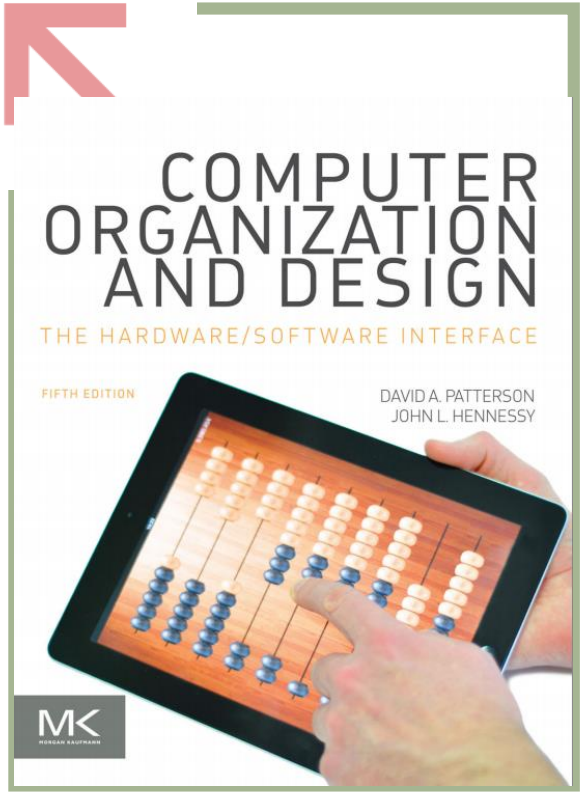
# Assembly programming

exception and exception handler

wangw6@sustc.edu.cn



# Topics

- Exception vs Interruption
- Common exception
- Exception handler
  - Register in coprocessor 0
  - trap instruction
- Lab assignment

# 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 to 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 |

Text Segment

| Bkpt | Address | Code | Basic | Source |
|---|---|---|---|---|
| | 0x00400000 | 0x23bdfffc | addi $29, $29, 0xfffffffc | 5: addi $sp, $sp, -4 |
| | 0x00400004 | 0xafa20000 | sw $2, 0x00000000($29) | 6: sw $v0, ($sp) |
| | 0x00400008 | 0x24020004 | addiu $2, $0, 0x00000004 | 9: li $v0, 4 |
| | 0x0040000c | 0x0000000c | syscall | 10: syscall |
| | 0x00400010 | 0x8fa20000 | 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 |

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

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

# 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 |

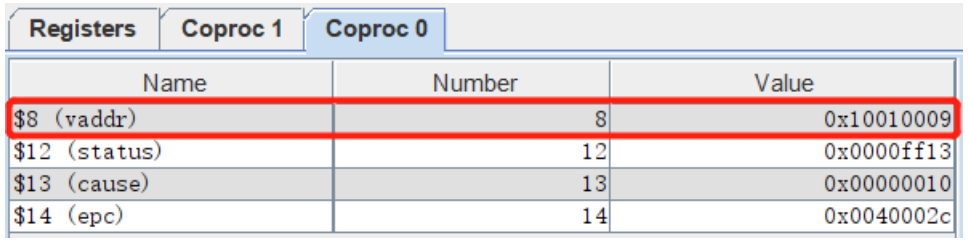
```
Error in : invalid program counter value: 0xffffffff
```

# Bad address exception continued

```
.include "../macro_print_str.asm"
.data
        str:    .asciiz "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
```

Which one will trigger the exception ?




```
.include "../macro_print_str.asm"
.data
        str:    .asciiz "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 0x0040002c: fetch address not aligned on word boundary 0x10010009
```

# Arithmetic exception

Will the ‘addu’ trigger an exception ?  How about ‘sub’, ’div’ ?  How about ‘addui $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 |
|---|---|---|
| $8 (vaddr) | 8 | 0x00000000 |
| $12 (status) | 12 | 0x0000ff13 |
| $13 (cause) | 13 | 0x00000030 |
| $14 (epc) | 14 | 0x00400010 |

Text Segment

| Bkpt | Address | Code | Basic | | Source |
|---|---|---|---|---|---|
| | 0x00400000 | 0x3c011001 | lui $1, 0x00001001 | 6: | lw $t0, addend1 |
| | 0x00400004 | 0x8c280000 | lw $8, 0x00000000($1) | | |
| | 0x00400008 | 0x3c011001 | lui $1, 0x00001001 | 7: | lw $t1, addend2 |
| | 0x0040000c | 0x8c290004 | lw $9, 0x00000004($1) | | |
| | 0x00400010 | 0x01092020 | add $4, $8, $9 | 8: | add $a0, $t0, $t1 |
| | 0x00400014 | 0x24020001 | addiu $2, $0, 0x00000001 | 10: | li $v0, 1 |
| | 0x00400018 | 0x0000000c | syscall | 11: | syscall |
| | 0x0040001c | 0x2402000a | addiu $2, $0, 0x0000000a | 13: | li $v0, 10 |
| | 0x00400020 | 0x0000000c | syscall | 14: | syscall |

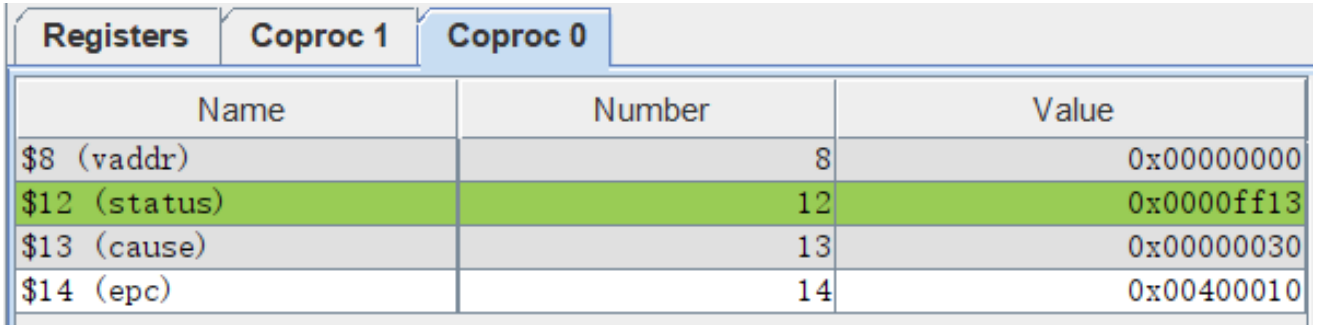
```
Runtime exception at 0x00400010: arithmetic overflow
```

# How MIPS acts when taking an exception?

1. It sets up the **EPC** to point to the restart location.
2. CPU changes into kernel mode and disables the interrupts (MIPS does this by setting EXL bit of SR register)
3. 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 also set.
4. CPU starts fetching instructions from the exception entry point and then goes to the **exception handler.**

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

# The register in Coprocessor 0




| Register name | Register number | Usage |
|---|---|---|
| BadVAddr | 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 |

# Related instructions

- **conditional trap**
  - teq, tne,…
- **break**
  - Terminate program execution with exception
- **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

How to trigger the trap?  What's the output if the trap is triggered?
Will the '\ndata over' be printed out or not ?

```
.data
        dmsg:   .asciiz "\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:    .asciiz "\nTrap generated"
```

# Lab Assignment（23:55 April 2, Tuesday）

1. Implement an arithmetic calculator which can conduct addition and multiplication on two integers, which is input by the user.
    1. In the following situation an exception will be triggered:
        1. the addition overflow
        2. the multiplication result exceeds the width of a word
    2. The exception handler(‘trap’ is sugguested) should 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)”, and the cause of the exception (the sum is overflow, the product is bigger than the Max value of a word)
        3. exit the program

# Sample Result

Welcome to use the simple arithmetic calculator on unsigned 31bit number:
Please input operator: +
Please input addend: 2147483647
Please input augend: 2147483647
Runtime exception at 0x4000…. ,the sum is overflow

Welcome to use the simple arithmetic calculator on unsigned 31bit number:
Please input operator: +
Please input addend: 15
Please input augend: 20
The sum of 15 and 20 is : 35

Welcome to use the simple arithmetic calculator on unsigned 31bit number:
Please input operator: *
Please input multiplicand : 2147483647
Please input multiplier : 2147483647
Runtime exception at 0x4000… ,the product is bigger than the Max value of a word

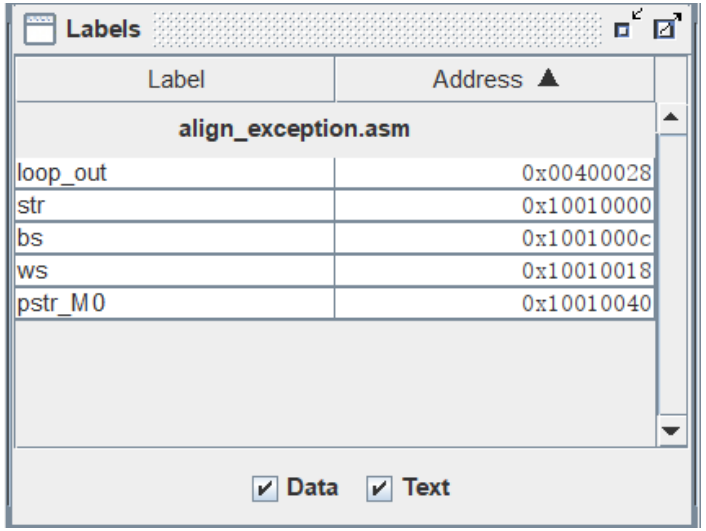
Welcome to use the simple arithmetic calculator on unsigned 31bit number:
Please input operator: *
Please input multiplicand : 15
Please input multiplier : 2
The product of 15 with 2 is : 30

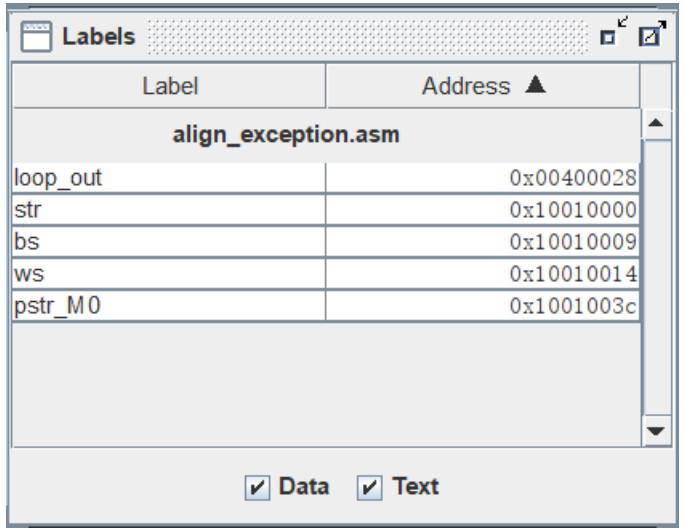
Welcome to use the simple arithmetic calculator on unsigned 31bit number:
Please input operator: /
The  operator / is not supported ,exit

# Tips：usage of ‘.align’

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

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




Data Segment

| 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 |

Data Segment

| 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*.