Arquitectura de Computadores

Ricardo.Sanz@upm.es

Curso 2004-2005

Arquitectura de Computadores

- Arquitectura de computadores es la disciplina que estudia la organización y funcionamiento de los computadores
- A veces se emplean los términos “arquitectura” y “organización” en sentidos distintos:
  - Arquitectura: Qué es capaz de hacer
  - Organización: Cómo lo hace

Hacer una suma

Ejemplo:

¿Cómo sumar el contenido de la posición 867 con el contenido de la posición 562 y almacenar el resultado en la posición 778?

Z ← X + Y

Dirección	Dato
1023	
:	
867	X
:	
778	Z
:	
562	Y
:	
2	
1	
0	

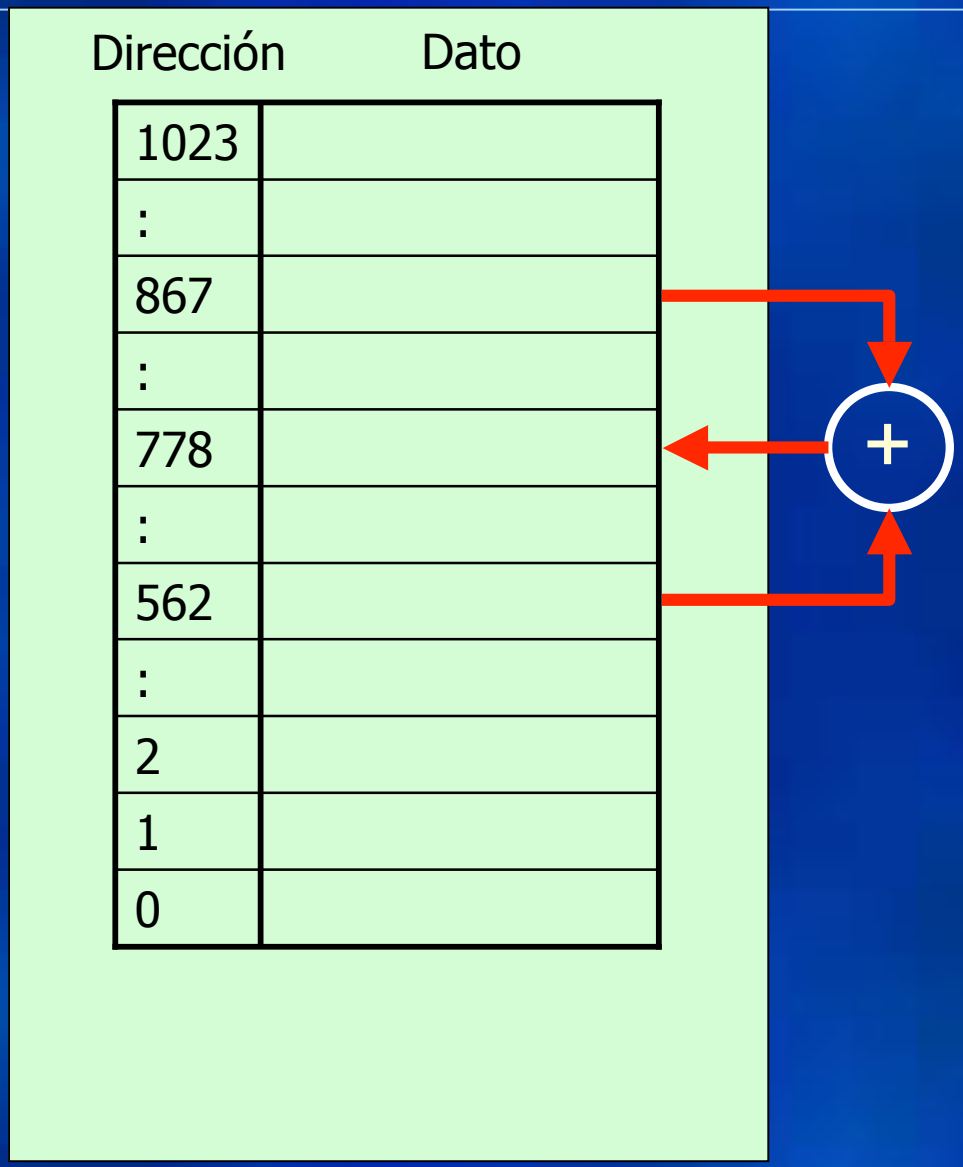
Memoria de 1024 × 8

Hacer una suma

Ejemplo:

¿Cómo sumar el contenido de la posición 867 con el contenido de la posición 562 y almacenar el resultado en la posición 778?

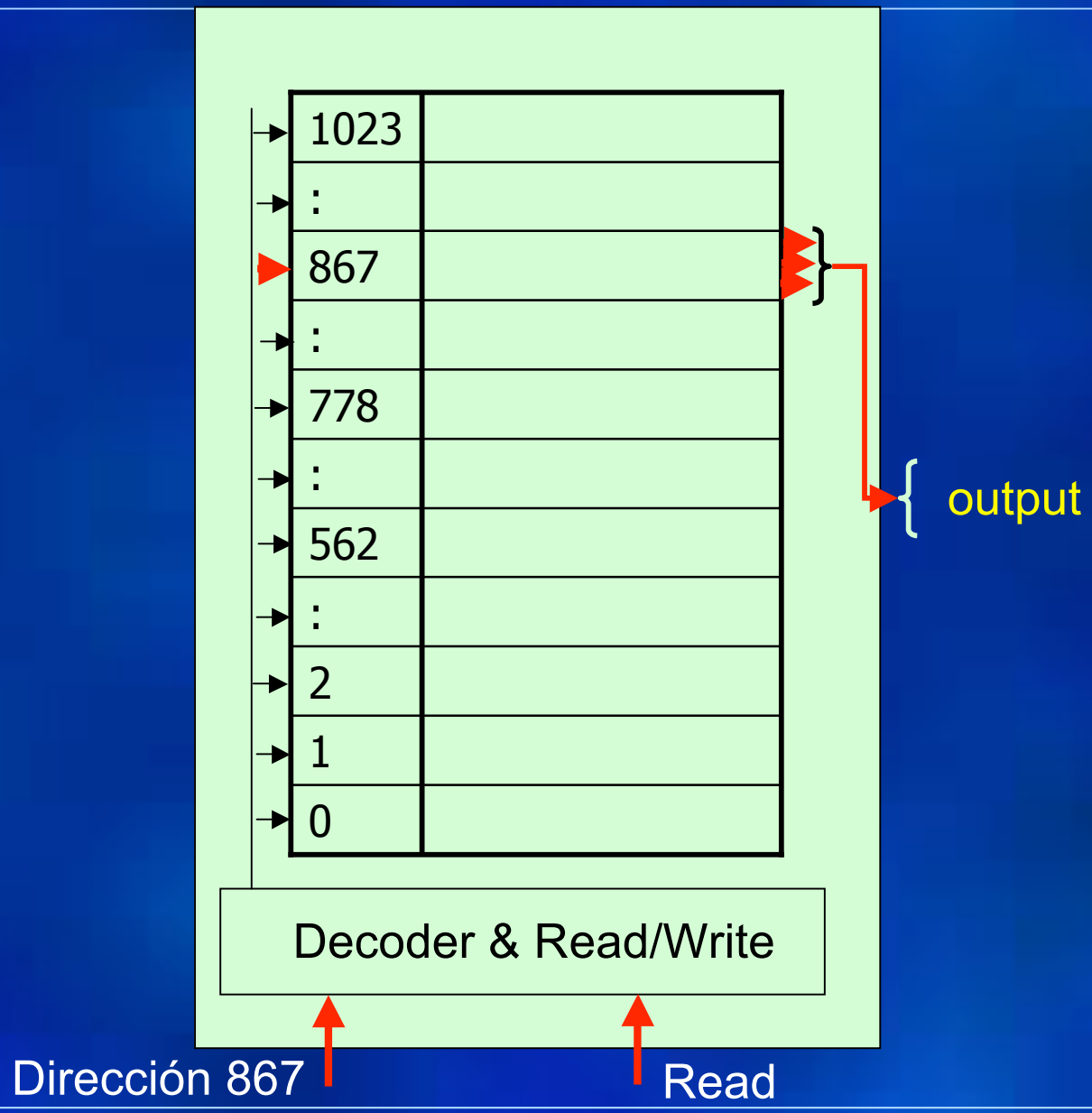
Memoria de 1024 × 8



Hacer una suma

Solución:

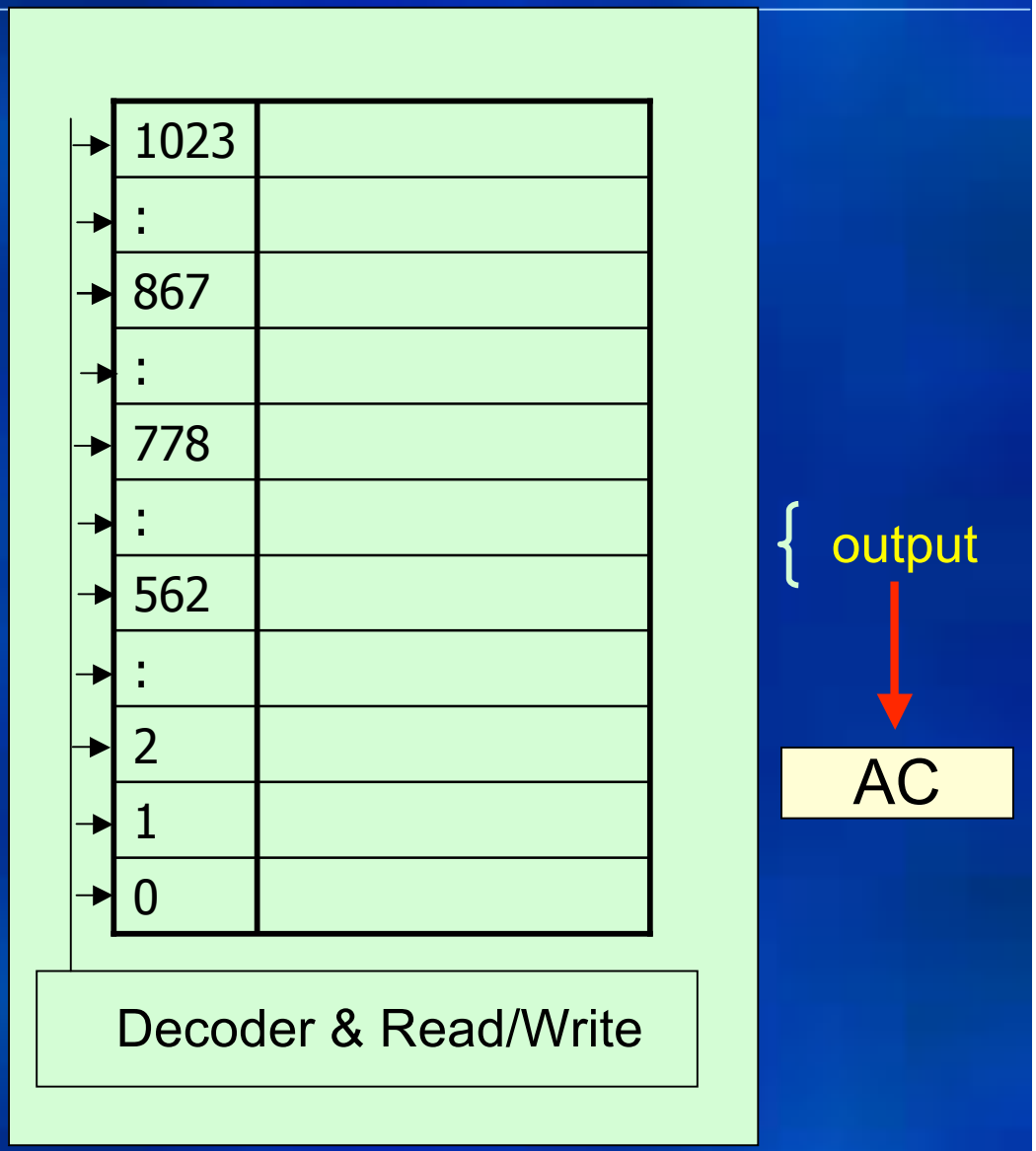
1. Leer en la memoria la posición 867



Hacer una suma

Solución:

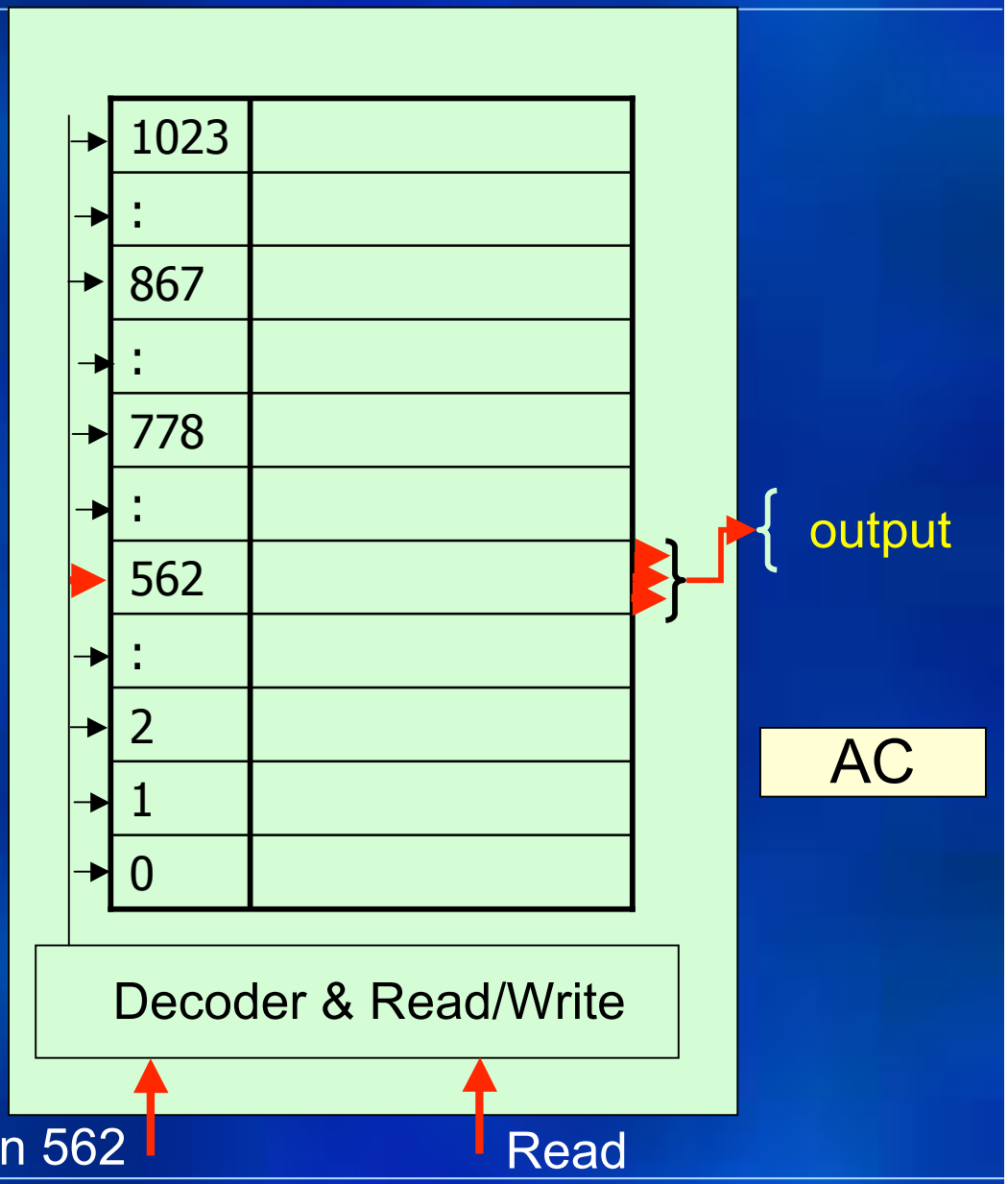
1. Leer en la memoria la posición 867
2. Almacenar lo leído en un registro externo



Hacer una suma

Solución:

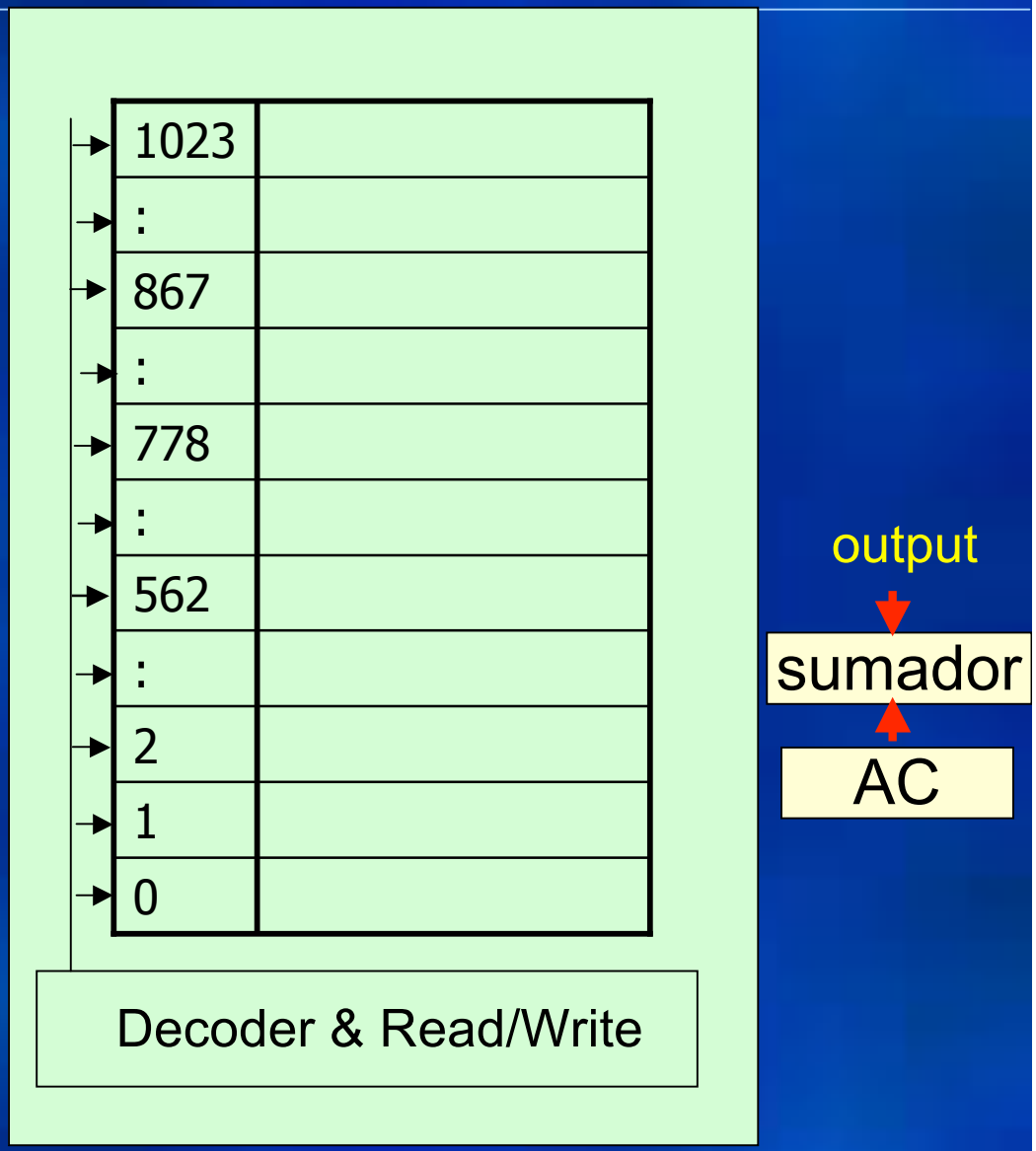
1. Leer en la memoria la posición 867
2. Almacenar lo leído en un registro externo
3. Leer en la memoria la posición 562



Hacer una suma

Solución:

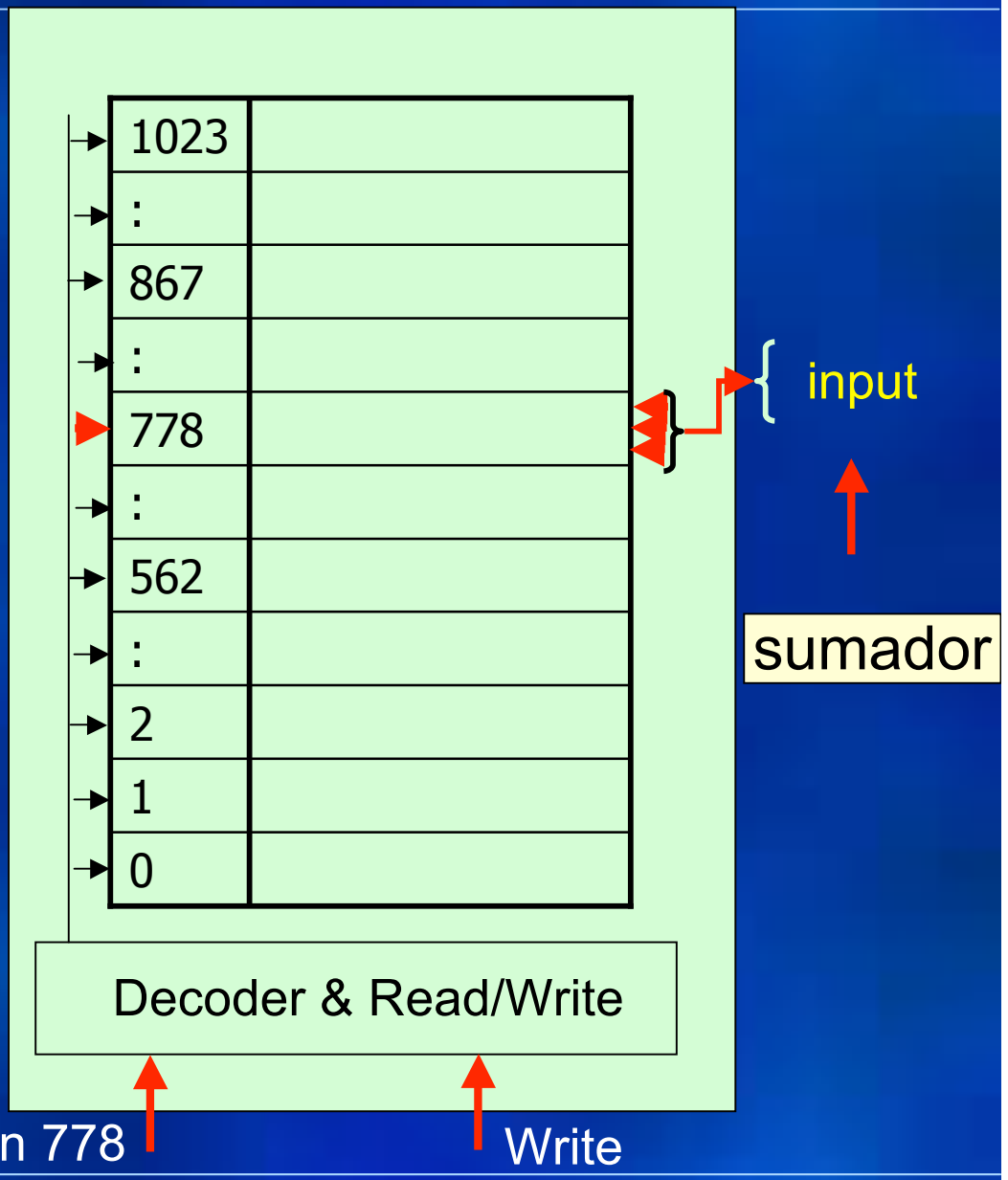
1. Leer en la memoria la posición 867
2. Almacenar lo leído en un registro externo
3. Leer en la memoria la posición 562
4. Sumar lo leído con el registro externo



Hacer una suma

Solución:

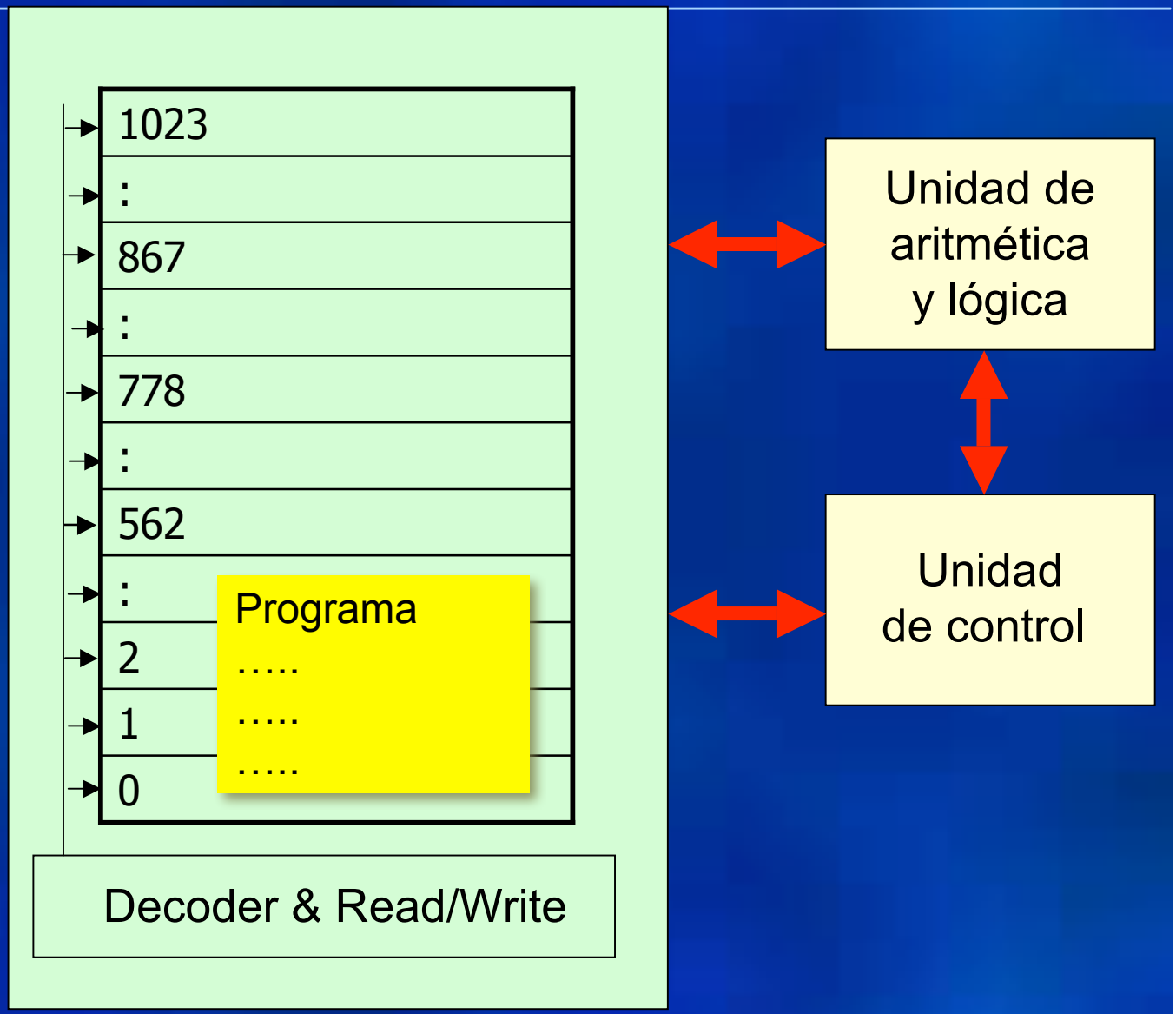
1. Leer en la memoria la posición 867
2. Almacenar lo leído en un registro externo
3. Leer en la memoria la posición 562
4. Sumar lo leído con el registro externo
5. Almacenar la suma en la posición 778



¿Cómo hacer mas que una suma ?

Solución general:

Usar un programa residente en memoria



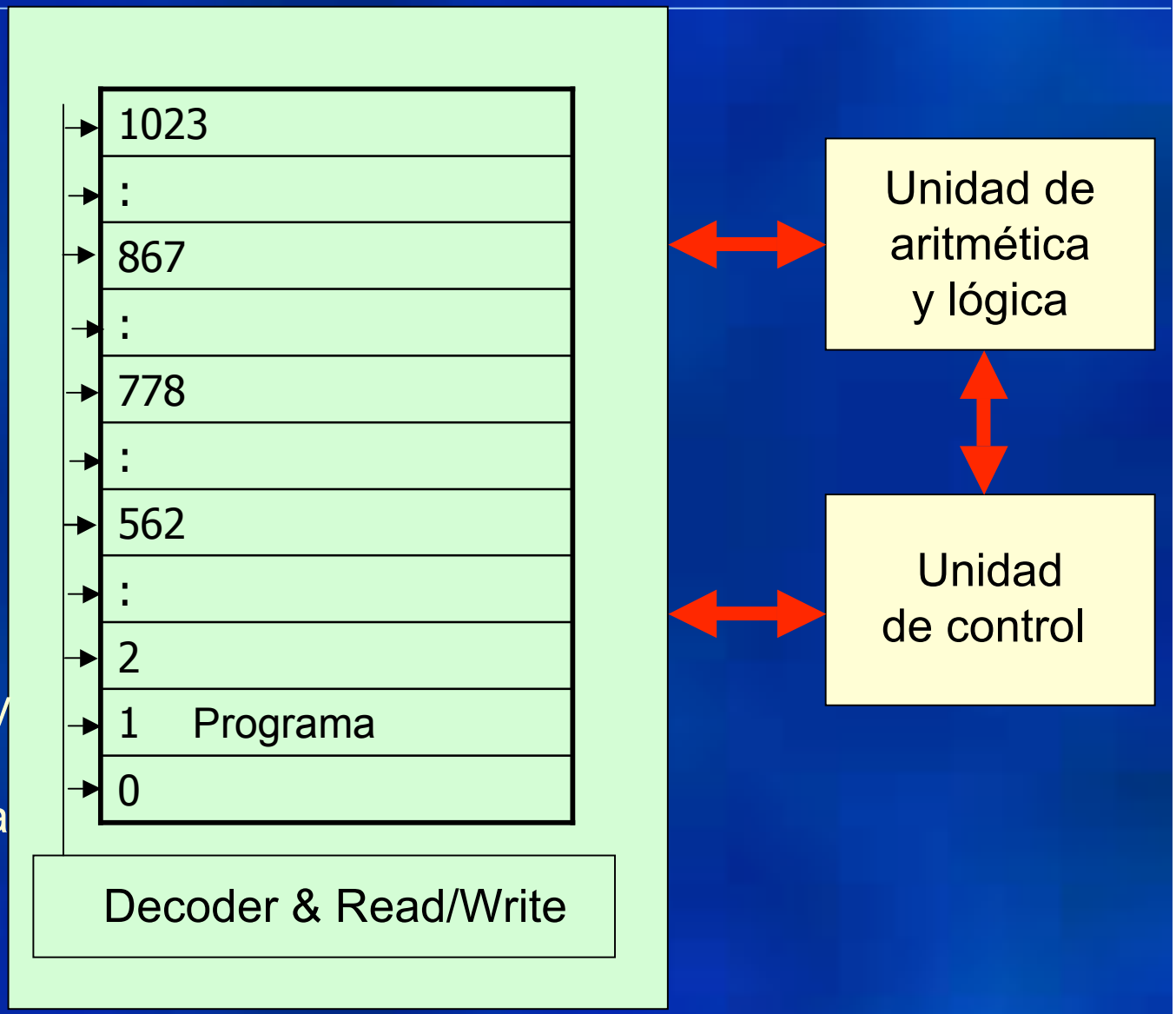
Hacer mas que una suma

Solución general:

El programa le da instrucciones a la Unidad de Control

Ejemplo:

00100001 110110011 significa leer en la memoria 110110011 y almacenar la lectura en el registro AC de la ALU.



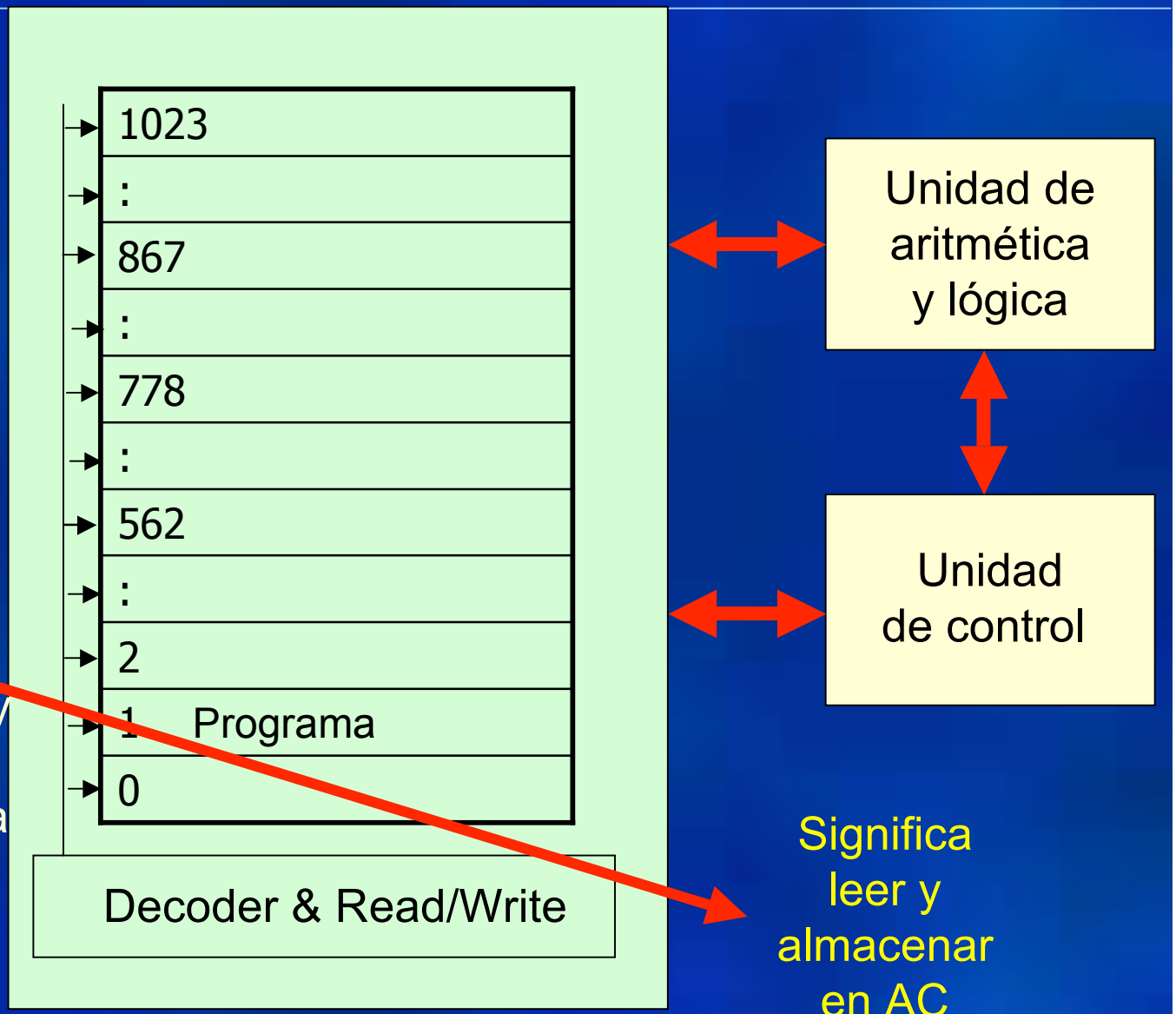
Hacer mas que una suma

Solución general:

El programa le da instrucciones a la Unidad Central.

Ejemplo:

00100001 110110011 significa leer en la memoria 110110011 y almacenar la lectura en el registro AC de la ALU.



Hacer mas que una suma

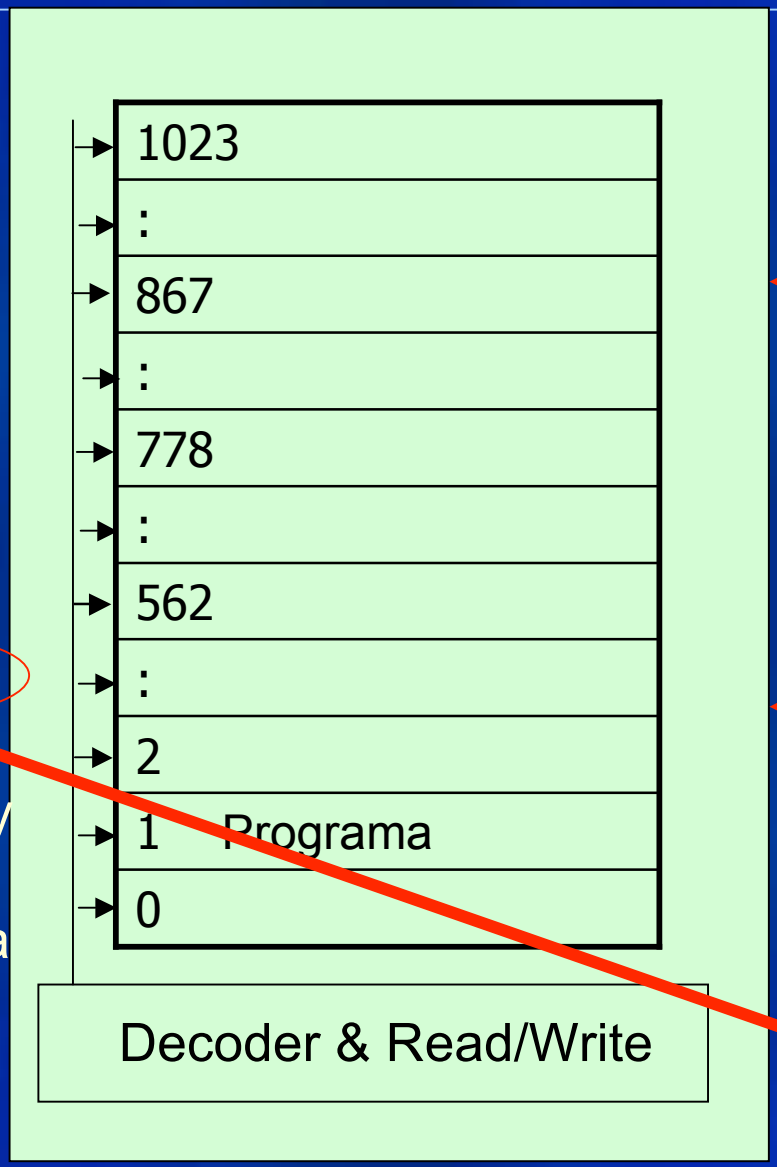
Solución general:

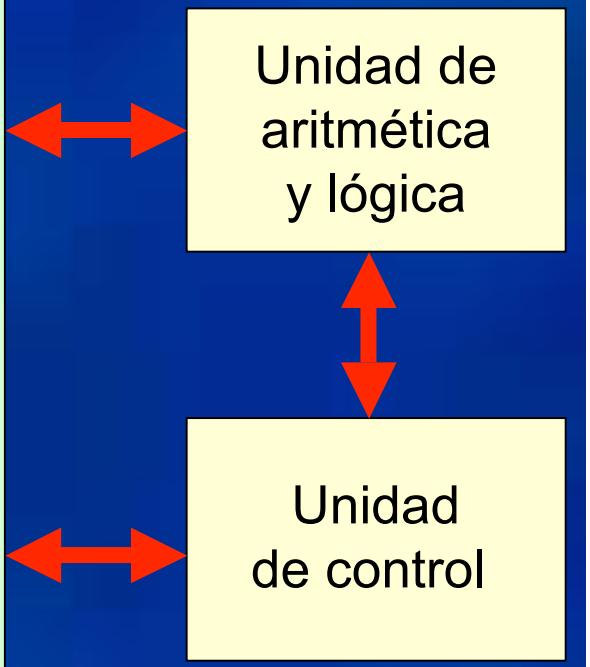
El programa le da instrucciones a la Unidad Central.

Ejemplo:

00100001 110110011 significa leer en la memoria 110110011 y almacenar la lectura en el registro AC de la ALU.

Memoria de 1024 × 8





Significa lo que se debe leer: 110110011 = 867

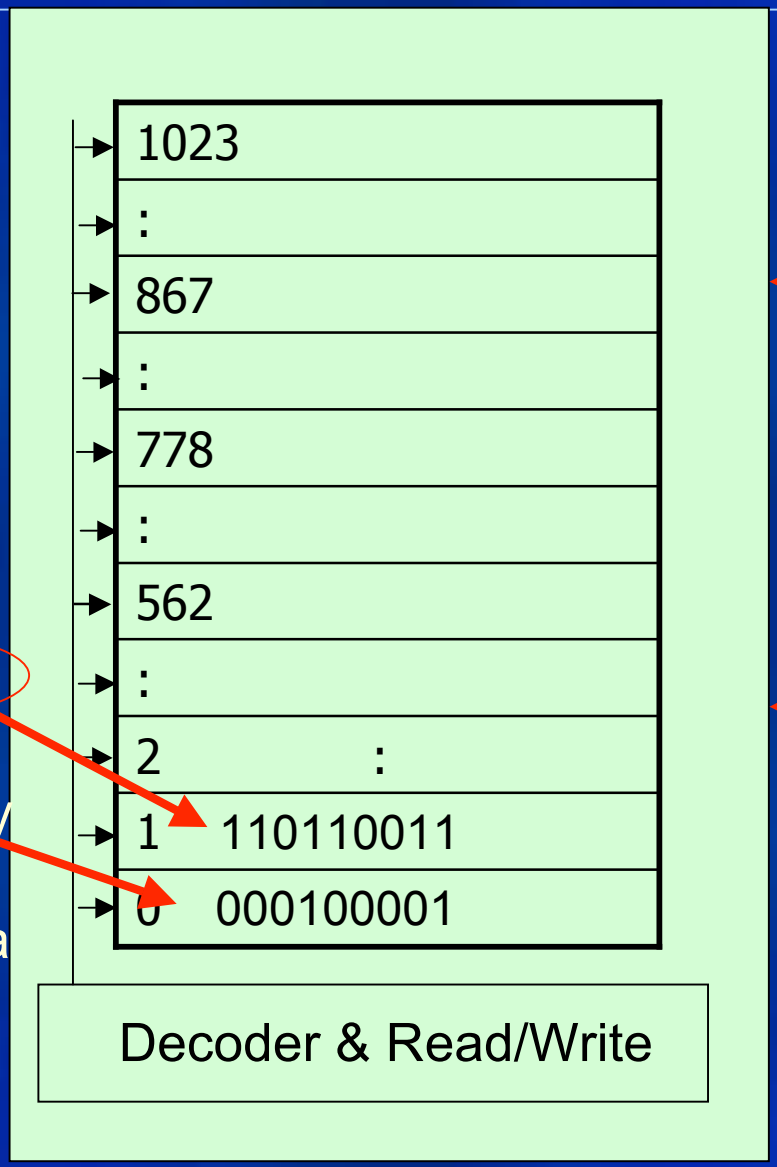
Hacer mas que una suma

Solución general:

El programa le da instrucciones a la Unidad Central.

Ejemplo:

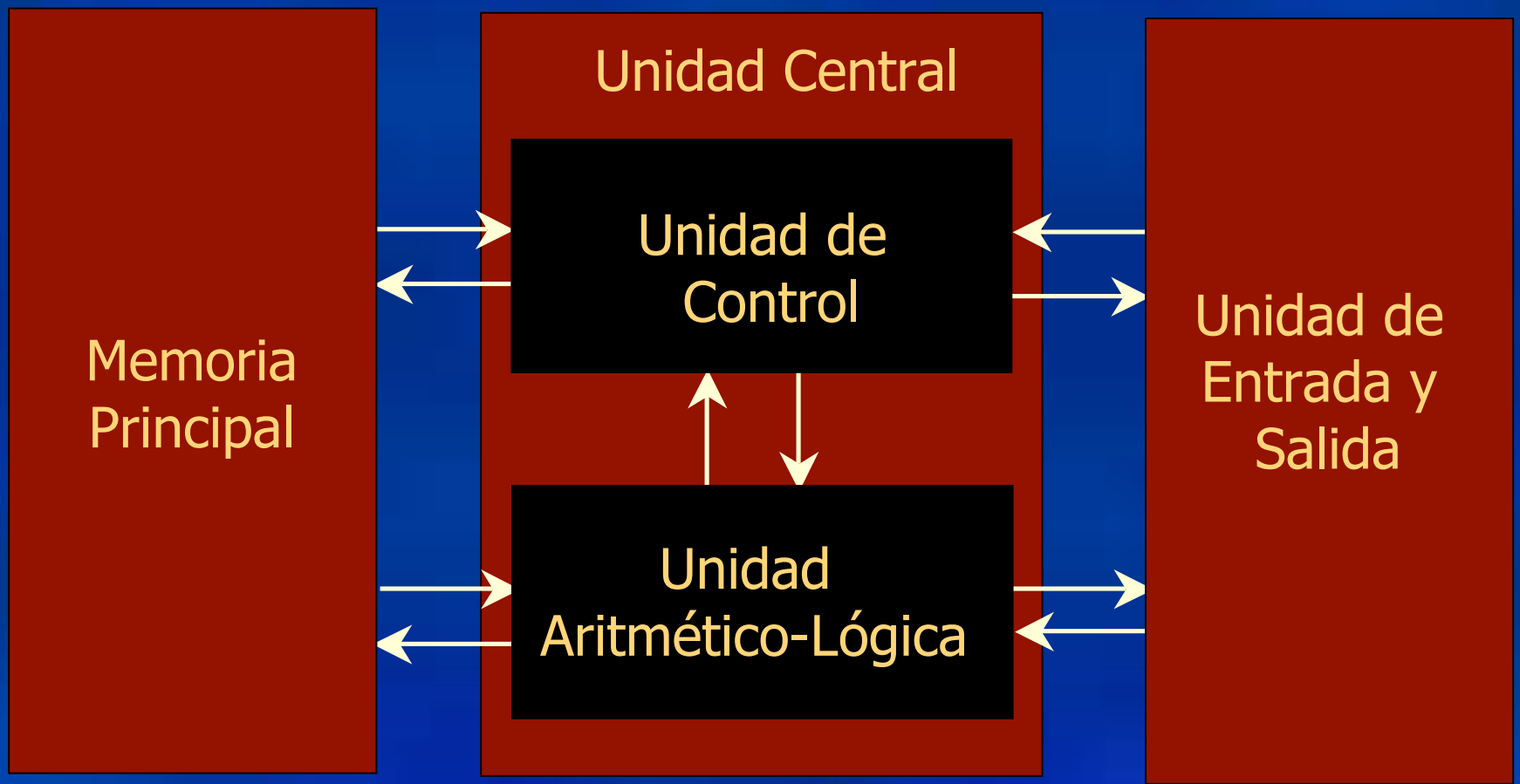
00100001 110110011 significa leer en la memoria 110110011 y almacenar la lectura en el registro AC de la ALU.



Principios de la arquitectura Von Neumann:

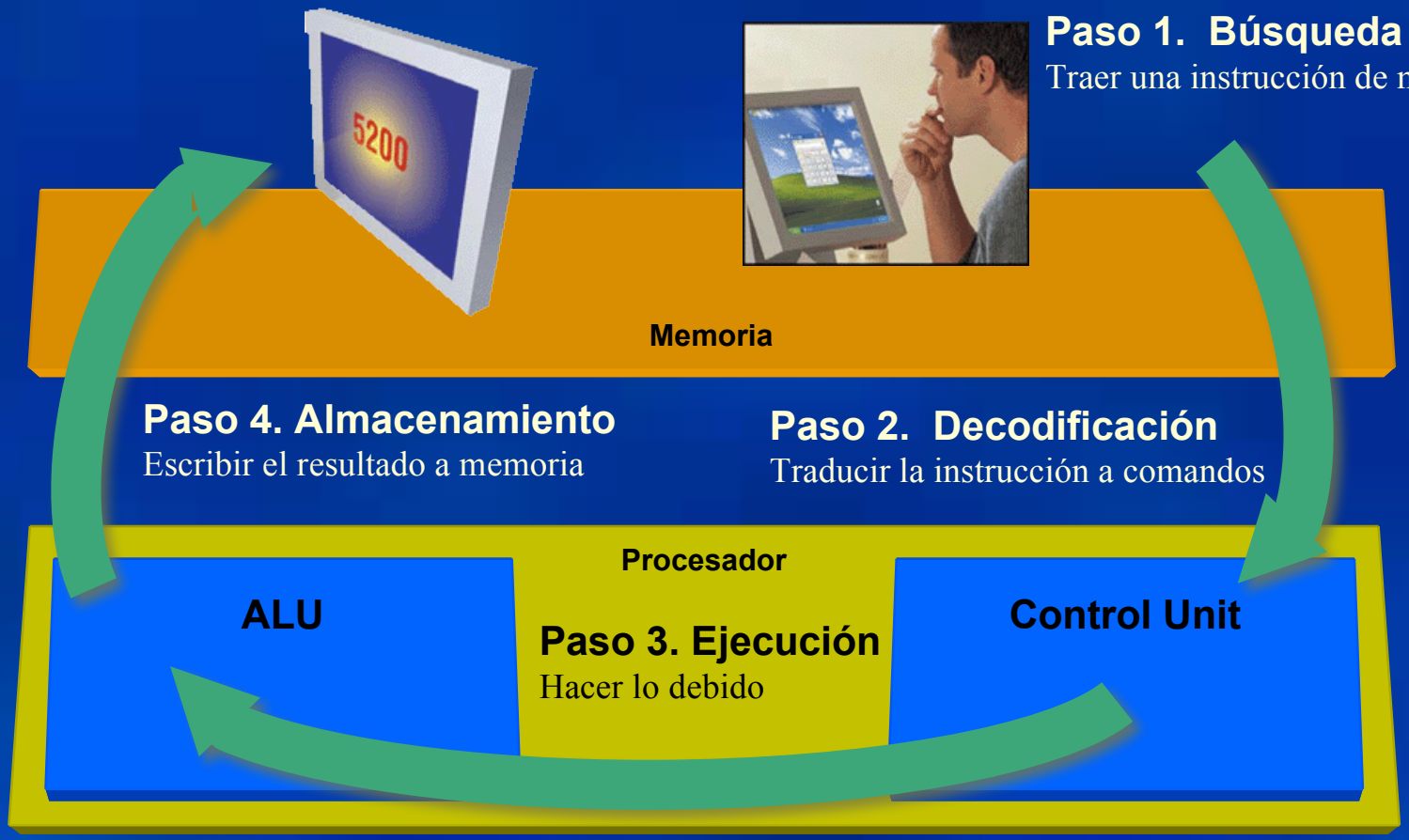
- Los datos y las instrucciones se almacenan en una sola memoria de lectura-escritura.
- Los contenidos de esta memoria se direccionan indicando su posición, sin considerar el tipo del de dato contenido en la misma.
- La ejecución se produce siguiendo una secuencia de instrucción tras instrucción (a no ser que dicha secuencia se modifique explícitamente).

Estructura Von Neumann



Ciclos de máquina

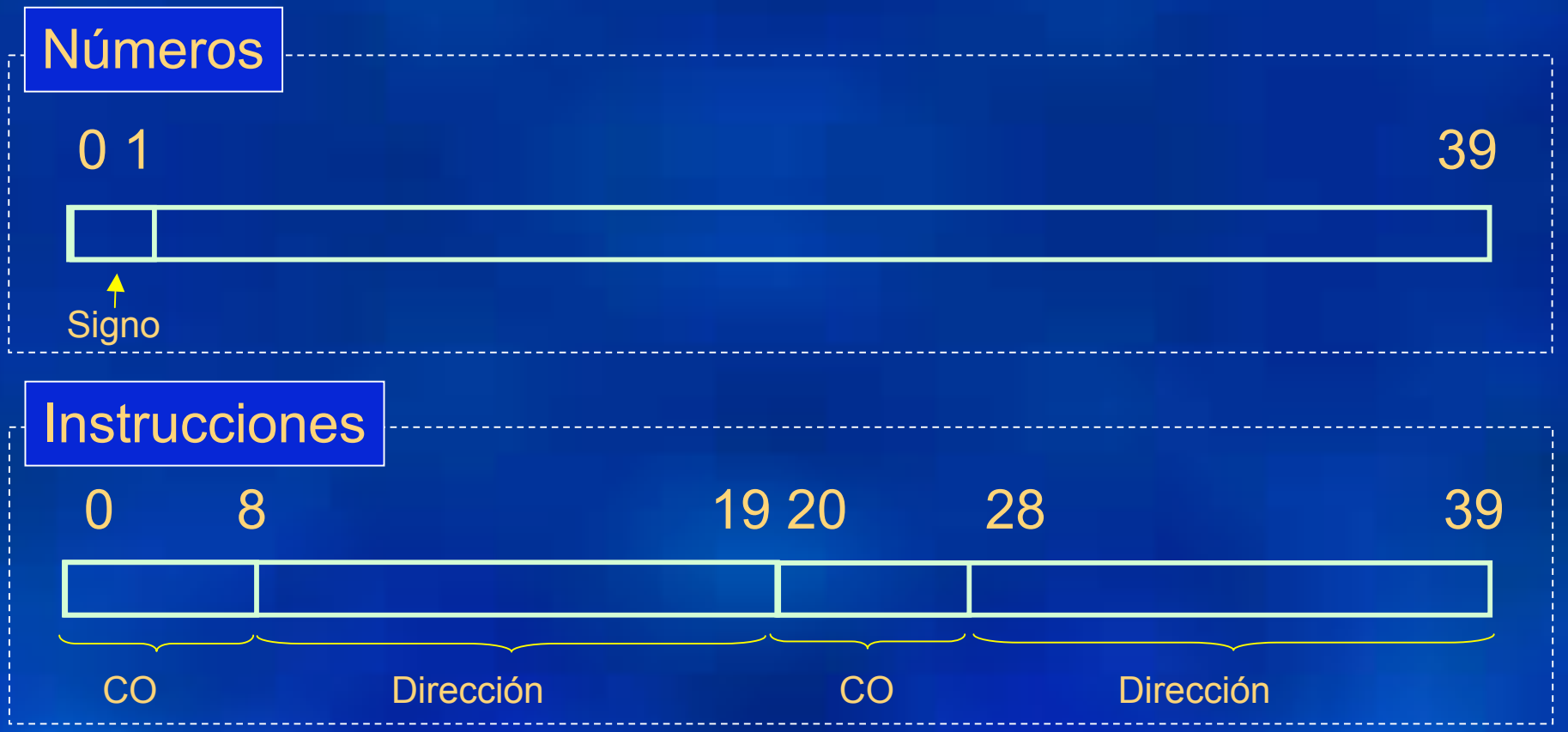
Cuatro pasos por ciclo de máquina



Datos en memoria

Ejemplo

- Memoria de 1000 x 40 bit words
  - Binary number
  - 2 x 20 bit instructions



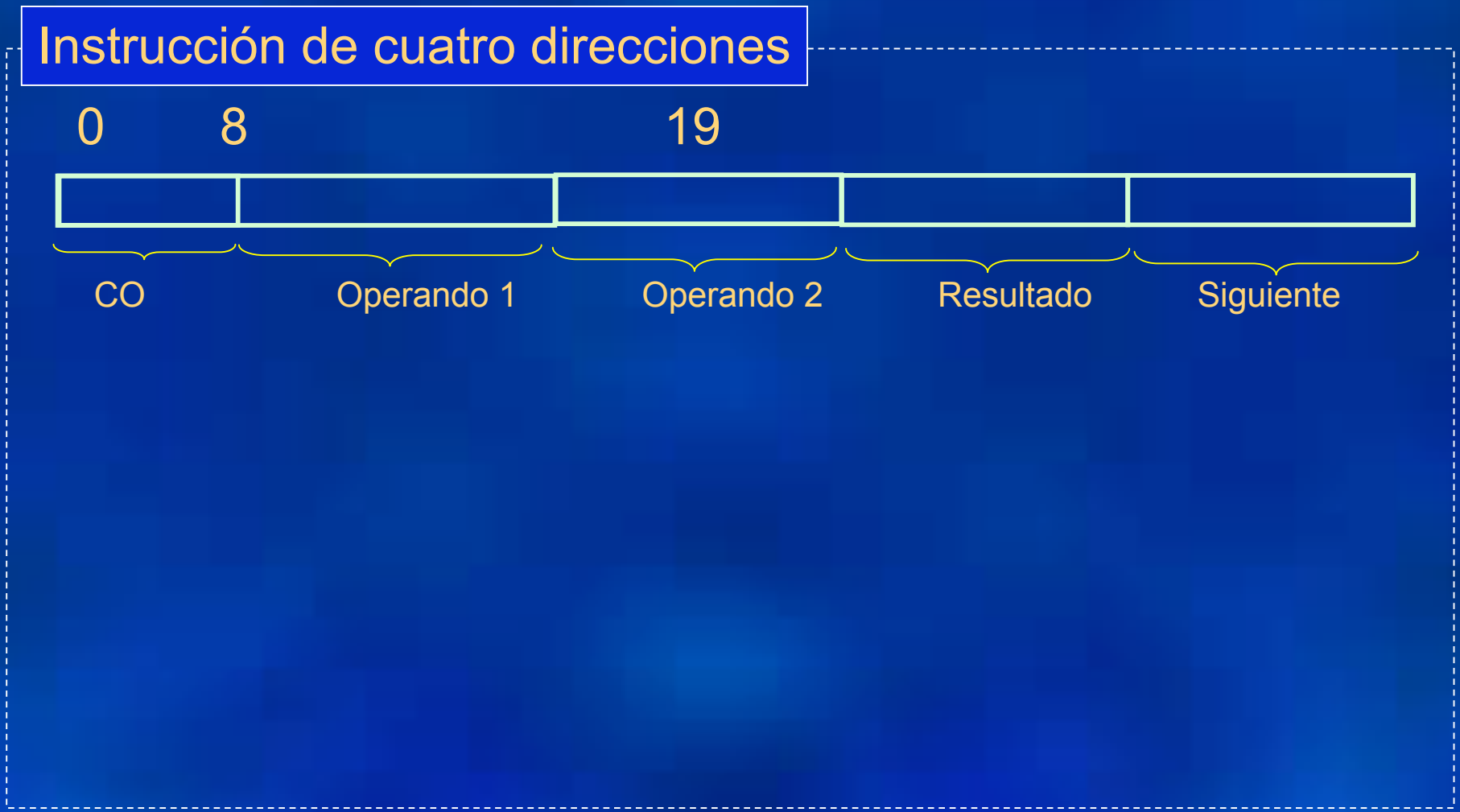
Formato de instrucción

Instrucciones

• La parte del código de operación (CO, OpCode, los primeros 8 bits) especifican qué instrucción será ejecutada.

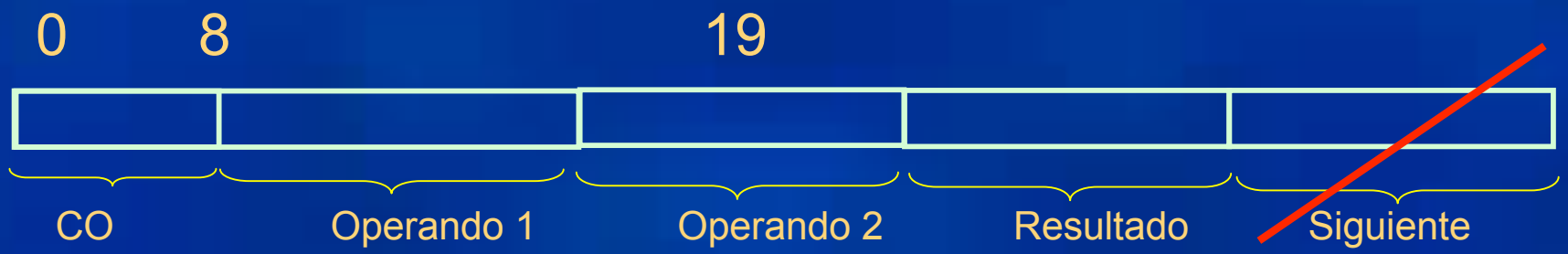
• La parte de la dirección (los 12 bits restantes) especifican cuál de las 1000 posiciones de memoria está implicada en la instrucción.

Instrucción de cuatro direcciones



Instrucción de tres direcciones

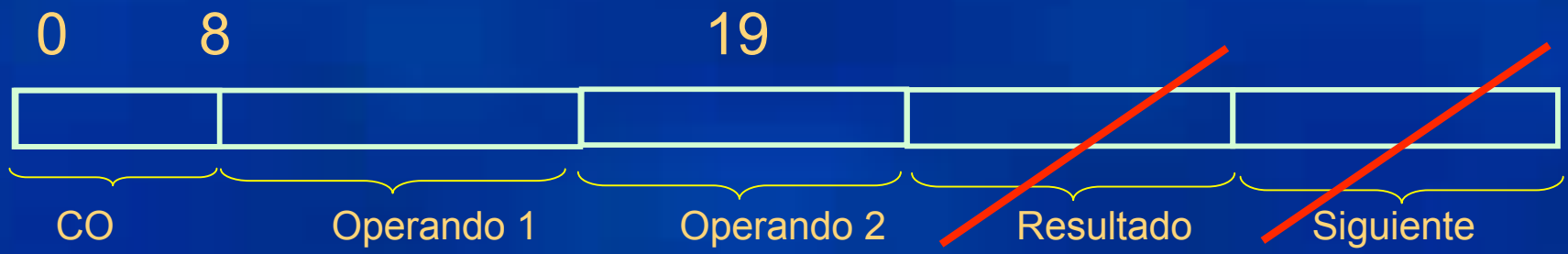
Instrucción de tres direcciones



- Se usa un contador de programa (PC) para saber cual es la siguiente instrucción

Instrucción de dos direcciones

Instrucción de dos direcciones



- Se usa un contador de programa (PC) para saber cual es la siguiente instrucción
- Se usa un acumulador (AC) para almacenar el resultado

Instrucción de una direcciones

Instrucción de una direcciones

- Se usa un contador de programa (PC) para saber cual es la siguiente instrucción
- Se usa un acumulador (AC) para almacenar el resultado
- Se usa un registro como operando

Instrucción de cero direcciones

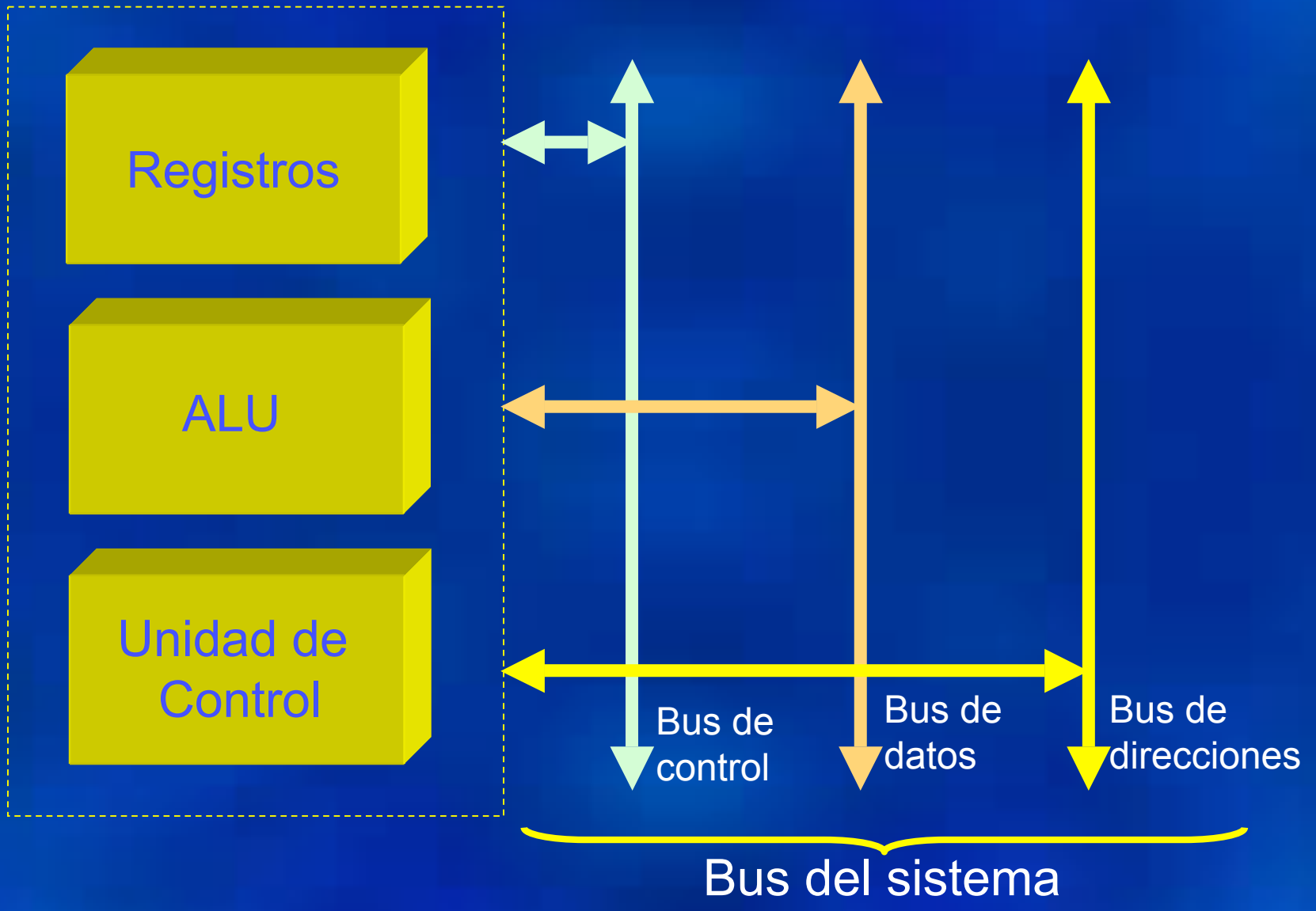
Instrucción de cero direcciones

- Se usa un contador de programa (PC) para saber cual es la siguiente instrucción
- Se opera sobre datos en la pila (stack)

Registros típicos de la CPU

- Memory Buffer Register (MBR)
- Memory Address Register (MAR)
- Instruction Register (IR)
- Instruction Buffer Register (IBR)
- Program Counter (PC)
- Accumulator (AC)
- Multiplier Quotient (MQ)

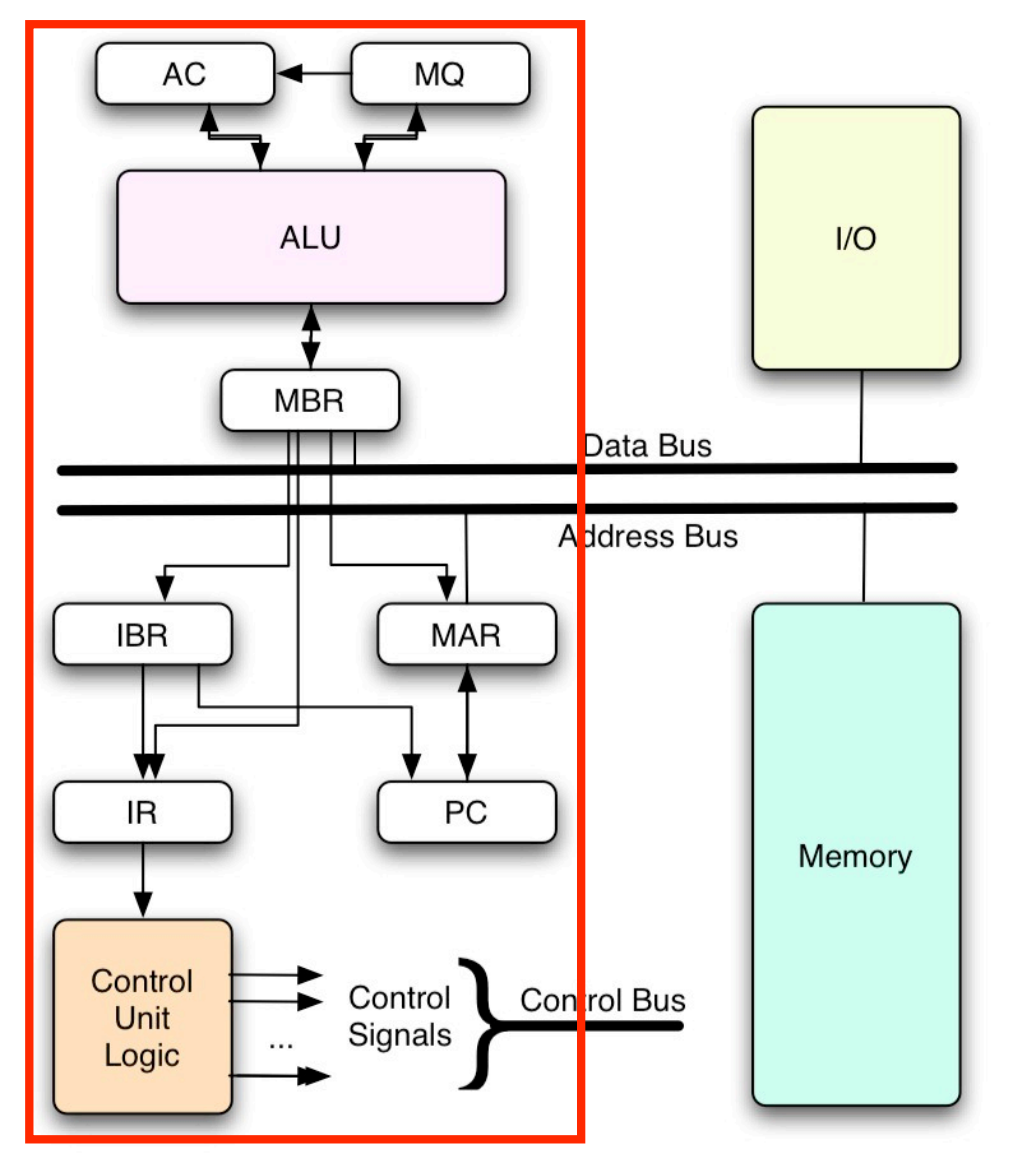
Organización básica de una CPU



Detalles

Partes de la CPU

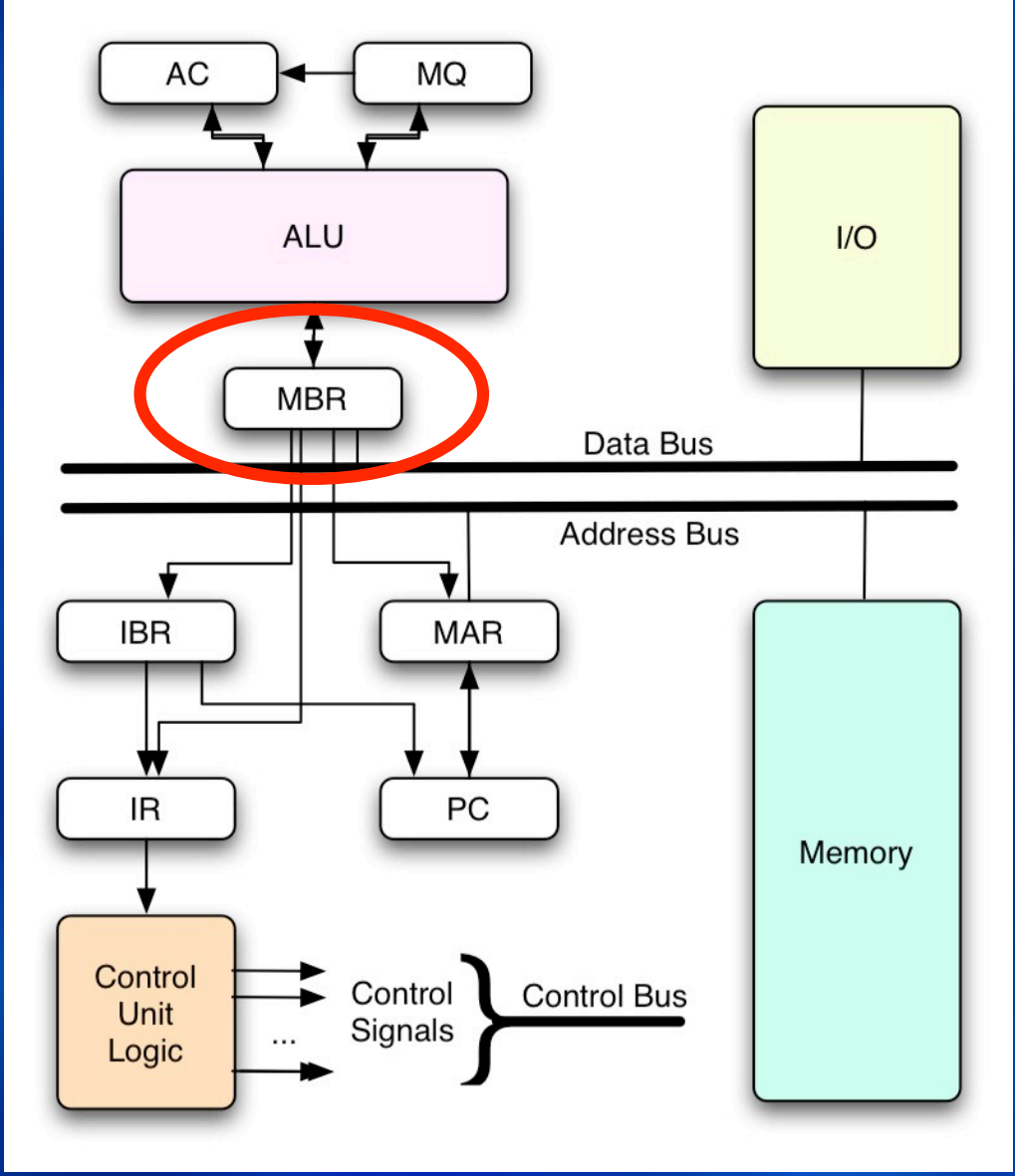
Unidad Aritmético-Lógica
Unidad de control
Registros
Caminos de datos



Detalles de la estructura

MBR:
Memory Buffer Register

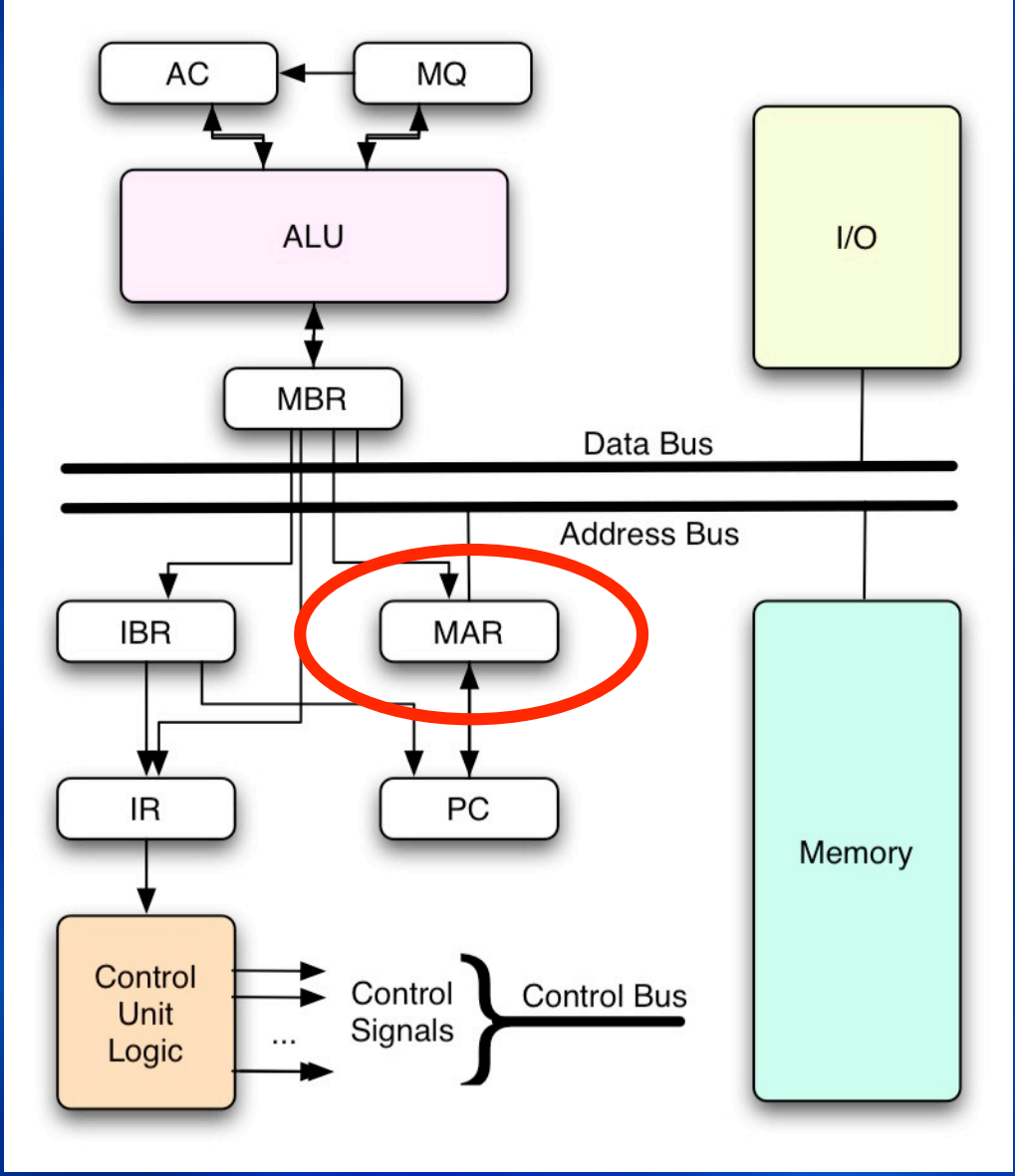
Contiene una palabra que debe ser almacenada en la memoria, o es usado para recibir una palabra procedente de la memoria.



Detalles de la estructura

MAR:
Memory Adress Register

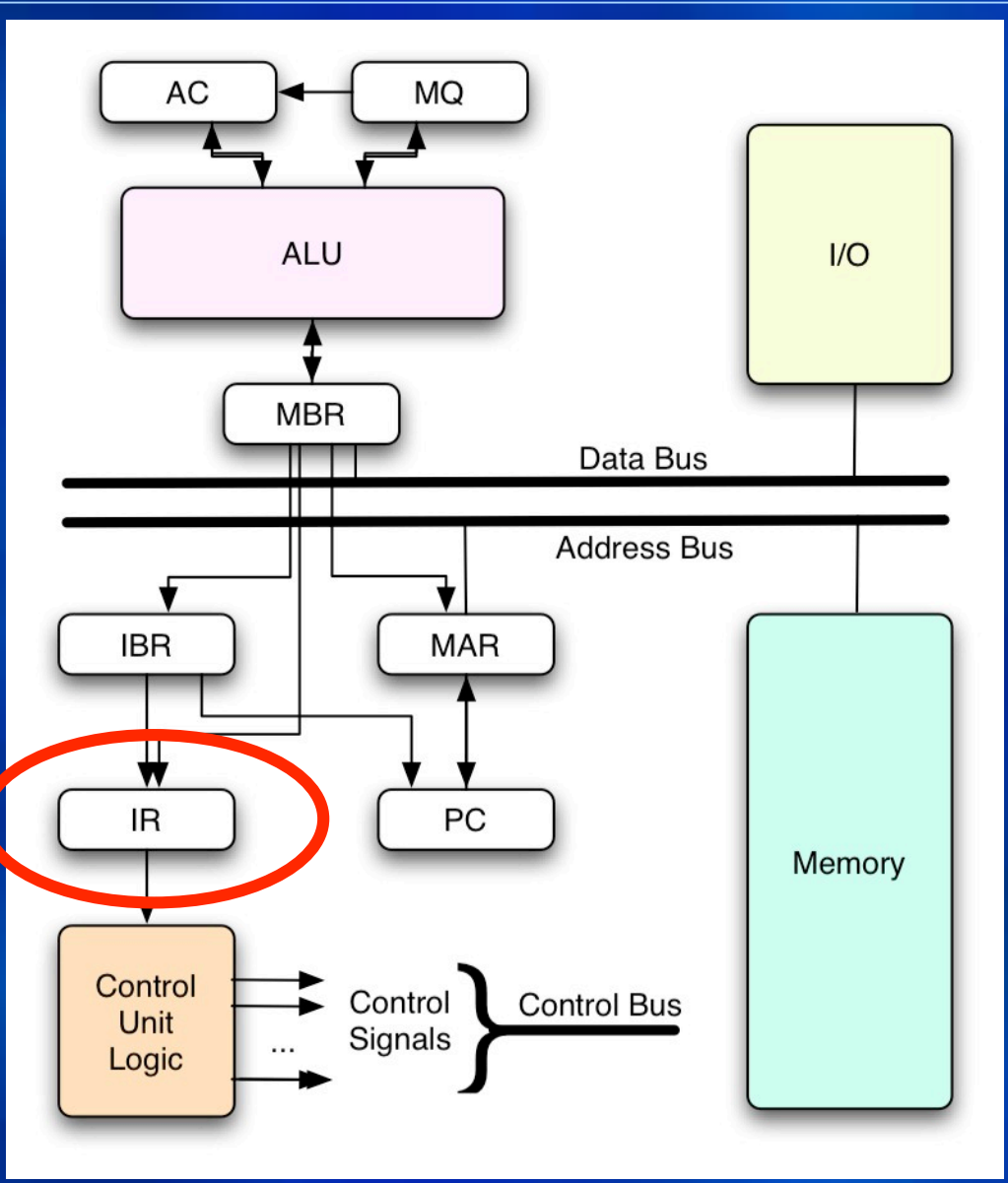
Especifica la dirección en memoria de la palabra que va a ser escrita o leída en MBR.



Detalles de la estructura

IR:
Instruction Register

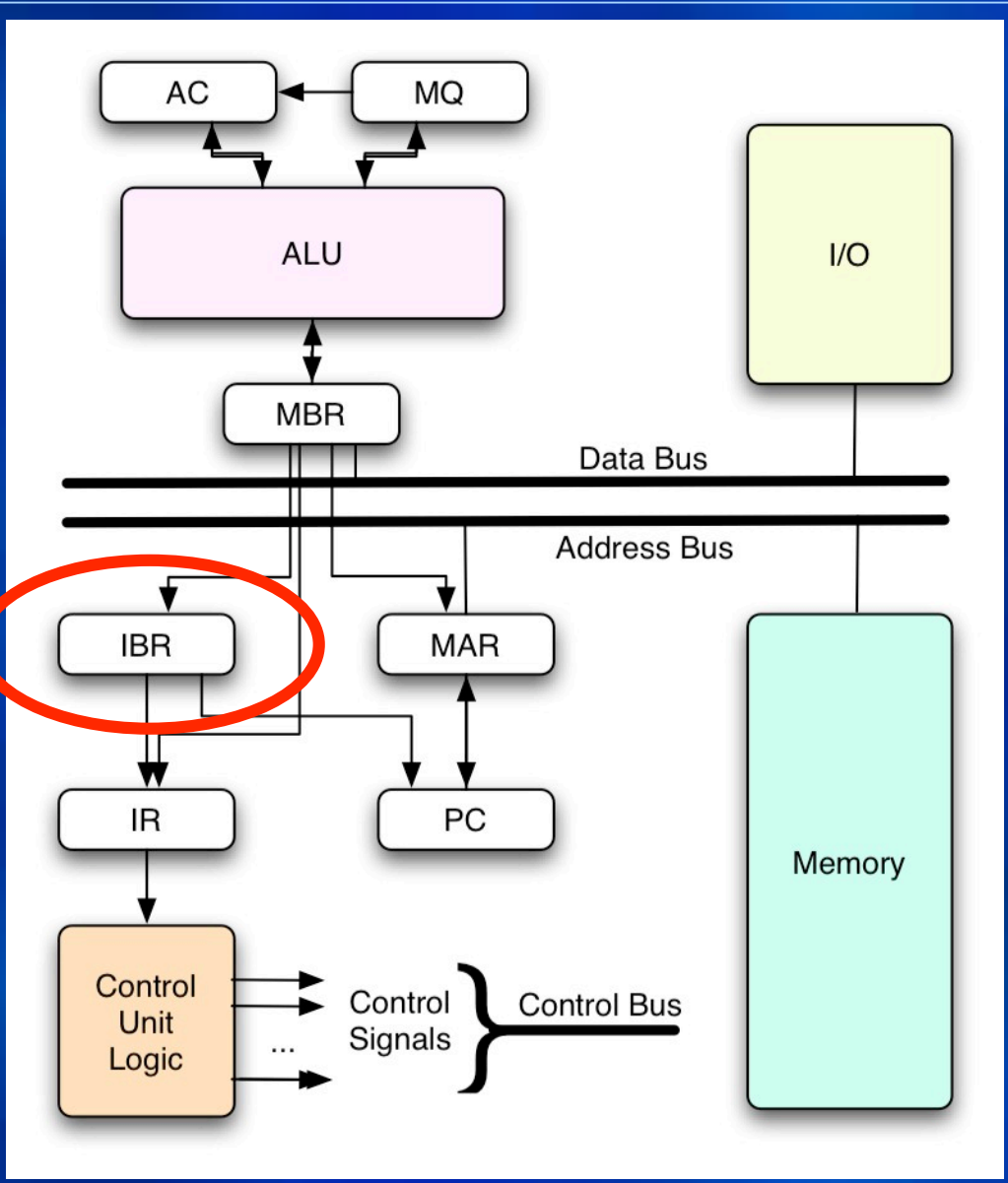
Contiene los 8 bits del código de operación de la instrucción que se va a ejecutar.



Detalles de la estructura

IBR:
Instruction Buffer Register

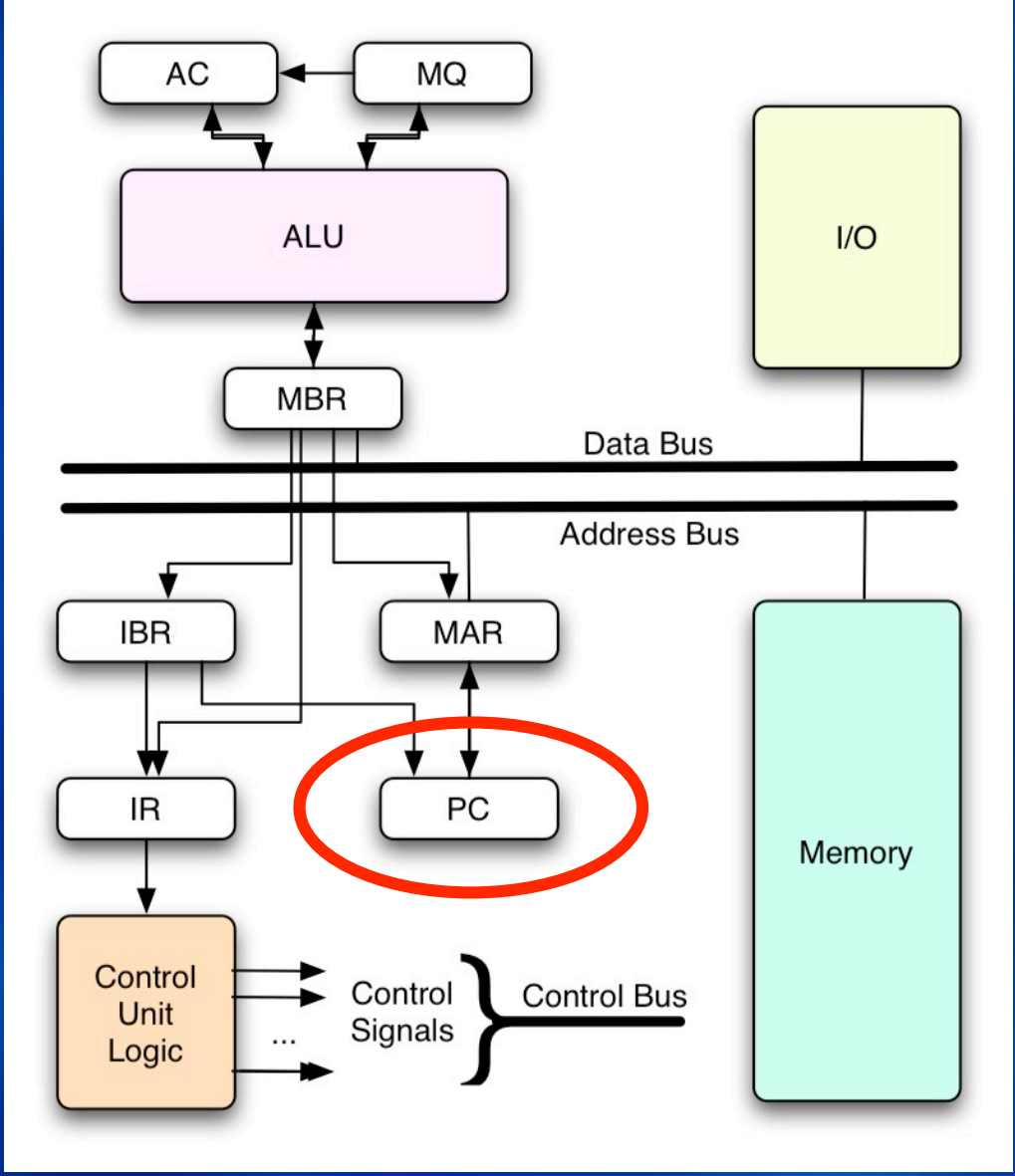
Empleado para almacenar temporalmente la instrucción contenida en la parte derecha de una palabra en memoria.



Detalles de la estructura

PC:
Program Counter

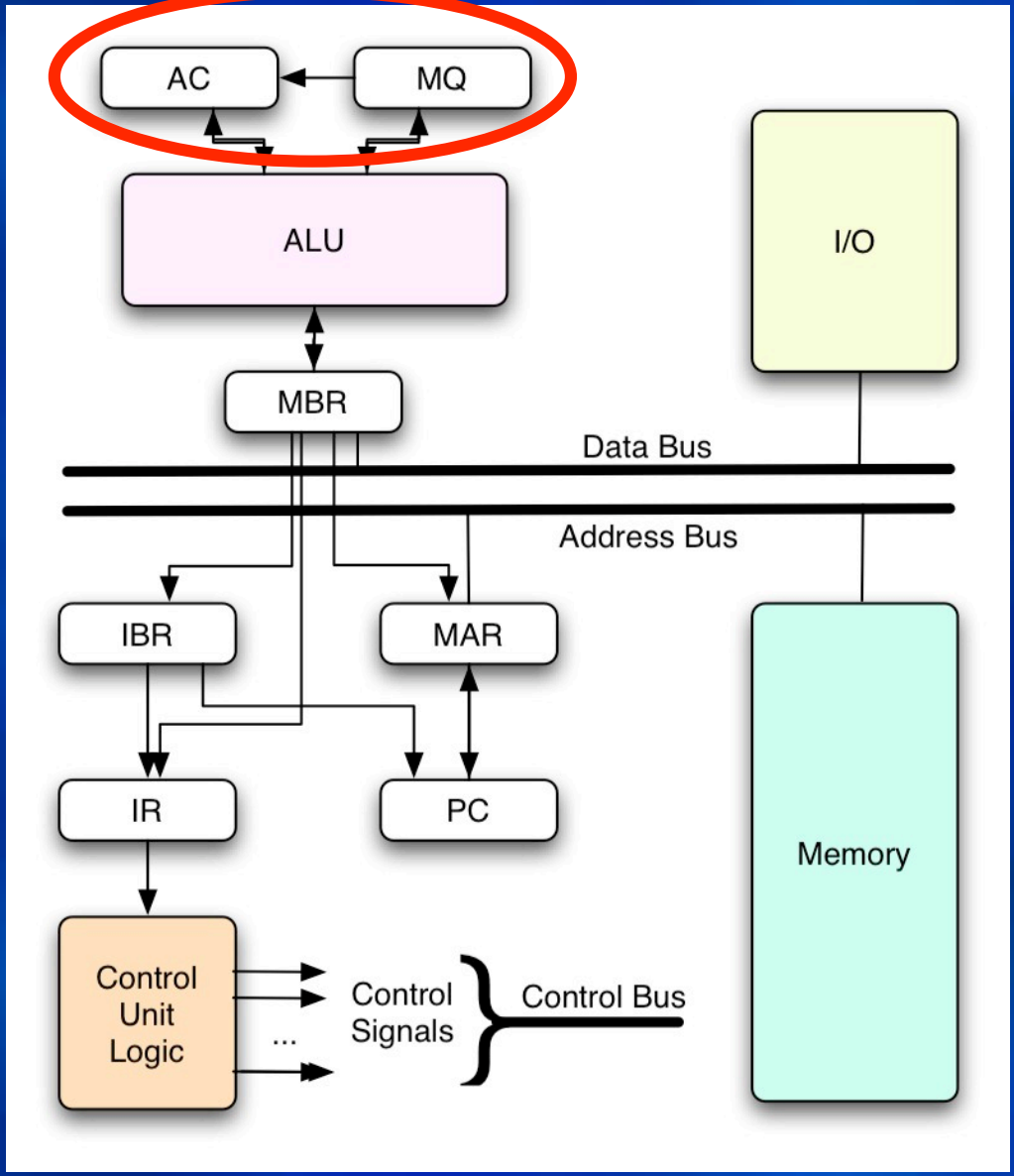
Contiene la dirección de la próxima pareja de instrucciones que van a ser captadas de la memoria.



Detalles de la estructura

AC y MQ:
Accumulator y Multiplier Quotient

Se emplean para almacenar operandos y resultados de operaciones de la ALU temporalmente. Por ejemplo, el resultado de multiplicar dos números de 40 bits es un número de 80 bits; los 40 bits más significativos se almacenan en AC y los menos significativos se almacenan en MQ.



Registros en CPU

Un procesador incluye registros visibles para el usuario y registros de control/estado.

Registros visibles:

- pueden referenciarse en las instrucciones de máquina.

- pueden ser:

1. Uso general
2. Datos
3. Direcciones
4. Códigos de condición

Registros de Propósito General

¿cuántos registros de propósito general ?

- Variable: 8, 32, … 1024
  - Más en RISC, menos en CISC
- Si hay muy pocos registros entonces se necesitan demasiados accesos a memoria.
- Muchos más registros no reducen considerablemente las referencias a memoria y hace la CPU más compleja.

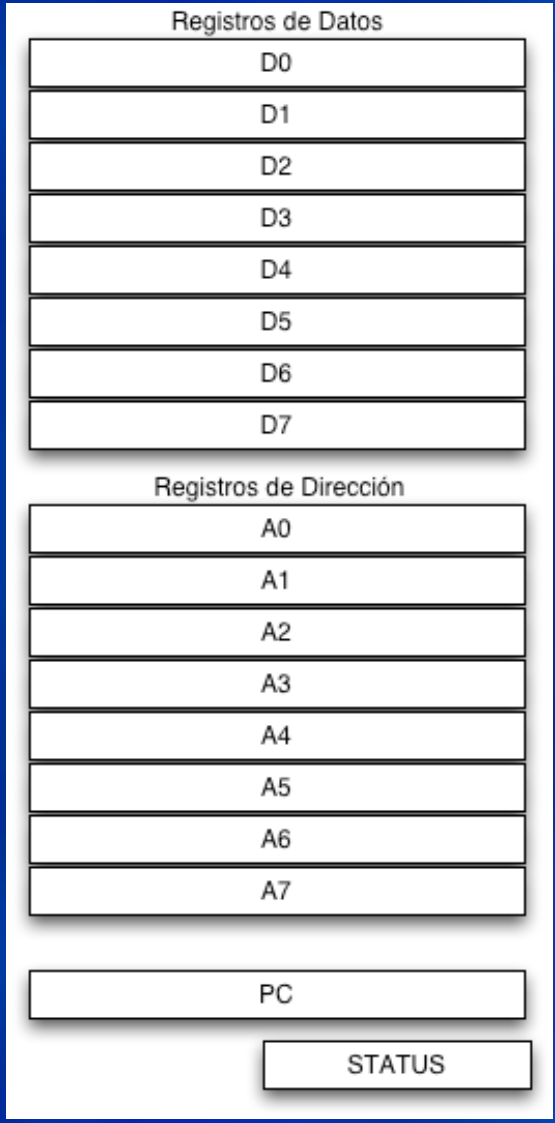
Tamaño de los Registros

¿De cuántos bits deben ser los registros?

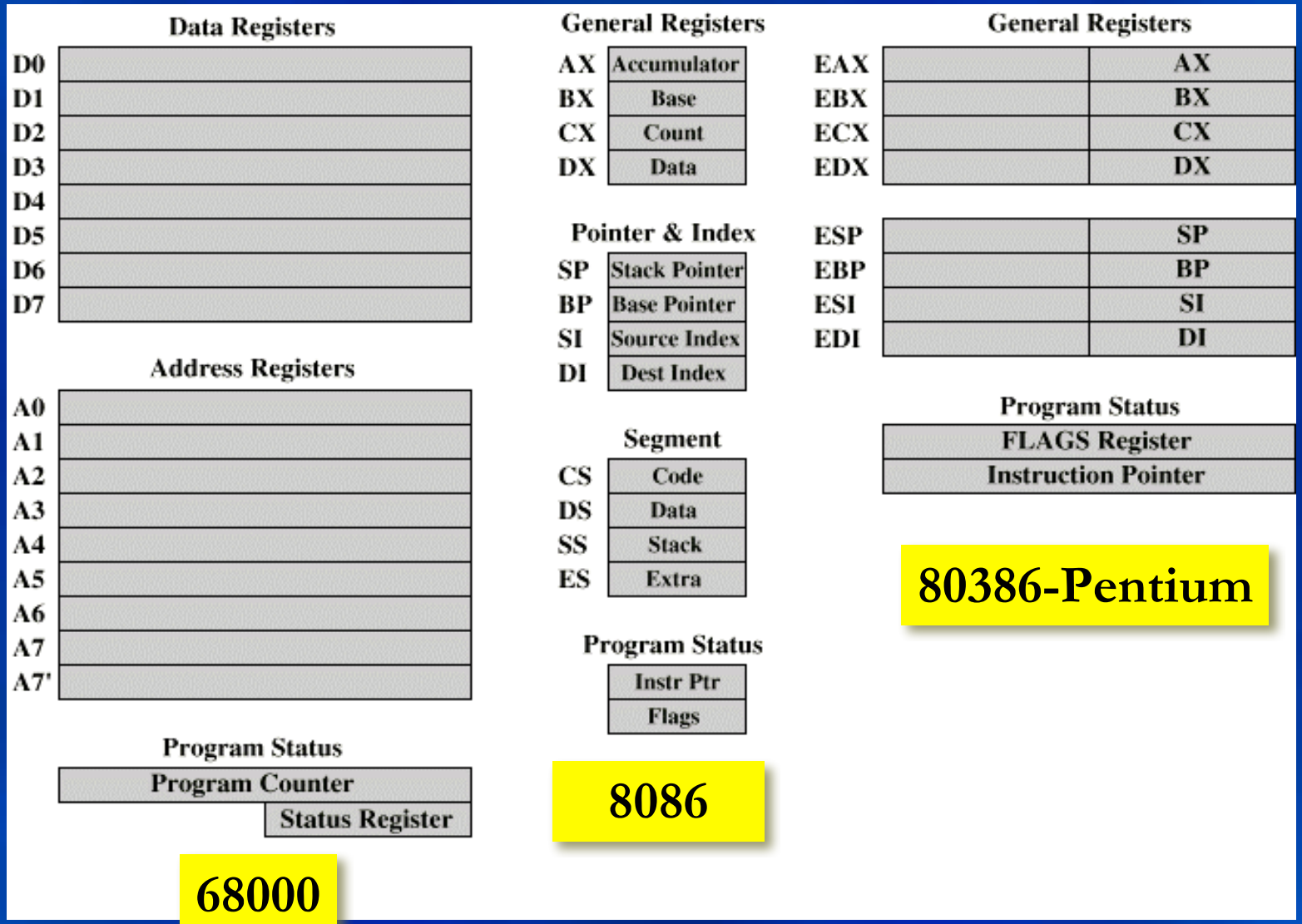
- Deben ser de un número suficiente tal que se puedan manejar las direcciones a memoria (posiblemente con direccionamiento indirecto).
- Además deben ser capaces de manejar una palabra de datos completa (tipicamanete un entero).
- A veces se combinan dos registros para conformar uno solo.

Registros del 68000

- El Motorola 68000 es un procesador clásico en el ámbito de los sistemas empotrados
- Su sucesor es el PowerPC (PPC)



Registros del 68000/8086/Pentium



Conjunto de instrucciones

Las instrucciones de una CPU a otra difieren bastante, sin embargo en todas las CPU se puede encontrar el siguiente conjunto de instrucciones:

1.- Instrucciones de transferencias de datos

2.- Instrucciones aritméticas

3.- Instrucciones lógicas

4.- Control de flujo

5.- Entrada / Salida

Instrucciones de Transferencia De Datos

- Se transfieren datos entre la memoria y los registros de la ALU o entre dos registros de la ALU.

CO	Instrucción	Descripción
00001010	`LOAD MQ`	Transferir el contenido del registro MQ a AC
00001001	`LOAD MQ,M(X)`	Transferir el contenido de la posición de memoria X a MQ
00100001	`STOR M(X)`	Transferir el contenido de AC a la posición de memoria X
00000001	`LOAD M(X)`	Transferir M(X) a AC
00000010	`LOAD –M(X)`	Transferir –M(X) a AC
00000100	`LOAD |M(X)|`	Transferir |M(X)| a AC

Instrucciones de Salto Incondicional

- Normalmente, la unidad de control ejecuta instrucciones secuencialmente en la memoria. Las instrucciones de salto pueden cambiar este orden (ej. Operaciones repetitivas).

Codop	Instrucción	Descripción
00001101	`JUMP M(X,8:19)`	Saltar a la instrucción indicada por la mitad izquierda de M(X)
00001110	`JUMP M(X,28:39)`	Saltar a la instrucción indicada por la mitad derecha de M(X)

Instrucciones de Salto Condicional

- El salto depende de una condición, esto permite puntos de decisión.

Codop	Instrucción	Descripción
00001111	`JUMP +M(X,8:19)`	Si AC ≥ 0 saltar a la instrucción indicada por la mitad izquierda de M(X)
00001000	`JUMP +M(X,28:39)`	Si AC ≥ 0 saltar a la instrucción indicada por la mitad derecha de M(X)

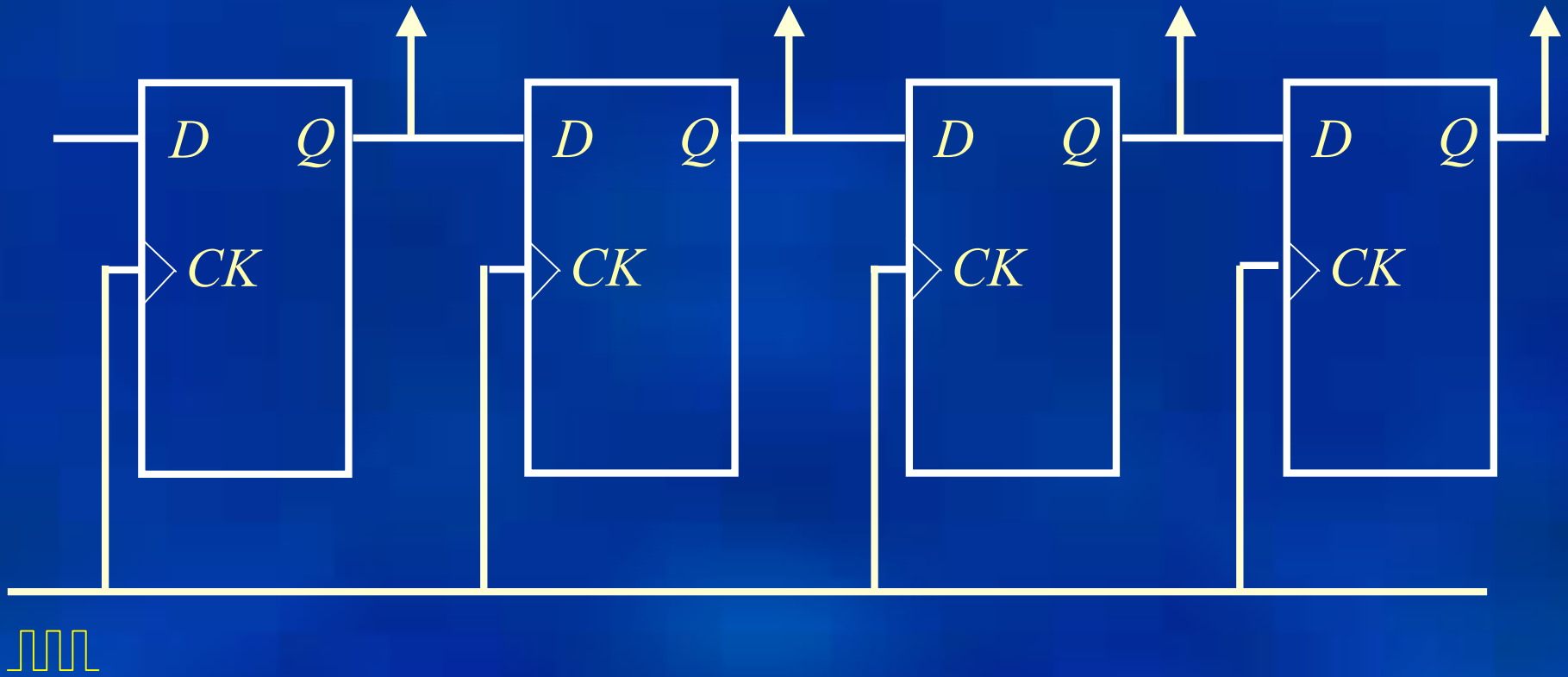
Instrucciones Aritméticas

- Son las operaciones realizadas por la ALU.

Codop	Instrucción	Descripción
00000101	`ADD M(X)`	AC ← AC + M(X)
00000111	`ADD |M(X)|`	AC ← AC + |M(X)|
00000110	`SUB M(X)`	AC ← AC - M(X)
00001000	`SUB |M(X)|`	AC ← AC - |M(X)|
00001011	`MUL M(X)`	[AC][MQ] ← AC × M(X)
00001100	`DIV M(X)`	[AC][MQ] ← AC ÷ M(X)
00010100	`LSH`	AC ← AC × 2
00010101	`RSH`	AC ← AC ÷ 2

¿Cómo se implementan por hardware?

RSH = desplazar a la derecha = división entre dos



Instrucciones de Modificación de Direcciones

- Permite que la ALU haga operaciones con las direcciones y las inserte en instrucciones almacenadas en memoria. Esto permite una considerable flexibilidad de direccionamiento en un programa.

Codop	Instrucción	Descripción
00010010	`STOR M(X,8:19)`	Reemplazar el campo de dirección de la izquierda de M(X) por los 12 bits de la derecha de AC.
00010011	`STOR M(X,28:39)`	Reemplazar el campo de dirección de la derecha de M(X) por los 12 bits de la derecha de AC.

CISC vs RISC

- Los tiempos de ejecución (ciclos de reloj) dependen de la instrucción
- Instrucciones más complejas necesitan más ciclos
- Se puede optimizar la CPU uniformizado los ciclos necesarios para cada instrucción
- RISC: Reduced Instruction Set Computer
- CISC: Complex Instruction Set Computer

Ejercicio

- Escribir un programa que sume el número almacenado en la posición 867 más el número almacenado en la posición 562. El resultado de la suma (sin considerar acarreo) se debe almacenar en la posición 778.

Ejercicio

- Escribir un programa que sume el número almacenado en la posición 867 más el número almacenado en la posición 562. El resultado de la suma (sin considerar acarreo) se debe almacenar en la posición 778.

```
LOAD M(867)        % transfiere el contenido de 867 a AC
ADD M(562)         % AC ← AC + M(562)
STOR M(778)        % transfiere AC a la memoria 778
```

¿Cómo se almacena el programa?

```
LOAD M(867)        00000001 001101100011
ADD M(562)         00000100 001000110010
STOR M(778)        00100001 001100001010
```

Ejercicio

- Escribir un programa que divida el número almacenado en la posición 867 por 8. El resultado de la división (sin considerar el resto ni la parte fraccionaria) se debe almacenar en la posición 778.

Ejercicio

- Escribir un programa que divida el número almacenado en la posición 867 por 8. El resultado de la división (sin considerar el resto ni la parte fraccionaria) se debe almacenar en la posición 778.

```
LOAD M(867)        % transfiere el contenido de 867 a AC
RSH                % divide el acumulador entre dos
RSH                % divide el acumulador entre dos
RSH                % divide el acumulador entre dos
STOR M(778)        % transfiere AC a la memoria 778
```

Ejercicio

- Escribir un programa que compare el número almacenado en la posición 867 con el número almacenado en la posición 562. Si el primero es menor que el segundo copiar el contenido de la memoria 500 en la memoria 501, de lo contrario se almacena en 501 el contenido de 867 menos el contenido de 562.

Ejercicio

- Escribir un programa que compare el número almacenado en la posición 867 con el número almacenado en la posición 562. Si el primero es menor que el segundo copiar el contenido de la memoria 500 en la memoria 501, de lo contrario se almacena en 501 el contenido de 867 menos el contenido de 562.

```
1. LOAD M(867)        % transfiere el contenido de 867 a AC
2. SUB M(562)         % AC ← AC - M(562)
3. JUMP +M(3,28:39)   % salta si AC ≥ 0 (i.e. M(867)≥ M(562))
4. LOAD M(500)        % transfiere el contenido de 500 a AC
5. STOR M(501)        % transfiere AC a la memoria 501
```

- En la posición derecha de 3 debe estar almacenado 5, de esta manera la tercera instrucción salta a la dirección 5 si AC ≥ 0.

Subrutinas

- Frecuentemente la misma pieza de código debe escribirse varias veces en muchas partes diferentes de un programa.
- En vez de repetir el código cada vez que sea necesario, hay una ventaja obvia si las instrucciones comunes se escriben solamente una vez.
- Un conjunto de instrucciones comunes que pueden utilizarse en un programa muchas veces se denomina **subrutina**.

Subrutinas

- Cada vez que la subrutina se utiliza en la parte del programa principal, se pasa al comienzo de la subrutina.
- Después que la subrutina ha sido ejecutada, se vuelve de nuevo al programa principal.

Uso de subrutinas

- Los registros y la memoria son comunes para el programa principal y para la subrutina,
  - **Ventaja**: la comunicación de los parámetros entre el programa principal y la subrutina es simple y rápida.
  - **Desventaja**: el programador puede olvidar que ciertos registros usados por en el programa principal no deben ser alterados en la subrutina, esto puede causar serios problemas y la detección de este error es difícil.

Ejemplo de Subrutinas

Se desea hacer un programa que realice las siguientes operaciones:

(0100) ← (0100) + 5
(0200) ← (0200) + 5
(0204) ← (0204) + 5

Ejemplo de Subrutinas

Se desea hacer un programa que realice las siguientes operaciones:

(0100) ← (0100) + 5
(0200) ← (0200) + 5
(0204) ← (0204) + 5

La mejor solución sería utilizando una subrutina que tenga como parámetro una dirección X y que realice la operación:

(X) ← (X) + 5

El programa principal debe cargar correctamente el registro X y llamar a la subrutina tres veces.

Ejemplo de Subrutinas

Se desea hacer un programa que realice las siguientes operaciones:

(0100) ← (0100) + 5
(0200) ← (0200) + 5
(0204) ← (0204) + 5

Programa principal

```
LOAD X,0100
CALL SUM5
LOAD X,0200
CALL SUM5
LOAD X,0204
CALL SUM5
```

Subrutina

```
SUM5 LOAD A,(X)
     ADD 5
     STORE (X),A
     RET
```

¿Qué pasa si el programa principal estaba usando A?

Ejemplo de Subrutinas

Se desea hacer un programa que realice las siguientes operaciones:

(0100) ← (0100) + 5
(0200) ← (0200) + 5
(0204) ← (0204) + 5

Solución: se usa la pila

Programa principal

```
LOAD X,0100
CALL SUM5
LOAD X,0200
CALL SUM5
LOAD X,0204
CALL SUM5
```

Subrutina

```
SUM5 PUSH A
     LOAD A,(X)
     ADD 5
     STORE (X),A
     POP A
     RET
```

Uso de la pila (Stack)

Existe una memoria direccionada por el registro SP (stack pointer). ¿Cómo se usa?

Cada vez que se hace PUSH X:

(SP) ← X
SP ← SP + 1

Cada vez que se hace POP X:

SP ← SP - 1
X ← (SP)

(es posible hacer PUSH X y luego POP Y)

Hemos visto

- Estructura básica de una CPU
- Ciclo de operación
- Formatos de instrucción
- Tipos de instrucciones
- Subrutinas