CHAPTER III: ADDITIONAL INFORMATION ON BASIC MECHANISMS

I/ Introduction:

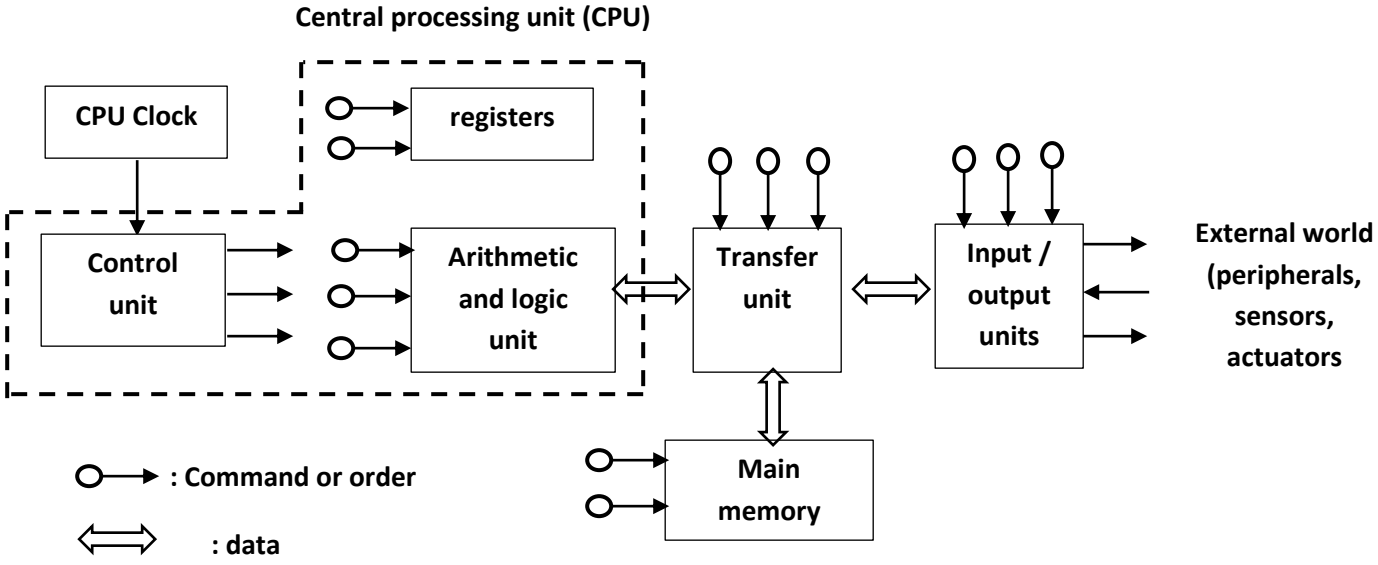
The implementation of an operating system requires knowledge of the characteristics of hardware elements, some of which participate in program execution and others which interrupt it. The operating system's function is to manage these elements to ensure reliable execution.

In this chapter, we'll look at processor components and interrupt concepts.

II/ Von Newmann architecture:

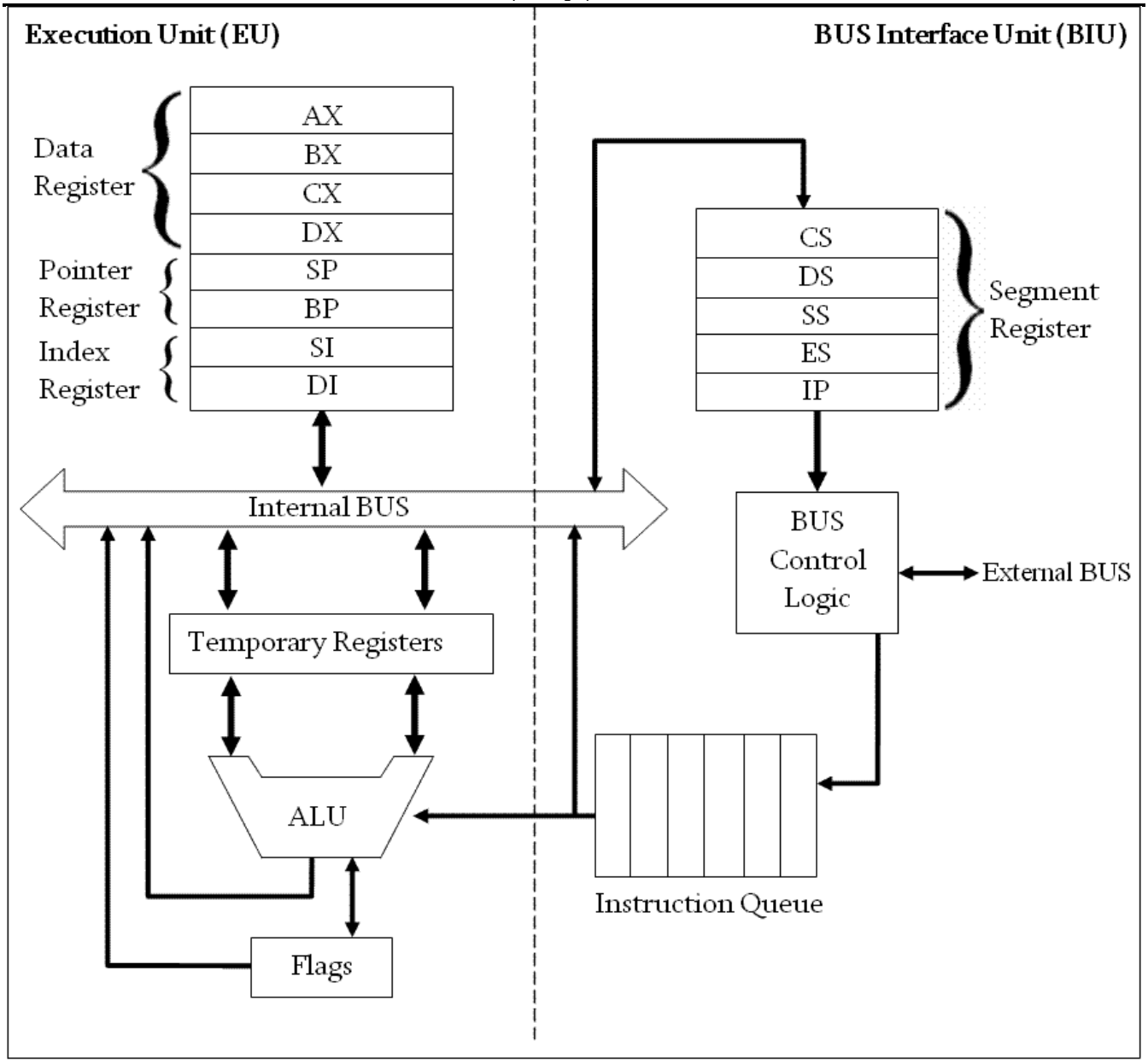
The Von Newmann machine consists of the following components:

- ✓ The main memory for storing data and running programs.
- ✓ The processor to perform the necessary calculations and processing.
- ✓ Peripherals for information exchange.



III/ The processor:

Unit capable of executing an instruction, it consists of :



1/ <u>The control and command unit:</u>

- ✓ It is used to control the correct operation of processing and controls instruction flow:
- ✓ *<u>Ordinal counter or instruction pointer (IP, CO):</u>* register containing the address of the memory word where the next instruction to be executed is located.
- ✓ *<u>Instruction register (RI):</u>* holds the current instruction while it is being interpreted.
- ✓ *<u>Instruction decoder (DI):</u>* analyzes the instruction's operation code to distribute the various elementary commands.
- ✓ *<u>Sequencer</u>*: orders all micro-commands to all units (memory, ALU, etc.) to process the instruction.

- ✓ *Status register (Processor Status Word PSW):* contains information on the status of the CPU and the instructions that have just been executed. It is an n-bit mask containing flags such as: overflow flag, parity flag, execution mode flag, protection bit, interrupt code, etc.

				O	D	I	T	S	Z		A		P		C
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

CF: carry indicator;
PF: parity indicator;
AF: auxiliary hold indicator;
ZF: zero indicator;
SF: sign indicator;
TF: trap indicator;
IF: interrupt authorization flag;
DF: decrement indicator;
OF: overflow flag.

- ✓ *The clock*: is a circuit that transmits electrical pulses at regular intervals, defining the sequential operation of the processor so that machine cycles are synchronized with the clock (the clock is a square-wave signal with a fixed frequency, such as 3Ghz).

❖ The 8086 microprocessor contains 14 registers divided into 4 groups in addition to IP and IR:

- **General registers**: 4 16-bit registers

AX = (**AH**,**AL**) ;

BX = (**BH**,**BL**) ;

CX = (**CH**,**CL**) ;

DX = (**DH**,**DL**)

They can also be considered as 8 8-bit registers. They are used to temporarily hold data. They are general registers, but can be used for specific operations. Example: AX = accumulator, CX = counter.

16 bits registers	High part 8 to 15 bits	Low part 0 to 7 bits	use
AX	AH	AL	Accumulator, multiplication, division
BX	BH	BL	Access memory
CX	CH	CL	Counter, repetition, offset
DX	DH	DL	In, out, multiplication, division
SI	/	/	Index source, lods, movs
DI			Index destination, stos, movs
BP,			Base pointer, stack
SP			Stack pointer, stack top

- **Pointer and index registers:** we have 4 registers (16-bit registers).

Pointers :

SP: Stack Pointer (the stack is an area where data is saved during program execution);

BP: Base Pointer, used to address data on the stack.

Index :

SI: Source Index ;

DI: Destination Index. They are used to transfer byte strings between two memory areas.

Note*: Pointers and indexes contain memory cell addresses.*

- **Segment registers:** we have 4 registers (16-bit registers).

CS: Code Segment register;

DS: Data Segment register;

SS: Stack Segment;

ES: Extra Segment, additional segment register for data;

Note: Segment registers, together with pointers and indexes, enable the 8086 microprocessor to address the entire memory.

❖ The 80x86 microprocessor contains 14 registers divided into 4 groups in addition to IP and IR:

A/ 32-bit registers: General registers are 32-bit in size, and the 8086's existing registers are still usable, but have been extended.

32 bits registers	High part 16 bits	Low part 16 bits	
		High part 8 to 15 bits	Low part 0 to 7 bits
EAX	AX	AH	AL
EBX	BX	BH	BL
ECX	CX	CH	CL
EDX	DX	DH	DL
ESI	SI	/	
EDI	DI		
EBP	BP		
ESP	SP		

B/ 64-bit registers

64 bits registers	High part 32 bits	High part 16 bits	Low part 16 bits	
			High part 8 to 15 bits	Low part 0 to 7 bits
RAX	EAX	AX	AH	AL
RBX	EBX	BX	BH	BL
RCX	ECX	CX	CH	CL
RDX	EDX	DX	DH	DL
RSI	ESI	SI	/	SIL
RDI	EDI	DI		DIL
RBP	EBP	BP		BPL
RSP	ESP	SP		SPL
R8	R8d	R8w		R8b
…..	….	….		….
R15	R15d	R15w		R15b

2/ Arithmetic and logical unit:

The ALU is made up of circuits whose purpose is to perform processing (calculations) on the operands under the control of the control unit.

IV/ Machine instruction format:

In fact, the microprocessor is only able to perform 3 types of operation:

- **LOAD r,[mem]**, i.e. load a data item from memory into a register r at an address supplied as a parameter
- **STORE [mem],r**, which stores a data item contained in a register r in memory at an address supplied as a parameter
- **OP r3, r2, r1**, where OP is an arithmetic or logical operation, i.e. set the result of r1 OP r2 in register r3.

In x86 microprocessors, only two operands are used in most instructions. Thus, **OP r1,r2** corresponds to **r1 = r1 OP r2**. In this case, operand **r1** is called **destination** and operand **r2** is called **source**.

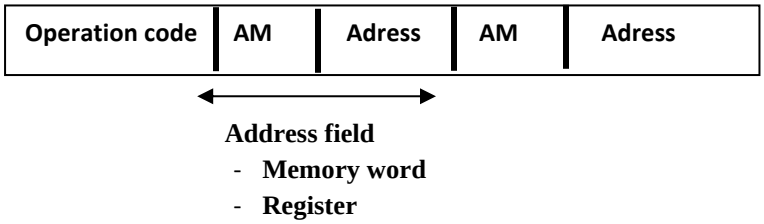
There are two classes of instructions, **RISC and CISC instruction sets**

- **RISC Reduced Instruction Set Computer**: the previous instruction format and memory addressing remain simple (i.e. there are few different ways of accessing memory).
- **CISC Complex Instruction Set Computer**: this type of architecture tends to combine a load or store instruction with a calculation, and memory addressing can be complex.

Example: *add [bx + cx * 4 + 8], ax*

V/ Addressing modes:

The addressing mode tells us how the instruction addresses its operands:



- **Memory word**
- **Register**

1/ Immediate addressing :

The operand is a constant contained in the address field.

Example: MOV AX, #100 ➞ load the AX register with the value 100.

2/ Direct addressing :

The operand is located in main memory, and the address field contains its actual address.

Example: MOV R1, 100 ➞ load the contents of memory word 100 into R1.

3/ Indirect addressing :

The address field contains the address of a pointer to memory.

Example: MOV R1, (R2) ➞ load into R1 the contents of an address located by the pointer whose address exists in R2.

4/ Indexed addressing :

The operand address is calculated by adding the contents of the address field to the contents of the index register (used in arrays).

Example 1: Add R1, (R2)+ // OR
R1 = R1 +M[R2]
R2 = R2 + d
Useful for stepping through arrays in a loop. R2 is the start of an array, d is the size of an element
We have also in indexed addressing tow modes: *incremented mode and decremented mode.*
Example 2: MOV AX, [SI +05]

5/ based addressing :

The contents of the base register + the contents of the address field.

Example: MOV AX, [BX + 04]

6/ relative addressing :

The contents of the ordinal counter + the contents of the address field (used in branching).

EA= PC + Address field value
PC= PC + Relative value.
PC is ordinal counter

7/ Addressing based on displacement:

```
mov ah,[bx+123h] ; <=> mov ah,[ds:bx+123h] => Opcode 8AA72301
```

8/ Indexed addressing with displacement:

```
mov ah,[di+123h] ; <=> mov ah,[ds:di+123h] => Opcode 8AA52301
```

9/ Based and indexed addressing:

```
mov ah,[bx+di] ; <=> mov ah,[ds:bx+di] => Opcode 8A21
mov [bp+si],ah ; <=> mov [ss:bp+si],ah => Opcode 8822
```

10/ Based and indexed addressing with displacement:

```
mov ah,[bx+si+123h] ; <=> mov ah,[ds:bx+si+123h] => Opcode 8AA02301
```

VI/ Executing an instruction:

For an instruction to be executed, its cycle passes through two stages:

- **The fetch cycle**, during which the instruction is fetched from memory and then decoded.
- **The execution cycle** during which the specified operation is performed by the ALU. The instruction then executes as follows:

1/ Instructions are placed in main memory in sequential addresses (except in the case of branching).

2/ The ordinal counter (instruction pointer) contains the address of the new instruction.

3/ A read command generated by the control unit reads the instruction from memory while sending its address to the address bus.

4/ The instruction is transferred on the data bus to the instruction register RI.

5/ The IP gives the address of the operand which is sent to the MC.

6/ The operation code is transmitted to the decoder, which determines the type of operation and transmits it to the sequencer (5, 6 are done at the same time).

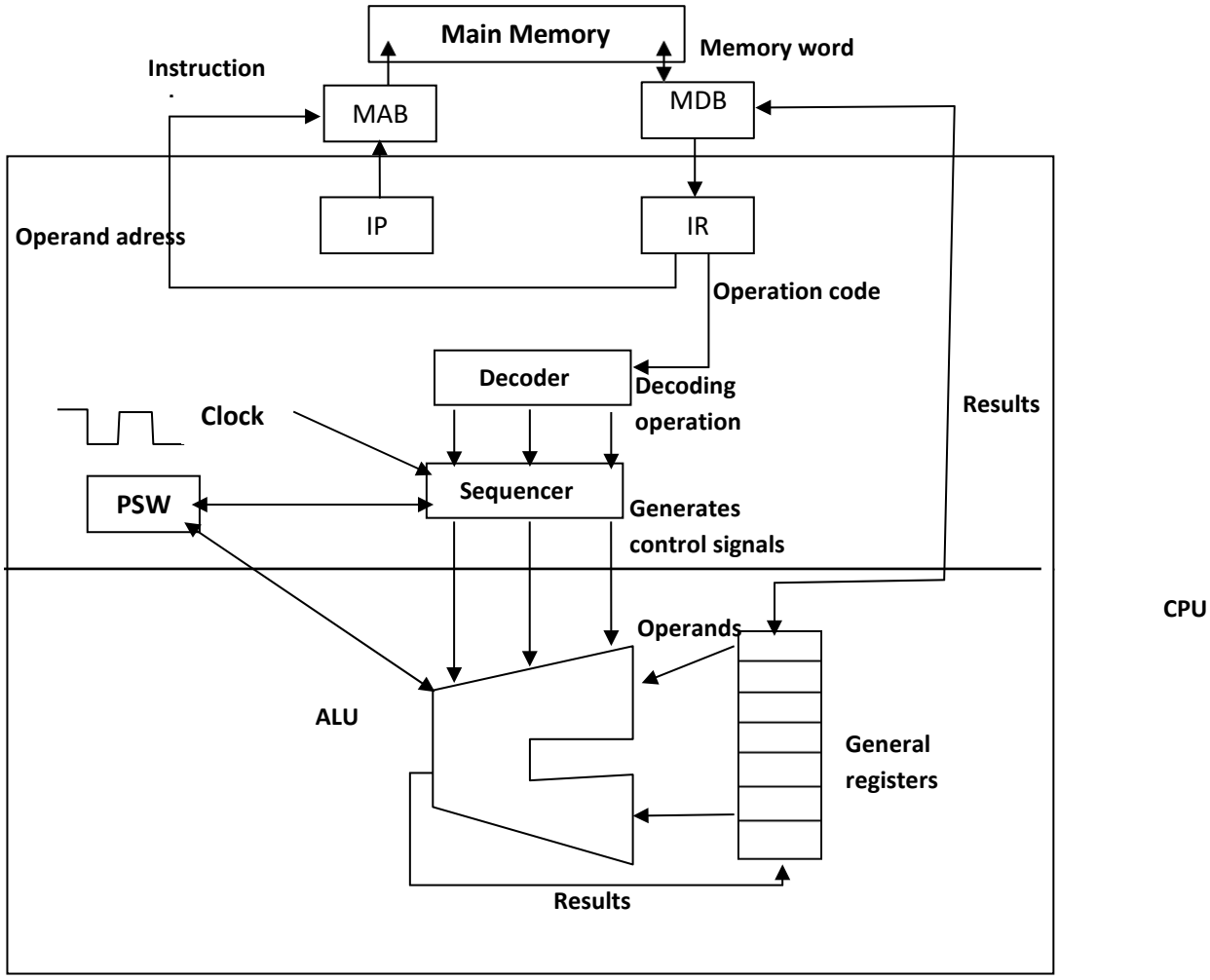
7/ The sequencer sends command signals to the memory to transmit the operand.

8/ The operand is transmitted to the ALU.

9/ The sequencer sends a command signal to the ALU to execute the selected operation.

10/ The IP is incremented.

11/ In the case of result storage, the sequencer sends a memory write command signal.



VII/ Interruptions:

There are two types of programs in a computer:

- User programs, which perform useful calculations.
- System programs, which supervise all events arriving at the machine.

These two types of programs share the processor in such a way as to take care of the user programs and consider the events produced.

The events produced in the machine are of two types:

- Program-related synchronized events, such as division by zero, attempted access to a forbidden zone, etc.
- hardware-related asynchronous events, such as end of I/O operation, clock signal, etc.

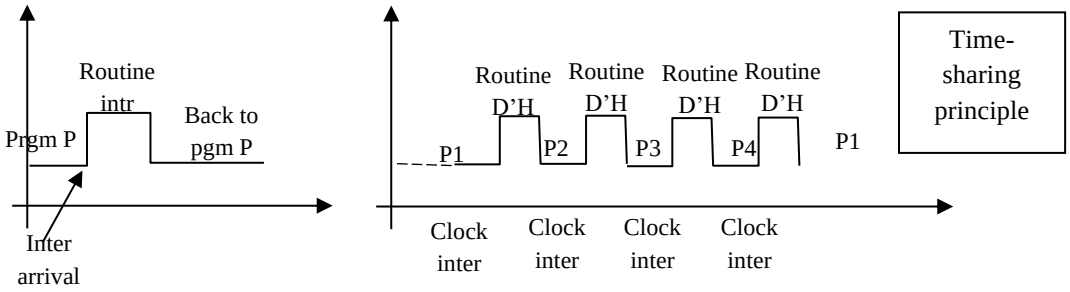
These events are the processor's way of continuously monitoring the state of a resource, and are referred to as interrupts.

1/ Definition:

An interrupt is a mechanism by which modules (I/O, memory, processes) can interrupt normal processor processing.

An interrupt is a response to an event that interrupts the execution of programs in progress at an observable point on the central processor. It takes the form of a signal sent to the processor, forcing it to suspend execution of the program in progress and triggering the execution of a predefined program called an "**interrupt routine**".

Example:



2/ Interrupt types:

There are two types of interruptions:

- External interruptions related to hardware.
- Internal program interruptions.

A/ External interrupts: caused by devices external to the processor, such as I/O units:

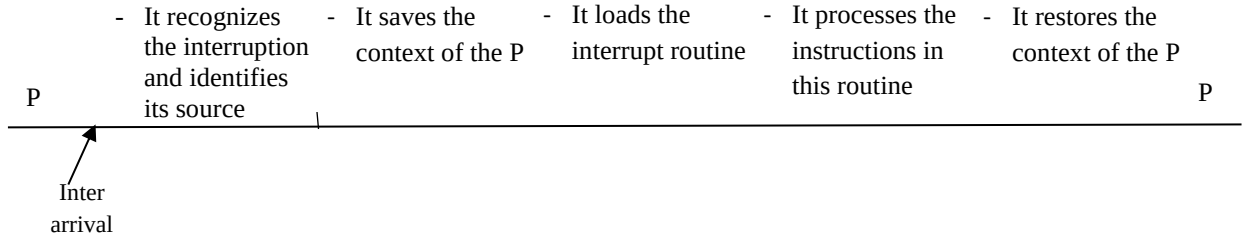
- A peripheral becomes ready.
- System reset.
- An error during I/O.
- End of I/O.

B/ Internal interrupts: which can be divided into two types:

- ✓ Clock interrupts: through these interrupts, the OS manages the execution time between all the processes in the system.
- ✓ Errors or traps: are caused by errors during program execution. A run must not be masked or delayed, for example:
  - Attempt to execute a prohibited or invalid operation.
  - Violation of access to a protected or non-existent area.
  - Division by zero.
  - Arithmetic overflow.
- ✓ Calls to supervisor: are caused by instructions calling system I/O functions, such as read instructions from keyboard, print instructions, ...

3/ Interruption management mechanism:

When a user process p is running, in time t, and the processor receives any interrupt with higher priority, then:



- ✓ **For an external interrupt**: the processor has an interrupt bit in the PSW register which is set to 1 by the hardware when an interrupt is triggered by a peripheral. The processor checks the value of this bit before starting the next instruction (so external interrupts are taken into account at the start of each instruction).
- ✓ **For an internal interrupt**: this occurs when an instruction is being executed (especially a trap or error):
  - - In the decoding stage (such as an attempt to execute a non-existent instruction, or a privileged instruction in slave mode).
  - - In the operand search stage (in the case of memory violation).
  - - In the operation initiation stage (in the case of overflow).

To identify the source of an interrupt, the technique commonly used is to consult a bit vector, where each bit is associated with a device.

Note: knowing that:

- ✓ **A program context** is all the information required to resume a process after it has been interrupted. It contains: its ordinal counter, its status word, the accumulator and the contents of the general registers.
- ✓ The operation of saving the context of one process and loading or restoring the context of another is called "**context switching**".

4/ Conditions for the arrival of an interruption:

For the processor to handle an interrupt immediately after its arrival, the following conditions must be met:

- The processor must be in an **observable point**, in other words, the code, which is being executed, must be a divisible code where it can be interrupted at any time and in any instruction. (In the opposite case, the code is said to be indivisible).
- The interruption system must be **active**, with the processor containing a mechanism for activating and deactivating the interruption system. It deactivates this system if it is executing interrupt-protected instructions (e.g. context-saving instructions).
- The interrupt must have a **higher priority** than the current program.
- The interrupt must be **unmasked**, where an interrupt can be masked because of its priority. When a hidden interrupt arrives, it will be delayed.

5/ Priorities between interrupts:

During a given period of time, there may be several simultaneous interrupt requests, and in order to process them fairly, the system defines a certain order of priority between interrupt levels. These priorities may be fixed or modifiable.