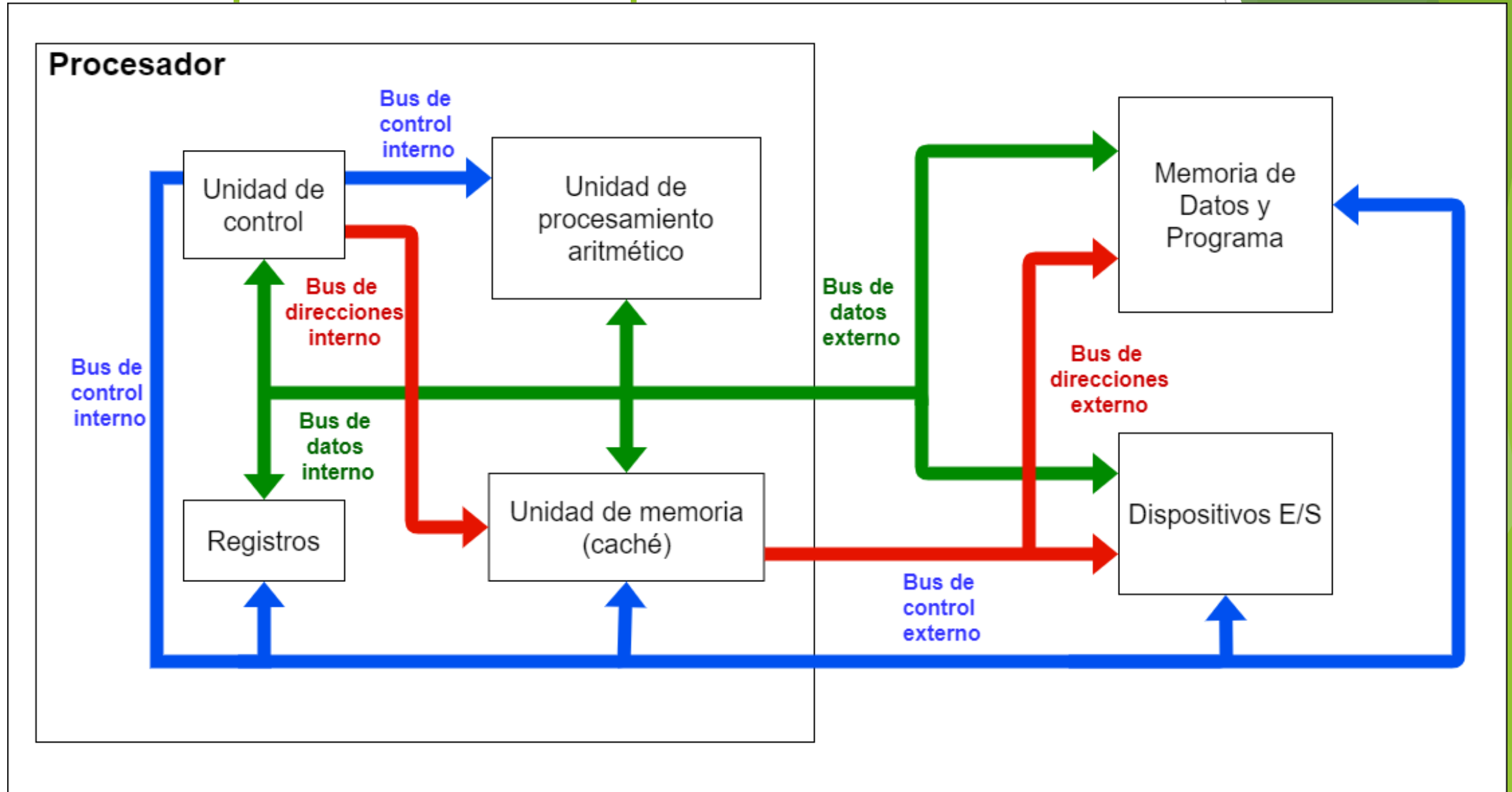


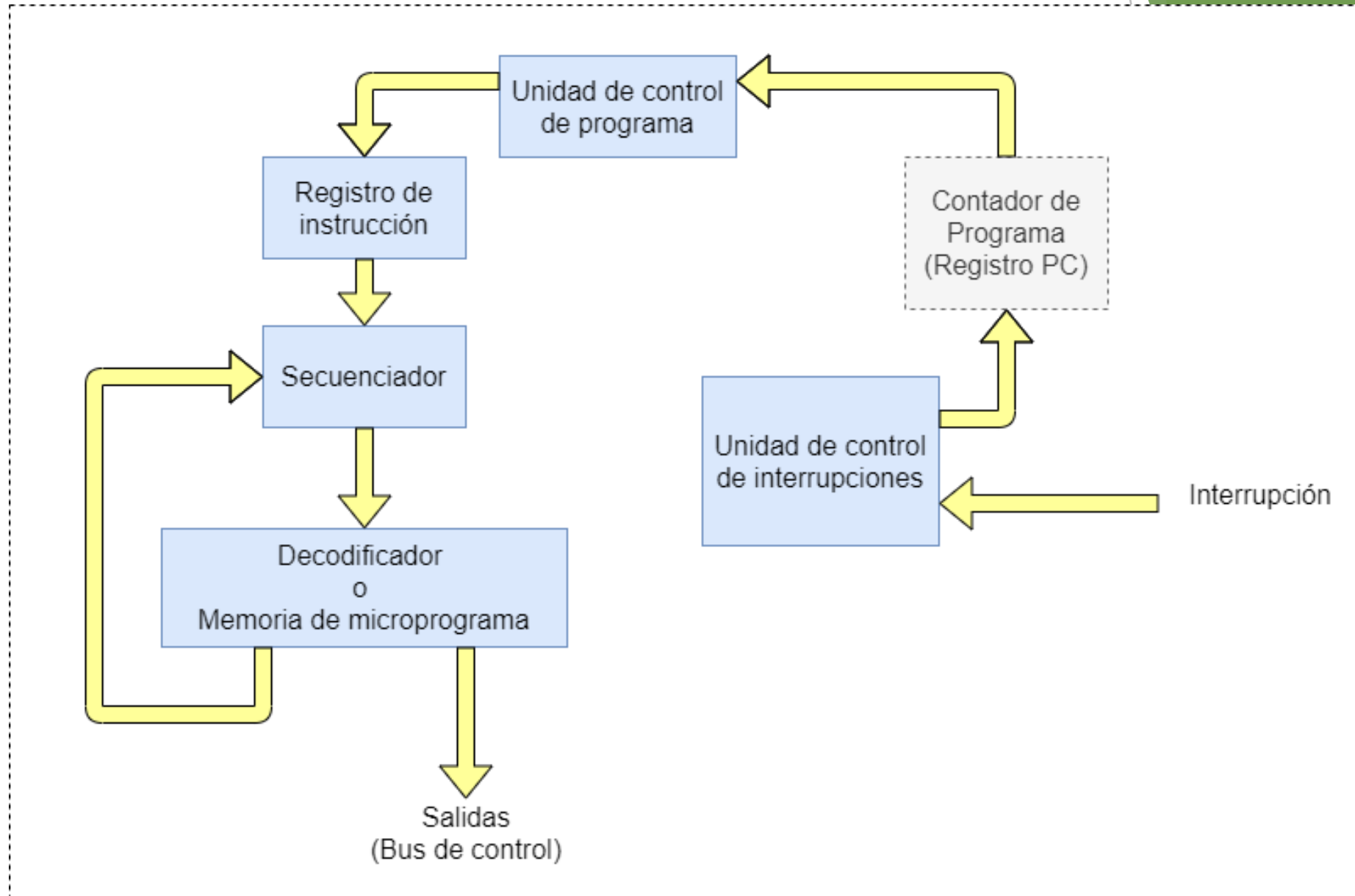
Tema 2: Presentación de un caso real (Programación en lenguaje ensamblador)

Objetivo: El alumno diseñará programas en lenguaje ensamblador para un procesador específico.

2.1: Arquitectura del procesador

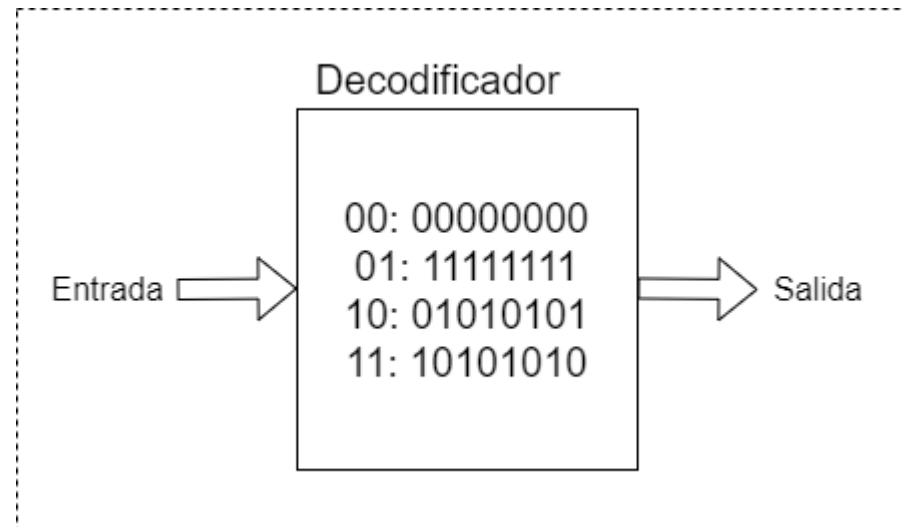


Unidad de control



Unidad de control

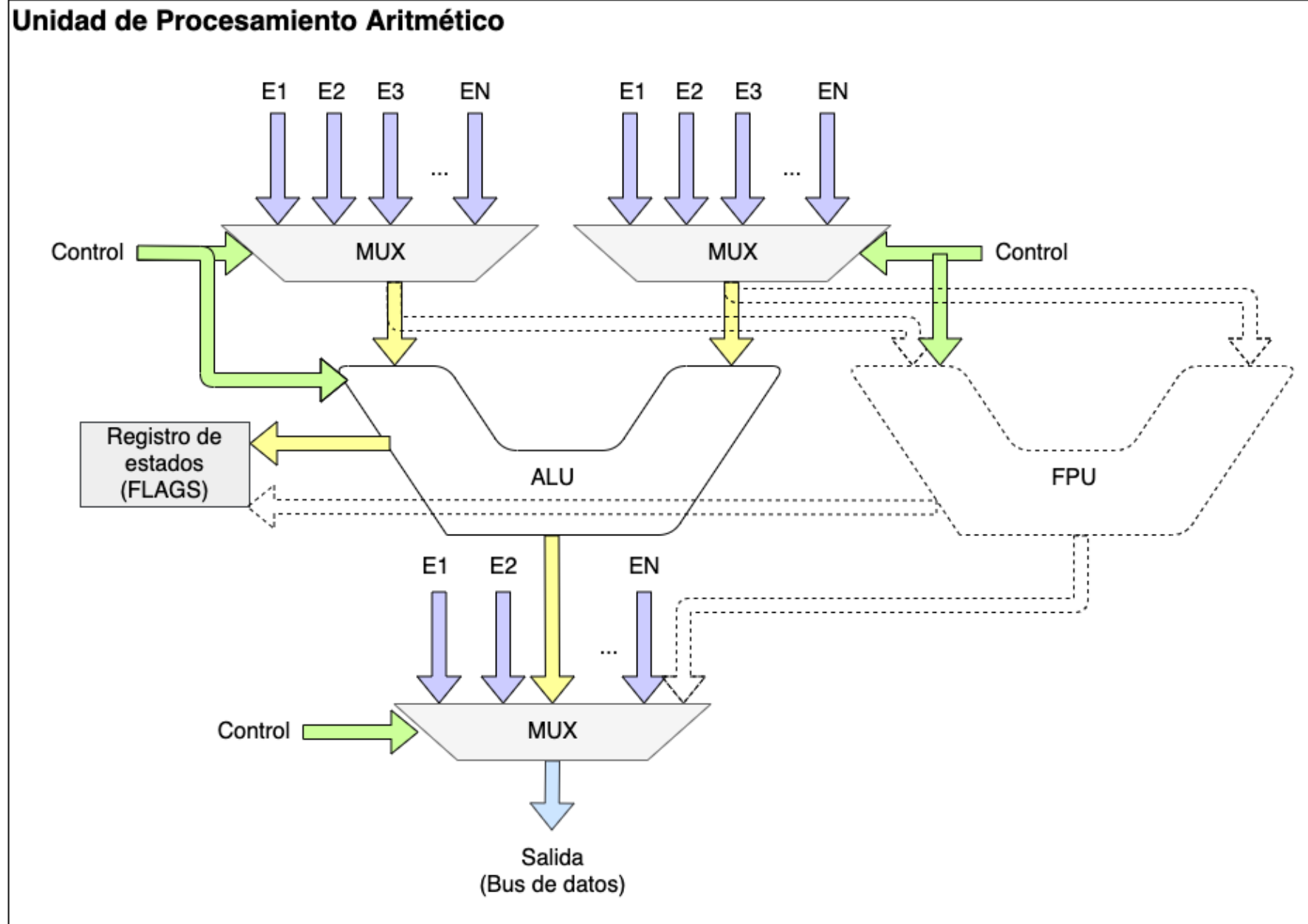
- ▶ **Secuenciador (Unidad lógica o de supervisión):** Sincroniza la ejecución de la instrucción con la velocidad de reloj.
- ▶ **Registro de instrucción:** registro que contiene la instrucción siguiente a ejecutarse (registro invisible al programador).
- ▶ **Decodificador o memoria de microprograma:** Es la lista de operaciones que se ejecutan por cada instrucción. Contiene las señales que se van a enviar dependiendo de la entrada de la instrucción



Unidad de control

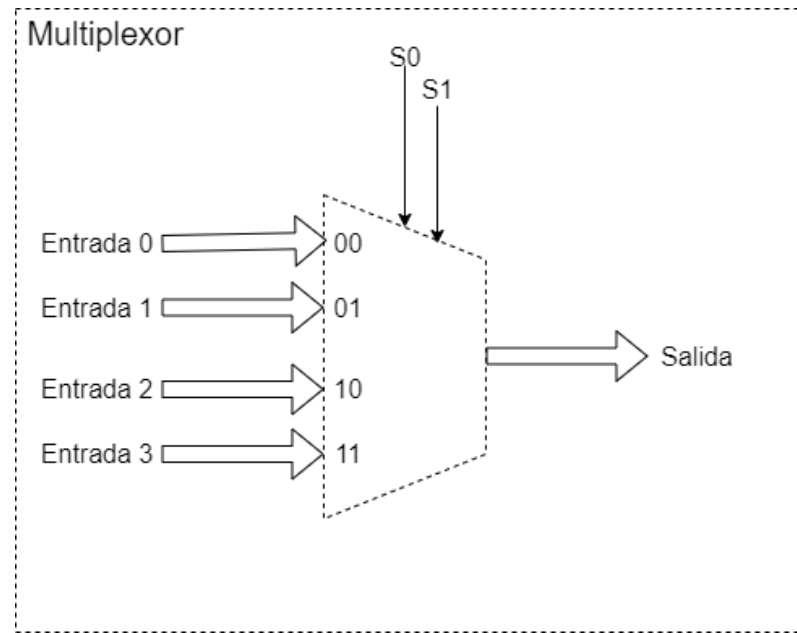
- ▶ **Unidad de control de programa:** Calcula la dirección de la siguiente instrucción a ejecutarse.
- ▶ **Unidad de control de interrupciones:** Es la unidad que se encarga del manejo de interrupciones.
 - ▶ Una **interrupción**, en computación, **es una señal de entrada** que indica que un evento importante **debe atenderse de inmediato**. El procesador va a detener su ejecución actual para atenderla.

Unidad de Procesamiento Aritmético



Unidad de Procesamiento Aritmético

- ▶ **ALU (Unidad Aritmético-Lógica):** Se encarga de realizar las operaciones aritméticas y lógicas en el procesador.
- ▶ **Multiplexor:** Es un circuito combinacional con varias entradas y una única salida de datos. La entrada pasa a la salida dependiendo de las líneas de selección



Unidad de Procesamiento Aritmético

- ▶ **FPU (*Floating-Point Unit*):** también conocido como co-procesador matemático. Se encarga de todas las operaciones matemáticas que involucran números de punto flotante o fracciones.
- ▶ **Registro de estados:** Contiene los valores que indican el estado de los componentes de la arquitectura.

Registro de estados (FLAGS, en Intel x86)

---	NT	IOP ₁	IOP ₀	O	D	I	T	S	Z	---	A	---	P	---	C
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Unidad de Procesamiento Aritmético

C (Acarreo): Suma o resta generan acarreo

P (Paridad): Número de 1's en un dato binario. 0 - paridad impar; 1 - paridad par

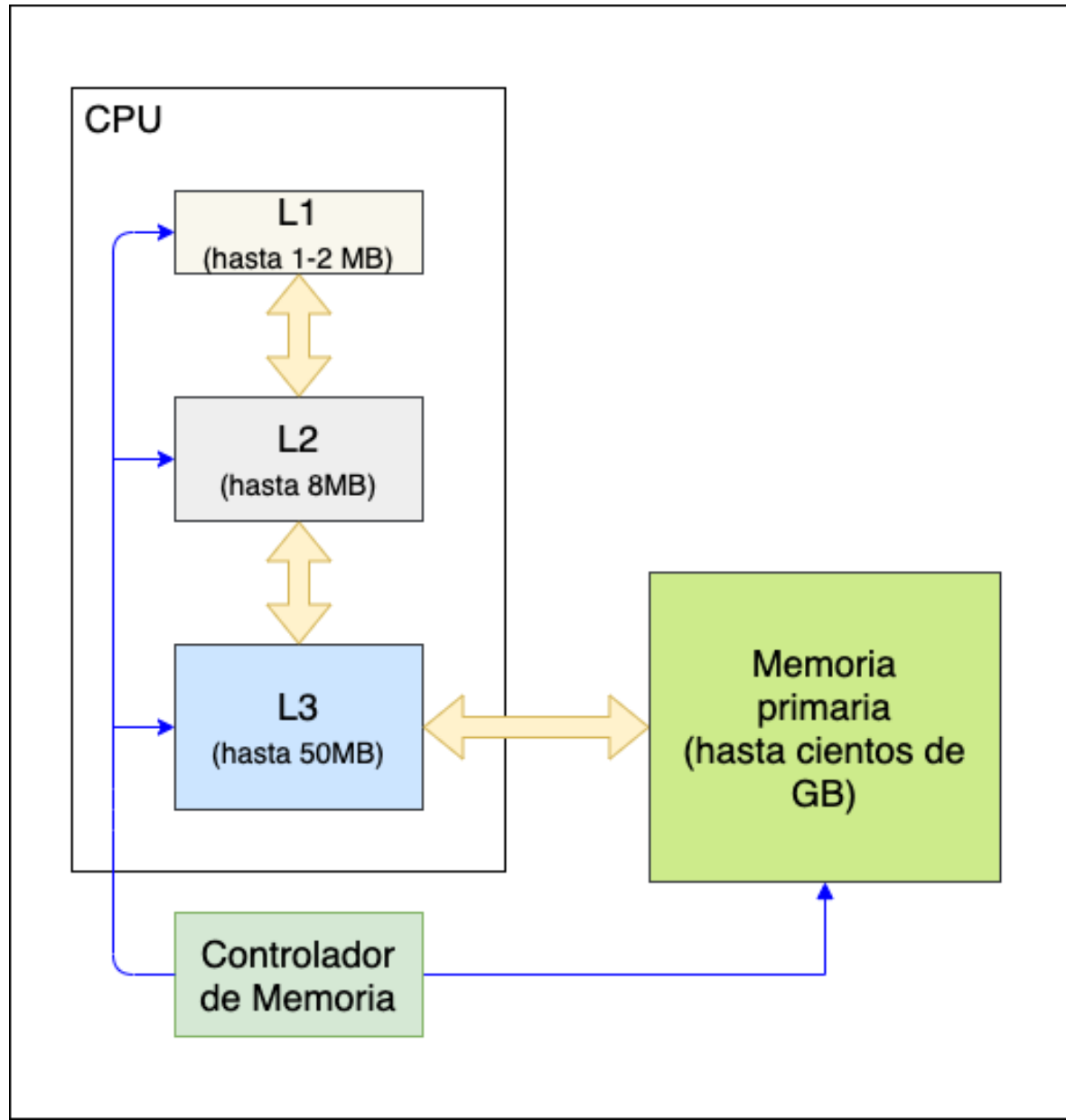
A (Acarreo auxiliar, o medio acarreo): Funciona como el acarreo pero a la mitad de los datos binarios

Z (Cero): Cuando el resultado de una operación es 0. Si resultado = 0, Z=1; si no, Z=0

S (Signo): Indica el signo del resultado de una operación aritmética. Si resultado < 0, S=1; si no, S=0

O (Desbordamiento): Cuando la operación de dos datos de un mismo tamaño requiere un dato de un tamaño mayor

Unidad de Memoria (caché)



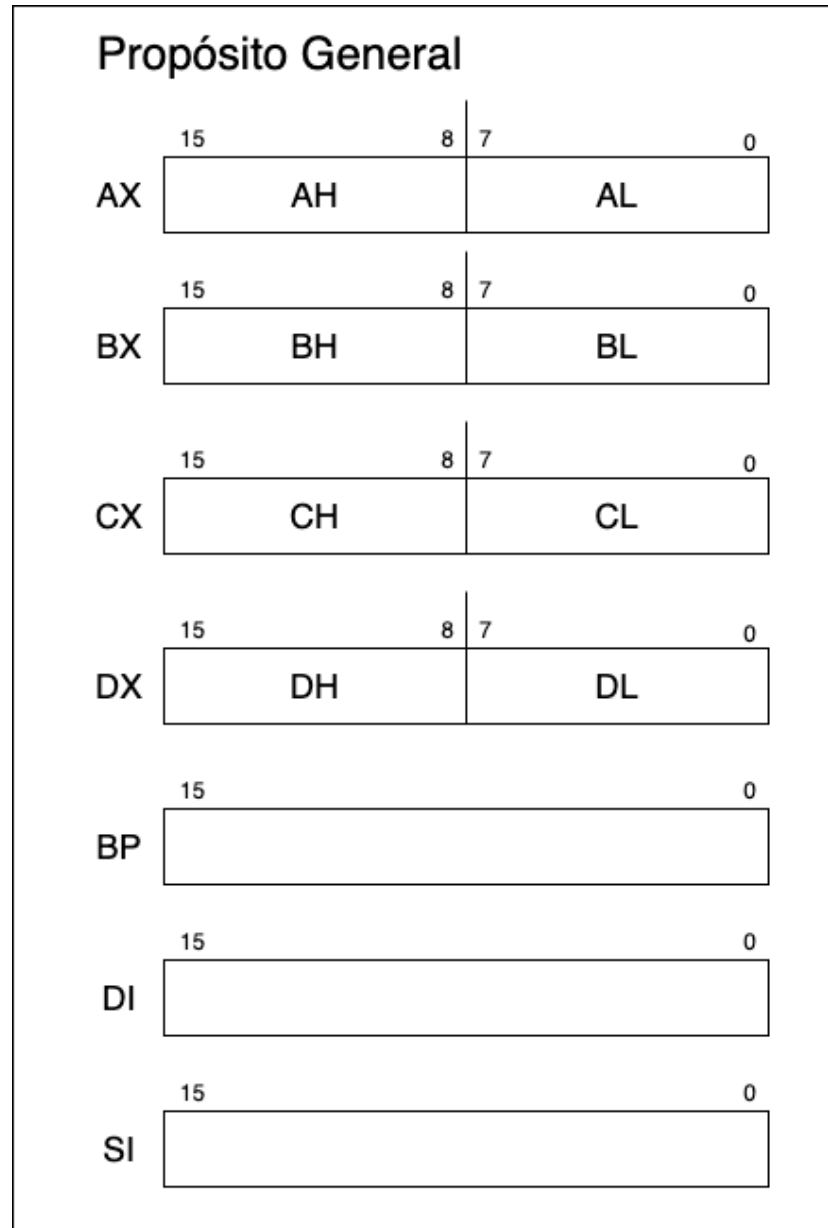
Unidad de Memoria (caché)

- ▶ La memoria caché es un **tipo de memoria específica** que está preparada para servir de **apoyo al procesador**. Actúa como un **sistema de almacenamiento de instrucciones y de datos** capaz de comunicarse con el procesador a gran velocidad.
- ▶ Dependiendo el nivel, es más veloz, pero también de menor capacidad de almacenamiento.
 - ▶ Caché L1 (nivel 1) es la más rápida, pero es la de menor tamaño, la mayoría puede llegar hasta 256 KB, pero existen algunas que llegan hasta los 2 MB.
 - ▶ Caché L2 (nivel 2) es la que sigue en velocidad, y es de mayor tamaño que la de nivel 1. Puede llegar hasta los 8 MB.
 - ▶ Caché L3 (nivel 3) es la caché más lenta dentro del procesador, pero es la de mayor tamaño, puede llegar hasta 50 MB.

Registros internos

- ▶ Propósito general
- ▶ Propósito específico
- ▶ Registros de segmento

Registros internos (propósito general)

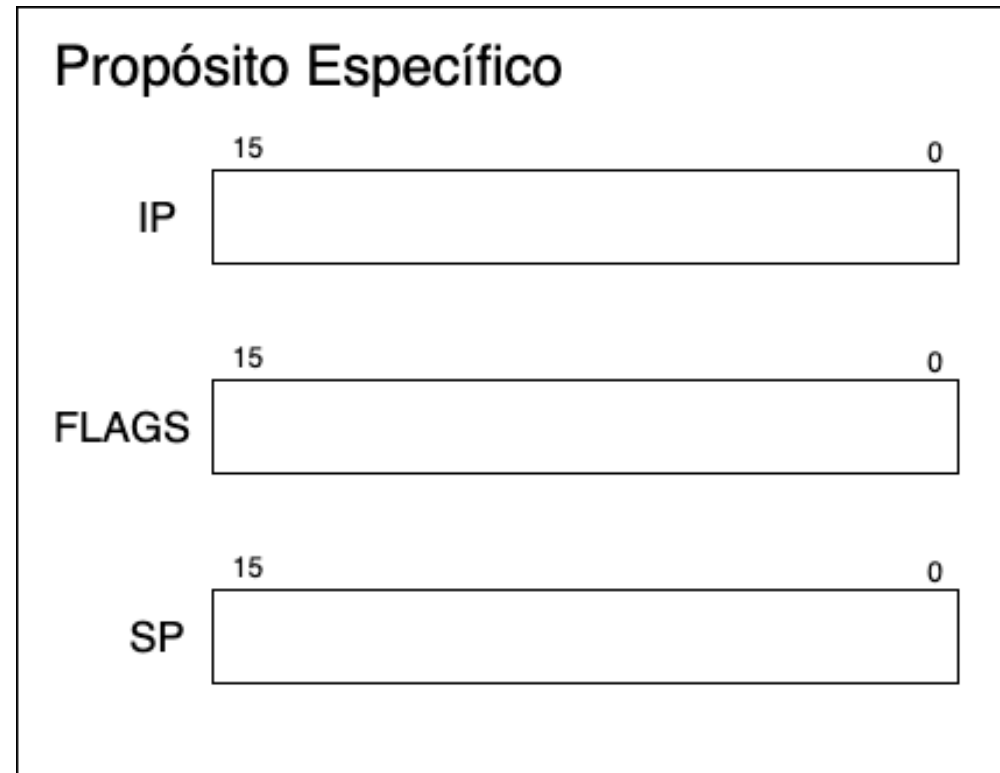


Registros internos (propósito general)

Los registros de propósito general pueden almacenar datos o direcciones de memoria, aunque en el Intel x86, estos registros pueden tener usos particulares:

- ▶ AX (Acumulador): Almacena el resultado de algunas operaciones aritméticas.
- ▶ BX (Índice Base): Apuntador a localidades de memoria de datos.
- ▶ CX (Contador): Se usa en algunas instrucciones lógicas y en loops.
- ▶ DX (Datos): Se usa en algunas operaciones aritméticas y de Entrada/Salida.
- ▶ BP (Apuntador a la Base [del *stack*]): Apuntador a la base de la pila.
- ▶ SI (Índice de origen): Se usa para apuntar al origen de operaciones de cadena.
- ▶ DI (Índice de destino): Se usa para apuntar al destino de operaciones de cadena.

Registros internos (propósito específico)



Registros internos (propósito específico)

- ▶ IP (Apuntador a Instrucciones): Direcciona la siguiente instrucción a ejecutar del segmento de código.
- ▶ SP (Apuntador a pila): Direcciona el área de memoria de la pila.
- ▶ FLAGS (Banderas): Registro de estados del procesador, utiliza una colección de bits y ciertos resultados de operaciones.

Registros internos (registros de segmento)

Registros de Segmento

CS	<div><div>15</div><div></div><div>0</div></div>
DS	<div><div>15</div><div></div></div>
SS	<div><div>15</div><div></div></div>
ES	<div><div>15</div><div></div></div>
FS	<div><div>15</div><div></div></div>

Registros internos (registros de segmento)

Generan direcciones de memoria cuando se combinan con otros registros (IP, BX, BP, SP, SI, DI).

- ▶ CS (Segmento de Código): Apuntador a la primera dirección de memoria en donde se almacena el código de un programa.
- ▶ DS (Segmento de Datos): Apuntador a la sección de memoria que contiene los datos utilizados por el programa.
- ▶ SS (Segmento de Pila): Apunta a la sección de memoria utilizada por la pila.
- ▶ ES (Segmento Extra): Apuntador a una sección de memoria adicional, que se puede utilizar para algunas instrucciones de cadena.
- ▶ FS y GS: Apuntadores a segmentos de memoria suplementarios.

2.2 Modos de direccionamiento

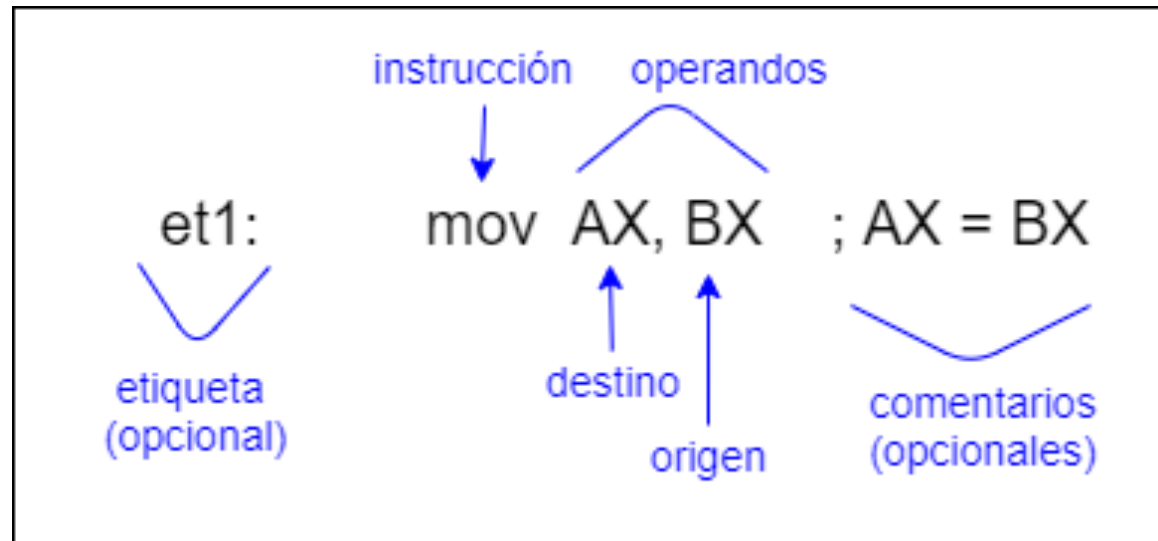
- ▶ Este término se refiere a la manera en la que se especifican los operandos de una instrucción.
- ▶ Los modos de direccionamiento especifican las reglas para interpretar o modificar el campo de dirección de la instrucción antes de que sea ejecutada.
- ▶ Los modos de direccionamiento nos sirven para:
 - ▶ Para desarrollar software más eficiente.
 - ▶ Conocer el esquema de direccionamiento (0,1,2,3 direcciones) empleado por cada instrucción.
- ▶ En arquitectura Intel x86 existen diferentes modos de direccionamiento dependiendo a qué segmento de memoria se está direccionando.
 - ▶ Modos de direccionamiento de memoria de datos.
 - ▶ Modos de direccionamiento de memoria de programa.
 - ▶ Modos de direccionamiento de memoria de la pila.

Modos de direccionamiento de memoria de datos

- ▶ Implícito
- ▶ Inmediato
- ▶ Directo
- ▶ De registros
- ▶ De registro indirecto
- ▶ De base más índice
- ▶ De registro relativo
- ▶ De base relativa más índice

Antes de comenzar:

Sintaxis de una instrucción en lenguaje ensamblador (arquitectura x86)



Operandos válidos:

- Registro a Registro
- Memoria a Registro
- Registro a Memoria
- Constante a Registro
- Constante a Memoria

```
mov ax,bx ; ax = bx
mov ax,[var] ; ax = [var]
mov [var], ax ; [var] = ax
mov ax, 5 ; ax = 0005h
mov [var], 5 ; [var] = 05h
```

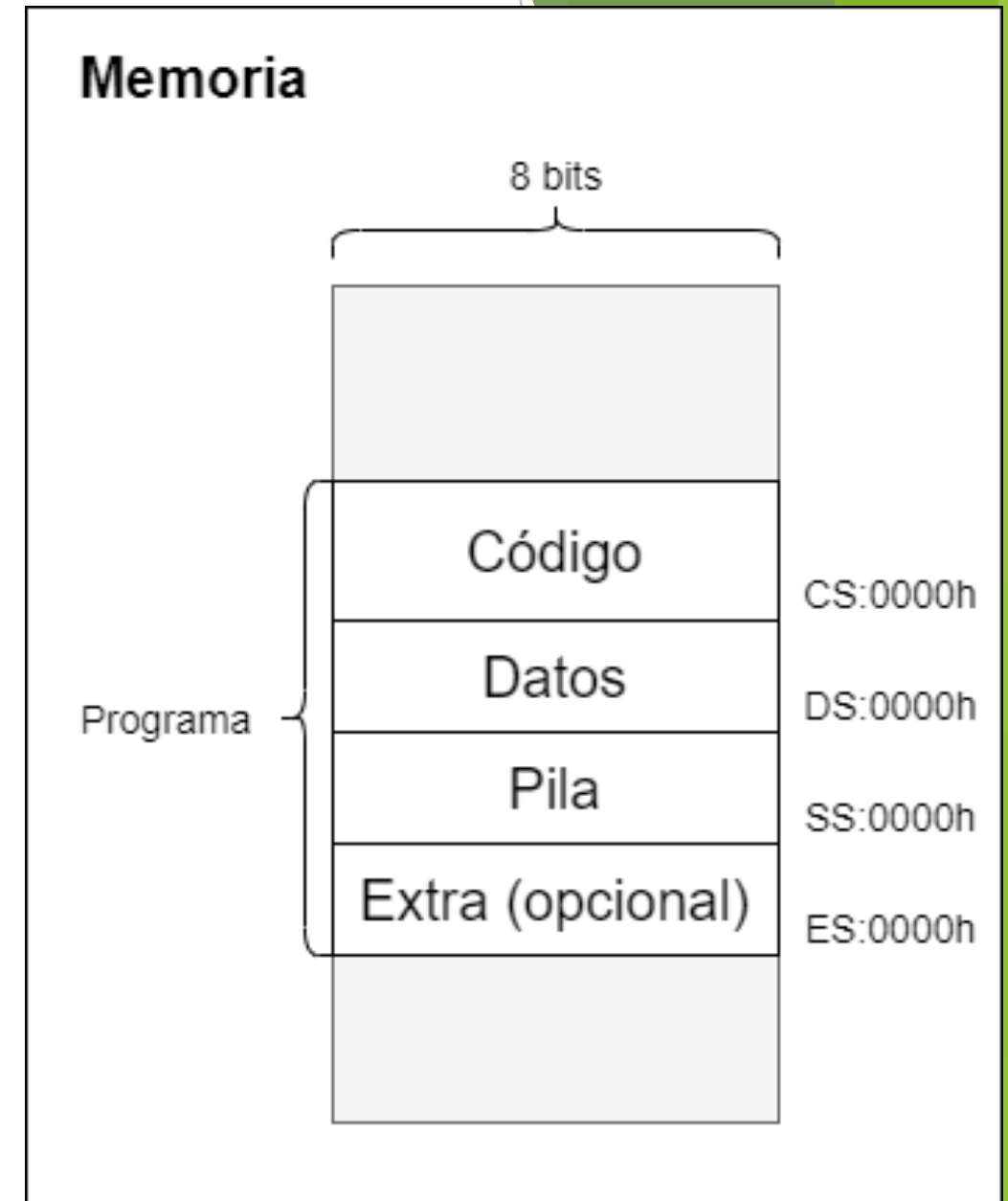
Antes de comenzar:

Sistemas numéricos en Intel x86

- ▶ En lenguaje ensamblador para arquitectura Intel x86, podemos utilizar los sistemas de numeración siguientes:
 - ▶ Decimal (default): 241 o 241d
 - ▶ Binario: 11110001b
 - ▶ Hexadecimal: F1h
 - ▶ Octal: 361o

Antes de comenzar: Memoria

- ▶ La memoria se divide en localidades, cada localidad almacena 8 bits (1 byte).
- ▶ El Sistema Operativo divide la memoria en segmentos, un segmento será un conjunto de bytes.
- ▶ Cada programa ocupa varios segmentos de memoria para almacenar información del mismo. Un programa contendrá: un segmento de código donde se almacenan las instrucciones; un segmento de datos, donde se almacenan los datos; un segmento de pila, para el uso de la pila; y un segmento extra que puede extender el uso de memoria de datos.
- ▶ Una dirección de memoria se representa por un segmento y un desplazamiento. Ejemplo: la dirección CS:0002h representa la localidad 2 de memoria del segmento de código.



Modos de direccionamiento de memoria de datos

Implícito

El operando está especificado en la instrucción misma. Es decir, no se especifica un operando, si no el procesador ya sabe sobre qué operando se aplica la operación.

► Ejemplos:

- `clc` ; clear carry, Bandera de Carry a 0
- `stc` ; set carry, Bandera de Carry a 1

Modos de direccionamiento de memoria de datos

Inmediato

Transfiere un dato inmediato (constante) a un registro o localidad de memoria.

► Ejemplos:

- `mov ax,1234h ; AX = 1234h`
- `mov [var], 12h ; ds:[var] = 12h`

Modos de direccionamiento de memoria de datos

Directo

Transfiere un dato (de tamaño byte) entre una localidad de memoria y un registro.

► Ejemplos:

- `mov ah,[var1] ; AH = [var1]`
- `mov [var2], al ; [var2] = al`
- `mov bx, ds:[var3] ; bx = [var3]`

Modos de direccionamiento de memoria de datos

De registros

Transfiere un dato entre 2 registros

No se permite cambiar el registro CS

No se permite MOV para mover registros de segmento a segmento

► Ejemplos:

► `mov ax,bx ; AX = BX`

► `mov DS, AX ; DS = AX`

► ~~`mov ss, cs`~~

Modos de direccionamiento de memoria de datos

De registro indirecto

Transfiere un dato entre un registro y una localidad de memoria direccionada por un registro índice (BX, DI, SI, BP)

► Ejemplos:

► `mov al,[bx] ; AL = ds:[BX]`

Modos de direccionamiento de memoria de datos

De base más índice

Transfiere un dato entre un registro y una localidad de memoria direccionada por un registro base (BX, BP) más un registro índice (DI, SI)

► Ejemplos:

► `mov al,[bx + di] ; AL = [BX+DI]`

Modos de direccionamiento de memoria de datos

De registro relativo

Transfiere un dato entre un registro y una localidad de memoria direccionada por un registro base o índice, más un desplazamiento (constante)

► Ejemplos:

► `mov al,[bx + 4] ; AL = [BX+4]`

Modos de direccionamiento de memoria de datos

De base relativa más índice

Transfiere un dato entre un registro y una localidad de memoria direccionada por un registro base y un índice, más un desplazamiento.

► Ejemplos:

► `mov al,[bx + di + 4]` ; $AL = [BX+DI+4]$

Modos de direccionamiento de memoria de programa

Se aplican sobre instrucciones de salto que afectan el flujo de ejecución de un programa.

- Instrucciones de salto (jmp, jc, js, jz, ja, etc.)
- Llamadas a procedimientos (call)

- ▶ Directo
- ▶ Relativo
- ▶ Indirecto

Modos de direccionamiento de memoria de programa

Directo

Comúnmente llamado ‘salto lejano’ porque puede saltar a cualquier posición de memoria de programa dentro del segmento de código.

- ▶ Ejemplos:
 - ▶ `jmp cs:0000h`
 - ▶ `jc cs:0040h`

Modos de direccionamiento de memoria de programa

Indirecto

Utiliza el contenido de un registro de 16 bits (AX, BX, CX, DX, SP, BP, DI, SI)

► Ejemplos:

- jmp BX
- jc AX

Modos de direccionamiento de memoria de programa

Relativo

Es relativo al registro IP. Se realiza haciendo desplazamientos hacia adelante o atrás, sumando un offset al registro IP.

Este tipo de salto no está disponible para el programador. Se produce por el ensamblador al utilizar etiquetas, el ensamblador calcula la dirección de la siguiente instrucción.

► Ejemplos:

- `jmp 4`
- `jc -3`

Modos de direccionamiento de memoria de la pila

La pila es importante en los microprocesadores. Puede guardar datos de manera temporal y también almacena direcciones de retorno cuando se utilizan llamadas a procedimientos.

La memoria de la pila es LIFO (*Last Input, First Output*) que describe la manera en la que se introducen y se obtienen los datos de ella.

Algunas instrucciones que hacen uso de la pila son:

- Instrucción PUSH: para colocar datos en la pila.
- Instrucción POP: para sacar datos de la pila.
- Instrucción CALL: llamada a procedimientos (el equivalente a una función o método en lenguaje de alto nivel), guarda la dirección de retorno en la pila.
- Instrucción RET: fin de procedimiento, saca la dirección de retorno de la pila.

Modos de direccionamiento de memoria de la pila

El direccionamiento de memoria de la pila se considera IMPLÍCITO, ya que se da por hecho que los datos siempre se van a introducir u obtener del tope de la pila.

Sin embargo, se pueden acceder datos de la memoria de la pila de manera DIRECTA accediendo a una dirección específica del segmento de pila SS:0000h, o INDIRECTA utilizando los registros apuntadores a la pila (SP y BP).

Modos de direccionamiento de memoria de la pila

Implícito

Introduce u obtiene los datos directamente del tope de la pila.

► Ejemplos:

- `push ax` ; Introduce el contenido del registro AX en el tope de la pila
- `pop bx` ; Obtiene el contenido del tope de la pila y lo guarda en registro BX

Modos de direccionamiento de memoria de la pila

Directo

Se transfiere un dato entre una localidad de memoria específica de la pila y un registro.

► Ejemplos:

- `mov ax,ss:[0000h]`
- `mov ss:[0000h],bx`

Modos de direccionamiento de memoria de la pila

Indirecto

Se transfiere un dato entre una localidad de memoria de pila apuntada por un registro apuntador de pila (SP, BP) y un registro.

► Ejemplos:

- `mov ax,[bp]`
- `mov ss:[sp],bx`

Ejemplo de modos de direccionamiento de memoria de datos

AX = 0000h
BX = 0001h
CX = 0002h
DI = 0003h
SI = 0002h

Instrucción	Resultado	Modo de Direccionamiento
mov ax, 5d		
mov cl, [var3]		
mov cx, ax		
mov ah, [bx+di]		
mov [bx], ah		
mov ch, [bx+5]		
mov ax, [bx+si+2]		

Memoria de Datos		
000Ah	BAh	
0009h	00h	
0008h	FEh	
0007h	00h	
0006h	71h	
0005h	54h	
0004h	12h	var5
0003h	CCh	var4
0002h	33h	var3
0001h	00h	var2
0000h	ABh	var1

Ejemplo de modos de direccionamiento de memoria de datos

AX = 0005h
BX = 0001h
CX = 0002h
DI = 0003h
SI = 0002h

Instrucción	Resultado	Modo de Direccionamiento
mov ax, 5d	AX = 0005h	Inmediato
mov cl, [var3]		
mov cx, ax		
mov ah, [bx+di]		
mov [bx], ah		
mov ch, [bx+5]		
mov ax, [bx+si+2]		

Memoria de Datos		
000Ah	BAh	
0009h	00h	
0008h	FEh	
0007h	00h	
0006h	71h	
0005h	54h	
0004h	12h	var5
0003h	CCh	var4
0002h	33h	var3
0001h	00h	var2
0000h	ABh	var1

Ejemplo de modos de direccionamiento de memoria de datos

AX = 0005h
BX = 0001h
CX = 0033h
DI = 0003h
SI = 0002h

Instrucción	Resultado	Modo de Direccionamiento
mov ax, 5d	AX = 0005h	Inmediato
mov cl, [var3]	CX = 0033h	Directo
mov cx, ax		
mov ah, [bx+di]		
mov [bx], ah		
mov ch, [bx+5]		
mov ax, [bx+si+2]		

Memoria de Datos		
000Ah	BAh	
0009h	00h	
0008h	FEh	
0007h	00h	
0006h	71h	
0005h	54h	
0004h	12h	var5
0003h	CCh	var4
0002h	33h	var3
0001h	00h	var2
0000h	ABh	var1

Ejemplo de modos de direccionamiento de memoria de datos

AX = 0005h
BX = 0001h
CX = 0005h
DI = 0003h
SI = 0002h

Instrucción	Resultado	Modo de Direccionamiento
mov ax, 5d	AX = 0005h	Inmediato
mov cl, [var3]	CX = 0033h	Directo
mov cx, ax	CX = 0005h	De registros
mov ah, [bx+di]		
mov [bx], ah		
mov ch, [bx+5]		
mov ax, [bx+si+2]		

Memoria de Datos		
000Ah	BAh	
0009h	00h	
0008h	FEh	
0007h	00h	
0006h	71h	
0005h	54h	
0004h	12h	var5
0003h	CCh	var4
0002h	33h	var3
0001h	00h	var2
0000h	ABh	var1

Ejemplo de modos de direccionamiento de memoria de datos

AX = 1205h
BX = 0001h
CX = 0005h
DI = 0003h
SI = 0002h

Instrucción	Resultado	Modo de Direccionamiento
mov ax, 5d	AX = 0005h	Inmediato
mov cl, [var3]	CX = 0033h	Directo
mov cx, ax	CX = 0005h	De registros
mov ah, [bx+di]	AX = 1205h	De base más índice
mov [bx], ah		
mov ch, [bx+5]		
mov ax, [bx+si+2]		

Memoria de Datos		
000Ah	BAh	
0009h	00h	
0008h	FEh	
0007h	00h	
0006h	71h	
0005h	54h	
0004h	12h	var5
0003h	CCh	var4
0002h	33h	var3
0001h	00h	var2
0000h	ABh	var1

Ejemplo de modos de direccionamiento de memoria de datos

Instrucción	Resultado	Modo de Direccionamiento
mov ax, 5d	AX = 0005h	Inmediato
mov cl, [var3]	CX = 0033h	Directo
mov cx, ax	CX = 0005h	De registros
mov ah, [bx+di]	AX = 1205h	De base más índice
mov [bx], ah	[0001h] = 12h	De registro indirecto
mov ch, [bx+5]		
mov ax, [bx+si+2]		

AX = 1205h
BX = 0001h
CX = 0005h
DI = 0003h
SI = 0002h

Memoria de Datos		
000Ah	BAh	
0009h	00h	
0008h	FEh	
0007h	00h	
0006h	71h	
0005h	54h	
0004h	12h	var5
0003h	CCh	var4
0002h	33h	var3
0001h	12h	var2
0000h	ABh	var1

Ejemplo de modos de direccionamiento de memoria de datos

AX = 1205h
BX = 0001h
CX = 7105h
DI = 0003h
SI = 0002h

Instrucción	Resultado	Modo de Direccionamiento
mov ax, 5d	AX = 0005h	Inmediato
mov cl, [var3]	CX = 0033h	Directo
mov cx, ax	CX = 0005h	De registros
mov ah, [bx+di]	AX = 1205h	De base más índice
mov [bx], ah	[0001h] = 12h	De registro indirecto
mov ch, [bx+5]	CX = 7105h	De registro relativo
mov ax, [bx+si+2]		

Memoria de Datos		
000Ah	BAh	
0009h	00h	
0008h	FEh	
0007h	00h	
0006h	71h	
0005h	54h	
0004h	12h	var5
0003h	CCh	var4
0002h	33h	var3
0001h	12h	var2
0000h	ABh	var1

Ejemplo de modos de direccionamiento de memoria de datos

AX = 7154h
BX = 0001h
CX = 7105h
DI = 0003h
SI = 0002h

Instrucción	Resultado	Modo de Direccionamiento
mov ax, 5d	AX = 0005h	Inmediato
mov cl, [var3]	CX = 0033h	Directo
mov cx, ax	CX = 0005h	De registros
mov ah, [bx+di]	AX = 1205h	De base más índice
mov [bx], ah	[0001h] = 12h	De registro indirecto
mov ch, [bx+5]	CX = 7105h	De registro relativo
mov ax, [bx+si+2]	AX = 7154h	De base relativa más índice

Memoria de Datos		
000Ah	BAh	
0009h	00h	
0008h	FEh	
0007h	00h	
0006h	71h	
0005h	54h	
0004h	12h	var5
0003h	CCh	var4
0002h	33h	var3
0001h	12h	var2
0000h	ABh	var1

2.3 Conjunto de Instrucciones

Lenguaje ensamblador

Lenguaje ensamblador

≠

Ensamblador

Lenguaje de programación a bajo nivel (acceso directo al hardware) que está constituido de mnemónicos.

Traductor del lenguaje ensamblador

Mnemónicos son palabras o conjuntos de caracteres que facilitan la comprensión y el entendimiento del lenguaje. Un mnemónico en lenguaje ensamblador sustituye a una operación en lenguaje máquina.

Lenguaje ensamblador

- ▶ Está específicamente diseñado para cada procesador, es decir, cada procesador tiene su propio lenguaje ensamblador.
- ▶ Cada instrucción en lenguaje ensamblador representa una instrucción en lenguaje máquina (0s y 1s).

Lenguaje ensamblador

► Características

- Es difícil de leer y entender
- Difícil de aprender
- Mantenimiento complicado
- Difícil de escribir
- Su velocidad de ejecución puede ser hasta 10 veces más rápido que un lenguaje de alto nivel
- Optimiza espacio de memoria
- Capacidad, se pueden implementar funciones difíciles o imposibles en lenguaje de alto nivel

Tipos de datos

- ▶ Se clasifican por su tamaño (longitud de bits)

- ▶ Byte - 8 bits

var1 db 12h

- ▶ Word - 16 bits = 2 bytes

var1 dw 1234h

- ▶ Doubleword - 32 bits = 4 bytes

var1 dd 12345678h

- ▶ Quadword - 64 bits = 8 bytes

var1 dq 123456789ABCDEF0h

Conjunto de instrucciones

- ▶ Operaciones de transferencia
 - ▶ MOV, LEA, PUSH, POP
- ▶ Operaciones aritméticas
 - ▶ ADD, SUB, INC, DEC, ADC, CMP, MUL, DIV
- ▶ Operaciones lógicas
 - ▶ AND, OR, NOT, XOR, TEST
- ▶ Operaciones de control de flujo
 - ▶ LOOP, CALL, JMP, JA, JAE, JE, JNE, JC, JNC, JZ, JNZ, etc.

Operaciones de transferencia

Operaciones de transferencia

MOV - *Move*

► Función

Copia el valor del operando origen al destino

► Sintaxis

```
mov destino, origen  
    {reg/mem} {reg/mem/constante}
```

► Ejemplos

```
mov ax, 1234h  
mov [var1], ax  
mov bx, [var1]
```

► Banderas afectadas

El valor de las banderas C, S, Z, A, P, y O quedan indefinidos (puede ser 0 o 1).

Operaciones de transferencia

LEA - *Load Effective Address*

► Función

Obtiene la dirección de desplazamiento de algún símbolo (variable o etiqueta) utilizado dentro del código fuente.

► Sintaxis

```
lea  destino, origen  
    {reg}    {símbolo}
```

► Ejemplos

```
lea ax, [et1]      ;AX = localidad de memoria del segmento de código  
lea bx, [var1]     ;BX = localidad de memoria del segmento de datos
```

► Banderas afectadas

No afecta al registro de banderas

Operaciones de transferencia

PUSH - *Push to stack*

► Función

Introduce un valor de 2 bytes al tope de la pila de programa. Modifica el valor del registro SP.

► Sintaxis

`push origen`
 {reg/mem/constante} (16 bits)

► Ejemplos

`push 1234h` ; constante de 16 bits
`push ax` ; registro de 16 bits
`push [var1]` ; var1 debe ser de tipo word

► Banderas afectadas

No afecta al registro de banderas

Operaciones de transferencia

POP - *Pop from stack*

► Función

Saca un valor de 2 bytes del tope de la pila de programa y lo almacena en el destino. Modifica el valor del registro SP.

► Sintaxis

```
pop destino  
    {reg/mem} (16 bits)
```

► Ejemplos

```
pop ax           ;pasa lo que hay en el tope de la pila a AX  
pop [var1]       ;pasa lo que hay en el tope de la pila a [var1] (de tipo word)
```

► Banderas afectadas

No afecta al registro de banderas

Operaciones aritméticas

Operaciones aritméticas

ADD - *Addition*

► Función

Suma dos números binarios. Suma el operando destino con el operando origen y guarda el resultado en el operando destino.

Ambos operandos deben ser del mismo tamaño.

► Sintaxis

```
add destino, origen  
    {reg/mem} {reg/mem/constante}
```

► Ejemplos

```
add ax, 1234h      ;AX = AX + 1234h  
add [var1], ah     ;[var1] = [var1] + AH, var1 debe ser de tipo byte  
add bx, [var2]     ;BX = BX + [var2], var2 debe ser de tipo word
```

► Banderas afectadas

El valor de las banderas C, S, Z, A, P, y O se modifica de acuerdo al resultado.

Operaciones aritméticas

SUB - *Subtraction*

► Función

Resta de dos números binarios. Resta el operando destino menos el operando origen y guarda el resultado en el operando destino. Ambos operandos deben ser del mismo tamaño.

► Sintaxis

```
sub destino, origen  
    {reg/mem} {reg/mem/constante}
```

► Ejemplos

```
sub ax, 1234h      ;AX = AX - 1234h  
sub [var1], ah     ;[var1] = [var1] - AH, var1 debe ser de tipo byte  
sub bx, [var2]     ;BX = BX - [var2], var2 debe ser de tipo word
```

► Banderas afectadas

El valor de las banderas C (borrow), S, Z, A, P, y O se modifica de acuerdo al resultado.

Operaciones aritméticas

INC - *Increment*

► Función

Realiza un incremento de 1 al operando destino.

► Sintaxis

```
inc destino  
    {reg/mem}
```

► Ejemplos

```
inc ax      ;AX = AX + 1  
inc bl      ;BL = BL + 1  
inc [var1]  ;[var1] = [var1] + 1, var1 puede ser de cualquier tipo
```

► Banderas afectadas

El valor de las banderas S, Z, P, y O se modifica de acuerdo al resultado.
No se afectan C ni A.

Operaciones aritméticas

DEC - *Decrement*

► Función

Realiza un decremento de 1 al operando destino.

► Sintaxis

```
dec destino  
    {reg/mem}
```

► Ejemplos

```
dec ax      ;AX = AX + 1  
dec bl      ;BL = BL + 1  
dec [var1]   ;[var1] = [var1] + 1, var1 puede ser de cualquier tipo
```

► Banderas afectadas

El valor de las banderas S, Z, P, y O se modifica de acuerdo al resultado.
No se afectan C ni A.

Operaciones aritméticas

ADC - *Addition with carry*

► Función

Suma dos números binarios más el bit de carry. Suma el operando destino con el operando origen y el valor del bit C, guarda el resultado en el operando destino. Operandos destino y origen deben ser del mismo tamaño.

► Sintaxis

```
adc destino, origen  
    {reg/mem} {reg/mem/constante}
```

► Ejemplos

```
adc ax, 1234h      ;AX = AX + 1234h + C  
adc [var1], ah     ;[var1] = [var1] + AH + C, var1 debe ser de tipo byte  
adc bx, [var2]     ;BX = BX + [var2] + C, var2 debe ser de tipo word
```

► Banderas afectadas

El valor de las banderas C, S, Z, A, P, y O se modifica de acuerdo al resultado.

Operaciones aritméticas

SBB - *Subtraction with borrow*

► Función

Resta de dos números binarios menos el bit de carry. Resta el operando destino menos el operando origen y menos el valor del bit C, guarda el resultado en el operando destino.

Operandos destino y origen deben ser del mismo tamaño.

► Sintaxis

```
sbb destino, origen  
      {reg/mem} {reg/mem/constante}
```

► Ejemplos

```
sbb ax, 1234h      ;AX = AX - 1234h - C  
sbb [var1], ah     ;[var1] = [var1] - AH - C, var1 debe ser de tipo byte  
sbb bx, [var2]     ;BX = BX - [var2] - C, var2 debe ser de tipo word
```

► Banderas afectadas

El valor de las banderas C (borrow), S, Z, A, P, y O se modifica de acuerdo al resultado.

Operaciones aritméticas

MUL - (Unsigned) *Multiplication*

► Función

Realiza la multiplicación sin signo entre un multiplicando (implícito) y un operando multiplicador (explícito). El resultado se almacena en AX si los operandos son de 8 bits, o en DX y AX si los operandos son más grandes.

► Sintaxis

`mul` **multiplicador**
{8 bits: reg/mem}
{16 bits: reg/mem}

12	→	Multiplicando
x 3	→	Multiplicador
36	→	Producto

Multiplicador	Multiplicando (implícito)	Producto
reg/mem (8 bits)	registro AL (8 bits)	AX (16 bits)
reg/mem (16 bits)	registro AX (16 bits)	DX:AX (32 bits)
reg/mem (32 bits)*	registro EAX (32 bits)	EDX:EAX (64 bits)

*existe pero no se abarcará en clase

Operaciones aritméticas

MUL - (Unsigned) *Multiplication*

- ¿Por qué el producto es el doble de tamaño que los operandos?

Supongamos dos números binarios del mismo tamaño (4 bits), donde el valor máximo de cada uno es 1111.

$$\begin{array}{r} 1111 \\ \times 1111 \\ \hline 1111 \\ + 1111 \\ 1111 \\ 1111 \\ \hline 11100001 \end{array}$$

Haciendo la multiplicación manualmente del valor máximo de cada números se puede observar que el resultado requiere 8 bits.

Asumimos que en el peor caso requeriremos 8 bits al hacer la multiplicación, a partir del máximo obtenido de la multiplicación, consideramos la longitud de ese resultado como base para los demás valores, pues éstos se podrán representar con los mismos 8 bits.

Operaciones aritméticas

MUL - (Unsigned) *Multiplication*

► Ejemplos

```
.data
var1    db    20h
var2    dw    1234h
.code
```

...

```
mov al, 0FFh    ;AL = FFh
mov bl, 0FFh    ;BL = FFh
mul bl          ;AX = BL * AL = FFh * FFh = FE01h
```

...

```
mov al, 35h     ;AL = 35h
mul [var1]       ;AX = [var1] * AL = 20h * 35h = 06A0h
```

...

```
mov ax, 1200h   ;AX = 1200h
mul [var2]       ;DX:AX = [var2] * AX = 1234h * 1200h = 0147A800h
                 ;DX = 0147h, AX = A800h
```

Operaciones aritméticas

MUL - (Unsigned) *Multiplication*

Banderas afectadas

c: } Si la parte alta es diferente de 0, entonces C = 1 y O = 1
o: } Si la parte alta es igual a 0, entonces C = 0 y O = 0

z: ?

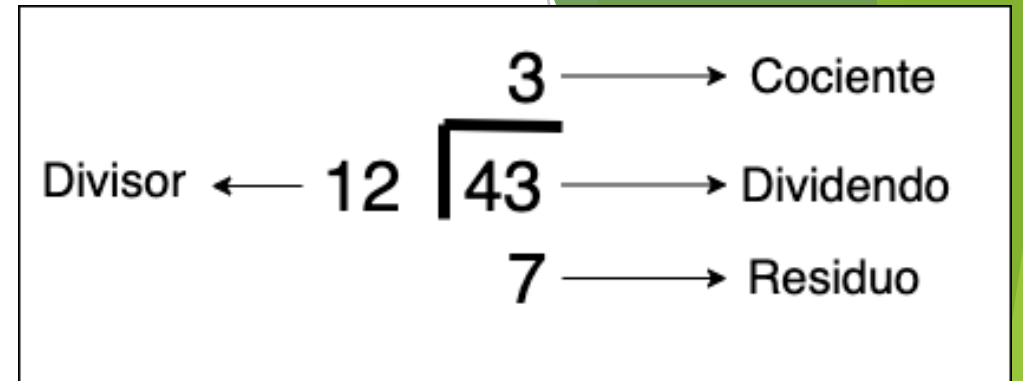
p: ?

s: ?

a: ?

Operaciones aritméticas

DIV - (Unsigned) *Division*



► Función

Realiza la división sin signo entre un dividendo (implícito) y un operando divisor (explícito). El resultado de la operación constará de dos partes: el cociente y el residuo. El cociente se almacena en la parte baja y el residuo en la parte alta. Dependiendo el tamaño de los operandos, el resultado se guardará en AX y DX (ver tabla a continuación).

► Sintaxis

`div divisor`
{8 bits: reg/mem}
{16 bits: reg/mem}

Divisor	Dividendo (implícito)	Cociente	Residuo
reg/mem (8 bits)	registro AX (16 bits)	AL (8 bits)	AH (8 bits)
reg/mem (16 bits)	registro DX:AX (32 bits)	AX (16 bits)	DX (16 bits)
reg/mem (32 bits)*	registro EDX: EAX (64 bits)	EAX (32 bits)	EDX (32 bits)

*existe pero no se abarcará en clase

Operaciones aritméticas

DIV - (Unsigned) *Division*

► Ejemplos

```
.data
var1    db    20h
var2    dw    1234h
.code
```

...

```
mov ax, 0FE02h    ;AX = FE02h
mov bl, 0FFh      ;BL = FFh
div bl            ;AL = AX / BL = FE02h / FFh = FFh
                  ;AH = AX % BL = FE02h % FFh = 01h
```

...

```
mov ax, 06A5h     ;AX = 06A5h
div [var1]         ;AL = AX / [var1] = 06A5h / 20h = 35h
                  ;AH = AX % [var1] = 06A5h % 20h = 05h
```

...

```
mov dx, 0147h     ;DX = 0147h
mov ax, 0A80Ah     ;AX = A80Ah
div [var2]         ;AX = DX:AX / [var2] = 0147A80Ah / 1234h = 1200h
                  ;DX = DX:AX % [var2] = 0147A80Ah % 1234h = 000Ah
```

Operaciones aritméticas

DIV - (Unsigned) *Division*

► Banderas afectadas

c: ?

o: ?

z: ?

p: ?

s: ?

a: ?

NOTA: División entre 0 produce una interrupción

Operaciones aritméticas

CMP - Arithmetic comparison

► Función

Compara dos números binarios restándolos. Resta el operando destino menos el operando origen. A diferencia de SUB, el resultado se descarta.

► Sintaxis

```
cmp destino, origen  
    {reg/mem} {reg/mem/constante}
```

► Ejemplos

```
cmp ax, 1234h      ;AX - 1234h, afecta banderas  
cmp [var1], ah     ;[var1] - AH, var1 debe ser de tipo byte, afecta banderas  
cmp bx, [var2]     ;BX - [var2], var2 debe ser de tipo word, afecta banderas
```

► Banderas afectadas

El valor de las banderas C (borrow), S, Z, A, P, y O se modifica de acuerdo al resultado.

Operaciones lógicas

IMPORTANTE: Las operaciones lógicas se realizan bit a bit y los operandos deben ser del mismo tamaño

Operaciones lógicas

AND - *Logical AND*

► Función

Conjunción lógica entre dos operandos. Se realiza la conjunción entre un operando destino y un operando origen, el resultado se almacena en el operando destino.

► Sintaxis

`and destino, origen`
{reg/mem} {reg/mem/constante}

► Ejemplos

`and ax, 00FFh`
`and [var1], ah`
`and bx, [var2]`

A	B	A and B
0	0	0
0	1	0
1	0	0
1	1	1

Banderas afectadas

c: 0

o: 0

s: }

z: }

p: }

a: Indefinido

Dependen del
resultado

Operaciones lógicas

OR - *Logical OR*

► Función

Disyunción lógica entre dos operandos. Se realiza la disyunción entre un operando destino y un operando origen, el resultado se almacena en el operando destino.

► Sintaxis

or destino, origen
 {reg/mem} {reg/mem/constante}

► Ejemplos

or al, 30h
or [var1], ah
or bh, al
or bx, [var2]

A	B	A or B
0	0	0
0	1	1
1	0	1
1	1	1

Banderas afectadas

c: 0

o: 0

s: }

z: }

p: }

a: Indefinido

Dependen del
resultado

Operaciones lógicas

XOR - *Logical XOR (eXclusive OR)*

► Función

Disyunción exclusiva lógica entre dos operandos. Se realiza la disyunción exclusiva entre un operando destino y un operando origen, el resultado se almacena en el operando destino.

► Sintaxis

```
xor destino, origen  
      {reg/mem} {reg/mem/constante}
```

► Ejemplos

```
xor ax, bx  
xor [var1], ah  
xor ax, 0000h  
xor bx, [var2]
```

A	B	A xor B
0	0	0
0	1	1
1	0	1
1	1	0

Banderas afectadas

c: 0
o: 0
s: }
z: } Dependenden del
p: } resultado
a: Indefinido

Operaciones lógicas

NOT - *Logical NOT (Complemento a 1)*

► Función

Aplica el complemento a 1 a un operando. Se realiza el complemento a 1 del operando destino y se almacena el resultado en el mismo operando.

► Sintaxis

```
not destino  
  {reg/mem}
```

► Ejemplos

```
not ax  
not [var1]
```

A	not A
0	1
1	0

Banderas afectadas

No afecta el registro de banderas

Operaciones lógicas

NEG - *Arithmetic NOT (Complemento a 2)*

► Función

Aplica el complemento a 2 a un operando. Se realiza el complemento a 2 del operando destino y se almacena el resultado en el mismo operando.

► Sintaxis

```
neg destino  
  {reg/mem}
```

► Ejemplos

```
neg ax  
neg [var1]
```

Banderas afectadas

c: 0 y o: 0, si destino = 0

c: 1 y o: 1, si destino \neq 0

s:

z:

p:

a:



Dependen del
resultado

Operaciones lógicas

TEST - *Logical comparison AND*

► Función

Conjunción lógica entre dos operandos. Se realiza la conjunción entre un operando destino y un operando origen. A diferencia de AND, en TEST el resultado se descarta.

► Sintaxis

```
test destino, origen  
    {reg/mem} {reg/mem/constante}
```

► Ejemplos

```
test ax, 00FFh  
test [var1], ah  
test bx, [var2]
```

Banderas afectadas

c: 0	
o: 0	
s:	Dependen del resultado
z:	
p:	
a: Indefinido	

Operaciones de control de flujo

Operaciones de control de flujo

JMP - Salto incondicional

► Función

Cambia el flujo de ejecución de un programa hacia una localidad de memoria de código especificada.

► Sintaxis

`jmp etiqueta`

► Ejemplos

```
mov ax,10d  
add cx,ax  
jmp et1  
sub cx,bx  
inc cx
```

```
et1:  
mov [var1],cx
```

Banderas afectadas

No afecta el registro de banderas

Operaciones de control de flujo

LOOP - Salto condicional

► Función

- Ejecuta un ciclo dependiendo del valor de CX.
- LOOP revisa el contenido de CX, si $CX \neq 0$, hace el salto y decrementa CX en 1.
- En un ciclo cerrado, LOOP se ejecutará CX veces

► Sintaxis

loop etiqueta

Banderas afectadas

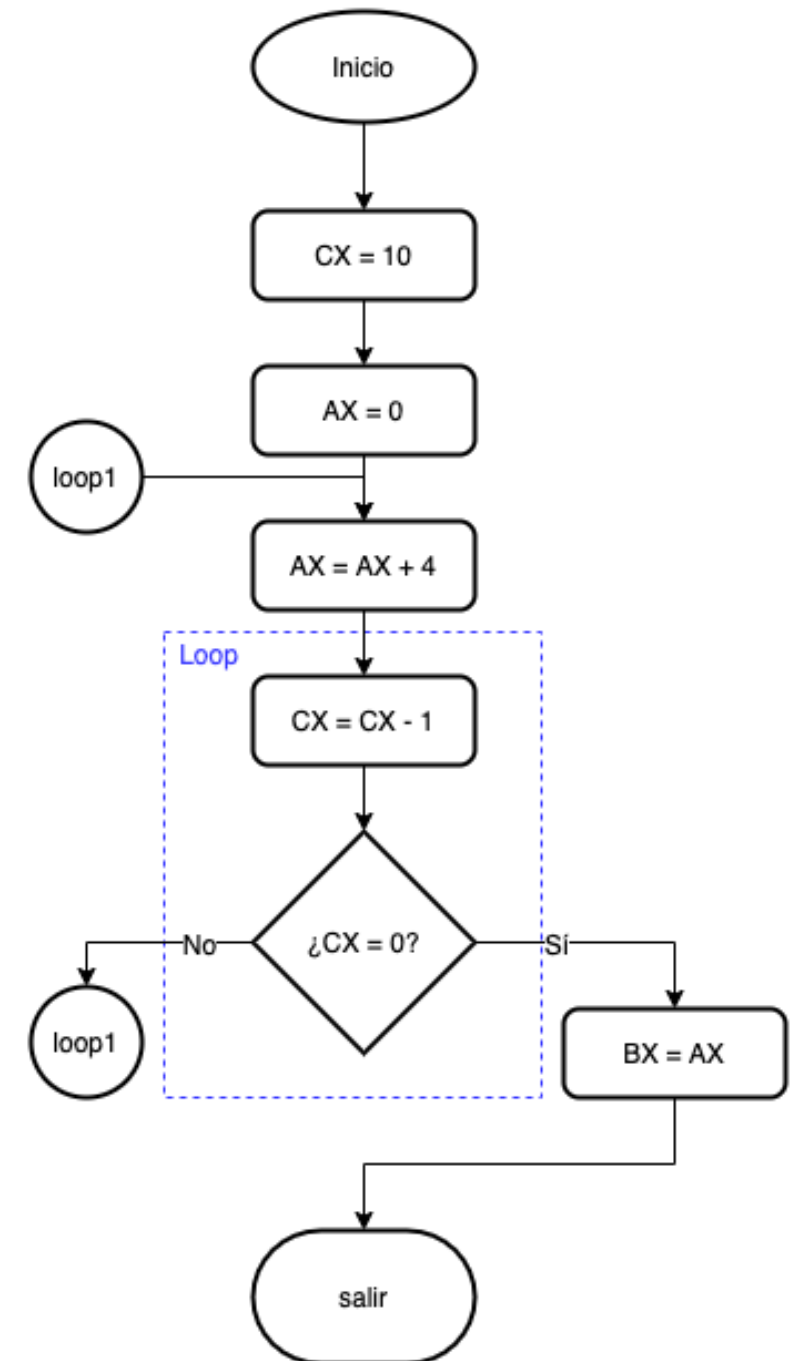
No afecta el registro de
banderas

Operaciones de control de flujo

LOOP - Salto condicional

Ejemplos

```
...  
mov ax,0      ;AX = 0000h  
mov cx,10d    ;CX = 000Ah  
loop1:  
add ax,4      ;AX = AX + 4  
loop loop1    ;Si CX != 0, salta a 'loop1' y CX = CX - 1  
mov bx,ax     ;BX = AX  
salir:  
....
```



Operaciones de control de flujo

Jump if... - Saltos condicionales

► Función

Conjunto de **instrucciones de salto condicional**.

Estas instrucciones de salto condicional hacen **uso del registro de banderas** para validar condiciones. Regularmente, estas instrucciones de salto están precedidas por una instrucción **CMP** o **TEST**, aunque también se pueden preceder por cualquier instrucción aritmética o lógica.

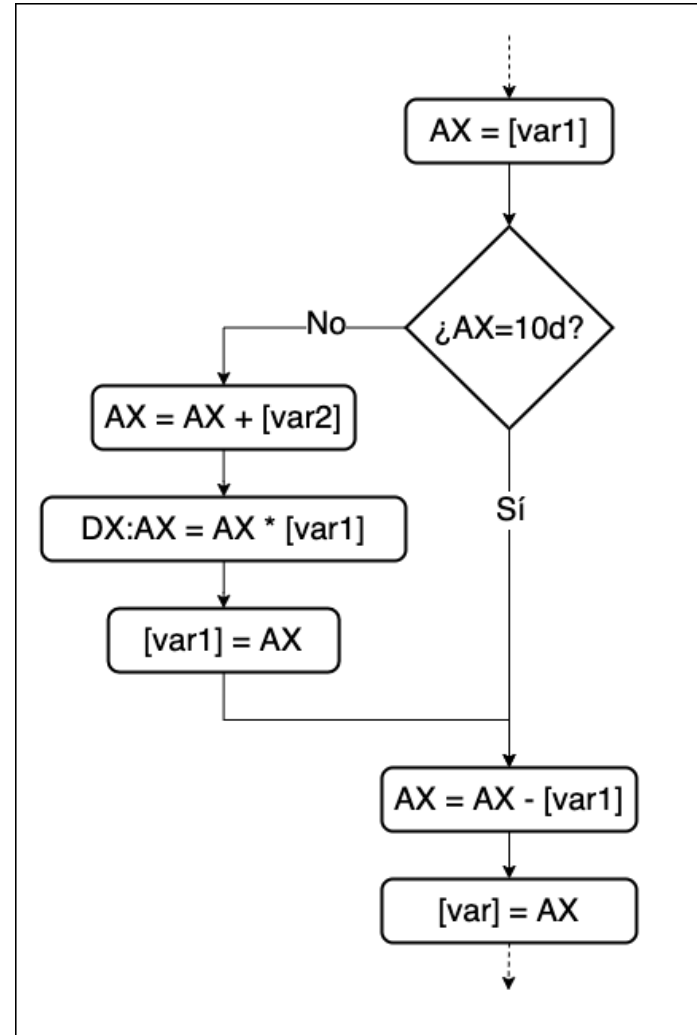
Instrucción	Descripción
ja	Jump if above (C=0 y Z=0)
jae	Jump if above or equal (C=0)
jb	Jump if below (C=1)
jbe	Jump if below or equal (C=1 y Z=1)
jg	Jump if greater (Z=0 y S=0)
jge	Jump if greater or equal (S=0)
jc	Jump if carry flag set (C=1)
js	Jump if sign flag set (S=1)
jz	Jump if zero flag set (Z=1)
jo	Jump if overflow flag set (O=1)
je	Jump if equals (Z=1)
jnz	Jump if not zero set (Z=0)
...	

Operaciones de control de flujo

Jump if... - Saltos condicionales

► Ejemplo:

```
...  
mov ax,[var1]  
cmp ax,10  
je flujo  
add ax,[var2]  
mul [var1]  
mov [var1],ax  
flujo:  
sub ax,[var1]  
mov [var1],ax  
...
```

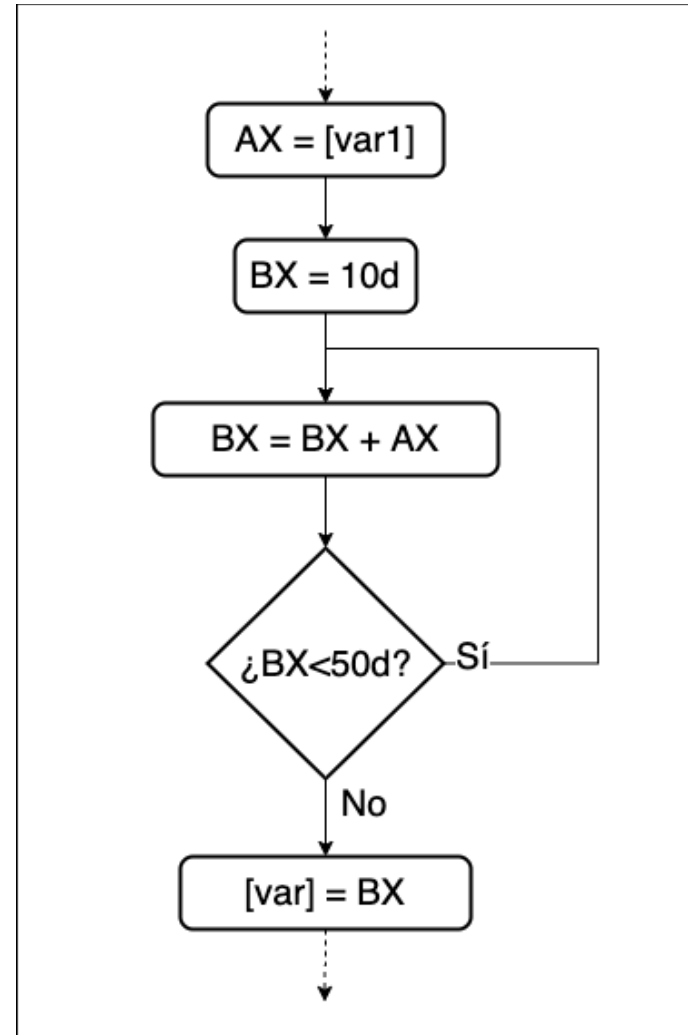


Operaciones de control de flujo

Jump if... - Saltos condicionales

► Ejemplo:

```
...  
mov ax,[var1]  
mov bx,10  
flujo1:  
add bx,ax  
cmp bx,50  
jb flujo1  
mov [var],bx  
...
```



Operaciones de control de flujo

Jump if... - Saltos condicionales

► Ejemplo:

```
...  
mov ax,[var1]  
mov bx,10  
cmp ax,bx  
jae flujo1  
sub ax,bx  
jmp flujo2  
flujo1:  
add ax,bx  
flujo2:  
mov [var],ax  
...
```

