Revision Setting Bits in words

Suppose we have a bit pattern `01010000` = `0x50` how do we set the bottom bit to 1?

Remember the truth table for `or`

$$
\begin{aligned}
0 \texttt{ or } 0 &= 0\\
1 \texttt{ or } 0 &= 1\\
0 \texttt{ or } 1 &= 1\\
1 \texttt{ or } 1 &= 1
\end{aligned}
$$

Thus to set the first bit of a bit pattern `01010000` = `0x50` we simply use `or` :

`01010000 or 00000001`

is equal to

`01010001`

Remember the `or` instruction works bitwise on numbers. So to set the third but of a binary number `x` we simple set `x` to

$$\texttt{x or } 0x04$$

You can use this method in a high-level language as well.

Testing bits

How do we test the if bit 3 is set in a particular number?

Remember the truth table for `and`

$$
\begin{aligned}
0 \texttt{ and } 0 &= 0\\
1 \texttt{ and } 0 &= 0\\
0 \texttt{ and } 1 &= 0\\
1 \texttt{ and } 1 &= 1
\end{aligned}
$$

Thus if we and number with `0x04`

`01010000 or 00000100`

we will get zero if bit 3 is set and `0x04` otherwise.

`01011100 or 00000100`

Exceptions and interrupts Remember if we use the `add` instruction to add two numbers and we get a number which is too big to represent in 32 bits we get an overflow.

On the mips processor an exception it thrown. An exception causes the processor to jump to a special piece of code call an exception (interrupt) handler which deals with the problem.

This is much like exceptions in high level languages:

```
try {
 /* Code that might raise an exception */
catch (expection_object) {
 /* Code to deal with the exception */
}
```

In C++ and many high-level languages there is no automatic exception mechanism for arithmetic overflow.

(Use ADA instead.)

Interrupts are the same as Exceptions Most modern processors (including the MIPS) allow external hardware to send an interrupt signal to the processor which forces the processor to finish executing the current instruction and go to an interrupt handler, and then possibly go back to executing the current program.

Every processor has a different convention about handling interrupts. Tobias has given you more than enough hints on how to write interrupt handlers, you *must* read the material you have been given.

Types of I/O handling There are two standard ways of dealing with I/O. We shall look at polled I/O and interrupt driven I/O.

Suppose I'm expecting a vistor to my house. The block of flats where I live the main locks at 9pm and there is no doorbell or intercom system at the main door.

Also suppose my vistor doesn't have a mobil phone. How do I know when to let my vistor in?

I don't want to wait outside the front door until he arives (not being Swedish he is not very punctual).

Luckily I can see the front door from my window.

One method, I could look out of the window every ten minutes and see if I can see my visitor and then let him in (polling).

The otherway we could agree on a protocol where by I'm *interrupted* by my vistor throwning a stone at the window and then I let him in.

What are the advantages and disadvantages of both methods?

Suppose my visitor didn't know when he was coming, he just said he would pop in. Which method would you use?

Suppose I'm throwing a party and I had a lot of people arriving between 8 and 8:30 which method would you use?

Synchronous and Asynchronous Protocols

Many I/O devices are use Synchronous protocols. A synchronous I/O device includes a clock and control lines and some protocol which is fixed with respect to the clock. That is at certain clock cycles certain values will be ready. Often many devices are connected together togther on the same bus, thus every device has to use the same clock speed.

An Asynchronous protocol requires the processor to ask the device for a value and wait for the device to return a that value.