Today: Beyond the 5-stage pipeline

- Amdahl’s Law

- Taxonomy of advanced processing.

- **Superpipelining**: Increasing the number of pipeline stages.

- **Superscalar**: Issuing several instructions in a single cycle.

- Hardware support for **Virtual Memory** and **Virtual Machines**.
Invented the “one ISA, many implementations” business model.
Amdahl’s Law (of Diminishing Returns)

If enhancement “E” makes multiply infinitely fast, but other instructions are unchanged, what is the maximum speedup “S”?

Where program spends its time

\[
S = \frac{1}{\text{(post-enhancement %)} / 100} = \frac{1}{48/100} = 2.08
\]

Attributed to Gene Amdahl -- “Amdahl’s Law”

What is the lesson of Amdahl’s Law?
Must enhance computers in a balanced way!
Amdahl’s Law in Action

The program spends 30% of its time running code that can not be recoded to run in parallel.

\[ S = \frac{1}{(30\% + \frac{70\%}{N}) / 100\%} \]

<table>
<thead>
<tr>
<th>CPUs</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>$\infty$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speedup</td>
<td>1.54</td>
<td>1.85</td>
<td>2.1</td>
<td>2.3</td>
<td>3.3</td>
</tr>
</tbody>
</table>

S(\infty)

2 3 # CPUs
Real-world 2006: 2 CPUs vs 4 CPUs

20 in iMac
Core Duo 2, 2.16 GHz
$1500

Mac Pro
2 Dual-Core Xeons, 2.66 GHz
$3200 w/ 20 inch display.
Real-world 2006: 2 CPUs vs 4 CPUs

<table>
<thead>
<tr>
<th></th>
<th>Speedmark 4.5</th>
<th>Adobe Photoshop CS2</th>
<th>Cinema 4D XL 9.5.21</th>
<th>Compressor 2.1</th>
<th>iMovie 6.0.1</th>
<th>iTunes 6.0.4</th>
<th>Unreal Tournament 2004</th>
<th>Finder</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OVERALL SCORE</strong></td>
<td>245</td>
<td>1:55</td>
<td>1:01</td>
<td>2:37</td>
<td>0:52</td>
<td>1:03</td>
<td>74.4</td>
<td>2:22</td>
</tr>
<tr>
<td><strong>20-inch iMac Core 2 Duo/2.16GHz</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Mac Pro 2.66GHz (Standard)</strong></td>
<td>299</td>
<td>1:25</td>
<td>0:28</td>
<td>1:47</td>
<td>0:38</td>
<td>0:48</td>
<td>91.3</td>
<td>2:01</td>
</tr>
</tbody>
</table>

Caveat: Mac Pro CPUs are server-class and have architectural advantages (better I/O, ECC DRAM, etc).

Source: Macworld

Simple video task: easier to parallelize.

ZIPing a file: very difficult to parallelize.
Taxonomy
5 Stage Pipeline: A point of departure

Seconds Program = Instructions Program / Cycles Instruction \times Seconds Cycle

At best, the 5-stage pipeline executes one instruction per clock, with a clock period determined by the slowest stage.

Processor has no “multi-cycle” instructions (ex: multiply with an accumulate register)

Filling all delay slots (branch, load)

Perfect caching
Superpipelining: Add more stages

Goal: Reduce critical path by adding more pipeline stages.

Example: 8-stage ARM XScale: extra IF, ID, data cache stages.

Difficulties: Added penalties for load delays and branch misses.

Ultimate Limiter: As logic delay goes to 0, FF clk-to-Q and setup.

Also, power!
**Superscalar:** Multiple issues per cycle

Goal: Improve CPI by issuing several instructions per cycle.

Example: CPU with floating point ALUs: Issue 1 FP + 1 Integer instruction per cycle.

Difficulties: Load and branch delays affect more instructions.

Ultimate Limiter: Programs may be a poor match to issue rules.
Out of Order: Going around stalls

Goal: Issue instructions out of program order

Example:

<table>
<thead>
<tr>
<th>Second Program</th>
<th>Instruction</th>
<th>Cycle</th>
<th>Second Cycle</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>F2,</td>
<td>34</td>
<td>1</td>
</tr>
<tr>
<td>LD</td>
<td>F4,</td>
<td>45</td>
<td>long</td>
</tr>
<tr>
<td>MULTD</td>
<td>F6, F4, F2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>ADDD</td>
<td>F8, F2, F2</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Difficulties: Bookkeeping is highly complex. A poor fit for lockstep instruction scheduling.

Ultimate Limiter: The amount of instruction level parallelism present in an application.
Dynamic Scheduling: End lockstep

Goal: Enable out-of-order by breaking pipeline in two: Fetch and Execution.

Example: IBM Power 5:

Limiters: Design complexity, instruction level parallelism.
Throughput and multiple threads

Goal: Use multiple CPUs (real and virtual) to improve (1) throughput of machines that run many programs (2) execution time of multi-threaded programs.

Example: Sun Niagara (8 SPARCs on one chip).

Difficulties: Gaining full advantage requires rewriting applications, OS, libraries.

Ultimate limiter: Amdahl’s law, memory system performance.
Superpipelining
Graphically Representing MIPS Pipeline

Can help with answering... is the ALU doing during cycle 4?
is there a hazard, why does it occur, and how can it be fixed?

Note: Some stages now overlap, some instructions take extra stages.

IF now takes 2 stages (pipelined I-cache)

ID and RF each get a stage.

ALU split over 3 stages

MEM takes 2 stages (pipelined D-cache)
Superpipelining techniques ...

- Split ALU and decode logic over several pipeline stages.

- **Pipeline memory**: Use more banks of smaller arrays, add pipeline stages between decoders, muxes.

- Remove “rarely-used” forwarding networks that are on critical path. This creates stalls, affects CPI.

- Pipeline the wires of frequently used forwarding networks.

*Also: Clocking tricks (example: negedge register file in COD3e pipeline)*
Recall: IBM Power Timing Closure

Recall: Pipelining SRAM memories ...

Architects specify number of rows and columns. Word and bit lines slow down as array grows larger!

Parallel Data I/O Lines

How could we pipeline this memory?

Add muxes to select subset of bits
ALU: Pipelining Unsigned Multiply

multiplicand: 1101 (13)
multiplier: * 1011 (11)

Partial products:

- 1101
- 1101
- 0000
- 1101

10001111 (143)

m bits x n bits = m+n bit product

Binary makes it easy:

- 0 => place 0 (0 x multiplicand)
- 1 => place a copy (1 x multiplicand)
Building Block: Full-Adder Variant

1-bit signals: x, y, z, s, Cin, Cout

If z = 1, \{Cout, s\} <= x + y + Cin
If z = 0, \{Cout, s\} <= y + Cin

z: one bit of multiplier
x: one bit of multiplicand
y: one bit of the “running sum”
Put it together: Array computes $P = A \times B$

To pipeline array:

Place registers between adder stages.

Use registers to delay selected $A$ and $B$ bits.

As drawn, combinational (slow!).
Virtex-5: DSP slice multiplier pipelining

*These signals are dedicated routing paths internal to the DSP48E column. They are not accessible via fabric routing resources.
Virtex-5: DSP slice multiplier pipelining
Pre Virtex-5: Pipelining 18x18 multipliers

Note: All inputs to the multiplier or adder less than required are sign extended.

Indicates optional pipeline stages.
Q. Could adding pipeline stages hurt the CPI for an application?
A. Yes, due to these problems:

<table>
<thead>
<tr>
<th>CPI Problem</th>
<th>Possible Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Taken branches cause longer stalls</td>
<td>Branch prediction, loop unrolling</td>
</tr>
<tr>
<td>Cache misses take more clock cycles</td>
<td>Larger caches, add prefetch opcodes to ISA</td>
</tr>
</tbody>
</table>
Recall: Control hazards ...

We avoid stalling by (1) adding a branch delay slot, and (2) adding comparator to ID stage.

If we add more early stages, we must stall.

Sample Program (ISA w/o branch delay slot)

**I1:** BEQ R4, R3, 25  
**I2:** AND R6, R5, R4  
**I3:** SUB R1, R9, R8

Time:

<table>
<thead>
<tr>
<th>Inst</th>
<th>t1</th>
<th>t2</th>
<th>t3</th>
<th>t4</th>
<th>t5</th>
<th>t6</th>
<th>t7</th>
<th>t8</th>
</tr>
</thead>
<tbody>
<tr>
<td>I1</td>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I2</td>
<td>IF</td>
<td>ID</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

EX stage computes if branch is taken.

If branch is taken, these instructions MUST NOT complete!
Solution: Branch prediction...

We update the PC based on the outputs of the branch predictor. If it is perfect, pipe stays full!

Dynamic Predictors: a cache of branch history

Time: t1 t2 t3 t4 t5 t6 t7 t8
Inst
I1: IF ID EX MEM WB
I2: IF ID
I3: IF
I4: 
I5: 
I6: 

EX stage computes if branch is taken

If we predicted incorrectly, these instructions MUST NOT complete!
Branch predictors cache branch history

Address of BNEZ instruction
0b0110[...01001000

Branch Target Buffer (BTB)
28-bit address tag
0b0110[...0100

Branch History Table (BHT)
2 bits

BNEZ R1 Loop

Update BHT/ BTB
for next time, once true behavior known

"Taken" or "Not Taken"

"Taken" Address

Hit

Must check prediction, kill instruction if needed.

80-90% accurate
Simple ("2-bit") Branch History Table Entry

Prediction for next branch. (1 = take, 0 = not take)
Initialize to 0.

Was last prediction correct? (1 = yes, 0 = no)
Initialize to 1.

Flip bit if prediction is not correct and "last predict correct" bit is 0.

Set to 1 if prediction bit was correct.
Set to 0 if prediction bit was incorrect.
Set to 1 if prediction bit flips.

After we “check” prediction ...

We do not change the prediction the first time it is incorrect. Why?

This branch taken 10 times, then not taken once (end of loop). The next time we enter the loop, we would like to predict “take” the first time through.

ADDI R4, R0, 11
loop: SUBI R4, R4, -1
      BNE R4, R0, loop
Spatial enhancements: many BHTs ...

0b0110[...01001000 BNEZ R1 Loop

Branch History Tables

(BHT00) (BHT01) (BHT10) (BHT11)

Detected patterns in:
if (x < 12)
if (x < 6)
code.

Yeh and Patt, 1992.

95% accurate
Hardware limits to superpipelining?

FO4 Delays

Historical limit: about 12

FO4: How many fanout-of-4 inverter delays in the clock period.

Thanks to Francois Labonte, Stanford
Superscalar

Basic Idea: Improve CPI by issuing several instructions per cycle.
Recall VLIW: Super-sized Instructions

Example: All instructions are 64-bit. Each instruction consists of two 32-bit MIPS instructions, that execute in parallel.

<table>
<thead>
<tr>
<th>Syntax: ADD $8 $9 $10</th>
<th>Semantics: $8 = $9 + $10</th>
</tr>
</thead>
<tbody>
<tr>
<td>opcode</td>
<td>rs</td>
</tr>
<tr>
<td>opcode</td>
<td>rs</td>
</tr>
</tbody>
</table>

Syntax: ADD $7 $8 $9  Semantics: $7 = $8 + $9

A 64-bit VLIW instruction

But what if we can’t change ISA execution semantics?
Sustaining Dual Instr Issues (no forwarding)

ADD R8, R0, R0
ADD R11, R0, R0
ADD R27, R26, R25
ADD R30, R29, R28
ADD R21, R20, R19
ADD R24, R23, R22
ADD R15, R14, R13
ADD R18, R17, R16
ADD R9, R8, R7
ADD R12, R11, R10

It's rarely this good ...

CS 194-6 L9: Advanced Processors I
We add 12 forwarding buses (not shown). (6 to each ID from stages of both pipes).

Worst-Case Instruction Issue

ADD R8, R0, R0
ADD R9, R8, R0
ADD R10, R9, R0
ADD R11, R10, R0

Dependencies force “serialization”
Example: Superscalar MIPS. Fetches 2 instructions at a time. If first integer and second floating point, issue in same cycle

**Integer instruction**  
**FP instruction**

- LD F0,0(R1)  
- LD F6,-8(R1)  
- LD F10,-16(R1)  
  ADDD F4,F0,F2  
- LD F14,-24(R1)  
  ADDD F8,F6,F2  
- LD F18,-32(R1)  
  ADDD F12,F10,F2  
- SD 0(R1),F4  
  ADDD F16,F14,F2  
- SD -8(R1),F8  
  ADDD F20,F18,F2  
- SD -16(R1),F12  
- SD -24(R1),F16

Why is the control for this CPU not so hard to do?
Superscalar: Visualizing the pipeline

Three instructions potentially affected by a single cycle of load delay, as FP register loads done in the “integer” pipeline.

<table>
<thead>
<tr>
<th>Type</th>
<th>Pipe</th>
<th>Stages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int. instruction</td>
<td>IF ID EX</td>
<td>MEM WB</td>
</tr>
<tr>
<td>FP instruction</td>
<td>IF ID EX</td>
<td>MEM WB</td>
</tr>
<tr>
<td>Int. instruction</td>
<td>IF ID EX</td>
<td>MEM WB</td>
</tr>
<tr>
<td>FP instruction</td>
<td>IF ID EX</td>
<td>MEM WB</td>
</tr>
<tr>
<td>Int. instruction</td>
<td>IF ID EX</td>
<td>MEM WB</td>
</tr>
<tr>
<td>FP instruction</td>
<td>IF ID EX</td>
<td>MEM WB</td>
</tr>
</tbody>
</table>
Limitations of “lockstep” superscalar

- Gets 0.5 CPI only for a 50/50 float/int mix with no hazards. For games/media, this may be OK.
- Extending scheme to speed up general apps (Microsoft Office, ...) is complicated.
- If one accepts building a complicated machine, there are better ways to do it.

Next Monday: Dynamic Scheduling
Virtual Memory
The Limits of Physical Addressing

“Physical addresses” of memory locations

CPU
A0-A31
D0-D31

Memory
A0-A31
D0-D31

Where we are ...

Data

All programs share one address space:
The **physical** address space

Machine language programs must be aware of the machine organization

No way to prevent a program from accessing **any** machine resource
Apple II: A physically-addressed machine

Apple II (1977)
CPU: 1000 ns
DRAM: 400 ns

<table>
<thead>
<tr>
<th>RAM Complement</th>
<th>Apple II System</th>
</tr>
</thead>
<tbody>
<tr>
<td>4K</td>
<td>$1298.00</td>
</tr>
<tr>
<td>48K</td>
<td>$2638.00</td>
</tr>
</tbody>
</table>
Apple II: A physically addressed machine

Apple ][ (1977)

<table>
<thead>
<tr>
<th>RAM Complement</th>
<th>Apple II System</th>
</tr>
</thead>
<tbody>
<tr>
<td>4K</td>
<td>$1,268.00</td>
</tr>
<tr>
<td>48K</td>
<td>$2,638.00</td>
</tr>
</tbody>
</table>

### RAM Organization and Usage

<table>
<thead>
<tr>
<th>Page #</th>
<th>Used for:</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>System Programs</td>
</tr>
<tr>
<td>1</td>
<td>System Stack</td>
</tr>
<tr>
<td>2</td>
<td>Input Buffer</td>
</tr>
<tr>
<td>3</td>
<td>Monitor Vector Locations</td>
</tr>
<tr>
<td>4</td>
<td>Tex/Lo-Res Graphics</td>
</tr>
<tr>
<td>5</td>
<td>Primary Page Storage</td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Text/Lo-Res Graphics</td>
</tr>
<tr>
<td>9</td>
<td>Secondary Page Storage</td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Hi-Res Graphics</td>
</tr>
<tr>
<td>31</td>
<td>Primary Page Storage</td>
</tr>
<tr>
<td>32</td>
<td>FREE RAM</td>
</tr>
<tr>
<td>63</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>Hi-Res Graphics</td>
</tr>
<tr>
<td>95</td>
<td>Secondary Page Storage</td>
</tr>
<tr>
<td>96</td>
<td>FREE RAM</td>
</tr>
<tr>
<td>191</td>
<td>I/O and softswitches</td>
</tr>
<tr>
<td>192</td>
<td>I/O shared ROM space</td>
</tr>
<tr>
<td>193</td>
<td></td>
</tr>
<tr>
<td>199</td>
<td></td>
</tr>
</tbody>
</table>
The Limits of Physical Addressing

"Physical addresses" of memory locations

All programs share one address space: The physical address space

Machine language programs must be aware of the machine organization

No way to prevent a program from accessing any machine resource
Solution: Add a Layer of Indirection

```
<table>
<thead>
<tr>
<th></th>
<th>“Virtual Addresses”</th>
<th>“Physical Addresses”</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>A0-A31</td>
<td></td>
</tr>
<tr>
<td>D0-D31</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Address</td>
<td>Virtual</td>
<td>Physical</td>
</tr>
<tr>
<td>Translation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memory</td>
<td>A0-A31</td>
<td></td>
</tr>
<tr>
<td></td>
<td>D0-D31</td>
<td></td>
</tr>
</tbody>
</table>
```

User programs run in an standardized virtual address space

Address Translation hardware managed by the operating system (OS) maps virtual address to physical memory

Hardware supports “modern” OS features: Protection, Translation, Sharing
MIPS R4000: Address Space Model

Process A

ASID = 12

2^{32} - 1

Address Error

2^{31}

2 GB

Process B

ASID = 13

2^{32} - 1

Address Error

2^{31}

2 GB

ASID = Address Space Identifier

Process A and B have independent address spaces

All address spaces use a standard memory map

May only be accessed by kernel/supervisor

When Process A writes its address 9, it writes to a different physical memory location than Process B’s address 9

To let Process A and B share memory, OS maps parts of ASID 12 and ASID 13 to the same physical memory locations.

Still works (slowly!) if a process accesses more virtual memory than the machine has physical memory
System Control Registers

Status (12): Indicates user, supervisor, or kernel mode

EntryLo0 (2): 8-bit ASID field codes virtual address space ID.

User cannot write supervisor/kernel bits. Supervisor cannot write kernel bit.

User cannot change address translation configuration
MIPS Address Translation: How it works

“Virtual Addresses”

CPU

A0-A31

D0-D31

Data

Translation Look-Aside Buffer (TLB)

“Physical Addresses”

Memory

A0-A31

D0-D31

Translation Look-Aside Buffer (TLB)

A small fully-associative cache of mappings from virtual to physical addresses

TLB also contains ASID and kernel/supervisor bits for virtual address

Fast common case: Virtual address is in TLB, process has permission to read/write it.
Page tables encode virtual address spaces

A virtual address space is divided into blocks of memory called pages

A machine usually supports pages of a few sizes (MIPS R4000):

<table>
<thead>
<tr>
<th>Page Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 Kbytes</td>
</tr>
<tr>
<td>16 Kbytes</td>
</tr>
<tr>
<td>64 Kbytes</td>
</tr>
<tr>
<td>256 Kbytes</td>
</tr>
<tr>
<td>1 Mbyte</td>
</tr>
<tr>
<td>4 Mbytes</td>
</tr>
<tr>
<td>16 Mbytes</td>
</tr>
</tbody>
</table>

A page table is indexed by a virtual address

A valid page table entry codes physical memory “frame” address for the page
The TLB caches page table entries

TLB caches page table entries.

Virtual Address

V page no.  offset

Page Table

E 60 = L 65G
for ASID

/ B = K
/ 78
/ 60 = 765G

Page Table

V

Access
Rights:

PA

Physical frame address

P page no.  offset

Physical Address

V = 0 pages either reside on disk or have not yet been allocated. OS handles V = 0 "Page fault"

MIPS handles TLB misses in software (random replacement). Other machines use hardware.

In this example, physical and virtual pages must be the same size!
MIPS R4000 TLB: A closer look ...

“Virtual Addresses”

A0-A31
CPU
D0-D31

“Physical Addresses”

Virtual

Translation
Look-Aside
Buffer
(TLB)

Physical

Memory
System

Data

Virtual Address with 1M \(2^{20}\) 4-Kbyte pages

| 39 | 32 31 29 28
|---|---|---|---|
| 8 | 20 | 12 11 | 0

Bits 31, 30 and 29 of the virtual address select user, supervisor, or kernel address spaces.

Physical space larger than virtual space!

Physical Address

Offset passed unchanged to physical memory

Offset

PFN

35

UC Regents Fall 2008 © UCB
Can TLB and caching be overlapped?

Virtual Page Number  Page Offset

Virtual Translation Look-Aside Buffer (TLB)

Physical

Index  Byte Select

Cache Tags  Valid  Cache Data

This works, but ...

Q. What is the downside?
A. Inflexibility. VPN size locked to cache tag size.
Can we cache virtual addresses?

"Virtual Addresses"

CPU

A0-A31

D0-D31

Virtual Cache

D0-D31

Virtual

Translation Look-Aside Buffer (TLB)

Main Memory

A0-A31

D0-D31

"Physical Addresses"

Only use TLB on a cache miss!

Downside: a subtle, difficult problem. What is it?

A. Synonym problem. If two address spaces share a physical frame, data may be in cache twice. Maintaining consistency is a challenge.
Virtualization
Parallels: Running Windows on a Mac

Like software emulating a PC, but different. Use an Intel-based Mac, runs on top of OS X. Uses hardware support to create a fast virtual PC that boots Windows.

+++ Reasonable performance.

Virtual CPU runs 33% slower than running on physical CPU.

2 GB physical memory for a 512 MB virtual PC to run w/o disk swaps.

Source: http://www.atpm.com/12.10/parallels.shtml
“Hardware assist?” What do we mean?

* In an emulator, we run Windows code by simulating the CPU in software.

* In a virtual machine, we let “safe” instructions (ex: ADD R3 R2 R1) run on the actual hardware.

* We use hardware features to deny direct execution of instructions that could “break out” of the virtual machine.

* We have seen an example of this sort of hardware feature earlier today ...
Recall: A LW that misses TLB launches a program!

MIPS handles TLB misses in software (random replacement). Other machines use hardware.

In this example, physical and virtual pages must be the same size!

V=0 pages either reside on disk or have not yet been allocated. OS handles V=0 "Page fault"
General Mechanism: Instruction Trap

Conceptually, we set CPU up to rewrite “unsafe” instructions with a function call.

Sample Program

<table>
<thead>
<tr>
<th>Instruction</th>
<th>What CPU Does</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND R4, R3, R2</td>
<td>ADD R4, R3, R2</td>
</tr>
<tr>
<td>Unsafe</td>
<td>JAL Unsafe_stub</td>
</tr>
<tr>
<td>ADD R5, R2, R2</td>
<td>NOP</td>
</tr>
</tbody>
</table>

CPU rewrite instructions??? We have already done this for pipelining ....
Recall: “Muxing in” NOPS to do stalls

Sample program

ADD R4, R3, R2
OR R5, R4, R2

Keep executing OR instruction until R4 is ready. Until then, send NOPS to IR 2/3.

Freeze PC and IR until stall is over.

New datapath hardware

(1) Mux into IR 2/3 to feed in NOP.

(2) Write enable on PC and IR 1/2

Let ADD proceed to WB stage, so that R4 is written to regfile.
Conclusion: Superpipelining, Superscalar

The 5 stage pipeline: a starting point for performance enhancements, a building block for multiprocessing.

* Superpipelining: Reduce critical path by adding more pipeline stages. Has the potential to hurt the CPI.

* Superscalar: Multiple instructions at once. Programs must fit the issue rules. Adds complexity.
Next Monday:

This Friday:
Project Meeting, 10 AM, 125 Cory

Final Presentation
Fri, Dec 5