



# Computer Organization

## Lab6 MIPS(5) - Exception, Interruption, Trap

## Topics

- **Exception** vs **Interrupt**
- **Common Exception**
- **Exception Handler**
  - **Register in coprocessor 0**
  - **Trap, eret (MIPS32)**

## Exception vs Interruption

- An **exception** is an event that **disrupts the normal flow of the execution of your code.**
  - When an exception occurs, **the CPU** will figure out what is wrong by **checking its status**, see if it can be corrected and then continue the execution of the normal code like nothing happened.
  - E.g. Accessing to the 0x0 address in user mode will trigger an **exception**.
- An **interruption** is an event **caused by a device which is external to the CPU.**
  - E.g. 'syscall' is an **interruption**.

# Common Exception

The following exceptions are the most common in the main processor:

- **Address error** exceptions
  - Which occur when the machine references a data item that is NOT on its proper **memory alignment** or when an **address is invalid** for the executing process.
- **Overflow** exceptions
  - Which occur when **arithmetic operations** compute **signed** values and the destination lacks the precision to store the result.
- **Bus exceptions**
  - Which occur when **an address is invalid** for the executing process.
- **Divide-by-zero** exceptions
  - Which occur when a **divisor is zero**.

# Bad Address Exception

```
.text
print_string:
    addi $sp,$sp,-4
    sw $v0,($sp)

    li $v0,4
    syscall

    lw $v0,($sp)
    addi $sp,$sp,4

    jr $ra
```

Registers	Coproc 1	Coproc 0
Name	Number	Value
\$8 (vaddr)	8	0x00000000
\$12 (status)	12	0x0000ff13
\$13 (cause)	13	0x00000010
\$14 (epc)	14	0x0040000c

Ekpt	Address	Code	Basic	Source
	0x00400000	0x23bdffff	addi \$29,\$29,0xffffffff	5: addi \$sp,\$sp,-4
	0x00400004	0xafaf0000	sw \$2,0x00000000(\$29)	6: sw \$v0,(\$sp)
	0x00400008	0x24020004	addiu \$2,\$0,0x00000004	9: li \$v0,4
	0x0040000c	0x0000000c	syscall	10: syscall
	0x00400010	0x0fa20000	lw \$2,0x00000000(\$29)	12: lw \$v0,(\$sp)
	0x00400014	0x23bd0004	addi \$29,\$29,0x00000004	13: addi \$sp,\$sp,4
	0x00400018	0x201fffff	addi \$31,\$0,0xffffffff	15: addi \$ra,\$zero,0xffffffff
	0x0040001c	0x03e00008	jr \$31	16: jr \$ra

Runtime exception at 0x0040000c: address out of range 0x00000000

\$a0's default value is 0x00000000, which is not allowed to access in user mode

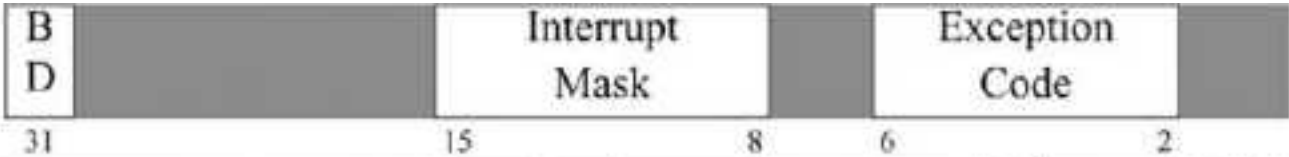
## The Register in Coprocessor 0

Register name	Register number	Usage
VAddr	8	memory address at which an offending memory reference occurred
Status	12	interrupt mask and enable bits
Cause	13	exception type and pending interrupt bits
EPC	14	address of instruction that caused exception

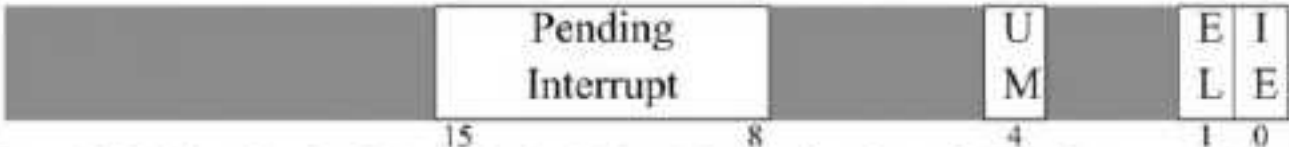
Registers	Coproc 1	Coproc 0	
Name	Number	Value	
\$8 (vaddr)	8	0x00000000	
\$12 (status)	12	0x0000ff13	
\$13 (cause)	13	0x00000030	
\$14 (epc)	14	0x00400010	

# Exception Control Registers

CAUSE:



STATUS:



BD = Branch Delay, UM = User Mode, EL = Exception Level, IE =Interrupt Enable

## EXCEPTION CODES

Number	Name	Cause of Exception	Number	Name	Cause of Exception
0	Int	Interrupt (hardware)	9	Bp	Breakpoint Exception
4	AdEL	Address Error Exception (load or instruction fetch)	10	RI	Reserved Instruction Exception
5	AdES	Address Error Exception (store)	11	CpU	Coprocessor Unimplemented
6	IBE	Bus Error on Instruction Fetch	12	Ov	Arithmetic Overflow Exception
7	DBE	Bus Error on Load or Store	13	Tr	Trap
8	Sys	Syscall Exception	15	FPE	Floating Point Exception



# Bad Address Exception continued

```
.data
    str: .asciiz "hello"
.text
print_string:
    addi $sp,$sp,-4
    sw $v0,($sp)

    la $a0,str
    li $v0,4
    syscall

    lw $v0,($sp)
    addi $sp,$sp,4

    addi $ra,$zero,0xffffffff
    jr $ra
```

\$ra	31	0xffffffff
pc		0xffffffff

Registers	Coproc 1	Coproc 0	
Name	Number	Value	
\$8 (vaddr)	8	0xffffffff	
\$12 (status)	12	0x0000ff13	
\$13 (cause)	13	0x00000010	
\$14 (epc)	14	0xffffffff	

Error in : invalid program counter value: 0xffffffff



## Bad Address Exception continued

Which one will trigger the exception ?

Registers	Coproc 1	Coproc 0
Name	Number	Value
\$8 (vaddr)	8	0x10010009
\$12 (status)	12	0x0000ff13
\$13 (cause)	13	0x00000010
\$14 (epc)	14	0x0040001c

```
.include "../macro_print_str.asm"
.data
```

```
str: .ascii "data is:"
bs: .byte 1:10
ws: .word 2:10
```

```
.text
print_string("data is:")
add $t0,$zero,$zero
addi $t1,$zero,10
```

```
loop_out:
lw $a0,bs
li $v0,1
syscall
add $t0,$t0,1
bne $t0,$t1,loop_out
```

```
end
```

```
.include "../macro_print_str.asm"
.data
```

```
str: .ascii "data is:"
bs: .byte 1:10
ws: .word 2:10
```

```
.text
print_string("data is:")
add $t0,$zero,$zero
addi $t1,$zero,10
```

```
loop_out:
lw $a0,ws
li $v0,1
syscall
add $t0,$t0,1
bne $t0,$t1,loop_out
```

```
end
```

Runtime exception at 0x0040001c: fetch address not aligned on word boundary 0x10010009

# Arithmetic Exception

Will the 'addu' trigger an exception? How about 'sub', 'div'? How about 'addiu \$a0, \$t0, -1'?

**.data**

addend1: .word 0x7fffffff

addend2: .word 0x7fffffff

**.text**

print\_string:

lw \$t0,addend1

lw \$t1,addend2

**add \$a0,\$t0,\$t1**

li \$v0,1

syscall

li \$v0,10

syscall

Registers	Coproc 1	Coproc 0
Name	Number	Value
\$0 (vaddr)	8	0x00000000
<b>\$12 (status)</b>	<b>12</b>	<b>0x0000ff13</b>
\$13 (cause)	13	0x00000030
\$14 (epc)	14	0x00400010

Text Segment				
Bkpt	Address	Code	Basic	Source
<input type="checkbox"/>	0x00400000	0x3c011001	lui \$1,0x00001001	6: lw \$t0,addend1
<input type="checkbox"/>	0x00400004	0x8c280000	lw \$8,0x00000000(\$1)	
<input type="checkbox"/>	0x00400008	0x3c011001	lui \$1,0x00001001	7: lw \$t1,addend2
<input type="checkbox"/>	0x0040000c	0x8c290004	lw \$9,0x00000004(\$1)	
<input type="checkbox"/>	0x00400010	0x01092020	<b>add \$4,\$8,\$9</b>	<b>8: add \$a0,\$t0,\$t1</b>
<input type="checkbox"/>	0x00400014	0x24020001	addiu \$2,\$0,0x00000001	10: li \$v0,1
<input type="checkbox"/>	0x00400018	0x0000000c	syscall	11: syscall
<input type="checkbox"/>	0x0040001c	0x2402000a	addiu \$2,\$0,0x0000000a	13: li \$v0,10
<input type="checkbox"/>	0x00400020	0x0000000c	syscall	14: syscall

Runtime exception at 0x00400010: arithmetic overflow

# How MIPS Acts When Taking An Exception?

- **Step1.** It sets up the **EPC** to point to the restart location
- **Step2.** CPU changes into **kernel mode** and **disables the interrupts** (MIPS does this by setting **EXL** bit of **Status register**)
- **Step3.** Set up the **Cause register** to indicate **which** is wrong so that software can tell the reason for the exception. If it is for **address exception**, for example, TLB miss and so on, the **BadVaddr register** is set.
- **Step4.** CPU starts fetching instructions from the exception entry point and then goes to the **exception handler**.

Up to **MIPS III**, **eret** instruction is used to return to the original location before falling into the exception. Note that **eret** behavior is: **clear the SR[EXL] bit and return control to the address stored in EPC.**

## Exception Related Instructions

### ➤ Conditional trap

- `teq $s0, $s1` ##trap(jump to the `ktext`), if `s0==s1`
- `tne $s0, $s1` ##trap(jump to the `ktext`), if `s0!=s1`
- `teqi $s0, 1` ##trap(jump to the `ktext`), if `s0==1`

### ➤ `mfc0,mtc0`

- `mfc0 $k0,$14` ##Move from coproc0 reg#14(`epc`) to `$k0`
- `mtc0 $k0,$14` ##Move from `$k0` to coproc0 reg#14(`epc`)

### ➤ `eret`

- Returns from an interrupt, exception or error trap.
- Similar to a branch or jump instruction, `eret` executes the next instruction before taking effect. Use this on R4000 processor machines in place of `rfe`.

# Demo

## **.data**

```
        dmsg:  .asciiiz "\ndata over"
```

## **.text**

```
main:   li $v0,5
        syscall
        teqi $v0,0
        la $a0,dmsg
        li $v0,4
        syscall
        li $v0,10
        syscall
```

## **.ktext** 0x80000180

```
        move $k0,$v0
        move $k1,$a0
        la $a0,msg
        li $v0,4
        syscall
```

```
        move $v0,$k0
        move $a0,$k1
```

```
        mfc0 $k0,$14
        addi $k0,$k0,4
        mtc0 $k0,$14
```

```
        eret
```

## **.kdata**

```
msg: .asciiiz "\nTrap generated"
```

Q1. How to trigger the trap?

Q2. When will the string  
“\ndata over” be printed out?

Q3. Use “break” in text  
segment and ktext segment  
separately,  
what happens?

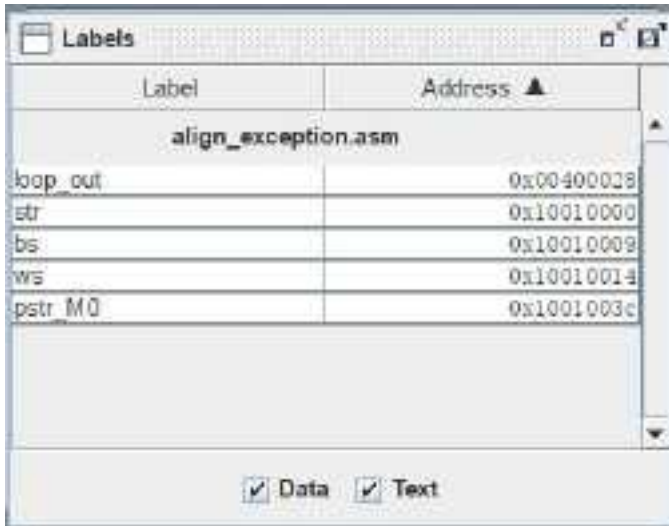
## Practice

1. Implementing a procedure to read a list of number from input, store them into an array and print every item of the array out:
  1. The size of the array and its space are determined by user's input
  2. The space of array item is determined by user's input
  3. While storing the array items into array, if the item exceed the bounday of array, an exception will be triggered and exit the program.
    1. For example: the size of array is 10 and its space is 10Bytes, the space of each array item is 4Bytes. Using loop to read the value of array items and write them into the array's space. While processing the 3th array item and write it into array's space, this value will pollute other areas. Your procedure should trigger an exception on this situation, print the warning infomation and exit the program.
  4. While all items are stored into the array correctly, print every item out.

The exception handler do the following things:

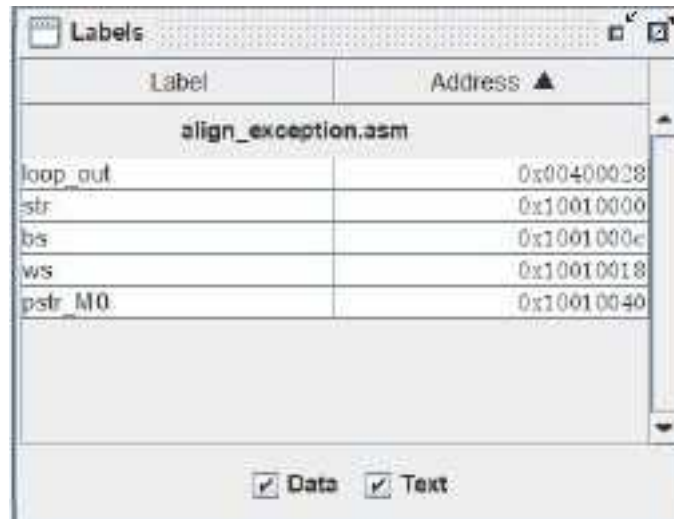
  1. stop the program running
  2. output prompt information, including “runtime exception at 0x\_\*\*\*”(the address of the instruction which triggered the exception), the cause of the exception (“ArrayItem pollutes other areas”, index is:\*\*\*) and the index of array item which triggered the exception.
  3. exit the program.

## Tips: usage of '.align'



Label	Address
align_exception.asm	
loop_out	0x00400028
str	0x10010000
bs	0x10010008
ws	0x10010014
pstr_M0	0x1001003c

☒ Data ☒ Text



Label	Address
align_exception.asm	
loop_out	0x00400028
str	0x10010000
bs	0x1001000c
ws	0x10010018
pstr_M0	0x10010040

☒ Data ☒ Text

.data

str: .ascii "data is:"

bs: .byte 1:10

ws: .word 2:10

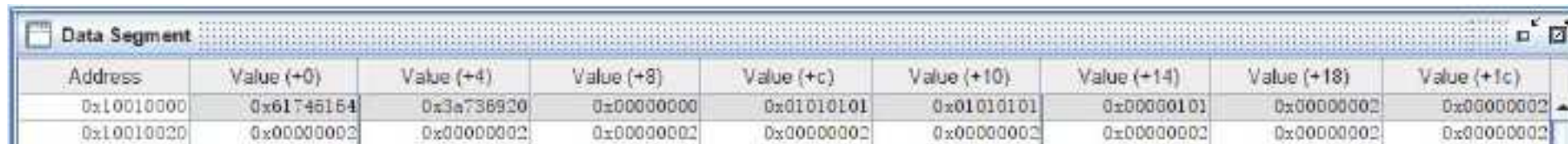
.data

str: .ascii "data is:"

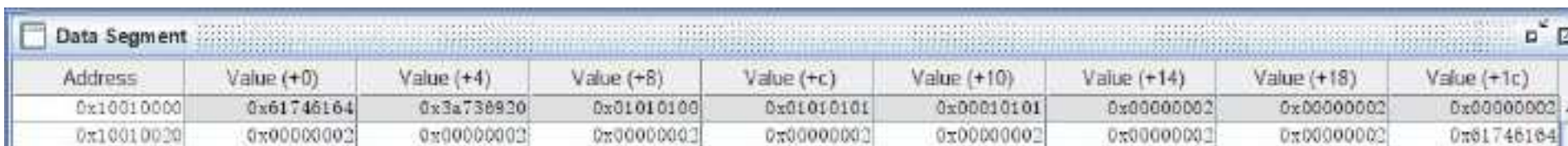
**.align 2**

bs: .byte 1:10

ws: .word 2:10



Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x61746164	0x3a736920	0x00000000	0x01010101	0x01010101	0x00000101	0x00000002	0x00000002
0x10010020	0x00000002	0x00000002	0x00000002	0x00000002	0x00000002	0x00000002	0x00000002	0x00000002



Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x61746164	0x3a736920	0x01010100	0x01010101	0x00010101	0x00000002	0x00000002	0x00000002
0x10010020	0x00000002	0x00000002	0x00000002	0x00000002	0x00000002	0x00000002	0x00000002	0x61746164



## Tips

**.align** Align the next datum on a  $2^n$  byte boundary.

For example, `.align 2` aligns the next value on a word boundary. `.align 0` turns off automatic alignment of `.half`, `.word`, `.float`, and `.double` directives until the next `.data` or `.kdata` directive.

**.kdata** subsequent items are stored in the kernel data segment, If the optional argument *addr* is present, subsequent items are stored starting at address *addr*.

**.ktext** subsequent items are stored in the kernel text segment, In SPIM, these items may only be instructions or words . If the optional argument *addr* is present, subsequent items are stored starting at address *addr*.