</td>
<td>RW</td>
<td>RW</td>
<td>RW</td>
<td>RW</td>
<td>RW</td>
<td>RW</td>
<td>RW</td>
</tr>
</tbody>
</table>
Problem 3

- Show the instruction occupying each stage in each cycle (no bypassing)

  if I1 is R1+R2 → R3 and I2 is R3+R4 → R5 and I3 is R7+R8 → R9

<table>
<thead>
<tr>
<th>CYC-1</th>
<th>CYC-2</th>
<th>CYC-3</th>
<th>CYC-4</th>
<th>CYC-5</th>
<th>CYC-6</th>
<th>CYC-7</th>
<th>CYC-8</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF I1</td>
<td>IF I2</td>
<td>IF I3</td>
<td>IF I3</td>
<td>IF I3</td>
<td>IF I4</td>
<td>IF I5</td>
<td>IF I6</td>
</tr>
<tr>
<td>D/R I1</td>
<td>D/R I2</td>
<td>D/R I2</td>
<td>D/R I2</td>
<td>D/R I3</td>
<td>D/R I4</td>
<td>D/R I5</td>
<td>D/R I6</td>
</tr>
<tr>
<td>ALU I1</td>
<td>ALU I2</td>
<td>ALU I1</td>
<td>ALU I2</td>
<td>ALU I2</td>
<td>ALU I3</td>
<td>ALU I3</td>
<td>ALU I4</td>
</tr>
<tr>
<td>DM I1</td>
<td>DM I2</td>
<td>DM I1</td>
<td>DM I2</td>
<td>DM I2</td>
<td>DM I3</td>
<td>DM I3</td>
<td>DM I4</td>
</tr>
</tbody>
</table>
Bypassing: 5-Stage Pipeline

Source: H&P textbook
Problem 4

• Show the instruction occupying each stage in each cycle (with bypassing)
  if I1 is R1+R2→R3 and I2 is R3+R4→R5 and I3 is R3+R8→R9.
  Identify the input latch for each input operand.

<table>
<thead>
<tr>
<th>CYC-1</th>
<th>CYC-2</th>
<th>CYC-3</th>
<th>CYC-4</th>
<th>CYC-5</th>
<th>CYC-6</th>
<th>CYC-7</th>
<th>CYC-8</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td>IF</td>
<td>IF</td>
<td>IF</td>
<td>IF</td>
<td>IF</td>
<td>IF</td>
<td>IF</td>
</tr>
<tr>
<td>D/R</td>
<td>D/R</td>
<td>D/R</td>
<td>D/R</td>
<td>D/R</td>
<td>D/R</td>
<td>D/R</td>
<td>D/R</td>
</tr>
<tr>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
</tr>
<tr>
<td>DM</td>
<td>DM</td>
<td>DM</td>
<td>DM</td>
<td>DM</td>
<td>DM</td>
<td>DM</td>
<td>DM</td>
</tr>
<tr>
<td>RW</td>
<td>RW</td>
<td>RW</td>
<td>RW</td>
<td>RW</td>
<td>RW</td>
<td>RW</td>
<td>RW</td>
</tr>
</tbody>
</table>
Problem 4

- Show the instruction occupying each stage in each cycle (with bypassing) if I1 is R1+R2 → R3 and I2 is R3+R4 → R5 and I3 is R3+R8 → R9.

Identify the input latch for each input operand.

<table>
<thead>
<tr>
<th>CYC-1</th>
<th>CYC-2</th>
<th>CYC-3</th>
<th>CYC-4</th>
<th>CYC-5</th>
<th>CYC-6</th>
<th>CYC-7</th>
<th>CYC-8</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF I1</td>
<td>IF I2</td>
<td>IF I3</td>
<td>IF I4</td>
<td>IF I5</td>
<td>IF</td>
<td>IF</td>
<td>IF</td>
</tr>
<tr>
<td>D/R I1</td>
<td>D/R I2</td>
<td>D/R I3</td>
<td>D/R I4</td>
<td>D/R I5</td>
<td>D/R</td>
<td>D/R</td>
<td>D/R</td>
</tr>
<tr>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
<td>ALU</td>
</tr>
<tr>
<td>DM I1</td>
<td>DM I2</td>
<td>DM I3</td>
<td>DM I4</td>
<td>DM I5</td>
<td>DM I6</td>
<td>DM I7</td>
<td>DM I8</td>
</tr>
<tr>
<td>RW I1</td>
<td>RW I2</td>
<td>RW I3</td>
<td>RW I4</td>
<td>RW I5</td>
<td>RW I6</td>
<td>RW I7</td>
<td>RW I8</td>
</tr>
</tbody>
</table>
Thank you!