

Diseño de una computadora digital Parte 1

Diseño de una computadora digital

El *diseño de una computadora digital* es:

La *organización del 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** cumpliendo determinadas funciones lógicas.
- 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

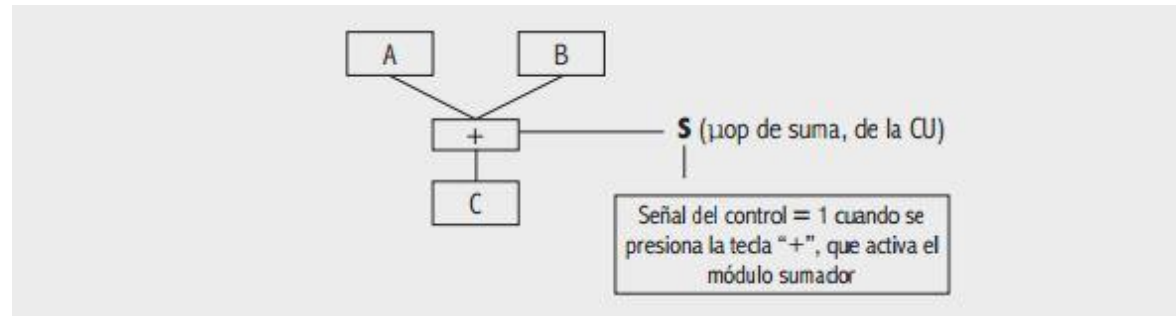
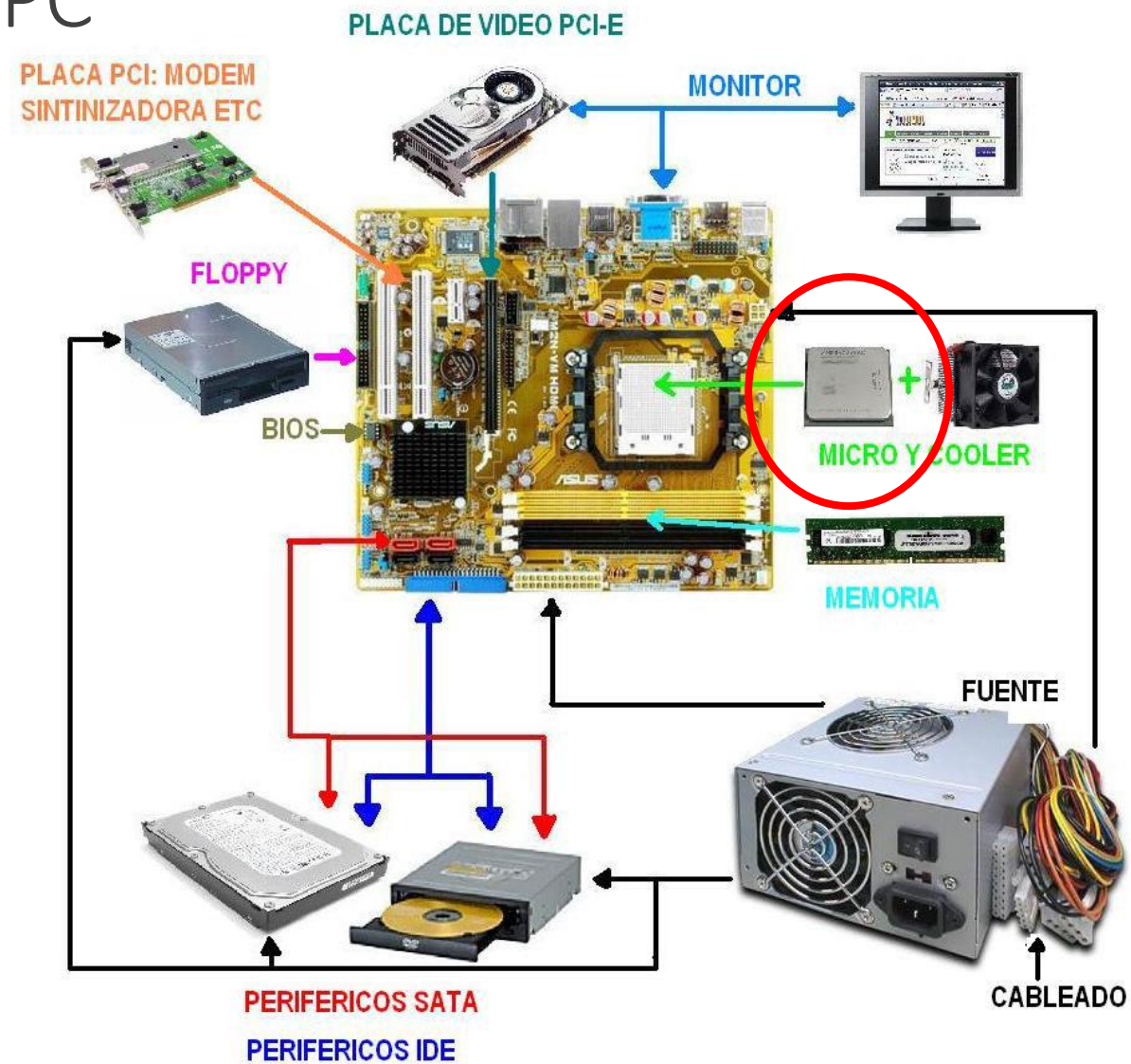


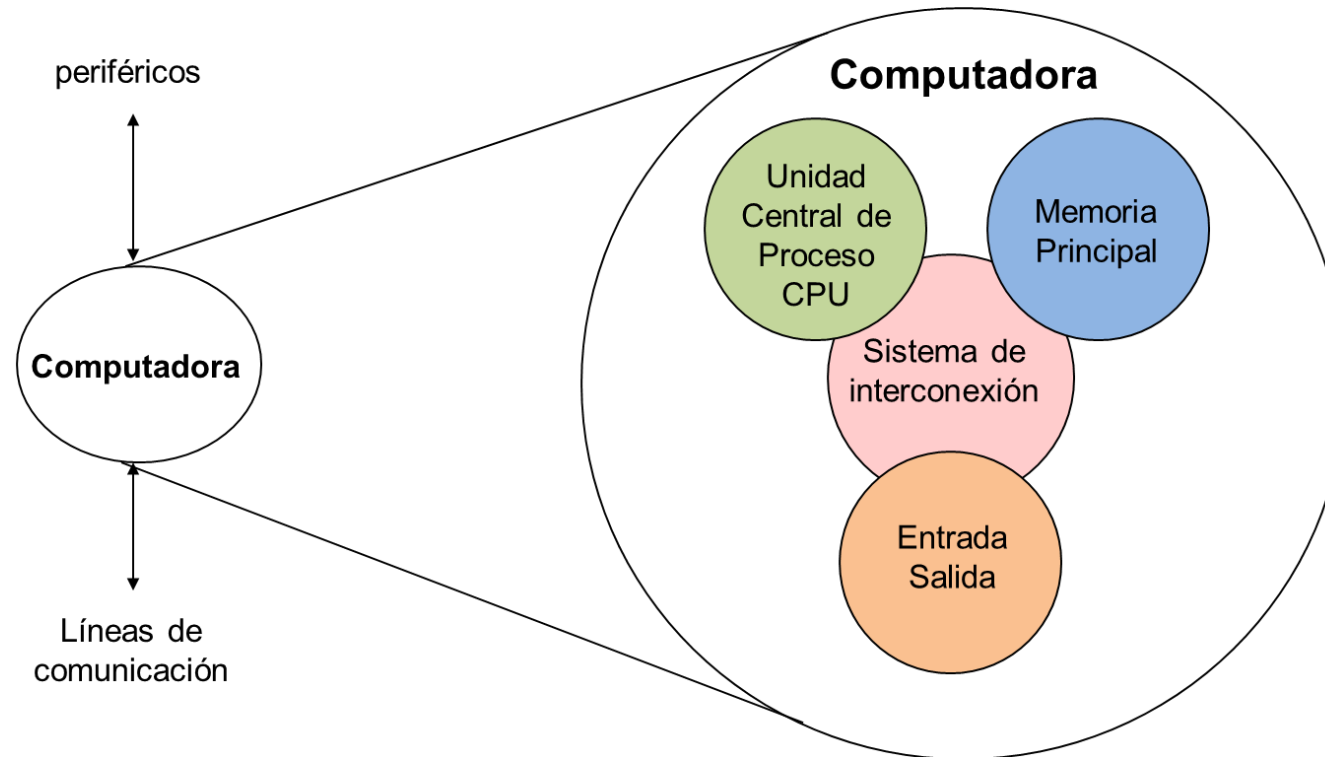
Diagrama de bloque de un dispositivo que suma los bits de los registros A y B.

- **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

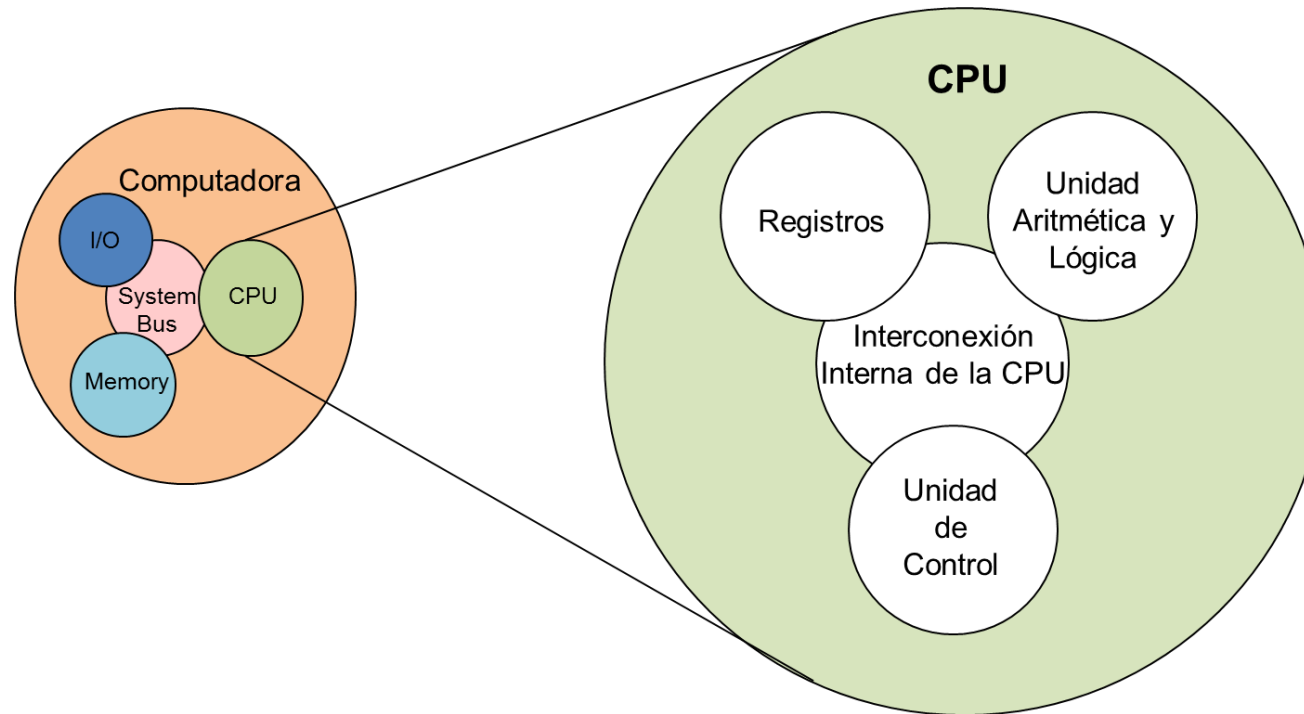
Partes de un PC



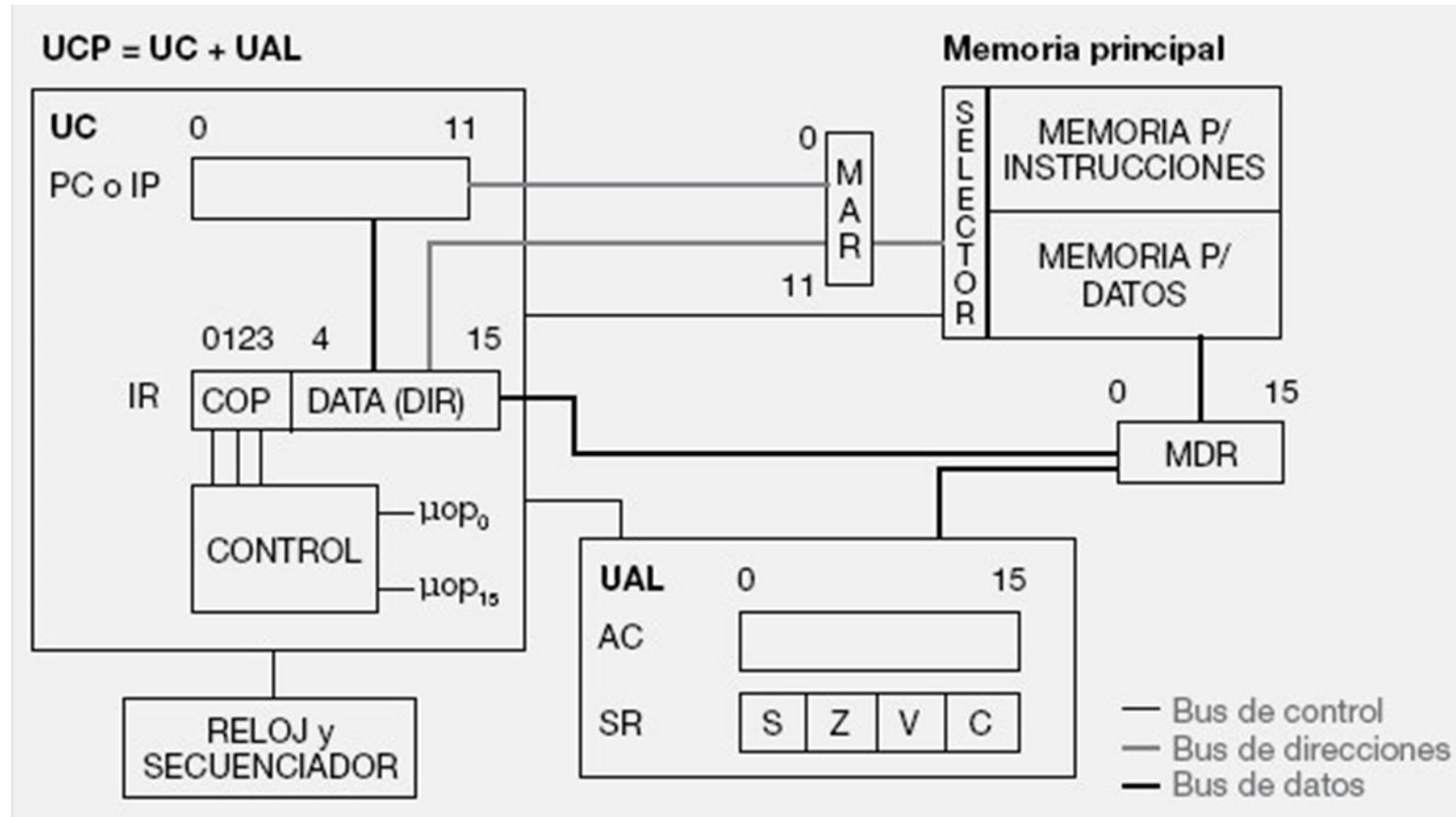
Modelo computadora digital



Modelo computadora digital



Modelo computadora "X"



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***.

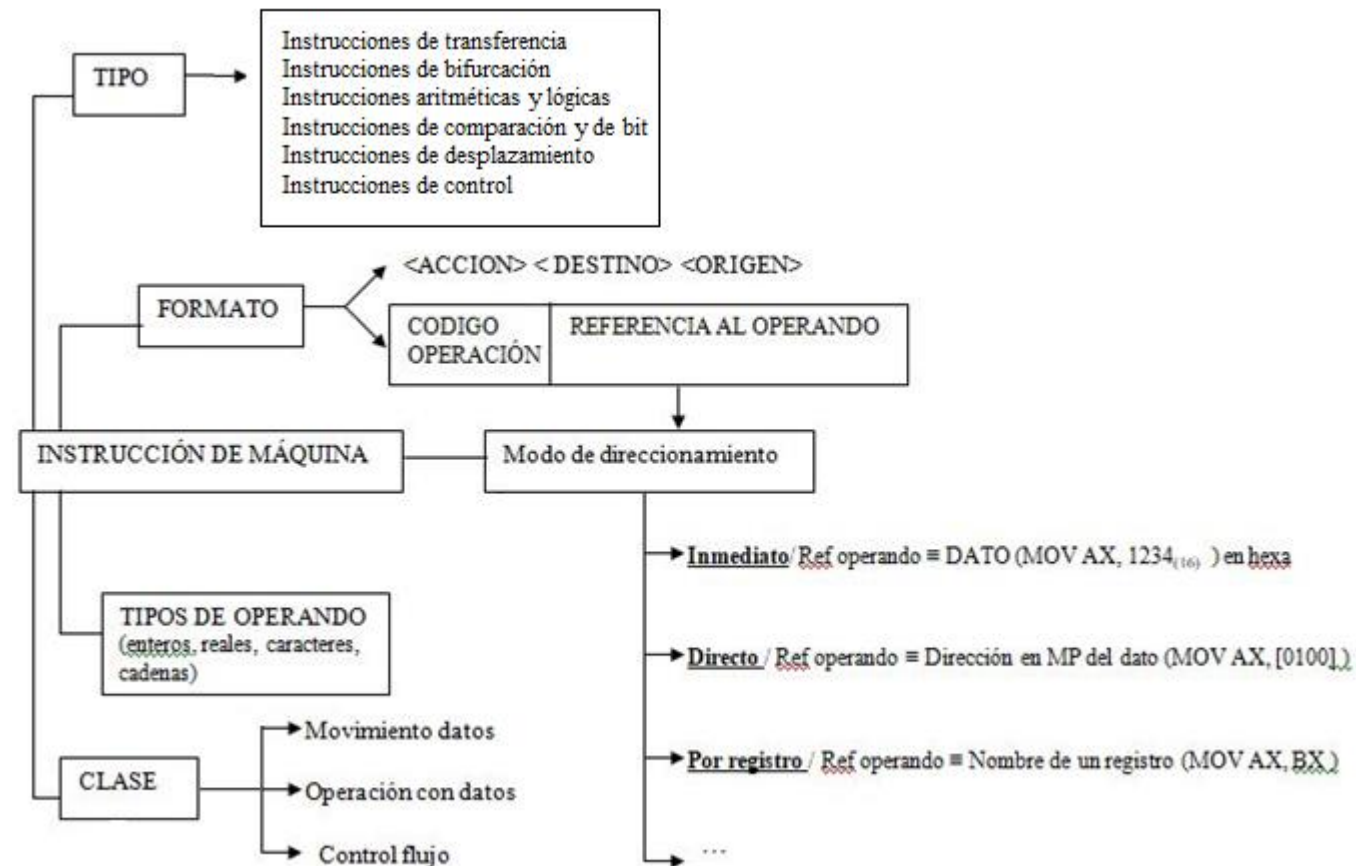
Instrucciones

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:

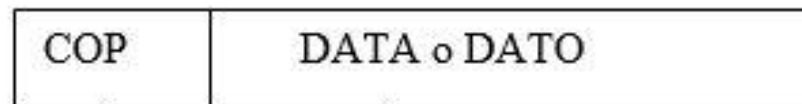
Es la combinación de bits que la unidad de control de la CPU interpreta para generar microoperaciones que permitan su ejecución.

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".

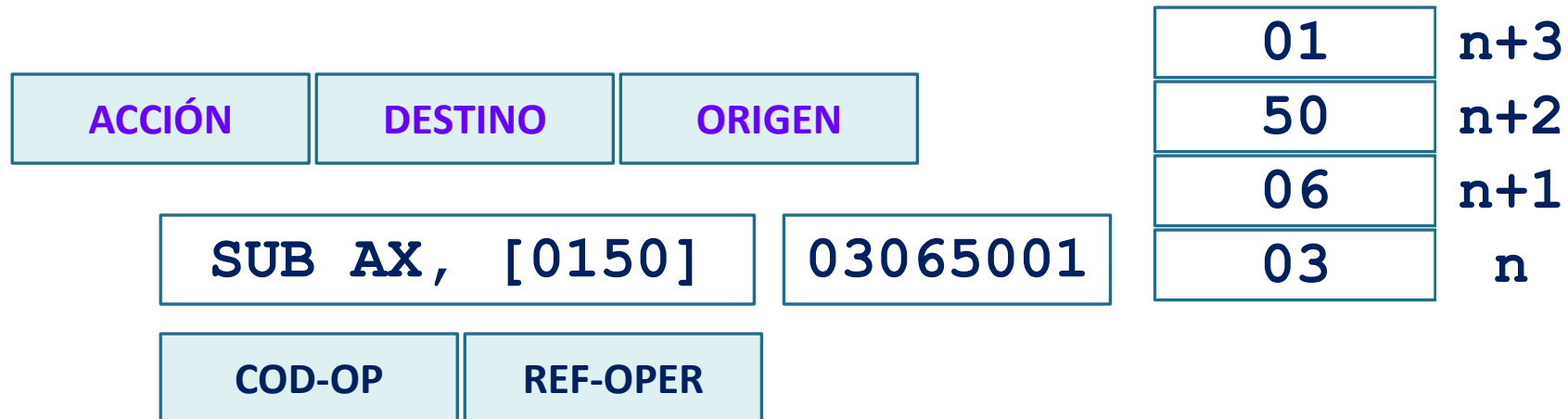
Datos sobre los que actúo:

- ♦ 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

Instrucciones

Para “máquinas de una dirección” el formato básico de la instrucción define:

- Una acción a realizar
- El punto destino donde se verá concretada la acción al terminar de ejecutarse la instrucción
- Un punto origen, donde se encuentra el operando que será afectado por la acción



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.

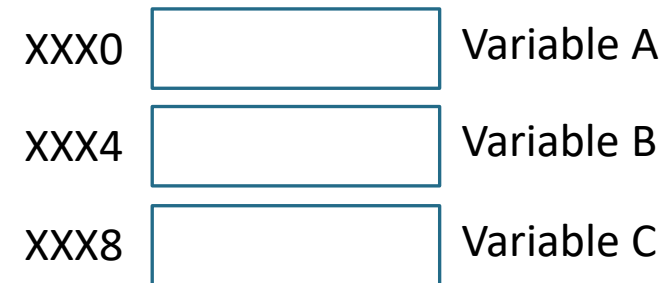
Lenguaje de Alto Nivel vs. Lenguaje de Bajo Nivel

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")  
}
```

En ejecución

En Memoria



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.

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.

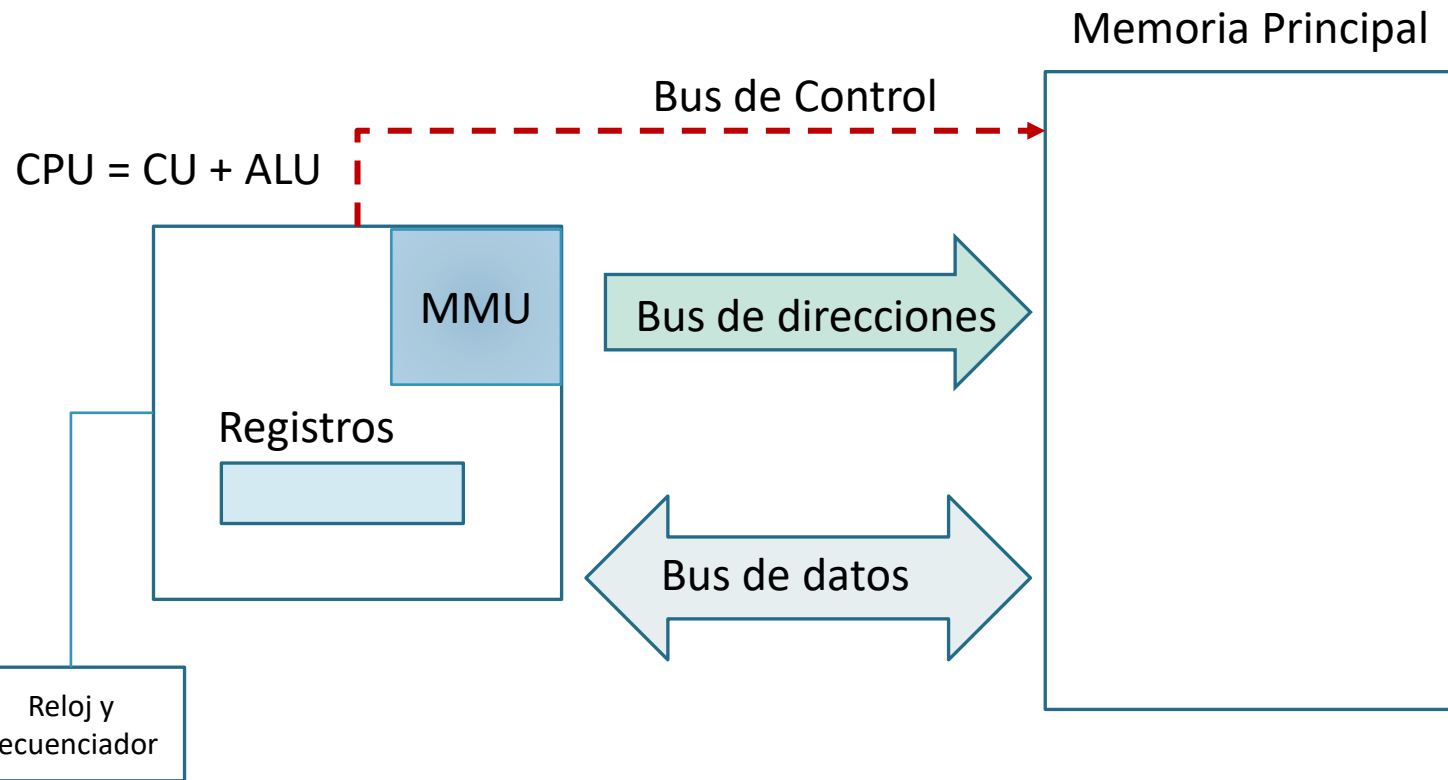
Estudiamos Assembler para comprender más la unidad de control (CU) y para entender las interacciones entre lo físico y lo lógico.

CPU – Unidad Central de Procesamiento

➤ La función de la CPU se puede separar en partes:

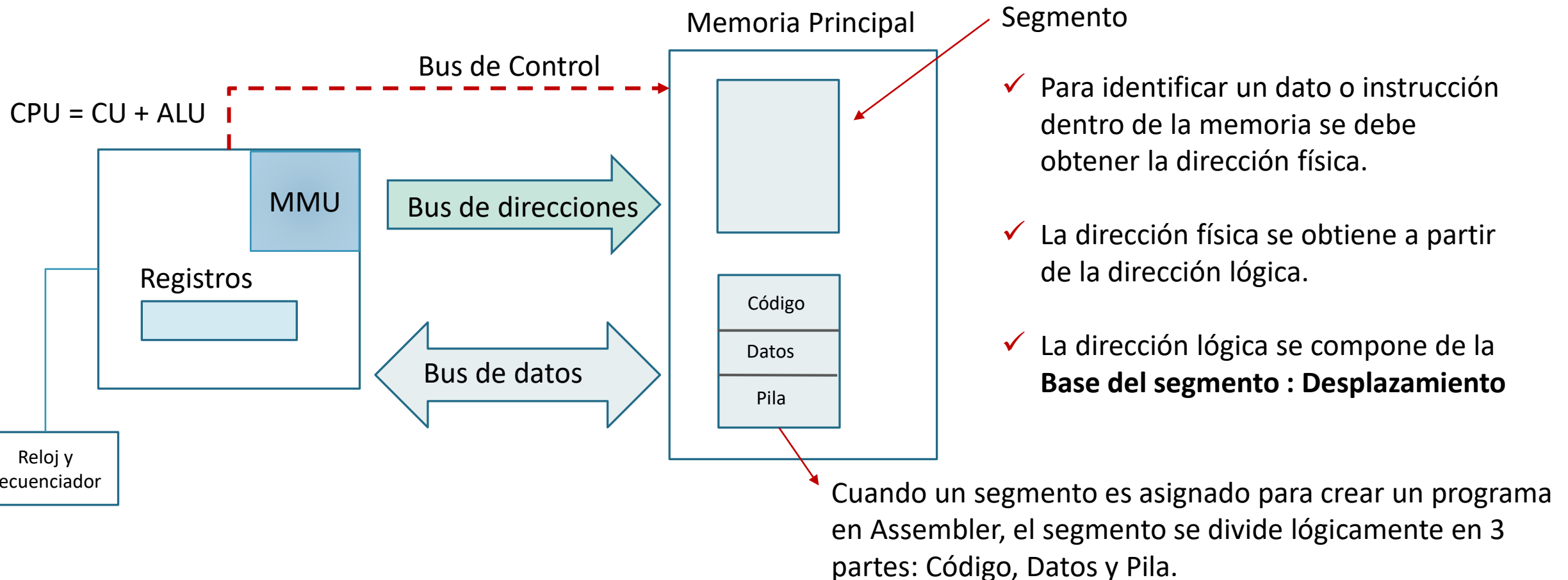
1. Tratamiento de instrucciones  Se encarga la **unidad de control (CU o *Control Unit*)**, sincronizado por el generador de pulsos de reloj
2. Operación de los datos  Se encarga la **unidad aritmético-lógica (ALU o *Aritmetic Logic Unit*)**
3. Calculo de dirección de Memoria  Se encarga la **unidad 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.

CPU – Memoria Principal – Modo real



Modo real - Direccionamiento

Dirección Lógica:

Base del segmento : Desplazamiento

También se la conoce como dirección Segmentada

Donde la base del segmento será el contenido de los registros CS, SS ó DS (puede haber también un extra segment ES) y el desplazamiento serán los contenidos del IP, SP o la dirección de memoria apuntada respectivamente.

Ejemplo en Assembler

```
13E0:0100 mov ax,0002
```

```
13E0:0103 mov bx,0004
```

```
13E0:0106 add ax,bx
```

```
13E0:0108 int 20
```

```
13E0:010A
```

Siempre van a ser de este tipo las direcciones segmentadas

CS:IP

SS:SP

DS: [mem]

Modo real - Direccionamiento

Calculo de la dirección física:

Dirección Física = Base del Segmento x 10_H + Desplazamiento

Por ejemplo para la instrucción `mov bx,0004` las direcciones serán:

Dirección Segmentada: **13E0:0103**

Dirección Física: $13E0 \times 10_H + 0103 = 13E00 + 0103 = 13F03$

Dirección física del comienzo del segmento

Ejemplo en Assembler

```
13E0:0100 mov ax,0002
```

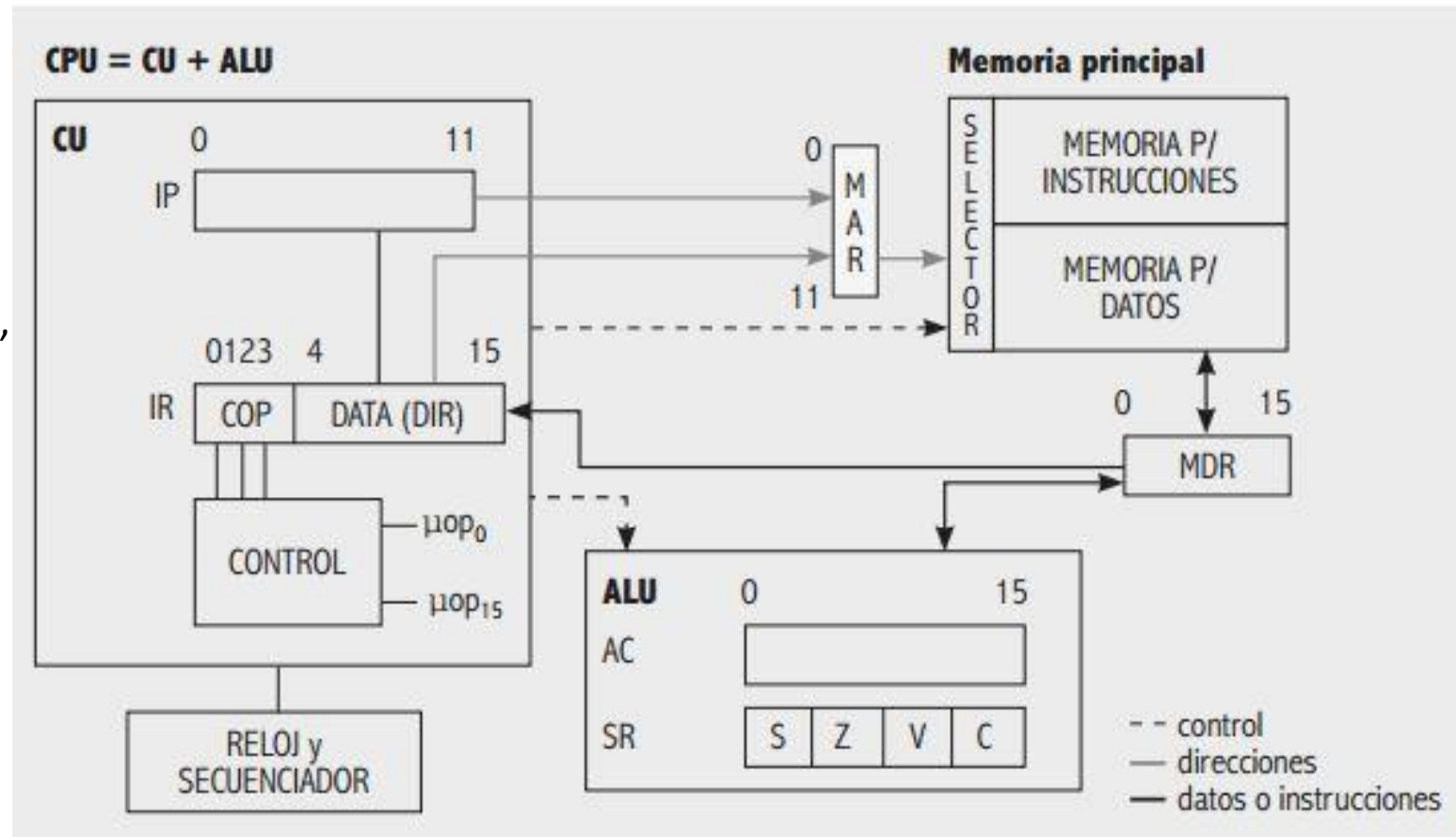
```
13E0:0103 mov bx,0004
```

```
13E0:0106 add ax,bx
```

```
13E0:0108 int 20
```

```
13E0:010A
```

Modelo de estudio



MAR: *Memory Address Register*

IP: *Instruction Pointer*

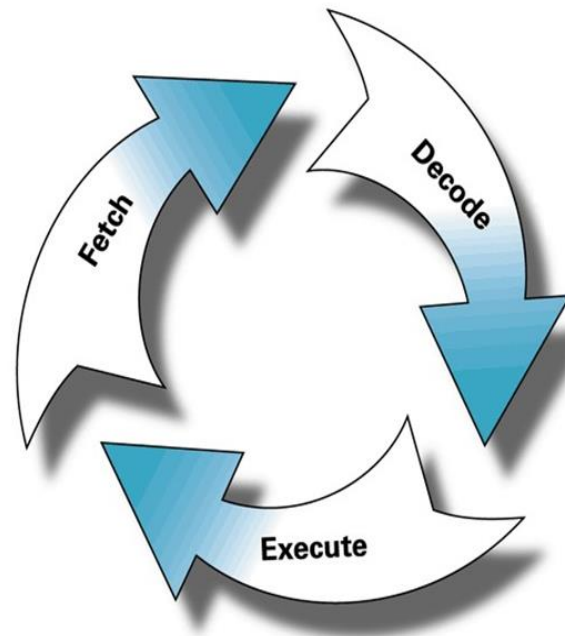
MDR: *Memory Data Register*

IR: *Instruction Register*

Ciclo de instrucción

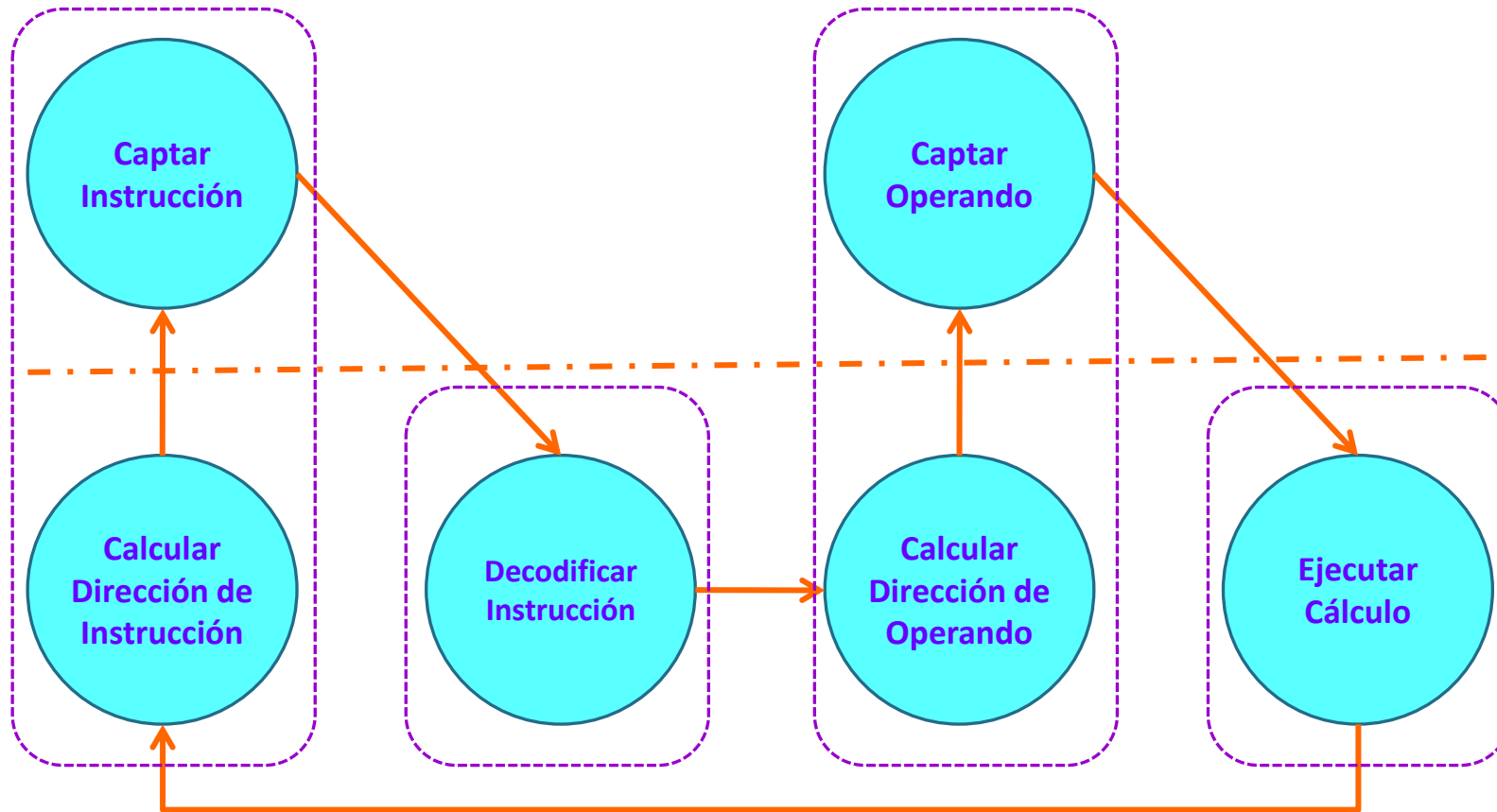
1. Recuperar la siguiente instrucción desde memoria (apuntada por el *program counter*) y luego incrementar el *program counter*.

2. Decodificar el patrón de bits en el registro de instrucción IR





3. Ejecutar la instrucción indicada en el registro de instrucción IR

Ciclo de instrucción



Fases de búsqueda y ejecución

Este proceso se puede dividir en las etapas siguientes:

1. Búsqueda de la instrucción en memoria. (BI)  **fase de búsqueda o fase fetch**
 2. Interpretación del código de instrucción. (Dec)
 3. Búsqueda del dato u operando afectado (si afecta a dato) por la instrucción. (BO)
 4. Generación de órdenes (Ejecución de la instrucción) al módulo que opera sobre ese dato y actualización del IP. (Ejec)
- 
- fase de ejecución o execute**

Diseño de una computadora digital

Segunda Parte

Profesora Silvana Panizzo

Emu8086

mov ax,0002

mov bx,0004

add ax,bx

int 20h

Ciclo de instrucción

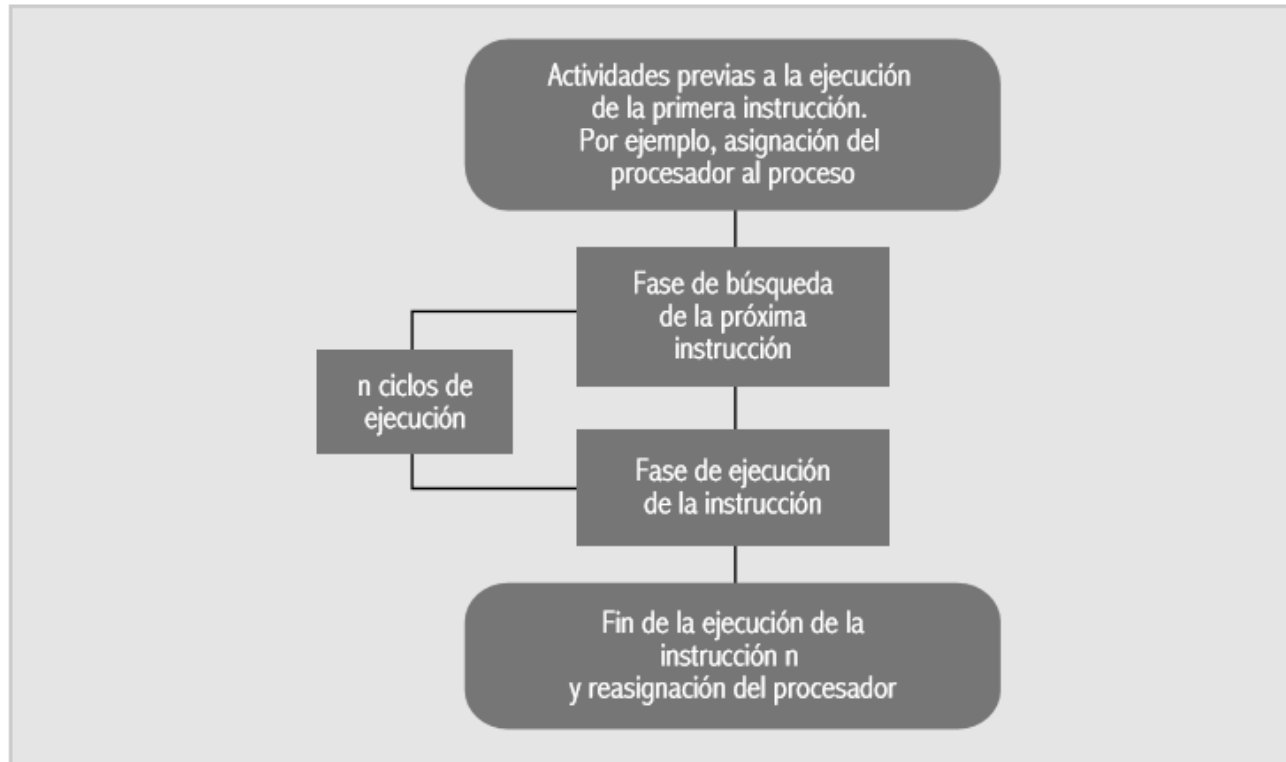


Fig. 8.2. Ciclo de instrucción para las n instrucciones de un programa.

Fase de búsqueda:

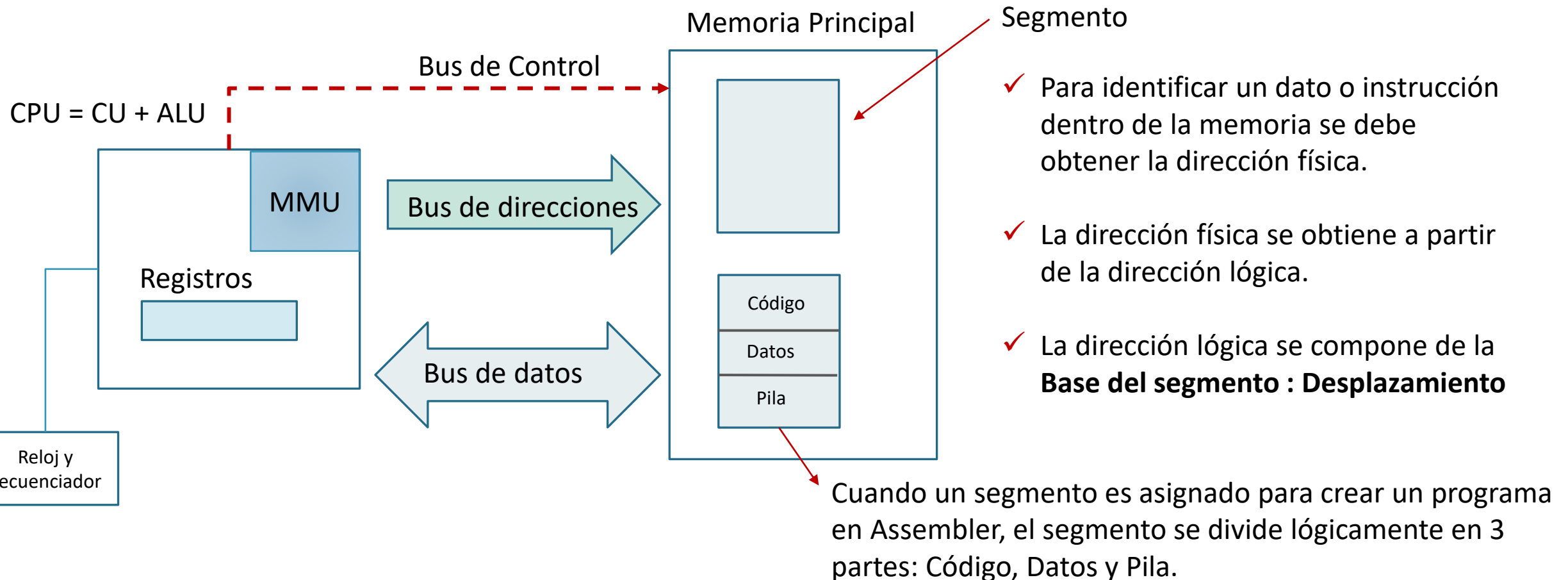
- Cálculo de la dirección física de la instrucción.
- Dar orden de lectura RD
- Se carga el registro IR

Fase de ejecución:

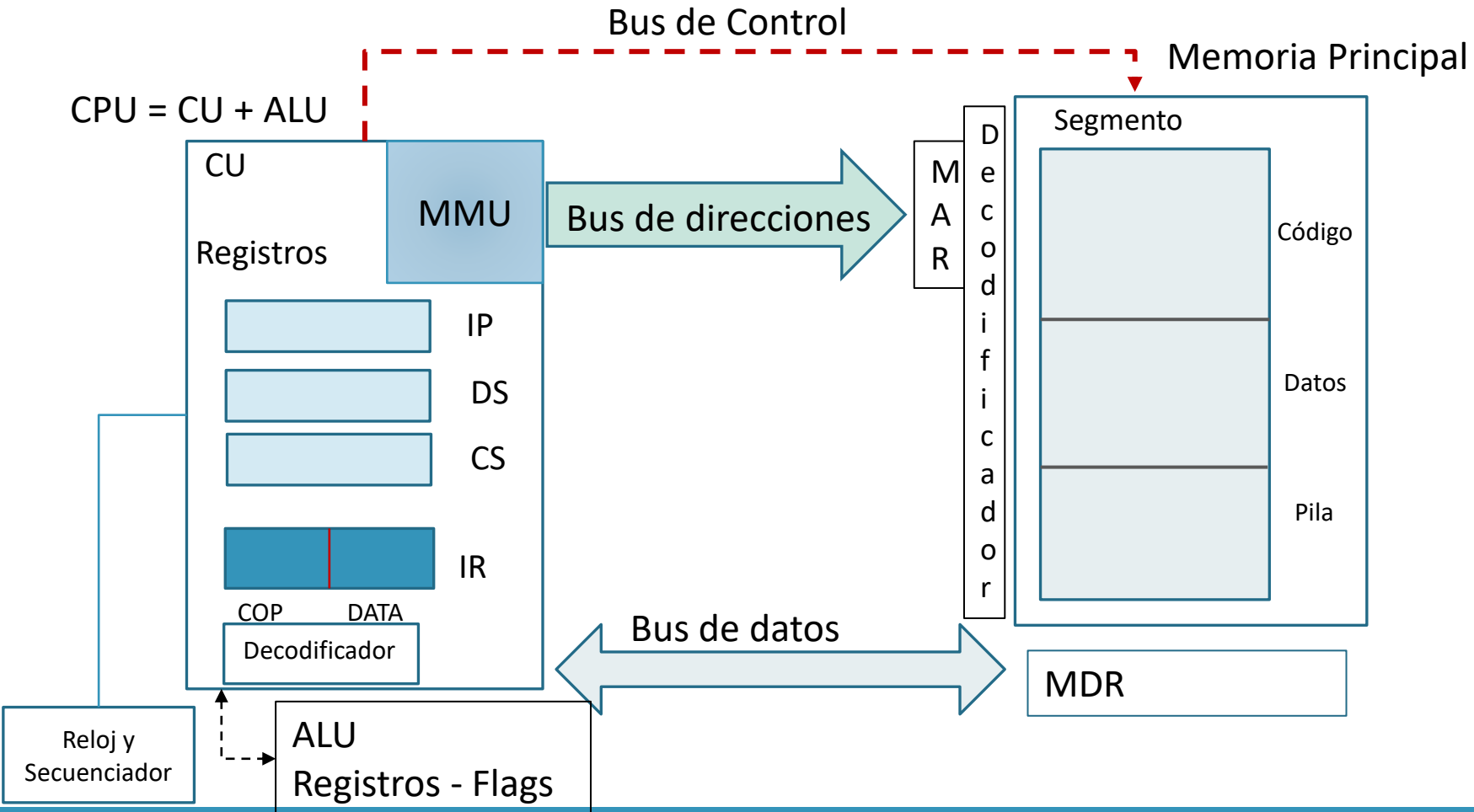
- Interpretar el código de la instrucción (Decodificar)
- Incrementar el IP
- Búsqueda del dato (**RD**) o Guardar el dato (**WR**) (si afecta)
- Generar orden al módulo para que opere el dato.

Repaso

CPU – Memoria Principal – Modo real

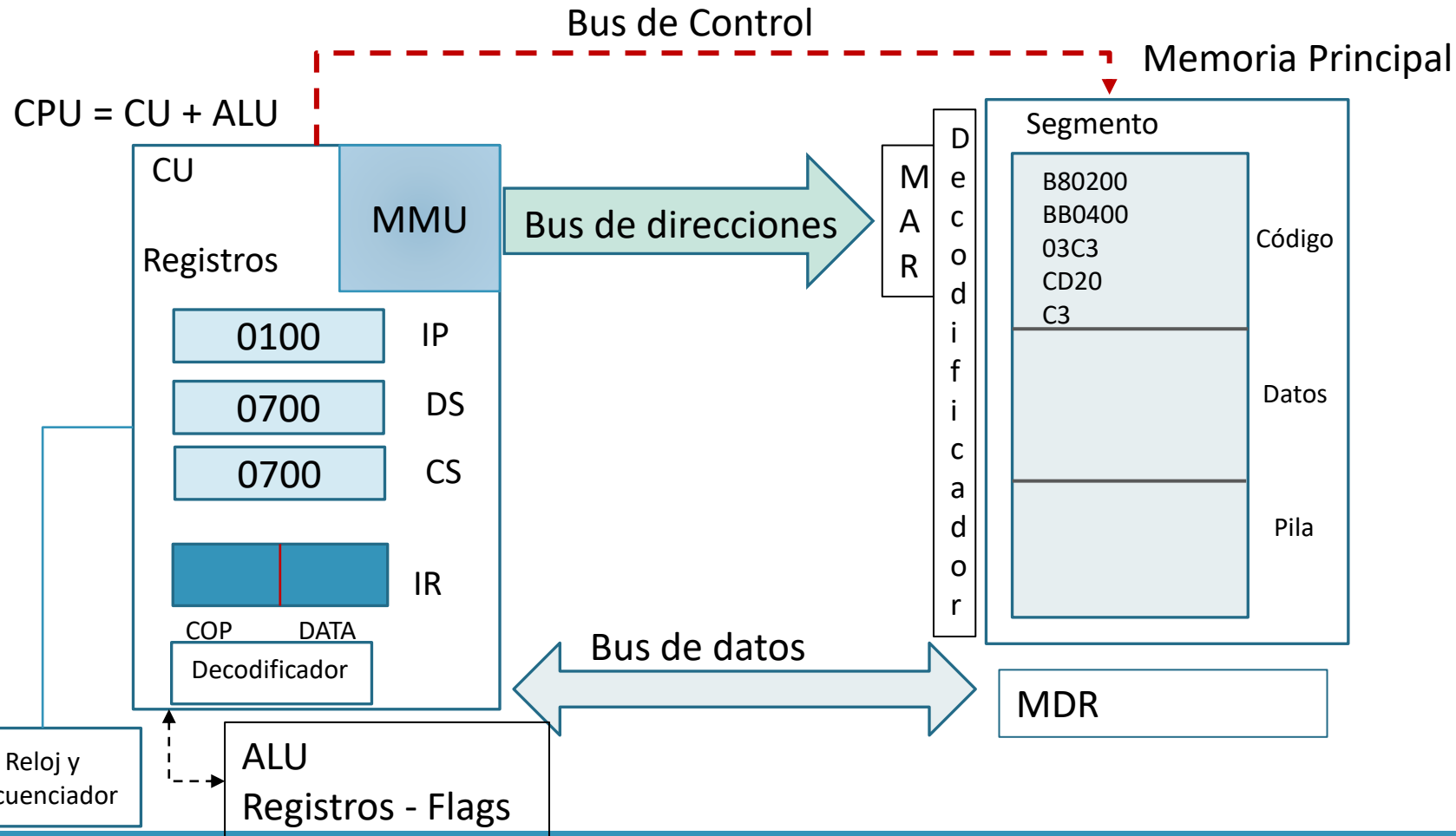


CPU – Memoria Principal – Modo real



```
0700:0100 mov ax,0002
0700:0103 mov bx,0004
0700:0106 add ax,bx
0700:0108 int 20h
0700:010A RET
```

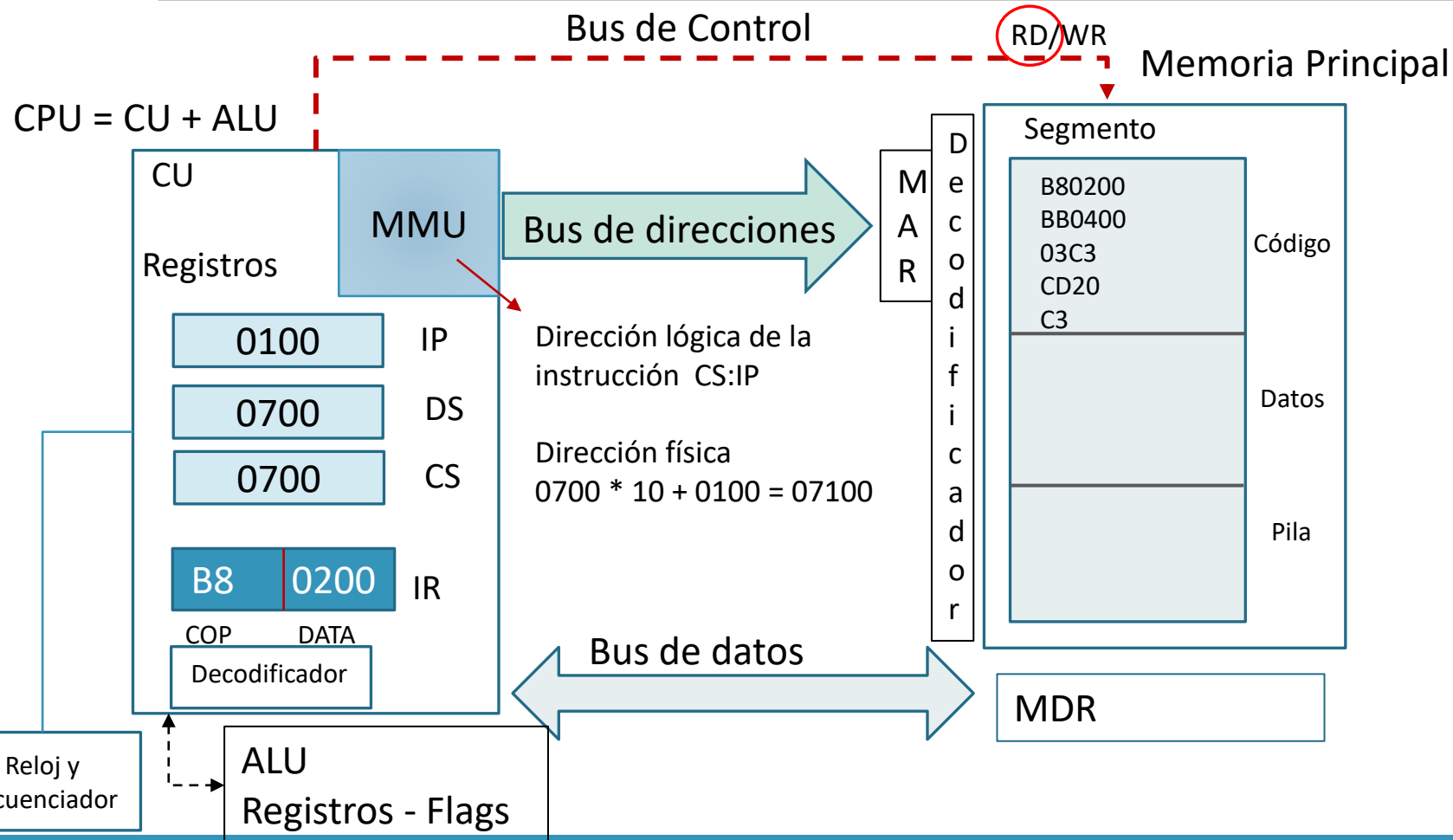
Ciclo de instrucción – Actividades previas



Actividades previas a la ejecución de la primer instrucción

```
0700:0100 mov ax,0002
0700:0103 mov bx,0004
0700:0106 add ax,bx
0700:0108 int 20h
0700:010A RET
```

Ciclo de instrucción – Fase búsqueda

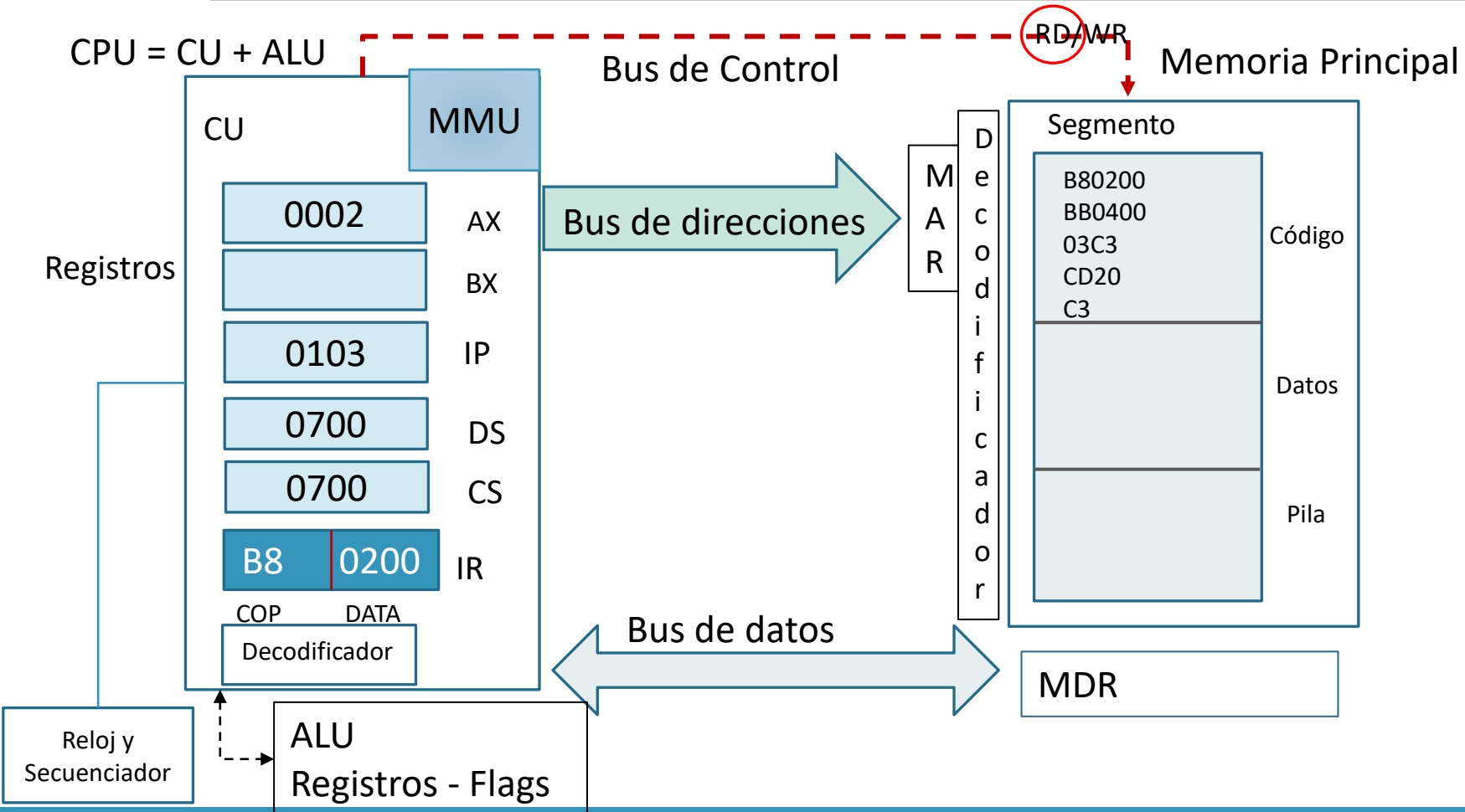


Fase de búsqueda:

- Cálculo de la dirección física de la instrucción.
- Dar orden de lectura RD
- Se carga el registro IR

```
0700:0100 mov ax,0002
0700:0103 mov bx,0004
0700:0106 add ax,bx
0700:0108 int 20h
0700:010A RET
```

Ciclo de instrucción – Fase ejecución

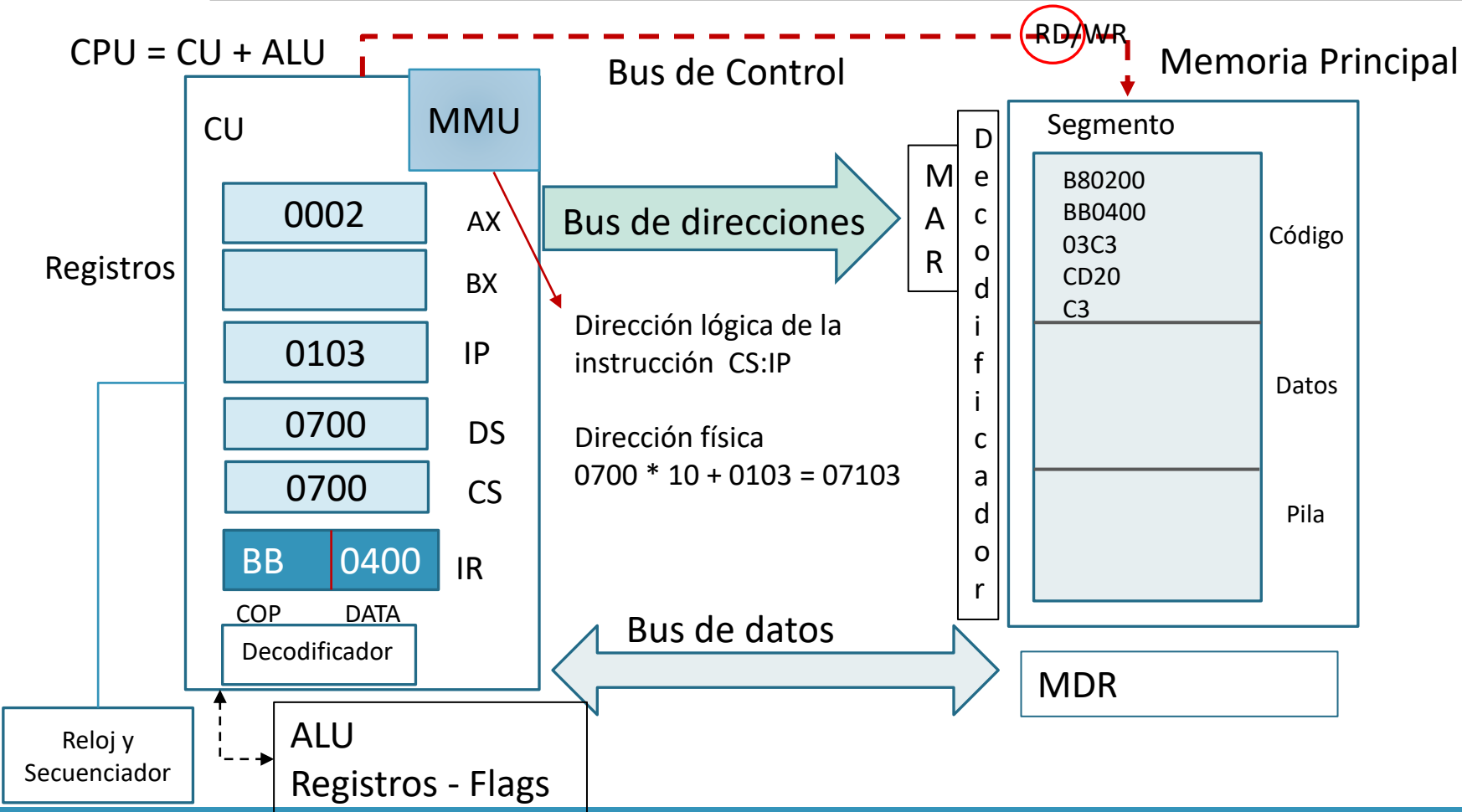


Fase de ejecución:

- Interpretar el código de la instrucción
- Incrementar IP
- Búsqueda del dato o Guarda el dato (si afecta)
- Generar orden al modulo para que opere el dato.

```
0700:0100 mov ax,0002
0700:0103 mov bx,0004
0700:0106 add ax,bx
0700:0108 int 20h
0700:010A RET
```

Ciclo de instrucción – Fase búsqueda



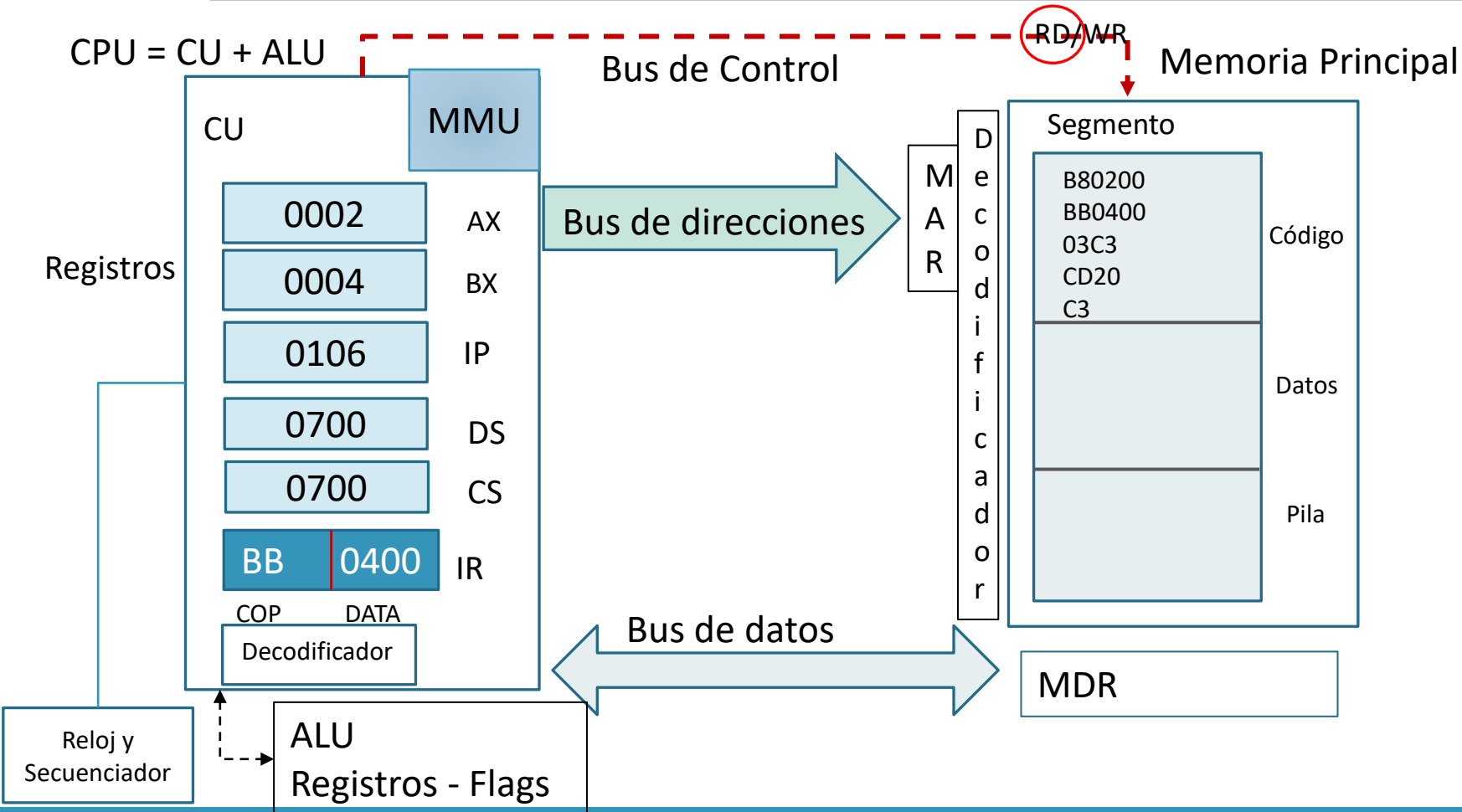
Fase de búsqueda:

- Calculo de la dirección física de la instrucción.
- Dar orden de lectura RD
- Se carga el registro IR

```

0700:0100 mov ax,0002
0700:0103 mov bx,0004
0700:0106 add ax,bx
0700:0108 int 20h
0700:010A RET
  
```

Ciclo de instrucción – Fase ejecución

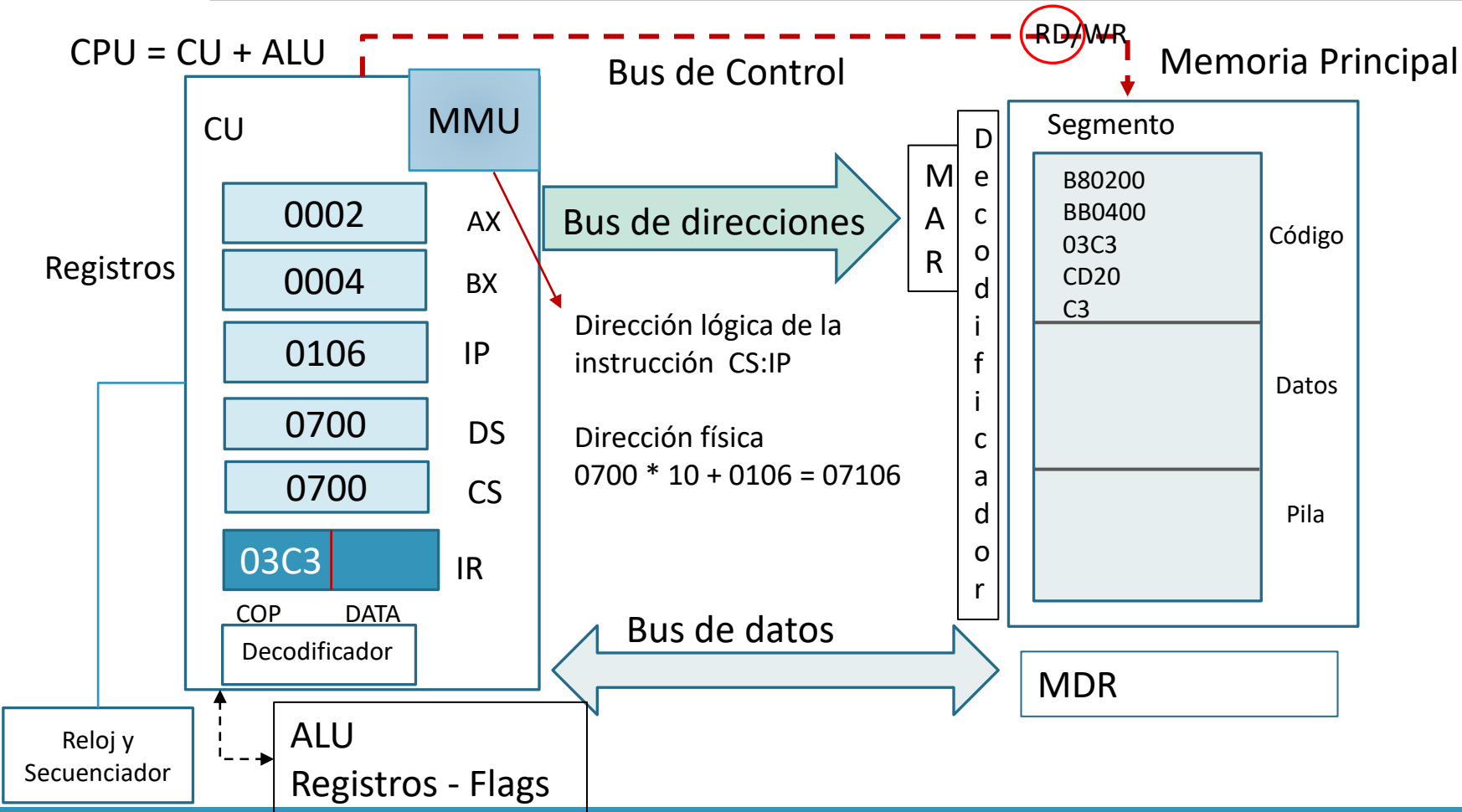


Fase de ejecución:

- Interpretar el código de la instrucción
- Incrementar IP
- Búsqueda del dato o Guarda el dato (si afecta)
- Generar orden al modulo para que opere el dato.

```
0700:0100 mov ax,0002
0700:0103 mov bx,0004
0700:0106 add ax,bx
0700:0108 int 20h
0700:010A RET
```

Ciclo de instrucción – Fase búsqueda



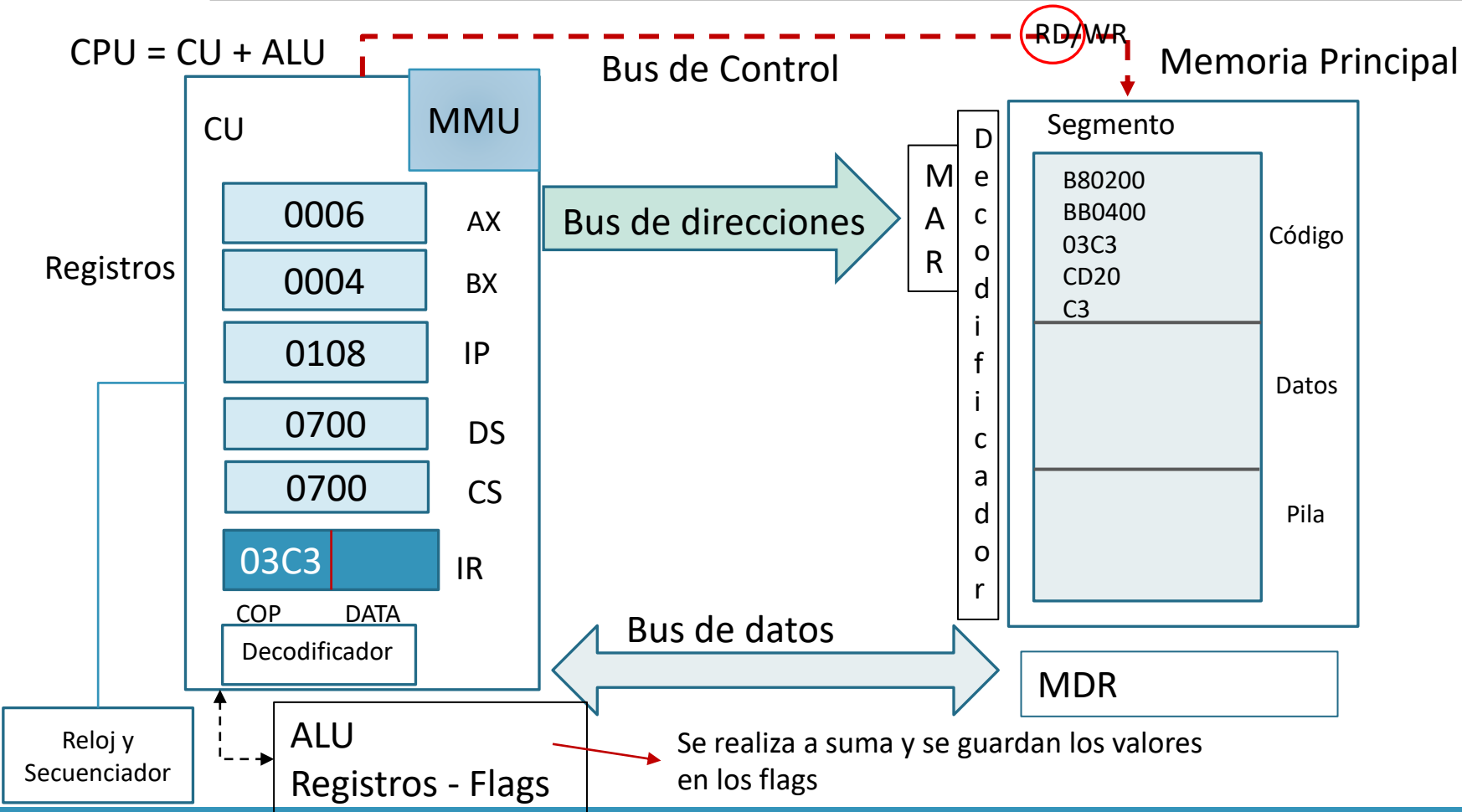
Fase de búsqueda:

- Calculo de la dirección física de la instrucción.
- Dar orden de lectura RD
- Se carga el registro IR

```

0700:0100 mov ax,0002
0700:0103 mov bx,0004
0700:0106 add ax,bx
0700:0108 int 20h
0700:010A RET
  
```


Ciclo de instrucción – Fase ejecución

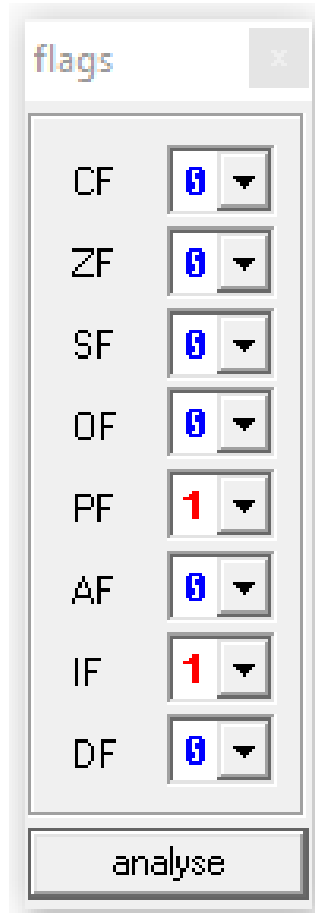


Fase de ejecución:

- Interpretar el código de la instrucción
- Incrementar IP
- Búsqueda del dato o Guarda el dato (si afecta)
- Generar orden al modulo para que opere el dato. (ALU)

```
0700:0100 mov ax,0002
0700:0103 mov bx,0004
0700:0106 add ax,bx
0700:0108 int 20h
0700:010A RET
```

Banderas y Registros



Overflow = OF
0 = no hay desbordamiento;
1 = sí lo hay

Dirección = DF
0 = hacia adelante;
1 = hacia atrás;

Interrupciones = IF
0 = desactivadas;
1 = activadas

Signo = SF
0 = positivo;
1 = negativo

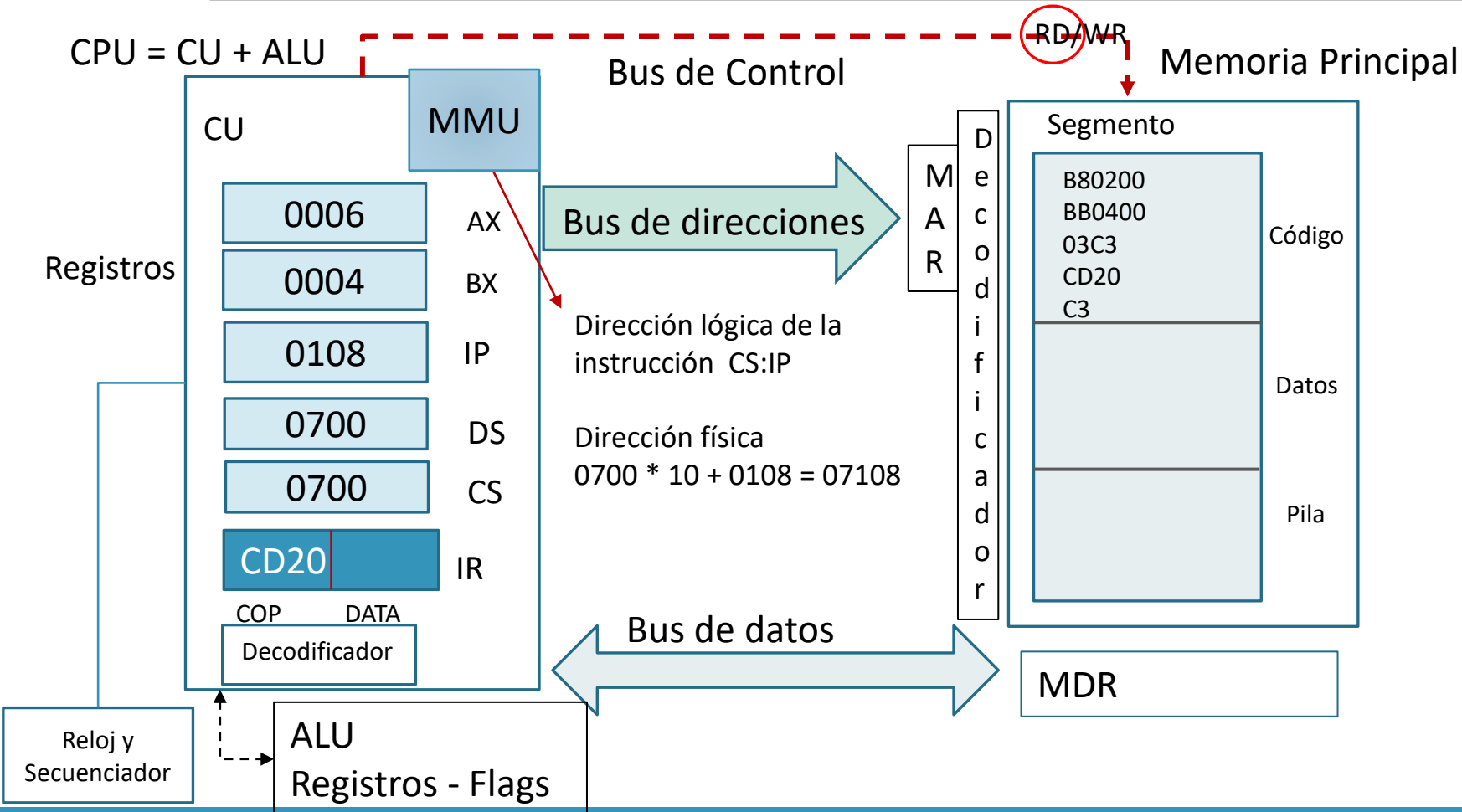
Cero = ZF
0 = no es cero;
1 = sí lo es

Auxiliary Carry = AF
0 = no hay acarreo
auxiliar;
1 = hay acarreo auxiliar

Parity = PF
0 = paridad non;
1 = paridad par;

Carry = CF
0 = no hay acarreo;
1 = Sí lo hay

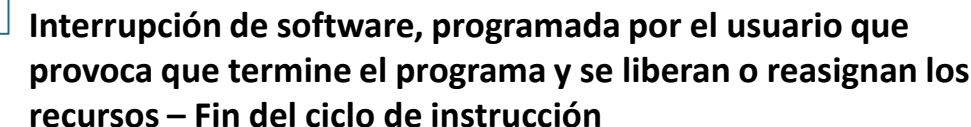
Ciclo de instrucción – Fase búsqueda



Fase de búsqueda:

- Calculo de la dirección física de la instrucción.
- Dar orden de lectura RD
- Se carga el registro IR

```
0700:0100 mov ax,0002
0700:0103 mov bx,0004
0700:0106 add ax,bx
0700:0108 int 20h
0700:010A RET
```



Emu8086

```
mov ah,[0300h]
```

```
add ah,[0301h]
```

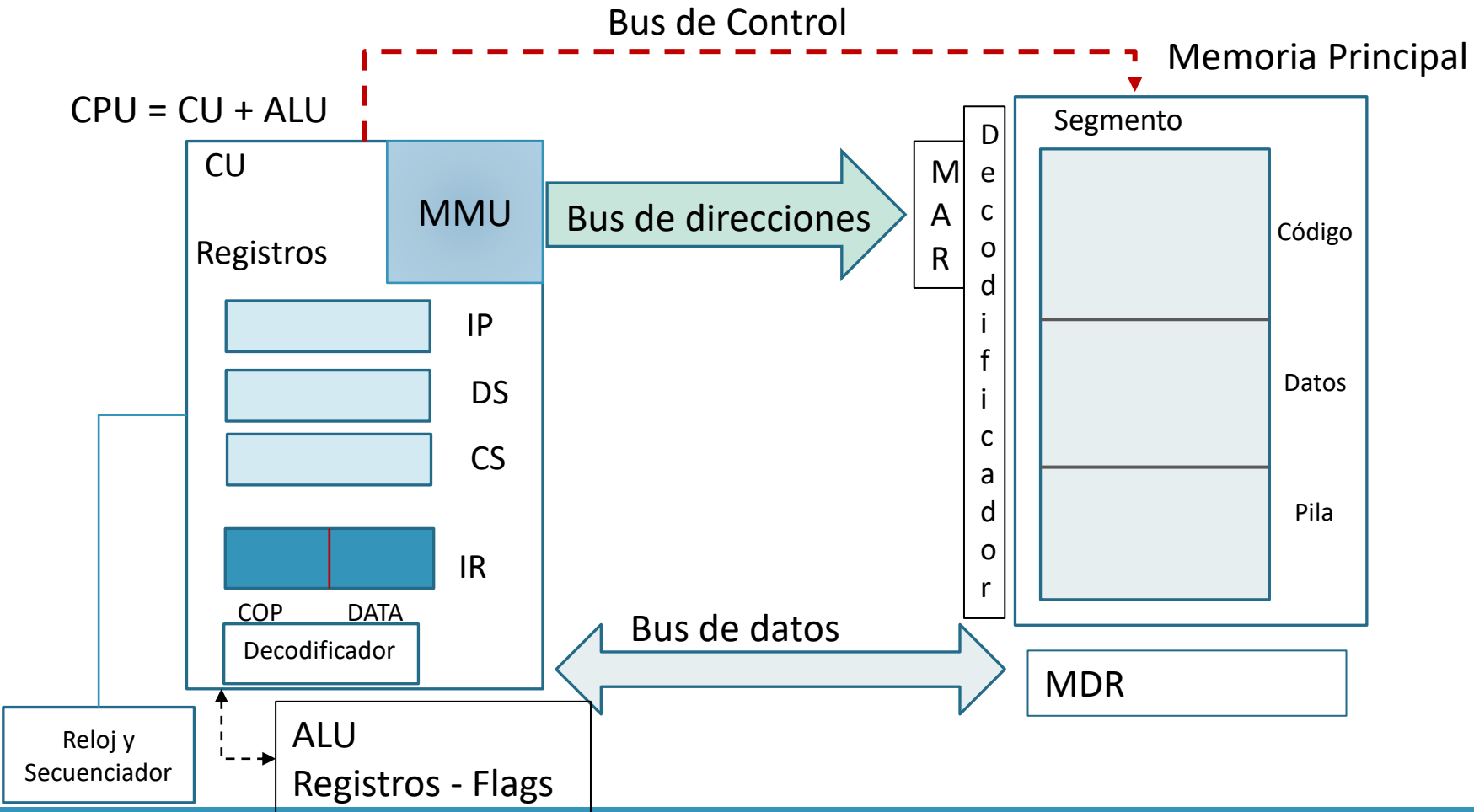
```
mov [0400h], ah
```

```
ret
```

0700:0300			
07300:	01	001	☺
07301:	FF	255	RES
07302:	00	000	NULL

0700:0400			
07400:	00	000	NULL

CPU – Memoria Principal – Modo real



```
0700:0100 mov ah,[0300]
0700:0104 add ah,[0301]
0700:0108 mov [0400], ah
0700:010C ret
```

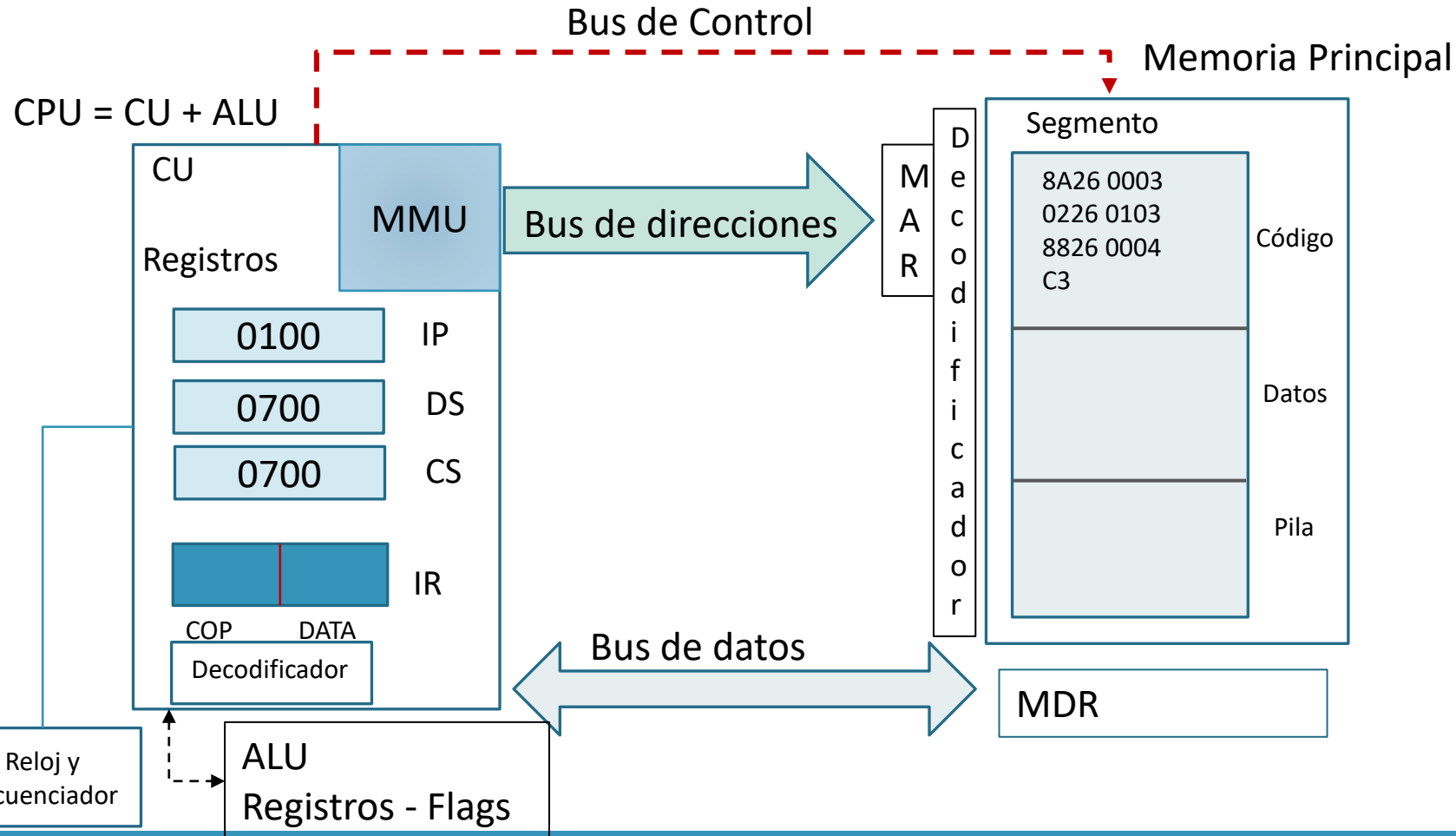
0700:0300			
07300:	01	001	☺
07301:	FF	255	RES
07302:	00	000	NULL

```
0700:0300 00.01
0700:0301 00.ff
```

0700:0400			
07400:	00	000	NULL

```
0700:0400 00.
```

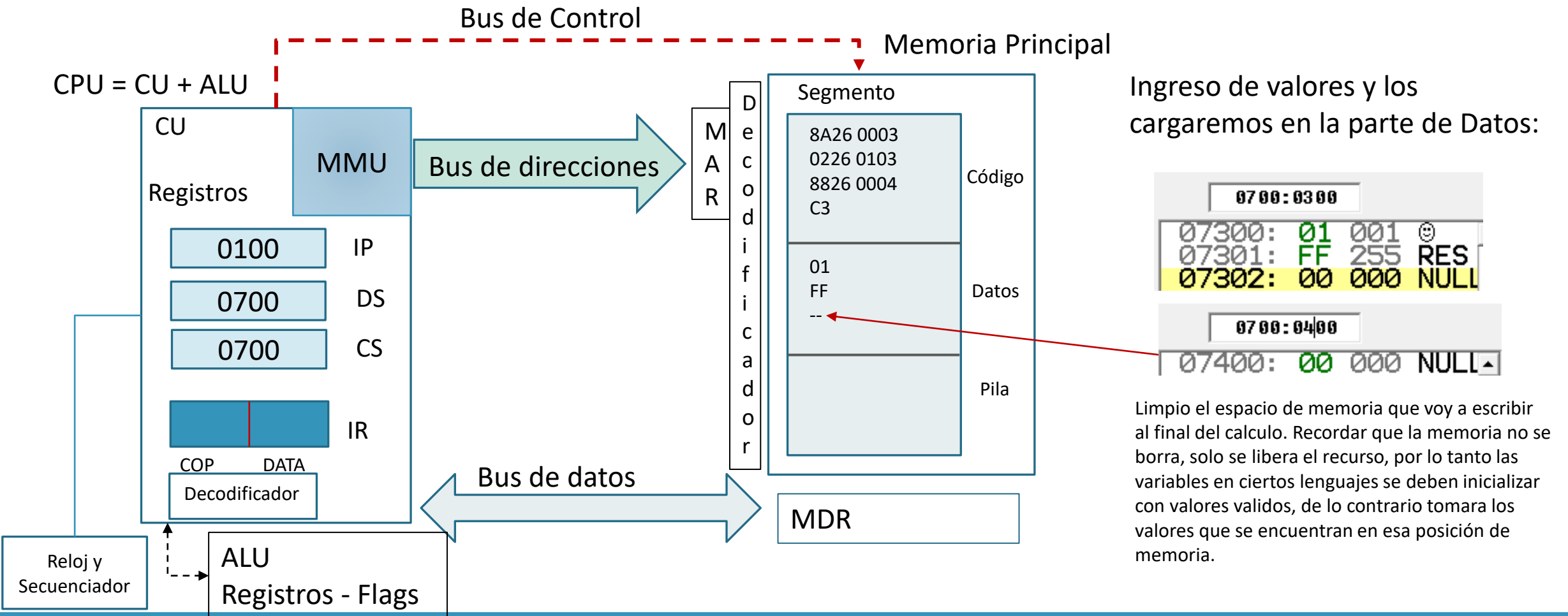
Ciclo de instrucción – Actividades previas



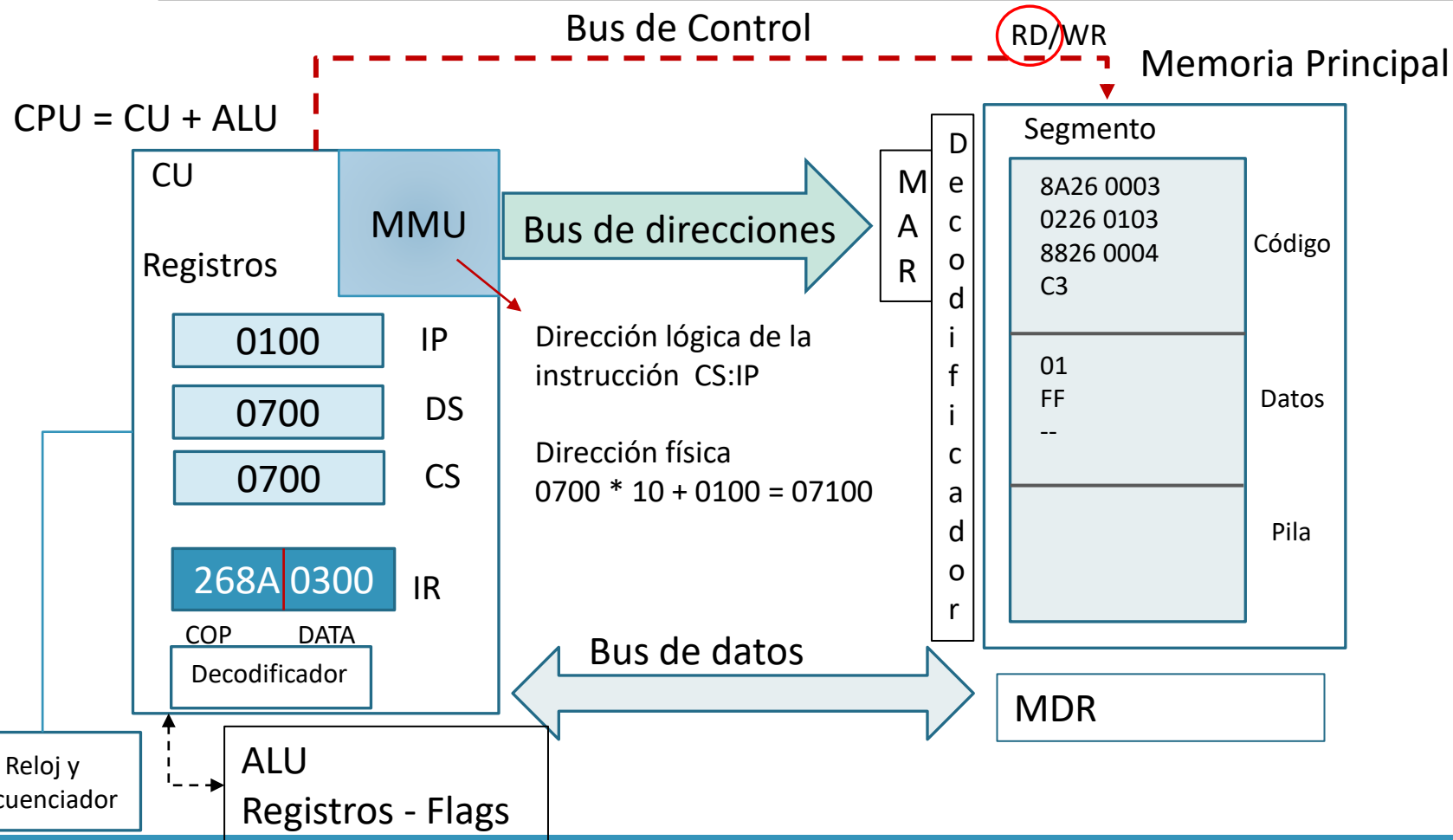
Actividades previas a la ejecución de la primer instrucción

```
0700:0100 mov ah,[0300]
0700:0104 add ah,[0301]
0700:0108 mov [0400], ah
0700:010C ret
```

Ciclo de instrucción – Actividades previas



Ciclo de instrucción – Fase búsqueda

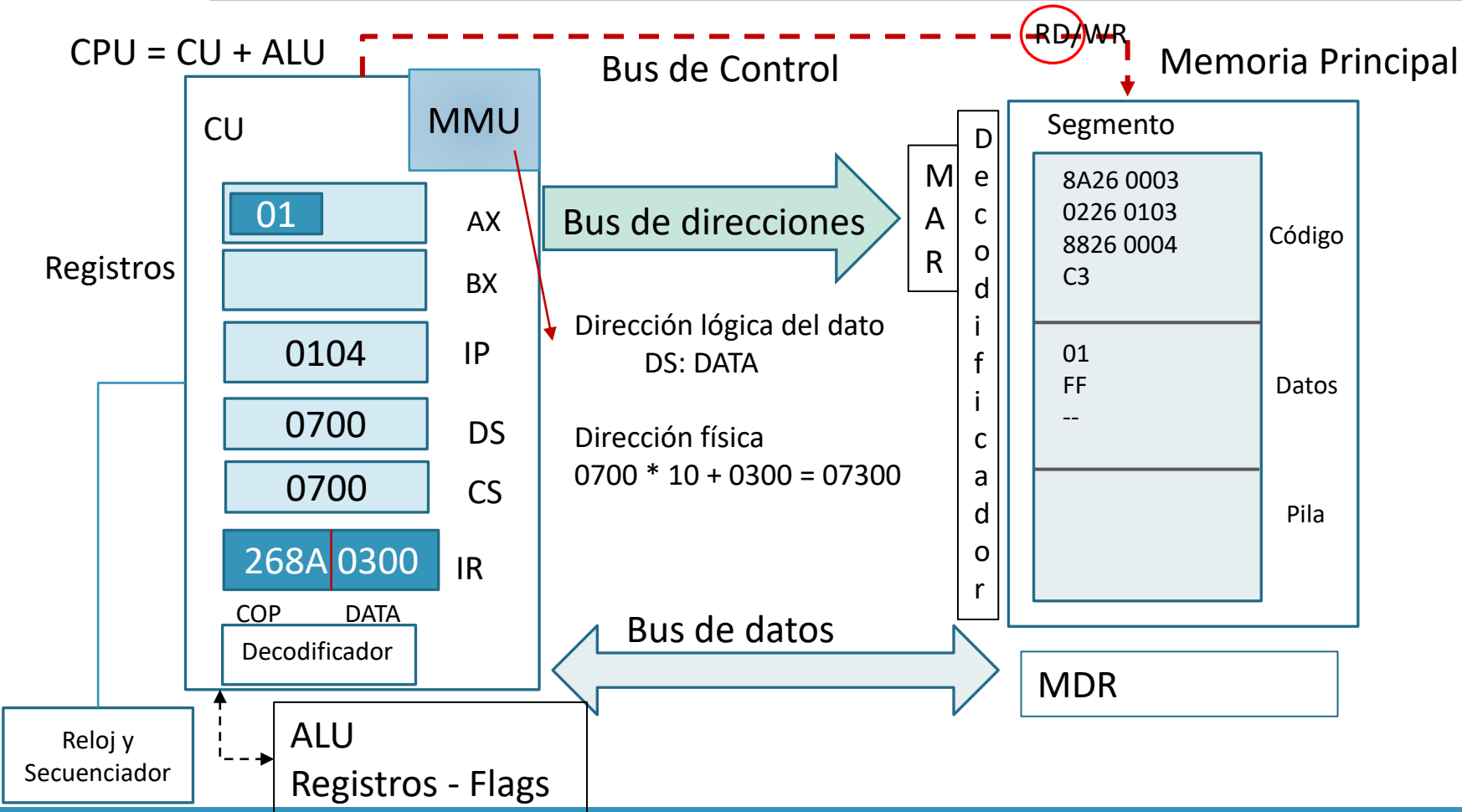


Fase de búsqueda:

- Calculo de la dirección física de la instrucción.
- Dar orden de lectura RD
- Se carga el registro IR

```
0700:0100 mov ah,[0300]
0700:0104 add ah,[0301]
0700:0108 mov [0400], ah
0700:010C ret
```

Ciclo de instrucción – Fase ejecución

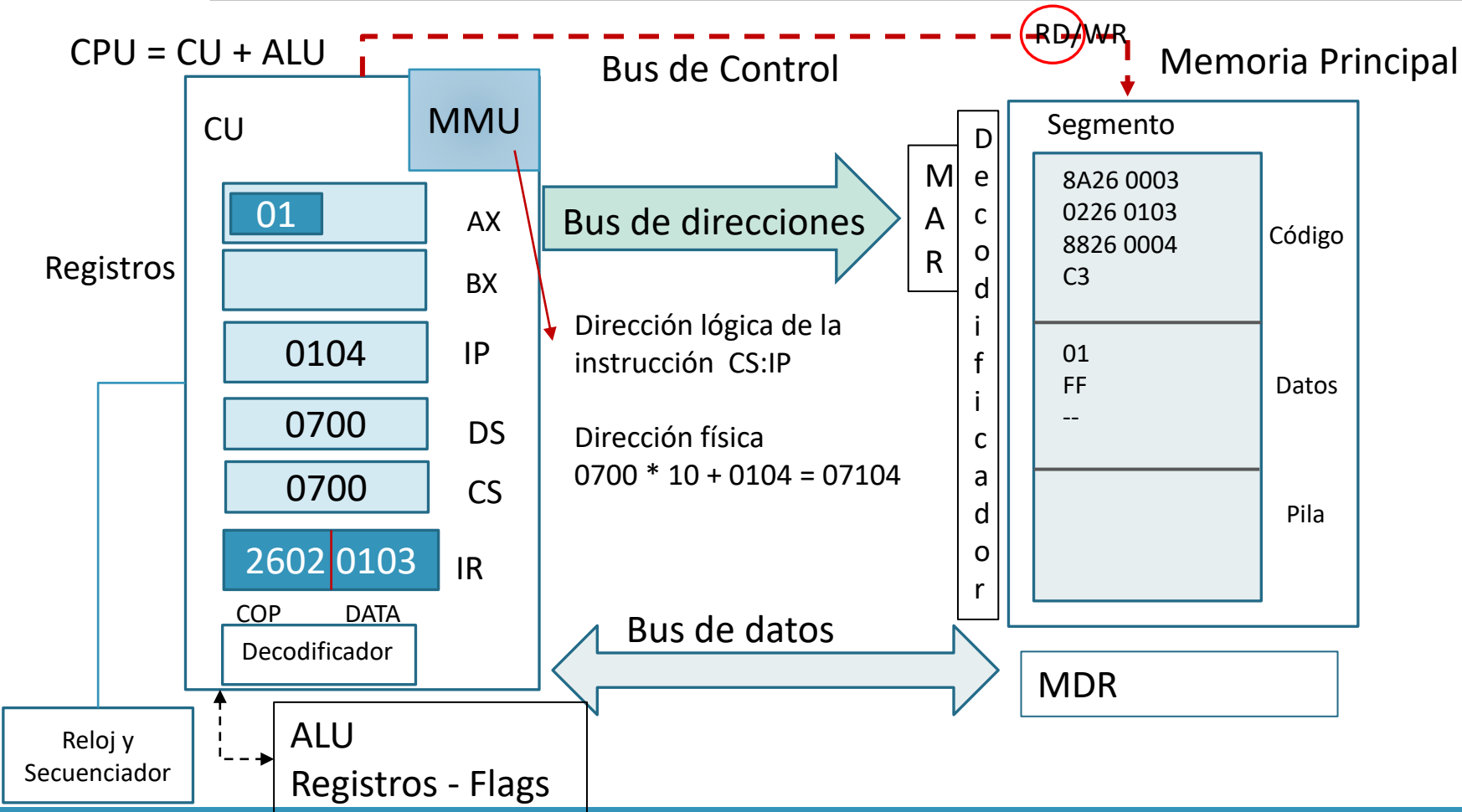


Fase de ejecución:

- Interpretar el código de la instrucción
- Incrementar IP
- **Búsqueda del dato** o Guarda el dato (si afecta)
- Generar orden al modulo para que opere el dato.

```
0700:0100 mov ah,[0300]
0700:0104 add ah,[0301]
0700:0108 mov [0400], ah
0700:010C ret
```

Ciclo de instrucción – Fase búsqueda

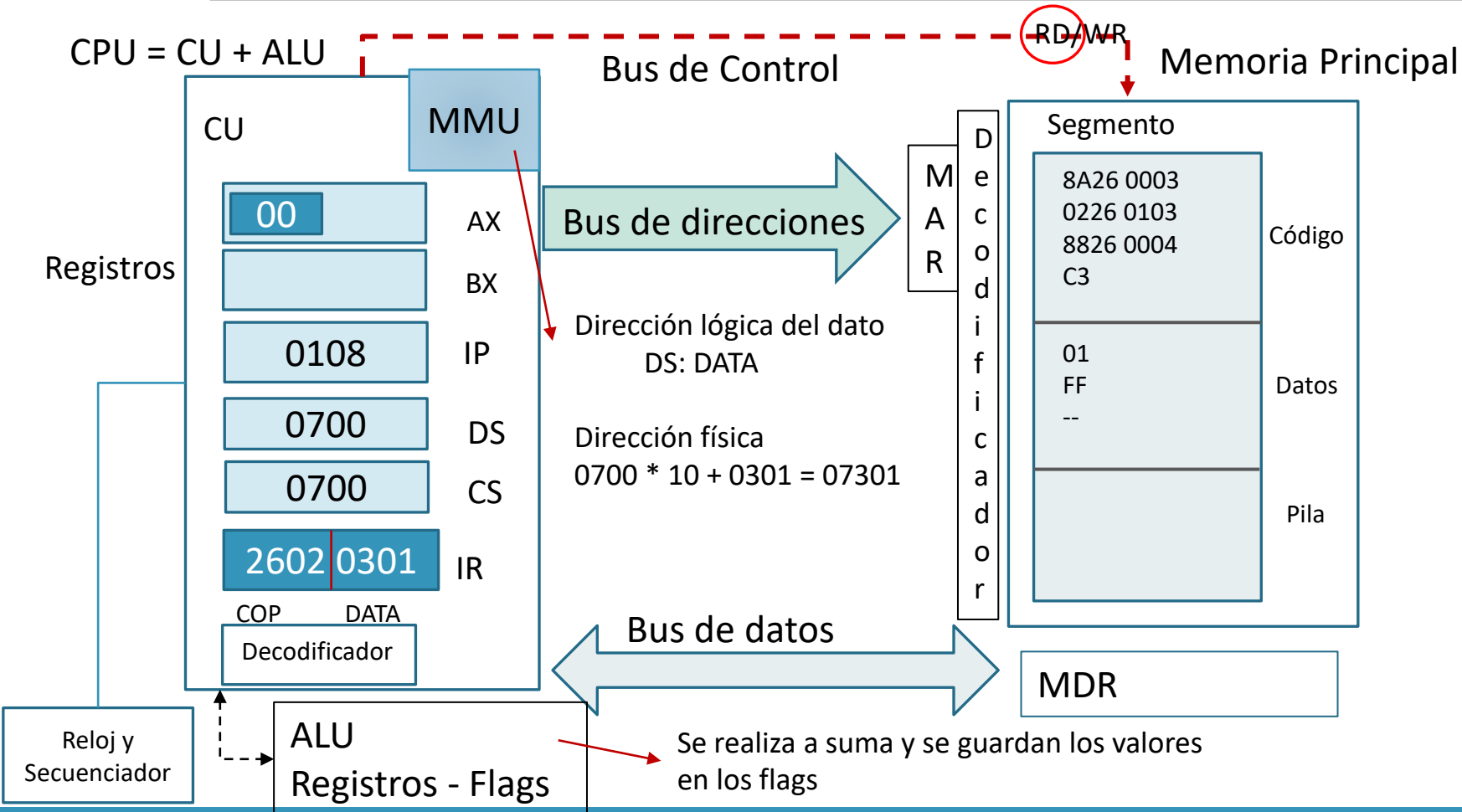


Fase de búsqueda:

- Calculo de la dirección física de la instrucción.
- Dar orden de lectura RD
- Se carga el registro IR

```
0700:0100 mov ah,[0300]
0700:0104 add ah,[0301]
0700:0108 mov [0400], ah
0700:010C ret
```

Ciclo de instrucción – Fase ejecución



Fase de ejecución:

- Interpretar el código de la instrucción
- Incrementar IP
- **Búsqueda del dato** o Guarda el dato (si afecta)
- **Generar orden al modulo para que opere el dato.**

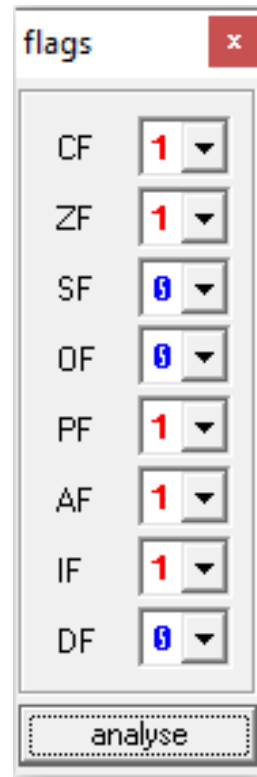
```
0700:0100 mov ah,[0300]
0700:0104 add ah,[0301]
0700:0108 mov [0400], ah
0700:010C ret
```

Banderas y Registros

Suma:

```

11111111
00000001 -> 1(10)
11111111 -> -1(10)
-----
00000000
  
```



Overflow = OF

0 = no hay desbordamiento;
1 = sí lo hay

Dirección = DF

0 = hacia adelante;
1 = hacia atrás;

Interrupciones = IF

0 = desactivadas;
1 = activadas

Signo = SF

0 = positivo;
1 = negativo

Cero = ZF

0 = no es cero;
1 = sí lo es

Auxiliary Carry = AF

0 = no hay acarreo auxiliar;
1 = hay acarreo auxiliar

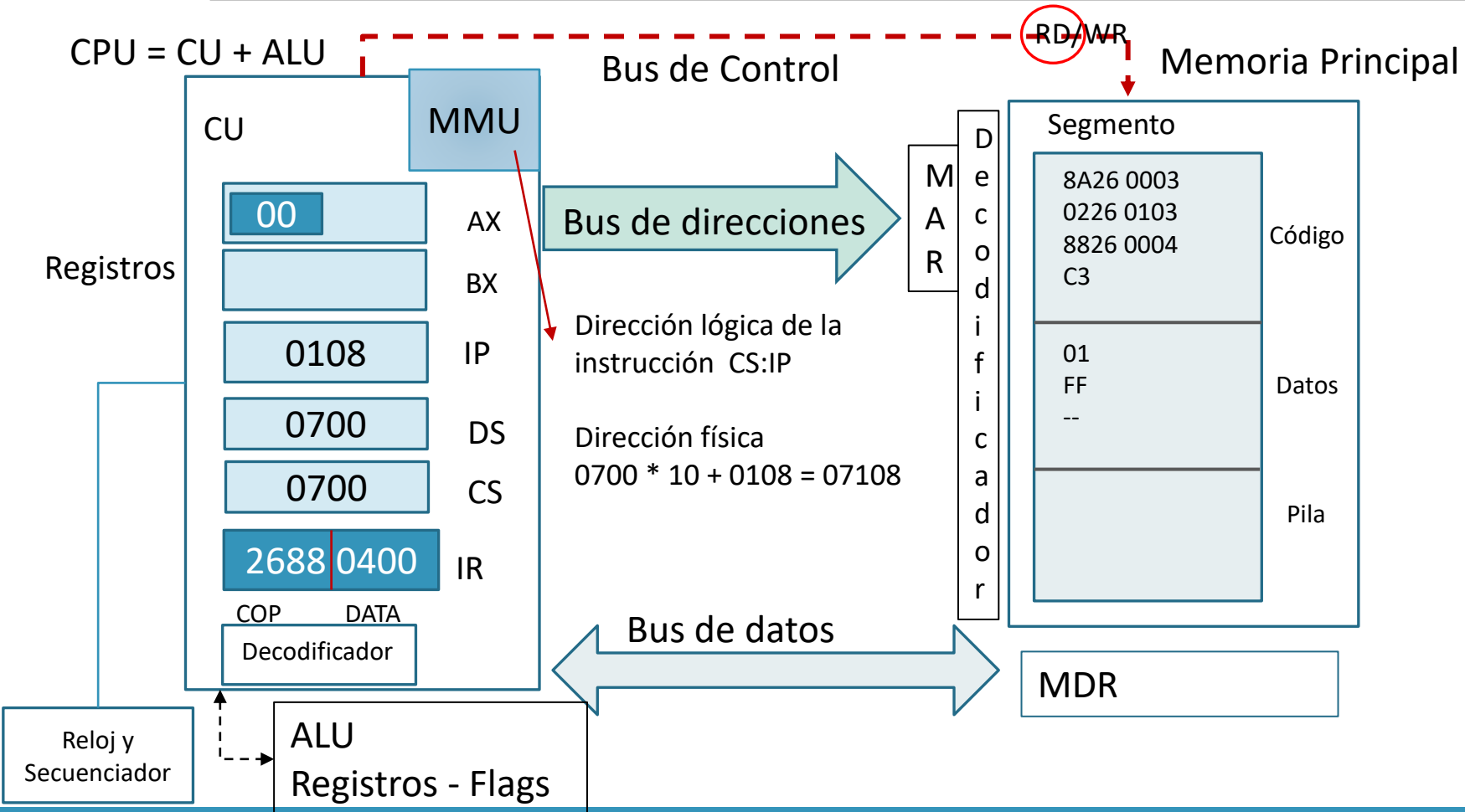
Parity = PF

0 = paridad non;
1 = paridad par;

Carry = CF

0 = no hay acarreo;
1 = Sí lo hay

Ciclo de instrucción – Fase búsqueda

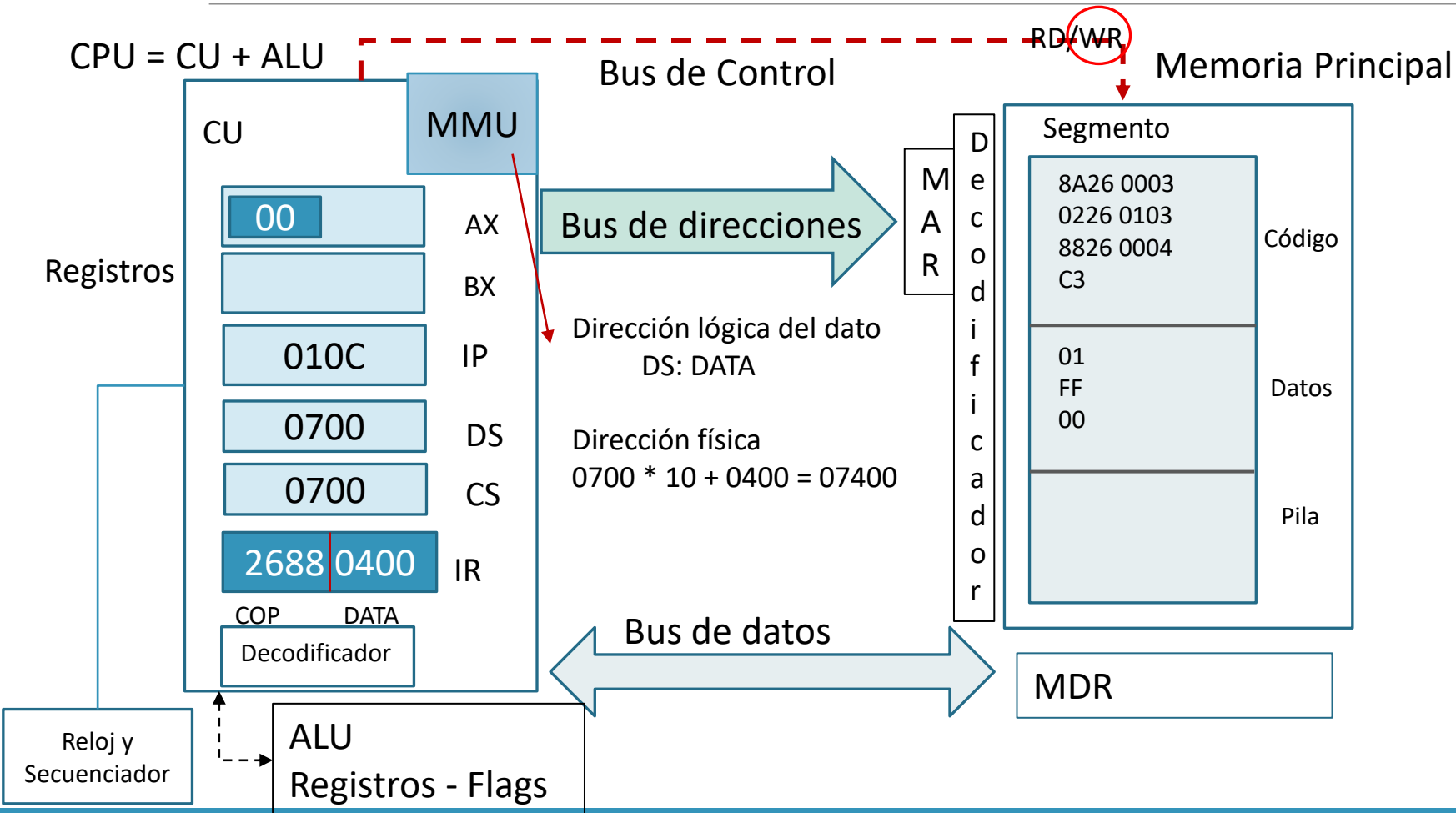


Fase de búsqueda:

- Calculo de la dirección física de la instrucción.
- Dar orden de lectura RD
- Se carga el registro IR

```
0700:0100 mov ah,[0300]
0700:0104 add ah,[0301]
0700:0108 mov [0400], ah
0700:010C ret
```

Ciclo de instrucción – Fase ejecución

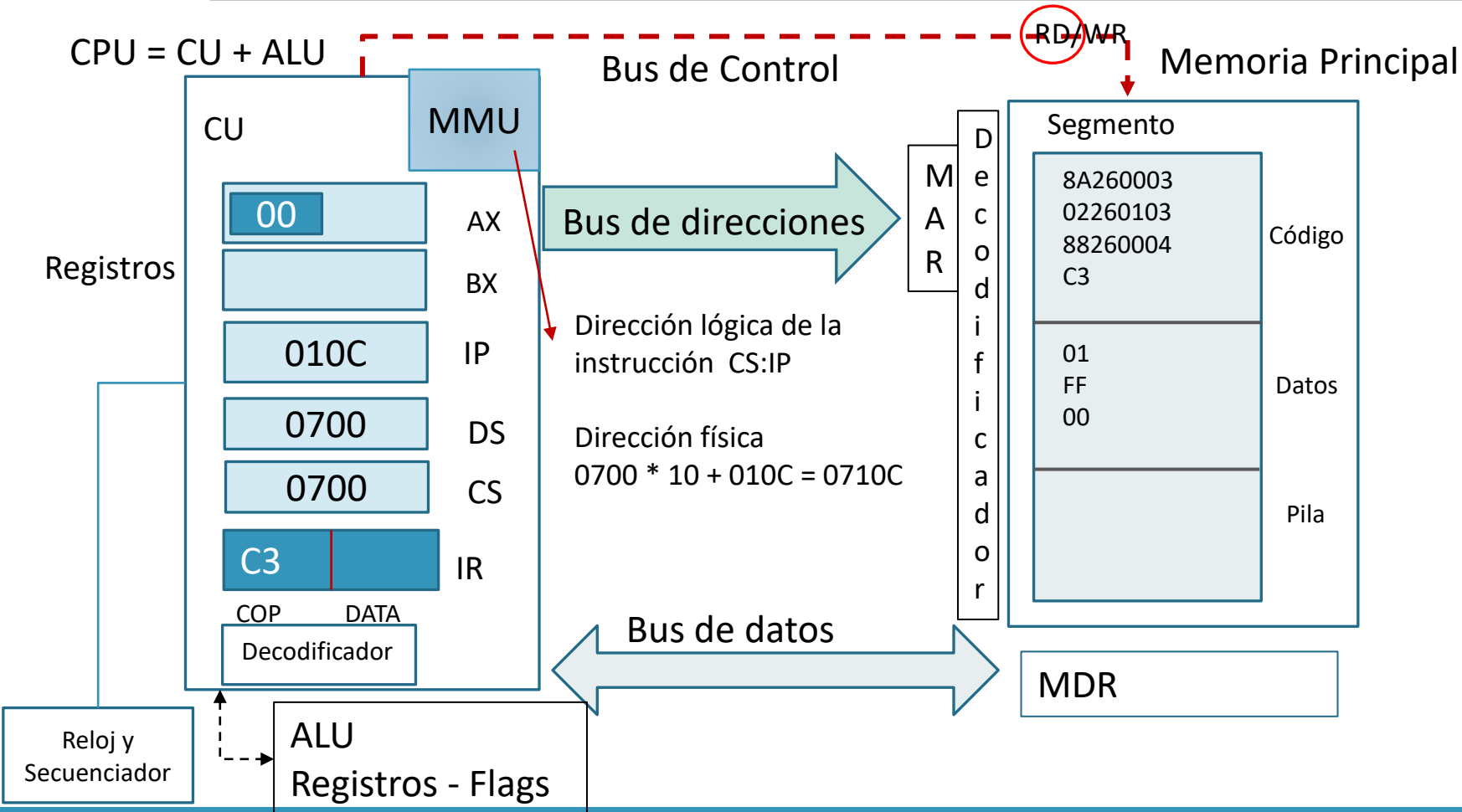


Fase de ejecución:

- Interpretar el código de la instrucción
- Incrementar IP
- Búsqueda del dato o **Guarda el dato**
- Generar orden al modulo para que opere el dato.

```
0700:0100 mov ah,[0300]
0700:0104 add ah,[0301]
0700:0108 mov [0400], ah
0700:010C ret
```

Ciclo de instrucción – Fase búsqueda

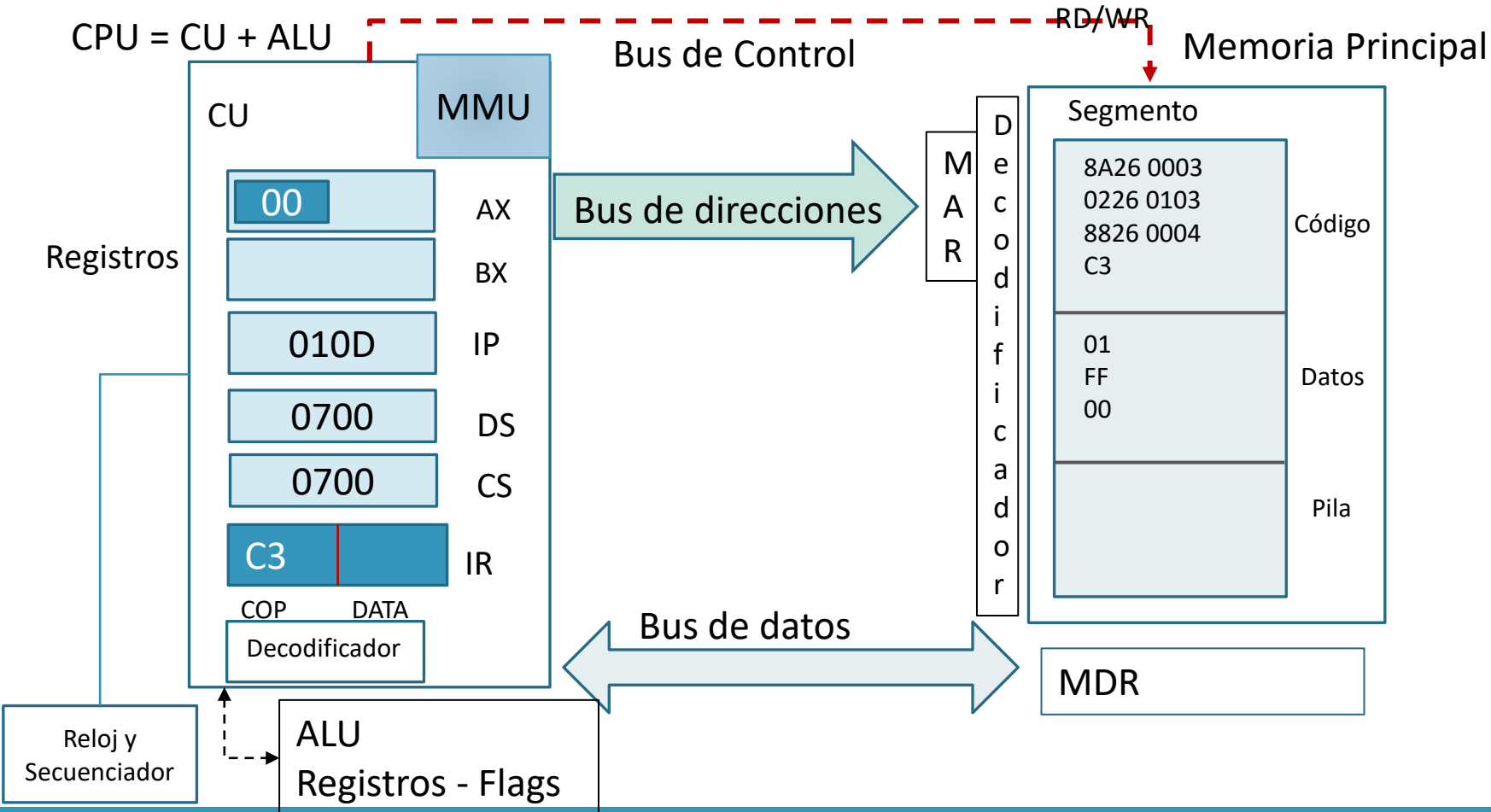


Fase de búsqueda:

- Calculo de la dirección física de la instrucción.
- Dar orden de lectura RD
- Se carga el registro IR

```
0700:0100 mov ah,[0300]
0700:0104 add ah,[0301]
0700:0108 mov [0400], ah
0700:010C ret
```


Ciclo de instrucción – Fase ejecución



Fase de ejecución:

- Interpretar el código de la instrucción
- Incrementar IP
- Búsqueda del dato o Guarda el dato
- Generar orden al modulo para que opere el dato.

```
0700:0100 mov ah,[0300]
0700:0104 add ah,[0301]
0700:0108 mov [0400], ah
0700:010C ret
```

Interrupción de software, programada por el usuario que provoca que termine el programa y se liberan o reasignan los recursos – Fin del ciclo de instrucción

DIRECCIONAMIENTO

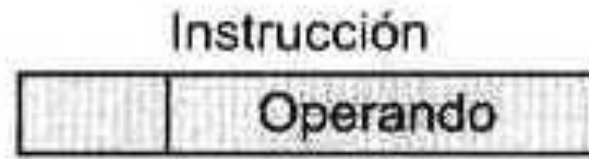
Direccionamiento

El campo o campos de direcciones en un formato de instrucción usual está bastante limitado.

Para poder subsanar este problema existen diversas técnicas de direccionamiento:

- Inmediato
- Directo
- Indirecto
- Registro
- Indirecto con registro
- Con desplazamiento
- Pila

Direccionamiento Inmediato



(a) Inmediato

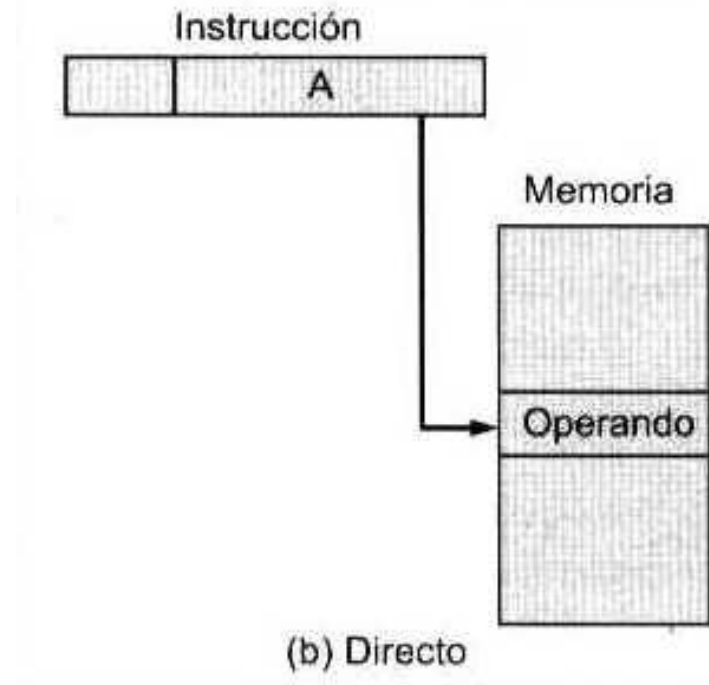
El operando se encuentra en la propia instrucción:

OPERANDO=A

Este modo puede utilizarse para definir y utilizar constantes, o para fijar valores iniciales de variables.

La ventaja del direccionamiento inmediato es que, una vez captada la instrucción, no se requiere una referencia a memoria para obtener el operando, ahorrándose un ciclo de memoria.

Direccionamiento Directo

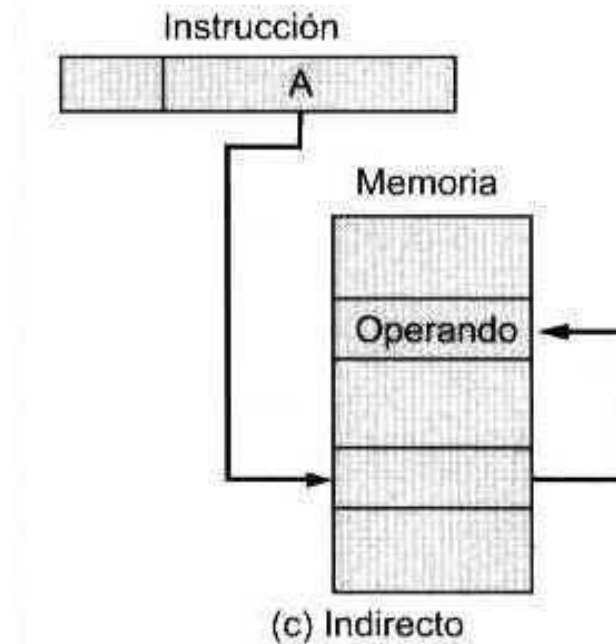


En el campo de direcciones contiene la dirección efectiva del operando.

$$EA=A$$

EA=Dirección real (efectiva) de la posición que contiene el operando que se referencia.

Direccionamiento Indirecto

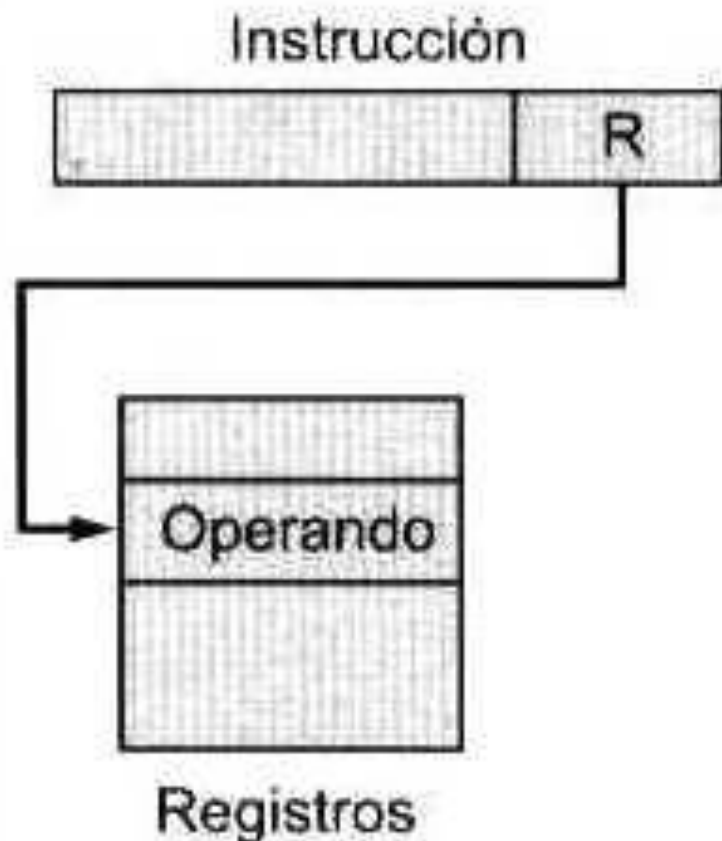


El problema del direccionamiento directo es que está limitado el rango de direcciones por la longitud de la palabra.

Una solución es hacer que el campo de direcciones referencie la dirección de una palabra de memoria que contenga la dirección completa del operando.

$$EA = (A) \quad (X) = \text{Contenido de la posición } X$$

Direccionamiento de Registros



(d) Registro

El direccionamiento de registros es similar al directo.

La única diferencia es que el campo de direcciones referencia a un registro, en lugar de una dirección de memoria principal:

$$EA = R$$

Las ventajas del direccionamiento de registros son que sólo es necesario un campo pequeño de direcciones en la instrucción ya que una referencia a registros consta de 3 o 4 bits, y no se requieren referencias a memoria.

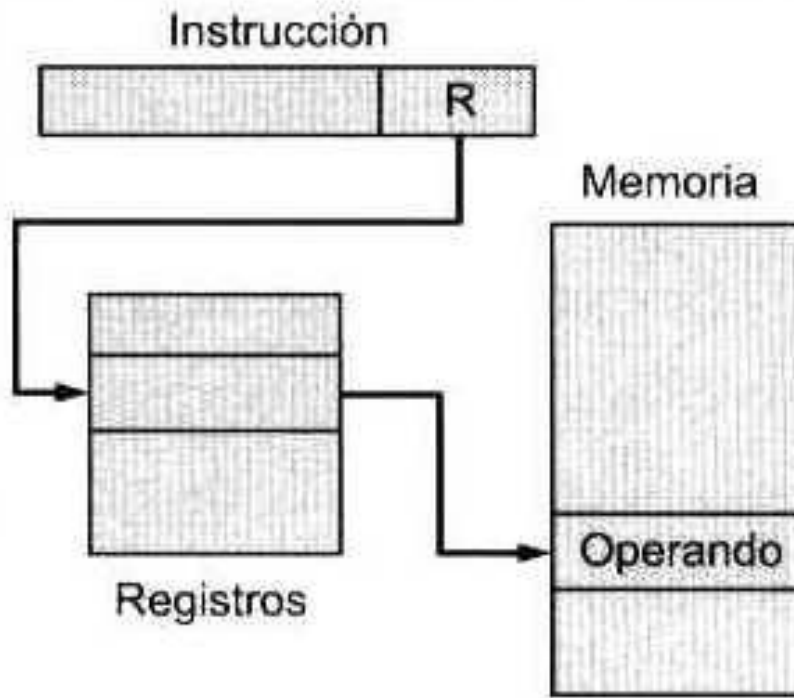
La desventaja es que el espacio de direcciones está muy limitado.

Direccionamiento Indirecto con Registro

La diferencia está en si el campo de direcciones hace referencia a una posición de memoria o a un registro

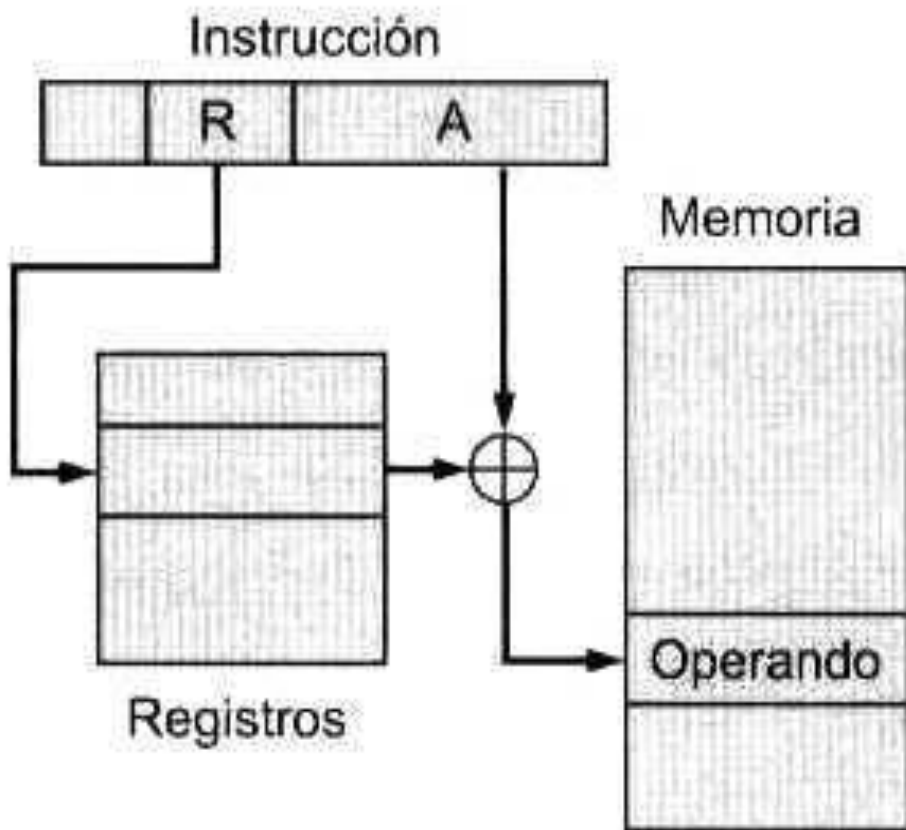
$$EA = (R)$$

El direccionamiento indirecto con registro emplea una referencia menos a memoria que el direccionamiento indirecto



(e) Indirecto con registro

Direccionamiento con desplazamiento



(f) Con desplazamiento

Un modo muy potente que combina las posibilidades de los direccionamientos directo, e indirecto con registro.

$$EA = A + (R)$$

Requiere que las instrucciones tengan dos campos de direcciones, al menos uno de ellos explícito.

Usos más comunes:

- Desplazamiento relativo
- Direccionamiento con registro base
- Indexado

Direccionamiento con desplazamiento

- Desplazamiento relativo

El registro referenciado implícitamente es el contador de programa (PC). Es decir, la dirección de instrucción actual se suma al campo de direcciones para producir el valor EA

- Direccionamiento con registro base

El registro referenciado contiene una dirección de memoria, y el campo de dirección contiene un desplazamiento desde dicha dirección. Es la forma mas común de implementar la segmentación.

- Indexado

El campo de dirección referencia una dirección de memoria principal, y el registro referenciado contiene un desplazamiento positivo desde esa dirección. Este uso es justo el opuesto de la interpretación del direccionamiento con registro base.

Direccionamiento de pila

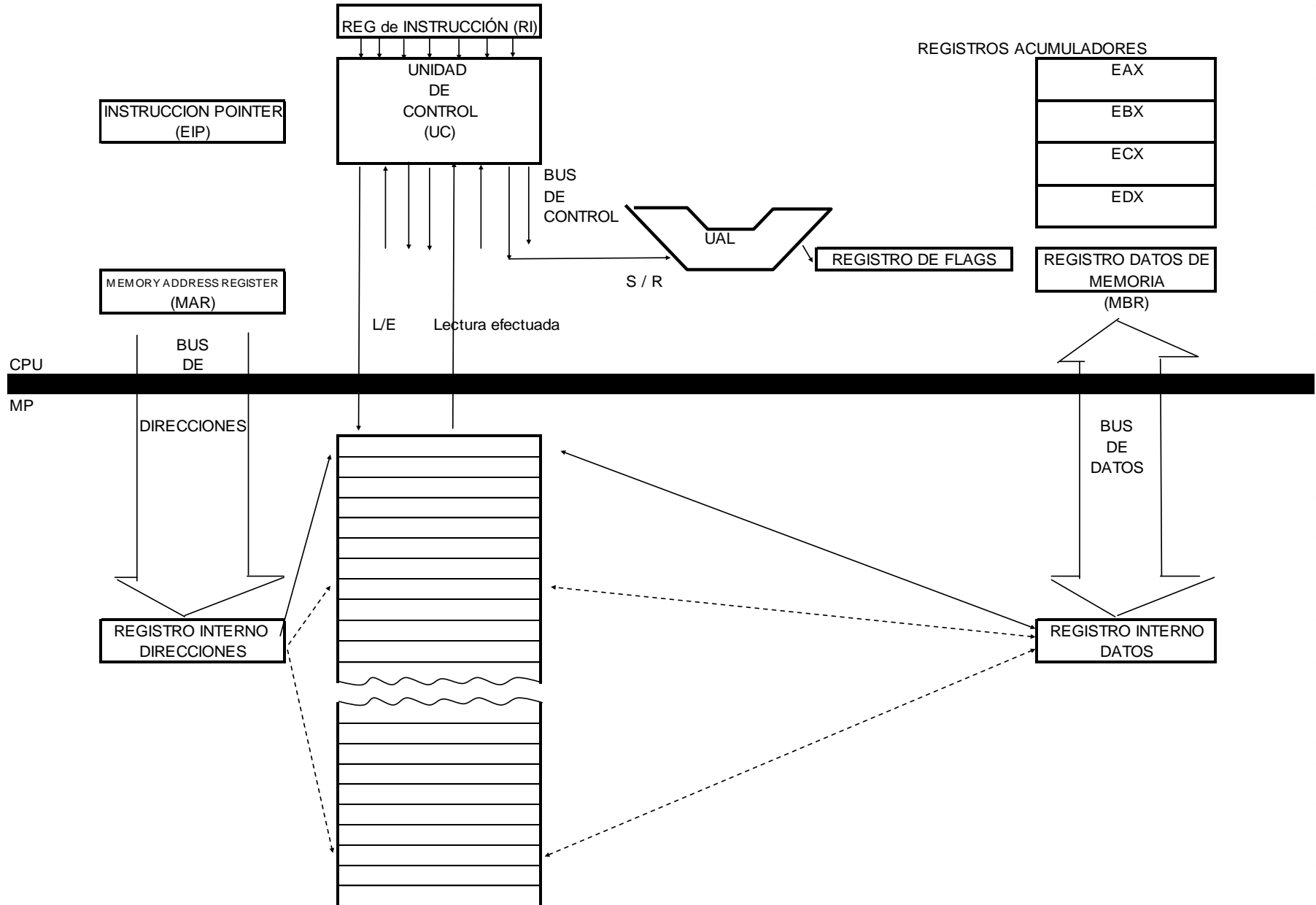
Una pila es un bloque de posiciones reservado. Los elementos se añaden en la cabecera de la pila de manera que , en cualquier instante, el bloque está parcialmente lleno. La pila tiene asociado un puntero, cuyo valor es la dirección de la cabecera o tope de la pila.

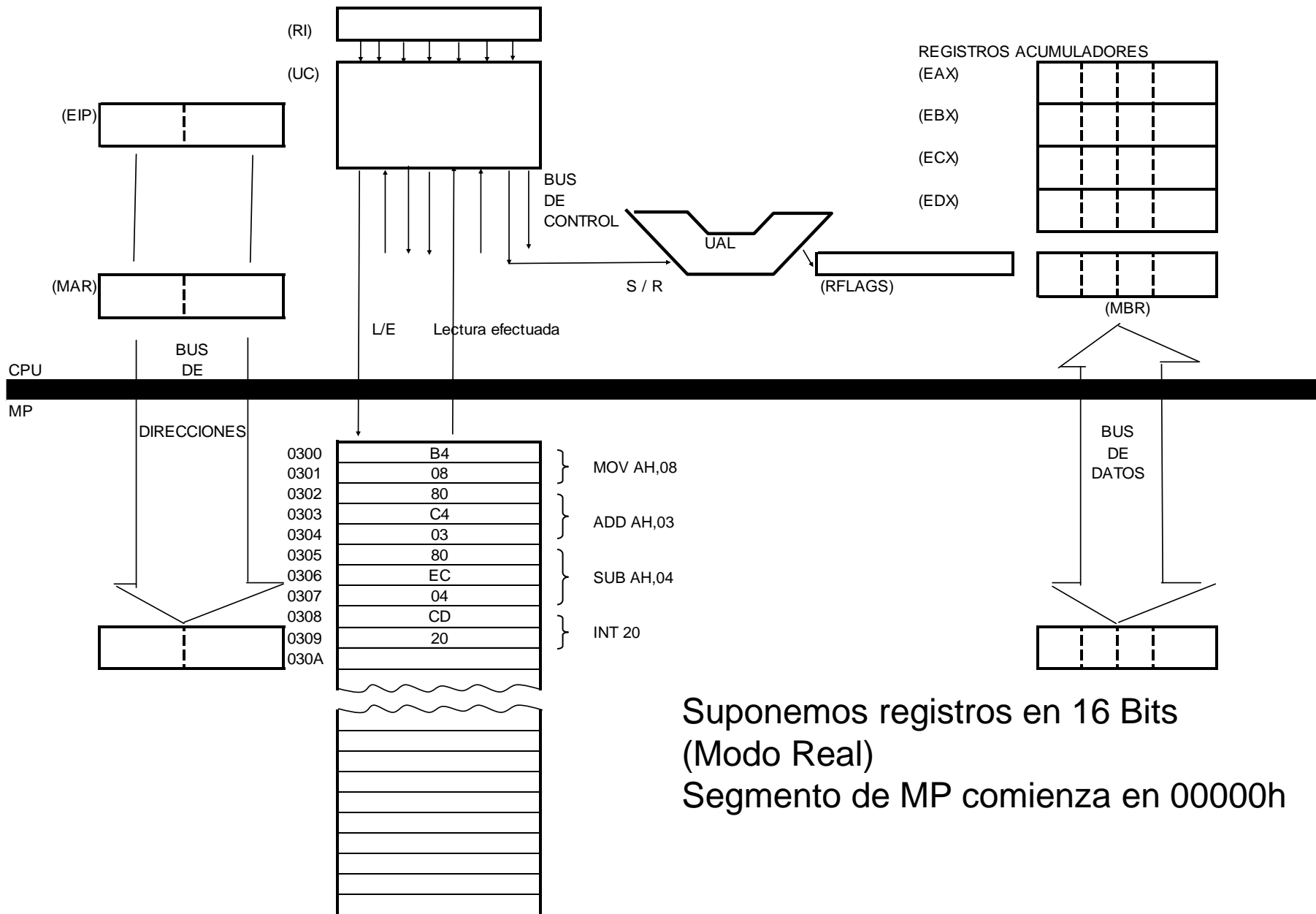


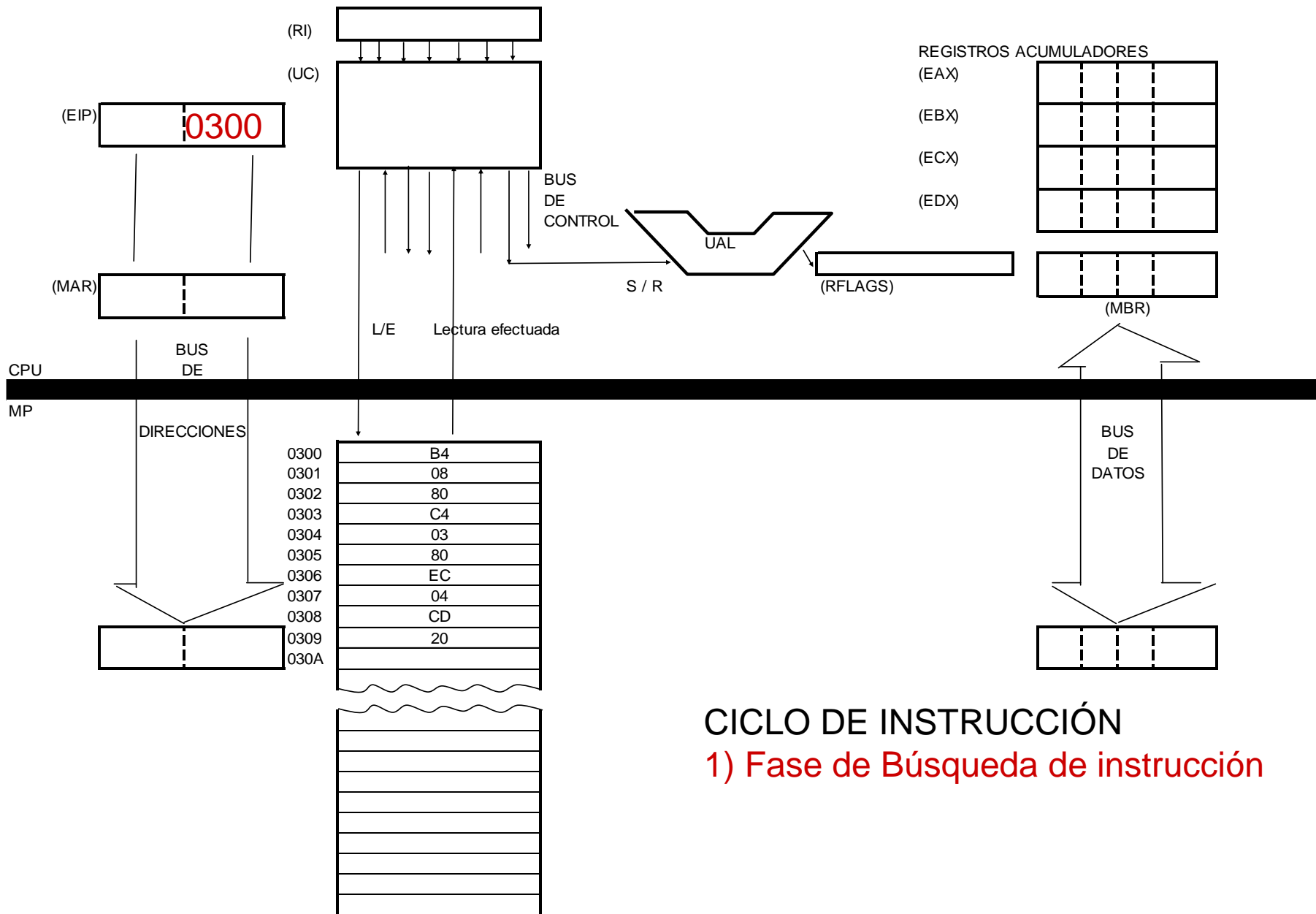
(g) Pila

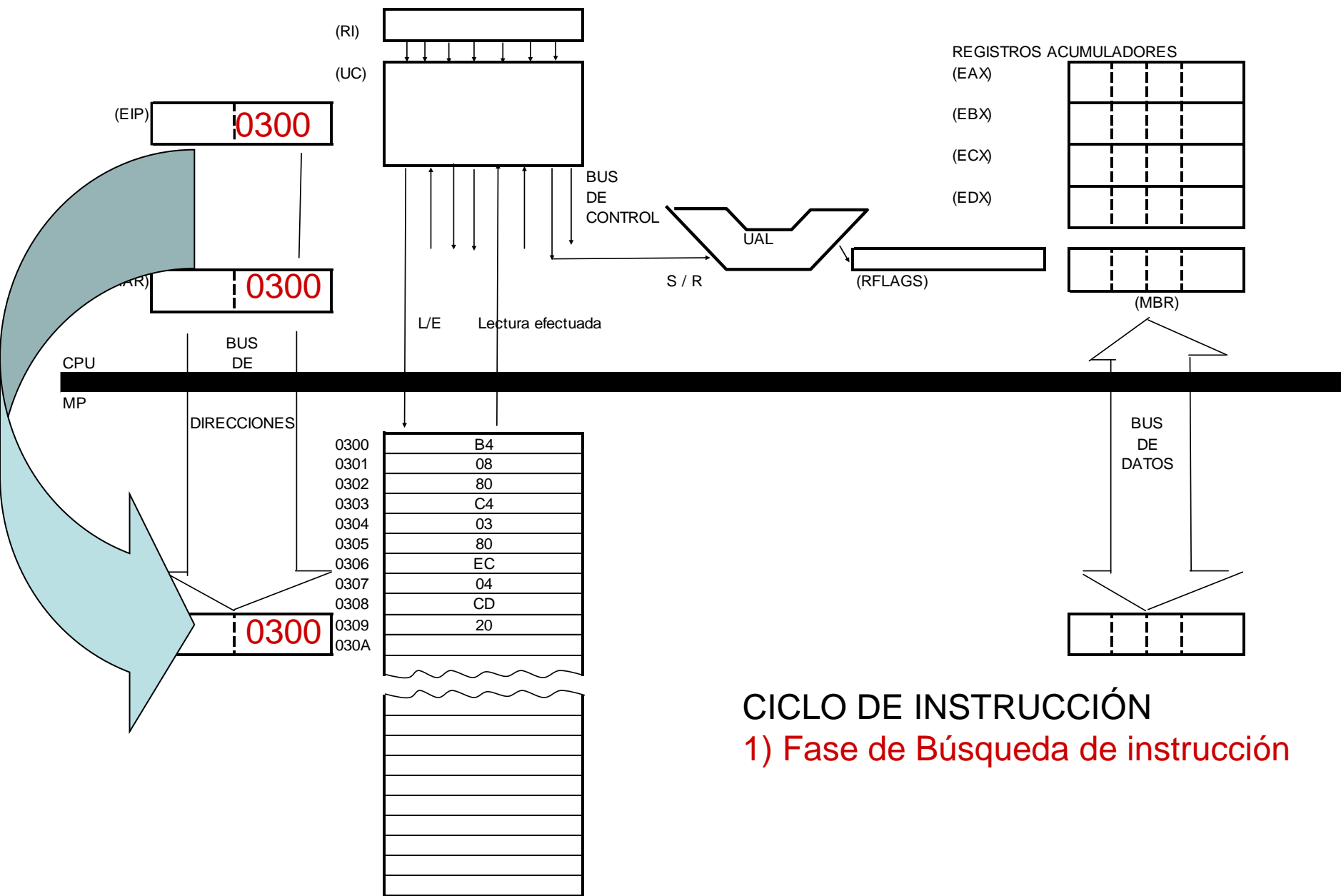
El modo de direccionamiento de pila es una forma de direccionamiento implícito. Las instrucciones operan implícitamente con la cabecera de la pila.

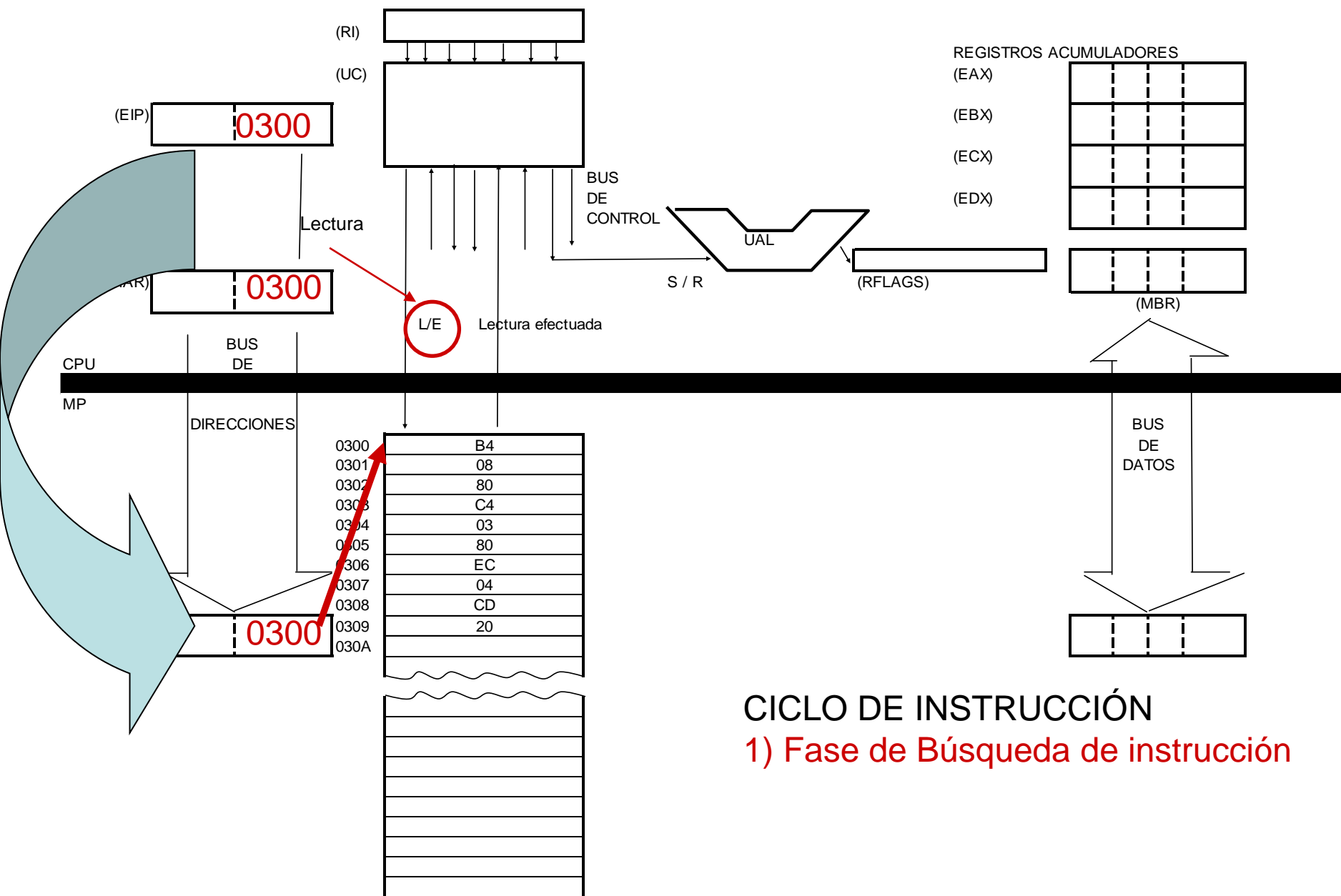
ESQUEMA GENERAL PARA ANALIZAR EL CICLO DE INSTRUCCION

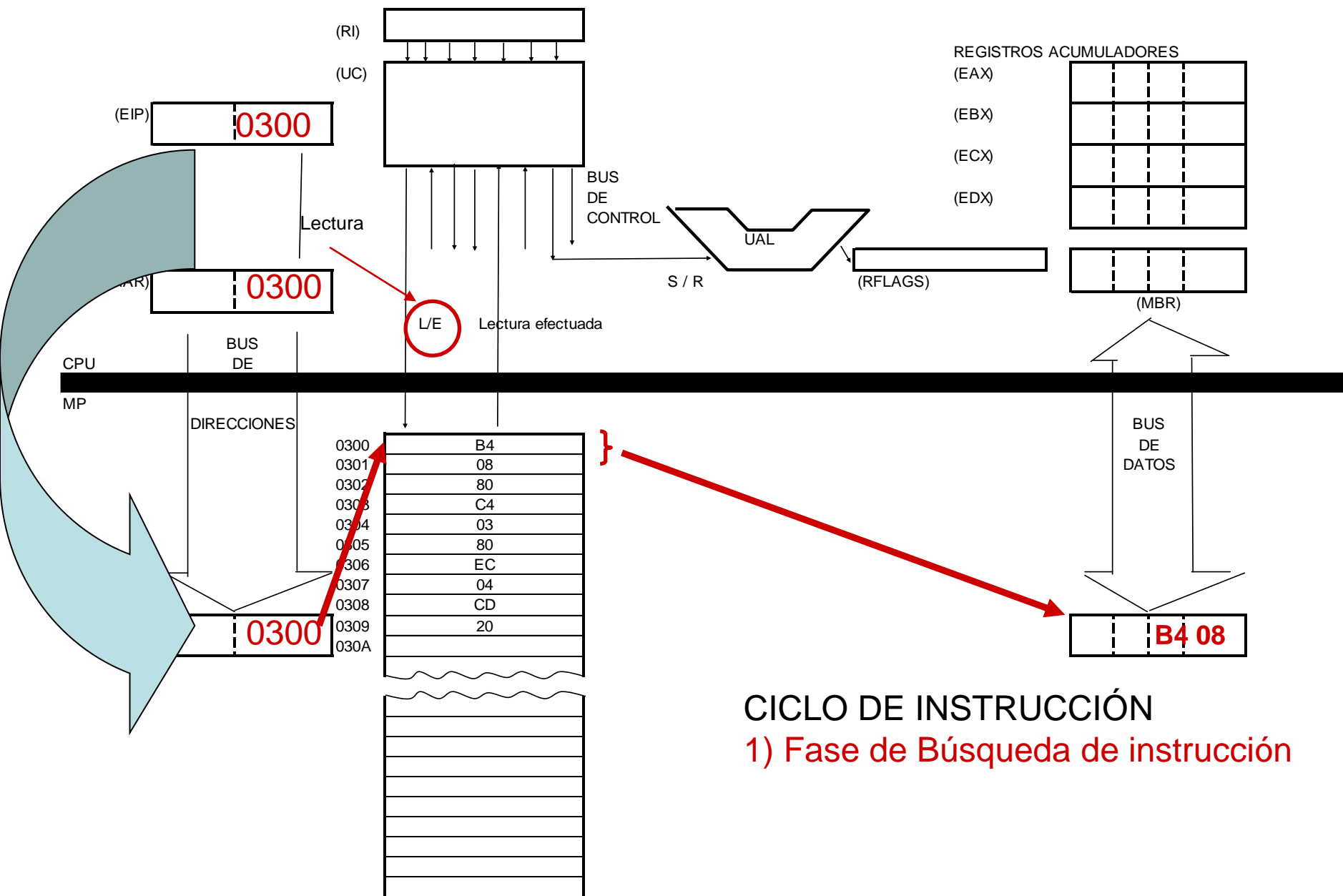




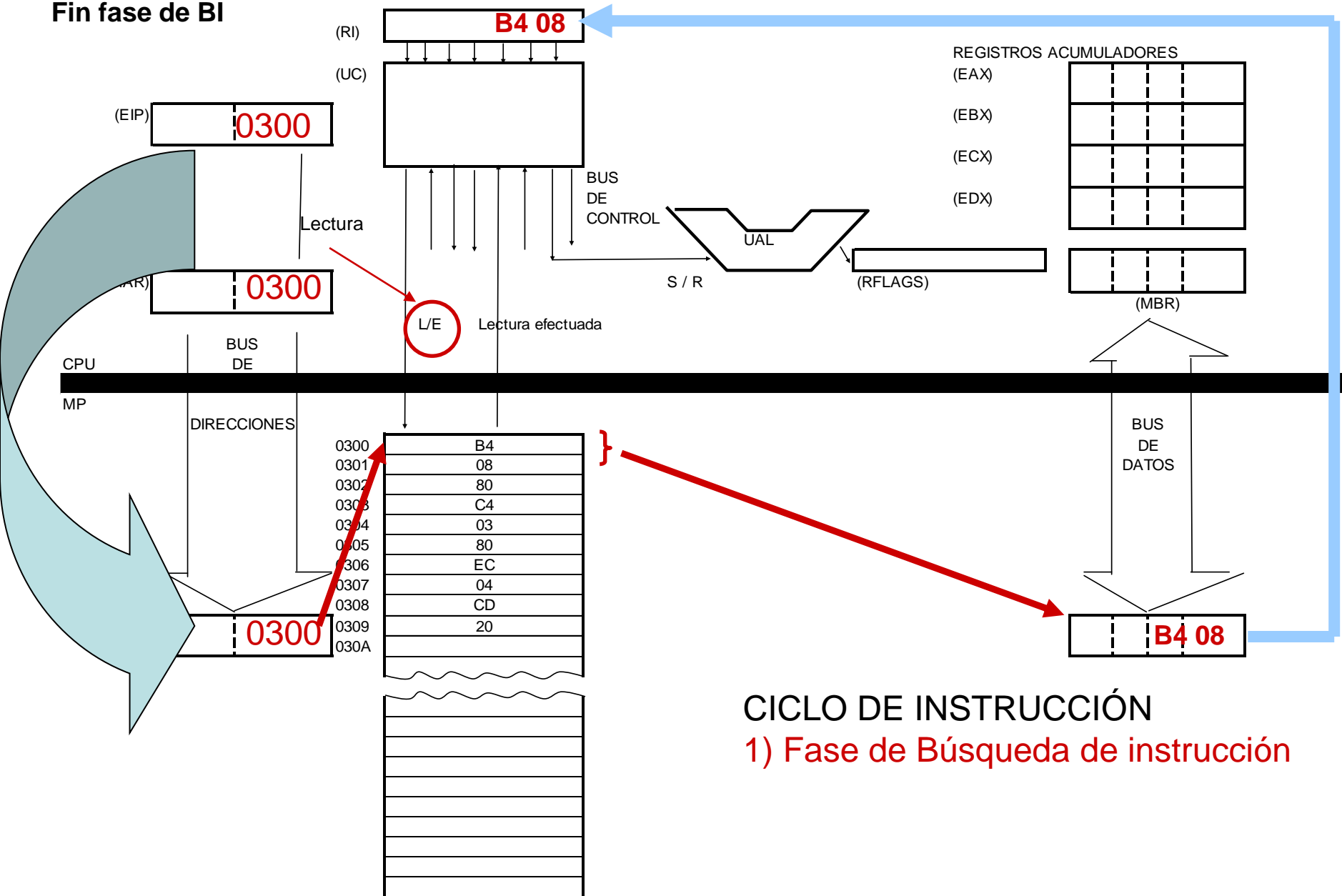




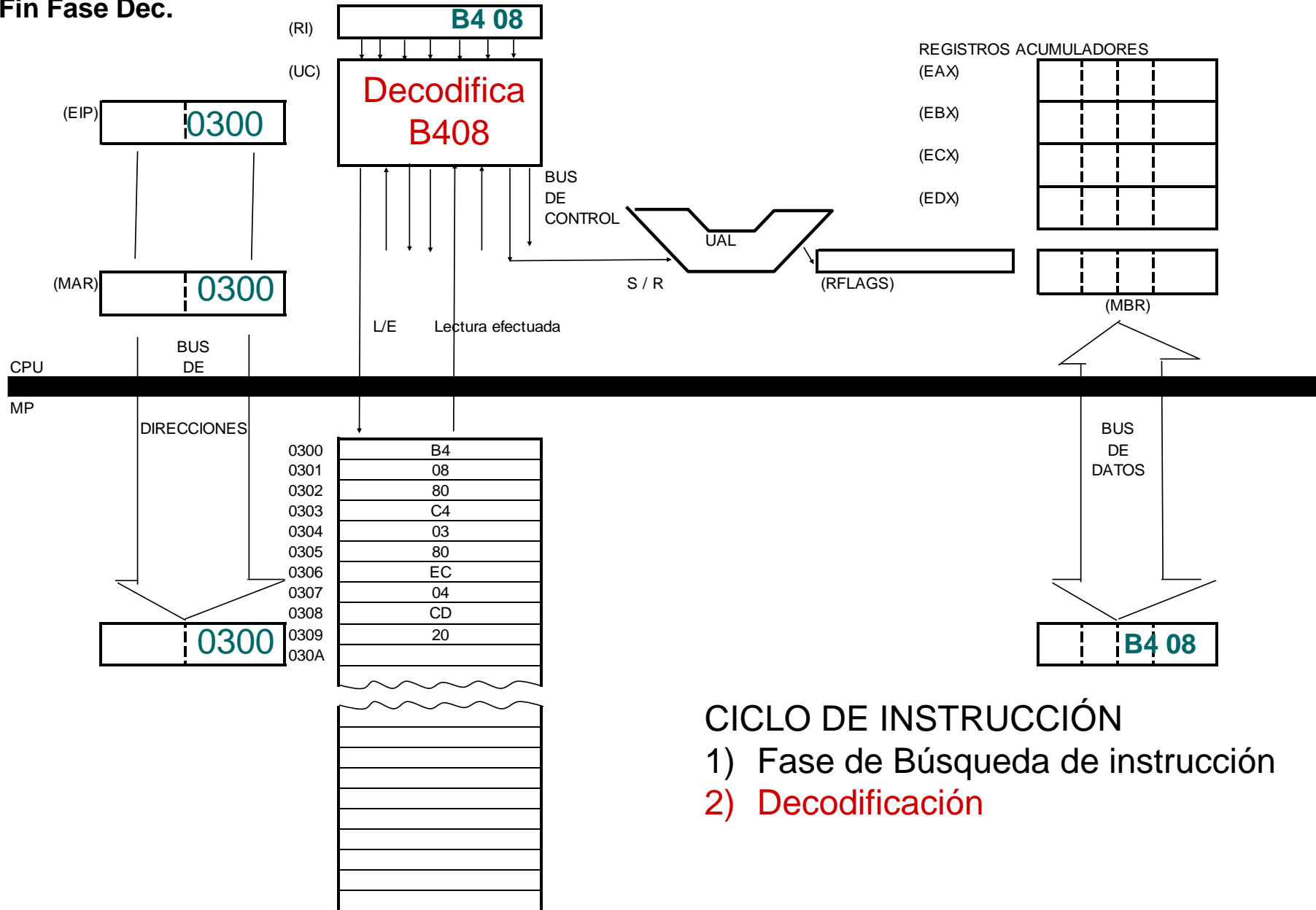




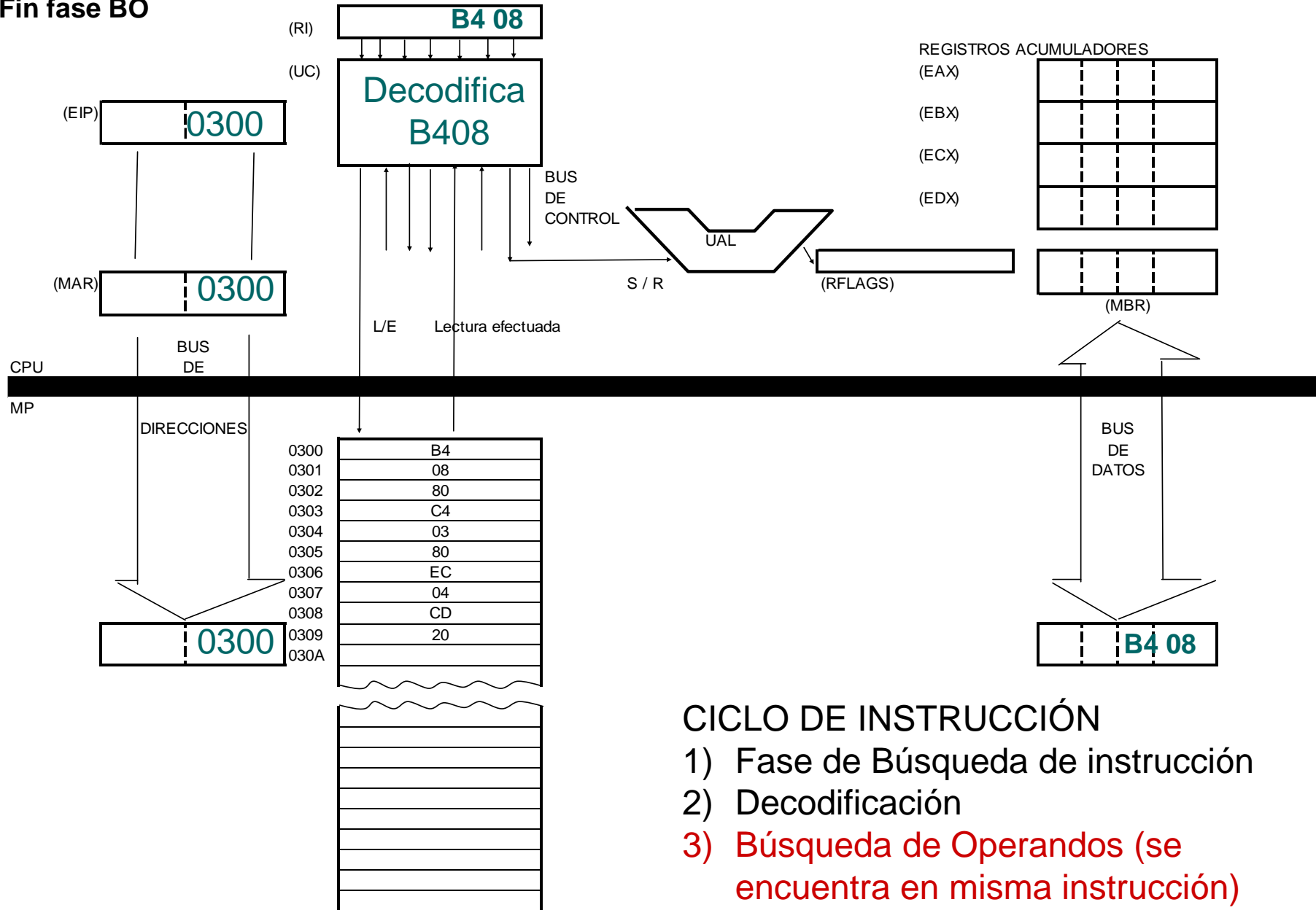
Fin fase de BI

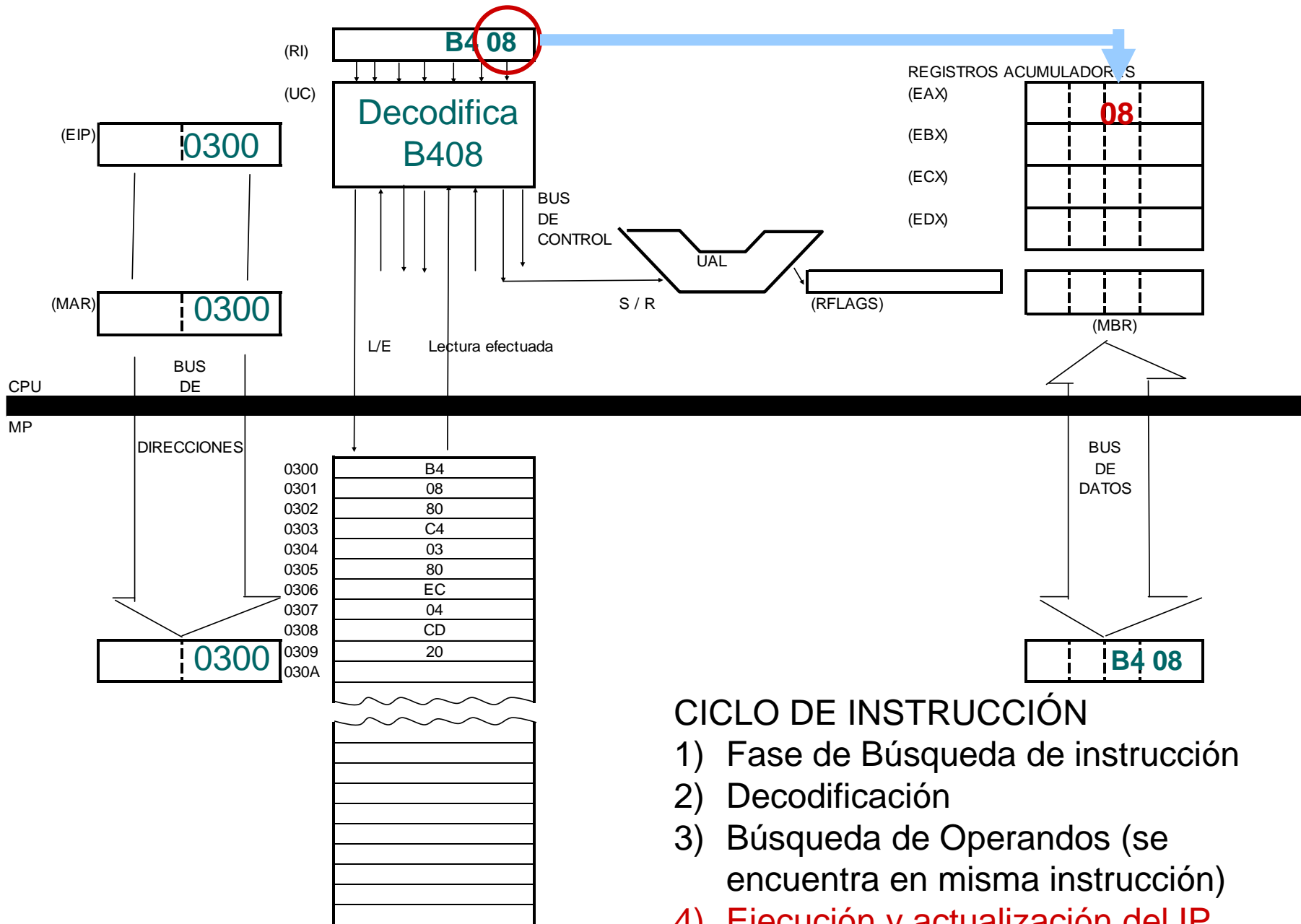


Fin Fase Dec.

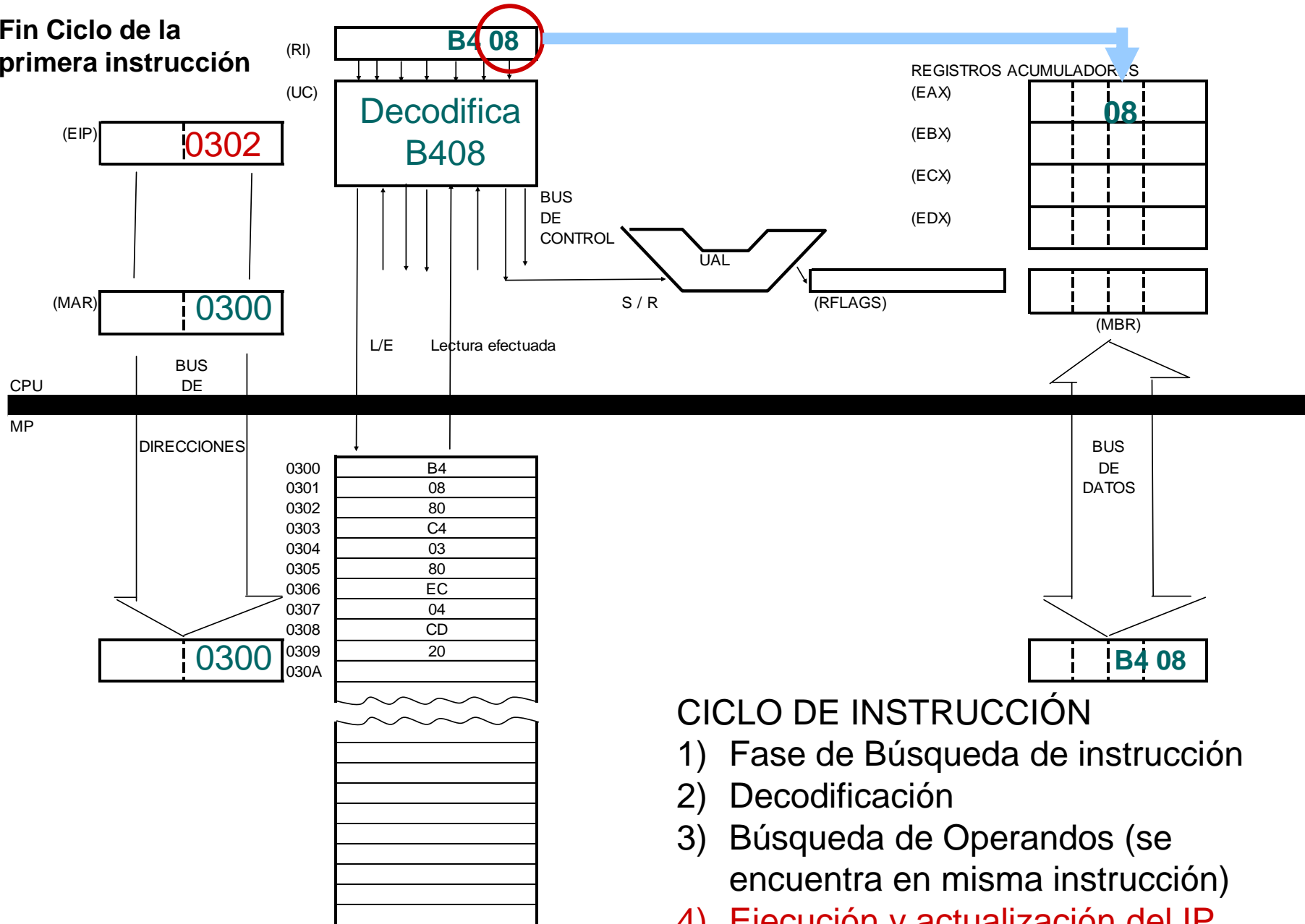


Fin fase BO



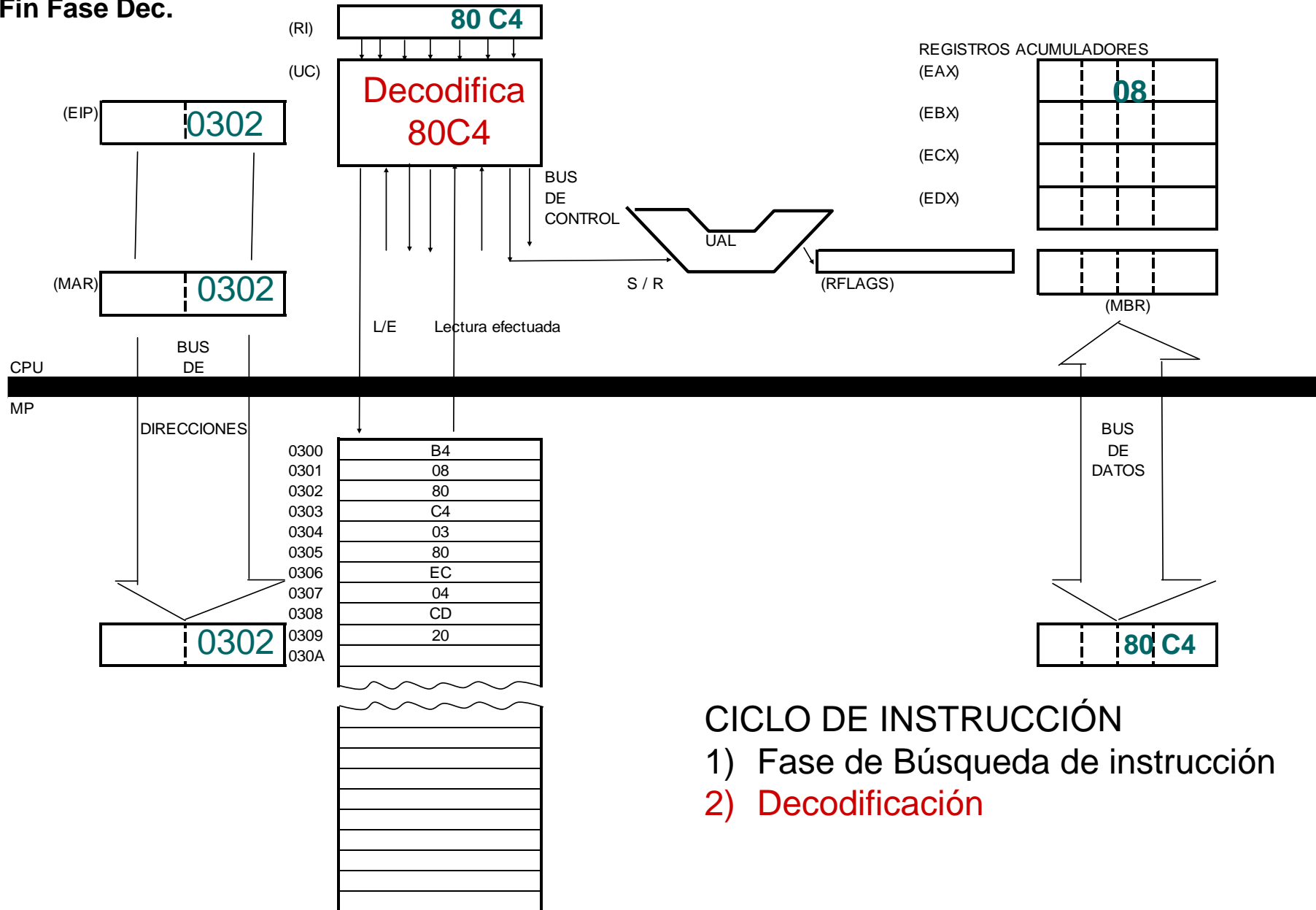


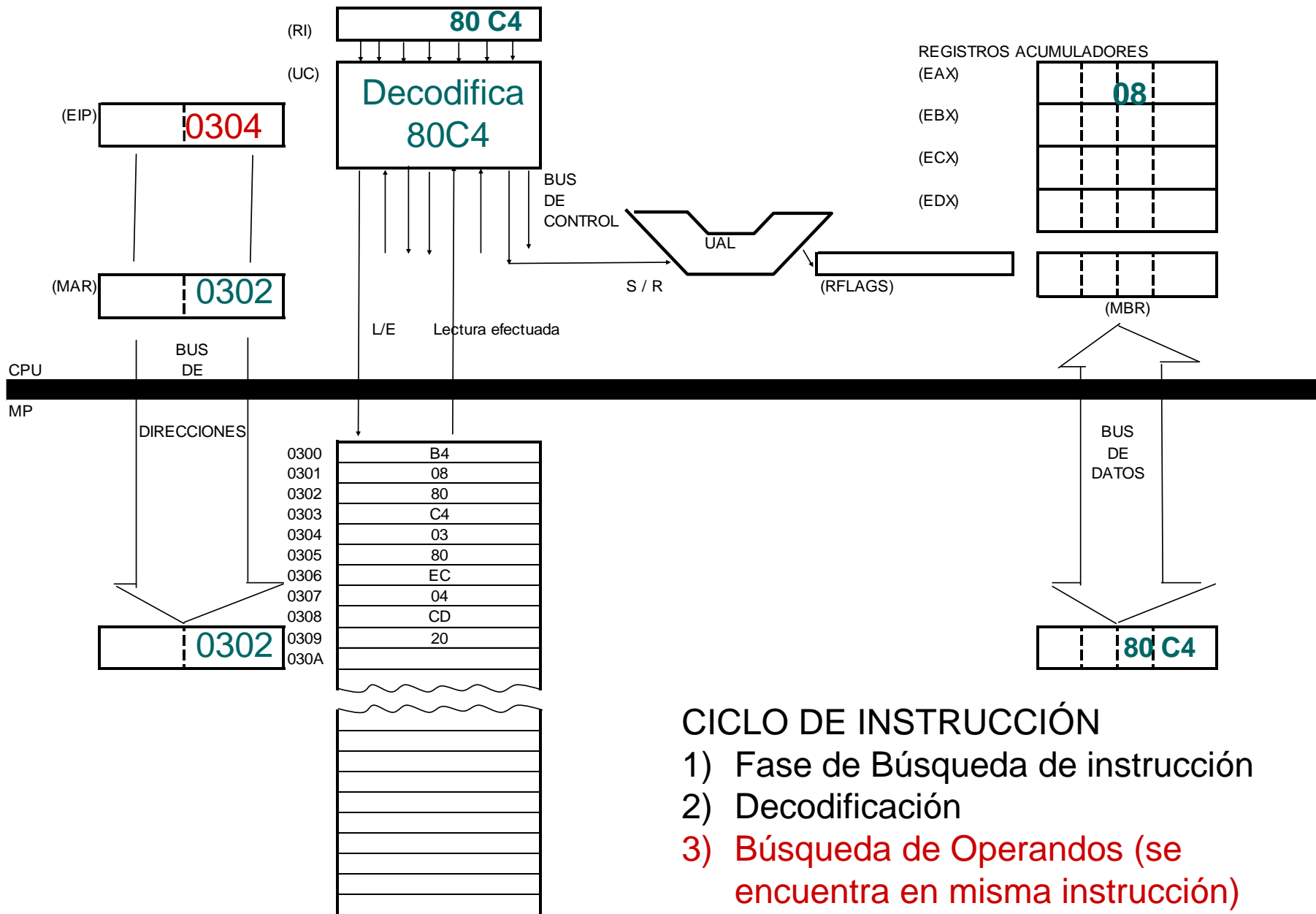
Fin Ciclo de la primera instrucción



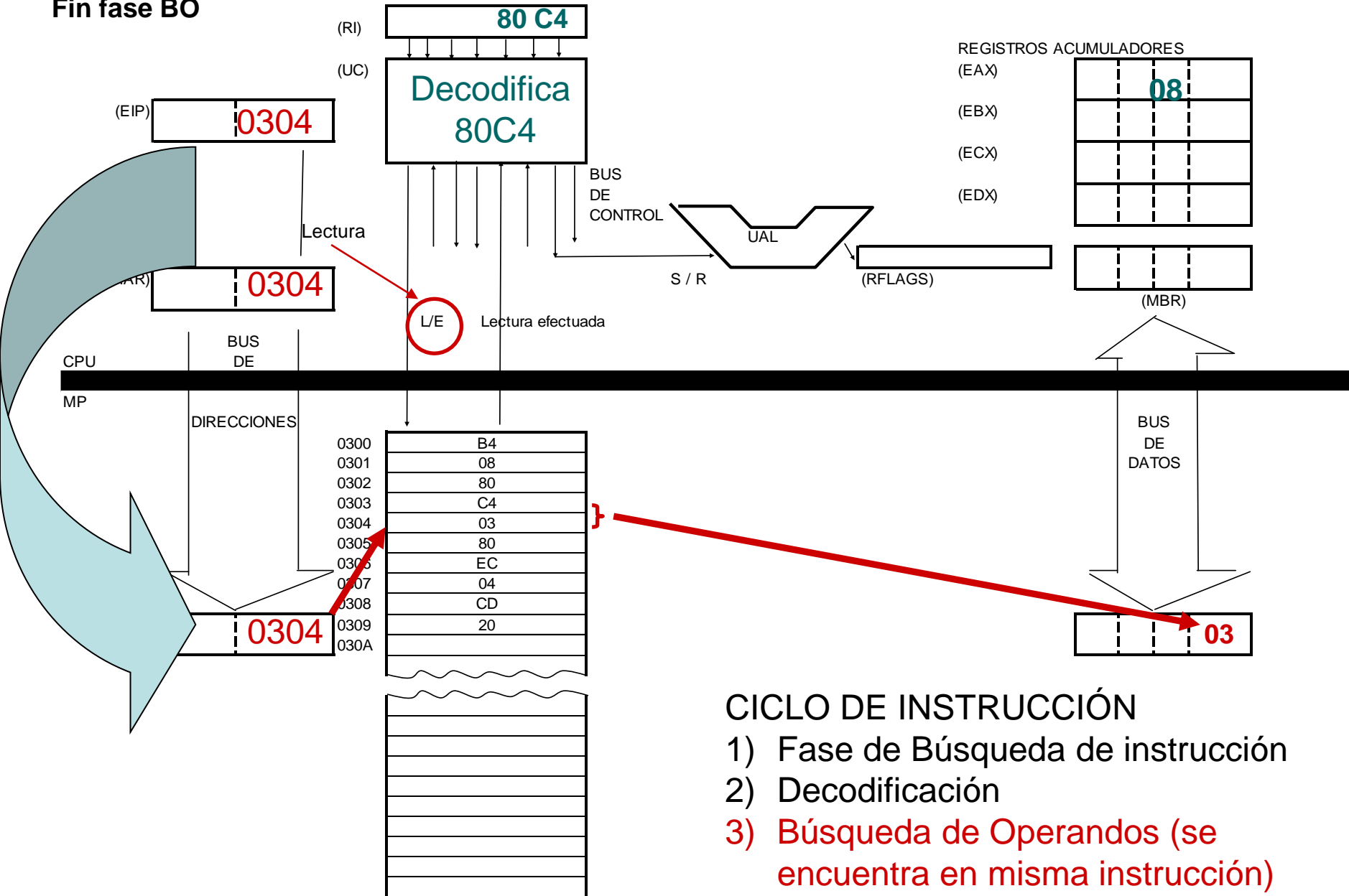


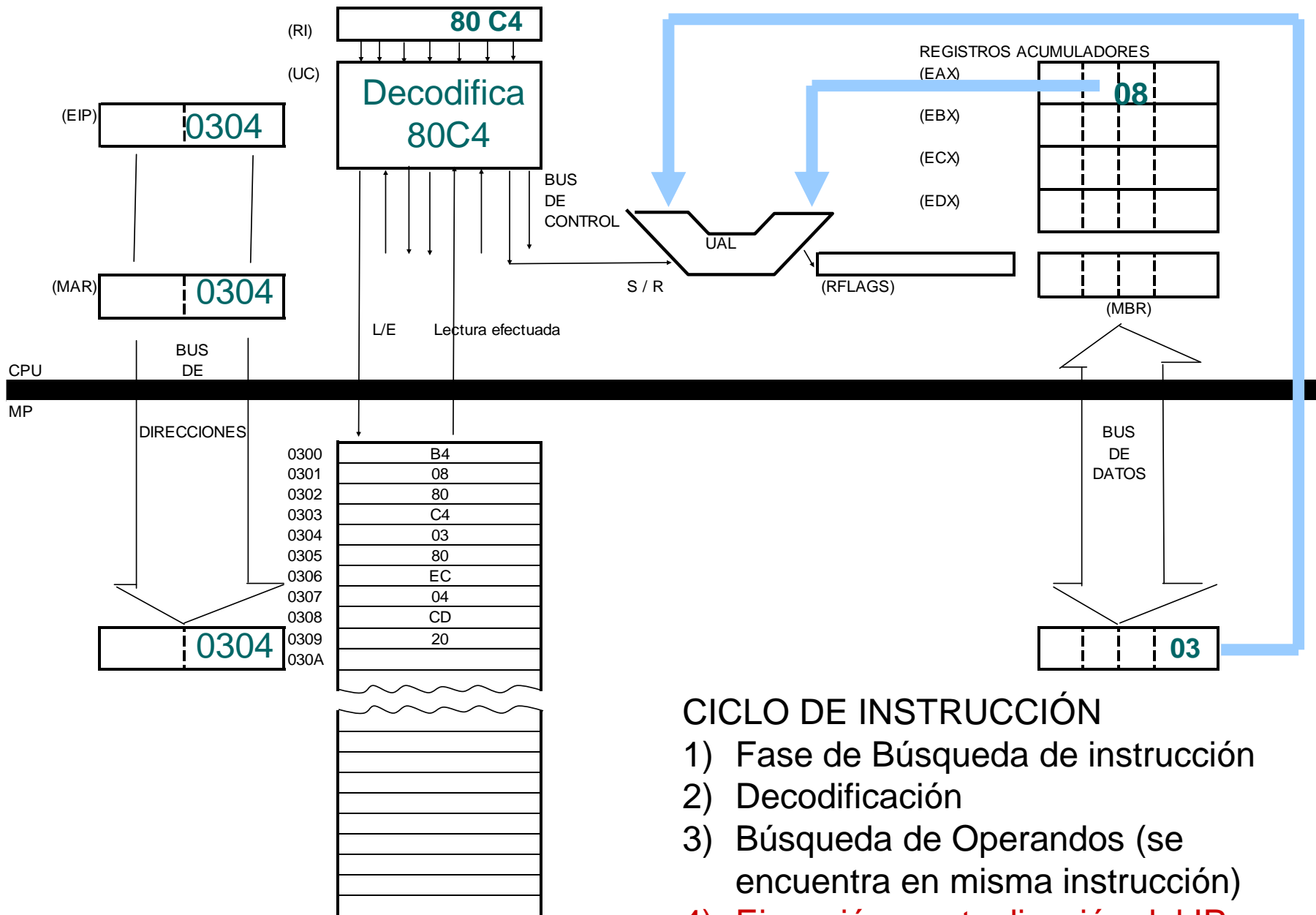
Fin Fase Dec.





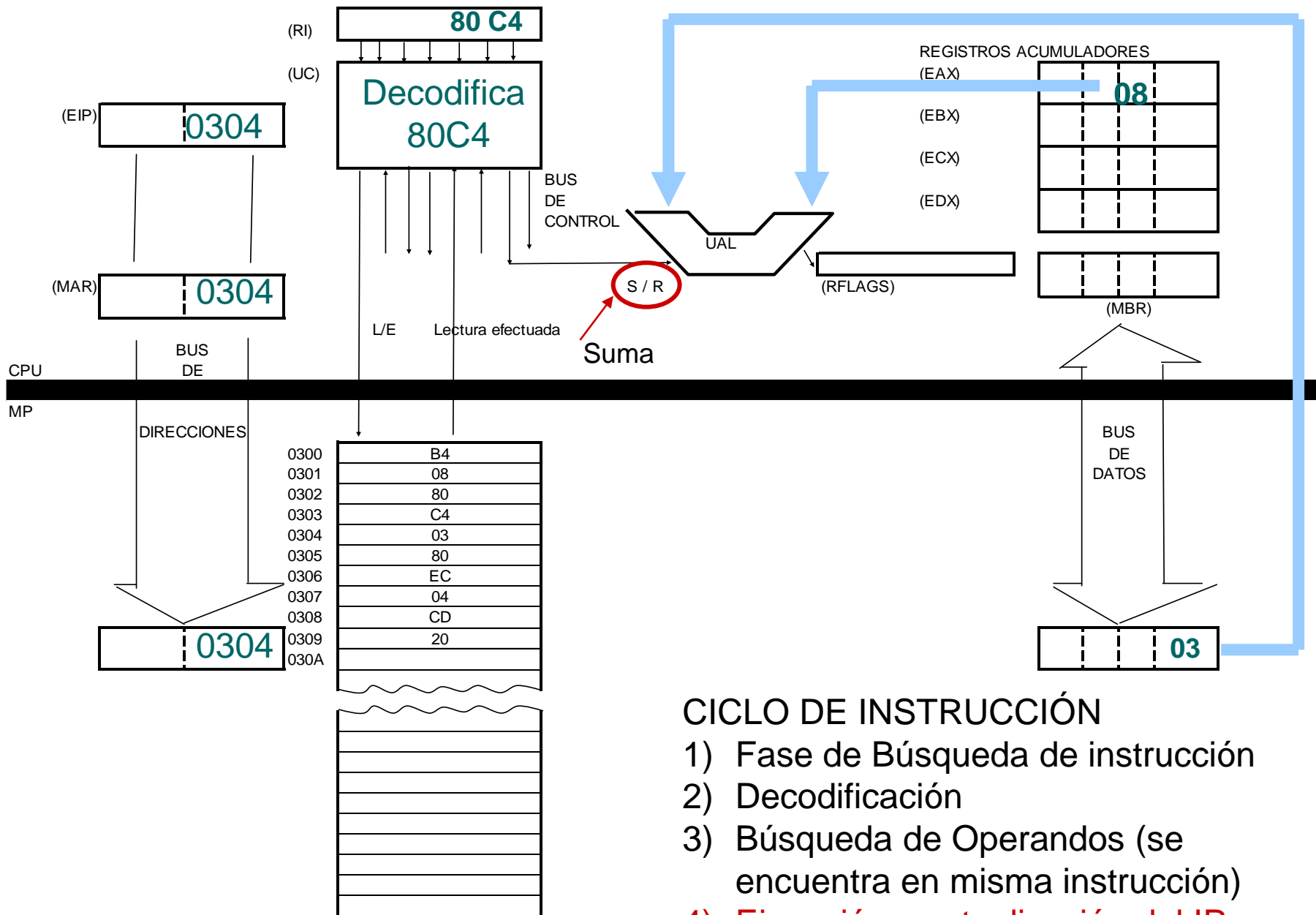
Fin fase BO





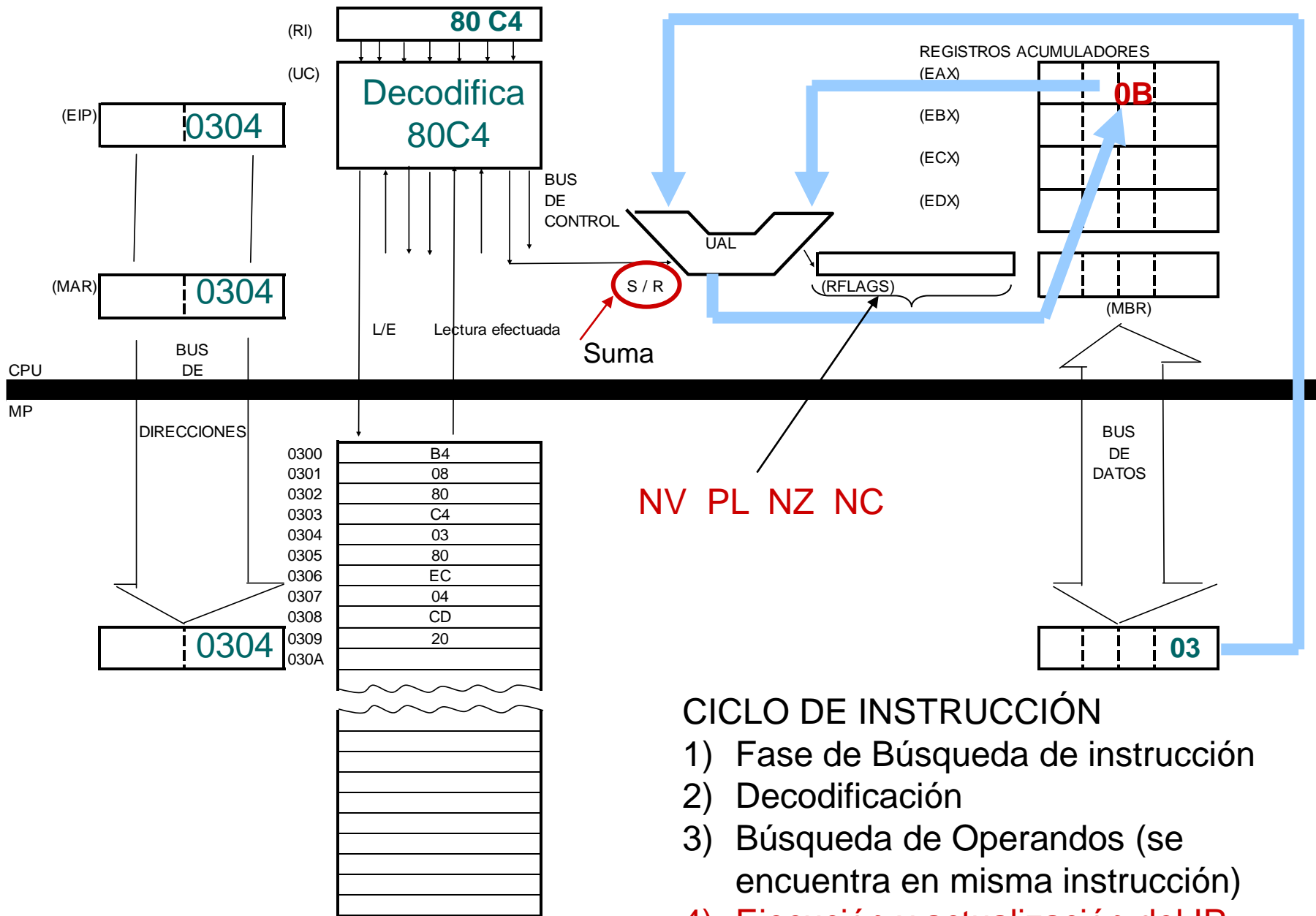
CICLO DE INSTRUCCIÓN

- 1) Fase de Búsqueda de instrucción
- 2) Decodificación
- 3) Búsqueda de Operandos (se encuentra en misma instrucción)
- 4) **Ejecución y actualización del IP**



CICLO DE INSTRUCCIÓN

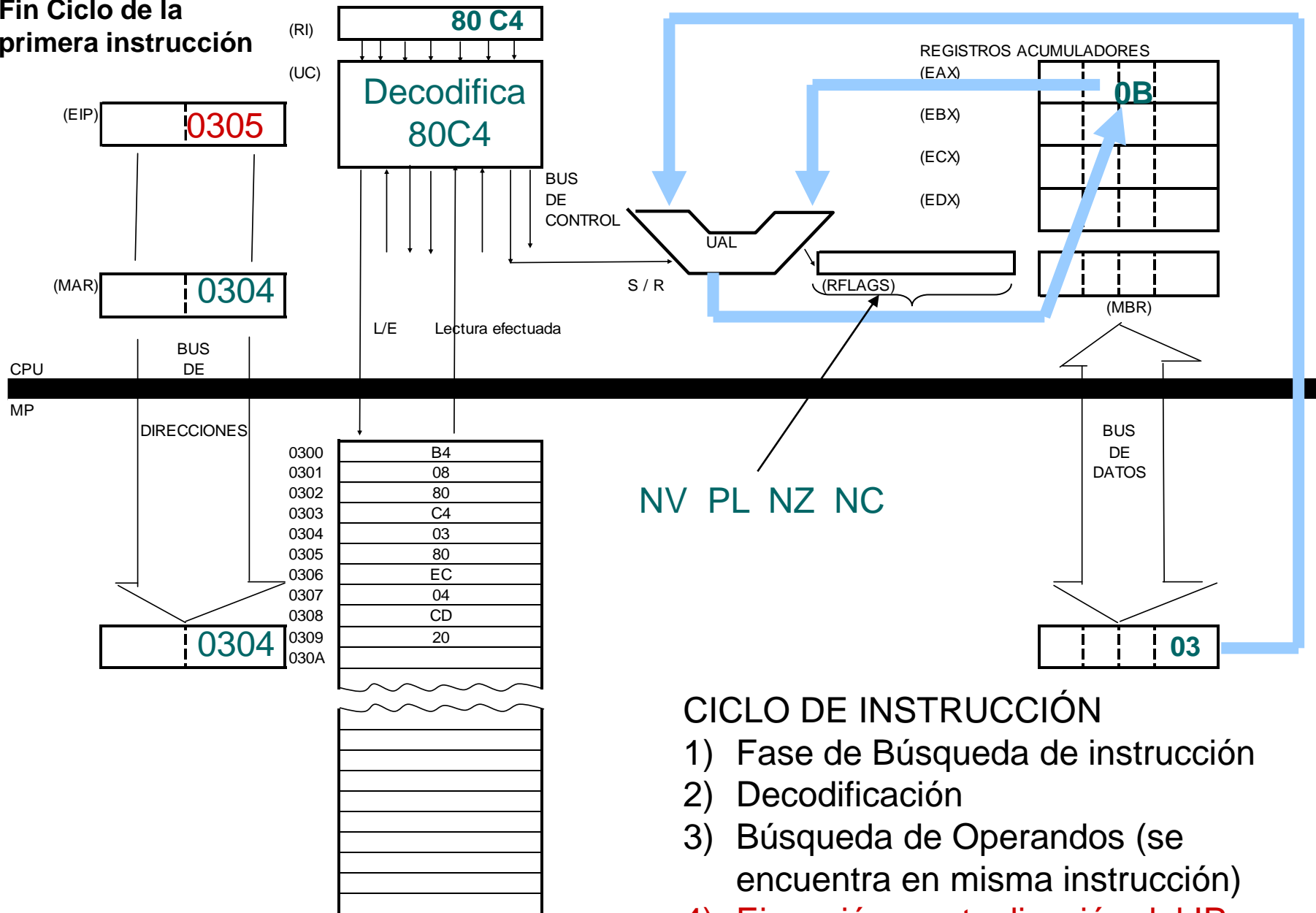
- 1) Fase de Búsqueda de instrucción
- 2) Decodificación
- 3) Búsqueda de Operandos (se encuentra en misma instrucción)
- 4) Ejecución y actualización del IP



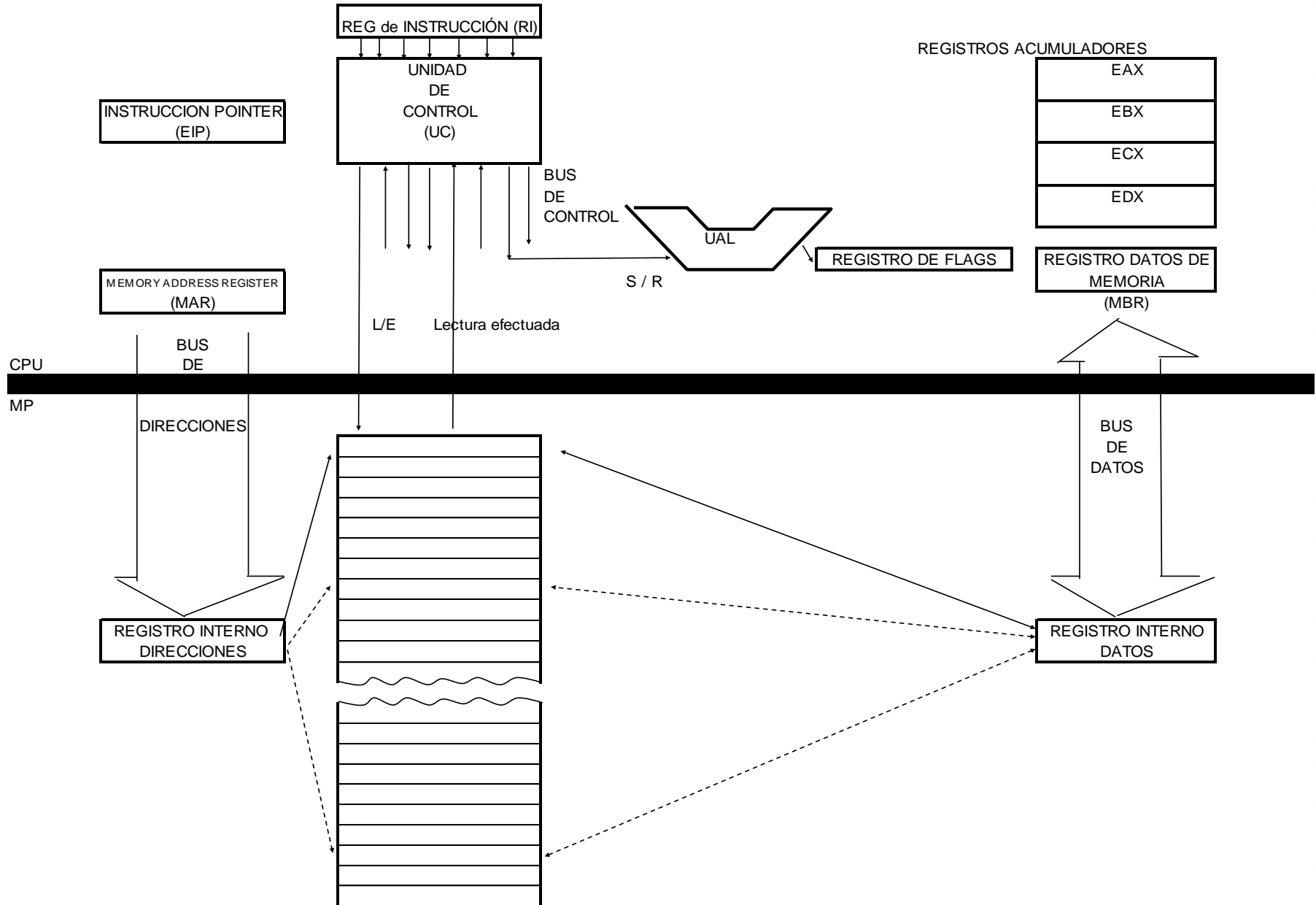
CICLO DE INSTRUCCIÓN

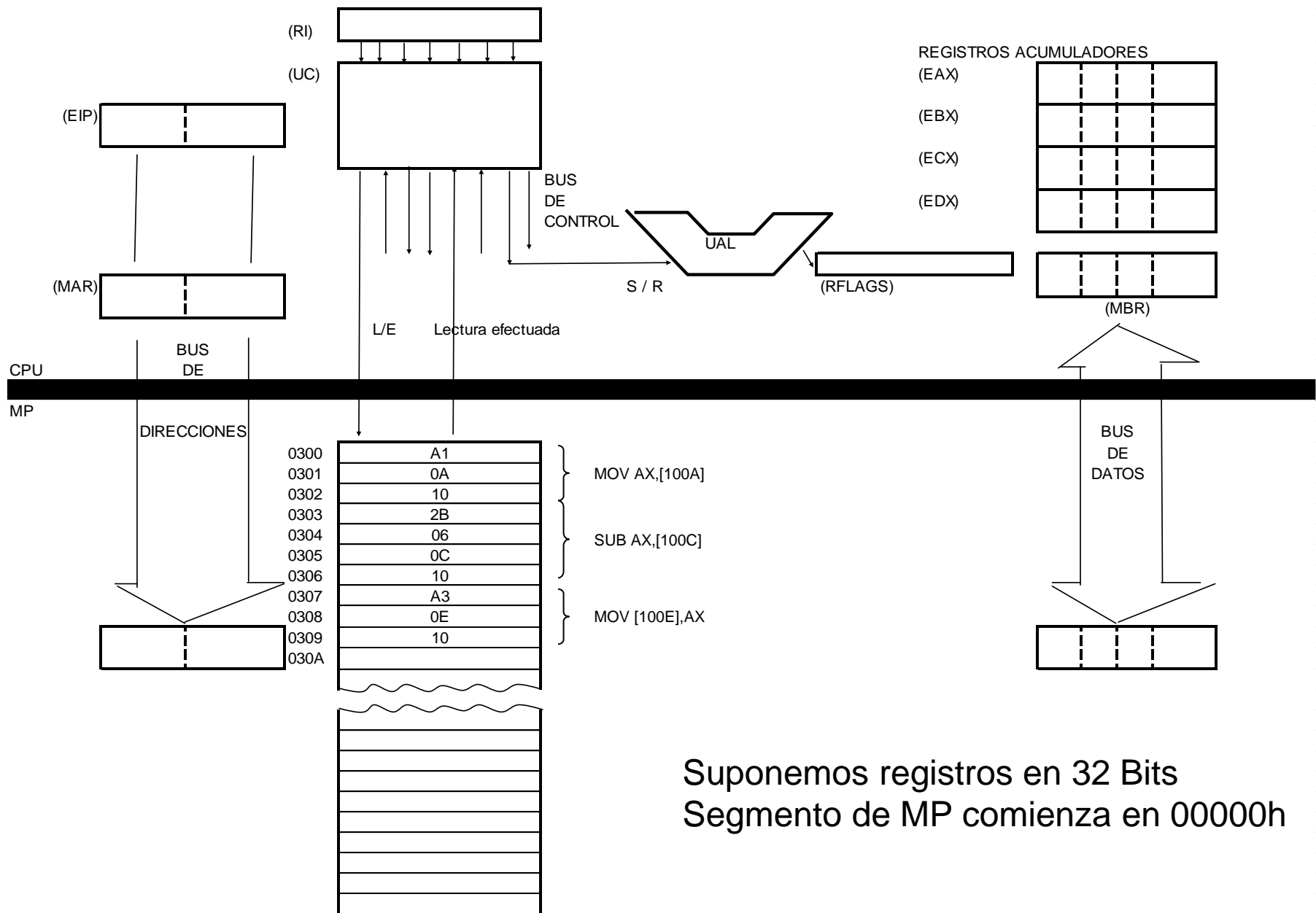
- 1) Fase de Búsqueda de instrucción
- 2) Decodificación
- 3) Búsqueda de Operandos (se encuentra en misma instrucción)
- 4) Ejecución y actualización del IP

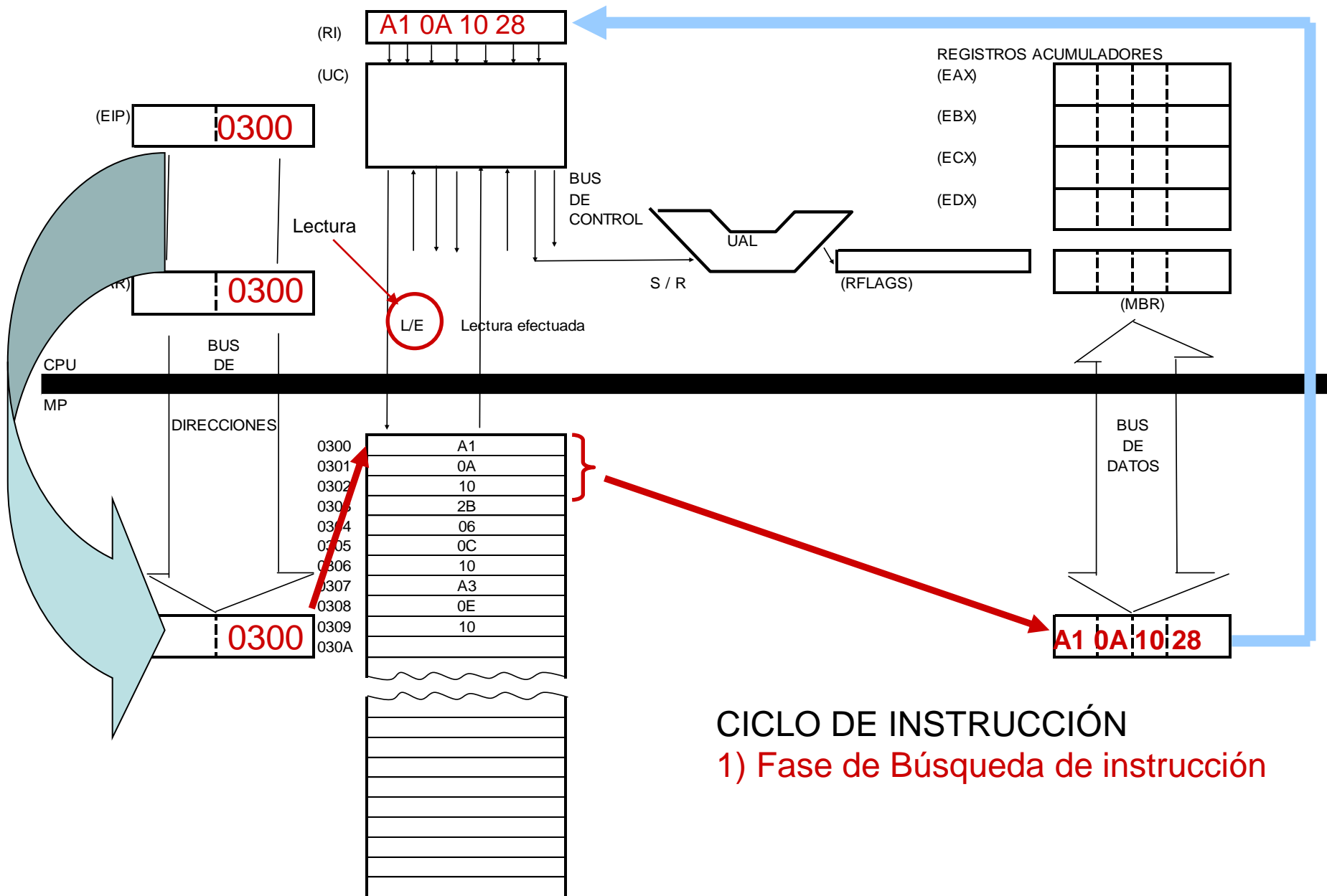
Fin Ciclo de la primera instrucción

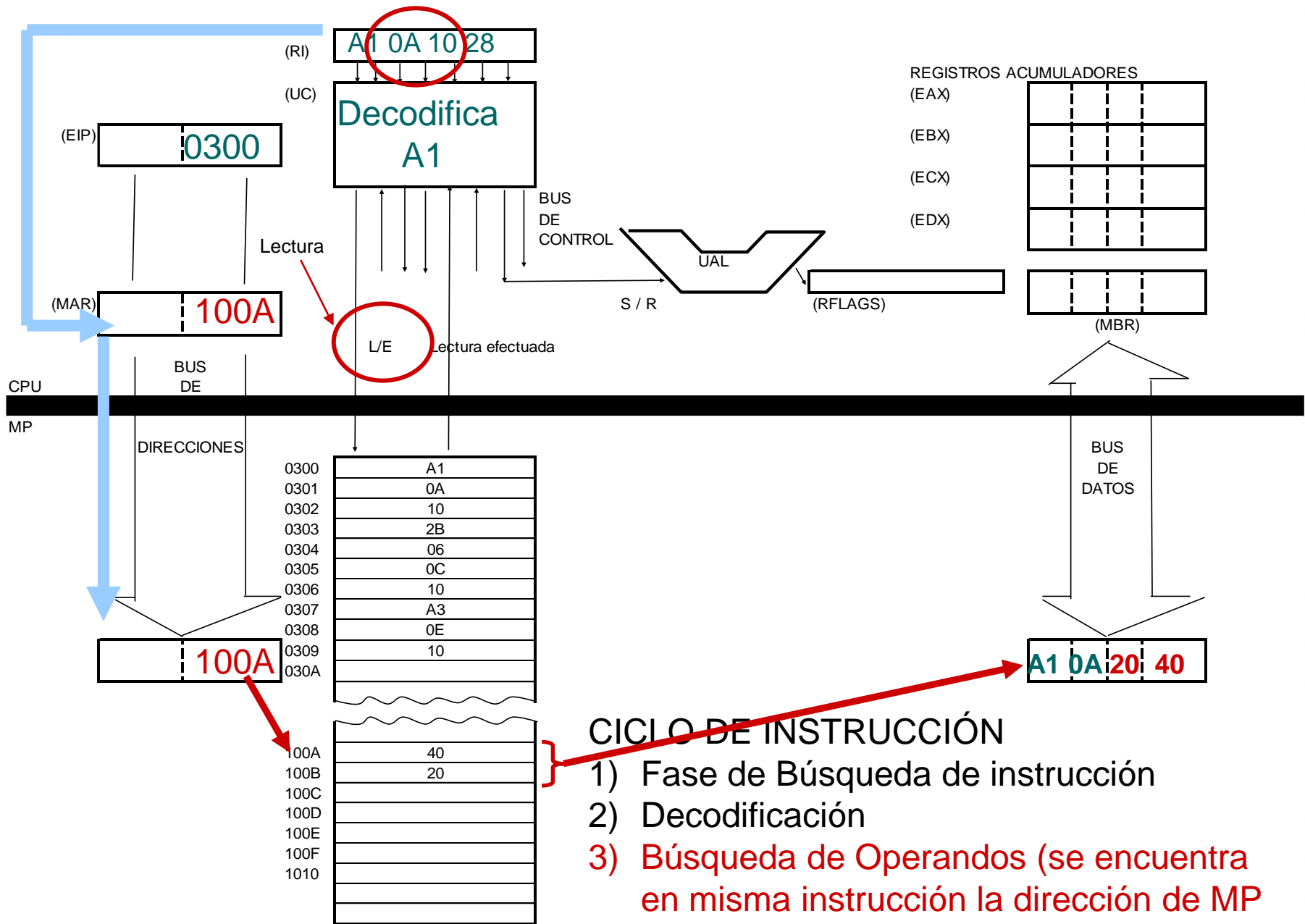


ESQUEMA GENERAL PARA ANALIZAR EL CICLO DE INSTRUCCION

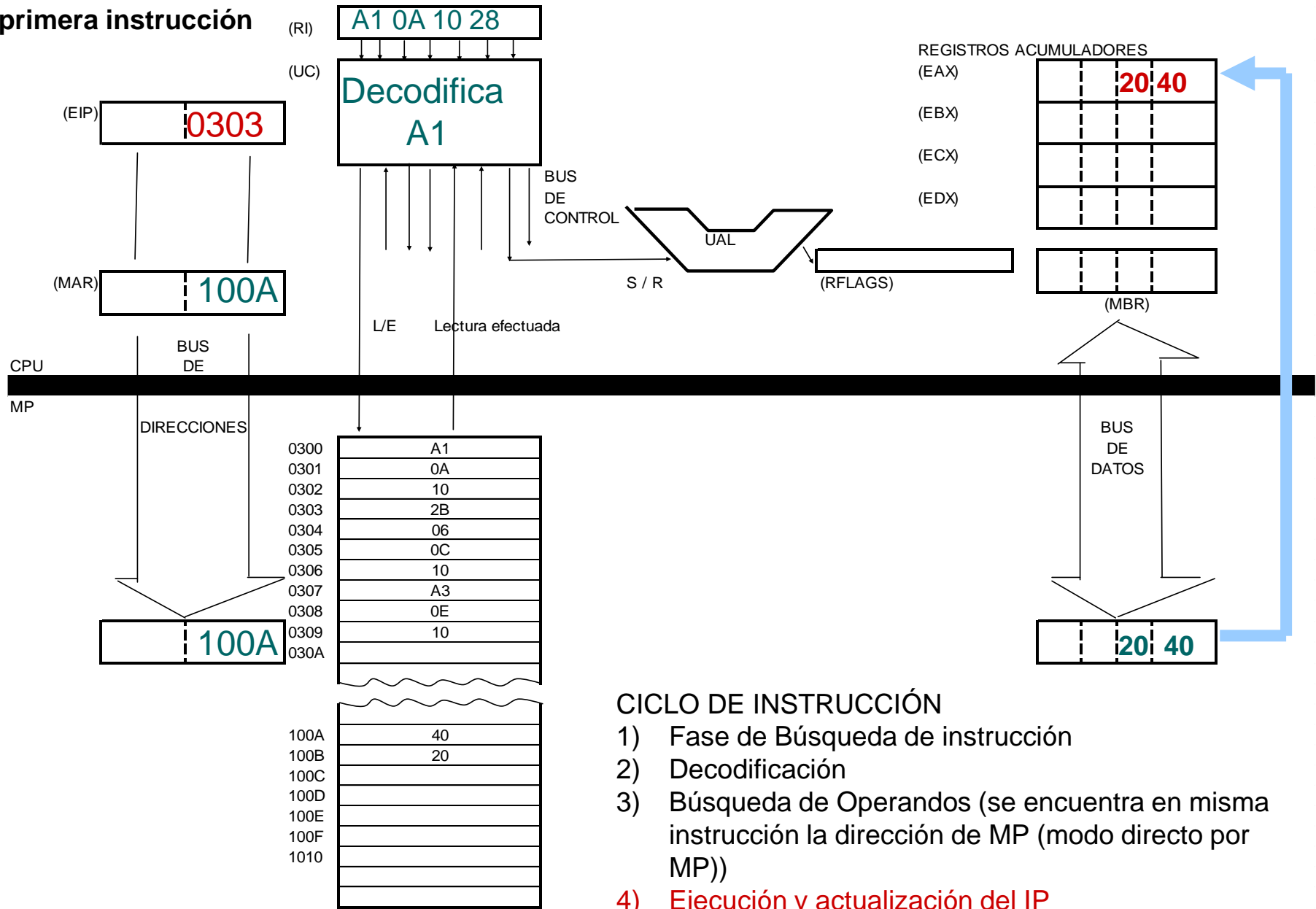


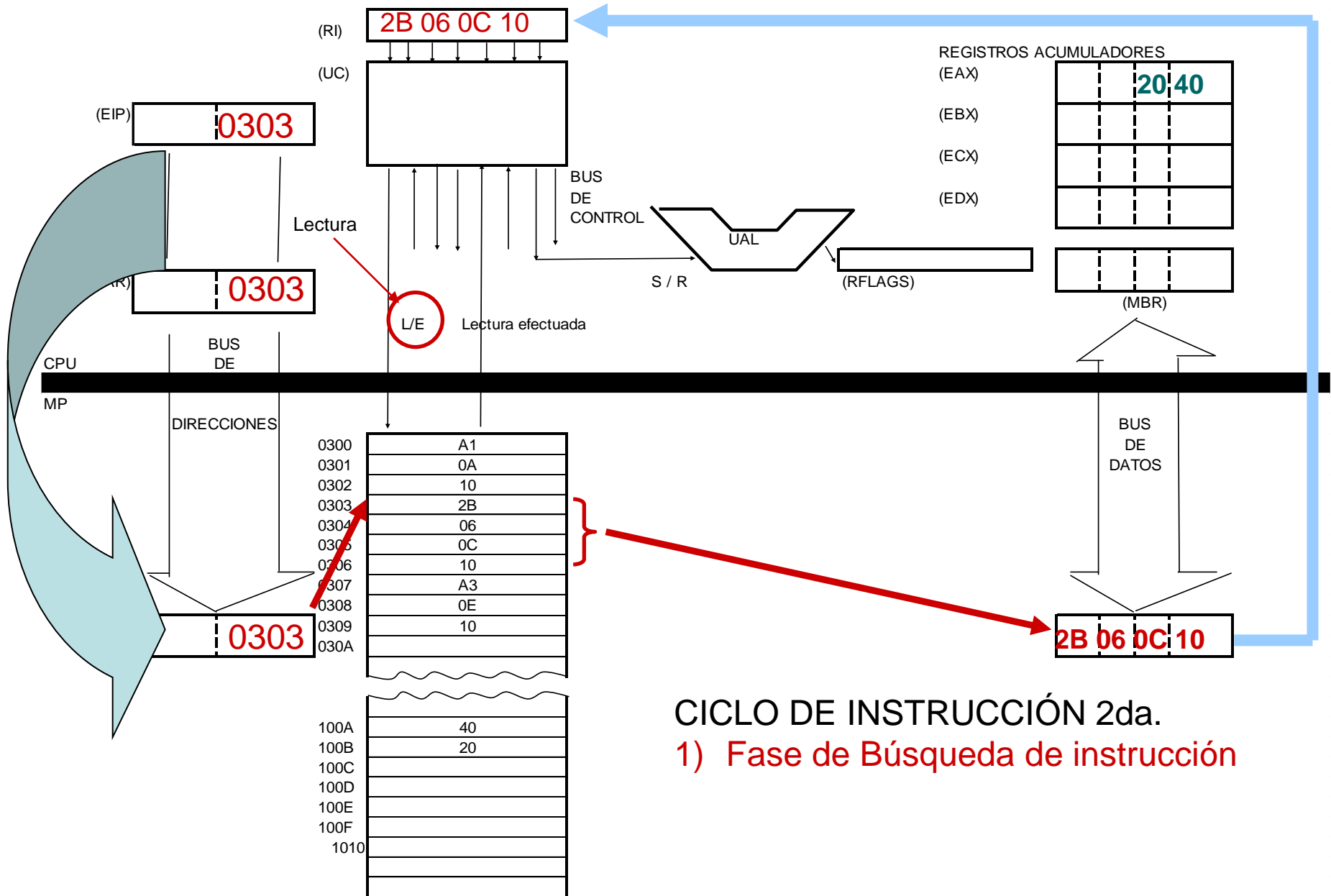


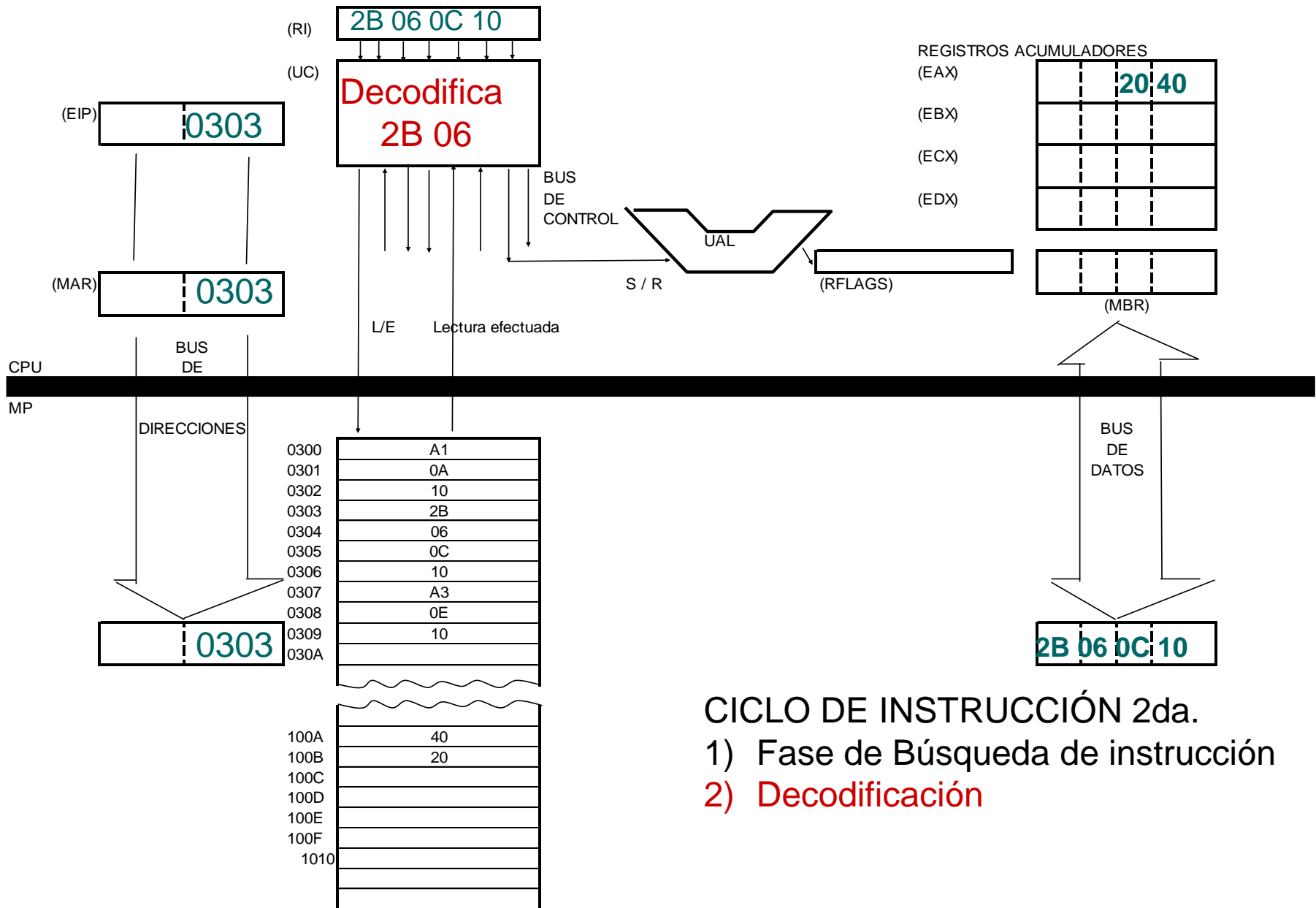


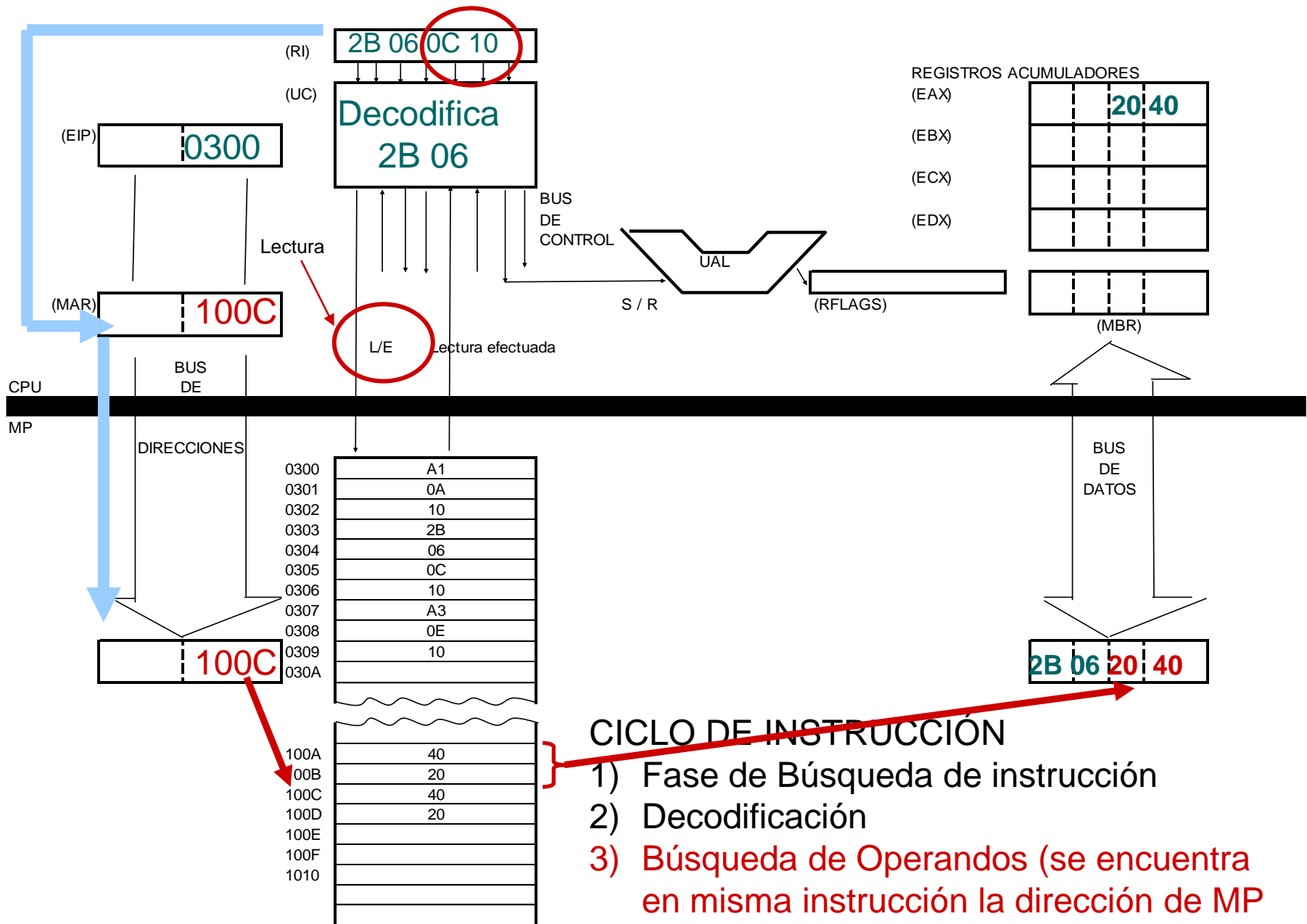


Fin Ciclo de la primera instrucción



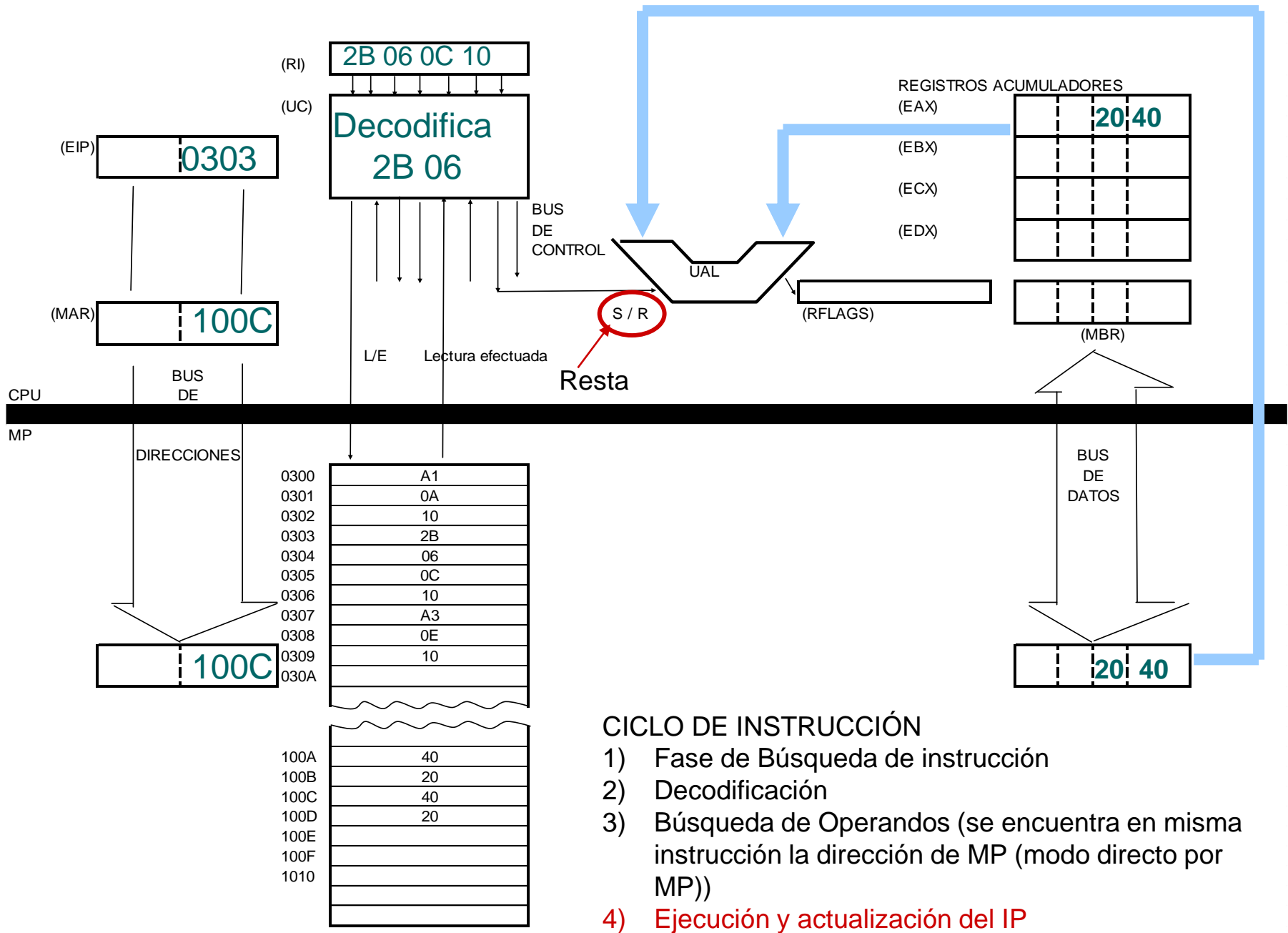




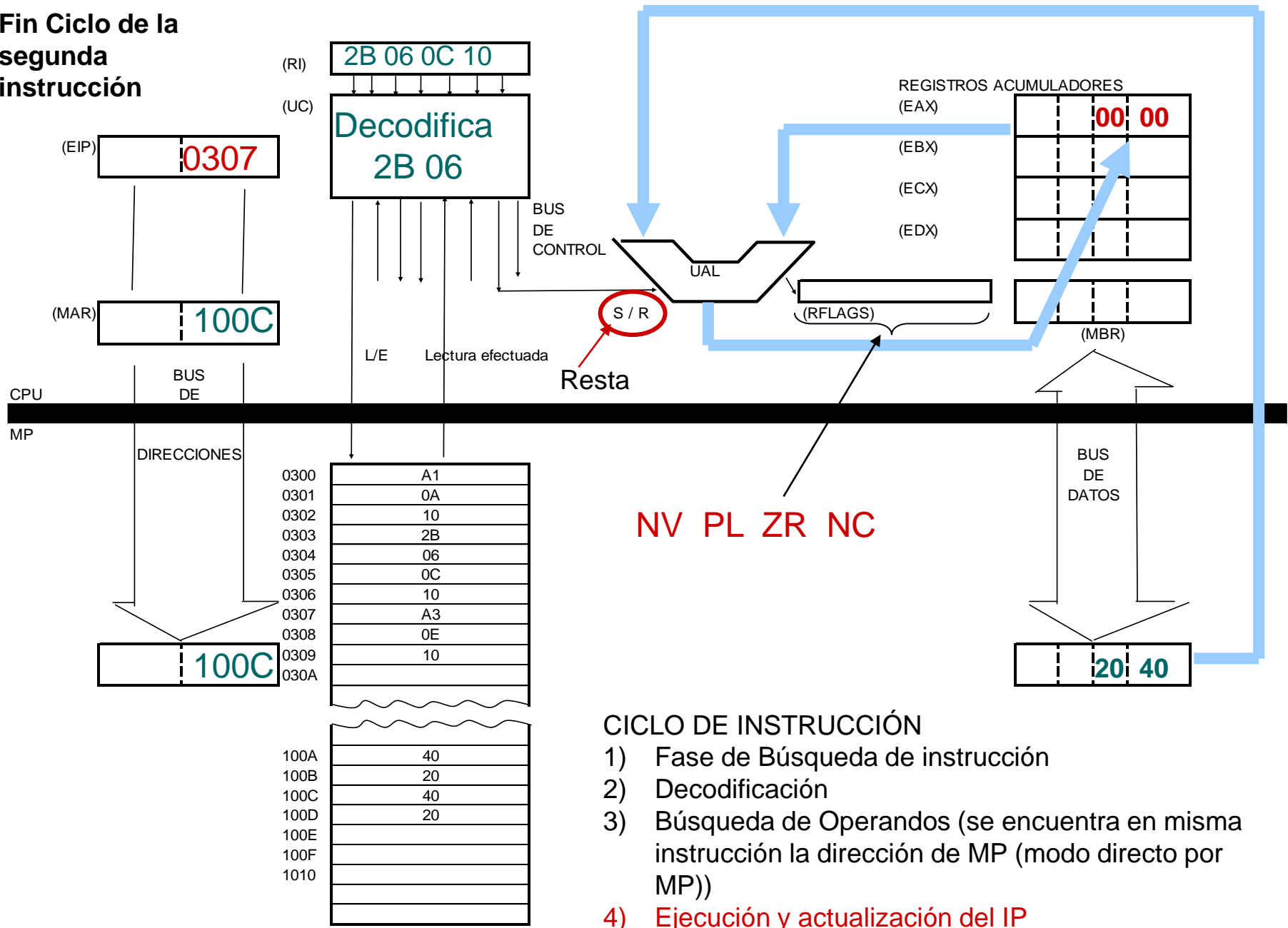


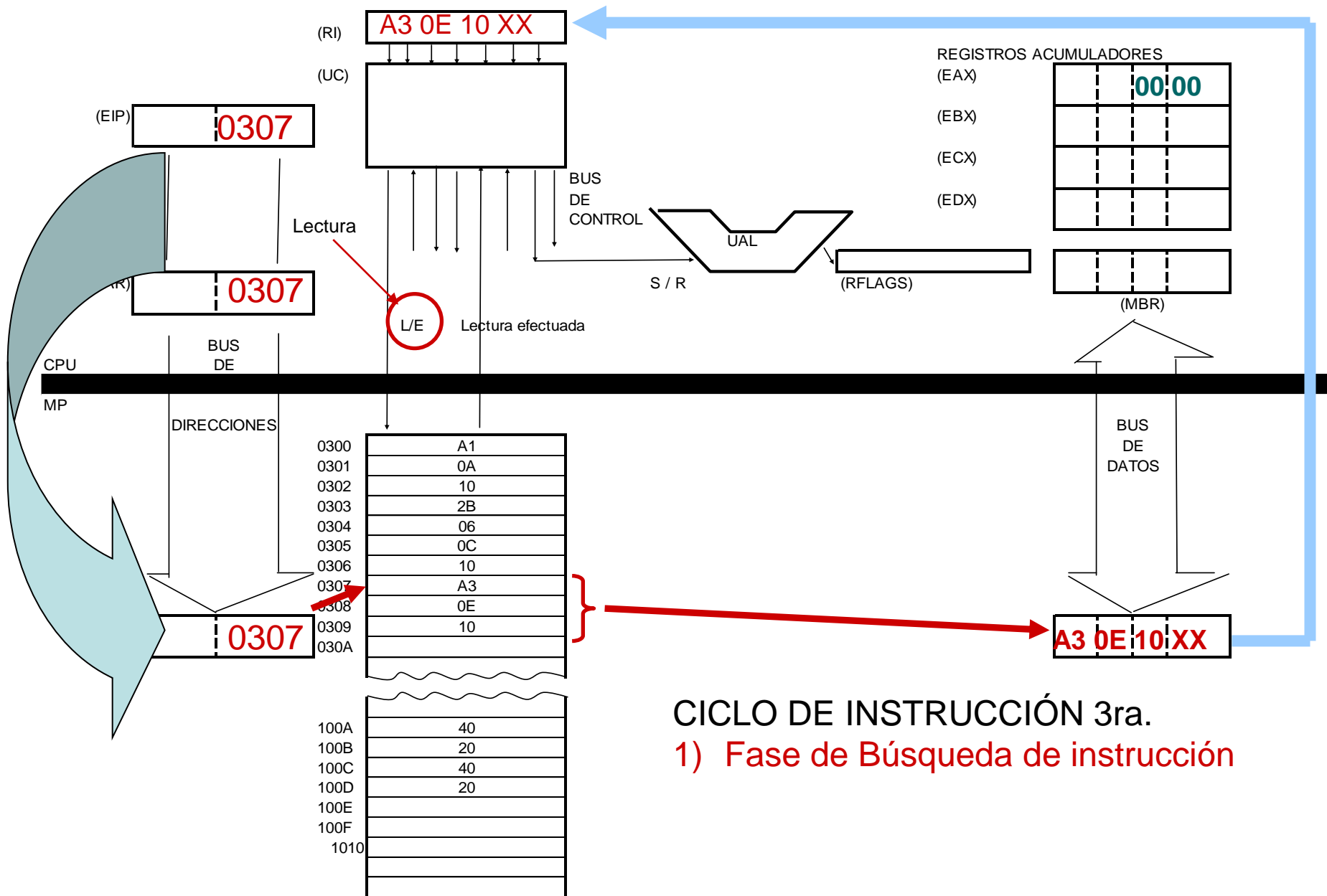
CICLO DE INSTRUCCIÓN

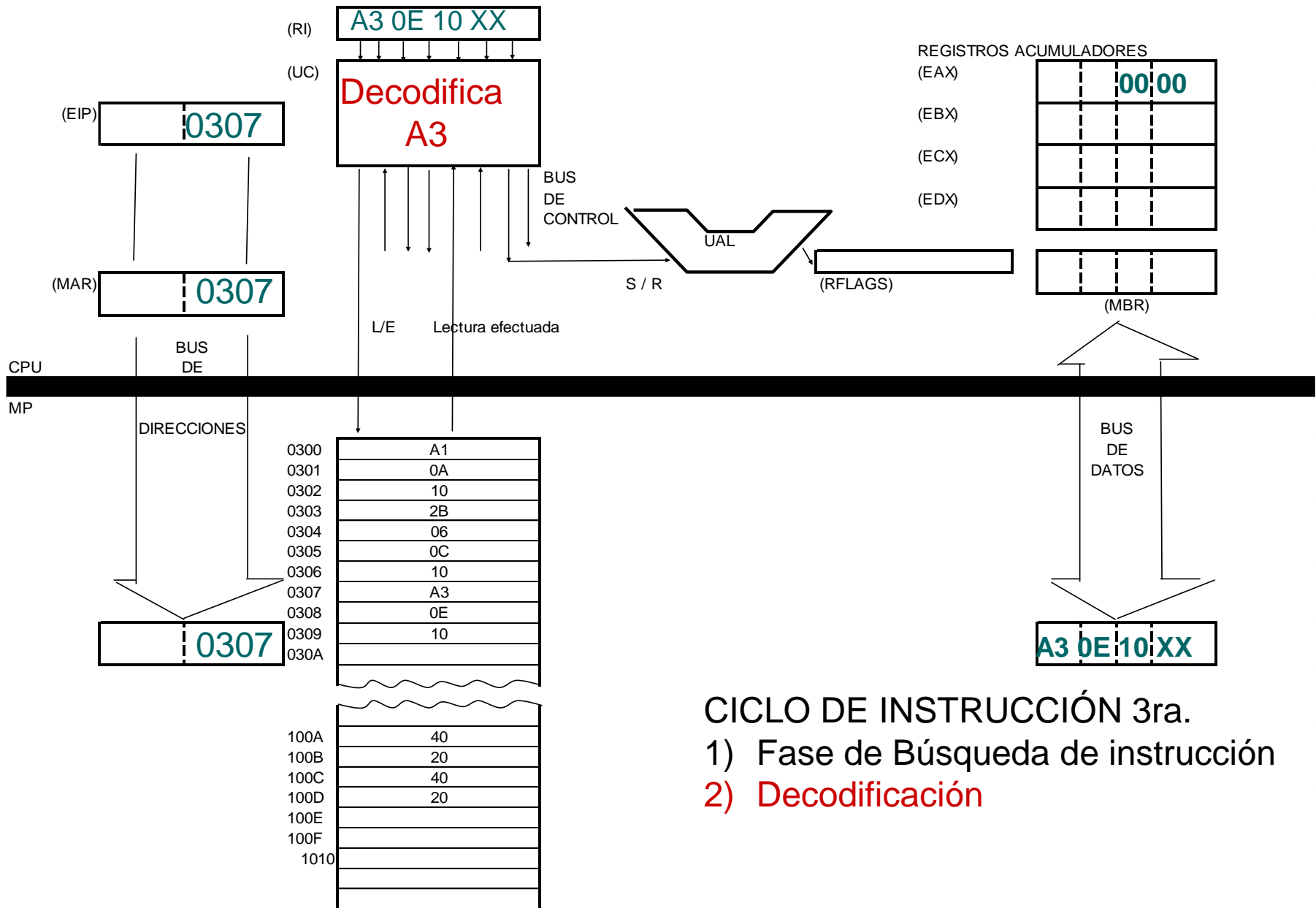
- 1) Fase de Búsqueda de instrucción
- 2) Decodificación
- 3) Búsqueda de Operandos (se encuentra en misma instrucción la dirección de MP (modo directo por MP))

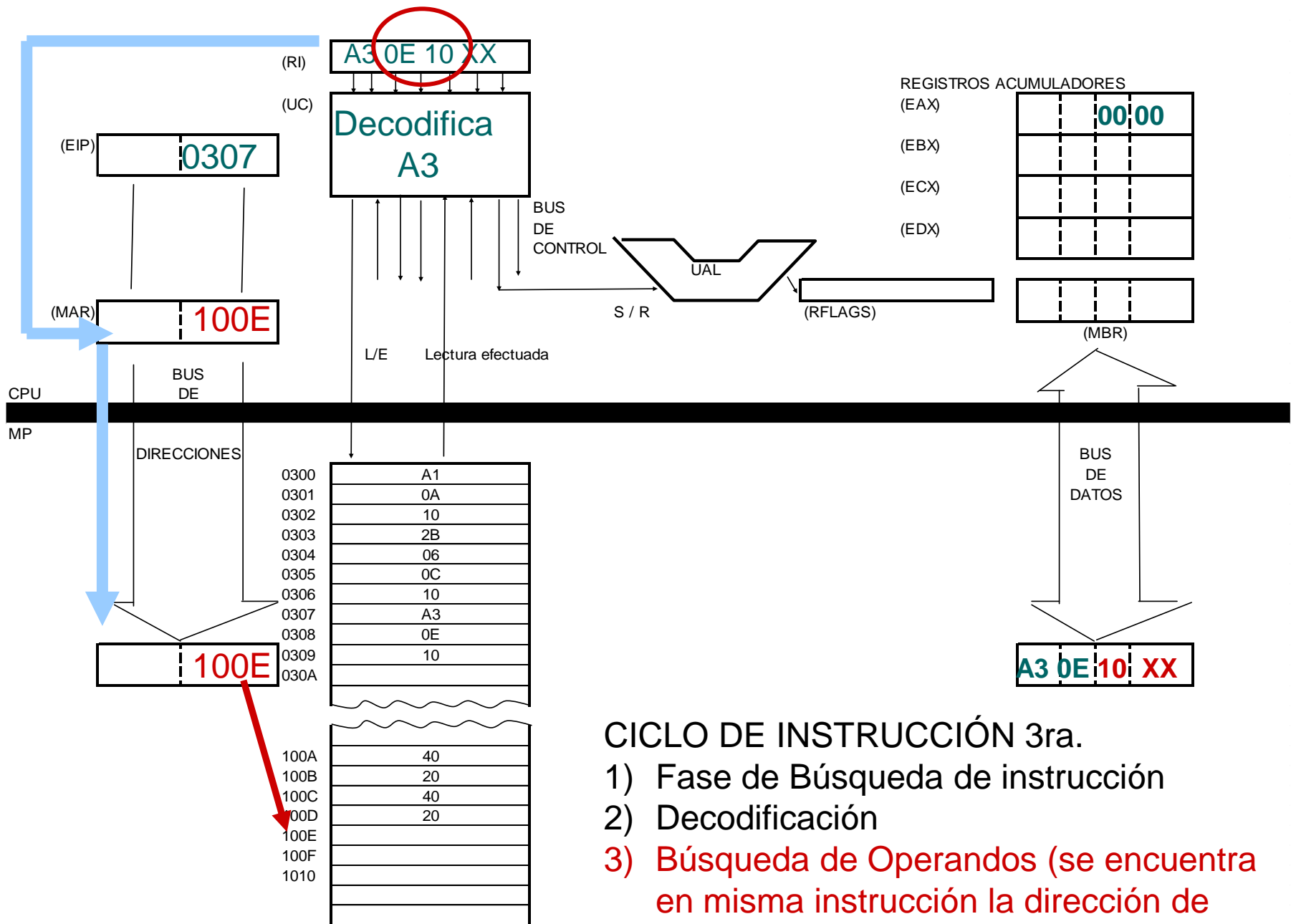


Fin Ciclo de la segunda instrucción









CICLO DE INSTRUCCIÓN 3ra.

- 1) Fase de Búsqueda de instrucción
- 2) Decodificación
- 3) Búsqueda de Operandos (se encuentra en misma instrucción la dirección de MP). Dirección donde se va a grabar AX

Fin Ciclo de la tercera instrucción

