Overview

- Back to processing cores

- How to run programs faster?

- What are the dependencies that we need to follow?

- In-order execution for complex pipeline

- Out-of-order execution
Compiler Scheduling

- Compiler scheduling can eliminate pipeline stalls
  - but register pressure increases

- Example: \( a = b + c \); \( d = e - f \)

1. \text{lw} r1, 4(r2); b
2. \text{lw} r3, 8(r2); c
3. add r5, r1, r3
4. \text{sw} r5, 0(r2); a
5. \text{lw} r1, 16(r2); e
6. \text{lw} r3, 20(r2); f
7. sub r5, r1, r3
8. \text{sw} r5, 12(r2); d
Classifying Data Hazards

Consider executing a sequence of
\[ r_k \leftarrow (r_i) \text{ op } (r_j) \]

Data-dependence

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

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

Anti-dependence (name dependence)

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

\[ r_1 \leftarrow (r_4) \text{ op } (r_5) \]

Classifying Data Hazards

Output-dependence

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

\[ r_3 \leftarrow (r_6) \text{ op } (r_7) \]

- **WAW example:** Load takes 2 memory cycles, ALU skips MEM stages

```
  lw $1, A($2) IF ID EX MEM1 MEM2 WB
  add $1, $2, 4 IF ID EX WB
```

- **Read-after-Read (RAR) hazard?**
Data Hazards: An Example

\[ I_1 \text{ DIVD} \quad f_6, \quad f_6, \quad f_4 \]
\[ I_2 \text{ LD} \quad f_2, \quad 45(r3) \]
\[ I_3 \text{ MULTD} \quad f_0, \quad f_2, \quad f_4 \]
\[ I_4 \text{ DIVD} \quad f_8, \quad f_6, \quad f_2 \]
\[ I_5 \text{ SUBD} \quad f_{10}, \quad f_0, \quad f_6 \]
\[ I_6 \text{ ADDD} \quad f_6, \quad f_8, \quad f_2 \]

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: \( I_1, I_2, I_3, I_4, I_5, I_6 \)

Instruction Scheduling

\[ I_1 \text{ DIVD} \quad f_6, \quad f_6, \quad f_4 \]
\[ I_2 \text{ LD} \quad f_2, \quad 45(r3) \]
\[ I_3 \text{ MULTD} \quad f_0, \quad f_2, \quad f_4 \]
\[ I_4 \text{ DIVD} \quad f_8, \quad f_6, \quad f_2 \]
\[ I_5 \text{ SUBD} \quad f_{10}, \quad f_0, \quad f_6 \]
\[ I_6 \text{ ADDD} \quad f_6, \quad f_8, \quad f_2 \]

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: \( I_1, I_2, I_3, I_4, I_5, I_6 \)
Dynamic Scheduling

- Allow hardware to make scheduling decisions

- Advantages:
  - simpler compiler
  - handles cases not resolvable at compile time
  - can run same code on different (ISA-compatible) pipeline

- Disadvantages:
  - hardware complexity significantly higher

Complex Pipeline Structure

IF → ID → ALU → Mem → WB
- GPR’s
- FPR’s
- Fadd
- Fmul
- Fdiv
When is it Safe to Issue an Instruction?

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?
- Control Dependency

Control 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] v WP[src2]
- WAR? Not possible
- WAW? WP[dest]
In-Order Issue Limitations: an example

1  LD  F2,  34(R2)  latency 1
2  LD  F4,  45(R3)  miss
3  MULTD  F6,  F4,  F2  3
4  SUBD  F8,  F2,  F2  1
5  DIVD  F4,  F2,  F8  4
6  ADDD  F10,  F6,  F4  1

In-order:  1 (2,1) . . . . . . . .  2 3 4 4 3 5 . . .  5 6 6

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 (no microcoding)
- 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, novel freon-based technology for cooling
- Fastest machine in world for 5 years (until 7600)
  - over 100 sold ($7-10M each)
IBM Memo on CDC6600

Thomas Watson Jr., IBM CEO, August 1963:

“Last week, Control Data ... announced the 6600 system. I understand that in the laboratory developing the system there are only 34 people including the janitor. Of these, 14 are engineers and 4 are programmers... Contrasting this modest effort with our vast development activities, I fail to understand why we have lost our industry leadership position by letting someone else offer the world’s most powerful computer.”

To which Cray replied: “It seems like Mr. Watson has answered his own question.”

Dynamic Scheduling by Scoreboard

- Idea: execute code out of order but preserve dependences
- Split ID stage into two: IS+RO

<table>
<thead>
<tr>
<th>IF</th>
<th>IS</th>
<th>RO</th>
<th>EX</th>
<th>WB</th>
</tr>
</thead>
</table>

- IS (Issue) – decode, check for structural and WAW hazards
- RO (Read Operands) – wait until no RAW hazards
- WB – wait until no WAR hazards (why can these happen?)

- Scoreboard:
  1. constructs data dependences
  2. determines when operands can be read and instruction executed
  3. decides when instruction can commit result
Four Steps to Instruction Execution

- **Step 1: issue**
  - if FU available (structural), and
  - if no earlier instruction writes to same destination (WAW), then
  - send instruction to FU

- **Step 2: read operands (a.k.a. dispatch)**
  - if no operand pending update (RAW), then
  - instruct FU to read operands and start execution

- **Step 3: execution**
  - inform scoreboard at completion

- **Step 4: write result (a.k.a. retire)**
  - if WAR hazard possible, stall at WB stage

Scoreboard: Highlights

- Still in-order issue
  - instruction that stalls at issue backs up pipeline
  - however out-of-order dispatch
  - *Instruction queue* between IF and IS can absorb some impact

- FU reads operands when both are available at the register file
  - no forwarding logic
  - but not as crucial as before – number of stages not fixed (dependent on FU)

- CDC6600 had “only” four read/write buses
  - additional structural hazard – we’ll ignore this one...
Parts of Scoreboard

- Instruction Status – in which of the four steps each instruction is

- FU Status – is FU available?
  - Busy – FU is busy
  - Op – operation to be performed
  - F_i, F_j, F_k – Destination and source registers
  - Q_j, Q_k – FUs producing F_j, F_k
  - R_j, R_k – Operand-ready flags; reset after operands are read

- Register Status – which FU will write each register (or blank)

\[ \Rightarrow \text{W}\text{P} \]

Scoreboard: Example

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Instruction Status</th>
<th>FU Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>fld 1 f6,34($2)</td>
<td>I 2 3 4</td>
<td>[ \text{Int} \ y \ \text{fld, f6} \ \text{F2} \ y ]</td>
</tr>
<tr>
<td>fld 2 f2,45($3)</td>
<td>I 2 3 4</td>
<td>[ \text{Mul1} \ y ]</td>
</tr>
<tr>
<td>fmul f0,f2,f4</td>
<td>I 2 3 4</td>
<td>[ \text{Mul2} \ y ]</td>
</tr>
<tr>
<td>fsub f8,f6,f2</td>
<td>I 2 3 4</td>
<td>[ \text{Add} \ y ]</td>
</tr>
<tr>
<td>fdiv f10,f0,f6</td>
<td>I 2 3 4</td>
<td>[ \text{Div} \ y ]</td>
</tr>
<tr>
<td>fadd f6,f8,f2</td>
<td>I 2 3 4</td>
<td>[ ]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Clock</th>
<th>Register Result Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>[ \text{F0 F2 F4 F6 F8 F10 F12 \ldots F30} ]</td>
</tr>
</tbody>
</table>

\[ \text{FU} \rightarrow \text{Inf} \]
### Scoreboard: Example

#### Instruction Status

<table>
<thead>
<tr>
<th>Instruction</th>
<th>I</th>
<th>R</th>
<th>E</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>fld _1 _6,34($2)</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>fld _2 _2,45($3)</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fmul _0,_2,_4</td>
<td>6</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fsub _8,_6,_2</td>
<td>7</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fdiv _10,_0,_6</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fadd _6,_8,_2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### FU Status

<table>
<thead>
<tr>
<th>Busy</th>
<th>Op</th>
<th>F_i</th>
<th>F_j</th>
<th>F_k</th>
<th>O_i</th>
<th>O_k</th>
<th>R_j</th>
<th>R_k</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int</td>
<td>Y</td>
<td>fdiv</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mul_1</td>
<td>Y</td>
<td>fmul</td>
<td>f0</td>
<td>f2</td>
<td>f4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add</td>
<td>Y</td>
<td>fsub</td>
<td>f8</td>
<td>f6</td>
<td>f2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Div</td>
<td>Y</td>
<td>fdiv</td>
<td>f10</td>
<td>16</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Register Result Status

<table>
<thead>
<tr>
<th>Clock</th>
<th>F0</th>
<th>F2</th>
<th>F4</th>
<th>F6</th>
<th>F8</th>
<th>F10</th>
<th>F12</th>
<th>F14</th>
<th>F16</th>
<th>F18</th>
<th>F20</th>
<th>F22</th>
<th>F24</th>
<th>F26</th>
<th>F28</th>
<th>F30</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Register Result Status

<table>
<thead>
<tr>
<th>Clock</th>
<th>F0</th>
<th>F2</th>
<th>F4</th>
<th>F6</th>
<th>F8</th>
<th>F10</th>
<th>F12</th>
<th>F14</th>
<th>F16</th>
<th>F18</th>
<th>F20</th>
<th>F22</th>
<th>F24</th>
<th>F26</th>
<th>F28</th>
<th>F30</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

ECE4750/CS4420 — Computer Architecture, Fall 2008, Suh
## Scoreboard: Example

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Status</th>
<th>FU Status</th>
<th>Register Result Status</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Busy</td>
<td>Op</td>
</tr>
<tr>
<td><strong>fld</strong> 6,34($2)</td>
<td>1 2 3 4</td>
<td>Int</td>
<td></td>
</tr>
<tr>
<td><strong>fld</strong> 2,45($3)</td>
<td>5 6 7 8</td>
<td>Mul_1</td>
<td></td>
</tr>
<tr>
<td><strong>fmul</strong> 0,2,4</td>
<td>6 9 19 20</td>
<td>Mul_2</td>
<td></td>
</tr>
<tr>
<td><strong>fsub</strong> 8,6,2</td>
<td>7 9 11 12</td>
<td>Add</td>
<td>Y</td>
</tr>
<tr>
<td><strong>fdiv</strong> 10,0,6</td>
<td>8 21</td>
<td>Div</td>
<td>Y</td>
</tr>
</tbody>
</table>

### Instruction Status

<table>
<thead>
<tr>
<th>Instruction</th>
<th>I</th>
<th>R</th>
<th>E</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>fld</strong> 6,34($2)</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td><strong>fld</strong> 2,45($3)</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td><strong>fmul</strong> 0,2,4</td>
<td>6</td>
<td>9</td>
<td>19</td>
<td>20</td>
</tr>
<tr>
<td><strong>fsub</strong> 8,6,2</td>
<td>7</td>
<td>9</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td><strong>fdiv</strong> 10,0,6</td>
<td>8</td>
<td>21</td>
<td>61</td>
<td>62</td>
</tr>
<tr>
<td><strong>fadd</strong> 6,8,2</td>
<td>13</td>
<td>14</td>
<td>16</td>
<td>22</td>
</tr>
</tbody>
</table>

### FU Status

<table>
<thead>
<tr>
<th>FU Status</th>
<th>Register Result Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUSY</td>
<td>F0 F2 F4 F6 F8 F10 F12 ... F30</td>
</tr>
<tr>
<td>Add</td>
<td>F0 F2 F4 F6 F8 F10 F12 ... F30</td>
</tr>
<tr>
<td>Div</td>
<td>F0 F2 F4 F6 F8 F10 F12 ... F30</td>
</tr>
</tbody>
</table>

### Clock

<table>
<thead>
<tr>
<th>Clock</th>
<th>21</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Register Result Status</th>
<th>F0 F2 F4 F6 F8 F10 F12 ... F30</th>
</tr>
</thead>
<tbody>
<tr>
<td>FU</td>
<td>Add</td>
</tr>
<tr>
<td>Div</td>
<td></td>
</tr>
</tbody>
</table>

ECE4750/CS4420 — Computer Architecture, Fall 2008, Suh
Scoreboard Limitations

- Available ILP
  - dependence chains
  - hardware limitations
    - CDC6600 cannot re-order across branches

- Number of scoreboard entries
  - *instruction window* – set of instructions examined as candidates for potential execution at any given point in time

- Number and type of functional units
  - structural hazards

- Name dependences
  - no renaming mechanism – Tomasulo’s (next)