CS 152
Computer Architecture and Engineering

Lecture 21 – Advanced Processors II

Thanks to Krste Asanovic ...

2005-4-7

John Lazzaro

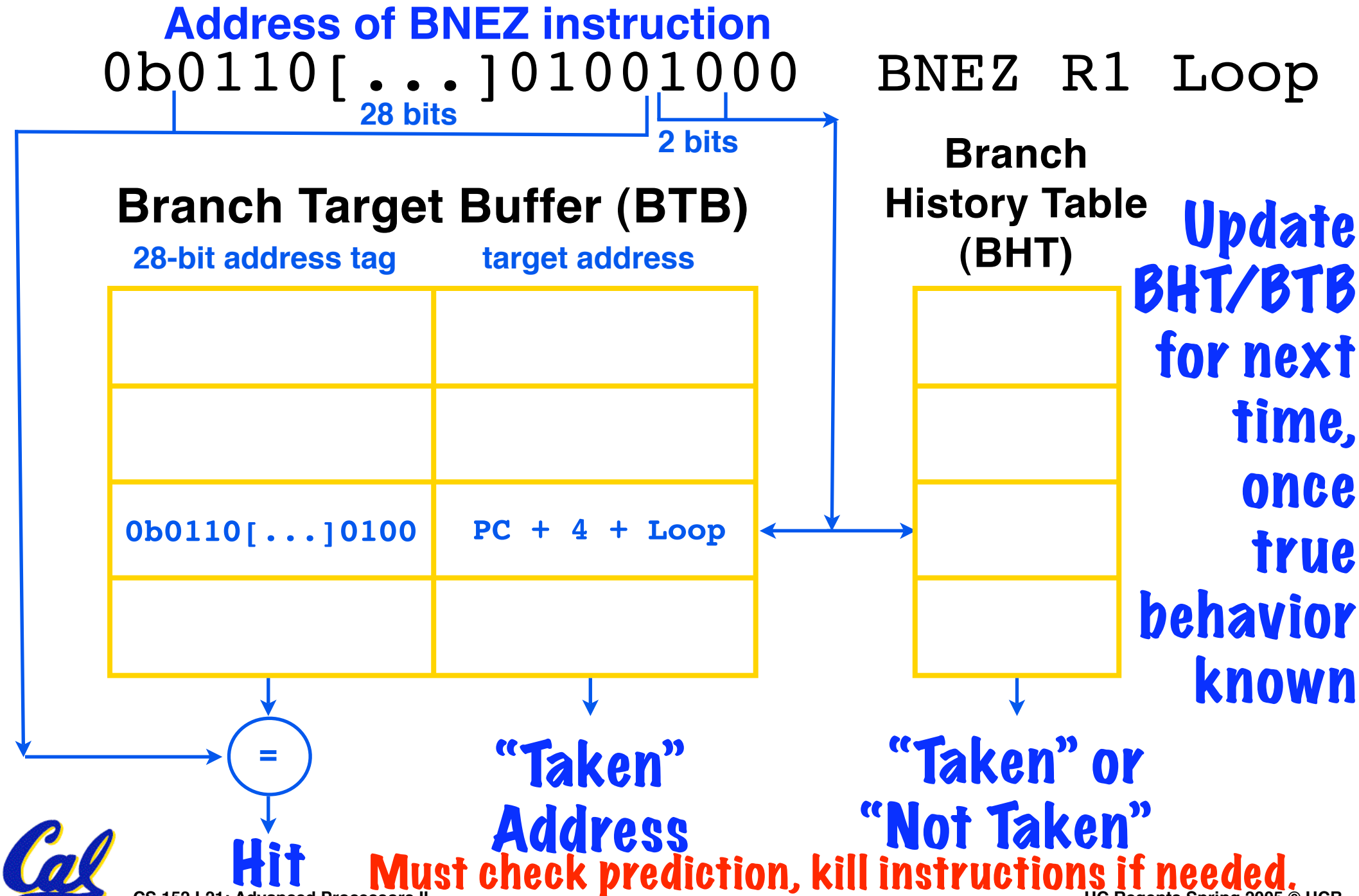
(www.cs.berkeley.edu/~lazzaro)

TAs: Ted Hong and David Marquardt

www-inst.eecs.berkeley.edu/~cs152/

UC Regents Spring 2005 © UCB

Last Time: Superpipelining & Superscalar

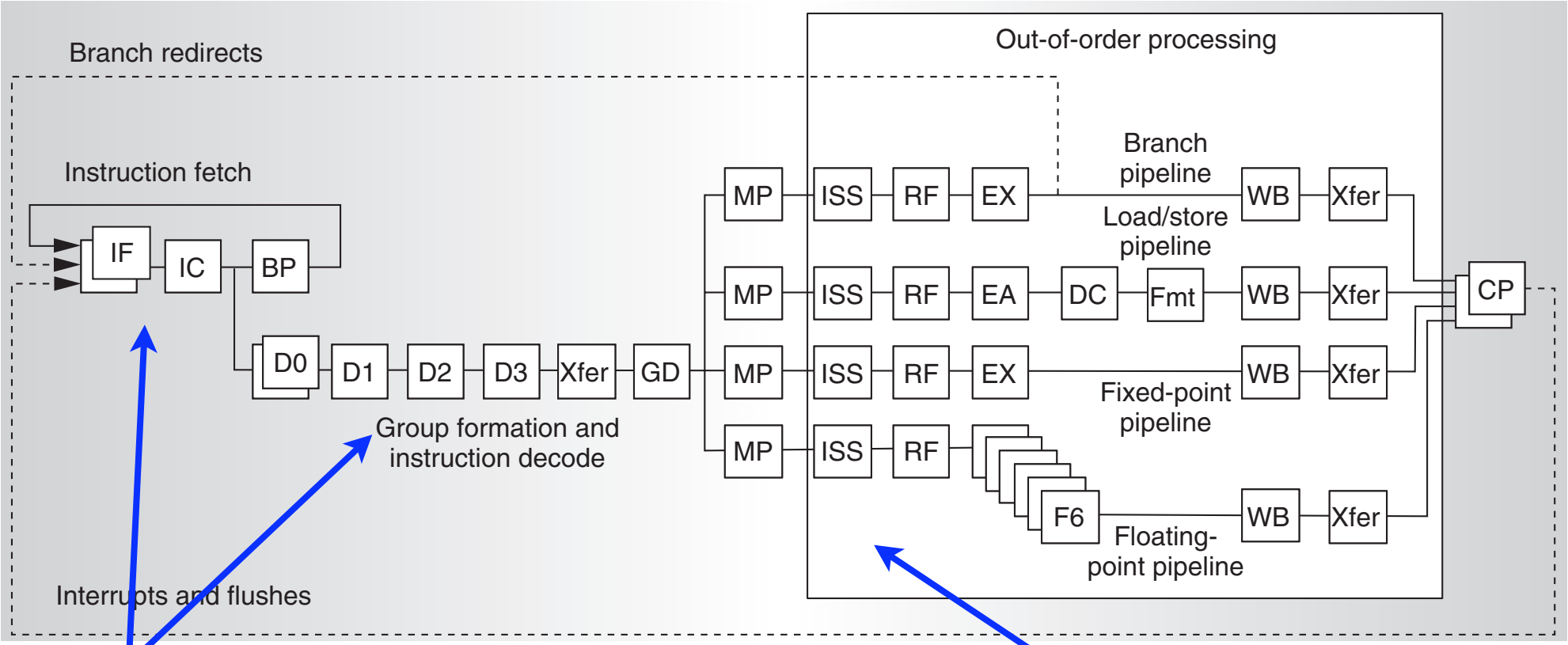


UC Regents Spring 2005 © UCB

Today: Dynamic Scheduling Overview

Goal: Enable out-of-order by breaking pipeline in two: fetch and execution.

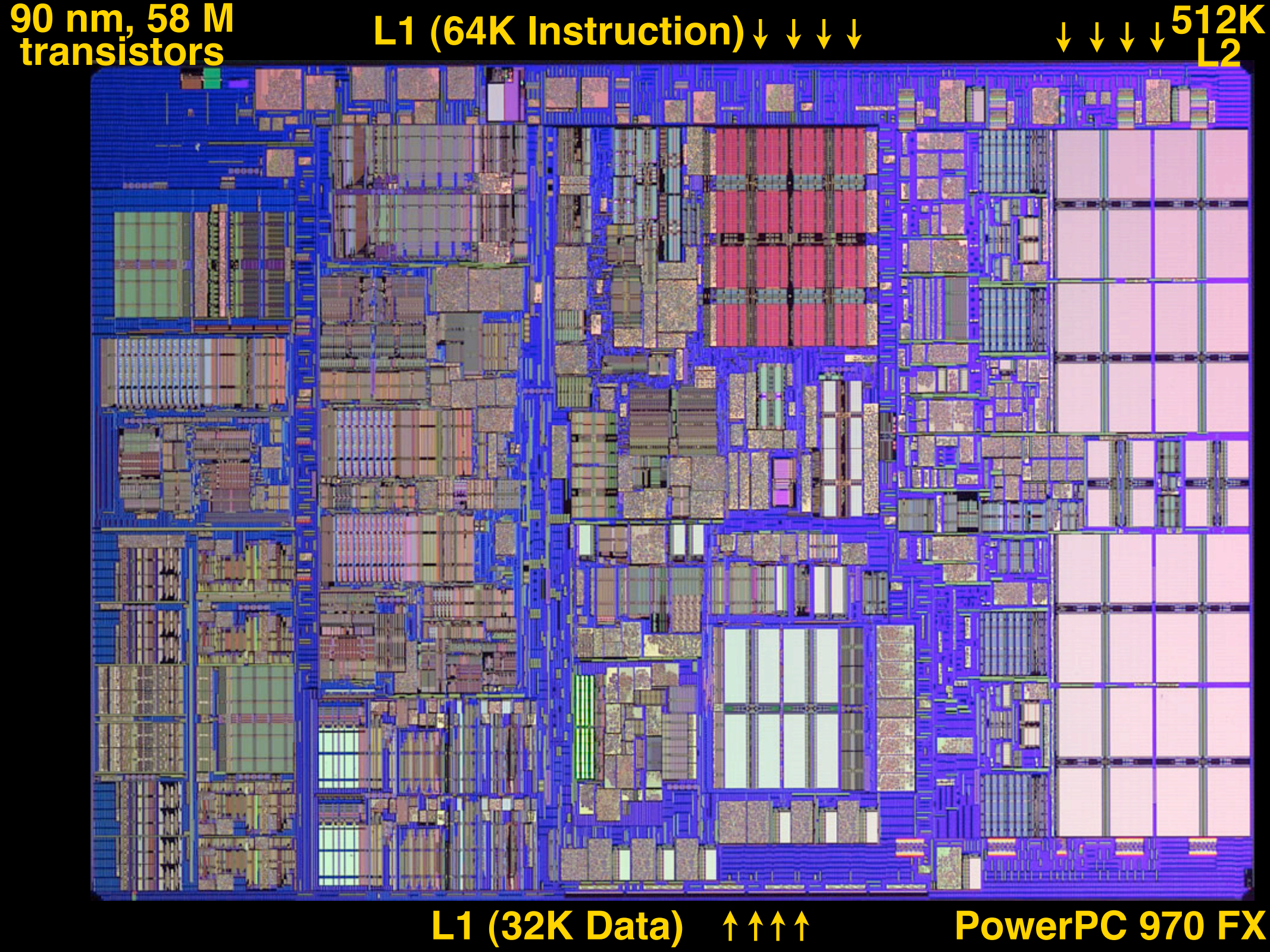
Example: IBM Power 5:



I-fetch and decode: like static pipelines

Today's focus: execution unit

UC Regents Spring 2005 © UCB


90 nm, 58 M transistors
L1 (64K Instruction) ↓ ↓ ↓ ↓
↓ ↓ ↓ ↓ 512K L2
L1 (32K Data) ↑↑↑↑
PowerPC 970 FX

Recall: WAR and WAW hazards ...

Write After Read (WAR) hazards. Instruction I2 expects to write over a data value after an earlier instruction I1 reads it. But instead, I2 **writes too early**, and I1 sees the new value.

Write After Write (WAW) hazards. Instruction I2 writes over data an earlier instruction I1 also writes. But instead, **I1 writes after I2**, and the final data value is incorrect.

Dynamic scheduling eliminates WAR and WAW hazards, making out-of-order execution tractable

UC Regents Spring 2005 © UCB

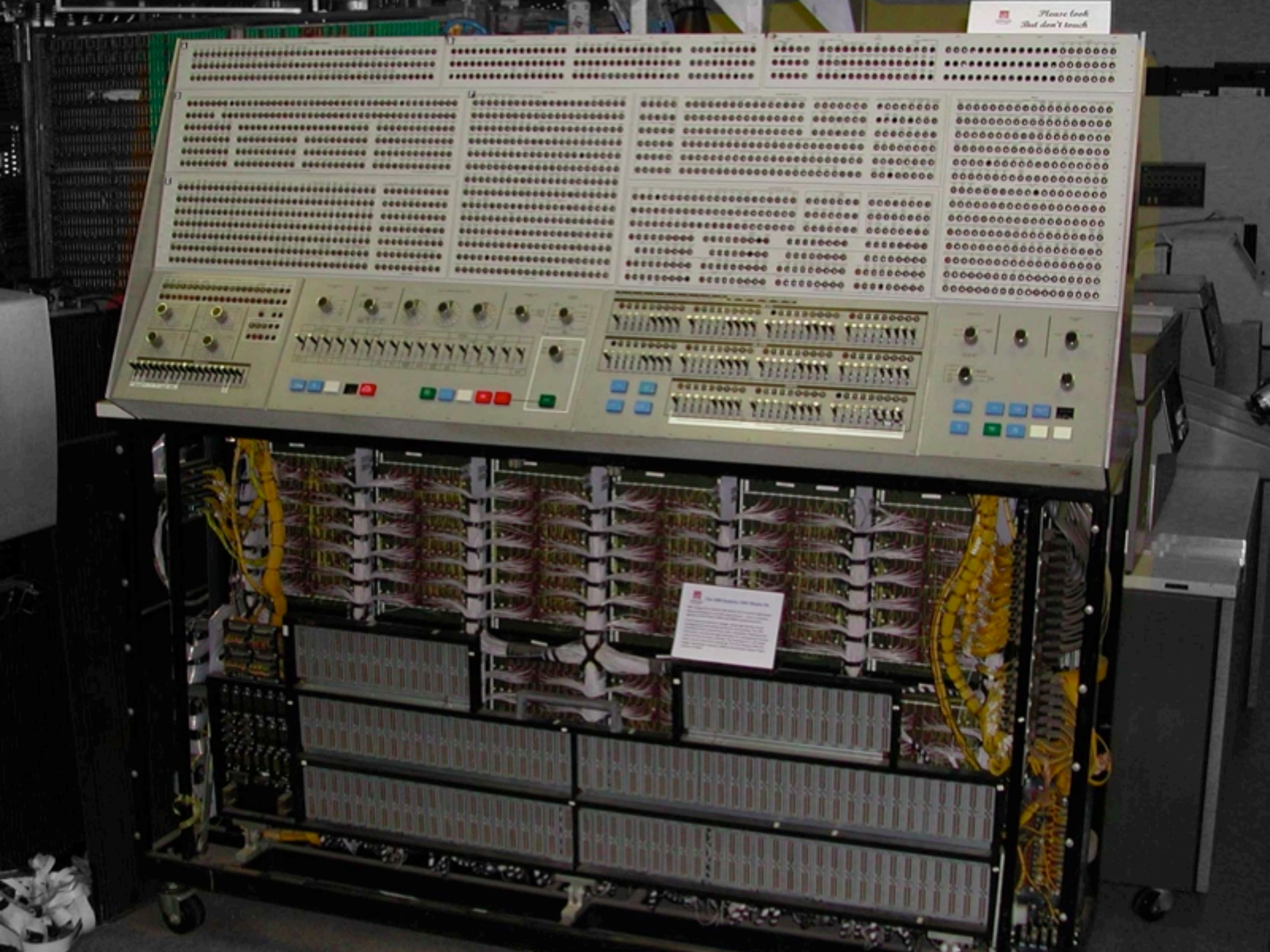
Dynamic Scheduling: A mix of 3 ideas

Imagine: an endless supply of registers ...

- **Top-down idea:** Registers that may be written only once (but may be read many times) eliminate WAW and WAR hazards.
- **Mid-level idea:** An instruction waiting for an operand to execute may trigger on the (single) write to the associated register. (eliminates RAW hazards)
- **Bottom-up idea:** To support "snooping" on register writes, attach all machine elements to a common bus.

Robert Tomasulo, IBM, 1967. FP unit for IBM 360/91

UC Regents Spring 2005 © UCB


Please look
But don't touch

Register Renaming

Imagine: an endless supply of registers???
How???

UC Regents Spring 2005 © UCB

Consider this simple loop ...

```
Loop:  LD    F0,0(R1)   ;F0= array element
       ADDD  F4,F0,F2   ;add scalar from F2
       SD    F4,0(R1)   ;store result
       SUBI  R1,R1,8    ;decrement pointer 8B (DW)
       BNEZ  R1,Loop    ;branch R1!=zero
       NOP              ;delayed branch slot
```

Every pass through the loop introduces the potential for WAW and/or WAR hazards for F0, F4, and R1.

UC Regents Spring 2005 © UCB

Given an endless supply of registers ...

Rename "architected registers" (Ri, Fi) to new "physical registers" (PRi, PFi) on each write.

```
        ADDI R1,R0,64
Loop:   LD   F0,0(R1)
        ADDD F4,F0,F2
        SD   F4,0(R1)
        SUBI R1,R1,8
        BNEZ R1,Loop
        NOP
```

R1→ PR01
F0→ PF00

```
        ADDI PR01,PR00,64
        LD PF00 0(PR01)
        ADDD PF04, PF00, PF02
        SD PF04, 0(PR01)
        SUBI PR11, PR01, 8
        BEQZ PR11 ENDLOOP
ITER2:  LD PF10 0(PR11)
        ADDD PF14, PF10, PF02
        SD PF14, 0(PR11)
        SUBI PR21, PR11, 8
        BEQZ PR21 ENDLOOP
ITER3:  LD PF20 0(PR21)
        [...]
```

What was gained?
An instruction may execute once all of its source registers have been written.

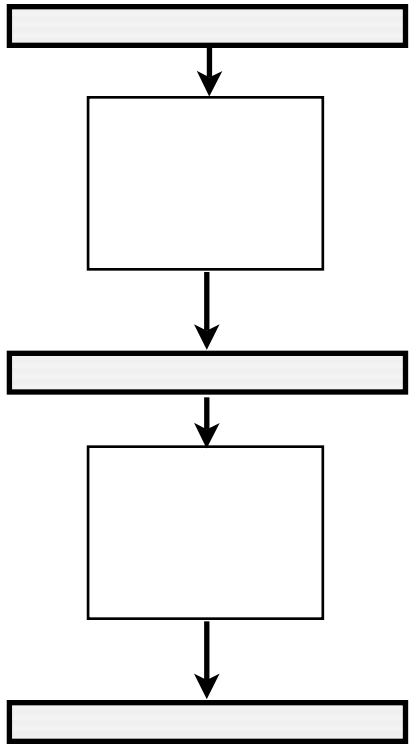
UC Regents Spring 2005 © UCB

Bus-Based CPUs

UC Regents Spring 2005 © UCB

A common bus == long wires == slow?

Pipelines in theory



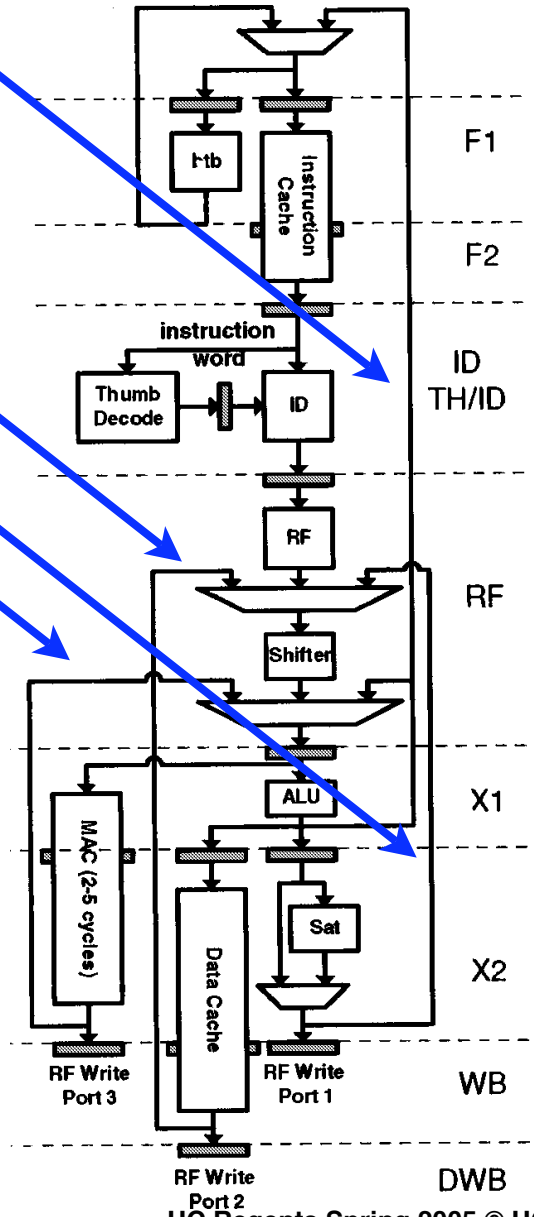
Wires are short, so clock periods can be short.

"wiring by abutment"

Long wires are the price we paid to avoid stalls

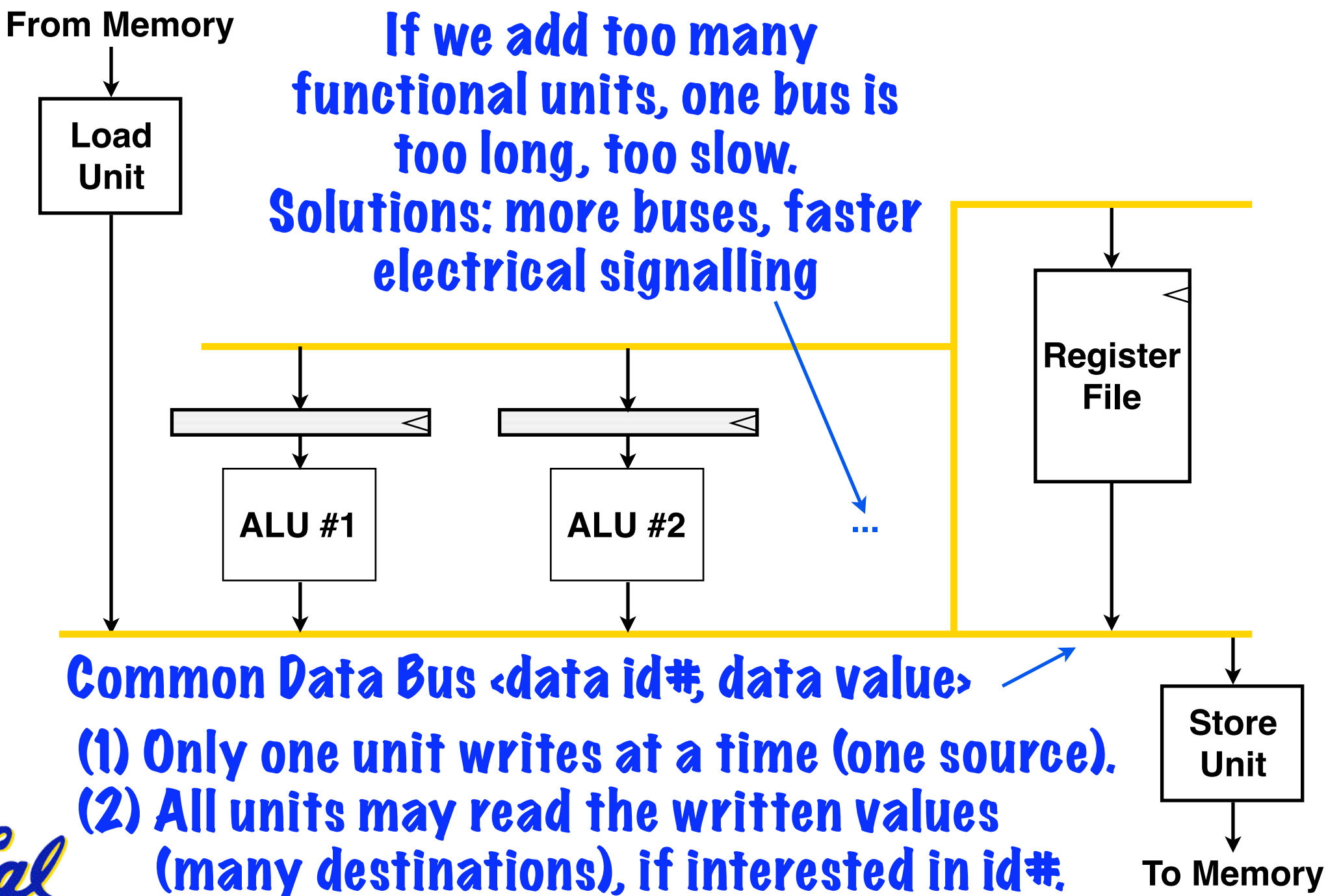
Conjecture: If processor speed is limited by long wires, lets do a design that fully uses the semantics of long wires by using a bus.

Pipelines in practice



UC Regents Spring 2005 © UCB

A bus-based multi-cycle computer



(1) Only one unit writes at a time (one source).
(2) All units may read the written values (many destinations), if interested in id#.

UC Regents Spring 2005 © UCB

Data-Driven Execution

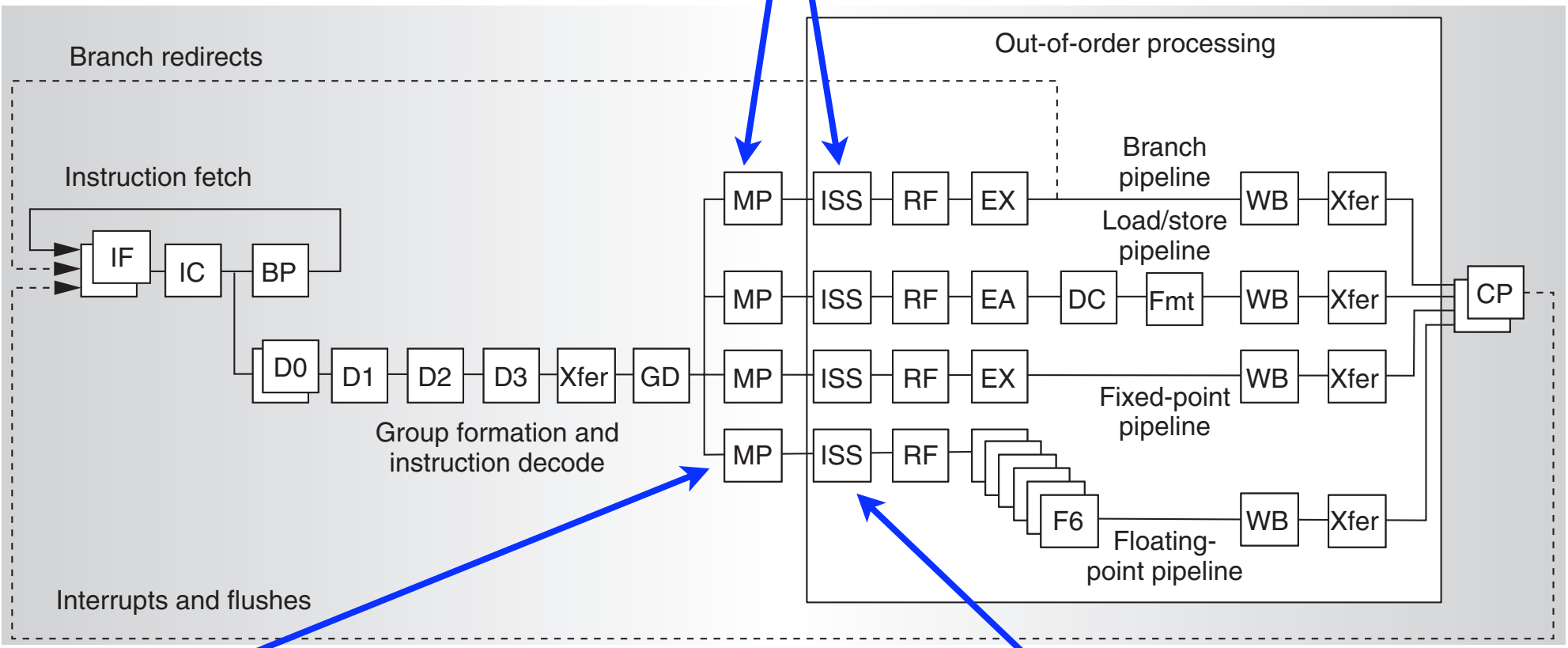
(Associative Control)

Caveat: In comparison to static pipelines, there is great diversity in dynamic scheduling implementations. Presentation that follows is a composite, and does not reflect any specific machine.

UC Regents Spring 2005 © UCB

Recall: IBM Power 5 block diagram ...

Queues between instruction fetch and execution.



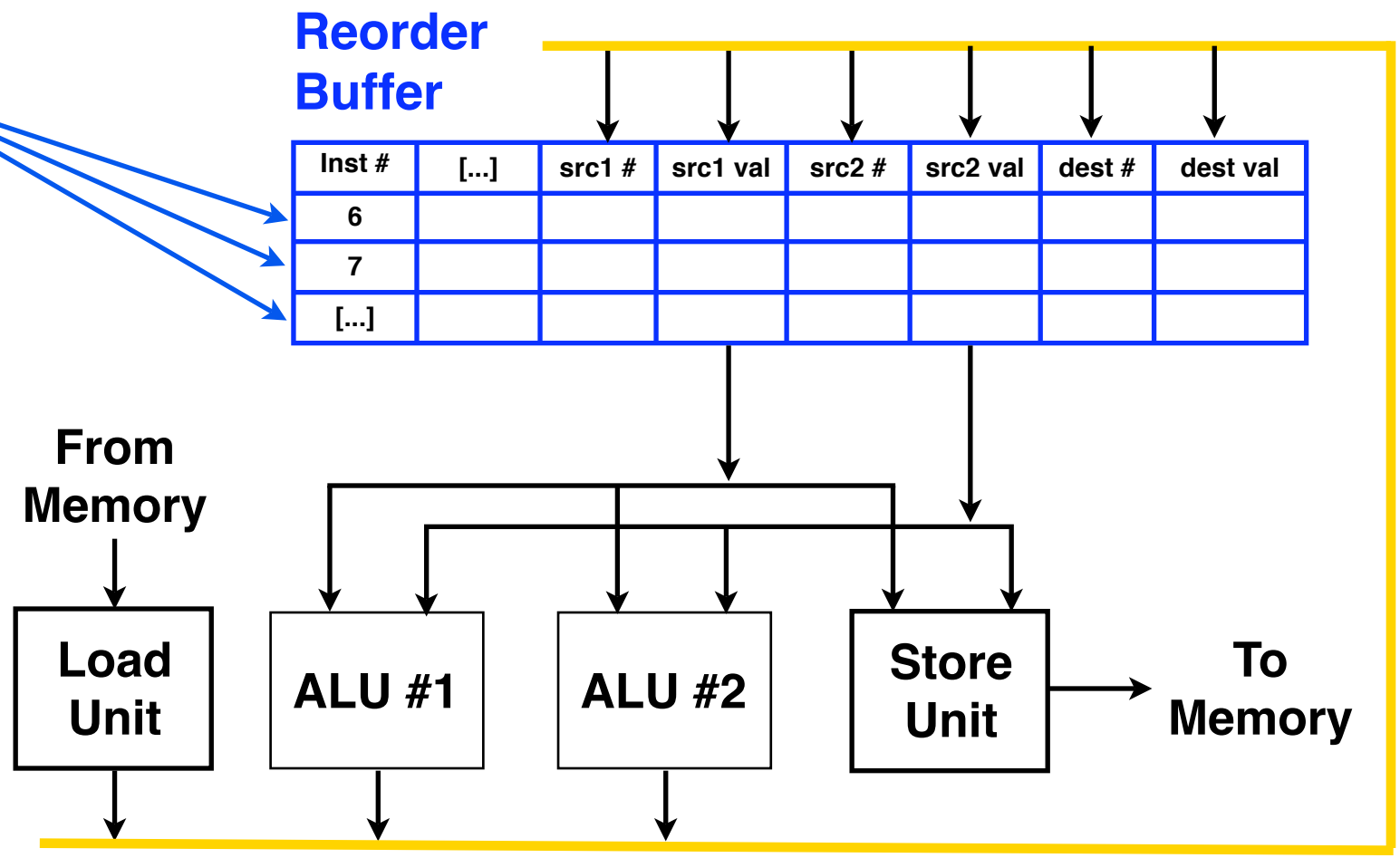
MP = "Mapping" from architected registers to physical registers (renaming).

ISS = Instruction Issue

UC Regents Spring 2005 © UCB

Instructions placed in "Reorder Buffer"

Each line holds physical <src1, src2, dest> registers for an instruction, and controls when it executes



Common Data Bus: <reg #, reg val>

Execution engine works on the physical registers, not the architecture registers.

UC Regents Spring 2005 © UCB

Circular Reorder Buffer: A closer look

Next instr to "commit", (complete).

Instruction opcode

Use bit (1 if line is in use)

Execute bit (0 if waiting ...)

Tail of List

Head of List

Inst#	Op	U	E	#1	#2	#d	P1	P2	Pd	P1 value	P2 value	Pd value
		0										
		0										
8		1										
9		1										
10		1										
		0										
		0										

Add next inst, in program order.

Physical register numbers

Valid bits for values

Copies of physical register values

UC Regents Spring 2005 © UCB

Example: The life of `ADD R3,R1,R2`

Issue: R1 "renamed" to PR21, whose value (13) was set by an earlier instruction. R2 renamed to PR22; it has not been written. R3 renamed to PR23.

Inst#	Op	U	E	#1	#2	#d	P1	P2	Pd	P1 value	P2 value	Pd value
9	Add	1	0	21	22	23	1	0	0	13	-	-

A write to PR22 appears on the bus, value 87. Both operands are now known, so 13 and 87 sent to ALU.

Inst#	Op	U	E	#1	#2	#d	P1	P2	Pd	P1 value	P2 value	Pd value
9	Add	1	1	21	22	23	1	1	0	13	87	-

ALU does the add, writing 100 to PR23.

Inst#	Op	U	E	#1	#2	#d	P1	P2	Pd	P1 value	P2 value	Pd value
9	Add	1	1	21	22	23	1	1	1	13	87	100

UC Regents Spring 2005 © UCB

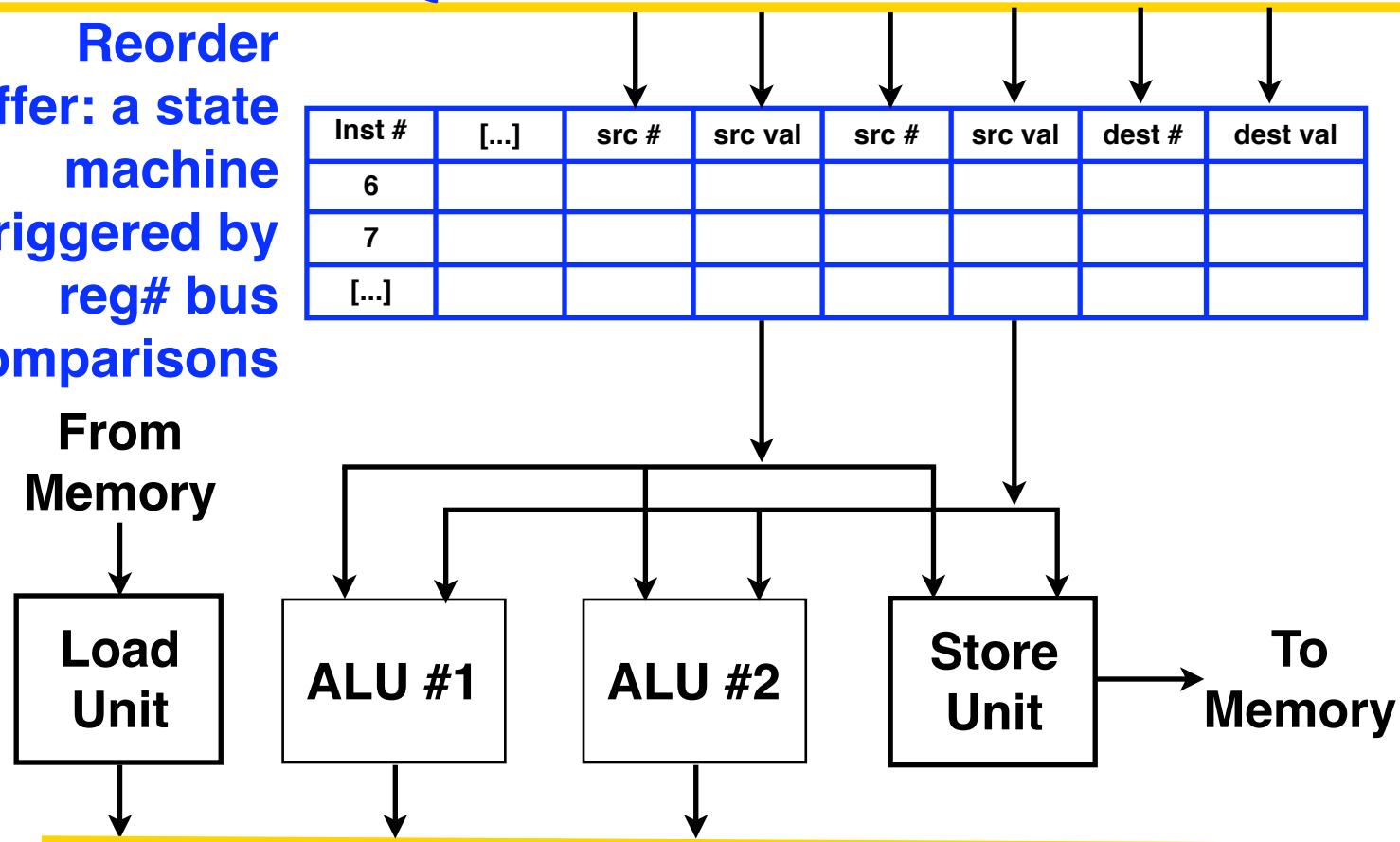
More details (many are still overlooked)

Example: Load/Store Disambiguation

Issue logic monitors bus to maintain a physical register file, so that it can fill in <val> fields during issue.

Reorder buffer: a state machine triggered by reg# bus comparisons

Inst #	[...]	src #	src val	src #	src val	dest #	dest val
6							
7							
[...]							



Common Data Bus: <reg #, reg val>

Q. Why are we storing each physical register value several times in the reorder buffer? See next topic ...

UC Regents Spring 2005 © UCB

Exceptions and Interrupts

Exception: An unusual event happens to an instruction during its execution. Examples: divide by zero, undefined opcode.

Interrupt: Hardware signal to switch the processor to a new instruction stream. Example: a sound card interrupts when it needs more audio output samples (an audio "click" happens if it is left waiting).

UC Regents Spring 2005 © UCB

Challenge: Precise Interrupt / Exception

Definition:

(or exception)

It must appear as if an interrupt is taken between two instructions (say I_i and I_{i+1})

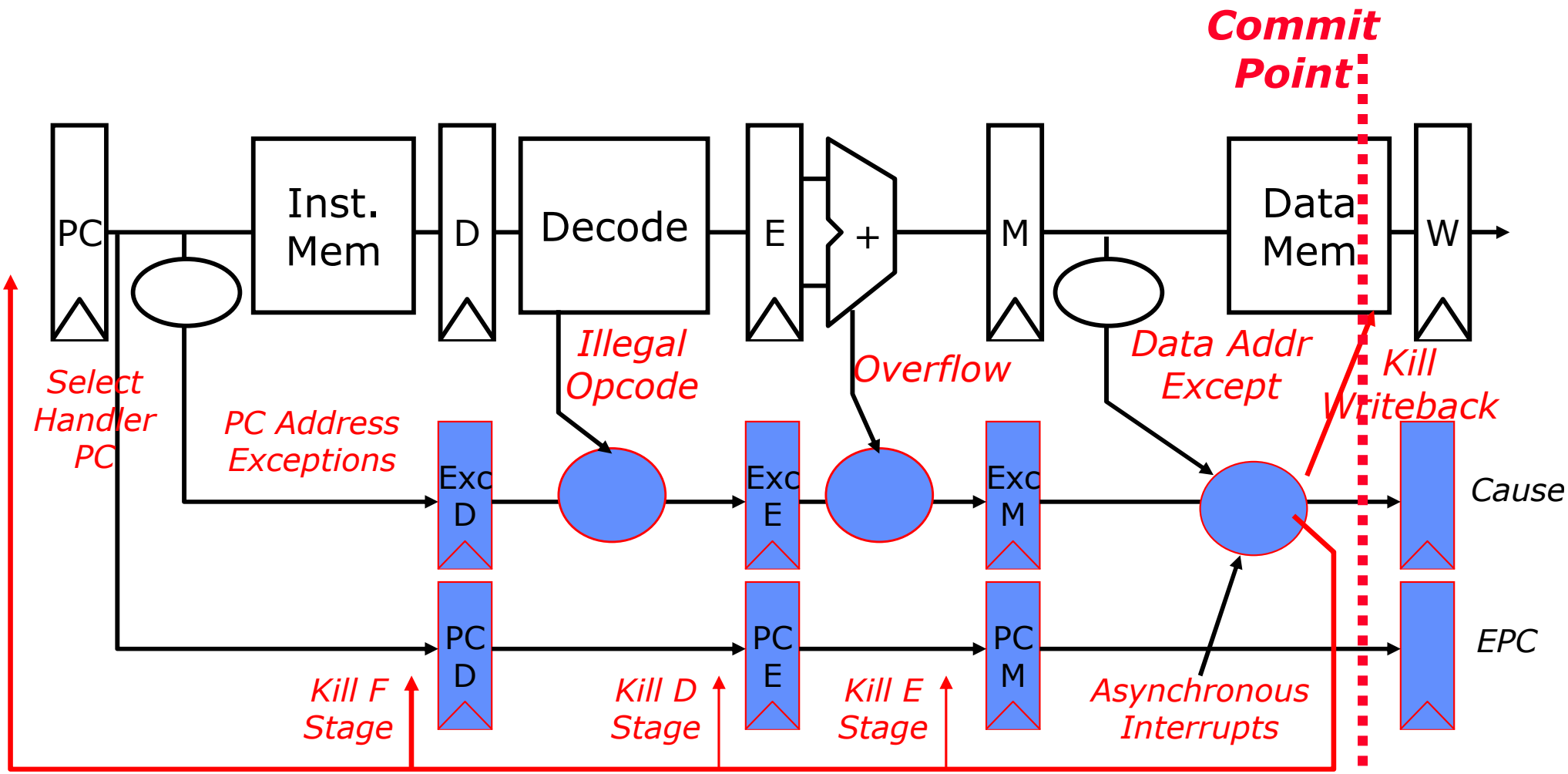
- the effect of all instructions up to and including I_i is totally complete
- no effect of any instruction after I_i has taken place

The interrupt handler either aborts the program or restarts it at I_{i+1} .

Follows from the "contract" between the architect and the programmer ...

UC Regents Spring 2005 © UCB

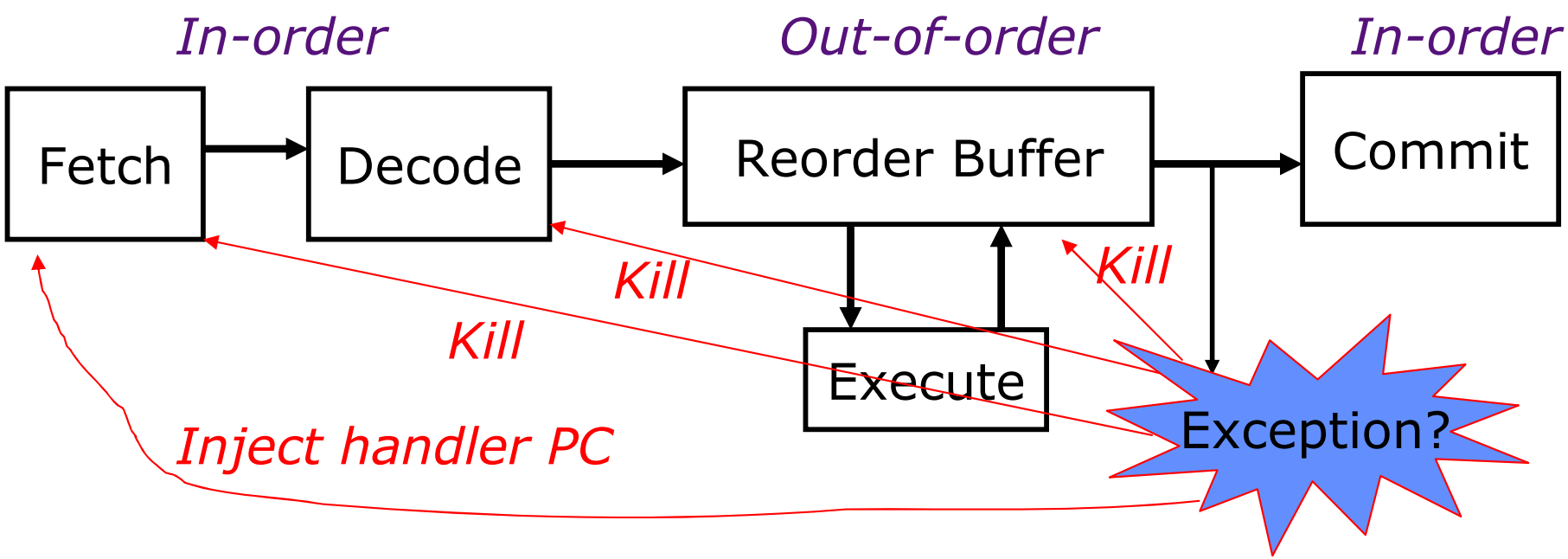
Precise Exceptions in Static Pipelines



Key observation: architected state only change in memory and register write stages.

UC Regents Spring 2005 © UCB

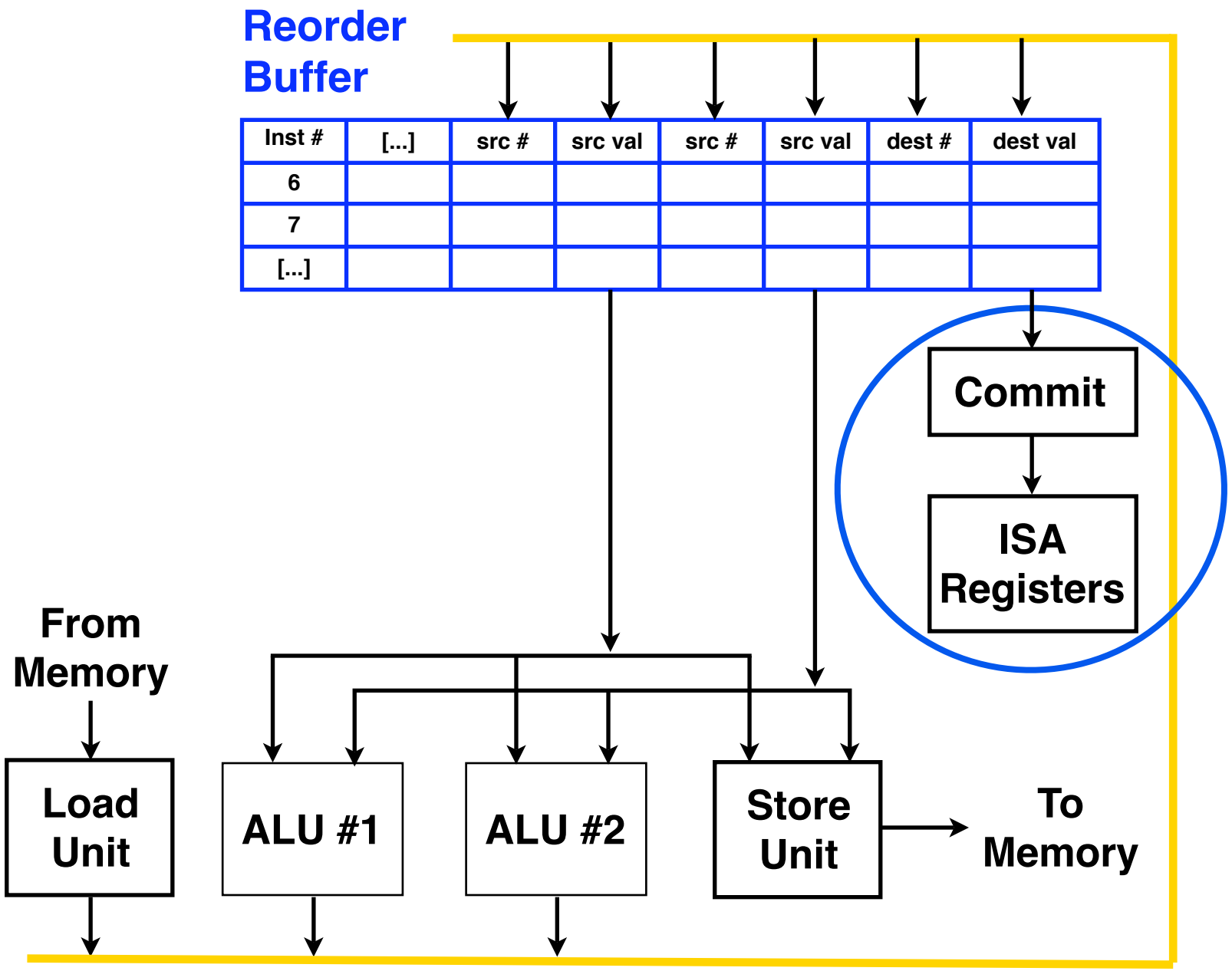
Dynamic scheduling and exceptions ...



Key observation: Only the architected state needs to be precise, not the physical register state. So, we delay removing instructions from the reorder buffer until we are ready to "commit" to that state changing the architected registers.

UC Regents Spring 2005 © UCB

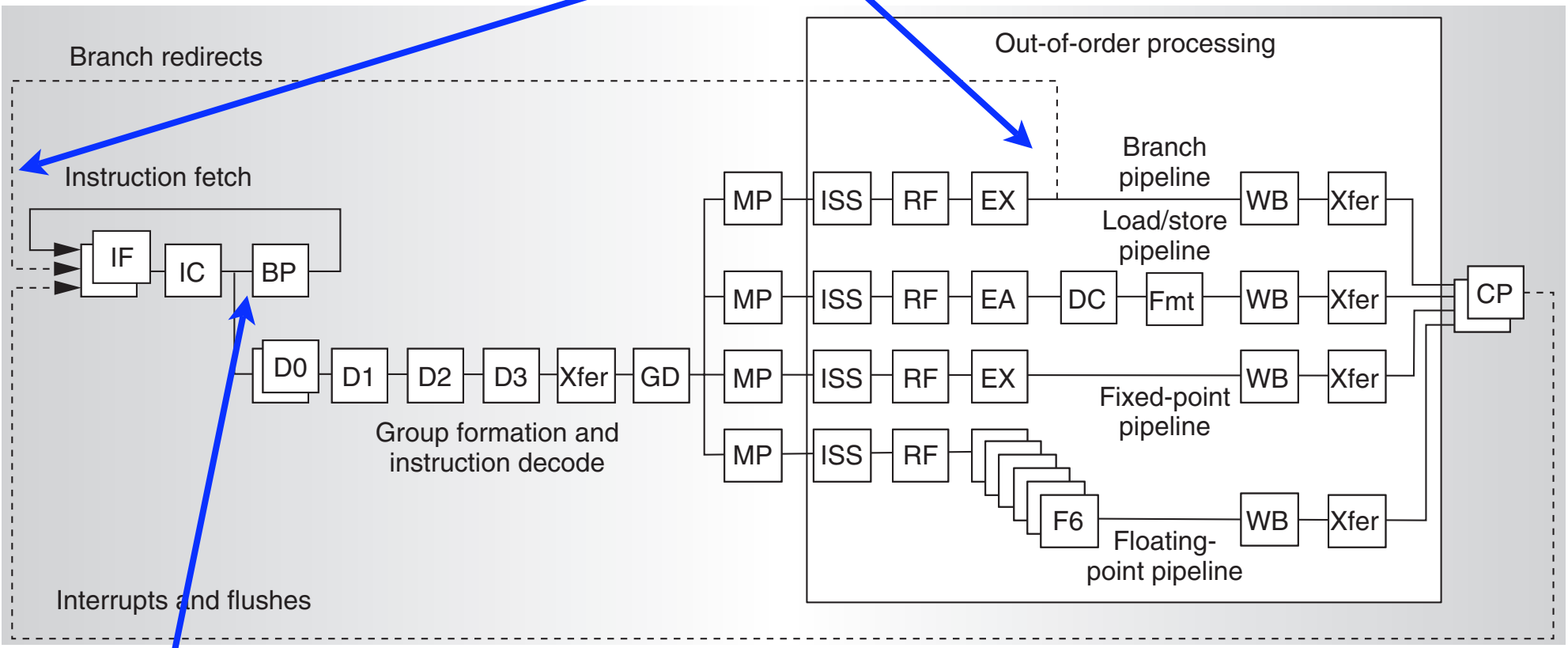
Add completion logic to data path ...



UC Regents Spring 2005 © UCB

Final thought: Branch prediction required

Because so many stages between predict and result!



BP = Branch prediction. On IBM Power 5, quite complex ... uses a predictor to predict the best branch prediction algorithm!

UC Regents Spring 2005 © UCB

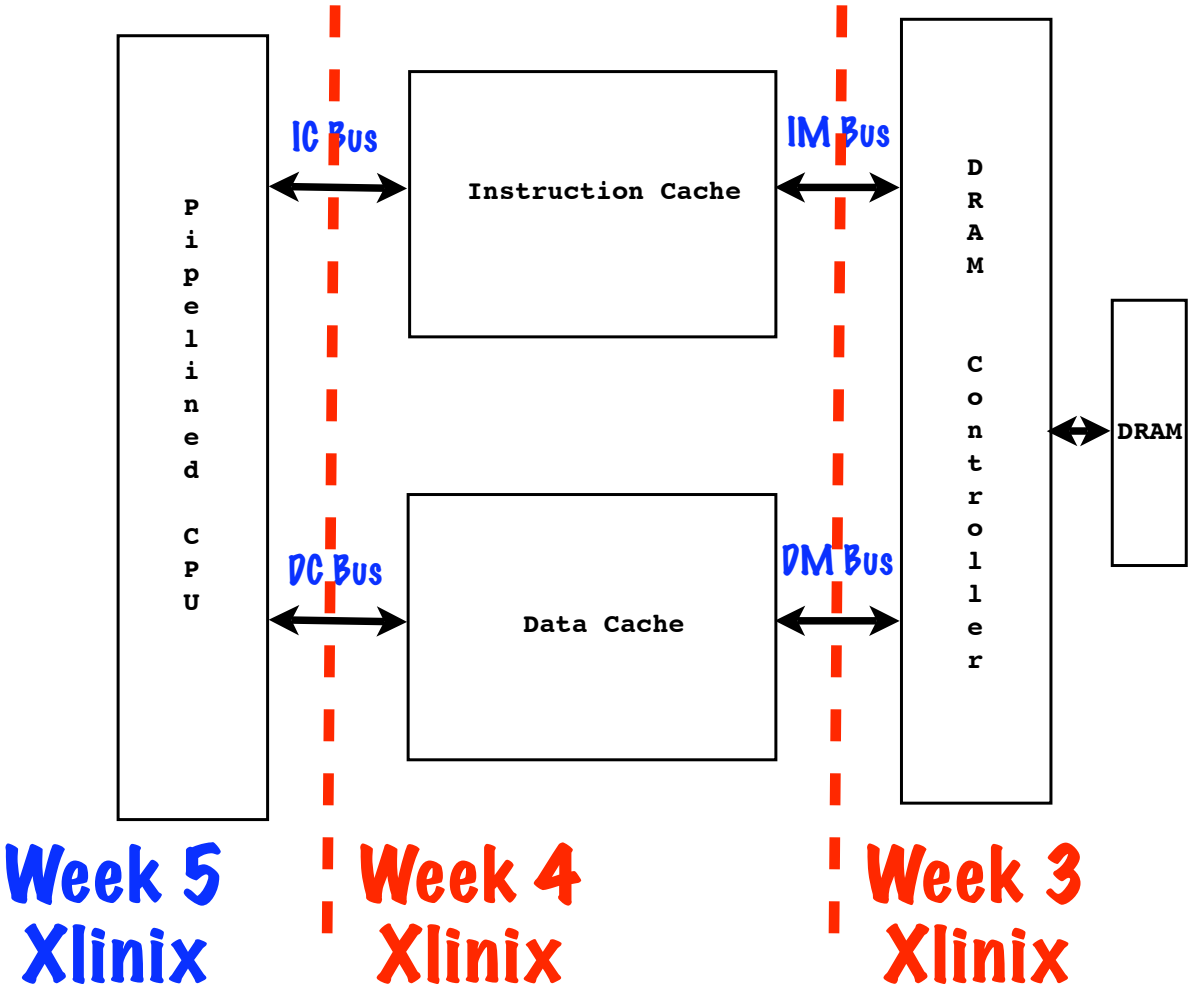
Conclusions: Dynamic Scheduling

- Three big ideas: register renaming, data-driven detection of RAW resolution, bus-based architecture.
- Very complex, but enables many things: out-of-order execution, multiple issue, loop unrolling, etc.
- Has saved architectures that have a small number of registers: IBM 360 floating-point ISA, Intel x86 ISA.

UC Regents Spring 2005 © UCB

Reminder: Friday Test Bench Checkoff

F 4/8				Final Project: Test Bench Checkoff



Test vector suite for Week 3/4/5 checkoffs, running in ModelSim (3/4) and SPIM (5).

Detailed block diagrams, state machines, and Lab 3 CPU changes

UC Regents Spring 2005 © UCB