Introduction to Superscalar Execution

Tom Kelliher, CS 240
Apr. 24, 2002

1 Administrivia

Announcements

No class Friday — GIG.

Assignment

Read 7.1–2.

Problems due 5/3, beginning of class: 7.7, 7.15–17, 7.32.

From Last Time

Data hazards in pipelines.

Outline

1. Introduction to superscalar pipelining

2. Data dependencies.

3. Out of order execution.
4. Multiple Instruction Issue

5. In-order execute pipeline and instruction scheduling.

Coming Up

Caches.

2 Introduction to Superscalar Pipelining

1. Historical Progression of IPC: $< 1$, $= 1$, $> 1$. The *entire* pipeline must be widened.

   Challenges: small register files, multiple-branch predictions, multiple line fetches from caches.

2. Range of parallelism: coarse- to fine-grained.

3. Superscalar techniques address ILP. *Let’s parallelize a sequential binary.*

4. What’s the upper bound on IPC? It depends.

   Text processing: low, mostly.

   Image processing, multimedia: high.

   Median operation on an image example:

   ```
   medianImage(image dest, image src)
   {
       for each pixel, p, in src
       p in dest = medianPixel(p in src);
   }

   medianPixel(pixel p)
   {
       find the <= 8 neighboring pixels of p;
       compute and return the median value;
   }
   ```
Challenges: exposing potential ILP to the compiler.

Example. Parallelize the following:

```c
sum = 0;
for (i = 0; i < last; ++i)
    sum += array[i];
```

5. Compiler techniques: loop unrolling, invariant code migration, strength reduction, etc.

### 2.1 Types of Data Dependencies

1. RAR. Not a problem at all.

2. RAW. A “true” dependency.

3. WAR. A “false” dependency.

4. WAW. Another “false” dependency.

Consider the code segment:

```c
r1 = r2 + r3
r4 = r1 + r5
r1 = r6 + r7
r8 = r1 + r4
```

ISA registers vs. physical registers. *Register renaming?*

Rename the previous example where the *Register Alias Table* (RAT) is initially:

```plaintext
r1 -> p12  r2 -> p6  r3 -> p9  r4 -> p15
r5 -> p1  r6 -> p10  r7 -> p8  r8 -> p14
```


Which dependencies were removed? Which remain?
2.2 Out of Order execution

1. What is it?

2. In-order completion.

3. How is it done?

2.3 Multiple Instruction Issue

1. In-order execution case.
   Structural hazard stalls.

2. Out of order execution case.
   Only stall if no free list entries.

3 In-Order Execute Pipeline

1. This is “simple” case.
   Consider two-instruction issue.
   Must consider:
   (a) Structural hazards.
   (b) Data hazards.
   (c) Control hazards.
   How to handle?

2. Instruction pairs must be aligned.
   (a) First instruction: R-format or branch.
(b) Second instruction: Memory access. (Add an address adder.)

3. If first instruction stalls, both stall.

4. Second instruction may stall due to data, control dependencies.

The pipeline:

What are the inter- and intra-instruction pair dependencies?

What are our options in increasing the functionality of that second ALU? (reg = reg op immed instrs, reg = reg op reg instrs) Additional dependencies?

3.1 Instruction Scheduling Example

Consider the code segment:

```c
sum = 0;
for (i = 0; i < last; ++i)
    sum += array[i];
```
Which might compile to:

```
  top:  lw $t0, 0($s1)
        addu $s2, $s2, $t0
        addi $s1, $s1, -4
        bne $s1, $0, top
```

How will the code be scheduled?

```
 ──     lw
 addi ───
 addu ──
 bne ──
```

The `addi` could be raised, but what’s it gain?

Suppose we unroll once:

```
  top:  lw $t0, 0($s1)
        addu $s2, $s2, $t0
        lw $t0, -4($s1)
        addu $s2, $s2, $t0
        addi $s1, $s1, -8
        bne $s1, $0, top
```

Where are the stalls? How can we introduce temp variables to eliminate some stalls?

How will the improved code schedule?

```
 addi    lw
 ───     lw
 addu ──
 addu ──
 bne ──
```

Is this an improvement?

By the way, what happens with the `lw` offsets?
Unroll twice more:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>addi</td>
<td>lw</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>addu</td>
<td>lw</td>
</tr>
<tr>
<td>addu</td>
<td>lw</td>
</tr>
<tr>
<td>addu</td>
<td></td>
</tr>
<tr>
<td>addu</td>
<td></td>
</tr>
<tr>
<td>bne</td>
<td></td>
</tr>
</tbody>
</table>

Is this an improvement?

When do you stop?