Multiple Issue

- Take advantage of the fact that we have multiple functional units
  - further decrease ideal CPI (<1)
- Two flavors of multiple-issue processors:
  - Superscalar
  - VLIW (Very long instruction Word)

<table>
<thead>
<tr>
<th></th>
<th>Issue structure</th>
<th>Hazard detection</th>
<th>Scheduling</th>
<th>Dist. 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>Sun UltraSPARC II/III</td>
</tr>
<tr>
<td>Superscalar (dynamic)</td>
<td>Dynamic</td>
<td>Hardware</td>
<td>Dynamic</td>
<td>Limited out-of-order exec.</td>
<td>IBM Power2</td>
</tr>
<tr>
<td>Superscalar (speculative)</td>
<td>Dynamic</td>
<td>Hardware</td>
<td>Dynamic with speculation</td>
<td>Out-of-order exec. with spec.</td>
<td>Pentium III/4 IBM RS64III, ...</td>
</tr>
<tr>
<td>VLIW</td>
<td>Static</td>
<td>Software</td>
<td>Static</td>
<td>No hazards between issue packets</td>
<td>i860, Trimedia</td>
</tr>
<tr>
<td>EPIC</td>
<td>Mostly static</td>
<td>Mostly software</td>
<td>Mostly static</td>
<td>Dependencies marked by compiler</td>
<td>Itanium</td>
</tr>
</tbody>
</table>
Superscalar architectures

- Issue a varying number of instructions per cycle
  - Statically scheduled using compiler techniques
  - Dynamically scheduled (e.g. using Tomasulo’s algorithm)
- Why a varying number of instructions?
  - Statically scheduled -> no out-of-order execution
  - Can check for hazards at issue time
  - Issue logic will issue instructions which cause a hazard

Some details

- Issue unit receives between one and n instructions from the instruction fetch unit (n being typically 4 or 8)
  → issue packet
- Instruction fetch unit examines each instruction in the issue packet in order
- If an instruction causes a structural hazard or a data hazard, it will not be issued
- Since the checking for structural and data hazards are complex operations, the instruction fetch unit is implemented as a pipeline, e.g
  - First stage checks for hazards within the issue packet
  - Second stage checks for hazards with already issued instructions
Dynamically scheduled superscalar MIPS

- Extend Tomasulo’s algorithm to handle multiple issues per cycle
  - Must issue instructions to reservation stations in order to maintain program semantics
- Note: we do not handle the details on how multiple issue works for Tomasulo’s algorithms

Limitations of ILP in hardware

- Need to be able to look arbitrarily far ahead to find instructions to issue
- Predict all branches perfectly
- Rename all registers to avoid WAR and WAW hazards
- Avoid data dependencies amongst instructions in an issue packet
- Handling memory dependencies among issuing instructions
- Provide enough replicated functional units to issue all instruction which are ready
Limitations of ILP in hardware (II)

- Finite number of Registers
  - For a single instruction stream
  - When considering thread-level parallelism
- Imperfect data dependence analysis depending on whether the data is located on the stack, the heap or is statically declared.

VLIW processors

- Issue a fixed number of instructions
  - As one large instruction or
  - As a fixed instruction packet
- Parallelism among instructions has to be explicitly indicated by the instructions
  - Statically scheduled by compiler
An example

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

Loop: L.D  F0, 0(R1)
      ADD.D F4, F0, F2
      S.D  F4, 0(R1)
      DADDUI R1, R1, #-8
      BNE  R1, R2, Loop

Assumptions

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

• 1 cycle branch delay
An example (III)

Loop:  L.D   F0, 0(R1)  
      s
      ADD.D  F4, F0, F2  
      s
      s
      S.D   F4, 0(R1)  
      DADDUI R1, R1, #-8  
      s
      BNE   R1, R2, Loop
      s

1  wait for F0 to propagate
2  wait for ADD to complete
3  wait for ADD to complete
4  wait for ADD to complete
5  wait for R1 to propagate
6  branch delay slot

Rescheduling the code

Loop:  L.D   F0, 0(R1)  
      DADDUI R1, R1, #-8  
      ADD.D  F4, F0, F2  
      s
      BNE   R1, R2, Loop
      S.D   F4, 8(R1)  

1
2
3
4
5
6 ←

Delayed branch slot

- Each loop iteration consists of 3 instructions of actual work (load, add, store) and 3 cycles loop overhead
Loop unrolling

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 (II)

- Eliminates three branches and three decrements
  - Reduces loop overhead
- The previous code sequence still contains many stalls, since many operations are still dependent on each other
- Requires a large number of registers
- Increase of the code size
- For general application of loop unrolling:
  - Two consecutive loops
    - First executes \((n \mod k)\) times, not unrolled
      - \(n\): number of loop iterations
      - \(k\): unroll depth of the loop
    - Second executes \((n/k)\) times, unrolled
Scheduled version of the unrolled loop

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

Scheduled version of the unrolled loop (II)

- Non trivial transformations in the previous code
  - Adjust the offsets of S.D instructions
  - Determine that it is legal to move the S.D after DADDUI and BNE
  - Determine that loads and stores of difference iterations can be interchanged
Effects of register renaming

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

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

No dependencies between the loop bodies of two iterations!

Limitations of loop unrolling

- Code size limitations
- Lower limit on the code amount which can be saved by loop unrolling
  - Consider that you need to prepend another (unrolled) loop of length \([ n \mod k]\)
- \textit{Register pressure}: shortfall of registers generated by aggressive unrolling
Static Multiple Issue: VLIW

- Hardware checking for dependencies in issue packets may be expensive and complex
- Compiler can examine instructions and decide which ones can be scheduled in parallel - group instructions into instruction packets - VLIW
- Hardware can then be simplified
- Processor has multiple functional units and each field of the VLIW is assigned to one unit
  - For example, VLIW could contain 5 fields and one has to contain ALU instruction or branch, two have to contain FP instructions and two have to be memory references
Example

- Assume VLIW contains 5 fields: ALU instruction or branch, two FP instructions and two memory references
- Ignore branch delay slot

Loop:
- L.D F0,0(R1)  
- stall, wait for F0 value to propagate
- ADD.D F4, F0, F2  
- stall, wait for FP add to be completed
- stall, wait for FP add to be completed
- S.D F4, 0(R1)  
- Memory reference
- DADDUI R1, R1, #8  
- stall, wait for R1 value to propagate
- BNE R1, R2, Loop  
- ALU instruction

Example

- Unroll seven times and rearrange

Loop:
- L.D F0,0(R1)  1
- L.D F0,0(R1)
- L.D F6,0(R1)
- L.D F10,-16(R1)
- L.D F14,-24(R1)
- L.D F18,-32(R1)
- L.D F22,-40(R1)
- L.D F26,-48(R1)
- ADD.D F4, F0, F2  3
- ADD.D F4, F0, F2
- ADD.D F8, F6, F2
- ADD.D F12, F10, F2
- ADD.D F16, F14, F2
- ADD.D F20, F18, F2
- ADD.D F24, F22, F2
- ADD.D F28, F26, F2

<table>
<thead>
<tr>
<th>ALU/branch</th>
<th>FP</th>
<th>FP</th>
<th>mem</th>
<th>mem</th>
</tr>
</thead>
</table>
### Example

**Loop:**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>L.D F0, 0(R1)</td>
<td>S.D F4, 0(R1)</td>
</tr>
<tr>
<td>L.D F6, -8(R1)</td>
<td>S.D F8, -8(R1)</td>
</tr>
<tr>
<td>L.D F10, -16(R1)</td>
<td>S.D F12, -16(R1)</td>
</tr>
<tr>
<td>L.D F10, 32(R1)</td>
<td>S.D F16, -24(R1)</td>
</tr>
<tr>
<td>L.D F20, 32(R1)</td>
<td>S.D F20, -32(R1)</td>
</tr>
<tr>
<td>L.D F20, -56(R1)</td>
<td>DADDUI R1, R1, -56</td>
</tr>
<tr>
<td>ADD.D F4, F0, F2</td>
<td>S.D F24, 16(R1)</td>
</tr>
<tr>
<td>ADD.D F8, F6, F2</td>
<td>BNE R1, R2, Loop</td>
</tr>
<tr>
<td>ADD.D F12, F10, F2</td>
<td>S.D F28, 8(R1)</td>
</tr>
<tr>
<td>ADD.D F16, F14, F2</td>
<td></td>
</tr>
<tr>
<td>ADD.D F20, F18, F2</td>
<td></td>
</tr>
<tr>
<td>ADD.D F24, F22, F2</td>
<td></td>
</tr>
<tr>
<td>ADD.D F28, F26, F2</td>
<td></td>
</tr>
</tbody>
</table>

**Example**

**Loop:**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>L.D F0, 0(R1)</td>
<td>S.D F4, 0(R1)</td>
</tr>
<tr>
<td>L.D F6, -8(R1)</td>
<td>S.D F8, -8(R1)</td>
</tr>
<tr>
<td>L.D F10, -16(R1)</td>
<td>S.D F12, -16(R1)</td>
</tr>
<tr>
<td>L.D F10, 32(R1)</td>
<td>S.D F16, -24(R1)</td>
</tr>
<tr>
<td>L.D F20, 32(R1)</td>
<td>S.D F20, -32(R1)</td>
</tr>
<tr>
<td>L.D F22, -40(R1)</td>
<td>DADDUI R1, R1, -56</td>
</tr>
<tr>
<td>ADD.D F4, F0, F2</td>
<td>S.D F24, 16(R1)</td>
</tr>
<tr>
<td>ADD.D F8, F6, F2</td>
<td>BNE R1, R2, Loop</td>
</tr>
<tr>
<td>ADD.D F12, F10, F2</td>
<td>S.D F28, 8(R1)</td>
</tr>
<tr>
<td>ADD.D F16, F14, F2</td>
<td></td>
</tr>
<tr>
<td>ADD.D F20, F18, F2</td>
<td></td>
</tr>
<tr>
<td>ADD.D F24, F22, F2</td>
<td></td>
</tr>
<tr>
<td>ADD.D F28, F26, F2</td>
<td></td>
</tr>
</tbody>
</table>

**ALU/branch**

<table>
<thead>
<tr>
<th>FP</th>
<th>FP</th>
<th>mem</th>
<th>mem</th>
</tr>
</thead>
</table>

---

*Slide based on a lecture by Jelena Mirkovic, University of Delaware*

*http://www.cs.umd.edu/~sunshine/courses/F04/CIS662/class16.pdf*
Example

Loop:

L.D F0, 0(R1)
L.D F6, -8(R1)
L.D F10, -16(R1)
L.D F14, -24(R1)
L.D F18, -32(R1)
L.D F22, -40(R1)
L.D F26, -48(R1)
ADD.D F4, F0, F2
ADD.D F8, F6, F2
ADD.D F12, F10, F2
ADD.D F16, F14, F2
ADD.D F20, F18, F2
ADD.D F24, F22, F2
ADD.D F28, F26, F2

S.D F4, 0(R1) ➞ 6
S.D F8, -8(R1) ➞ 7
S.D F12, -16(R1)
S.D F16, -24(R1)
S.D F20, -32(R1)
DADDUI R1, R1, #-56
S.D F24, 16(R1)
BNE R1, R2, Loop
S.D F28, 8(R1)

Example

Loop:

L.D F0, 0(R1)
L.D F6, -8(R1)
L.D F10, -16(R1)
L.D F14, -24(R1)
L.D F18, -32(R1)
L.D F22, -40(R1)
L.D F26, -48(R1)
ADD.D F4, F0, F2
ADD.D F8, F6, F2
ADD.D F12, F10, F2
ADD.D F16, F14, F2
ADD.D F20, F18, F2
ADD.D F24, F22, F2
ADD.D F28, F26, F2

S.D F4, 0(R1) ➞ 6
S.D F8, -8(R1) ➞ 7
S.D F12, -16(R1)
S.D F16, -24(R1)
S.D F20, -32(R1)
DADDUI R1, R1, #-56
S.D F24, 16(R1)
BNE R1, R2, Loop
S.D F28, 8(R1)
Example

Loop:

<table>
<thead>
<tr>
<th>ALU/branch</th>
<th>FP</th>
<th>FP</th>
<th>mem</th>
<th>mem</th>
</tr>
</thead>
<tbody>
<tr>
<td>L.D F0,0(R1)</td>
<td>S.D F4, 0(R1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L.D F6, -8(R1)</td>
<td>S.D F8, -8(R1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L.D F10, -16(R1)</td>
<td>S.D F12, -16(R1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L.D F14, -24(R1)</td>
<td>S.D F16, -24(R1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S.D F20, 24(R1)</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DADDUI R1, R1, # -56</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S.D F24, 16(R1)</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BNE R1, R2, Loop 9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S.D F28, 8(R1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L.D F22, -40(R1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L.D F26, -48(R1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD.D F4, F0, F2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD.D F8, F6, F2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD.D F12, F10, F2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD.D F16, F14, F2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD.D F20, F18, F2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD.D F24, F22, F2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD.D F28, F26, F2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Overall 9 cycles for 7 iterations 1.29 per iteration
But VLIW was always half-full