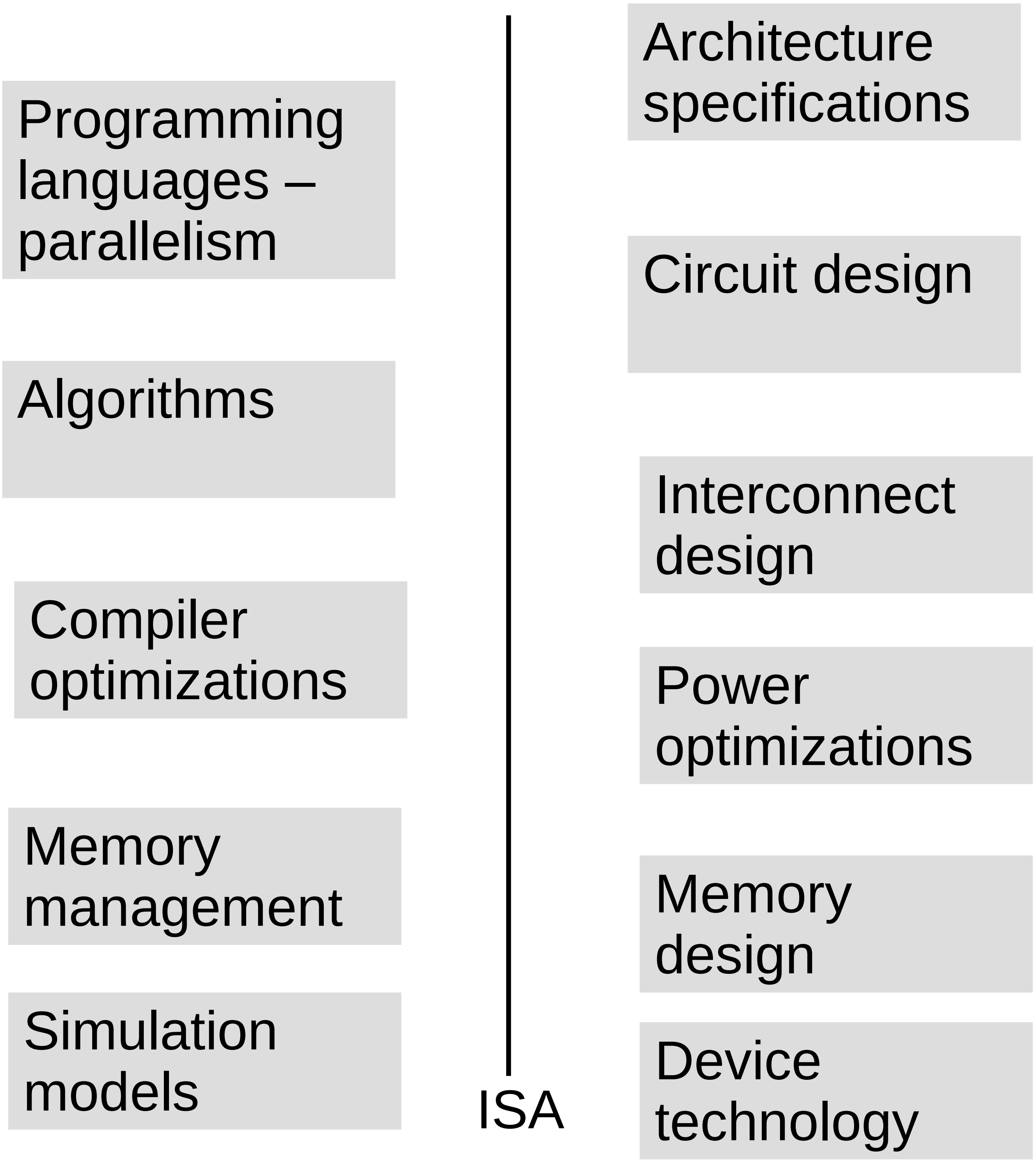
Advanced topics

- Forwarding unit design, Branch Predictors
- Superscalar processors/Dynamic scheduling - Algorithms for Out-of-Order (OOO) Execution
- Hardware of OOO – Re-order buffer and Reservation stations
- Multi-core processors, multiple issue processors
- Multithreading
- Parallelism in instructions: SIMD, VLIW
- Interrupts/Exceptions/I/O

- Virtual memory, Page tables, translation look aside buffer (TLB)
- Multi-core processors and cache hierarchy
- Cache coherence protocols in multi-core processors, consistency models
- Victim cache, banked caches
- Non-uniform Memory/Cache architectures
- Cache compression and compaction
- Prefetching using buffers
- Cache side channel attacks, security
- GPU/TPU architectures



Programming languages – parallelism

Algorithms

Compiler optimizations

Memory management

Simulation models

ISA

Architecture specifications

Circuit design

Interconnect design

Power optimizations

Memory design

Device technology

Processor pipeline

Exceptions

Branch/Jump.

- Control – hardest to design. Control hazard – tough to resolve
- Exception and interrupts - Another form of control hazard
- Exception - any unexpected change in control flow caused by internal events

  Overflow. , invalid opcode.
- Interrupt- triggered by an external event, can be asynchronous

Exceptions

Divide by zero

Num / 0

Errors

Pc

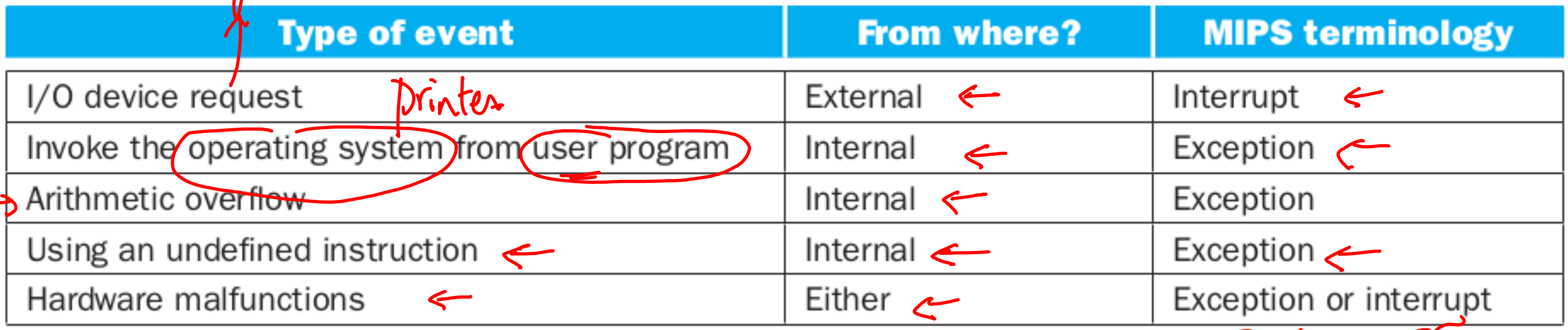
Instruction

→ Invalid opcode

→ LW → < 32bit >

- Control – hardest to design. Control hazard – tough to resolve
- Exception and interrupts - Another form of control hazard
- Exception - any unexpected change in control flow caused by internal events
- Interrupt- triggered by an external event, can be asynchronous

Processor



Type of event	From where?	MIPS terminology
I/O device request	External	Interrupt
Invoke the operating system from user program	Internal	Exception
Arithmetic overflow	Internal	Exception
Using an undefined instruction	Internal	Exception
Hardware malfunctions	Either	Exception or interrupt

Printer

Computer organization and design- Henessey and Patterson

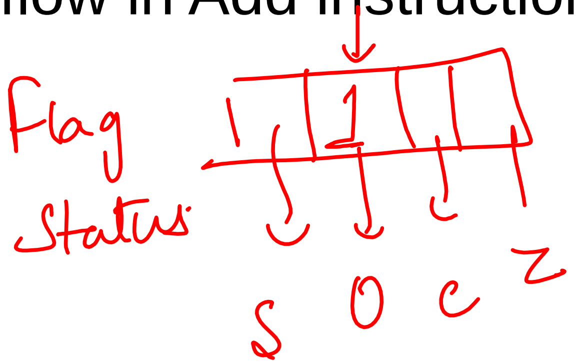
Unexpected

Assume arithmetic overflow in Add instruction

add $1, $2, $1;

sw $3, 400($1);

add $5, $1, $2;



Assume Divide by 0 error in the Div instruction

Div $1, $2, $1; ZERO

sw $3, 400($1);

add $5, $1, $2;

```
add $1, $2, $1;          arithmetic overflow
sw   $3, 400($1);
add $5,  $1, $2;
```

Invalid $1 causes error in the following instructions

Exceptions at different stages

Instr.

PC address

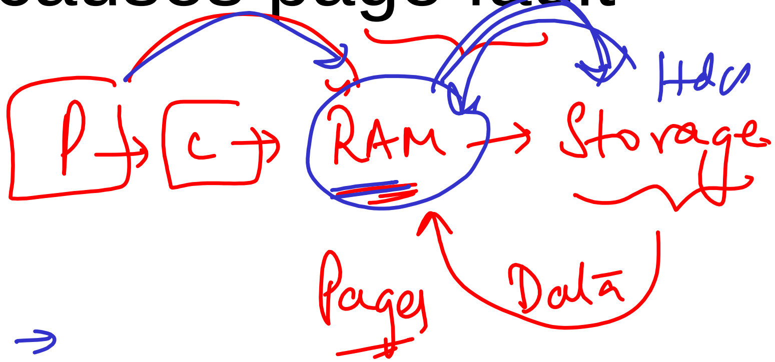
Word aligned
address.
Virtual
mem
Phy mem
16GB

100 program

- IF:
  - fetching instruction from memory causes page fault
  - Misaligned memory access

Page

4kB, 16kB



- ID: illegal opcode
- EX: division by 0, overflow

ALU →

- MEM: fetching data causes page fault; illegal address
- WB: no exceptions

data

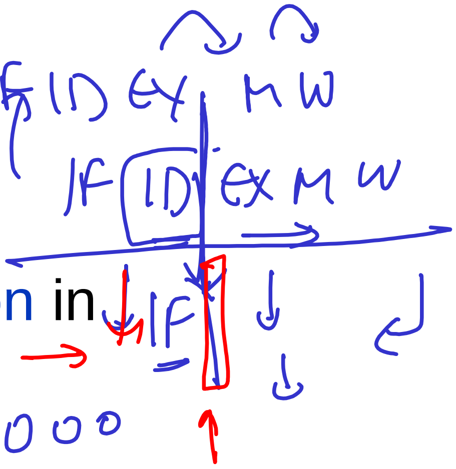
RISC-V/MIPS

5-bit → 32 reg

⇒ 6-bit → 64 regs

32-reg

Exception handling



Ileg Invalid opcode

In The Hardware

- The pipeline has to stop executing the offending instruction in midstream,
- let all preceding instructions complete,
- flush all succeeding instructions, / pipeline reg.
- set a register to show the cause of the exception,
- save the address of the offending instruction, and
- then jump to a predefined address/vectored interrupt (address of the <exception handler code)>

LW· 011000

000100

EPC Registers

→ Add ① PC / Instr

② - Cause.

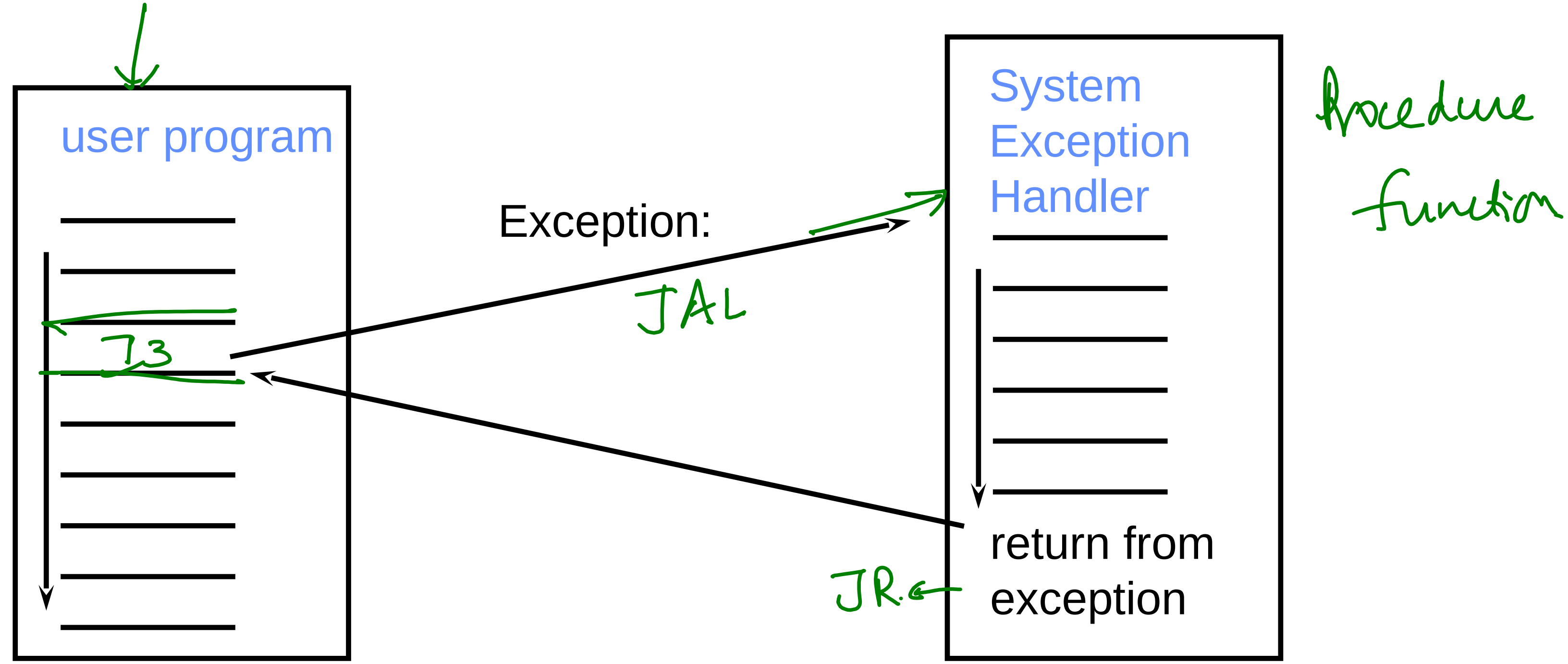
4 exc Table.

00	÷ 0	0011H
01	opcode	0022H
10		

In The Software

The software (OS) looks at the cause of the exception and deals with it.

- OS kills the program or resumes the instruction
- Depends on processor implementation and ISA



user program

Exception:

System Exception Handler

return from exception

Context / PC / Reg

- Save the address of the offending instruction
- Save any other information needed to return back

MIPS support

EPC register = exception program counter – 32 bit

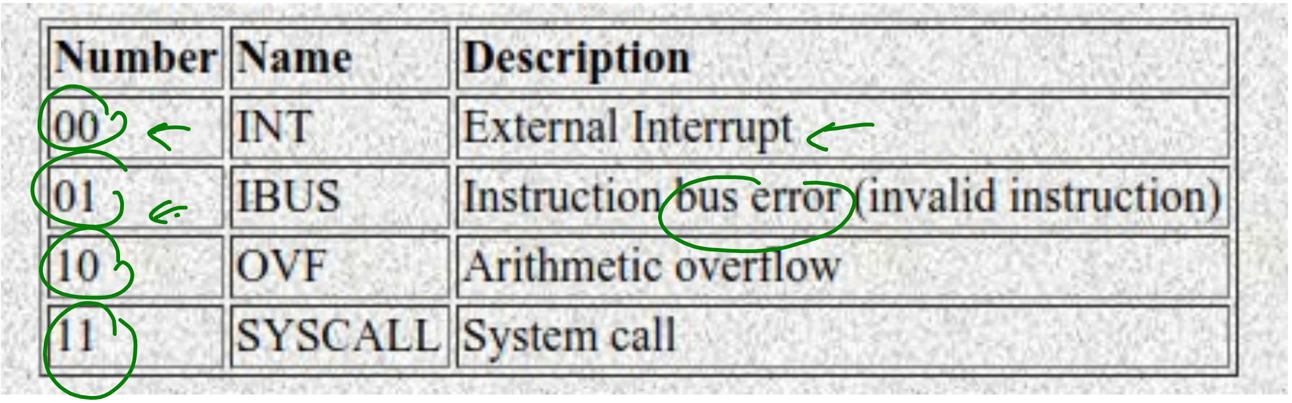
contains address of instruction that caused the exception

We need to record what caused an exception

→ 1. Cause register = 32-bit status register used to record cause of exception. (some bits used)

2. Vectored interrupt: OS can determine the cause based on the address

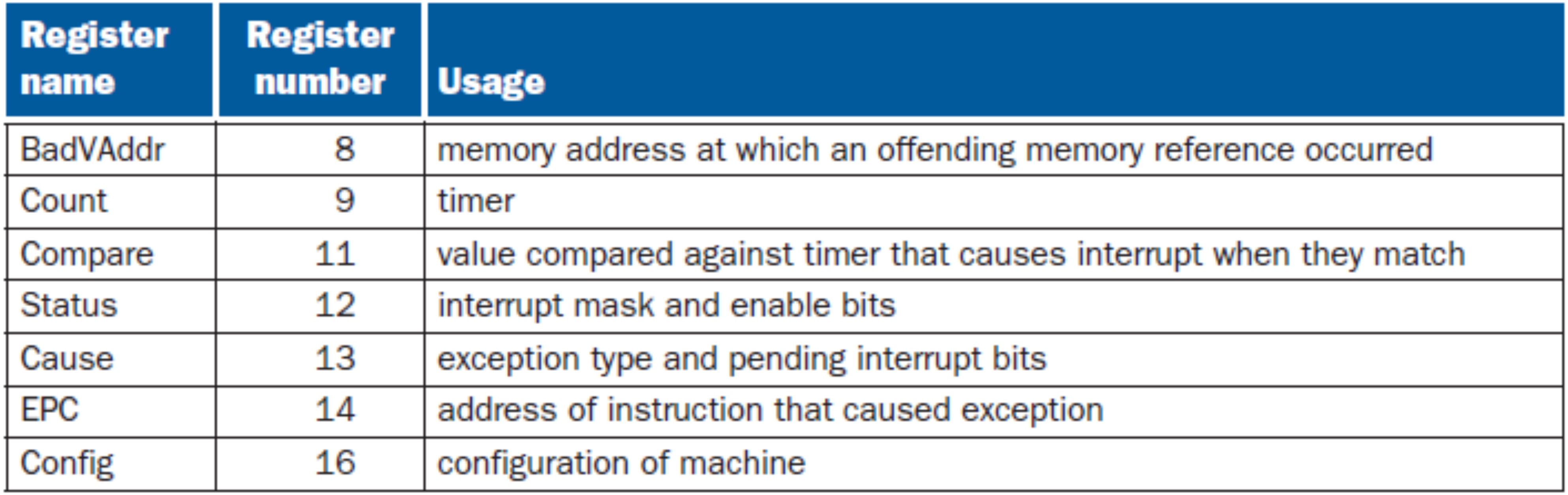
pc → 0x00 lw.



Number	Name	Description
00	INT	External Interrupt
01	IBUS	Instruction bus error (invalid instruction)
10	OVF	Arithmetic overflow
11	SYSCALL	System call

Co-processor for MIPS

- Contains registers useful for handling exceptions
- Not accessible in **user mode**. Available only in Kernel mode
- Includes the status register, cause register, BadVaddr, and EPC (Exception Program Counter).

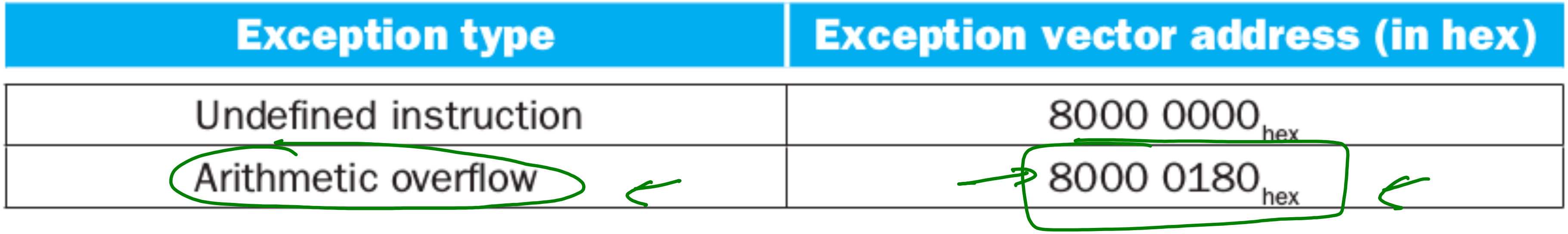


Register name	Register number	Usage
BadVAddr	8	memory address at which an offending memory reference occurred
Count	9	timer
Compare	11	value compared against timer that causes interrupt when they match
Status	12	interrupt mask and enable bits
Cause	13	exception type and pending interrupt bits
EPC	14	address of instruction that caused exception
Config	16	configuration of machine

MIPS support

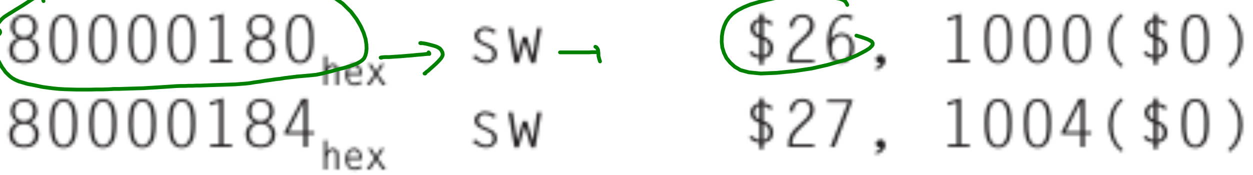
- Control signals to write EPC , Cause, and any other Status registers
- Write exception address into EPC, increase PC mux input lines to set exception address (MIPS uses 8000 00180$_{\text{hex}}$).
- Undo PC = PC + 4, since want EPC to point to offending instruction (not PC+4)
  - So, do PC = PC – 4
- Flush all succeeding instructions

MIPS support – for vectored interrupts



Exception type	Exception vector address (in hex)
Undefined instruction	8000 0000_{hex}
Arithmetic overflow	8000 0180_{hex}

IF ID EX.

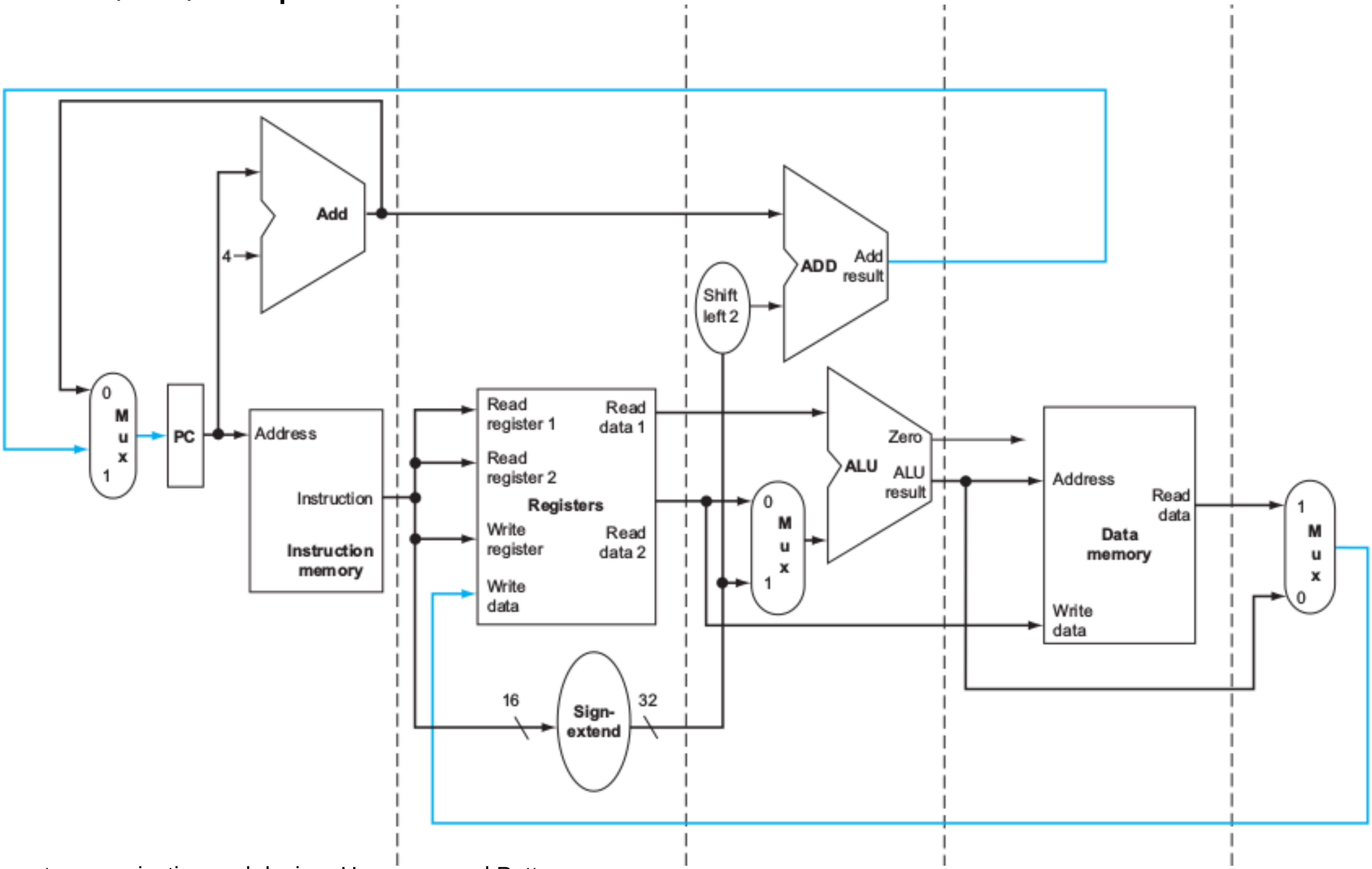


```
80000180hex    sw     $26, 1000($0)
80000184hex    sw     $27, 1004($0)
```

JR

Computer organization and design- Henessey and Patterson

```
lw $1, 4($3)
add $2, $3, $4          // causes overflow – when is it detected
or $3, $1, $2
bne $1, $3, Loop
```



Computer organization and design- Henessey and Patterson

lw $1, 4($3)

add $2, $3, $4 → // causes overflow – detect in EX stage

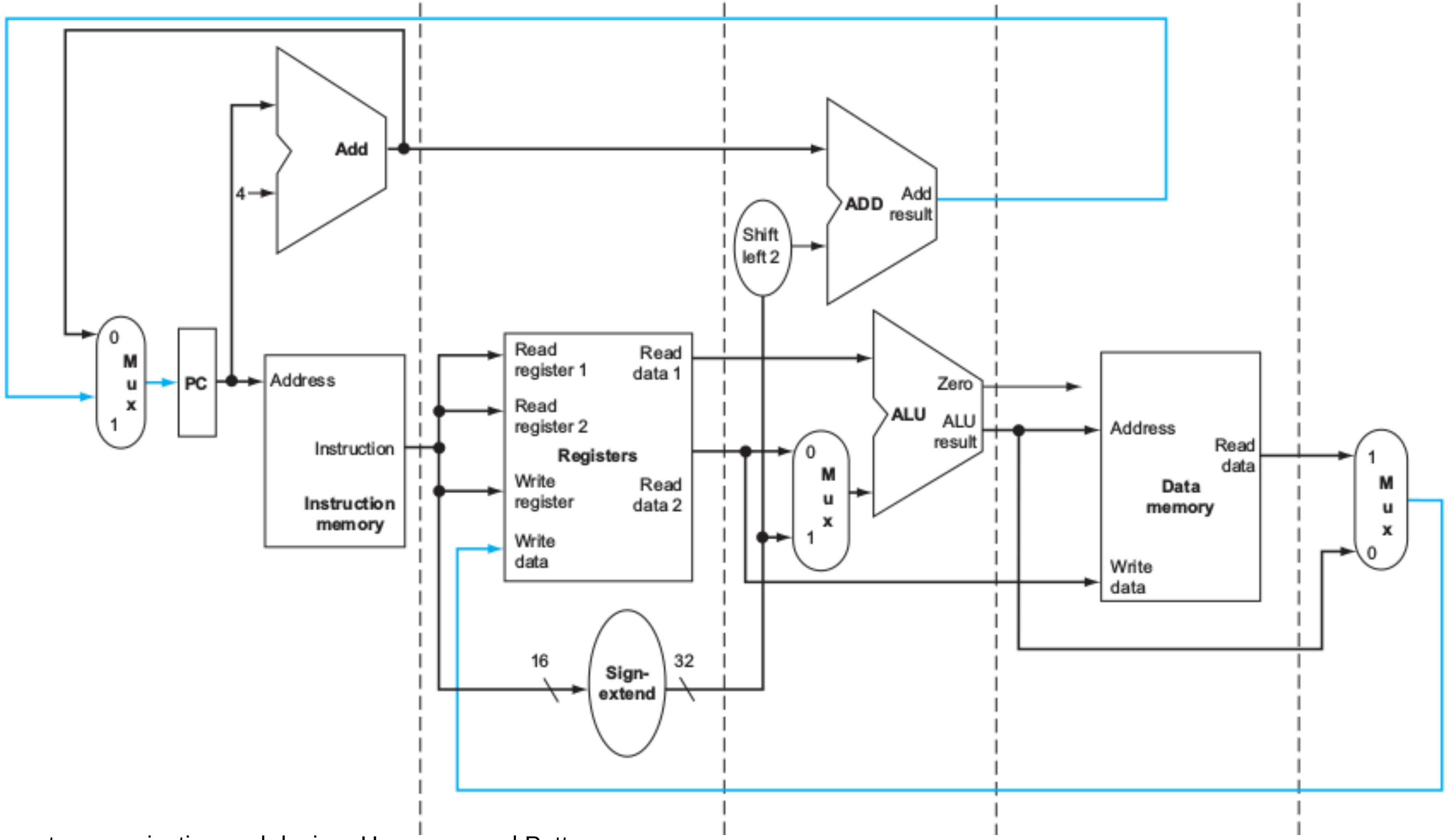
or $3, $1, $2 ← Pipeline reg.

bne $1, $3, L ←

IH – ESR

~~BNE~~ ~~Or~~ Add LW

Mem WB



Computer organization and design- Henessey and Patterson

lw $1, 4($3)

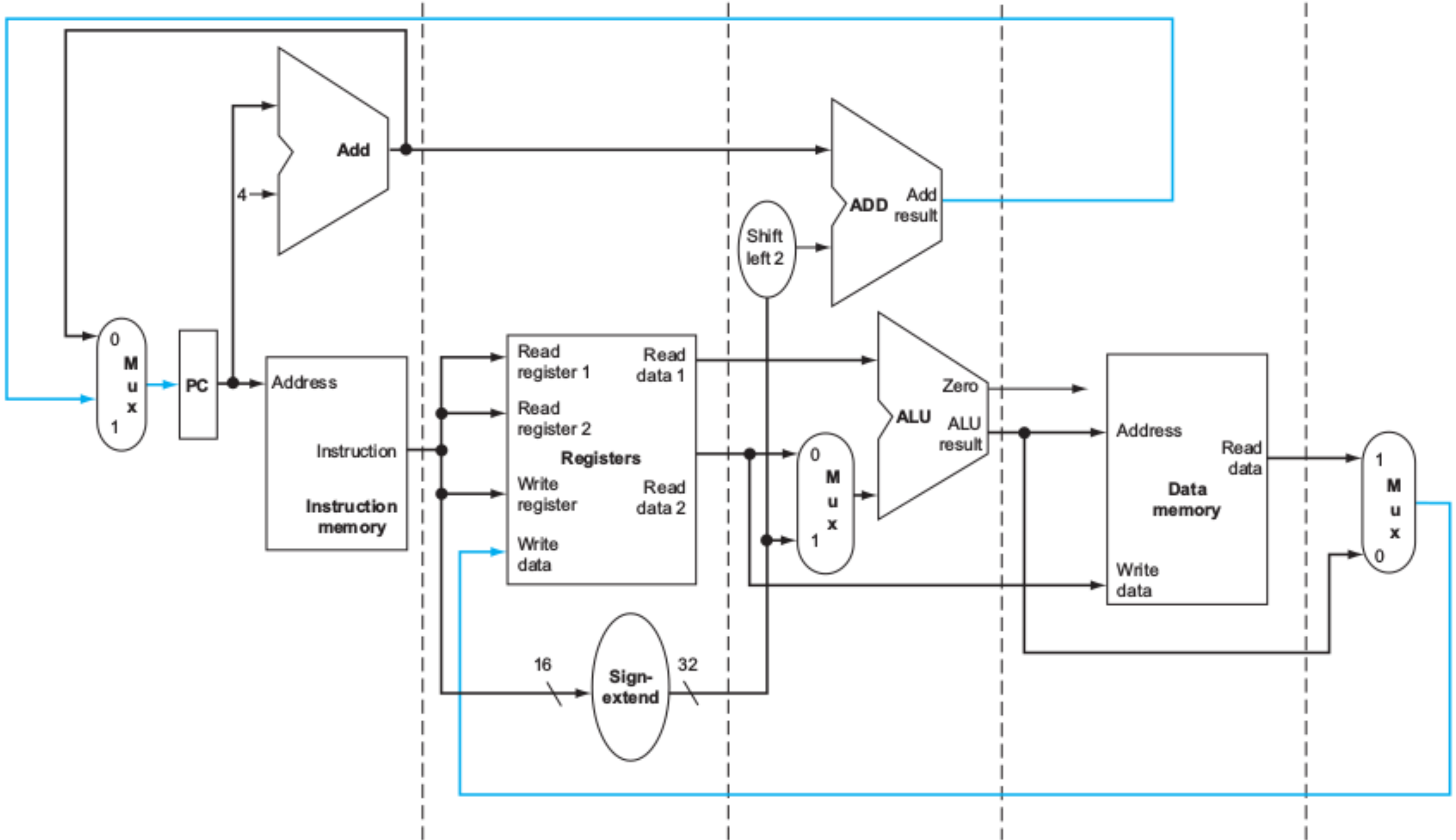
add $2, $3, $4 // causes overflow – detect in EX stage

or $3, $1, $2

bne $1, $3, L

Nop Nop Nop LW

ISR/ESR – SW instr



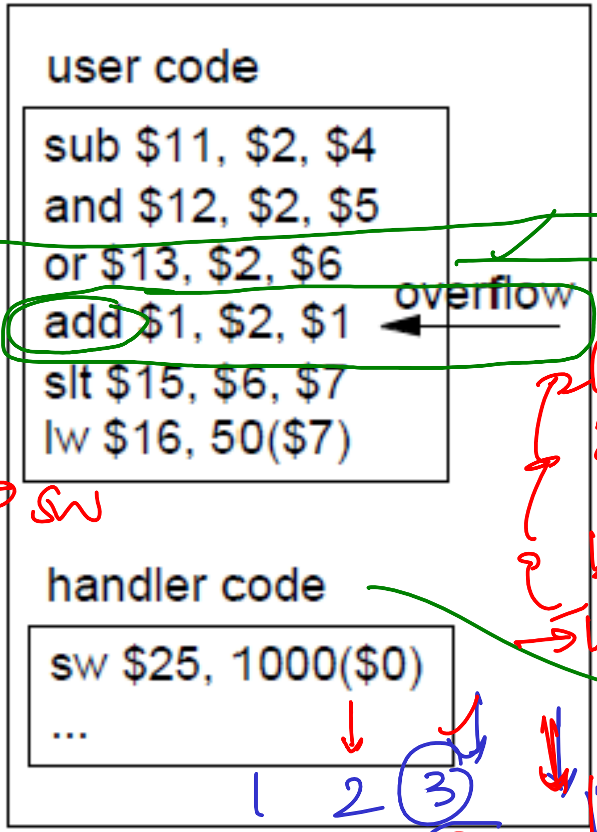
Computer organization and design- Henessey and Patterson

EPC = 4.

Cause.

Current PC =

Timing diagram?


user code

sub $11, $2, $4
and $12, $2, $5
or $13, $2, $6
add $1, $2, $1
slt $15, $6, $7
lw $16, 50($7)

overflow

handler code

sw $25, 1000($0)
...

PC+4 → SW

Arn

1 2 3 4 W

0 → OR IF ID EX M W

4 → Add → IF ID EX Nop Nop

8 PC+4 SLT → IF ID ~~EX~~ ~~M~~ ~~W~~

12 PC+4 LW → IF ~~ID~~ ~~EX~~ ~~M~~ ~~W~~

16 PC → SW

SW

IF - - - .

1 2 3

→ LW IF ID EX M

I Add → IF ID = EX

LW → IF 2 ID

PC ⇒ SW → → IF

Add IF ID EX

Sub IF ID

→ Controller

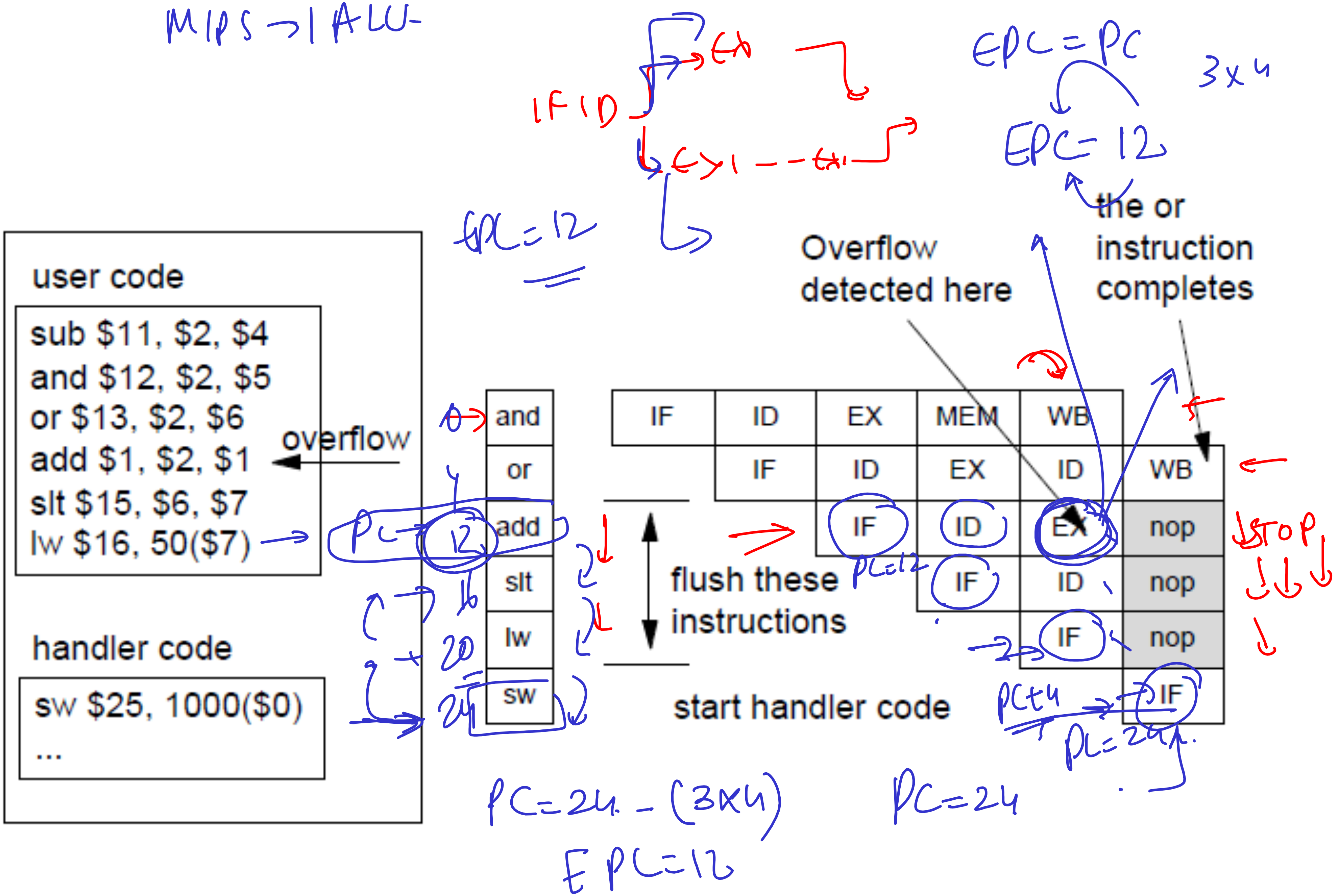
EPC

- Phase.
- Stalls.

1 stall

3 instr

MIPS →| ALU-



user code

```
sub $11, $2, $4
and $12, $2, $5
or $13, $2, $6
add $1, $2, $1
slt $15, $6, $7
lw $16, 50($7)
```

overflow

handler code

```
sw $25, 1000($0)
...
```

Overflow detected here

the or instruction completes

and	IF	ID	EX	MEM	WB	
or		IF	ID	EX	ID	WB
add			IF	ID	EX	nop
slt				IF	ID	nop
lw					IF	nop
sw						IF

flush these instructions

start handler code

Precise and imprecise exceptions

Precise --> If the pipeline can be stopped so that

- instructions just before the faulting instruction are completed

- the faulting (and future) instruction can be ~~restarted~~ stopped without altering the machine state

If it is an overflow --> restart from next instruction

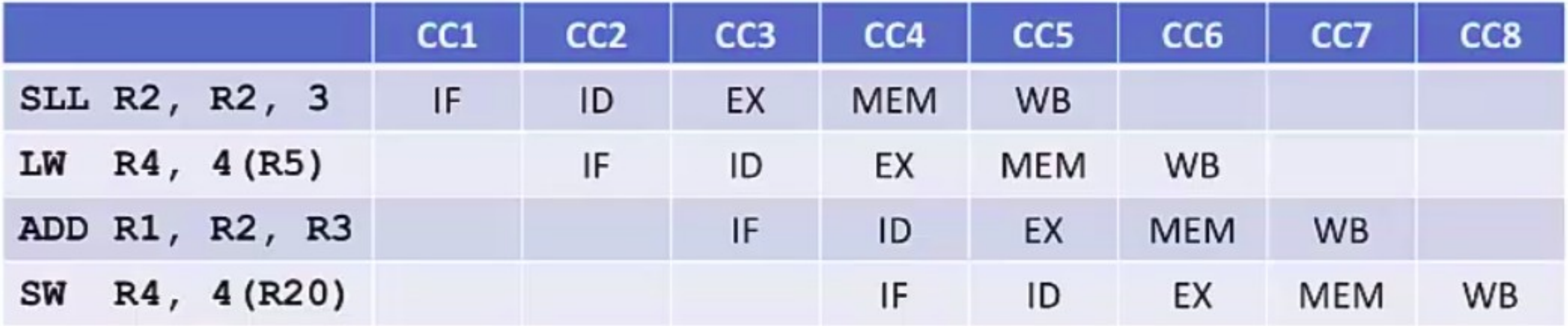
Add
↓
ESR

ESR:

Add. → IF ID EX1 .. EX2 - EX3 M W ←

SW → IF ID EX M W ←

Out of order completion or floating point pipelines where future instruction has already completed --> imprecise



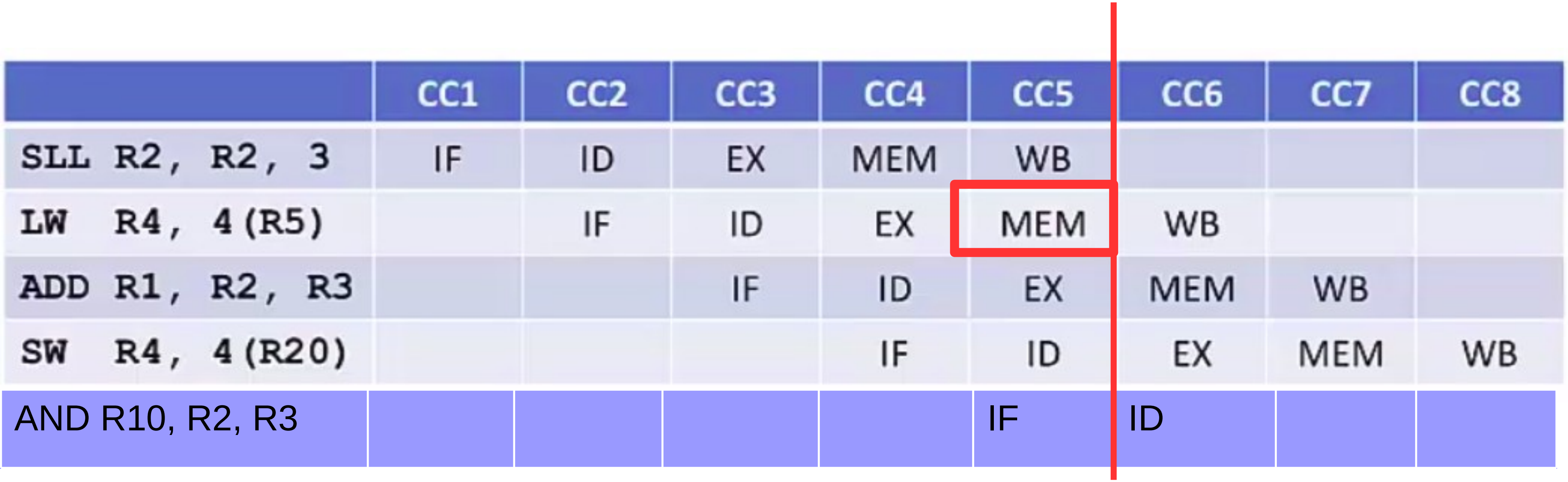
	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8
SLL R2, R2, 3	IF	ID	EX	MEM	WB			
LW R4, 4(R5)		IF	ID	EX	MEM	WB		
ADD R1, R2, R3			IF	ID	EX	MEM	WB	
SW R4, 4(R20)				IF	ID	EX	MEM	WB
AND R10, R2, R3					IF			

Suppose that LW has a misaligned address (not aligned on the word boundary)

When is it detected?

What should happen next?

Which pipeline registers to clear?

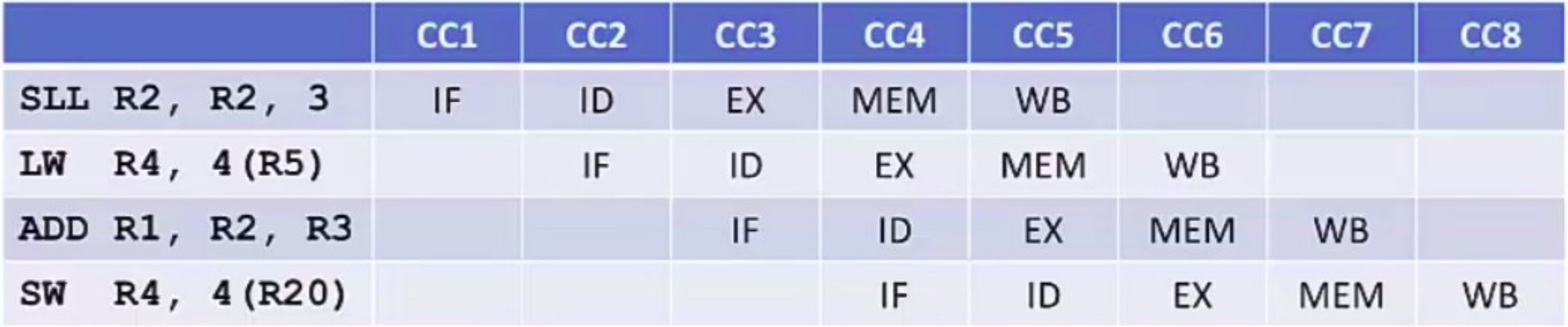


	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8
SLL R2, R2, 3	IF	ID	EX	MEM	WB			
LW R4, 4(R5)		IF	ID	EX	MEM	WB		
ADD R1, R2, R3			IF	ID	EX	MEM	WB	
SW R4, 4(R20)				IF	ID	EX	MEM	WB
AND R10, R2, R3					IF	ID		

Detected in EX stage – after computing the address?
Detected in MEM – while accessing memory?

IF/ID, ID/EX, EX/MEM to be cleared

Save PC --> EPC
Assume AND is in IF and has not written PC to PC+4
What should be EPC?



	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8
SLL R2, R2, 3	IF	ID	EX	MEM	WB			
LW R4, 4(R5)		IF	ID	EX	MEM	WB		
ADD R1, R2, R3			IF	ID	EX	MEM	WB	
SW R4, 4(R20)				IF	ID	EX	MEM	WB
AND R10, R2, R3					IF	ID		

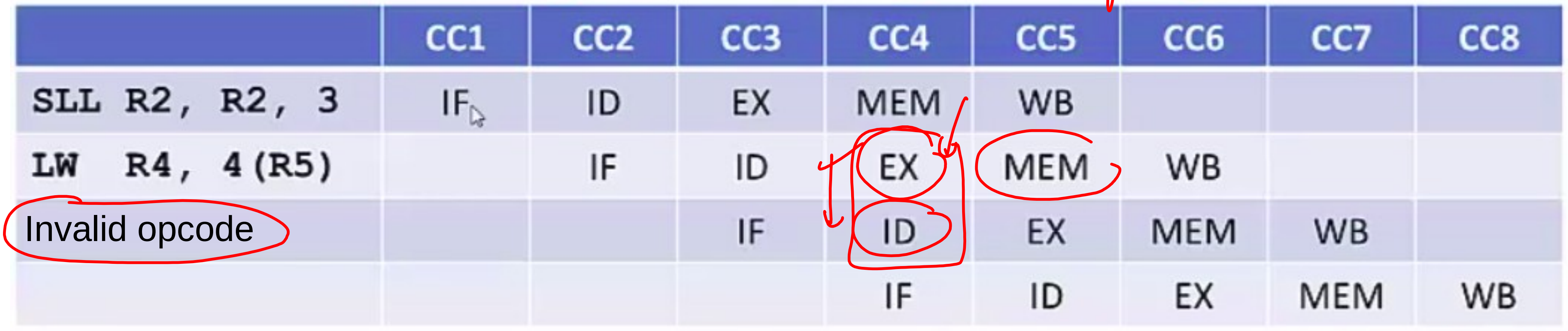
AND is in IF and has not written PC to PC+4
PC is pointing to AND

Instruction causing exception is PC – C or PC-12 --> EPC
Depends on the stage in which LW causes exception

Now make PC get the ISR address

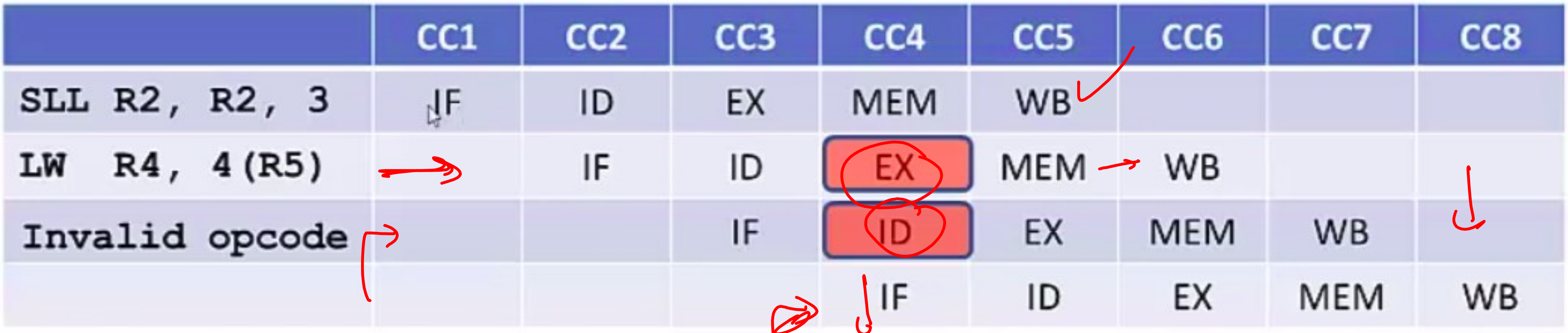
What if ADD was BEQ?

Misaligned mem addr



	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8
SLL R2, R2, 3	IF	ID	EX	MEM	WB			
LW R4, 4(R5)		IF	ID	EX	MEM	WB		
Invalid opcode			IF	ID	EX	MEM	WB	
				IF	ID	EX	MEM	WB

Multiple exceptions in the same clock cycle
LW – misaligned memory access
Next instruction is invalid opcode



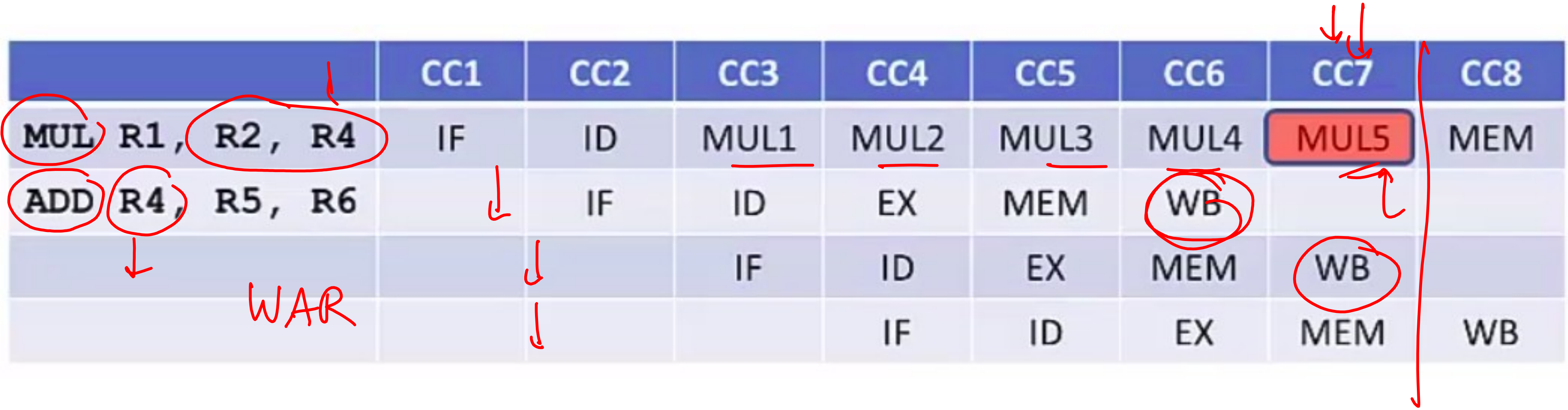
	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8
SLL R2, R2, 3	IF	ID	EX	MEM	WB			
LW R4, 4(R5)		IF	ID	EX	MEM	WB		
Invalid opcode			IF	ID	EX	MEM	WB	
				IF	ID	EX	MEM	WB

ESR →

Multiple exceptions in the same clock cycle

1st instruction takes precedence

Complex hardware



	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8
MUL R1, R2, R4	IF	ID	MUL1	MUL2	MUL3	MUL4	MUL5	MEM
ADD R4, R5, R6		IF	ID	EX	MEM	WB		
			IF	ID	EX	MEM	WB	
				IF	ID	EX	MEM	WB

WAR

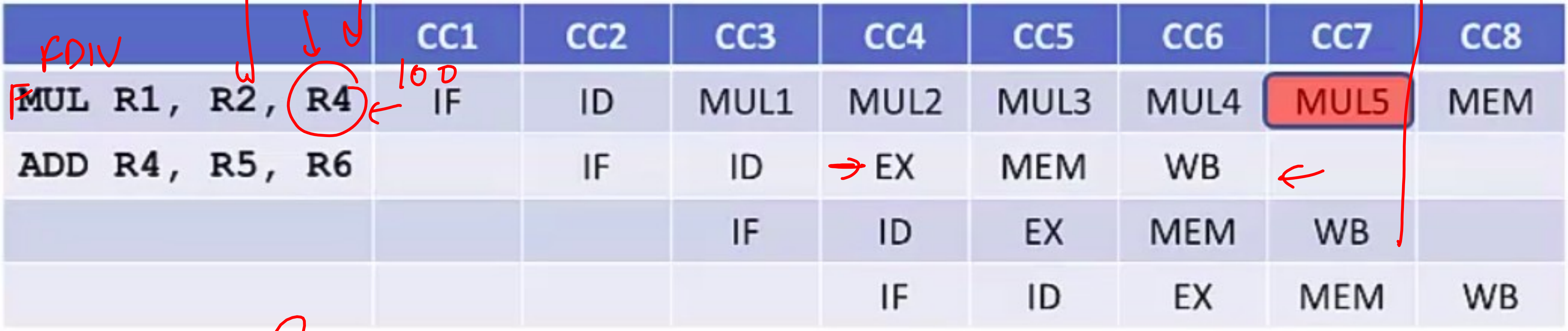
Out of order completion

Multiple clock cycle execution

MUL --> overflow

80.

80 81 82 83



	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8
MUL R1, R2, R4	IF	ID	MUL1	MUL2	MUL3	MUL4	MUL5	MEM
ADD R4, R5, R6		IF	ID	EX	MEM	WB		
			IF	ID	EX	MEM	WB	
				IF	ID	EX	MEM	WB

FDIV

100

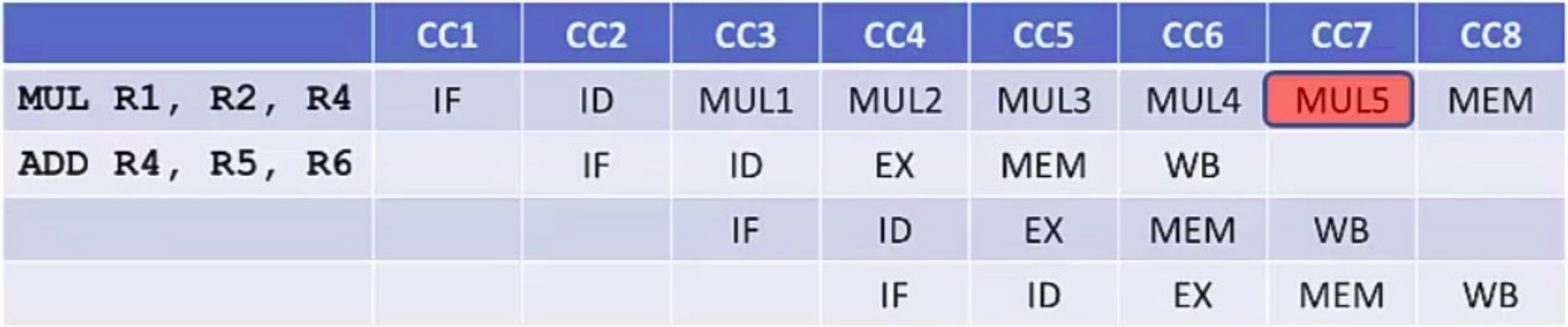
Snapshots Checkpoints Rollback

Add should not even have executed as per the exception rules!

Now, ADD has finished and exited the pipeline and also overwritten R4

Cannot even find out which value of R4 caused exception in MUL

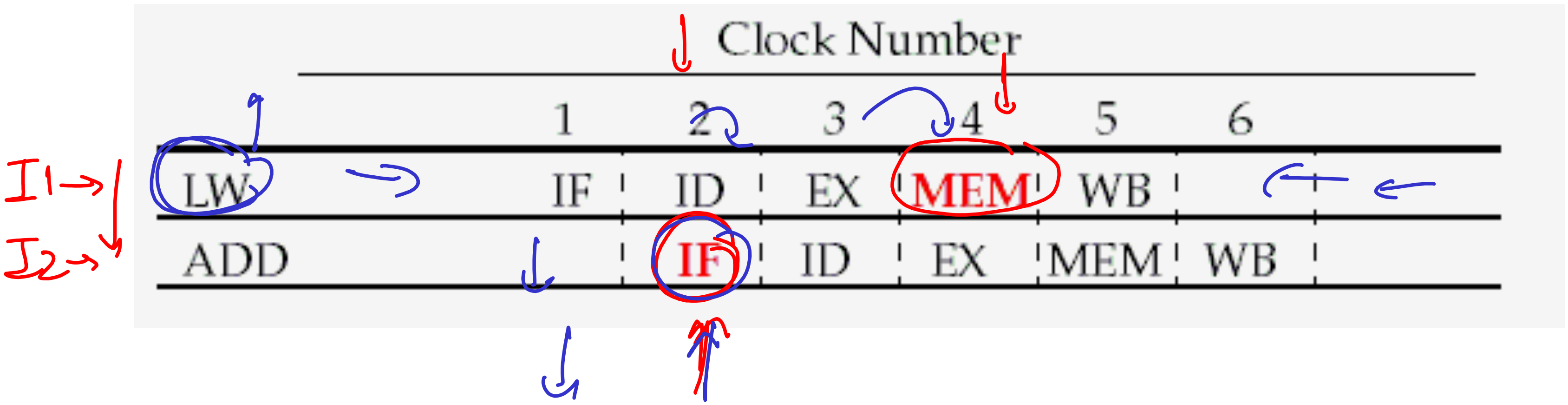
Imprecise exception



	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8
MUL R1, R2, R4	IF	ID	MUL1	MUL2	MUL3	MUL4	MUL5	MEM
ADD R4, R5, R6		IF	ID	EX	MEM	WB		
			IF	ID	EX	MEM	WB	
				IF	ID	EX	MEM	WB

How can we solve this?
Need to roll back architectural status or machine state
to prior to MUL and restore R4

Flush MUL and ADD



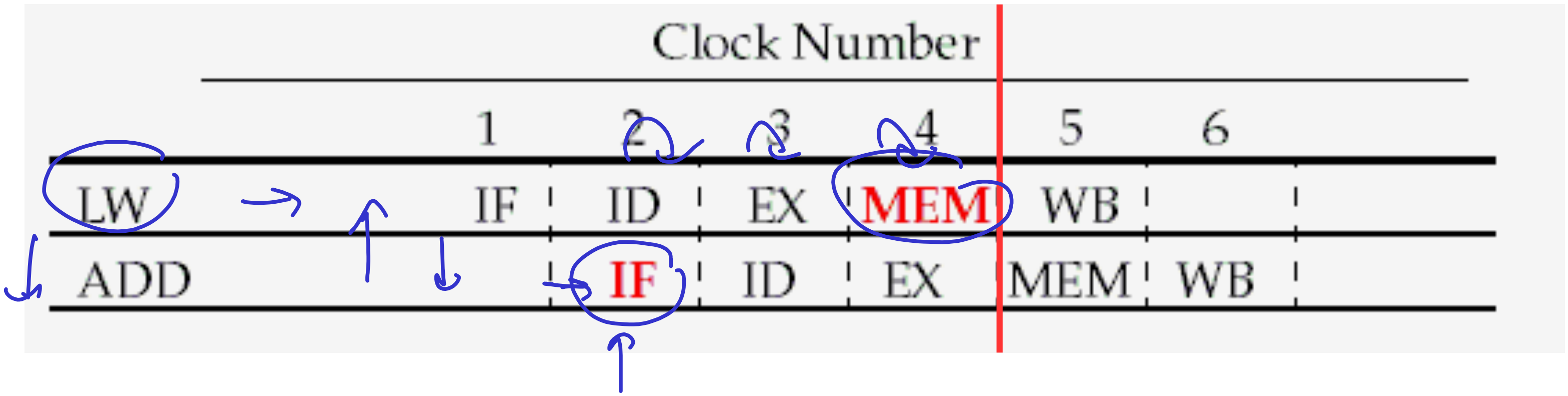
Clock Number

		1	2	3	4	5	6
I1 →	LW	IF	ID	EX	MEM	WB	
I2 →	ADD		IF	ID	EX	MEM	WB

ADD instruction page fault occurs before (in time) the LW page fault.

We must finish the LW before handling the ADD page fault (if we are implementing precise exceptions.)

We would then detect the LW's exception first and resolve it



Clock Number

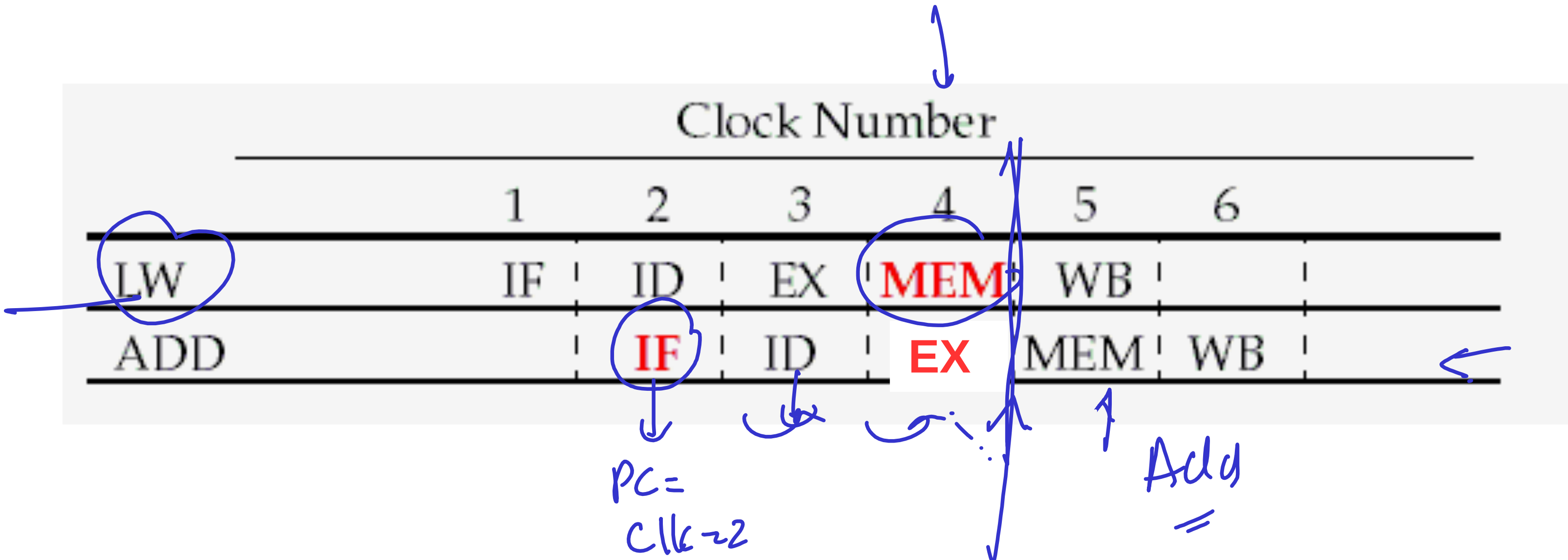
	1	2	3	4	5	6
LW	IF	ID	EX	MEM	WB	
ADD		IF	ID	EX	MEM	WB

Wait to handle an exception until a “last” point --> well defined point in the pipeline after which the machine state changes

--> such as write back or the end of the memory stage

Set an exception field in the pipeline stage and move ahead

Memory stage will look at this field to decide which instruction should be the precise exception point



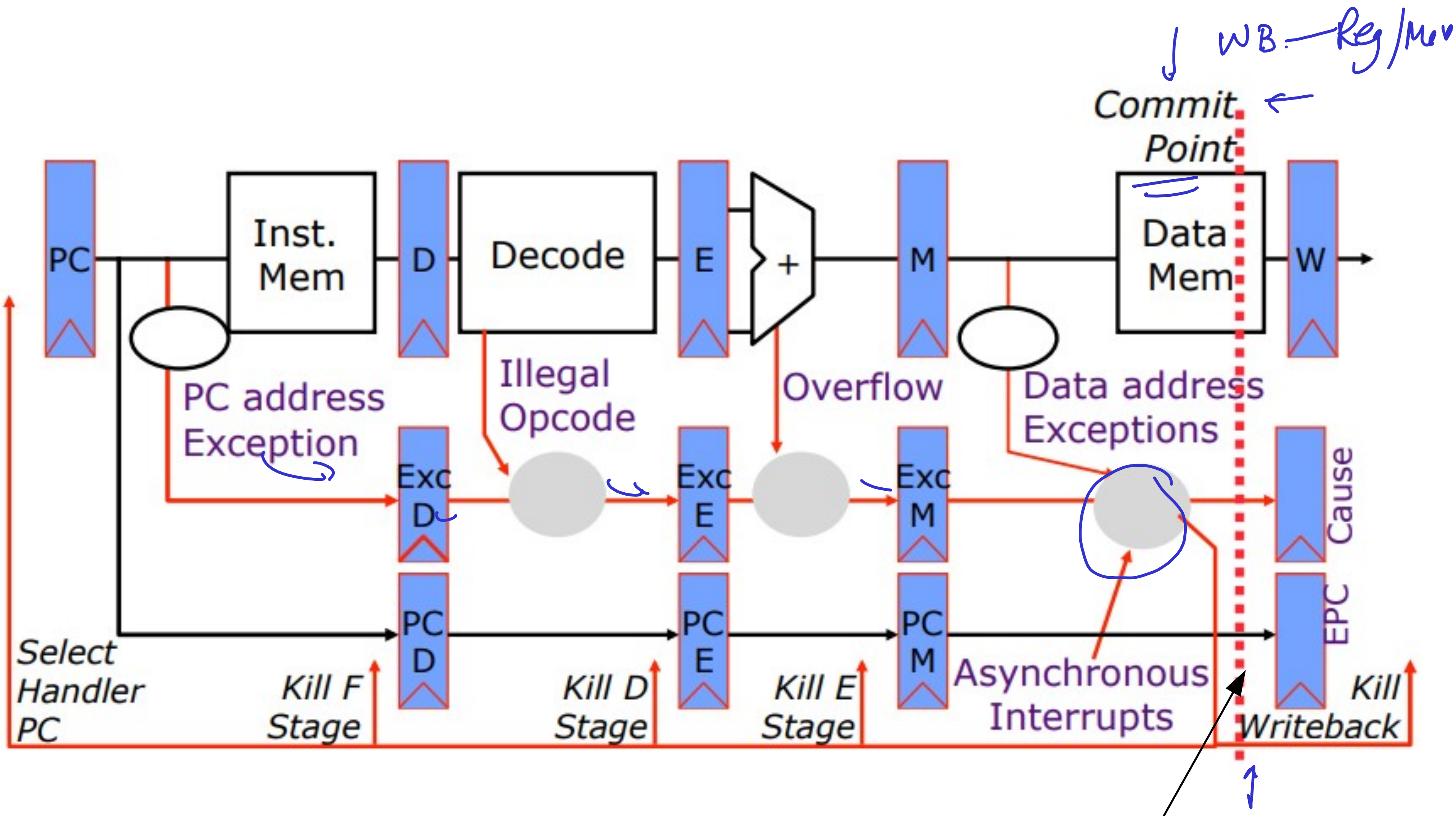
	Clock Number					
	1	2	3	4	5	6
LW	IF	ID	EX	MEM	WB	
ADD		IF	ID	EX	MEM	WB

PC=
Clk=2

Add

Imprecise exception --> Wait till MEM stage

Add --> Make a note of the IF exception, but dont resolve it until a certain point, until we are sure there are no previous exceptions



If exception at commit point: update Cause and EPC registers

Taken from CS152, UCB

MIPS support

- Additional instructions:
- mfc0 = instruction to put EPC into one of general-purpose regs. E.g. mfc0 $s1, $epc

so that we can return from exception handler using jr.

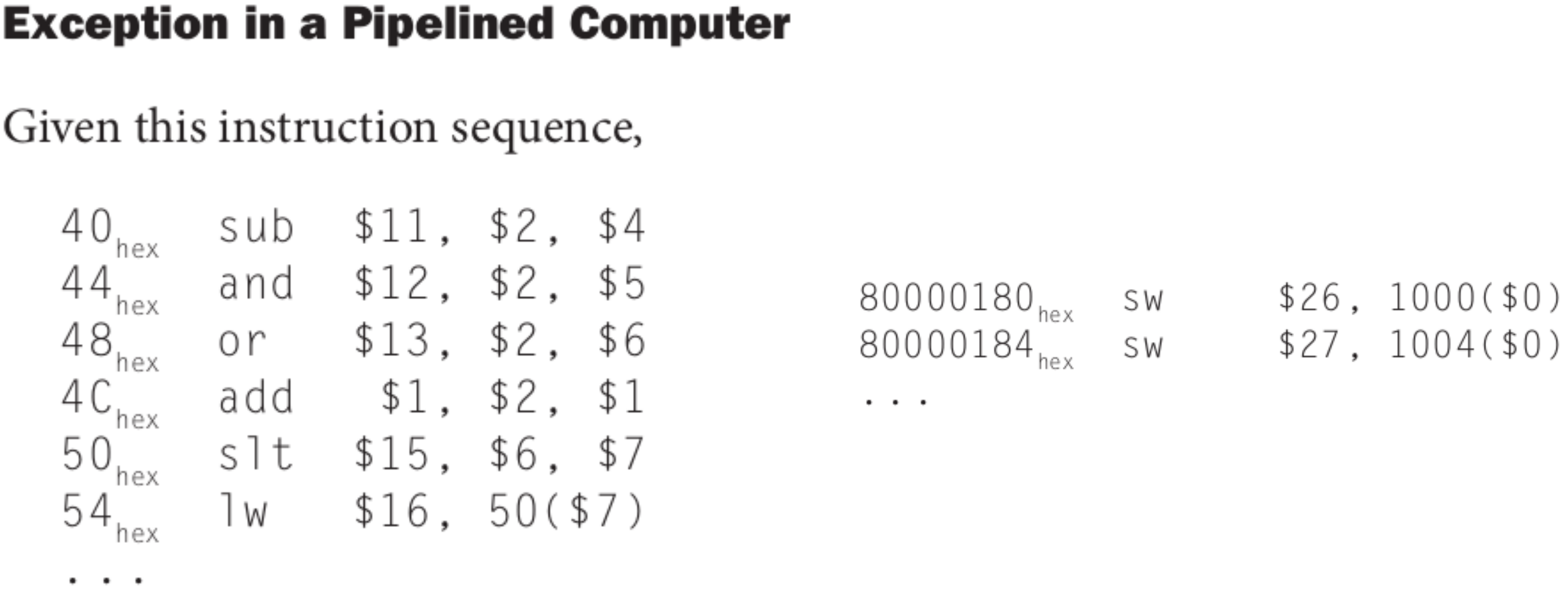
- syscall

Executes a system call. The system call number should be set in register $v0

- rfe- Return from exception.

Example

- Assume $1 overflows after add. User will never know what value of original $1 caused the exception.
  - So, it is important to stop execution in the middle of the pipeline (EX) and prevent writeback
  - Introduce an EX. Flush
  - Save the offending instruction in EPC
  - Output from ALU --> should generate an overflow flag to the control unit which inturn generates the flush signals



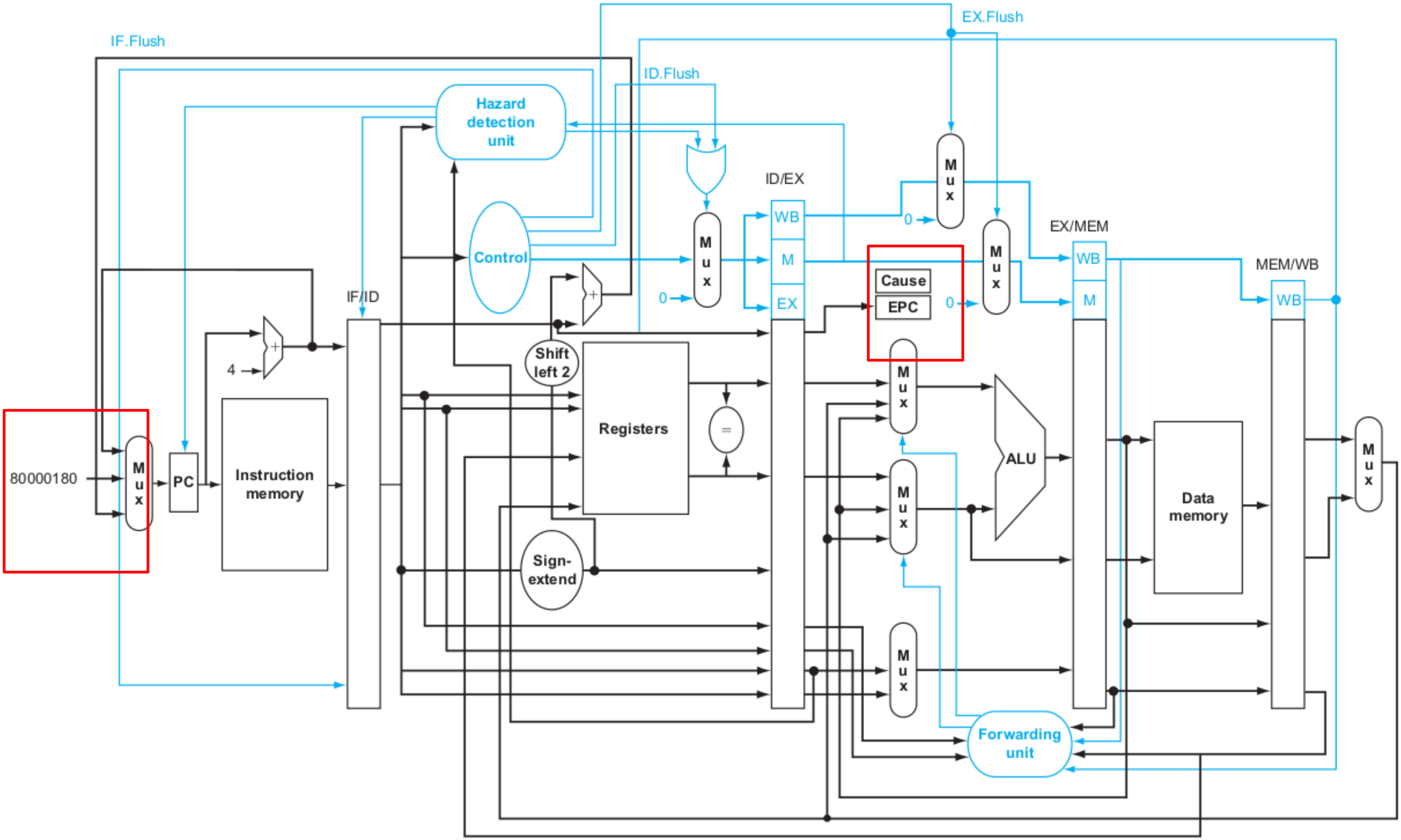
Exception in a Pipelined Computer

Given this instruction sequence,

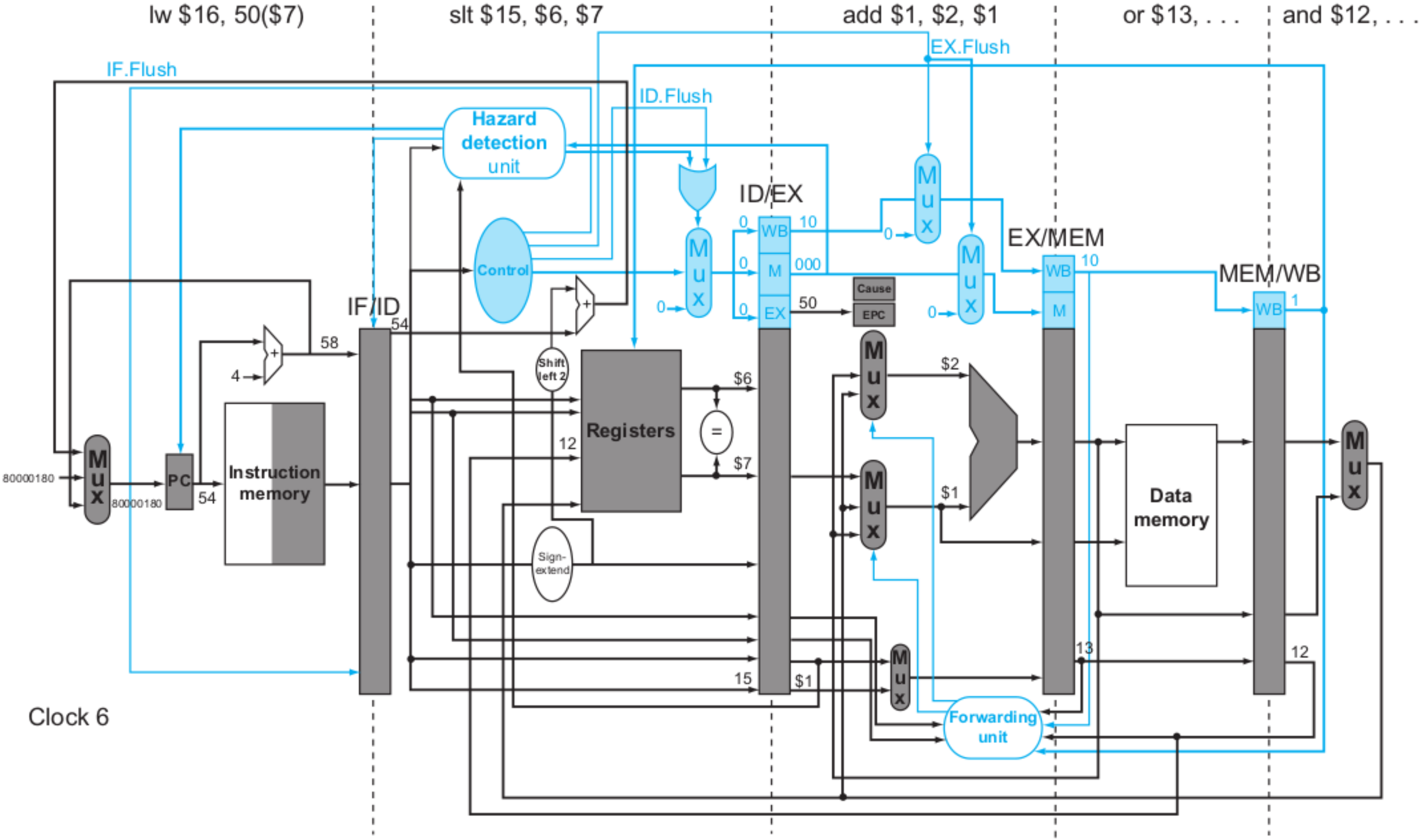
```
40hex   sub  $11, $2, $4        80000180hex  sw     $26, 1000($0)
44hex   and  $12, $2, $5        80000184hex  sw     $27, 1004($0)
48hex   or   $13, $2, $6        . . .
4Chex   add   $1, $2, $1
50hex   slt  $15, $6, $7
54hex   lw   $16, 50($7)
. . .
```

Show what happens in the pipeline if an overflow exception occurs in the add instruction.

Additions to the MIPS architecture



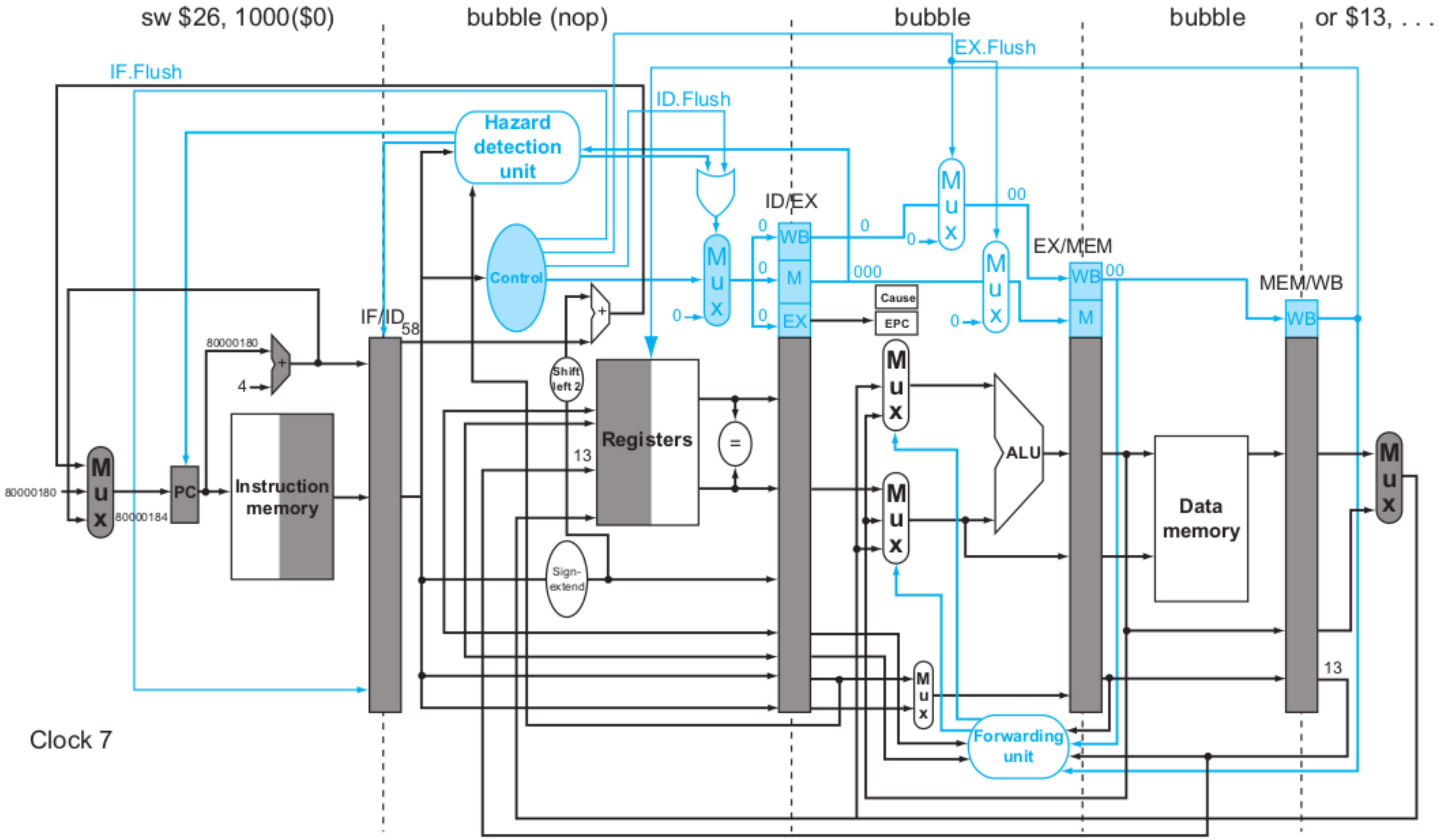
Overflow detected in EX of clock 6. Causes Flush of ADD and



Clock 6

Computer organization and design- Henessey and Patterson

Prior instructions complete. Future instructions flushed. Start from ISR



Computer organization and design- Henessey and Patterson

Acknowledgements

- CS305 – IIT Bombay – Bhaskaran Raman
- CS152: Computer architecture: UCB http://www-inst.eecs.berkeley.edu/~cs152/sp12/lectures/L05-PipeliningII.pdf
- CMSC 611: UMN http://ece-research.unm.edu/jimp/611/slides/chap3_5.html
- CSCE430/830 – Univ of Maine
- Computer organization and design- Henessey and Patterson