**8086 Interrupts**

Table of Contents

[Classifications 3](#_Toc83227053)

[Interrupt Service Routines 5](#_Toc83227054)

[Interruption Process 6](#_Toc83227055)

[Interrupt Vectors 8](#_Toc83227056)

[Interrupt Types 9](#_Toc83227057)

An **interrupt** is a process by which a normal program execution is interrupted by some **external signal** or by a **special instruction** in the program. The microprocessor stops its current execution to deal with the interrupt.

The microprocessor needs to interact with external devices such as I/O devices and memory. If we had to deal with just one external device, we would never need interrupts. However, this is impractical. In reality, we need to deal with **multiple devices**.

From this comes the second option. We could ask each device in turn if they require our services, a process called **polling**. This works, but it is **slow** and wastes too much time in the process of asking and waiting for a response. Additionally, we will frequently find that a device did not even have a task when asked, which means even more wasted time.

Instead, we use **interrupts**. The external device contacts the microprocessor when it needs its services. This is particularly useful because I/O devices tend to be significantly slower than the microprocessor.

## Classifications

In the Intel 8086, interrupts can be of three types:

1. **Hardware Interrupt** – These interrupts come from **external signals** and are applied to the or the pins. These interrupts are **user-defined** since they occur due to user actions. For example, if we connect a USB device, drivers need to be installed, which causes an interrupt.
2. **Software Interrupt** – These are interrupts that are caused by the **interrupt instruction**, . These are also **user-defined**.
3. **Pre-Defined Interrupts** – These are interrupts caused by errors while executing instructions, such as trying to divide a number by .

**Hardware interrupts** can in turn be classified into two groups:

1. **Maskable Interrupts** – These are interrupts that can be delayed or rejected. They are user-defined interrupts and come through the pin.
2. **Non-Maskable Interrupts** – These are interrupts that cannot be delayed or rejected. They are system interrupts or major system faults and come through the pin.

Note that only interrupt requests arriving at are given an acknowledgement, through the pin . Interrupts that arrive at the pin are not given an acknowledgement.

Interrupts are prioritized as:

1. Reset
2. Internal interrupts and exceptions (e.g. divide by )
3. Software interrupts (INT n, INTO)
4. Non-maskable interrupts (NMI)
5. External hardware interrupt (INTR/INTA)
6. Single-Step Debugging

The priority order is maintained if multiple interrupts occur **simultaneously**, not while an interrupt is already ongoing. An ongoing interrupt cannot be interrupted.

The priority is handled by the **Programmable Interrupt Controller**.

## Interrupt Service Routines

An interrupt is an **emergency signal** that needs to be serviced. The microprocessor may respond to it as soon as possible.

When the microprocessor receives an interrupt signal, it suspends the currently executing program and jumps to an **interrupt service routine** (ISR). The ISR responds to the incoming interrupt. Each interrupt has its own ISR.

The ISR is not a user-defined program. It is a **system program**.

After the ISR has finished its work, the original program resumes execution from where it left off.

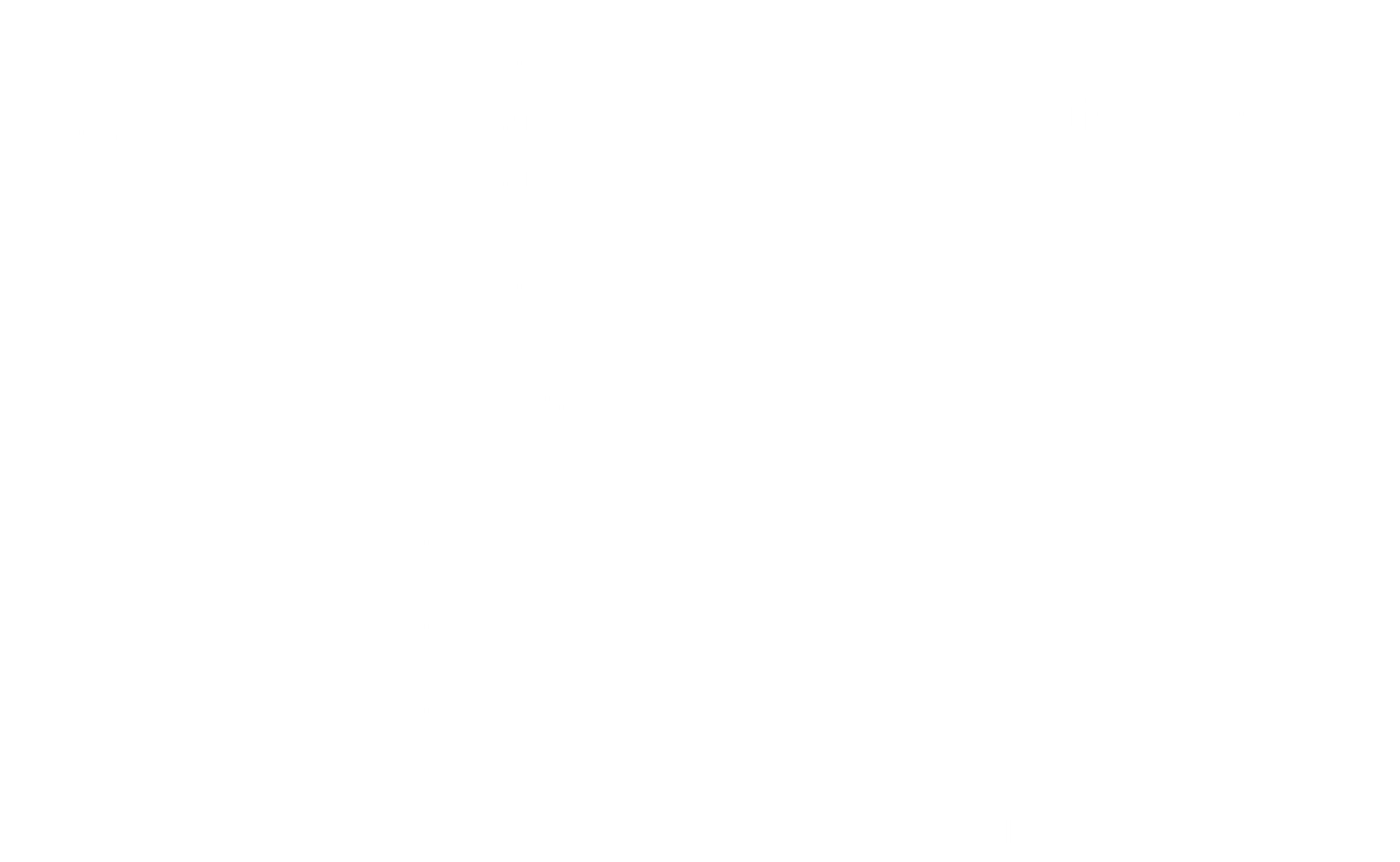
## Interruption Process

There are two **flag bits** associated with interrupts, (Interrupt Flag) and (Trap Flag). is set if the microprocessor is accepting interrupts. is set if we are in single-step debugging mode.

At the end of each **instruction cycle**, the 8086 checks to see if any interrupts have been requested. If it has, then the following actions take place:

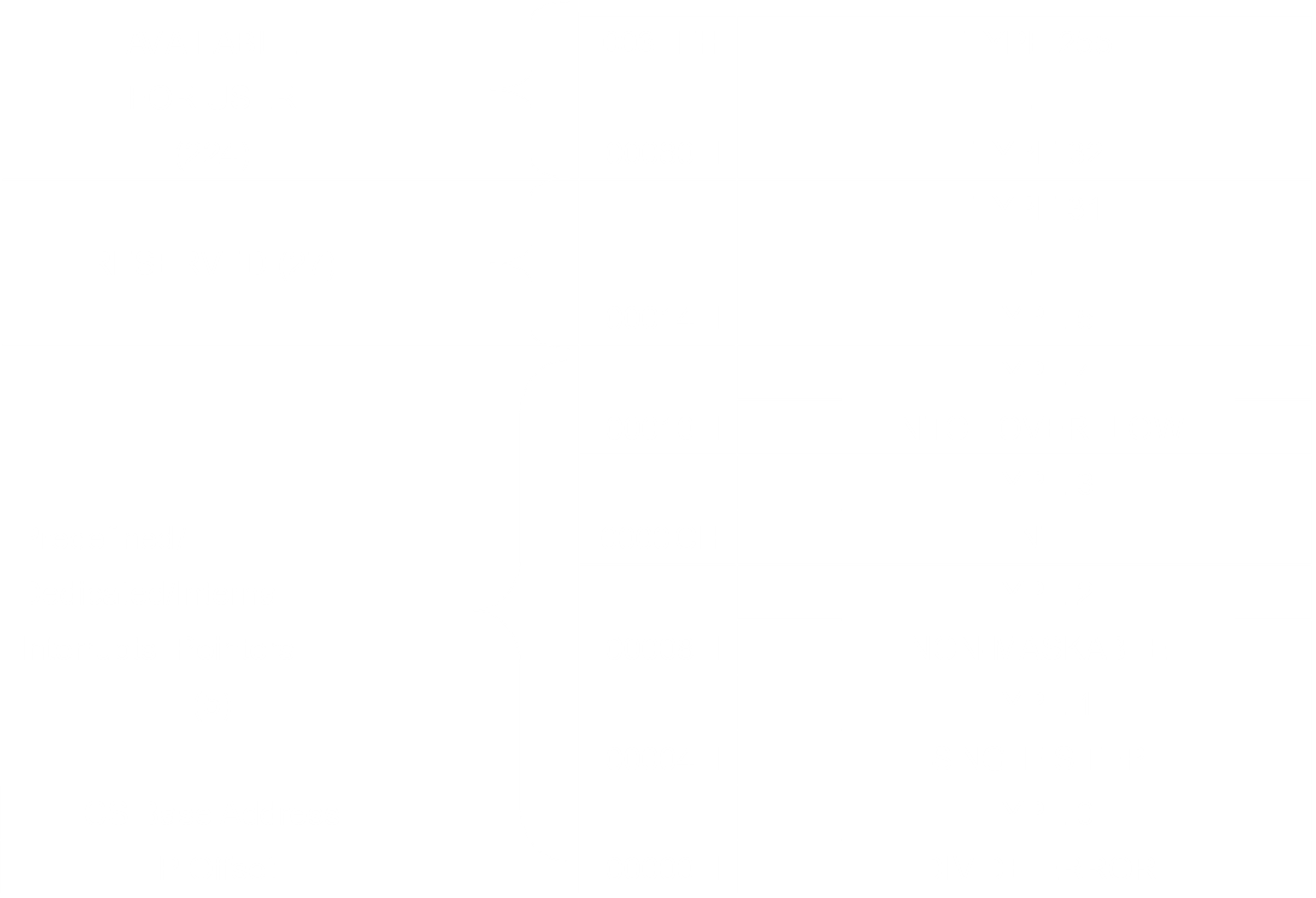
1. **is decremented** by and the **flag register** is pushed onto the stack.
2. **is disabled** by clearing in the flag register.
3. **is reset**, since the ISR is a system program and single-step debugging should not be allowed in it.
4. **is decremented** by again and the contents of are pushed onto the stack.
5. **is decremented** by again and the contents of are pushed onto the stack.
6. An **indirect jump** is made to the start of the **ISR** written to respond to the interrupt. The ISR has new values for , and the flags.

On the other side, once the ISR has finished execution, the contents, the contents and the flag register are **popped** from the stack one by one and then execution of the previous program resumes.



## Interrupt Vectors

An **Interrupt Vector** is a pointer to the start of an ISR in memory. All interrupts are mapped onto an interrupt vector using the **Interrupt Vector Table** (IVT). This is a table, located in the first of memory (from to ) whose sole purpose is to hold the Interrupt Vectors for different interrupts.



From the Interrupt Vector Table it can be seen that there are  **types** of interrupts, of which we know next to none. For each interrupt type, there is a single entry in the IVT. However, each entry takes up **four memory locations**. Since the table starts at , it ends at .

In a single entry, the first two bytes, the **low word**, hold the **offset address**, or the value, of the ISR, while the last two bytes, the **high word**, hold the **segment address**, or the value of the ISR.

Interrupt types through are **predefined interrupts**. These are the same in all Intel processors. Interrupt types through are **reserved interrupts** used by Intel for research purposes. Interrupt types through are made available to **users**.

### Interrupt Types

Each interrupt in the IVT can be identified by a number from to . Intel calls this number the **type**.

When the microprocessor gets an INTR signal and sends back an INTA signal, the interrupter releases the type number of the interrupt required by it. The microprocessor multiplies the type number by to produce the address required in the IVT, and then reads bytes of memory starting from that address.

The predefined interrupt types are:

* **Type 0 (Divide Error)** – A Divide Error interrupt occurs when the result of a division overflows or an attempt is made to divide by .
* **Type 1 (Trap)** – If the Trap Flag (TF) is set, the Trap interrupt occurs after the execution of each instruction, thus allowing single-step debugging. When this interrupt is accepted, TF is cleared.
* **Type 2 (NMI)** – When a logic appears on the NMI pin, an NMI interrupt occurs.
* **Type 3 (Breakpoint)** – This is a special software interrupt, called with the command INT 3. This interrupt is used to put a breakpoint in a program for debugging purposes. The code after the breakpoint is not executed.
* **Type 4 (Overflow)** – An overflow interrupt occurs due to the INTO command, which can be called if an overflow occurs, as reflected by the Overflow Flag (OF).