

# Diseño de una computadora digital

---

Primera Parte

Profesora Silvana Panizzo

# Diseño de una computadora digital

---

## OBJETIVOS:

- Comprender el funcionamiento de una computadora básica en relación con su diseño interno.
- Demostrar la importancia del estudio del Algebra de Boole y la teoría de circuitos lógicos.
- Ver un modelo de computadora con una estructura muy sencilla para entender su funcionamiento.
- Para lograr entender la estructura interna de la computadora se hará referencia a **algunas instrucciones básicas** del lenguaje de **programación Assembler**, que permitirán entender el movimiento de las instrucciones o los datos de un programa a través de los registros internos que componen esta computadora.

# Diseño de una computadora digital

---

El *diseño de una computadora digital* es:

La *organización de hardware* relacionado por rutas de control y datos.

Permite *el flujo de señales binarias* para transformar datos de entrada en información útil al usuario.

Su diseño *es abstracto* ya que se ocupa de *ensamblar* los módulos que la componen.

# Módulo de cálculo en una computadora digital

---

Un *módulo*:

- Está constituido por una **configuración determinada de compuertas**.
- Donde **hay circuitos** que son:
  - abstractos
  - **cajas negras** que cumplen determinada función lógica
- Cada **módulo realiza una o varias operaciones** sobre datos codificados en **binario**, que se almacenan en registros asociados al módulo mientras dura la operación.
- Si dentro de un modulo, una **operación se aplica a un registro esta se** denomina **microoperación y se activa en un instante de tiempo sincronizado por los pulsos del reloj**

# Módulo de cálculo en una computadora digital

## *Ejemplo: Sumador binario paralelo*

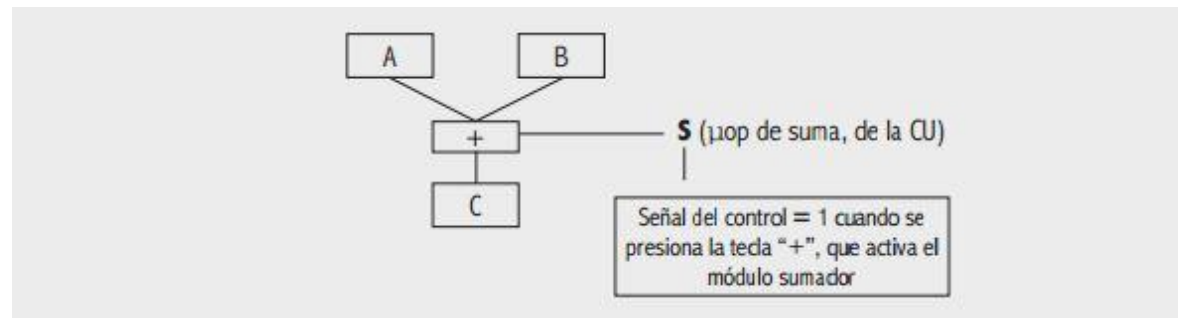


Diagrama de bloque de un dispositivo que suma los bits de los registros A y B.

- **Función:** operar datos binarios para obtener en la salida el resultado de su suma.
- Los *datos de entrada* se almacenan en forma temporal en los registros A y B y, tras la orden de *comando S*, el *resultado* se obtiene sobre el registro C.
- La orden de comando es la microoperación de suma que habilita al registro C para que actúe de receptor del resultado.
- La orden puede ser una señal “1” generada por otro módulo, cuya función es “dar órdenes” en el caso de que este sumador pertenezca a una computadora

# Lenguaje de Alto Nivel vs. Lenguaje de Bajo Nivel

Lenguaje de alto nivel

```
#include <stdio.h>
```

```
Int main( )
```

```
{
```

```
    int A, int B, long int C;
```

```
    A = 1;      (Guarda en MP en "dir 1")
```

```
    B = 2;      (Guarda en MP en "dir 2")
```

```
    C= A+B;     (Guarda en "dir 3" el resultado)
```

```
    printf( "%ld", C); (Muestra resultado  
                        guardado en "dir 3")
```

```
}
```

En Memoria

XXX0



Variable A

XXX4



Variable B

XXX7



Variable C

En ejecución

Variables en Memoria

# Lenguaje de Alto Nivel vs. Lenguaje de Bajo Nivel

- Se **necesita un programa traductor**.
- Puede surgir un **ejecutable**
- **Por cada sentencia source o fuente** puede haber **una o n sentencias en lenguaje de maquina**.

## Lenguaje de alto nivel

```
Int main( )  
{  
    int A, int B;  
    A = 1;      (Guarda en MP en "dir 1")  
    B = 2;      (Guarda en MP en "dir 2")  
    C = A+B;     (Guarda en "dir 3" el resultado)  
    printf( "%d", C); (Muestra resultado  
                      guardado en "dir 3")  
}
```

## Lenguaje simbólico de bajo nivel (Assembler)

Son 3 instrucciones en Assembler  
para obtener el mismo resultado.

13E0:0100 mov ah,[0300]  
13E0:0104 add ah,[0301]  
13E0:0108 mov [0400], ah

## En Memoria: Lenguaje de maquina: Binario

### Código de instrucción

13E0:0100 8A260003  
13E0:0104 02260103  
13E0:0108 88260004

La maquina no puede entender el  
lenguaje simbólico y se debe traducir a  
lenguaje de maquina, es decir a binario.

# Lenguaje de Alto Nivel vs. Lenguaje de Bajo Nivel

Tanto el *lenguaje simbólico de alto nivel* como el *lenguaje de bajo nivel (Assembler)* permiten obtener “*las mismas instrucciones*” en *lenguaje de maquina*, las que el procesador “entiende” y puede ejecutar.

## ***Maximizar eficiencia de la computadora:***

- Para un experto en Informática, *dos buenas herramientas* son el conocimiento de las *características de diseño de su computadora* y el aprovechamiento de las *facilidades del sistema operativo*, o sea, maximizar la eficiencia de la computadora.



# Lenguaje de Alto Nivel Lenguaje de Bajo Nivel

---

## **Programa:**

- ✓ *Conjunto ordenado de instrucciones que resuelve una tarea.*

## **Compiladores:**

- ✓ *Las herramientas de compilación traducen las sentencias de alto nivel en instrucciones de bajo nivel antes de su ejecución, creando lo que se denomina un ejecutable o código binario para luego ser ejecutado.*
- ✓ *Por ejemplo: Un archivo .exe*

## **Interpretes:**

- ✓ *Los interpretes realizan la traducción de las sentencias de alto nivel en instrucciones de bajo nivel en momento de ejecución.*
- ✓ *Por ejemplo: La interpretación de código HTML por un navegador, el navegador realiza la tarea de interprete.*

# Relación entre el diseño del hardware y la ejecución de instrucciones

---

La ***programación en lenguaje de maquina implica:***

- El conocimiento del tipo de instrucciones en **código de máquina** (o **código nativo**) a las que obedece la computadora.

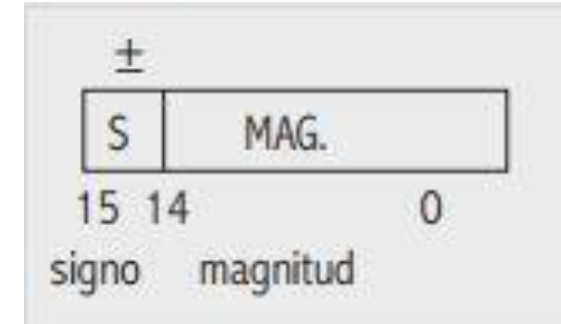
## ***Assembler***

- Para ***simplificar el trabajo*** con ***largas secuencias binarias*** a la hora de programar surge un lenguaje simbólico de ***bajo nivel*** conocido como ***Assembler***
- ***Cada computadora y sus compatibles*** tiene su propio assembler.
- Tiene ***intima relación con el diseño***.
- ***Estudiamos Assembler para comprender más la unidad de control (CU) y para entender las interacciones entre lo físico y lo lógico.***

# Formato de Datos vs Formato de instrucción

- **Formato de datos** →

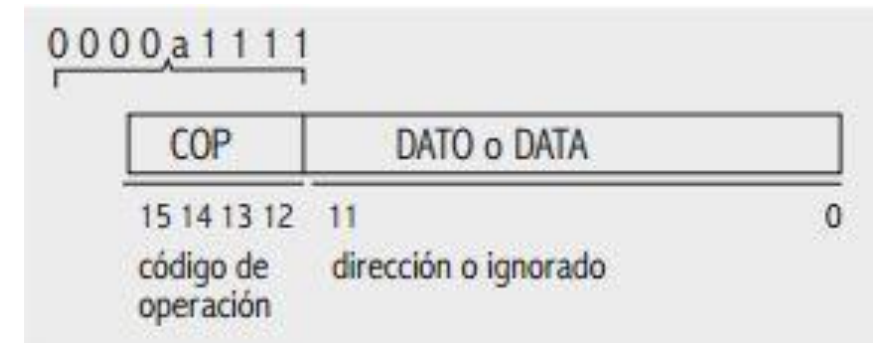
- Los datos son de tipo entero signados de 16 bits ( 1 para signo y 15 para la magnitud)



Formato de datos

- **Formato de instrucción:** →

- Las instrucciones se almacenan como palabras consecutivas a partir de la palabra 000H



Formato de instrucción

# Instrucciones

---

Operación expresada mediante la codificación binaria de cadenas de unos (1) y ceros (0).

Se le denomina lenguaje máquina

El lenguaje máquina es distinto para cada computador. Excepto cuando existe compatibilidad entre familias

Repertorio de instrucciones o set de instrucciones: Conjunto de órdenes que puede ejecutar un computador

Lenguaje ensamblador: Set de instrucciones expresado con mnemónicos

# Instrucciones

---

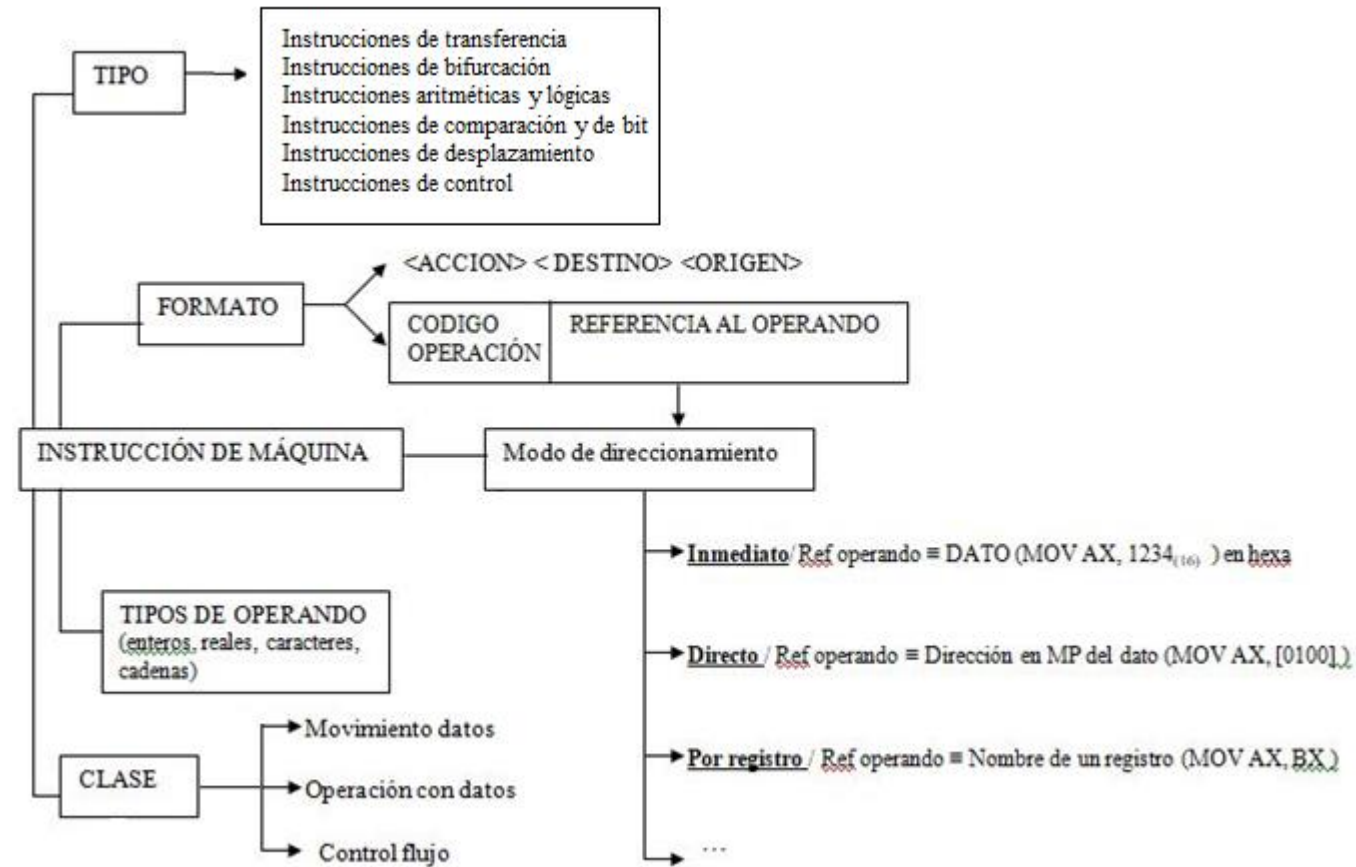
- Cuando la computadora ***realiza una tarea compleja***, a pedido del usuario, *ejecuta una serie de pasos simples* representados por su *propio juego de instrucciones*.
- Estas instrucciones ***constituyen su lenguaje de máquina o lenguaje nativo***.
- Como ya se indicó, ***no es usual que el programador plantee la tarea en términos de secuencias binarias***, sino que se ***utiliza un lenguaje simbólico*** más orientado a su modalidad de expresión que a la de la computadora.
- Sin embargo, ***todo programa que utiliza un lenguaje simbólico debe traducirse a código de máquina antes de su ejecución***.

# Instrucciones

---

- Por el momento no se entrará en detalle respecto de esta ***herramienta “que traduce” instrucciones simbólicas a instrucciones de máquina.***
- Considérese ***la notación simbólica como una forma alternativa para representar instrucciones binarias***, teniendo siempre presente que la computadora sólo ejecuta códigos de instrucción en lenguaje de máquina.
- Así como se establece un código de representación de caracteres, unidades elementales que constituyen los datos (que ingresan, por ejemplo, por teclado), ***también hay un código de representación de instrucciones, unidades elementales que constituyen los programas.***

# Instrucciones



# Instrucciones

---

## **Código de una instrucción:**

- Es la combinación de bits que la unidad de control de la CPU interpreta para generar microoperaciones que permitan su ejecución.

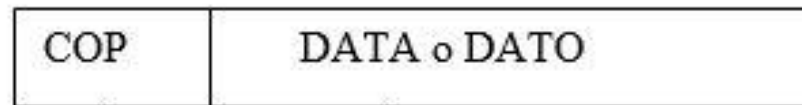
## **Formato de la instrucción:**

- El formato más simple es el que asigna un grupo de bits para representar una “acción” y otro grupo para representar el “dato” al que afecta esta acción.



# Instrucciones

## Formato básico de una instrucción:






El primer grupo de bits se denomina código de operación (OPCODE). La cantidad de bits del COP determina el número de acciones distintas que se podrían definir, según la fórmula siguiente:  
"n bits" determinan " $2^n$  códigos de operaciones distintos".

Datos sobre los que actúo:

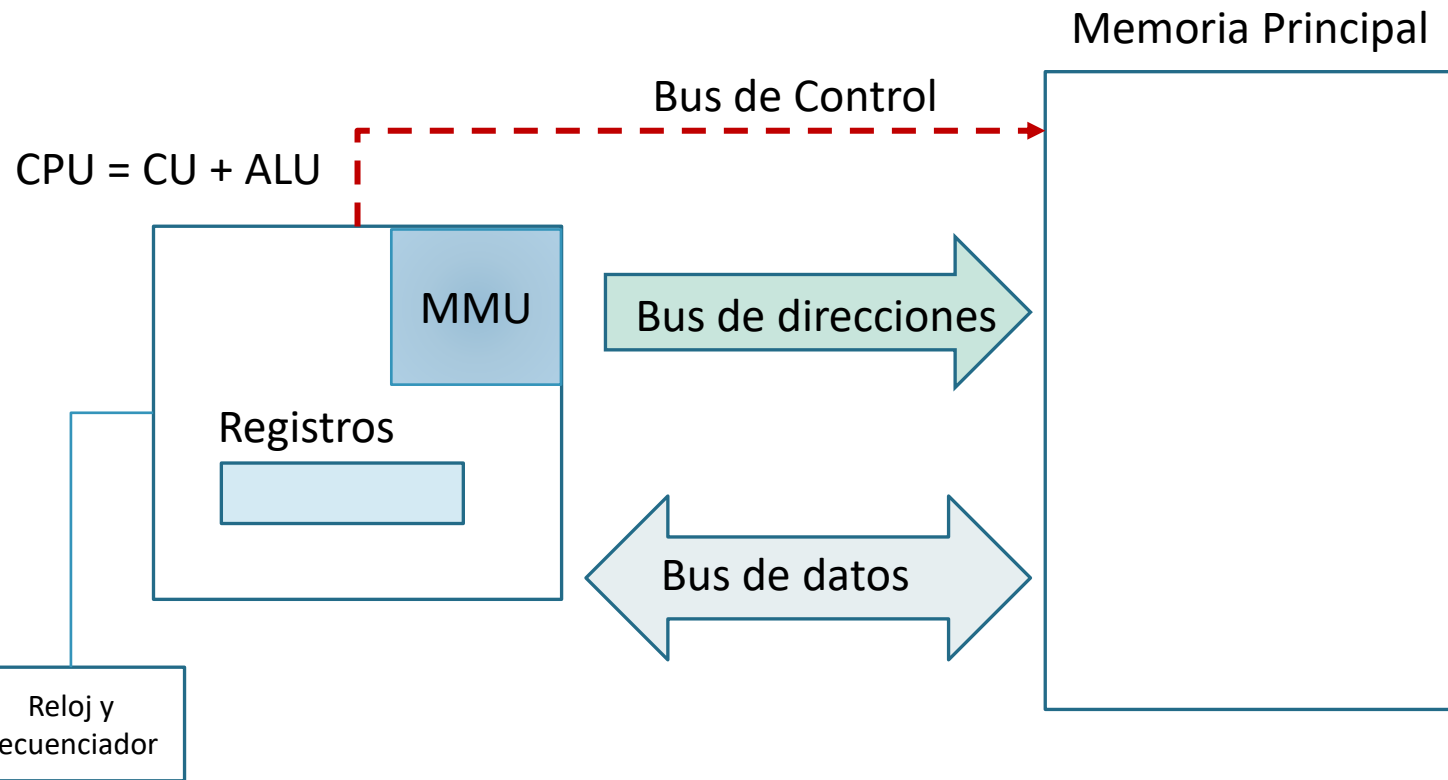
- ♦ En general es la posición del dato en memoria
- ♦ La cantidad de bits permite representar cualquier dato en memoria.
- ♦ A veces un dato esta en un registro de la CPU.
- ♦ Puede no haber dato

# CPU – Unidad Central de Procesamiento

---

- La función de la CPU se puede separar en partes:
  1. Tratamiento de instrucciones  Se encarga la **unidad de control (CU o *Control Unit*)**, sincronizado por el generador de pulsos de reloj
  2. Operación de los datos  Se encarga la **unidad aritmético-lógica (ALU o *Aritmetic Logic Unit*)**
  3. Calculo de dirección de Memoria  Se encarga la **unidad de Manejo de Memoria (MMU o *Memory Manager Unit*)**

# CPU – Memoria Principal



- Los datos e instrucciones se transportan a través **bus de datos**, este camino es bidireccional y permite la transferencia de grupos de bits desde o hacia la Memoria Principal.
- La MMU se comunica con la Memoria principal a través del bus de direcciones o address bus
- A través del bus de control se envían ordenes de Lectura (RD) o Escritura (WR) hacia la Memoria Principal según si es necesario escribir en una posición de memoria o leer desde la misma.

# Registros

## Registros de calculo de 16 y 32 bits

**Tabla 8-1. Registros de cálculos de 16 bits.**

15	8 7	0
AH	AL	Ax Registro acumulador (operación E/S y de cadena)
BH	BL	Bx Registro base (registro base para direccionamiento)
CH	CL	Cx Registro contador (para bucles, iteraciones, desplazamientos y rotaciones)
DH	DL	Dx Registro para datos (almacenado de datos, direcciones de puertos, extensión de Ax en multiplicación y división)

MOV AX, 1234	AX	15	8 7	0
MOV AH, 12	AH	12	34	
MOV AL, 34	AL	12		34

- Los cuatro registros de calculo en una IA-32:
  - ✓ EAX (Acumulador)
  - ✓ EBX (Base)
  - ✓ ECX (Contador)
  - ✓ EDX (Datos)
- Todos los registros se pueden utilizar en operaciones de cálculos aritméticos y lógicos.

# Registros

## Registros punteros

**Tabla 8-2. Registros punteros.**

	31	16	15	0	
EIP					IP
ESP					SP
EBP					BP
ESI					SI
EDI					DI

**IP** Registro puntero de instrucción.

**SP** Registro puntero de pila. Modo de direccionamiento a la pila.

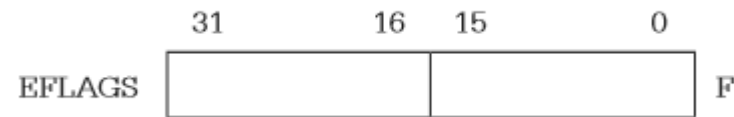
**BP** Registro base para pila. Modo de direccionamiento base a la pila.

**SI** Registro índice fuente. Modo de direccionamiento indexado.

**DI** Registro índice destino. Modo de direccionamiento indexado.

# Registros

## Registro de estado



**Registro de estado:** en este registro se alojan, por nombrar algunas, todas las banderas aritméticas, banderas de modo de trabajo del microprocesador, banderas asociadas a interrupciones, etc.

## Registro de segmento

**Tabla 8-3. Registros de segmento**

15	0
CS	
SS	
DS	
ES	
FS	
GS	

Registro de base de segmento de código

Registro de base de segmento de pila o *stack*

Registros de base de segmentos de datos



Los registros de segmento almacenan la referencia binaria a la base de un segmento en memoria, esto es, donde empieza la zona de memoria para ese objeto, tendremos seis de ellos.

# Modo real vs Modo Protegido

---

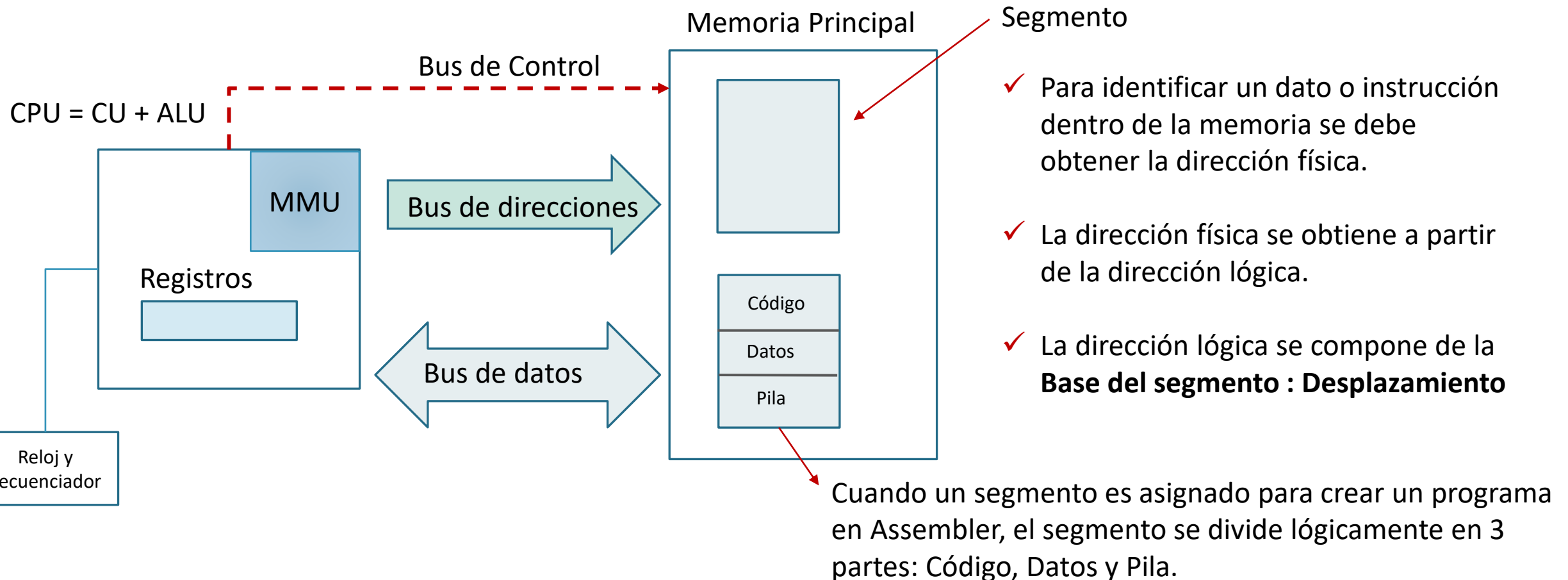
## **Modo real:**

- ✓ 20 bits de espacio de direcciones segmentado ( $2^{20} = 1$  MB de memoria maximo)
- ✓ Acceso directo del software a las rutinas del BIOS y el hardware periférico
- ✓ No tiene conceptos de protección de memoria o multitarea a nivel de hardware.
- ✓ Palabra de 16 bits.
- ✓ La memoria es segmentada. (Tamaño máximo de segmento  $2^{16} = 64$  KB).
- ✓ Rango de direcciones de un segmento de 0000 a FFFF en Hexadecimal.
- ✓ Cada dirección identifica 8 bits de almacenamiento.

## **Modo Protegido: (Se vera mas adelante con mas detalle)**

- ✓ En este modo existe la Memoria Virtual.
- ✓ Trabaja con Memoria Principal de hasta 4GB y Memoria Virtual de hasta 64TB (a partir del 80386).
- ✓ La memoria es segmentada, con o sin paginación.
- ✓ Protección de programas.
- ✓ Capacidad de Multitareas.
- ✓ Posible Conexión a Memoria Cache.

# CPU – Memoria Principal – Modo real





# Modo real

---

Dirección Lógica:

**Base del segmento : Desplazamiento**

CS : IP	ES : DATA
DS : DATA	SS : SP

**Ejemplo en Assembler**

13E0:0100 mov ax,0002

13E0:0103 mov bx,0004

13E0:0106 add ax,bx

13E0:0108 int 20

13E0:010A

Calculo de la dirección física:

**Dirección Física = Base del segmento \* 10<sub>H</sub> + Desplazamiento**

Ejemplo:

Dirección lógica: 13E0 : 0100

Dirección física: 13E0 \* 10 + 0100 = 13F00

# Instrucciones - Assembler

---

## **Instrucción MOV**

Transferencia de datos entre celdas de memoria, registros y acumulador.

Sintaxis:

**MOV destino, fuente**

Donde “destino” es el lugar a donde se moverán los datos y “fuente” es el lugar donde se encuentran dichos datos.

**MOV AX,0006h**

**MOV BX,AX**

**MOV AX,4C00h**

**MOV AH,[0300]**

# Instrucciones - Assembler

---

## **Instrucción ADD**

Adición de los operandos.

Sintaxis:

**ADD destino, fuente**

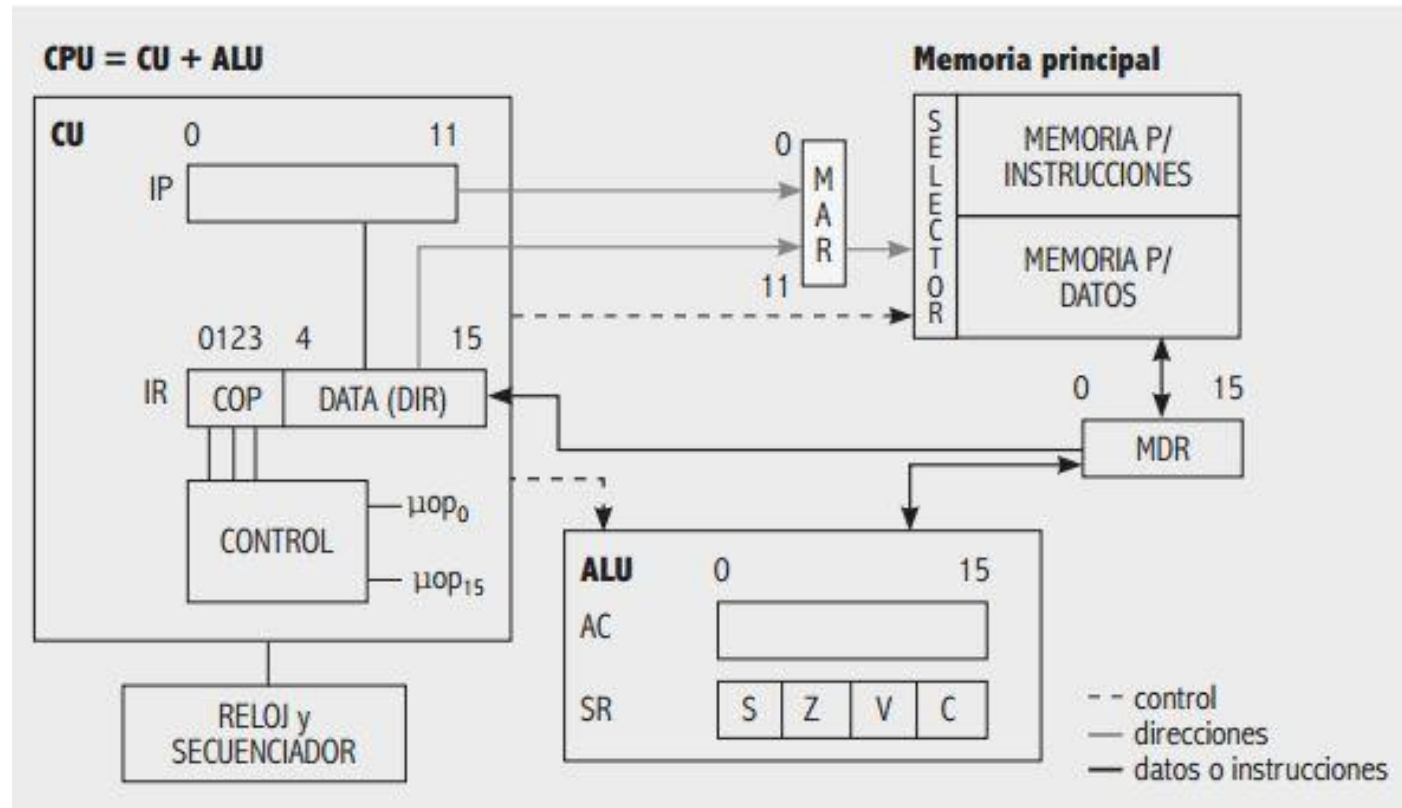
Suma los dos operandos y guarda el resultado en el operando “destino”.

**ADD AX,0006h**

**ADD BX,AX**

**ADD AH,[0301]**

# Modelo de estudio



MAR: *Memory Address Register*

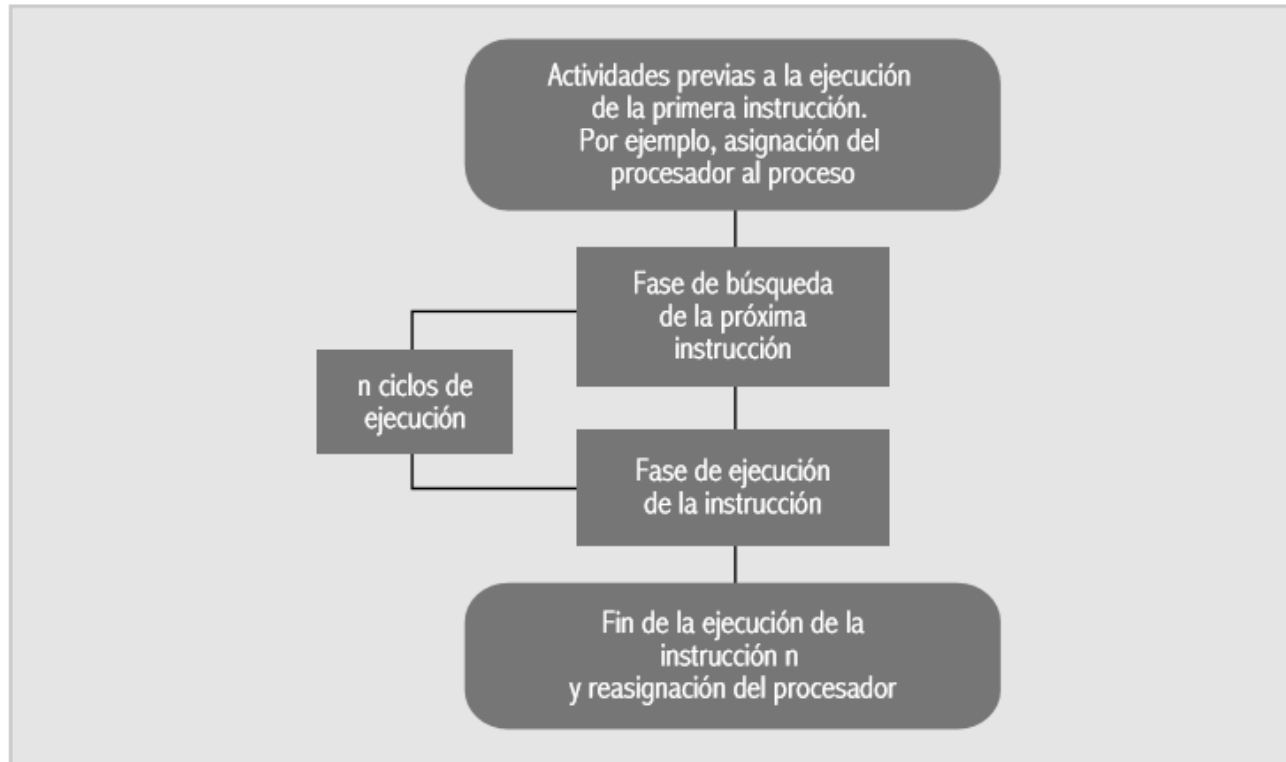
IP: *Instruction Pointer*

MDR: *Memory Data Register*

IR: *Instruction Register*

Arquitectura básica de "X"

# Ciclo de instrucción



**Fig. 8.2.** Ciclo de instrucción para las  $n$  instrucciones de un programa.

# Fases de búsqueda y ejecución

---

- Este proceso se puede dividir en las etapas siguientes:
  1. Búsqueda de la instrucción en memoria. → **fase de búsqueda o fase fetch**
  2. Interpretación del código de instrucción.
  3. Búsqueda del dato afectado (si afecta a dato) por la instrucción.
  4. Generación de órdenes al módulo que opera sobre ese dato.

} **fase de ejecución o execute**