Chapter 2
Instruction-Level Parallelism and Its Exploitation

Overview
- Instruction level parallelism
- Dynamic Scheduling Techniques
  - Scoreboarding
  - Tomasulo’s Algorithm
- Reducing Branch Cost with Dynamic Hardware Prediction
  - Basic Branch Prediction and Branch-Prediction Buffers
  - Branch Target Buffers
- Overview of Superscalar and VLIW processors

CPI Equation
Pipeline CPI = Ideal pipeline CPI + Structural stalls + RAW stalls + WAR stalls + WAW stalls + Control stalls

<table>
<thead>
<tr>
<th>Technique</th>
<th>Reduces</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loop unrolling</td>
<td>Control stalls</td>
</tr>
<tr>
<td>Basic pipeline scheduling</td>
<td>RAW stalls</td>
</tr>
<tr>
<td>Dynamic scheduling with scoreboard</td>
<td>RAW stalls</td>
</tr>
<tr>
<td>Dynamic scheduling with register renaming</td>
<td>WAR and WAW stalls</td>
</tr>
<tr>
<td>Dynamic branch prediction</td>
<td>Control stalls</td>
</tr>
<tr>
<td>Issuing multiple instructions per cycle</td>
<td>Ideal CPI</td>
</tr>
<tr>
<td>Compiler dependence analysis</td>
<td>Ideal CPI and data stalls</td>
</tr>
<tr>
<td>Software pipelining and trace scheduling</td>
<td>Ideal CPI and data stalls</td>
</tr>
<tr>
<td>Speculation</td>
<td>All data and control stalls</td>
</tr>
<tr>
<td>Dynamic memory disambiguation</td>
<td>RAW stalls involving memory</td>
</tr>
</tbody>
</table>

Instruction Level Parallelism
- Potential overlap among instructions
- Few possibilities in a basic block
  - Blocks are small (6-7 instructions)
  - Instructions are dependent
- Exploit ILP across multiple basic blocks
  - Iterations of a loop
    for \( i = 1000; i > 0; i = i - 1 \)
    \( x[i] = x[i] + g \);
  - Alternative to vector instructions

Basic Pipeline Scheduling
- Find sequences of unrelated instructions
- Compiler’s ability to schedule
  - Amount of ILP available in the program
  - Latencies of the functional units
- Latency assumptions for the examples
  - Standard MIPS integer pipeline
  - No structural hazards (fully pipelined or duplicated units)
- Latencies of FP operations:

<table>
<thead>
<tr>
<th>Instruction producing result</th>
<th>Instruction using result</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP ALU op</td>
<td>FP ALU op</td>
<td>1</td>
</tr>
<tr>
<td>FP ALU op</td>
<td>SD</td>
<td>2</td>
</tr>
<tr>
<td>LD</td>
<td>FP ALU op</td>
<td>1</td>
</tr>
<tr>
<td>LD</td>
<td>SD</td>
<td>0</td>
</tr>
</tbody>
</table>

Sample Pipeline

```
IF  ID  FP1  FP2  FP3  FP4  DM  WB
```

```
IF  ID  FP1  FP2  FP3  FP4  DM  WB
```

```
IF  ID  DM  WB
```

```
IF  ID  DM  WB
```

```
IF  ID  DM  WB
```

```
IF  ID  DM  WB
```
**Basic Scheduling**

for (i = 1000; i > 0; i=i-1)

\[ x[i] = x[i] + s; \]

Sequential MIPS Assembly Code

Loop: LD F0, 0(R1)
ADD F4, F0, F2
SD 0(R1), F4
BNEZ R1, Loop

Pipelined execution:

<table>
<thead>
<tr>
<th>Loop</th>
<th>LD F0, 0(R1)</th>
<th>Stall</th>
<th>ADD F4, F0, F2</th>
<th>Stall</th>
<th>SD 0(R1), F4</th>
<th>Stall</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>SUBI R1, R1, #8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>ADD F4, F0, F2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Stall</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>BNEZ R1, Loop</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Scheduled pipelined execution:

<table>
<thead>
<tr>
<th>Loop</th>
<th>LD F0, 0(R1)</th>
<th>Stall</th>
<th>ADD F4, F0, F2</th>
<th>Stall</th>
<th>BNEZ R1, Loop</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Loop Unrolling**

Unrolled loop (four copies):

Loop: LD F0, 0(R1)
ADD F4, F0, F2
SD 0(R1), F4
SUBI R1, R1, #8
BNEZ R1, Loop

Scheduled Unrolled loop:

Loop: LD F0, 0(R1)
ADD F4, F0, F2
SD 0(R1), F4
SUBI R1, R1, #8
BNEZ R1, Loop

**Dynamic Scheduling**

- Scheduling separates dependent instructions
  - Static – performed by the compiler
  - Dynamic – performed by the hardware
- Advantages of dynamic scheduling
  - Handles dependences unknown at compile time
  - Simplifies the compiler
  - Optimization is done at run time
- Disadvantages
  - Can not eliminate true data dependences

**Out-of-order execution (1/2)**

- Central idea of dynamic scheduling
  - In-order execution:
    - DIV F0, F2, F4
    - ADD F2, F4, F4
  - Out-of-order execution:
    - DIV F0, F2, F4
    - ADD F2, F4, F4

**Out-of-Order Execution (2/2)**

- Separate issue process in ID:
  - Issue
    - decode instruction
    - check structural hazards
    - in-order execution
  - Read operands
    - Wait until no data hazards
    - Read operands
- Out-of-order execution/completion
  - Exception handling problems
  - WAR hazards

**Dynamic Scheduling with a Scoreboard**

- Details in Appendix A.7
- Allows out-of-order execution
  - Sufficient resources
  - No data dependencies
- Responsible for issue, execution and hazards
- Functional units with long delays
  - Duplicated
  - Fully pipelined
- CDC 6600 – 16 functional units
MIPS with Scoreboard

Scoreboard Operation

- Scoreboard centralizes hazard management
  - Every instruction goes through the scoreboard
  - Scoreboard determines when the instruction can read its operands and begin execution
  - Monitors changes in hardware and decides when an stalled instruction can execute
  - Controls when instructions can write results

- New pipeline

<table>
<thead>
<tr>
<th>ID</th>
<th>Read Regs</th>
<th>Execution</th>
<th>Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>Issue</td>
<td>Read</td>
<td>Execution</td>
<td>Write</td>
</tr>
</tbody>
</table>

Scoreboard Data Structure

- Instruction status – indicates pipeline stage
- Functional unit status
  - Busy – functional unit is busy or not
  - Op – operation to perform in the unit (+, -, etc.)
  - Fi – destination register
  - Fj, Fk – source register numbers
  - Qj, Qk – functional unit producing Fj, Fk
  - Rj, Rk – flags indicating when Fj, Fk are ready
- Register result status – FU that will write registers

Execution Process

- Issue
  - Functional unit is free (structural)
  - Active instructions do not have same Rd (WAW)
- Read Operands
  - Checks availability of source operands
  - Resolves RAW hazards dynamically (out-of-order execution)
- Execution
  - Functional unit begins execution when operands arrive
  - Notifies the scoreboard when it has completed execution
- Write result
  - Scoreboard checks WAR hazards
  - Stalls the completing instruction if necessary

Scoreboard Data Structure (1/3)

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Issue</th>
<th>Read Operands</th>
<th>Execution completed</th>
<th>Write</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD F6, 3(R2)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>LD F2, 45(R3)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>MUL F0, F2, F4</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIV F10, F6, F12</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD F12, F0, F10</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Scoreboard Data Structure (2/3)

<table>
<thead>
<tr>
<th>Name</th>
<th>Busy</th>
<th>Op</th>
<th>Fi</th>
<th>Fj</th>
<th>Fk</th>
<th>Qj</th>
<th>Qk</th>
<th>Rj</th>
<th>Rk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>Y</td>
<td>Load</td>
<td>F2</td>
<td>F3</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mult1</td>
<td>Y</td>
<td>Mut</td>
<td>F0</td>
<td>F2</td>
<td>F4</td>
<td>Integer</td>
<td>N</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>Mult2</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add</td>
<td>Y</td>
<td>Sub</td>
<td>F8</td>
<td>F6</td>
<td>F2</td>
<td>Integer</td>
<td>N</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>Divide</td>
<td>Y</td>
<td>Div</td>
<td>F10</td>
<td>F0</td>
<td>F6</td>
<td>Mut1</td>
<td>N</td>
<td>Y</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instruction status</th>
<th>Instruction status</th>
<th>Instruction status</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instruction status</th>
<th>Instruction status</th>
<th>Instruction status</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Scoreboard Data Structure (3/3)

<table>
<thead>
<tr>
<th>Instruction status</th>
<th>Instruction</th>
<th>Issue</th>
<th>Read operands</th>
<th>Execution complete</th>
<th>Partial result</th>
</tr>
</thead>
<tbody>
<tr>
<td>SD</td>
<td>F1, F2 (X)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>SD</td>
<td>F1, F2 (X)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>MOVF</td>
<td>F1, F2, F3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>MOVF</td>
<td>F1, F2, F3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>ADD</td>
<td>F1, F2, F3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>ADD</td>
<td>F1, F2, F3</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

Scoreboard Algorithm

<table>
<thead>
<tr>
<th>Instruction status</th>
<th>Wait until</th>
<th>Bookkeeping</th>
</tr>
</thead>
<tbody>
<tr>
<td>Issue</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Read operands</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Execution complete</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write result</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Scoreboard Limitations

- Amount of available ILP
- Number of scoreboard entries
  - Limited to a basic block
  - Extended beyond a branch
- Number and types of functional units
  - Structural hazards can increase with DS
- Presence of anti- and output- dependences
  - Lead to WAR and WAW stalls

Tomasulo Approach

- Another approach to eliminate stalls
  - Combines scoreboard with
  - Register renaming (to avoid WAR and WAW)
- Designed for the IBM 360/91
  - High FP performance for the whole 360 family
  - Four double precision FP registers
  - Long memory access and long FP delays
- Can support overlapped execution of multiple iterations of a loop

Tomasulo Approach

Stages

- Issue
  - Empty reservation station or buffer
  - Send operands to the reservation station
  - Use name of reservation station for operands
- Execute
  - Execute operation if operands are available
  - Monitor CDB for availability of operands
- Write result
  - When result is available, write it to the CDB
Example (1/2)

<table>
<thead>
<tr>
<th>Instruction status</th>
<th>Issue</th>
<th>Execute</th>
<th>Write result</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD F0, F1, F2</td>
<td>V</td>
<td>V</td>
<td>V</td>
</tr>
<tr>
<td>ADD F0, F1, F2</td>
<td>V</td>
<td>V</td>
<td>V</td>
</tr>
<tr>
<td>ADD F0, F1, F2</td>
<td>V</td>
<td>V</td>
<td>V</td>
</tr>
<tr>
<td>ADD F0, F1, F2</td>
<td>V</td>
<td>V</td>
<td>V</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reservation status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>----</td>
</tr>
<tr>
<td>ADD</td>
</tr>
<tr>
<td>ADD</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Register status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Field</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>Q1</td>
</tr>
</tbody>
</table>

Tomasulo’s Algorithm

<table>
<thead>
<tr>
<th>Instruction status</th>
<th>Wait until</th>
<th>Action or lockup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Issue</td>
<td>Status or buffer empty</td>
<td></td>
</tr>
<tr>
<td>Execute</td>
<td>(SR[Q=0] and DI[Q=0])</td>
<td></td>
</tr>
<tr>
<td>Write result</td>
<td>Execution completed at r and CDB available</td>
<td></td>
</tr>
</tbody>
</table>

Dynamic Hardware Prediction

- Importance of control dependences
  - Branches and jumps are frequent
  - Limiting factor as ILP increases (Amdahl’s law)
- Schemes to attack control dependences
  - Static
    - Basic (stall the pipeline)
    - Predict-not-taken and predict-taken
    - Delayed branch and canceling branch
  - Dynamic predictors
- Effectiveness of dynamic prediction schemes
  - Accuracy
  - Cost

Basic Branch Prediction Buffers

a.k.a. Branch History Table (BHT) - Small direct-mapped cache of T/NT bits

Loop Iterations

Loop: LD F0, F1(R1)
MULT F4,F0,F2
SD F4,F3,F5
SUBI R1, R1, #8
BNEZ R1, Loop

Basic Branch Prediction Buffers

IR: Branch Instruction
PC: Branch Target
BHT: T (predict taken)
NT (predict not-taken)
PC + 4
N-bit Branch Prediction Buffers

Use an n-bit saturating counter
Only the loop exit causes a misprediction
2-bit predictor almost as good as any general n-bit predictor

Branch-Target Buffers

• Further reduce control stalls (hopefully to 0)
• Store the predicted address in the buffer
• Access the buffer during IF

Performance Issues

• Limitations of branch prediction schemes
  – Prediction accuracy (80% - 95%)
  • Type of program
  • Size of buffer
  – Penalty of misprediction
• Fetch from both directions to reduce penalty
  – Memory system should:
    • Dual-ported
    • Have an interleaved cache
    • Fetch from one path and then from the other

Prediction with BTF

Five Primary Approaches in use for Multiple-issue Processors