CIS 429/529

Complex Pipelining: VLIW

Prof. Michel A. Kinsky
Data Hazards: An Example

$I_1$ DIVD $f6$, $f6$, $f4$

$I_2$ LD $f2$, $45(r3)$

$I_3$ MULTD $f0$, $f2$, $f4$

$I_4$ DIVD $f8$, $f6$, $f2$

$I_5$ SUBD $f10$, $f0$, $f6$

$I_6$ ADDD $f6$, $f8$, $f2$

RAW Hazards

WAR Hazards

WAW Hazards
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*

$I_1$  DIVD  $f_6$,  $f_6$,  $f_4$

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

$I_3$  MULTD  $f_0$,  $f_2$,  $f_4$

$I_4$  DIVD  $f_8$,  $f_6$,  $f_2$

$I_5$  SUBD  $f_{10}$,  $f_0$,  $f_6$

$I_6$  ADDD  $f_6$,  $f_8$,  $f_2$
# Out-of-order Completion

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Operation</th>
<th>Values</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_1$</td>
<td>DIVD</td>
<td>f6, f6, f4</td>
<td>4</td>
</tr>
<tr>
<td>$I_2$</td>
<td>LD</td>
<td>f2, 45(r3)</td>
<td>1</td>
</tr>
<tr>
<td>$I_3$</td>
<td>MULTD</td>
<td>f0, f2, f4</td>
<td>3</td>
</tr>
<tr>
<td>$I_4$</td>
<td>DIVD</td>
<td>f8, f6, f2</td>
<td>4</td>
</tr>
<tr>
<td>$I_5$</td>
<td>SUBD</td>
<td>f10, f0, f6</td>
<td>1</td>
</tr>
<tr>
<td>$I_6$</td>
<td>ADDD</td>
<td>f6, f8, f2</td>
<td>1</td>
</tr>
</tbody>
</table>

**in-order comp**

1  2  1  2  3  4  3  5  4  6  5  6

**out-of-order comp**

1  2  2  3  1  4  3  5  5  4  6  6

_ _ means completion
Little’s Law

Parallelism = Throughput * Latency

or

\[ \bar{N} = \bar{T} \times \bar{L} \]

Throughput per Cycle

One Operation

Latency in Cycles
Pipelined ILP Machine

Max Throughput, Six Instructions per Cycle

One Pipeline Stage

Latency in Cycles

Two Integer Units, Single Cycle Latency

Two Load/Store Units, Three Cycle Latency

Two Floating-Point Units, Four Cycle Latency

• How much instruction-level parallelism (ILP) required to keep machine pipelines busy?
Superscalar Control Logic Scaling

• Each issued instructions must make interlock checks against $W^*L$ instructions, i.e., growth in interlocks $\propto W^*(W^*L)$

• For in-order machines, $L$ is related to pipeline latencies

• For out-of-order machines, $L$ also includes time spent in instruction buffers (instruction window or ROB)

• As $W$ increases, larger instruction window is needed to find enough parallelism to keep machine busy $\rightarrow$ greater $L$

  ‣ Out-of-order control logic grows faster than $W^2$ ($\sim W^3$)
Out-of-Order Control Complexity

[ SGI/MIPS Technologies Inc., 1995 ]
VLIW: Very Long Instruction

- Multiple operations packed into one instruction
- Each operation slot is for a fixed function
- Constant operation latencies are specified
- Architecture requires guarantee of:
  - Parallelism within an instruction → no x-operation RAW check
  - No data use before data ready → no data interlocks
Early VLIW Machines

• FPS AP120B (1976)
  ‣ scientific attached array processor
  ‣ first commercial wide instruction machine

• Multiflow Trace (1987)
  ‣ available in configurations with 7, 14, or 28 operations/instruction
  ‣ 28 operations packed into a 1024-bit instruction word

• Cydrome Cydra-5 (1987)
  ‣ 7 operations encoded in 256-bit instruction word
  ‣ rotating register file
VLIW Compiler

• The compiler:
  ▶ Schedules to maximize parallel execution
  ▶ Guarantees intra-instruction parallelism
  ▶ Schedules to avoid data hazards (no interlocks)
    • Typically separates operations with explicit NOPs
Intel EPIC IA-64

• EPIC is the style of architecture
  ‣ Explicitly Parallel Instruction Computing

• IA-64 is Intel’s chosen ISA
  ‣ IA-64 = Intel Architecture 64-bit
  ‣ An object-code compatible VLIW

• Itanium (aka Merced) is first implementation (cf. 8086)
  ‣ First customer shipment expected 1997 (actually 2001)
  ‣ McKinley, second implementation shipped in 2002
# IA-64 Instruction Format

- Template bits describe grouping of these instructions with others in adjacent bundles
- Each group contains instructions that can execute in parallel
Problems with “Classic” VLIW

• Knowing branch probabilities
  ‣ Profiling requires an significant extra step in build process

• Object code size
  ‣ instruction padding wastes instruction memory/cache
  ‣ loop unrolling/software pipelining replicates code

• Scheduling variable latency memory operations
  ‣ caches and/or memory bank conflicts impose statically unpredictable variability
Problems with “Classic” VLIW

- Scheduling for statically unpredictable branches
  - optimal schedule varies with branch path
- Object-code compatibility
  - have to recompile all code for every machine, even for two machines in same generation
Loop Unrolling

• Unroll inner loop to perform 4 iterations at once
  ‣ Is this code correct?

```c
for (i=0; i<N; i++)
```

```c
for (i=0; i<N; i+=4)
{
}
```
Superscalar and VLIW Machines

• Superscalar architecture implements instruction-level parallelism
  ‣ Single Instructions-Single Data (SISD) format

• VLIW machines show the advantages and limitations of instruction-level parallelism
  ‣ Multiple Instructions-Multiple Data (MIMD)

• We will further explore MIMD types of execution with multicore processors

• Single Instructions-Multiple Data (SIMD) execution with vector processor (next week)