

CSE 140

Computer Architecture

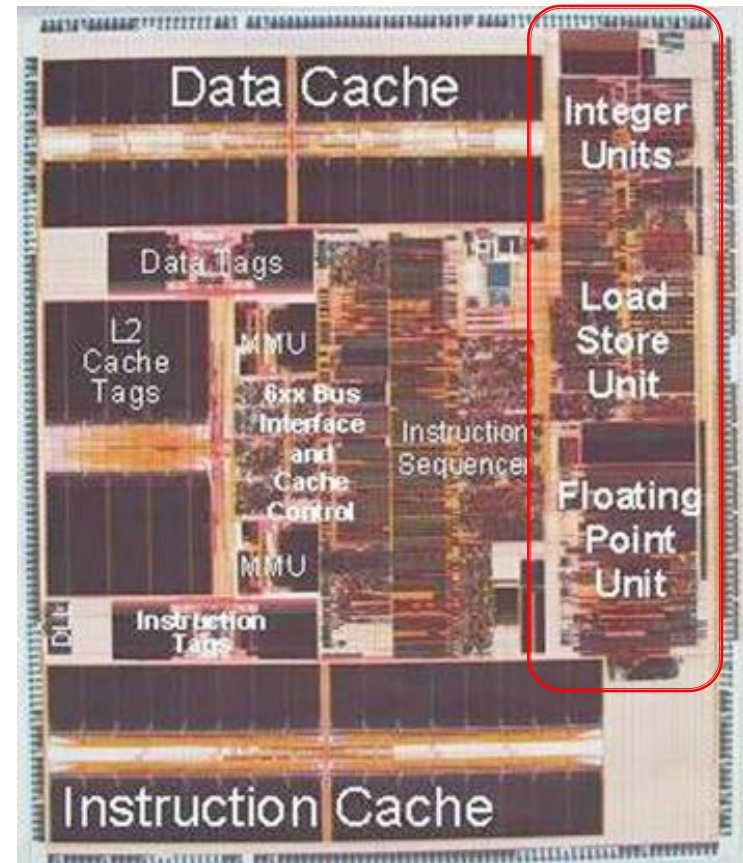
Lecture 12 – Complex Pipeline

Announcement

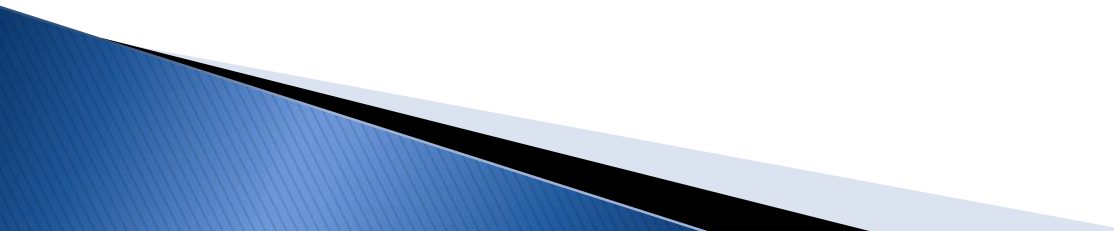
- ▶ No new assignment this week
 - Wrap up HW #3 and Project #1 in labs
- ▶ Project #1
 - Due 10/11 (Friday) at 11:59pm
- ▶ Midterm #1
 - 10/17, during lecture (not 10/10)
 - Lectures #1 - #8 (No virtual memory)
 - HW #1 - #3
 - Review on 10/15
- ▶ Reading assignment
 - Chapter 4.9, 4.10, 4.12
 - Do all **Participation Activities** in each section
 - Due Thursday (10/10) at 11:59pm

An Actual CPU – Early PowerPC

- ▶ Cache
 - 32 KB Instructions and 32 KB Data L1 caches
 - External L2 Cache interface with integrated controller and cache tags, supports up to 1 MByte external L2 cache
 - Dual Memory Management Units (MMU) with Translation Lookaside Buffers (TLB)
- ▶ Pipelining
 - 5 stages
 - Superscalar (3 inst/cycle)
 - 6 execution units (2 integer, 1 complex multi-cycle integer, **1 double precision IEEE floating point**, 1 load/store, 1 branch)



Complex Pipelining: Motivation

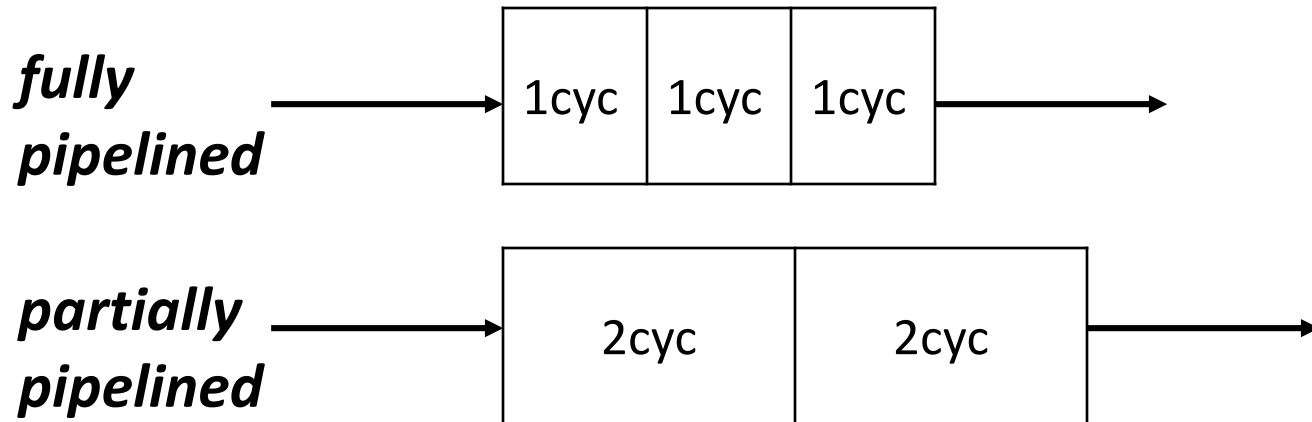
- ▶ Pipelining becomes complex when we want high performance in the presence of:
 - Long latency or partially pipelined floating-point units
 - Memory systems with variable access time
 - Multiple arithmetic and memory units
- 

Floating-Point Unit (FPU)

- ▶ Much more hardware than an integer unit
 - Single-cycle FPU is a bad idea - why?
- ▶ It is common to have several FPU's
 - It is common to have different types of FPU's Fadd, Fmul, Fdiv, ...
- ▶ An FPU may be pipelined, partially pipelined or not pipelined
- ▶ To operate several FPU's concurrently the FP register file needs to have more read and write ports

Functional Unit Characteristics

- ▶ Functional units have internal pipeline registers
 - operands are latched when an instruction enters a functional unit
 - inputs to a functional unit (e.g., register file) can change during a long latency operation

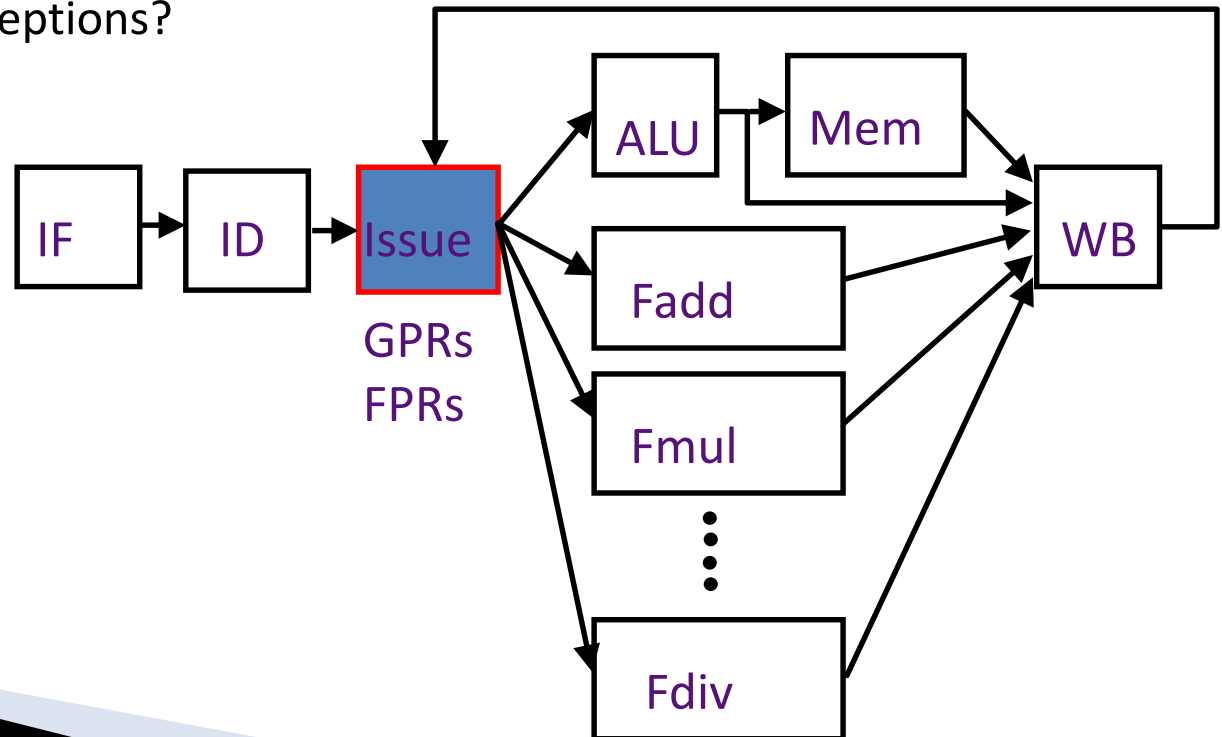


Floating-Point ISA

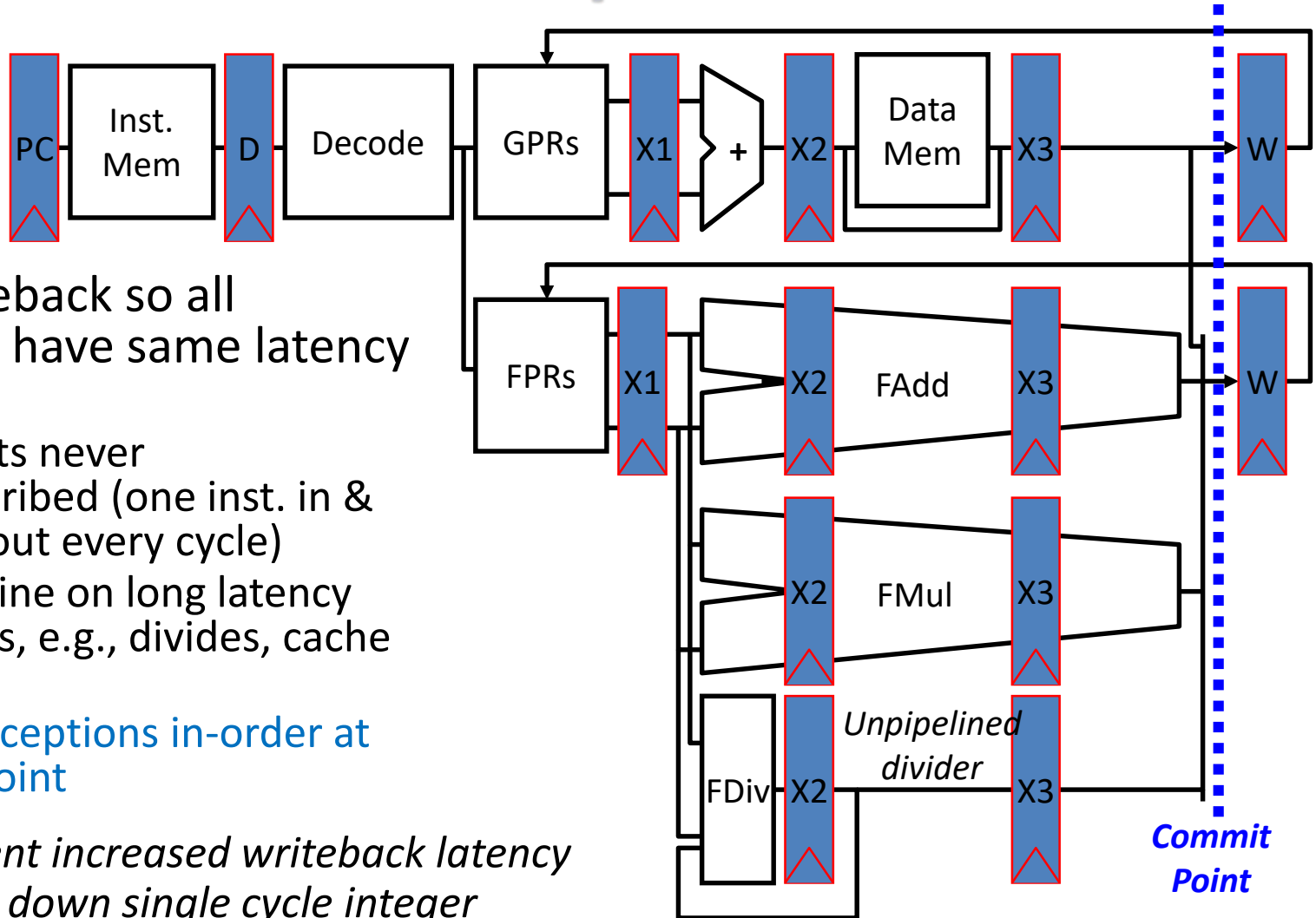
- ▶ Interaction between the floating-point datapath and the integer datapath is determined largely by the ISA
- ▶ MIPS ISA
 - separate register files for FP and Integer instructions
 - the only interaction is via a set of move/convert instructions (some ISA's don't even permit this)
 - separate load/store for **FPR's** and **GPR's (general purpose registers)** but both use **GPR's** for address calculation
 - separate conditions for branches
 - FP branches are defined in terms of set condition codes
 - Read Ch. 3 for more details

Complex Pipeline Control Issues

- ▶ **Structural conflicts at the execution stage** if some FPU or memory unit is not pipelined and takes more than one cycle
- ▶ **Structural conflicts at the write-back stage** due to variable latencies of different functional units
- ▶ **Out-of-order write hazards** due to variable latencies of different functional units
- ▶ How to handle exceptions?



Complex In-Order Pipeline

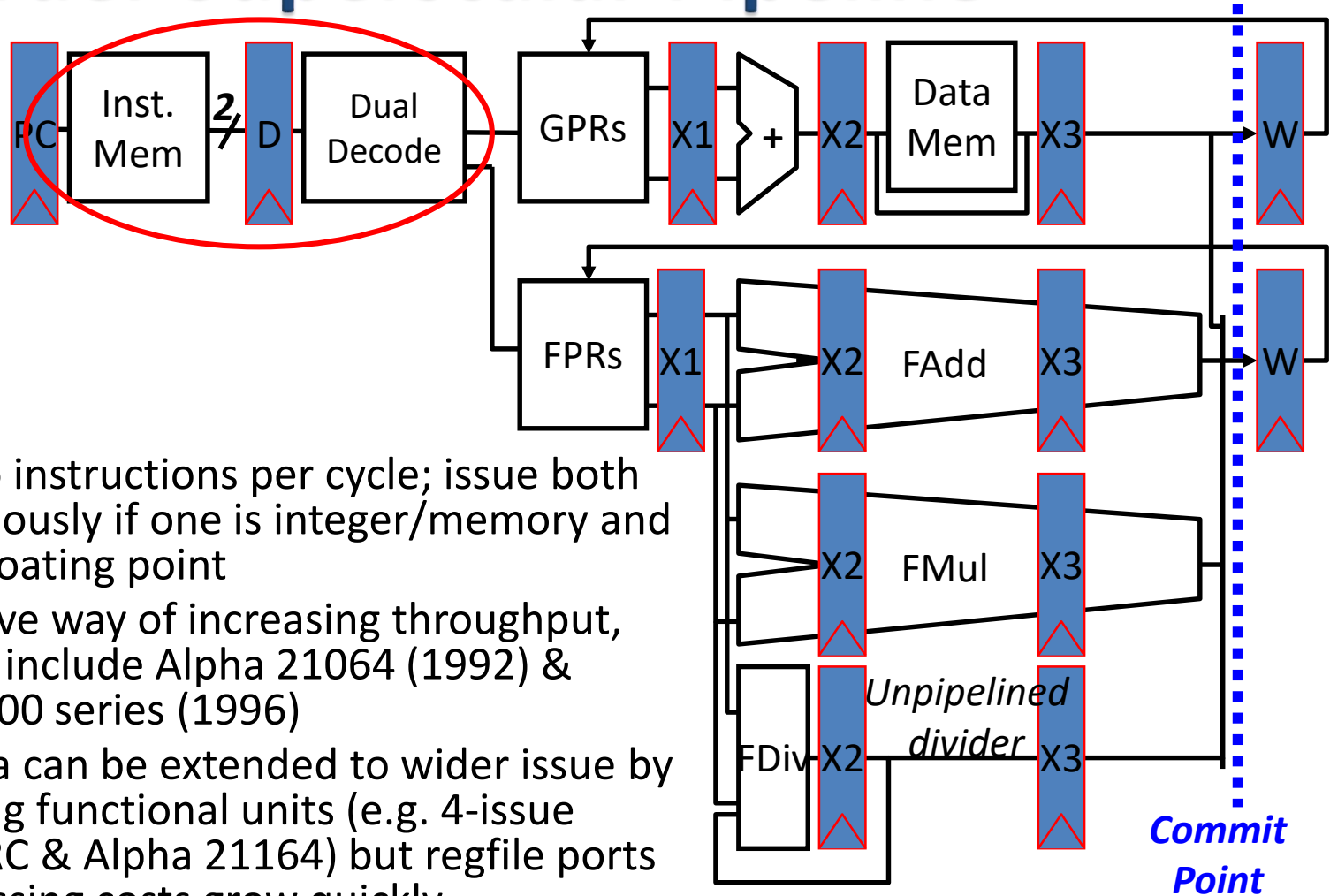


- ▶ Delay writeback so all operations have same latency to W stage

- Write ports never oversubscribed (one inst. in & one inst. out every cycle)
- Stall pipeline on long latency operations, e.g., divides, cache misses
- Handle exceptions in-order at commit point

How to prevent increased writeback latency from slowing down single cycle integer operations? **Bypassing**

In-Order Superscalar Pipeline



- ▶ Fetch two instructions per cycle; issue both simultaneously if one is integer/memory and other is floating point
- ▶ Inexpensive way of increasing throughput, examples include Alpha 21064 (1992) & MIPS R5000 series (1996)
- ▶ Same idea can be extended to wider issue by duplicating functional units (e.g. 4-issue UltraSPARC & Alpha 21164) but regfile ports and bypassing costs grow quickly

Boosting Performance

- ▶ How do we keep more functional units busy?
- ▶ How can we do useful work during long latency operations?
- ▶ Allow instructions to execute **out-of-order (OOO)**.
- ▶ Possible, but what problems will it cause?
 - hazards avoidance becomes more complex!


Types of Data Hazards

Consider executing a sequence of

$$r_k \leftarrow r_i \text{ op } r_j$$


type of instructions

Data-dependence

$$\begin{array}{l} r_3 \leftarrow r_1 \text{ op } r_2 \\ r_5 \leftarrow r_3 \text{ op } r_4 \end{array}$$



Read-after-Write
(RAW) hazard

Anti-dependence

$$\begin{array}{l} r_3 \leftarrow r_1 \text{ op } r_2 \\ r_1 \leftarrow r_4 \text{ op } r_5 \end{array}$$


Write-after-Read
(WAR) hazard

Output-dependence

$$\begin{array}{l} r_3 \leftarrow r_1 \text{ op } r_2 \\ r_3 \leftarrow r_6 \text{ op } r_7 \end{array}$$


Write-after-Write
(WAW) hazard

Register vs. Memory Dependence

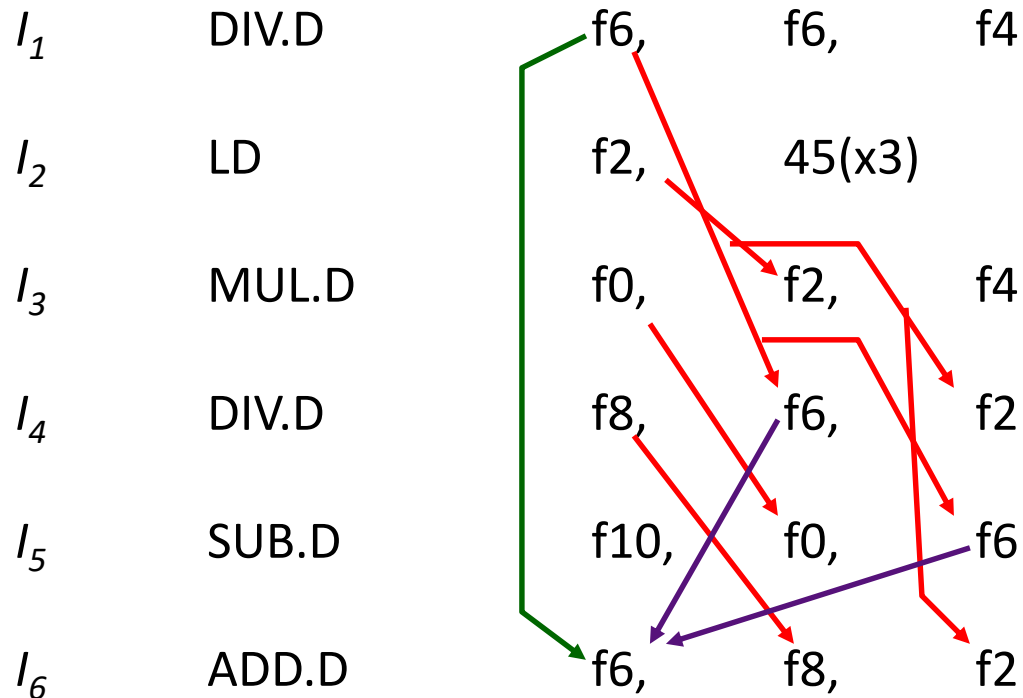
- ▶ Data hazards due to register operands can be determined at the decode stage, but data hazards due to memory operands can only be determined after computing the effective address

Store: $M[r1 + disp1] \leftarrow r2$

Load: $r3 \leftarrow M[r4 + disp2]$

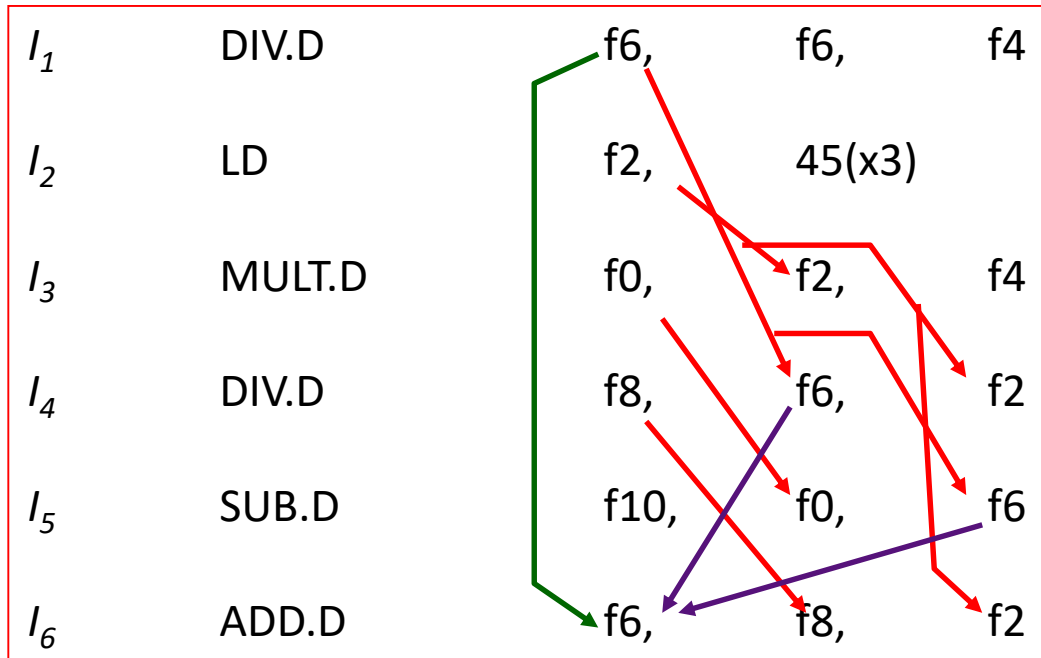
Does $(r1 + disp1) = (r4 + disp2)$?

Data Hazards: An Example



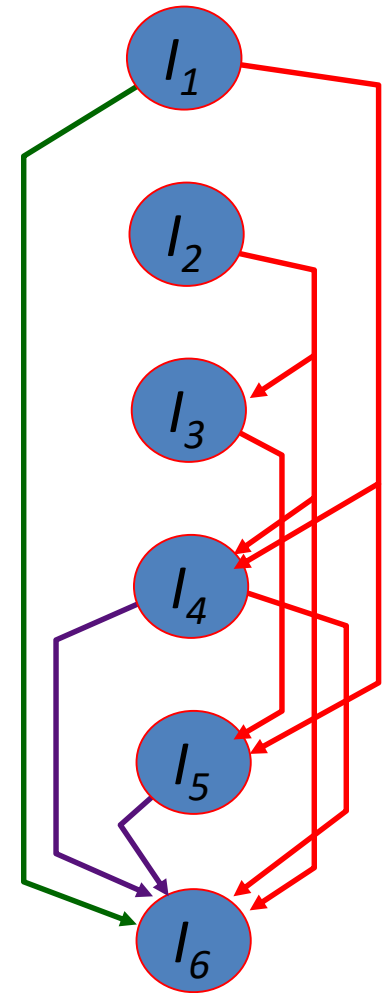
RAW Hazards
WAR Hazards
WAW Hazards

Instruction Scheduling



Valid orderings:

in-order	I_1	I_2	I_3	I_4	I_5	I_6
out-of-order	I_2	I_1	I_3	I_4	I_5	I_6
out-of-order	I_1	I_2	I_3	I_5	I_4	I_6



Out-of-order Completion *In-order Issue*

					Latency
I_1	DIV.D	f6,	f6,	f4	4
I_2	LD	f2,	45(x3)		1
I_3	MULT.D	f0,	f2,	f4	3
I_4	DIV.D	f8,	f6,	f2	4
I_5	SUB.D	f10,	f0,	f6	1
I_6	ADD.D	f6,	f8,	f2	1

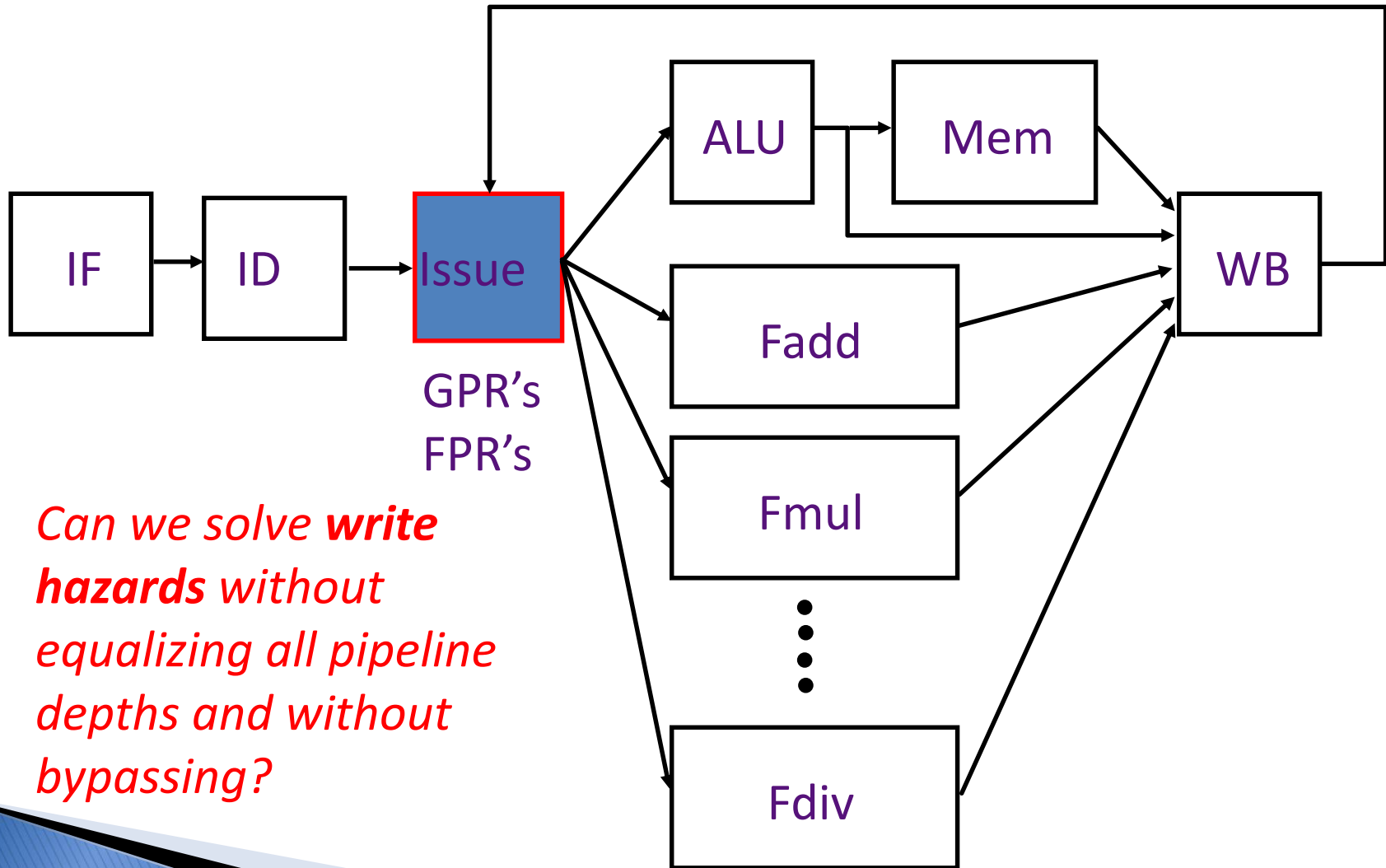
Completion
order has to
follow instr.
order

Completion
order doesn't
need s to
follow instr.
order

in-order comp 1 2 1 2 3 4 3 5 4 6 5 6

out-of-order comp 1 2 2 3 1 4 3 5 5 4 6 6

Complex Pipeline



*Can we solve **write hazards** without equalizing all pipeline depths and without bypassing?*

When is it Safe to Issue an Instruction?

Suppose a data structure keeps track of all the instructions in all the functional units

- ▶ The following checks need to be made before the Issue stage can dispatch an instruction
 - Is the required functional unit available?
 - Is the input data available? → RAW?
 - Is it safe to write the destination? → WAR? WAW?
 - Is there a structural conflict at the WB stage?

A Data Structure for Correct Issues

Keeps track of the status of Functional Units

<i>Unit Name</i>	<i>Busy</i>	<i>Op</i>	<i>Dest</i>	<i>Src1</i>	<i>Src2</i>
Int					
Mem					
Add1					
Add2					
Add3					
Mult1					
Mult2					
Div					

*The instruction **i** at the Issue stage consults this table*

FU available?

RAW?

WAR?

WAW?

check the busy column

search the dest column for i's sources

search the source columns for i's destination

search the dest column for i's destination

An entry is added to the table if no hazard is detected;

An entry is removed from the table after Write-Back