Interrupt

- An **Interrupt** can be caused by:

1. An external **hardware** applying voltage to the **INTR** pin of the microprocessor, which indicates that the external device, such as printer or keyboard, requires service (hardware-based interrupt).
2. By executing the **INT xx** *instruction* (software-based interrupt).
3. Due to an **internal** interrupt (e.g., divide by zero).

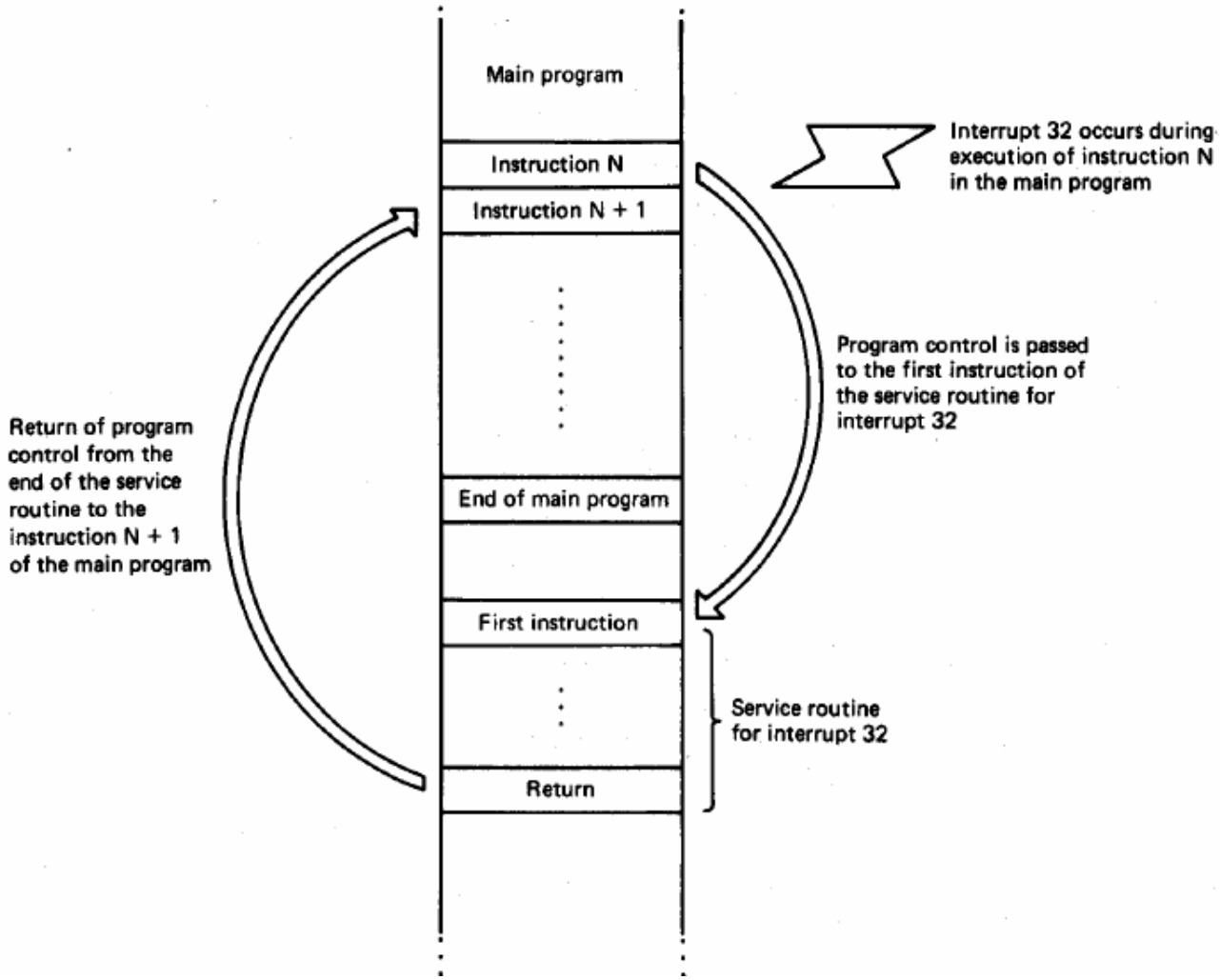

Figure -1- Interrupt program context switching mechanism.

- When an interrupt signal occurs, the MPU must **suspend** what it is doing in the main part of the program and **pass control** to a special routine (called the **interrupt-service routine**) that performs the function required by the external device.

- For example, when the printer causes an interrupt, the printer **driver** (interrupt service routine) is called, but how does the microprocessor know **where to jump**?

- The **8086** microprocessor is capable of implementing any combination of up to **256 interrupts**.

An **interrupt vector table** is stored in the first **1 kbyte** of memory (starting at address 00000h and ending at 003FFh). This is a pointer table to indicate the location of service routines corresponding to interrupt **types 0 to 255**.

- The CS and IP in the interrupt vector table indicate the location of the service routine for the corresponding interrupt.

- For example, when an interrupt of **type 32** occurs, the microprocessor looks at address 32 * 4 = 128 = **80h** and (after pushing the current CS and IP) loads the new values of CS and IP from 80h to jump to the interrupt service routine.

- The first 32 interrupt types have dedicated functions or are reserved (see Figure-2-).

- Interrupts are serviced on a **priority** basis. **Type 0** identifies the **highest priority** interrupt, and type 255 identifies the lowest-priority interrupt.

- If an interrupt-service routine has been initiated to perform a function assigned to a specific priority level, only interrupts with **higher priority** are allowed to interrupt the active service routine. Lower-priority devices have to **wait** until the current service routine is completed.

- If the IF flag is **set** (**IF = 1**), it **enables** the external interrupt request pin (**INTR**) input for operation (i.e., enables hardware interrupts). If **IF = 0** the INTR pin is disabled and external devices cannot interrupt the microprocessor operation.

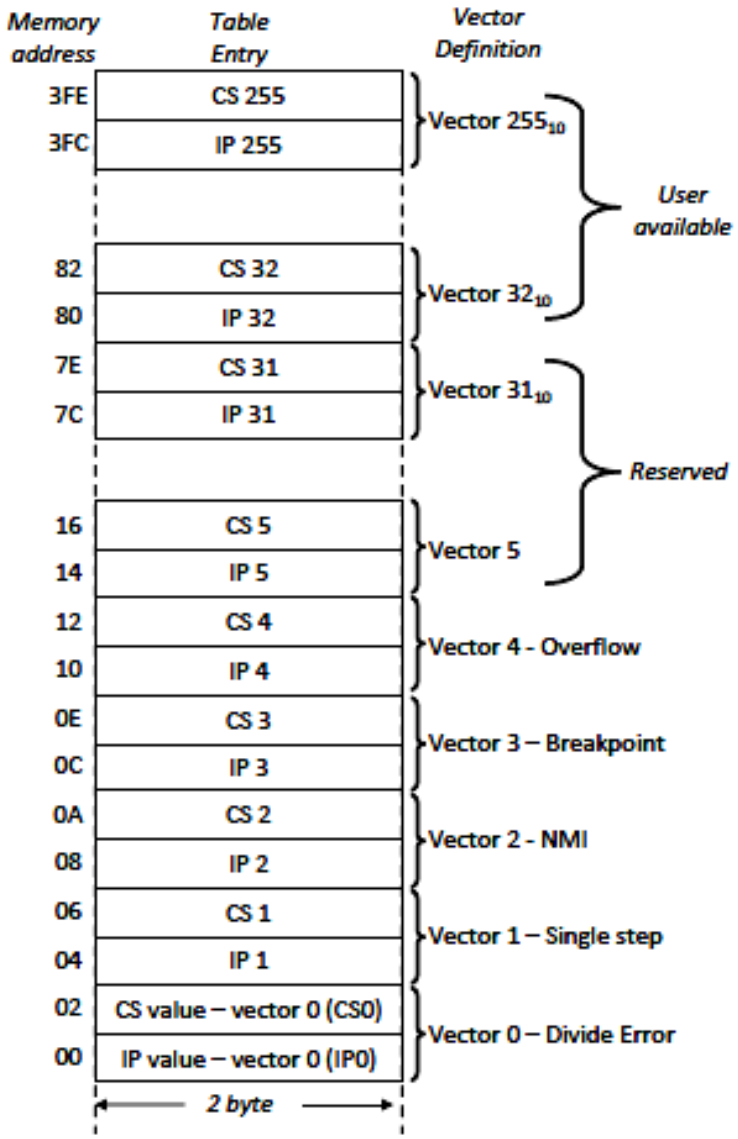

Figure -2- Interrupt vector table of the 8086 microprocessor

INTERRUPT INSTRUCTIONS

Mnemonic	Meaning	Format	Operation	Flags Affected
CLI	Clear interrupt flag	CLI	0 → (IF)	IF
STI	Set interrupt flag	STI	1 → (IF)	IF
INT n	Type n software interrupt	INT n	(Flags) → ((SP) − 2) 0 → TF,IF (CS) → ((SP) − 4) (2 + 4 · n) → (CS) (IP) → ((SP) − 6) (4 · n) → (IP)	TF, IF
IRET	Interrupt return	IRET	((SP)) → (IP) ((SP) + 2) → (CS) ((SP) + 4) → (Flags) (SP) + 6 → (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	None

Figure -3- Interrupt instructions

A number of instructions are provided in the instruction set of the 8086 microprocessors for use with interrupt processing. Figure -3- lists these instructions, with brief descriptions of their functions. For instance, the first two instructions, STI and CLI, permit manipulation of the interrupt flag through software. STI stands for set interrupt enable flag. Execution of this instruction enables the external interrupt request (INTR) input for operation—that is it Sets interrupt flag (IF). On the other hand, execution of CLI (clear interrupt enable flag) disables the external interrupt input by resetting IF.

The next instruction listed in Fig.-3- is the software-interrupt instruction **INT n**. It is used to initiate a vectored call of a subroutine. Executing the instruction causes

program control to be transferred to the subroutine pointed to by the vector for the number **n** specified in the instruction.

For example, execution of the instruction INT 50 initiates execution of a subroutine whose starting point is identified by vector 50 in the pointer table of Fig. -2-. First, the MPU saves the old flags on the stack, clears TF and IF, and saves the old program context, CS and IP on the stack. Then it reads the values of IP_{50} and CS_{50} from addresses-000C8 H and 000CA H, respectively, in memory, loads them into the IP and CS registers, calculates the physical address CS_{50}:IP_{50}, and starts to fetch instruction from this new location in program memory.

An interrupt-return (**IRET**) instruction must be included at the end of each interrupt-service routine. It is required to pass control back to the point in the program where execution was terminated due to the occurrence of the interrupt. As shown in Fig.-3-, when executed, IRET causes the old values of IP, CS, and flags to be popped from the stack back into the internal registers of the MPU. This restores the original program environment.

INTO is the interrupt-on-overflow instruction. This instruction must be included after arithmetic instructions that can result in an overflow condition, such as divide. It test the overflow flag, and if the flag is found to be set, a type 4 internal interrupt is initiated. this condition causes program control to be passed to an overflow service routine located at the starting address identified by the vector IP_4 at 00010H and CS_4 at 00012 H of the pointer table in Fig.-2-

The last two instructions associated with the interrupt interface are halt (**HLT**) and wait (**WAIT**). They produce similar responses by the 8086 and permit the operation of the MPU to be synchronized to an event in external hardware.

For instance, when HLT is executed, the MPU suspends operation and enters the idle state. It no longer executes instructions; instead, it remains idle waiting for the occurrence of an external hardware interrupt or reset interrupt. With the occurrence of either of these events, the MPU resumes execution with the corresponding service routine.

If the WAIT instruction is used instead of the HLT instruction, the MPU checks the logic level of the $\overline{TEST}$ input prior to going into the idle state. Only if $\overline{TEST}$ is at logic 1 will the MPU go into the idle state. While in the idle state, the MPU continues to check the logic level at $\overline{TEST}$, looking for its transition to the 0 logic level. As $\overline{TEST}$ switches to 0, execution resumes with the next sequential instruction in the program.

Ex: At what address are CS_{50} and IP_{50} stored in memory?

Solution:

Address= 4 × 50 = 200

and expressing it as a hexadecimal number results in

Address= C8 H

Therefore, IP_{50} is stored at 000C8H and CS_{50} at 000CAH.