The Basics of Exception Handling

MIPS uses two coprocessors: C0 and C1 for additional help. C0 primarily helps with exception handling, and C1 helps with floating point arithmetic. Each coprocessor has a few registers.

Interrupts

Initiated outside the instruction stream

Arrive asynchronously (at no specific time),

Example:

- I/O device status change
- I/O device error condition

Traps

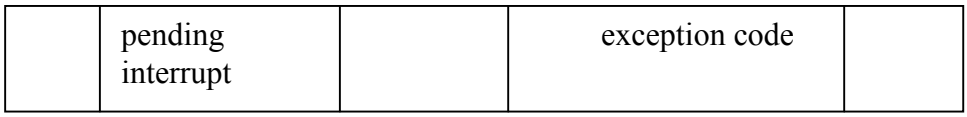
Occur due to something in instruction stream.

Examples:

- Unaligned address error
- Arithmetic overflow
- System call

MIPS coprocessor C0 has a cause register (Register 13) that contains a 4-bit code to identify the cause of an exception

Cause register

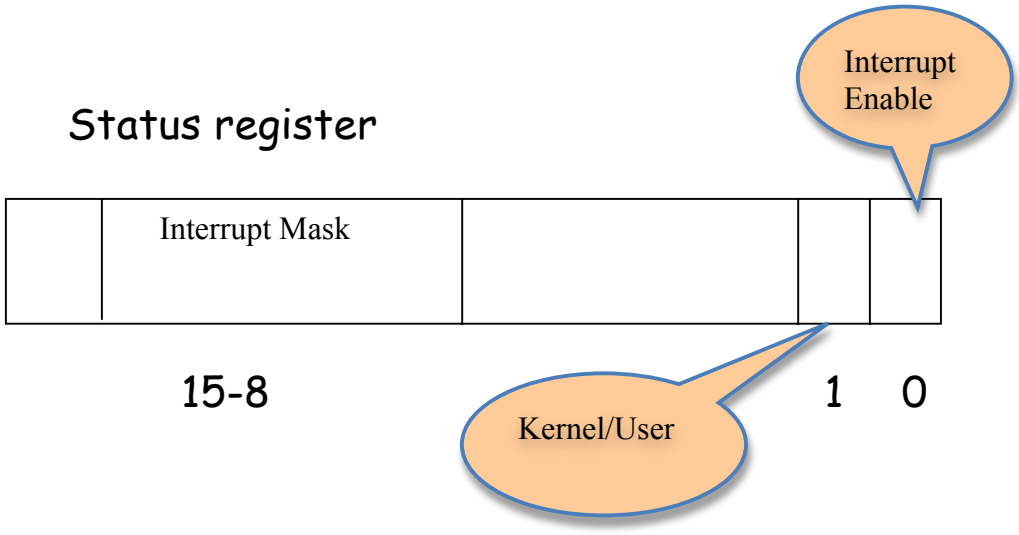


[Exception Code = 0 means I/O interrupt

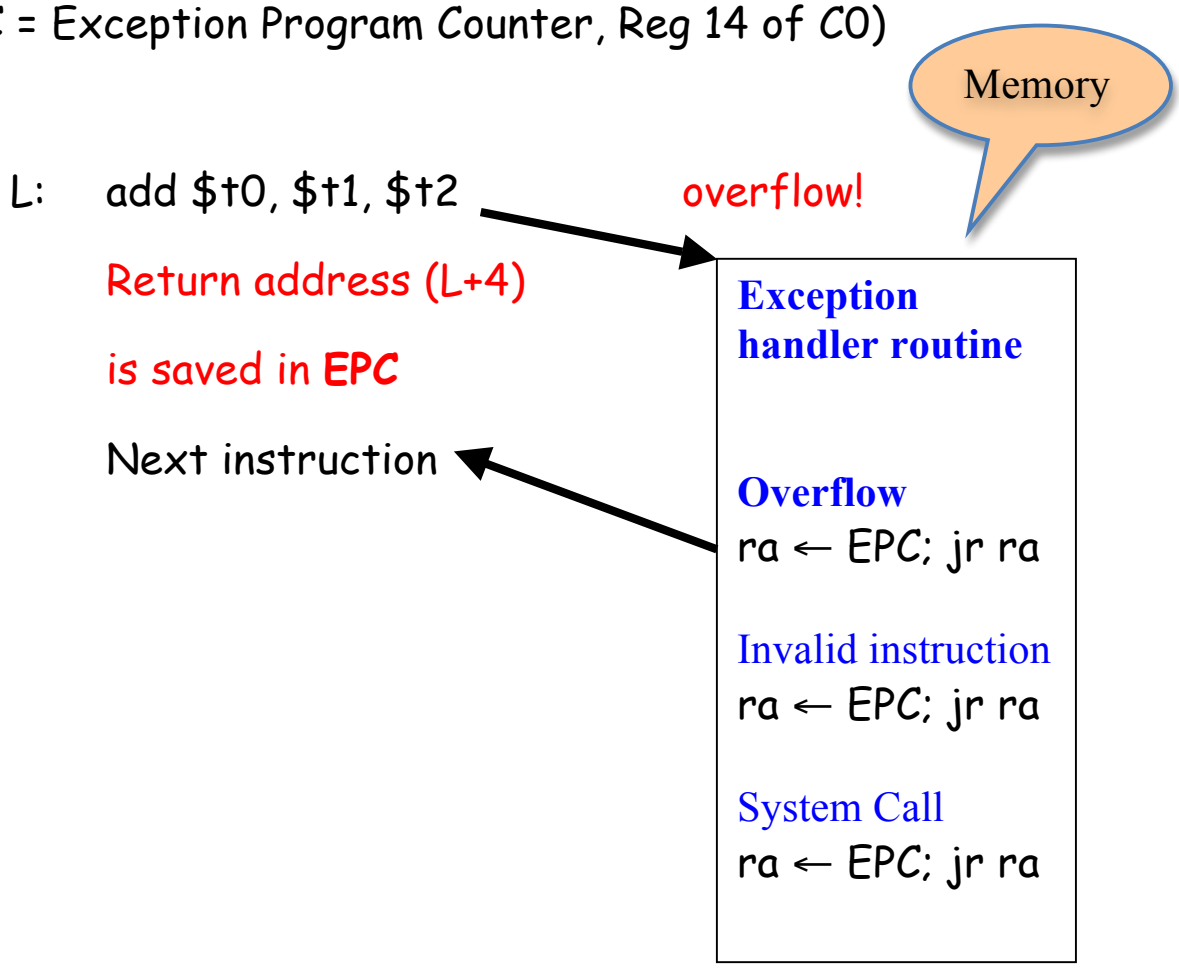
= 12 means arithmetic overflow etc]

MIPS instructions that cause overflow (or some other violation) lead to an ***exception***, which sets the exception code. It then switches to the kernel mode (designated by a bit in the status register of C0, register 12) and transfers control to a predefined address to invoke a routine (exception handler) for handling the exception.

Status register



(EPC = Exception Program Counter, Reg 14 of C0)



The Exception Handler determines the cause of the exception by looking at the exception code bits. Then it jumps to the appropriate exception handling routine. Finally, it returns to the main program.

Exceptions cause mostly unscheduled procedure calls.

Example: Read one input from a Keyboard

Consider reading a value from the keyboard. Assume that the interrupt enable bit is set to 1. The first line, ".text 0x80000080" places the code explicitly at the memory location where the *interrupt service routine* is called.

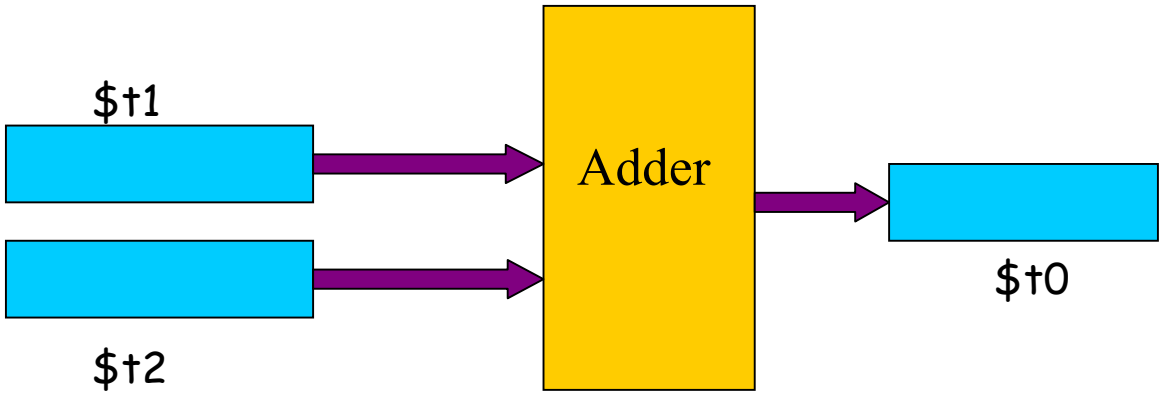
```
.text       0x80000080
            mfc0 $k0, $13          # $k0 = $Cause;
            mfc0 $k1, $14          # $k1 = $EPC;
            andi $k0, $k0, 0x003c # $k0 &= 0x003c (hex);
                                   # Filter the Exception Code;
            bne $k0, $zero, NotIO # if ($k0 ≠ 0) go to NotIO
                                   # Exception Code 0 => I/O instr.
            sw $ra, save0($0)      # save0 = $ra;
            jal ReadByte           # ReadByte(); (Get the byte).
            lw $ra, save0($0)      # $ra = save0;
            jr $k1                 # return;
NotIO:      Other routines here
```

Note that procedure ReadByte must save all registers that it plans to use, and restore them later.

Understanding Logic Design

Appendix C of your Textbook on the CD

When you write add $t0, $t1, $t2, you imagine something like this:

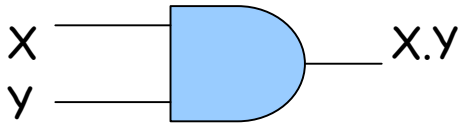


What kind of hardware can ADD two binary integers?

We need to learn about GATES and BOOLEAN ALGEBRA that are foundations of logic design.

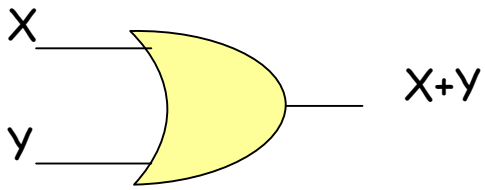
AND gate

X	y	X.y
0	0	0
0	1	0
1	0	0
1	1	1



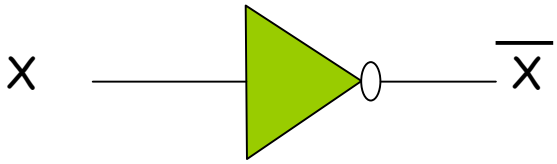
OR gate

X	y	X+y
0	0	0
0	1	1
1	0	1
1	1	1



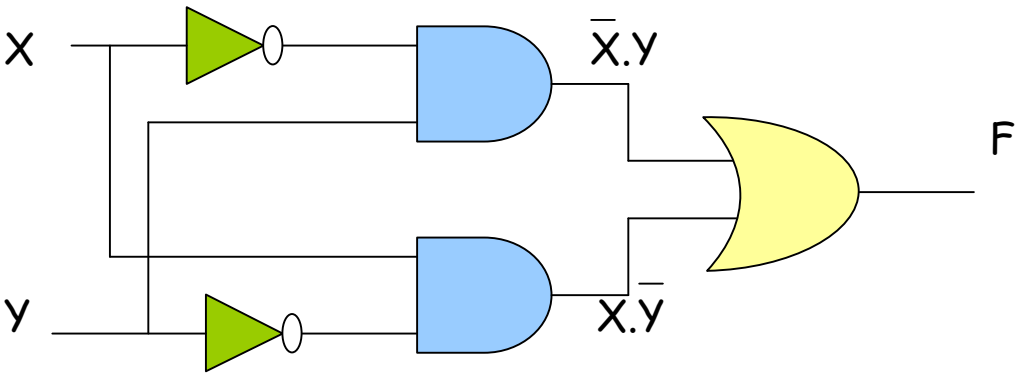
NOT gate

X	$\overline{X}$
0	1
1	0



Typically, logical 1 = +3.5 volt, and logical 0 = 0 volt. Other representations are possible.

Analysis of logical circuits



What is the value of F when X=0 and Y=1?

Draw a truth table.

X	Y	F
0	0	0
0	1	1
1	0	1
1	1	0

This is the **exclusive or** (XOR) function. In algebraic form $F = \bar{X}.Y + X.\bar{Y}$

More practice

1. Let $\bar{A}.B + A.C = 0$. What are the values of A, B, C?
2. Let $(A + B + C).(\bar{A} + \bar{B} + C) = 0$. What are the possible values of A, B, C?

- Draw truth tables.
- Draw the logic circuits for the above two functions.