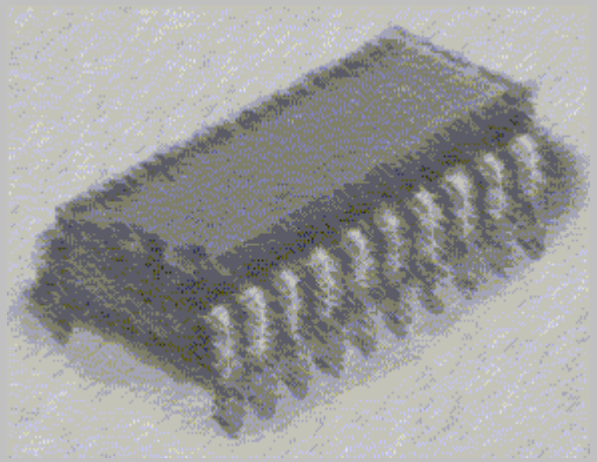
# Lesson 2. Data displaying on the screen. MS-DOS and ROM-BIOS services



***Computer Structure and Organization***

Graduate in Computer Sciences

Graduate in Computer Engineering



Automatic Department

# Contents

- I8086 flag register
- Transfer data instruction (ongoing)
- Control transfer instructions
- Arithmetic instruction: compare instruction
- Character strings representation: the ASCII code
- Interrupt
- Interrupt instructions
- MS-DOS interrupt services. INT 21h.
- ROM-BIOS interrupt services. INT 10h. INT 16h.

# I8086 flag register

- Flags register indicates the result of the previous instruction execution. Despite of a 16 bits register is used, not all of their bits has an specific meaning

### i8086 flag register

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | O | D | I | T | D | Z | | A | | P | | C |
| | O: overflow<br>D: direction<br>I: enabled interrupt<br>T: traps<br>S: signed | | | | | | | | Z: zero<br>A: auxiliary carry (BCD)<br>P: parity<br>C: carry | | | | | | |
| Shadowed bits are not used by i8086 | | | | | | | | | | | | | | | |

### Code View flag bits representation

| Flag | On (1) | Off (0) |
|---|---|---|
| Carry | CY | NC |
| Parity | PE | PO |
| Auxiliary carry | AC | NA |
| Zero | ZR | NZ |
| Signed | NG | PL |
| Interrupt | EI | DI |
| Direction | DN | UP |
| Overflow | OV | NV |

# Data transfer instruction

- **Mnemonic:** LEA
- **Format:** LEA target, source
- **Description:**

  Gets source effective address . Source segment address is stored in DS its offset in source

- **Example:**

  LEA DX, OPERANDO1

- **Using OFFSET directive :**

  MOV DX, OFFSET OPERANDO1

  It works in assembly time not in the execution one.

  Only for constant offsets.

# Control transfer instructions (I)

- Modifies normal sequence of program execution by changing program counter value (IP).

**Types of instructions:**

- **Jumps:**
  - **Unconditional:** JMP label( IP ← label)
  - **Conditional**: J{condition} label
    IF condition, IP ← label. IF not, IP ← next instruction
- **Routines calls:** (jumps with address ret
  - **Procedures**: CALL
  - **Interrupts**:
    - INT
      - Software
      - BIOS:
        - S.O.
        - Hardware

## Control transfer instructions (II)

- The most i80x86/8088 used conditions are:

J [N] { Z – Zero; E - Equal; C - Carry; S - Signed; O – Overflow }

J [N] { G – Great; L – Less; A - Above*; B - Bellow* } [E]

J [N] { P - Parity; PE - Parity even; PO – Parity odd; CXZ - CX=0 }

* Unsigned operands

## Control transfer instructions (and III)

- **Loops**: operation (IP decrement) + Conditional jump on operation result

- LOOP Label
  CX ← CX -1;
  IF CX ≠ 0 then IP← Label, if not IP ← next instruction

**Example:**

```
MOV CX, 4
Bucle:
    INC BX
    ADD BX, CX
    LOOP Bucle
```

## Compare instruction

- **Mnemonic:** CMP
- **Format:** CMP target, source
- **Description:**
  Compares source and target operands and properly modifies the flag register. It internally works by subtracting source and target operand. Operands are equal if the result is zero. Source is greater than target if the result is positive. And target is greater than source otherwise.
- **Examples:**
  - CMP AX, DX ; Compares AX and DX
  - CMP CL, 'A' ; Compares CL and A ASCII code
  - CMP DL, [BX] ; Compares DL and pointed by BX memory
    ; position content

## Character string representation (I)

**Alphanumerical representations:**

- Coded with different number of bits (6, 7, 8, 16) each character to be represented.

- Alphanumerical codes examples:
  - 6 bits (64 available characters) Fieldata and BCDIC
  - 7 bits (128 available characters) ASCII
  - 8 bits (256 available characters) extended ASCII and EBCDIC
  - 16 bits (65536 available characters) UNICODE

## Character string representation (II)

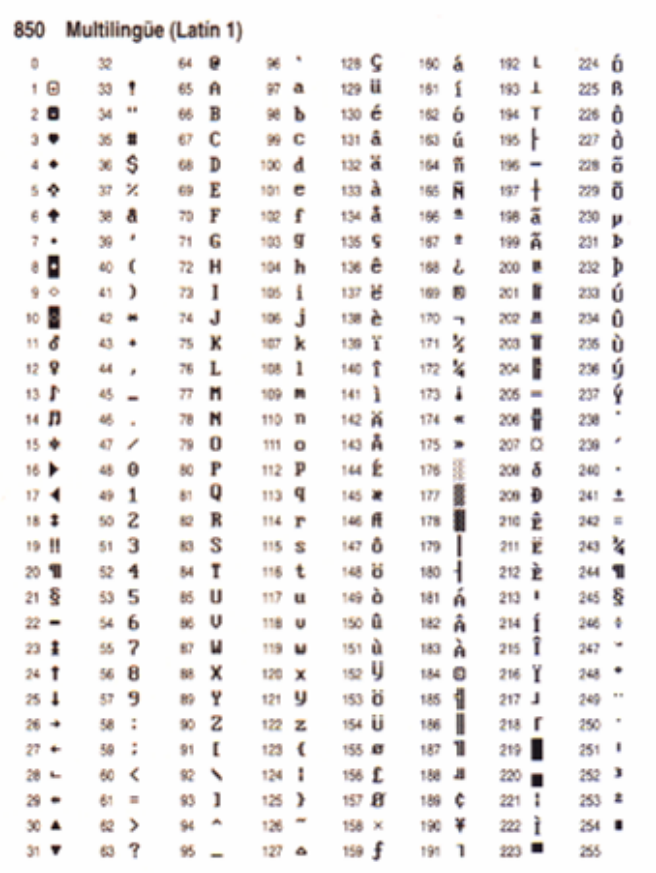
- Sentences are formed by joining characters:

- **Fixed length strings:**
  A maximum length is defined for all strings


- **Variable length strings:**
  - Delimiter character


  - Explicit length

## Character string representation (and III)




0...32 → Control


## Interrupts (I)

- Interrupts are sent signals to the CPU to ask for the current instruction execution be suspended and to accept a specific request.

- Interrupts types:
  - **Hardware interrupts:** are generated by microprocessor circuits and related to an event like pressing a key of the keyboard.
    - **Masked interrupts.** Can be disabled.
    - **Unmasked interrupts.** Cannot be disabled. Belong to emergency situations like, parity error, overflow, …
  - **Software interrupts:** are generated by program sentences and provide different services.

## Interrupts (and II)

- How to call an interrupt service:
  - Identify needed interrupt
  - Prepare parameters
  - Choose required number interrupt function
  - Call the interrupt

- Interrupt calls mean a jump to another memory code area and require to store IP and Flags registers.

## Interrupt instructions (I)

- **Mnemonic:** INT
- **Format:** INT interrupt_type
- **Description:**
  INT jumps to specified interrupt address. i8086 interrupt addresses are calculated by multiplying interrup_type by 4 . Interrupt_type is an usnsigned number between 0 and 255.

  Interrupt array is composed of two words: offset is the first one and segment address the second word
- **Example:**
  - INT 21h

## Interrupt instructions (and II)

- **Mnemonic:** IRET
- **Format:** IRET
- **Description:**
  Flag register is restored and IP gets the returning address from the top of the stack. It's used to finish the end of an interrupt procedure.
- **Example:**
  - IRET

## MS-DOS interrupt services (I)

| INT 21h | |
|---|---|
| **AH** | **Function** |
| 01h | Program waits till a key is depressed on the keyboard. Read character ASCII code is returned on AL and echoed on the screen |
| 02h | Write a single character on the screen. Character ASCII code must be stored on DL. |
| 08h | Program waits till a key is depressed on the keyboard. Read character ASCII code is returned on AL without echoed on the screen |
| 09h | Display a memory stored string character on the screen. Stricng must end with $. String address must be stored on DS:DX. |
| 0Ah | Read a string character from keyboard and store it on memory. Memory storage address must be stored in DS:DX. |

## MS-DOS interrupt services (II)

### Function 1h

- Function number must be stored on AH
- Returns the ASCII code of the read character on AL
- Echoed read character on the screen

**In code segment**

```
mov ah, 01h
int 21h
```

### Function 8h

- Function number must be stored on AH
- Returns the ASCII code of the read character on AL
- Read character is not displayed on the screen. It's an useful function to read passwords from keyboard.

**In code segment**

```
mov ah, 08h
int 21h
```

## MS-DOS interrupt services (III)

### Function 2h

- Function number must be stored on AH
- Desired character ASCII code must be stored on DL
- Displayed ASCII code is returned on AL
- There are two ways of passing the ASCII code

**In code segment**

```
mov ah, 02h
mov dl, 'A'
int 21h
```

```
mov ah, 02h
mov dl, 41h
int 21h
```

## MS-DOS interrupt services (IV)

### Function 0Ah

- Function number must be stored on AH
- DS:DX store offset and segment address of the string
- Next buffer structure must be defined in data segment:
  - Maximum number of character to be read plus 1 to store ENTER
  - One byte in which the number of read characters will be returned
  - So many bytes such as wished character to be read Usually DUP directive is used to define them.
  - One more byte to store ENTER key

**In data segment**

```
Cadena DB 9,?,?,?,?,?,?,?,?,?,?
```

**Definition alternative:**

```
Cadena DB 9,?,9 DUP(?)
```

**In code segment**

```
mov ah, 0Ah
lea dx, cadena
int 21h
```

## MS-DOS interrupt services (VI)

### Function 09h

- Function number must be stored on AH
- DS:DX store offset and segment address of the string to be displayed
- String must en with $ character and be defined on data segment

**In data segment**

```
Cadena DB 'Hello world$'
```

**In code segment**

```
mov ah, 09h
lea dx, cadena
int 21h
```

## MS-DOS interrupt services (and VII)

### Function 4Ch

- Function number must be stored on AH
- ERRORLEVEL number to be returned must be stored on AL
- Returned ERRORLEVEL can be processed by MS-DOS using:
  IF ERRORLEVEL n action

**In code segment**

```
mov ah, 4Ch
mov al, 1
int 21h
```

# ROM-BIOS

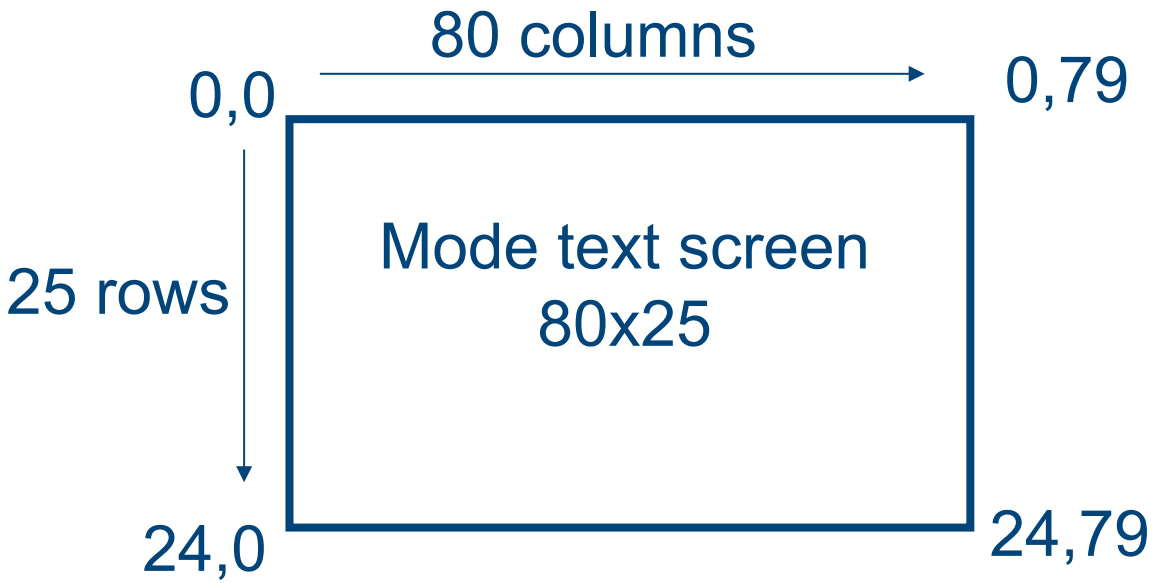
- Direct access to handle with hardware services
- Interrupt calls are needed to use such services
- Basic Input Output Services (E/S) we will use are:
  - INT 10H: Video services (output on the screen)
  - INT 16H: Keyboard services (input form keyboard)

# Video services summary. BIOS

| INT 10h | |
|---|---|
| **AH** | **Función** |
| 00h | Set video mode |
| 02h | Postion the cursor on text mode screen using specified coordinates |
| 06h | Window scroll up |
| 07h | Window scroll down |
| 09h | Write character and atribute in current position. |
| 0Ah | Write character and last atribute in current position |
| 0Eh | Teletype writing character mode (write a character and cursor is positioned on next colum) |

# BIOS video output. Interrupt 10H. (I)

- **Function 00h:** Set video mode.
- Video mode must be stored on AL.
- Some of the vide modes are:
  - AL = 1 40x25 Text 16 colours
  - AL = 3 80x25 Text 16 colours




**In code segment**

```
mov ah, 0
mov al, 3          ;80x25
int 10h
```

# BIOS video output. Interrupt 10H. (II)

- **Function 02h**: Position the cursor in specified coordinated on text mode screen.
- Required parameters to be passed:
  - DH = Row (0...24)
  - DL = Column (0...39 / 79 depend on video mode)
  - BH = Page(0...3 in video mode 1, 0...7 in video mode 3) Default page is 0.

**In code segment:**

```
mov ah, 2
mov dh,12       ;Row12
mov dl,20       ;Column 20
mov bh, 0       ;Page 0
int 10h
```

# BIOS video output. Interrupt 10H. (III)

- **Functions 06h / 07h**: window scroll up (06h) / scroll down(07h)
- Required parameter are:
  - AL = number of lines to scroll.  Screen is cleared if AL = 0
  - CH = upper left corner row
  - CL = upper left corner column
  - DH = lower right corner row
  - DL = lower right corner column
  - BH = Attribute to use on scrolled / cleared area ( e.g.: 07h=white on black)

  This function is used to clear screen by using: AL=0, in text mode 80x25 rows will be 0,0 y 24,79

→ Continues

# BIOS video output. Interrupt 10H. (IV)

**Character attributes:**

| Attribute byte on 0, 1, 2 and 3 video modes | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | P | Background | | | I | Foreground | | |

P = background intensity or blinking frequency

I = Foreground intensity

| P or I bit values used by default | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Black | Blue | Green | Cyan | Red | Magenta | Brown | White | Grey | Soft blue | Soft green | Soft cyan | Soft red | Soft Magenta | Yellow | Hard intenso |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |

# BIOS video output. Interrupt 10H. (V)

**In segment code**

```
;Scrolls up a 3x3 area
;Fills with blank spaces cleared area
mov ah, 6
mov ch, 10        ;Upper left row corner 10
mov cl, 20        ;Upper left column corner 20
mov dh,12         ;Lower right row corner 12
mov dl,22         ;Lower right column corner 22
mov bh, 7         ;White on black attribute
mov al, 3         ;Number of lines to be scrolled
int 10h
```

# BIOS video output. Interrupt 10H. (VI)

- **Function 09h**: writes attribute and character on current cursor position once or more times.
- Required parameters to be passed are:
  - AL = ASCII code of the wished character
  - BH = Page (0 to use active page)
  - BL = Attribute (e.g.: 07h=normal, 0Fh=high intensity)
  - CX = number of times to be written (repetition factor)

**In code segment**

```
mov ah, 9
mov bh, 0       ;Page0
mov bl, 7       ;White on black attribute
mov cx, 4       ;Display four times stored
mov al,'A'      ;on AL character 'A'
int 10h
```

# BIOS video output. Interrupt 10H. (VII)

- **Function 0Ah**: writes a character in current cursor position with previous attribute.
- Next parameters are required:
  - AL = Character ASCII code to be written
  - BH = Page (0 is active page)
  - CX = number of times to be written (repetition factor)

**In code segment:**

```
mov ah, 0Ah
mov bh, 0       ;Page 0
mov cx, 1       ;Display only once 'z'
mov al,'z'      ;character stored on AL
int 10h
```

# BIOS video output. Interrupt 10H. (and VIII)

- **Function 0Eh**: Writes a character on the screen and positions cursor on next column (teletype mode)
- Required parameters are:
  - AL = Character ASCII code to write
  - BH = Page (0 is active page)

**In code segment**

```
mov ah, 0Eh
mov al, 'H'     ; Display 'H' character and
int 10h         ; cursor is positioned on next
                ; column
```

# Keyboard services summary. BIOS. (1)

| INT 16h | |
|---|---|
| **AH** | **Function** |
| 00h / 10h(1) | Read a character from buffer keyboard. Wait till a key is depressed if buffer is empty |
| 01h / 11h(1) | Return buffer state:<br>ZF = 1 if buffer is empty<br>ZF = 0 if a key is on buffer |
| | (1) Expanded keyboards (keys F1 to F12, arrow keys, etc.) |

# BIOS keyboard services. Interrupt 16h. (I)

- **Function 00h (or 10h(1)).** Read a character from keyboard buffer. Wait till a key is depressed if buffer is empty.
- Read character ASCII code is returned on AL and key code on AH.

(1) Expanded keyboards (keys F1 to F12, arrow keys, etc.)

**In data segment:**

```
Caracter_leido DB ?
Codigo_ident   DB ?
```

**In code segment:**

```
mov ah, 0
int 16h
mov Caracter_leido, al
mov Codigo_ident,ah
```

# BIOS keyboard services. Interrupt 16h. (and II)

- **Function 01h (or 11h[1]).** Returns buffer state. Read character is not removed from keyboard buffer. To flush buffer functions 00h or 10h must be used.
- Returns:
  - ZF = 1 if buffer is empty.
  - ZF = 0 if there is a key on buffer, then read character ASCII code is returned on AL and key code on AH.

**In code segment:**
Function 01h (int 16h) is used to implement an infinite loop till any key is depressed.

```
Lazo:
        ;inside loop code
        mov ah, 1
        int 16h
jz Lazo    ;If no key is depressed repeat
mov ah,0 ;Function 0h is used to flush
int 16h    ;the keyboard buffer
```