Interrupts and Interrupt Service Routines

WHAT IS AN INTERRUPT?

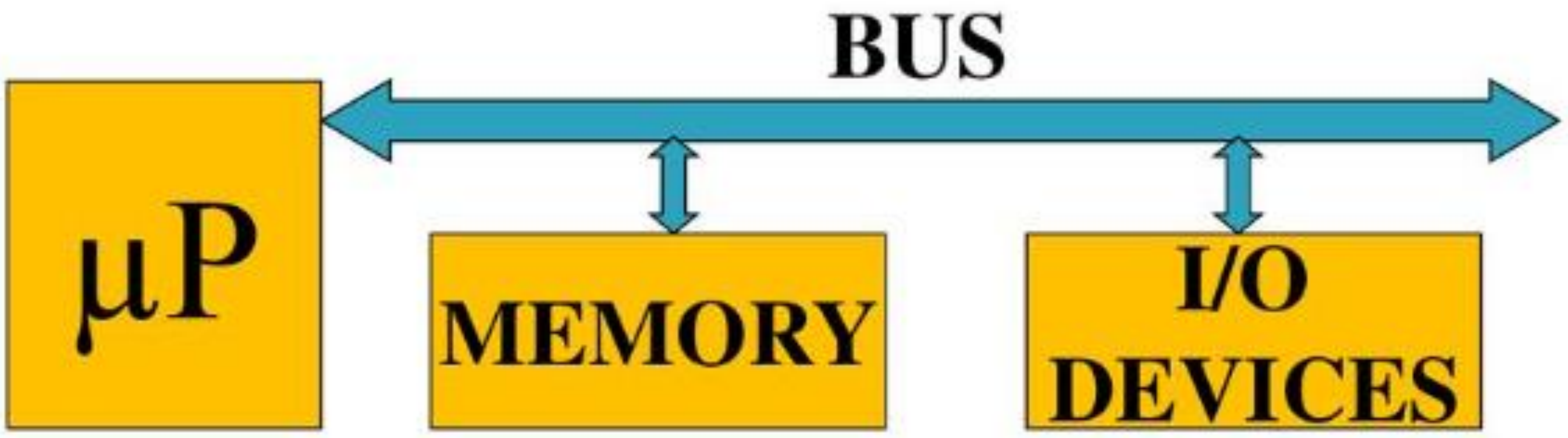
- **Capability to suspend the execution of running program and execution of another program to fulfill specific requirement upon request**

- **After finishing the second program, automatically return to the first program and start execution from where it was left**

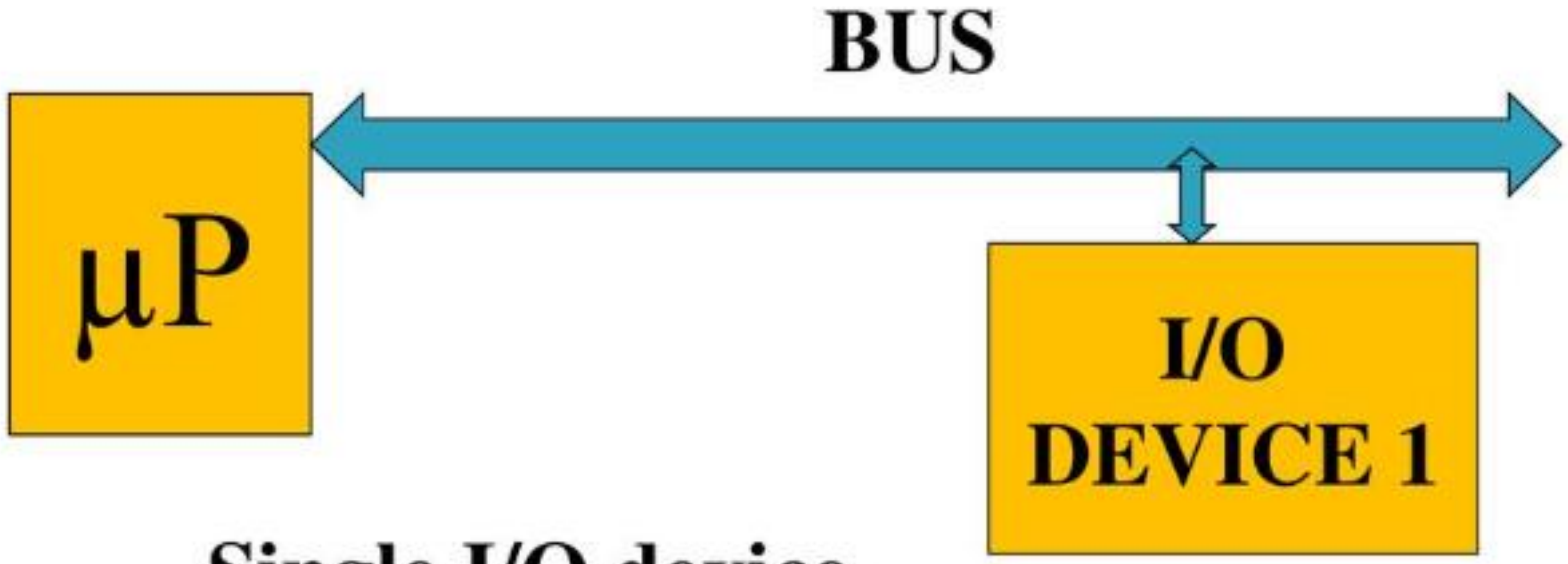
WHY INTERRUPT CAPABILITY IS NECESSARY?

- To understand this - review of how µP/µC communicate with the outside world must be understood.

BASIC BLOCK DIAGRAM OF A MP

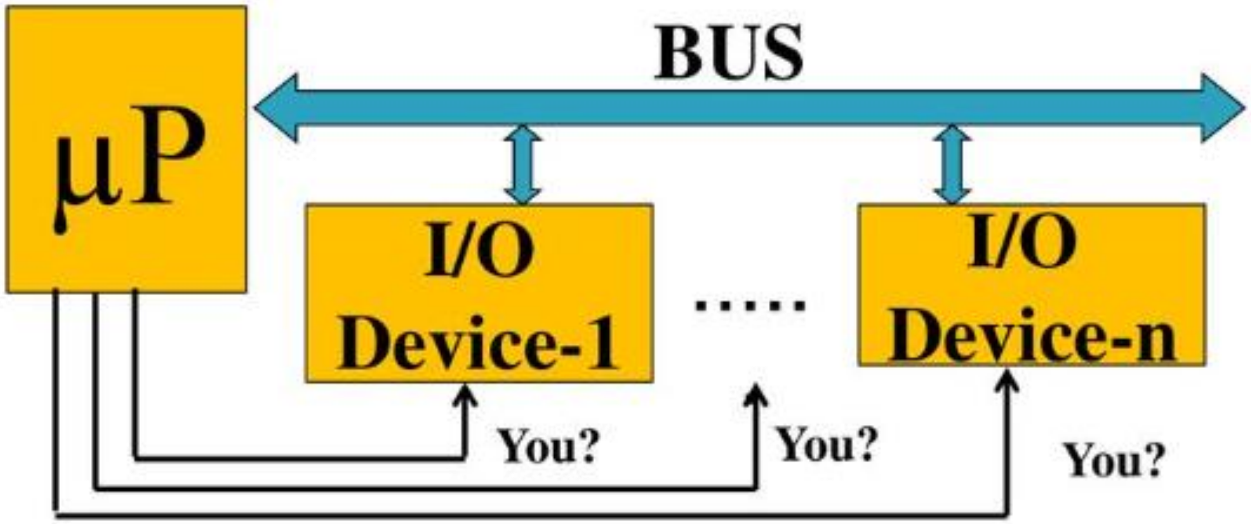


First type of communication - Dedicated I/O



Single I/O device

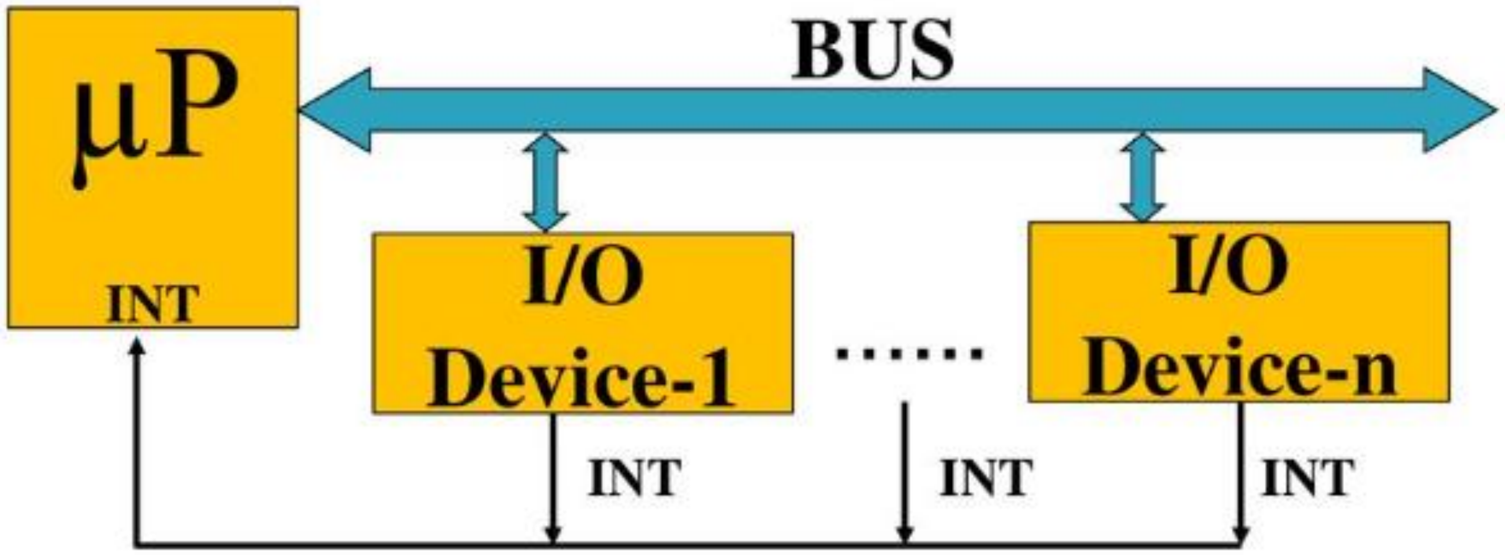
SECOND TYPE OF COMMUNICATION - POLLED I/O OR PROGRAMMED I/O



Disadvantage:

- Not fast enough
- Wastes too much microprocessor time

Third type of communication - Interrupted I/O



- Interrupts are useful when I/O devices are slow

- Most microprocessors allow normal program execution to be interrupted by some external signal or by a special instruction in the program.
- In response to an interrupt, the microprocessor stops executing its current program and calls a procedure which “services” the interrupt.
- A special instruction --- IRET --- at the end of interrupt-service procedure returns execution to the interrupted main program.

SOURCES OF 8086 INTERRUPTS

8086 interrupts can be classified into two types:

- **1) Predefined interrupt**
  Some error condition produced by execution of an instruction, e.g., trying to divide some number by zero. (*Interrupt due to exceptions*)

- **2) User defined interrupt**
  i) *Hardware interrupt*
  An external signal applied to *NMI*, *INTR* pins
  ii) *Software interrupt*
  Execution of interrupt instruction INT

NMI (Non Maskable Interrupt):

Any interrupt request at NMI input pin cannot be masked or disabled by any means; type is implicit

INTR:

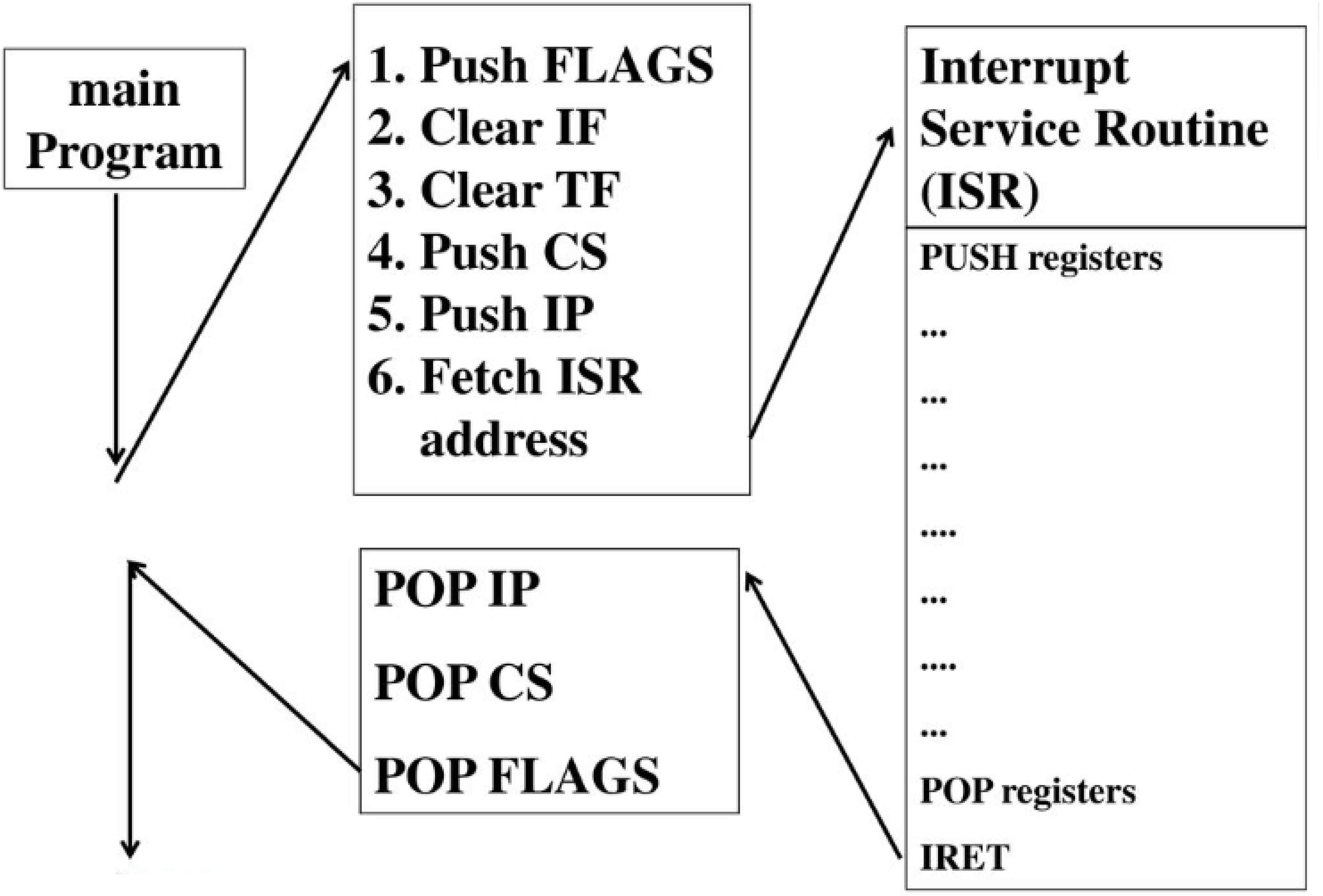
This hardware interrupt can be masked using Interrupt Flag (IF); 256 Types (00h to FFh); to handle more than one interrupts that occur at a time, Programmable Interrupt Controller is required.

INT:

This is a software interrupt; the type is specified in the instruction

At the end of each instruction cycle, 8086 checks if any interrupt service has been requested.

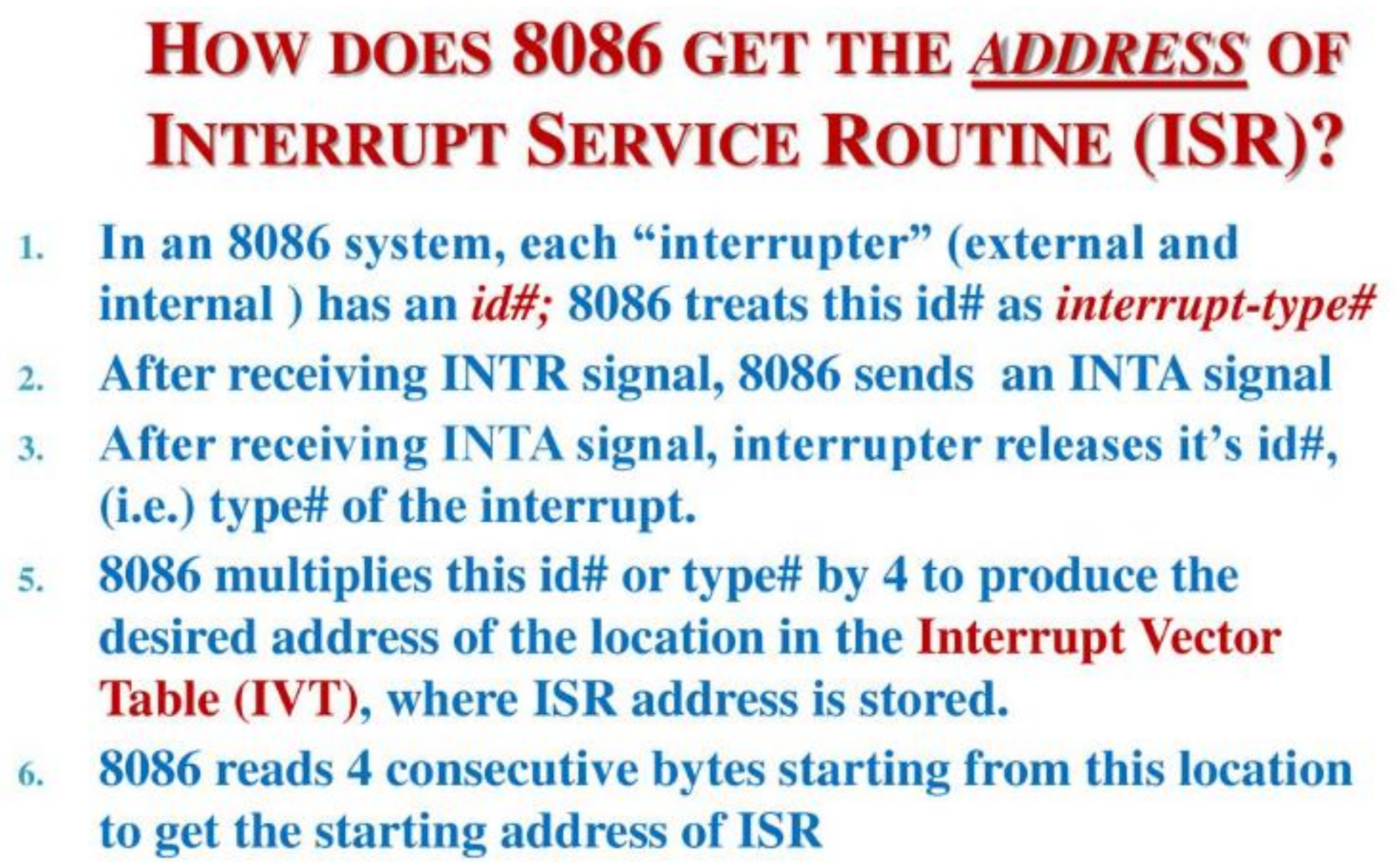
1. **8086 decrements SP by 2 and pushes flag register content into the stack.**
2. **8086 disables INTR input by clearing IF flag in flag register**
3. **The TF (trap) flag in flag register is Reset**
4. **Decrements SP again by 2 and pushes current CS content into the stack.**
5. **Decrements SP again by 2 and pushes current IP content into the stack.**
6. **Does an indirect far jump to the start of the procedure written to respond to the interrupt.**


main Program
1. Push FLAGS
2. Clear IF
3. Clear TF
4. Push CS
5. Push IP
6. Fetch ISR address
Interrupt Service Routine (ISR)
PUSH registers
...
...
...
....
...
....
...
POP registers
IRET
POP IP
POP CS
POP FLAGS

How does 8086 get to Interrupt Service Routine?

1. **8086 loads its CS and IP registers with the *address of ISR*.**

2. **So, the next instruction to be executed is the first instruction of ISR**



How does 8086 get the *address* of Interrupt Service Routine (ISR)?

1. In an 8086 system, each "interrupter" (external and internal) has an *id#;* 8086 treats this id# as *interrupt-type#*
2. After receiving INTR signal, 8086 sends an INTA signal
3. After receiving INTA signal, interrupter releases it's id#, (i.e.) type# of the interrupt.
5. 8086 multiplies this id# or type# by 4 to produce the desired address of the location in the Interrupt Vector Table (IVT), where ISR address is stored.
6. 8086 reads 4 consecutive bytes starting from this location to get the starting address of ISR

First two bytes are loaded into IP
Second two bytes are loaded into CS

INTERRUPT VECTOR TABLE (IVT)

- In an 8086 system, the first 1Kbytes (1024 bytes) of memory, from 00000 to 003FF, is set aside as a Table for storing the starting addresses of interrupt service routines.

- Since 4 bytes are required to store CS and IP values for each ISR, the Table can hold the starting addresses for up to 256 ISRs.

- The starting address of an ISR is often called the interrupt vector or the interrupt pointer.

- So the Table is referred to as interrupt-vector table or interrupt-pointer table.

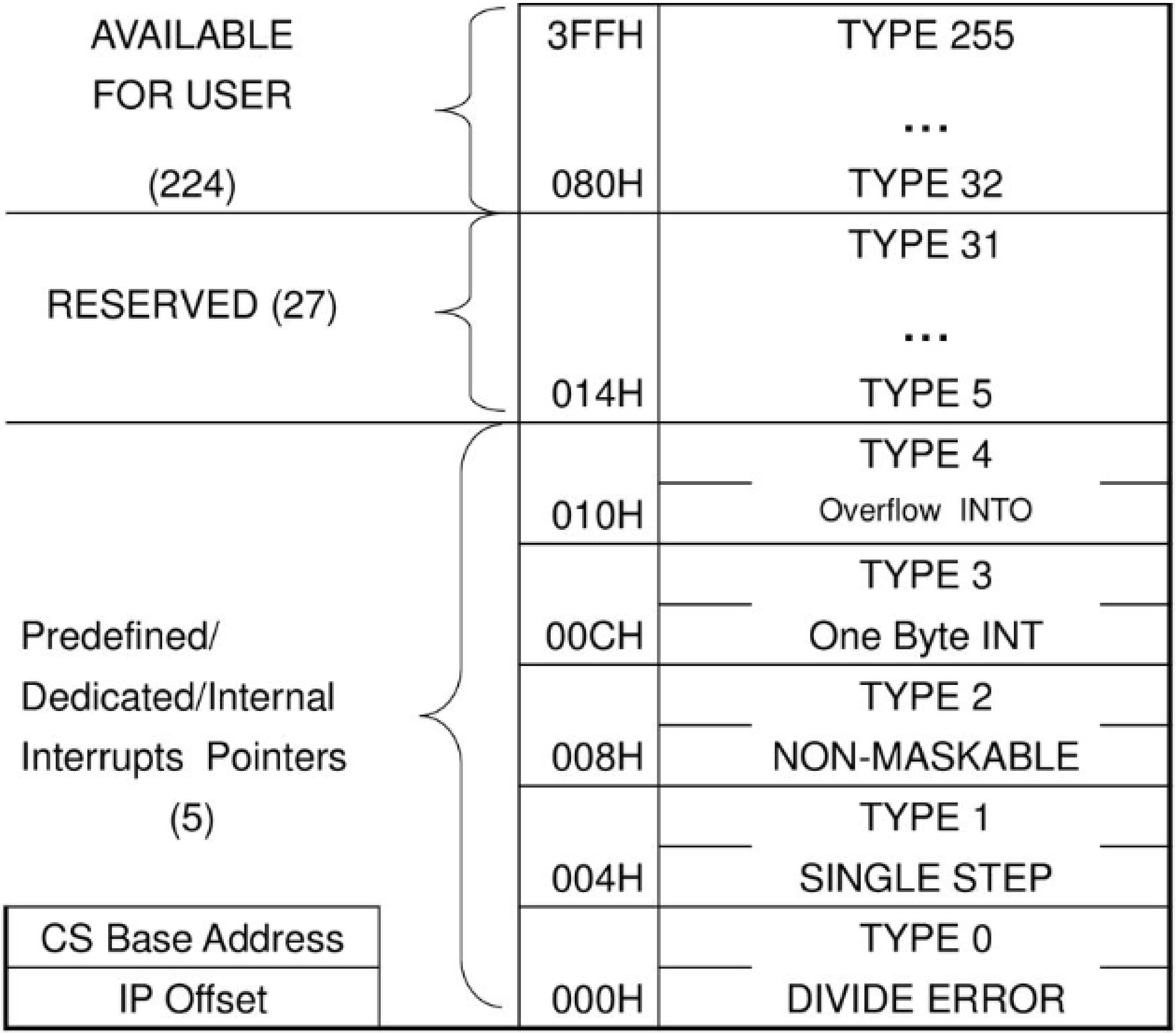
- The 256 interrupt vectors are arranged in the table in memory as follows in the next slide.

Note :

- The IP value is put in as the lower word of the vector and CS as higher word of the vector

- Each double word interrupt vector is identified by a number from 0 to 255 (00h to FFh)

- INTEL calls this number as the TYPE of the interrupt


AVAILABLE FOR USER (224)
RESERVED (27)
Predefined/ Dedicated/Internal Interrupts Pointers (5)
CS Base Address
IP Offset
3FFH
080H
014H
010H
00CH
008H
004H
000H
TYPE 255
...
TYPE 32
TYPE 31
...
TYPE 5
TYPE 4
Overflow INTO
TYPE 3
One Byte INT
TYPE 2
NON-MASKABLE
TYPE 1
SINGLE STEP
TYPE 0
DIVIDE ERROR

WHAT HAPPENS IF TWO OR MORE INTERRUPTS OCCUR AT THE SAME TIME?

Higher priority interrupts will be served first

PRIORITIES OF 8086 INTERRUPTS

Interrupt Type	Priority
DIVIDE ERROR, INT N, INT0	HIGHEST
NMI	
INTR	
SINGLE STEP	LOWEST

DIFFERENCES BETWEEN CALL AND INT:

S.No	CALL Instruction	INTn instruction
1	Upon the execution ,the control will jump to any one of the 1 MB of memory locations .	Upon execution the control will jump to a fixed location in the vector table.
2	The user can insert in the sequence of instructions of a program	Can occur at any time activated by hardware
3	Once initiated it cannot be masked	Can be masked
4	When initiated ,it stores the CS:IP of the next instruction on the stack	When initiated ,it stores the CS:IP of the next instruction and also the flag register on the stack.
5	The last instruction of the subroutine will be RET	The last instruction of the ISS will be IRET

THE INTERRUPT SEQUENCE OF 8086 MICROPROCESSOR:

1. External interface sends an interrupt signal, to the Interrupt Request (INTR) pin, or an internal interrupt occurs.
2. The CPU finishes the present instruction (for a hardware interrupt) and sends Interrupt Acknowledge (INTA) to hardware interface.
3. The interrupt type N is sent to the Central Processor Unit (CPU) via the Data bus from the hardware interface.
4. The contents of the flag registers are pushed onto the stack.
5. Both the interrupt (IF) and (TF) flags are cleared. This disables the INTR pin and the trap or single-step feature.

6. **The contents of the code segment register (CS) are pushed onto the Stack.**
7. **The contents of the instruction pointer (IP) are pushed onto the Stack.**
8. **The ISR address from the interrupt vector table (IVT) is fetched, by finding (4 x N). Where N is the 'type' of interrupt.**
9. **In the IVT, the first 2 bytes starting from 4 x N is placed into the IP and the next 2 bytes from (4xN +2) is placed into the CS, so that the next instruction is executed from the ISR addressed by the interrupt vector.**
10. **While returning from the ISR by the Interrupt Return (IRET) instruction, the IP, CS and Flag (PSW) registers are popped from the stack and returns to the state prior**

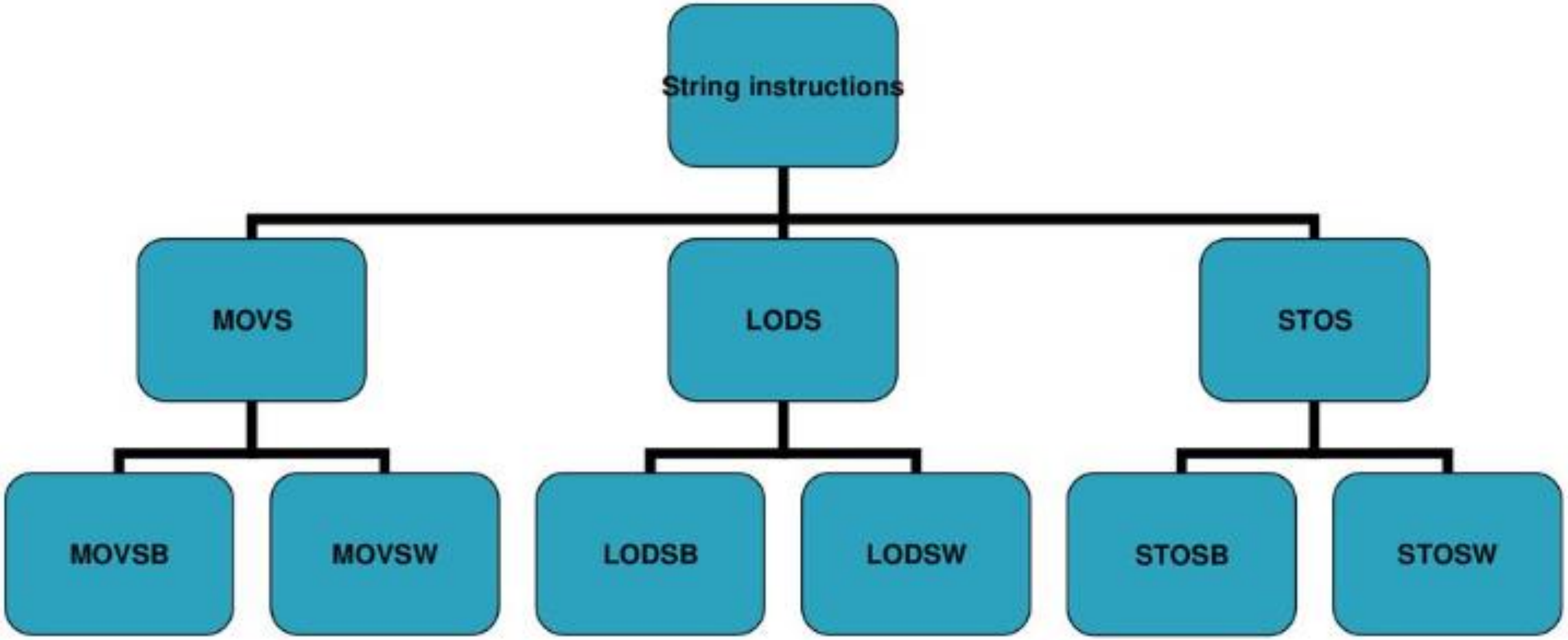
BYTE AND STRING PROGRAMMING

STRINGS

- A string is a series of bytes or words stored in successive memory locations

- 8086 can perform the following operations on strings
  - Moving a string from one place in memory to another
  - Compare two strings
  - Search a string for a specified character

STRING INSTRUCTIONS



B stands for byte, and W stands for word

MOVING A STRING

MOVSB/ MOVSW Instruction

- Copies a byte or word from a location in the data segment (DS) to the location in the extra segment (ES)
- Offset of source in data segment must be in SI register
- Offset of destination in extra segment must be in DI register
- For multiple byte/word moves the count is stored in CX register

Role of Direction Flag in Moving a String

- DF = 0
  - SI & DI will be incremented by 1 or 2 after every byte or word is moved
- DF = 1
  - SI & DI will be decremented by 1or 2 after every byte or word is moved

LEA, CLD & REP INSTRUCTIONS FOR MOVING A STRING

- **LEA** (Load Effective Address)
  - this instruction determines the offset of the variable or memory location named as the source and puts it in the specified 16-bit register
- **CLD** (Clear Direction Flag)
- **REP** (Repeat)
  - A prefix written before one of the string instruction
  - Causes string instruction to be repeated until CX=0

LODSB AND LODSW

For **LODSB** this is how it is done

- AL = DS:[SI]

- if D = 0 then SI = SI + 1

- else SI = SI – 1

- And **LODSW** is the same except that SI by 2

STOSB AND STOSW

For STOSB Store byte in AL into ES:[DI]. Update DI.

For STOSW Store word in AX into ES:[DI]. Update DI.

How STOSB executes?

- ES:[DI] = AL
  if DF = 0 then DI = DI + 1
- Else DI = DI - 1

- And STOSW executes the same as STOSB but DI is incremented or decremented by 2