Programming the 8259 PIC: A Tech-Tip Example and Boilerplate

Thomas W. Jenkins
Associate Professor, New Mexico State University,
Department of Engineering Technology
PO Box 3001, MSC 3566, Las Cruces, NM 88003-8001, tjenkins@nmsu.edu

Abstract - *The Intel® 8259 Programmable Interrupt Controller (PIC) is a common electronic device whose many and varied applications include integrating it into simple Intel based microcomputer systems. This device is also frequently used in Engineering Technology classes when discussing computer hardware or microcomputer system architectures and interfacing. Regularly these devices are utilized in the application portion of these classes or in student projects. The author presents an example of the programming of this device in MASM – an assembly language. This can be used as a boilerplate for describing the device's actions or in the use of this device in electronic or computer hardware laboratories or other applications.*

Introduction:

Most microcomputer systems designed today require that many I/O devices such as keyboards, communication ports, sensors, displays, and other components interface to the central processing unit (CPU) via a method call interrupts. In this method, the interfacing device (or sensor, etc.) requests service from the processor by generating an asynchronous signal to the CPU called an interrupt. The CPU will respond to the interrupt signal by completing its current instruction execution, saving the CPU state, and then respond to this request by executing a small program called an Interrupt Service Routine (ISR) that is specific to the requesting device. After completing the ISR, the CPU returns to the program it was executing before servicing the interrupting device via the ISR.

Most microprocessor CPUs typically have one input pin (INT) which when asserted causes the CPU to execute microcode to respond to this signal. Specifically, the CPU will automatically execute the following sequence of events via this microcode when a TRUE is detected on the INT input pin of the CPU.

1. The CPU completes its current instruction, which is in the instruction register.
2. The Code Segment (CS) and Instruction Pointer (IP) and the flags register values are placed on the stack thusly saving the "machine state".
3. The CPU sends a handshake signal out via a pin called the interrupt acknowledge (INTA)
4. After acknowledging the interrupt signal, the CPU looks on it's data bus for a numeric value N. This number N (0-255 decimal) is used to read an entry from the Interrupt Vector Table (IVT) which is located at the lowest memory locations 0-400H. At the Nth entry in the table, there is a four-byte value that is a pointer/address to a memory location where a short program resides – the ISR. The ISR is a machine code subroutine that is specifically designed to "service" the external hardware interrupt. The CPU places this four-byte value into its' CS and IP registers. Thus the CPU responds to this request for service by executing this small program that is specific to the requesting device. The particular ISR to be executed is determined by the value N which is read from the data bus.
5. The ISR is ended with a Return from Interrupt (IRET) instruction which permits the CPU to return to the code it was executing before the interrupt occurred. This is accomplished by popping the saved CS, IP, and flags from the stack before the CPU returns to the program it was executing before servicing the interrupting device.

Consequently, the technique of real-time programming using interrupts permits the CPU to be doing another task (executing another program) and only stop temporarily to execute an ISR when an external device (event, sensor, etc.) requests CPU action before returning to the original task.

With the CPU having only one INT pin, this would seem to limit the number of devices that could request an interrupt, to only one device. The Intel® 8259 Programmable Interrupt Controller (PIC), which is designed to be used in common real-time interrupt driven microcomputer systems, is specifically designed to handle this situation. The PIC, can **multiplex** up to eight prioritized hardware external interrupts, thusly allowing eight devices to request service asynchronously from the CPU. It can be cascaded for up to 64 vectored priority interrupts without additional circuitry. The PIC is designed to minimize overhead and additional software and hardware in handling multiple external interrupts.

The CPU and the PIC are easily hardware interfaced with many texts discussing the technique. Simply stated the CPU and PIC have a handshake interface with two lines: the PIC has an INTR line (output) which connects directly to the CPU INT line(input); while

the INTA line (output) from the CPU attaches to the INTA line (input) on the PIC. The PIC and CPU also have common connections via the shared data bus.

As the name "programmable" implies, the PIC can be programmed allowing a variety of implementations and applications. Typically, the primary usage is to permit multiplexing *multiple* devices that utilize interrupts (hard drive, printer, communication ports, floppy drive, external sensors, and user-defined devices) to use the interrupt techniques with a single microprocessor CPU. The programming of the PIC is usually done automatically as part of a microcomputer systems bootstrap or initialization routine that is contained in an operating system program. However, it is often useful to modify the standard PIC initialization to a non-typical application or it may be necessary to initialize the PIC to a specific task in small user-defined applications where large operating system programs are not needed nor even wanted.

For example, small microprocessor based systems are often used to collect data from a variety of sensors on a pipeline, irrigation system, photovoltaic array, or other instrumented applications. In this user-defined specialized application, a very limited microprocessor based system with minimal hardware is desired to minimize cost, weight, energy usage, and space. A large memory intensive operating system such as WINDOWS® is definitely a burden here. In these types of applications, user defined programming of the PIC is required.

Many microprocessor system architecture or interfacing classes in Engineering Technology programs have students design (and build) simple microprocessor based systems to demonstrate basic microcomputer architecture concepts or concepts relating to interfacing of the microprocessor to various devices such as memory, IO devices, sensors, or other similar applications. The PIC is often a main component of these systems and thus requires the instructor and students to understand the initialization of this device to function within this system. Intel technical manuals are often difficult for students to understand at their level of experience and many textbooks give only superficial treatment of this device. Therefore, the use of an example boilerplate and accompanying in-line documentation is offered as a supplement to these resources and as an aid in these activities.

The author has used this boiler plate in a senior Hardware Design and Microcomputer Architecture class that has students design, build, integrate, test, and document a single board microprocessor based computer system. Small assembly language routines are constructed to permit the hardware to do specified tasks such as display a real-time clock and function as a four-function calculator. The boiler-plate provided below is given to the students as a supplement to technical manuals, lectures, and text material over these devices. The boiler-plate approach gives good documentation on the function of the device and relieves the student of much of the burden of the coding, which permits more investment in learning interfacing and functionality.

Programming the 8259 PIC

The following provides an example of assembly language code, which initializes the PIC into an example typical configuration and provides an easily modifiable boilerplate for users to change the code to meet their specific requirements. The following is written in MASM code format and the author assumes a degree of assembly language, general programming, and PIC familiarity by the readers.

```
;*************************************************************************
; semicolon on any line begins a comment – nonexecutable documentation
;*************************************************************************
;              8259 Programmable Interrupt Controller (PIC)
;The PIC has two internal registers which we will program – they are
;generally accessed in Intel architecture via I/O ports 20 and 21 hex.
;NOTE: the use of equates (equ) is a common and useful form of assigning
;mnemonic names to constants and thus increasing the program's
;useful documentation.

PIC_P1        equ   20h   ;8259 control port: A0=0 (address line 0)
PIC_P2        equ   21h   ;8259 control port: A0=1


;In this example, we will…
;Initialize the 8259 PIC to handle three out of a possilbe eight
;hardware interrupts – these will be on the input lines labeled on the PIC
;as IR0, IR1, and IR7. These interrupts could be the timer, keyboard, hard-
;drive or some sensor input. The 8259 should be edge-triggered on these
;pins, single-mode, auto generate an End-of-Interrupt (EOI), 8088 CPU,
;with interrupt offset mask value of 8 – more on this below.

;This example requires three initialization control words: ICW1,ICW2 and
;ICW4, and one operational control word (OCW1) be sent to the device
;ports (equated to mnemonic names of PIC_P1 and PIC_P2 above.)
;These control words' port destination internal register destination is
;determined by the address bus line A0 value 0/1 which will directly
;correspond to decoding to the assigned ports 20 or 21Hex
;
;The PIC is programmed via operational command words (ICW or OCW). If
;we need to initialize the 8259 PIC controller into a single unit,
;edge triggered interupt controller, then the first control word (ICW1)
;performs the following initialization: (x indicates a don't care bit)
;
;X/0  X/0  X/0   1       0        1           1          1   =   17h
;{Not Used}     Must   Edge      {IVT         Single     Needs
;               be    Trigger   interval      8259       ICW4
;               a 1             of 4 bytes
;                               per vector}
;IVT = interrupt vector table
;
```

```
;
;The PIC has eight interrupt input pins IR0-IR7. The internal register at
;standard IO port 20H (PIC_P1 see above) holds a value which is added to
;the interrupt line which is activated to produce an Interrupt Number.
;This number is the true interrupt value used by the CPU. This
;modification of the interrupt line-to-interrupt number, allows for
;generating any eight consecutively numbered interrupts with a PIC as well
;as permitting the cascading of PIC's.
;Most commonly, the 8259 uses the value 8 as its interrupt number offset –
;therefore interrupt numbers 8-0Fh. These are most commonly used with a
;single PIC implementation and standard INTEL architecture.
;
;The second control word (ICW2) performs the following initialization:
; 0   0    0    0   1     X/0   X/0   X/0  (x=don't care)    =  08h
;     This is the number of the interrupt generated by the
;     IR0 pin (0+8=8), i.e. the interrupt offset number. Therefore, an
;      interrupt on the first pin IR0 (0) causes this
;     interrupt number 8 to be sent to the CPU via the data bus. This
;      number is multiplied by 4 by the CPU and that value is used
;     to find the corresponding ISR's address in the Interrupt Vector
;    Table (IVT). An interrupt on IR7 would thusly generater an interrupt
;    number of 7+8=15 decimal or 0FH(HEX)
;
;The third control word is not required because the 8259A PIC is being
;initialized as a single unit ONLY! (They can be cascaded)
;
;The fourth control word (ICW4) performs the following initialization:
;0      0       0        0       1          1       1      1  =  0Fh
;            Non     {Non-buffered       Auto      8088
;            SFNM            mode}       EOI       CPU
;Thusly,
ICW1          equ   17h   ;  command word 1 (ICW1)
ICW2          equ   08h   ;   "      word 2 (ICW2)
ICW4          equ   0Fh   ;   "      word 4 (ICW4)
;
;The first/only operation command word (OCW1) is the enable/disable mask
;for the eight interrupt lines: o=enable, 1=disable in the relative bit
;position for that interrupt. For example, int0=> bit 0 of the byte mask
OCW1          equ   01111100B ;8259 int. enable mask values to
                              ;enable interrupts  0,1,7
;             equ   7CH       ;enables IR0, IR1, and IR7 on the 8259
;
;
;*************************************************************************
;*************************************************************************
;                                INIT_8259
;*************************************************************************
;Initialize the 8259 PIC to handle three(3) hardware interrupts on lines
;IR0, IR1, and IR7. These interrupts could be the timer, keyboard,
;and some sensor input. The 8259 should be edge-triggered on these pins,
;single-mode, auto EOI, 8088 CPU, with interrupt offset mask value of 8.

;This example requires three initialization control words: ICW1, 2, and
;4, and one(1)operational control word (OCW1) be sent to the device
;ports. These control words port destination is determined by the address
```

```
;line A0 value 0/1 which will directly correspond to ports 20 or 21Hex

;It is recommended that the interrupt processing be disabled here then
;re-enabled at the end-of-this-code using the set/clear interrupt flag
;instructions: STI CLI
  cli                   ;turn off interrrupt processing while we
                        ;manipulate the PIC registers.
;Start to initialize the 8259A PIC by sending it the first control
;"word" (really 8 bits - 1 byte) as defined in the equates (equ) above.
;
 mov al,ICW1            ;load the 8259 PIC with the first
 out PIC_P1,al          ;initialization word.  A0=0 therefore port 20H
;
;Continue to initialize the 8259A PIC with its second control word.
;
 mov al,ICW2             ;load the 8259 PIC with the second
 out PIC_P2,al           ;initialization word. A0=1 therefore port 21H
;
;Finish initializing the 8259A PIC with the last control word.
;
 mov al,ICW4             ;load the 8259 PIC with the fourth
 out PIC_P2,al           ;initialization word. A0=1
;
;The 8259A PIC is now initialized as a single unit, edge triggered
;interrupt controller.
;
;Now send the 8259 then operational command word (OCW1) which will set
;the enable/disable bits for the three valid interrupts (0,1,7)
;
 mov al,OCW1             ;load the 8259 enable/disable IRQ mask
 out PIC_P2,al           ; A0=1


;It is recommended that the interrupt processing be re-enabled here using
;the set/clear interrupt flag instructions: STI and CLI
  sti                   ;turn ON interrrupt processing after we have
                        ;manipulated the PIC registers – allowing
                         ;the PIC to operate.
;
;done – the PIC is now initialized and ready to accept interrupts!!
;*************************************************************************
;*************************************************************************
;                               INIT_8259
;*************************************************************************
```

Conclusion:

The 8259 PIC is a versatile device that has many applications in electronics and microcomputer systems. Understanding their potential applications means understanding how to program the device to function in its different modes. An example boilerplate makes their programming an easier endeavor and permits their full potential in applications and as teaching tools.

References:

1. INTEL Hardware Design Homepage including specifications for the 8259 Programmable Interrupt Controller; http://developer.intel.com/
2. *The Intel Microprocessors: 8086/8088, 80186, 80286, 80386, 80486, Pentium, Pentium Pro, and Pentium II, Pentium III, Pentium 4*, Seventh Edition, Barry B. Brey, Prentice-Hall ©2005
3. *Embedded Controllers: 80186/80188 and 80386EX*, Barry B. Brey, Prentice-Hall, ©1998
4. *The 8088 and 8086 Microprocessors*, WA Triebel and A. Singh, Prentice Hall, 2000.
5. *The 80386, 80486, and Pentium Processors: Hardware, Software, and Interfacing*, Walter Tribel, Prentice Hall, 1998