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)
- Todos los procesadores Intel, AMD y otros están basados en el original 8086/8, y son compatibles.
  - En el arranque, Pentiums, Athlons etc se ven como un 8086: Instruction Pointer apunta a FFFF0H
- 8086 es un procesador de 16-bit
  - 16-bit data registers
  - 16 or 8 bit external data bus
- Algunas técnicas para optimizar la performance, por ejemplo la Unidad de Prefetch
- Segmentos: Offset memory model
- Formato de datos Little-Endian

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 menos significativo

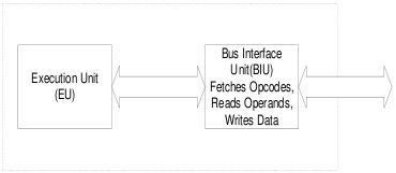
Dir	Big Endian	Little Endian
0	$byte_n$	$byte_0$
1	$byte_{n-1}$	$byte_1$
2	$byte_{n-2}$	$byte_2$
...		
n	$byte_0$	$byte_n$

8086/8088

- Los accesos a memoria enlentecen el ciclo de ejecución
  - El 8086/8 usa una cola de instrucciones para mejorar la performance
  - Mientras el procesador decodifica y ejecuta una instrucción, la unidad de interfaz con el bus puede leer nuevas instrucciones, pues en ese momento el bus está en desuso
- El IBM PC original usa el microprocesador 8088
  - 8088 es similar al 8086 pero tiene un bus externo de 8 bits y una cola de instrucciones de solo 4 bytes
  - El bus de 8-bit reduce la performance, pero baja los costos
- La arquitectura del 8086 y el 8088 se puede considerar en conjunto
  - Algunos clones del PC usaban el 8086 para mejorar la performance

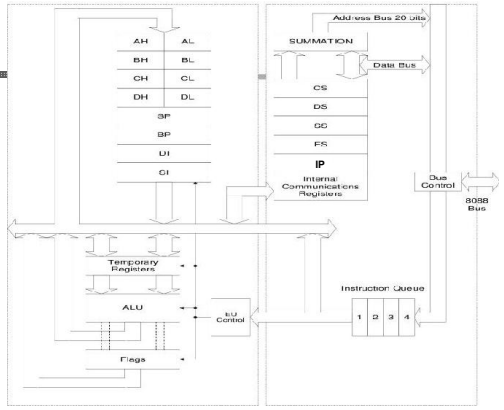
Unidades funcionales del 8086/8088

- Execution unit (EU) – ejecuta las instrucciones
- Bus interface unit (BIU) – fetch de instructiones y operandos, escritura de resultados
- Prefetch queue: 8086/6 bytes, 8088/4 bytes



8086/8088 MPU

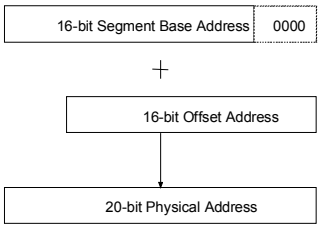
Organización Interna del 8086/8088

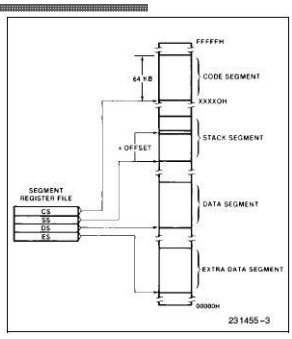


Elementos de la BIU

- Instruction Queue: la próxima instrucción u operando puede ser leído desde memoria mientras el procesador ejecuta la instrucción corriente
  - Dado que la interfaz de memoria es mucho más lenta que el procesador, la cola de instrucciones mejora la performance global del sistema.
- Registros de Segmento:
  - CS, DS, SS y ES: registers de 16 bit
  - Usados como base para generar las direcciones de 20 bits
  - Permiten al 8086/8088 direccionar 1Mbyte de memoria
  - El programa puede utilizarlos para apuntar a diferentes segmentos durante la ejecución
- Instruction Pointer (IP) contiene el offset de la dirección de la próxima instrucción (distancia en bytes desde la dirección contenida en el registro CS)

Direcciones de 20 bits en el 8086/8088

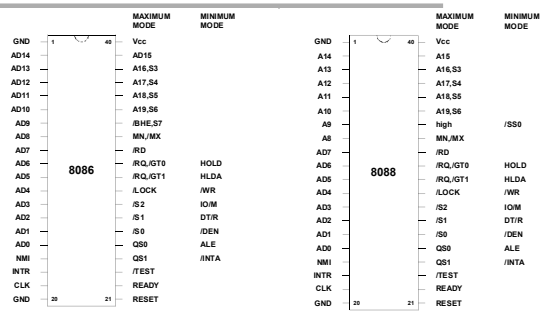


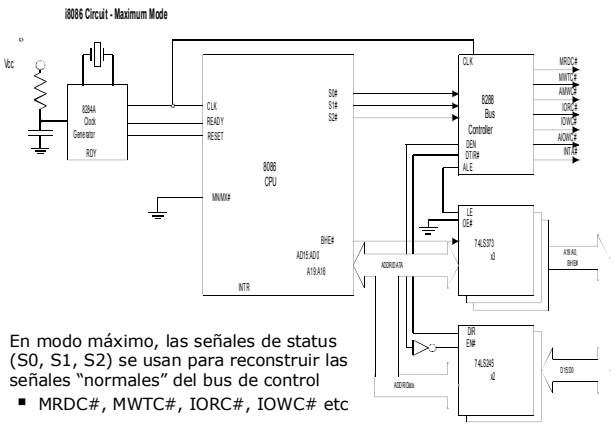


Construyendo un sistema basado en el 8086/8

- Los microprocesadores 8086/8 necesitan circuitos extra para construir un sistema
- Los buses de datos y direcciones se multiplexan en los mismos pines del procesador
  - Se necesita lógica extra para demultiplexar direcciones y datos y poder acceder a RAMs y ROMs
- Modos de funcionamiento
  - Máximo
  - Mínimo
- El Modo Máximo el 8086/8 necesita *al menos* los siguientes circuitos extra: 8288 Bus Controller, 8284A Clock Generator, 74HC373s y 74HC245s

Modos de funcionamiento





- En modo máximo, las señales de status (S0, S1, S2) se usan para reconstruir las señales "normales" del bus de control
  - MRDC#, MWTC#, IORC#, IOWC# etc

Evolución: 80186/80188

- Set de Instrucciones aumentado
- Componentes del sistema "on-chip"
  - Clock generator
  - Controlador DMA
  - Controlador interrupciones
  - Timer
  - etc…
- No utilizado en PCs
- Popular en sistemas embebidos
  - Controladores
  - Hardware dedicado

Señal RESET

- RESET es activa en nivel bajo. Pone al 8086/8 en un estado definido
- Limpia los registros de flags, segmento, etc
- Pone la dirección de programa efectiva en 0FFFF0h (CS=0F000h, IP=0FFF0h)
  - Programas en el 8086/8 siempre arrancan en FFFF0H después de un Reset
  - En esta dirección deben instalarse las rutinas de inicialización del sistema: en el PC, la ROM BIOS
- Esta característica se mantiene en las últimas generaciones de procesadores

Direccionamiento: memoria y E/S por separado (i)

- Los procesadores Intel tienen en espacio de direccionamiento de E/S separado de la memoria principal (Código y Datos)
- Decodificación de direcciones de dispositivos de E/S por separado
  - Uso de las señales IOR# and IOW#
- Se corresponden con instrucciones separadas para acceder la E/S y la memoria
  - MOV AL, [BX] ; acceso a memoria
  - IN AL, 2Ch ; acceso a E/S
- Algunos procesadores tienen un espacio de direcciones unificado. Los dispositivos de E/S son decodificados en el mapa de memoria principal
  - "E/S mapeada en memoria"

Direccionamiento:
memoria y E/S por separado (ii)

- Ventajas de la E/S mapeada en memoria
  - Dispositivos de E/S accedidos por instrucciones normales - no se necesitan instrucciones separadas
  - Se reduce el tamaño del set de instrucciones
  - No se necesitan pines especiales (IOR#, IOW#)
- Ventajas de espacios de direccionamiento separados
  - Todo el mapa de direcciones disponible para memoria
  - La E/S puede usar instrucciones más pequeñas y rápidas
  - Fácil distinguir accesos de E/S en lenguaje ensamblador
  - Menos hardware para decodificar E/S.

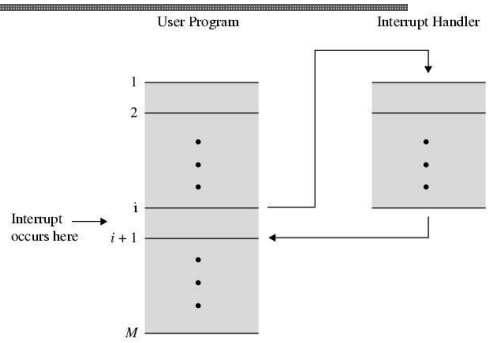
Interrupciones

- Mecanismo que permite interrumpir la secuencia normal de procesamiento de la CPU ante condiciones particulares
- Pueden ser de alguna de las siguientes clases:
  - Programa
    - Ej. overflow, división por cero, protección de memoria
  - Timer
    - Generado por un temporizador interno del procesador
    - Usado en pre-emptive multi-tasking
  - E/S
    - Provocada por el controlador de E/S
  - Fallo de Hardware
    - Ej. Error de paridad de memoria

Ciclo de Interrupciones

- Se agrega al ciclo de instrucción
- Procesador chequea por la interrupción
  - Indicado por una señal de interrupción
- Si no hay interrupción, fetch próxima instrucción
- Si hay una interrupción pendiente:
  - Suspender ejecución del programa corriente
  - Salvar contexto
  - Hacer que PC apunte a la dirección de inicio del manejador de la interrupción
  - Procesar interrupción
  - Restaurar contexto y continuar el programa interrumpido

Transferencia de Control vía Interrupciones

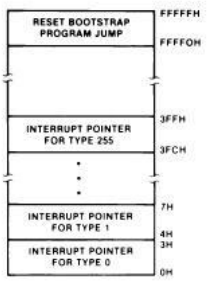


Interrupciones

- Tipos
  - Hardware: dispositivos de entrada salida
  - Internas: división entre cero
  - Software: llamadas al sistema
  - No enmascarables.
- Cada interrupción lleva asociado un número que identifica al servicio que se debe invocar.

Vector de Interrupciones

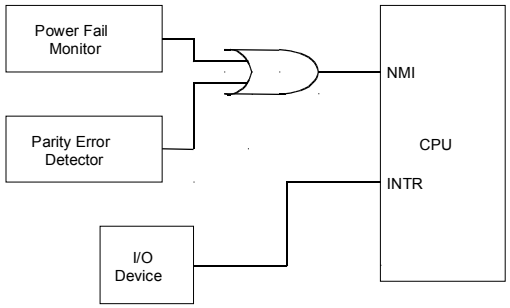
- Las localizaciones de memoria 00000H a 003FFH están reservadas para las interrupciones
- Existen 256 tipos posibles de interrupciones
  - Cada Handler o Rutina de Servicio de Interrupción está direccionada por un puntero de 4 bytes: 16 bits de segmento y 16 bits de offset
- Las rutinas y los punteros deben instalarse antes de habilitar las interrupciones
  - Servicios de la BIOS
  - Servicios del Sistema Operativo



Interrupciones enmascarables y no enmascarables

- Las interrupciones pueden enmascararse globalmente usando la Interrupt Enable Flag (IE o I)
- IE es seteada por la instrucción STI y reseteada mediante CLI
- Interrupciones no enmascarables (Non Maskable Interrupts-NMI) son prioritarias y como su nombre indica NO se pueden enmascarar
- Uso de la NMI
  - Parity Error
  - Power fail
  - Etc…

Ejemplo de NMI



Generalidades 8086/8080

- Los bytes y palabras pueden residir en cualquier lugar de la memoria (no es necesario que estén alineados)
- Puede operar con números
  - Binarios (con o sin signo de 8 ó 16 bits)
  - BCD
- Dispone de 92 instrucciones.
- Existen 7 modos de direccionamiento.
- Frecuencia típica
  - 4,77 MHz (8080)
  - 8 MHz (8086)
- Las instrucciones más rápidas se ejecutan en dos ciclos de reloj y las lentas en 206 ciclos.
- Se pueden direccionar hasta 64K puertos de E/S.

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

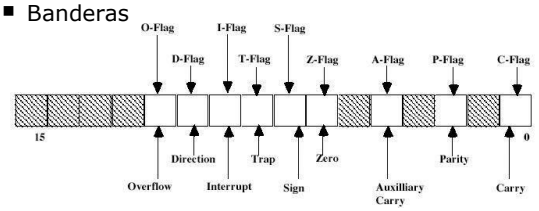
← Bit hex

Registros (ii)

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

Registros (iii)

- Registros de índice
  - SI, índice origen.
  - DI, índice destino.
- Puntero a instrucción
- Banderas



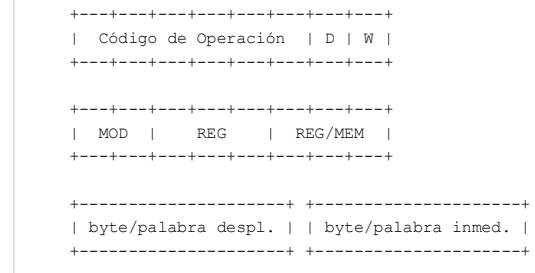
Modos de direccionamiento (i)

- Se entiende por modos de direccionamiento a las formas diferentes que pueden tomar los parámetros de las instrucciones del procesador.
- Distinguiremos fundamentalmente cuatro modos diferentes:
  - 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.
  - VALOR o 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.
    - Ej: MOV Ax,500

Modos de direccionamiento (ii)

- DIRECTO
  - 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,[127]
  - En este ejemplo el offset de la dirección de memoria se codifica junto con la instrucción y el segmento se asume a DS. Si MEMORIA es un array de bytes que representa a la memoria:
  - AL := MEMORIA[DS:127]
- INDIRECTO
  - Se utiliza el modo indirecto cuando se referencia a una dirección de memoria a través de uno o varios registros
  - Ej: MOV AL,[Bx]
  - Aquí el offset de la dirección de memoria esta contenido en el registro Bx y al igual que el caso anterior como no se especifica el segmento se asume DS. Ejemplo:
  - AL := MEMORIA [DS:Bx]

Formato de instrucción (i)



Formato de instrucción (ii)

- El **código de operación** ocupa 6 bits.
- **D** indica que el operando destino está en el campo registro.
- **W** indica que los operandos son de tipo palabra.
- **MOD** indica el modo de direccionamiento
  - 00 sin desplazamiento.
  - 01 desplazamiento de 8 bits
  - 10 desplazamiento de 16 bits
  - 11 registro
- **REG** indica el registro involucrado en la instrucción
- **R/M**, en el caso de modo registro (MOD=11) se codifica igual que el campo REG; en caso contrario se indica la forma en que se direcciona la memoria

Set de Instrucciones

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

- Ver cartilla del curso

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'.

Operadores

- Operador +, -, *, /, mod, shl, shr, and, or y xor.
  - Formato: valor1 oper valor2
  - Ejemplos
    - 75+25
    - 80*25
    - 1002/123
    - 1100001 SHR 2
- Operador not
  - Formato: oper valor
  - Ejemplo: not 1100001b
- Operador offset y seg
  - Formato: oper {etiqueta|variable}

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
```

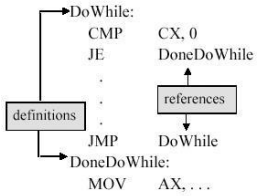
Instrucciones (1/2)

- 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)

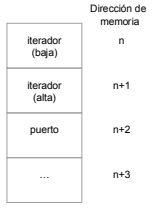


- Los comentarios comienzan con un ';'

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:

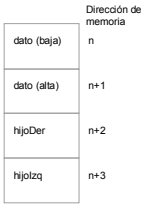
```
iterador:integer;
puerto:byte;
…
```



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:

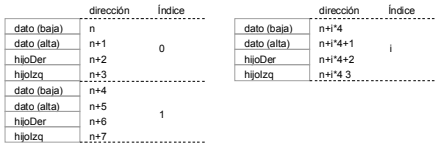
```
type nodo=
 record
   dato:integer;
   hijoDer:byte;
   hijoIzq:byte;
 end
```



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;
```



Acceso a memoria (1/3)

- Todos las direcciones se especifica como direcciones segmentadas (segmento:desplazamiento).
- El desplazamiento se define según la expresión:

  {Bx|Bp}[+{Si|Di}][+desplazamiento]|
  {Si|Di}[+desplazamiento]|
  desplazamiento
- Ejemplos de desplazamientos:
  - BX
  - BX+DI
  - BX+SI+2
- Ejemplos de direcciones segmentadas:
  - ES:[BP+SI]
  - [BP+4]
- Las direcciones segmentadas son traducidas automáticamente por el hardware multiplicando el segmento por 16 y luego sumando el desplazamiento.

Acceso a memoria(2/3)

- Segmentos utilizados

	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)

Acceso a memoria(3/3)

- Desde la perspectiva del programador
  - Las direcciones son siempre segmentadas.
  - Nunca puede especificar una dirección real de 20 bits.
- El microprocesador 8086 permite que a lo sumo uno de los operandos de la mayoría de las instrucciones esté almacenado en memoria.
- Ejemplos:
  - inc [bp]
  - mov [bx],ax
  - xor [bx],al
  - mov [bx],[bx+2] Prohibido

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

MUL

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

- 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.

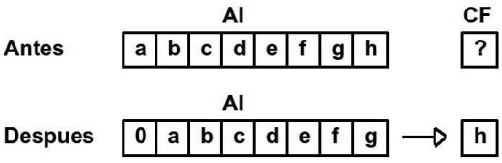
SHR

Formato: SHR op_1, op_2

Tipo Args: (r,m) (1, CL)

Lógica: corre op_1, op_2 lugares a la der.

Ej: SHR Al,1



- SHL, SAR, ROL y ROR.

IN

Formato: IN op_1, op_2

Tipo Args: (AL, AX) (i,DX)

Lógica:

si op_1 = **Al**
Al = I/O(op_2)
sino si op_1 = **AX**
Ax = I/O(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.

CALL y RET

Formato: CALL op_1

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

Lógica:

Si llamada FAR
    PUSH CS
    PUSH IP
    CS := segmento op_1
    IP := offset op_1
sino { llamada NEAR }
    PUSH IP
    IP := op_1
fin si

Formato: RET

Lógica:

POP IP
si procedimiento FAR
    POP CS
fin si

- JMP

JC (JB)

Formato: JC op_1

Tipo Args: (d)

Lógica:

Si $CF = 1$
$IP \leftarrow op_1 + IP$

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.

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

Instrucciones prohibidas

- No permitido
  - mul 4
  - mov ES,4
  - mov AX,[BX*4]
  - mov AX,BX*4
  - mov AX,BX+1
  - puhs BL
  - push 4
  - mov BL,AX

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):integer;
  iterStr: integer;
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 prt [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,9876
mov ds,ax
call len
```

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

Preguntas