80x86 Interrupts

Saidalavi Kalady

National Institute of Technology, Calicut

Interrupts

- In this lecture we examine a technique called INTERRUPT PROCESSING.
- An interrupt/exception is a HARDWARE/SOFTWARE INITIATED PROCEDURE that interrupts whatever program is currently executing.
- We provide examples and a detailed explanation of the interrupt structure of the Intel family of microprocessors.
- Interrupts are particularly useful when interfacing I/O devices that provide or require data at relatively low data transfer rates.

Purpose of Interrupts

- Before interrupts were invented, computers used a scheme called POLLING to interface with I/O devices.
- This process was such a tremendous waste of time that designers developed another process, INTERRUPT PROCESSING, to handle this situation.
- Unlike the polling technique, interrupt processing allows the microprocessor to execute other software while the keyboard operator is thinking about what key to type next.
- As soon as a key is pressed, the keyboard encoder debounces the switch and puts out one pulse that interrupts the microprocessor.
- The microprocessor executes other software until the key is actually pressed, when it reads a key and returns to the program that was interrupted.
- As a result, the microprocessor can print reports or complete any other task while the operator is typing a document and thinking about what to type next.

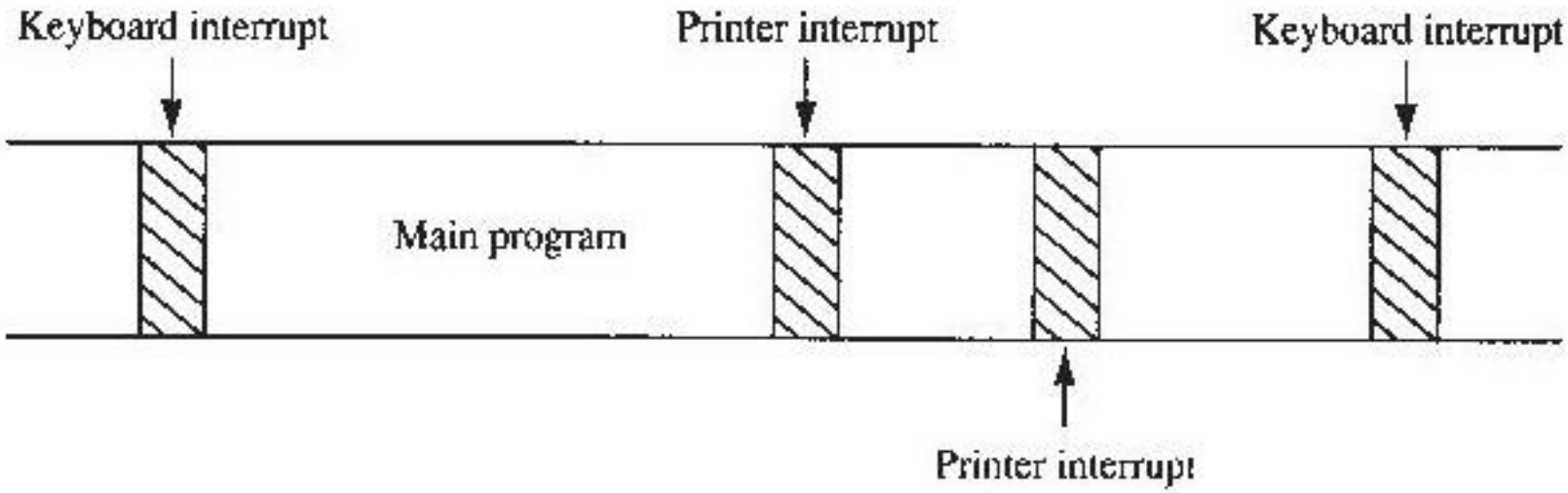


Figure: A time line that indicates interrupt usage in a typical system.

Interrupts in 80x86

- The interrupts of the entire Intel family of microprocessors include two hardware pins that request interrupts (INTR and NMI), and one hardware pin ($\overline{\text{INTA}}$) that acknowledges the interrupt requested through INTR.
- In addition to the pins, the microprocessor also has software interrupts INT, INTO , INT 3, and BOUND.
- Two flag bits, IF (interrupt flag) and TF (trap flag), are also used with the interrupt structure and a special return instruction, IRET (or IRETD in the 80386, 80486, or Pentium-Pentium 4).

Interrupt vectors

- The interrupt vectors and vector table are crucial to an understanding of hardware and software interrupts.
- The interrupt vector table is located in the first 1024 bytes of memory at addresses 000000*H*-0003*FFH*.
- It contains 256 different four-byte INTERRUPT VECTORS.
- An interrupt vector contains the address (segment and offset) of the INTERRUPT SERVICE PROCEDURE.
- The Figures in next two slides illustrate the interrupt vector table for the microprocessor.

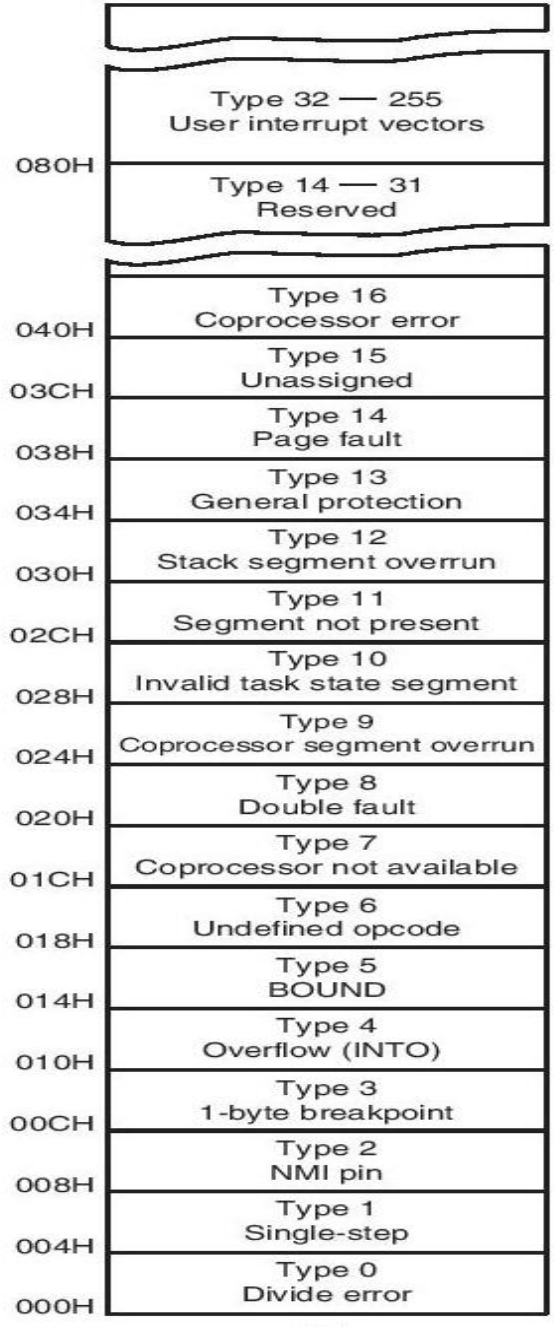


Figure: The interrupt vector table for the microprocessor

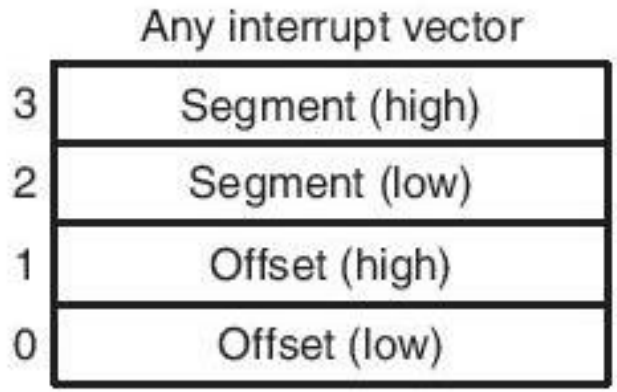


Figure: The contents of an interrupt vector.

Interrupt vectors

- The first five interrupt vectors are identical in all Intel microprocessor family members, from the 8086 to the Pentium.
- Other interrupt vectors exist for the 80286 that are upward-compatible to the 80386, 80486, and Pentium-Core2, but not downward-compatible to the 8086 or 8088.
- Intel reserves the first 32 interrupt vectors for their use in various microprocessor family members.
- The last 224 vectors are available as user interrupt vectors.
- Each vector is four bytes long in the real mode and contains the starting address of the interrupt service procedure.
- The first two bytes of the vector contain the offset address and the last two bytes contain the segment address.

Interrupt instructions

- Of the five software interrupt instructions available to the microprocessor, INT and INT3 are very similar, BOUND and INTO are conditional, and IRET is a special interrupt return instruction.
- The BOUND instruction, which has two operands, compares a register with two words of memory data.
- For example, if the instruction $BOUND AX, DATA$ is executed, AX is compared with the contents of $DATA$ and $DATA + 1$ and also with $DATA + 2$ and $DATA + 3$.
- If AX is less than the contents of $DATA$ and $DATA + 1$, a type 5 interrupt occurs.
- If AX is greater than $DATA + 2$ and $DATA + 3$, a type 5 interrupt occurs.
- If AX is within the bounds of these two memory words, no interrupt occurs.

Interrupt instructions

- The INTO instruction checks or tests the overflow flag (O).
- If $O = 1$, the INTO instruction calls the procedure whose address is stored in interrupt vector type number 4.
- If $O = 0$, then the INTO instruction performs no operation and the next sequential instruction in the program executes.

Interrupt instructions

- The INT n instruction calls the interrupt service procedure that begins at the address represented in vector number n.
- For example, an INT $80H$ or INT 128 calls the interrupt service procedure whose address is stored in vector type number 80H (000200H–00203H).
- To determine the vector address, just multiply the vector type number (n) by 4, which gives the beginning address of the four-byte long interrupt vector.
- For example, INT $5 = 4 \times 5$ or 20 (14H). The vector for INT 5 begins at address $0014H$ and continues to $0017H$.

Interrupt instructions

- Each INT instruction is stored in two bytes of memory: The first byte contains the opcode, and the second byte contains the interrupt type number.
- The only exception to this is the INT 3 instruction, a one-byte instruction.
- The INT 3 instruction is often used as a breakpoint-interrupt because it is easy to insert a one-byte instruction into a program.
- Breakpoints are often used to debug faulty software.

Interrupt instructions

- The IRET instruction is a special return instruction used to return for both software and hardware interrupts.
- The IRET instruction is much like a far *RET*, because it retrieves the return address from the stack.
- It is unlike the near return because it also retrieves a copy of the flag register from the stack.
- An IRET instruction removes six bytes from the stack: two for the IP, two for the CS, and two for the flags.

Interrupt instructions

- In the 80386–Core2, there is also an IRETD instruction because these microprocessors can push the *EFLAG* register (32 bits) on the stack, as well as the 32-bit *EIP* in the protected mode and 16-bit code segment register.
- If operated in the real mode, we use the IRET instruction with the 80386–Core2 microprocessors.
- If the Pentium 4 operates in 64-bit mode, an IRETQ instruction is used to return from an interrupt.
- The IRETQ instruction pops the *EFLAG* register into *RFLAGS* and also the 64-bit return address is placed into the *RIP* register.

Operation on real mode interrupts

- When the microprocessor completes executing the current instruction, it determines whether an interrupt is active by checking (1) instruction executions, (2) single-step, (3) NMI, (4) coprocessor segment overrun, (5) INTR, and (6) INT instructions in the order presented.
- If one or more of these interrupt conditions are present, the following sequence of events occurs:
  1. The contents of the flag register are pushed onto the stack.
  2. Both the interrupt (IF) and trap (TF) flags are cleared. This disables the INTR pin and the trap or single-step feature.
  3. The contents of the code segment register (CS) are pushed onto the stack.
  4. The contents of the instruction pointer (IP) are pushed onto the stack.
  5. The interrupt vector contents are fetched, and then placed into both IP and CS so that the next instruction executes at the interrupt service procedure addressed by the vector.

Operation on real mode interrupts

- Whenever an interrupt is accepted, the microprocessor stacks the contents of the flag register, CS and IP; clears both IF and TF; and jumps to the procedure addressed by the interrupt vector.
- After the flags are pushed onto the stack, IF and TF are cleared.
- These flags are returned to the state prior to the interrupt when the IRET instruction is encountered at the end of the interrupt service procedure.
- Therefore, if interrupts were enabled prior to the interrupt service procedure, they are automatically re-enabled by the IRET instruction at the end of the procedure.
- The return address (in CS and IP) is pushed onto the stack during the interrupt.

Operation on real mode interrupts

- Sometimes the return address points to the next instruction in the program; sometimes it points to the instruction or point in the program where the interrupt occurred.
- Interrupt type numbers 0, 5, 6, 7, 8, 10, 11, 12, and 13 push a return address that points to the offending instruction, instead of to the next instruction in the program.
- This allows the interrupt service procedure to possibly retry the instruction in certain error cases.
- Some of the protected mode interrupts (types 8, 10, 11, 12, and 13) place an error code on the stack following the return address.
- The error code identifies the selector that caused the interrupt.
- In cases where no selector is involved, the error code is a 0.

Operation on protected mode interrupts

- In the protected mode, interrupts have exactly the same assignments as in the real mode, but the interrupt vector table is different.
- In place of interrupt vectors, protected mode uses a set of 256 interrupt descriptors that are stored in an interrupt descriptor table (IDT).
- The interrupt descriptor table is 256×8 (2K) bytes long, with each descriptor containing eight bytes.
- The interrupt descriptor table is located at any memory location in the system by the interrupt descriptor table address register (IDTR).
- Each entry in the IDT contains the address of the interrupt service procedure in the form of a segment selector and a 32-bit offset address.
- It also contains the P bit (present) and DPL bits to describe the privilege level of the interrupt.
- The Figure in next slide shows the contents of the interrupt descriptor.

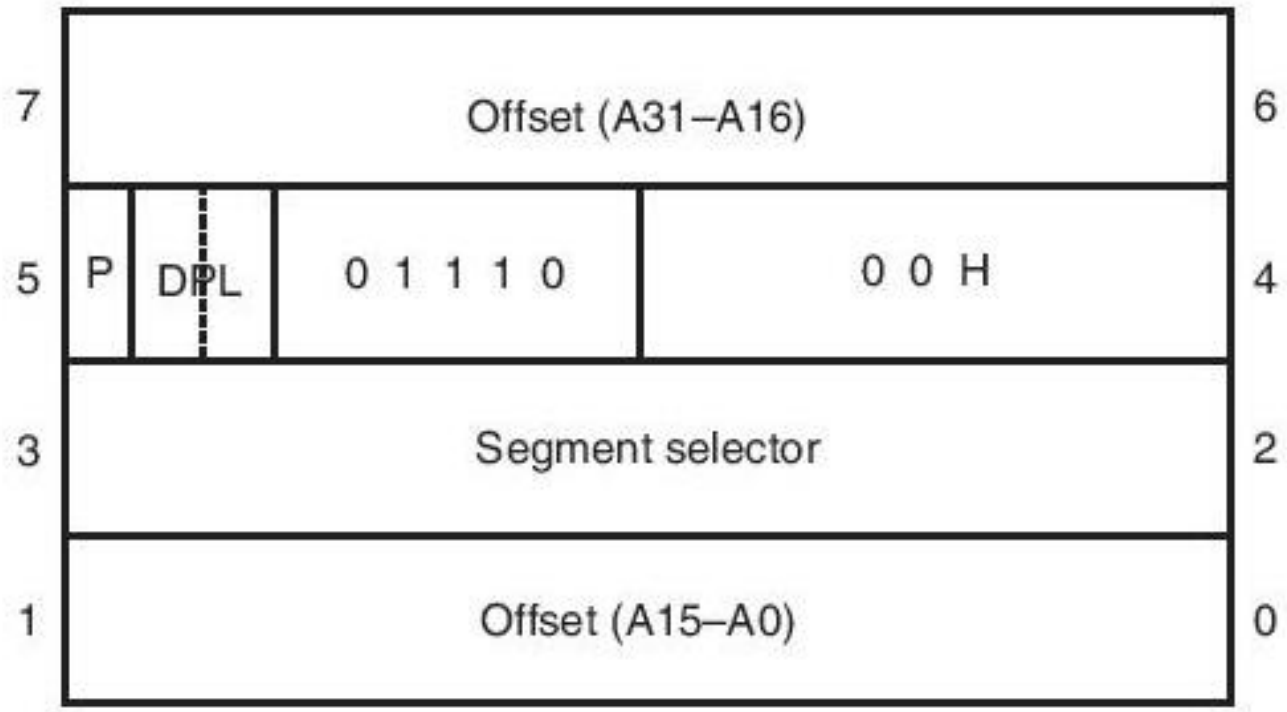
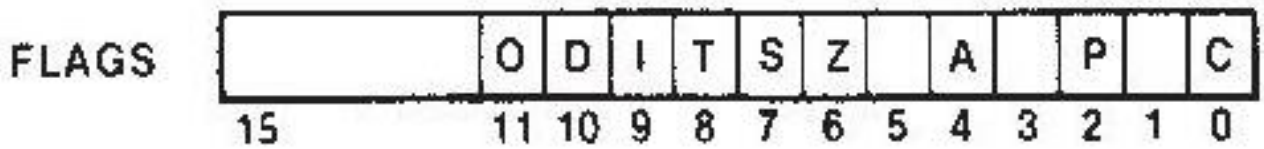


Figure: The protected mode interrupt descriptor.

Operation on protected mode interrupts

- Real mode interrupt vectors can be converted into protected mode interrupts by copying the interrupt procedure addresses from the interrupt vector table and converting them to 32-bit offset addresses that are stored in the interrupt descriptors.
- A single selector and segment descriptor can be placed in the global descriptor table that identifies the first 1M byte of memory as the interrupt segment.
- Other than the IDT and interrupt descriptors, the protected mode interrupt functions like the real mode interrupt.
- We return from both interrupts by using the IRET or IRETD instruction. The only difference is that in protected mode the microprocessor accesses the IDT instead of the interrupt vector table.
- In the 64-bit mode of the Pentium 4 and Core2, an IRETQ must be used to return from an interrupt. This is one reason why there are different drivers and operating systems for the 64-bit mode.

Interrupt flag bits



- The interrupt flag (IF) and the trap flag (TF) are both cleared after the contents of the flag register are stacked during an interrupt.
- The Figure above illustrates the contents of the flag register and the location of IF and TF.

Interrupt flag bits

- When the IF bit is set, it allows the INTR pin to cause an interrupt;
- when the IF bit is cleared, it prevents the INTR pin from causing an interrupt.
- When TF $= 1$, it causes a trap interrupt (type number 1) to occur after each instruction executes.
- This is why we often call trap a single-step.
- When TF $= 0$, normal program execution occurs.
- The interrupt flag is set and cleared by the STI and CLI instructions, respectively.
- There are no special instructions that set or clear the trap flag.

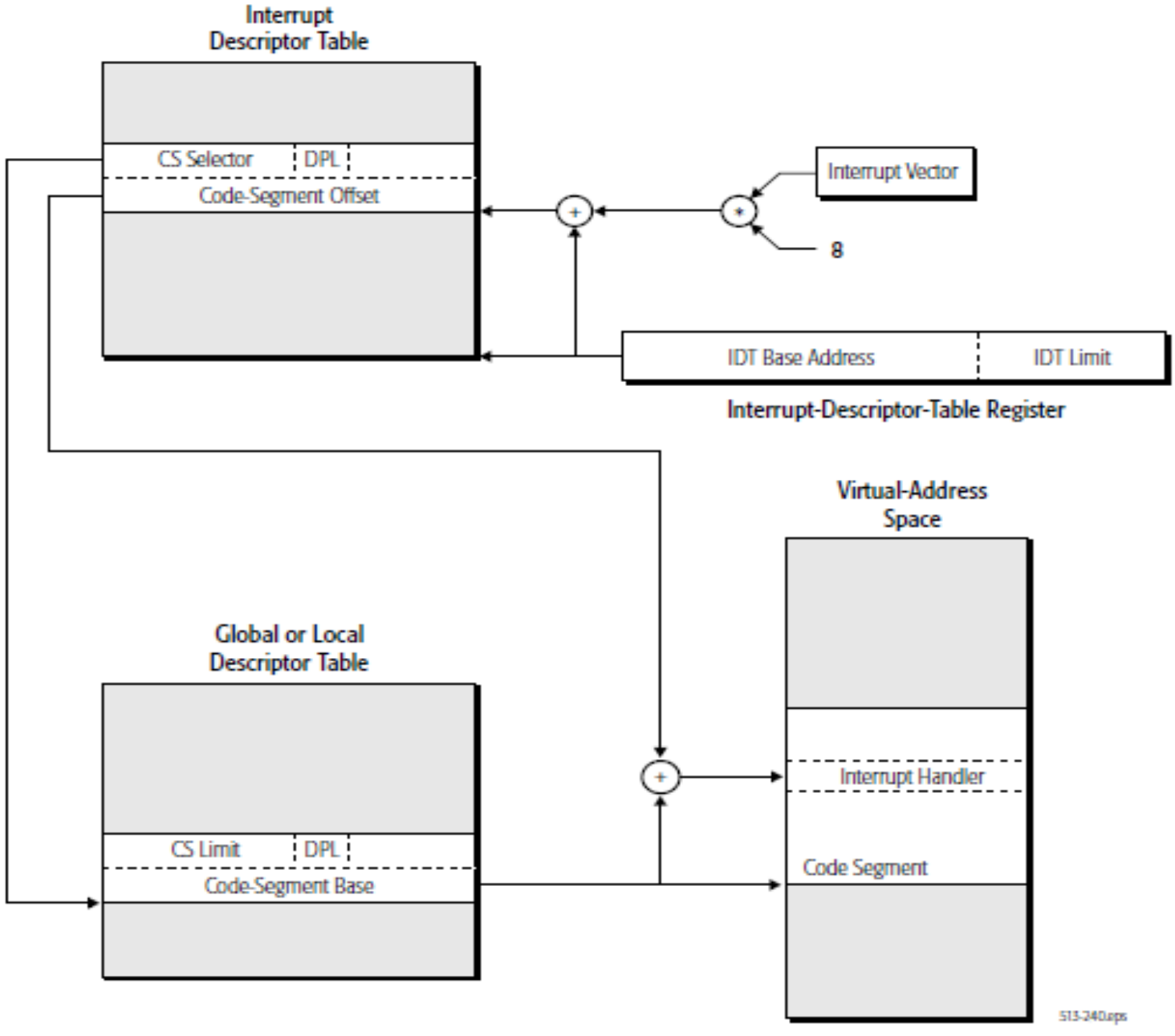


Figure: Protected Mode Interrupts

Hardware Interrupts

- The microprocessor has two hardware interrupt inputs: non-maskable interrupt (NMI) and interrupt request (INTR).
- Whenever the NMI input is activated, a type 2 interrupt occurs because NMI is internally decoded.
- The INTR input must be externally decoded to select a vector.
- Any interrupt vector can be chosen for the INTR pin, but we usually use an interrupt type number between 20H and FFH.
- Intel has reserved interrupts 00H through 1FH for internal and future expansion.
- The $\overline{\text{INTA}}$ signal is also an interrupt pin on the microprocessor, but it is an output that is used in response to the INTR input to apply a vector type number to the data bus connections D_7 - D_0.

Hardware Interrupts

- The Figure in the next slide shows the three user interrupt connections on the microprocessor.
- The non-maskable interrupt (NMI) is an edge-triggered input that requests an interrupt on the positive edge (0-to-1 transition).
- After a positive edge, the NMI pin must remain a logic 1 until it is recognized by the microprocessor.
- Note that before the positive edge is recognized, the NMI pin must be a logic 0 for at least two clocking periods.
- The NMI input is often used for parity errors and other major system faults, such as power failures.
- Power failures are easily detected by monitoring the AC power line and causing an NMI interrupt whenever AC power drops out.

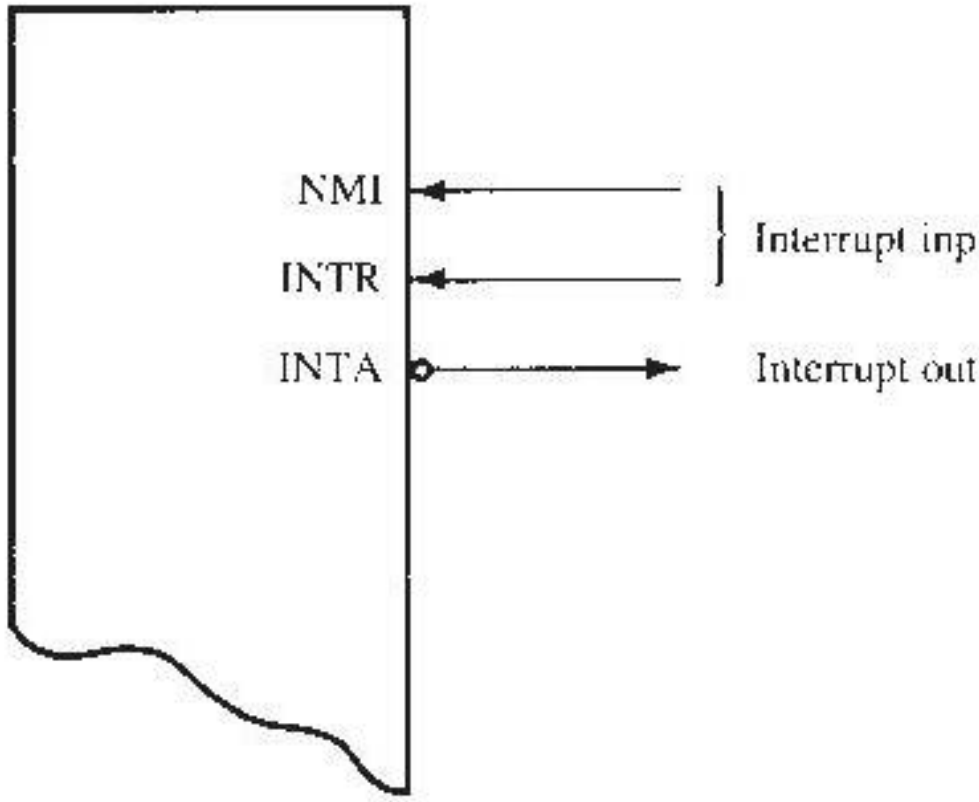


Figure: The interrupt pins on all versions of the Intel microprocessor.

INTR & $\overline{\text{INTA}}$

- The interrupt request input (INTR) is level-sensitive, which means that it must be held at a logic 1 level until it is recognized.
- The INTR pin is set by an external event and cleared inside the interrupt service procedure.
- This input is automatically disabled once it is accepted by the microprocessor and re-enabled by the IRET instruction at the end of the interrupt service procedure.
- The 80386–Core2 use the IRETD instruction in the protected mode of operation.
- In the 64-bit mode, an IRETQ is used in protected mode.
- The microprocessor responds to the INTR input by pulsing the $\overline{\text{INTA}}$ output in anticipation of receiving an interrupt vector type number on data bus connections D_7 - D_0.

8259A Programmable Interrupt Controller

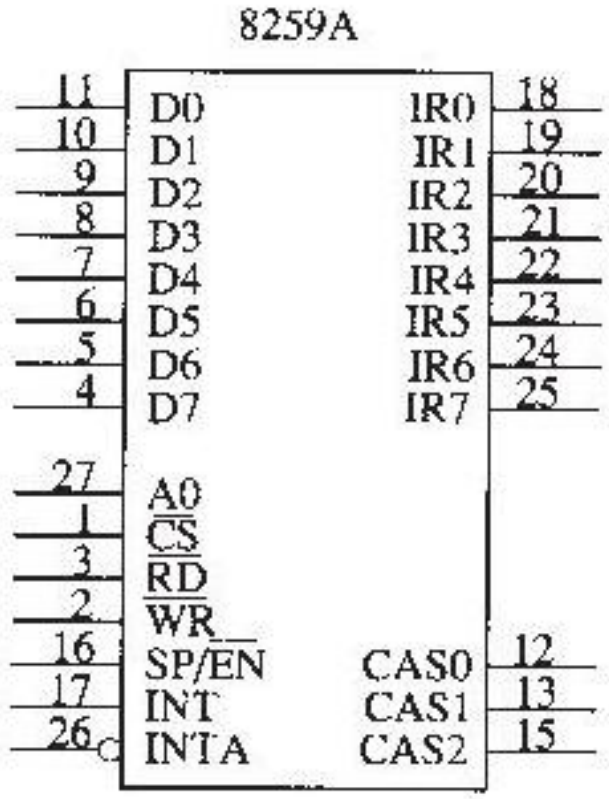


Figure: The pin-out of the 8259A programmable interrupt controller (PIC).

8259A Programmable Interrupt Controller

- The 8259A programmable interrupt controller (PIC) adds eight vectored priority encoded interrupts to the microprocessor.
- This controller can be expanded, without additional hardware, to accept up to 64 interrupt requests.
- This expansion requires a master 8259A and eight 8259A slaves.
- A pair of these controllers still resides and is programmed in the latest chip sets from Intel and other manufacturers.
- The 8259A is easy to connect to the microprocessor because all of its pins are direct connections except the $\overline{CS}$ pin, which must be decoded, and the $\overline{WR}$ pin, which must have an I/O bank write pulse.

Command words

- The 8259A is programmed by INITIALIZATION and OPERATION COMMAND WORDS.
- INITIALIZATION COMMAND WORDS (ICWs) are programmed before the 8259A is able to function in the system and dictate the basic operation of the 8259A.
- OPERATION COMMAND WORDS (OCWs) are programmed during the normal course of operation.
- The OCWs control the operation of the 8259A.
- The OCWs are used to direct the operation of the 8259A once it is programmed with the ICW.
- The OCWs are selected when the A_0 pin is at a logic 0 level, except for OCW_1, which is selected when A_0 is a logic 1.

Command words

- There are four initialization command words (ICWs) for the 8259A that are selected when the A_0 pin is a logic 1.
- When the 8259A is first powered up, it must be sent ICW_1, ICW_2, and ICW_4.
- If the 8259A is programmed in cascade mode by ICW_1, then we also must program ICW_3.
- So if a single 8259A is used in a system, ICW_1, ICW_2, and ICW_4 must be programmed.
- If cascade mode is used in a system, then all four ICWs must be programmed.

THANK YOU