INTERRUPT INTERFACE OF THE 8088 AND 8086 MICROPROCESSOR

11.1 Interrupt Mechanism, Types and Priority
11.2 Interrupt Vector Table
11.3 Interrupt Instructions
11.4 Enabling/Disabling of Interrupts
11.5 External Hardware-Interrupt Interface Signals
11.6 External Hardware-Interrupt Sequence
INTERRUPT INTERFACE OF THE 8088 AND 8086 MICROPROCESSOR

11.7 82C59A Programmable Interrupt Controller
11.8 Interrupt Interface Circuits Using the 82C59A
11.9 Software Interrupts
11.10 Nonmaskable Interrupt
11.11 Reset
11.12 Internal Interrupt Function

11.1 Interrupt Mechanism, Types and Priority

- Interrupts provide a mechanism for quickly changing program environment. Transfer of program control is initiated by the occurrence of either an event internal to the MPU or an event in its external hardware.
- The section of program to which control is passed is called the interrupt service routine.
- The 8088 and 8086 microprocessor are capable of implementing any combination of up to 256 interrupts.
- Interrupts are divided into five groups:
  - External hardware interrupts
  - Nonmaskable interrupts
  - Software interrupts
  - Internal interrupts
  - reset
11.1 Interrupt Mechanism, Types and Priority

- Interrupt program context switching mechanism

- Hardware, software, and internal interrupts are serviced on a priority basis.
- Each interrupt is given a different priority level by assigning it a type number. Type 0 identifies the highest-priority interrupt, and type 255 identifies the lowest-priority interrupt.
- Tasks that must not be interrupted frequently are usually assigned to higher-priority levels and those that can be interrupted to lower-priority levels.
- Once an interrupt service routine is initiated, it could be interrupted only by a function that corresponds to a higher-priority level.
11.1 Interrupt Mechanism, Types and Priority

- Types of interrupts and their priority

Increasing priority

- Reset
- Internal interrupts and exceptions
- Software interrupts
- Nonmaskable interrupts
- External hardware interrupts

11.2 Interrupt Vector Table

- An *address pointer table* is used to link the interrupt type numbers to the locations of their service routines in the program-storage memory.

- The address pointer table contains 256 address pointers (vectors), which are identified as vector 0 through vector 255. One pointer corresponds to each of the interrupt types 0 through 255.

- The address pointer table is located at the low-address end of the memory address space. It starts at $0000_{16}$ and ends at $003F_{16}$. This represents the first 1 Kbytes of memory.
11.2 Interrupt Vector Table

Interrupt vector table of the 8088/8086

**EXAMPLE**

At what address are CS\textsubscript{50} and IP\textsubscript{50} stored in memory?

Solution:

Each vector requires four consecutive bytes of memory for storage. Therefore, its address can be found by multiplying the type number by 4. Since CS\textsubscript{50} and IP\textsubscript{50} represent the words of the type 50 interrupt pointer, we get

Address = 4 \times 50 = 200

converting to binary form gives

Address = 11001000\textsubscript{2} = C8\textsubscript{16}

Therefore, IP\textsubscript{50} is stored at 000C8\textsubscript{16} and CS\textsubscript{50} at 000CA\textsubscript{16}. 
### 11.3 Interrupt Instructions

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Meaning</th>
<th>Format</th>
<th>Operation</th>
<th>Flags affected</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLI</td>
<td>Clear interrupt flag</td>
<td>CLI</td>
<td>$0 \rightarrow (IF)$</td>
<td>IF</td>
</tr>
<tr>
<td>STI</td>
<td>Set interrupt flag</td>
<td>STI</td>
<td>$1 \rightarrow (IF)$</td>
<td>IF</td>
</tr>
<tr>
<td><strong>INT n</strong></td>
<td>Type n software interrupt</td>
<td>INT n</td>
<td>$(\text{flags}) \rightarrow ((\text{SP})-2)$ $0 \rightarrow TF, IF$ $(\text{CS}) \rightarrow (\text{CS})$ $(2+4n) \rightarrow (\text{CS})$ $(\text{IP}) \rightarrow ((\text{SP}) - 6)$ $(4n+6) \rightarrow (\text{IP})$</td>
<td>TF, IF</td>
</tr>
<tr>
<td>IRET</td>
<td>Interrupt return</td>
<td>IRET</td>
<td>$(\text{SP}) \rightarrow (\text{IP})$ $(\text{SP})+4 \rightarrow (\text{CS})$ $(\text{SP})+6 \rightarrow (\text{IP})$</td>
<td>All</td>
</tr>
<tr>
<td>INTO</td>
<td>Interrupt on overflow</td>
<td>INTO</td>
<td>INT 4 steps</td>
<td>TF, IF</td>
</tr>
<tr>
<td>HLT</td>
<td>Halt</td>
<td>HLT</td>
<td>Wait for an external interrupt or reset to occur</td>
<td>None</td>
</tr>
<tr>
<td>WAIT</td>
<td>Wait</td>
<td>WAIT</td>
<td>Wait for TEST input to go active</td>
<td></td>
</tr>
</tbody>
</table>

### 11.4 Enabling/Disabling of Interrupts

- An **interrupt-enable flag** bit (IF) is provided within the 8088/8086 MPUs.
- The ability to initiate an external hardware interrupt at the INTR input is enabled by setting IF or masked out by resetting it. Executing the STI or CLI instructions, respectively, does this through software.
- During the initiation sequence of a service routine for an external hardware interrupt, the MPU automatically clears IF. This masks out the occurrence of any additional external hardware interrupts.
11.5 External Hardware-Interrupt Interface Signals

- Minimum-mode interrupt interface
  - Key interrupt interface signals: INTR and INTA

Minimum-mode 8088 and 8086 system external hardware interrupt interface

- Maximum-mode interrupt interface
  - 8288 bus controller is added in the interface. The INTA and ALE signals are produced by the 8288.
  - The bus priority lock signal LOCK is also added. This signal ensures that no other device can take over control of the system bus until the interrupt-acknowledge bus cycle is completed.

Maximum-mode 8088 and 8086 system external hardware interrupt interface
### 11.5 External Hardware-Interrupt Interface Signals

Maximum-mode interrupt interface

<table>
<thead>
<tr>
<th>Status inputs</th>
<th>CPU cycle</th>
<th>8288 command</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_2$</td>
<td>$S_1$</td>
<td>$S_0$</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Interrupt bus status code to the 8288 bus controller

---

### 11.6 External Hardware-Interrupt Sequence

```
COMPLETE CURRENT INSTRUCTION
  \rightarrow INTERNAL INTERRUPT?\rightarrow NMI
  \rightarrow INTR\rightarrow TF
  \rightarrow EXECUTE NEXT INSTRUCTION
    IF ACKNOWLEDGE INTERRUPT
    \rightarrow READ TYPE NUMBER
    \rightarrow COMPLETE CURRENT INSTRUCTION
    \rightarrow LET TEMP = TF
```
11.6 External Hardware-Interrupt Sequence

Flow chart of the interrupt processing sequence of the 8088 and 8086 microprocessor

11.6 External Hardware-Interrupt Sequence

Interrupt-acknowledge bus cycle
11.6 **External Hardware-Interrupt Sequence**

**Interrupt service routine**

- To save registers and parameters on the stack:
  - PUSH XX
  - PUSH YY
  - PUSH ZZ

- Main body of the service routine:

- To restore register and parameters from the stack:
  - POP ZZ
  - POP YY
  - POP XX

- Return to main program:
  - IRET

---

**EXAMPLE**

The circuit in the next slide is used to count interrupt requests. The interrupting device interrupts the microprocessor each time the interrupt-request input signal transitions from 0 to 1. The corresponding interrupt type number generated by the 74LS244 is 60H.

- **a.** Describe the hardware operation for an interrupt request.
- **b.** What is the value of the type number sent to the microprocessor?
- **c.** Assume that (CS)=(DS)=1000H and (SS)=4000H; the main program is located at offsets of 200H; the count is held at 100H; the interrupt-service routine starts at offset 1000H from the beginning of another code segment at 2000H:0000H; and the stack starts at an offset of 500H from the stack segment. Make a map showing the memory address space.
- **d.** Write the main program and the service routine.
11.6 External Hardware-Interrupt Sequence

**EXAMPLE**

```
8088
INTR
AD1 - AD0
+5V

74LS74
Q    CLK
CLR   D
+0V

74LS244
2Y4 - 2Y1
1Y4 - 1Y1
2A4 - 2A1
2A2 - 1A4 - 1A2
1A1

Interrupting Device

Interrupt Request

Interrupt Type Numbers
= 01100000
AD7 - AD0 = 2Y4 2Y3 2Y2 2Y1
1Y4 1Y3 1Y2 1Y1
```

**Solution:**

a. A positive transition at the CLK input of the flip-flop (interrupt request) make the Q output of the flip-flop logic 1 and presents a positive level signal at the INTR input of the 8088. When 8088 recognized this as an interrupt request, it responds by generating the INTA signal. The logic 0 output on the line clears the flip-flop and enables the 74LS244 buffer to present the type number to the 8088. This number is read of the data bus by the 8088 and is used to initiate the interrupt-service routine.

b. From the inputs and outputs of the 74LS244, we see the type number is
   \[ AD_7...AD_2AD_0 = 2Y_42Y_32Y_22Y_11Y_41Y_31Y_21Y_1 = 01100000 \]
   \[ AD_7...AD_2AD_0 = 60H \]
Solution:
c. The memory organization is in the right figure

d. The flowcharts of the main program and interrupt-service routine
11.6 External Hardware-Interrupt Sequence

Solution:

;Main Program, START = 1000H:0200H

START:

`MOV AX,1000H ;Setup data segment at 1000H:0000H`
`MOV DS,AX`
`MOV AX,4000H ;Setup stack segment at 4000H:0000H`
`MOV SS,AX`
`MOV SP,0FF00H ;TOP is at 4000H:0FF00H`
`MOV AX,0000H ;Segment for interrupt vector table`
`MOV ES,AX`
`MOV AX,1000H ;Service routine offset`
`MOV AX,2000H ;Service routine segment`
`MOV [ES:18H],AX`

;Enable interrupts
`STI`
`HERE: JMP HERE ;Wait for interrupt`

;Interrupt Service Routine, SRVRTN = 2000H:1000H

`SRVRTN: PUSH AX ;Save register to be used`
`MOV AL,[0100H] ;Get the count`
`INC AL ;Increment the count`
`DAA ;Decimal adjust, just the count`
`MOV [0100H],AL ;Save the updated count`
`TOP AX ;Restore the register used`
`IRET ;Return from the interrupt`

---

11.7 82C59A Programmable Interrupt Controller

- The 82C59A is an LSI peripheral IC that is designed to simplify the implementation of the interrupt interface in the 8088- and 8086-based microcomputer system.
- The 82C59A is known as a **programmable interrupt controller** or **PIC**.
- The operation of the PIC is programmable under software control.
- The 82C59A can be cascaded to expand from 8 to 64 interrupt inputs.
11.7 82C59A Programmable Interrupt Controller

Block diagram of the 82C59A

Block diagram and pin layout of the 82C59A

Internal architecture of the 82C59A
11.7 82C59A Programmable Interrupt Controller

- Internal architecture of the 82C59A
  - Eight functional parts of the 82C59A
    - The data bus buffer
    - The read/write logic
    - The control logic
    - The in-service register
    - The interrupt-request register
    - The priority resolver
    - The interrupt-mask register
    - The cascade buffer/comparator

- Programming the 82C59A
  - Two types of command words are provided to program the 82C59A: the initialization command words (ICW) and the operational command words (OCW).
  - ICW commands (ICW1, ICW2, ICW3, ICW4) are used to load the internal control registers of the 82C59A to define the basic configuration or mode in which it is used.
  - The OCW commands (OCW1, OCW2, OCW3) permit the 8088 or 8086 microprocessor to initiate variations in the basic operating modes defined by the ICW commands.
  - The MPU issues commands to the 82C59A by initiating output (I/O-mapped) or write (Memory-mapped) cycles.
11.7 82C59A Programmable Interrupt Controller

- Programming the 82C59A

- Initialization command words
  - ICW₁

![Diagram of 82C59A Initialization Sequence]

![Diagram of ICW₁ Configuration]
11.7 82C59A Programmable Interrupt Controller

**EXAMPLE**

What value should be written into ICW₁ in order to configure the 82C59A so that ICW₄ is needed in the initialization sequence, the system is going to use multiple 82C59As, and its inputs are to be level sensitive? Assume that all unused bits are to be logic 0.

**Solution:**

Since ICW₄ is to be initialized, D₀ must be logic 1, \( D₀ = 1 \)

For cascaded mode of operation, D₁ must be 0, \( D₁ = 0 \)

And for level-sensitive inputs, D₃ must be 1, \( D₃ = 1 \)

Bits D₂ and D₅ through D₇ are don’t-care states and are 0.

\[ D₂ = D₅ = D₆ = D₇ = 0 \]

Moreover, D₄ must be fixed at the 1 logic level, \( D₄ = 1 \)

This gives the complete command word

\[ D₇D₆D₅D₄D₃D₂D₁D₀ = 00011001₂ = 19₁₆ \]

**Initialization command words**

- ICW₂ is used for type number determination

![Diagram of 82C59A Interrupt Controller](image-url)
11.7 82C59A Programmable Interrupt Controller

EXAMPLE

What should be programmed into register ICW₂ if the type numbers output on the bus by the device are to range from F₀₁₆ through F₇₁₆?

Solution:

To set the 82C59A up so that type numbers are in the range of F₀₁₆ through F₇₁₆, its device code bits must be

\[ D₂D₁D₀ = 1110₂ \]

The lower three bits are don’t-care states and all can be 0s. This gives the word

\[ D₇D₆D₅D₄D₃D₂D₁D₀ = 11110000₂ = F₀₁₆ \]

11.7 82C59A Programmable Interrupt Controller

- Initialization command words
  - ICW₃ is required only for cascaded mode of operation
11.7 82C59A Programmable Interrupt Controller

EXAMPLE

Assume that a master PIC is to be configured so that its IR$_0$ through IR$_3$ inputs are to accept inputs directly from external devices, but IR$_4$ through IR$_7$ are to be supplied by the INT outputs of slaves. What code should be used for the initialization command word ICW$_3$?

Solution:

For IR$_0$ through IR$_3$ to be configured to allow direct inputs from external devices, bits D$_0$ through D$_3$ of ICW$_3$ must be logic 0:

\[ D_3D_2D_1D_0 = 0000_2 \]

The other IR inputs of the master are to be supplied by INT outputs of slaves. Therefore, their control bits must be all 1:

\[ D_7D_6D_5D_4 = 1111_2 \]

This gives the complete command word

\[ D_7D_6D_5D_4D_3D_2D_1D_0 = 11110000_2 = F0_{16} \]

Initialization command words

ICW$_4$ is used to configure device for use with the 8088 or 8086 and selects various features in its operation.
11.7 82C59A Programmable Interrupt Controller

- Operational command words
  - OCW is used to access the contents of the interrupt-mask register (IMR). Setting a bit to logic 1 masks out the associated interrupt input.

```
<table>
<thead>
<tr>
<th></th>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
<td>1</td>
<td>M7</td>
<td>M6</td>
<td>M5</td>
<td>M4</td>
<td>M3</td>
<td>M2</td>
<td>M1</td>
</tr>
</tbody>
</table>
```

**Interrupt mask:**
- 1 = Mask set
- 0 = Mask reset

---

**EXAMPLE**

What should be the OCW code if interrupt inputs IR0 through IR3 are to be masked and IR4 through IR7 are to be unmasked?

**Solution:**

For IR0 through IR3 to be masked, their corresponding bits in the mask register must be make logic 1:

$$D_3D_2D_1D_0 = 1111_2$$

On the other hand, for IR4 through IR7 to be unmasked, D4 through D7 must be logic 0:

$$D_7D_6D_5D_4 = 0000_2$$

This gives the complete command word

$$D_7D_6D_5D_4D_3D_2D_1D_0 = 00011111_2 = 0F_{16}$$
11.7 82C59A Programmable Interrupt Controller

- Operational command words
  - OCW₂ is used to select appropriate priority scheme and assigns an IR level for the scheme.

**EXAMPLE**

What OCW₂ must be issued to the 82C59A if the priority scheme rotate on nonspecific EOI command is to be selected?

**Solution:**

To enable the rotate on nonspecific EOI command priority scheme, bits D₇ through D₅ must be set to 101. Since a specific level does not have to be considered, the rest of the bits in the command word can be 0. This gives OCW₂ as

\[ D_7D_6D_5D_4D_3D_2D_1D_0 = 10100000_2 = A0_{16} \]
11.7 82C59A Programmable Interrupt Controller

- Operational command words
  - OCW₃ permits reading of the contents of the ISR or IRR registers through software.

EXAMPLE

Write a program that will initialize an 82C59A with the initialization command words ICW₁, ICW₂, ICW₃ derived in the previous examples, and ICW₄ is equal to 1F₁₆. Assume that the 82C59A resides at address A000₁₆ in the memory address space.

Solution:

Since the 82C59A resides in the memory address space, we can use a series of move instructions to write the initialization command words into its registers. Note that the memory address for an ICW is A000₁₆ if A₀ = 0, and it is A00₁₆ if A₀ = 1. However, before doing this, we must first disable interrupts. This is done with the instruction

```
CLI ; Disable interrupts
```
### 11.7 82C59A Programmable Interrupt Controller

Next we will create a data segment starting at address 00000H:

```
MOV AX, 0 ; Create a data segment at 00000H
MOV DS, AX
```

Now we are ready to write the command words to the 82C59A:

```
MOV AL, 19H ; Load ICW1
MOV [0A000H], AL ; Write ICW1 to 82C59A
MOV AL, 0F0H ; Load ICW2
MOV [0A001H], AL ; Write ICW2 to 82C59A
MOV AL, 0F0H ; Load ICW3
MOV [0A001H], AL ; Write ICW3 to 82C59A
MOV AL, 1FH ; Load ICW4
MOV [0A001H], AL ; Write ICW4 to 82C59A
```

Initialization is now complete and the interrupts can be enabled:

```
STI ; Enable interrupts
```
11.8 Interrupt Interface Circuits Using the 82C59A

For applications that require more than eight interrupt-request inputs, several 82C59As are connected into a master/slave configuration.

Master/slave connection of the 82C59A interface
11.8 Interrupt Interface Circuits Using the 82C59A

Maximum-mode interrupt interface for the 8088 microcomputer using the 82C59A

EXAMPLE

Analyze the circuit in the following figure and write an appropriate main program and a service routine that counts as a decimal number the positive edges of the clock signal applied to the IR0 input of the 82C59A.
11.8 Interrupt Interface Circuits Using the 82C59A

Solution:
Lets first determine the I/O addresses of the 82C59A registers:

\[ A_{15}A_{14}A_{13}A_{12}A_{11}A_{10}A_{9}A_{8}A_{7}A_{6}A_{5}A_{4}A_{3}A_{2}A_{1}A_{0} = 1111111100000000_2 \text{ for } A_1 = 0, \ M/IO = 0 \text{ and} \]
\[ = 1111111100000010_2 \text{ for } A_1 = 1, \ M/IO = 0 \]

These two I/O addresses are FF00H and FF02H, respectively. The address FF00H is for the ICW1 and FF02H is for the ICW2, ICW3, ICW4, and OCW1 command words.
The command words are:

- **ICW1** = 00010011₂ = 13H
- **ICW2** = 01001000₂ = 48H
- **ICW3** = not needed
- **ICW4** = 00000011₂ = 03H
- **OCW1** = 11111110₂ = FEH

Software organization:
11.8 Interrupt Interface Circuits Using the 82C59A

Flowcharts of the main program and service routine:

Main Program
- Set up data segment, stack segment, and stack pointer
- Set up the interrupt vector
- Initialize 82C59A
- Enable interrupts
- Wait for interrupt

SRV72
- Save processor status
- Increment the count
- Restore processor status
- Return

Program:
```
;MAIN PROGRAM
CLI ;Start with interrupt disabled

START:
    MOV  AX, 0 ;Extra segment at 00000H
    MOV  ES, AX
    MOV  AX, 1000H ;Data segment at 01000H
    MOV  DS, AX
    MOV  AX, 0FF00H ;Stack segment at 0FF00H
    MOV  SS, AX
    MOV  SP, 100H ;Top of stack at 10000H

    MOV  AX, OFFSET SRV72 ;Get offset for SRV72
    MOV  [ES:120H], AX ;Set up the IP
    MOV  AX, SEG SRV72 ;Get CS for the service routine
    MOV  [ES:122H], AX ;Set up the CS
```
11.8 **Interrupt Interface Circuits Using the 82C59A**

**Program:**

MOV DX, 0FF00H ; ICW1 address  
MOV AL, 13H ; Edge trig input, single 8259A  
OUT DX, AL  
MOV DX, 0FF02H ; ICW2, ICW4, OCW1 address  
MOV AL, 48H ; ICW2, type 72  
OUT DX, AL  
MOV AL, 03H ; ICW4, AEOI, nonbuf mode  
OUT DX, AL  
MOV AL, 0FEH ; OCW1, mask all but IR0  
OUT DX, AL  
STI ; Enable the interrupts

---

11.8 **Interrupt Interface Circuits Using the 82C59A**

**Program:**

SRV72: PUSH AX ; Save register to be used  
MOV AL, [COUNT] ; Get the count  
INC AL ; Increment the count  
DAA ; Decimal adjust the count  
MOV [COUNT], AL ; Save the new count  
POP AX ; Restore the register used  
IRET ; Return from interrupt
11.9 Software Interrupts

- The 8088 and 8086 microcomputer systems are capable of implementing up to 256 software interrupts.
- The INT n instruction is used to initiate a software interrupt. The software interrupt service routine vectors are also located in the memory locations in the vector table.
- Software interrupts are of higher priority than the external interrupts and are not masked out by IF.
- The software interrupts are actually vectored subroutine calls.

11.10 Nonmaskable Interrupt

- The nonmaskable interrupt (NMI) is initiated from external hardware.
- Differences between NMI and other external interrupts:
  - NMI can not be masked out with the interrupt flag.
  - Request for NMI service are signaled to the 8088/8086 microprocessor by applying logic 1 at the NMI input, not the INTR input.
  - NMI input is positive edge-triggered. Therefore, a request for NMI is automatically latched internal to the MPU.
- NMI automatically vectors from the type 2 vector location in the pointer table (000816 ~000A16)
- Typically, the NMI is assigned to hardware events that must be responded to immediately, such power failure.
11.11 Reset

The RESET input of the 8088 and 8086 microprocessors provides a hardware means for initializing the microcomputer.

Bus and control signal status of the 8088/8086 during system reset
11.11 Reset

- When the MPU recognizes the RESET input, it initiates its internal initialization routine. At completion of initialization, the flags are all cleared, the registers are set to the values in the following table.

<table>
<thead>
<tr>
<th>CPU COMPONENT</th>
<th>CONTENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flags</td>
<td>Clear</td>
</tr>
<tr>
<td>Instruction pointer</td>
<td>0000H</td>
</tr>
<tr>
<td>CS Register</td>
<td>FFFFH</td>
</tr>
<tr>
<td>DS Register</td>
<td>0000H</td>
</tr>
<tr>
<td>SS Register</td>
<td>0000H</td>
</tr>
<tr>
<td>ES Register</td>
<td>0000H</td>
</tr>
<tr>
<td>Queue</td>
<td>Empty</td>
</tr>
</tbody>
</table>

- The external hardware interrupts are disabled after the initialization.
- Program execution begins at address $FFFF0_{16}$ after reset. This storage location contains an instruction that will cause a jump to the startup (boot-strap) program that is used to initialize the reset of the microcomputer system’s resources, such as I/O ports, the interrupt flag, and data memory.
- After the system-level initialization is complete, another jump can be performed to the starting point of the microcomputer’s operating system or application program.
11.12 Internal Interrupt Functions

- Four of the 256 interrupts of the 8088 and 8086 are dedicated to internal interrupt functions.
- Internal interrupts differ from external hardware interrupts in that they occur due to the result of executing an instruction, not an event that takes place in external hardware.
- Internal interrupts are not masked out with IF flag.
- Internal interrupts of the 8088 and 8086 MPU:
  - Divide error (Type number 0)
  - Single step (Type number 1)
  - Breakpoint interrupt (Type number 3)
  - Overflow error (Type number 4)

---

### Internal interrupt vector locations

<table>
<thead>
<tr>
<th>Memory address</th>
<th>Vector number</th>
<th>Internal function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001H</td>
<td>4</td>
<td>Overflow error</td>
</tr>
<tr>
<td>0010H</td>
<td>3</td>
<td>Breakpoint</td>
</tr>
<tr>
<td>0005H</td>
<td>1</td>
<td>Single step</td>
</tr>
<tr>
<td>0004H</td>
<td>0</td>
<td>Divide error</td>
</tr>
</tbody>
</table>