INTERRUPT INTERFACE OF THE 8088 AND 8086 MICROPROCESSOR

INTERRUPT INTERFACE OF THE 8088 AND 8086 MICROPROCESSOR

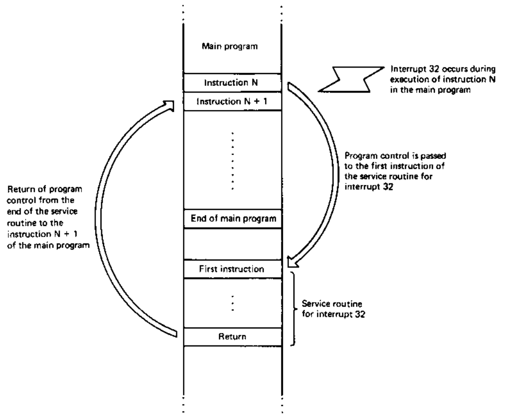
INTERRUP INTERFACE OF THE 8088 AND 8086 MICROPROCESSOR

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



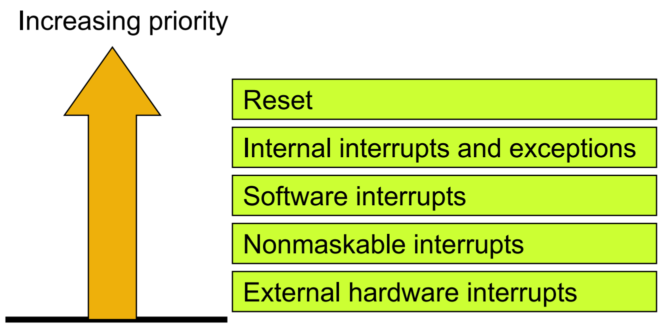
Main program
Instruction N
Instruction N + 1
End of main program
First instruction
Return
Interrupt 32 occurs during execution of instruction N in the main program
Program control is passed to the first instruction of the service routine for interrupt 32
Return of program control from the end of the service routine to the instruction N + 1 of the main program
Service routine for interrupt 32

11.1 Interrupt Mechanism, Types and Priority

- Hardware, software, and internal interrupts are serviced on a ***priority*** basis.
- Each interrupts 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

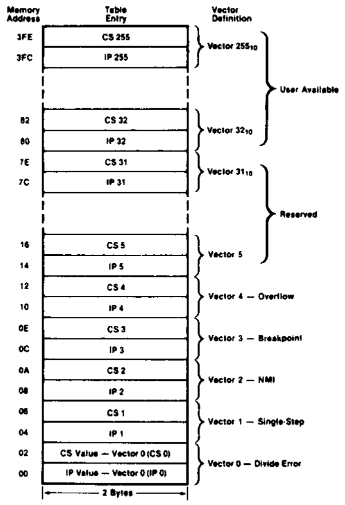


11.2 Interrupt Vector Table

- 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 00000_{16} and ends at $003FE_{16}$. This represents the first 1 Kbytes of memory.

11.2 Interrupt Vector Table

- Interrupt vector table of the 8088/8086



11.2 Interrupt Vector Table

EXAMPLE

At what address are CS_{50} and IP_{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_{50} and IP_{50} represent the words of the type 50 interrupt pointer, we get

$$\text{Address} = 4 \times 50 = 200$$

converting to binary form gives

$$\text{Address} = 11001000_2 = C8_{16}$$

Therefore, IP_{50} is stored at $000C8_{16}$ and CS_{50} at $000CA_{16}$.

11.3 Interrupt Instructions

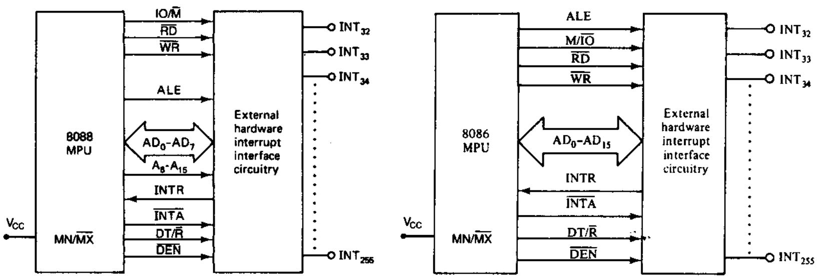
Mnemonic	Meaning	Format	Operation	Flags affected
CLI	Clear interrupt flag	CLI	$0 \to (IF)$	IF
STI	Set interrupt flag	STI	$1 \to (IF)$	IF
INT n	Type n software interrupt	INT n	$(Flags) \to ((SP)-2)$ $0 \to TF, IF$ $(CS) \to ((SP)-4)$ $(2+4xn) \to (CS)$ $(IP) \to ((SP)-6)$ $(4xn) \to (IP)$	TF, IF
IRET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

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

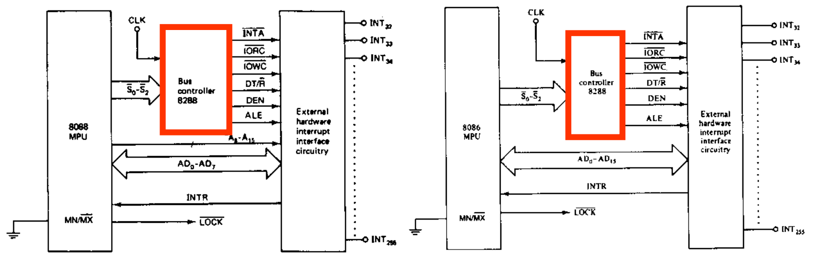
- Minimum-mode interrupt interface
  - Key interrupt interface signals: INTR and $\overline{\text{INTA}}$



Minimum-mode 8088 and 8086 system external hardware interrupt interface

11.5 External Hardware-Interrupt Interface Signals

- Maximum-mode interrupt interface
  - 8288 bus controller is added in the interface. The $\overline{\text{INTA}}$ and ALE signals are produced by the 8288.
  - The bus priority lock signal $\overline{\text{LOCK}}$ 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

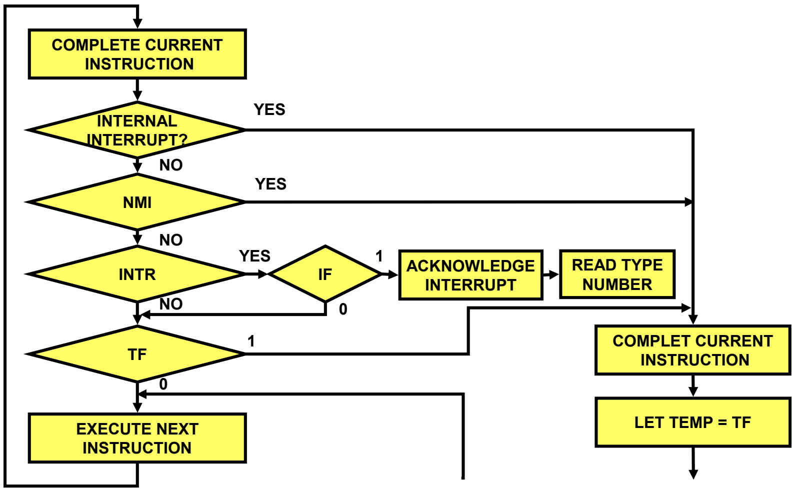
11.5 External Hardware-Interrupt Interface Signals

■Maximum-mode interrupt interface

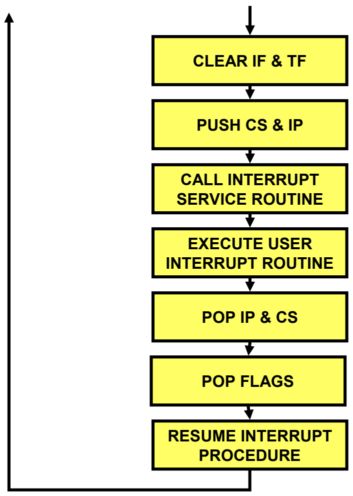
Status inputs			CPU cycle	8288 command
$\overline{S}_2$	$\overline{S}_1$	$\overline{S}_0$		
0	0	0	Interrupt acknowledge	$\overline{\text{INTA}}$
0	0	1	Read I/O port	$\overline{\text{IORC}}$
0	1	0	Write I/O port	$\overline{\text{IOWC}}$, $\overline{\text{AIOWC}}$
0	1	1	Halt	None
1	0	0	Instruction fetch	$\overline{\text{MRDC}}$
1	0	1	Read memory	$\overline{\text{MRDC}}$
1	1	0	Write memory	$\overline{\text{MWTC}}$, $\overline{\text{AMWC}}$
1	1	1	Passive	None

Interrupt bus status code to the 8288 bus controller

11.6 External Hardware-Interrupt Sequence



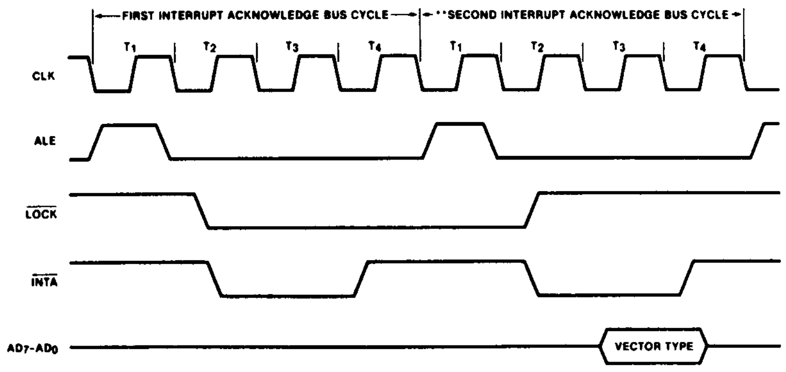
11.6 External Hardware-Interrupt Sequence


CLEAR IF & TF
PUSH CS & IP
CALL INTERRUPT SERVICE ROUTINE
EXECUTE USER INTERRUPT ROUTINE
POP IP & CS
POP FLAGS
RESUME INTERRUPT PROCEDURE

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

11.6 External Hardware-Interrupt Sequence

- Interrupt-acknowledge bus cycle


FIRST INTERRUPT ACKNOWLEDGE BUS CYCLE
**SECOND INTERRUPT ACKNOWLEDGE BUS CYCLE
T1
T2
T3
T4
CLK
ALE
LOCK
INTA
AD7-AD0
VECTOR TYPE

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

11.6 External Hardware-Interrupt Sequence

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.

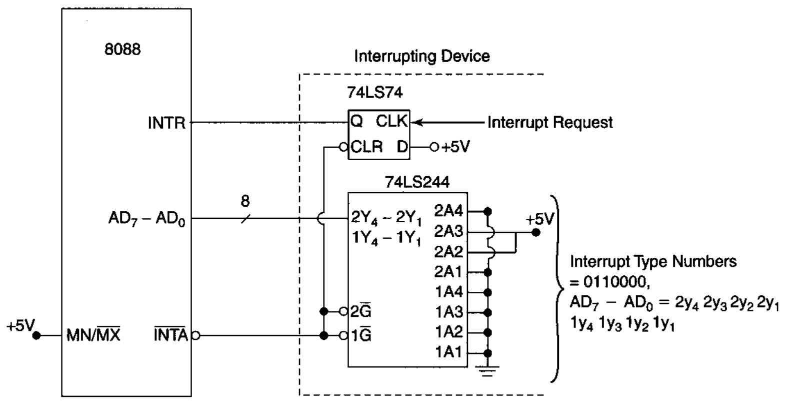
b. What is the value of the type number sent to the microprocessor?

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



國立台灣大學
生物機電系
林達德

11.6 External Hardware-Interrupt Sequence

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 $\overline{\text{INTA}}$ 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

$AD_7 \ldots AD_1AD_0 = 2Y_42Y_32Y_22Y_11Y_41Y_31Y_21Y_1 = 01100000_2$

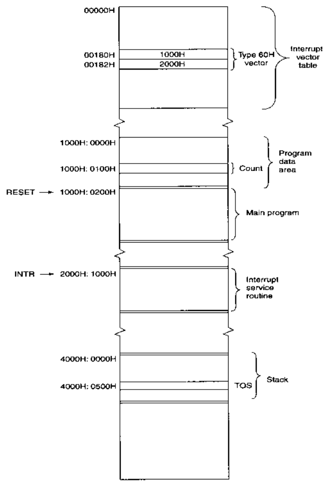
$AD_7 \ldots AD_1AD_0 = 60H$

國立台灣大學
生物機電系
林達德

11.6 External Hardware-Interrupt Sequence

Solution:

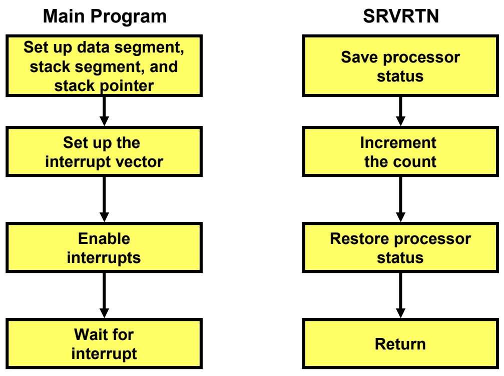
c. The memory organization is in the right figure



Sequence

Solution:

d. The flowcharts of the main program and interrupt-service routine



11.6 External Hardware-Interrupt Sequence

Solution:

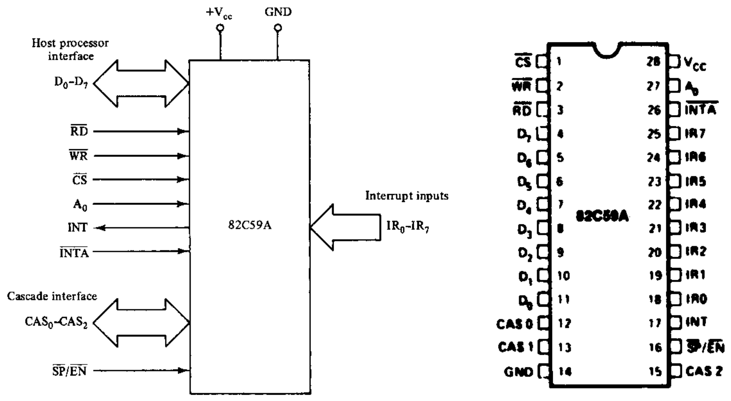
```
;Main Program, START = 1000H:0200H

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,0500H          ;TOS is at 4000H;0500H
           MOV AX,0000H          ;Segment for interrupt vector table
           MOV ES,AX
           MOV AX,1000H          ;Service routine offset
           MOV [ES:180H],AX
           MOV AX,2000H          ;Service routine segment
           MOV [ES:182H],AX
           STI                   ;Enable interrupts
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 asdjust the count
           MOV [0100H],AL        ;Save the updated count
           POP 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

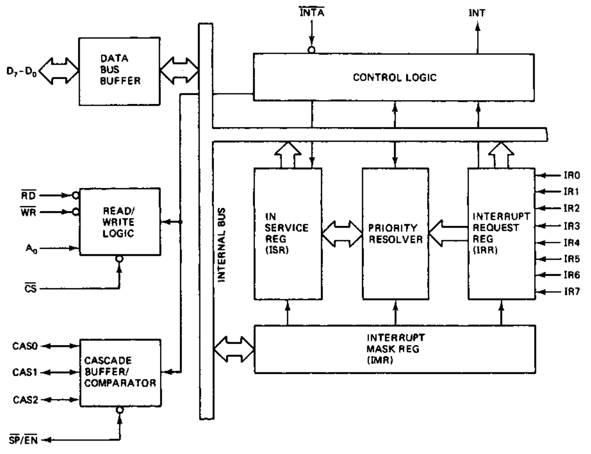
- Block diagram of the 82C59A



Block diagram and pin layout of the 82C59A

11.7 82C59A Programmable Interrupt Controller

- Internal architecture of the 82C59A



11.7 82C59A Programmable Interrupt Controller

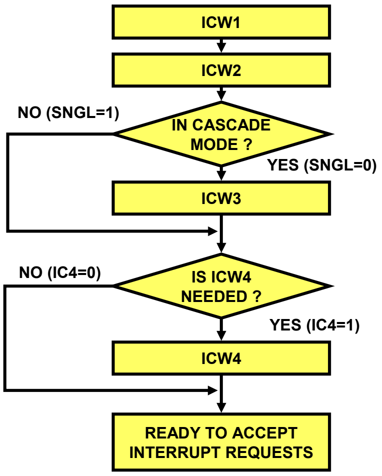
- Internal architecture of the 82C59A
  - Eight functional parts of the 82C59A
    - The data bus buffer
    - The read/write logic
    - The control logic
    - The in-service register
    - The interrupt-request register
    - The priority resolver
    - The interrupt-mask register
    - The cascade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

11.7 82C59A Programmable Interrupt Controller

- 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 (ICW_1, ICW_2, ICW_3, ICW_4) 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 (OCW_1, OCW_2, OCW_3) 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

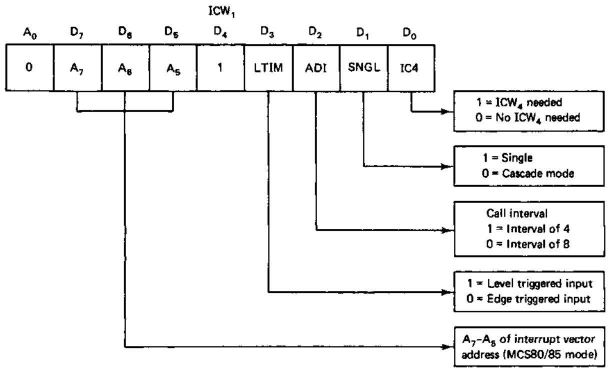
- Programming the 82C59A



Initialization sequence of the 82C59A

Controller

- Initialization command words
  - ICW$_1$



11.7 82C59A Programmable Interrupt Controller

EXAMPLE

What value should be written into ICW_1 in order to configure the 82C59A so that ICW_4 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 ICW_4 is to be initialized, D_0 must be logic 1, $D_0 = 1$

For cascaded mode of operation, D_1 must be 0, $D_1 = 0$

And for level-sensitive inputs, D_3 must be 1, $D_3 = 1$

Bits D_2 and D_5 through D_7 are don't-care states and are 0.

$$D_2 = D_5 = D_6 = D_7 = 0$$

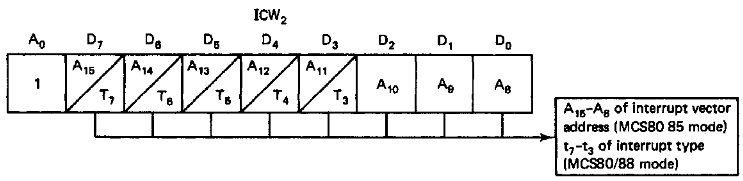
Moreover, D_4 must be fixed at the 1 logic level, $D_4 = 1$

This gives the complete command word

$$D_7D_6D_5D_4D_3D_2D_1D_0 = 00011001_2 = 19_{16}$$

11.7 82C59A Programmable Interrupt Controller

- Initialization command words
  - ICW_2 is used for type number determination


ICW2
A0 D7 D6 D5 D4 D3 D2 D1 D0
1
A15 T7
A14 T6
A13 T5
A12 T4
A11 T3
A10
A9
A8
A15–A8 of interrupt vector address (MCS80 85 mode) t7–t3 of interrupt type (MCS80/88 mode)

11.7 82C59A Programmable Interrupt Controller

EXAMPLE

What should be programmed into register ICW_2 if the type numbers output on the bus by the device are to range from $F0_{16}$ through $F7_{16}$?

Solution:

To set the 82C59A up so that type numbers are in the range of $F0_{16}$ through $F7_{16}$, its device code bits must be

$$D_7D_6D_5D_4D_3 = 11110_2$$

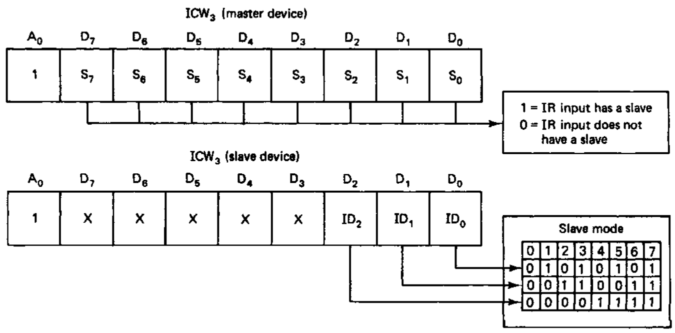
The lower three bits are don't-care states and all can be 0s. This gives the word

$$D_7D_6D_5D_4D_3D_2D_1D_0 = 11110000_2 = F0_{16}$$

11.7 82C59A Programmable Interrupt Controller

Initialization command words

- ICW_3 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:

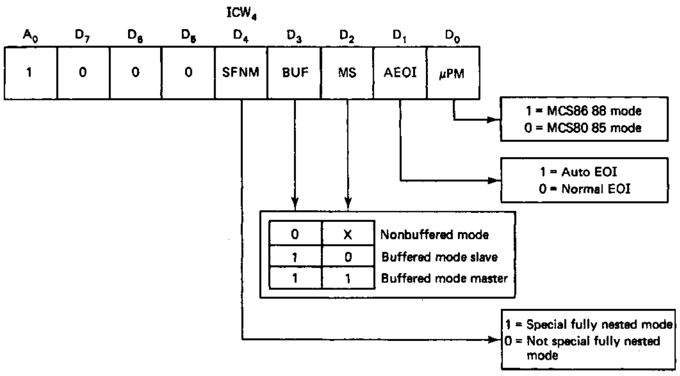
$$D_7D_6D_5D_4 = 1111_2$$

This gives the complete command word

$$D_7D_6D_5D_4D_3D_2D_1D_0 = 11110000_2 = F0_{16}$$

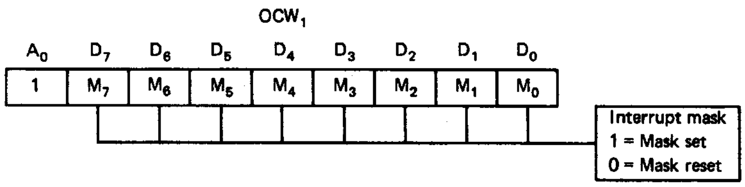
11.7 82C59A Programmable Interrupt Controller

- 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_1 is used to access the contents of the interrupt-mask register (IMR). Setting a bit to logic 1 masks out the associated interrupt input.



OCW_1
A_0 D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
1 M_7 M_6 M_5 M_4 M_3 M_2 M_1 M_0
Interrupt mask
1 = Mask set
0 = Mask reset

11.7 82C59A Programmable Interrupt Controller

EXAMPLE

What should be the OCW_1 code if interrupt inputs IR_0 through IR_3 are to be masked and IR_4 through IR_7 are to be unmasked?

Solution:

For IR_0 through IR_3 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 IR_4 through IR_7 to be unmasked, D_4 through D_7 must be logic 0:

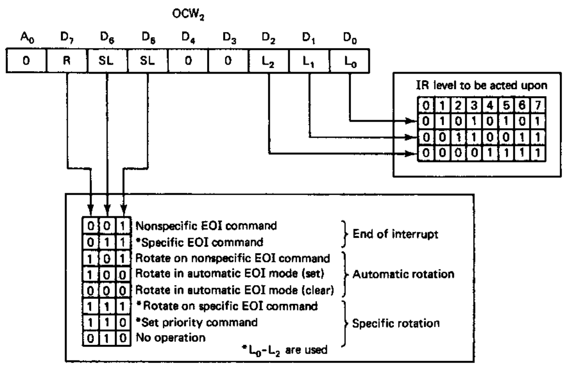
$$D_7D_6D_5D_4 = 0000_2$$

This gives the complete command word

$$D_7D_6D_5D_4D_3D_2D_1D_0 = 00001111_2 = 0F_{16}$$

11.7 82C59A Programmable Interrupt Controller

- Operational command words
  - OCW$_2$ is used to select appropriate priority scheme and assigns an IR level for the scheme.


OCW$_2$
A$_0$ D$_7$ D$_6$ D$_5$ D$_4$ D$_3$ D$_2$ D$_1$ D$_0$
0 R SL SL 0 0 L$_2$ L$_1$ L$_0$
IR level to be acted upon
0 1 2 3 4 5 6 7
0 1 0 1 0 1 0 1
0 0 1 1 0 0 1 1
0 0 0 0 1 1 1 1
0 0 1 Nonspecific EOI command
0 1 1 *Specific EOI command
End of interrupt
1 0 1 Rotate on nonspecific EOI command
1 0 0 Rotate in automatic EOI mode (set)
0 0 0 Rotate in automatic EOI mode (clear)
Automatic rotation
1 1 1 *Rotate on specific EOI command
1 1 0 *Set priority command
0 1 0 No operation
Specific rotation
*L$_0$-L$_2$ are used

11.7 82C59A Programmable Interrupt Controller

EXAMPLE

What OCW$_2$ must be issued to the 82C59A if the priority scheme rotate on nonspecific EOI command is to be selected?

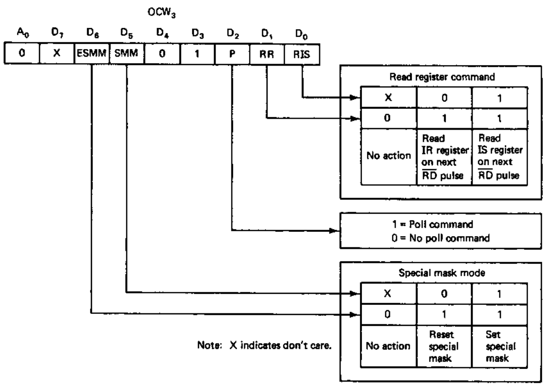
Solution:

To enable the rotate on nonspecific EOI command priority scheme, bits D$_7$ through D$_5$ 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 OCW$_2$ as

$$D_7D_6D_5D_4D_3D_2D_1D_0 = 10100000_2 = A0_{16}$$

11.7 82C59A Programmable Interrupt Controller

- Operational command words
  - OCW_3 permits reading of the contents of the ISR or IRR registers through software.



OCW_3
A_0 D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0
0 X ESMM SMM 0 1 P RR RIS
Read register command
X 0 1
0 1 1
No action
Read IR register on next $\overline{RD}$ pulse
Read IS register on next $\overline{RD}$ pulse
1 = Poll command
0 = No poll command
Special mask mode
X 0 1
0 1 1
No action
Reset special mask
Set special mask
Note: X indicates don't care.

11.7 82C59A Programmable Interrupt Controller

EXAMPLE

Write a program that will initialize an 82C59A with the initialization command words ICW_1, ICW_2, ICW_3 derived in the previous examples, and ICW_4 is equal to $1F_{16}$. Assume that the 82C59A resides at address $A000_{16}$ 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 $A000_{16}$ if $A_0 = 0$, and it is $A001_{16}$ if $A_0 = 1$. 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 00000_{16}:

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

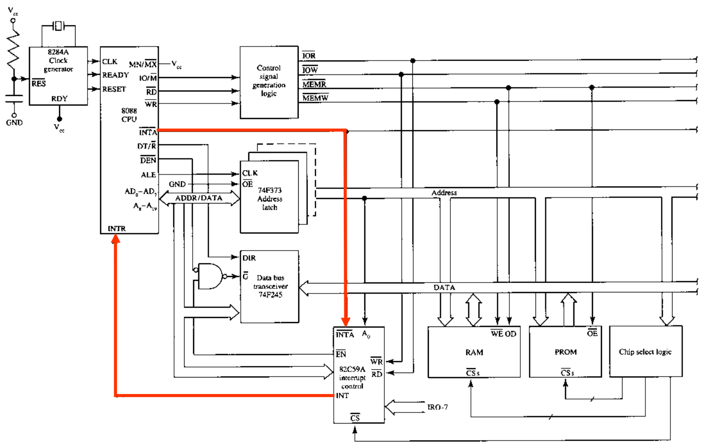
Now we are ready to write the command words to the 82C59A:

```
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

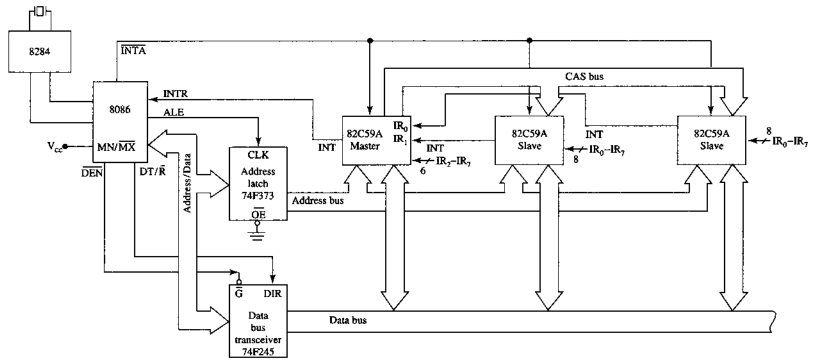
```
STI                    ;Enable interrupts
```

11.8 Interrupt Interface Circuits Using the 82C59A



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

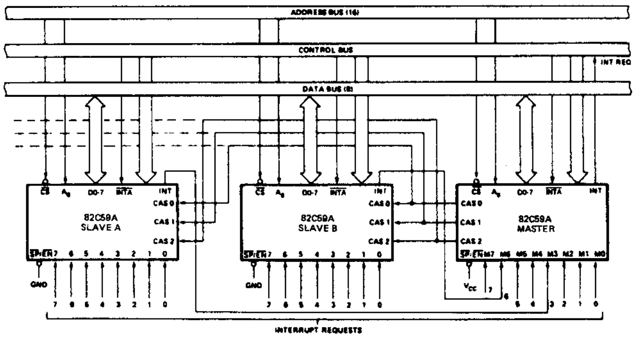
11.8 Interrupt Interface Circuits Using the 82C59A



Minimum-mode interrupt interface for the 8086 microcomputer using the 82C59A

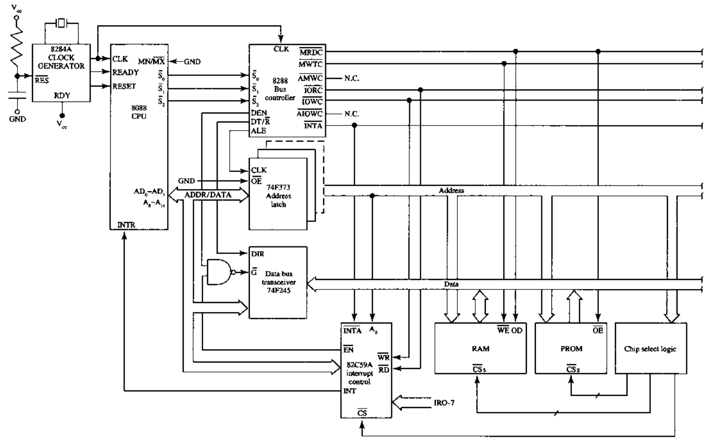
11.8 Interrupt Interface Circuits Using the 82C59A

- 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

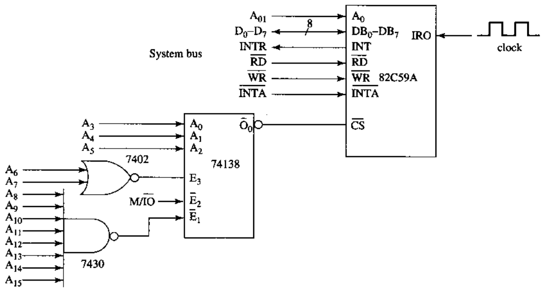


Maximum-mode interrupt interface for the 8088 microcomputer using the 82C59A

11.8 Interrupt Interface Circuits 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 IR_0 input of the 82C59A.



11.8 Interrupt Interface Circuits Using the 82C59A

Solution:

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

$A_{15}A_{14}A_{13}A_{12}A_{11}A_{10}A_9A_8A_7A_6A_5A_4A_3A_2A_1A_0$
$= 1111111100000000_2$ for $A_1 = 0$, M/IO = 0 and
$= 1111111100000010_2$ for $A_1 = 1$, M/IO = 0

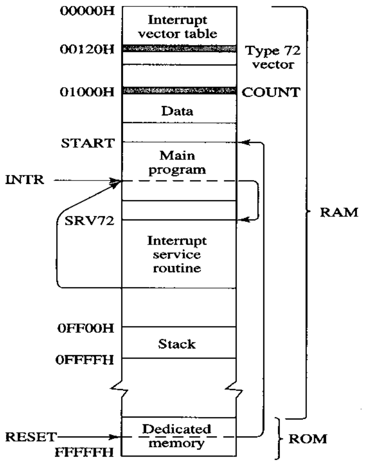
These two I/O addresses are FF00H and FF02H, respectively. The address FF00H is for the ICW_1 and FF02H is for the ICW_2, ICW_3, ICW_4, and OCW_1 command words.

The command words are:

$ICW_1 = 00010011_2 = 13H$
$ICW_2 = 01001000_2 = 48H$
ICW_3 = not needed
$ICW_4 = 00000011_2 = 03H$
$OCW_1 = 11111110_2 = FEH$

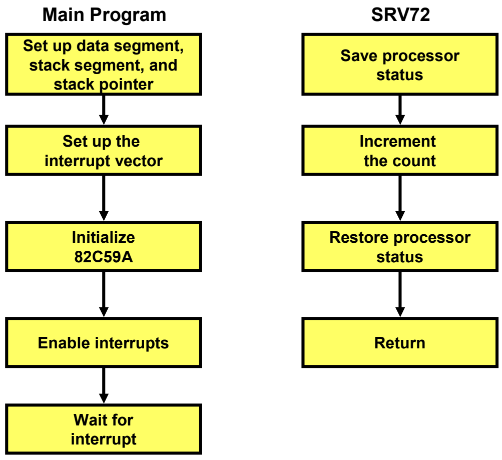
11.8 Interrupt Interface Circuits Using the 82C59A

Software organization:



11.8 Interrupt Interface Circuits Using the 82C59A

Flowcharts of the main program and service routine:



the 82C59A

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

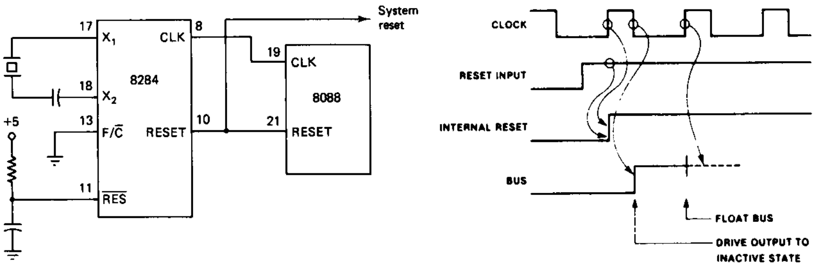
- The 8088 and 8086 microcomputer systems are capable of implementing up to 256 software interrupts.
- The **INT n** instruction is used to initiate a software interrupt. The software interrupt service routine vectors are also located in the memory locations in the vector table.
- Software interrupts 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 (0008_{16} ~$000A_{16}$)
- Typically, the NMI is assigned to hardware events that must be responded to immediately, such power failure.

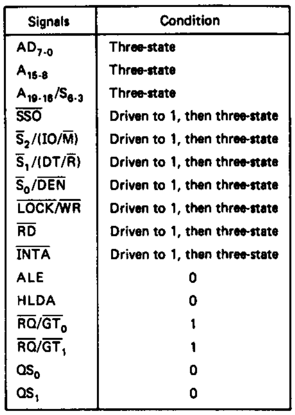
11.11 Reset

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



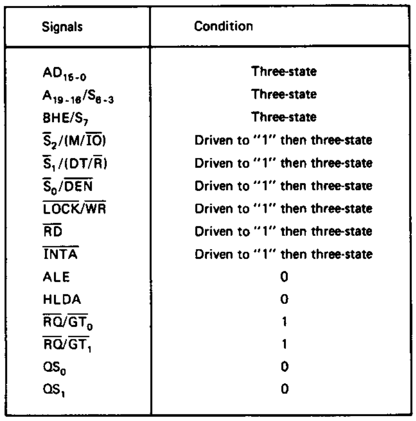
Reset interface and timing sequence of the 8088

11.11 Reset

- 

Signals	Condition
AD_{7-0}	Three-state
A_{15-8}	Three-state
A_{19-16}/S_{6-3}	Three-state
$\overline{SSO}$	Driven to 1, then three-state
$\overline{S_2}/(IO/\overline{M})$	Driven to 1, then three-state
$\overline{S_1}/(DT/\overline{R})$	Driven to 1, then three-state
$\overline{S_0}/\overline{DEN}$	Driven to 1, then three-state
$\overline{LOCK}/\overline{WR}$	Driven to 1, then three-state
$\overline{RD}$	Driven to 1, then three-state
$\overline{INTA}$	Driven to 1, then three-state
ALE	0
HLDA	0
$\overline{RQ}/\overline{GT_0}$	1
$\overline{RQ}/\overline{GT_1}$	1
QS_0	0
QS_1	0

8088 signal status



Signals	Condition
AD_{15-0}	Three-state
A_{19-16}/S_{6-3}	Three-state
$\overline{BHE}/S_7$	Three-state
$\overline{S_2}/(M/\overline{IO})$	Driven to "1" then three-state
$\overline{S_1}/(DT/\overline{R})$	Driven to "1" then three-state
$\overline{S_0}/\overline{DEN}$	Driven to "1" then three-state
$\overline{LOCK}/\overline{WR}$	Driven to "1" then three-state
$\overline{RD}$	Driven to "1" then three-state
$\overline{INTA}$	Driven to "1" then three-state
ALE	0
HLDA	0
$\overline{RQ}/\overline{GT_0}$	1
$\overline{RQ}/\overline{GT_1}$	1
QS_0	0
QS_1	0

8086 signal status

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.

CPU COMPONENT	CONTENT
Flags	Clear
Instruction pointer	0000H
CS Register	FFFFH
DS Register	0000H
SS Register	0000H
ES Register	0000H
Queue	Empty

11.11 Reset

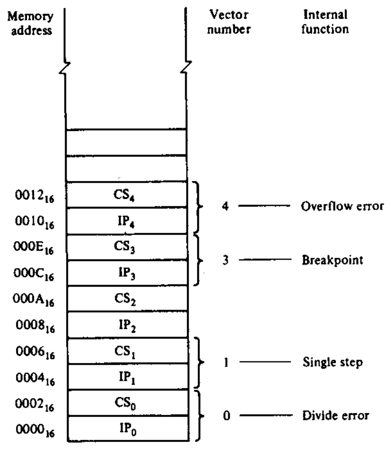
- The external hardware interrupts are disabled after the initialization.
- Program execution begins at address $FFFF0_{16}$ after reset. This storage location contains an instruction that will cause a jump to the startup (boot-strap) program that is used to initialize the reset of the microcomputer 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)

11.12 Internal Interrupt Functions

- Internal interrupt vector locations



Memory address
Vector number
Internal function
0012_{16}
CS_4
0010_{16}
IP_4
$000E_{16}$
CS_3
$000C_{16}$
IP_3
$000A_{16}$
CS_2
0008_{16}
IP_2
0006_{16}
CS_1
0004_{16}
IP_1
0002_{16}
CS_0
0000_{16}
IP_0
4 Overflow error
3 Breakpoint
1 Single step
0 Divide error