EECS 322 Computer Architecture

Pipeline Control,

Data Hazards

and Branch Hazards

Instructor: Francis G. Wolff
wolff@eeecs.cwru.edu
Case Western Reserve University

This presentation uses powerpoint animation: please view show
Models

**Single-cycle model** (non-overlapping)
- The instruction *latency* executes in a single cycle
- Every instruction and clock-cycle must be stretched to the slowest instruction (p.438)

**Multi-cycle model** (non-overlapping)
- The instruction *latency* executes in multiple-cycles
- The clock-cycle must be stretched to the slowest step
- Ability to share functional units within the execution of a single instruction

**Pipeline model** (overlapping, p. 522)
- The instruction *latency* executes in multiple-cycles
- The clock-cycle must be stretched to the slowest step
- The *throughput* is mainly one clock-cycle/instruction
- Gains efficiency by overlapping the execution of multiple instructions, increasing hardware utilization. (p. 377)
Recap: Can pipelining get us into trouble?

- Yes: **Pipeline Hazards**
  - **structural hazards:** attempt to use the same resource two different ways at the same time
    - e.g., multiple memory accesses, multiple register writes
    - solutions:
      - multiple memories (separate instruction & data memory)
      - stretch pipeline
  - **control hazards:** attempt to make a decision before condition is evaluated
    - e.g., any conditional branch
    - solutions: prediction, delayed branch
  - **data hazards:** attempt to use item before it is ready
    - e.g., add r1, r2, r3; sub r4, r1, r5; lw r6, 0(r7); or r8, r6, r9
    - solutions: forwarding/bypassing, stall/bubble
Review: Single-Cycle Datapath

2 adders: PC+4 adder, Branch/Jump offset adder

Harvard Architecture: Separate instruction and data memory
Review: Multi vs. Single-cycle Processor Datapath

Combine adders: add $1\frac{1}{2}$ Mux & 3 temp. registers, A, B, ALUOut
Combine Memory: add 1 Mux & 2 temp. registers, IR, MDR

Single-cycle = 1 ALU + 2 Mem + 4 Muxes + 2 adders + OpcodeDecoders
Multi-cycle = 1 ALU + 1 Mem + 5½ Muxes + 5 Reg (IR,A,B,MDR,ALUOut) + FSM
Multi-cycle Processor Datapath

Single-cycle = 1 ALU + 2 Mem + 4 Muxes + 2 adders + OpcodeDecoders

Multi-cycle = 1 ALU + 1 Mem + 5½ Muxes + 5 Reg (IR, A, B, MDR, ALUOut) + FSM

5x32 = 160 additional FFs for multi-cycle processor over single-cycle processor
213 + 16 = 229 additional FFs for pipeline over multi-cycle processor
Overhead

**Single-cycle model**
- 8 ns Clock (125 MHz), *(non-overlapping)*
- 1 ALU + 2 adders
- 0 Muxes
- 0 Datapath Register bits (Flip-Flops)

**Multi-cycle model**
- 2 ns Clock (500 MHz), *(non-overlapping)*
- 1 ALU + Controller
- 5 Muxes
- 160 Datapath Register bits (Flip-Flops)

**Pipeline model**
- 2 ns Clock (500 MHz), *(overlapping)*
- 2 ALU + Controller
- 4 Muxes
- 373 Datapath + 16 Controlpath Register bits (Flip-Flops)
Pipeline Control: Controlpath Register bits

**Figure 6.29**

- **Instruction Path:**
  - IF/ID
  - ID/EX
  - EX/MEM
  - MEM/WB

- **Control Path:**
  - 9 control bits
  - 5 control bits
  - 2 control bits
### Pipeline Control: Controlpath table

#### Figure 5.20, Single Cycle

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Reg Dst</th>
<th>ALU Src</th>
<th>Mem Reg</th>
<th>Reg Wrt</th>
<th>Mem Red</th>
<th>Mem Wrt</th>
<th>Branch</th>
<th>ALU op1</th>
<th>ALU op0</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-format</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>lw</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>sw</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>beq</td>
<td>X</td>
<td>0</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

#### Figure 6.28

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Reg Dst</th>
<th>ALU Op1</th>
<th>ALU Op0</th>
<th>ALU Src</th>
<th>Branch</th>
<th>Mem Red</th>
<th>Mem Wrt</th>
<th>Reg Wrt</th>
<th>Mem Reg</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-format</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>lw</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>sw</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>beq</td>
<td>X</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>X</td>
</tr>
</tbody>
</table>
Pipeline Hazards

**Pipeline hazards**
- Solution #1 always works (for non-realtime) applications:
  stall, delay & procrastinate!

**Structural Hazards** (i.e. fetching same memory bank)
- Solution #2: partition architecture

**Control Hazards** (i.e. branching)
- Solution #1: stall! but decreases throughput
- Solution #2: guess and back-track
- Solution #3: delayed decision: delay branch & fill slot

**Data Hazards** (i.e. register dependencies)
- Worst case situation
- Solution #2: re-order instructions
- Solution #3: forwarding or bypassing: delayed load
Pipeline Datapath and Controlpath
Figure 6.30
Figure 6.30
Pipeline single stepping

<table>
<thead>
<tr>
<th>Clock</th>
<th>IF/ID</th>
<th>ID/EX</th>
<th>EX/MEM</th>
<th>MEM/WB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&lt;PC, IR&gt;</td>
<td>&lt;PC, A, B, S, Rt, Rd&gt;</td>
<td>&lt;PC, Z, ALU, B, R&gt;</td>
<td>&lt;MDR, ALU, R&gt;</td>
</tr>
<tr>
<td>0</td>
<td>&lt;0,?&gt;</td>
<td>&lt;?,&lt;?,?,?,?&gt;,?&gt;</td>
<td>&lt;?,&lt;?,?,?,?&gt;</td>
<td>&lt;?,&lt;?,?&gt;</td>
</tr>
<tr>
<td>1</td>
<td>&lt;4,lw $10,20($1)&gt;</td>
<td>&lt;0,?&gt;</td>
<td>&lt;?,&lt;?,?,?,?&gt;,?&gt;</td>
<td>&lt;?,&lt;?,?&gt;</td>
</tr>
<tr>
<td>2</td>
<td>&lt;8,sub $11,$2,$3&gt;</td>
<td>&lt;8,C$1→3,C$10→8,20,$10,0&gt;</td>
<td>&lt;0,?&gt;</td>
<td>&lt;?,&lt;?,?&gt;</td>
</tr>
<tr>
<td>3</td>
<td>&lt;12,and $12,$4,$5&gt;</td>
<td>&lt;8,C$2→4,C$3→4,X,$3,$11&gt;</td>
<td>&lt;4+20&lt;&lt;2→84,0,20+3→23,8,$10&gt;&lt;?,&lt;?,?&gt;</td>
<td>&lt;0,?&gt;,&lt;?,?,?,?&gt;,?&gt;</td>
</tr>
<tr>
<td>4</td>
<td>&lt;16,or $13,$6,$7&gt;</td>
<td>&lt;12,C$4→6,C$5→7,X,$5,$12&gt;&lt;X,1,4-4=0,4,$11&gt;</td>
<td>Mem[23]→9,23,$10</td>
<td>&lt;0,?&gt;,&lt;?,?,?,?&gt;,?&gt;</td>
</tr>
<tr>
<td>5</td>
<td>&lt;20,add $14,$8,$9&gt;</td>
<td>&lt;16,C$6,C$7,X,$7,$13&gt;</td>
<td>&lt;X,0,1,7,$12&gt;</td>
<td>X,0,$11&gt;</td>
</tr>
</tbody>
</table>

Contents of Register 1 = C$1 = 3; C$2=4; C$3=4; C$4=6; C$5=7; C$10=8; … Memory[23]=9;

Formats: add $rd,$rs=A,$rt=B; lw $rt=B,@($rs=A)
**Clock 1: Figure 6.31a**

- **IF**: `lw $10, 20($1)`
- **ID**: before<1>
- **EX**: before<2>
- **MEM**: before<3>
- **WB**: before<4>

**Clock 1**

- `PC=4` - Instruction memory
- `IR=lw $10, 20($1)`

**Control**

- `PC=0`
- `Address`
- `Instruction memory`

**IF/ID**

- Control
- `00`
- `000`
- `0000`

**ID/EX**

- `00`
- `00`
- `M`
- `0`
- `0`
- `EX`

**EX/MEM**

- `00`
- `00`
- `00`
- `M`
- `0`
- `0`

**MEM/WB**

- `00`
- `00`
- `00`
- `1`
- `1`
- `WB`
- `WB`
- `WB`
- `WB`
- `WB`

**ALU**

- Add
- Add extend
- Shift left 2
- ALU result
- ALU control
- ALUOp
- Zero
- ALUSrc

**RegWrite**

- Read register 1
- Read register 2
- Write register
- Write data
- Instruction [15–0]
- Instruction [20–16]
- Instruction [15–11]

**MemRead**

- Address
- Write data
- Data memory

**MemWrite**

- Branch
- MemRead
- MemWrite
- 1

**MemReg**

- 0

**Clock 1: Figure 6.31a**

- IR = `lw $10, 20($1)`

- PC = 4

- Control signals:
  - `Control` signal to ALU and other components.

- ALU operations:
  - ADD operation for generating the result.
  - Shift left 2 operation on the result.

- Register write signals:
  - RegWrite signals to write the result to memory.

- Address calculation:
  - Address calculation for accessing memory.

- Data memory access:
  - Access to data memory through MemRead and MemWrite signals.

- Branching:
  - Branching condition is evaluated based on the ALU result.

- Instruction execution flow:
  - IF stage for fetching the instruction.
  - ID stage for identifying the opcode and operands.
  - EX stage for executing ALU operations.
  - MEM stage for memory operations.
  - WB stage for writing back the result to registers.

- Clock signals:
  - Clock signals for each stage, indicating the progression through the pipeline.

- Pipeline stages:
  - IF: Instruction fetch
  - ID: Instruction decode
  - EX: Execute
  - MEM: Memory access
  - WB: Write back
Figure 6.31b
Figure 6.32a
Clock 4: Figure 6.32b

IF: or $13, $6, $7
ID: and $12, $2, $3
MEM: lw $10, ...

PC=12
A=C$4
B=C$5
S=X

ALU=20+C$1
MDR=Mem[20+C$1]
D=$10

ALU=20+C$1
MDR=Mem[20+C$1]
D=$10

Clock 4
Data Dependencies: that can be resolved by forwarding

Figure 6.36

Data Hazards

Resolved by forwarding

At same time: Not a hazard

Forward in time: Not a hazard

Value of register $2:$

<table>
<thead>
<tr>
<th>CC 1</th>
<th>CC 2</th>
<th>CC 3</th>
<th>CC 4</th>
<th>CC 5</th>
<th>CC 6</th>
<th>CC 7</th>
<th>CC 8</th>
<th>CC 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>-20</td>
<td>-20</td>
<td>-20</td>
<td>-20</td>
<td>-20</td>
<td></td>
</tr>
</tbody>
</table>

Program execution order (in instructions)

sub $2, $1, $3

and $12, $2, $5

or $13, $6, $2

add $14, $2, $2

sw $15, 100($2)
Data Hazards: arithmetic

<table>
<thead>
<tr>
<th>CC 1</th>
<th>CC 2</th>
<th>CC 3</th>
<th>CC 4</th>
<th>CC 5</th>
<th>CC 6</th>
<th>CC 7</th>
<th>CC 8</th>
<th>CC 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10/−20</td>
<td>−20</td>
<td>−20</td>
<td>−20</td>
<td>−20</td>
</tr>
<tr>
<td>Value of register $2$ : 10 10 10 10 10/−20 −20 −20 −20 −20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value of EX/MEM : X X X −20 X X X X X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Value of MEM/WB : X X X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Program execution order (in instructions)

- sub $2, $1, $3
- and $12, $2, $5
- or $13, $6, $2
- add $14, $2, $2
- sw $15, 100($2)

Forwards in time: Can be resolved

At same time: Not a hazard

Figure 6.37
Data Dependencies: no forwarding

**sub $2,$1,$3**

**and $12,$2,$5**

Suppose every instruction is dependant = 1 + 2 stalls = 3 clocks

\[
\text{MIPS} = \frac{\text{Clock}}{\text{CPI}} = \frac{500 \text{ Mhz}}{3} = 167 \text{ MIPS}
\]
Data Dependencies: no forwarding

A dependant instruction will take = 1 + 2 stalls = 3 clocks
An independent instruction will take = 1 + 0 stalls = 1 clocks

Suppose 10% of the time the instructions are dependant?
Averge instruction time = 10%*3 + 90%*1 = 0.10*3 + 0.90*1 = 1.2 clocks

MIPS = \( \text{Clock} = \frac{500 \text{ Mhz}}{1.2} \) = 417 MIPS (10% dependency)

MIPS = \( \text{Clock} = \frac{500 \text{ Mhz}}{3} \) = 167 MIPS (100% dependency)

MIPS = \( \text{Clock} = \frac{500 \text{ Mhz}}{1} \) = 500 MIPS (0% dependency)
Data Dependencies: with forwarding

sub $2,$1,$3

and $12,$2,$5

Detected Data Hazard 1a
ID/EX.$rs = EX/M.$rd

Suppose every instruction is dependant = 1 + 0 stalls = 1 clock

\[ \text{MIPS} = \frac{\text{Clock}}{\text{CPI}} = \frac{500 \text{ Mhz}}{1} = 500 \text{ MIPS} \]
Data Dependencies: Hazard Conditions

**Data Hazard Condition**
occurs whenever a data source needs a previous unavailable result due to a data destination.

**Example**

\[
\begin{align*}
\text{sub} & \quad \text{ID/EX.$rs} \\
& \quad \text{ID/EX.$rt} \\
\text{and} & \quad \text{MEM/WB.$rdest} = \\
& \quad \text{ID/EX.$rs} \\
& \quad \text{ID/EX.$rt}
\end{align*}
\]

**Data Hazard Detection**
is always comparing a destination with a source.

<table>
<thead>
<tr>
<th>Source</th>
<th>Hazard Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID/EX.$rs</td>
<td>1a.</td>
</tr>
<tr>
<td>ID/EX.$rt</td>
<td>1b.</td>
</tr>
<tr>
<td>ID/EX.$rs</td>
<td>2a.</td>
</tr>
<tr>
<td>ID/EX.$rt</td>
<td>2b.</td>
</tr>
</tbody>
</table>
Data Dependencies: Hazard Conditions

1a Data Hazard:
sub $2, $1, $3
and $12, $2, $5

EX/MEM.$rd = ID/EX.$rs
sub $rd, $rs, $rt
and $rd, $rs, $rt

1b Data Hazard:
sub $2, $1, $3
and $12, $1, $2

EX/MEM.$rd = ID/EX.$rt
sub $rd, $rs, $rt
and $rd, $rs, $rt

2a Data Hazard:
sub $2, $1, $3
and $12, $1, $5
or $13, $2, $1

MEM/WB.$rd = ID/EX.$rs
sub $rd, $rs, $rt
sub $rd, $rs, $rt
and $rd, $rs, $rt

2b Data Hazard:
sub $2, $1, $3
and $12, $1, $5
or $13, $6, $2

MEM/WB.$rd = ID/EX.$rt
sub $rd, $rs, $rt
sub $rd, $rs, $rt
and $rd, $rs, $rt
Data Dependencies: Worst case

**Data Hazard:**

\[
\text{sub} \quad \$2, \quad \$1, \quad \$3 \quad \text{sub} \quad \$rd, \quad \$rs, \quad \$rt
\]

\[
\text{and} \quad \$12, \quad \$2, \quad \$2 \quad \text{and} \quad \$rd, \quad \$rs, \quad \$rt
\]

\[
\text{or} \quad \$13, \quad \$2, \quad \$2 \quad \text{and} \quad \$rd, \quad \$rs, \quad \$rt
\]

**Data Hazard 1a:**  \( \text{EX/MEM.} \$rd = \text{ID/EX.} \$rs \)

**Data Hazard 1b:**  \( \text{EX/MEM.} \$rd = \text{ID/EX.} \$rt \)

**Data Hazard 2a:**  \( \text{MEM/WB.} \$rd = \text{ID/EX.} \$rs \)

**Data Hazard 2b:**  \( \text{MEM/WB.} \$rd = \text{ID/EX.} \$rt \)
Data Dependencies: Hazard Conditions

<table>
<thead>
<tr>
<th>Hazard Type</th>
<th>Source</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>1a.</td>
<td>ID/EX.$rs</td>
<td>EX/MEM.$rdest</td>
</tr>
<tr>
<td>1b.</td>
<td>ID/EX.$rt</td>
<td>EX/MEM.$rdest</td>
</tr>
<tr>
<td>2a.</td>
<td>ID/EX.$rs</td>
<td>MEM/WB.$rdest</td>
</tr>
<tr>
<td>2b.</td>
<td>ID/EX.$rt</td>
<td>MEM/WB.$rdest</td>
</tr>
</tbody>
</table>

Pipeline Registers:

- ID/EX: $rs, $rt, $rd
- EX/MEM: $rd
- MEM/WB: $rd
Figure 6.38
Data Hazards: Loads

Figure 6.44
Data Hazards: load stalling

Program execution order (in instructions)

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Time (in clock cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw $2, 20($1)</td>
<td>CC 1</td>
</tr>
<tr>
<td>and $4, $2, $5</td>
<td>CC 2</td>
</tr>
<tr>
<td>or $8, $2, $6</td>
<td>CC 4</td>
</tr>
<tr>
<td>add $9, $4, $2</td>
<td>CC 5</td>
</tr>
<tr>
<td>slt $1, $6, $7</td>
<td>CC 7</td>
</tr>
</tbody>
</table>

Figure 6.45
Data Hazards: Hazard detection unit (page 490)

Stall Condition

Source \[ \begin{align*} 
\text{IF/ID.} & .rs \\
\text{IF/ID.} & .rt 
\end{align*} \]

\[
\Rightarrow \quad \text{Destination} \quad \text{ID/EX.} .rt \land \text{ID/EX.} .\text{MemRead}=1
\]

Stall Example

\[
\begin{array}{ccc}
\text{lw} & \$2, & 20(\$1) \\
\text{and} & \$4, & \$2, \$5 \\
\end{array}
\quad \begin{array}{ccc}
\text{lw} & \text{\$rt, addr(\$rs)} \\
\text{and} & \text{\$rd, $rs, $rt} \\
\end{array}
\]

No Stall Example: *(only need to look at next instruction)*

\[
\begin{array}{ccc}
\text{lw} & \$2, & 20(\$1) \\
\text{and} & \$4, & \$1, \$5 \\
\text{or} & \$8, & \$2, \$6 \\
\text{lw} & \text{\$rt, addr(\$rs)} \\
\text{and} & \text{\$rd, $rs, $rt} \\
\text{or} & \text{\$rd, $rs, $rt} \\
\end{array}
\]
Data Hazards: Hazard detection unit (page 490)

**No Stall Example:** *(only need to look at next instruction)*

\[
\begin{align*}
\text{lw} & \quad $2,\ 20($1) & \quad \text{lw} & \quad $rt,\ \text{addr($rs)} \\
\text{and} & \quad $4,\ $1,\ $5 & \quad \text{and} & \quad $rd,\ $rs,\ $rt \\
\text{or} & \quad $8,\ $2,\ $6 & \quad \text{or} & \quad $rd,\ $rs,\ $rt
\end{align*}
\]

**Example**
load: assume half of the instructions are immediately followed by an instruction that uses it.

What is the average number of clocks for the load?

load instruction time: \[50\% \times (1 \text{ clock}) + 50\% \times (2 \text{ clocks}) = 1.5\]
Hazard Detection Unit: when to stall

Figure 6.46
Data Dependency Units

Forwarding Condition

\[
\begin{align*}
\text{Source} & \quad \text{Destination} \\
\text{ID/EX.} & \quad \text{EX/MEM.} \\
\text{ID/EX.} & \quad \text{EX/} \\
\text{ID/EX.} & \quad \text{MEM/WB.} \\
\end{align*}
\]

\[
\begin{align*}
\text{ID/EX.} & \quad \text{EX/MEM.} \\
\text{ID/EX.} & \quad \text{EX/} \\
\text{ID/EX.} & \quad \text{MEM/WB.} \\
\end{align*}
\]

Stall Condition

\[
\begin{align*}
\text{Source} & \quad \text{Destination} \\
\text{IF/ID.} & \quad \text{ID/EX.} \\
\text{IF/ID.} & \quad \text{ID/EX.} \\
\end{align*}
\]

\[
\begin{align*}
\text{IF/ID.} & \quad \text{ID/EX.} \\
\text{IF/ID.} & \quad \text{ID/EX.} \\
\end{align*}
\]

\[\text{ID/EX.} \land \text{ID/EX. MemRead} = 1\]
Data Dependency Units

Pipeline Registers

Stalling Comparisons

Forwarding Comparisons

\[ \begin{align*}
\text{Stall Condition} & \quad \text{Source} \\
\text{IF/ID.$rs} & \quad \text{IF/ID.$rt} \\
\text{IF/ID.$rt} & \quad \text{Destination} \\
& \quad \{ \text{IF/EX.$rs} \land \text{IF/EX.$rt} \} = \text{ID/EX.$rt} \land \text{ID/EX.MemRead=1} 
\end{align*} \]
Branch Hazards: Soln #1, Stall until Decision made (fig. 6.4)

@3C: add $4, $5, $6
@40: beq $1, $3, 7
@44: and $12, $2, $5
@48: or $13, $6, $2
@4C: add $14, $2, $2
@50: lw $4, 50($7)

Program execution order (in instructions)
add $4, $5, $6
beq $1, $2, 40
lw $3, 300($0)

Time
2 4 6 8 10 12 14 16

Instruction fetch
Reg ALU Data access Reg
Instruction fetch Reg ALU Data access Reg
Instruction fetch

Stall
Decision made in ID stage: do load
Branch Hazards: Soln #2, Predict until Decision made

beq $1,$3,7
and $12,$2,$5

Predict false branch

discard “and $12,$2,$5” instruction

lw $4, 50($7)

Decision made in ID stage: discard & branch
Branch Hazards: Soln #3, Delayed Decision

beq $1,$3,7

add $4,$6,$6

lw $4, 50($7)

Move instruction before branch

Do not need to discard instruction

Decision made in ID stage: branch
Branch Hazards: Soln #3, Delayed Decision

beq $1,$3,7

and $12, $2, $5

lw $4, 50($7)

Decision made in ID stage: do branch
Branch Hazards: Decision made in the ID stage (figure 6.4)

beq $1,$3,7

nop

No decision yet: insert a nop

Iw $4, 50($7)
Branch Hazards: Soln #2, Predict until Decision made

Program execution order

Time (in clock cycles)

CC 1    CC 2    CC 3

Predict false branch

40 beq $1, $3, 7

44 and $12, $2, $5

48 or $13, $6, $2

52 add $14, $2, $2

72 lw $4, 50($7)

Same effect as 3 stalls

Branch Decision made in MEM stage: Discard values when wrong prediction

Branch Decision made in MEM stage:
Discard values when wrong prediction

Figure 6.50
Figure 6.51

Flush: if wrong prediction, add nops

Early branch comparison

Hazard detection unit

Instruction memory

ALU

Shift left 2

Registers

Control
Performance

load: assume half of the instructions are immediately followed by an instruction that uses it (i.e. data dependency)
load instruction time = 50%*(1 clock) + 50%*(2 clocks) = 1.5

Jump: assume that jumps always pay 1 full clock cycle delay (stall). Jump instruction time = 2

Branch: the branch delay of misprediction is 1 clock cycle that 25% of the branches are mispredicted.

branch time = 75%*(1 clocks) + 25%*(2 clocks) = 1.25
Performance, page 504

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Single-Cycle</th>
<th>Multi-Cycle Clocks</th>
<th>Pipeline Cycles</th>
<th>Instruction Mix</th>
</tr>
</thead>
<tbody>
<tr>
<td>loads</td>
<td>1</td>
<td>5</td>
<td>1.5</td>
<td>23%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(50% dependancy)</td>
<td></td>
</tr>
<tr>
<td>stores</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>13%</td>
</tr>
<tr>
<td>arithmetic</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>43%</td>
</tr>
<tr>
<td>branches</td>
<td>1</td>
<td>3</td>
<td>1.25</td>
<td>19%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(25% dependancy)</td>
<td></td>
</tr>
<tr>
<td>jumps</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>2%</td>
</tr>
<tr>
<td>Clock speed</td>
<td>125 Mhz 8 ns</td>
<td>500 Mhz 2 ns</td>
<td>500 Mhz 2 ns</td>
<td></td>
</tr>
<tr>
<td>CPI</td>
<td>1</td>
<td>4.02</td>
<td>1.18</td>
<td>= Σ Cycles*Mix</td>
</tr>
<tr>
<td>MIPS</td>
<td>125 MIPS</td>
<td>125 MIPS</td>
<td>424 MIPS</td>
<td>= Clock/CPI</td>
</tr>
</tbody>
</table>

Also known as the instruction latency with in a pipeline

Pipeline throughput

Load instruction time = 50%*(1 clock) + 50%*(2 clocks) = 1.5
Branch time = 75%*(1 clocks) + 25%*(2 clocks) = 1.25