**CG1112 Engineering Principles and Practices II for CEG**

**Week 5 Tutorial Part 2 – Interrupts**

Question 1.

What are hardware interrupts? Why are they needed? Give some examples of hardware interrupts in your laptop or PC.

A hardware interrupt consists of a series of request lines built into the CPU or MCU. Its purpose is to let hardware signal to the CPU to get its attention.

The CPU can be busy running code, but when an interrupt request line is triggered (either by a rising edge, falling edge or change in signal level), the CPU stops what it is doing, and runs a special piece of code called an interrupt service routine (ISR) to handle the interrupt. The CPU then resumes what it was doing, where it left off.

Examples on a PC or laptop include interrupts from the keyboard when someone has pressed a key, or from the network interface when a frame of data has come in over the network, etc.

Similarly, what are software interrupts? Give some examples of how software interrupts can be used.

A software interrupt is an interrupt that is triggered using a special machine instruction, rather than a hardware line. Intel machines, for example, use the INT command to trigger software interrupts:

INT <interrupt number>

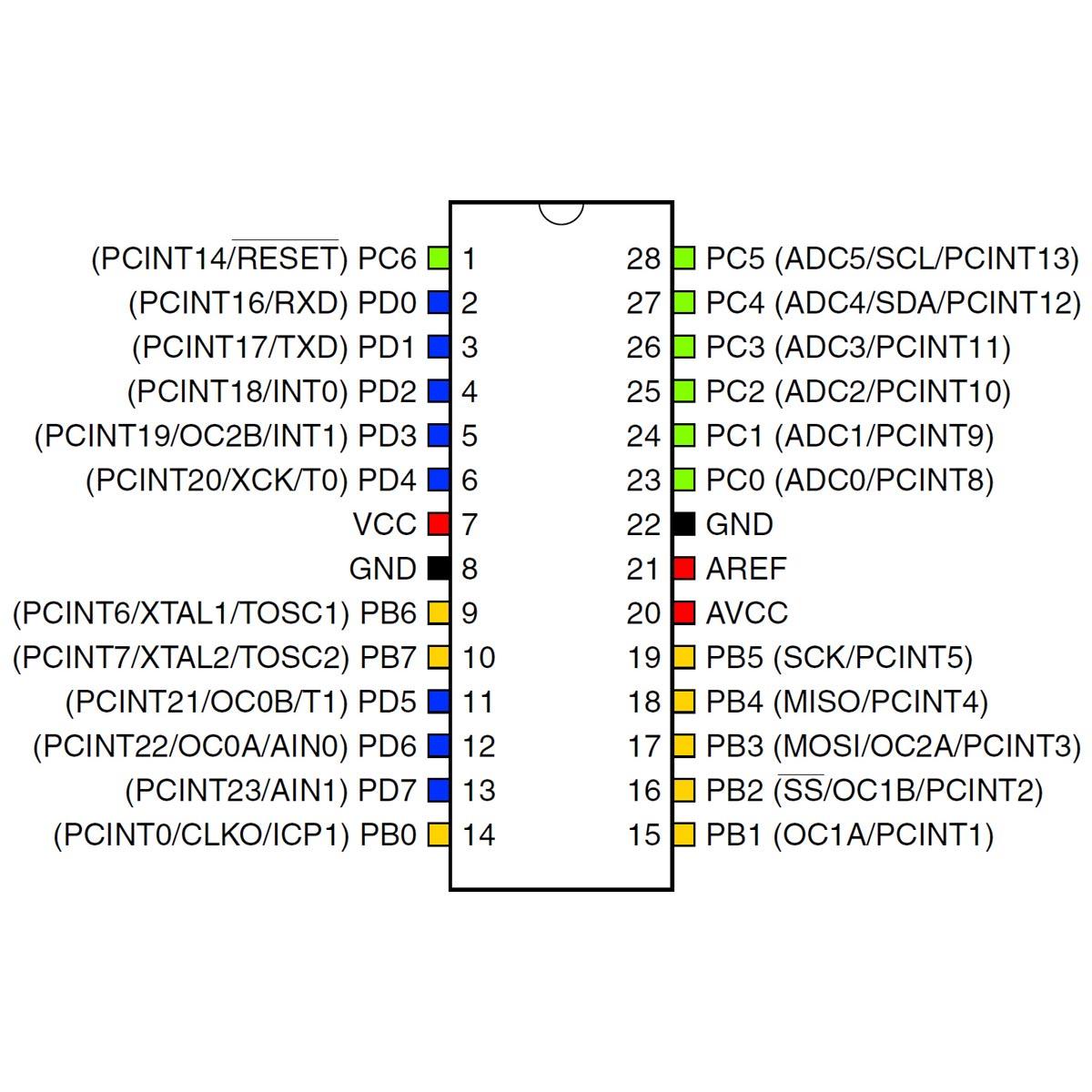
Here <interrupt number> is an interrupt identification number (analogous to the interrupt request lines in hardware interrupts), ranging from 0 to 255. Like hardware interrupts, software interrupts cause the CPU to jump to an ISR, with a different ISR for each of the 256 possible software interrupts. The address of each ISR is flexible and the CPU can be configured to jump to the correct ISR.

Software interrupts are very useful for providing flexible entry points for software APIs (application programming interfaces).

For example, on the LINUX operating system, if a program wishes to make a request to the OS (e.g. to write to the screen), it can do this by executing INT 128 (INT 80h, if you understand hexadecimal). The flexibility in setting the ISR’s address for INT 128 means that it is possible for LINUX programmers to change the entry point of the OS, without affecting programs written for LINUX.

Question 2.

What interrupt request lines are available on the Atmega328P? Describe these lines and how they are used.



There are two main types of interrupt request lines:

* Two external interrupt lines INT0 and INT1 (PD2 and PD3)
* 23 “pin change” interrupt request lines PCINT0 to PCINT23 (note: No PCINT15).

INT0 and INT1 are much more flexible. They can be triggered by:

* A low signal level
* A change in signal level
* Rising edge
* Falling edge

In addition INT0 and INT1 each have their own ISR.

The pin change interrupt requests (PCINT0 to PCINT23) only respond to changes in voltage levels. In addition, the 23 PCINT lines are grouped into 2 groups of 8 lines each and one group of 7 lines. All lines in the same group will trigger the execution of the same ISR. I.e. there are only 3 unique ISRs even though there are 23 lines.

Question 3.

Discuss how hardware interrupts are implemented. In particular, talk about how hardware interrupts are detected, how the CPU/MCU decides with ISR to run, how control is handed over to the ISR, and how execution resumes normally after the ISR has completed execution.

Detection:

* Every interrupt request line is assigned an index number.
* The CPU tests the state of the interrupt request lines many times a second. (On the MIPS range of microprocessors, for example, the CPU tests the lines at the end of each instruction execution).
* When one line is detected to have been triggered, the CPU takes note of its index number.
* The CPU consults a table called the Interrupt Vector Table using the index number of the triggered line. This table tells the CPU where the ISR for this particular line is. (This table is set up at power-up with the correct ISR addresses. Like-wise the ISRs are actually loaded into the addresses indicated. This is usually done by the OS)
* The CPU saves the contents of the Program Counter (PC) onto the “process stack” – a data structure similar to what you have learnt in CS2040C, which tells the CPU where to get the next program instruction for execution.
* The CPU loads the address of the ISR into PC, causing it to execute the ISR code.
* The ISR ends with a Return from Interrupt (RETI) instruction, that causes the CPU pop the stack containing the previous PC value into PC. This causes execution to resume at the point of interruption.

Nested interrupts:

* What if an interrupt occurs while processing another interrupt?
  + Interrupts have priority levels.
  + If the new interrupt has a lower priority than the current interrupt, it is ignored until processing for the current interrupt completes.
  + If the new interrupt has a higher priority than the current one, PC is again saved on the process stack, and the vector for the new interrupt is loaded into PC, causing it to execute the new ISR.
  + When the new interrupt exits, PC is popped off the stack, causing the previous ISR to resume.