Introducción al 8086

Facultad de Ingeniería

Universidad de la República

Instituto de Computación

Contenido

- Generalidades 80x86
- Modos de direccionamiento
- Set de instrucciones
- Assembler
- Compilando algunos ejemplos

Microprocesadores Intel 8086/8088

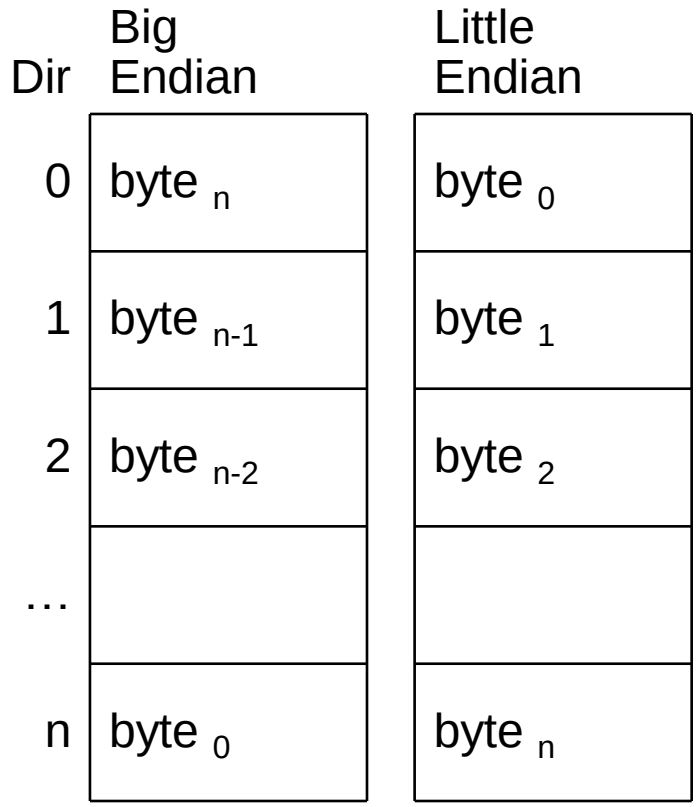
- Los procesadores Intel 8086 y 8088 son la base del IBM-PC y compatibles
(8086 introducido en 1978, primer IBM-PC en 1981)
- Diseño CISC
- Todos los procesadores Intel, AMD y otros están basados en el original 8086/8, y son compatibles con éste.
- 8086 es un procesador de 16-bit
- 16-bit data registers
- Bus de datos externo de 16 bits (8086) u 8 bits (8088)
- Algunas técnicas para optimizar la performance, por ejemplo la Unidad de Prefetch
- Modelo de memoria segmentado
- Formato de datos Little-Endian
- Manejan Stack por hardware

Orden de los Bytes: Little-Endian y Big-Endian

- Indican de qué forma se almacena en memoria la secuencia de bytes que representan un **escalar** multi-byte

- *Little endian* indica que el byte menos significativo de la secuencia de bytes será almacenado en la dirección de memoria menor

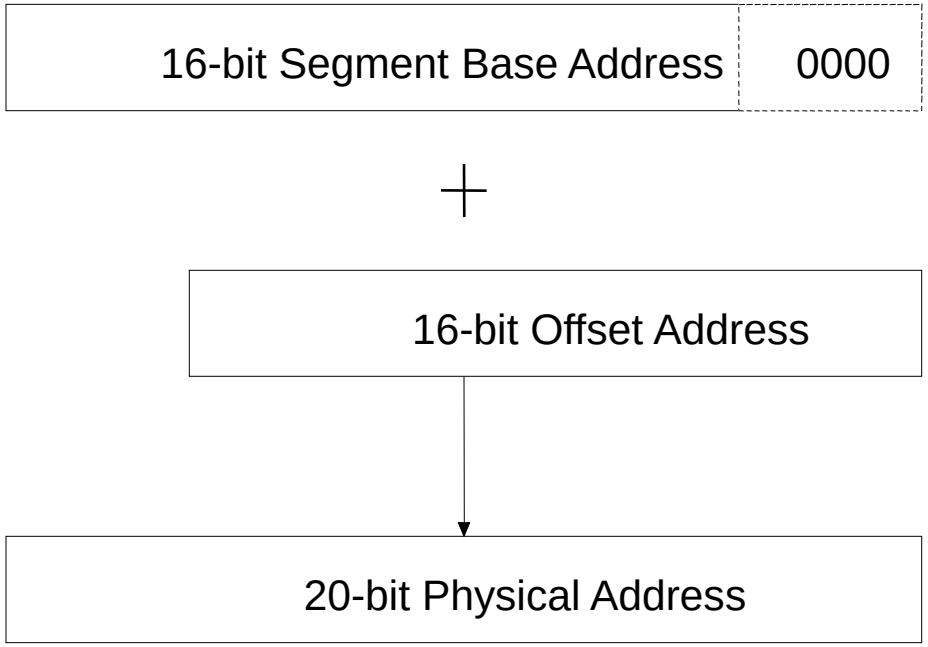
- En la imagen se muestra cómo una secuencia de bytes, $byte_n$… $byte_0$, se guarda en memoria en cada caso. $byte_0$ es el menos significativo y $byte_n$ es el más significativo

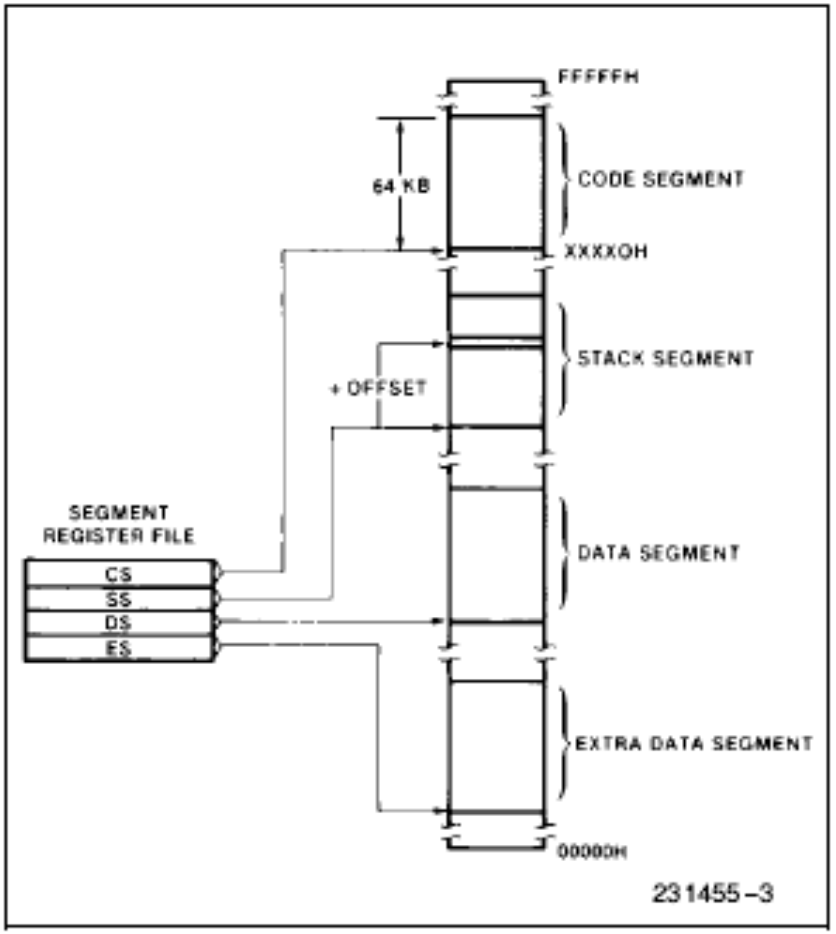


Modelo de memoria segmentado

- 8086 direcciona 1Mb de memoria principal
- Registros de 16 bits → direccionan 64K
- Para direccionar 1Mb se necesitan 20 bits
- Registros de Segmento
- dirección = registro de segmento * 16 + desplazamiento
- 8086 dispone de 4 registros de segmento de 16 bits
  - CS – Code Segment
  - DS – Data Segment
  - SS – Stack Segment
  - ES – Extra Segment

Modelo de memoria segmentado



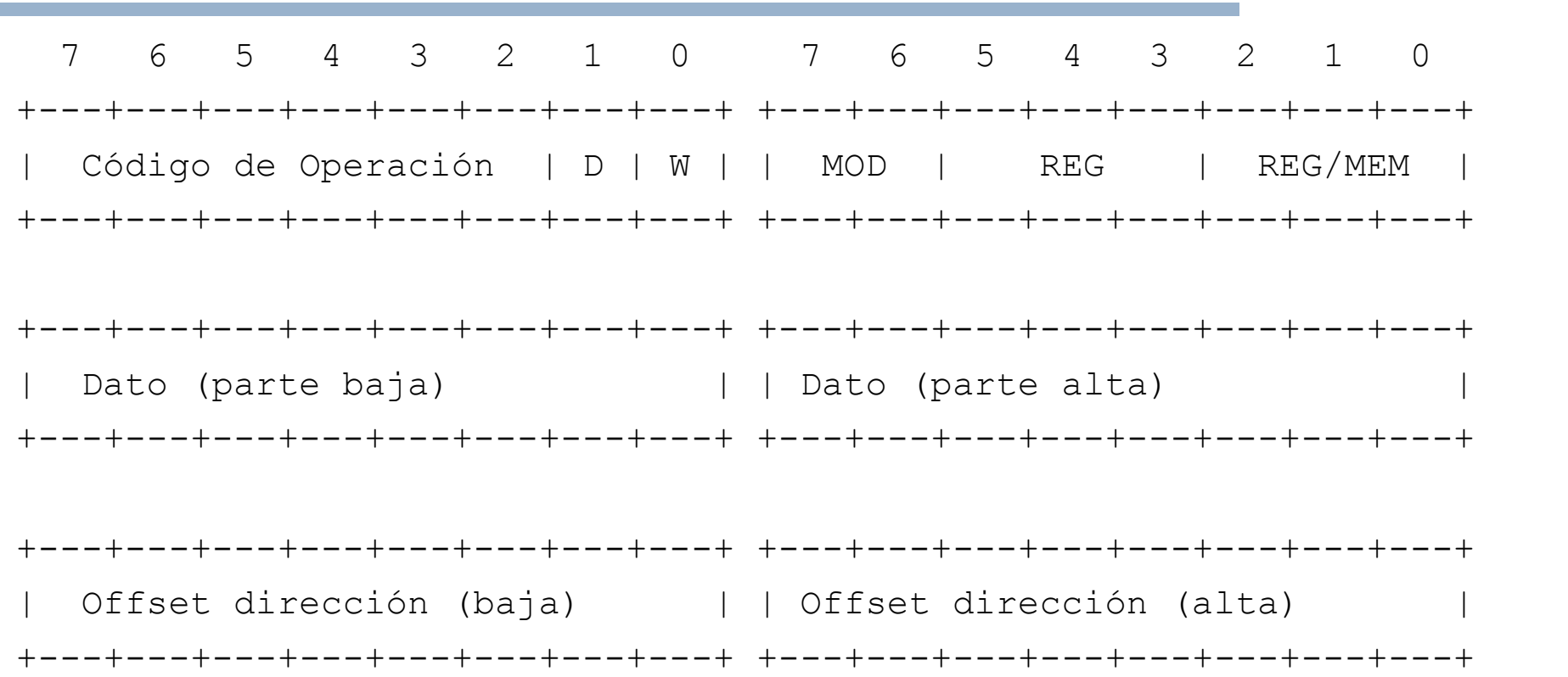


dirección = registro de segmento * 16 + desplazamiento

Arquitectura del 8086

- Set de instrucciones
  - CISC
  - Largo variable
  - ALU con soporte de multiplicación y división entera
  - Manejo de Strings (mover, comparar, buscar, etc)
- Arquitectura de 2 operandos
- Formato de instrucción
  - Variable (1 a 6 bytes)
  - Soporte de prefijos que modifican parámetros de la instrucción

Formato de instrucción (i)



D – sentido operacion (0 desde reg)

W – op usa palabras (1) o bytes (0)

MOD – modo de direccionamiento

REG – indica registro

REG/MEM – 2do reg o regs involucrados en direccionamiento indirecto

Set de Registros (i)

- Datos o almacenamiento temporal
- AX, acumulador.
- BX, base.
- CX, contador.
- DX, dato.

Registro	Byte Superior	Byte Inferior
AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL
	FEDCBA98	76543210

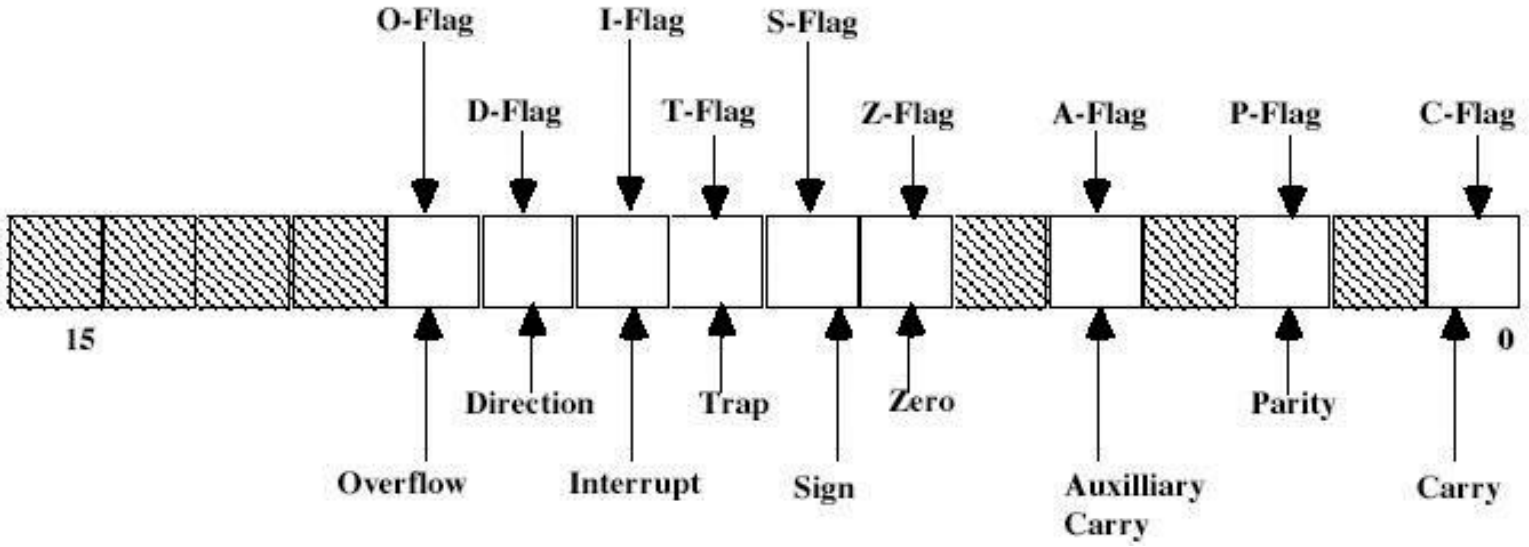
← Bit hex

Set de Registros (ii)

- Segmento
- CS, código
- DS, dato.
- SS, pila.
- ES, extra.
- Punteros a pila
- SP, tope de la pila.
- BP, base a la pila.

Set de Registros (iii)

- Registros de índice
- SI, índice origen.
- DI, índice destino.
- IP, Puntero a instrucción
- Banderas (Flags)



Modos de direccionamiento (i)

- INMEDIATO
  - Utilizado cuando se hace referencia a un valor constante que se codifica junto con la instrucción. Es decir dicho parámetro representa a su valor y no a una dirección de memoria o un registro que lo contiene.
  - Pueden ser de 8 o 16 bits
  - Ej: MOV AX,500

- DIRECTO A REGISTRO
  - Un parámetro que direcciona a un registro está utilizando el modo de direccionamiento REGISTRO.
  - Ej: MOV AX,BX
  - En este ejemplo los dos parámetros direccionan un registro.

Modos de direccionamiento (ii)

- DIRECTO A MEMORIA

•Se utiliza el modo directo cuando se referencia a una dirección de memoria y la misma está codificada junto con la instrucción.

•Ej: MOV AL,[384]

•El offset (16 bits) de la dirección de memoria se codifica junto con la instrucción y el segmento se asume a DS (con prefijo se puede utilizar otro segmento)

- INDIRECTO

•Se utiliza el modo indirecto cuando se referencia a una dirección de memoria a través de uno o dos registros para especificar el desplazamiento

•Se restringe los registros que se pueden utilizar para este modo:

{ BX | BP } [+ { SI | DI }]

{ SI | DI }

•Ej: MOV AL,[BX+SI] MOV DX, [DI]

Modos de direccionamiento (iii)

- INDIZADO (directo + indirecto)
  - Es la combinación de los modos directo e indirecto
  - Aplican las mismas restricciones sobre los registros que se pueden utilizar

{ BX | BP } [+ { SI | DI }] { + desplazamiento }

{ SI | DI } { + desplazamiento }

- Ej: MOV AL,[BX+SI+15] MOV DX, SS:[DI+4]

	CS	SS	DS	ES
IP	Si			
SP		si		
BP	Prefijo	por defecto	prefijo	prefijo
BX	Prefijo	prefijo	por defecto	prefijo
SI	Prefijo	prefijo	por defecto	prefijo
DI	Prefijo	prefijo	por defecto	por defecto (cadenas)

Entrada/Salida

- Los procesadores Intel tienen el espacio de direccionamiento de E/S separado de la memoria principal (Código y Datos)
- Se utilizan dos instrucciones para realizar E/S:
  - IN
  - OUT
- Ejemplos:
  - IN AX, DX
  - IN AL, 0x20
  - OUT DX, BX
  - OUT 0x20, BL

Set de Instrucciones

- Aritméticas
- Lógicas
- Desplazamiento
- Manejo de Flags
- Bifurcación Incondicional
- Bifurcación Condicional
- Interrupciones
- Manejo de Stack

- Ver cartilla del curso

Instrucciones assembler

Formato: código op1, op2

Tipo Args: indica la forma puede tomar cada parámetro

- i, operando inmediato (1 o 2 bytes)
- d , desplazamiento inmediato (1 byte)
- r, registro de uso general (8 o 16 bits)
- R, registro de uso general (16 bits)
- m, palabra de memoria (1 o 2 bytes)
- M, palabra de memoria (2 bytes)
- CL, el nombre de un registro en particular

Lógica: pseudo código con la lógica de la instrucción.

Descripción: semántica de la instrucción.

Banderas:

- X, afecta apropiadamente el valor de la flag.
- ?, el valor de la flag luego es indeterminado
- -, no afecta el valor de flag

ADD

Formato: ADD op_1, op_2

Tipo Args: (r,m) (r,m,i)

Lógica: $op_1 \leftarrow op_1 + op_2$

Descripción: Suma los dos operandos y almacena el resultado en op_1, por lo tanto ambos deben tener el mismo tipo

Banderas:

OF	DF	IF	TF	SF	ZF	AF	PF	CF
X	-	-	-	X	X	X	X	X

- ADC, SUB y SBB

MOV

Formato: MOV op_1, op_2

Tipo Ops: (r, m) (r, m, i)

Lógica: $op_1 \leftarrow op_2$

Descripción: Copia un byte o una palabra (según el tipo de los operadores) desde el operando fuente al operando destino.
Ambos operandos deben ser del mismo tipo (byte o palabra).
El contenido especificado por el elemento fuente se copia sobre el elemento destino, quedando inalterado el elemento fuente.

Atención: **No se puede mover memoria a memoria.**

Banderas:

OF	DF	IF	TF	SF	ZF	AF	PF	CF
-	-	-	-	-	-	-	-	-

MUL

Formato: MUL *op*

Tipo Args: (r,m)

Lógica:

si *op* es de tipo byte ⇒
Ax = Al * *op*
sino si *op* es de tipo palabra ⇒
Dx, Ax = Ax * op
fin si
si la mitad superior del resultado es 0
CF = 0
sino
CF = 1
fin si
OF = CF

- DIV

INC

Formato: INC op

Tipo Args: (r,m)

Lógica: $op = op + 1$

Descripción: Incrementa el operando destino.

Banderas:

OF	DF	IF	TF	SF	ZF	AF	PF	CF
X	-	-	-	X	X	X	X	-

- DEC, CBW (expande signo de AL a AH) y NOT

CMP

Formato: CMP op_1, op_2

Tipo Args: (r,m) (r,m,i)

Lógica: $op_1 - op_2$

Descripción: Resta op_1 de op_2 pero solo afecta las flags ignorando el resultado. Los operandos quedan inalterados pudiéndose consultar las flags mediante una operación de bifurcación condicional.

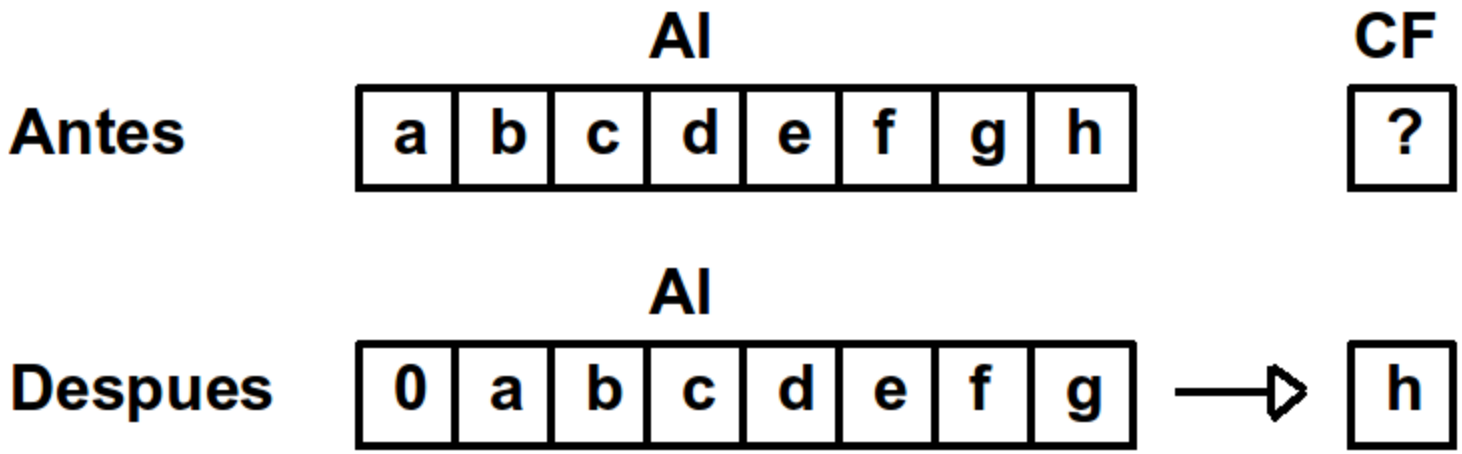
Banderas:

OF	DF	IF	TF	SF	ZF	AF	PF	CF
X	-	-	-	X	X	X	X	X

- AND, OR, XOR y NOT.

SHR

Formato:	SHR	op_1,	op_2
Tipo Args:	(r,m)	(1, CL)	
Lógica:	corre op_1, op_2 lugares a la der.		
	Ej:	**SHR AI,1**	



- SHL, SAR, ROL y ROR.

IN

Formato:	IN	op_1,	op_2
Tipo Args:		(AL, AX)	(i,DX)

Lógica:

si $\mathrm{op}_1 = \mathbf{Al}$

$\mathbf{Al} = \mathrm{I/O}(\mathrm{op}_2)$

sino si $\mathrm{op}_1 = \mathbf{AX}$

$\mathbf{Ax} = \mathrm{I/O}(\mathrm{op}_2)$

fin si

Descripción: Transfiere un byte o una palabra de una puerta de entrada del procesador al registro Al o Ax, respectivamente.

El nro. de la puerta se puede especificar mediante:

- Un valor fijo (de o a 255).
- Un valor variable, el contenido en el registro Dx (de 0 a 65535), pudiéndose acceder a 64K puertas de entrada.

- OUT y MOV

CLC

Formato: CLC

Lógica: CF := 0

Descripción: Borra la bandera de acarreo (CF) sin afectar a ninguna otra bandera.

Banderas:

OF	DF	IF	TF	SF	ZF	AF	PF	CF
-	-	-	-	-	-	-	-	0

- STC, CLI y STI.

JC (JB)

Formato: JC op_1

Tipo Args: (d)

Lógica:

Si $\mathrm{CF} = 1$

$\mathrm{IP} \leftarrow op_1 + \mathrm{IP}$

Descripción: Transfiere el control a la instrucción (IP + op_1) si se cumple la condición CF=1 .

El desplazamiento es un valor con signo de 8 bits, es decir esta comprendido entre -128 y 127.

- JA, JNB, JNA, JE, JNE, JG, JNG, JO, JNO, JS y JNS.

CALL y RET

Formato: CALL op_1

Tipo Args: (R,M,a,A,W)

Lógica:

```
Si llamada FAR
        PUSH CS
        PUSH IP
        CS := segmento op1
        IP := offset op1
sino { llamada NEAR }
        PUSH IP
        IP := op1
fin si
```

Formato: RET

Lógica:

```
POP IP
si procedimiento FAR
        POP CS
fin si
```

- JMP: Igual que call sin guardar dirección de retorno

PUSH y POP

Formato:	PUSH op_1
Tipo Args:	(R,M)
Lógica:	SP=SP-2 MOV SS:[SP],op_1
Descripción:	Decrementa el puntero del stack, SP y transfiere la palabra especificada por op_1 a la dirección SS:SP

Formato:	POP op_1
Tipo Args:	(R,M)
Lógica:	MOV op_1,SS:[SP] SP=SP+2
Descripción:	Transfiere la palabra en tope del stack al operando op_1 y luego incrementa el puntero del stack, SP.

- PUSHF y POPF

Constantes

- Valores binarios, tiras de ceros y unos.
- Terminan con b o B.
- Ej: 100011 b.
- Valores octales.
- Terminan con o, O, q o Q.
- Ej: 664 o.
- Valores hexadecimales
- Empiezan con 0..9.
- Terminan con h o H.
- Ejemplo: 0FFFh
- Valores decimales.
- Strings, secuencias de caracteres ASCII.
- Van entre comillas simples.
- ‘Hola mundo’.

Directivas(1/5)

- La directiva EQU
- La forma de esta directiva es:

  identificador EQU expresión
- Ejemplo:

```
NULL EQU 0
TAM_ELEM EQU 4*8
interElem EQU CX
MASK EQU 100010 b
MASK_2 EQU MASK SHR 2
```

Directivas(2/5)

- Las directivas DB, DW, DDW y DUP
- La forma de estas directivas es:

  etiqueta {*DB*|*DW*|*DDW*} expresión1, expresión2,...

  cantidad *dup* (valor)
- Ejemplo:

```
iterElem DW 0
vectorChico DB 1,2,3,4,5,6
vectorGrande DB 1024 dup(0)
...
mov ax,[iterElem]
mov bl,[vectorChico+2]
```

Directivas(3/5)

- La directiva MACRO
- La forma de esta directiva es:

```
nombreMacro MACRO [parametro[,parametro...]]
  instrucciones
ENDM
```

- Ejemplo:

```
sqr MACRO registro
  mov AX,registro
  mul registro
  mov registro,AX
ENDM
```

Directivas(4/5)

- Las directivas byte ptr y word ptr
- La forma de esta directiva es:

  {byte|word} ptr elemento
- Ejemplo:

```
mov byte ptr [ES:BX], 0
mul word ptr [DI]
```

Directivas(5/5)

- La directiva PROC
- La forma de esta directiva es:

  nombreProc PROC [NEAR|FAR]
- La directiva ENDP
- La forma de esta directiva es:

  nombreProc ENDP
- Definición de un procedimiento:

  nombreProc PROC atributo

  …

  nombreProc ENDP

Instrucciones (1/2)

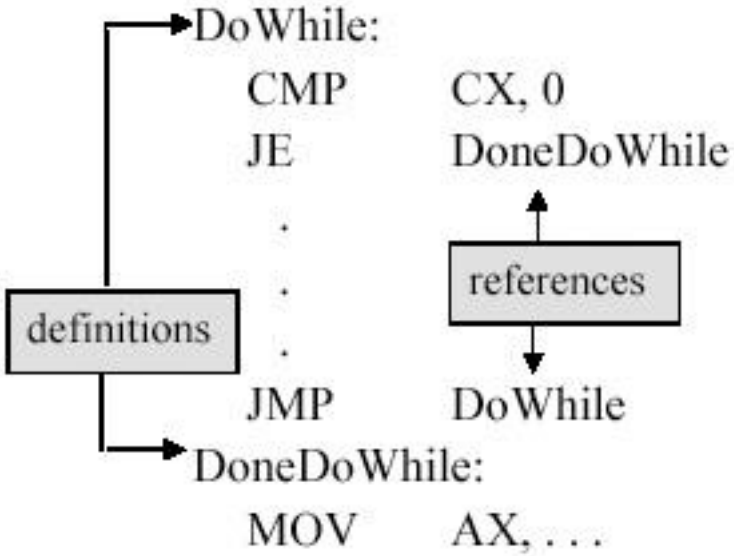
- Es una secuencia de 1 a 6 bytes.
- Formato

  [etiqueta] nombreInstruccion [operandos] [comentarios]
- Etiqueta (i)

- Nombre simbólico que referencia la primera posición de la instrucción.
- Puede estar formada por
- Letra A a Z.
- Números 0 a 9.
- Caracteres especiales @ - . $.

Instrucciones (2/2)

- Etiqueta (ii)



- Los comentarios comienzan con un ‘;’

Ejemplo

- Calcular la suma de 0 hasta BX y retornar en AX el resultado

```
; el parametro viene en BX
; el resultado se devuelve en AX
suma PROC
  xor AX,AX
  xor CX, CX
while:
  cmp BX,CX
  je fin
  inc CX
  add AX, CX
  jmp while
fin:
  ret
suma ENDP
```

Acceso a estructuras de datos en memoria (1/3)

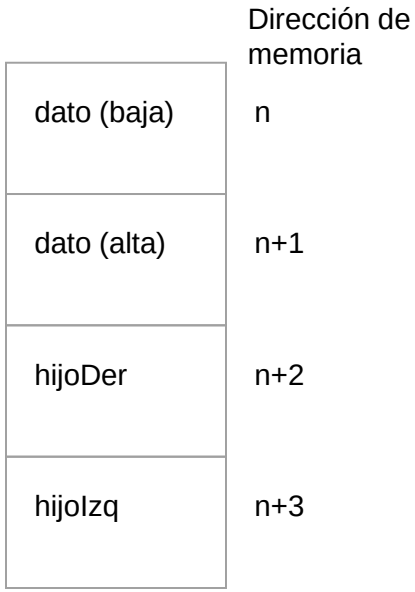
- Las variables que se definen contiguas en el programa aparecen contiguas en memoria. Cada una de ellas ocupa tantos bytes como sean necesario por su tipo.
- Ejemplo:

```
iterador:short;

puerto:byte;

…
```

Acceso a estructuras de datos en memoria (2/3)

- Los campos de una variable de tipo estructurado se almacenan en posiciones contiguas de memoria en el orden en que aparecen declaradas y ocupando tantos bytes como sean necesarios para alojar al tipo del campo.
- Ejemplo:

```
type nodo=
 record
   dato:short;
   hijoDer:byte;
   hijoIzq:byte;
 end
```



Acceso a estructuras de datos en memoria (3/3)

- Los elementos de una variable de tipo array se almacenan en posiciones contiguas de memoria en el orden en que aparecen declaradas.
- Ejemplo:

```
type arbol=array[0..(MAX_NODOS-1)] of nodo;
```

	dirección	Índice
dato (baja)	n	0
dato (alta)	n+1	
hijoDer	n+2	
hijoIzq	n+3	
dato (baja)	n+4	1
dato (alta)	n+5	
hijoDer	n+6	
hijoIzq	n+7	

...

	dirección	Índice
dato (baja)	n+i*4	i
dato (alta)	n+i*4+1	
hijoDer	n+i*4+2	
hijoIzq	n+i*4 3	

Compilado de estructuras de control (1/2)

- if-then-else

Alto nivel	Asembler

`if (i<>0) then`
`  {bloque del if}`
`else`
`  {bloque del else}`
	`cmp i,0`
	`  je else`
	`  ; aquí va el bloque que`
	`  ; corresponde al then`
	`  jmp finIf`
	`else:`
	`  ; aquí va el bloque que`
	`  ; corresponde al else`
	`finIf:`

Compilado de estructuras de control (2/2)

- while

Alto nivel	Asembler

`while (i<n) do`
`  {bloque del while}`
	`while:`
	`  cmp i,n`
	`  jae finWhile`
	`  ; aquí va el bloque que`
	`  ; corresponde al cuerpo`
	`  ; del while`
	`  jmp while`
	`finWhile:`

Compilado una función(1/3)

- La función len retorna el largo de un string.
- Alto nivel:

```
String=array[0..(LARGO_MAXIMO)] of byte;

function len(str: String):short;
  iterStr: short;
begin
  iterStr:=0;
  while (str[iterStr]<>NULL) do
    iterStr:=iterStr+1;
  len:=iterStr;
end
```

Compilado una función(2/3)

- Asembler

```
NULL EQU 0

; el desplazamiento de str viene en bx
; el resultado se devuelve en di
len proc
  xor di,di
while:
  cmp byte ptr [bx+di],NULL
  je fin
  inc di
  jmp while
fin:
  ret
len endp
```

Compilado una función(3/3)

- Invocando a la función

```
miString db 'hola mundo'
         db NULL
…
mov bx, offset miString
call len
cmp di, …
```

```
;el string esta en la
;direccion segmentada 100:1000
mov bx,1000
mov ax,100
mov ds,ax
call len
```

```
;el string esta en la
;direccion absoluta 0x98765
mov bx,5
mov ax,0x9876
mov ds,ax
call len
```

Pasaje de parametros via Stack

- Asembler

```
NULL EQU 0
; el desplazamiento y segmento de str vienen en el stack
; el resultado se devuelve en el stack
len proc
  pop ax  ; recupero dir de retorno
   pop bx  ; obtengo desplazamiento de str
   pop ds  ; segmento de str
   xor di,di
while:
   cmp byte ptr [bx+di],NULL
   je fin
   inc di
   jmp while
fin:
   push di   ; guardo resultado en stack
   push ax   ; coloco dir de retorno
   ret
len endp
```

Pasaje de parametros via Stack

- Invocando a la función

```
miString db 'hola mundo'
         db NULL
…
mov bx, segment miString
push bx
mov bx, offset miString
push bx
call len
pop di
cmp di, …
```

```
;el string esta en la
;direccion segmentada 100:1000
mov ax,100
push ax
mov ax,1000
push ax
call len
```

```
;el string esta en la
;direccion absoluta 0x98765
mov ax,5
push ax
mov ax,0x9876
push ax
call len
```

Pasaje de parametros via Stack/preservacion de regs

- Asembler

```
NULL EQU 0
; el desplazamiento y segmento de str vienen en el stack
; el resultado se devuelve en el stack
len proc
  push bp
  mov bp, sp
  push ds
  push bx
  push di
  mov bx, [bp+6]
  mov ds, bx
  mov bx, [bp+4]
  xor di,di
while:
  cmp byte prt [bx+di],NULL
  je fin
  inc di
  jmp while
fin:
  mov [bp+6], di
  mov di, [bp+2]
  mov [bp+4], di
  pop di
  pop bx
  pop ds
  pop bp
  add sp, 2
  ret
len endp
```

Referencias

- 8088-8086/8087 Programación Ensamblador en entorno MS DOS, Miguel Angel Rodriguez-Roselló, Anaya, 1988.
- Art of Assembly Language, http://webster.cs.ucr.edu/AoA/DOS/AoADosIndex.html