EECS 322 Computer Architecture

Pipeline Control,

Data Hazards

and Branch Hazards
Models

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

**Multi-cycle model** (non-overlapping)
- Each instruction executes in variable number of 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)
- Each instruction executes in several cycles
- The clock-cycle must be stretched to the slowest step
- Gains efficiency by overlapping the execution of multiple instructions, increasing hardware utilization. (p. 377)
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)
Figure 6.25
Pipeline Control: Controlpath Register bits

Figure 6.29
## Pipeline Control: Controlpath Register bits

### 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>Bra- nch</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>Bra- nch</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: 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
Figure 6.30
### Pipeline Datapath and Controlpath

<table>
<thead>
<tr>
<th>Clock</th>
<th>Instruction</th>
<th>Next Instruction</th>
<th>Registers</th>
<th>Memory Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>&lt;0,?&gt;</td>
<td>&lt;?,?,?,?,??&gt;</td>
<td>&lt;?,?,?,?,?&gt;</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>&lt;4,lw $10,20($1)&gt;</td>
<td>&lt;0,?,?,?,??&gt;</td>
<td>&lt;?,?,?,?,?&gt;</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>&lt;8,sub $11,$2,$3&gt;</td>
<td>&lt;4,C$1→3,C$10→8,20,$10,0&gt;</td>
<td>&lt;0,?,?,?,?&gt;</td>
<td></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;</td>
<td>&lt;?,?,?,?&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;</td>
<td>X,1,4-4=0,4,$11</td>
<td>Mem[23]→9,23,$10</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>X,0,1,7,$12</td>
<td>X,0,$11</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
Figure 6.31b
Clock 4: Figure 6.32b

IF: or $13, $6, $7
ID: and $12, $2, $3
MEM: lw $10, . . .
WB: before<1>

32 April 10, 2000
Data Dependencies: that can be resolved by forwarding

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)

Value of register $2:

- CC 1: 10
- CC 2: 10
- CC 3: 10
- CC 4: —
- CC 5: —
- CC 6: —
- CC 7: —
- CC 8: 20
- CC 9: 20

Time (in clock cycles)

Data Dependencies

Data Hazards

Resolved by forwarding

At same time: Not a hazard

Forward in time: Not a hazard

Figure 6.36
Data Hazards: arithmetic

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)

Value of register $2:

<table>
<thead>
<tr>
<th>Time (in clock cycles)</th>
<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>Value of EX/MEM:</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>−20</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Value of MEM/WB:</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

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

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

\[
\text{MIPS} = \frac{\text{Clock}}{\text{CPI}} = \frac{500 \text{ Mhz}}{1} = 500 \text{ MIPS} \quad (0\% \text{ dependency})
\]
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**
```
sub $2, $1, $3 sub $rd, $rs, $rt
and $12, $2, $5 and $rd, $rs, $rt
```

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

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

sub $2, $1, $3  sub $rd, $rs, $rt
and $12, $2, $2  and $rd, $rs, $rt
or $13, $2, $2  and $rd, $rs, $rt

Data Hazard 1a: EX/MEM.$rd = ID/EX.$rs
Data Hazard 1b: EX/MEM.$rd = ID/EX.$rt
Data Hazard 2a: MEM/WB.$rd = ID/EX.$rs
Data Hazard 2b: MEM/WB.$rd = ID/EX.$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

(a. No forwarding)

(b. With forwarding)
Data Hazards: Loads

Program execution order (in instructions)

lw $2, 20($1)
and $4, $2, $5
or $8, $2, $6
add $9, $4, $2
slt $1, $6, $7

Time (in clock cycles)

CC 1 CC 2 CC 8 CC 9

Backwards in time: Cannot be resolved
Forwards in time: Can be resolved
At same time: Not a hazard

Figure 6.44
Data Hazards: load stalling

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

### Stall Condition

<table>
<thead>
<tr>
<th>Source</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF/ID.$rs</td>
<td>= ID/EX.$rt \land ID/EX.MemRead=1</td>
</tr>
<tr>
<td>IF/ID.$rt</td>
<td></td>
</tr>
</tbody>
</table>

### Stall Example

- **lw** $2$, 20($1)  
- **and** $4$, $2$, $5$  
- **lw** $rt$, addr($rs)$
- **and** $rd$, $rs$, $rt$

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

- **lw** $2$, 20($1)$  
- **and** $4$, $1$, $5$  
- **or** $8$, $2$, $6$
- **lw** $rt$, addr($rs)$
- **and** $rd$, $rs$, $rt$
- **or** $rd$, $rs$, $rt$
Data Hazards: Hazard detection unit (page 490)

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

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

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

Source
ID/EX.$rs
ID/EX.$rt

} = EX/MEM.$rd

Destination

ID/EX.$rs
ID/EX.$rt

} = MEM/WB.$rd

Stall Condition

Source
IF/ID.$rs
IF/ID.$rt

} = ID/EX.$rt \land ID/EX.MemRead=1

Destination
Data Dependency Units

Pipeline Registers

Stalling Comparisons

Forwarding Comparisons

IF/ID

ID/EX

EX/MEM

MEM/WB

$rs$

$\text{IF/ID.}\$rs$

$\text{ID/EX.}\$rs$

$\text{EX/MEM.}\$rs$

$\text{MEM/WB.}\$rs$

$\text{Source}$

$\text{Destination}$

$rt$

$\text{IF/ID.}\$rt$

$\text{ID/EX.}\$rt$

$\text{EX/MEM.}\$rt$

$\text{MEM/WB.}\$rt$

$rd$

$\text{IF/ID.}\$rd$

$\text{ID/EX.}\$rd$

$\text{EX/MEM.}\$rd$

$\text{MEM/WB.}\$rd$

\{ \text{IF/ID.}\$rs \land \text{ID/EX.}\$rt \land \text{ID/EX.}\text{MemRead}=1 \}

\{ \text{Source} \}$

\{ \text{Destination} \}$
Branch Hazards: Soln #1, Stall until Decision made (fig. 6.4)

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

Soln #1: Stall until Decision is made

Program execution order (in instructions)

Instruction fetch  Reg  ALU  Data access  Reg

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

Iw $4, 50($7)

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

Clock

1 2 3 4 5 6 7 8

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

\[
\text{beq } \$1, \$3, 7
\]

and \(\$12, \$2, \$5\)

\[
\text{lw } \$4, 50(\$7)
\]
Branch Hazards: Decision made in the ID stage (figure 6.4)

Clock

1 2 3 4 5 6 7 8

beq $1,$3,7

nop

No decision yet: insert a nop

lw $4, 50($7)

Decision: do load
Branch Hazards: Soln #2, Predict until Decision made

Program execution order

Time (in clock cycles)

CC 1  CC 2  CC 3  CC 4  CC 5  CC 6  CC 7  CC 8  CC 9

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

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

Figure 6.50

EECS 322 April 10, 2000
Figure 6.51

Flush: if wrong prediction, add nops

Early branch comparison

Hazard detection unit

Forwarding unit

Control

IF/ID

IF/Flush

Instruction memory

PC

Shift left 2

4

Registers

ALU

Data memory

MEM/WB

EX/MEM

EX

M

ID/EX

WB

M

Mux

Mux

Mux

Mux

Mux
Performance

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

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 = \(75\% \times (1 \text{ clocks}) + 25\% \times (2 \text{ clocks}) = 1.25\)
## 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>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>jumps</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>2%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Clock speed</th>
<th>125 Mhz</th>
<th>500 Mhz</th>
<th>500 Mhz</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 ns</td>
<td>2 ns</td>
<td>2 ns</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CPI</th>
<th>1</th>
<th>4.02</th>
<th>1.18</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>MIPS</th>
<th>125 MIPS</th>
<th>125 MIPS</th>
<th>424 MIPS</th>
</tr>
</thead>
</table>

\[
\text{MIPS} = \text{Clock} \div \text{CPI} = \sum \text{Cycles} \times \text{Mix}
\]
1. Convert the RISCEE 1 Architecture into a pipeline Architecture (like Figure 6.30) (showing the number data and control bits).

2. Build the control line table (like Figure 6.28) for the RISCEE3 pipeline architecture for RISCEE1 instructions: (addi, subi, load, store, beq, jmp, jal).

3. Homework 6.23, p534
4. Homework 6.24, p534