EXPLOITING ILP

B649
Parallel Architectures and Pipelining
What is ILP?

Pipeline CPI = Ideal pipeline CPI + Structural stalls + Data hazard stalls + Control stalls

• Use hardware techniques to minimize stalls
  ★ branch prediction
  ★ dynamic scheduling
  ★ speculation

• Pick instructions across branches (i.e., across basic blocks) to overlap
  ★ on MIPS average dynamic branch frequency is 15% to 25%
  ★ Pick instructions across loop iterations

```c
for (i=1; i<=1000; i++)
    x[i] = x[i] + y[i];
```
Data Dependences

• Conditions for data dependence from instruction $i$ to instruction $j$
  ★ $i$ and $j$ access a common memory location / register
  ★ at least one of those accesses is a write
  ★ there is a valid control-flow path from $i$ to $j$

• Data dependence is transitive
  ★ $j$ depends on $i$, $k$ depends on $j$ $\Rightarrow k$ depends on $i$

• Three types of data dependences
  ★ true dependence (RAW)
  ★ anti-dependence (WAR)
  ★ output dependence (WAW)
Control Dependences

• Dependences arising out of program control-flow
• Prevent reordering to maintain program correctness (just like data dependences)

```java
if (c1) {
    S1;
}
if (c2) {
    S2;
}
S3;
```
Control Dependences

• Dependences arising out of program control-flow
• Prevent reordering to maintain program correctness (just like data dependences)

```c
if (c1) {
    S1;
}
if (c2) {
    S2;
}
S3;
```
Another (Equivalent) View

• Data dependences
  ★ true dependences (RAW)

• Name dependences
  ★ anti-dependences (WAR) and output dependences (WAW)
  ★ not “true” dependences

• Control dependences
Another (Equivalent) View

- Data dependences
  - *true dependences (RAW)*

- Name dependences
  - *anti-dependences (WAR) and output dependences (WAW)*
  - *not “true” dependences*

- Control dependences

Both software and hardware will reorder to improve performance as long as the “observed behavior” of the program does not change.

*(Principle of observational equivalence)*
BASIC COMPILER TECHNIQUES: LOOP UNROLLING AND SCHEDULING
Simple Example

\[
\text{for } (i=1000; \ i>0; \ i--) \\
x[i] = x[i] + s;
\]
Simple Example

```
for (i=1000; i>0; i--)
x[i] = x[i] + s;
```

<table>
<thead>
<tr>
<th>Loop</th>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>L.D</td>
<td>F0,0(R1)</td>
<td>F0=array element</td>
</tr>
<tr>
<td>ADD.D</td>
<td>F4,F0,F2</td>
<td>add scalar in F2</td>
</tr>
<tr>
<td>S.D</td>
<td>F4,0(R1)</td>
<td>store result</td>
</tr>
<tr>
<td>DADDUI</td>
<td>R1,R1,#-8</td>
<td>decrement pointer</td>
</tr>
<tr>
<td>BNE</td>
<td>R1,R2,Loop</td>
<td>branch R1!=R2</td>
</tr>
</tbody>
</table>
## Pipelined Machine

<table>
<thead>
<tr>
<th>Instruction producing result</th>
<th>Instruction using result</th>
<th>Latency in clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP ALU op</td>
<td>Another FP ALU op</td>
<td>3</td>
</tr>
<tr>
<td>FP ALU op</td>
<td>Store double</td>
<td>2</td>
</tr>
<tr>
<td>Load double</td>
<td>FP ALU op</td>
<td>1</td>
</tr>
<tr>
<td>Load double</td>
<td>Store double</td>
<td>0</td>
</tr>
</tbody>
</table>
Scheduled Loop

\[
\text{for } (i=1000; i>0; i--) \\
x[i] = x[i] + s;
\]

<table>
<thead>
<tr>
<th>Instruction producing result</th>
<th>Instruction using result</th>
<th>Latency in clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP ALU op</td>
<td>Another FP ALU op</td>
<td>3</td>
</tr>
<tr>
<td>FP ALU op</td>
<td>Store double</td>
<td>2</td>
</tr>
<tr>
<td>Load double</td>
<td>FP ALU op</td>
<td>1</td>
</tr>
<tr>
<td>Load double</td>
<td>Store double</td>
<td>0</td>
</tr>
</tbody>
</table>

Loop:

- L.D
- stall
- ADD.D
- stall
- stall
- S.D
- DADDUI
- stall
- BNE

Clock cycle issued

- L.D: F0,0(R1) 1
- stall: 2
- ADD.D: F4,F0,F2 3
- stall: 4
- stall: 5
- S.D: F4,0(R1) 6
- DADDUI: R1,R1,#-8 7
- stall: 8
- BNE: R1,R2,Loop 9
Scheduled Loop

for (i=1000; i>0; i--)
    x[i] = x[i] + s;

<table>
<thead>
<tr>
<th>Instruction producing result</th>
<th>Instruction using result</th>
<th>Latency in clock cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>FP ALU op</td>
<td>Another FP ALU op</td>
<td>3</td>
</tr>
<tr>
<td>FP ALU op</td>
<td>Store double</td>
<td>2</td>
</tr>
<tr>
<td>Load double</td>
<td>FP ALU op</td>
<td>1</td>
</tr>
<tr>
<td>Load double</td>
<td>Store double</td>
<td>0</td>
</tr>
</tbody>
</table>

Loop:  
L.D  F0,0(R1)  1  
stall  2  
ADD.D F4,F0,F2  3  
stall  4  
stall  5  
S.D  F4,0(R1)  6  
DADDUI R1,R1,#-8  7  
stall  8  
BNE  R1,R2,Loop  9

Clock cycle issued

Loop:  
L.D  F0,0(R1)  1  
DADDUI R1,R1,#-8  2  
ADD.D F4,F0,F2  3  
stall  4  
stall  5  
S.D  F4,0(R1)  6  
BNE  R1,R2,Loop  7  

Loop Unrolling

```
Loop:  L.D  F0,0(R1) ; F0 = array element
      ADD.D F4,F0,F2 ; add scalar in F2
      S.D  F4,0(R1) ; store result
      DADDUI R1,R1,#-8 ; decrement pointer
                     ; 8 bytes (per DW)
      BNE   R1,R2,Loop ; branch R1!=R2

Loop:  L.D  F0,0(R1)
      ADD.D F4,F0,F2
      S.D  F4,0(R1) ; drop DADDUI & BNE
      L.D  F6,-8(R1)
      ADD.D F8,F6,F2
      S.D  F8,-8(R1) ; drop DADDUI & BNE
      L.D  F10,-16(R1)
      ADD.D F12,F10,F2
      S.D  F12,-16(R1) ; drop DADDUI & BNE
      L.D  F14,-24(R1)
      ADD.D F16,F14,F2
      S.D  F16,-24(R1)
      DADDUI R1,R1,#-32
      BNE   R1,R2,Loop
```
Loop Unrolling

<table>
<thead>
<tr>
<th>Loop:</th>
<th>Code</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>L.D</td>
<td>F0,0(R1)</td>
<td>F0=element of an array</td>
</tr>
<tr>
<td>ADD.D</td>
<td>F4,F0,F2</td>
<td>Add scalar in F2</td>
</tr>
<tr>
<td>S.D</td>
<td>F4,0(R1)</td>
<td>Store result</td>
</tr>
<tr>
<td>DADDUI</td>
<td>R1,R1,#-8</td>
<td>Decrement pointer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8 bytes per DW (per DW)</td>
</tr>
<tr>
<td>BNE</td>
<td>R1,R2,Loop</td>
<td>Branch R1!=R2</td>
</tr>
</tbody>
</table>

Takes 27 cycles

After unrolling:

<table>
<thead>
<tr>
<th>Loop:</th>
<th>Code</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>L.D</td>
<td>F0,0(R1)</td>
<td></td>
</tr>
<tr>
<td>ADD.D</td>
<td>F4,F0,F2</td>
<td></td>
</tr>
<tr>
<td>S.D</td>
<td>F4,0(R1)</td>
<td></td>
</tr>
<tr>
<td>L.D</td>
<td>F6,-8(R1)</td>
<td></td>
</tr>
<tr>
<td>ADD.D</td>
<td>F8,F6,F2</td>
<td></td>
</tr>
<tr>
<td>S.D</td>
<td>F8,-8(R1)</td>
<td></td>
</tr>
<tr>
<td>L.D</td>
<td>F10,-16(R1)</td>
<td></td>
</tr>
<tr>
<td>ADD.D</td>
<td>F12,F10,F2</td>
<td></td>
</tr>
<tr>
<td>S.D</td>
<td>F12,-16(R1)</td>
<td></td>
</tr>
<tr>
<td>L.D</td>
<td>F14,-24(R1)</td>
<td></td>
</tr>
<tr>
<td>ADD.D</td>
<td>F16,F14,F2</td>
<td></td>
</tr>
<tr>
<td>S.D</td>
<td>F16,-24(R1)</td>
<td></td>
</tr>
<tr>
<td>DADDUI</td>
<td>R1,R1,#-32</td>
<td></td>
</tr>
<tr>
<td>BNE</td>
<td>R1,R2,Loop</td>
<td></td>
</tr>
</tbody>
</table>
Schedule Unrolled Loop

<table>
<thead>
<tr>
<th>Loop:</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>L.D</td>
<td>F0,0(R1)</td>
</tr>
<tr>
<td>L.D</td>
<td>F6,-8(R1)</td>
</tr>
<tr>
<td>L.D</td>
<td>F10,-16(R1)</td>
</tr>
<tr>
<td>L.D</td>
<td>F14,-24(R1)</td>
</tr>
<tr>
<td>ADD.D</td>
<td>F4,F0,F2</td>
</tr>
<tr>
<td>ADD.D</td>
<td>F8,F6,F2</td>
</tr>
<tr>
<td>ADD.D</td>
<td>F12,F10,F2</td>
</tr>
<tr>
<td>ADD.D</td>
<td>F16,F14,F2</td>
</tr>
<tr>
<td>S.D</td>
<td>F4,0(R1)</td>
</tr>
<tr>
<td>S.D</td>
<td>F8,-8(R1)</td>
</tr>
<tr>
<td>DADDUI</td>
<td>R1,R1,#-32</td>
</tr>
<tr>
<td>S.D</td>
<td>F12,16(R1)</td>
</tr>
<tr>
<td>S.D</td>
<td>F16,8(R1)</td>
</tr>
<tr>
<td>BNE</td>
<td>R1,R2,Loop</td>
</tr>
</tbody>
</table>
Schedule Unrolled Loop

<table>
<thead>
<tr>
<th>Loop:</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>L.D</td>
<td>F0,0(R1)</td>
</tr>
<tr>
<td>L.D</td>
<td>F6,-8(R1)</td>
</tr>
<tr>
<td>L.D</td>
<td>F10,-16(R1)</td>
</tr>
<tr>
<td>L.D</td>
<td>F14,-24(R1)</td>
</tr>
<tr>
<td>ADD.D</td>
<td>F4,F0,F2</td>
</tr>
<tr>
<td>ADD.D</td>
<td>F8,F6,F2</td>
</tr>
<tr>
<td>ADD.D</td>
<td>F12,F10,F2</td>
</tr>
<tr>
<td>ADD.D</td>
<td>F16,F14,F2</td>
</tr>
<tr>
<td>S.D</td>
<td>F4,0(R1)</td>
</tr>
<tr>
<td>S.D</td>
<td>F8,-8(R1)</td>
</tr>
<tr>
<td>DADDUI</td>
<td>R1,R1,#-32</td>
</tr>
<tr>
<td>S.D</td>
<td>F12,16(R1)</td>
</tr>
<tr>
<td>S.D</td>
<td>F16,8(R1)</td>
</tr>
<tr>
<td>BNE</td>
<td>R1,R2,Loop</td>
</tr>
</tbody>
</table>

Takes 14 cycles
Loop Unrolling

- Unroll a small number of times (called unroll factor)
  ★ reduces branches
  ★ bigger body enables better instruction scheduling
- Rename registers to avoid name dependences
  ★ too much unrolling can cause register pressure
- Reorder instructions to reduce stalls
- Generate startup and / or cleanup loops for iterations that are not multiple of unroll factor
BRANCH PREDICTION
Static Branch Prediction

• Delay slots help
  ★ can schedule instructions in the delay slots from the branch direction taken more often

• Static prediction approaches
  ★ predict taken (average misprediction for SPEC is 34%, ranging from 9% to 59%)
  ★ use profile information
Misprediction Rate Based on Profile Data
Spec92 Benchmarks

The graph shows the misprediction rate for various benchmarks under integer and floating point operations. The benchmarks include compress, eqntott, espresso, gcc, li, doduc, ear, hydro2d, mdjfp, and su2cor.

- Compress has a misprediction rate of 12% under integer operations.
- Eqntott has a misprediction rate of 22% under integer operations.
- Espresso has a misprediction rate of 18% under integer operations.
- Gcc and li both have a misprediction rate of 11% and 12% respectively under integer operations.
- Doduc, ear, hydro2d, mdjfp, and su2cor have misprediction rates of 5%, 6%, 9%, 10%, and 15% respectively under floating point operations.
Dynamic Branch Prediction

- “Branch prediction buffer” or “branch history table”
- Small 1-bit memory (cache) indexed by lower bits of address
Dynamic Branch Prediction

• “Branch prediction buffer” or “branch history table”
• Small r-bit memory (cache) indexed by lower bits of address

Problem: Predicts incorrectly twice

Address bits

bits to index BPB
Two-bit Predictors

![Diagram of two-bit predictors showing logic flow and outcomes for prediction of taken or not taken based on two-bit prediction system.]
Prediction Accuracy: 4K 2-bit Entries (Spec89 Benchmarks)
Prediction Accuracy: 4K vs Infinite

- **nasa7**: 1% (4K), 0% (Infinite)
- **matrix300**: 0% (4K), 0% (Infinite)
- **tomcatv**: 1% (4K), 0% (Infinite)
- **doduc**: 5% (4K), 5% (Infinite)
- **spice**: 9% (4K), 9% (Infinite)
- **fpppp**: 9% (4K), 9% (Infinite)
- **gcc**: 12% (4K), 11% (Infinite)
- **espresso**: 5% (4K), 5% (Infinite)
- **eqntott**: 18% (4K), 18% (Infinite)
- **li**: 10% (4K), 10% (Infinite)

**SPEC89 benchmarks**
Correlating Branch Predictors

• Motivating example

```java
if (aa == 2)
    aa = 0;
if (bb == 2)
    bb = 0;
if (aa != bb) {
    ...
}
```

• Idea

★ “correlate” last m branches
General Correlating Predictors

- \((m,n)\) predictor
  - use last \(m\) branches to predict using \(n\) bit saturating counters

Address bits

- Bits to index BPB
- Global shift register \((m\text{ bits})\)
- Branch Prediction Buffer \(n\text{ bits}\)
General Correlating Predictors

- $(m,n)$ predictor
  - Use last $m$ branches to predict using $n$ bit saturating counters
General Correlating Predictors

- \((m,n)\) predictor
  - use last \(m\) branches to predict using \(n\) bit saturating counters

if \(b = \) number of entries in BTB,

\[
\text{number of bits in BTB} = 2^m \times 2^n \times b
\]
Comparison of 2-bit Predictors

The chart compares the frequency of mispredictions for various SPEC benchmarks using 2-bit predictors with different entry sizes:

- 4096 entries: 2 bits per entry
- Unlimited entries: 2 bits per entry
- 1024 entries: (2,2)

The benchmarks and their misprediction rates are as follows:

- nasa7: 1%
- matrix300: 0%
- tomcatv: 1%
- doduc: 5%
- spice: 9%
- fpppp: 9%
- gcc: 11%
- espresso: 5%
- eqntott: 18%
- li: 10%

The chart illustrates the impact of different predictor sizes on misprediction rates for various applications.
Tournament Predictors (Spec89 Benchmarks)

![Graph showing conditional branch misprediction rate vs. total predictor size for different predictor types.]
DYNAMIC SCHEDULING
Comments on Assignment 1

• Start NOW!
• Use course blog for discussions
• Needs work, but rewards await you
• You may mix languages
  ★ e.g., parsing might be easier with a scripting language
• Matrix-matrix computation == matrix multiply
• Note differences in Tomasulo’s approach for assignment from the textbook Figure 2.9
• Extra credit possibilities (case-by-case)
  ★ forwarding in pipelined architecture
  ★ speculation in Tomasulo’s approach
  ★ other applications
  ★ transformations such as, loop unrolling
Why Dynamic Scheduling?

• Reduces stalls
  ★ tolerates data hazards
  ★ tolerates cache misses

• Code optimized for one pipeline can run on another

• Can work with statically scheduled code

• BUT, needs significantly more hardware
Dynamic Scheduling: Idea

• Out-of-order execution
• Out-of-order completion
• Additional issues:
  ★ WAR and WAW hazards
  ★ precise exceptions
• Implementation:
  ★ split ID into Issue and Read Operands
  ★ multiple instructions in execution need multiple functional units
• In order issue, out-of-order execution
Dynamic Scheduling: Idea

- Out-of-order execution
- Out-of-order completion
- Additional issues:
  - WAR and WAW hazards
  - Precise exceptions
- Implementation:
  - Split ID into Issue and Read Operands
  - Multiple instructions in execution need multiple functional units
- In order issue, out-of-order execution

DIV.D F0, F2, F4
ADD.D F10,F0,F8
SUB.D F12,F8,F14
Dynamic Scheduling: Idea

- Out-of-order execution
- Out-of-order completion
- Additional issues:
  - WAR and WAW hazards
  - Precise exceptions
- Implementation:
  - Split ID into Issue and Read Operands
  - Multiple instructions in execution need multiple functional units
- In order issue, out-of-order execution

```
DIV.D F0, F2, F4
ADD.D F10, F0, F8
SUB.D F12, F8, F14
```

```
DIV.D F0, F2, F4
ADD.D F6, F0, F8
SUB.D F8, F10, F14
MUL.D F6, F10, F8
```
Tomasulo’s Approach

• Invented by Robert Tomasulo for IBM 360
  ★ 360 had only 4 FP regs and long floating point delays
• Avoids RAW and WAW hazards by *register renaming*
  ★ use of *reservation stations*

```
DIV.D    F0, F2, F4
ADD.D    F6,F0,F8
S.D      F6,0(R1)
SUB.D    F8,F10,F14
MUL.D    F6,F10,F8
```
Tomasulo’s Approach

• Invented by Robert Tomasulo for IBM 360
  ★ 360 had only 4 FP regs and long floating point delays

• Avoids RAW and WAW hazards by register renaming
  ★ use of reservation stations

DIV.D F0, F2, F4
ADD.D F6, F0, F8
S.D F6, 0(R1)
SUB.D F8, F10, F14
MUL.D F6, F10, F8

DIV.D F0, F2, F4
ADD.D S, F0, F8
S.D S, 0(R1)
SUB.D T, F10, F14
MUL.D F6, F10, T
Tomasulo’s Approach: Basic Structure
Tomasulo’s Approach: Steps

• Issue
  ★ get next instruction from queue
  ★ move to a matching reservation station (or stall if none available)
  ★ get operand values if in registers, else keep track of units that will produce them

• Execute
  ★ if an operand not available, monitor the CDB
  ★ if all operands are ready, execute the instruction when the functional unit becomes available
  ★ loads and stores take two steps: read register and wait for memory

• Write results
  ★ functional units write into CDB (and from there into registers)
  ★ stores write to memory when both value and store register are available
Tomasulo’s Approach: Fields

- Reservation station:
  - Op: operation
  - Qj, Qk: operands, to come from reservation stations
  - Vj, Vk: operands, available in registers
  - A: effective address (initialized with immediate value)
  - Busy: bit to indicate the reservation station is busy

- Register file:
  - Qi: the number of the reservation station whose result will go into this register; blank (or zero) indicates the register already has the value
Example

1. L.D F6,32(R2)
2. L.D F2,44(R3)
3. MUL.D F0,F2,F4
4. SUB.D F8,F2,F6
5. DIV.D F10,F0,F6
6. ADD.D F6,F8,F2
### Example

#### Instruction status

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Issue</th>
<th>Execute</th>
<th>Write Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>L.D F6,32(R2)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>L.D F2,44(R3)</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>MUL.D F0,F2,F4</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUB.D F8,F2,F6</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIV.D F10,F0,F6</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD.D F6,F8,F2</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Reservation stations

<table>
<thead>
<tr>
<th>Name</th>
<th>Busy</th>
<th>Op</th>
<th>Vj</th>
<th>Vk</th>
<th>Qj</th>
<th>Qk</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load1</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Load2</td>
<td>yes</td>
<td>Load</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>45 + Regs[R3]</td>
</tr>
<tr>
<td>Add1</td>
<td>yes</td>
<td>SUB</td>
<td></td>
<td>Mem[34 + Regs[R2]]</td>
<td>Load2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add2</td>
<td>yes</td>
<td>ADD</td>
<td></td>
<td></td>
<td>Add1</td>
<td>Load2</td>
<td></td>
</tr>
<tr>
<td>Add3</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mult1</td>
<td>yes</td>
<td>MUL</td>
<td>Regs[F4]</td>
<td></td>
<td>Load2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mult2</td>
<td>yes</td>
<td>DIV</td>
<td>Mem[34 + Regs[R2]]</td>
<td></td>
<td>Mult1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Register status

<table>
<thead>
<tr>
<th>Field</th>
<th>F0</th>
<th>F2</th>
<th>F4</th>
<th>F6</th>
<th>F8</th>
<th>F10</th>
<th>F12</th>
<th>...</th>
<th>F30</th>
</tr>
</thead>
<tbody>
<tr>
<td>Qi</td>
<td>Mult1</td>
<td>Load2</td>
<td>Add2</td>
<td>Add1</td>
<td>Mult2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Instruction status

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Issue</th>
<th>Execute</th>
<th>Write Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>L.D F6,32(R2)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>L.D F2,44(R3)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>MUL.D F0,F2,F4</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>SUB.D F8,F2,F6</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DIV.D F10,F0,F6</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD.D F6,F8,F2</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

### Reservation stations

<table>
<thead>
<tr>
<th>Name</th>
<th>Busy</th>
<th>Op</th>
<th>Vj</th>
<th>Vk</th>
<th>Qj</th>
<th>Qk</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load1</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Load2</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add1</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add2</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add3</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mult1</td>
<td>yes</td>
<td>MUL</td>
<td>Mem[45 + Regs[R3]]</td>
<td>Regs[F4]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mult2</td>
<td>yes</td>
<td>DIV</td>
<td>Mem[34 + Regs[R2]]</td>
<td>Mult1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Register status

<table>
<thead>
<tr>
<th>Field</th>
<th>F0</th>
<th>F2</th>
<th>F4</th>
<th>F6</th>
<th>F8</th>
<th>F10</th>
<th>F12</th>
<th>…</th>
<th>F30</th>
</tr>
</thead>
<tbody>
<tr>
<td>Qi</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

|     | Mult1 |     |     |     |     |     |     |   |     |
|     |       |     |     |     |     |     |     |   |     |
Observations

• RAW hazards handled by waiting for operands
• WAR and WAW hazards handled by register renaming
  ★ only WAR and WAW hazards between instructions currently in the pipeline are handled; is this a problem?
  ★ larger number of hidden names reduces name dependences
• CDB implements forwarding
Loop Example

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Opcode</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>L.D</td>
<td>F0,0(R1)</td>
<td></td>
</tr>
<tr>
<td>MUL.D</td>
<td>F4,F0,F2</td>
<td></td>
</tr>
<tr>
<td>S.D</td>
<td>F4,0(R1)</td>
<td></td>
</tr>
<tr>
<td>DADDIU</td>
<td>R1,R1,-8</td>
<td></td>
</tr>
<tr>
<td>BNE</td>
<td>R1,R2,loop</td>
<td></td>
</tr>
</tbody>
</table>
## Loop Example

### Instruction status

<table>
<thead>
<tr>
<th>Instruction</th>
<th>From iteration</th>
<th>Issue</th>
<th>Execute</th>
<th>Write Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>L.D F0,0(R1)</td>
<td>1</td>
<td>√</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>MUL.D F4,F0,F2</td>
<td>1</td>
<td>√</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S.D F4,0(R1)</td>
<td>1</td>
<td>√</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L.D F0,0(R1)</td>
<td>2</td>
<td>√</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>MUL.D F4,F0,F2</td>
<td>2</td>
<td>√</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S.D F4,0(R1)</td>
<td>2</td>
<td>√</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Reservation stations

<table>
<thead>
<tr>
<th>Name</th>
<th>Busy</th>
<th>Op</th>
<th>Vj</th>
<th>Vk</th>
<th>Qj</th>
<th>Qk</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load1</td>
<td>yes</td>
<td>Load</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Regs[R1] + 0</td>
</tr>
<tr>
<td>Load2</td>
<td>yes</td>
<td>Load</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Regs[R1] – 8</td>
</tr>
<tr>
<td>Add1</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add2</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add3</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mult1</td>
<td>yes</td>
<td>MUL</td>
<td>Regs[F2]</td>
<td></td>
<td></td>
<td>Load1</td>
<td></td>
</tr>
<tr>
<td>Mult2</td>
<td>yes</td>
<td>MUL</td>
<td>Regs[F2]</td>
<td></td>
<td></td>
<td>Load2</td>
<td></td>
</tr>
<tr>
<td>Store1</td>
<td>yes</td>
<td>Store</td>
<td>Regs[R1]</td>
<td></td>
<td></td>
<td>Mult1</td>
<td></td>
</tr>
<tr>
<td>Store2</td>
<td>yes</td>
<td>Store</td>
<td>Regs[R1] – 8</td>
<td></td>
<td></td>
<td>Mult2</td>
<td></td>
</tr>
</tbody>
</table>

### Register status

<table>
<thead>
<tr>
<th>Field</th>
<th>F0</th>
<th>F2</th>
<th>F4</th>
<th>F6</th>
<th>F8</th>
<th>F10</th>
<th>F12</th>
<th>...</th>
<th>F30</th>
</tr>
</thead>
<tbody>
<tr>
<td>Qi</td>
<td>Load2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Summary of Tomasulo’s Approach

• Need to check WAR and WAW hazards
  ★ through registers
  ★ through loads and stores

• Works very well if branches predicted accurately
  ★ instruction not allowed to execute unless all preceding branches finished

• Out-of-order completion results in imprecise exceptions

• Widely popular
  ★ high performance without compiler assistance
  ★ can hide cache latencies
  ★ reasonable performance for code difficult to schedule statically
  ★ key component of speculation
HARDWARE-BASED SPECULATION
Speculation: Handling Control Dependences

• Fetch, issue, and **execute** instructions as if branch predictions always right

• Mechanism to handle situation where prediction was incorrect

• Combines three key ideas:
  ★ dynamic branch prediction
  ★ speculation to execute without waiting for control dependences to resolve
  ★ dynamic scheduling
Speculation: Handling Control Dependences

• Fetch, issue, and **execute** instructions as if branch predictions always right

• Mechanism to handle situation where prediction was incorrect

• Combines three key ideas:
  ★ dynamic branch prediction
  ★ speculation to execute without waiting for control dependences to resolve
  ★ dynamic scheduling

Data flow execution
Extending Tomasulo’s Algorithm

• Separate execution from completion
  ★ execute: when data dependences are resolved
  ★ commit: when control dependences are resolved

• Out-of-order execution, but in-order commit

• Store uncommitted instructions in a reorder buffer (ROB)

• Written results found in ROB, until committed
  ★ similar to store buffer

• Register file / memory written upon commit
Tomasulo’s Approach + Speculation
Tomasulo’s Approach + Speculation

Fields in ROB
1. Instruction type
2. Destination
3. Value
4. Ready
Speculation Steps

**Issue → Execute → Write → Commit**

- **Issue**
  - ★ get instruction from queue
  - ★ issue if empty reservation station and empty ROB slot
    - ◆ otherwise, stall
  - ★ update control fields to indicate buffers are in use
  - ★ send the reserved ROB entry number to the reservation station for tagging
Speculation Steps

\[ \text{Issue} \rightarrow \text{Execute} \rightarrow \text{Write} \rightarrow \text{Commit} \]

• Execute
  ★ if an operand not ready, monitor the CDB
    ★ checks RAW hazards
  ★ execute when both operands available
    ★ loads require two steps (why?)
    ★ stores only need base register (why?)
Speculation Steps

Issue → Execute → Write → Commit

• Write
  ★ upon completion, write result+tag on CDB
  ★ CDB is read by ROB and any waiting reservation stations
  ★ mark reservation station available
  ★ for store:
    ★ write in ROB’s Value field if value available
    ★ otherwise, keep monitoring CDB for the value
Speculation Steps

*Issue* $\rightarrow$ *Execute* $\rightarrow$ *Write* $\rightarrow$ **Commit**

- **Commit** (also called “completion” or “graduation”)
  - normal commit
    - update the register with the result, remove entry from ROB
  - store
    - update memory with the result, remove entry from ROB
  - correctly predicted branch
    - finish the branch
  - incorrectly predicted branch
    - flush the ROB
    - restart at the correct successor
## Example

<table>
<thead>
<tr>
<th></th>
<th>Instruction</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>L.D</td>
<td>F6,32(R2)</td>
</tr>
<tr>
<td>2.</td>
<td>L.D</td>
<td>F2,44(R3)</td>
</tr>
<tr>
<td>3.</td>
<td>MUL.D</td>
<td>F0,F2,F4</td>
</tr>
<tr>
<td>4.</td>
<td>SUB.D</td>
<td>F8,F2,F6</td>
</tr>
<tr>
<td>5.</td>
<td>DIV.D</td>
<td>F10,F0,F6</td>
</tr>
<tr>
<td>6.</td>
<td>ADD.D</td>
<td>F6,F8,F2</td>
</tr>
</tbody>
</table>
### Reorder buffer

<table>
<thead>
<tr>
<th>Entry</th>
<th>Busy</th>
<th>Instruction</th>
<th>State</th>
<th>Destination</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>no</td>
<td>L.D</td>
<td>F6,32(R2)</td>
<td>Commit</td>
<td>F6 Mem[34 + Regs[R2]]</td>
</tr>
<tr>
<td>2</td>
<td>no</td>
<td>L.D</td>
<td>F2,44(R3)</td>
<td>Commit</td>
<td>F2 Mem[45 + Regs[R3]]</td>
</tr>
<tr>
<td>3</td>
<td>yes</td>
<td>MUL.D</td>
<td>F0,F2,F4</td>
<td>Write result</td>
<td>F0 #2 × Regs[F4]</td>
</tr>
<tr>
<td>4</td>
<td>yes</td>
<td>SUB.D</td>
<td>F8,F2,F6</td>
<td>Write result</td>
<td>F8 #2 − #1</td>
</tr>
<tr>
<td>5</td>
<td>yes</td>
<td>DIV.D</td>
<td>F10,F0,F6</td>
<td>Execute</td>
<td>F10</td>
</tr>
<tr>
<td>6</td>
<td>yes</td>
<td>ADD.D</td>
<td>F6,F8,F2</td>
<td>Write result</td>
<td>F6 #4 + #2</td>
</tr>
</tbody>
</table>

### Reservation stations

<table>
<thead>
<tr>
<th>Name</th>
<th>Busy</th>
<th>Op</th>
<th>Vj</th>
<th>Vk</th>
<th>Qj</th>
<th>Qk</th>
<th>Dest</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load1</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Load2</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add1</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add2</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Add3</td>
<td>no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mult1</td>
<td>no</td>
<td>MUL.D</td>
<td>Mem[45 + Regs[R3]]</td>
<td>Regs[F4]</td>
<td></td>
<td></td>
<td></td>
<td>#3</td>
</tr>
<tr>
<td>Mult2</td>
<td>yes</td>
<td>DIV.D</td>
<td>Mem[34 + Regs[R2]]</td>
<td>#3</td>
<td></td>
<td></td>
<td></td>
<td>#5</td>
</tr>
</tbody>
</table>

### FP register status

<table>
<thead>
<tr>
<th>Field</th>
<th>F0</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>F4</th>
<th>F5</th>
<th>F6</th>
<th>F7</th>
<th>F8</th>
<th>F10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reorder #</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>6</td>
<td></td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Busy</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>...</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>
Loop Example

<table>
<thead>
<tr>
<th>Instruction</th>
<th>R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>L.D</td>
<td>F0,0(R1)</td>
</tr>
<tr>
<td>MUL.D</td>
<td>F4,F0,F2</td>
</tr>
<tr>
<td>S.D</td>
<td>F4,0(R1)</td>
</tr>
<tr>
<td>DADDIU</td>
<td>R1,R1,-8</td>
</tr>
<tr>
<td>BNE</td>
<td>R1,R2,loop</td>
</tr>
</tbody>
</table>
## Loop Example

<table>
<thead>
<tr>
<th>Entry</th>
<th>Busy</th>
<th>Instruction</th>
<th>State</th>
<th>Destination</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>no</td>
<td>L.D F0,0(R1)</td>
<td>Commit</td>
<td>F0</td>
<td>Mem[0 + Regs[R1]]</td>
</tr>
<tr>
<td>2</td>
<td>no</td>
<td>MUL.D F4,F0,F2</td>
<td>Commit</td>
<td>F4</td>
<td>#1 × Regs[F2]</td>
</tr>
<tr>
<td>3</td>
<td>yes</td>
<td>S.D F4,0(R1)</td>
<td>Write result</td>
<td>0 + Regs[R1]</td>
<td>#2</td>
</tr>
<tr>
<td>4</td>
<td>yes</td>
<td>DADDIU R1,R1,#-8</td>
<td>Write result</td>
<td>R1</td>
<td>Regs[R1] – 8</td>
</tr>
<tr>
<td>5</td>
<td>yes</td>
<td>BNE R1,R2,Loop</td>
<td>Write result</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>yes</td>
<td>L.D F0,0(R1)</td>
<td>Write result</td>
<td>F0</td>
<td>Mem[#4]</td>
</tr>
<tr>
<td>7</td>
<td>yes</td>
<td>MUL.D F4,F0,F2</td>
<td>Write result</td>
<td>F4</td>
<td>#6 × Regs[F2]</td>
</tr>
<tr>
<td>8</td>
<td>yes</td>
<td>S.D F4,0(R1)</td>
<td>Write result</td>
<td>0 + #4</td>
<td>#7</td>
</tr>
<tr>
<td>9</td>
<td>yes</td>
<td>DADDIU R1,R1,#-8</td>
<td>Write result</td>
<td>R1</td>
<td>#4 – 8</td>
</tr>
<tr>
<td>10</td>
<td>yes</td>
<td>BNE R1,R2,Loop</td>
<td>Write result</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### FP register status

<table>
<thead>
<tr>
<th>Field</th>
<th>F0</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>F4</th>
<th>F5</th>
<th>F6</th>
<th>F7</th>
<th>F8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reorder #</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>Busy</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>
Observations on Speculation

• Speculation enables precise exception handling
  ★ defer exception handling until instruction ready to commit

• Branches are critical to performance
  ★ prediction accuracy
  ★ latency of misprediction detection
  ★ misprediction recovery time

• Must avoid hazards through memory
  ★ WAR and WAW already taken care of (how?)
  ★ for RAW
    ★ don’t allow load to proceed if an active ROB entry has Destination field matching with A field of load
    ★ maintain program order for effective address computation (why?)
SUPERSCALAR PROCESSORS
Improving ILP: Multiple Issue

- Statically scheduled superscalar processors
- VLIW (Very Long Instruction Word) processors
- Dynamically scheduled superscalar processors
## Multiple Issue Processor Types

<table>
<thead>
<tr>
<th>Common name</th>
<th>Issue structure</th>
<th>Hazard detection</th>
<th>Scheduling</th>
<th>Distinguishing characteristic</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Superscalar (static)</td>
<td>dynamic</td>
<td>hardware</td>
<td>static</td>
<td>in-order execution</td>
<td>mostly in the embedded space: MIPS and ARM</td>
</tr>
<tr>
<td>Superscalar (dynamic)</td>
<td>dynamic</td>
<td>hardware</td>
<td>dynamic</td>
<td>some out-of-order execution, but no speculation</td>
<td>none at the present</td>
</tr>
<tr>
<td>Superscalar (speculative)</td>
<td>dynamic</td>
<td>hardware</td>
<td>dynamic with speculation</td>
<td>out-of-order execution with speculation</td>
<td>Pentium 4, MIPS R12K, IBM Power5</td>
</tr>
<tr>
<td>VLIW/LIW</td>
<td>static</td>
<td>primarily software</td>
<td>static</td>
<td>all hazards determined and indicated by compiler (often implicitly)</td>
<td>most examples are in the embedded space, such as the TI C6x</td>
</tr>
<tr>
<td>EPIC</td>
<td>primarily static</td>
<td>primarily software</td>
<td>mostly static</td>
<td>all hazards determined and indicated explicitly by the compiler</td>
<td>Itanium</td>
</tr>
</tbody>
</table>
Dyn. Scheduling+Multiple Issue+Speculation

- **Design parameters**
  - two-way issue (two instruction issues per cycle)
  - pipelined and separate integer and FP functional units
  - dynamic scheduling, but not out-of-order issue
  - speculative execution

- **Task per issue**: assign reservation station and update pipeline control tables (i.e., control signals)

- **Two possible techniques**
  - do the task in half a clock cycle
  - build wider logic to issue any pair of instructions together

- **Modern processors use both** (4 or more way superscalar)
## Loop Example

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>R2,0(R1) ;R2=array element</td>
</tr>
<tr>
<td>DADDIU</td>
<td>R2,R2,#1 ;increment R2</td>
</tr>
<tr>
<td>SD</td>
<td>R2,0(R1) ;store result</td>
</tr>
<tr>
<td>DADDIU</td>
<td>R1,R1,#8 ;increment pointer</td>
</tr>
<tr>
<td>BNE</td>
<td>R2,R2,Loop ;branch if not last</td>
</tr>
</tbody>
</table>
Two-Way Issue, Without Speculation

<table>
<thead>
<tr>
<th>Iteration number</th>
<th>Instructions</th>
<th>Issues at clock cycle number</th>
<th>Executes at clock cycle number</th>
<th>Memory access at clock cycle number</th>
<th>Write CDB at clock cycle number</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LD R2,0(R1)</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>First issue</td>
</tr>
<tr>
<td>1</td>
<td>DADDIU R2,R2,#1</td>
<td>1</td>
<td>5</td>
<td></td>
<td>6</td>
<td>Wait for LW</td>
</tr>
<tr>
<td>1</td>
<td>SD R2,0(R1)</td>
<td>2</td>
<td>3</td>
<td>7</td>
<td></td>
<td>Wait for DADDIU</td>
</tr>
<tr>
<td>1</td>
<td>DADDIU R1,R1,#8</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td></td>
<td>Execute directly</td>
</tr>
<tr>
<td>1</td>
<td>BNE R2,R3,LOOP</td>
<td>3</td>
<td>7</td>
<td></td>
<td></td>
<td>Wait for DADDIU</td>
</tr>
<tr>
<td>2</td>
<td>LD R2,0(R1)</td>
<td>4</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>Wait for BNE</td>
</tr>
<tr>
<td>2</td>
<td>DADDIU R2,R2,#1</td>
<td>4</td>
<td>11</td>
<td></td>
<td>12</td>
<td>Wait for LW</td>
</tr>
<tr>
<td>2</td>
<td>SD R2,0(R1)</td>
<td>5</td>
<td>9</td>
<td>13</td>
<td></td>
<td>Wait for DADDIU</td>
</tr>
<tr>
<td>2</td>
<td>DADDIU R1,R1,#8</td>
<td>5</td>
<td>8</td>
<td></td>
<td>9</td>
<td>Wait for BNE</td>
</tr>
<tr>
<td>2</td>
<td>BNE R2,R3,LOOP</td>
<td>6</td>
<td>13</td>
<td></td>
<td></td>
<td>Wait for DADDIU</td>
</tr>
<tr>
<td>3</td>
<td>LD R2,0(R1)</td>
<td>7</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>Wait for BNE</td>
</tr>
<tr>
<td>3</td>
<td>DADDIU R2,R2,#1</td>
<td>7</td>
<td>17</td>
<td></td>
<td>18</td>
<td>Wait for LW</td>
</tr>
<tr>
<td>3</td>
<td>SD R2,0(R1)</td>
<td>8</td>
<td>15</td>
<td>19</td>
<td></td>
<td>Wait for DADDIU</td>
</tr>
<tr>
<td>3</td>
<td>DADDIU R1,R1,#8</td>
<td>8</td>
<td>14</td>
<td></td>
<td>15</td>
<td>Wait for BNE</td>
</tr>
<tr>
<td>3</td>
<td>BNE R2,R3,LOOP</td>
<td>9</td>
<td>19</td>
<td></td>
<td></td>
<td>Wait for DADDIU</td>
</tr>
</tbody>
</table>
Two-Way Issue, With Speculation

<table>
<thead>
<tr>
<th>Iteration number</th>
<th>Instructions</th>
<th>Issues at clock number</th>
<th>Executes at clock number</th>
<th>Read access at clock number</th>
<th>Write CDB at clock number</th>
<th>Commits at clock number</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LD</td>
<td>R2,0(R1)</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5 First issue</td>
</tr>
<tr>
<td>1</td>
<td>DADDIU</td>
<td>R2,R2,#1</td>
<td>1</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>Wait for LW</td>
</tr>
<tr>
<td>1</td>
<td>SD</td>
<td>R2,0(R1)</td>
<td>2</td>
<td>3</td>
<td>7</td>
<td>7</td>
<td>Wait for DADDIU</td>
</tr>
<tr>
<td>1</td>
<td>DADDIU</td>
<td>R1,R1,#8</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>8</td>
<td>Commit in order</td>
</tr>
<tr>
<td>1</td>
<td>BNE</td>
<td>R2,R3,LOOP</td>
<td>3</td>
<td>7</td>
<td>8</td>
<td>8</td>
<td>Wait for DADDIU</td>
</tr>
<tr>
<td>2</td>
<td>LD</td>
<td>R2,0(R1)</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>No execute delay</td>
</tr>
<tr>
<td>2</td>
<td>DADDIU</td>
<td>R2,R2,#1</td>
<td>4</td>
<td>8</td>
<td>9</td>
<td>9</td>
<td>Wait for LW</td>
</tr>
<tr>
<td>2</td>
<td>SD</td>
<td>R2,0(R1)</td>
<td>5</td>
<td>6</td>
<td>9</td>
<td>10</td>
<td>Wait for DADDIU</td>
</tr>
<tr>
<td>2</td>
<td>DADDIU</td>
<td>R1,R1,#8</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>11</td>
<td>Commit in order</td>
</tr>
<tr>
<td>2</td>
<td>BNE</td>
<td>R2,R3,LOOP</td>
<td>6</td>
<td>10</td>
<td>11</td>
<td>11</td>
<td>Wait for DADDIU</td>
</tr>
<tr>
<td>3</td>
<td>LD</td>
<td>R2,0(R1)</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>Earliest possible</td>
</tr>
<tr>
<td>3</td>
<td>DADDIU</td>
<td>R2,R2,#1</td>
<td>7</td>
<td>11</td>
<td>12</td>
<td>12</td>
<td>Wait for LW</td>
</tr>
<tr>
<td>3</td>
<td>SD</td>
<td>R2,0(R1)</td>
<td>8</td>
<td>9</td>
<td>12</td>
<td>13</td>
<td>Wait for DADDIU</td>
</tr>
<tr>
<td>3</td>
<td>DADDIU</td>
<td>R1,R1,#8</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>14</td>
<td>Executes earlier</td>
</tr>
<tr>
<td>3</td>
<td>BNE</td>
<td>R2,R3,LOOP</td>
<td>9</td>
<td>13</td>
<td>11</td>
<td>14</td>
<td>Wait for DADDIU</td>
</tr>
</tbody>
</table>

Compare with 14 and 19, without speculation
ADVANCED TECHNIQUES FOR INSTRUCTION DELIVERY AND SPECULATION
Increasing Fetch Bandwidth: **Branch Target Buffers**

- **PC of instruction to fetch**
- **Look up**
- **Predicted PC**

**Number of entries in branch-target buffer**

**Branch predicted taken or untaken**

- **No:** instruction is not predicted to be branch; proceed normally
- **Yes:** then instruction is branch and predicted PC should be used as the next PC

© 2007 Elsevier, Inc. All rights reserved.
Increasing Fetch Bandwidth: 

**Branch Target Buffers**

- **IF**
  - Entry found in branch-target buffer?
- **ID**
  - Is instruction a taken branch?
- **EX**
  - Branch correctly predicted; continue execution with no stalls
  - Mispredicted branch, kill fetched instruction; restart fetch at other target; delete entry from target buffer
  - Enter branch instruction address and next PC into branch-target buffer
  - Send out predicted PC
  - Send PC to memory and branch-target buffer
  - Normal instruction execution
Increasing Fetch Bandwidth: *Branch Target Buffers*

<table>
<thead>
<tr>
<th>Instruction in buffer</th>
<th>Prediction</th>
<th>Actual branch</th>
<th>Penalty cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>yes</td>
<td>taken</td>
<td>taken</td>
<td>0</td>
</tr>
<tr>
<td>yes</td>
<td>taken</td>
<td>not taken</td>
<td>2</td>
</tr>
<tr>
<td>no</td>
<td>taken</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>no</td>
<td>not taken</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Increasing Fetch Bandwidth: 
**Branch Target Buffers: Variation**

![Diagram showing branch target buffer operation]

- **PC of instruction to fetch**
- **Look up**
- **Target instruction(s)**
  - +
  - Predicted PC

**Number of entries in branch-target buffer**

- **Branch predicted taken or not taken**
- **No:** instruction is not predicted to be branch; proceed normally
- **Yes:** then instruction is branch and predicted PC should be used as the next PC
Increasing Fetch Bandwidth

- Return address predictors
  ★ assuming a special instruction for return (not a jump)
  ★ returns are indirect (why?)
  ★ more important for OO and dynamic languages (esp. VMs)
Return Address Prediction Accuracy

![Graph showing return address prediction accuracy across different buffer entries for various applications, including go, m88ksim, cc1, compress, xlisp, ijpeg, perl, and vortex.](image-url)
Increasing Fetch Bandwidth

• Return address predictors
  ★ assuming a special instruction for return (not a jump)
  ★ returns are indirect (why?)
  ★ more important for OO and dynamic languages (esp. VMs)

• Integrated (stand-alone) instruction fetch units (not just a pipeline stage)
  ★ integrated branch prediction
  ★ instruction prefetch (when might this be useful?)
  ★ instruction memory access and buffering (e.g., trace cache on Pentium 4)
Improving Speculation: Register Renaming

• Use an extended set of “physical” registers, instead of “architectural” registers
  ★ beware of the terminology difference with memory system

• Physical registers hold values, instead of reservation stations or ROB

• Map of architectural to physical registers

• Commit steps:
  ★ make the map entry for written architectural reg. “permanent”
  ★ deallocate physical registers containing “older” value

• Deallocating physical registers
  ★ look at the source operands to find unused physical registers
  ★ wait until next instruction writing the same architectural register commits (how does this work?)
Improving Speculation

• Limiting speculation
  ★ avoid speculating when it may cause an expensive exception, such as TLB miss or L2 miss

• Speculating through multiple branches
  ★ speculate on a subsequent branch while the previous one still pending
    ★ useful when high branch frequency, clustered branches, or long functional unit delays
  ★ speculating on more than branch in **one** cycle
    ★ no architecture combines multiple branch prediction in one cycle with full speculation

• Value prediction
INTEL PENTIUM 4
Overall Architecture
Overall Architecture
Overall Architecture
Overall Architecture
Overall Architecture
# Characteristics

<table>
<thead>
<tr>
<th>Feature</th>
<th>Size</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Front-end branch-target buffer</td>
<td>4K entries</td>
<td>Predicts the next IA-32 instruction to fetch; used only when the execution trace cache misses.</td>
</tr>
<tr>
<td>Execution trace cache</td>
<td>12K uops</td>
<td>Trace cache used for uops.</td>
</tr>
<tr>
<td>Trace cache branch-target buffer</td>
<td>2K entries</td>
<td>Predicts the next uop.</td>
</tr>
<tr>
<td>Registers for renaming</td>
<td>128 total</td>
<td>128 uops can be in execution with up to 48 loads and 32 stores.</td>
</tr>
<tr>
<td>Functional units</td>
<td>7 total: 2 simple ALU, complex ALU, load, store, FP move, FP arithmetic</td>
<td>The simple ALU units run at twice the clock rate, accepting up to two simple ALU uops every clock cycle. This allows execution of two dependent ALU operations in a single clock cycle.</td>
</tr>
<tr>
<td>L1 data cache</td>
<td>16 KB; 8-way associative; 64-byte blocks write through</td>
<td>Integer load to use latency is 4 cycles; FP load to use latency is 12 cycles; up to 8 outstanding load misses.</td>
</tr>
<tr>
<td>L2 cache</td>
<td>2 MB; 8-way associative; 128-byte blocks write back</td>
<td>256 bits to L1, providing 108 GB/sec; 18-cycle access time; 64 bits to memory capable of 6.4 GB/sec. A miss in L2 does not cause an automatic update of L1.</td>
</tr>
</tbody>
</table>
Branch Misprediction Rate

The diagram shows the branch misprediction rate for various benchmarks. The x-axis represents branch mispredictions per 1000 instructions, while the y-axis lists the benchmarks: gzip, vpr, gcc, mcf, crafty, wupwise, swim, mgrid, applu, and mesa.

- gzip has the highest misprediction rate among the listed benchmarks.
- mesa has a moderate misprediction rate.
- swim and mgrid have the lowest misprediction rates.

The data indicates that gzip and mcf have significantly higher misprediction rates compared to the others.
Percentage Mispredicted uops

- gzip
- vpr
- gcc
- mcf
- crafty
- wupwise
- swim
- mgrid
- applu
- mesa

Misspeculation percentage

© 2007 Elsevier, Inc. All rights reserved.
Data Cache Misses

![Bar charts showing L1 and L2 data cache misses per 1000 instructions for various programs.](chart.png)
CPI

- gzip: 1.59
- vpr: 2.49
- gcc: 1.49
- mcf: 5.85
- crafty: 1.53
- wupwise: 1.24
- swim: 3.25
- mgrid: 1.19
- applu: 1.73
- mesa: 1.45
Intel Pentium 4 vs AMD Opteron: CPI

The chart compares the CPI (cycles per instruction) for various benchmarks on Intel Pentium 4 and AMD Opteron processors. The benchmarks include gzip, vpr, gcc, mcf, crafty, wupwise, swim, mgrid, applu, and mesa. The CPI values for each benchmark are shown on the x-axis, with lighter gray bars representing the Pentium 4 and darker gray bars representing the AMD Opteron.
Intel Pentium 4 vs AMD Opteron: Performance

Pentium 4: 3.2 GHz
Opteron: 2.6 GHz
Intel Pentium 4 vs IBM Power5

Pentium 4: 3.2 GHz
Power5: 1.9 GHz
Recap

• Many advanced techniques on modern processors
  ★ pipelining
  ★ dynamic scheduling
  ★ branch prediction
  ★ speculation
  ★ multiple issue
  ★ branch target buffers
  ★ register renaming
  ★ ...

• Too much complexity $\Rightarrow$ Multiple cores