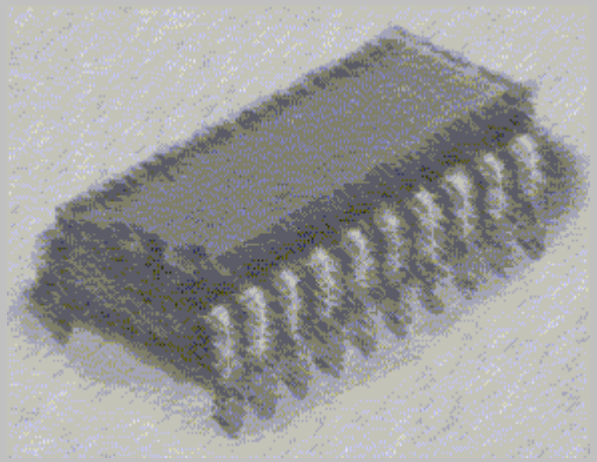
# Lesson 5. Instruction formats. Directives, string instructions and several modules programs

***Computer Structure and Organization***

Graduated in Computer Sciences /
Graduated in Computer Engineering



Computer Structure and Organization
Graduated in Computer Sciences /
Graduated in Computer Engineering

Automatic Department

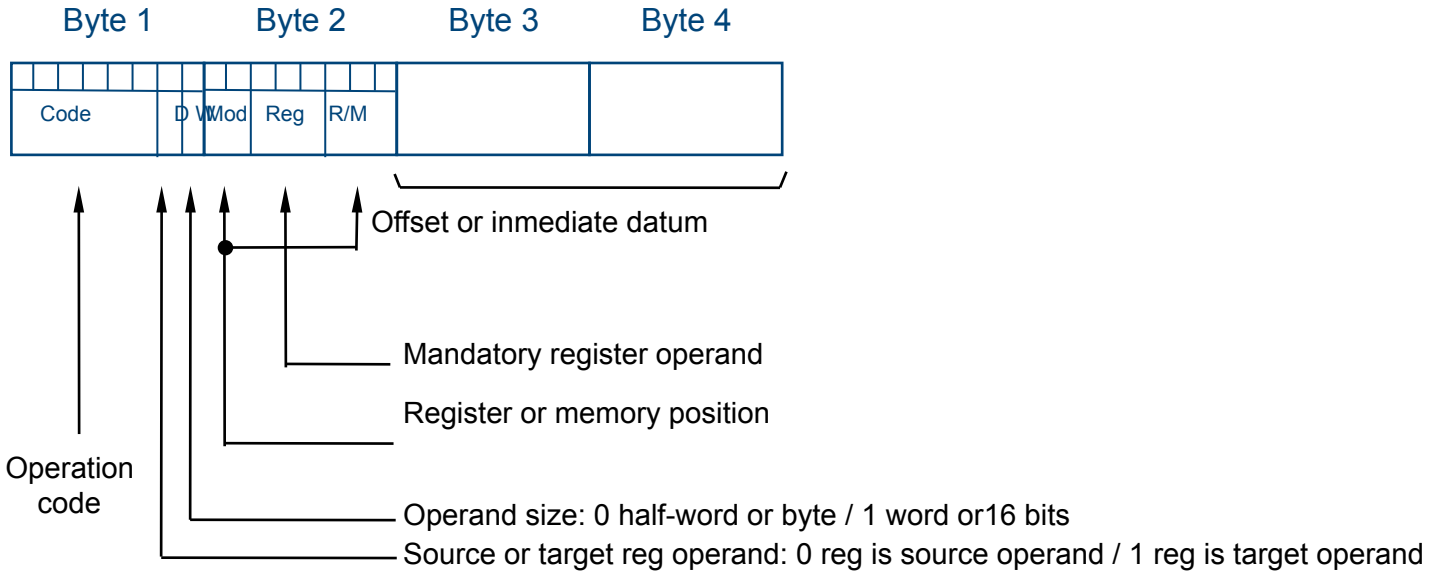
## Contents

- i8086 instruction formats
- String instructions
- Directives
- Assembling several modules programs

# i80x86/8088 instruction formats (I)

- i80x86/88 have several instruction format types
- Format sizes vary from one byte to six bytes code length

## Formato de las instrucciones registro-registro y registro-memoria:



# i80x86/8088 instruction formats (II)

- **Instruction format first byte contains following information:**
  - Operation code (6 bits)
  - Direction bit register (D):
    - IF D = 1 field REG = target operand
    - IF D = 0 field REG = source operand
  - Data size bit (W): specifies is the instruction uses half word or word size operands:
    - IF W = 0 size is 8 bits (or 16 bits)
    - IF W = 1 size is 16 bits (or 32 bits)

# i80x86/8088 instruction formats (III)

- **Operands are contained on the second byte** (It's mandatory that one of them is a register)
- **REG** table is used to code desired register

| REG | W=0 | W=1 |
|---|---|---|
| 000 | AL | AX |
| 001 | CL | CX |
| 010 | DL | DX |
| 011 | BL | BX |
| 100 | AH | SP |
| 101 | CH | BP |
| 110 | DH | SI |
| 111 | BH | DI |

- **MOD** specifies the addressing mode

| MOD | Function: |
|---|---|
| 00 | Memory addressing mode without offset |
| 01 | Memory addressing mode with a half-word size offset |
| 10 | Memory addressing mode with one word size offset |
| 11 | Register to register addressing mode |

# i80x86/8088 instruction formats (and IV)

- **Operands are contained on the second byte** (*ongoing*):
- Depending on MOD field, **R/M** is used to identify one register or a memory postion according to the bellow table
- O8 means half-word offsset (8 or 16 bits) and O16 indicates one word offset (16 bits or 32 bits)

| MOD = 11 | | | EFFECTIVE MEMORY ADDRESS | | | |
|---|---|---|---|---|---|---|
| R/M | W=0 | W=1 | R/M | MOD = 00 | MOD = 01 | MOD = 10 |
| 000 | AL | AX | 000 | [BX]+[SI] | [BX]+[SI]+O8 | [BX]+[SI]+O16 |
| 001 | CL | CX | 001 | [BX]+[DI] | [BX]+[DI]+O8 | [BX]+[DI]+O16 |
| 010 | DL | DX | 010 | [BP]+[SI] | [BP]+[SI]+O8 | [BP]+[SI]+O16 |
| 011 | BL | BX | 011 | [BP]+[DI] | [BP]+[DI]+O8 | [BP]+[DI]+O16 |
| 100 | AH | SP | 100 | [SI] | [SI]+O8 | [SI]+O16 |
| 101 | CH | BP | 101 | [DI] | [DI]+O8 | [DI]+O16 |
| 110 | DH | SI | 110 | direct mem. pos. | [BP]+O8 | [BP]+O16 |
| 111 | BH | DI | 111 | [BX] | [BX]+O8 | [BX]+O16 |

## String instructions (I)

- **Mnemonic:** MOVS / MOVSB / MOVSW (MOV BYTE / WORD STRING)
- **Format:** MOVS target, source
  MOVSB / MOSW
- **Description:** Pointed DS:SI byte or word is transferred to the pointed ES:DI byte or word. SI and DI will be incremented / decremented in the number of transferred bytes according to DF value ((DF=0 → increment; DF=1 → decrement). Instruction REP is used as repetition factor.
- **Operands:** MOVS uses ES:DI as target and DS:SI as source by default.

- **Examples:**
  - movsb
  - movs tabla1, tabla2

## String instructions (II)

- **Mnemonic:** LODS / LODSB / LODSW (Load BYTE / WORD to AL / AX)
- **Format:** LODS source
  LODSB/LODSW
- **Description:** get a byte or word from a string. Transfers one byte or word from the pointed string by DS:SI to AL or AX register. SI will be incremented / decremented in the number of transferred bytes according to DF value ((DF=0 → increment; DF=1 → decrement).
- **Operands:** LODS uses DS:SI as source by default.

- **Examples:**
  - lods cadena
  - lodsb

## String instructions (III)

- **Mnemonic:** STOS / STOSB / STOSW (Store BYTE / WORD from AL / AX)
- **Format:** STOS target
  STOSB/STOSW
- **Description:** stores one byte or word on a string. Transfers one byte (AL) or one word (AX) to pointed ES:DI string. DI will be incremented / decremented in the number of transferred bytes according to DF value ((DF=0 → increment; DF=1 → decrement). Instruction REP is used as repetition factor
- **Operands:** STOS uses ES:DI as target by default.

- **Examples:**
  - stos cadena
  - stosw

## String instructions (IV)

- **Mnemonic:** CMPS / CMPSB / CMPSW (Compare BYTE / WORD)
- **Format:** CMPS target, source
  CMPSB/CMPSW
- **Description:** Strings bytes or word are compared. Pointed by DS:SI (source string) and ES:DI (target string) will be compared. SI and DI will be incremented / decremented in the number of compared bytes according to DF value ((DF=0 → increment; DF=1 → decrement). Instruction REP is used as repetition factor.
- **Operands:** CMPS uses ES:DI as target and DS:SI as source by default.
- **Examples:**
  - cmps cadena1, cadena2
  - cmpsb
  - cmpsw

## String instructions (V)

- **Mnemonic:** SCAS / SCASB / SCASW (Scan BYTE / WORD)
- **Format:** SCAS target
  SCASB/SCASW
- **Description:** searches a byte or word among string positions. AL is used for byte size strings and AX for word size one. Result is not stored but flags are modified. DI will be incremented / decremented in the operand size according to DF value ((DF=0 → increment; DF=1 → decrement). Instruction REP is used as repetition factor.
- **Operands:** target se emplean como referencia, pues por defecto SCAS emplea ES:DI como target.
- **Examples:**
  - scas cadena
  - sacsb

## String instructions (and VI)

- **Mnemonic:** REP / REPZ / REPE / REPNZ / REPNE (Repeat)
- **Format:** REP / REPZ / REPE / REPNZ / REPNE instrucción de cadena
- **Description:** precedes string instructions to repeat action according to CX value (from CX to cero) or till Zero flag changes. This repetition can be unconditional (REP) or conditional (REPZ / REPE till ZF=1 or REPNZ / REPNE till ZF=0).
- **Operands:** hasn't any operand
- **Restrictions:** for string instructions only
- REP is used with MOVS, LODS and STOS; REPE, REPZ, REPNE and REPNZ are used with CMPS and SCAS

- **Example:**
  - rep movsb

## Directives (I)

- Directives are instruction to indicate what the assembly program must do.
- Directives are used to reserve memory storage space, to name program variables, to build data structures, etc.
- Directives can be classified into:
  - Data directives
  - Conditional directives
  - Listing directives
  - Macros directives

# Directives (II)
# Data directives (I) Symbol definition (I)

- **Mnemonic:** EQU (EQUIVALENT)
- **Format:** name EQU expression
- **Description:** assignes a name to the value of an expression, This name cannot be redefined.
- **Example:**
  - columnas EQU 80

# Directives (III)
# Data directives (II) Symbol definition (and II)

- **Mnemonic:** =
- **Format:** name= expression
- **Description:** asignes a name to a value expression. Name can be redefined. It's useful for macros.
- **Examples:**
  - valor = 10
  - valor = valor + 1

# Directives (IV)
# Data directives (III) Data definition (I)

- **Mnemonic:** DB (DEFINE BYTE)
- **Format:** [variable_name] DB expression
- **Description:** memory storage reservation for a byte datum and following ones. “variable_name” is optional and will be the assigned name to the first byte of the arrange.

- **Operands:** “expresion” is the first value of the variable. It can be:
  - Positive or negative expression or contant (-128 ≤ expression ≤127 signed).
  - Undefined value by using “?” symbol.
  - Character string delimited by single or double quotes.
  - n1 DUP (n2) that means to repeat n1 as meny times as n2 indicates.
- **Examples:**
  - valores DB 30, -15, 20 - Datos DB 10 DUP(0)
  - DB 12*3 - cadena DB “Hola mundo”

# Directives (V)
# Data directives (IV) Data definition (II)

- **Mnemonic:** DD (DEFINE DOUBLE)
- **Format:** [variable_name] DD expression
- **Description:** memory storage reservation for a 32 bits datum and following ones. “variable_name” is optional and will be the assigned name to the first byte of the arrange.

- **Operands:** “expresion” is the first value of the variable. It can be:
  - Positive or negative expression or contant.
  - Undefined value by using “?” symbol.
  - A full memory address (segment and offset).
  - n1 DUP (n2) that means to repeat n1 as meny times as n2 indicates.
- **Examples:**
  - valores DD 300, -150, 2000
  - DD 120*3
  - Direc DD tabla

# Directives (VI)
# Data directives (V) Data definition (III)

- **Mnemonic:** DQ (DEFINE QUADWORD)
- **Format:** [variable_name] DQ expression
- **Description:** memory storage reservation for a 64 bits datum and following ones. “variable_name” is optional and will be the assigned name to the first byte of the arrange.

- **Operands:** “expresion” is the first value of the variable. It can be:
  - Positive or negative expression or contant.
  - Undefined value by using “?” symbol.
  - n1 DUP (n2) that means to repeat n1 as meny times as n2 indicates.

- **Examples:**
  - valores DQ 300, -150, 2000
  - DQ 120*3
  - datos DQ 4 DUP (0) ; Equivale a DQ 0, 0, 0 , 0

# Directives (VII)
# Data directives (VI) Data definition (IV)

- **Mnemonic:** DT (DEFINE TENBYTE)
- **Format:** [variable_name] DT expression
- **Description:** ten bytes storage space are reserved to store packe BCD digits. First byte is reserved to store sign number 00h (positive) 80h (negative. 9 remaining bytes are used to store 18 decimal numbeers. “variable_name” is optional and will be the assigned name to the first byte of the arrange.
- **Operands:** “expresion” is the first value of the variable. It can be:
  - Positive or negative expression or contant.
  - Undefined value by using “?” symbol.
  - n1 DUP (n2) that means to repeat n1 as meny times as n2 indicates.

- **Examples:**
  - valores DT 0123456789
  - negat DT -0123456789

# Directives (VIII)
# Data directives (VII) Data definition (and V)

- **Mnemonic:** DW (DEFINE WORD)
- **Format:** [variable_name] DW expression
- **Description:** memory storage reservation for a 16 bits datum and following ones. “variable_name” is optional and will be the assigned name to the first byte of the arrange.
- **Operands:** “expresion” is the first value of the variable. It can be:
  - Positive or negative expression or contant.
  - Undefined value by using “?” symbol.
  - An offset.
  - n1 DUP (n2) that means to repeat n1 as meny times as n2 indicates.

- **Examples:**
  - valores DW 300, -150, 2000
  - DW 120*3

## Directives (IX)
## Data directives (VIII). External references (I)

- **Mnemonic:** PUBLIC
- **Format:** PUBLIC symbol
- **Description:** specified symbols can be accessed by other modules during link modules operation (LINK). To use these symbols, EXTRN senteces must be used in the modules that want to access it.
- **Example:**
  - PUBLIC dato
  - dato DB 23h

## Directives (X)
## Data directives (IX). External references (and II)

- **Mnemonic:** EXTRN
- **Format:** EXTRN symbol:type
- **Description:** identifies symbols which were defined in other modules by using PUBLIC.
- **Example:**
  - EXTRN dato:byte

# Directives (XI)
# Data directives (X). Segment definitions (I)

- **Mnemonic:** SEGMENT
- **Format:** name SEGMENT [allignment] [combination ]
- **Description:** indicates the begining of the “name” segment.

- **Alignment (optional):**
  - BYTE
  - WORD
  - PARA
  - PAGE

- **Example:**
  - datos SEGMENT
  - datos ENDS

- **Combination (optional):**
  - PUBLIC
  - COMMON
  - AT
  - STACK
  - MEMORY

## Directives (XII)
## Data directives (XI). Segment definitions (II)

- **Mnemonic:** ENDS (END SEGMENT)
- **Format:** segment_name ENDS
- **Description:** states “segment_name” ends or “structure_name” ends.
- **Operands:** segment_name is mandatory

# Directives (XIII)
# Data directives (XII). Segment definitions (and III)

- **Mnemonic:** ASSUME
- **Format:** ASSUME segment_register:segment_name
- **Description:** states each segment name to be use by assembler program..
- **Operandos:** “segment_register”: DS, CS, SS or ES
  “segment_rname”: defined by SEGMENT directive.
- **Example:**
  - ASSUME CS:codigo
  - ASSUME ES:@DATA

# Directives (XIV)
# Data directives (XIII) Procedure definitions (I)

- **Mnemonic:** PROC (PROCEDURE)
- **Format:** procedure_name PROC [atributte]
- **Description:** estiblishes the begining of procedure_name declaration
- **Operands:** atributte = NEAR or FAR (NEAR by default)
- **Example:**
  - rutina PROC
  - rutina ENDP

## Directives (XV)
## Data directives (XIV) Procedure definitions (and II)

- **Mnemonic:** ENDP (END PROCEDURE)
- **Format:** procedure_name ENDP
- **Description:** stats the end of the procedure
- **Operand:** procedure_name is mandatory

## Directives (XVI)
## Data directives (XV) Macro definitions (I)

- **Mnemonic:** MACRO
- **Format:** macro_name MACRO parameters_list
- **Description:** specifies the name and the list of parametrs to be uses by the macro. Parameters are separated by commas.
- **Example:**
  - add3 MACRO operando1, operando2, resultado
  - *macro body*
  - ENDM

## Directives (XVII)
## Data directives (XVI) Macro definitions (II)

- **Mnemonic:** ENDM (END MACRO)
- **Format:** ENDM
- **Description:** states macro end.

# Directives (XVIII)
# Data directives (XVII) Macro definitions (and III)

- **Mnemonic:** LOCAL
- **Format:** LOCAL label
- **Description:** indicates the labels to be changed in macro expansion. LOCAL can be used in macro definition only.
- **Example:**

```
Delay MACRO number
    LOCAL goon
    mov cx, number
    goon:    loop goon
ENDM
```

# Directives (XIX)
# Data directives (XVIII). Block definitions

- **Mnemonic:** STRUC (STRUCTURE)
- **Format:** structure_name STRUC
- **Description:** defines new data type. Memory storage is not reserved by definition. To access to each field: structure_name.field.
- **Operands:** estándar data directives DB, DW, DD, DQ and DT.

- **Example:**

```
parametros      STRUC
p1              dw ?
p2              db ?
parametros      ENDS
```

## Directives (XX)
## Data directives (XIX). MASM control (I)

- **Mnemonic:** END
- **Format:** END [expression]
- **Description:** states end source file. “expresión” operand indicates begining source program address. If several modules are used only main module can use this expression.
- **Example:** END begin

## Directives (XXI)
## Data directives (and XX). MASM control (and II)

- **Mnemonic:** .RADIX
- **Format:** .RADIX expression
- **Description:** establishes number base by default. Expression is allwais expressed in 10 base.
- **Example:** .RADIX 16

## Directives (XXII)
## Conditional directives

- Assembly program ignore or not some portions of the source file.

```
IFxxx [condition]
...
ELSE
...
ENDIF
```

- ELSE statement is optional

- **Example:**
  - PRUEBA = 0
  - IF PRUEBA EQ 0
  - ... ; *Instructions used in test only*
  - ...
  - ENDIF

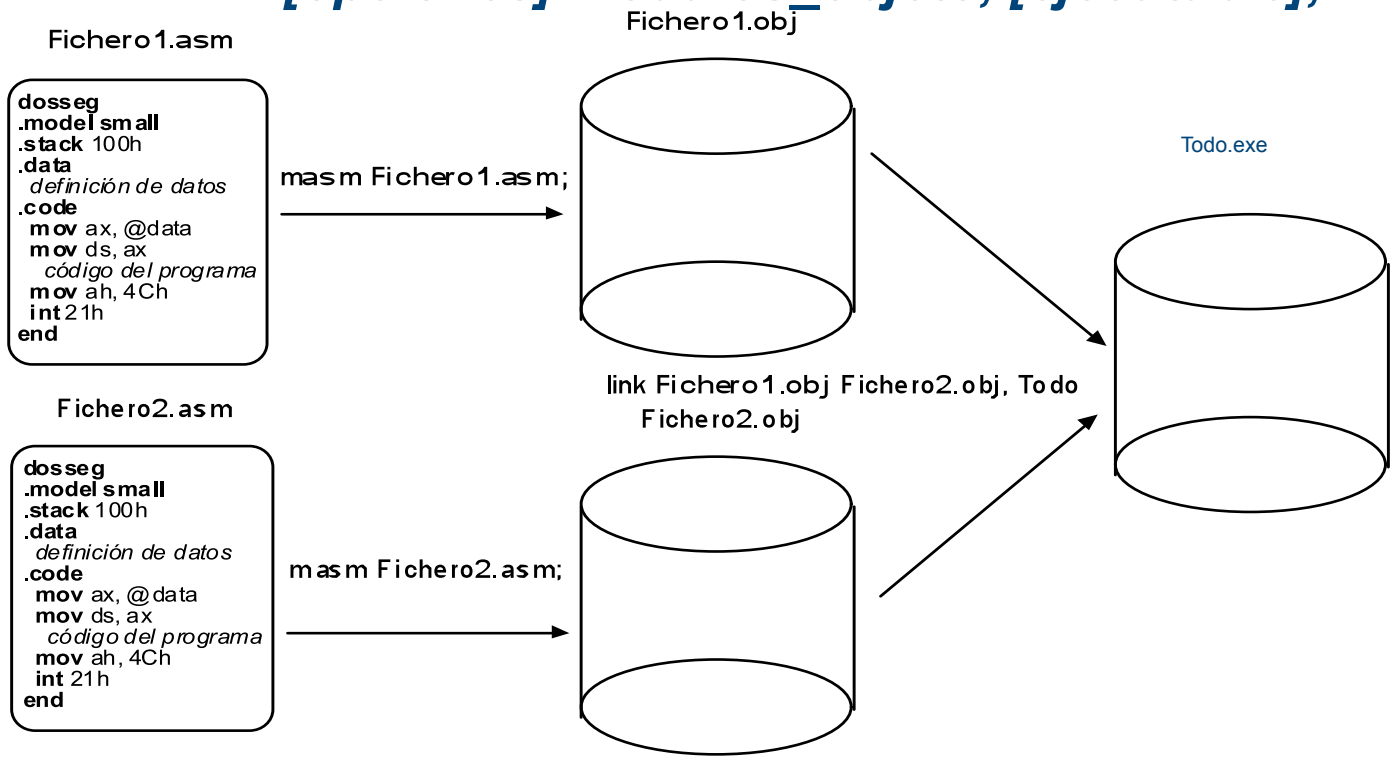
## Directives (and XXIII)
## Listing directives

- What to print and printing format
- Listing directives are classified into:
  - **List format:**
    - PAGE, TITLE, SUBTTL
  - **Macro list:**
    - .LALL: list macros and their expanssions
    - .SALL: macros and their expanssions are not printed
  - **Remarks:**
    - COMMENT
  - **Menssages:**
    - %OUT: output a message during assembly process

## Several modules assembling (I)

- EXTRN and PUBLIC directives are used to share data information and definition between different modules during assembling and linking processes
- Link syntax when individual OBJ are created is:

***LINK [opciones] módulos_objeto, [ejecutable];***



# Several modules assembling (and II)

**Program 1**

```
Extrn texto2: byte
Extrn leer: far
Public variable
Dosseg
.model small
.stack 100h
.data
 texto db 'Pulsa tecla$'
 variable db ?
.code
  inicio: mov ax, @data
  mov ds, ax
  lea dx, texto
  mov ah, 9
```

**Program 1 (ongoing)**

```
int 21h
call leer
lea dx, texto2,
mov ah, 9
int 21h
Mov dl, variable
Mov ah, 2
Int 21h
mov ah,4Ch
int 21h
End inicio
```

**Program 2**

```
Extrn variable: byte
Public texto2
Public Leer
Dosseg
.model small
.stack 100h
.data
 texto2db 'Has pulsado$'
.code
Leer proc far
  mov ax, @data
  mov ds, ax
  mov ah, 1
  int 21h
```

**Program 2 (ongoing)**

```
mov variable, al
ret
Leer endp
End
```

**Assembling and linking the program**

Masm prg1;

Masm prg2

Link prg1 prg2, todo