

Re: Universal grammar

Source: <http://sci.tech-archive.net/Archive/sci.lang/2006-11/msg00832.html>

- *From:* "Paul J Kriha" <paul.nospam.kriha@xxxxxxxxxxxxxxxx>
 - *Date:* Mon, 13 Nov 2006 19:36:56 +1300
-

LEE Sau Dan <danlee@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx> wrote in message
<news:87ac2wzllj.fsf@xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx>

"Paul" ==
Paul J Kriha
<paul.nospam.kriha@xxxxxxxxxxxxxxxx>
writes:

>>>> 1) RESET cannot be caught or masked.
>>>> 2) INTERRUPT is caught, handled and then the system resumes
>>>> from it. RESET abruptly brings to the whole system to a
>>>> "start" state and never resumes from where it left off.
>>>> 3) INTERRUPT is used by peripherals to catch the attention of
>>>> the CPU, > > asking CPU to do something for them quickly. >
>>>> > RESET is usually raised by operators (often human) to bring
>>>> the > > system to a "start" state.

Paul> This is a software engineers vocabulary. The distinction
Paul> between interrupts made in software-engineer-speak is
Paul> artificial. No such distinction actually exists between h/w
Paul> reset and other h/w interrupts.

There is a difference. (see below)

No

Paul> Technically speaking, there is no difference what-so-ever
Paul> between how a CPU reacts to signals on various interrupt
Paul> lines. Reset interrupt, timer interrupt and various other
Paul> hardware device interrupts cause the CPU interrupt whatever
Paul> it is doing and jump at a predetermined hardwired
Paul> address.

It's not hardwired. It's programmable. The address to jump to is
stored in memory, and the CPU may modify it. That's not what
"hardwired" means.

Re: Universal grammar

Oh, fluf. Of course the address of the actual interrupt vector **MUST BE HARDWIRED** in the CPU. When an interrupt signal reaches it, th CPU finishes the current instruction and if NOT prevented by interrupt masking it slavishly jumps at a hardwired address in the interrupt vector table. The contents of the interrupt vector table are in turn programmable which means we can define where the actual interrupt server support code resides.

If you are using IBM compatible PC, go to Accessories, System Tools, and start System Information. Expand "Components", click on "Ports". Look at, for example, IRQ4 (interrupt pin number 4). IRQ4 is dedicated to the serial interface COM1:

Since the beginning of time (early eighties) its `_hardwired_ eight` byte interrupt vector is sitting at addresses `x03F8` to `x03FF`. Remember, `x03F8` is `_hardwired_` in the CPU!!!

When COM2 serial communications device drops the voltage on the interrupt pin IRQ3, the CPU will use hardwired addresses `x02F8`–`x02FF` to tell it where the actual code to serve this interrupt is. Remember, `x02F8` is `_hardwired_` in the CPU!!!

Paul> At that level the main (and often the only) difference
Paul> between various interrupts is the actual code stored at
Paul> those addresses (they usually happen to be further jump
Paul> instructions to various blocks of codes implementing the
Paul> various h/w servers).

There is a big difference between interrupts and reset signals.

For interrupts, the CPU has to prepare for resuming from it. Thus, before jumping to the address of the interrupt handler, the CPU has to save the address of the current (or next) instruction (the return address) somewhere (usually on the execution stack), so that after handling the interrupt, a specific instruction can cause the CPU to **resume** to the previously executing instruction (actually the next one).

It depends on which CPU we talking about but these days the actual functionality is almost exactly the same as executing a common subroutine jump. Usually the CPU invokes the same internal microcode to store current context on the stack and so on. Some architectures have a separate interrupt stack to make sure that the CPU doesn't run out of stack memory while attempting to service a critical interrupt.

Re: Universal grammar

Paul> If you manage to change these arrays of jump instruction,
Paul> aka interrupt vectors, you modify the function of the
Paul> interrupts lines. So on many old as well as new CPUs you
Paul> could swap functionality of reset line and say disk channel
Paul> interrupt. (Not that one would normally want to do that).

Paul> In fact we can view the whole machine's memory as the reset
Paul> interrupt's interrupt server.

No. A RESET signal is different. A RESET causes the CPU to clear its register contents, reset the "address for next instruction" to some *hardwired* value, and execute instruction from that address onward. No return address is saved onto the stack, because the CPU is *not* expected to *resume* from a RESET signal. (Actually, the stack is also reset by clearing the stack-pointer register. So, it's meaningless to store a return address.)

Some of what you say here is done by the init code in the PROM not by CPU internal microcode as you imply. It's totally unnecessary to waste expensive microcode memory to perform tasks unique to RESET interrupt which get performed only once in a blue moon.

So, there are major differences between these 2 kinds of signals:

Execution flow:

RESET: CPU jumps to a hardwired memory and starts execution there.

INTERRUPT: CPU *saves* the current address onto stack, and then jumps to a (usually) programmable location to continue execution.

and where is such programmable location being defined?

The interrupt handler can resume from the interrupt by executing a specific instruction that causes the CPU to read the saved address back from the stack, and jump to that address.

Purpose:

RESET: It is usually a system-wide signal. Not only the CPU but also peripherals react to this signal and go to an "initial" state.

This is all pure fantasy.

There is no system-wide RESET wire going to all peripherals.

INTERRUPT: It is usually a one-way communication channel with which peripherals asks for attention from the CPU. The CPU won't send interrupts to peripherals. Peripherals won't send interrupts

Re: Universal grammar

to one another.

None of this is (absolutely) true.

Lee Sau Dan

pjk

.