Complex Pipelining

Daniel Sanchez
Computer Science and Artificial Intelligence Laboratory
M.I.T.
Complex Pipelining: Motivation

Instruction pipelining becomes complex when we want high performance in the presence of

- Multi-cycle operations, for example:
  - Full or partially pipelined floating-point units, or
  - Long-latency operations, e.g., divides

- Variable-latency operations, for example:
  - Memory systems with variable access time

- Replicated functional units, for example:
  - Multiple floating-point or memory units
CDC 6600
Seymour Cray, 1963

- A fast pipelined machine with 60-bit words
  - 128 Kword main memory capacity, 32 banks
- Ten functional units (parallel, unpipelined)
  - Floating Point: adder, 2 multipliers, divider
  - Integer: adder, 2 incrementers, ...
- Hardwired control
- Dynamic scheduling of instructions using a scoreboard
- Ten Peripheral Processors for Input/Output
  - A fast multi-threaded 12-bit integer ALU
- Very fast clock, 10 MHz (FP add in 4 clocks)
- >400,000 transistors, 750 sq. ft., 5 tons, 150 kW, new freon-based cooling technology
- Fastest machine in world for 5 years (until CDC 7600)
  - Over 100 sold ($7-10M each)
CDC 6600: Datapath

Address Regs
8 x 18-bit

Index Regs
8 x 18-bit

Operand Regs
8 x 60-bit

Central Memory

result

addr

oprnd

10 Functional Units

Inst. Stack
8 x 60-bit

IR

September 25, 2023
CDC 6600: A Load/Store Architecture

- Separate instructions to manipulate three types of registers
  - 8 60-bit data registers (X)
  - 8 18-bit address registers (A)
  - 8 18-bit index registers (B)

- All arithmetic and logic instructions are reg-to-reg

\[
\text{opcode} \quad i \quad j \quad k \quad \text{Ri} \leftarrow (\text{Rj}) \text{ op } (\text{Rk})
\]

- Only Load and Store instructions refer to memory!

\[
\text{opcode} \quad i \quad j \quad \text{disp} \quad \text{Ri} \leftarrow \text{M}[\text{Rj} + \text{disp}]
\]

- Touching address registers 1 to 5 initiates a load
  - 6 to 7 initiates a store
  - very useful for vector operations
CDC6600: Vector Addition

\[ B1 \leftarrow -n \]

loop:

\[ \text{JZE } B1, \text{ exit} \]

\[ A1 \leftarrow B1 + a1 \quad \text{load into } X1 \]
\[ A2 \leftarrow B1 + b1 \quad \text{load into } X2 \]
\[ X6 \leftarrow X1 + X2 \]
\[ A6 \leftarrow B1 + c1 \quad \text{store } X6 \]
\[ B1 \leftarrow B1 + 1 \]

jump loop

\[ Ai = \text{address register} \]
\[ Bi = \text{index register} \]
\[ Xi = \text{data register} \]

more on vector processing later...
We will present complex pipelining issues more abstractly ...
Floating Point ISA

Interaction between the Floating point datapath and the Integer datapath is determined largely by the ISA

RISC-V ISA (with F/D extensions)
- separate register files for FP and Integer instructions
  *the only interaction is via a set of move instructions (some ISAs don’t even permit this)*
- separate load/store for FPRs and GPRs, but both use GPRs for address calculation
- branches only take GPRs as sources; FP compare instructions write result to a GPR, which can then be used in a branch
Floating Point Unit

Much more hardware than an integer unit

Single-cycle floating point unit is a bad idea - why?

• it is common to have several floating point units

• it is common to have different types of FPUs
  \textit{Fadd, Fmul, Fdiv, ...}

• an FPU may be pipelined, partially pipelined or not pipelined

\textit{To operate several FPUs concurrently, the register file needs to have more read and write ports}
Functional Unit Characteristics

- **fully pipelined**
  - Operands are latched when an instruction enters a functional unit.
  - Inputs to a functional unit (e.g., register file) can change during a long latency operation.

- **partially pipelined**
  - Busy state.
  - Accept state.

Functional units have internal pipeline registers.
Realistic Memory Systems

Latency of access to the main memory is usually much higher than one cycle and often unpredictable. *Solving this problem is a central issue in computer architecture.*
Realistic Memory Systems

Latency of access to the main memory is usually much higher than one cycle and often unpredictable. *Solving this problem is a central issue in computer architecture*

Common approaches to improving memory performance

- separate instruction and data memory ports
  - no self-modifying code
- caches
  - single cycle except in case of a miss ⇒ stall
- interleaved memory
  - multiple memory accesses ⇒ bank conflicts
- split-phase memory operations
  - out-of-order responses
Complex Pipeline Structure
Complex Pipeline Control Issues

- Structural hazards at the execution stage if some FPU or memory unit is not pipelined and takes more than one cycle

- Structural hazards at the write-back stage due to variable latencies of different function units

- Out-of-order write hazards due to variable latencies of different function units
Complex Pipeline Control Issues

- Structural hazards at the execution stage if some FPU or memory unit is not pipelined and takes more than one cycle

- Structural hazards at the write-back stage due to variable latencies of different function units

- Out-of-order write hazards due to variable latencies of different function units

- How to handle exceptions?
Complex In-Order Pipeline

- Delay writeback so all operations have same latency to W stage
  - Write ports never oversubscribed (one inst. in & one inst. out every cycle)
Complex In-Order Pipeline

- Delay writeback so all operations have same latency to W stage
  - Write ports never oversubscribed (one inst. in & one inst. out every cycle)

How to prevent increased writeback latency from slowing down single-cycle integer operations?
Complex In-Order Pipeline

- Delay writeback so all operations have same latency to W stage
  - Write ports never oversubscribed (one inst. in & one inst. out every cycle)

How to prevent increased writeback latency from slowing down single-cycle integer operations?

Bypassing
Complex In-Order Pipeline

How should we handle data hazards for long-latency operations?
Complex In-Order Pipeline

How should we handle data hazards for long-latency operations?

Stall pipeline on long latency operations, e.g., divides, cache misses
How should we handle data hazards for long-latency operations?

Stall pipeline on long latency operations, e.g., divides, cache misses

Exceptions?
Complex In-Order Pipeline

How should we handle data hazards for long-latency operations?

Stall pipeline on long latency operations, e.g., divides, cache misses

Exceptions?

Speculate that exceptions won’t occur and detect them and recover in program order at commit point
Superscalar In-Order Pipeline

- Fetch two instructions per cycle; issue both simultaneously if one is integer/memory and other is floating-point

```
PC
Inst. Mem

Dual Decode

GPRs
X1
+
X2

Data Mem
X3

W

2

PC

Mem

D

Dual Decode

GPRs

Data Mem

W

FPRs

X1

Fadd

X3

W

Fmul

X3

Unpipelined divider

FDiv

X2

Commit Point

X3
```
Superscalar In-Order Pipeline

- Fetch two instructions per cycle; issue both simultaneously if one is integer/memory and other is floating-point
- Inexpensive way of increasing throughput
  - Examples: Alpha 21064 (1992)  
    MIPS R5000 series (1996)
Superscalar In-Order Pipeline

- Fetch two instructions per cycle; issue both simultaneously if one is integer/mem and other is floating-point
- Inexpensive way of increasing throughput
- Can be extended to wider issue but register file ports and bypassing costs grow quickly
  - E.g., 4-issue UltraSPARC
Dependence Analysis

Needed to Exploit Instruction-level Parallelism
Types of Data Hazards

Consider executing a sequence of

\[ r_k \leftarrow (r_i) \text{ op } (r_j) \]

type of instructions
Types of Data Hazards

Consider executing a sequence of

\[ r_k \leftarrow (r_i) \text{ op } (r_j) \]

type of instructions

Data-dependence

\[ r_3 \leftarrow (r_1) \text{ op } (r_2) \quad \text{Read-after-Write} \]
\[ r_5 \leftarrow (r_3) \text{ op } (r_4) \quad \text{(RAW) hazard} \]
Types of Data Hazards

Consider executing a sequence of
type of instructions

\[ \begin{align*}
  r_k &\leftarrow (r_i) \text{ op } (r_j) \\
  r_3 &\leftarrow (r_1) \text{ op } (r_2) & \text{Read-after-Write} \\
  r_5 &\leftarrow (r_3) \text{ op } (r_4) & \text{(RAW) hazard}
\end{align*} \]
Types of Data Hazards

Consider executing a sequence of

\[ r_k \leftarrow (r_i) \text{ op } (r_j) \]

type of instructions

**Data-dependence**

\[ r_3 \leftarrow (r_1) \text{ op } (r_2) \]  
Read-after-Write (RAW) hazard

\[ r_5 \leftarrow (r_3) \text{ op } (r_4) \]

**Anti-dependence**

\[ r_3 \leftarrow (r_1) \text{ op } (r_2) \]  
Write-after-Read (WAR) hazard

\[ r_1 \leftarrow (r_4) \text{ op } (r_5) \]
Types of Data Hazards

Consider executing a sequence of

\[ r_k \leftarrow (r_i) \ op \ (r_j) \]

type of instructions

**Data-dependence**

\[
\begin{align*}
    r_3 & \leftarrow (r_1) \ op \ (r_2) & \text{Read-after-Write (RAW) hazard} \\
    r_5 & \leftarrow (r_3) \ op \ (r_4)
\end{align*}
\]

**Anti-dependence**

\[
\begin{align*}
    r_3 & \leftarrow (r_1) \ op \ (r_2) & \text{Write-after-Read (WAR) hazard} \\
    r_1 & \leftarrow (r_4) \ op \ (r_5)
\end{align*}
\]
Types of Data Hazards

Consider executing a sequence of

\[ r_k \leftarrow (r_i) \text{ op } (r_j) \]

type of instructions

---

**Data-dependence**

\[
\begin{align*}
    r_3 &\leftarrow (r_1) \text{ op } (r_2) & \text{(Read-after-Write (RAW) hazard)} \\
    r_5 &\leftarrow (r_3) \text{ op } (r_4)
\end{align*}
\]

---

**Anti-dependence**

\[
\begin{align*}
    r_3 &\leftarrow (r_1) \text{ op } (r_2) & \text{(Write-after-Read (WAR) hazard)} \\
    r_1 &\leftarrow (r_4) \text{ op } (r_5)
\end{align*}
\]

---

**Output-dependence**

\[
\begin{align*}
    r_3 &\leftarrow (r_1) \text{ op } (r_2) & \text{(Write-after-Write (WAW) hazard)} \\
    r_3 &\leftarrow (r_6) \text{ op } (r_7)
\end{align*}
\]
Types of Data Hazards

Consider executing a sequence of
type of instructions

Data-dependence
\[ r_3 \leftarrow (r_1) \text{ op } (r_2) \]
Read-after-Write (RAW) hazard

Anti-dependence
\[ r_3 \leftarrow (r_1) \text{ op } (r_2) \]
Write-after-Read (WAR) hazard

Output-dependence
\[ r_3 \leftarrow (r_1) \text{ op } (r_2) \]
Write-after-Write (WAW) hazard
Detecting Data Hazards

Range and Domain of instruction $i$

$R(i) = \text{ Registers (or other storage) modified by instruction } i$

$D(i) = \text{ Registers (or other storage) read by instruction } i$
Detecting Data Hazards

Range and Domain of instruction \( i \)

\[ R(i) = \text{Registers (or other storage) modified by instruction } i \]

\[ D(i) = \text{Registers (or other storage) read by instruction } i \]

Suppose instruction \( j \) follows instruction \( i \) in the program order. Executing instruction \( j \) before the effect of instruction \( i \) has taken place can cause a

- **RAW hazard** if \( R(i) \cap D(j) \neq \emptyset \)
- **WAR hazard** if \( D(i) \cap R(j) \neq \emptyset \)
- **WAW hazard** if \( R(i) \cap R(j) \neq \emptyset \)
Register vs. Memory Data Dependences

- Data hazards due to register operands can be determined at the decode stage *but*

- Data hazards due to memory operands can be determined only after computing the effective address

```latex
\text{store} \quad M[(a1) + \text{offset1}] \leftarrow (a2) \\
\text{load} \quad a3 \leftarrow M[(a4) + \text{offset2}]
```

Does \((a1) + \text{offset1} == (a4) + \text{offset2}\)?
Register vs. Memory
Data Dependences

• Data hazards due to register operands can be determined at the decode stage \textit{but}

• Data hazards due to memory operands can be determined only after computing the effective address

\begin{align*}
\text{store} & \quad M[(a1) + \text{offset1}] \leftarrow (a2) \\
\text{load} & \quad a3 \leftarrow M[(a4) + \text{offset2}] \\
\end{align*}

\textit{Does (a1) + offset1 == (a4) + offset2 ?}

In lecture 10, we’ll see how to handle memory dependencies
For now, we focus only on register dependencies
## Data Hazards: An Example

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>I₁</td>
<td>FDIV.D</td>
<td>f6,</td>
<td>f6,</td>
</tr>
<tr>
<td>I₂</td>
<td>FLD</td>
<td>f2,</td>
<td>45(r3)</td>
</tr>
<tr>
<td>I₃</td>
<td>FMUL.D</td>
<td>f0,</td>
<td>f2,</td>
</tr>
<tr>
<td>I₄</td>
<td>FDIV.D</td>
<td>f8,</td>
<td>f6,</td>
</tr>
<tr>
<td>I₅</td>
<td>FSUB.D</td>
<td>f10,</td>
<td>f0,</td>
</tr>
<tr>
<td>I₆</td>
<td>FADD.D</td>
<td>f6,</td>
<td>f8,</td>
</tr>
</tbody>
</table>
Data Hazards: An Example

<table>
<thead>
<tr>
<th>$I_1$</th>
<th>FDIV.D</th>
<th>f6, f6, f4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_2$</td>
<td>FLD</td>
<td>f2, 45(r3)</td>
</tr>
<tr>
<td>$I_3$</td>
<td>FMUL.D</td>
<td>f0, f2, f4</td>
</tr>
<tr>
<td>$I_4$</td>
<td>FDIV.D</td>
<td>f8, f6, f2</td>
</tr>
<tr>
<td>$I_5$</td>
<td>FSUB.D</td>
<td>f10, f0, f6</td>
</tr>
<tr>
<td>$I_6$</td>
<td>FADD.D</td>
<td>f6, f8, f2</td>
</tr>
</tbody>
</table>

**RAW Hazards**

**WAR Hazards**

**WAW Hazards**
Data Hazards: An Example

\[ I_1 \: \text{FDIV.D} \: f6, \: f6, \: f4 \]
\[ I_2 \: \text{FLD} \: f2, \: 45(r3) \]
\[ I_3 \: \text{FMUL.D} \: f0, \: f2, \: f4 \]
\[ I_4 \: \text{FDIV.D} \: f8, \: f6, \: f2 \]
\[ I_5 \: \text{FSUB.D} \: f10, \: f0, \: f6 \]
\[ I_6 \: \text{FADD.D} \: f6, \: f8, \: f2 \]

RAW Hazards
WAR Hazards
WAW Hazards
Data Hazards: An Example

\begin{align*}
I_1 & : \text{FDIV.D} & \text{f6, f6, f4} \\
I_2 & : \text{FLD} & \text{f2, 45(r3)} \\
I_3 & : \text{FMUL.D} & \text{f0, f2, f4} \\
I_4 & : \text{FDIV.D} & \text{f8, f6, f2} \\
I_5 & : \text{FSUB.D} & \text{f10, f0, f6} \\
I_6 & : \text{FADD.D} & \text{f6, f8, f2}
\end{align*}

RAW Hazards
WAR Hazards
WAW Hazards
Data Hazards: An Example

\[I_1 \quad \text{FDIV.D} \quad f6, f6, f4\]
\[I_2 \quad \text{FLD} \quad f2, 45(r3)\]
\[I_3 \quad \text{FMUL.D} \quad f0, f2, f4\]
\[I_4 \quad \text{FDIV.D} \quad f8, f6, f2\]
\[I_5 \quad \text{FSUB.D} \quad f10, f0, f6\]
\[I_6 \quad \text{FADD.D} \quad f6, f8, f2\]

- RAW Hazards
- WAR Hazards
- WAW Hazards
Data Hazards: An Example

\begin{align*}
I_1 & : \text{FDIV.D} & f6, & f6, & f4 \\
I_2 & : \text{FLD} & f2, & 45(r3) \\
I_3 & : \text{FMUL.D} & f0, & f2, & f4 \\
I_4 & : \text{FDIV.D} & f8, & f6, & f2 \\
I_5 & : \text{FSUB.D} & f10, & f0, & f6 \\
I_6 & : \text{FADD.D} & f6, & f8, & f2 \\
\end{align*}

\textbf{RAW Hazards}  \\
\textbf{WAR Hazards}  \\
\textbf{WAW Hazards}
Data Hazards: An Example

\[
\begin{align*}
I_1 & : \text{FDIV.D} & \text{f6, f6, f4} \\
I_2 & : \text{FLD} & \text{f2, 45(r3)} \\
I_3 & : \text{FMUL.D} & \text{f0, f2, f4} \\
I_4 & : \text{FDIV.D} & \text{f8, f6, f2} \\
I_5 & : \text{FSUB.D} & \text{f10, f0, f6} \\
I_6 & : \text{FADD.D} & \text{f6, f8, f2}
\end{align*}
\]

RAW Hazards
WAR Hazards
WAW Hazards
Data Hazards: An Example

$I_1$  FDIV.D  f6, f6, f4

$I_2$  FLD  f2, 45(r3)

$I_3$  FMUL.D  f0, f2, f4

$I_4$  FDIV.D  f8, f6, f2

$I_5$  FSUB.D  f10, f0, f6

$I_6$  FADD.D  f6, f8, f2

RAW Hazards
WAR Hazards
WAW Hazards
Data Hazards: An Example

\[
\begin{align*}
I_1 & \quad \text{FDIV.D} & \quad f6, f6, f4 \\
I_2 & \quad \text{FLD} & \quad f2, 45(r3) \\
I_3 & \quad \text{FMUL.D} & \quad f0, f2, f4 \\
I_4 & \quad \text{FDIV.D} & \quad f8, f6, f2 \\
I_5 & \quad \text{FSUB.D} & \quad f10, f0, f6 \\
I_6 & \quad \text{FADD.D} & \quad f6, f8, f2
\end{align*}
\]

\textbf{RAW Hazards}

\textbf{WAR Hazards}

\textbf{WAW Hazards}
Data Hazards: An Example

\[ I_1 \quad \text{FDIV.D} \quad f6, \quad f6, \quad f4 \]
\[ I_2 \quad \text{FLD} \quad f2, \quad 45(r3) \]
\[ I_3 \quad \text{FMUL.D} \quad f0, \quad f2, \quad f4 \]
\[ I_4 \quad \text{FDIV.D} \quad f8, \quad f6, \quad f2 \]
\[ I_5 \quad \text{FSUB.D} \quad f10, \quad f0, \quad f6 \]
\[ I_6 \quad \text{FADD.D} \quad f6, \quad f8, \quad f2 \]

RAW Hazards
WAR Hazards
WAW Hazards
Data Hazards: An Example

$I_1$  FDIV.D  $f_6$, $f_6$, $f_4$

$I_2$  FLD  $f_2$, $45(r3)$

$I_3$  FMUL.D  $f_0$, $f_2$, $f_4$

$I_4$  FDIV.D  $f_8$, $f_6$, $f_2$

$I_5$  FSUB.D  $f_{10}$, $f_0$, $f_6$

$I_6$  FADD.D  $f_6$, $f_8$, $f_2$

RAW Hazards
WAR Hazards
WAW Hazards
Data Hazards: An Example

\[ I_1 \quad \text{FDIV.D} \quad f6, \quad f6, \quad f4 \]
\[ I_2 \quad \text{FLD} \quad f2, \quad 45(r3) \]
\[ I_3 \quad \text{FMUL.D} \quad f0, \quad f2, \quad f4 \]
\[ I_4 \quad \text{FDIV.D} \quad f8, \quad f6, \quad f2 \]
\[ I_5 \quad \text{FSUB.D} \quad f10, \quad f0, \quad f6 \]
\[ I_6 \quad \text{FADD.D} \quad f6, \quad f8, \quad f2 \]

**RAW Hazards**

**WAR Hazards**

**WAW Hazards**
# Instruction Scheduling

<table>
<thead>
<tr>
<th>$I_1$</th>
<th>FDIV.D</th>
<th>f6, f6, f4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_2$</td>
<td>FLD</td>
<td>f2, 45(r3)</td>
</tr>
<tr>
<td>$I_3$</td>
<td>FMUL.D</td>
<td>f0, f2, f4</td>
</tr>
<tr>
<td>$I_4$</td>
<td>FDIV.D</td>
<td>f8, f6, f2</td>
</tr>
<tr>
<td>$I_5$</td>
<td>FSUB.D</td>
<td>f10, f0, f6</td>
</tr>
<tr>
<td>$I_6$</td>
<td>FADD.D</td>
<td>f6, f8, f2</td>
</tr>
</tbody>
</table>

![Diagram of instruction scheduling]
Instruction Scheduling

Valid orderings:
in-order  \( I_1 \quad I_2 \quad I_3 \quad I_4 \quad I_5 \quad I_6 \)
out-of-order
out-of-order
Instruction Scheduling

Valid orderings:
in-order

out-of-order

out-of-order
Instruction Scheduling

Valid orderings:

in-order: $I_1, I_2, I_3, I_4, I_5, I_6$

out-of-order: $I_2, I_1, I_3, I_4, I_5, I_6$

out-of-order: $I_1, I_2, I_3, I_5, I_4, I_6$
# Out-of-order Completion

## In-order Issue

<table>
<thead>
<tr>
<th>Cycle</th>
<th>Instruction</th>
<th>Operands</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FDIV.D</td>
<td>f6, f6, f4</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>FLD</td>
<td>f2, 45(r3)</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>FMUL.D</td>
<td>f0, f2, f4</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>FDIV.D</td>
<td>f8, f6, f2</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>FSUB.D</td>
<td>f10, f0, f6</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>FADD.D</td>
<td>f6, f8, f2</td>
<td>1</td>
</tr>
</tbody>
</table>

**cycle**

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
</table>

**in-order comp**

| 1 | 2 |

**out-of-order comp**

| 1 | 2 |
### Out-of-order Completion

#### In-order Issue

<table>
<thead>
<tr>
<th>$I_1$</th>
<th>FDIV.D</th>
<th>f6, f6, f4</th>
<th>Latency 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_2$</td>
<td>FLD</td>
<td>f2, 45(r3)</td>
<td>1</td>
</tr>
<tr>
<td>$I_3$</td>
<td>FMUL.D</td>
<td>f0, f2, f4</td>
<td>3</td>
</tr>
<tr>
<td>$I_4$</td>
<td>FDIV.D</td>
<td>f8, f6, f2</td>
<td>4</td>
</tr>
<tr>
<td>$I_5$</td>
<td>FSUB.D</td>
<td>f10, f0, f6</td>
<td>1</td>
</tr>
<tr>
<td>$I_6$</td>
<td>FADD.D</td>
<td>f6, f8, f2</td>
<td>1</td>
</tr>
</tbody>
</table>

#### Cycle

<table>
<thead>
<tr>
<th>cycle</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
</table>

#### In-order comp

| in-order comp | 1 | 2 | 1 | 2 | 3 | 4 | 3 | 5 | 4 | 6 | 5 | 6 |

#### Out-of-order comp

| out-of-order comp | 1 | 2 |
# Out-of-order Completion

## In-order Issue

<table>
<thead>
<tr>
<th>Cycle</th>
<th>Instruction</th>
<th>Operands</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FDIV.D f6, f6, f4</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>FLD f2, 45(r3)</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>FMUL.D f0, f2, f4</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>FDIV.D f8, f6, f2</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>FSUB.D f10, f0, f6</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>FADD.D f6, f8, f2</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>in-order comp</th>
<th>out-of-order comp</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2</td>
<td>1 2 2 3 1 4 3 5 5 4 6 6</td>
</tr>
</tbody>
</table>
### Out-of-order Completion

#### In-order Issue

<table>
<thead>
<tr>
<th>Cycle</th>
<th>Instruction</th>
<th>Operands</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FDIV.D</td>
<td>f6, f6, f4</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>FLD</td>
<td>f2, 45(r3)</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>FMUL.D</td>
<td>f0, f2, f4</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>FDIV.D</td>
<td>f8, f6, f2</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>FSUB.D</td>
<td>f10, f0, f6</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>FADD.D</td>
<td>f6, f8, f2</td>
<td>1</td>
</tr>
</tbody>
</table>

#### What problems can out-of-order comp cause?
Out-of-order Completion

In-order Issue

<table>
<thead>
<tr>
<th>cycle</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>in-order comp</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>out-of-order comp</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>6</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

What problems can out-of-order comp cause?

- Structural hazards
- Data hazards
- Exceptions
Scoreboard: A Hardware Data Structure to Detect Hazards Dynamically
Can we solve write hazards without equalizing all pipeline depths and without bypassing?
When is it Safe to Issue an Instruction?

- Approach: Stall issue until sure that issuing will cause no dependence problems...

- Suppose a data structure keeps track of all the instructions in all the functional units

- The following checks need to be made before the Issue stage can dispatch an instruction
  - Is the required function unit available?
  - Is the input data available? ⇒ RAW?
  - Is it safe to write the destination? ⇒ WAR? WAW?
  - Is there a structural conflict at the WB stage?
A Data Structure for Correct Issues

**Keeps track of the status of Functional Units**

<table>
<thead>
<tr>
<th>Name</th>
<th>Busy</th>
<th>Op</th>
<th>Dest</th>
<th>Src1</th>
<th>Src2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mem</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mult1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mult2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Div</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A Data Structure for Correct Issues

Keeps track of the status of Functional Units

<table>
<thead>
<tr>
<th>Name</th>
<th>Busy</th>
<th>Op</th>
<th>Dest</th>
<th>Src1</th>
<th>Src2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mem</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mult1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mult2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Div</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The instruction $i$ at the Issue stage consults this table

- FU available?
- RAW?
- WAR?
- WAW?
A Data Structure for Correct Issues

Keeps track of the status of Functional Units

<table>
<thead>
<tr>
<th>Name</th>
<th>Busy</th>
<th>Op</th>
<th>Dest</th>
<th>Src1</th>
<th>Src2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mem</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mult1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mult2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Div</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The instruction \( i \) at the Issue stage consults this table

FU available? check the busy column
RAW?
WAR?
WAW?
A Data Structure for Correct Issues

Keeps track of the status of Functional Units

<table>
<thead>
<tr>
<th>Name</th>
<th>Busy</th>
<th>Op</th>
<th>Dest</th>
<th>Src1</th>
<th>Src2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mem</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mult1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mult2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Div</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The instruction \(i\) at the Issue stage consults this table

- FU available? check the busy column
- RAW? search the dest column for \(i\)’s sources
- WAR?
- WAW?
A Data Structure for Correct Issues

Keeps track of the status of Functional Units

<table>
<thead>
<tr>
<th>Name</th>
<th>Busy</th>
<th>Op</th>
<th>Dest</th>
<th>Src1</th>
<th>Src2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mem</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mult1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mult2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Div</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The instruction i at the Issue stage consults this table

- **FU available?** check the busy column
- **RAW?** search the dest column for i’s sources
- **WAR?** search the source columns for i’s destination
- **WAW?**
A Data Structure for Correct Issues

Keeps track of the status of Functional Units

<table>
<thead>
<tr>
<th>Name</th>
<th>Busy</th>
<th>Op</th>
<th>Dest</th>
<th>Src1</th>
<th>Src2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mem</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mult1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mult2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Div</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The instruction i at the Issue stage consults this table

- FU available? check the busy column
- RAW? search the dest column for i’s sources
- WAR? search the source columns for i’s destination
- WAW? search the dest column for i’s destination
A Data Structure for Correct Issues

Keeps track of the status of Functional Units

<table>
<thead>
<tr>
<th>Name</th>
<th>Busy</th>
<th>Op</th>
<th>Dest</th>
<th>Src1</th>
<th>Src2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mem</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mult1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mult2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Div</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The instruction $i$ at the Issue stage consults this table

- FU available? check the busy column
- RAW? search the dest column for $i$’s sources
- WAR? search the source columns for $i$’s destination
- WAW? search the dest column for $i$’s destination

An entry is added to the table if no hazard is detected;
An entry is removed from the table after Write-Back
Simplifying the Data Structure Assuming In-order Issue

• Suppose the instruction is not dispatched by the Issue stage
  • If a RAW hazard exists
  • or if the required FU is busy

• Suppose operands are latched by the functional unit on issue

Can the dispatched instruction cause a

WAR hazard?
WAW hazard?
Simplifying the Data Structure
Assuming In-order Issue

• Suppose the instruction is not dispatched by the Issue stage
  • If a RAW hazard exists
  • or if the required FU is busy

• Suppose operands are latched by the functional unit on issue

  Can the dispatched instruction cause a
  WAR hazard?  NO: Operands read at issue
  WAW hazard?
Simplifying the Data Structure
Assuming In-order Issue

• Suppose the instruction is not dispatched by the Issue stage
  • If a RAW hazard exists
  • or if the required FU is busy

• Suppose operands are latched by the functional unit on issue

Can the dispatched instruction cause a

WAR hazard?  NO: Operands read at issue
WAW hazard?  YES: Out-of-order completion
Simplifying the Data Structure Assuming In-order Issue

- Suppose the instruction is not dispatched by the Issue stage
  - If a RAW hazard exists
  - or if the required FU is busy
- Suppose operands are latched by the functional unit on issue

Can the dispatched instruction cause a

- WAR hazard? NO: Operands read at issue
- WAW hazard? YES: Out-of-order completion

<table>
<thead>
<tr>
<th>Name</th>
<th>Busy</th>
<th>Op</th>
<th>Dest</th>
<th>Src1</th>
<th>Src2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mem</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>......</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Simplifying the Data Structure Assuming In-order Issue

- Suppose the instruction is not dispatched by the Issue stage
  - If a RAW hazard exists
  - or if the required FU is busy

- Suppose operands are latched by the functional unit on issue

Can the dispatched instruction cause a

- WAR hazard? NO: Operands read at issue
- WAW hazard? YES: Out-of-order completion

<table>
<thead>
<tr>
<th>Name</th>
<th>Busy</th>
<th>Op</th>
<th>Dest</th>
<th>Src1</th>
<th>Src2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int</td>
<td></td>
<td></td>
<td>Write</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mem</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>......</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Simplifying the Data Structure
Assuming In-order Issue

- Suppose the instruction is not dispatched by the Issue stage
  - If a RAW hazard exists
  - or if the required FU is busy

- Suppose operands are latched by the functional unit on issue

Can the dispatched instruction cause a

- WAR hazard? NO: Operands read at issue
- WAW hazard? YES: Out-of-order completion

<table>
<thead>
<tr>
<th>Name</th>
<th>Busy</th>
<th>Op</th>
<th>Dest</th>
<th>Src1</th>
<th>Src2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int</td>
<td></td>
<td></td>
<td>Write</td>
<td>Pending</td>
<td></td>
</tr>
<tr>
<td>Mem</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>......</td>
<td></td>
<td></td>
<td>No read is pending after issue</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Simplifying the Data Structure

- No WAR hazard
  - \( \Rightarrow \) no need to keep \( src1 \) and \( src2 \)

<table>
<thead>
<tr>
<th>Name</th>
<th>Busy</th>
<th>Op</th>
<th>Dest</th>
<th>Src1</th>
<th>Src2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int</td>
<td></td>
<td></td>
<td>Write Pending</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mem</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add1</td>
<td></td>
<td></td>
<td>No read is pending after issue</td>
<td></td>
<td></td>
</tr>
<tr>
<td>......</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Simplifying the Data Structure

- No WAR hazard
  ⇒ no need to keep src1 and src2

- The Issue stage does not dispatch an instruction in case of a WAW hazard
  ⇒ a register name can occur at most once in the dest column
  Can be encoded as a bit vector

<table>
<thead>
<tr>
<th>Name</th>
<th>Busy</th>
<th>Op</th>
<th>Dest</th>
<th>Src1</th>
<th>Src2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mem</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>......</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Write Pending
No read is pending after issue
Scoreboard for In-order Issues

Busy[FU#] : a bit-vector to indicate FU’s availability.
(FU = Int, Add, Mult, Div)
These bits are hardwired to FU's.

WP[reg#] : a bit-vector to record the registers for which writes are pending.
These bits are set to true by the Issue stage and set to false by the WB stage

Issue checks the instruction (opcode dest src1 src2) against the scoreboard (Busy & WP) to dispatch

FU available?
RAW?
WAR?
WAW?
Scoreboard for In-order Issues

**Busy**[FU#] : a bit-vector to indicate FU’s availability.  
(FU = Int, Add, Mult, Div)  
These bits are hardwired to FU's.

**WP**[reg#] : a bit-vector to record the registers for which writes are pending.  
These bits are set to true by the Issue stage and set to false by the WB stage.

Issue checks the instruction (opcode dest src1 src2) against the scoreboard (Busy & WP) to dispatch.

FU available?  
RAW?  
WAR?  
WAW?
Scoreboard for In-order Issues

Busy[FU#] : a bit-vector to indicate FU’s availability. (FU = Int, Add, Mult, Div)
These bits are hardwired to FU's.

WP[reg#] : a bit-vector to record the registers for which writes are pending.
These bits are set to true by the Issue stage and set to false by the WB stage

Issue checks the instruction (opcode dest src1 src2) against the scoreboard (Busy & WP) to dispatch

FU available?  Busy[FU#]
RAW?           WP[src1] or WP[src2]
WAR?           WAW?
Scoreboard for In-order Issues

Busy[FU#] : a bit-vector to indicate FU's availability.
(FU = Int, Add, Mult, Div)
These bits are hardwired to FU's.

WP[reg#] : a bit-vector to record the registers for which writes are pending.
These bits are set to true by the Issue stage and set to false by the WB stage

Issue checks the instruction (opcode dest src1 src2) against the scoreboard (Busy & WP) to dispatch

FU available?  Busy[FU#]
RAW?          WP[src1] or WP[src2]
WAR?          cannot arise
WAW?
Scoreboard for In-order Issues

Busy[FU#] : a bit-vector to indicate FU's availability. (FU = Int, Add, Mult, Div)
These bits are hardwired to FU's.

WP[reg#] : a bit-vector to record the registers for which writes are pending.
These bits are set to true by the Issue stage and set to false by the WB stage

Issue checks the instruction (opcode dest src1 src2) against the scoreboard (Busy & WP) to dispatch

FU available?  Busy[FU#]
RAW?          WP[src1] or WP[src2]
WAR?          cannot arise
WAW?          WP[dest]
Scoreboard Dynamics

<table>
<thead>
<tr>
<th>Issue time</th>
<th>Functional Unit Status</th>
<th>Registers Reserved for Writes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Int(1) Add(1) Mult(3) Div(4) WB</td>
<td></td>
</tr>
<tr>
<td></td>
<td>f6, f6, f4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>f2, 45(r3)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>f0, f2, f4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>f8, f6, f2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>f10, f0, f6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>f6, f8, f2</td>
<td></td>
</tr>
</tbody>
</table>

September 25, 2023
## Scoreboard Dynamics

<table>
<thead>
<tr>
<th>Issue time</th>
<th>Functional Unit Status</th>
<th>Registers Reserved for Writes</th>
</tr>
</thead>
<tbody>
<tr>
<td>t0</td>
<td>Int(1)</td>
<td>Add(1)</td>
</tr>
<tr>
<td>I₁</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I₂</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I₃</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I₄</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I₅</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I₆</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Instructions:
- **FDIV.D f6, f6, f4**
- **FLD f2, 45(r3)**
- **FMUL.D f0, f2, f4**
- **FDIV.D f8, f6, f2**
- **FSUB.D f10, f0, f6**
- **FADD.D f6, f8, f2**
# Scoreboard Dynamics

## Functional Unit Status

<table>
<thead>
<tr>
<th></th>
<th>Int(1)</th>
<th>Add(1)</th>
<th>Mult(3)</th>
<th>Div(4)</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>t0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I1</td>
<td></td>
<td></td>
<td>f6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Registers Reserved for Writes

<table>
<thead>
<tr>
<th></th>
<th>f6</th>
</tr>
</thead>
<tbody>
<tr>
<td>t0</td>
<td></td>
</tr>
</tbody>
</table>

### Issue Time

- **I1**: FDIV.D, f6, f6, f4
- **I2**: FLD, f2, 45(r3)
- **I3**: FMUL.D, f0, f2, f4
- **I4**: FDIV.D, f8, f6, f2
- **I5**: FSUB.D, f10, f0, f6
- **I6**: FADD.D, f6, f8, f2

### WB Time

- **Check Busy[Fu#]**
- **Check WP[src1, src2]**
- **Check WP[dest]**

---

September 25, 2023
## Scoreboard Dynamics

<table>
<thead>
<tr>
<th>Issue time</th>
<th>Functional Unit Status</th>
<th>Registers Reserved for Writes</th>
</tr>
</thead>
<tbody>
<tr>
<td>t0</td>
<td>I₁</td>
<td>f6</td>
</tr>
<tr>
<td>t1</td>
<td>I₂</td>
<td>f2</td>
</tr>
</tbody>
</table>

### Functional Unit Status
- **Int(1)**: Add(1), Mult(3), Div(4)
- **WB**

### Issue time

<table>
<thead>
<tr>
<th>I₁</th>
<th>FDIV.D</th>
<th>f6, f6, f4</th>
</tr>
</thead>
<tbody>
<tr>
<td>I₂</td>
<td>FLD</td>
<td>f2, 45(r3)</td>
</tr>
<tr>
<td>I₃</td>
<td>FMUL.D</td>
<td>f0, f2, f4</td>
</tr>
<tr>
<td>I₄</td>
<td>FDIV.D</td>
<td>f8, f6, f2</td>
</tr>
<tr>
<td>I₅</td>
<td>FSUB.D</td>
<td>f10, f0, f6</td>
</tr>
<tr>
<td>I₆</td>
<td>FADD.D</td>
<td>f6, f8, f2</td>
</tr>
</tbody>
</table>

### Check Busy[Fu#]
- Check WP[src1, src2]
- Check WP[dest]
# Scoreboard Dynamics

<table>
<thead>
<tr>
<th>Issue time</th>
<th>Functional Unit Status</th>
<th>Registers Reserved for Writes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Int(1) Add(1) Mult(3) Div(4) WB</td>
<td></td>
</tr>
<tr>
<td>t0</td>
<td>1</td>
<td>f6</td>
</tr>
<tr>
<td>t1</td>
<td>2</td>
<td>f6</td>
</tr>
<tr>
<td>t2</td>
<td></td>
<td>f6</td>
</tr>
</tbody>
</table>

| $I_1$ | FDIV.D | f6, f6, f4 |
| $I_2$ | FLD    | f2, 45(r3) |
| $I_3$ | FMUL.D | f0, f2, f4 |
| $I_4$ | FDIV.D | f8, f6, f2 |
| $I_5$ | FSUB.D | f10, f0, f6 |
| $I_6$ | FADD.D | f6, f8, f2 |

Check Busy[Fu#]  
Check WP[src1, src2]  
Check WP[dest]
## Scoreboard Dynamics

### Functional Unit Status

<table>
<thead>
<tr>
<th>Issue time</th>
<th>Int(1)</th>
<th>Add(1)</th>
<th>Mult(3)</th>
<th>Div(4)</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>t0</td>
<td></td>
<td></td>
<td></td>
<td>f6</td>
<td></td>
</tr>
<tr>
<td>t1</td>
<td></td>
<td>f2</td>
<td></td>
<td>f6</td>
<td></td>
</tr>
<tr>
<td>t2</td>
<td></td>
<td></td>
<td></td>
<td>f6</td>
<td>f2</td>
</tr>
<tr>
<td>t3</td>
<td></td>
<td>f0</td>
<td></td>
<td>f6</td>
<td>f6</td>
</tr>
</tbody>
</table>

### Register Reserved for Writes

- f6
- f6, f2
- f6, f2
- f6, f0

### Instruction Details

- **I₁**: FDIV.D, f6, f6, f4
- **I₂**: FLD, f2, 45(r3)
- **I₃**: FMUL.D, f0, f2, f4
- **I₄**: FDIV.D, f8, f6, f2
- **I₅**: FSUB.D, f10, f0, f6
- **I₆**: FADD.D, f6, f8, f2

**Check Busy[Fu#]**
- Check WP[src1, src2]
- Check WP[dest]
# Scoreboard Dynamics

## Functional Unit Status

<table>
<thead>
<tr>
<th>Time</th>
<th>Int(1)</th>
<th>Add(1)</th>
<th>Mult(3)</th>
<th>Div(4)</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>t0</td>
<td></td>
<td></td>
<td></td>
<td>f6</td>
<td></td>
</tr>
<tr>
<td>t1</td>
<td>I₂</td>
<td>f2</td>
<td></td>
<td>f6</td>
<td></td>
</tr>
<tr>
<td>t2</td>
<td></td>
<td></td>
<td></td>
<td>f6</td>
<td>f2</td>
</tr>
<tr>
<td>t3</td>
<td>I₃</td>
<td>f0</td>
<td></td>
<td>f6</td>
<td></td>
</tr>
<tr>
<td>t4</td>
<td></td>
<td>f0</td>
<td></td>
<td>f6</td>
<td></td>
</tr>
</tbody>
</table>

## Registers Reserved for Writes

- t0: f6
- t1: f6, f2
- t2: f6, f2
- t3: f6, f0
- t4: f6, f0

---

### Issue time

- $I₁$: FDIV.D  
- $I₂$: FLD  
- $I₃$: FMUL.D  
- $I₄$: FDIV.D  
- $I₅$: FSUB.D  
- $I₆$: FADD.D

### WB time

- Check Busy[Fu#]  
- Check WP[src1, src2]  
- Check WP[dest]
# Scoreboard Dynamics

## Functional Unit Status

<table>
<thead>
<tr>
<th>t0</th>
<th>Int(1)</th>
<th>Add(1)</th>
<th>Mult(3)</th>
<th>Div(4)</th>
<th>WB</th>
<th>Registers Reserved for Writes</th>
</tr>
</thead>
<tbody>
<tr>
<td>t0</td>
<td>$I_1$</td>
<td></td>
<td></td>
<td>f6</td>
<td></td>
<td>f6</td>
</tr>
<tr>
<td>t1</td>
<td>$I_2$</td>
<td>f2</td>
<td></td>
<td>f6</td>
<td></td>
<td>f6, f2</td>
</tr>
<tr>
<td>t2</td>
<td></td>
<td></td>
<td></td>
<td>f6</td>
<td>f2</td>
<td>f6, f2</td>
</tr>
<tr>
<td>t3</td>
<td>$I_3$</td>
<td>f0</td>
<td></td>
<td>f6</td>
<td></td>
<td>f6, f0</td>
</tr>
<tr>
<td>t4</td>
<td></td>
<td>f0</td>
<td></td>
<td>f6</td>
<td></td>
<td>f6, f0, $I_1$</td>
</tr>
<tr>
<td>t5</td>
<td>$I_4$</td>
<td></td>
<td>f0, f8</td>
<td></td>
<td></td>
<td>f0, f8</td>
</tr>
</tbody>
</table>

## Instruction Details

- $I_1$: FDIV.D
- $I_2$: FLD
- $I_3$: FMUL.D
- $I_4$: FDIV.D
- $I_5$: FSUB.D
- $I_6$: FADD.D

## Check Busy
- [Fu#]

## Check Write Permissions
- [src1, src2]
- [dest]
## Scoreboard Dynamics

### Functional Unit Status

<table>
<thead>
<tr>
<th>Issue time</th>
<th>Int(1)</th>
<th>Add(1)</th>
<th>Mult(3)</th>
<th>Div(4)</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>t0</td>
<td></td>
<td></td>
<td>f6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t1</td>
<td></td>
<td>I₂</td>
<td>f2</td>
<td>f6</td>
<td></td>
</tr>
<tr>
<td>t2</td>
<td></td>
<td></td>
<td>f6</td>
<td>f2</td>
<td></td>
</tr>
<tr>
<td>t3</td>
<td>I₃</td>
<td></td>
<td>f0</td>
<td>f6</td>
<td></td>
</tr>
<tr>
<td>t4</td>
<td></td>
<td></td>
<td>f0</td>
<td>f6</td>
<td></td>
</tr>
<tr>
<td>t5</td>
<td>I₄</td>
<td></td>
<td>f0, f8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t6</td>
<td></td>
<td></td>
<td>f8</td>
<td>f0</td>
<td></td>
</tr>
</tbody>
</table>

### Registers Reserved for Writes

<table>
<thead>
<tr>
<th>WB time</th>
<th>t0</th>
<th>t1</th>
<th>t2</th>
<th>t3</th>
<th>t4</th>
<th>t5</th>
<th>t6</th>
</tr>
</thead>
<tbody>
<tr>
<td>f6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f6, f2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f6, f2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f6, f0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f6, f0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f0, f8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f0, f8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Instructions

- **I₁**: FDIV.D
- **I₂**: FLD
- **I₃**: FMUL.D
- **I₄**: FDIV.D
- **I₅**: FSUB.D
- **I₆**: FADD.D

### Check Busy[Fu#]
- Check Busy[Fu#]
- Check WP[src1, src2]
- Check WP[dest]

---

September 25, 2023
# Scoreboard Dynamics

## Functional Unit Status

<table>
<thead>
<tr>
<th></th>
<th>Int(1)</th>
<th>Add(1)</th>
<th>Mult(3)</th>
<th>Div(4)</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>t0</td>
<td></td>
<td></td>
<td></td>
<td>f6</td>
<td></td>
</tr>
<tr>
<td>t1</td>
<td>I_2</td>
<td>f2</td>
<td></td>
<td>f6</td>
<td></td>
</tr>
<tr>
<td>t2</td>
<td></td>
<td></td>
<td>f6</td>
<td></td>
<td>I_2</td>
</tr>
<tr>
<td>t3</td>
<td>I_3</td>
<td>f0</td>
<td></td>
<td>f6</td>
<td>I_1</td>
</tr>
<tr>
<td>t4</td>
<td></td>
<td>f0</td>
<td></td>
<td>f6</td>
<td>f6, f0</td>
</tr>
<tr>
<td>t5</td>
<td>I_4</td>
<td>f0, f8</td>
<td></td>
<td>f0, f8</td>
<td>I_3</td>
</tr>
<tr>
<td>t6</td>
<td></td>
<td>f8</td>
<td></td>
<td>f0, f8</td>
<td></td>
</tr>
<tr>
<td>t7</td>
<td>I_5</td>
<td>f10</td>
<td></td>
<td>f8</td>
<td>f8, f10</td>
</tr>
</tbody>
</table>

## Registers Reserved for Writes

<table>
<thead>
<tr>
<th></th>
<th>f6</th>
</tr>
</thead>
<tbody>
<tr>
<td>t0</td>
<td></td>
</tr>
<tr>
<td>t1</td>
<td>f6, f2</td>
</tr>
<tr>
<td>t2</td>
<td>f6, f2</td>
</tr>
<tr>
<td>t3</td>
<td>f6, f0</td>
</tr>
<tr>
<td>t4</td>
<td>f6, f0</td>
</tr>
<tr>
<td>t5</td>
<td>f0, f8</td>
</tr>
<tr>
<td>t6</td>
<td>f0, f8</td>
</tr>
<tr>
<td>t7</td>
<td>f8, f10</td>
</tr>
</tbody>
</table>

## Instructions

- **FDIV.D**
  - Issues: \( I_1 \)
  - **Issue time**: t0
  - **Check Busy[Fu#]**
  - **Check WP[src1, src2]**
  - **Check WP[dest]**

- **FLD**
  - Issues: \( I_2 \)
  - **Issue time**: t1
  - **Issue time**: t2
  - **Check Busy[Fu#]**
  - **Check WP[src1, src2]**

- **FMUL.D**
  - Issues: \( I_3 \)
  - **Issue time**: t3
  - **Check Busy[Fu#]**
  - **Check WP[src1, src2]**

- **FDIV.D**
  - Issues: \( I_4 \)
  - **Issue time**: t4
  - **Issue time**: t5
  - **Check Busy[Fu#]**
  - **Check WP[src1, src2]**

- **FSUB.D**
  - Issues: \( I_5 \)
  - **Issue time**: t6
  - **Check Busy[Fu#]**
  - **Check WP[src1, src2]**

- **FADD.D**
  - Issues: \( I_6 \)
  - **Issue time**: t7
  - **Check Busy[Fu#]**
  - **Check WP[src1, src2]**
Scoreboard Dynamics

<table>
<thead>
<tr>
<th>Issue time</th>
<th>Functional Unit Status</th>
<th>Registers Reserved for Writes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Int(1) Add(1) Mult(3)</td>
<td></td>
</tr>
<tr>
<td>t0</td>
<td>I₁</td>
<td>f6</td>
</tr>
<tr>
<td>t1</td>
<td>I₂</td>
<td>f6, f2</td>
</tr>
<tr>
<td>t2</td>
<td>f6, f2</td>
<td></td>
</tr>
<tr>
<td>t3</td>
<td>I₃</td>
<td>f6</td>
</tr>
<tr>
<td>t4</td>
<td>I₁</td>
<td>f6, f0</td>
</tr>
<tr>
<td>t5</td>
<td>f0, f8</td>
<td></td>
</tr>
<tr>
<td>t6</td>
<td>f0, f8</td>
<td></td>
</tr>
<tr>
<td>t7</td>
<td>I₅</td>
<td>f8, f10</td>
</tr>
<tr>
<td>t8</td>
<td>f8, f10</td>
<td></td>
</tr>
</tbody>
</table>

- **I₁**: FDIV.D
- **I₂**: FLD
- **I₃**: FMUL.D
- **I₄**: FDIV.D
- **I₅**: FSUB.D
- **I₆**: FADD.D

Check Busy[Fu#]
Check WP[src1, src2]
Check WP[dest]
# Scoreboard Dynamics

<table>
<thead>
<tr>
<th>Issue time</th>
<th>Functional Unit Status</th>
<th>Registers Reserved for Writes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Int(1) Add(1) Mult(3)</td>
<td></td>
</tr>
<tr>
<td>t0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>t1</td>
<td>I₂</td>
<td>f6, f2</td>
</tr>
<tr>
<td>t2</td>
<td>I₁</td>
<td>f6</td>
</tr>
<tr>
<td>t3</td>
<td>I₃</td>
<td>f6, f0</td>
</tr>
<tr>
<td>t4</td>
<td>I₁</td>
<td>f6, f0</td>
</tr>
<tr>
<td>t5</td>
<td>I₄</td>
<td>f0, f8</td>
</tr>
<tr>
<td>t6</td>
<td>I₃</td>
<td>f0, f8</td>
</tr>
<tr>
<td>t7</td>
<td>I₅</td>
<td>f8, f10</td>
</tr>
<tr>
<td>t8</td>
<td>I₅</td>
<td>f8, f10</td>
</tr>
<tr>
<td>t9</td>
<td>I₄</td>
<td>f8, f10</td>
</tr>
</tbody>
</table>

**Functional Unit Status**

- **Int(1)**: Integer Unit
- **Add(1)**: Add Unit
- **Mult(3)**: Multiply Unit
- **Div(4)**: Divide Unit
- **WB**: Write Back Unit

**Instructions**

- **FDIV.D**: f6, f6, f4
- **FLD**: f2, 45(r3)
- **FMUL.D**: f0, f2, f4
- **FDIV.D**: f8, f6, f2
- **FSUB.D**: f10, f0, f6
- **FADD.D**: f6, f8, f2

**Check Busy[Fu#]**
- Check Busy[FlD]
- Check Busy[FDIV.D]
- Check Busy[FMUL.D]

**Check WP[src1, src2]**
- Check WP[dest]

---

September 25, 2023

MIT 6.5900 Fall 2023

L06-31
Scoreboard Dynamics

<table>
<thead>
<tr>
<th>Issue time</th>
<th>Functional Unit Status</th>
<th>Registers Reserved for Writes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Int(1) Add(1) Mult(3)</td>
<td></td>
</tr>
<tr>
<td>t0</td>
<td>I₁</td>
<td>f6</td>
</tr>
<tr>
<td>t1</td>
<td>f2</td>
<td>f6, f2</td>
</tr>
<tr>
<td>t2</td>
<td>f6</td>
<td>f6, f2</td>
</tr>
<tr>
<td>t3</td>
<td>f0</td>
<td>f6, f0</td>
</tr>
<tr>
<td>t4</td>
<td>f0</td>
<td>f6, f0</td>
</tr>
<tr>
<td>t5</td>
<td>f0 f8</td>
<td>f0, f8</td>
</tr>
<tr>
<td>t6</td>
<td>f8</td>
<td>f0, f8</td>
</tr>
<tr>
<td>t7</td>
<td>f10</td>
<td>f8, f10</td>
</tr>
<tr>
<td>t8</td>
<td>f8</td>
<td>f8, f10</td>
</tr>
<tr>
<td>t9</td>
<td>f8</td>
<td>f8, f10</td>
</tr>
<tr>
<td>t10</td>
<td>f6</td>
<td>f6</td>
</tr>
</tbody>
</table>

I₁: FDIV.D  f6, f6, f4
I₂: FLD      f2, 45(r3)
I₃: FMUL.D   f0, f2, f4
I₄: FDIV.D   f8, f6, f2
I₅: FSUB.D   f10, f0, f6
I₆: FADD.D   f6, f8, f2

Check Busy[Fu#]
Check WP[src1, src2]
Check WP[dest]
## Scoreboard Dynamics

<table>
<thead>
<tr>
<th>Issue time</th>
<th>Functional Unit Status</th>
<th>Registers Reserved for Writes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Int(1) Add(1) Mult(3) Div(4)</td>
<td></td>
</tr>
<tr>
<td>t0</td>
<td>I₁</td>
<td>f6</td>
</tr>
<tr>
<td>t1</td>
<td>I₂</td>
<td>f2</td>
</tr>
<tr>
<td>t2</td>
<td></td>
<td>f6</td>
</tr>
<tr>
<td>t3</td>
<td>I₃</td>
<td>f0</td>
</tr>
<tr>
<td>t4</td>
<td></td>
<td>f0</td>
</tr>
<tr>
<td>t5</td>
<td>I₄</td>
<td>f0, f8</td>
</tr>
<tr>
<td>t6</td>
<td></td>
<td>f8</td>
</tr>
<tr>
<td>t7</td>
<td>I₅</td>
<td>f10</td>
</tr>
<tr>
<td>t8</td>
<td></td>
<td>f8</td>
</tr>
<tr>
<td>t9</td>
<td></td>
<td>f8</td>
</tr>
<tr>
<td>t10</td>
<td>I₆</td>
<td>f6</td>
</tr>
<tr>
<td>t11</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Functional Operations

- **I₁**: FDIV.D
  - f6, f6, f4
- **I₂**: FLD
  - f2, 45(r3)
- **I₃**: FMUL.D
  - f0, f2, f4
- **I₄**: FDIV.D
  - f8, f6, f2
- **I₅**: FSUB.D
  - f10, f0, f6
- **I₆**: FADD.D
  - f6, f8, f2

### Check Busy
- [Fu#]

### Check WP
- [src1, src2]
- [dest]
Preview: Anatomy of a Modern Out-of-Order Superscalar Core
Preview: Anatomy of a Modern Out-of-Order Superscalar Core

- L06 (Today): Complex pipes w/in-order issue
Preview: Anatomy of a Modern Out-of-Order Superscalar Core

- L06 (Today): Complex pipes w/ in-order issue
- L07: Out-of-order exec & renaming
Preview: Anatomy of a Modern Out-of-Order Superscalar Core

- **L06 (Today):** Complex pipes w/in-order issue
- **L07:** Out-of-order exec & renaming
- **L08:** Branch prediction
Preview: Anatomy of a Modern Out-of-Order Superscalar Core

- L06 (Today): Complex pipes w/ in-order issue
- L07: Out-of-order exec & renaming
- L08: Branch prediction
- L09: Speculative execution and recovery
Preview: Anatomy of a Modern Out-of-Order Superscalar Core

- **L06 (Today):** Complex pipes w/in-order issue
- **L07:** Out-of-order exec & renaming
- **L08:** Branch prediction
- **L09:** Speculative execution and recovery
- **L10:** Advanced Memory Ops