VLIW Processors

Loop:
- LW  r1, 0(r2)
- MUL r3, r1, r4
- SW  r3, 0(r5)
- ADDW r2, r1, r3
- ADDW r3, r5, r4
- ADDW r7, r7, -1
- BHT2 r7, 1 loop

Architecture Exposes:
- Number of Functional Units
- Functional Unit Latency
- Branch Resolution Latency

Compiler must schedule VLIW instructions given these constraints.

7 cycles / 1 cycle, 64 cycles to process 64 FLMs
448 total cycles.
VLW PIPELINE DIAGRAM

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4

ADDW r2, r2, r4
ADDW r1, 0(r2)
ADDW r2, r2, r4
ADDW r7, r7, -1

MUL r3, r1, r4
ADDW r7, r7, -1

F D
F D x0 x1 x2 x3
F D
F D
F D
F D
F D
F D
F D
F D
F D
F D
F D
F D
F D
F D
F D
F D
F D
F D

ADDW r3, r5, r4
SW r3, 0(r3)
**LOOP UNROLLING**

Unroll loop to amortize loop overhead, reduce ans. # of iterations

<table>
<thead>
<tr>
<th>Y-PIPE</th>
<th>X-PIPE</th>
<th>L-PIPE</th>
<th>S-PIPE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

10 cycles / 1 ita, 16 ita to process 64 entries
160 total cycles

**UNROLL BY FACTOR OF EIGHT?**

Unroll by factor of eight?

<table>
<thead>
<tr>
<th>Y-PIPE</th>
<th>X-PIPE</th>
<th>L-PIPE</th>
<th>S-PIPE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

14 cycles / 1 ita, 8 ita to process 64 entries
112 total cycles
SOFTWARE PIPELINING

Take instructions from multiple iterations to create new iterations that can run at higher peak throughput.

**prototipe**

```
LW r1, 0(r2)
MUL r3, r1, r4
```

**loop**

```
SW r3, 0(r5)  MUL r3, r1, r4  LW r1, 0(r2)  
SW r2, 0(r5)  MUL r2, r1, r4  LW r1, 0(r2)  
SW r3, 0(r5)  MUL r3, r1, r4  SW r2, 0(r5)  
```

**epilogue**

```
SW r3, 0(r5)  MUL r3, r1, r4  SW r2, 0(r5)  
```

**Original**

```
loop:
LW r1, 0(r2)
MUL r3, r1, r4
ADDU r2, r1, r4
SW r3, 0(r5)
ADDU r3, r5, r4
ADDU r7, r2, -1
J+2 r7, loop
```

```
main loop:
SW r3, 0(r5)
MUL r3, r1, r4
ADDU r2, r1, r4
ADDU r5, r5, r4
ADDU r7, r2, -1
BSTL r7, loop
```

```
epilogue:
SW r3, 0(r5)
ADDU r5, r5, r4
MUL r3, r1, r4
SW r3, 0(r5)
```

**Pipeline Startup**

```
| itr 0 | LW X SW | Achieve full throughput |
| itr 1 | LW X SW | START 2nditr befor first iteration is finished |
| itr 2 | LW X SW |                             |
| itr 3 | LW X SW | LET PIPELINE DRAIN |
```

---

National Brand

---

"C++"
Scanning SW PIPELINE loop on XW PROCESSOR

<table>
<thead>
<tr>
<th>Y-PIPE</th>
<th>X-PIPE</th>
<th>L-PIPE</th>
<th>S-PIPE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Add w r2</td>
<td>lw r1</td>
<td></td>
</tr>
<tr>
<td>MUL r3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MUL r3</td>
<td>Add w r7</td>
<td>lw r1</td>
<td>sw r3</td>
</tr>
<tr>
<td></td>
<td>Add w r5</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Add w r2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Prologue = 6 cycles
Main loop = 4 cycles / 1 iteration, 62 iterations = 248 cycles
Epilogue = 5 cycles
Total = 259 cycles

SW PIPELINING produces more compact code, uses less registers, and can deal with irregularly sized input arrays better than loop unrolling.

SW PIPELINING does not require loop overhead (essentially SAME # of iterations)

SW PIPELINING allows code to quickly get up to peak throughput with an epilogue + prologue per loop.
Loop Unrolling + SW Pipelining

Use loop unrolling to amortize loop overhead.
Use SW pipelining to reduce reg pressure, code size, + flexibility

First unroll loop

```
loop:
    lw    r1, 0(r2)
    lw    r2, 4(r2)
    lw    r3, 8(r2)
    lw    r4, 12(r2)
    mul   r8, r1, r4
    mul   r9, r3, r4
    mul   r10, r3, r4
    mul   r11, r6, r4
    sw    r8, 0(r5)
    sw    r9, 4(r5)
    sw    r10, 8(r5)
    sw    r11, 12(r5)
    addw  r2, r2, 16
    addw  r5, r5, 16
    addw  r7, r7, -4
    bne   r7, loop
```
<table>
<thead>
<tr>
<th>Y-PIPE</th>
<th>X-PIPE</th>
<th>L-PIPE</th>
<th>S-PIPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>mul r8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mul r9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mul r10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mul r11</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Main Loop**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>mul r8</td>
<td>addw r7</td>
<td></td>
<td>sra r8,0(r1)</td>
</tr>
<tr>
<td>mul r9</td>
<td>bshz r7</td>
<td></td>
<td>sra r9,4(r2)</td>
</tr>
<tr>
<td>mul r10</td>
<td>addw r5,16</td>
<td></td>
<td>sra r10,3(r3)</td>
</tr>
<tr>
<td>mul r11</td>
<td>addw r2,16</td>
<td></td>
<td>sra r11,4(r5)</td>
</tr>
</tbody>
</table>

**Epilogue**

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Might still need fix up code in case input not multiple of 4

**Prologue = 9 cycles**

**Main Loop = 4 cycles / iter, 11 iter = 56 cycles**

**Epilogue = 8 cycles**

**Total = 73 cycles**!