

Organización de Computadoras

CURSO 2024

TURNO RECURSANTES

CLASE 7

Resumen de clase 7

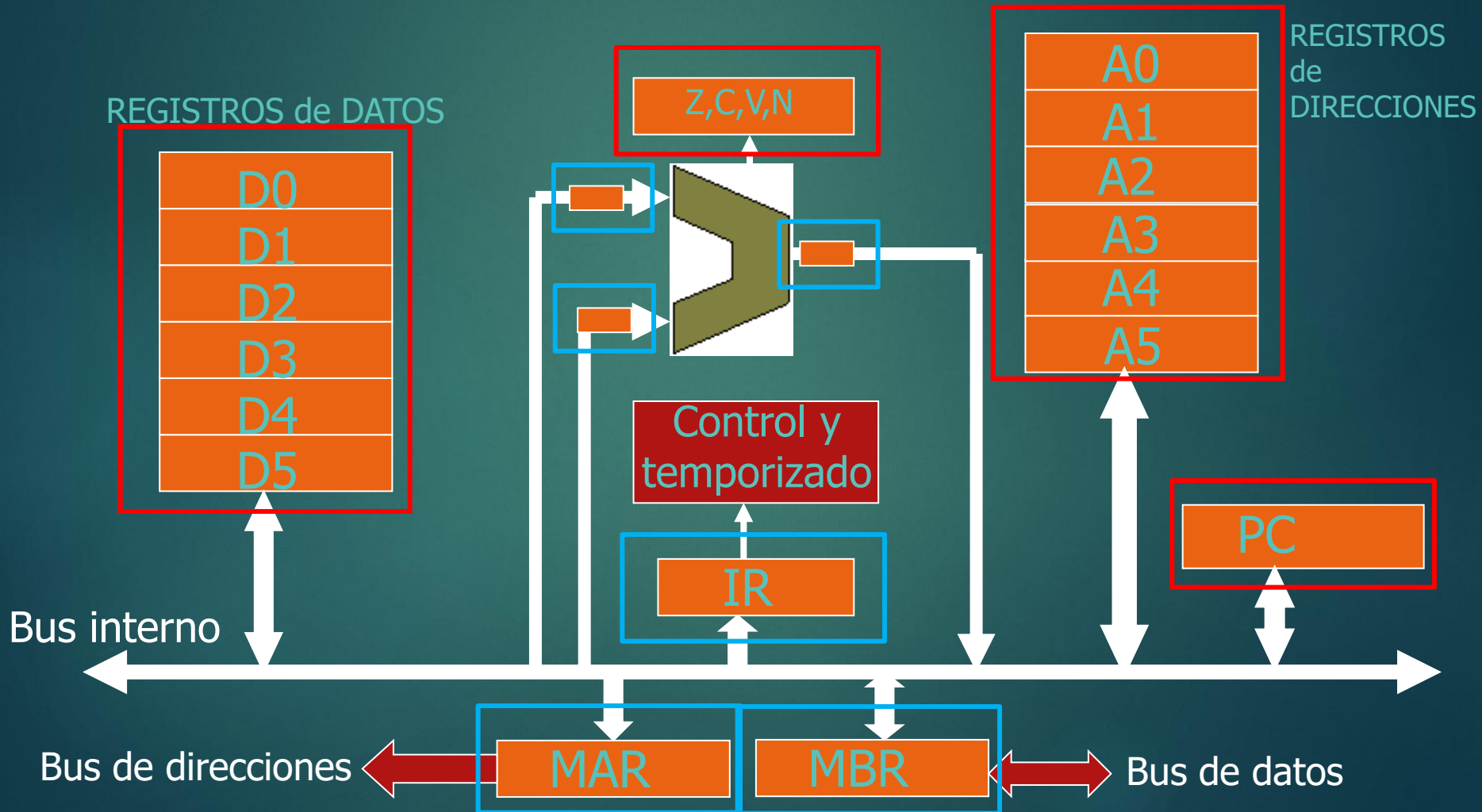
- Organización de Registros
- Instrucciones

Organización de registros

De acuerdo a la posibilidad de acceso, existen 2 tipos de registros:

- Registros visibles (al programador)
 - Referenciables por medio de instrucciones
 - Utilizables explícitamente por el programador
- Registros invisibles (al programador)
 - No referenciables por instrucciones
 - Principalmente utilizados por la UC
 - Vinculados a control, estado, o temporales

Registros visibles e invisibles



Registros visibles al usuario

De acuerdo al uso, los registros pueden ser:

- de “Propósito general”
- de Datos
- de Direcciones
- de Estado (banderas de estado o códigos de condición)

Registros visibles al usuario

Registros de propósito general

- Son de “uso universal”.
- Pueden contener indistintamente el operando en cualquier tipo de instrucción.
- Generalmente el uso genérico está restringido de alguna manera. Por ejemplo, dedicados a operaciones en PF o enteros, direcciones u operandos, etc.

Registros visibles al usuario

Registros de datos

- Son usados en operaciones que involucren datos (aritméticas, lógicas, etc.).
- Las operaciones en punto fijo usan registros distintos de las de punto flotante.
- Normalmente no se usan para manipular direcciones (aunque hay excepciones, como el BX en el 8086).
- Están relacionados al tipo de dato y tamaño de la ALU.

Registros visibles al usuario

8

Registros de direcciones

- Se usan para manipular direcciones.
- Típicamente asociados a modos de direccionamiento (ej. registro índice para el direccionamiento indexado).
- Incluso pueden tener tamaños distintos a los registros de datos.
- EL tamaño está relacionado con la capacidad de direccionamiento de la CPU, el espacio de direcciones y la forma que administra dicho espacio.

Registros visibles al usuario

9

Registros de estado

- Almacenan bits referidos al estado operativo de la CPU.
- El más importante es el que contiene los bits de estado de la ALU (carry, overflow, etc.), comúnmente llamado Status Register (SR), Flags (FR), Program status word (PSW).
- También pueden almacenar bits referidos a la operación de la CPU (interrupciones, paros de emergencia, errores, etc.).
- El tamaño es muy variable.
- Pueden ser utilizados por las instrucciones de bifurcación condicional.
- Pueden o no ser alterables por el programador.

Características de los Registros

10

Las características más importantes de los registros son:

- Tamaño
- Cantidad
- Uso

Características de los Registros

11

Tamaño:

- Los registros de direcciones, en principio, deben ser capaces de almacenar la dirección completa, o eventualmente, una fracción de la misma.
- Los de datos deben tener un tamaño que les permita almacenar la mayoría de los tipos de datos.
- Algunas máquinas permiten que 2 registros (contiguos) puedan ser utilizados separados, o como un solo registro para almacenar valores de doble longitud.

Características de los Registros

12

Cantidad:

- Es deseable tener muchos registros dentro de la CPU, porque mas registros significa más información en la CPU.
- Pero cuanto más registros, mayor es la cantidad de bits para poder identificarlo, por lo que afecta al tamaño de la instrucción.
- Menos registros significa más referencias y accesos a memoria (que es mucho más lenta que la CPU).
- Un número típico es entre 8 y 32 reg. Más registros no resultan en una gran mejora y aumenta el tamaño de la instrucción.
- 2do cuatrimestre se verá esto en procesadores RISC.

Características de los Registros

Uso:

- Un tema muy controvertido es usar todos los registros como “de propósito general” o especializar su uso.
- Si son todos de propósito general, se facilita el uso de los registros, pero el tamaño de las instrucciones tiende a ser mayor.
- Si son especializados, donde está implícito en el código de operación el registro a usar (ej. Acumulador), se ahorran bits en la instrucción pero limitan la flexibilidad del programador.
- No hay una receta.

Registros de control

Un tipo especial de registros son los Registros de Control.

- Empleados para controlar la operación de la CPU (es decir, forman parte de la Unidad de Control).
- Pueden ser visibles o invisibles al usuario.
- Los 4 esenciales para la ejecución de instrucciones:
 - Contador de programa (PC)
 - Registro de instrucción (IR)
 - Registro de dirección de memoria (MAR)
 - Registro buffer de memoria (MBR)

Registros de control

- Esos 4 registros se emplean para el movimiento de información entre la CPU y Memoria.
- Dentro de la CPU los datos se deben presentar a la ALU para procesamiento, ésta puede acceder al MBR y a los registros visibles por el usuario.
- Pueden haber también registros temporales adicionales para intercambiar datos con el MBR y demás registros visibles.

Organización de registros

CPU x86 Intel (principales)

Como ejemplo de estudio, vamos a revisar la estructura de registros de la familia de procesadores 80x86 de Intel.

El 80x86 tiene 4 conjuntos de registros:

- De datos
- De direcciones
- De segmentos
- De control

En el caso del Pentium los registros se extienden en tamaño, y se agregan algunos más.

Organización de registros

CPU x86 Intel (principales)

Procesadores 80x86 – Registros de datos

8		8	
AH	<i>AX</i>	AL	
BH	<i>BX</i>	BL	
CH	<i>CX</i>	CL	
DH	<i>DX</i>	DL	

De “uso general”
(de Datos)

Organización de registros

CPU x86 Intel (principales)

Procesadores 80x86 – Registros de direcciones

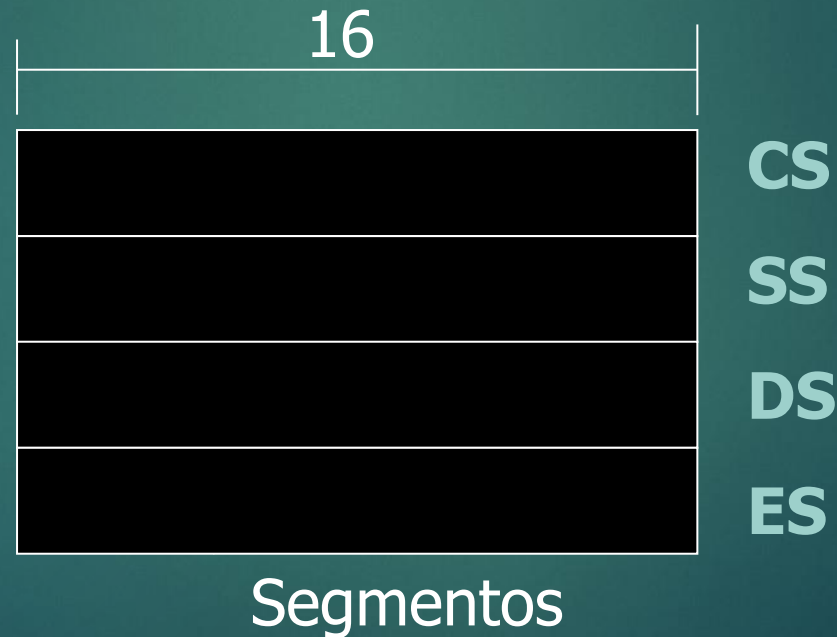


De “direccionamiento”

Organización de registros

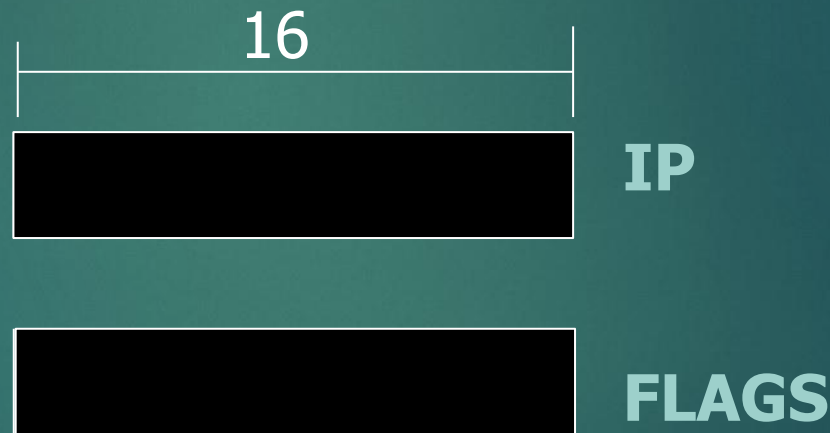
CPU x86 Intel (principales)

Procesadores 80x86 – Registros de segmentos



Organización de registros CPU x86 Intel (principales)

Procesadores 80x86 – Registros de control



PC y banderas

Organización de registros

CPU x86 Intel (principales)

Funciones principales de los registros de datos

- AX : acumulador, es el principal en las operaciones aritméticas
- BX : dato y puntero base (dir. de memoria)
- CX : dato y contador en instrucciones repetitivas
- DX : dato, y 2do operando en instrucciones de multiplicación y división

Organización de registros

CPU x86 Intel (principales)

Funciones principales de los registros de direcciones

- SI y DI : apuntadores (registros índice) que utilizan las instrucciones que recorren arreglos o tablas.
- BP : registro base, apuntador a “área de trabajo”
- SP : registro puntero de pila o stack

Organización de registros

CPU x86 Intel (principales)

Funciones principales de los registros de segmentos

- CS: registro de segmento de código
- DS: registro de segmento de datos
- SS: registro de segmento de pila
- ES: registro de segmento “extra”

Los registros de segmentos determinan las direcciones reales para acceso a la memoria. Se “combinan” con las direcciones efectivas (“EA”) obtenidas del modo de direccionamiento empleado en la instrucción.



Organización de registros

CPU x86-PII Intel (principales)

Procesador Pentium - Registros de datos

16	8	8
EAX	AH AX	AL
EBX	BH BX	BL
ECX	CH CX	CL
EDX	DH DX	DL

De “uso general”
(de Datos)

Organización de registros

CPU x86-PII Intel (principales)

Procesador Pentium - Registros de direcciones

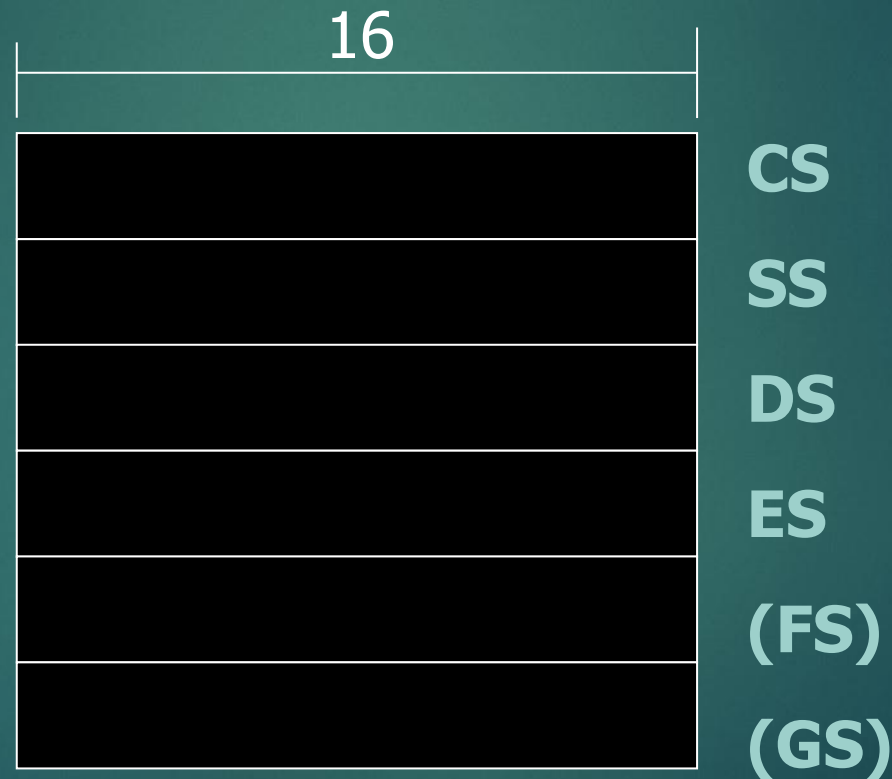
16	16
ESP	<i>SP</i>
EBP	<i>BP</i>
ESI	<i>SI</i>
EDI	<i>DI</i>

De “direccionamiento”

Organización de registros

CPU x86-Pii Intel (principales)

Procesador Pentium - Registros de segmento



Segmentos

Organización de registros

CPU x86-PII Intel (principales)

Procesador Pentium - Registros de control



PC y banderas

Registros CPU 68000 Motorola

Otra estructura de registros con una filosofía diferente a Intel, es la de la familia de procesadores 68000 de Motorola.

El 68000 tiene 3 conjuntos de registros:

- De datos
- De direcciones
- De control

En el 68000 los registros de datos y direcciones no están especializados.

En las referencias de datos, todos los registros de datos se pueden usar idénticamente.

En las referencias de direcciones, todos los registros de direcciones se pueden usar idénticamente.

Registros CPU 68000 Motorola

Procesador 68000 - Registros de datos



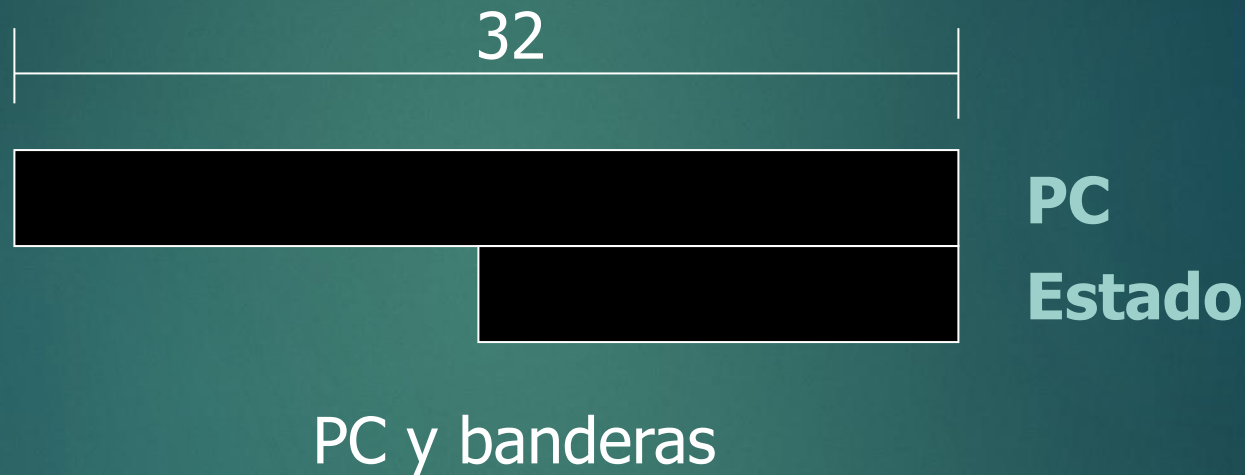
Registros CPU 68000 Motorola

Procesador 68000 - Registros de direcciones

32	
	A0
	A1
	A2
	A3
	A4
	A5
	A6
Apuntador del stack usuario	A7
Apuntador del stack supervisor	A7'

Registros CPU 68000 Motorola

Procesador 68000 - Registros de control



Instrucciones en Assembly del 80x86

32

Como se vio anteriormente, las instrucciones de máquina del MSX88 típicamente tienen el siguiente formato:

1	ADD <i>dest,fuente</i>	Suma <i>fuente</i> y <i>dest</i>	$(dest) \leftarrow (dest) + (fuente)$
1	ADC <i>dest,fuente</i>	Suma <i>fuente</i> , <i>dest</i> y <i>flag C</i>	$(dest) \leftarrow (dest) + (fuente) + C$
1	SUB <i>dest,fuente</i>	Resta <i>fuente</i> a <i>dest</i>	$(dest) \leftarrow (dest) - (fuente)$
1	SBB <i>dest,fuente</i>	Resta <i>fuente</i> y <i>flag C</i> a <i>dest</i>	$(dest) \leftarrow (dest) - (fuente) - C$
1	CMP <i>dest,fuente</i>	Compara <i>fuente</i> con <i>dest</i>	$(dest) - (fuente)$
5	NEG <i>dest</i>	Negativo de <i>dest</i>	$(dest) \leftarrow \text{CA2}(dest)$
5	INC <i>dest</i>	Incrementa <i>dest</i>	$(dest) \leftarrow (dest) + 1$
5	DEC <i>dest</i>	Decrementa <i>dest</i>	$(dest) \leftarrow (dest) - 1$

1. *dest/fuente* son: *reg/reg*, *reg/mem*, *reg/op.inm*, *mem/reg*, *mem/op.inm*.

5. *dest* solo puede ser *mem* o *reg*.

mem puede ser una etiqueta (dir.directo) o [BX], siendo (BX) una dirección de memoria (dir.indirecto).

Instrucciones en Assembly del 80x86

33

Es decir, son de la forma :

COP destino,fuente

Donde:

- COP es un código de operación, por ejemplo ADD
 - fuente es una referencia al 1er operando
 - Destino es una referencia al resultado, e implícitamente el 2do operando.

Este formato de la instrucción en Assembly se puede vincular al formato en lenguaje de máquina siguiente:

ADD	Ref. resultado: destino	Ref. Op1: fuente
-----	-------------------------	------------------

Instrucciones en Assembly del 80x86

34

- El nombre destino y fuente proviene del hecho que si hay un movimiento de datos, es desde la derecha (fuente) hacia la izquierda (destino).
- En una suma hay 2 operandos y el resultado se almacena en el lugar del operando izquierdo (destino).
- Los operandos y el resultado pueden estar en un registro o en la memoria.

Instrucciones en Assembly del 80x86

35

Si llamamos:

- mem = especificación de una dirección de memoria
- reg = especificación de un registro de la CPU
- inm = dato inmediato

Entonces las instrucciones son de la forma:

- Instrucción mem, reg
- Instrucción reg , mem
- Instrucción reg , reg
- Instrucción reg , inm
- Instrucción mem, inm

Instrucciones en Assembly del 80x86

Ejemplos: Direccionamiento a registro

➤ `ADD AX,BX`  significa: $AX = AX + BX$

Es decir, la instrucción suma el contenido del registro AX con el del BX y el resultado se guarda en AX (operando de 16 bits).

También puede sumar en 8 bits si los registros referenciados son de 8 bits.

➤ `ADD AL,AH`  significa : $AL = AL + AH$

La misma estructura sirve para una instrucción de movimiento de datos.

➤ `MOV AL,CH`  significa : $AL = CH$

Instrucciones en Assembly del 80x86

Ejemplos: Direccionamiento inmediato


➤ `ADD AX,35AFh`  `AX= AX+35AFh`

Cuando la referencia es directamente un número, y si la instrucción lo admite, el dato se toma como inmediato (16 bits).

También puede tomar el operando inmediato en 8 bits, si se referencia un registro de 8 bits.

➤ `ADD AL,15`  `AL= AL+15`

La misma estructura sirve para una instrucción de movimiento de datos.

➤ `MOV AL,3Eh`  `AL= 3Eh`

Instrucciones en Assembly del 80x86

Ejemplos: Direccionamiento directo

➤ `ADD AX, [35AFh]` ➡ `AX = AX + contenido direcc. 35AFh y 35B0h`

Cuando la referencia es un número entre corchetes (no válido para ASM88), el número es interpretado como directo. El tamaño del operando depende del tamaño del registro destino, si es de 16 bits el operando referenciado indirectamente es de 16 bits.

También puede tomar un operando directo en 8 bits, si se referencia un registro destino de 8 bits.

➤ `ADD AL, DATO` ➡ `AL = AL + variable contenida en DATO (8 bits)`

La misma estructura sirve para una instrucción de movimiento de datos:

`MOV CH, NUM1` ➡ `CH = variable contenida en NUM1 (8 bits)`

Instrucciones en Assembly del 80x86

Ejemplos: Direccionamiento Indirecto por registro

- `ADD AX, [BX]` ➡ `AX = AX + dato almacenado en dirección contenida en BX y siguiente.`

Cuando la referencia es un registro entre corchetes, el registro es interpretado como registro base y su contenido es la dirección donde reside el operando (indirecto vía registro). El tamaño del operando depende del tamaño del registro destino, si es de 16 bits el operando referenciado indirectamente es de 16 bits.


También puede tomar un operando indirecto en 8 bits, si se referencia un registro de 8 bits.

- `MOV [BX], AL` ➡ Mueve el dato contenido en AL a la dirección contenida en BX

Instrucciones en Assembly del 80x86


40

Ejemplos: Direccionamiento base + índice

- `MOV CX, [BX+SI]`  en CX se carga el dato almacenado en la direcc. BX+SI y la siguiente

Cuando la referencia es un registro base sumado a un registro índice, entre corchetes, la suma de ambos registros es la dirección donde reside el operando (base indexado). El tamaño del operando depende del tamaño del registro destino, si es de 16 bits el operando referenciado indirectamente es de 16 bits.

También puede tomar un operando base indexado en 8 bits, si se referencia un registro de 8 bits.

- `MOV [BX+DI], AL`  en la direcc. BX+DI se almacena el dato cargado en AL

Instrucciones en Assembly del 80x86

Ejemplos: Direccionamiento Relativo por registro (Base con offset)



➤ `MOV AL, [BX+2]` ➡ `AL`= dato almacenado en dir `BX+2`

Cuando la referencia es un registro base sumado a un número, entre corchetes, la suma de ambos términos es la dirección donde reside el operando (base con offset). El tamaño del operando depende del tamaño del registro destino, si es de 16 bits el operando referenciado indirectamente es de 16 bits, y si es de 8 el operando es de 8.

➤ `MOV [BX+2Ah], AX` ➡ dir `BX+2Ah` y la que sigue `<=` dato en `AX` (16 bits)

Instrucciones en Assembly del 80x86

Ejemplos: Direccionamiento relativo base+índice

- `MOV AL, [BX+SI+2]`  `AL= dato almacenado en la dir BX+SI+2`
- `MOV [BX+DI+2Ah], AX`  `dato almacenado en la dir BX+DI+2Ah y la que sigue = AX (16 bits)`

Formatos de instrucción-

Criterios de diseño

- Cuando se diseña el repertorio de instrucciones hay que definir si se usarán instrucciones cortas, largas, o combinadas.
- En general se prefiere hacer las instrucciones (por lo menos, las más usadas) cortas, porque la velocidad de ejecución de las instrucciones depende de la velocidad de transferencia (“ancho de banda”) de la memoria, es decir de la cantidad de bits que puede transferir por segundo. En general, la velocidad del procesador es varias veces mayor que el de la memoria.
- Instrucciones más cortas se leen más rápido, y el efecto que produce es que el procesador “parece” más rápido.

Formatos de instrucción-

Criterios de diseño

- De todas maneras, las instrucciones deben tener suficientes bits para expresar todas las operaciones deseadas.
- La experiencia demuestra que se requiere dejar bits libres en los diferentes campos que componen la instrucción, para permitir cambios o extensiones en el futuro de la familia.
- Otro aspecto importante es el tamaño del campo de datos, es decir, cuantos bits se le asigna a un campo que es una referencia numérica (de dato o dirección).

Ejemplo para MSX88

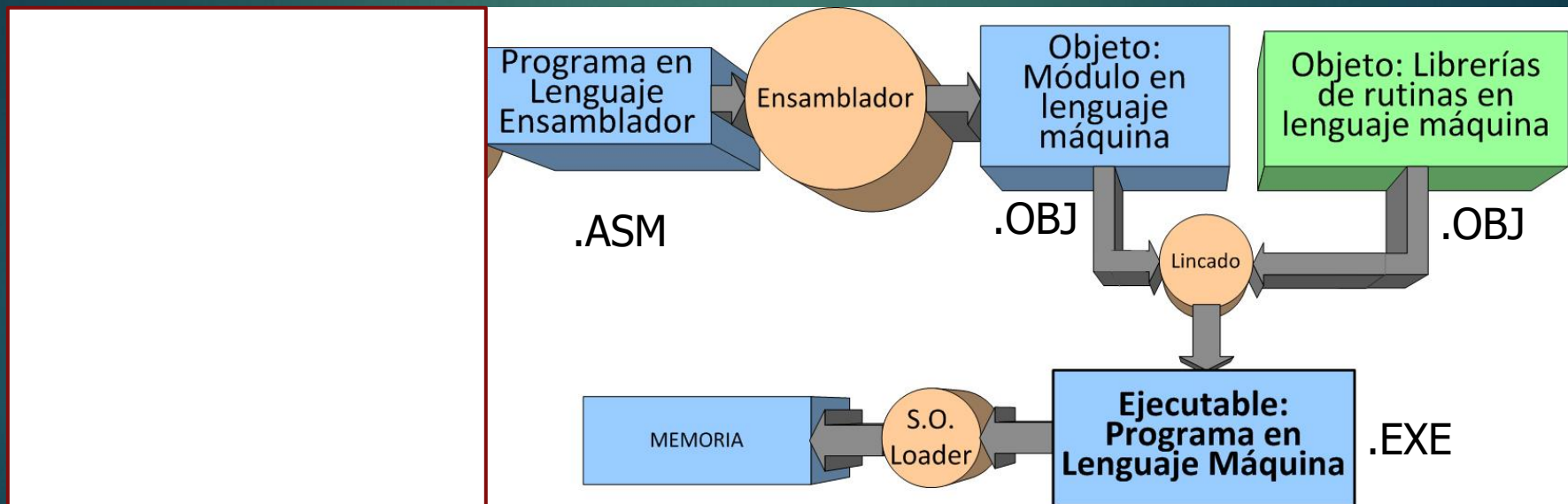
45

El proceso de desarrollo de un programa en el MSX88 (y en general en el lenguaje Assembly) requiere de 3 pasos:

- 1) Editar el programa (archivo fuente) con un editor de texto. Este archivo deberá tener (como es típico) la extensión asm (por ejemplo prueba.asm)
- 2) Ensamblar el programa fuente (por ejemplo prueba.asm) con un programa ensamblador que, en el caso del paquete de archivos del MSX88, es el Asm88. La salida del ensamblador son 2 archivos: el módulo objeto (que llevará la extensión prueba.o) y un archivo con formato imprimible (prueba.lst).
- 3) Enlazar el módulo objeto con un programa “enlazador”, en este caso Link88. La salida es un módulo ejecutable que llevará la extensión exe (por ejemplo prueba.exe).

Lenguajes

Proceso de desarrollo de un programa en Assmebly:



Ejemplo para MSX88

47

Una vez disponible el archivo ejecutable, se puede cargar en el simulador del MSX88 y ejecutarlo para comprobar su funcionamiento.

El ejemplo es el siguiente:

```
ORG 2000H
MOV BX,3000H
MOV AX,[BX]
ADD BX, 02H
MOV CX,[BX]
ADD AX,CX
PUSH AX
POP DX
HLT

ORG 3000h
DB 55h, 33h, 44h, 22h
END
```

Ejemplo para MSX88

48

En el archivo para salida a impresora de la siguiente filmina, se pueden observar 4 columnas:

- Las direcciones en las que se carga cada instrucción de máquina (columna Dir.)
- El programa ejecutable en el lenguaje de máquina (columna código máquina)
- Un índice para referenciar las líneas de la salida impresa (columna línea)
- El programa en el lenguaje Assembly (columna Código en lenguaje ensamble).

Ejemplo para MSX88

Dir.	Código máquina	Línea	Código en lenguaje assembly
		1	ORG 2000H
2000	BB 00 30	2	MOV BX,3000H
2003	8B 07	3	MOV AX,[BX]
2005	81 C3 02 00	4	ADD BX, 02H
2009	8B 0F	5	MOV CX,[BX]
200B	03 C1	6	ADD AX,CX
200D	50	7	PUSH AX
200E	5A	8	POP DX
200F	F4	9	HLT
		10	
		11	org 3000h
3000	55 33 44 22	12	db 55h, 33h, 44h, 22h
		13	END

SÍMBOLOS:

Nombre: Tipo: Valor:

Ejemplo para MSX88

El proceso de ensamblado toma el código de operación y los argumentos de la instrucción para convertirla en el código absoluto (lenguaje de máquina).

Consideremos, por ejemplo, la primera instrucción del ejemplo anterior:

MOV BX,3000H

Es una instrucción del tipo “MOV” (mover) inmediato a registro.

En la cartilla de instrucciones del procesador se tiene:

MOV = Move:

	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0	7 6 5 4 3 2 1 0
Register/memory to/from register	1 0 0 0 1 0 d w	mod reg r/m	(DISP-LO)	(DISP-HI)		
Immediate to register/memory	1 1 0 0 0 1 1 w	mod 0 0 0 r/m	(DISP-LO)	(DISP-HI)	data	data if w = 1
Immediate to register	1 0 1 1 w reg	data	data if w = 1			
Memory to accumulator	1 0 0 0 0 0 0 w	addr-lo	addr-hi			
Accumulator to memory	1 0 1 0 0 0 1 w	addr-lo	addr-hi			
Register/memory to segment register	1 0 0 0 1 1 1 0	mod 0 SR r/m	(DISP-LO)	(DISP-HI)		
Segment register to register/memory	1 0 0 0 1 1 0 0	mod 0 SR r/m	(DISP-LO)	(DISP-HI)		

- 1) COP: 1011 (es decir B_{16})
- 2) W=1 (porque al ser BX de 16 bits la operación de movimiento se hace en 16 bits)
- 3) reg es un número de registro que se obtiene de la siguiente tabla:

Ejemplo para MSX88

51

BX

AL = 8-bit accumulator
 AX = 16-bit accumulator
 CX = Count register
 DS = Data segment
 ES = Extra segment
 Above/below refers to unsigned value.
 Greater = more positive,
 Less = less positive (more negative) signed values
 if d = 1 then "to"; if d = 0 then "from".
 if w = 1 then word instruction; if w = 0 then byte instruction

if s:w = 01 then 16 bits of immediate data form the operand
 if s:w = 11 then an immediate data byte is sign extended to
 form the 16-bit operand.

if v = 0 then "count" = 1; if v = 1 then "count" in (CL)
 x = don't care

z is used for string primitives for comparison with ZF FLAG

SEGMENT OVERRIDE PREFIX

0 0 1 reg 1 1 0

if mod = 11 then r/m is treated as a REG field
 if mod = 00 then DISP = 0*, disp-low and disp-high are absent
 if mod = 01 then DISP = disp-low sign-extended to 16-bits, disp-high is absent
 if mod = 10 then DISP = disp-high: disp-low
 if r/m = 000 then EA = (BX) + (SI) + DISP
 if r/m = 001 then EA = (BX) + (DI) + DISP
 if r/m = 010 then EA = (BP) + (SI) + DISP
 if r/m = 011 then EA = (BP) + (DI) + DISP
 if r/m = 100 then EA = (SI) + DISP
 if r/m = 101 then EA = (DI) + DISP
 if r/m = 110 then EA = (BP) + DISP*
 if r/m = 111 then EA = (BX) + DISP

base indexed

indexed

base

REG is assigned according to the following table:

16-Bit (w = 1)	8-Bit (w = 0)
000 AX	000 AL
001 CX	001 CL
010 DX	010 DL
011 BX	011 BL
100 SP	100 AH
101 BP	101 CH
110 SI	110 DH
111 DI	111 BH

3) El primer byte de la instrucción queda: 1011 1011 = BB₁₆

4) Los 2 bytes restantes son el número inmediato: 3000₁₆

5) La instrucción en lenguaje absoluto queda finalmente: BB 00 30

Notar que el número 3000H se carga invertido (primero la parte baja y luego la alta).

Referencias

- Organización de los registros

Capítulo 11 apartado 11.2. Stallings. 5ta Ed.

- Formatos de instrucciones

Capítulo 10 apartado 10.3. y 10.4. Stallings. 5ta Ed.

- Links de interés

http://www.intel.com/museum/online/hist_micro/hof/index.htm

- Simulador MSX88

Descargas en página web de cátedra