SNS COLLEGE OF TECHNOLOGY

An Autonomous Institution
Coimbatore-35

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A+' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

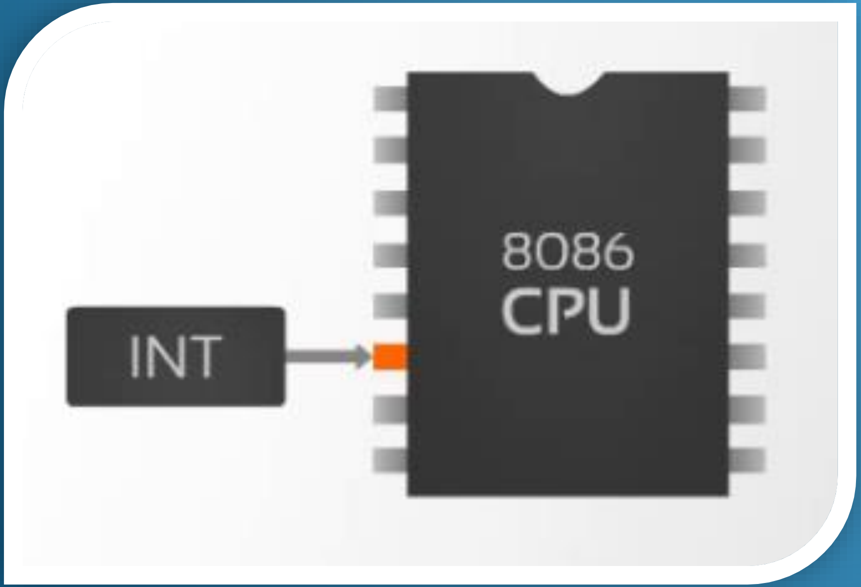
19ITT204 - MICROCONTROLLER AND EMBEDDED SYSTEMS

II YEAR/ IV SEMESTER

UNIT I ARCHITECTURE OF 8086 MICROPROCESSOR

TOPIC – INTERRUPTS

Interrupts on 8086 Microprocessor



- When your phone rings during a lecture, what will happen?
- When you are studying then your cell phone rings – what will you do?
- When you finish talking on the phone then you will continue with your study.
- Now your phone rings again and someone also knocking at your door then what will you do?
- When being interrupted, you will perform some pre-defined action.
- Interrupt has priority – some interrupt is more important than the others. For example, answering your phone is more important

Introduction

- An interrupt is used to cause a temporary halt in the execution of program.
- The meaning of ‘interrupts’ is to break the sequence of operation.
- **While the Microprocessor 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, IRET returns the control back again to the main program. Interrupt processing is an alternative to polling.

Need for Interrupt:

- **Interrupts are particularly useful when interfacing I/O devices, that provide or require data at relatively low data transfer rate.**

Sources of Interrupts:

Three types of interrupts sources are there:

1. An external signal applied to NMI or INTR input pin(hardware interrupt)
2. Execution of Interrupt instruction(software interrupt)
3. Interrupt raised due to some error condition produced in 8086 instruction execution process. (Divide by zero, overflow errors etc)

8086 Interrupt Sources

An 8086 interrupt can come from any one of the following three sources:

1. An external signal applied to the non-maskable interrupt (NMI 17 pin) pin or to the interrupt (INTR 18 pin) pin. An interrupt caused by a signal applied to one of these inputs is called *hardware interrupt*.
2. The execution of the instruction INT n, where n is the interrupt type that can take any value between 00H and FFH. This is called *software interrupt*.
3. An error condition such as divide-by-0, which is produced in the 8086 by the execution of the DIV/IDIV instruction or the trap interrupt.

How to get key typed in the keyboard or a keypad?

- **Polling :-**

The CPU executes a program that check for the available of data, If a key is pressed then read the data, otherwise keep waiting or looping!!!

- **Interrupt:-**

The CPU executes other program, as soon as a key is pressed, the Keyboard generates an interrupt. The CPU will response to the interrupt read the data. After that returns to the original program. So by proper use of interrupt, the CPU can serve many devices at the "same time"

Polling vs Interrupt

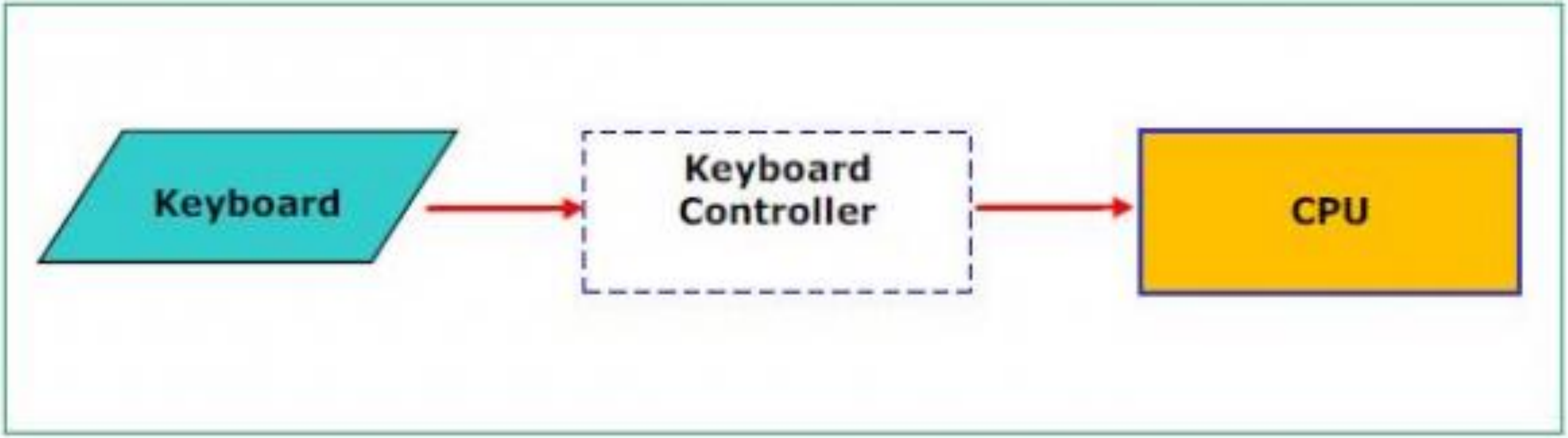
- ❑ The keyboard controller can hold only a single keystroke. Therefore, the keyboard controller must be freed before the next keystroke arrives.
- ❑ The keystroke is passed to the CPU by putting it in the keyboard buffer. So, the keyboard controller keeps on passing the keystroke input to the CPU,

But how does the CPU attend to it?

The CPU is not at the disposal of the keyboard controller; it is usually busy doing several other operations. So, we need some mechanism to indicate to the CPU that a keystroke has arrived. How is this done? There are two approaches to making sure that the CPU pays attention:

- ➢ Polling-based
- ➢ Interrupt-based

Example: Polling Vs Interrupt



Keystroke causes interrupt

Polling Based System:-

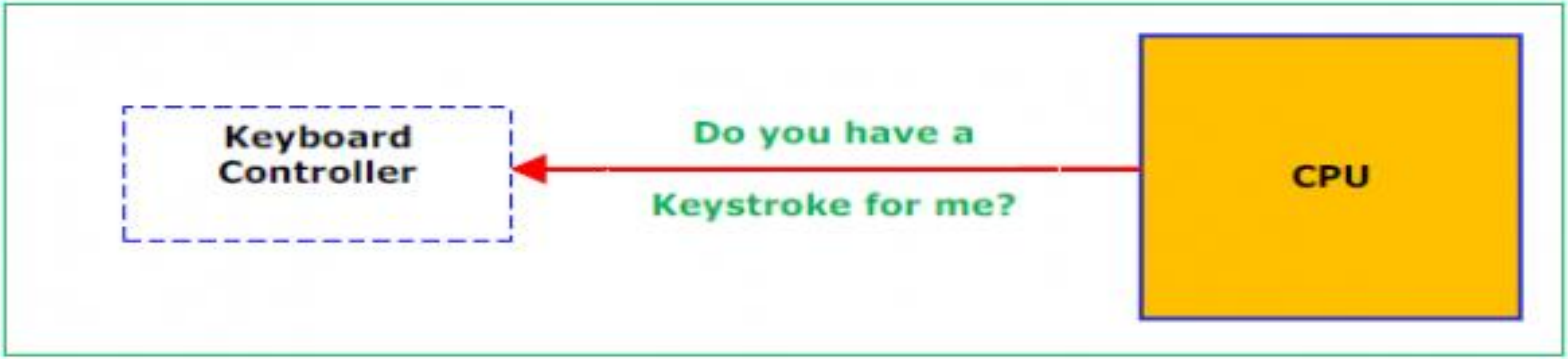
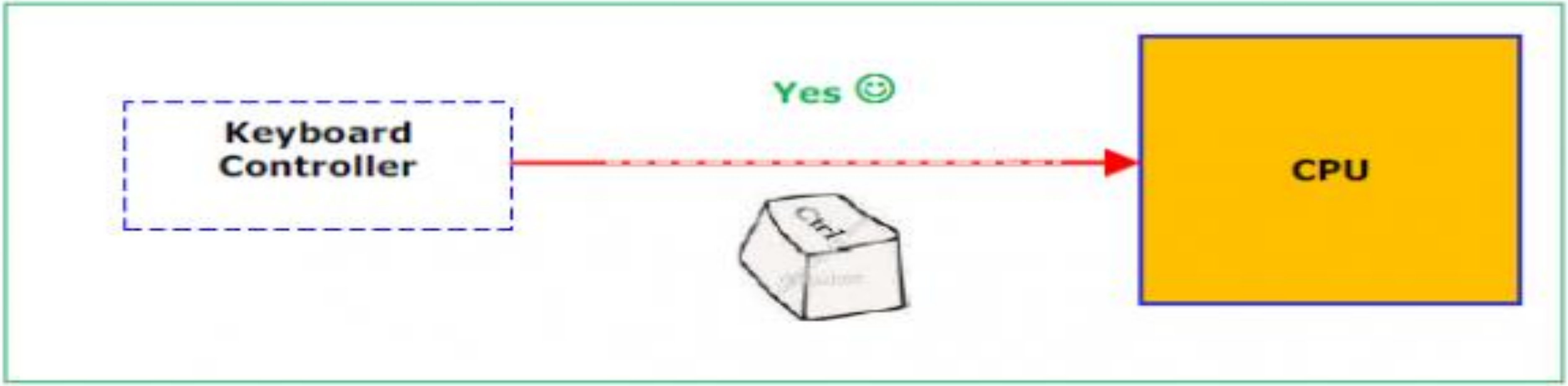
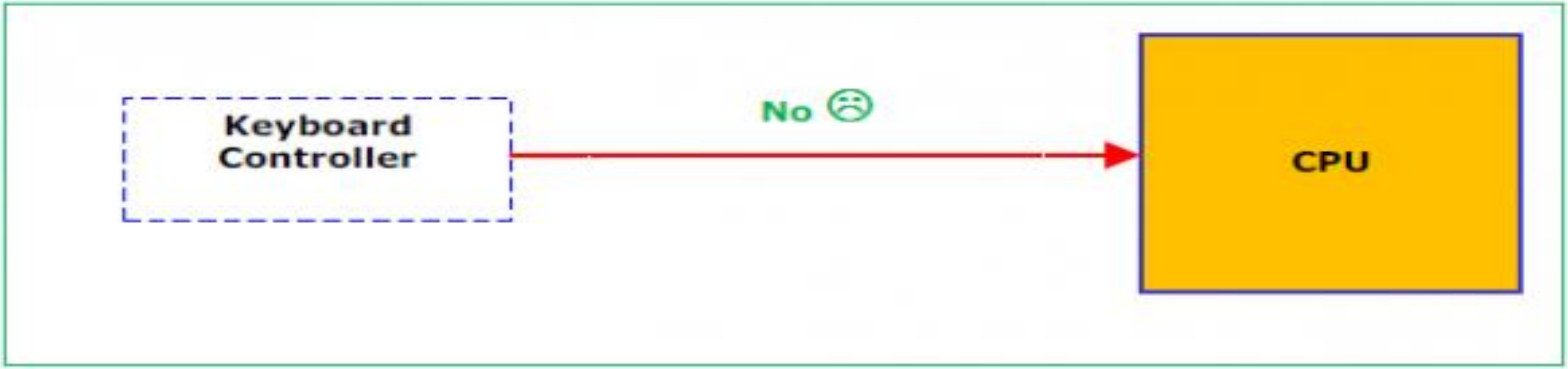


Figure 2: Polling-based interrupt handling

- The CPU executes a program that check for the available of data If a key is pressed then read the data, otherwise keep waiting or looping!!!

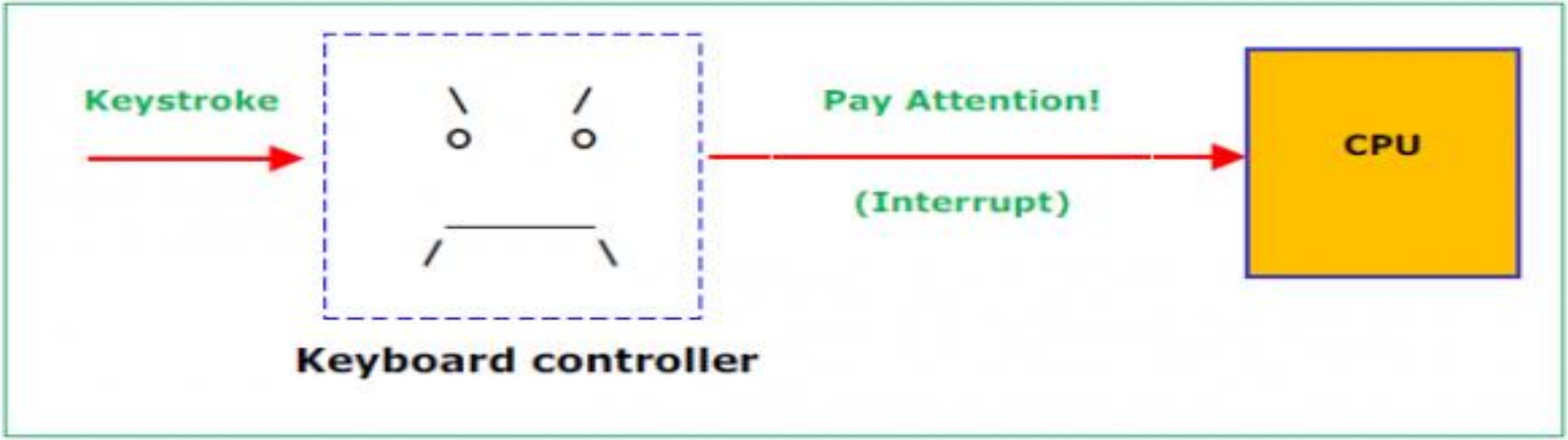


Keystroke passed to the CPU



No keystroke for CPU

Interrupt-based systems



Interrupt-based approach

- The CPU executes other program, as soon as a key is pressed, the Keyboard generates an interrupt. The CPU will response to the interrupt – read the data. After that returns to the original program. So by proper use of interrupt, the CPU can serve many devices at the "same time"

Example of interrupt

- How to control a robot that has sensors to detect obstacles and makes a turn

Controlling a robot by using Polling & Interrupt

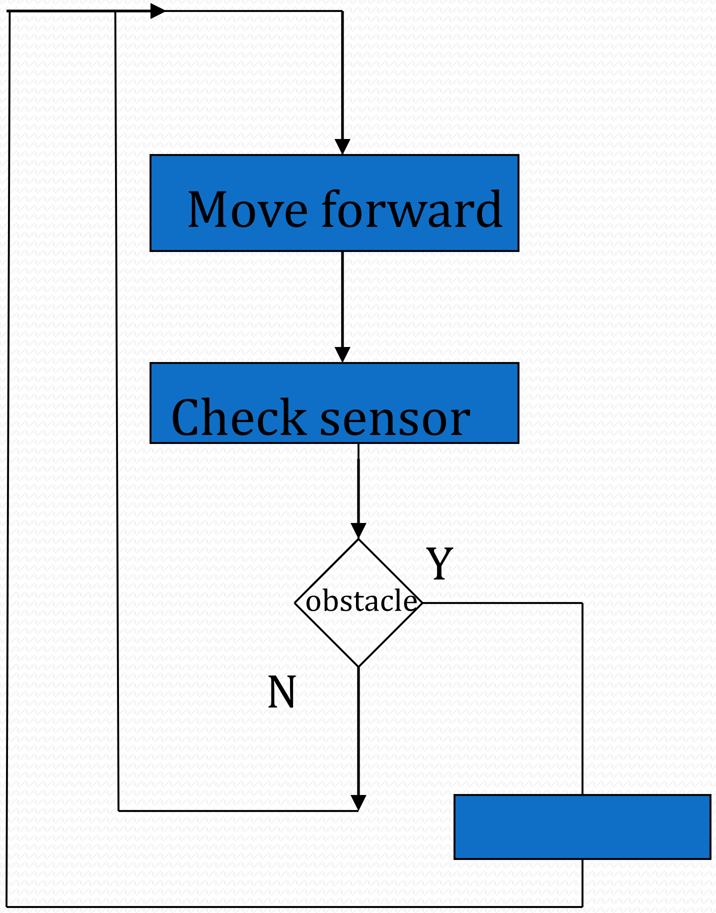
Polling

- Move forward in a pre-defined unit
- Check sensor reading
- Do nothing if no obstacle or turn if obstacle detected
- Loop back and move forward again

Interrupt

- Keeping moving until interrupted by the sensor
- Interrupt received then do pre-defined operation
- After finishing the interrupt service return to normal operation i.e keep moving forward again

Polling vs Interrupt Control of a robot



Move forward

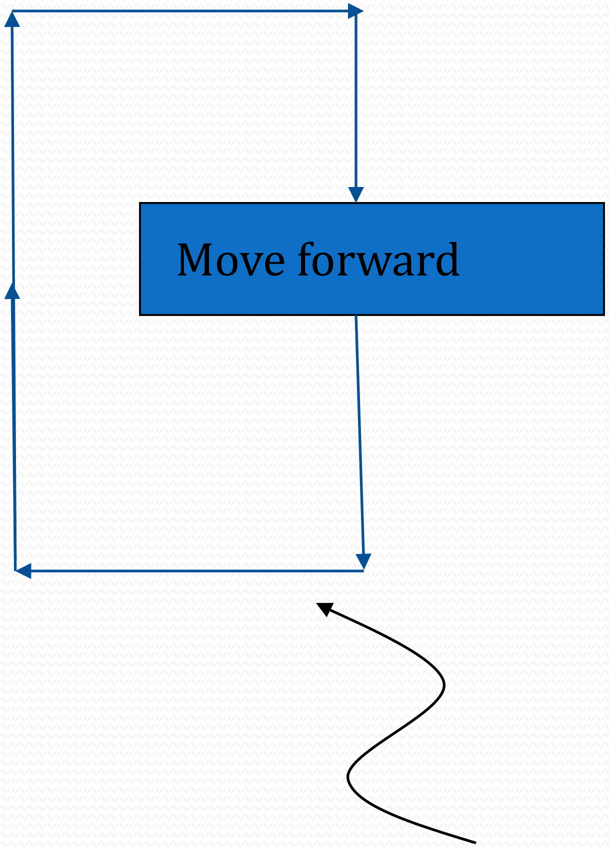
Check sensor

obstacle

Y

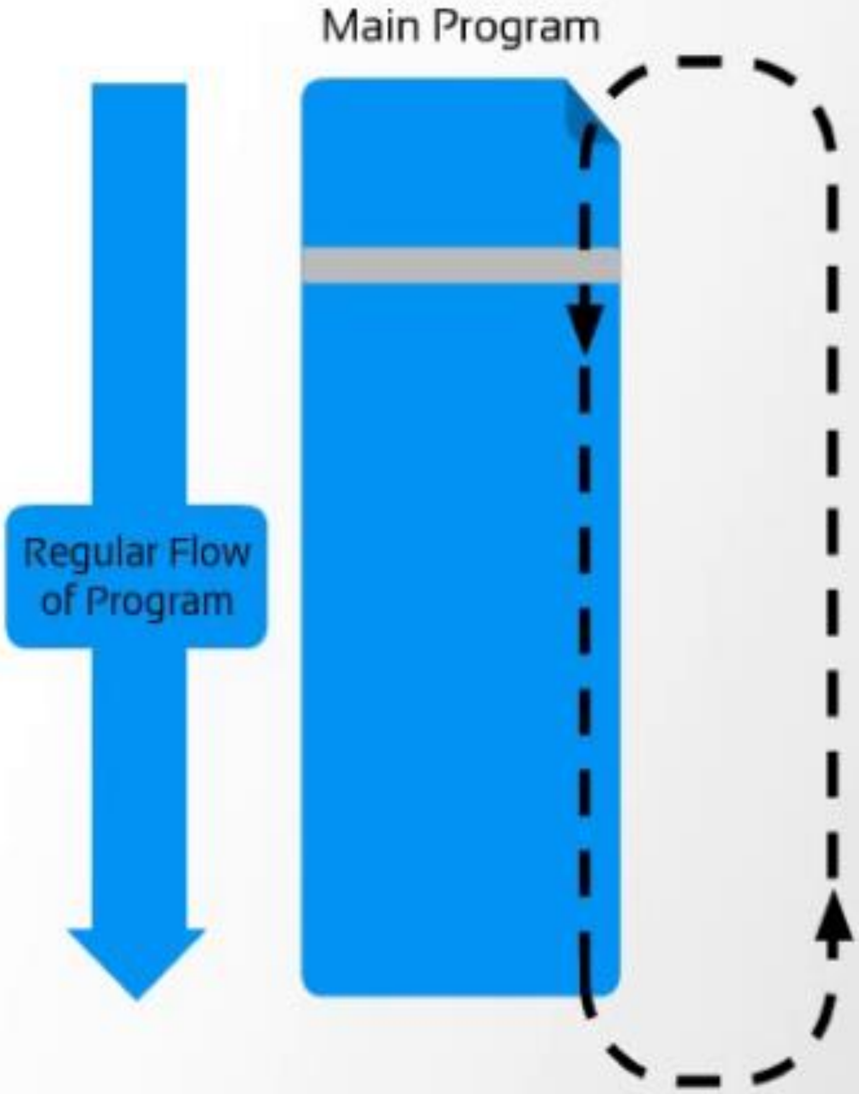
N

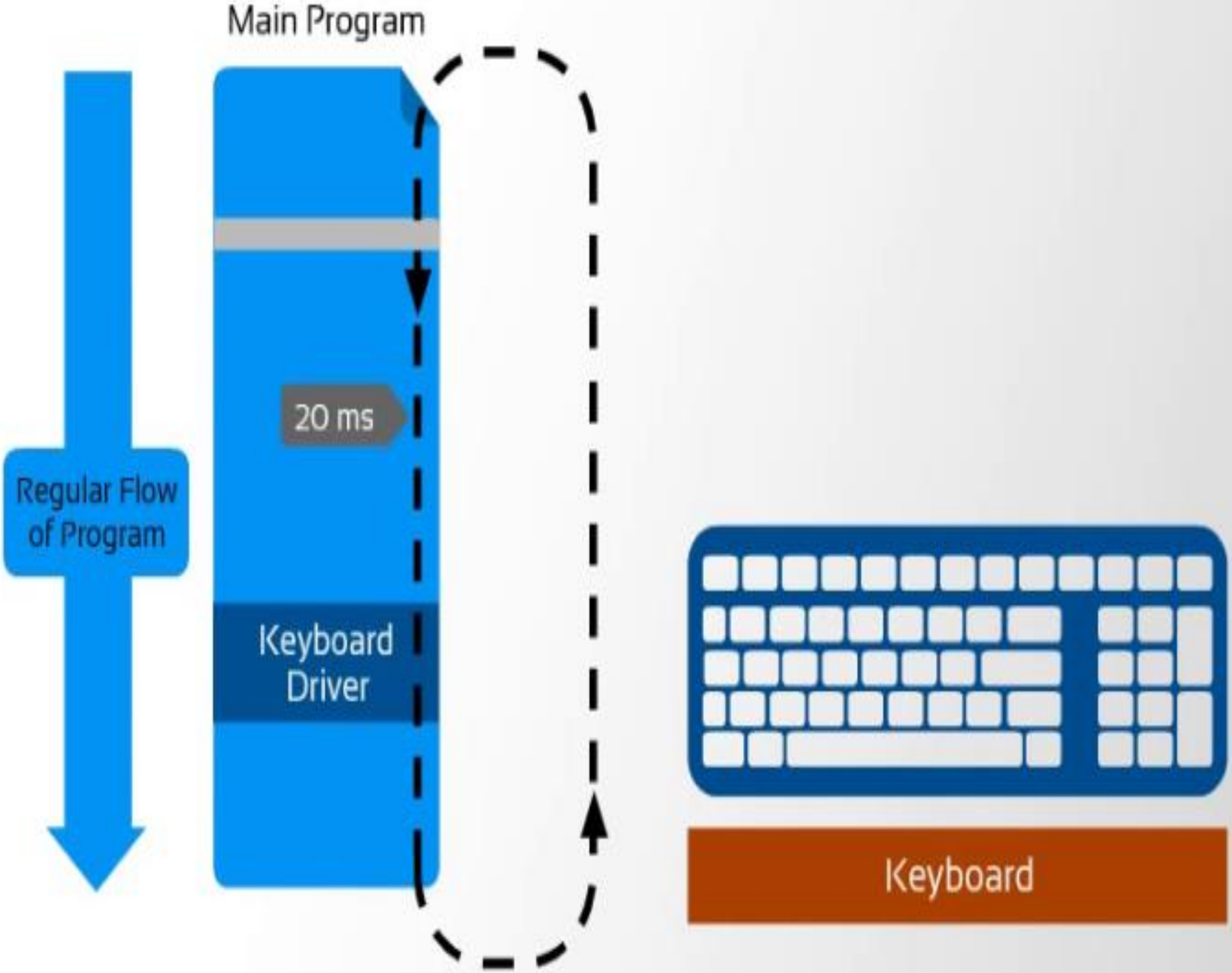
Stop or turn

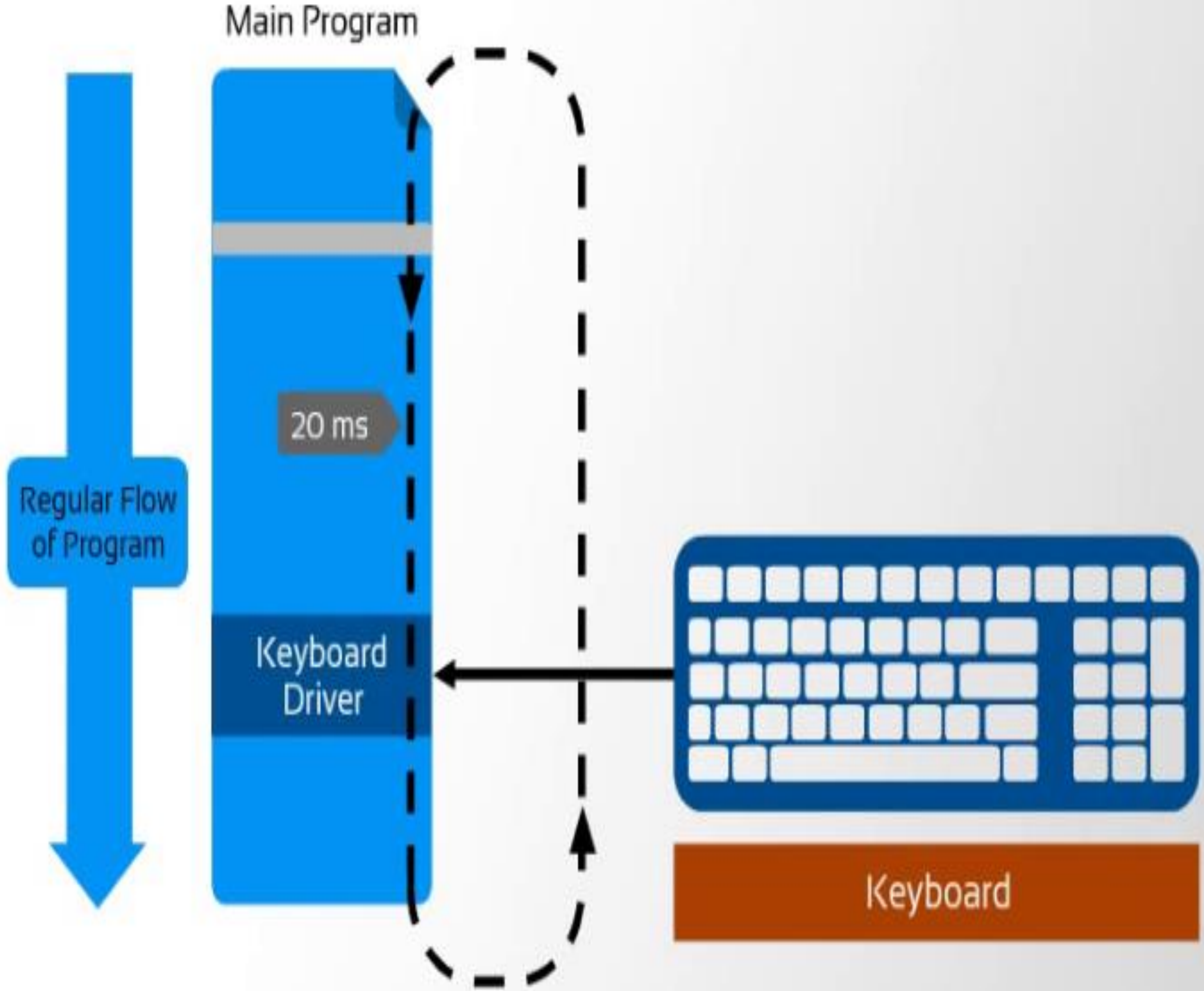


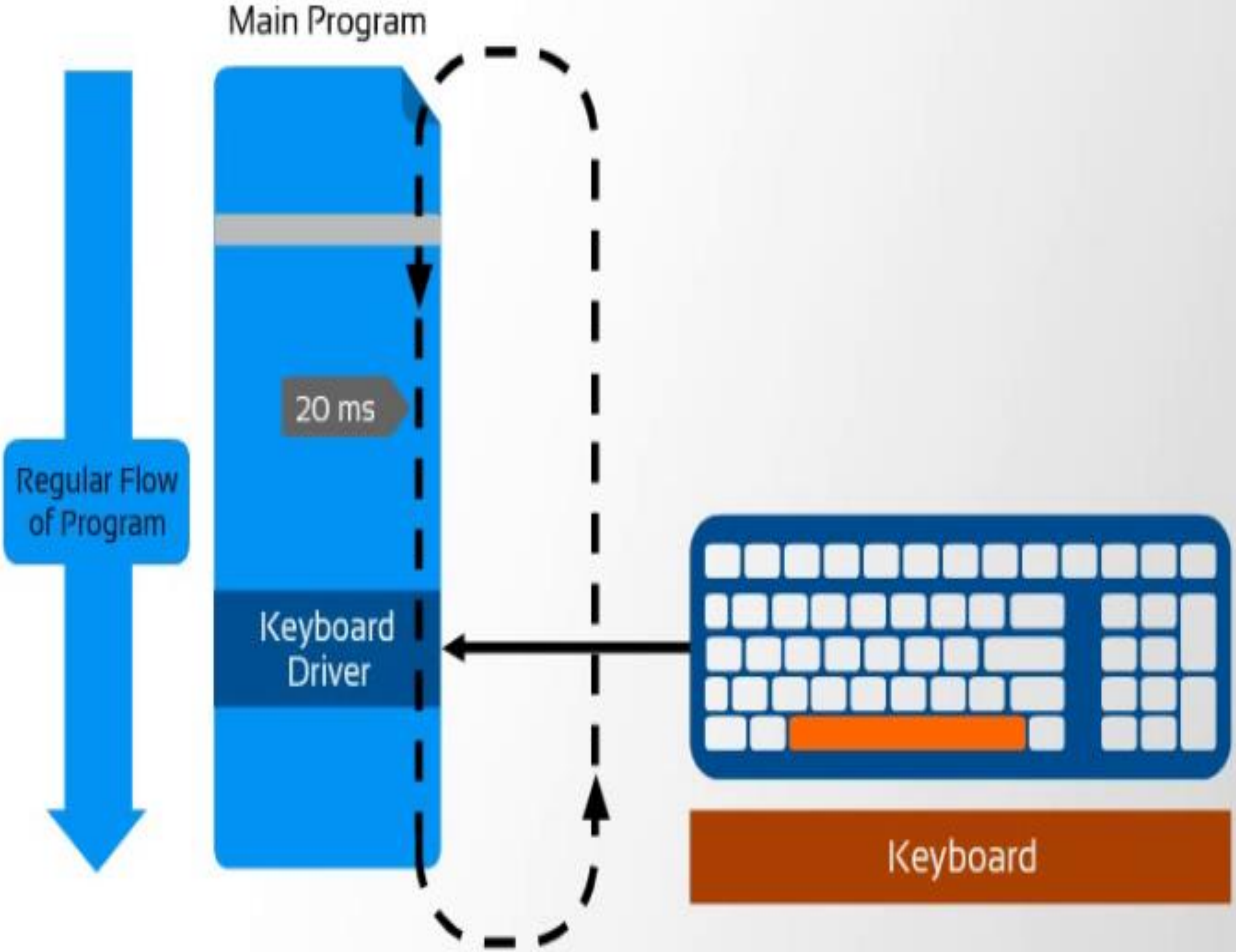
Move forward

interrupt

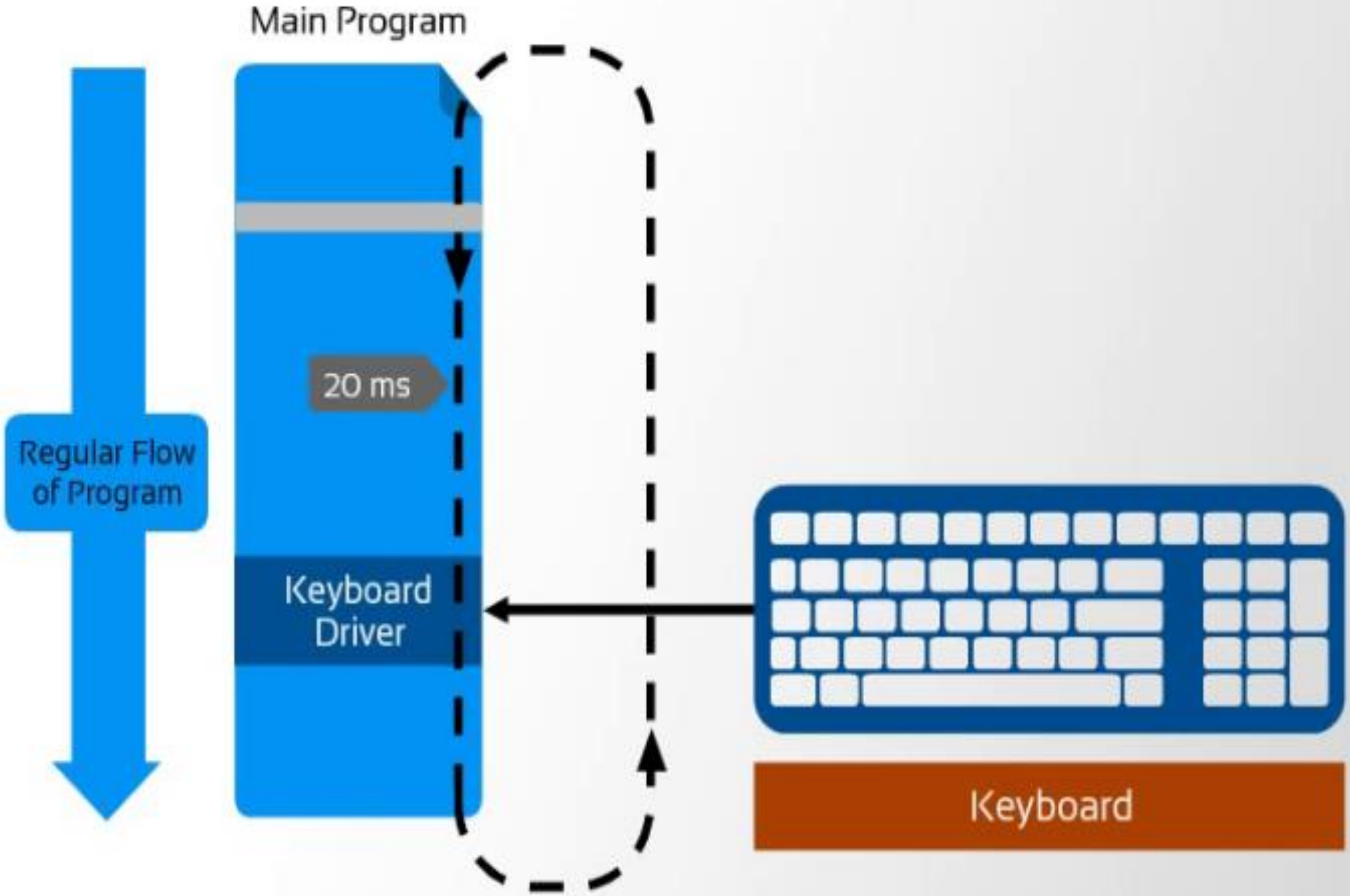

Main Program
Regular Flow of Program


Main Program
Regular Flow of Program
20 ms
Keyboard Driver
Keyboard

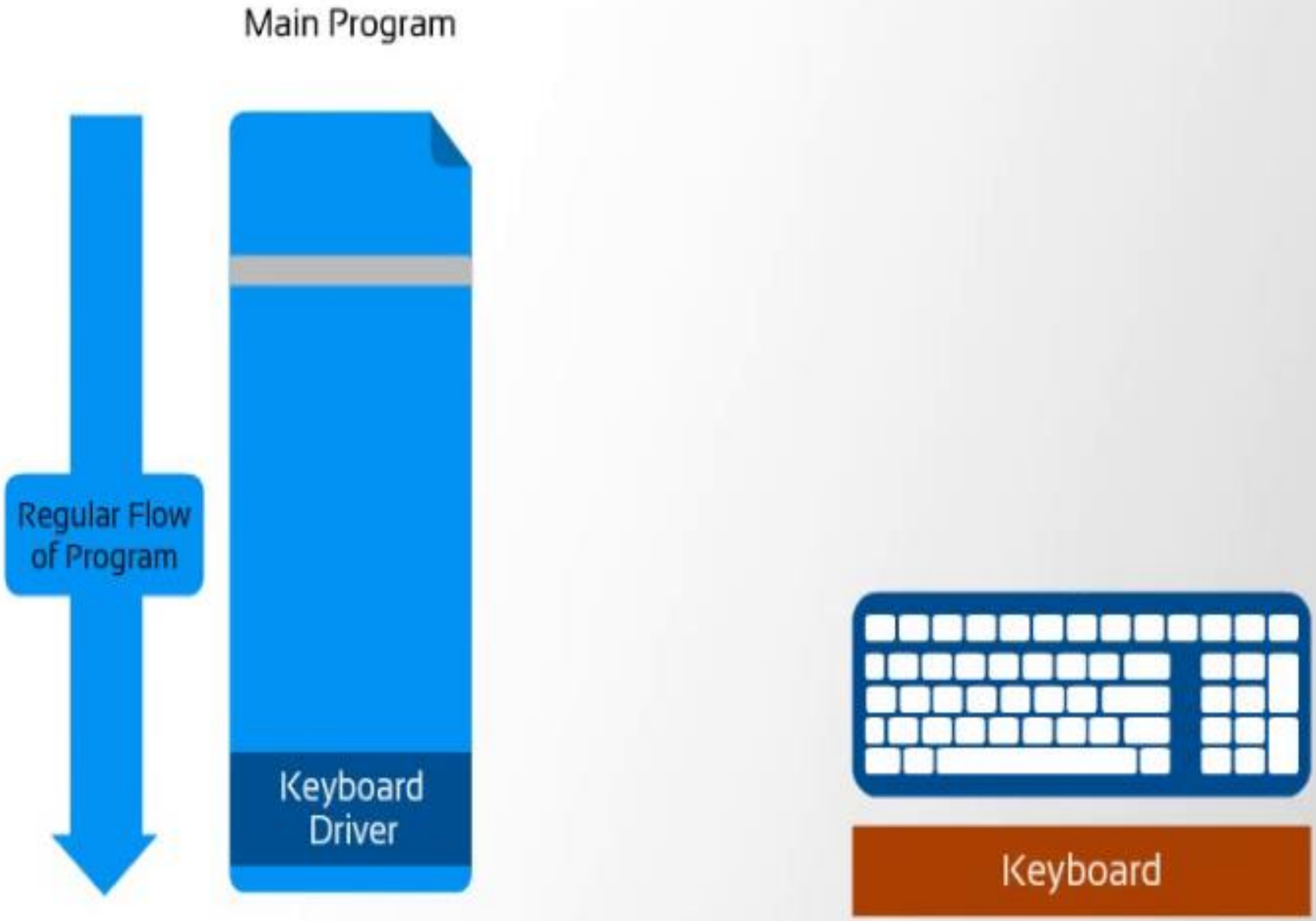

Main Program
20 ms
Regular Flow of Program
Keyboard Driver
Keyboard

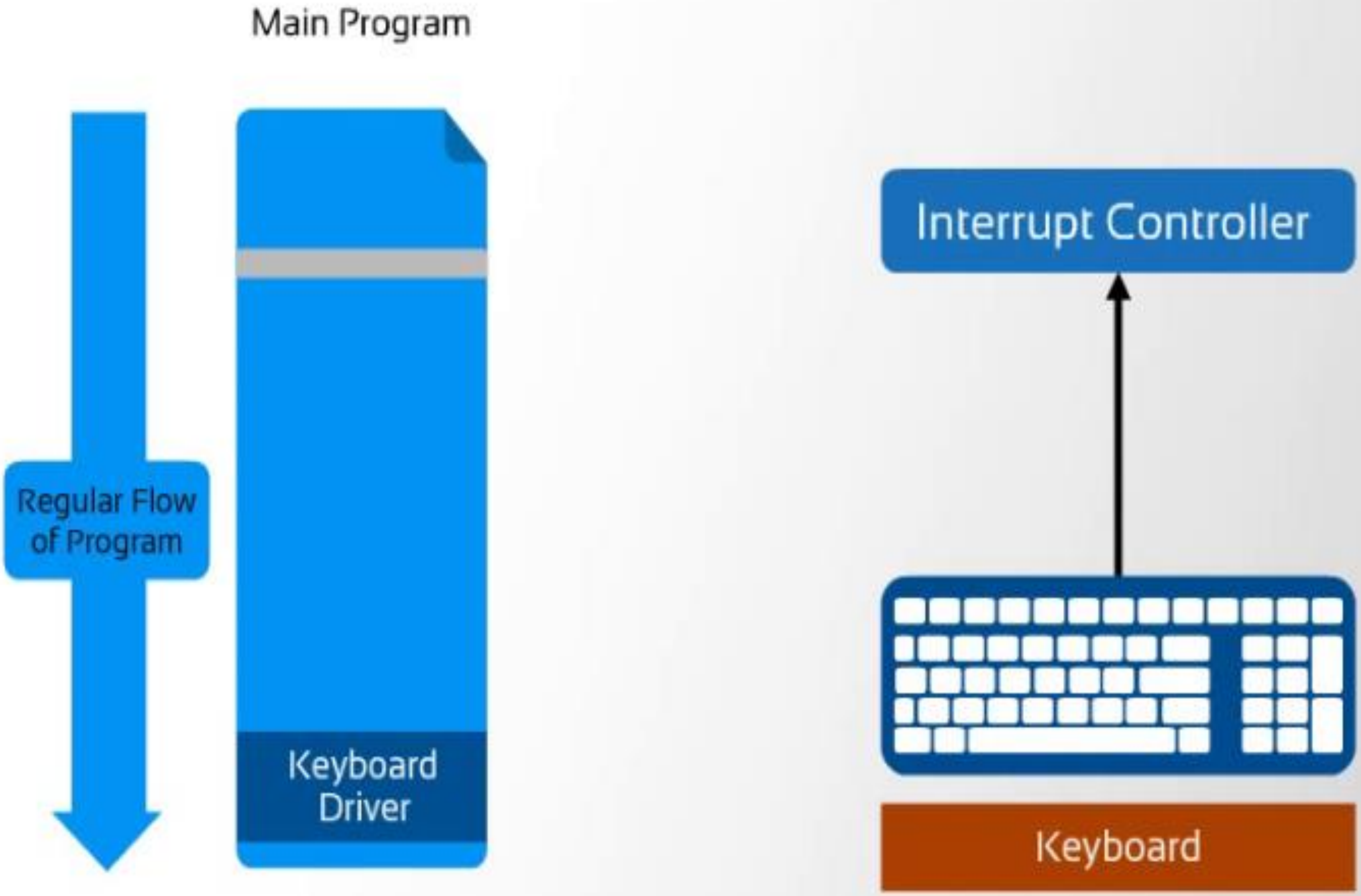

Main Program
Regular Flow of Program
20 ms
Keyboard Driver
Keyboard

This is called Polling.



How would the processor work with an Interrupt





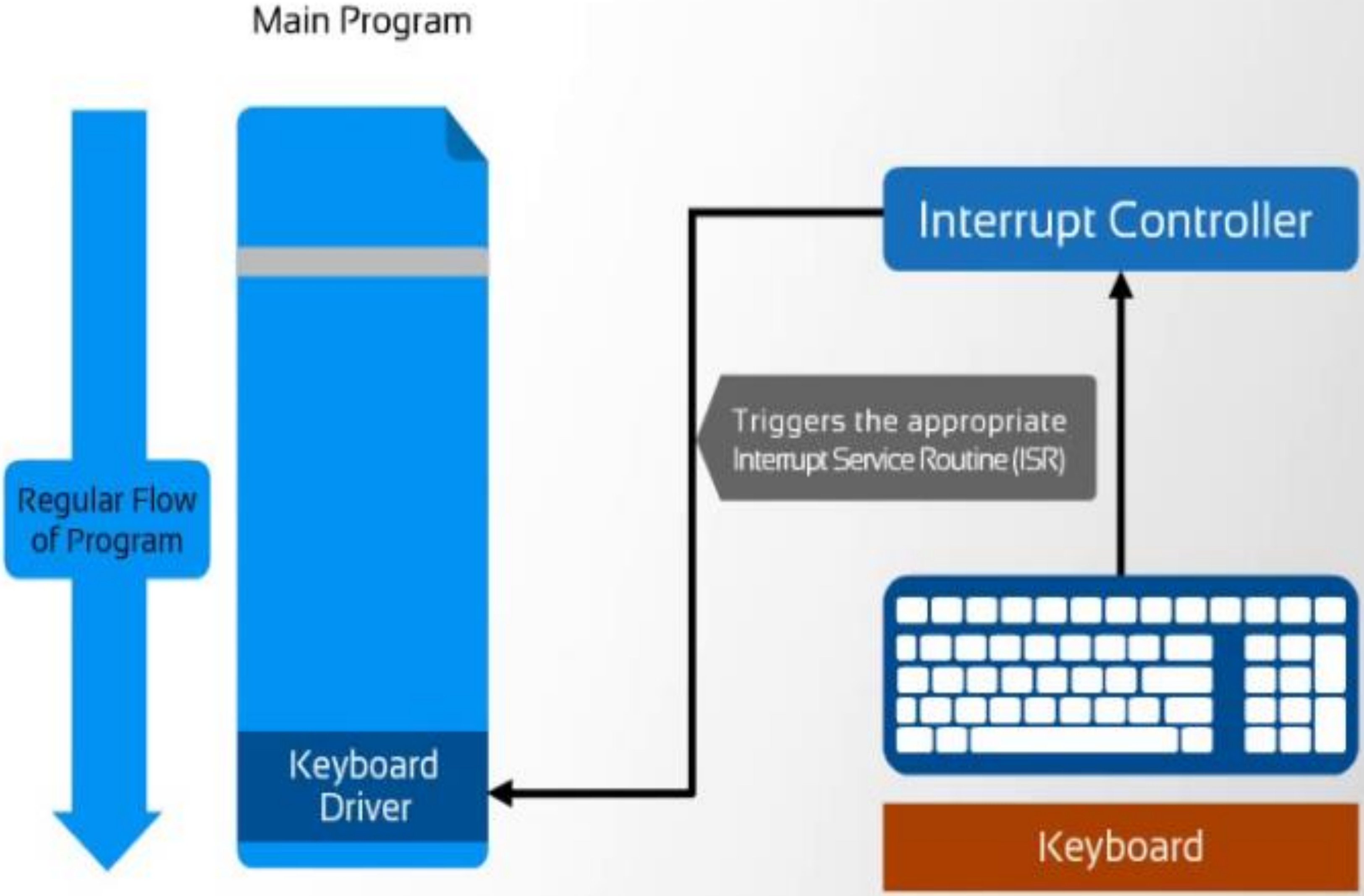
Main Program

Regular Flow of Program

Keyboard Driver

Interrupt Controller

Keyboard



Main Program

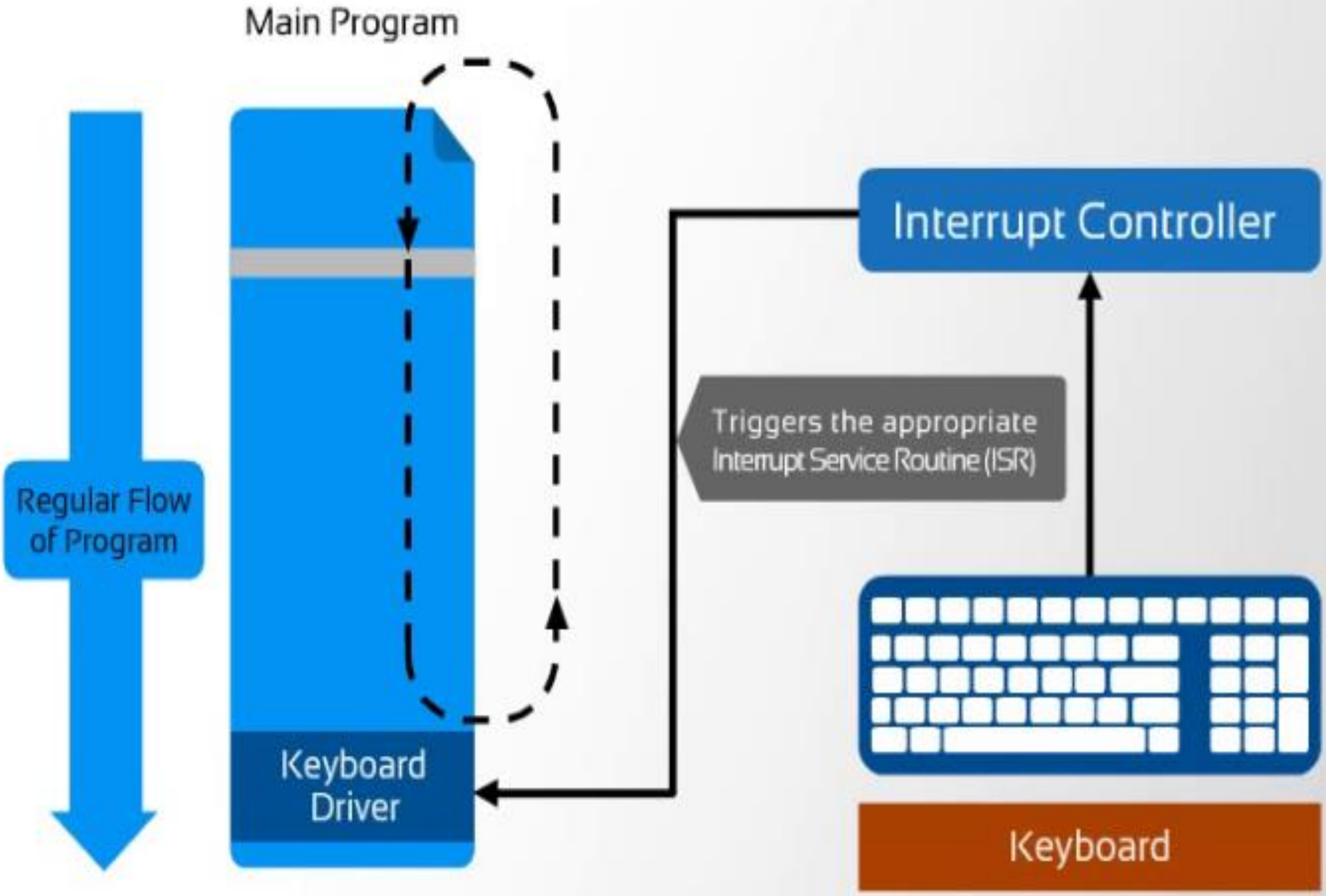
Regular Flow of Program

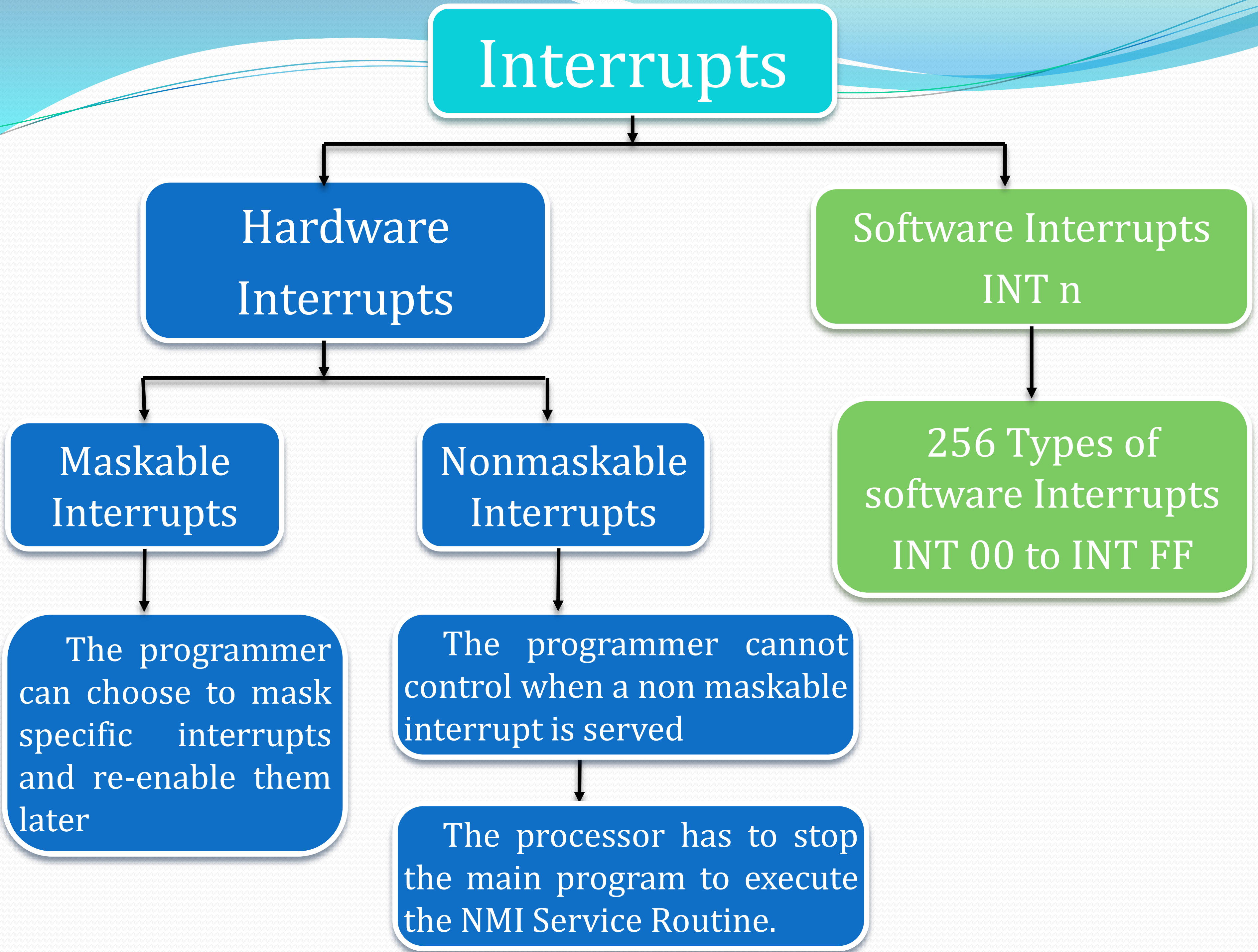
Keyboard Driver

Interrupt Controller

Triggers the appropriate Interrupt Service Routine (ISR)

Keyboard


Main Program
Regular Flow of Program
Keyboard Driver
Interrupt Controller
Triggers the appropriate Interrupt Service Routine (ISR)
Keyboard



Interrupts

Hardware Interrupts

Software Interrupts
INT n

Maskable Interrupts

Nonmaskable Interrupts

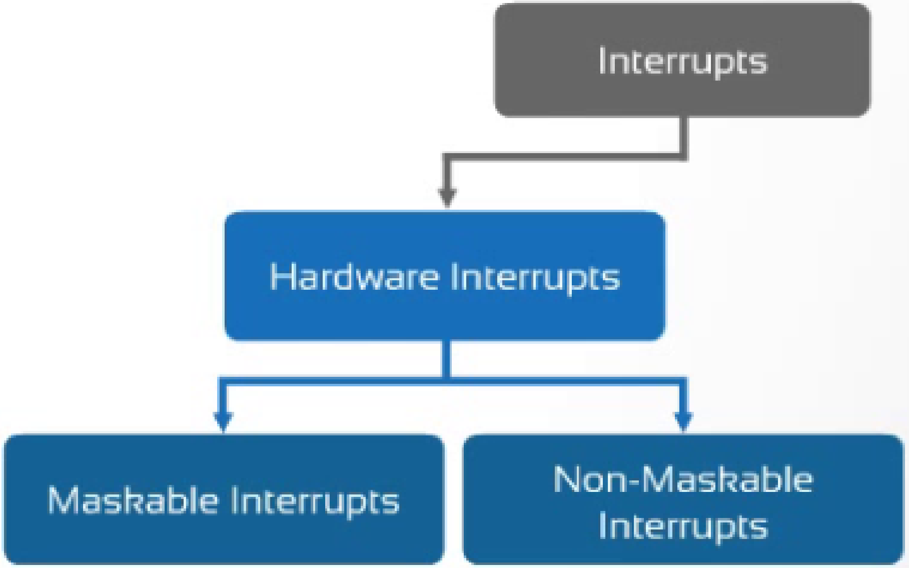
256 Types of software Interrupts
INT 00 to INT FF

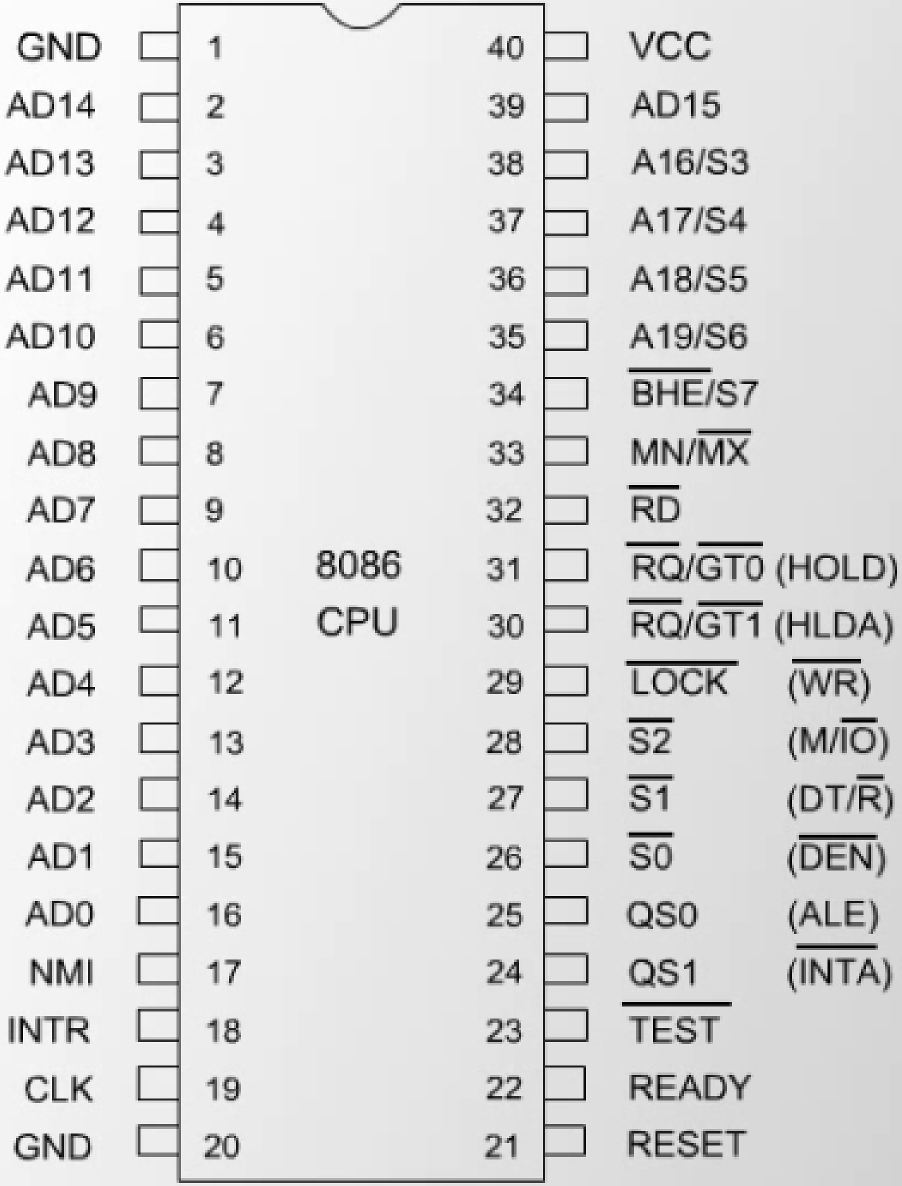
The programmer can choose to mask specific interrupts and re-enable them later

The programmer cannot control when a non maskable interrupt is served

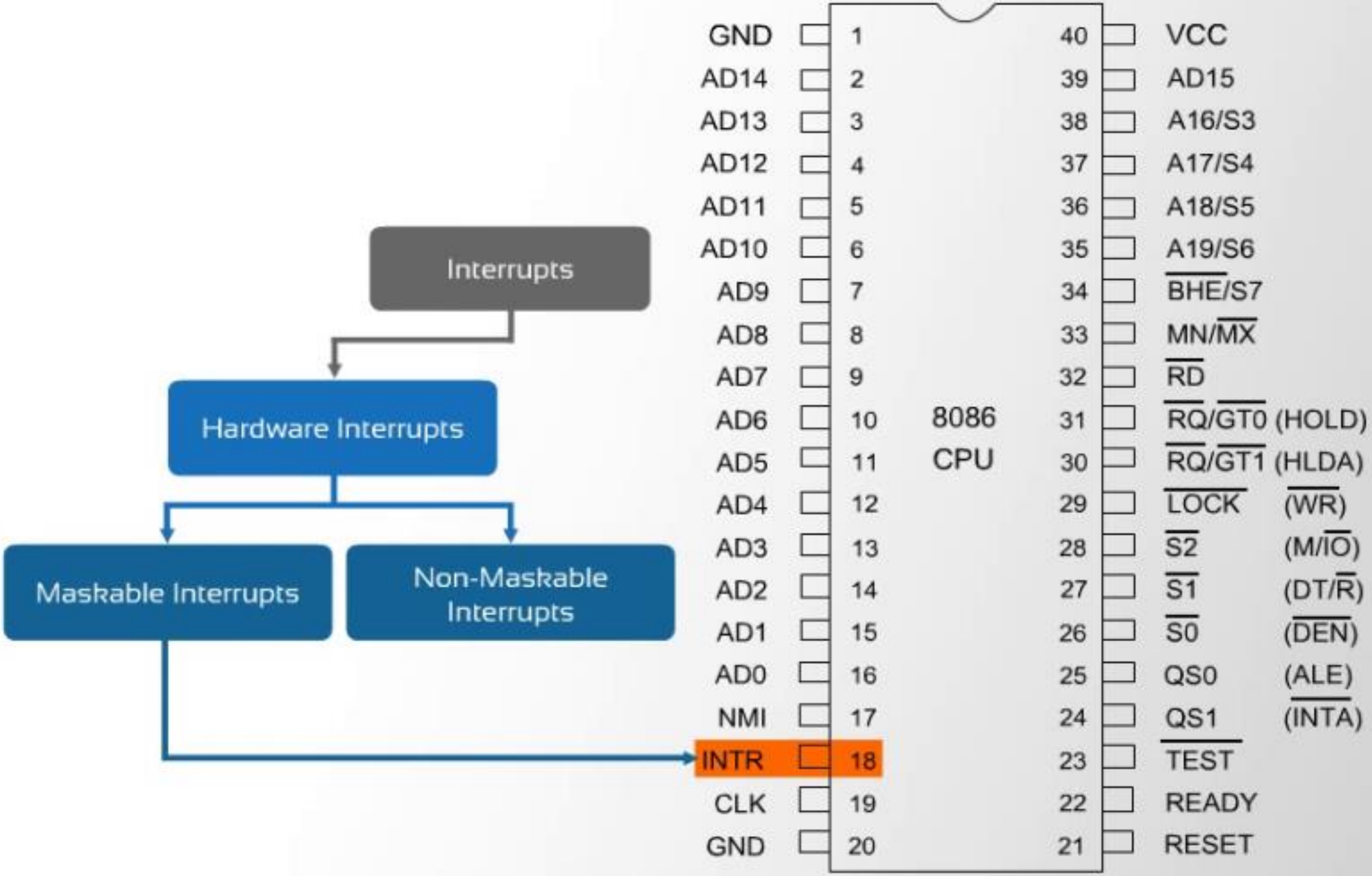
The processor has to stop the main program to execute the NMI Service Routine.

8086 CPU

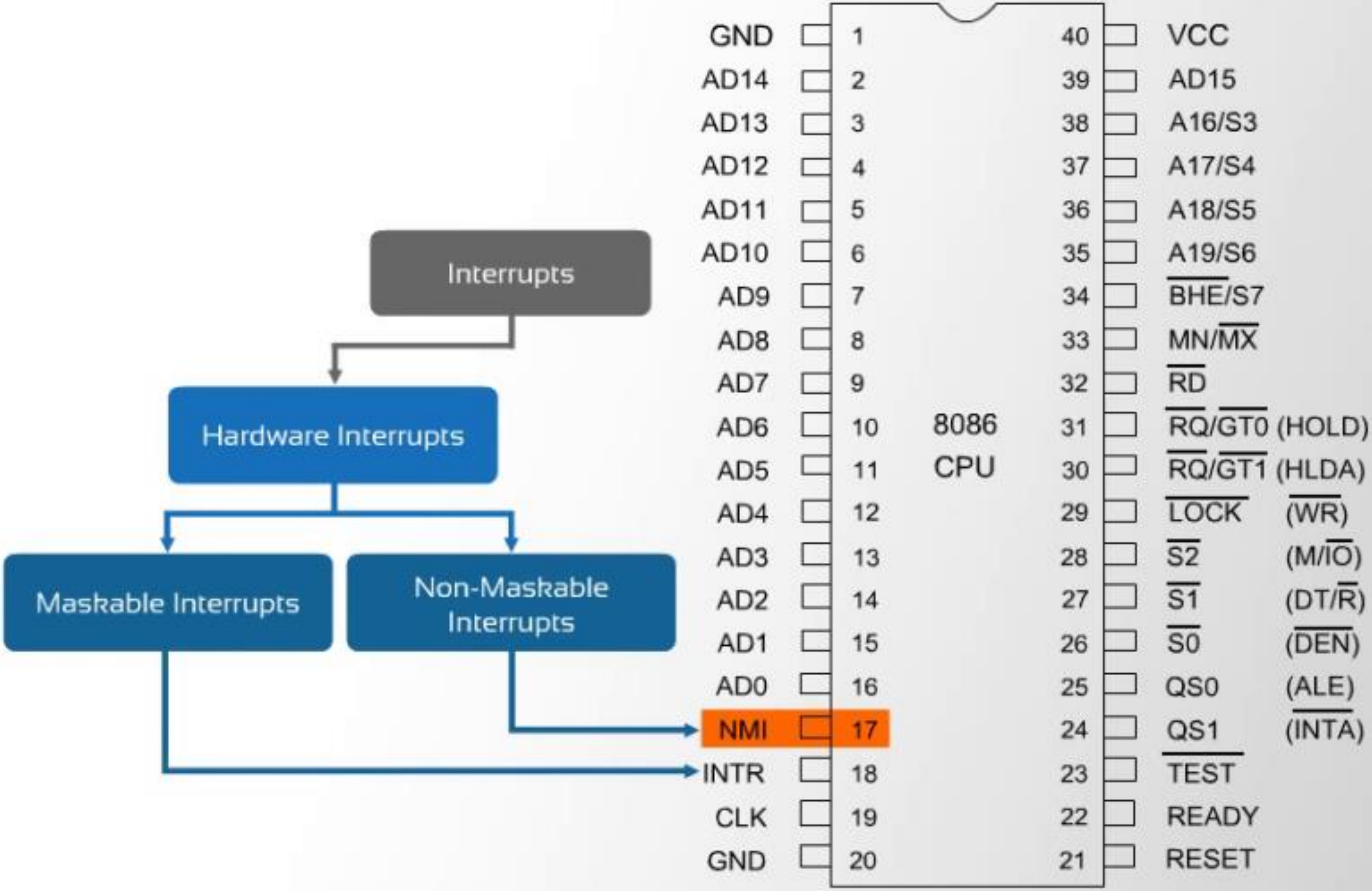

Interrupts
Hardware Interrupts
Maskable Interrupts
Non-Maskable Interrupts


8086
CPU
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
40 VCC
39 AD15
38 A16/S3
37 A17/S4
36 A18/S5
35 A19/S6
34 $\overline{BHE}$/S7
33 MN/$\overline{MX}$
32 $\overline{RD}$
31 $\overline{RQ}$/$\overline{GT0}$ (HOLD)
30 $\overline{RQ}$/$\overline{GT1}$ (HLDA)
29 $\overline{LOCK}$ ($\overline{WR}$)
28 $\overline{S2}$ (M/$\overline{IO}$)
27 $\overline{S1}$ (DT/$\overline{R}$)
26 $\overline{S0}$ ($\overline{DEN}$)
25 QS0 (ALE)
24 QS1 ($\overline{INTA}$)
23 $\overline{TEST}$
22 READY
21 RESET

8086 CPU


Interrupts
Hardware Interrupts
Maskable Interrupts
Non-Maskable Interrupts
8086
CPU
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
40 VCC
39 AD15
38 A16/S3
37 A17/S4
36 A18/S5
35 A19/S6
34 $\overline{BHE}$/S7
33 MN/$\overline{MX}$
32 $\overline{RD}$
31 $\overline{RQ}$/$\overline{GT0}$ (HOLD)
30 $\overline{RQ}$/$\overline{GT1}$ (HLDA)
29 $\overline{LOCK}$ ($\overline{WR}$)
28 $\overline{S2}$ (M/$\overline{IO}$)
27 $\overline{S1}$ (DT/$\overline{R}$)
26 $\overline{S0}$ ($\overline{DEN}$)
25 QS0 (ALE)
24 QS1 ($\overline{INTA}$)
23 $\overline{TEST}$
22 READY
21 RESET

8086 CPU



Interrupts

Hardware Interrupts

Maskable Interrupts

Non-Maskable Interrupts

Processing of Interrupt by the Processor

1. Executes the INT instruction
2. Interrupts the INT instruction during the assembly time
3. Moves the INT instruction to the Vector Table

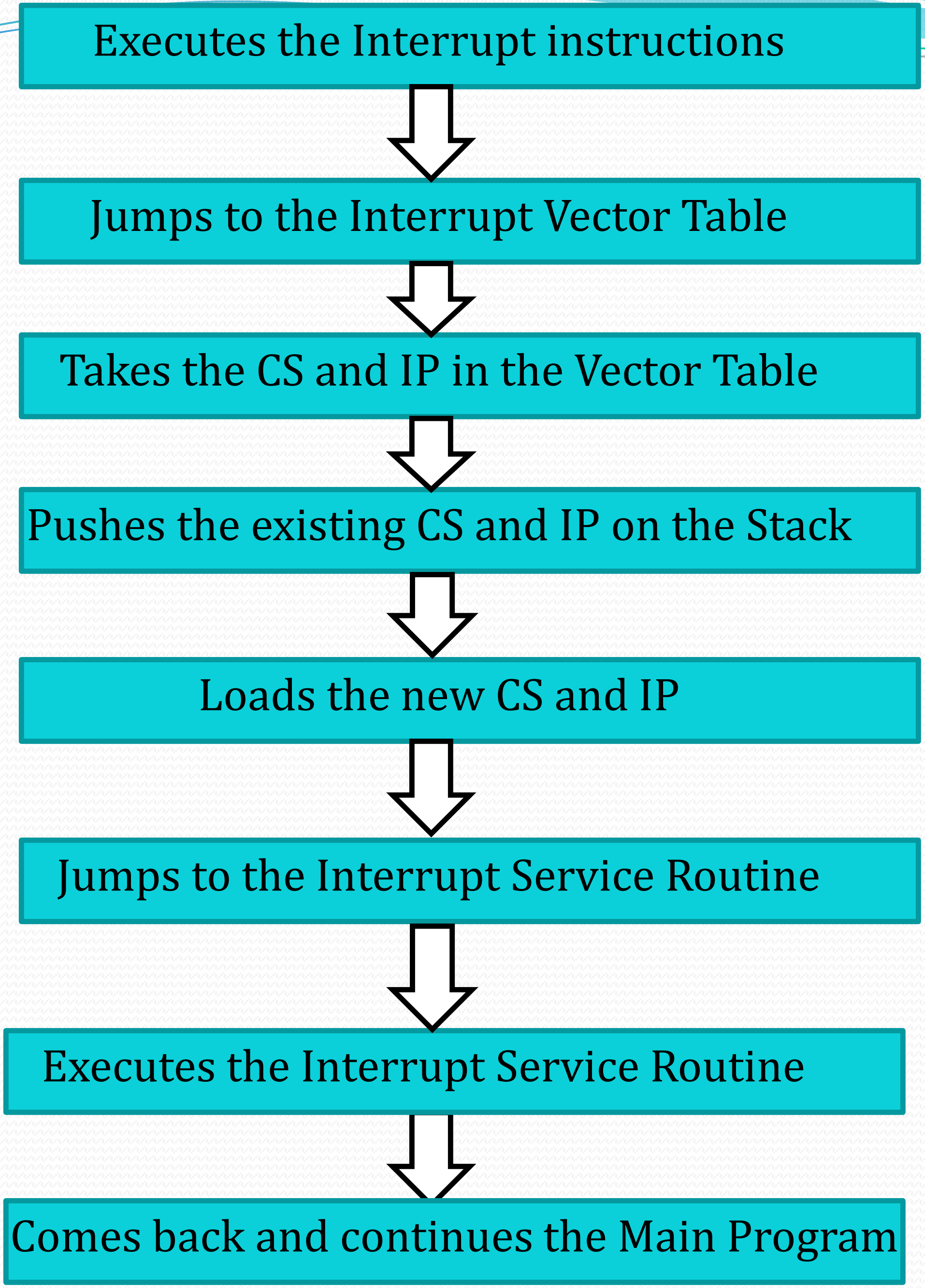
- Vector Table occupies location 00000H to 0003FFh of the program memory.
- It contains the code segment (CS) and Instruction Pointer (IP) for each kind of interrupt.

8086 Interrupt Processing

If an interrupt has been requested, the 8086 Microprocessor processes it by performing the following series of steps:

1. Pushes the content of the flag register onto the stack to preserve the status of the interrupt (IF) and trap flags (TF), by decrementing the stack pointer (SP) by 2
2. Disables the INTR interrupt by clearing IF in the flag register
3. Resets TF in the flag register, to disable the single step or trap interrupt
4. Pushes the content of the code segment (CS) register onto the stack by decrementing SP by 2
5. Pushes the content of the instruction pointer (IP) onto the stack by decrementing SP by 2
6. Performs an indirect far jump to the start of the interrupt service routine (ISR) corresponding to the received interrupt.

Steps involved in processing an interrupt instruction by the processor



Executes the Interrupt instructions

Jumps to the Interrupt Vector Table

Takes the CS and IP in the Vector Table

Pushes the existing CS and IP on the Stack

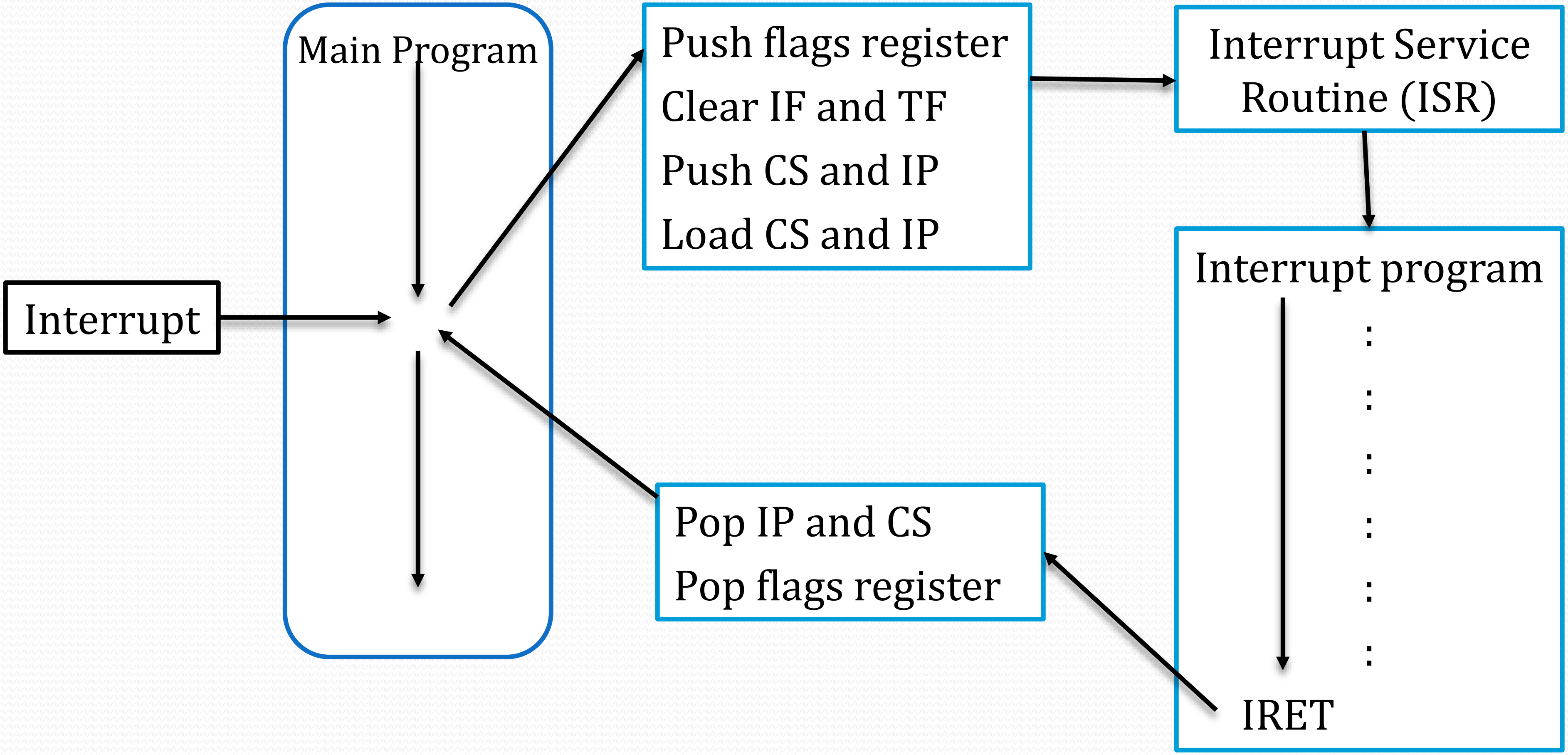
Loads the new CS and IP

Jumps to the Interrupt Service Routine

Executes the Interrupt Service Routine

Comes back and continues the Main Program

Processing of an Interrupt by the 8086



Main Program

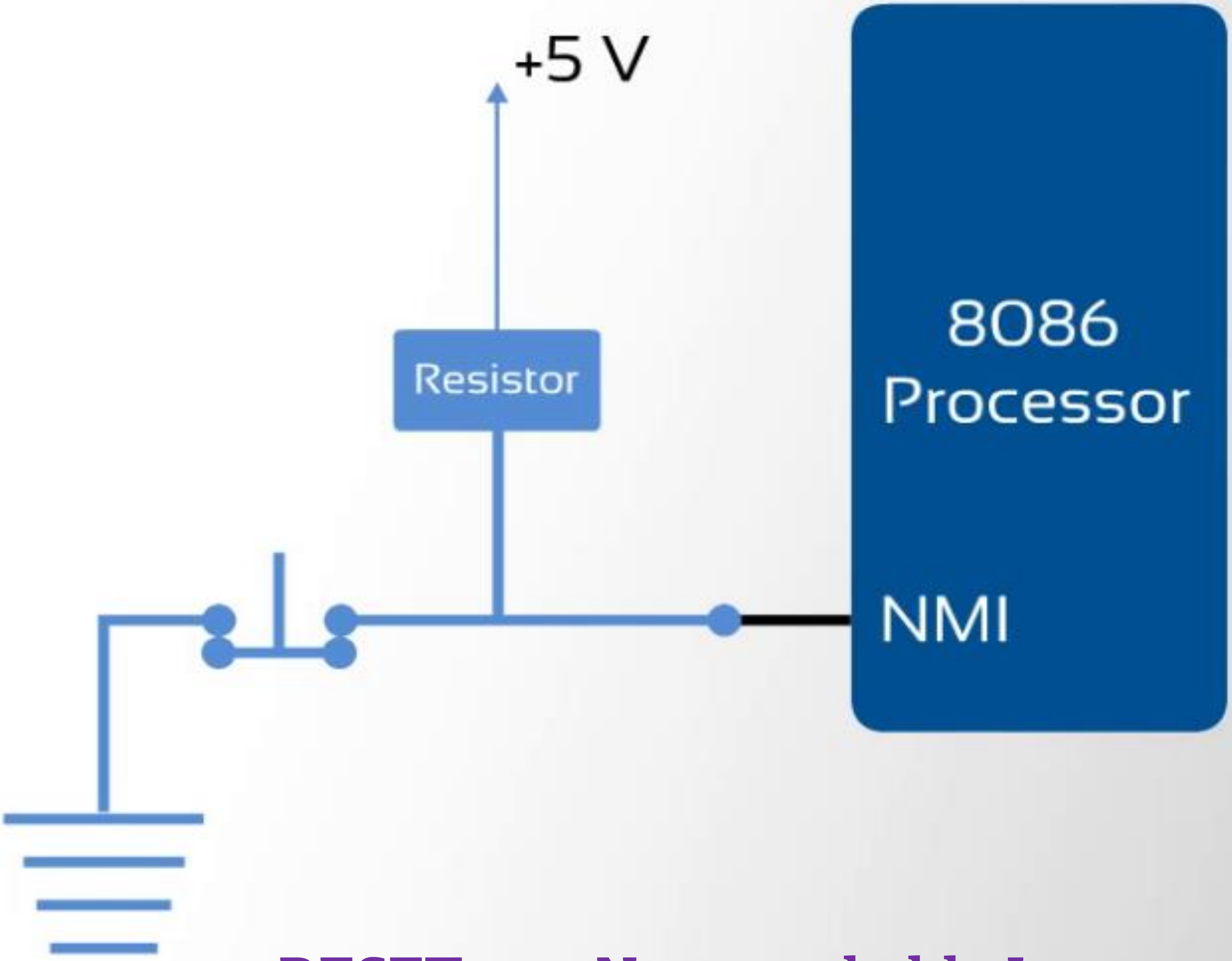
Interrupt

Push flags register
Clear IF and TF
Push CS and IP
Load CS and IP

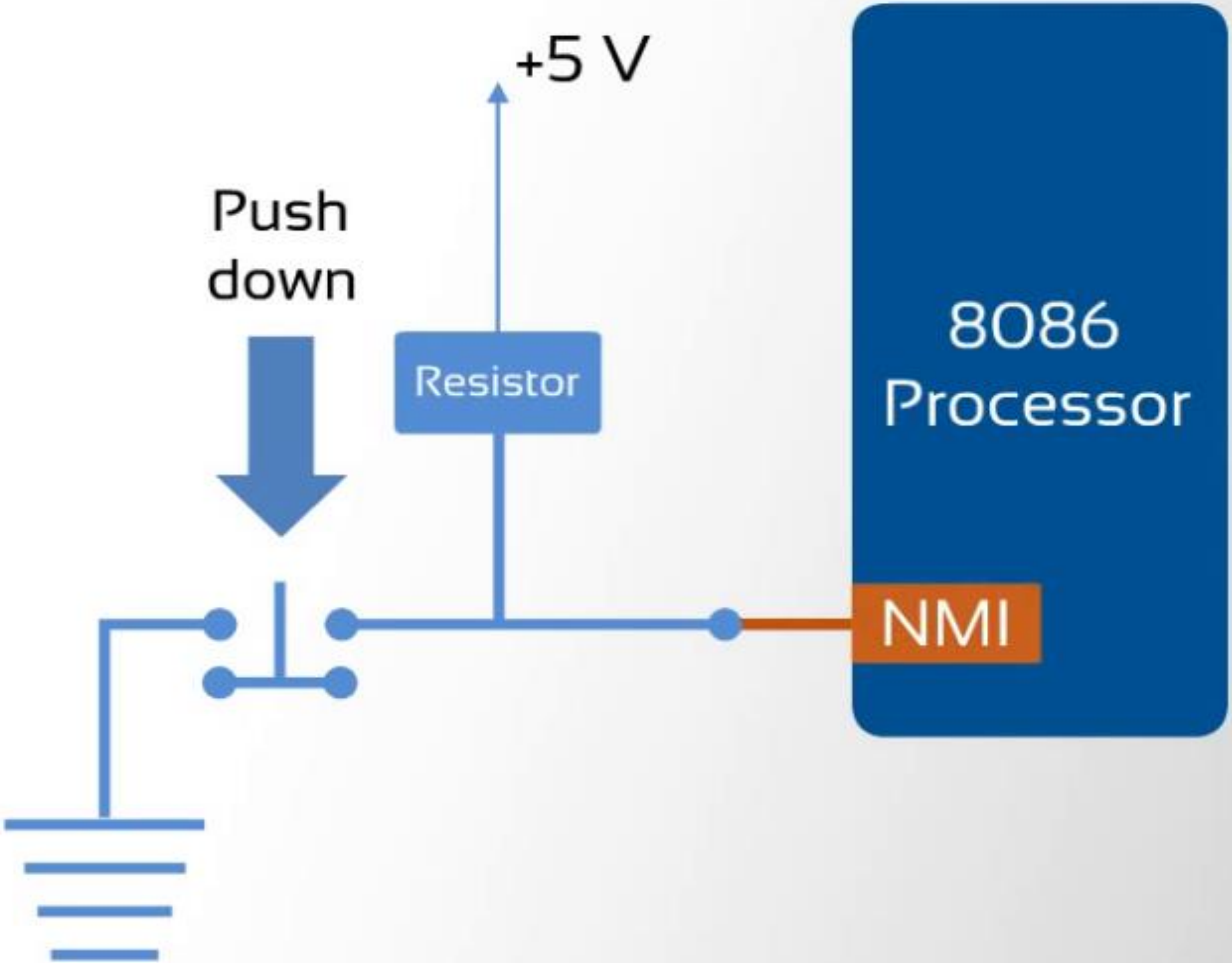
Interrupt Service Routine (ISR)

Interrupt program
:
:
:
:
:
:
IRET

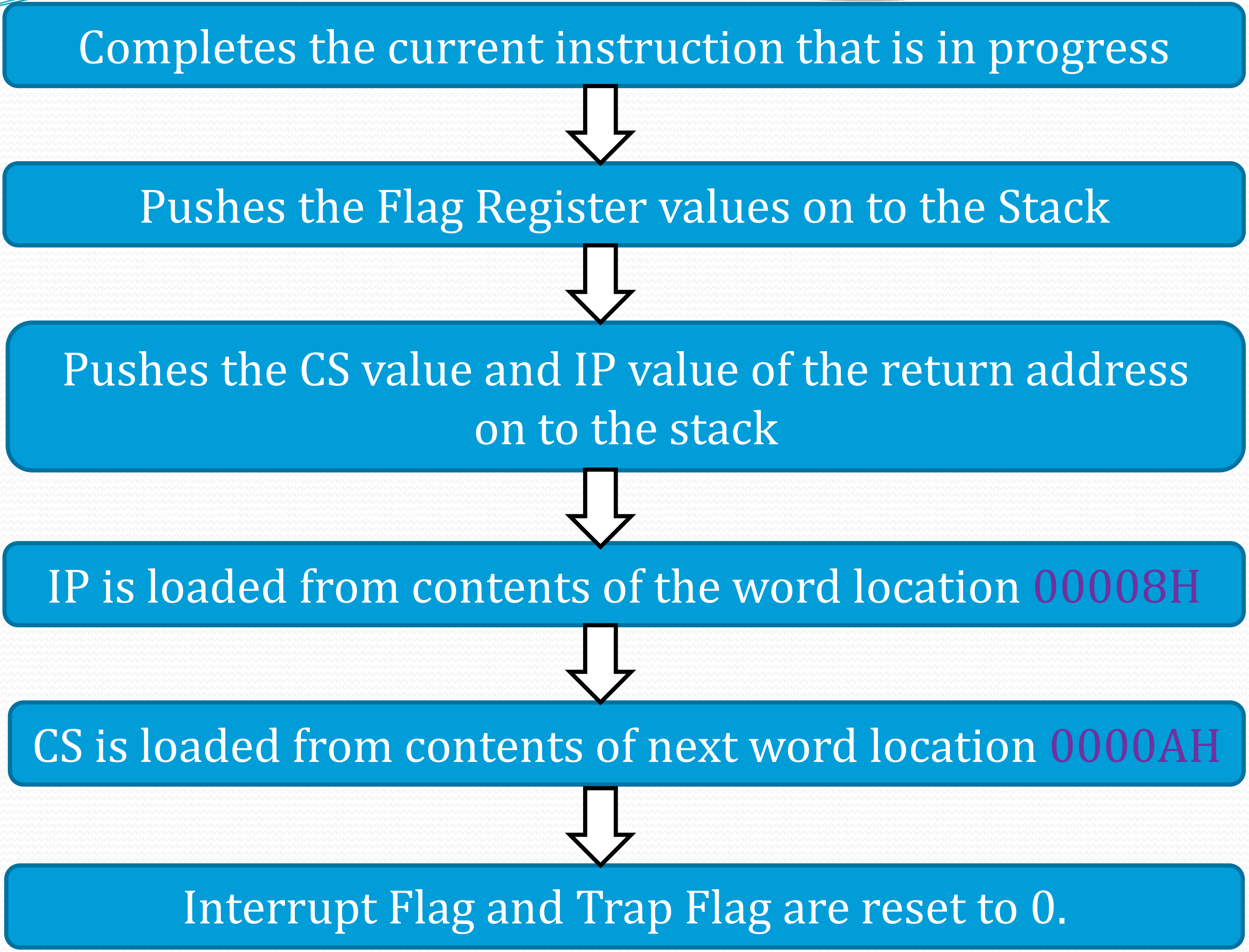
Pop IP and CS
Pop flags register



RESET as a Non-maskable Interrupts


+5 V
Push
down
Resistor
8086
Processor
NMI

Process sequence in the processor



Completes the current instruction that is in progress

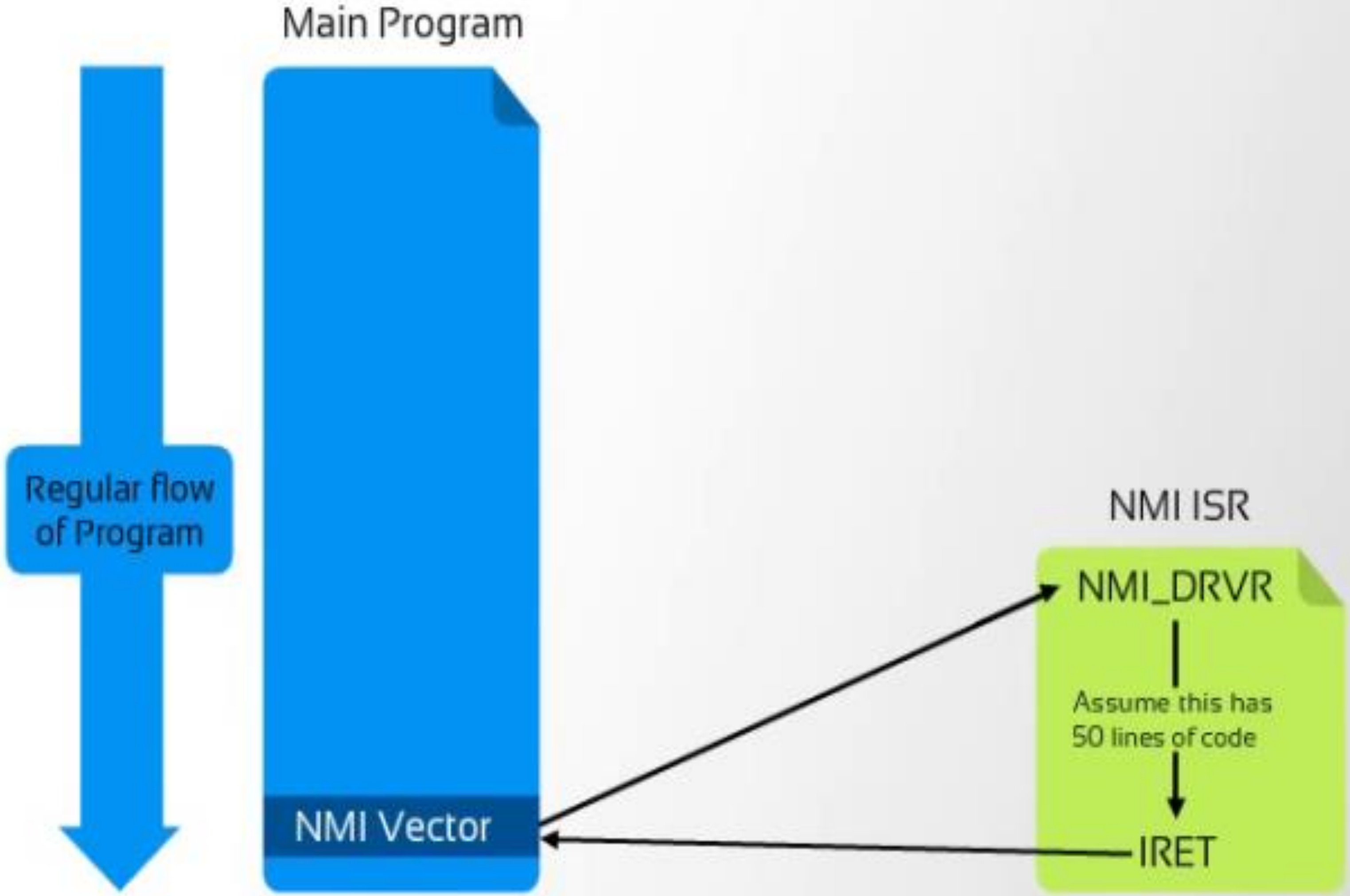
Pushes the Flag Register values on to the Stack

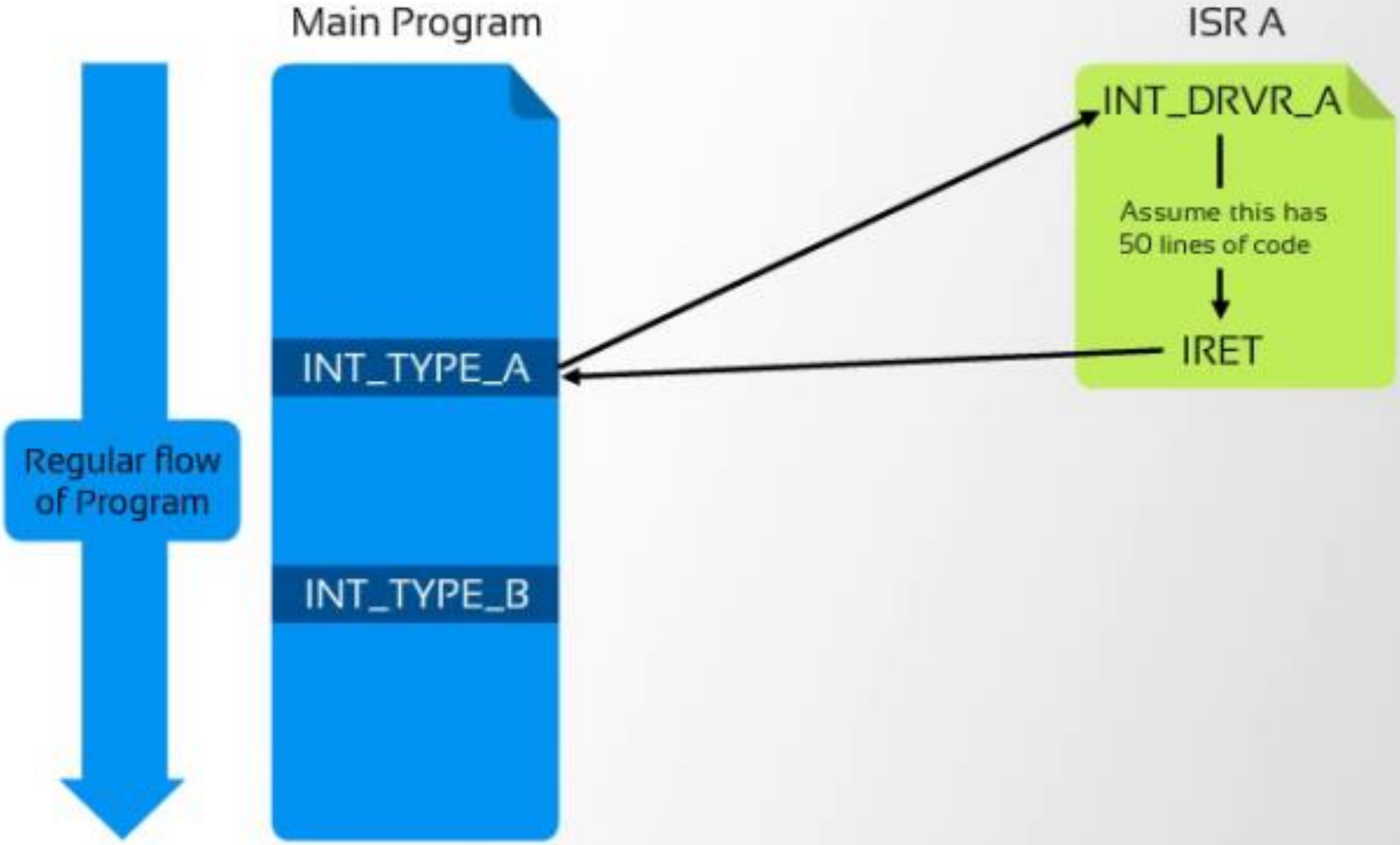
Pushes the CS value and IP value of the return address on to the stack

IP is loaded from contents of the word location 00008H

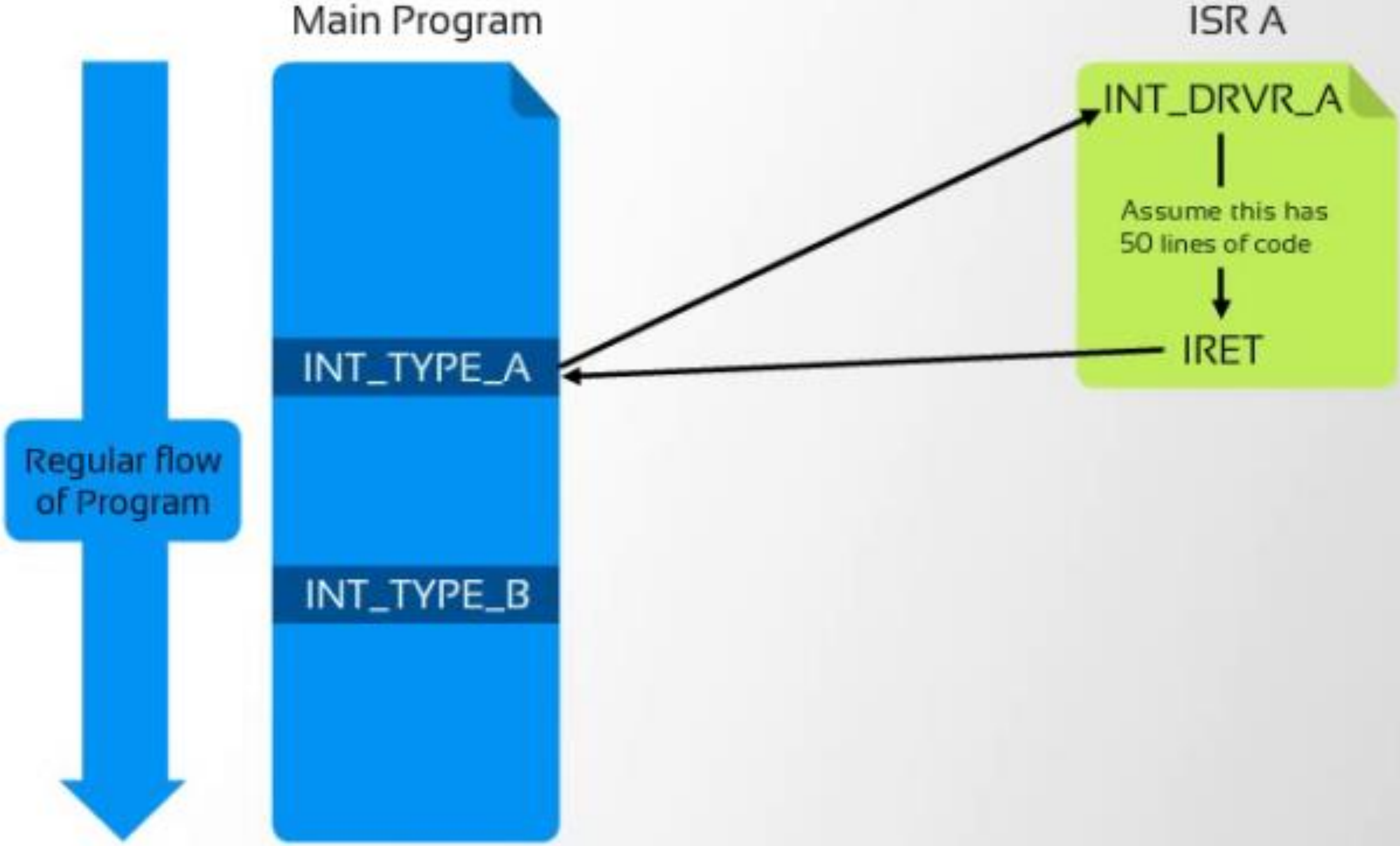
CS is loaded from contents of next word location 0000AH

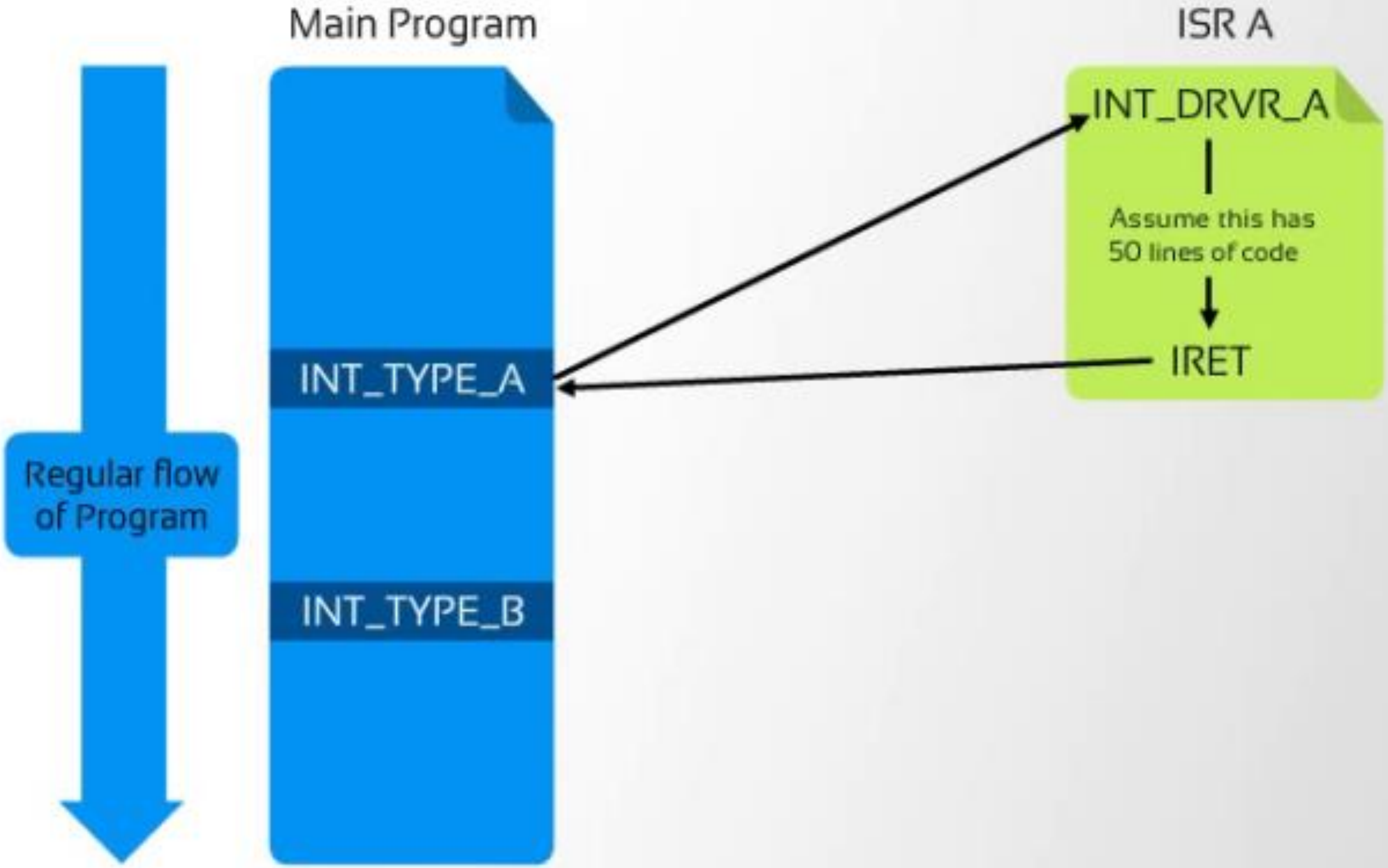
Interrupt Flag and Trap Flag are reset to 0.


Main Program
Regular flow of Program
NMI Vector
NMI ISR
NMI_DRVR
Assume this has 50 lines of code
IRET

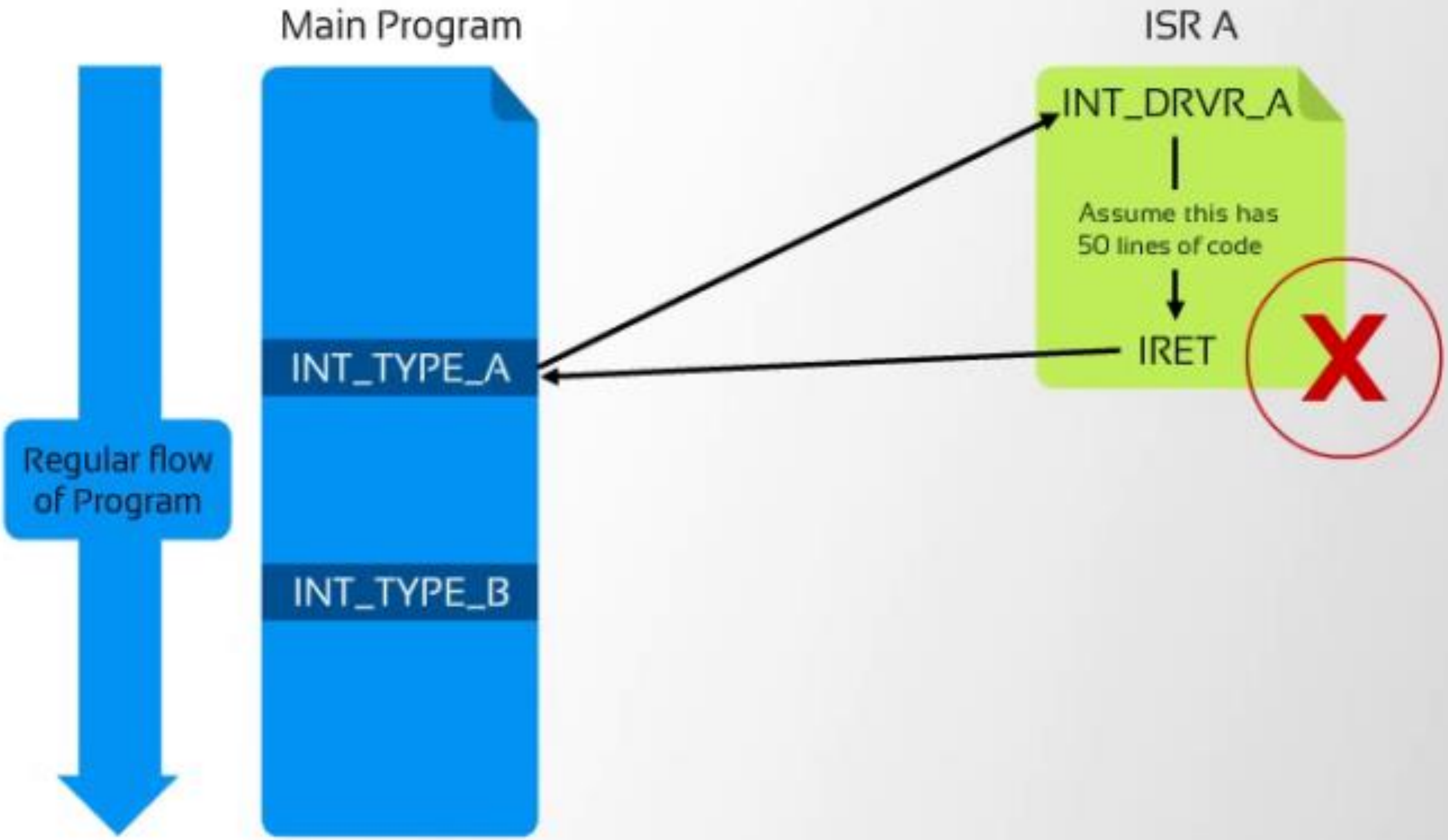

Main Program
ISR A
INT_DRVR_A
Assume this has 50 lines of code
IRET
INT_TYPE_A
Regular flow of Program
INT_TYPE_B

The processor pushes the Flag, the CS, and the IP before executing the ISR.

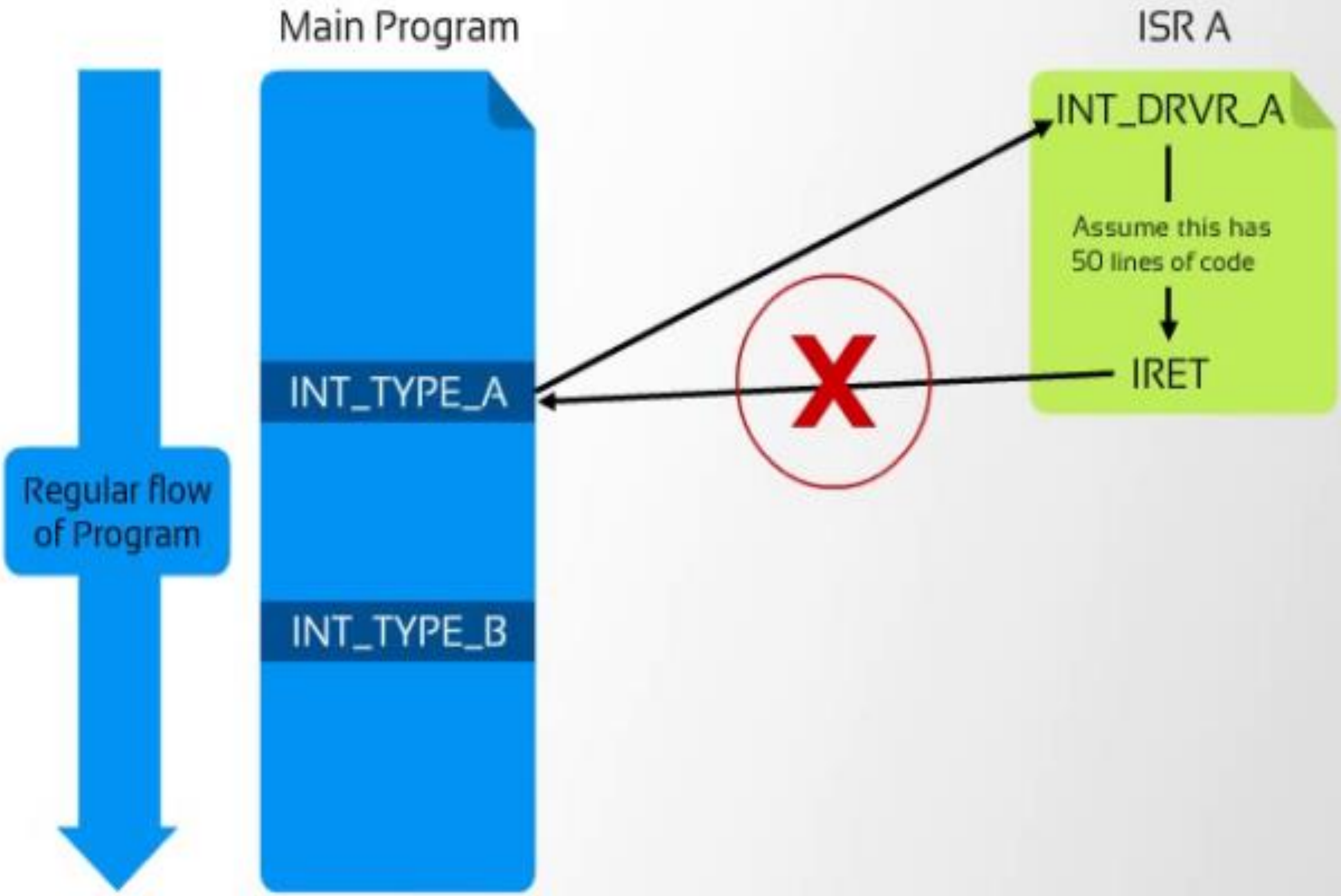




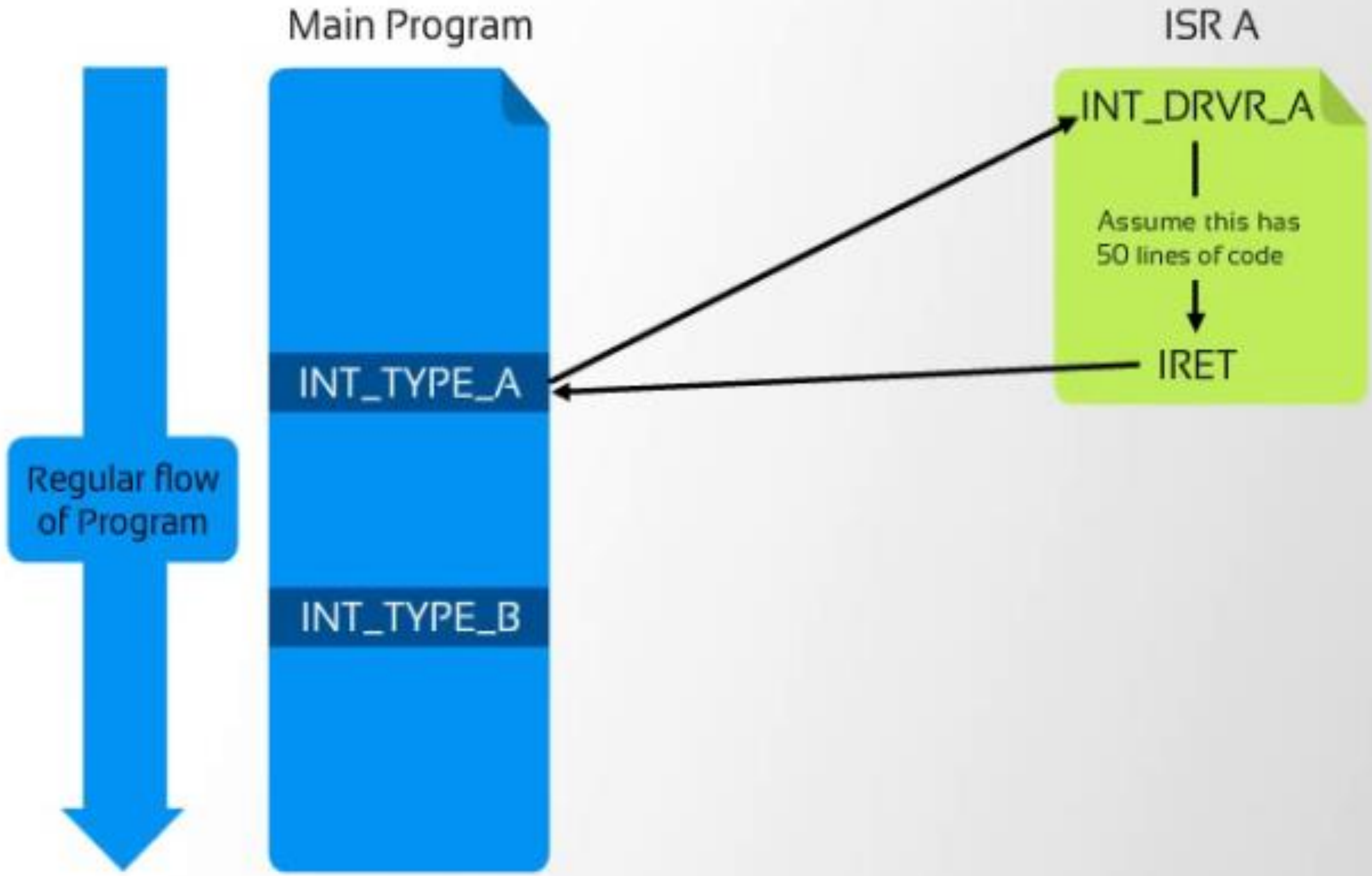
Interrupt Return Instruction

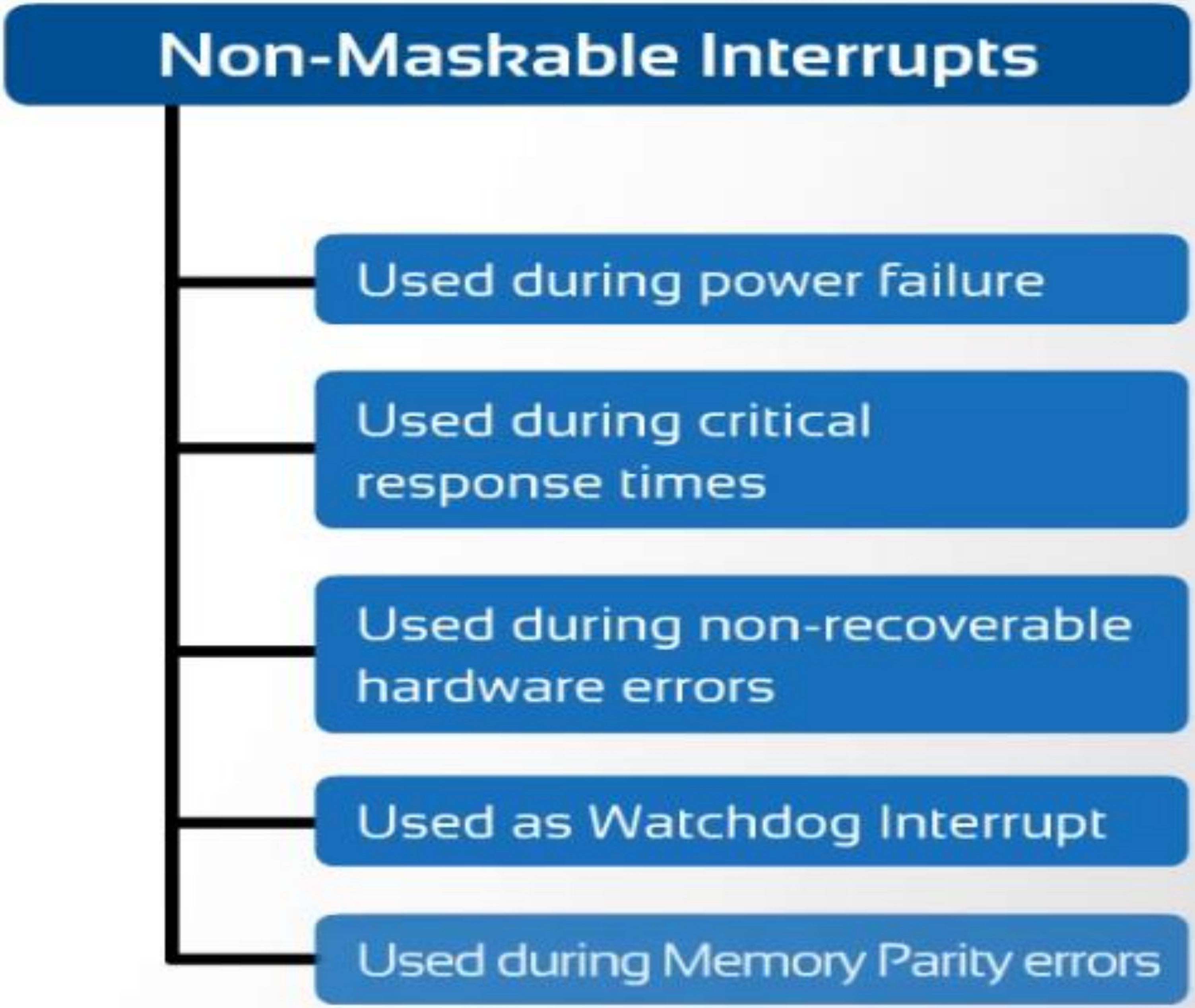


Main Program
ISR A
INT_DRVR_A
Assume this has 50 lines of code
IRET
INT_TYPE_A
Regular flow of Program
INT_TYPE_B
X


Main Program
ISR A
INT_DRVR_A
Assume this has 50 lines of code
IRET
INT_TYPE_A
Regular flow of Program
INT_TYPE_B
X

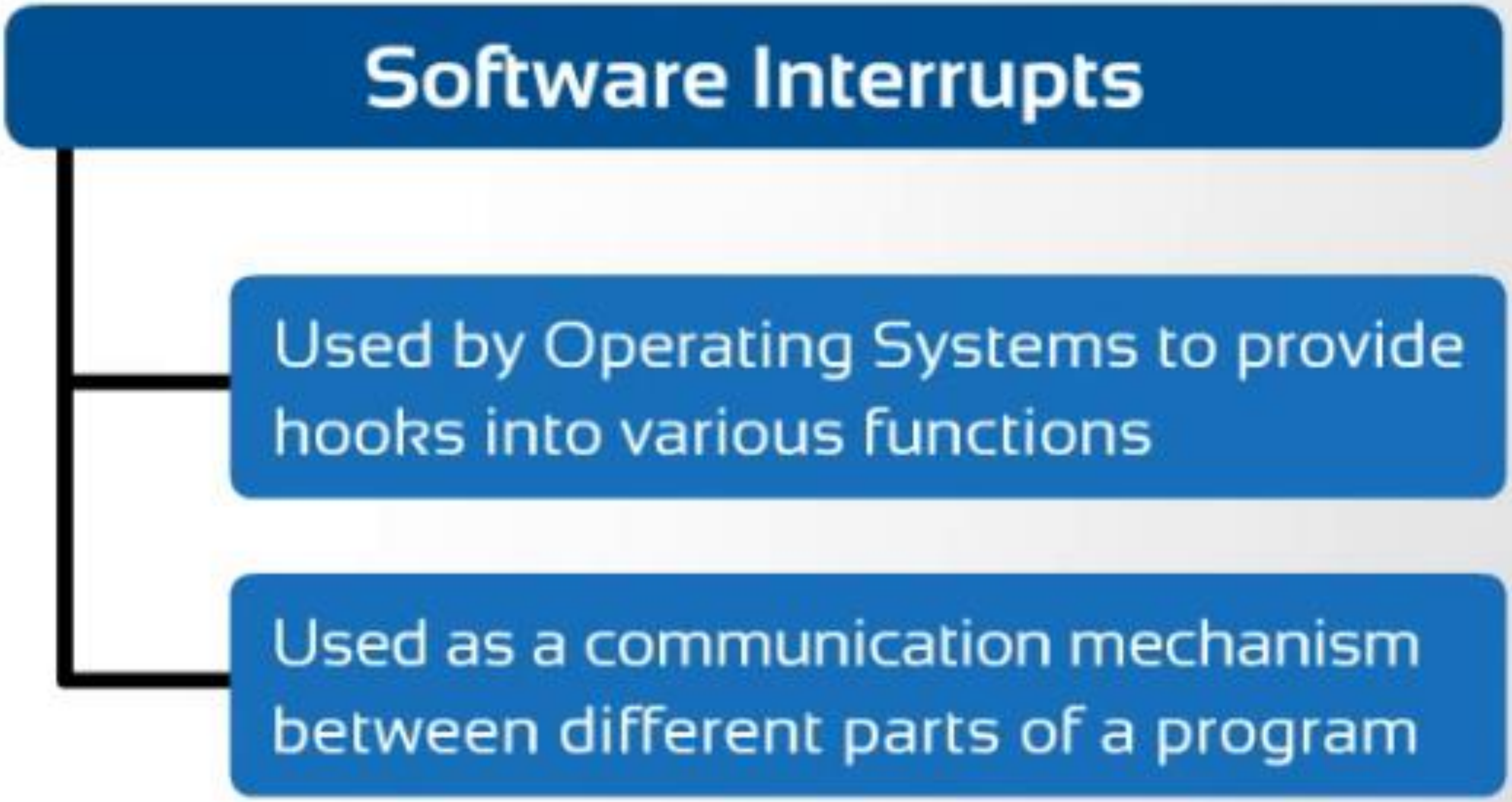
The execution of the IRET instruction by the processor pops the Flag Registers, IP, and CS.





Non-Maskable Interrupts

- Used during power failure
- Used during critical response times
- Used during non-recoverable hardware errors
- Used as Watchdog Interrupt
- Used during Memory Parity errors

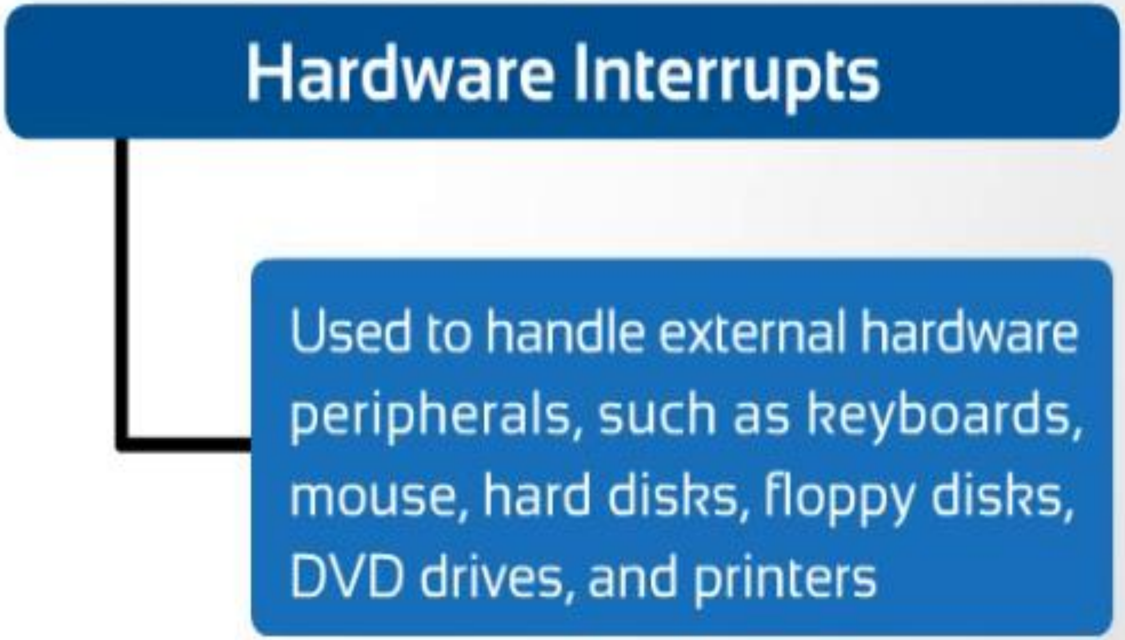


Software Interrupts

- Used by Operating Systems to provide hooks into various functions
- Used as a communication mechanism between different parts of a program

Hardware Interrupts

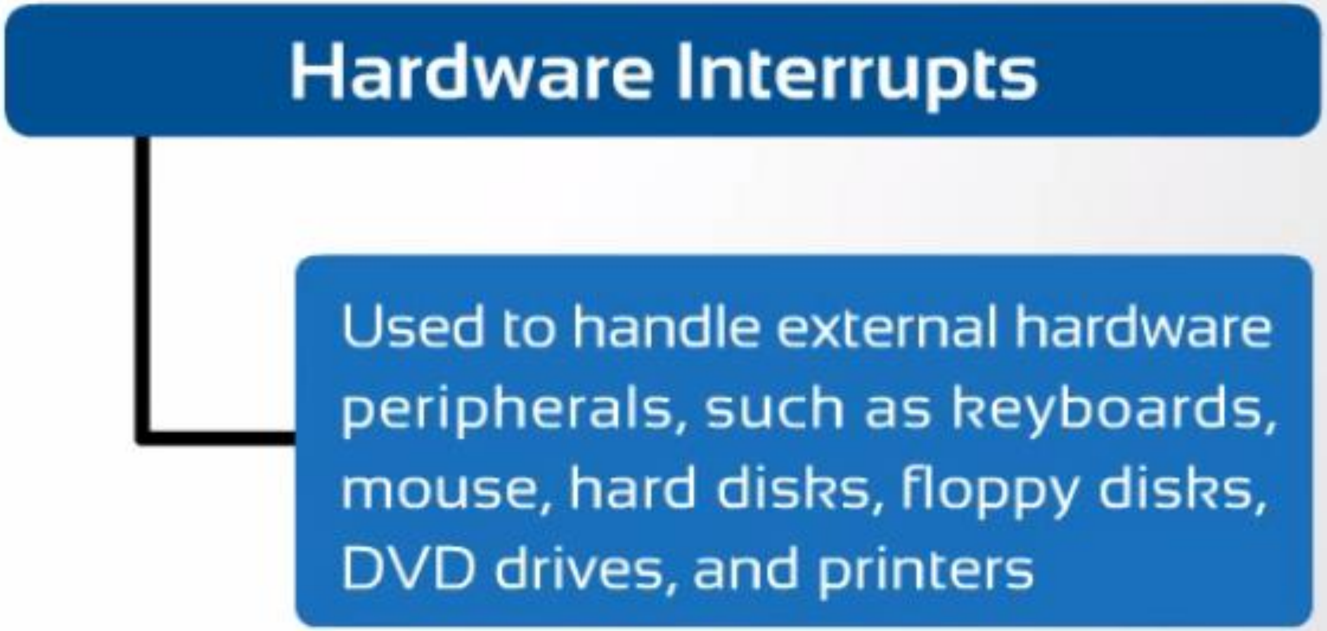
Used to handle external hardware peripherals, such as keyboards, mouse, hard disks, floppy disks, DVD drives, and printers



Hardware Interrupts

Used to handle external hardware peripherals, such as keyboards, mouse, hard disks, floppy disks, DVD drives, and printers

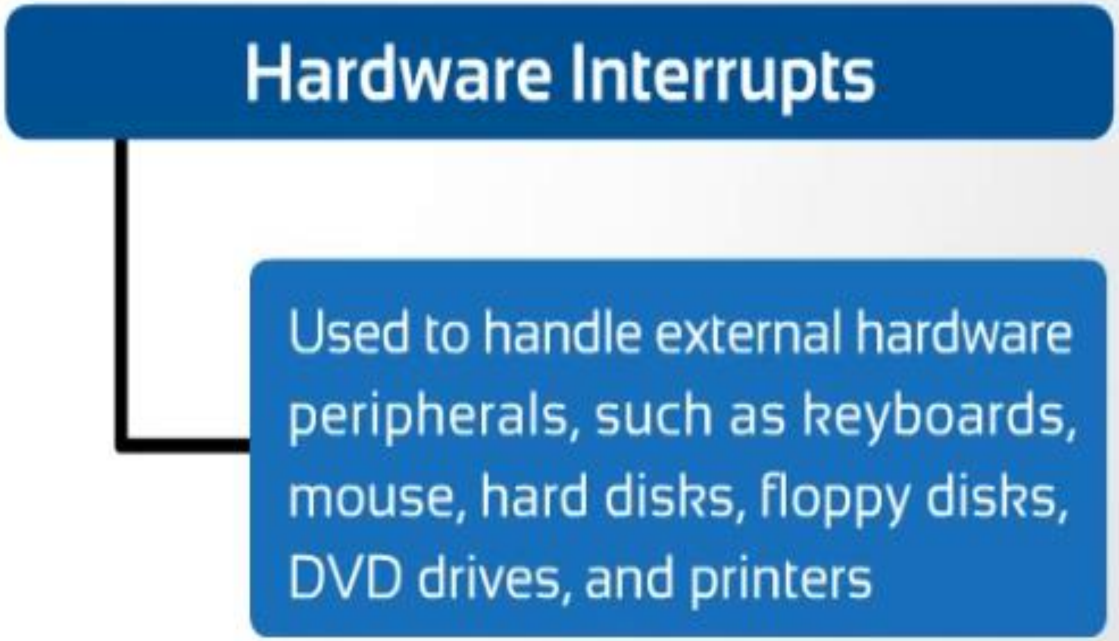
Interrupts are expensive in terms of time and processing power.



Hardware Interrupts

Used to handle external hardware peripherals, such as keyboards, mouse, hard disks, floppy disks, DVD drives, and printers

Without Interrupts, the systems are very simple.



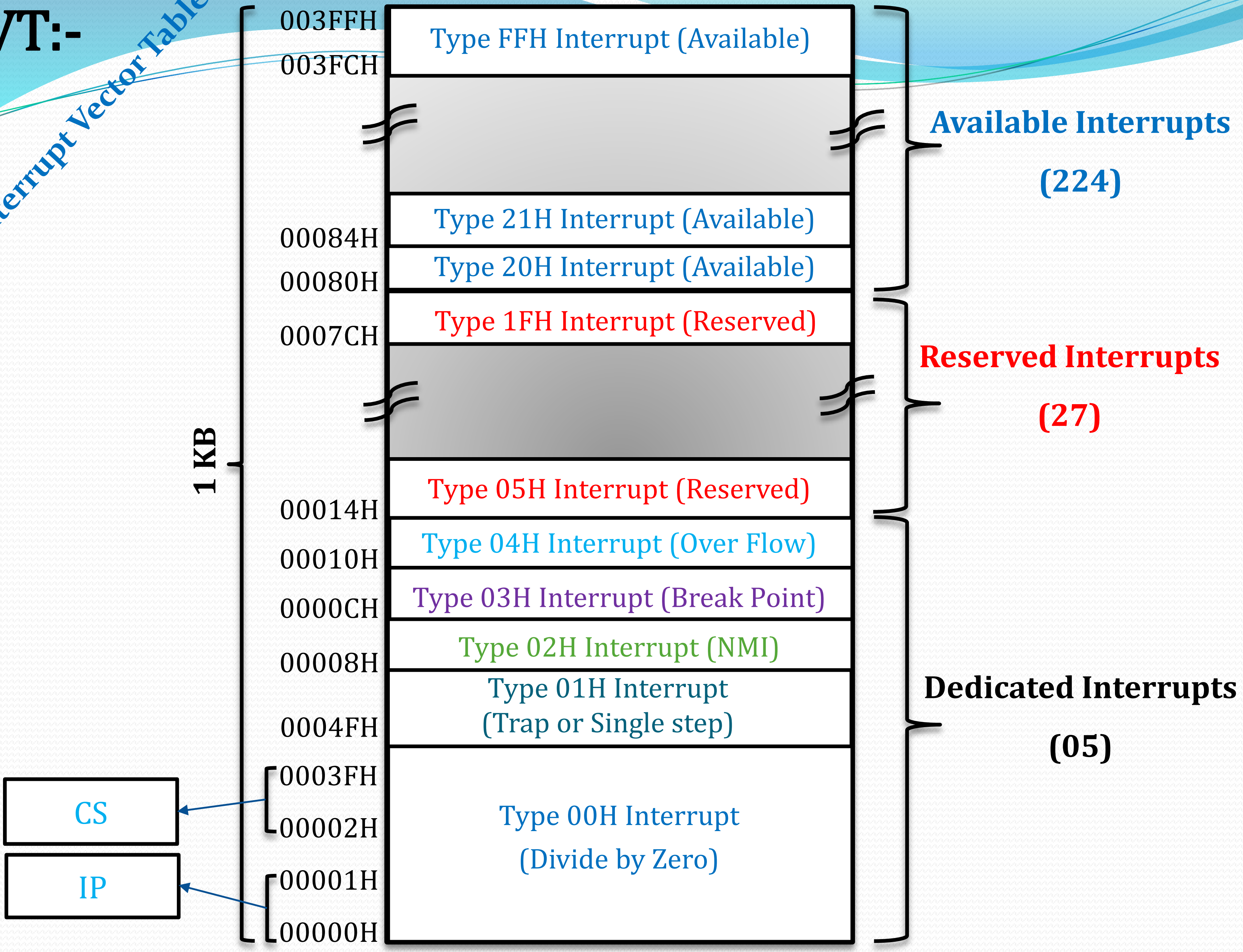
Hardware Interrupts

Used to handle external hardware peripherals, such as keyboards, mouse, hard disks, floppy disks, DVD drives, and printers

All microprocessor architectures have in-built interrupt service capability.

IVT:-

Interrupt Vector Table



003FFH

003FCH

Type FFH Interrupt (Available)

00084H

Type 21H Interrupt (Available)

00080H

Type 20H Interrupt (Available)

0007CH

Type 1FH Interrupt (Reserved)

00014H

Type 05H Interrupt (Reserved)

00010H

Type 04H Interrupt (Over Flow)

0000CH

Type 03H Interrupt (Break Point)

00008H

Type 02H Interrupt (NMI)

0004FH

Type 01H Interrupt
(Trap or Single step)

0003FH

00002H

00001H

00000H

Type 00H Interrupt
(Divide by Zero)

CS

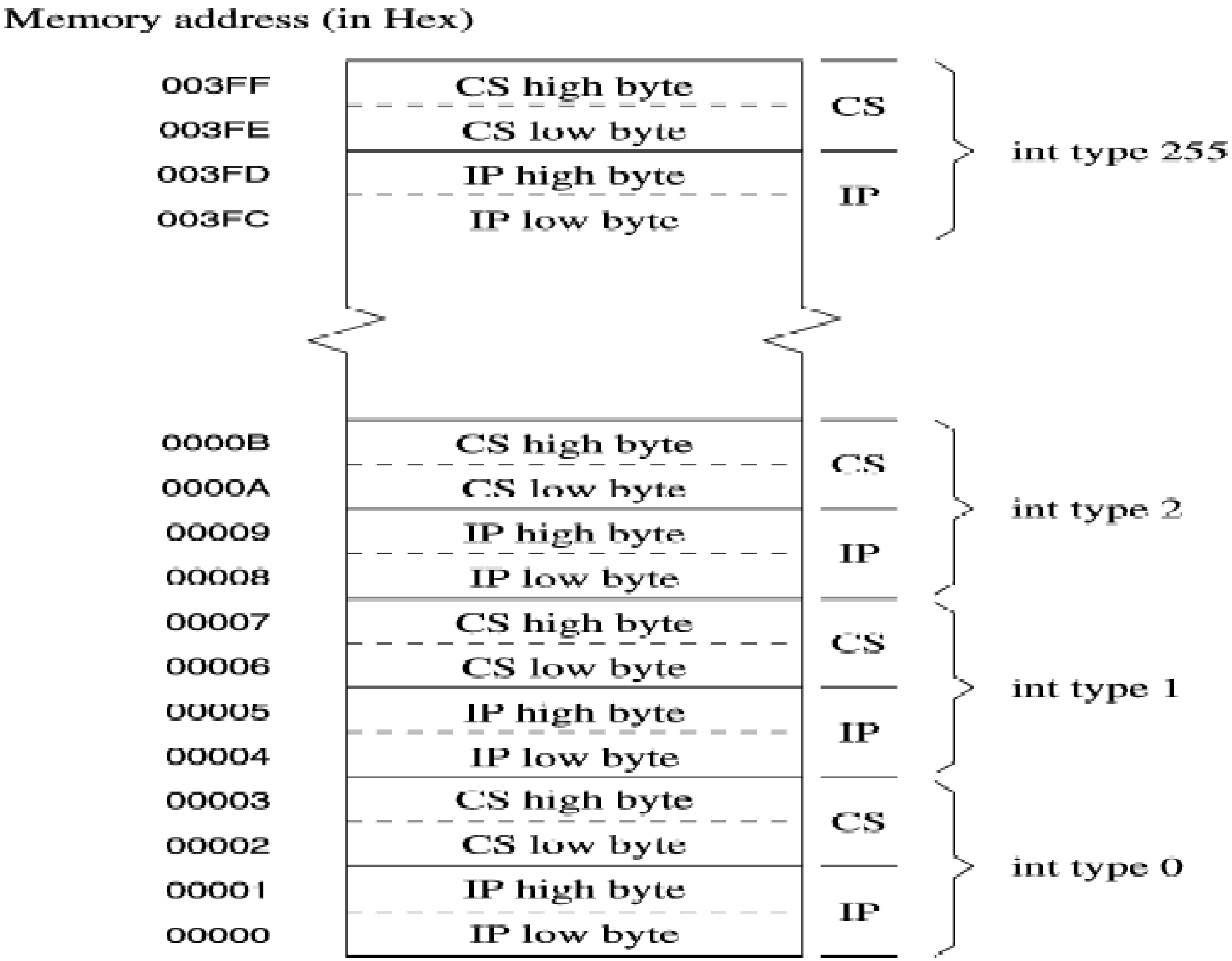
IP

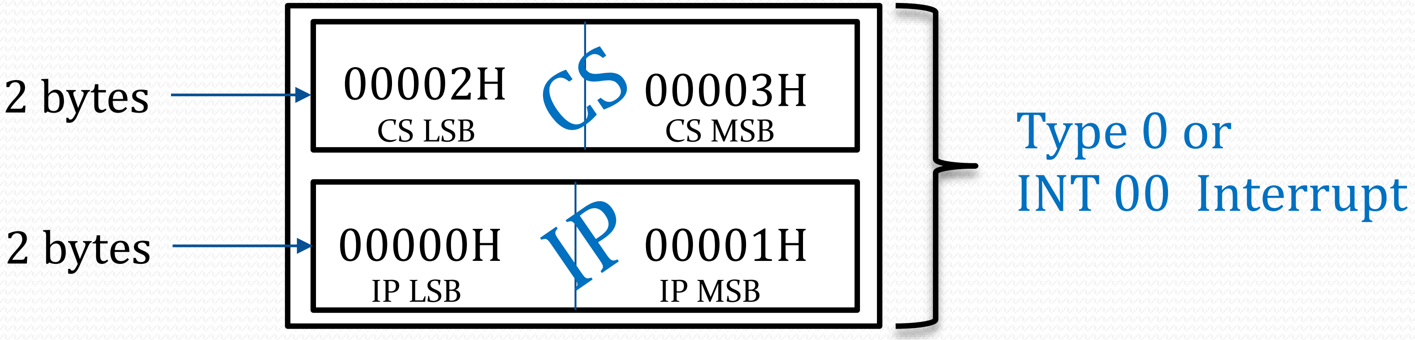
1 KB

Available Interrupts (224)

Reserved Interrupts (27)

Dedicated Interrupts (05)


Memory address (in Hex)
003FF
003FE
003FD
003FC
CS high byte
CS low byte
IP high byte
IP low byte
CS
IP
int type 255
0000B
0000A
00009
00008
CS high byte
CS low byte
IP high byte
IP low byte
CS
IP
int type 2
00007
00006
00005
00004
CS high byte
CS low byte
IP high byte
IP low byte
CS
IP
int type 1
00003
00002
00001
00000
CS high byte
CS low byte
IP high byte
IP low byte
CS
IP
int type 0



CS LSB MSB

Given a vector, where is the ISR address stored in memory ?

Offset = Type number X 4

Example:- INT 02h

Offset = 02 x 4 = 08

= 00008h

256 Interrupts Of 8086 are Divided in To 3 Groups

1. **Type 00 to Type 04 interrupts-**

   These are used for fixed operations and hence are called dedicated interrupts

2. **Type 05 to Type 31 interrupts**

   Not used by 8086,reserved for higher processors like 80286 80386 etc.

3. **Type 32 to Type 255 interrupts**

   Available for user, called user defined interrupts these can be H/W interrupts and activated through INTR line or can be S/W interrupts.

- **Type – 0 :- Divide by Zero Error Interrupt**

  Quotient is large cant be fit in al/ax or divide by zero

- **Type –1:- Single step or Trap Interrupt**

  Used for executing the program in single step mode by setting trap flag.

- **Type – 2:- Non-Maskable Interrupt**

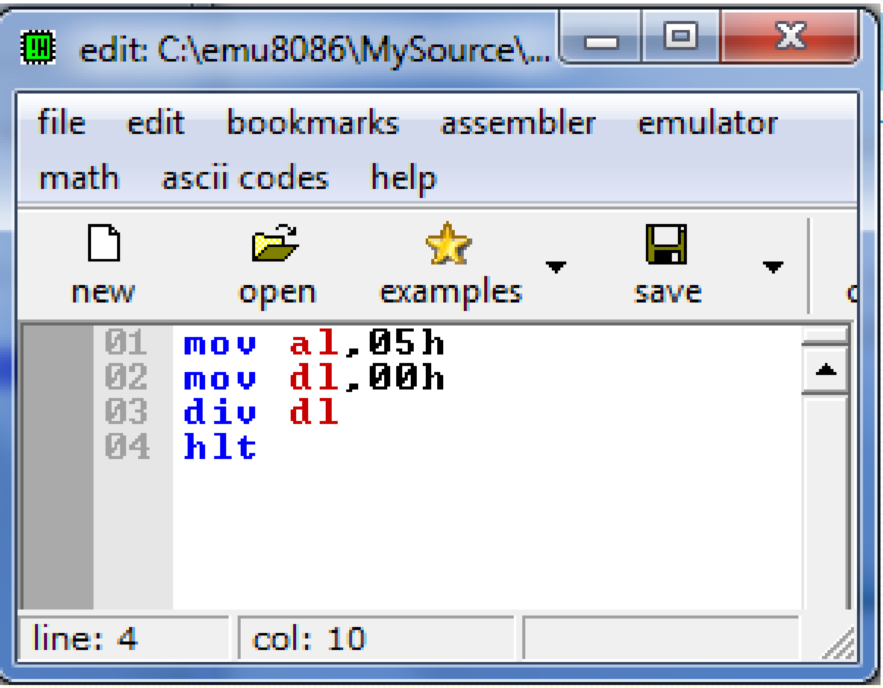
  This interrupt is used for executing ISR of NMI pin (positive edge signal), NMI can’t be masked by S/W.

- **Type – 3:- One-byte INT instruction interrupt**

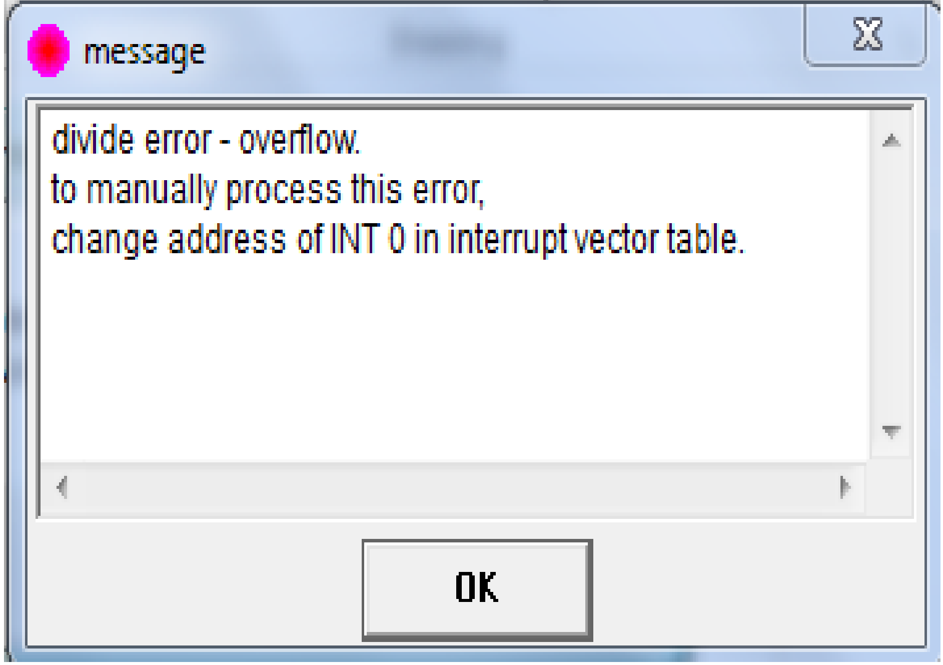
  Used for providing break points in the program

- **Type – 4 Over flow Interrupt**

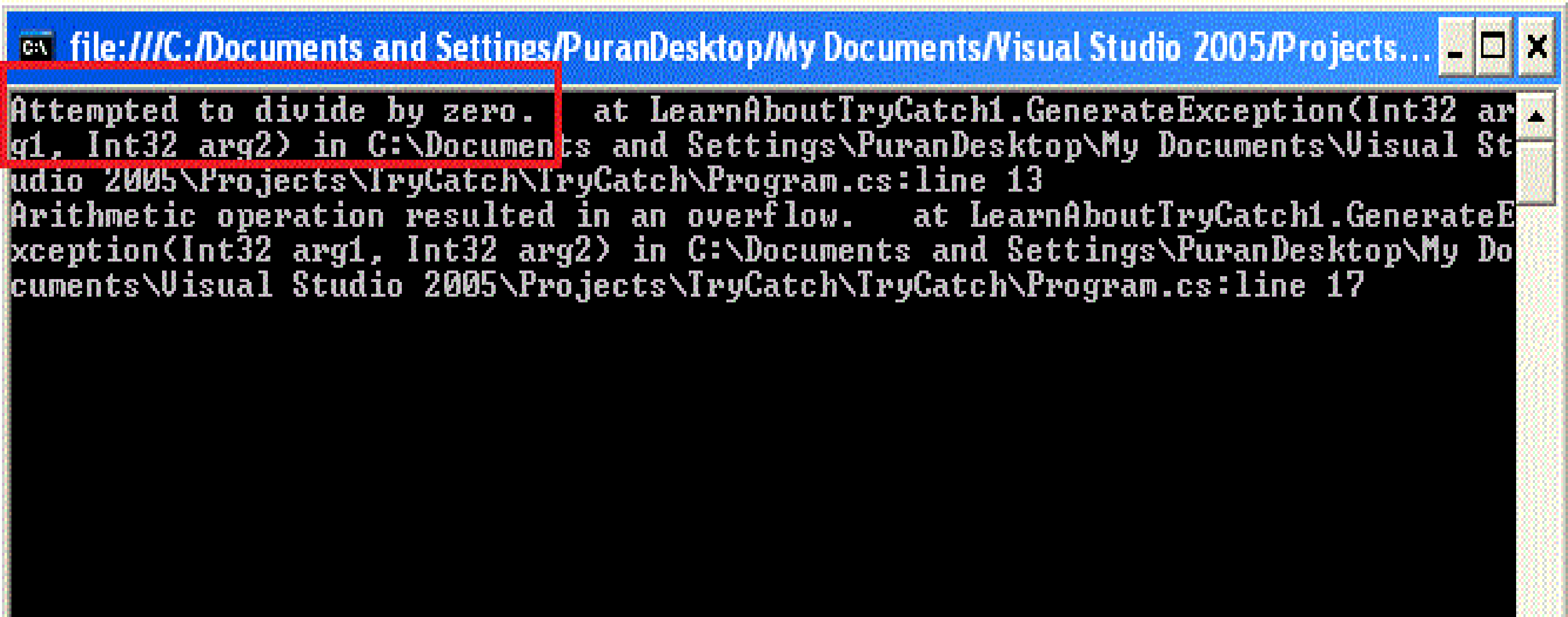
  Used to handle any overflow error after signed arithmetic.



```
mov  al,05h
mov  dl,00h
div  dl
hlt
```



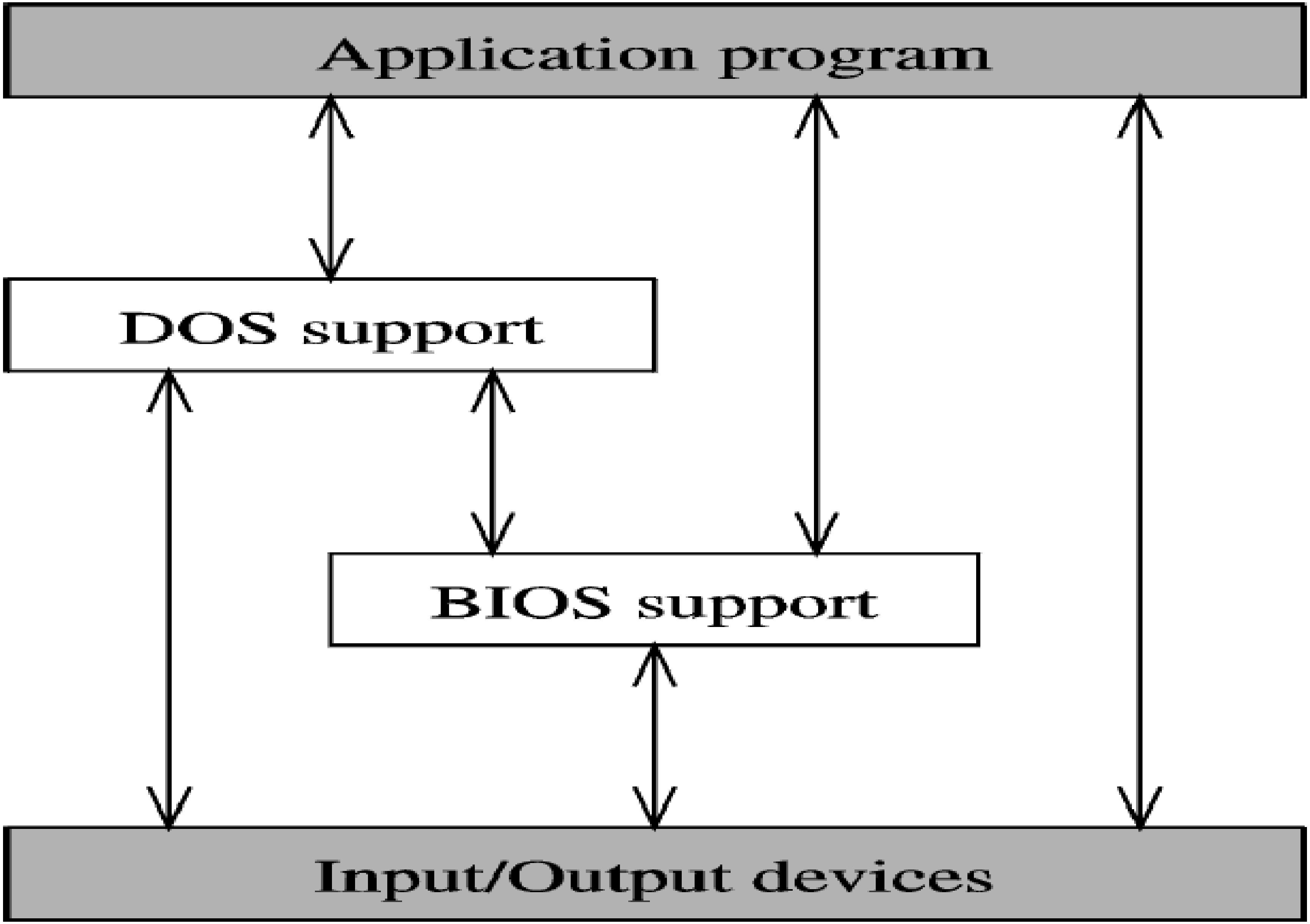
divide error - overflow.
to manually process this error,
change address of INT 0 in interrupt vector table.


file:///C:/Documents and Settings/PuranDesktop/My Documents/Visual Studio 2005/Projects...

```
Attempted to divide by zero.   at LearnAboutTryCatch1.GenerateException(Int32 ar
g1, Int32 arg2) in C:\Documents and Settings\PuranDesktop\My Documents\Visual St
udio 2005\Projects\TryCatch\TryCatch\Program.cs:line 13
Arithmetic operation resulted in an overflow.   at LearnAboutTryCatch1.GenerateE
xception(Int32 arg1, Int32 arg2) in C:\Documents and Settings\PuranDesktop\My Do
cuments\Visual Studio 2005\Projects\TryCatch\TryCatch\Program.cs:line 17
```

BIOS Interrupt or Function Calls

- The BIOS (basic input/output system) is boot firmware, which is designed to be the first program run by a PC when powered on.
- The initial function of the BIOS is to identify, test, and initialize system devices such as the video display card, hard disk, floppy disk, and other hardware.
- The BIOS prepares the machine for a known state, so that the software stored on the compatible media can be loaded, executed, and given control of the PC.
- *BIOS function calls*, also known as BIOS interrupts, are stored in the system ROM and in the video BIOS ROM present in the PC.


Application program
DOS support
BIOS support
Input/Output devices

DOS interrupts

- INT 21h is provided by DOS.
- When MS-DOS is loaded into the computer, INT 21H can be invoked to perform some extremely useful functions.
- These functions are commonly referred to as DOS INT 21H function calls.
- Data input and output through the keyboard and monitor are the most commonly used functions.

Below are two examples that use DOS interrupts.

1. Display the message defined with variable DATA DB ‘Microprocessor,’$’

```
MOV AH,09                Option 9 to display string of data
MOV DX, OFFSET DATA      DX= offset address of data
INT 21H                  Invoke the interrupt
```

2. Inputting a single character, with echo.

```
MOV AH, 01     Option 01 to input one character
INT 21H        Invoke the interrupt
```

DOS (Disk Operating System) Interrupts

1. Function call 01: Read the key board

   Input parameter AH = 01 read a character from keyboard. Echo it on CRO screen and return the ASCII code of the key pressed in AL output parameter: AL = ASCII code of character

2. Function call 02h: Display on CRT screen

   Input parameter: AH = 02

   Dl = ASCII character to be displayed on CRT screen

3. Function call 03: Read character from com1

   Input parameter: AH = 03h

   Function: Reads data from com port

   Output parameter: AL = ASCII code of character

4. Function call 04: Write character to com1

   Input parameter: AH = 04h DL = ASCII code of character to be transmitted

   Function: Writes data to com port

5. Function call 05: Write to Lpt1

   Input parameter: AL = 05H

   DL = ASCII code of character to be printed

   Function: Print the character available in dl on printer attached to Lpt1

6. Function call 09: Display a character string

   Input parameter: AH = 09, DS:DX= Address of character string

   Function: Displays the characters available in the string to CRT till a $ symbol

7. Function call 0Ah: Buffered key board input

   Input parameter: AH = 0Ah

   DS:DX = Address of keyboard input buffer

   Function: The ASCII codes of the characters received from keyboard are stored in keyboard buffer from 3rd byte. 1st byte of buffer = size of buffer upto 255. It receives the characters till specified no.Of characters are received or enter key is presses which ever is earlier

BIOS interrupt

INT 10H subroutines are burned into the ROM BIOS of the 8086 based and compatibles and are used to communicate with the computer user screen video. Much of the manipulation of screen text or graphics is done through INT 10H. Among them are changing the color of characters or background, clearing screen, and changing the locations of cursor. Below are two examples that use BIOS interrupts.

1. Clearing the screen:

```
MOV AX, 0600H ;scroll entire screen
MOV BH, 07 ;normal attribute
MOV CX, 0000 ;start at 00,00
MOV DX, 184FH ;end at 18, 4F
INT 10H ;invoke the interrupt
```

BIOS (Basic Input/output System) INTERRUPTS

INT 10H:Video Service Interrupt:-It Controls The Video Display

a) Function Call 00: Select Video Mode

Input Parameter: AL = Mode Number
AH = 00h

Function: It Changes The Display Mode And Clears The Screen

AL = 00	40 X 25 Black And White
AL = 04	320 X 200 Color
AL = 10h	640 X 350 X 16 Color

b) Function Call 03: Read Cursor Position

Input Parameter: AH = 03
BH = Page Number

Function: Reads Cursor Position On Screen

Output Parameters: CH = Starting Line
CL = Ending Line
DH = Current Row
DL = Current Column

C) Function Call 0E: Write Character On CRT Screen And Advance Cursor

Input Parameter: Ah = 0Eh
AH = ASCII Code Of The Character
BH = Page(text Mode)
BH = Color(graphics)

Function: Display Character Available In Al On Screen

Other Interrupts:-

INT 11H:- *Determine the type of equipment installed. Register AX should contain* FFFFH and instruction INT 11H to be executed. On return, register AX will indicate the equipment's attached to computer

INT14H:- Control the serial communication port attached to the computer. Ah should contain the function call

a) Function Call 00:initialize The Com Port

b) Function Call 01: Send A Character

c) Function Call 02:receive A Character

INT 16H:- Keyboard interrupt AH should contain the function call

a) Function call 00: Read keyboard character, It will return ASCII code of the character

b) Function call 01: Get key board status

Thank You