EGC442 Class Notes 4/14/2023

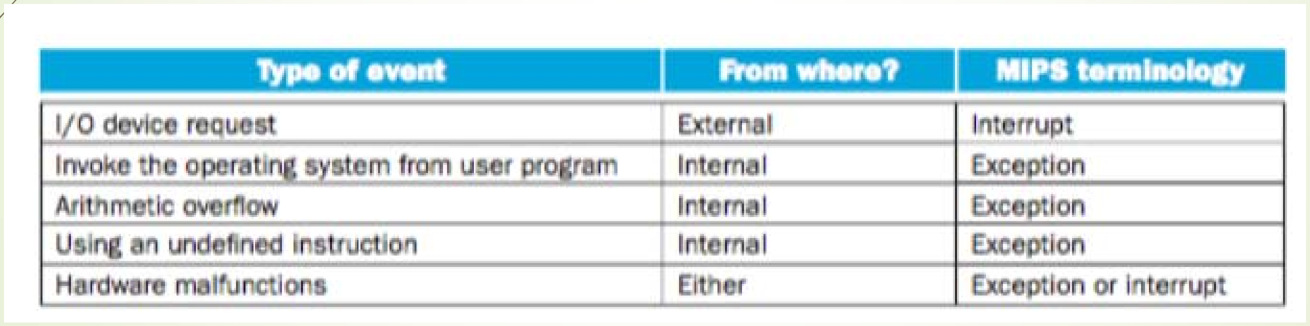
Baback Izadi
Division of Engineering Programs
bai@engr.newpaltz.edu

Test 2:

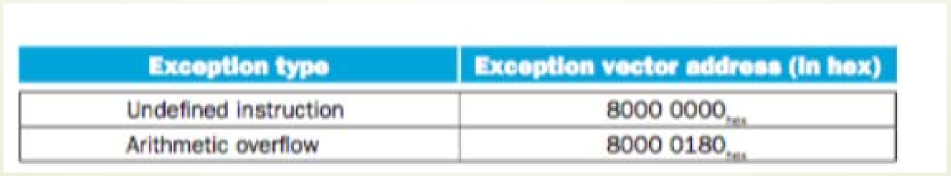
- Chapter 4
  - ALU design
- Chapter 5
  - Design of data path and control
  - Pipelined processor
  - Correcting for various hazards
  - Advanced pipeline concepts

Exception

- ***Exception***: Also called ***interrupt***. An unscheduled event that disrupts program execution; used to detect overflow.
- ***Interrupt***: An exception that comes from outside of the processor. (Some architectures use the term *interrupt* for all exceptions.)
- ***Vectored interrupt***: An interrupt for which the address to which control is transferred is determined by the cause of the exception



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



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

Exception Example

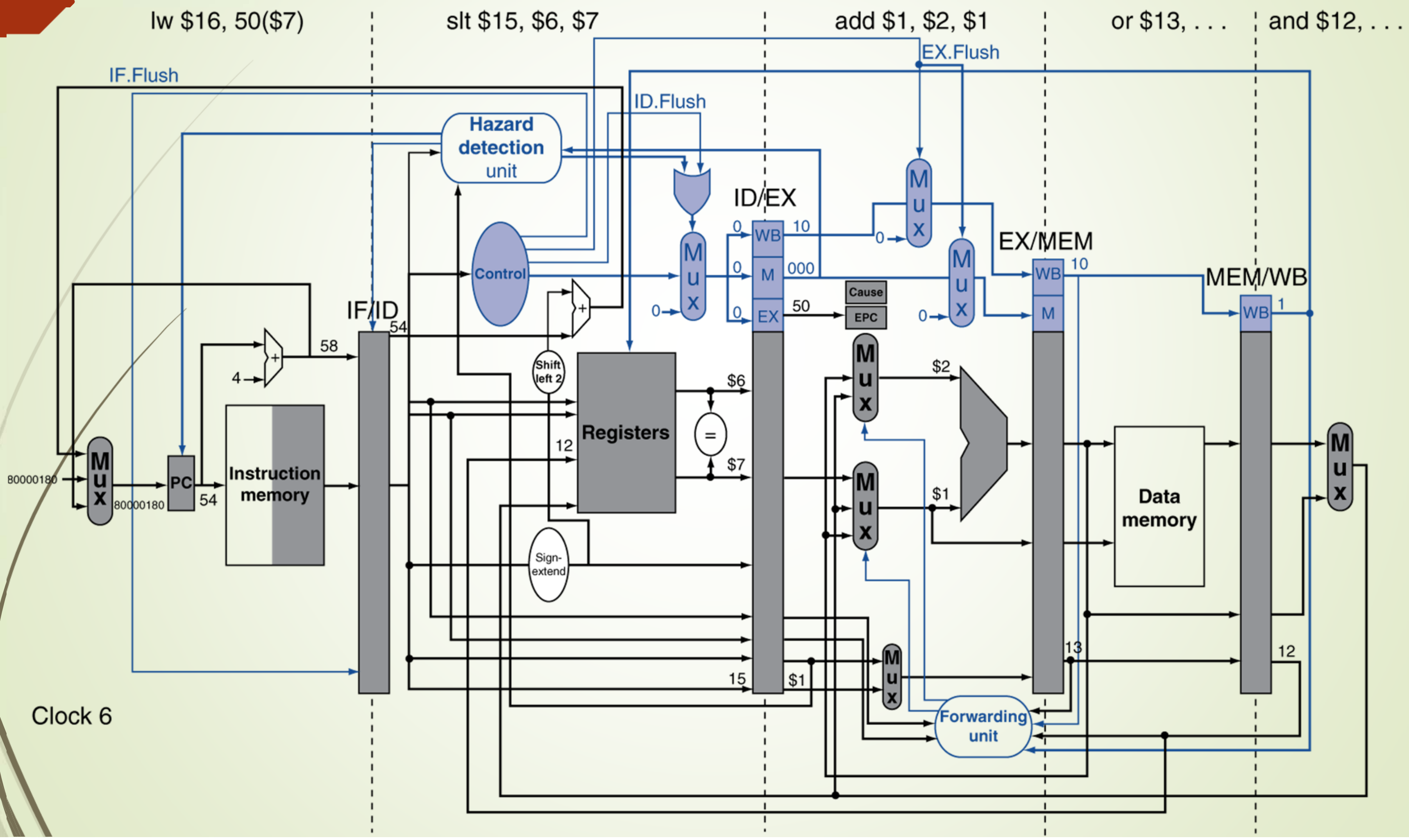
- Exception on add in

```
40 sub  $11, $2, $4
44 and  $12, $2, $5
48 or   $13, $2, $6
4C add  $1,  $2, $1
50 slt  $15, $6, $7
54 lw   $16, 50($7)
…
```

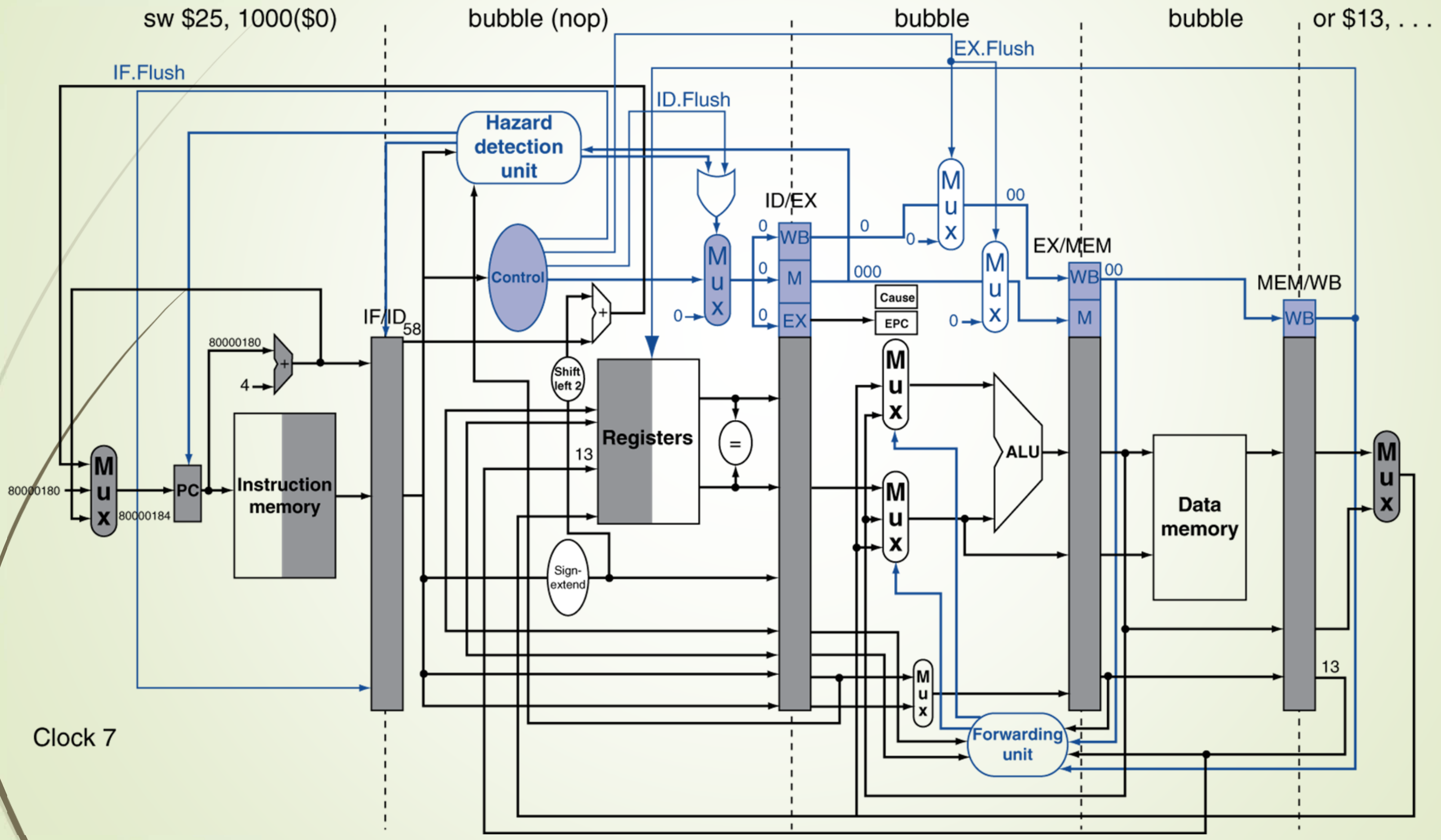
- Handler

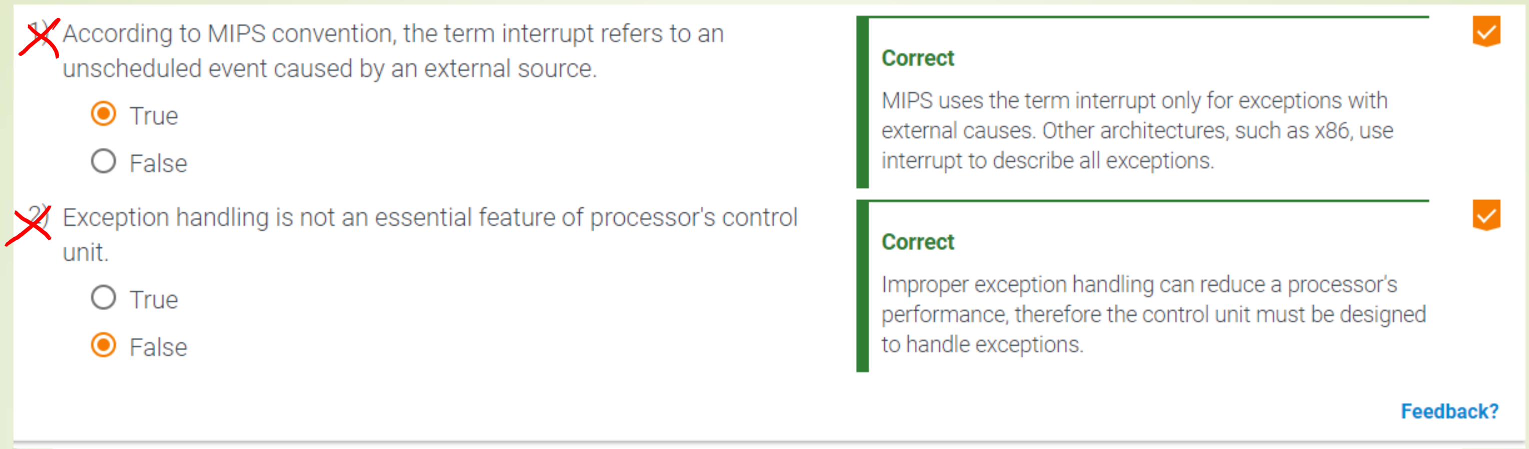
```
80000180  sw   $25, 1000($0)
80000184  sw   $26, 1004($0)
…
```

Exception Example



Exception Example





1) According to MIPS convention, the term interrupt refers to an unscheduled event caused by an external source.

[✓ True] False

Correct ✓

MIPS uses the term interrupt only for exceptions with external causes. Other architectures, such as x86, use interrupt to describe all exceptions.

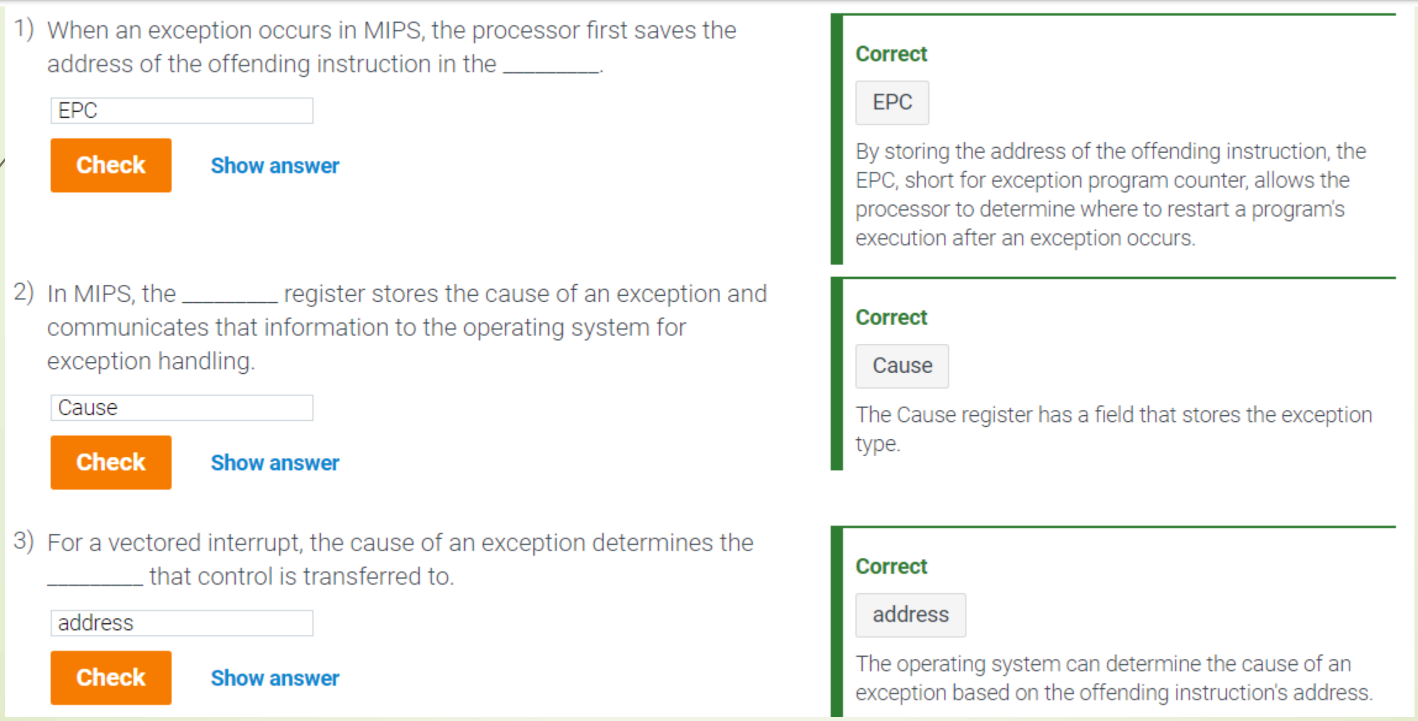
2) Exception handling is not an essential feature of processor's control unit.

True [✓ False]

Correct ✓

Improper exception handling can reduce a processor's performance, therefore the control unit must be designed to handle exceptions.

Feedback?



1) When an exception occurs in MIPS, the processor first saves the address of the offending instruction in the ________.

EPC

Check Show answer

Correct

EPC

By storing the address of the offending instruction, the EPC, short for exception program counter, allows the processor to determine where to restart a program's execution after an exception occurs.

2) In MIPS, the ________ register stores the cause of an exception and communicates that information to the operating system for exception handling.

Cause

Check Show answer

Correct

Cause

The Cause register has a field that stores the exception type.

3) For a vectored interrupt, the cause of an exception determines the ________ that control is transferred to.

address

Check Show answer

Correct

address

The operating system can determine the cause of an exception based on the offending instruction's address.

4 1) In a pipeline implementation, offending arithmetic overflow instructions are detected in the ________ stage of the pipeline to prevent the results from being written to the ________ stage.

- IF, ID
- EX, MEM
- [✓ EX, WB]

Correct

The EX.Flush signal prevents the instruction from fully executing and writing results to the WB, or write back, stage.

5 2) In the majority of MIPS implementations, multiple thrown exceptions are interrupted ________.

- according to which instruction causes the largest exception
- [✓ according to which offending instruction is earliest]
- randomly

Correct

MIPS processors interrupt the earliest instruction first.

6 3) A(n) ________ is always associated with an exact instruction in pipelined computers.

- [✓ precise interrupt]
- imprecise interrupt

Correct

Designing precise interrupts is difficult and so some processors have imprecise interrupts.

```
add $1, $2, $1  # arithmetic overflow
XXX $1, $2, $1  # undefined instruction
sub $1, $2, $1  # hardware error
```

7 Which exception should be recognized first in the above sequence?

- [✓ arithmetic overflow]
- undefined instruction
- hardware error

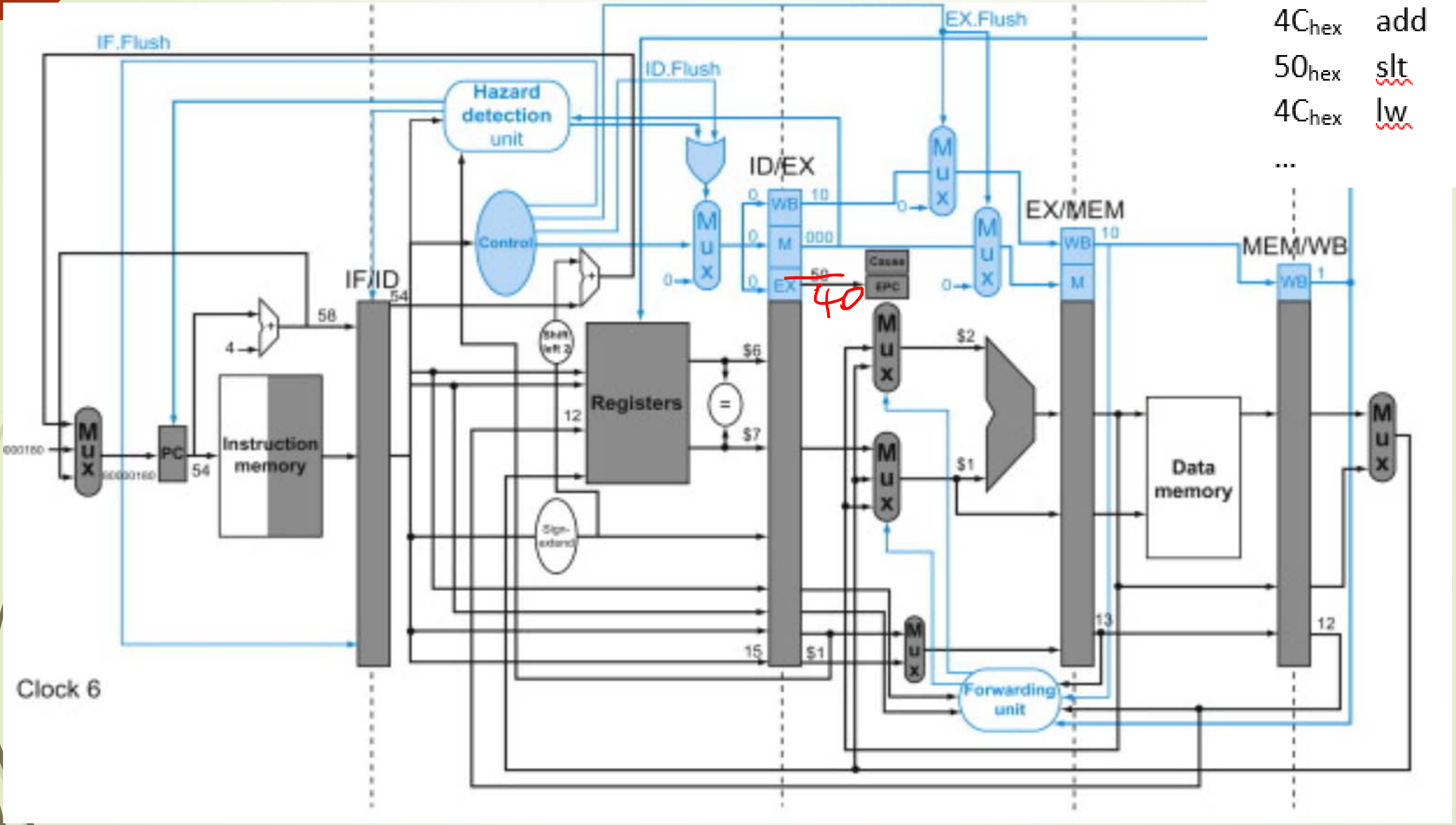
Correct

The add instruction is logically executed first. The overflow is detected in the EX stage and invokes the operating system to handle the exception.

7. Show what happens in the pipeline if an overflow exception occurs in the *sub* instruction.

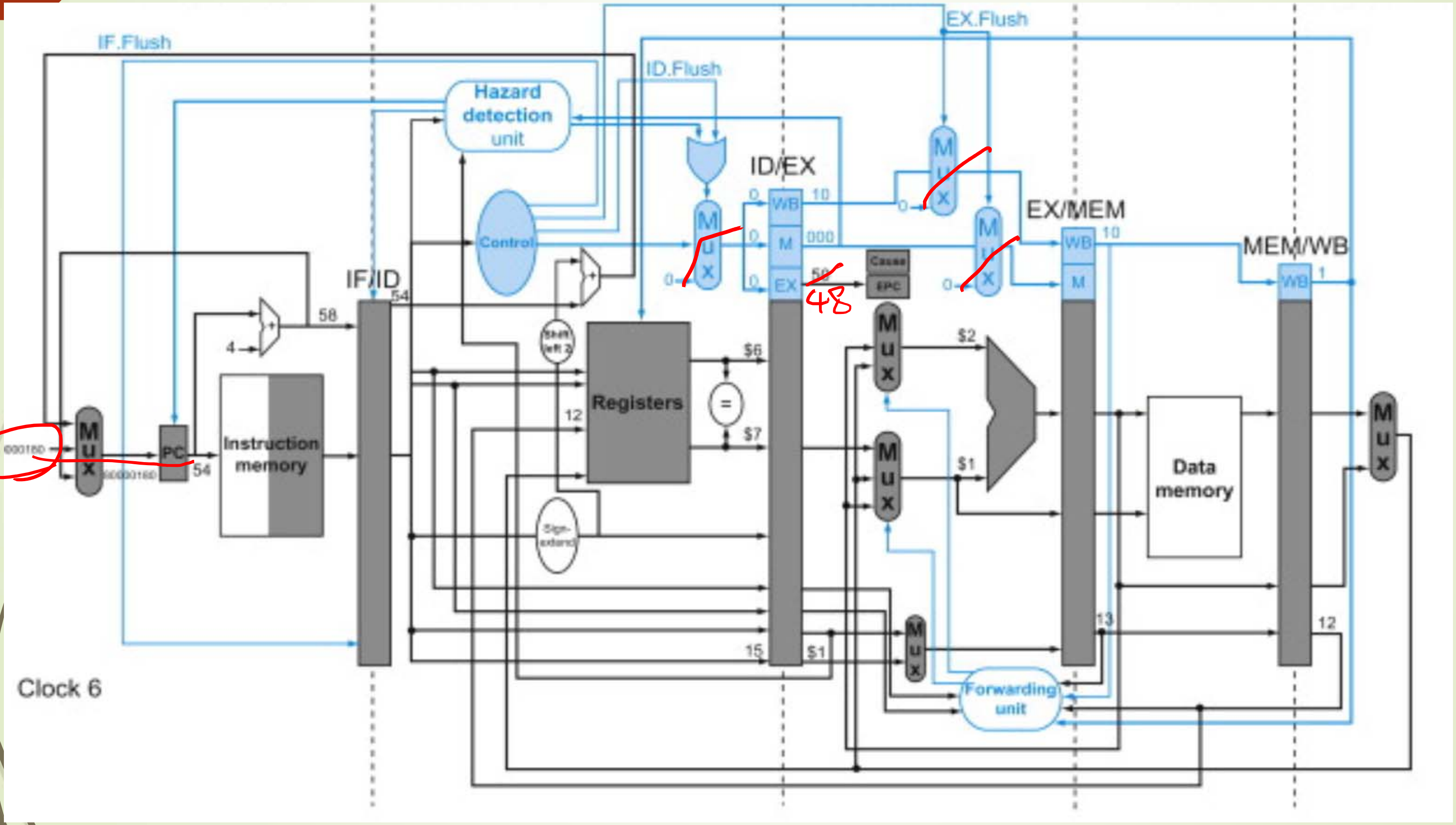
40_{hex}	and	\$11, \$2, \$4
44_{hex}	sub	\$12, \$2, \$4
48_{hex}	or	\$13, \$2, \$6
$4C_{hex}$	add	\$1, \$2, \$1
50_{hex}	slt	\$15, \$6, \$7
$4C_{hex}$	lw	\$16, 50(\$7)
...		

or \$13, \$2, \$6 **sub \$12, \$2, \$5** **and \$11, \$2, \$4**

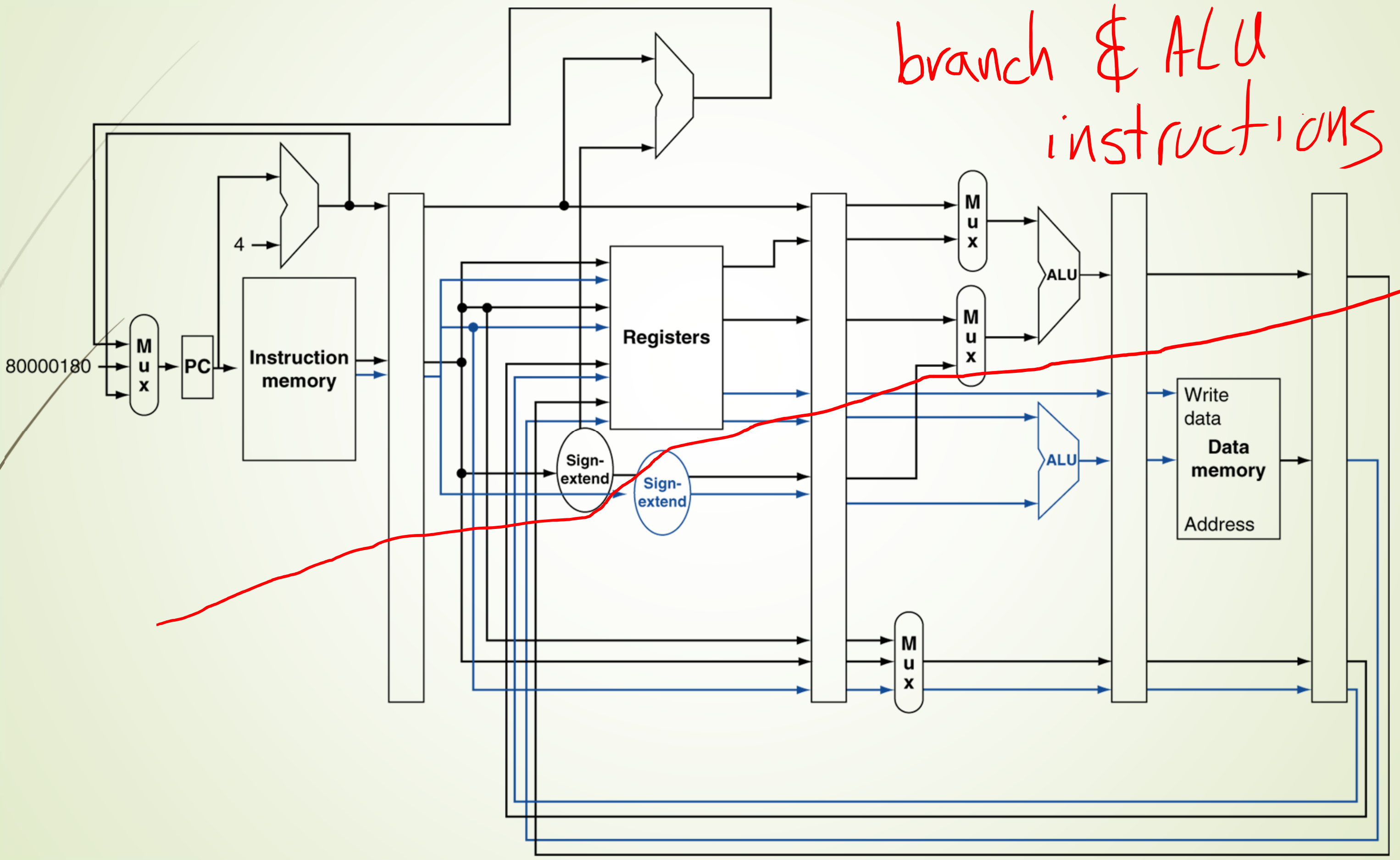


Show what happens in the pipeline if an overflow exception occurs in the sub instruction.

~~add~~ SW ~~or \$13, \$2, \$6~~ NOP ~~sub \$12, \$2, \$5~~ NOP **and \$11, \$2, \$4**

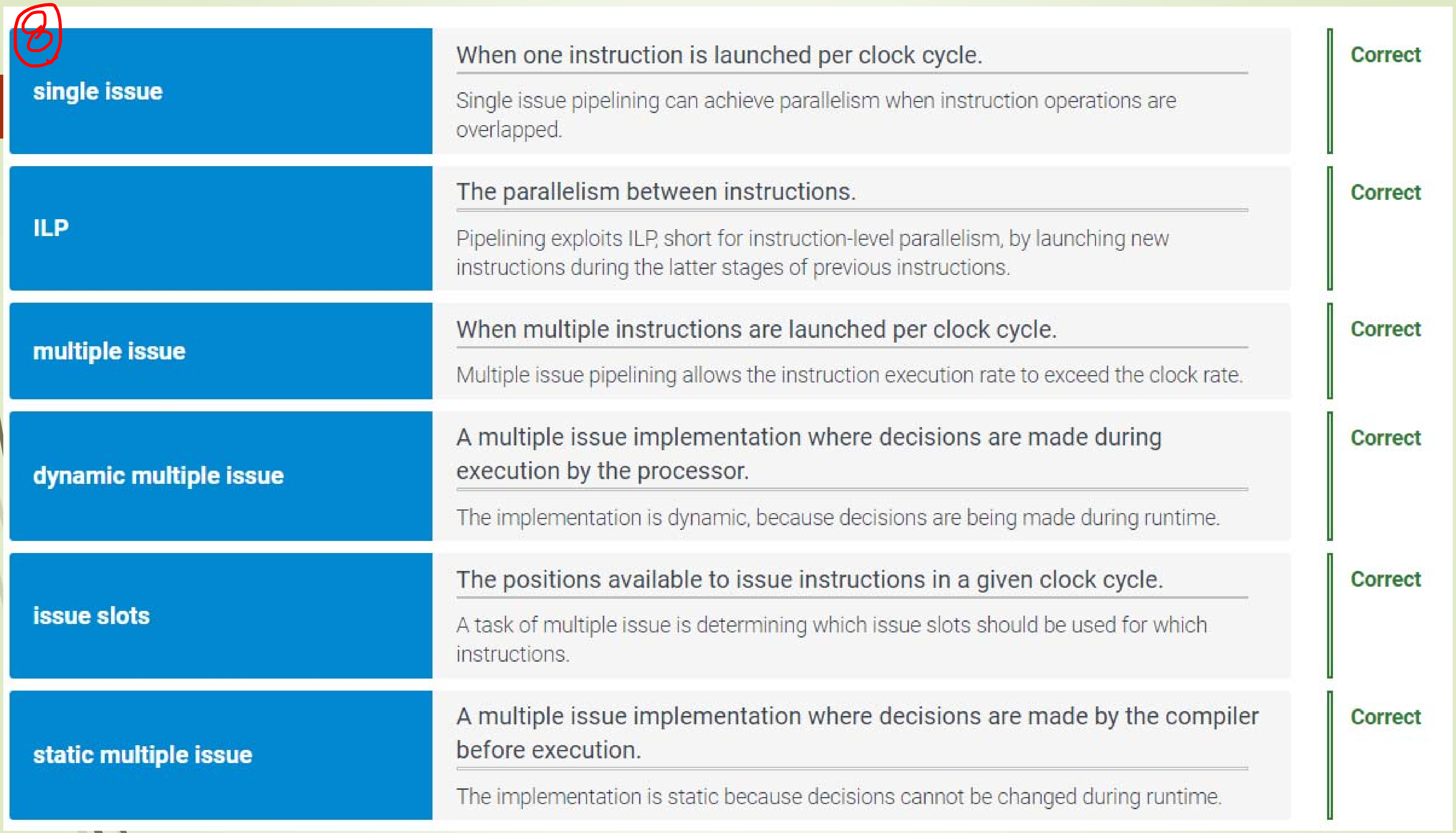


MIPS with Static Dual Issue



branch & ALU instructions

Load & store instructions



8

single issue	When one instruction is launched per clock cycle. Single issue pipelining can achieve parallelism when instruction operations are overlapped.	Correct
ILP	The parallelism between instructions. Pipelining exploits ILP, short for instruction-level parallelism, by launching new instructions during the latter stages of previous instructions.	Correct
multiple issue	When multiple instructions are launched per clock cycle. Multiple issue pipelining allows the instruction execution rate to exceed the clock rate.	Correct
dynamic multiple issue	A multiple issue implementation where decisions are made during execution by the processor. The implementation is dynamic, because decisions are being made during runtime.	Correct
issue slots	The positions available to issue instructions in a given clock cycle. A task of multiple issue is determining which issue slots should be used for which instructions.	Correct
static multiple issue	A multiple issue implementation where decisions are made by the compiler before execution. The implementation is static because decisions cannot be changed during runtime.	Correct

Scheduling Example

- Schedule this for dual-issue MIPS

```
Loop: lw   $t0, 0($s1)        # $t0=array element
      addu $t0, $t0, $s2      # add scalar in $s2
      sw   $t0, 0($s1)        # store result
      addi $s1, $s1,-4        # decrement pointer
      bne  $s1, $zero, Loop # branch $s1!=0
```

	ALU/branch	Load/store	cycle
Loop:	nop	lw $t0, 0($s1)	1
	addi $s1, $s1,-4	nop	2
	addu $t0, $t0, $s2	nop	3
	bne $s1, $zero, Loop	sw $t0, 4($s1)	4

- IPC = 5/4 = 1.25 (c.f. peak IPC = 2)

9. Show how the following loop can be scheduled on a static two-issue pipeline for MIPS?

```
Loop:  lw    $t0, 0($a1)
       add   $t0, $t0, $a3
       sw    $t0, 0($a1)
       addi  $a1, $a1, -12
       bne   $a1, $0, Loop
```

Computer the overall IPC.

	ALU/Branch		Load / Store	
Loop	nop		lw	$t0, 0($s1)
	nop			
	add	$t0, $t0, $a3	sw	$t0, 0($a1)
	addi	$a1, $a1, -12		
	bne	$a1, $0, Loop		

- IPC = 5/5 = 1

9. Show how the following loop can be scheduled on a static two-issue pipeline for MIPS?

```
Loop:  lw   $t0, 0($a1)
       add  $t0, $t0, $a3
       sw   $t0, 0($a1)
       addi $a1, $a1, -12
       bne  $a1, $0, Loop
```

Reorder the instructions to avoid as many pipeline stalls as possible. Computer the overall IPC.

	ALU/Branch		Load / Store	
Loop	nop		lw	$t0, 0($s1)
	addi	$a1, $a1, -12		
	add	$t0, $t0, $a3		
	bne	$a1, $0, Loop	sw	$t0, 12($a1)

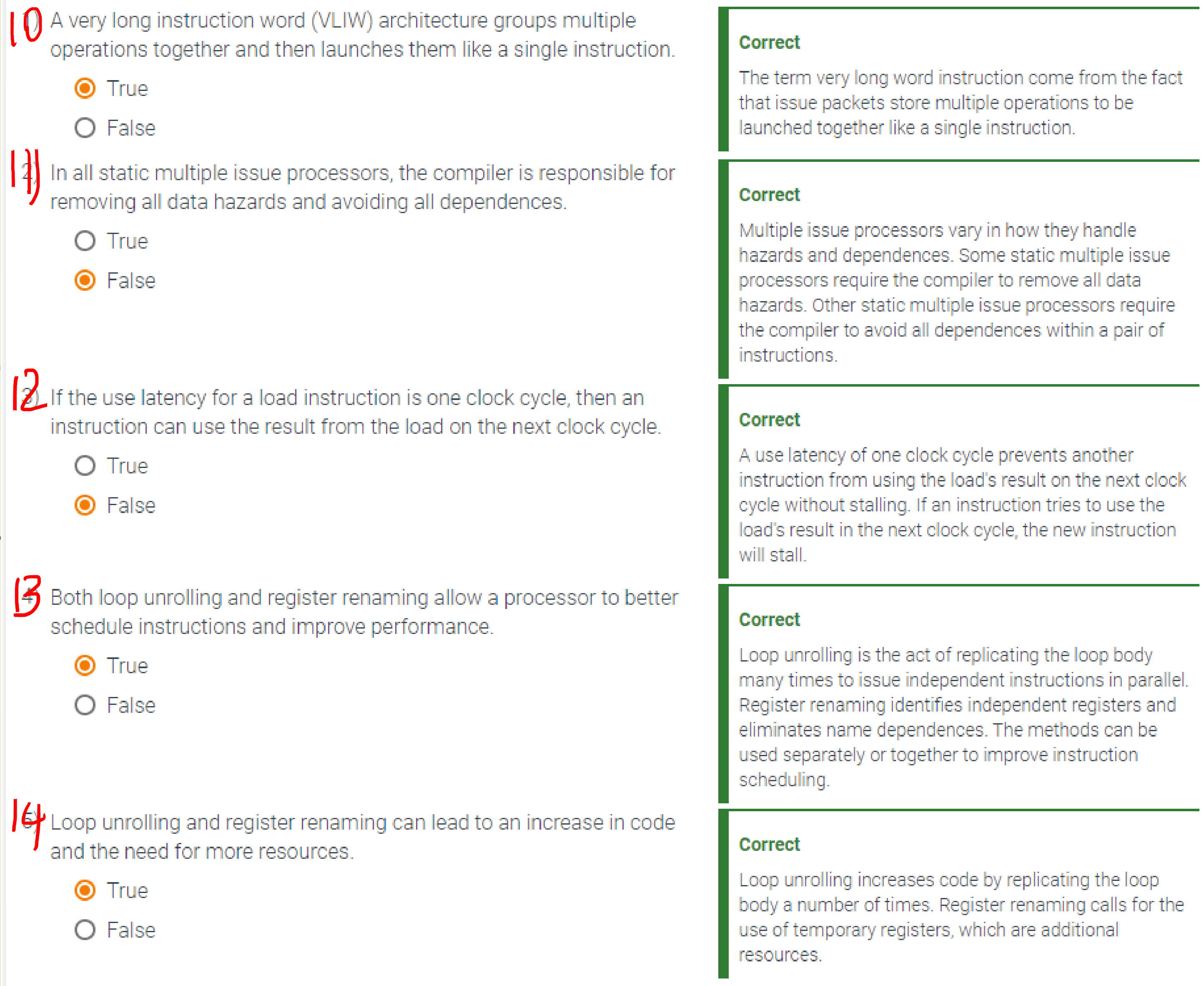
- IPC = 5/4 = 1.25

Loop Unrolling Example

```
lw   $t0, 0($s1)
addu $t0, $t0, $s2
sw   $t0, 0($s1)
addi $s1, $s1,-4


lw   $t0, 0($s1)
addu $t0, $t0, $s2
sw   $t0, 0($s1)
addi $s1, $s1,-4


lw   $t0, 0($s1)
addu $t0, $t0, $s2
sw   $t0, 0($s1)
addi $s1, $s1,-4


lw   $t0, 0($s1)
addu $t0, $t0, $s2
sw   $t0, 0($s1)
addi $s1, $s1,-4
```

```
lw   $t0, 0($s1)
addu $t0, $t0, $s2
sw   $t0, 0($s1)
addi $s1, $s1,-16

lw   $t0, 12($s1)
addu $t0, $t0, $s2
sw   $t0, 12($s1)

lw   $t0, 8($s1)
addu $t0, $t0, $s2
sw   $t0, 8($s1)

lw   $t0, 4($s1)
addu $t0, $t0, $s2
sw   $t0, 4($s1)
```

```
lw   $t0, 0($s1)
addu $t0, $t0, $s2
sw   $t0, 0($s1)
addi $s1, $s1,-16

lw   $t1, 12($s1)
addu $t1, $t1, $s2
sw   $t1, 12($s1)

lw   $t2, 8($s1)
addu $t2, $t2, $s2
sw   $t2, 8($s1)

lw   $t3, 4($s1)
addu $t3, $t3, $s2
sw   $t3, 4($s1)
```



10 A very long instruction word (VLIW) architecture groups multiple operations together and then launches them like a single instruction.

[✓ True] False

Correct

The term very long word instruction come from the fact that issue packets store multiple operations to be launched together like a single instruction.

11 In all static multiple issue processors, the compiler is responsible for removing all data hazards and avoiding all dependences.

True [✓ False]

Correct

Multiple issue processors vary in how they handle hazards and dependences. Some static multiple issue processors require the compiler to remove all data hazards. Other static multiple issue processors require the compiler to avoid all dependences within a pair of instructions.

12 If the use latency for a load instruction is one clock cycle, then an instruction can use the result from the load on the next clock cycle.

True [✓ False]

Correct

A use latency of one clock cycle prevents another instruction from using the load's result on the next clock cycle without stalling. If an instruction tries to use the load's result in the next clock cycle, the new instruction will stall.

13 Both loop unrolling and register renaming allow a processor to better schedule instructions and improve performance.

[✓ True] False

Correct

Loop unrolling is the act of replicating the loop body many times to issue independent instructions in parallel. Register renaming identifies independent registers and eliminates name dependences. The methods can be used separately or together to improve instruction scheduling.

14 Loop unrolling and register renaming can lead to an increase in code and the need for more resources.

[✓ True] False

Correct

Loop unrolling increases code by replicating the loop body a number of times. Register renaming calls for the use of temporary registers, which are additional resources.

15) Show would the following loop unrolling and register renaming can be used for 4 iteration of the following on a static two-issue pipeline for MIPS? Computer the overall IPC.

```
Loop:   lw   $t0, 0($a1)
        add  $t0, $t0, $a3
        sw   $t0, 0($a1)
        addi $a1, $a1, -12
        bne  $a1, $0, Loop
```

```
lw   $t0, 0($s1)
add  $t0, $t0, $a3
sw   $t0, 0($a1)
addi $a1, $a1,-12


lw   $t0, 0($s1)
add  $t0, $t0, $a3
sw   $t0, 0($a1)  -12
addi $a1, $a1,-12
                      -24

lw   $t0, 0($s1)
add  $t0, $t0, $a3
sw   $t0, 0($a1)
addi $a1, $a1,-12


lw   $t0, 0($s1)
add  $t0, $t0, $a3
sw   $t0, 0($a1)
addi $a1, $a1,-12
```

```
lw   $t0, 0($s1)
add  $t0, $t0, $a3
sw   $t0, 0($a1)
addi $a1, $a1,-48

lw   $t0, 0($s1)
add  $t0, $t0, $a3
sw   $t0, 36($a1) -12
                0
lw   $t0, 0($s1)      -24
add  $t0, $t0, $a3
sw   $t0, 24($a1)


lw   $t0, 0($s1)
add  $t0, $t0, $a3
sw   $t0, 12($a1)
```

```
lw   $t0, 0($s1)
add  $t0, $t0, $a3
sw   $t0, 0($a1)
addi $a1, $a1,-48

lw   $t1, 0($s1)
add  $t1, $t1, $a3
sw   $t1, 36($a1)


lw   $t2, 0($s1)
add  $t2, $t2, $a3
sw   $t2, 24($a1)


lw   $t3, 0($s1)
add  $t3, $t3, $a3
sw   $t3, 12($a1)
```

Loop Unrolling Example

WR

Read

	ALU/branch	Load/store	cycle
Loop:	addi $s1, $s1,-48	lw $t0, 0($s1)	1
	nop	lw $t1, 12($s1)	2
	add $t0, $t0, $a3	lw $t2, 8($s1)	3
	add $t1, $t1, $a3	lw $t3, 4($s1)	4
	add $t2, $t2, $a3	sw $t0, 48($s1)	5
	add $t3, $t3, $a3	sw $t1, 36($s1)	6
	nop	sw $t2, 24($s1)	7
	bne $a1, $zero, Loop	sw $t3, 12($s1)	8

```
lw   $t0, 0($s1)
add  $t0, $t0, $a3
sw   $t0, 0($a1)
addi $a1, $a1,-48

lw   $t1, 0($s1)
add  $t1, $t1, $a3
sw   $t1, 36($a1)

lw   $t2, 0($s1)
add  $t2, $t2, $a3
sw   $t2, 24($a1)

lw   $t3, 0($s1)
add  $t3, $t3, $a3
sw   $t3, 12($a1)
```

- IPC = 14/8 = 1.75
  - Closer to 2, but at cost of registers and code size