Exception handling in Pipelined Processors

Due to the overlapping of instruction execution, multiple interrupts can occur in the same clock cycle. Sources of interrupt in the MIPS are as follows:

F	Misaligned memory access, Protection violation, Page fault
D	Undefined opcode
X	Arithmetic overflow
M	Misaligned memory access Protection violation, page fault

The need for **precise interrupts** is an additional cause for complication.

Two simple cases

Example 1. I/O device interrupt

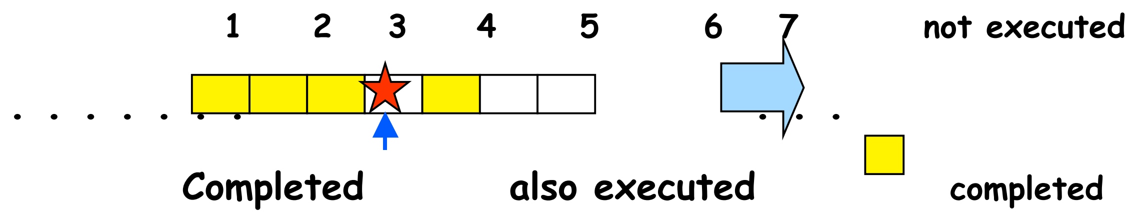
When the interrupt occurs, the current instruction completes, and the current context is saved. After the interrupt is serviced, the context is restored, and the execution resumes from the next instruction.

Example 2. Page fault / arithmetic overflow

The current instruction cannot be completed. So it is aborted, the exception is handled, and the execution resumes from the instruction causing the exception.

Precise vs. Imprecise Interrupts

To implement precise interrupts, the interrupt handler needs to create the illusion of sequential instruction execution.



The above scenario is feasible with some pipelined processors. If the interrupt is precise, then you must be able to draw a line between completed and unexecuted instructions.

To implement a *precise interrupt*,

- *Undo* all instructions after the interrupting instruction, and
- Restart from the faulting instruction.

Otherwise, the interrupt becomes imprecise.

Example of difficult cases

LW r4, X	F	D	X	M*	W	
ADD r1, r2, r3		F*	D	X	M	W

The Problem The second instruction interrupts first! If the second instruction is restarted first, and then the first instruction is restarted, then second instruction is executed twice!

A Solution Let the hardware post interrupts for each instruction. When instruction enters the W stage, check the interrupt flags, and handle the flags in instruction order.

An important rule

Interrupt handling is simplified if we can disable or defer all writes (that affect the state of the computation) until the outcome of all the previous instructions are satisfactorily resolved.

This avoids backtracking, shadow registers etc.

For such situations (and a few other situations too), rename registers provide a convenient solution.

Rename registers

Rename registers form a pool of registers that can be temporarily used to store results until the instruction is "committed". These can be useful in implementing precise interrupts.

Instruction 1 (completed)

Instruction 2 (completed)

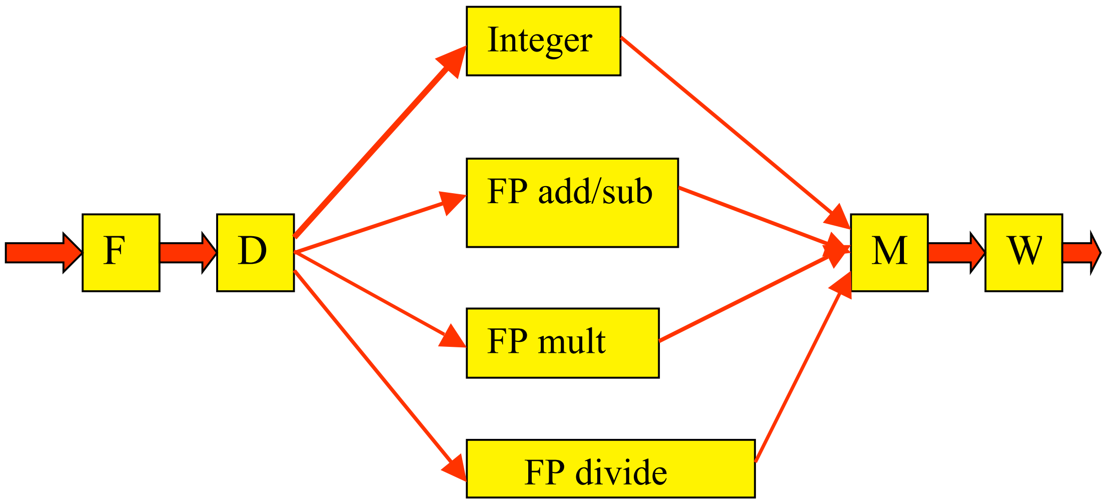
Instruction 3 (generates an interrupt)

Instruction 4 (completed, with result in rename register)

Instruction 5 (not executed yet)

The result of instruction 4 will be written into a rename register first. It is written into the final destination (i.e committed or graduated) after all previous instructions have completed their executions.

MIPS with multicycle FP units



Latency of a functional unit

Number of clock cycles needed to produce a result.

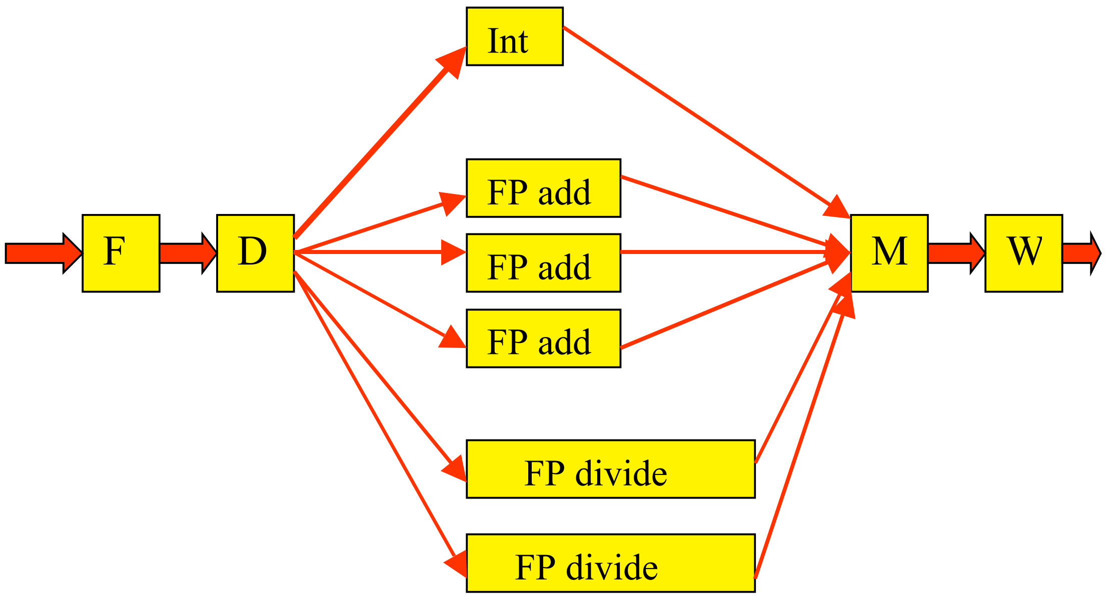
Repetition Rate (or Initiation Interval)

How often can the input operands be fed into a functional unit?

Improving Repetition Rate

Method 1

Using N copies of a functional unit, repetition rate can be improved by a factor of N.



Method 2

Use pipelined functional units.

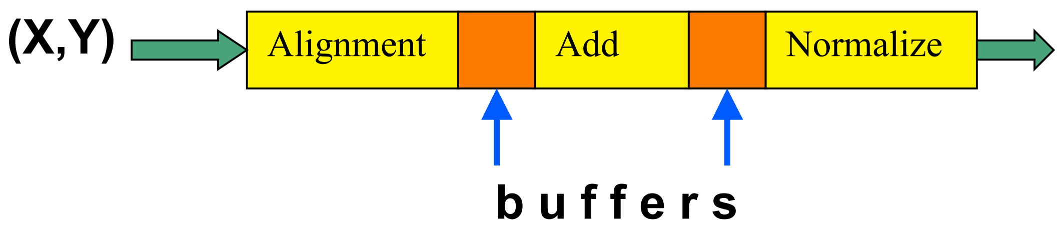
How can we reduce latency?

Pipelined Functional Units

$X = 3.0 \times 10^{17}$, $Y = 4.0 \times 10^{15}$

Compute $S = X+Y$

1. Alignment	$x = 300 \times 10^{15}$, $y = 4 \times 10^{15}$
2. Add	$S = 304 \times 10^{15}$
3. Normalize	$S = 3.04 \times 10^{17}$



A three-stage add pipeline

If each stage takes 1 clock cycle, then the repetition rate is reduced to 1 clock cycle.

Additional Problems with Multicycle Units

Problem 1. New type of structural hazards

Two instructions contend for the same functional unit.

Was not possible in the first version of MIPS!

	1	2	3	4	5	6	7
F0:=F4*F6	F	D	m	m	m	m	…
F4:=F8*F2		F	D	o	o	o	

Two instructions try to write into the register file at the same time.

	1	2	3	4	5	6	7
F2 := F4+F6	F	D	a	a	a	M	W
F10:= F4/F6		F	D	d	d	d	…
F8 := M[R2]			F	D	X	M	W

Note. The functional units are not pipelined here.

How will the first example change if the multiplier is pipelined?

Problem 2. Write-After-Write (WAW) Hazard

Instruction 1 F2:= F2/F4

Instruction 2 F2:=F8+F6

1:	F	D	d	d	d	d	d	d	M	W
2:		F	D	a	a	a	M	W		

The final value of F2 is incorrect!

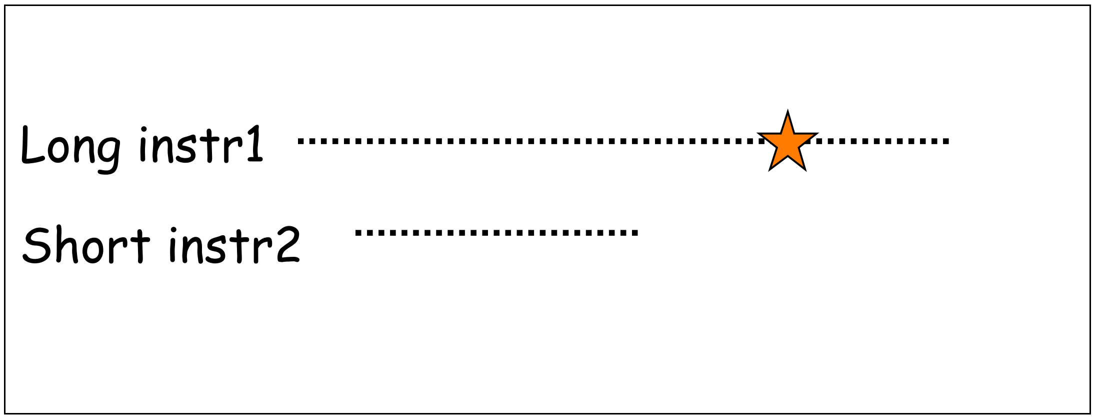
Possible solutions

1. Disable "W" of instruction 1.
2. Defer the completion of instruction 2.

As a general strategy, if instructions are retired in program order, then such problems are solved. Rename register helps.

Problem 3.

Are interrupts precise?



If 1 is restarted, then 2 is executed twice. To undo instruction 2, you must save the old value of F10.

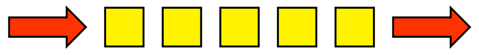
A solution is to retire instructions in program order. Save the result of 2 until 1 is completed.

Superscalar vs Superpipelined CPU

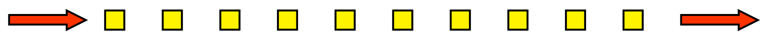
MIPS R 4000 is a 64-bit RISC.

It is a superpipelined processor with eight stages in its instruction pipeline.

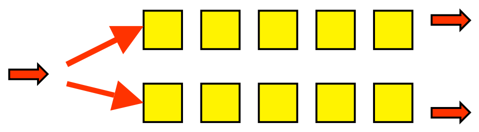
It keeps control unit simple, and maximizes clock speed. The compiler resolves hazards.



pipelined

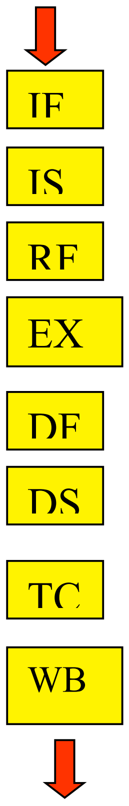


superpipelined (one long pipeline)

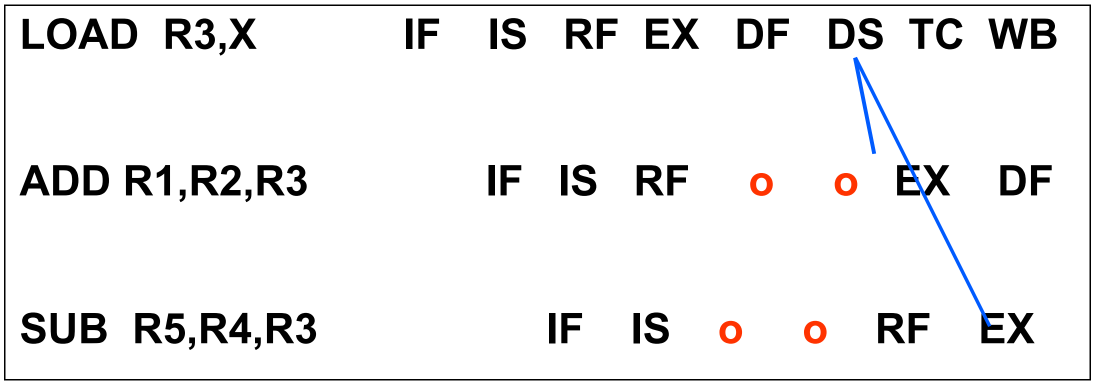


superscalar (multiple pipelines)

MIPS 4000 pipeline



IF	Instruction fetch (first half)
IS	Instruction fetch (second half)
RF	Register fetch (decode)
EX	Execute (integer & FP units)
DF	Data fetch (first half)
DS	Data fetch (second half)
TC	Tag check
WB	Write back



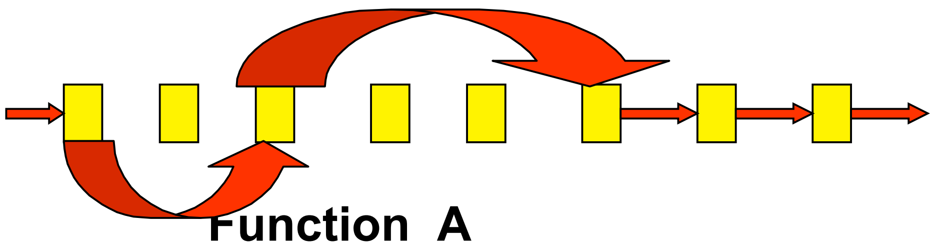
```
LOAD  R3,X          IF   IS   RF  EX   DF   DS  TC  WB

ADD R1,R2,R3             IF   IS   RF   o    o   EX   DF

SUB  R5,R4,R3               IF   IS   o    o   RF   EX
```

Two-cycle load delay

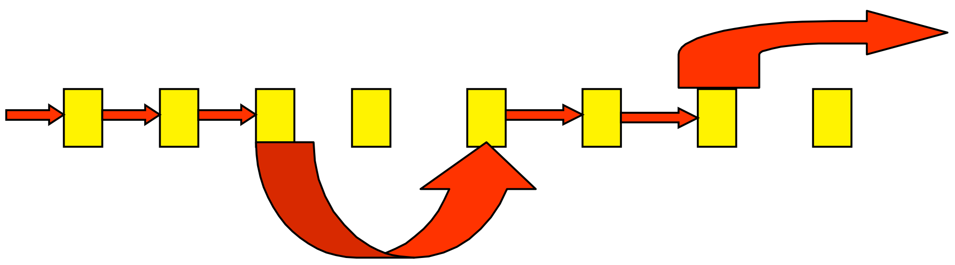
MIPS 4000 Multifunction Pipes

Unifunctional pipes. A separate pipe for each function like ADD MULT. Used in CRAY.

Multifuction pipes. Each function is implemented by reconfiguring some basic functional blocks at run time.



Function A



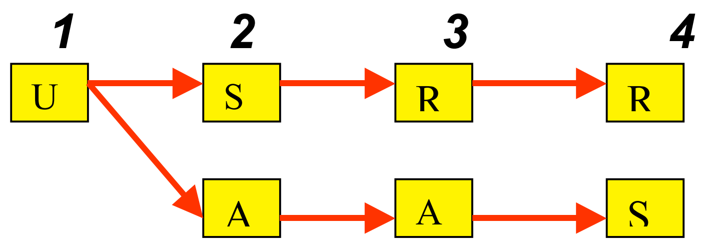
Function B

First used in Texas Instruments ASC

MIPS R4000 uses multifunction pipes.

Eight stages of FP pipeline

Stage	*Unit*	*Description*
A	FP add	Mantissa add
D	FP div	Divide
E	FP mul	Exception test
M	FP mul	Multiplier first stage
N	FP mul	Multiplier second stage
R	FP add	Rounding
S	FP add	Operand shift
U	all	Unpack FP numbers



ADD = U, S+A, A+R, R+S

Latency = 4 cycles

Initiation interval = 3 cycles

Structural hazards with FP stages

The following table examines when an ADD operation can be scheduled after a MUL operation without causing a structural hazard.

clock	1	2	3	4	5	6	7	8
MUL	U	M	M	M	M	N	N+A	R
ADD		U	S+A	A+R	R+S			
ADD			U	S+A	A+R	R+S		
ADD				U	S+A	A+R	R+S	
ADD (stall)						U	S+A	A*+R
ADD (stall)						U	S+A*	

Control unit resolves hazard by appropriately delaying certain stages. Verify that there is no hazard if a MUL follows an ADD.