Module 3

Interrupts - Types of Interrupts and Interrupt Service Routine. Handling Interrupts in 8086, Interrupt programming. Basic Peripherals and their Interfacing with 8086 - Programmable Interrupt Controller - 8259 - Architecture.

Interrupts

- ‘Interrupt' means break the sequence of operation.
- While the CPU is executing a program, an 'interrupt' breaks the normal sequence of execution of instructions, diverts its execution to some other program called *Interrupt Service Routine* (ISR).
- After executing ISR, the control is transferred back again to the main program which was being executed at the time of interruption.
- Whenever a number of devices interrupt a CPU at a time, and if the processor is able to handle them properly, it is said to have multiple interrupt processing capability.

Real World Analogy

- *Suppose you are reading a novel and have completed up to page 100. At this instant, your younger brother distracts you. You will somehow mark the line and the page you are reading, so that you may be able to continue after you attend to him. Say you have marked page number 101. You will now go to his room to solve his problem. While you are helping him a friend of yours comes and asks you for a textbook. Now, there are two options in front of you. The first is to make the friend wait till you complete serving your brother, and thereafter you serve his request. In this, you are giving less priority to your friend.*
- *The second option is to ask your brother to wait; remember the solution of his problem at the intermediate state; serve the friend; and after the friend is served, continue with the solution that was in the intermediate state. In this case, it may be said that you have given higher priority to your friend. After serving both of them, again you may continue reading from page 101 of the novel. Here, first you are interrupted by your brother. While you are serving your brother, you are again interrupted by a friend. This type of sequence of appearance of interrupts is called nested interrupt, i.e. interrupt within interrupt.*

- In case of 8086, there are two interrupt pins, viz. NMI and INTR.
- The NMI is a nonmaskable interrupt input pin which means that any interrupt request at NMI input cannot be masked or disabled by any means.
- The INTR interrupt, however, may be masked using the Interrupt Flag (IF). The INTR, further, is of 256 types.
- The INTR types may be from 00 to FFH (or 00 to 255). If more than one type of INTR interrupt occurs at a time, then an external chip called programmable interrupt controller is required to handle them.

- Interrupt Service Routines (ISRs) are the programs to be executed by interrupting the main program execution of the CPU, after an interrupt request appears.

- After the execution of ISR, the main program continues its execution further from the point at which it was interrupted.

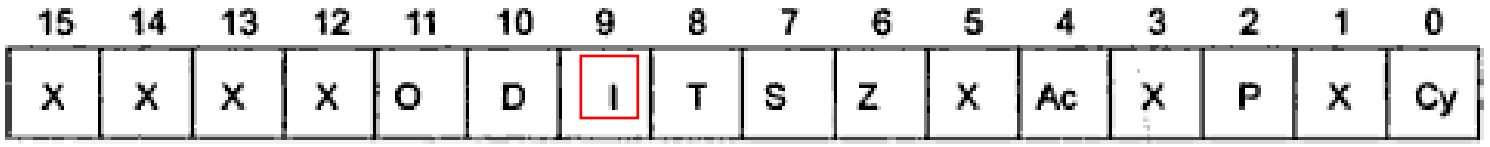
INTERRUPT CYCLE OF 8086

Types of Interrupts

- Broadly, there are two types of interrupts.
- The first out of them is external interrupt and the second is internal interrupt.
- In external interrupt, an external device or a signal interrupts the processor from outside or, in other words, the interrupt is generated outside the processor, for example, a keyboard interrupt.
- The internal interrupt, on the other hand, is generated internally by the processor circuit, or by the execution of an interrupt instruction. The examples of this type are divide by zero interrupt, overflow interrupt, interrupts due to INT instructions, etc.

Interrupt Flag

- *In the 8086 Flag register,* Interrupt Flag: If this flag is set, the maskable interrupt are recognized by the CPU, otherwise they are ignored.

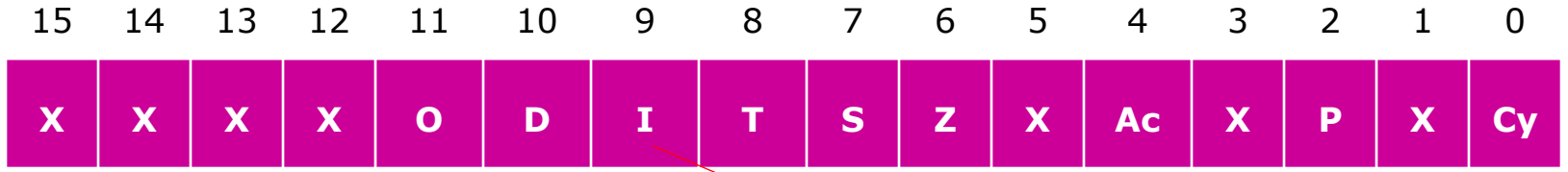


O — Overflow flag
D — Direction flag
I — Interrupt flag
T — Trap flag
S — Sign flag
Z — Zero flag
Ac — Auxiliary carry flag
P — Parity flag
Cy — Carry flag
X — Not used

Fig. 1.4 *Flag Register of 8086*

Architecture — Execution Unit (EU)

Flag Register



Interrupt Flag

Causes the 8086 to recognize external mask interrupts; clearing IF disables these interrupts.

What happens when an interrupt occurs?

Suppose an external device interrupts the CPU at the interrupt pin while the CPU is executing an instruction of a program.

1. The CPU first completes the execution of the current instruction.
2. The IP is then incremented to point to the next instruction.
3. If interrupt is a NMI, TRAP or Divide by Zero, the CPU acknowledges the requesting device on its INTA pin immediately.
4. If it is an INT request, the CPU checks the IF flag.
5. If the IF is set(1) , the interrupt request is acknowledged using the INTA pin.
6. If the IF is not set(0), the interrupt requests are ignored.
7. The responses to the NMI, TRAP and Divide by Zero interrupt requests are independent of the IF flag.

What happens when an interrupt occurs? (contd.)

8. After an interrupt is acknowledged , the CPU computes the vector address from the type of the interrupt.

9. This vector address may be passed to the interrupt structure of the CPU internally (in case of software interrupts, NMI, TRAP and Divide by Zero interrupts) or externally, i.e. from an interrupt controller in case of external interrupts.

10. (The contents of IP and CS are next pushed to the stack. The contents of IP and CS now point to the address of the next instruction of the main program from which the execution is to be continued after executing the ISR. The PSW is also pushed to the stack.)

11. The Interrupt Flag (IF) is cleared.

12. The TF is also cleared, after every response to the single step interrupt.

What happens when an interrupt occurs? (contd.)

13. The control is then transferred to the interrupt service routine for serving the interrupting device.

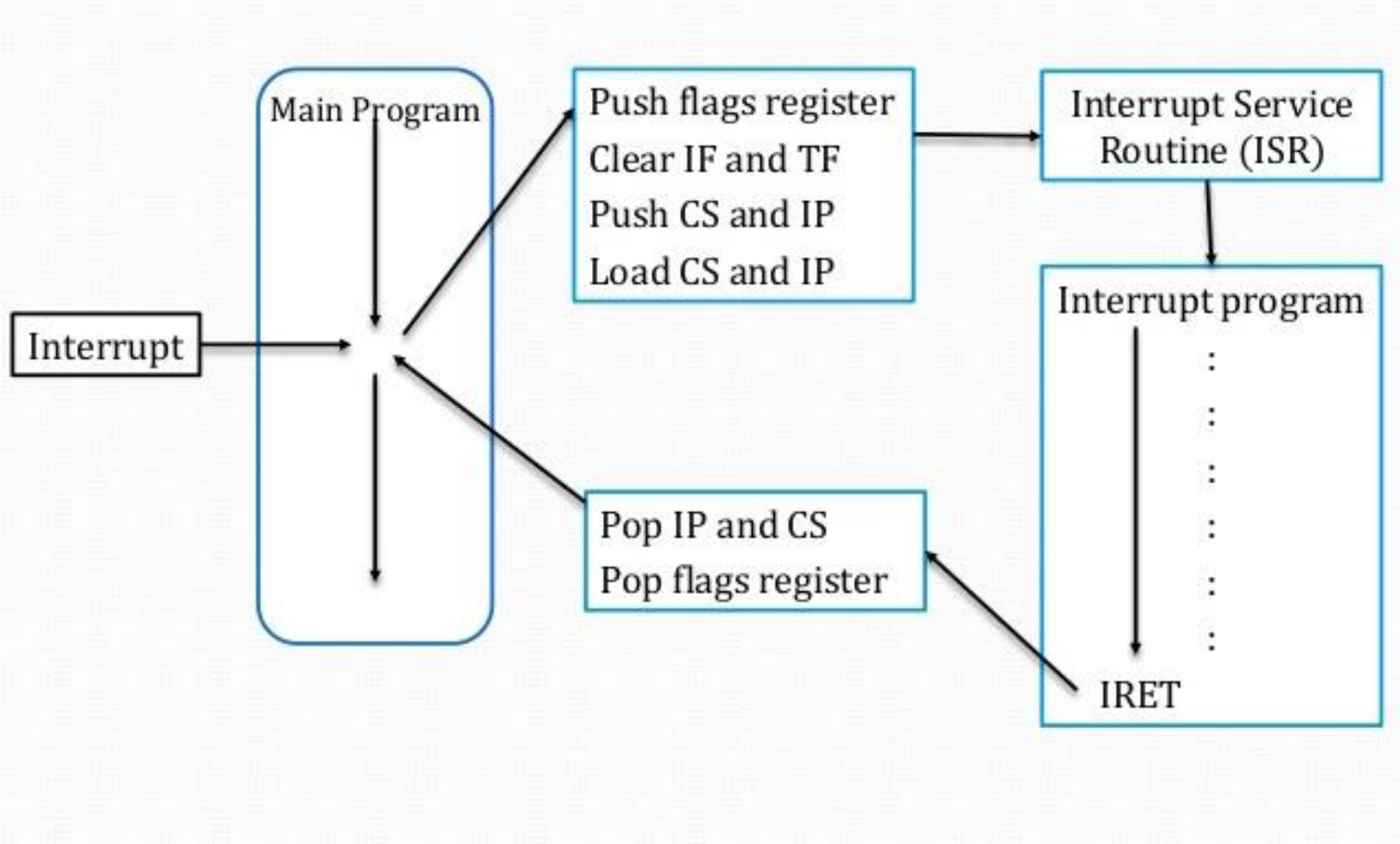
14. The new address of Interrupt Service Routine is found out from the interrupt vector table. The execution of the ISR starts.

15. At the end of ISR the last instruction should be IRET.

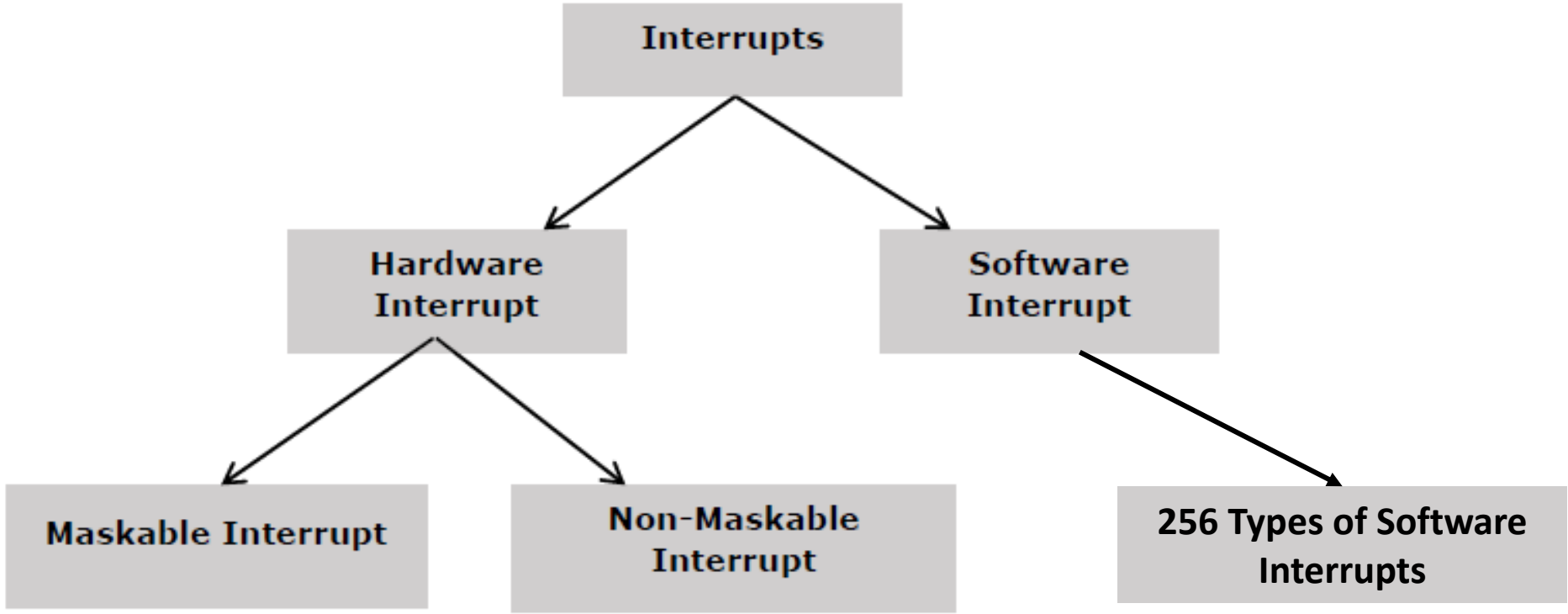
16. When the CPU executes IRET , the contents of flags, IP and CS which were saved at the start by the CALL instruction are now retrieved to the respective registers.

17. The execution continues onwards from this address, received by IP and CS.

What happens when an interrupt occurs?



Types of Interrupts

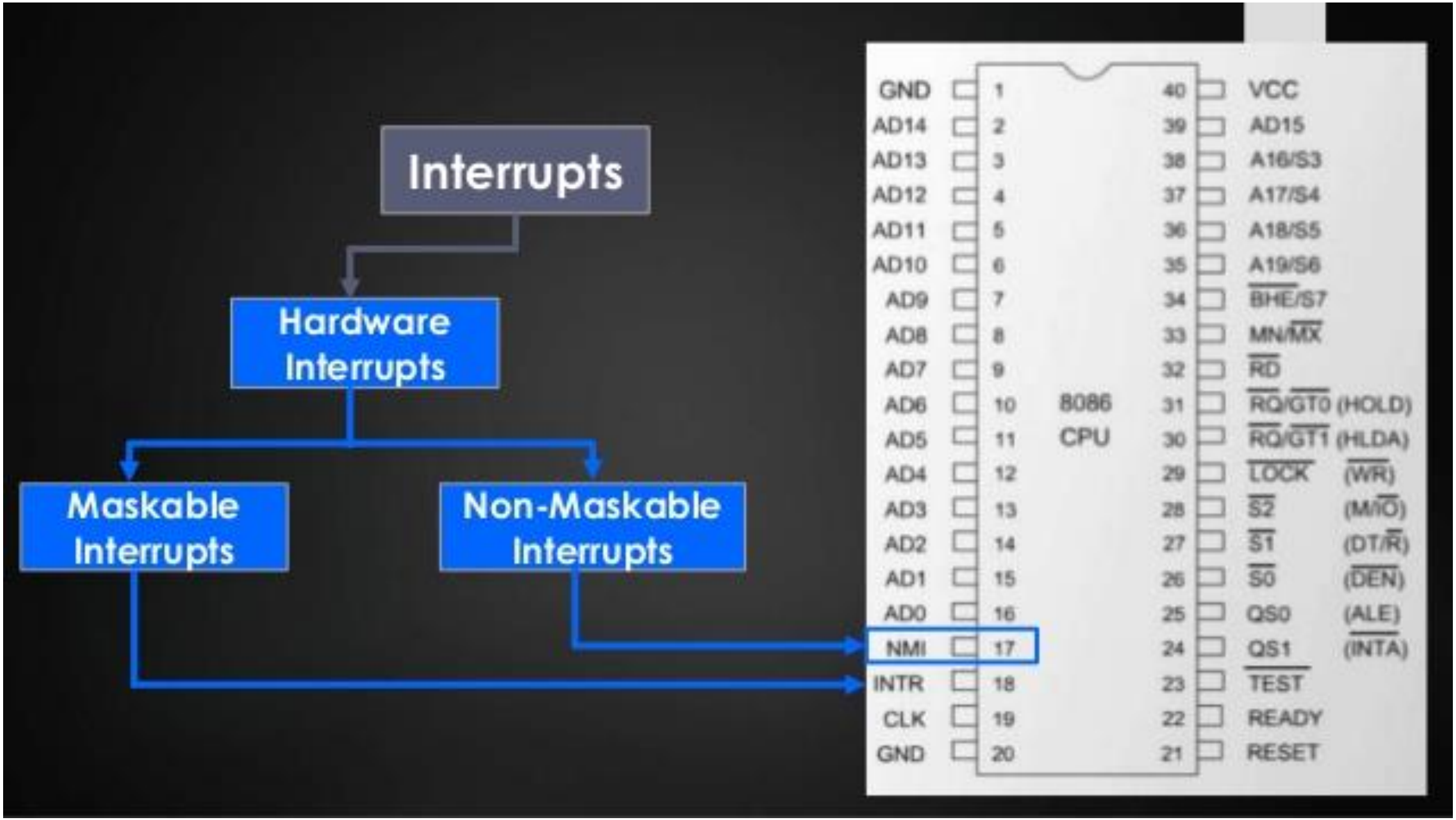


Hardware Interrupts

- Hardware interrupt is caused by any peripheral device by sending a signal through a specified pin to the microprocessor.

- The 8086 has two hardware interrupt pins, i.e. NMI and INTR.

- NMI is a non-maskable interrupt and INTR is a maskable interrupt having lower priority. One more interrupt pin associated is INTA called interrupt acknowledge.


Interrupts
Hardware Interrupts
Maskable Interrupts
Non-Maskable Interrupts
GND 1
AD14 2
AD13 3
AD12 4
AD11 5
AD10 6
AD9 7
AD8 8
AD7 9
AD6 10
AD5 11
AD4 12
AD3 13
AD2 14
AD1 15
AD0 16
NMI 17
INTR 18
CLK 19
GND 20
8086 CPU
40 VCC
39 AD15
38 A16/S3
37 A17/S4
36 A18/S5
35 A19/S6
34 BHE/S7
33 MN/MX
32 RD
31 RQ/GT0 (HOLD)
30 RQ/GT1 (HLDA)
29 LOCK (WR)
28 S2 (M/IO)
27 S1 (DT/R)
26 S0 (DEN)
25 QS0 (ALE)
24 QS1 (INTA)
23 TEST
22 READY
21 RESET

Software Interrupts

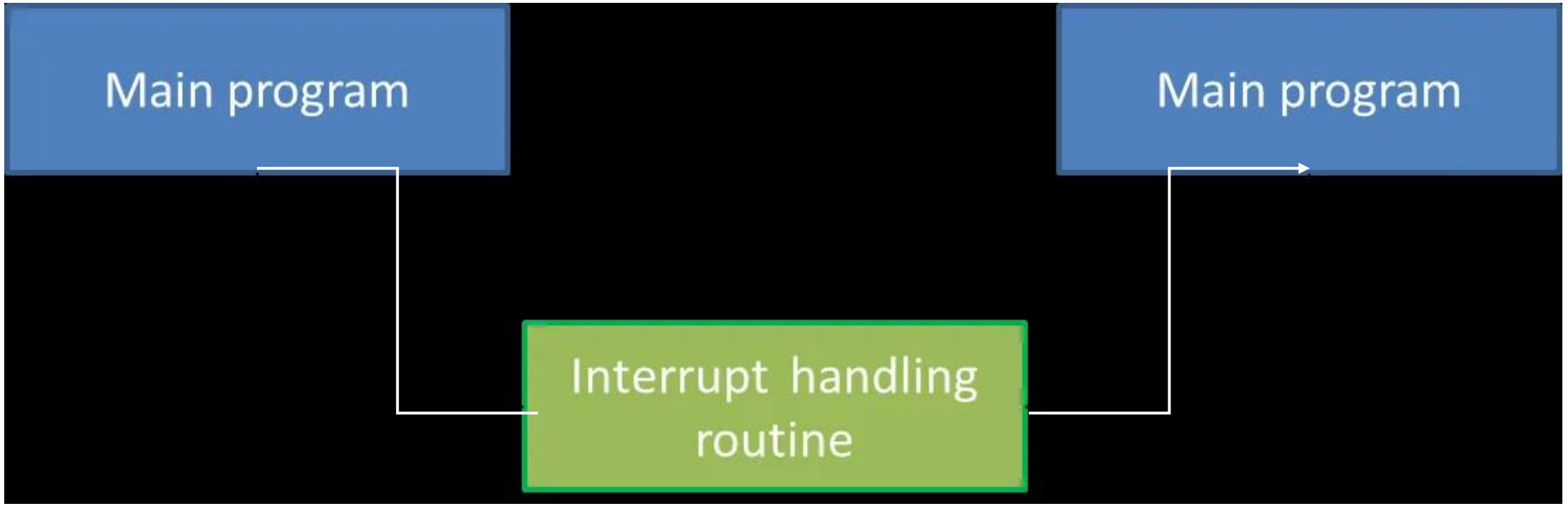
- Some instructions are inserted at the desired position into the program to create interrupts.

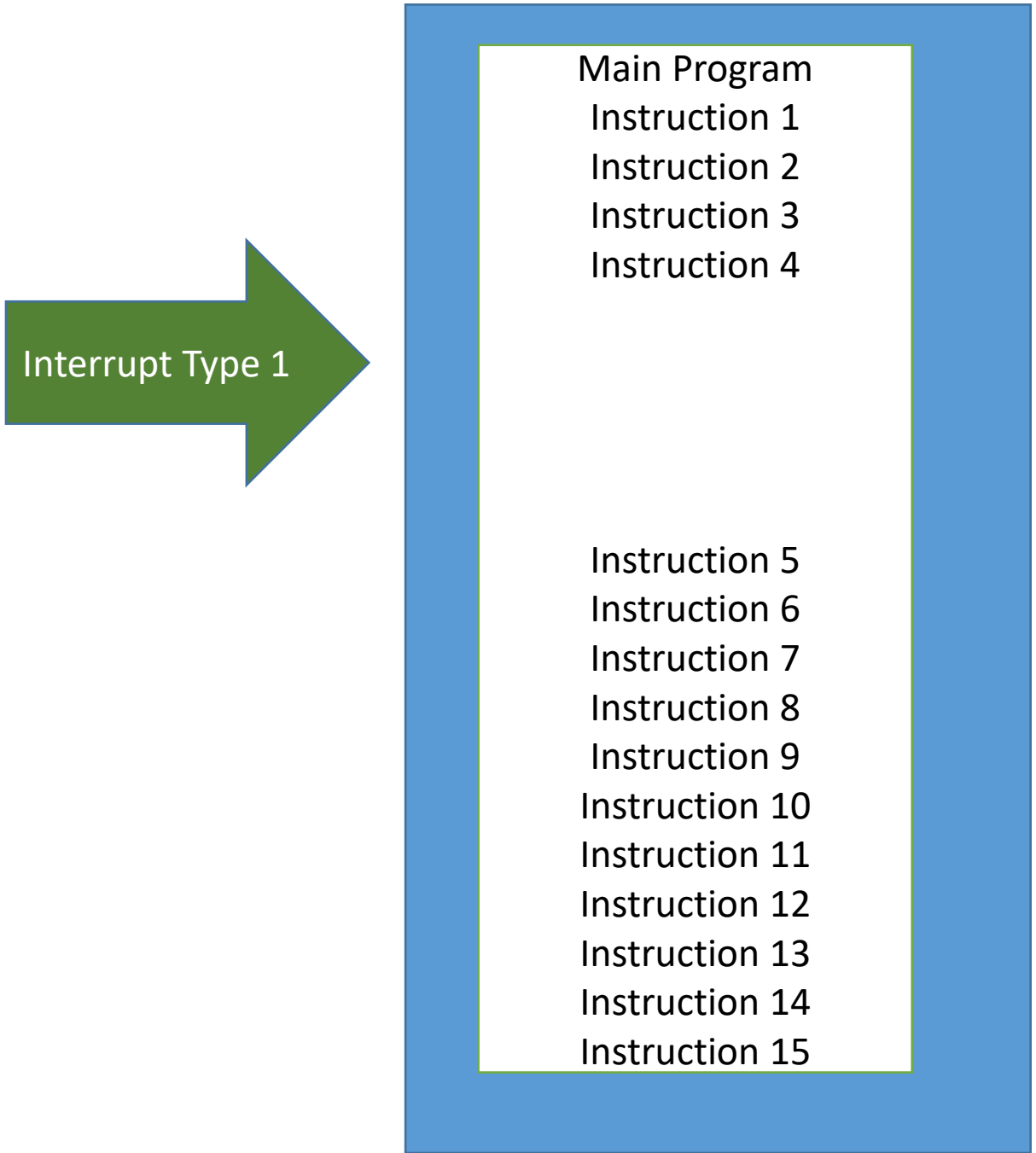
- These interrupt instructions can be used to test the working of various interrupt handlers. It includes −

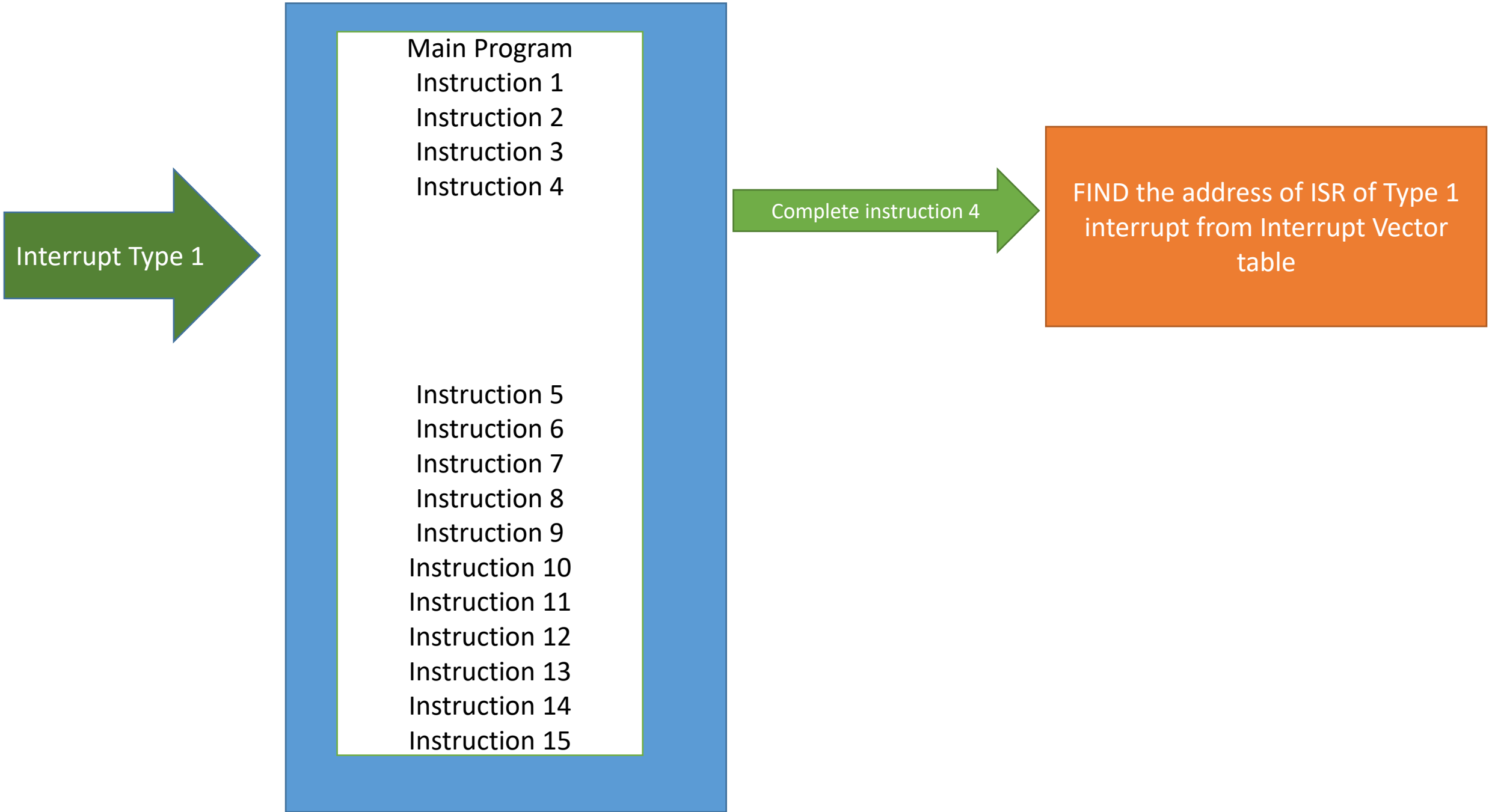
- **INT- Interrupt instruction with type number**

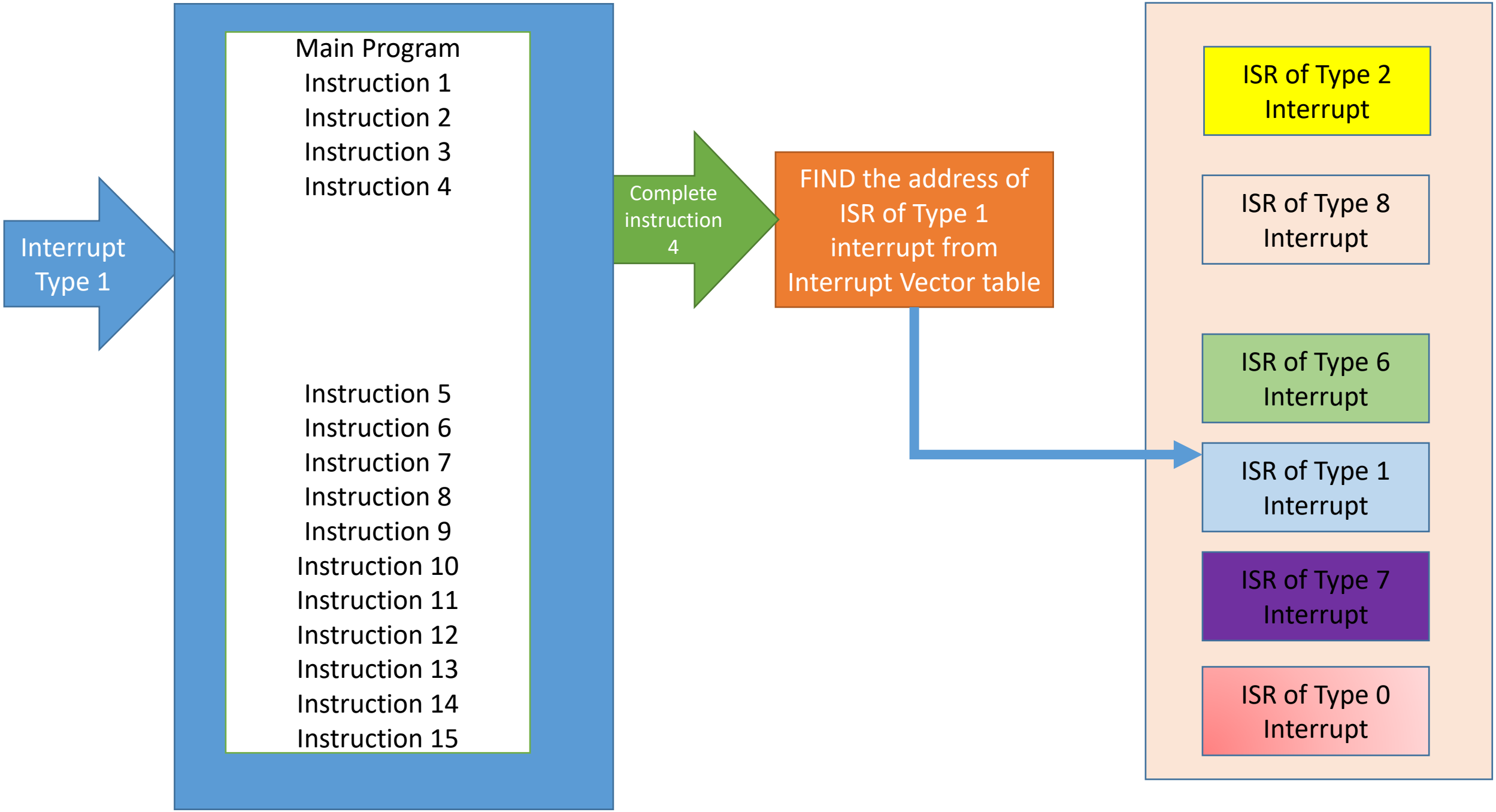
- **TYPE 0** interrupt represents division by zero situation.
- **TYPE 1** interrupt represents single-step execution during the debugging of a program.
- **TYPE 2** interrupt represents non-maskable NMI interrupt.
- **TYPE 3** interrupt represents break-point interrupt.
- **TYPE 4** interrupt represents overflow interrupt.

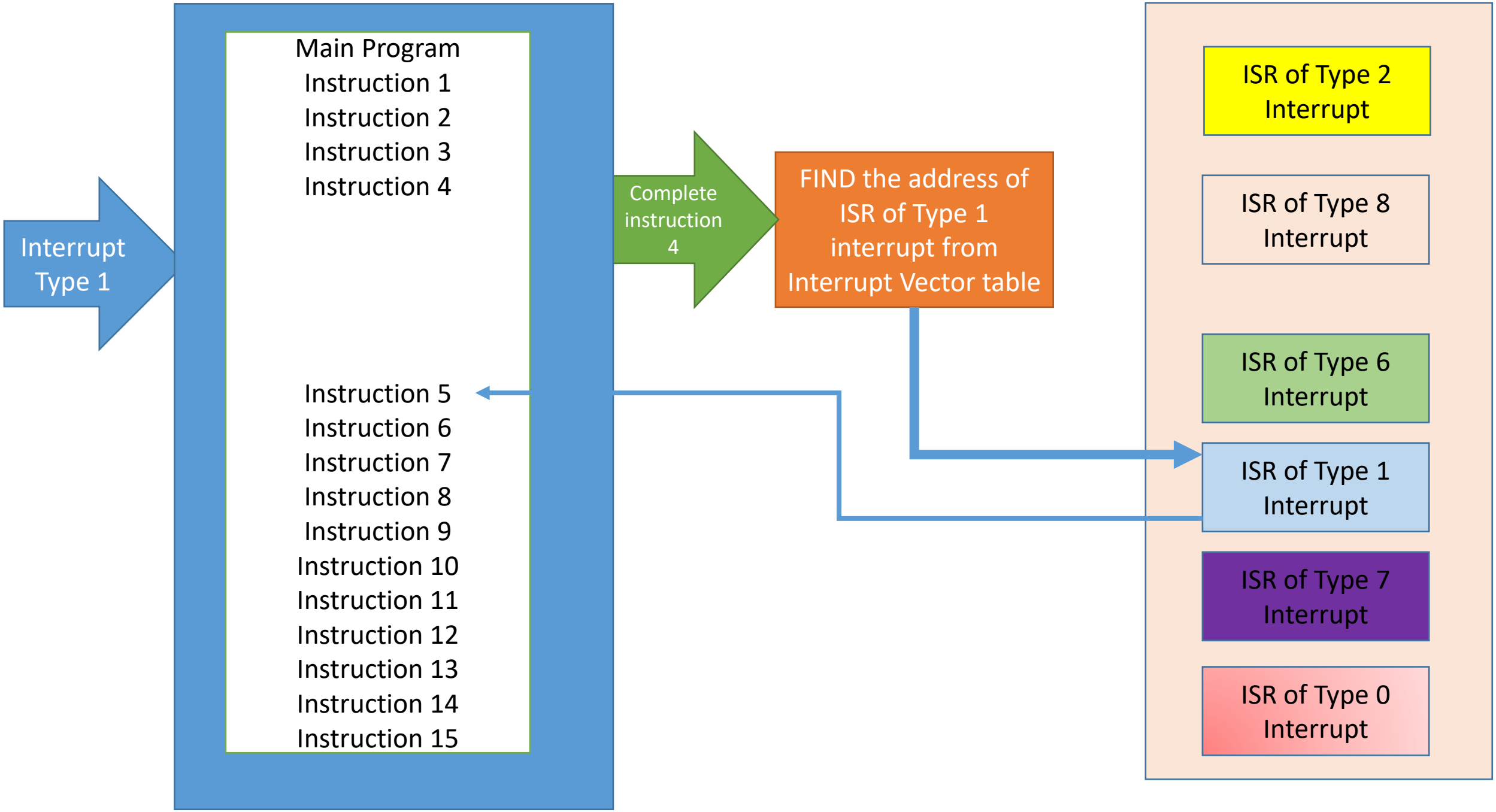
Interrupt Service Routine




Interrupt Type 1
Main Program
Instruction 1
Instruction 2
Instruction 3
Instruction 4
Instruction 5
Instruction 6
Instruction 7
Instruction 8
Instruction 9
Instruction 10
Instruction 11
Instruction 12
Instruction 13
Instruction 14
Instruction 15


Interrupt Type 1
Main Program
Instruction 1
Instruction 2
Instruction 3
Instruction 4
Instruction 5
Instruction 6
Instruction 7
Instruction 8
Instruction 9
Instruction 10
Instruction 11
Instruction 12
Instruction 13
Instruction 14
Instruction 15
Complete instruction 4
FIND the address of ISR of Type 1 interrupt from Interrupt Vector table


Main Program
Instruction 1
Instruction 2
Instruction 3
Instruction 4
Instruction 5
Instruction 6
Instruction 7
Instruction 8
Instruction 9
Instruction 10
Instruction 11
Instruction 12
Instruction 13
Instruction 14
Instruction 15
Interrupt Type 1
Complete instruction 4
FIND the address of ISR of Type 1 interrupt from Interrupt Vector table
ISR of Type 2 Interrupt
ISR of Type 8 Interrupt
ISR of Type 6 Interrupt
ISR of Type 1 Interrupt
ISR of Type 7 Interrupt
ISR of Type 0 Interrupt


Main Program
Instruction 1
Instruction 2
Instruction 3
Instruction 4
Instruction 5
Instruction 6
Instruction 7
Instruction 8
Instruction 9
Instruction 10
Instruction 11
Instruction 12
Instruction 13
Instruction 14
Instruction 15
Interrupt Type 1
Complete instruction 4
FIND the address of ISR of Type 1 interrupt from Interrupt Vector table
ISR of Type 2 Interrupt
ISR of Type 8 Interrupt
ISR of Type 6 Interrupt
ISR of Type 1 Interrupt
ISR of Type 7 Interrupt
ISR of Type 0 Interrupt

How the 8086 finds out the address of an ISR?

- Every external and internal interrupt is assigned with a type (N), that is either implicit (in case of NMI, TRAP and divide by zero) or specified in the instruction INT N (in case of internal interrupts).
- In case of external interrupts, the type is passed to the processor by an external hardware like programmable interrupt controller.
- In the zeroth segment of physical address space, i.e. CS = 0000, Intel has reserved 1,024 locations for storing the interrupt vector table.
- The 8086 supports a total of 256 types of the interrupts, i.e. from 00 to FFH.

How the 8086 finds out the address of an ISR? (contd.)

- Each interrupt requires 4 bytes, i.e. two bytes each for IP and CS of its ISR.
- Thus a total of 1,024 bytes are required for 256 interrupt types, hence the interrupt vector table starts at location 0000:0000 and ends at 0000:03FFH.
- The interrupt vector table contains the IP and CS of all the interrupt types stored sequentially from address 0000:0000 to 0000:03FF H.
- The interrupt type N is multiplied by 4 and the hexadecimal multiplication obtained gives the offset address in the zeroeth code segment at which the IP and CS addresses of the interrupt service routine (ISR) are stored.
- The execution automatically starts from the new CS:IP.

Interrupt Response Sequence

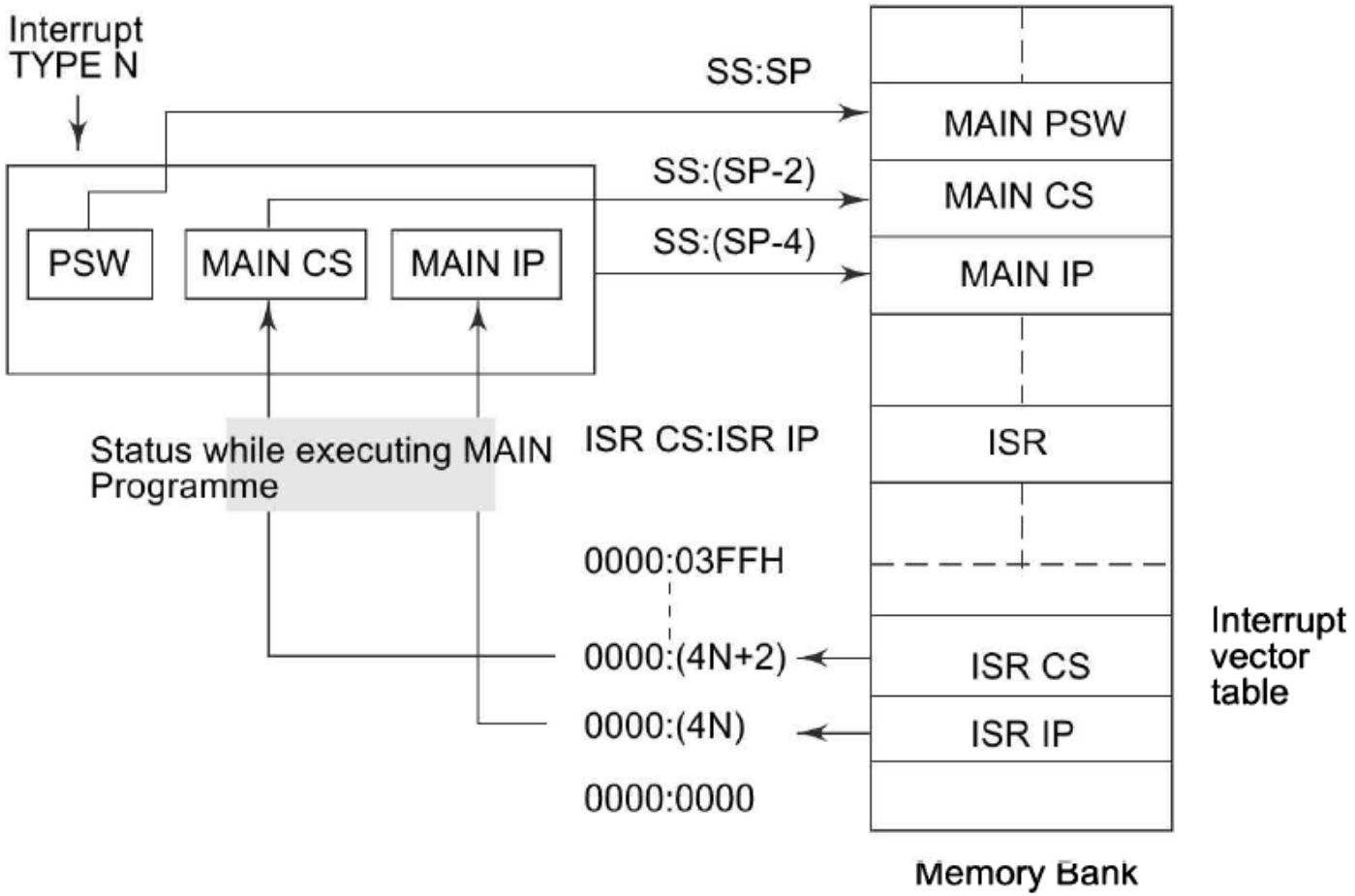


Fig. 4.4 *Interrupt Response Sequence*

Structure of Interrupt Vector Table

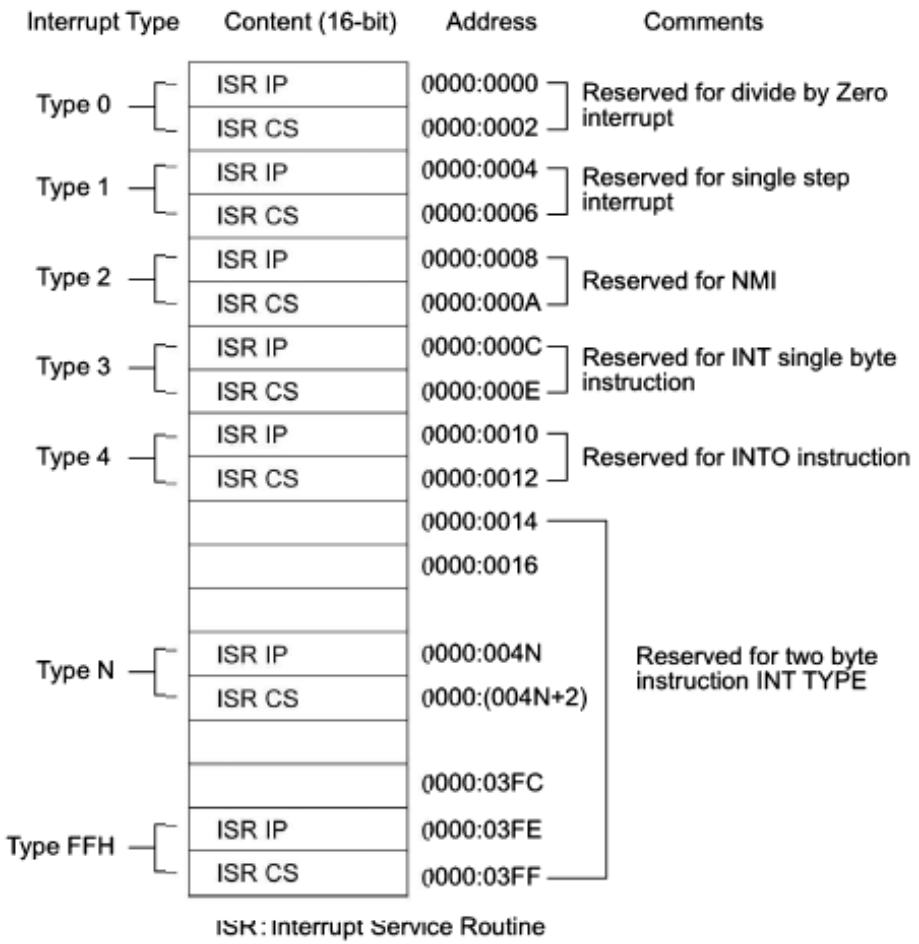
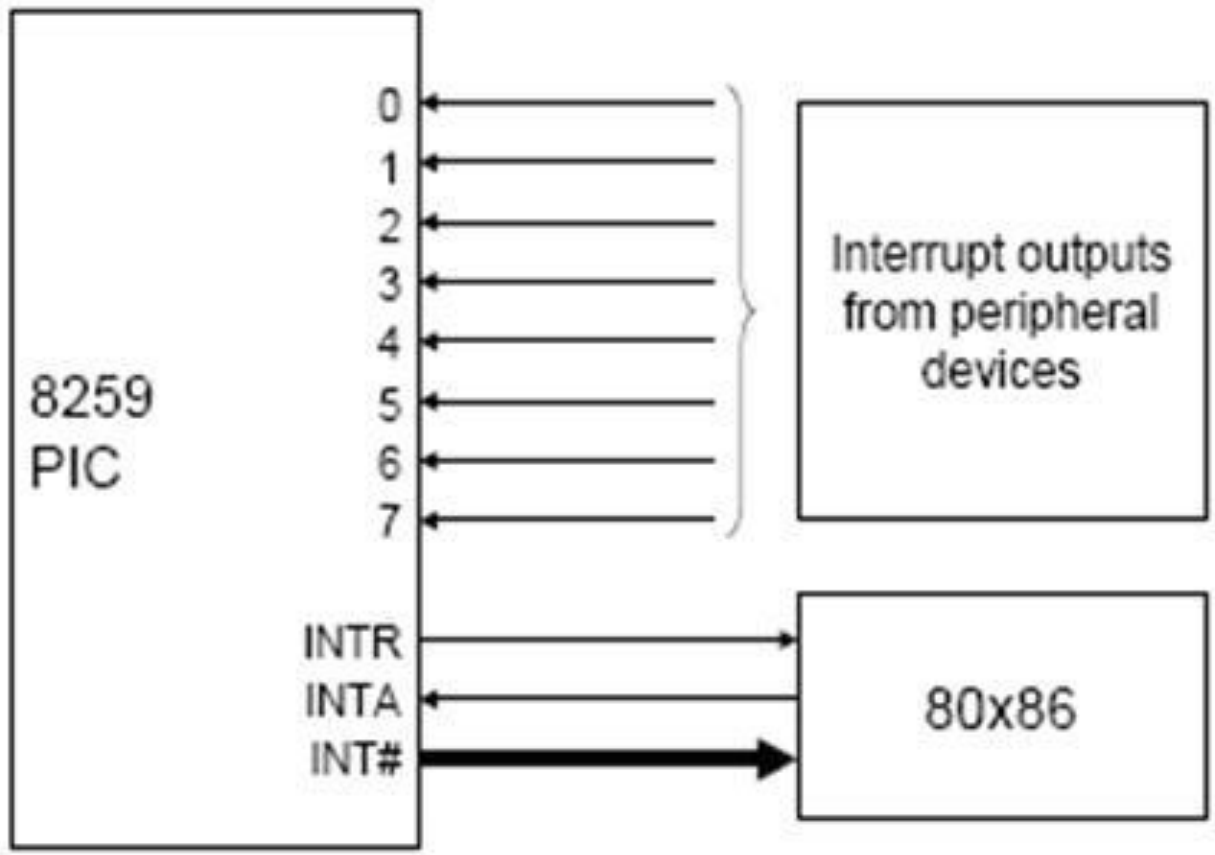


Fig. 4.5 *Structure of Interrupt Vector Table of 8086/88*

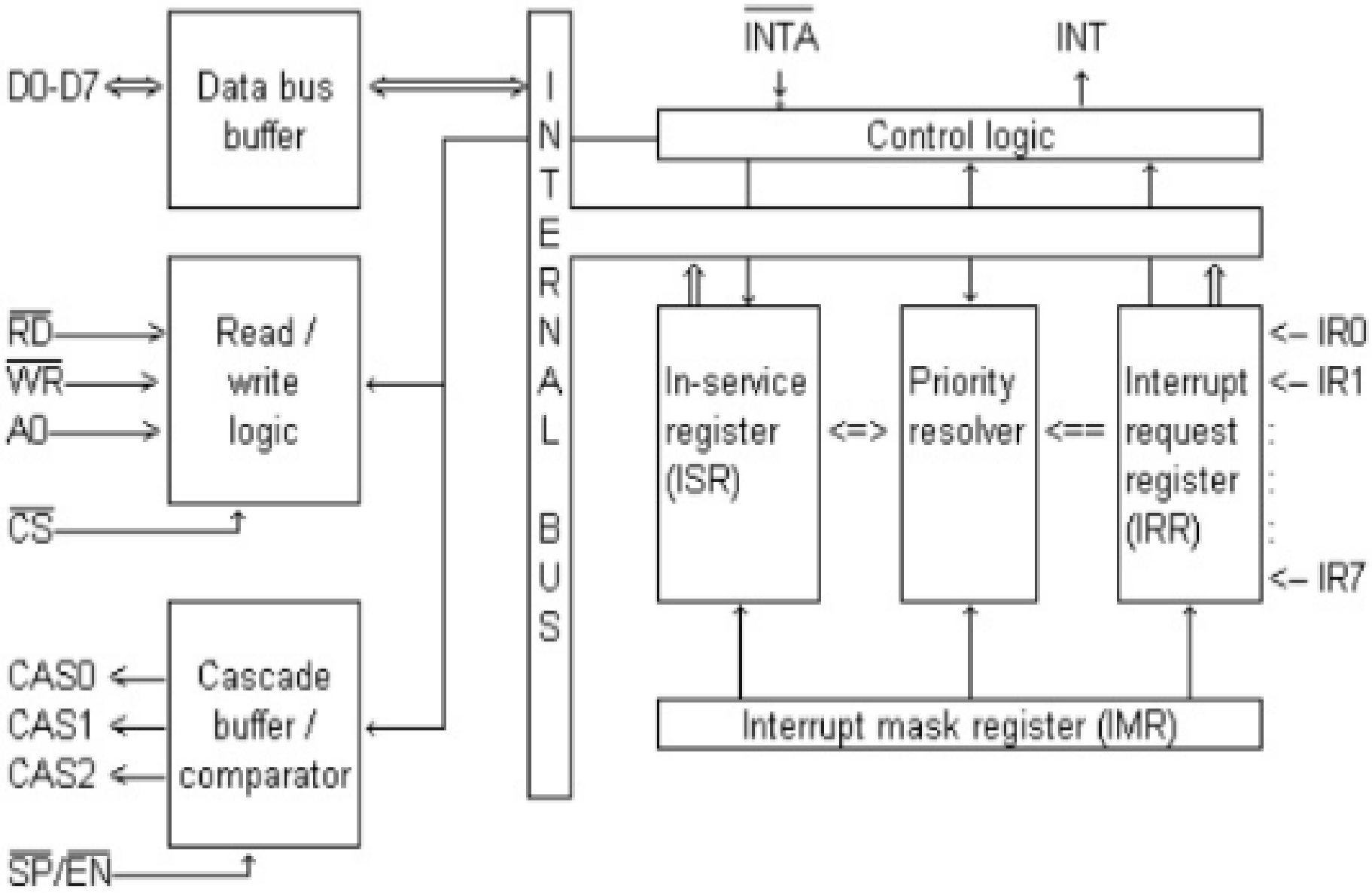
8259A - PROGRAMMABLE INTERRUPT CONTROLLER

- A programmable interrupt controller is able to handle a number of interrupts at a time.
- This controller takes care of a number of simultaneously appearing interrupt requests along with their types and priorities.
- This relieves the processor from all these tasks.
- The programmable interrupt controller 8259A from Intel is one such device.
- Its predecessor 8259 was designed to operate only with 8-bit processors like 8085. A modified version, 8259A was later introduced that is compatible with 8-bit as well as 16-bit processors.

The 8259 PIC


8259
PIC
0
1
2
3
4
5
6
7
INTR
INTA
INT#
Interrupt outputs
from peripheral
devices
80x86

8259 internal block diagram



8259 Architecture - The functional explanation

- **Interrupt Request Register (IRR) -** The interrupts at IRQ input lines are handled by Interrupt Request Register internally. IRR stores all the interrupt requests in it in order to serve them one by one on the priority basis.

- **In-Service Register (ISR)** - This stores all the interrupt requests those are being served, i.e. ISR keeps a track of the requests being served.

- **Priority Resolver** - This unit determines the priorities of the interrupt requests appearing simultaneously.

- The highest priority is selected and stored into the corresponding bit of ISR during INTA pulse.

- The IR_0 has the highest priority while the IR_7 has the lowest one, normally in fixed priority mode.

- The priorities however may be altered by programming the 8259A in rotating priority mode.

- **Interrupt Mask Register (IMR)** This register stores the bits required to mask the interrupt inputs. IMR operates on IRR at the direction of the Priority Resolver.

- **Interrupt Control Logic** This block manages the interrupt and the interrupt acknowledge signals to be sent to the CPU for serving one of the eight interrupt requests.

- This also accepts the interrupt acknowledge (INTA) signal from CPU that causes the 8259A to release vector address on to the data bus.

- **Data Bus Buffer** This buffer interfaces internal 8259A bus to the microprocessor system data bus.

- Control words, status and vector information pass through data buffer during read or write operations.

- **Read/Write Control Logic** This circuit accepts and decodes commands from the CPU.

- This block also allows the status of the 8259A to be transferred on to the data bus.

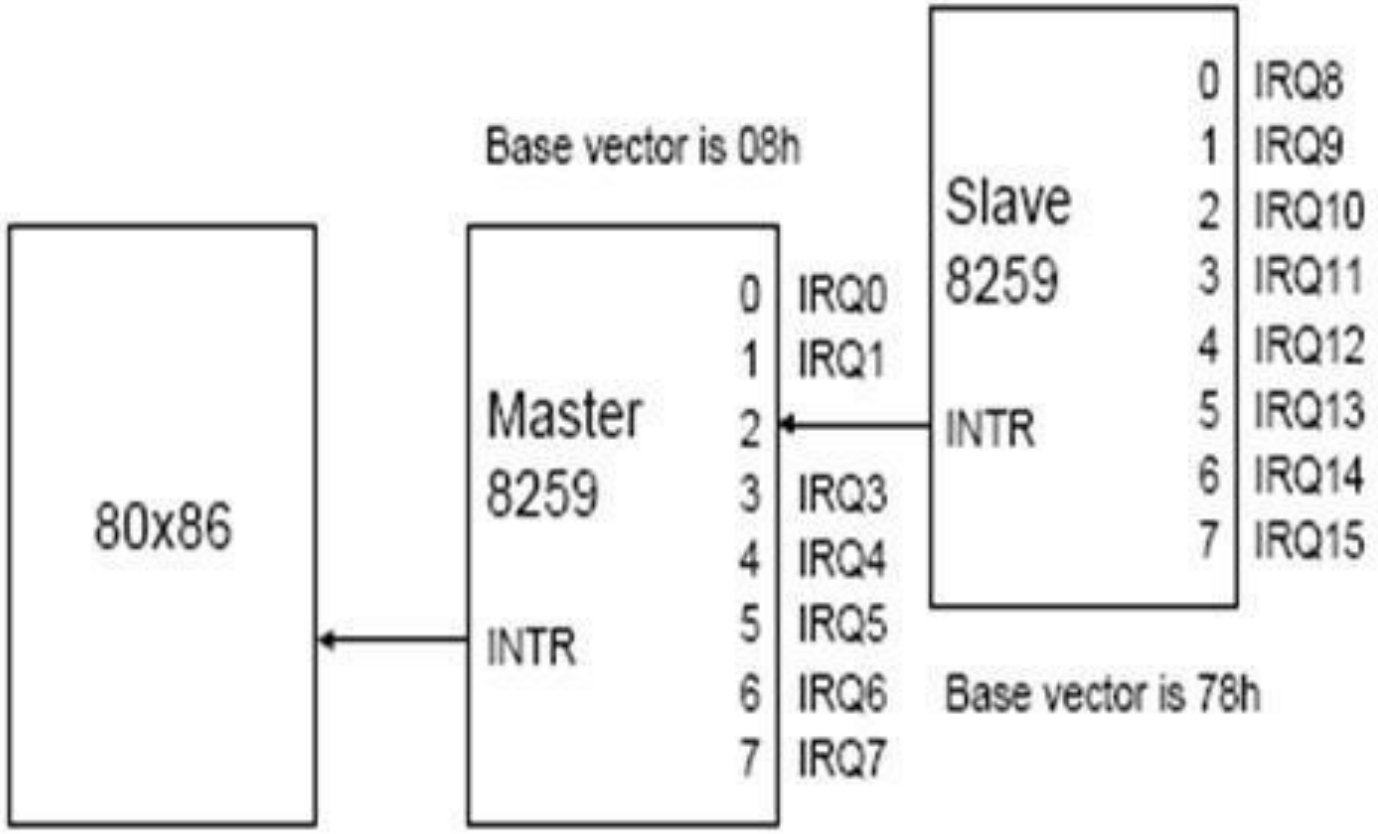
- **Cascade Buffer/Comparator**

- This block stores and compares the IDs of all the 8259As used in the system. The three I/O pins CASO-2 are outputs when the 8259A is used as a master.

- The same pins act as inputs when the 8259A is in the slave mode. The 8259A in the master mode, sends the ID of the interrupting slave device on these lines.

- The slave thus selected, will send its pre-programmed vector address on the data bus during the next INTA pulse.

Connecting Multiple Devices to Processors



Interrupt Sequence in an 8086-8259A System

The interrupt sequence in an 8086-8259A system is described as follows:

1. *When an interrupt is received in $IR_0 – IR_7$, corresponding IRR(Interrupt Req Reg) bit is set.*
2. *8259A resolves priority and sends an INT signal to CPU.*
3. *The CPU acknowledges with INTA pulse.*
4. *Upon receiving an INTA signal from the CPU, the highest priority ISR(In Service Register) bit is set and the corresponding IRR bit is reset. The 8259A does not drive data bus during this period.*
5. *The 8086 will initiate a second INTA pulse. During this period 8259A releases vector address on to data bus from where it is read by the CPU.*
6. *This completes the interrupt cycle.*

Interrupt Acknowledge Sequence of 8086

NON MASKABLE INTERRUPT

- The processor 8086/88 has a Non-Masksable Interrupt input pin (NMI), that has the highest priority among the external interrupts.

- TRAP(Single Step-Type 1) is an internal interrupt having the highest priority amongst all the interrupts except the Divide By Zero (Type0) exception.

- The NMI is activated on a positive transition (low to high voltage).

MASKABLE INTERRUPT (INTR)

- The processor 8086/88 also provides a pin INTR, that has lower priority as compared to NMI.
- Further the priorities within the INTR types are decided by the type of the INTR signal that is to be passed to the processor via data bus by some external device like the programmable interrupt controller.
- The INTR signal is level triggered and can be masked by resetting the interrupt flag. It is internally synchronized with the high transition of the CLK.
- For the INTR signal, to be responded to in the next instruction cycle, it must go high in the last clock cycle of the current instruction or before that.
- The INTR requests appearing after the last clock cycle of the current instruction will be responded to after the execution of the next instruction.
- The status of the pending interrupts is checked at the end of each instruction cycle.

- If the IF is set, the processor is ready to respond to any INTR interrupt if the IF is reset, the processor will not serve any interrupt appearing at this pin.

- However, once the processor responds to an INTR signal, the IF is automatically reset.

- If one wants the processor to further respond to any type of INTR signal, the IF should again be set. The interrupt acknowledge sequence is as shown in Fig. 4.6.

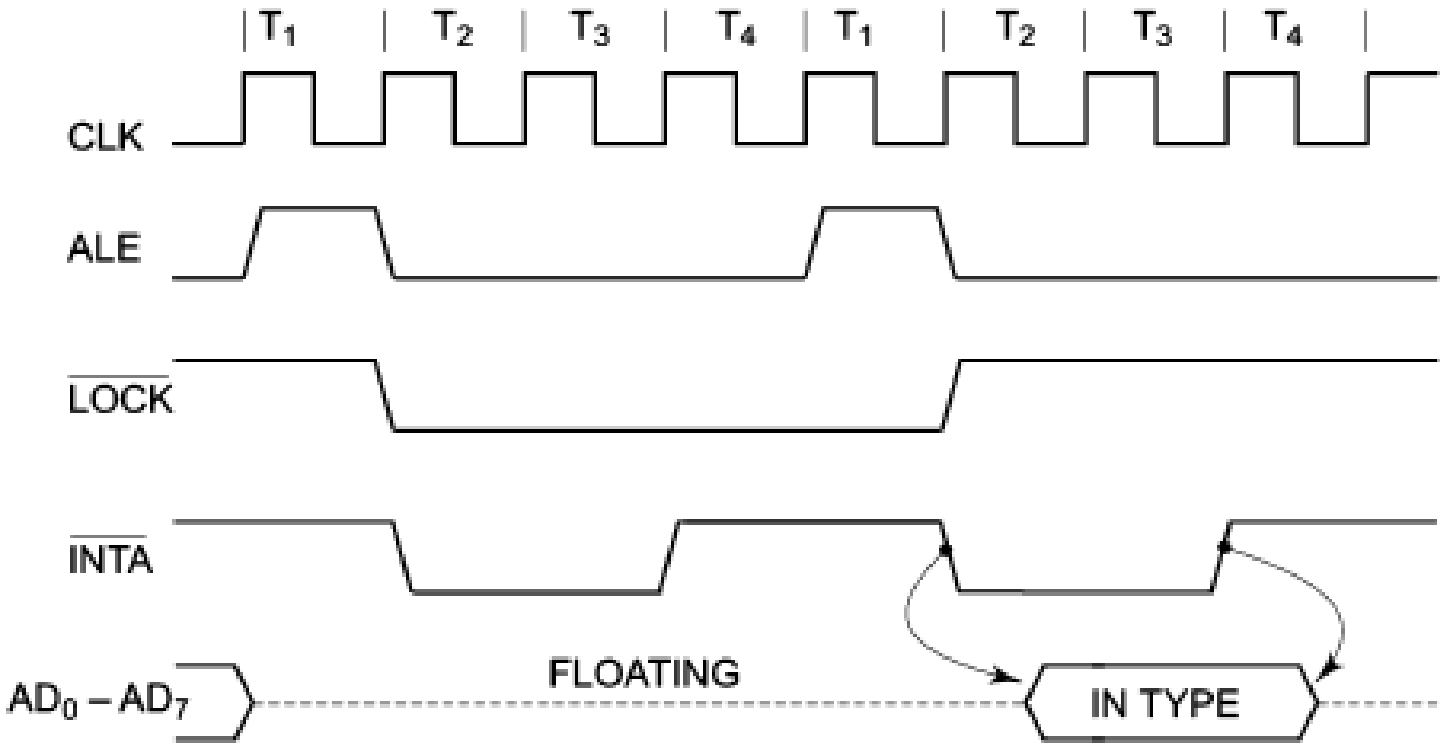


Fig. 4.6 *Interrupt Acknowledge Sequence of 8086*

- Suppose an external signal interrupts the processor and the pin $\overline{\text{LOCK}}$ goes low at the trailing edge of the first ALE pulse that appears after the interrupt signal preventing the use of bus for any other purpose.
- The pin $\overline{\text{LOCK}}$ remains low till the start of the next machine cycle.
- With the trailing edge of $\overline{\text{LOCK}}$, the $\overline{\text{INTA}}$ goes low and remains low for two clock states before returning back to the high state.
- It remains high till the start of the next machine cycle, i.e. next trailing edge of ALE.
- Then $\overline{\text{INTA}}$ again goes low, remains low for two states before returning to the high state. The first trailing edge of ALE floats the bus AD_0-AD_7 , while the second trailing edge prepares the bus to accept the type of
- the interrupt. The type of the interrupt remains on the bus for a period of two cycles.

INTERRUPT PROGRAMMING

- While programming for any type of interrupt, the programmer must, either externally or through the program, set the interrupt vector table for that type preferably with the CS and IP addresses of the interrupt service routine.

- The method of defining the interrupt service routine for software as well as hardware interrupt is the same.

- Figure 4.7 shows the execution sequence in case of a software interrupt. It is assumed that the interrupt vector table is initialised suitably to point to the interrupt service routine.

- Figure 4.8 shows the transfer of control for the nested interrupts.

Nested Interrupts

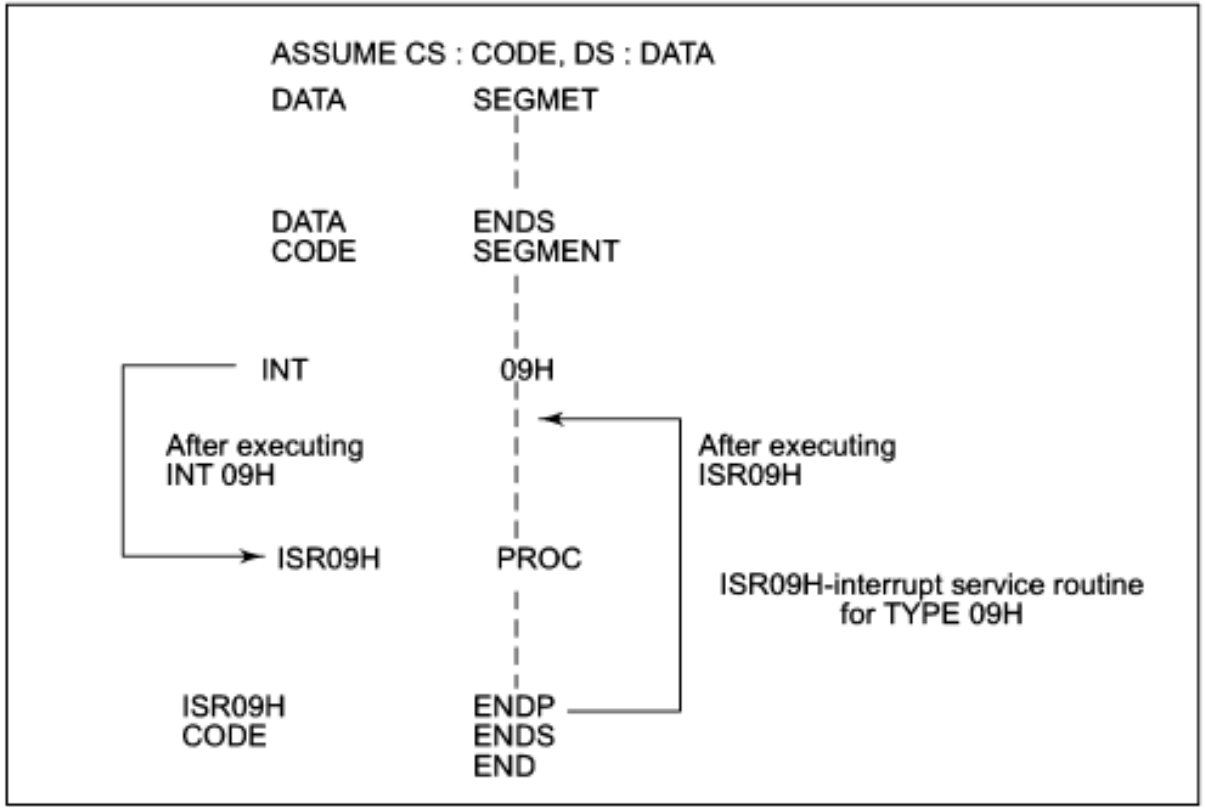


Fig. 4.7 *Transfer of Control during Execution of an Interrupt Service Routine*

Nested Interrupts

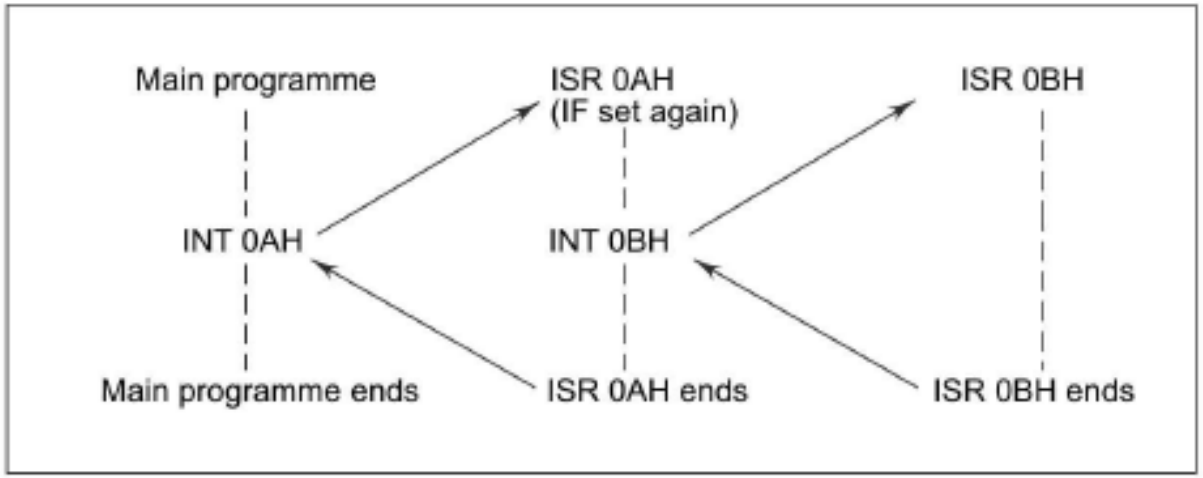


Fig. 4.8 ***Transfer of Control for Nested Interrupts***

Example: Write a program that gives display `IRT2 is OK' if a hardware signal appears on $IRQ_2$ pin and `IRT3 is OK' if it appears on IRQ_3 pin of PC 10 Channel.

```
ASSUME     CS:CODE, DS:DATA
DATA         SEGMENT
MSG1         DB "IRT2 IS OK" ,0AH, 0DH, "$"
MSG2         DB "IRT3 IS OK",0AH,0DH, "$"
DATA         ENDS
CODE         ES

START:     MOV AX, CODE
           MOV DS, AX                ; Set IVT for Type 0AH
           MOV DX, OFFSET ISR1
           MOV AX,250AH              ; IRQ2 is equivalent to Type 0AH
           INT 21H
           MOV DX, OFFSET ISR2       ; Set IVT for Type 0BH
           MOV AX, 250BH             ; IRQ3 is equivalent to TYPE 0BH
           INT 21H
HERE :     JUMP HERE
                                     ; ISR1 and ISR2 dispaly the message
ISR1       PROC  LOCAL
           MOV AX, DATA
           MOV DS, AX
```

```
            MOV DX, OFFSET MSG1             ; Display message MSG1
            MOV AH, 09H
            INT 21 H
            IRET
ISR1        ENDP
ISR2        PROC LOCAL
            MOV AX, DATA
            MOV DS, AX
            MOV DX, OFFSET MSG2             ; Display message MSG2
            MOV AH,09H
            INT 21H
            IRET
ISR2        ENDP
CODE        ENDS
    END     START
```