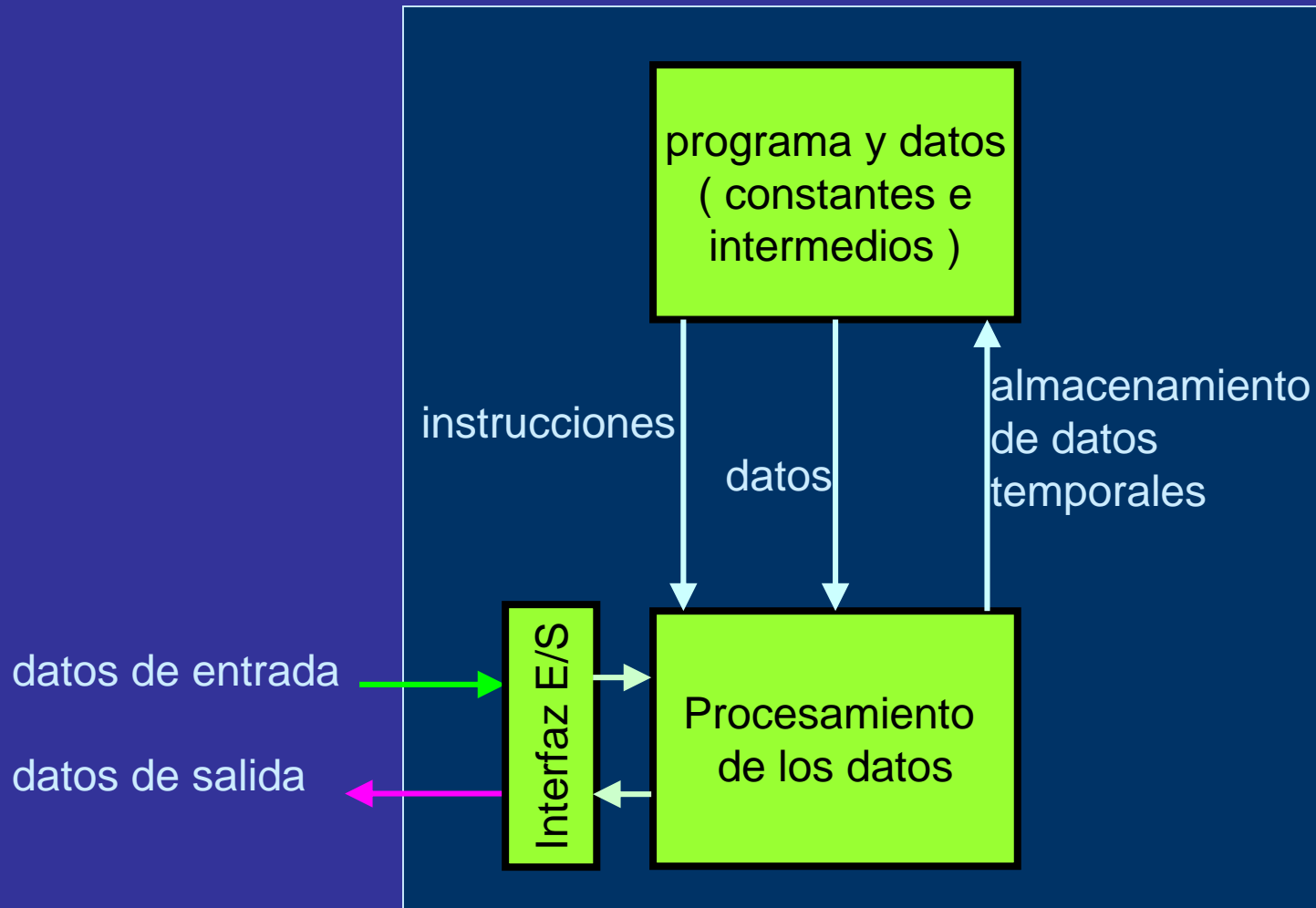


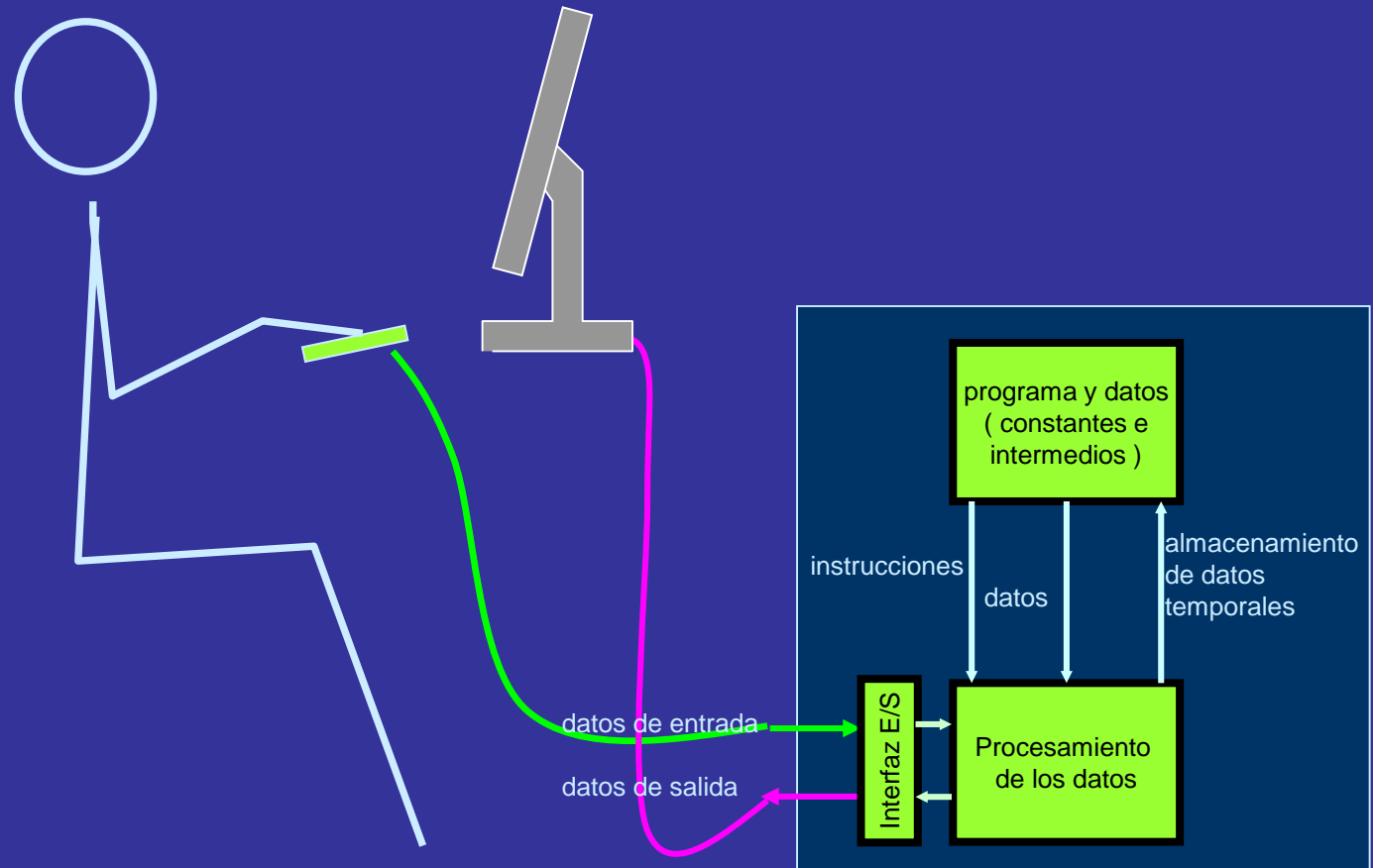
Microprocesadores y Microcontroladores

Facultad de Ingeniería – Universidad Nacional de Cuyo

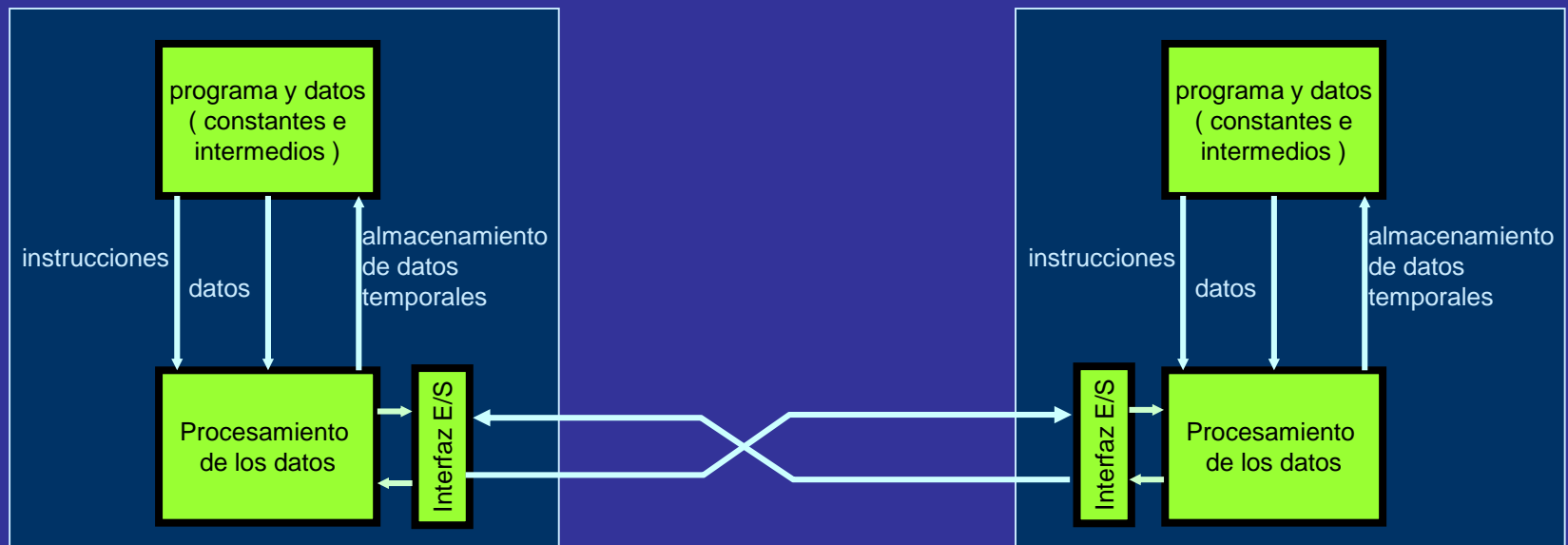
Sistema de Cómputo Programable



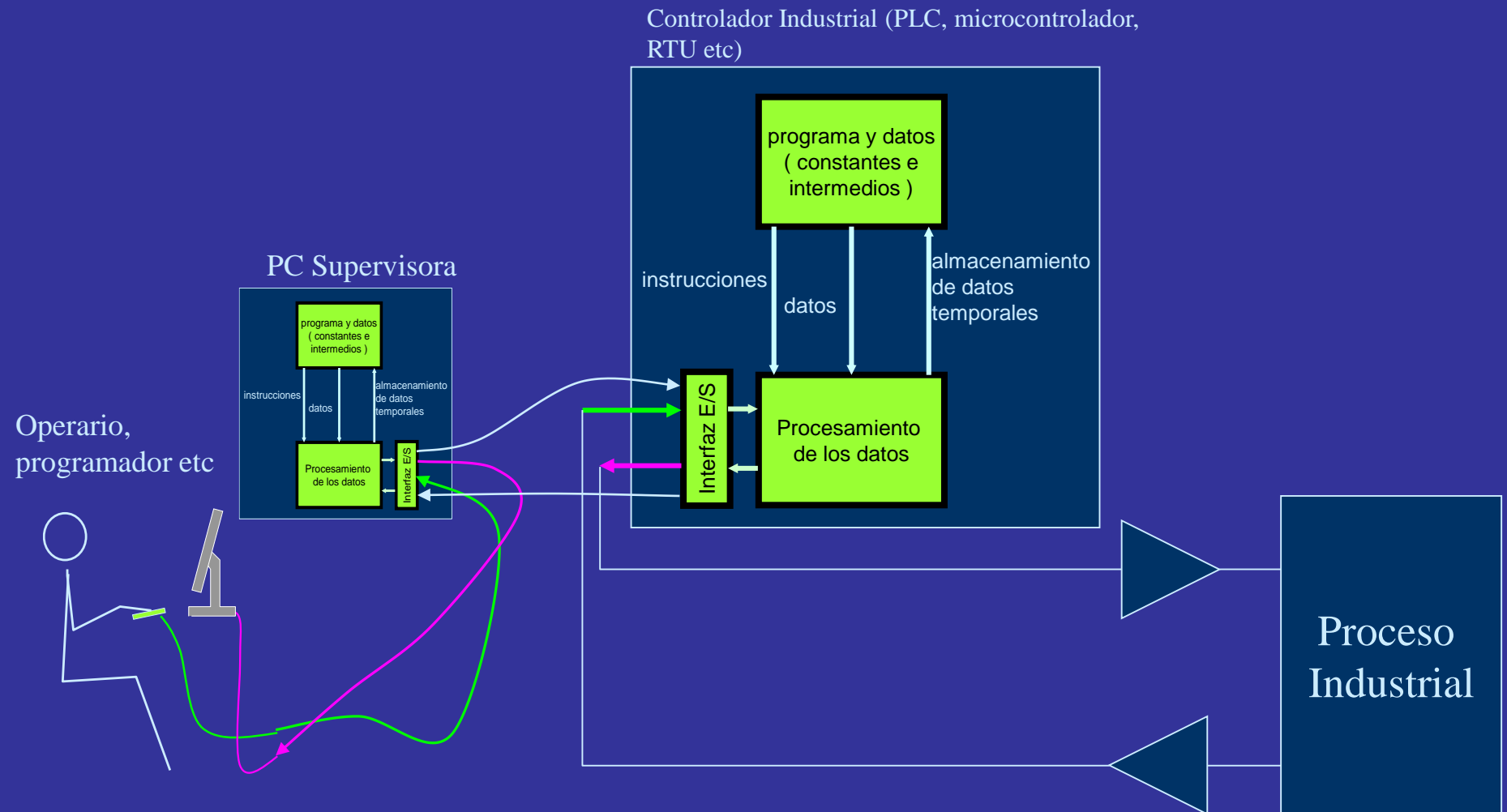
Sistema de Cómputo Programable (2)



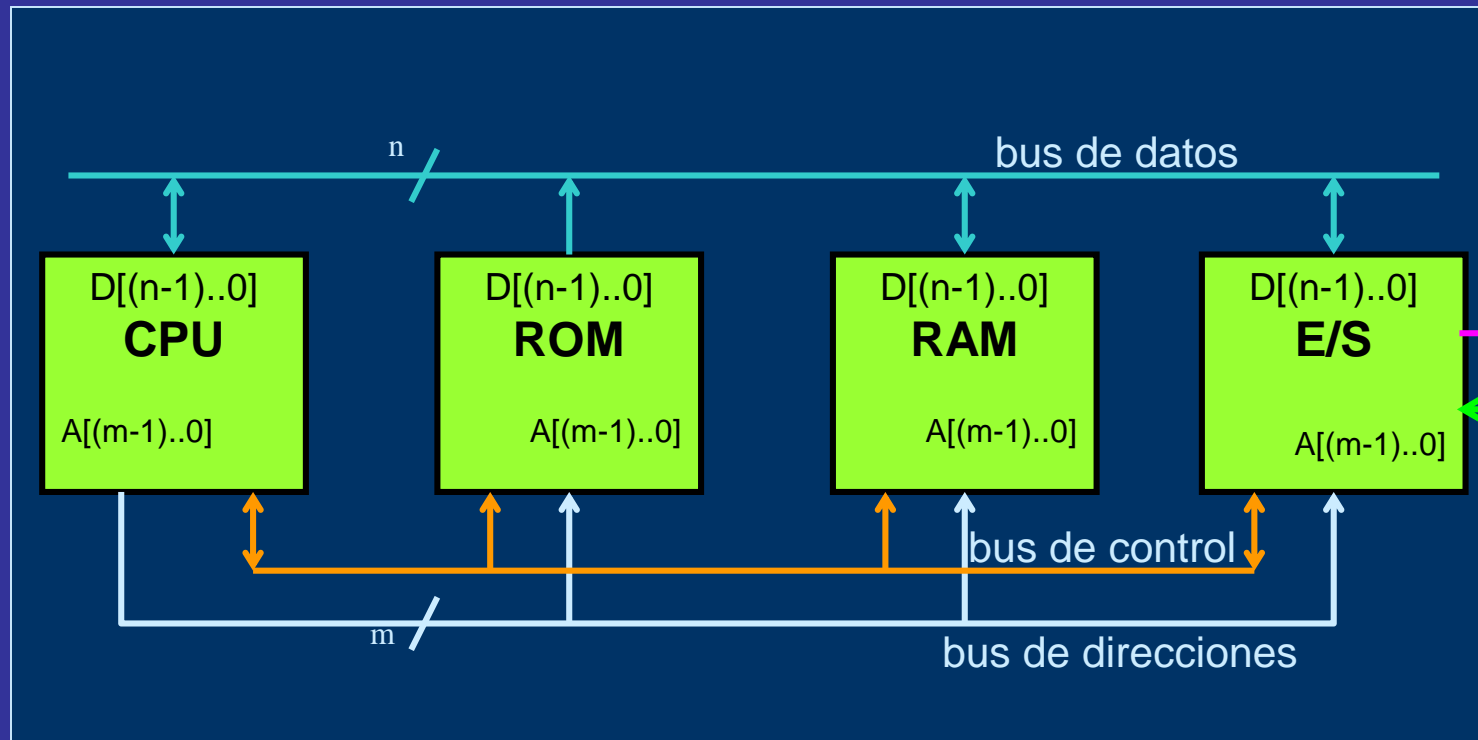
Sistema de Cómputo Programable (3)



Sistema de Cómputo Programable (4)



Arquitectura Von Neumann



CPU: Unidad de Control y Procesamiento

RAM: Random Access Memory

ROM: Read Only Memory (EPROM, FLASH, PROM etc)

E/S: Interfaces de Entrada-Salida “mapeadas” como RAM

Data Bus: n bits

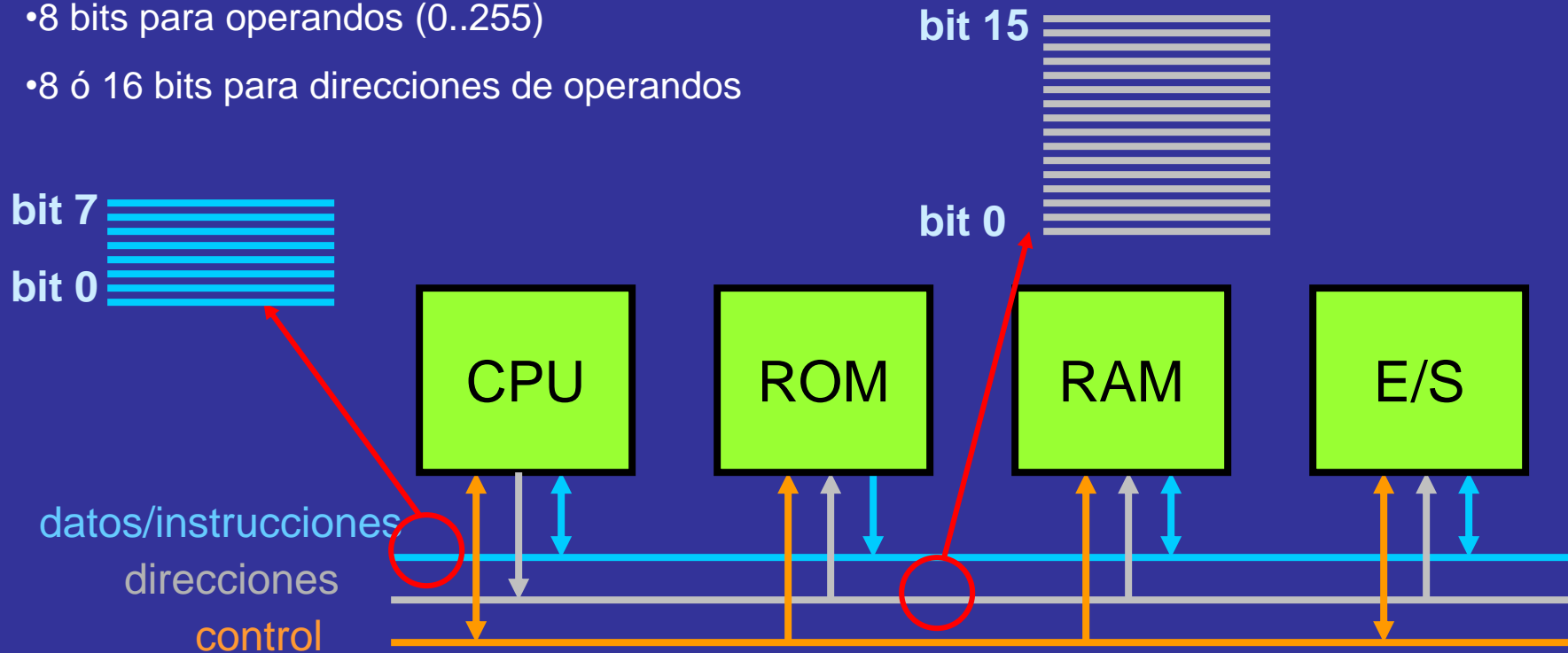
Address Bus: m bits

BUS DE DATOS de 8 bits

- 8 bits para código de instrucción (hasta 256)
- 8 bits para operandos (0..255)
- 8 ó 16 bits para direcciones de operandos

BUS DE DIRECCIONES de 16 bits

- Hasta 65536 posiciones RAM-ROM-ES

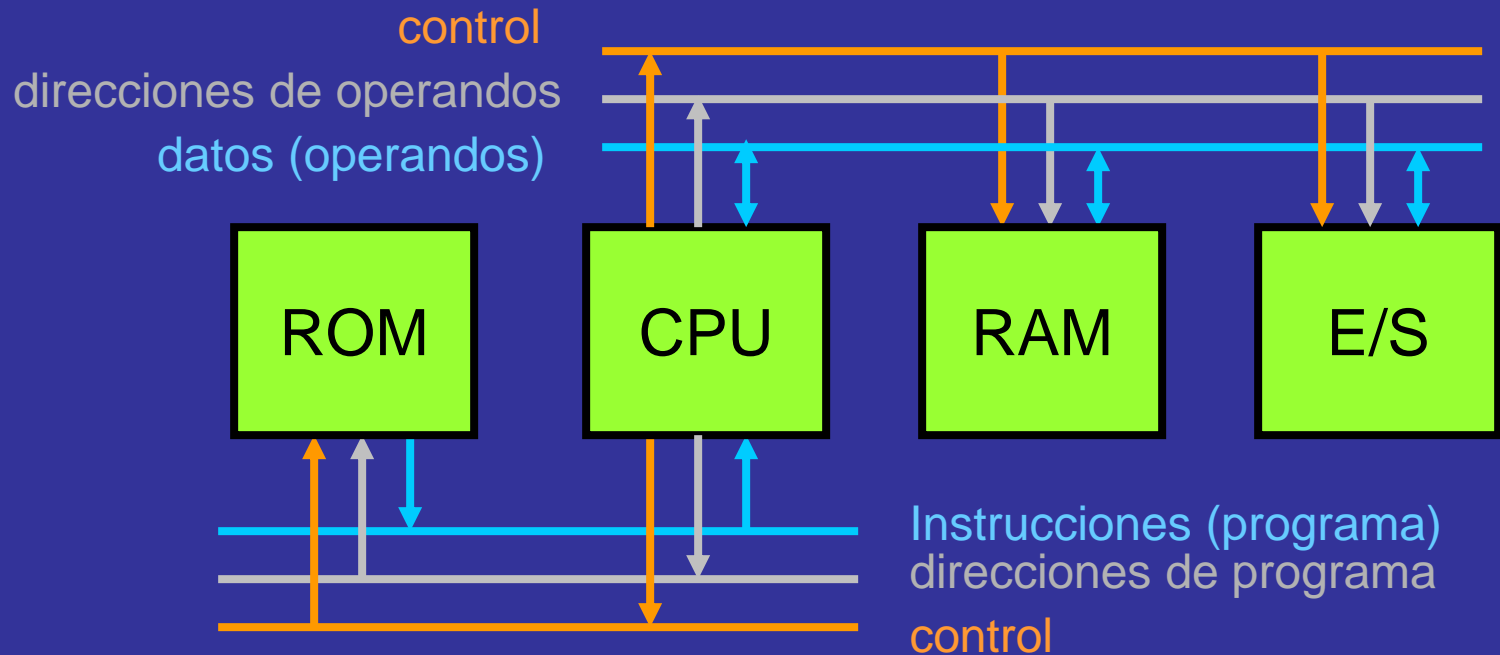


Microcontroladores/Microprocesadores Von Neumann

- Más de un ciclo de reloj por instrucción
- Repertorio de instrucciones complejo (CISC)
- Normalmente mayor capacidad de direccionamiento y de pila

Arquitectura Harvard

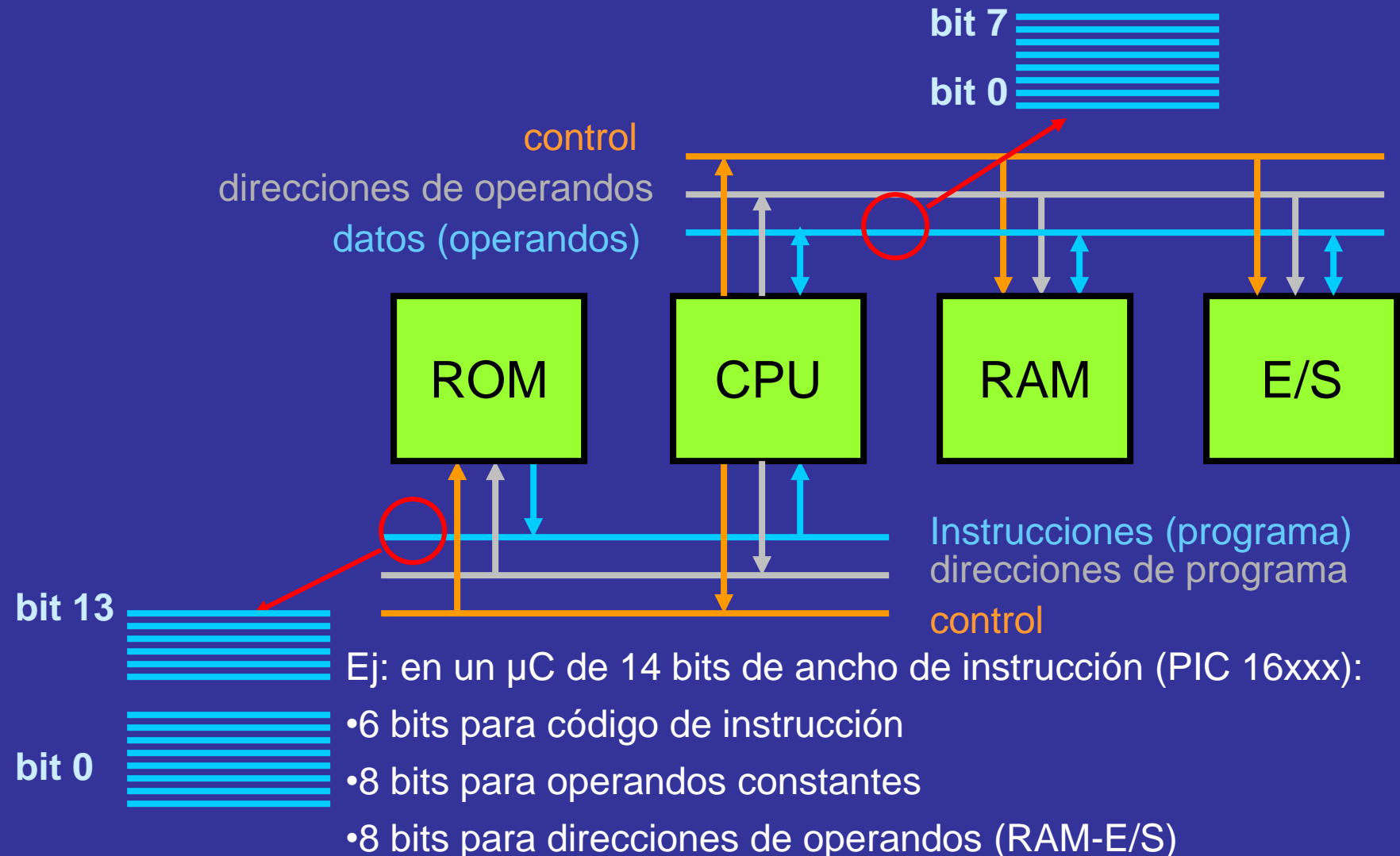
(μ C de Microchip, Atmel etc)

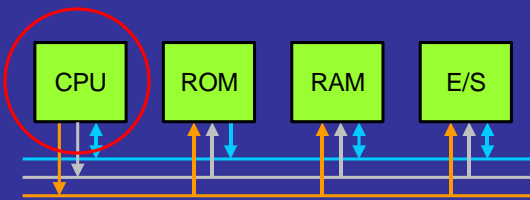


- Instrucciones en un ciclo
- Repertorio de instrucciones reducido (RISC)

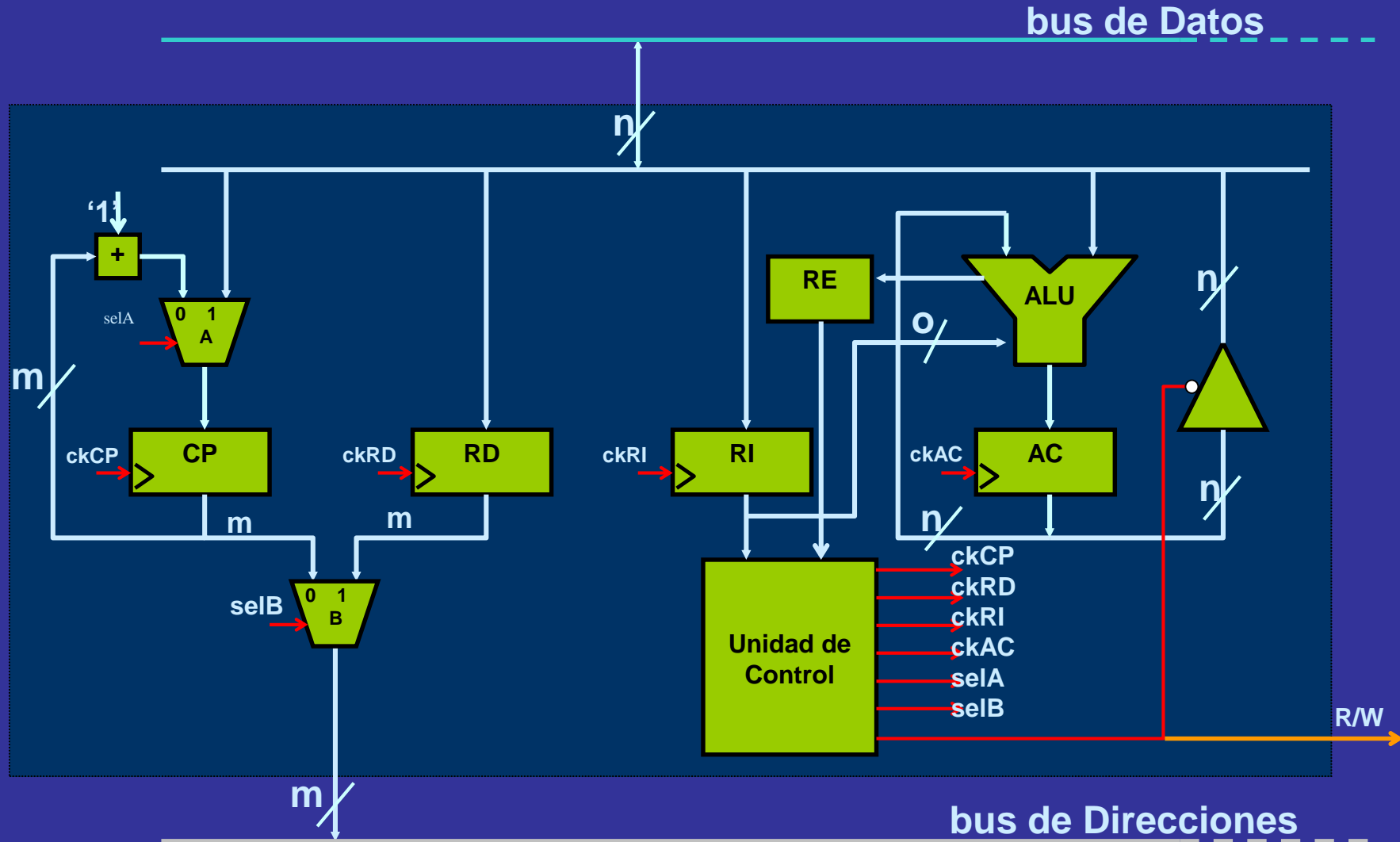
Arquitectura Harvard

(μ C de Microchip, Atmel)

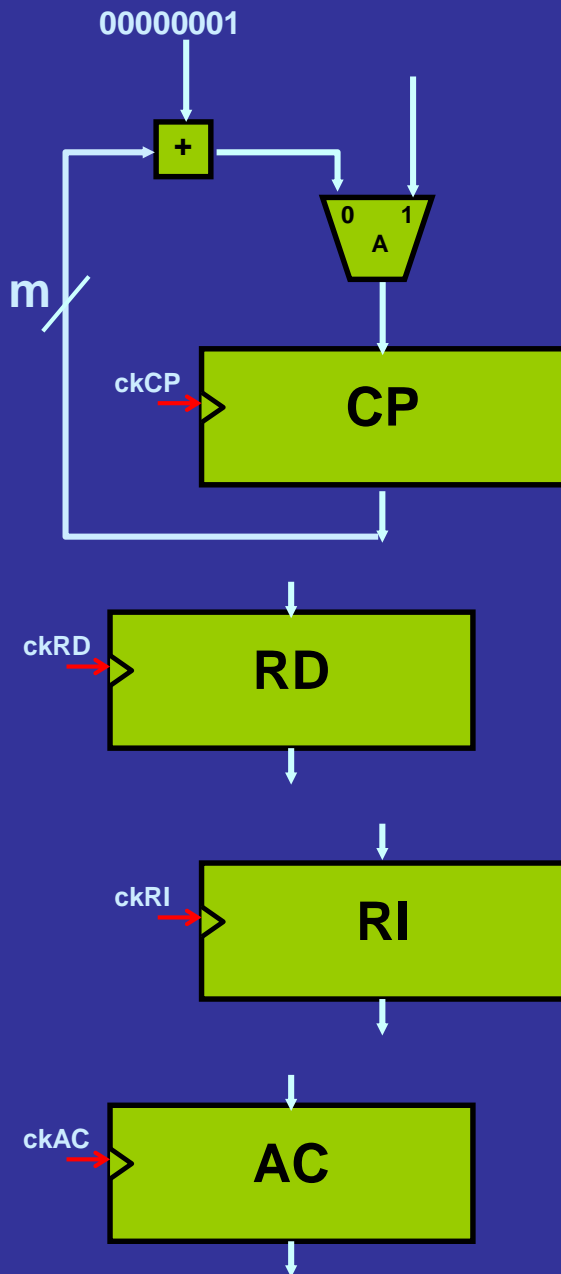




CPU tipo Von Neumann



Registros básicos de la CPU Von Neumann



CP: Contador de programa.

- a) (Mux en 0) Incrementa de 1 en 1
- b) (Mux en 1) Carga valor (salto de programa)

RD: Registro de Direcciones.

Carga dirección de operando, desde el bus de datos, para presentar en el bus de direcciones.

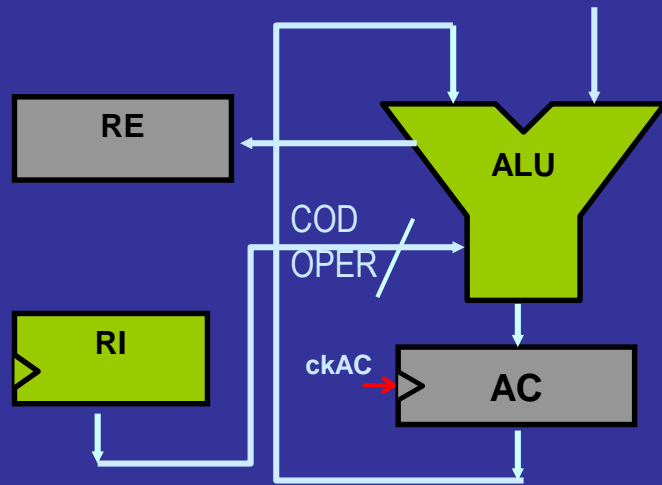
RI: Registro de Instrucciones.

Carga código de operación para ser decodificada en la Unidad de Control y la ALU.

AC: Acumulador

Almacena los resultados de la ALU

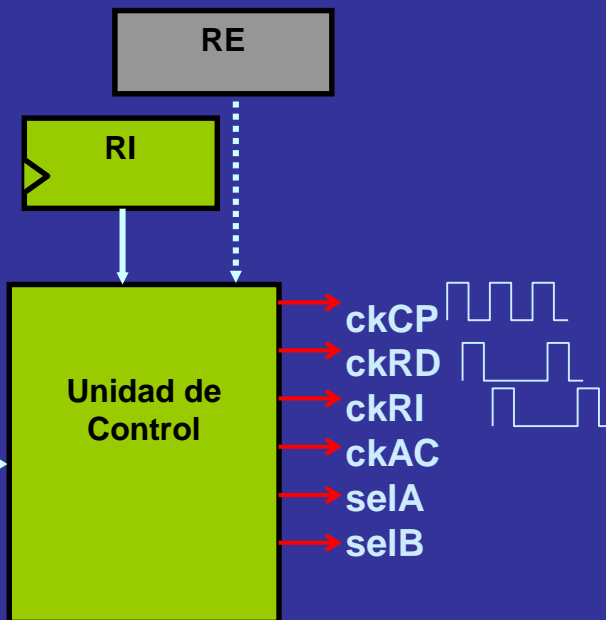
ALU y Unidad de Control



ALU: Unidad Aritmético-Lógica

Realiza la operación indicada por el código de operación (proveniente del RI), entre el resultado almacenado en el acumulador AC y el dato que ingresa por la rama derecha. El resultado solamente es tomado por el acumulador AC cuando se da el clock ckAC.

El resultado de la operación (Cero, Positivo, Negativo, Acarreo, Desborde etc) activa los bits correspondientes en el Registro de Estado RE



UC: Unidad de Control

Es un sistema secuencial que, en función del código de operación (proveniente del RI) - y en ocasiones de los bits del registro de estado RE - genera los pulsos de reloj ckCP, ckRD, ckRI, ckAC y los niveles para los multiplexores, selA y selB.

Su velocidad depende de la del reloj de sistema (generador de pulsos)

Ejemplo: Un juego de instrucciones (1)

c o d	Mnemónico		Sintaxis	Explicación
0	LEEC	2	LEEC k	Carga constante k en el acumulador AC. El segundo byte es k. $AC \leftarrow k$
1	ANDC	2	ANDC k	AND bit a bit entre el contenido de AC y la constante k. El segundo byte es k. El resultado se carga en AC. $AC \leftarrow AC \text{ and } k$
2	ORC	2	ORC k	OR bit a bit entre el contenido de AC y la constante k. El segundo byte es k. $AC \leftarrow AC \text{ or } k$
3	RESTAC	2	RESTAC k	Resta al contenido de AC la constante k. El segundo byte es k. $AC \leftarrow AC - k$
4	SUMAC	2	SUMAC k	Suma aritmética entre el contenido de AC y la constante k. El segundo byte es k. $AC \leftarrow AC + k$
5	SALTA	2	SALTA D	Salto incondicional del programa. D es la dirección donde continúa la ejecución del programa. $CP \leftarrow D$
6	SALTAZ	2	SALTAZ D	Salto de programa si el resultado de la última operación de la ALU es cero (bit Z del RE activado). D es la dirección donde continúa la ejecución del programa. $CP \leftarrow D \text{ si } Z='1'$
7	SALTAN	2	SALTAN D	Salto de programa si el resultado de la última operación de la ALU es negativo (bit N del RE activado). D es la dirección donde continúa la ejecución del programa. $CP \leftarrow D \text{ si } N='1'$

juego de instrucciones (2)

c o d	Mnemónico		Sintaxis	Explicación
8	LEE	2	LEE [X]	Carga variable X en el acumulador AC. El segundo byte es la dirección de N. $AC \leftarrow [X]$
9	AND	2	AND [X]	AND bit a bit entre AC y la variable X. El segundo byte es la dirección de X. El resultado se carga en AC. $AC \leftarrow AC \text{ and } [X]$
A	OR	2	OR [X]	OR bit a bit entre AC y la variable X. El segundo byte es la dirección de X. El resultado se carga en AC. $AC \leftarrow AC \text{ or } [X]$
B	RESTA	2	RESTA [X]	Resta al contenido de AC la variable X. El segundo byte es la dirección de X. El resultado se carga en AC. $AC \leftarrow AC - [X]$
C	SUMA	2	SUMA [X]	Suma aritmética entre el contenido de AC y la variable N. El segundo byte es la dirección de N. El resultado queda en AC. $AC \leftarrow AC + [N]$
D	NOT	1	NOT	Complementa a '1' el acumulador $AC \leftarrow \text{not } AC$
E	ESCRIBE	2	ESCRIBE [X]	Escribe en variable N el valor de AC. $[N] \leftarrow AC$
F	NOP	1	NOP	No realiza operación aritmético-lógica. Sólo se incrementa CP

Una instrucción en lenguaje de alto nivel (Basic, Pascal, C etc) tal como:

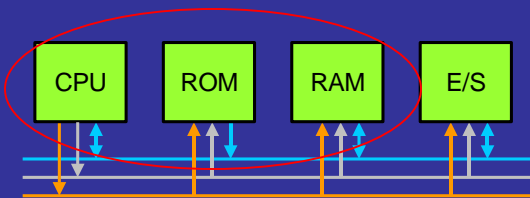
Y = 6 + X

ó

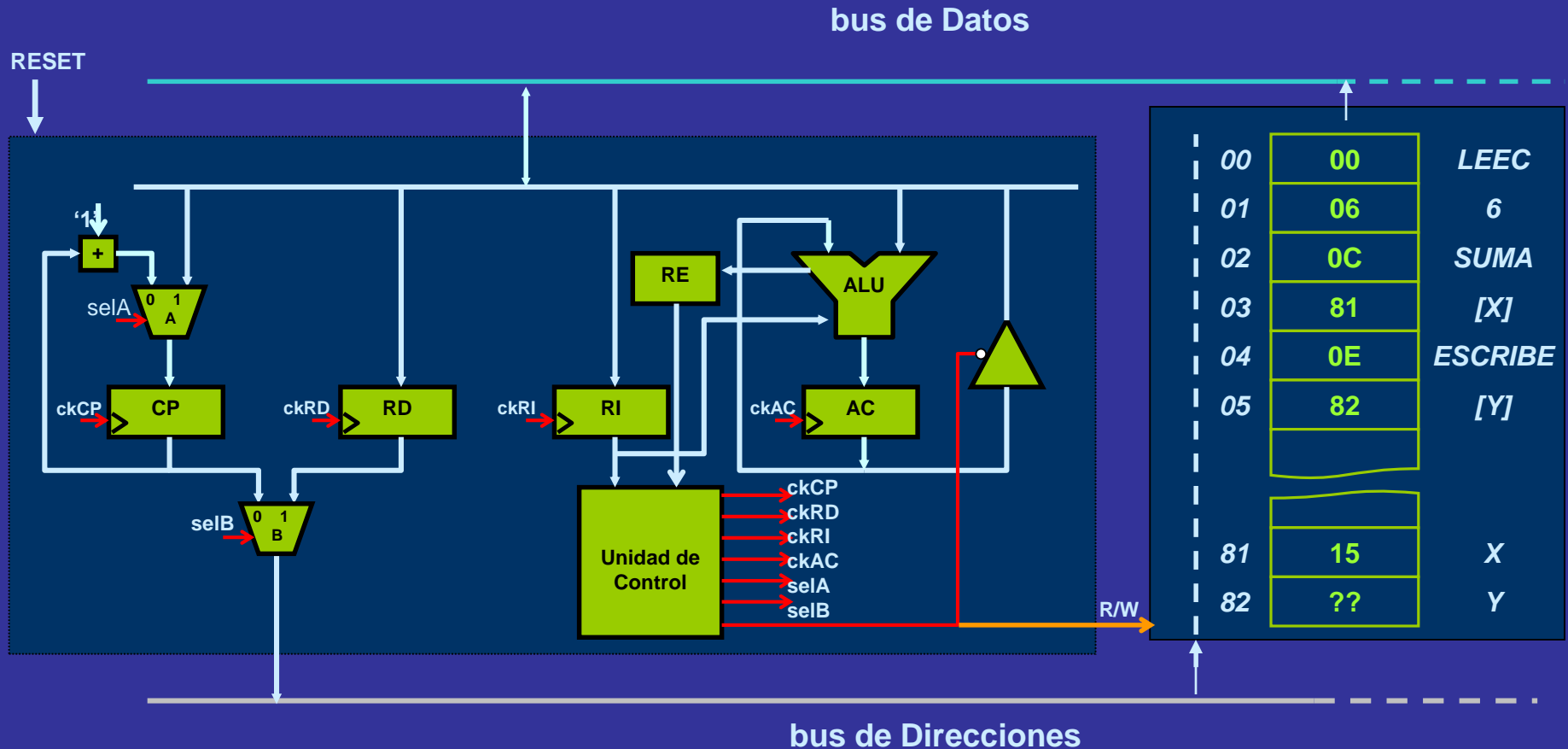
Y = X + 6

Se realiza en el microprocesador como:

LEEC	6	<i>lee constante 6</i>
SUMA	[X]	<i>suma variable X</i>
ESCRIBE	[Y]	<i>escribe en Y</i>
ó		
LEE	[X]	<i>lee variable X</i>
SUMAC	6	<i>suma constante 6</i>
ESCRIBE	[Y]	<i>escribe en Y</i>



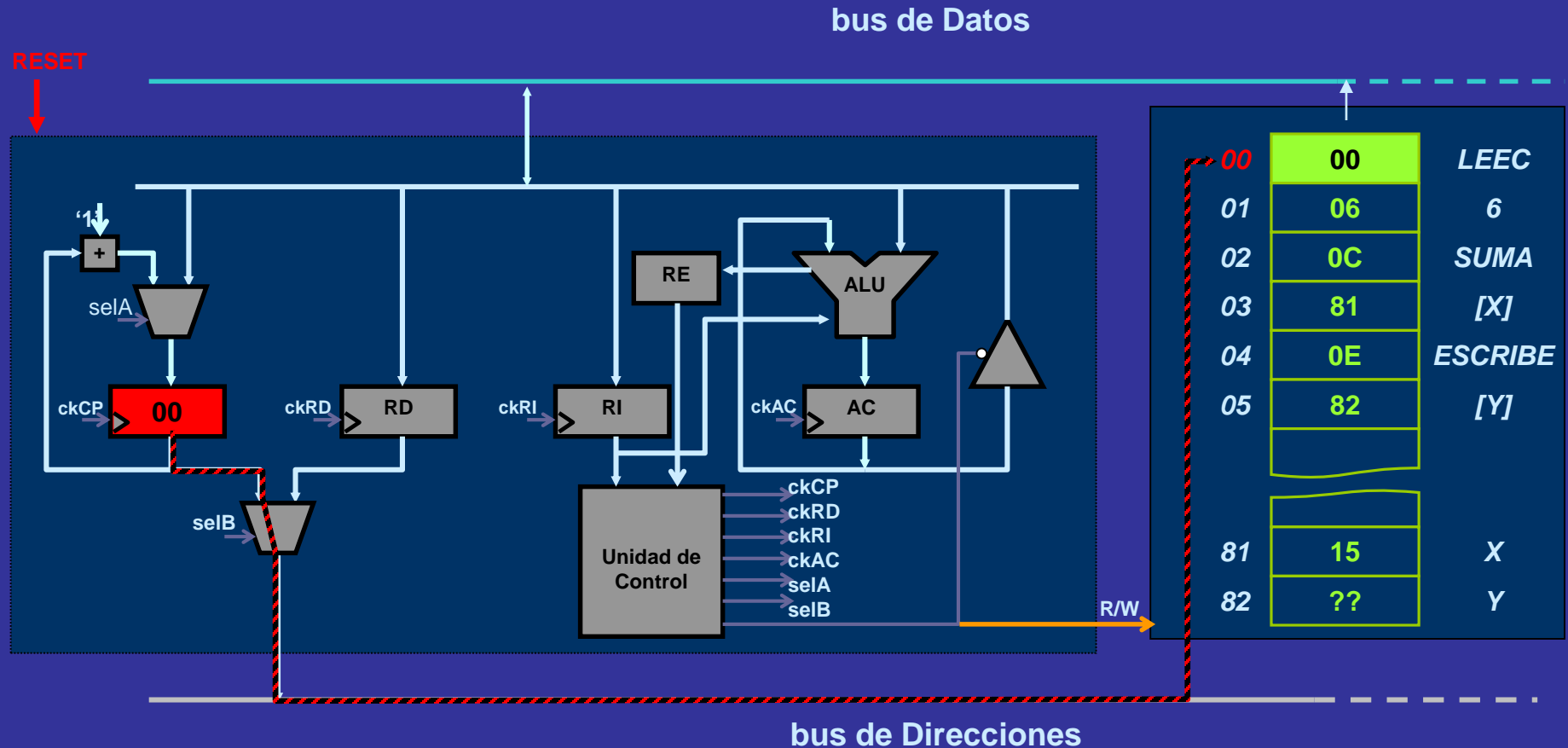
Operación a nivel registros



00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
LEEC	ANDC	ORC	REST AC	SUMA C	SALT A	SALT AZ	SALT AN	LEE	AND	OR	REST A	SUMA	NOT	ESCR IBE	NOP

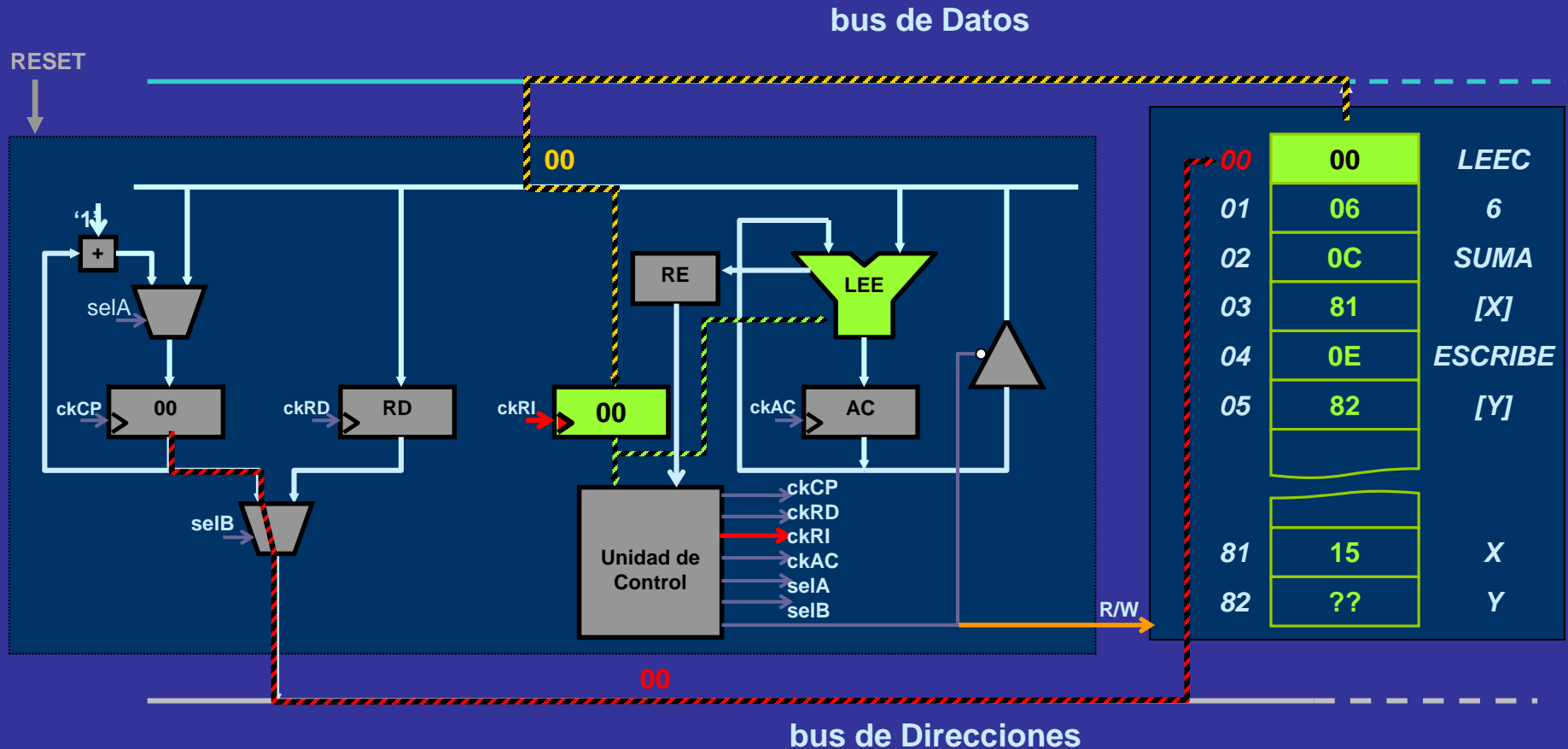
1 - Un reset pone al Contador de programa CP en cero (inicio de programa).

La señal RW está en '1', por lo que la ROM/RAM es leída



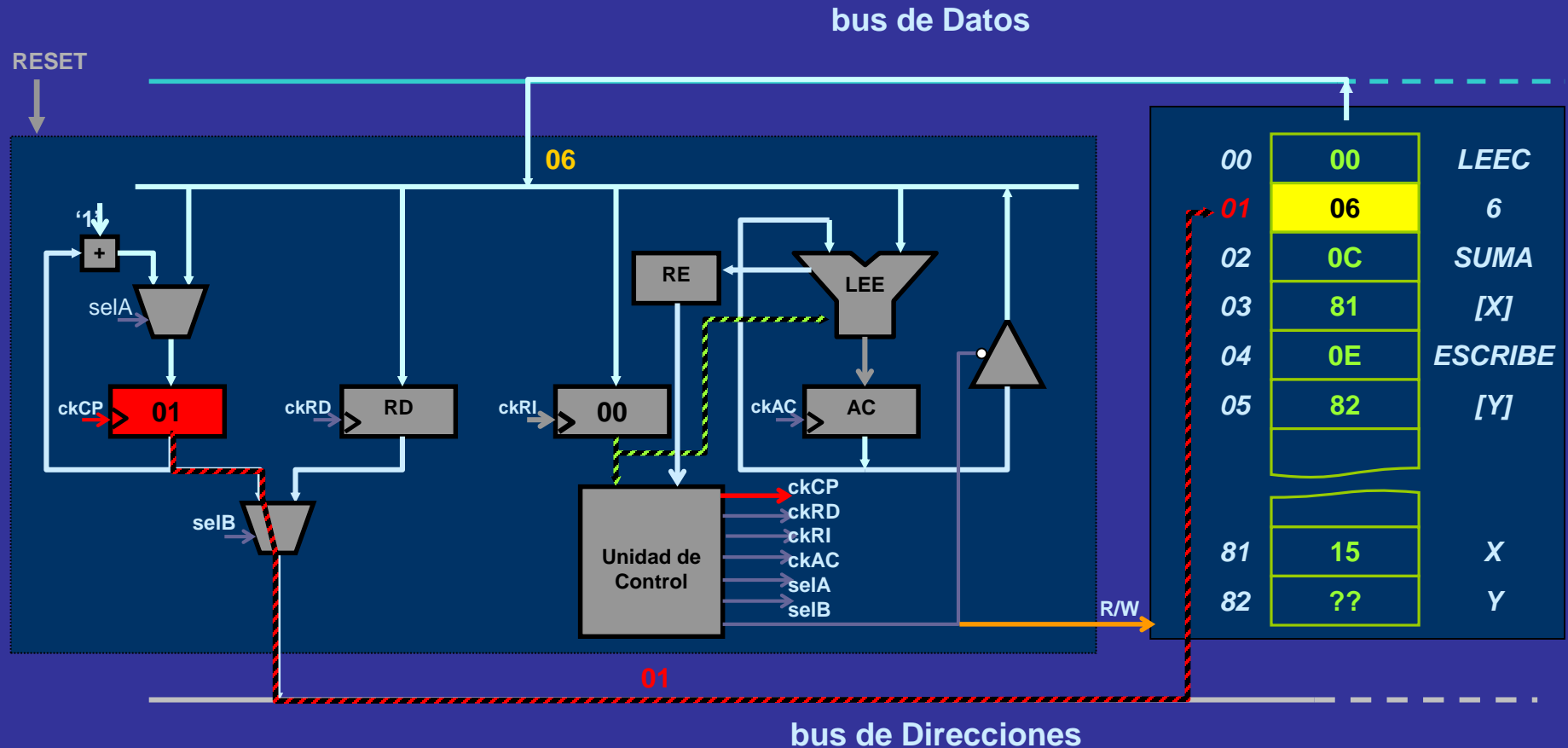
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
LEEC	ANDC	ORC	REST AC	SUMA C	SALT A	SALT AZ	SALT AN	LEE	AND	OR	REST A	SUMA	NOT	ESCR IBE	NOP

2 – El valor 00 (LEERC) es almacenado en el Registro de Instrucciones RI.
La Unidad de Control decodifica la instrucción, y la ALU se pone en modo LEER (deja pasar el dato presente en su rama derecha)



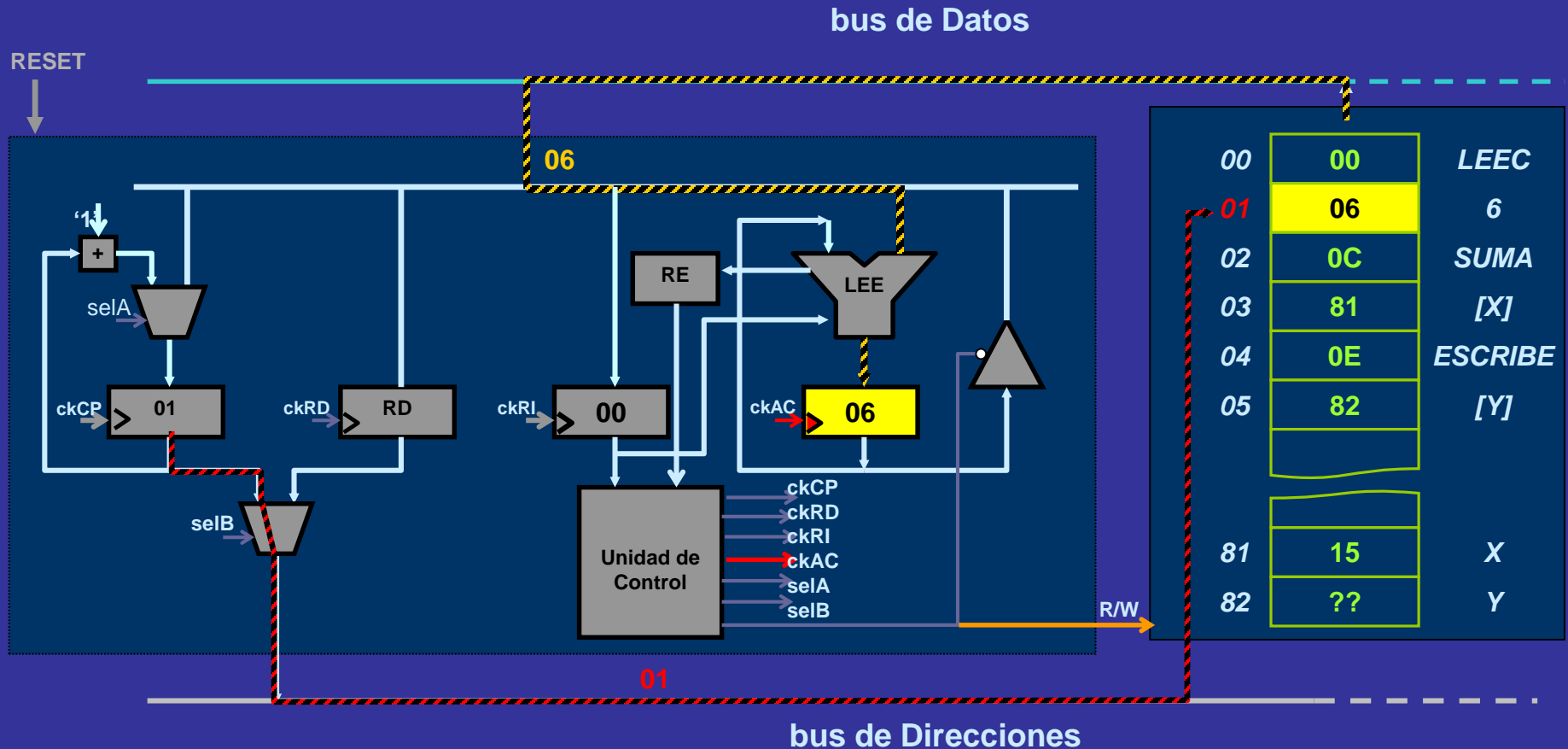
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
LEEC	ANDC	ORC	REST AC	SUMA C	SALTA	SALTA Z	SALT AN	LEE	AND	OR	REST A	SUMA	NOT	ESCR IBE	NOP

3 – Al activar ckCP el contador de programa CP se incrementa, por lo que ahora se direcciona la posición '01' de la memoria
En esa posición está la CONSTANTE a ser leída, en este caso '06'



00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
LEEC	ANDC	ORC	REST AC	SUMA C	SALT A	SALT AZ	SALT AN	LEE	AND	OR	REST A	SUMA	NOT	ESCR IBE	NOP

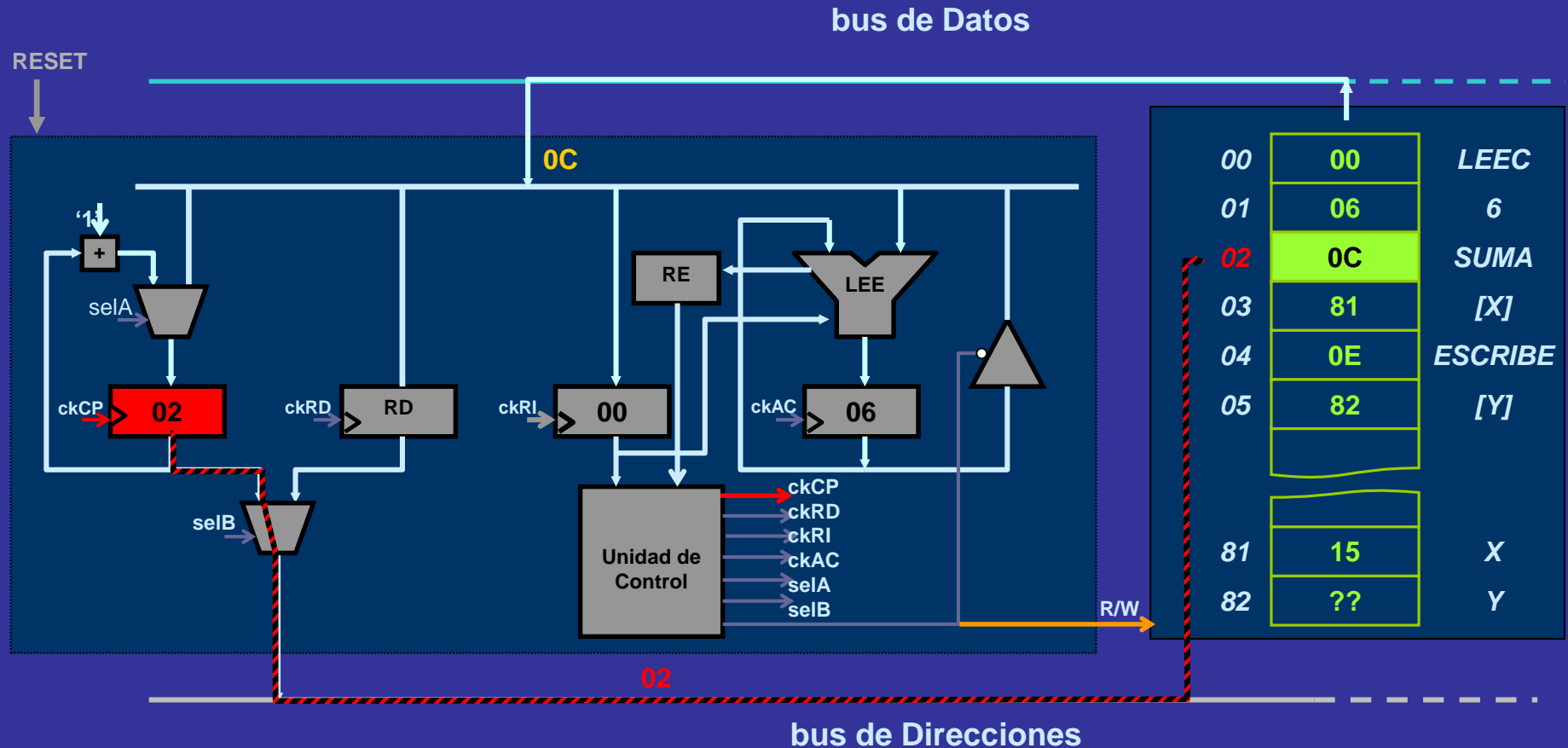
4 – Dicha constante se almacena en el acumulador AC
 Una vez almacenada también está presente en la rama izquierda de la ALU
 y en la entrada del buffer Triestate



00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
LEEC	ANDC	ORC	REST AC	SUMA C	SALT A	SALT AZ	SALT AN	LEE	AND	OR	REST A	SUMA	NOT	ESCR IBE	NOP

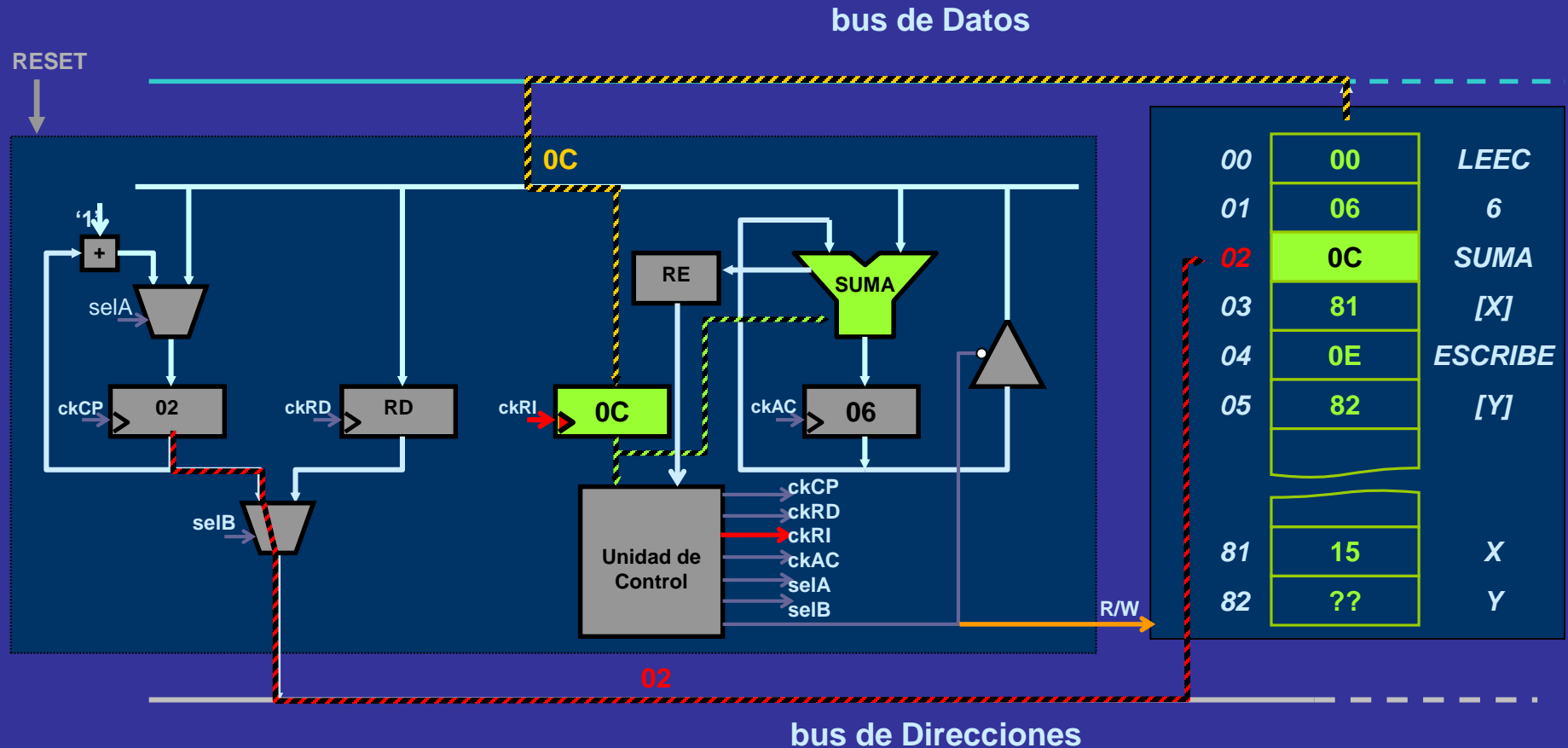
5 – El contador de programa CP se incrementa, apuntando ahora a la posición '02' de la memoria.

En esa posición está el CÓDIGO de la siguiente instrucción, en este caso '0C' (Sumar variable)



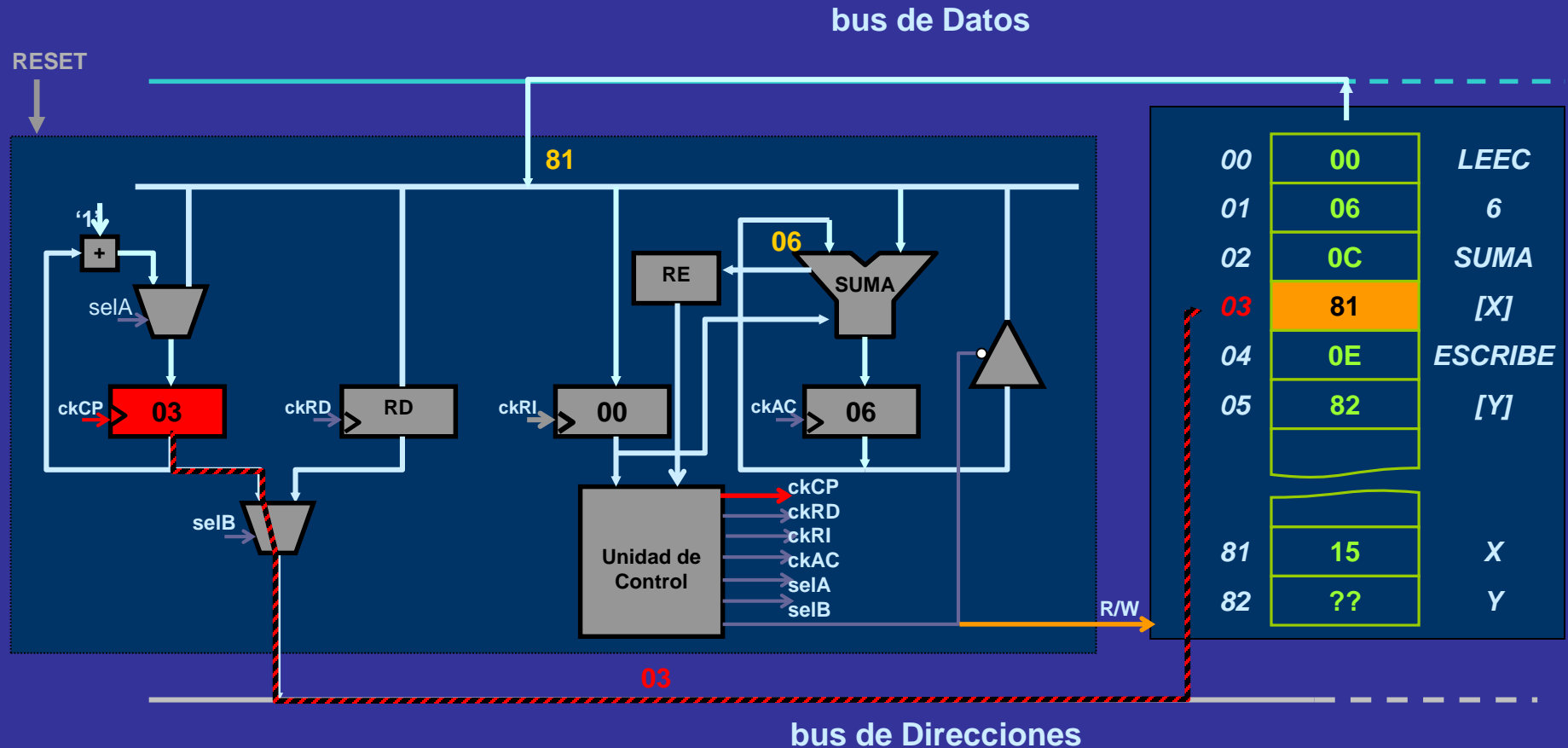
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
LEEC	ANDC	ORC	REST AC	SUMA C	SALT A	SALT AZ	SALT AN	LEE	AND	OR	REST A	SUMA	NOT	ESCR IBE	NOP

La Unidad de Control decodifica la instrucción, y la ALU se pone en modo SUMAR (suma aritmética de ambas ramas)



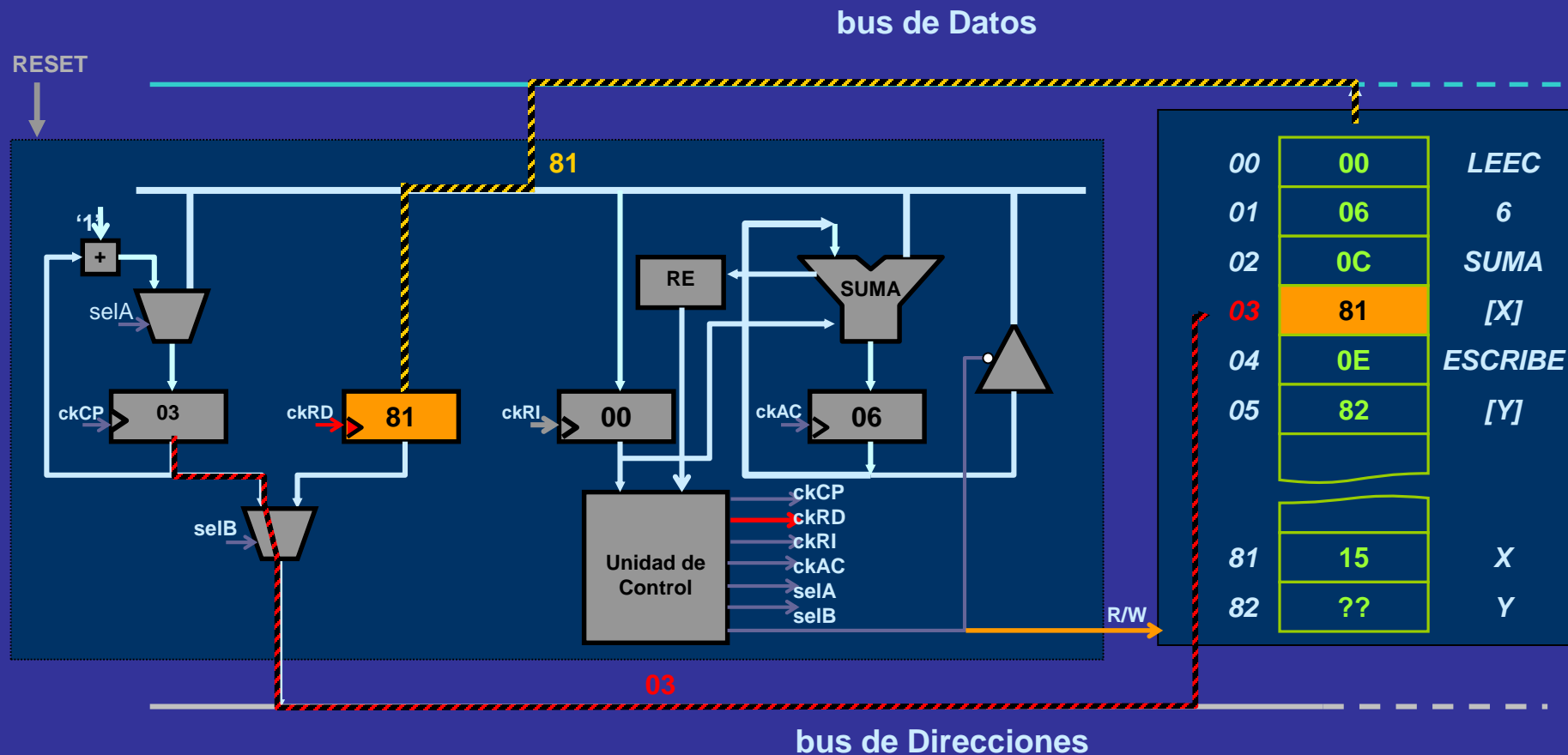
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
LEEC	ANDC	ORC	REST AC	SUMA C	SALT A	SALT AZ	SALT AN	LEE	AND	OR	REST A	SUMA	NOT	ESCR IBE	NOP

5 – El contador de programa CP se incrementa, por lo que ahora se direcciona la posición '03' de la memoria. Como la instrucción SUMA es sumar variable, lo que sigue no es el valor sino la DIRECCIÓN en la que se encuentra la variable a sumar, en este caso '81'



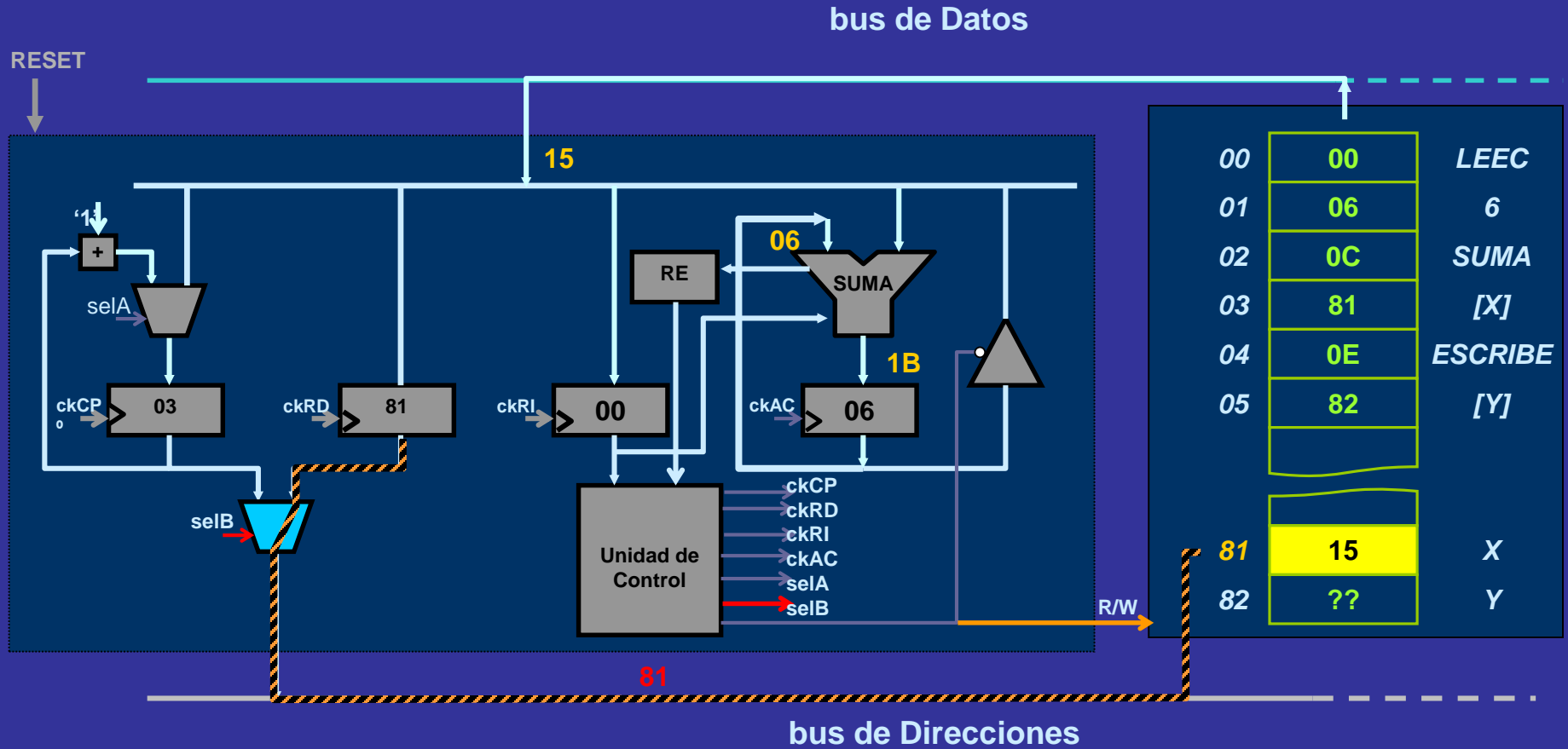
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
LEEC	ANDC	ORC	REST AC	SUMA C	SALT A	SALT AZ	SALT AN	LEE	AND	OR	REST A	SUMA	NOT	ESCR IBE	NOP

6 – Dicha dirección se almacena en el Registro de Direcciones RD



00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
LEEC	ANDC	ORC	REST AC	SUMA C	SALT A	SALT AZ	SALT AN	LEE	AND	OR	REST A	SUMA	NOT	ESCR IBE	NOP

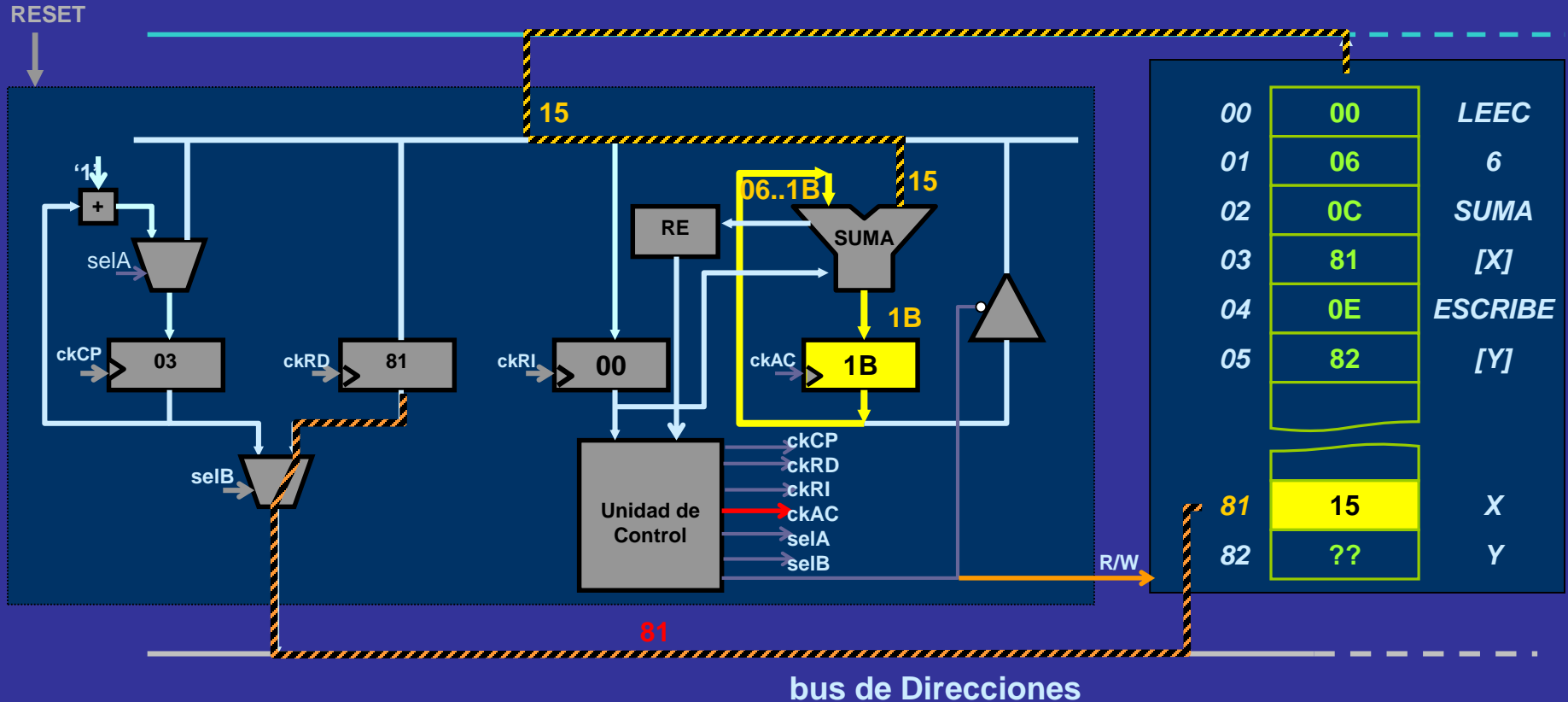
7 – Moviendo selB el multiplexor inferior deja pasar el valor de RD (81) al bus de direcciones. Con este mecanismo, el '81' se transfirió desde el bus de datos al bus de direcciones para apuntar a la variable X.



00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
LEEC	ANDC	ORC	REST AC	SUMA C	SALT A	SALT AZ	SALT AN	LEE	AND	OR	REST A	SUMA	NOT	ESCR IBE	NOP

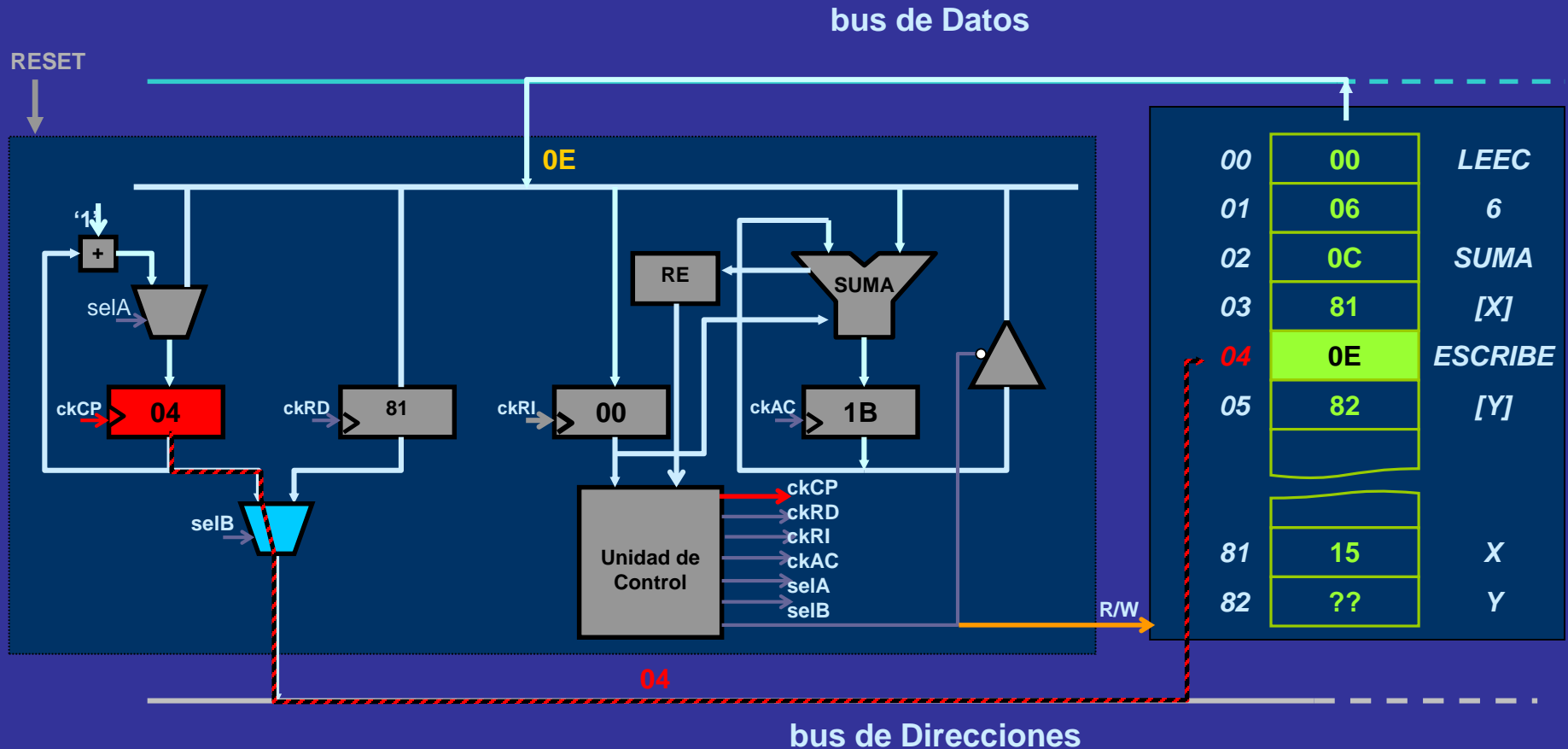
8 – La Memoria presenta el valor almacenado en dicha posición, en este caso '15' (hexadecimal). La ALU, que está en modo suma, presenta el resultado de la misma a la entrada de AC, en este caso $06+15=1B$ (hexadecimal) El resultado se almacena en el acumulador, y también está presente en la rama izquierda de la ALU y en la entrada del buffer Triestate

bus de Datos



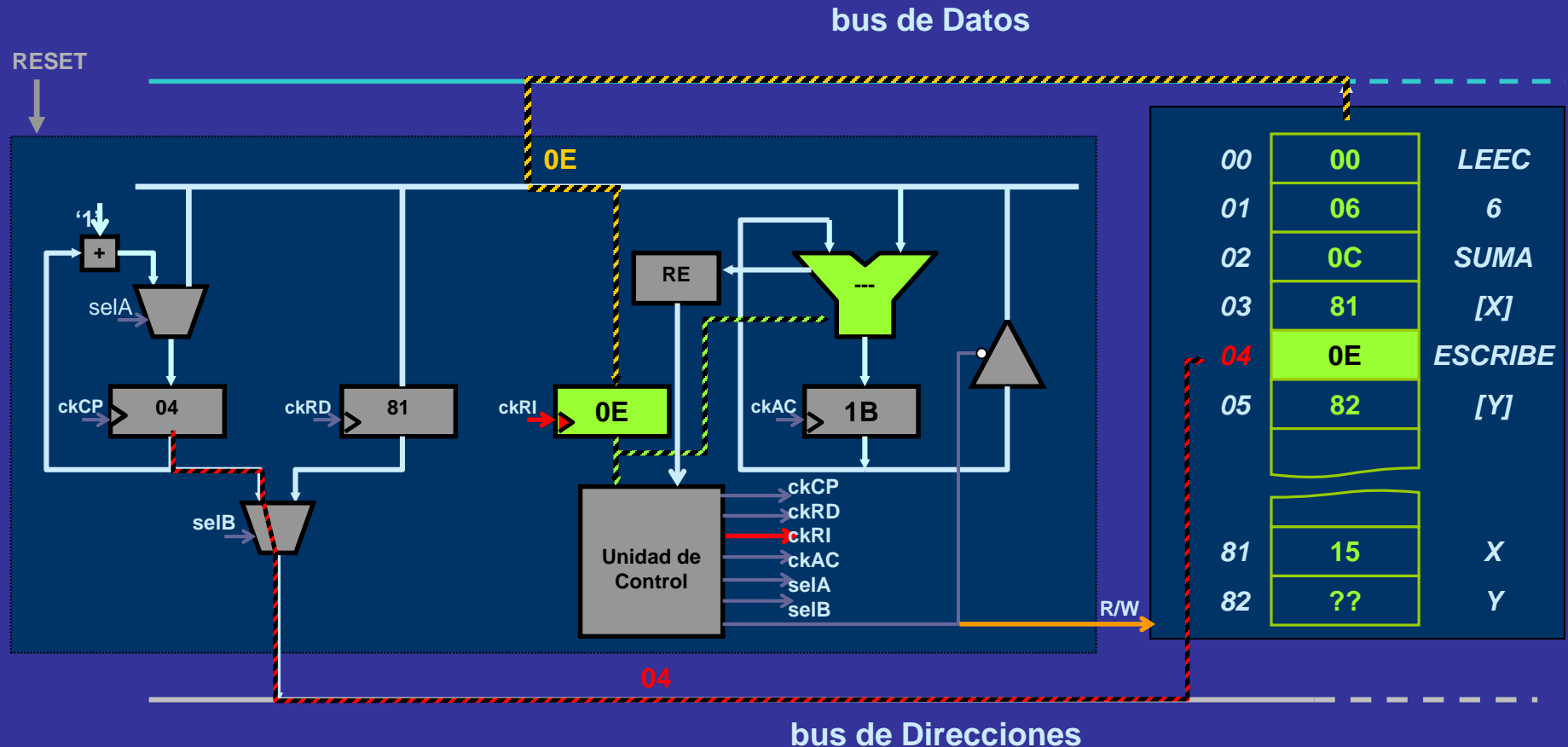
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
LEEC	ANDC	ORC	REST AC	SUMA C	SALT A	SALT AZ	SALT AN	LEE	AND	OR	REST A	SUMA	NOT	ESCR IBE	NOP

9 – El contador de programa CP se incrementa y el selector selB vuelve a dejar pasar el CP al bus de direcciones apuntando a la posición '04' de la memoria. En esa posición está el CÓDIGO de la siguiente instrucción, en este caso '0E' (Escribir variable)



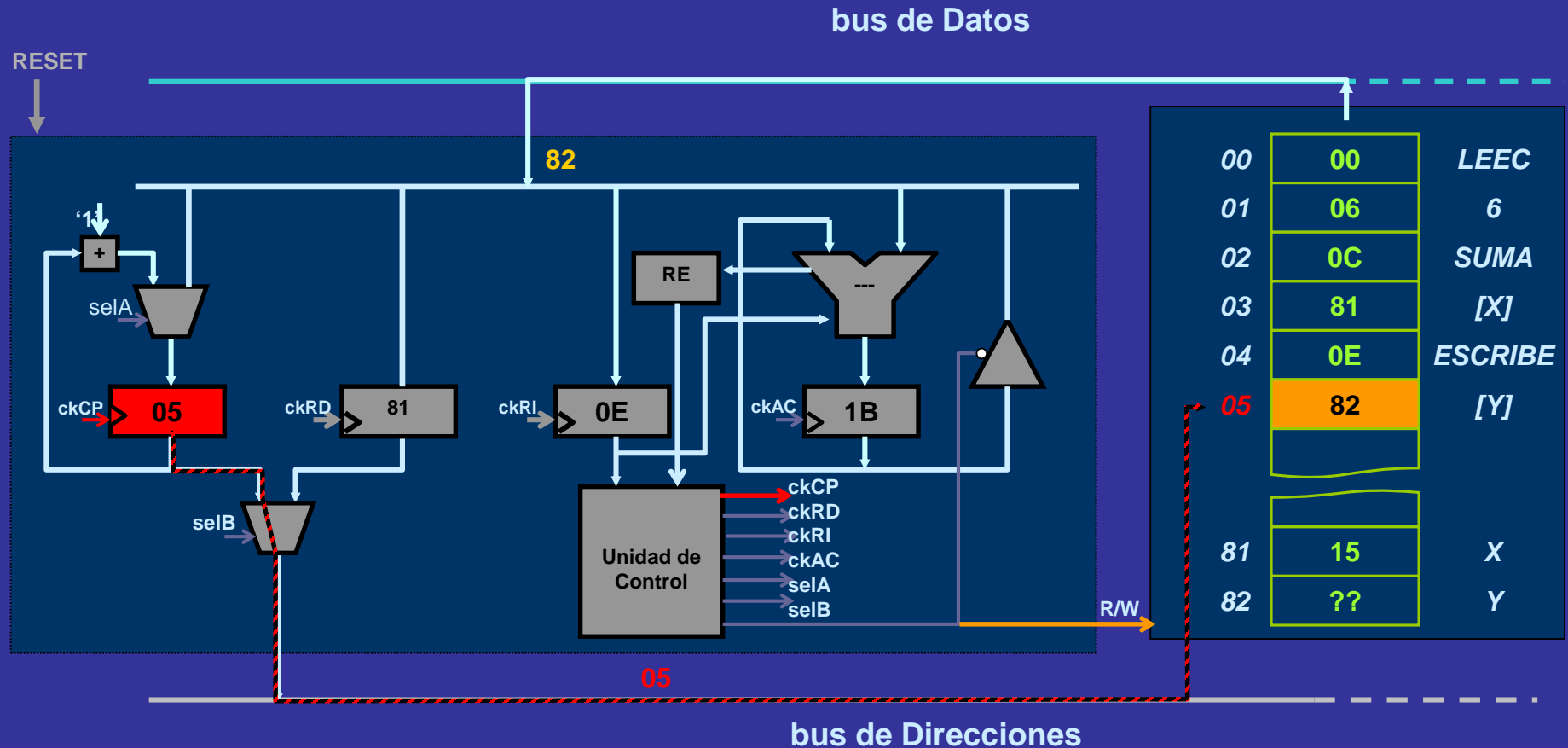
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
LEEC	ANDC	ORC	REST AC	SUMA C	SALT A	SALT AZ	SALT AN	LEE	AND	OR	REST A	SUMA	NOT	ESCR IBE	NOP

10 – Este valor es almacenado en el Registro de Instrucciones RI.
La Unidad de Control decodifica la instrucción. La operación que realice la ALU es indiferente.



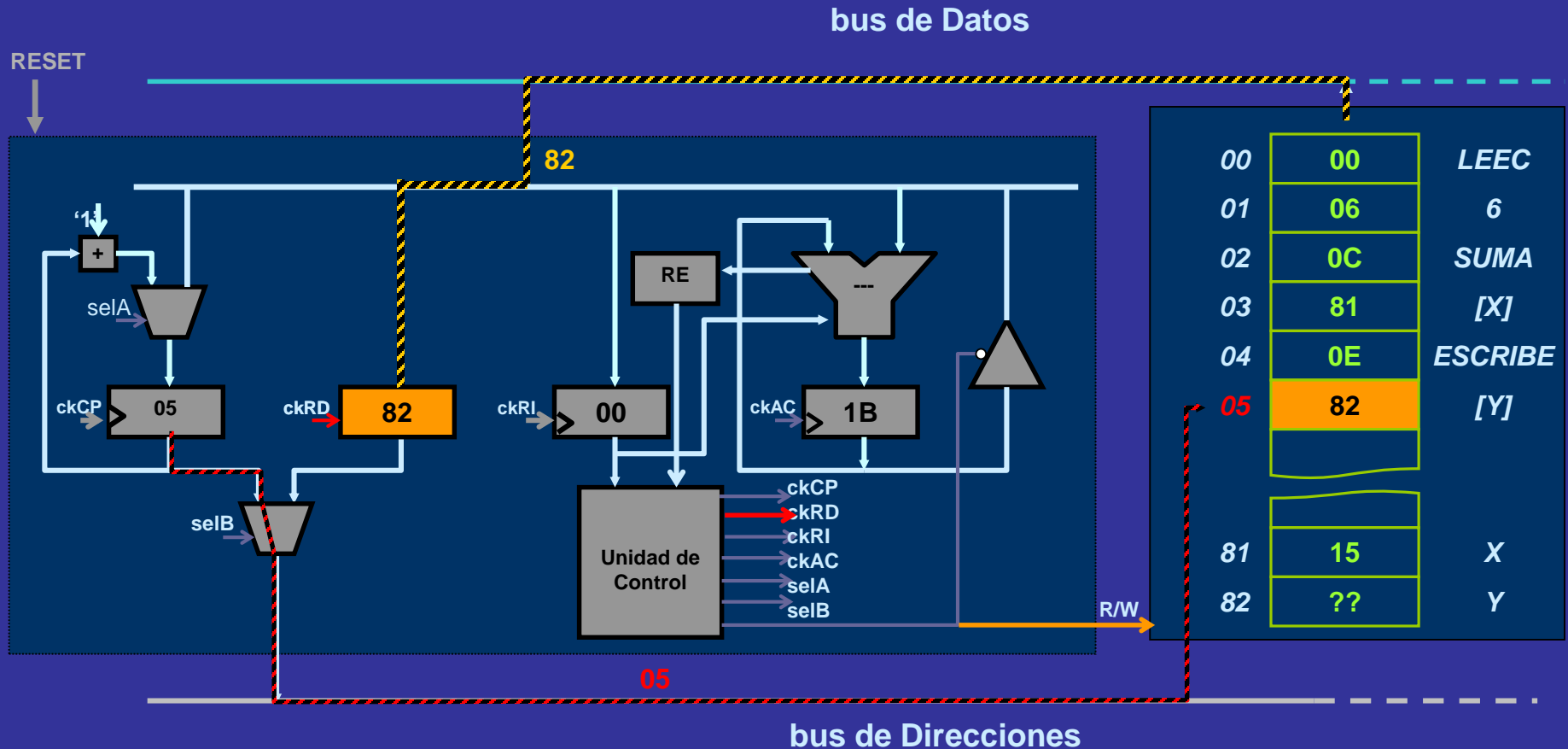
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
LEEC	ANDC	ORC	REST AC	SUMA C	SALT A	SALT AZ	SALT AN	LEE	AND	OR	REST A	SUMA	NOT	ESCR IBE	NOP

11 – El contador de programa CP se incrementa, por lo que ahora se direcciona la posición '05' de la memoria. Esa posición (82) es la DIRECCIÓN de Y, donde debe escribirse lo que contiene el acumulador



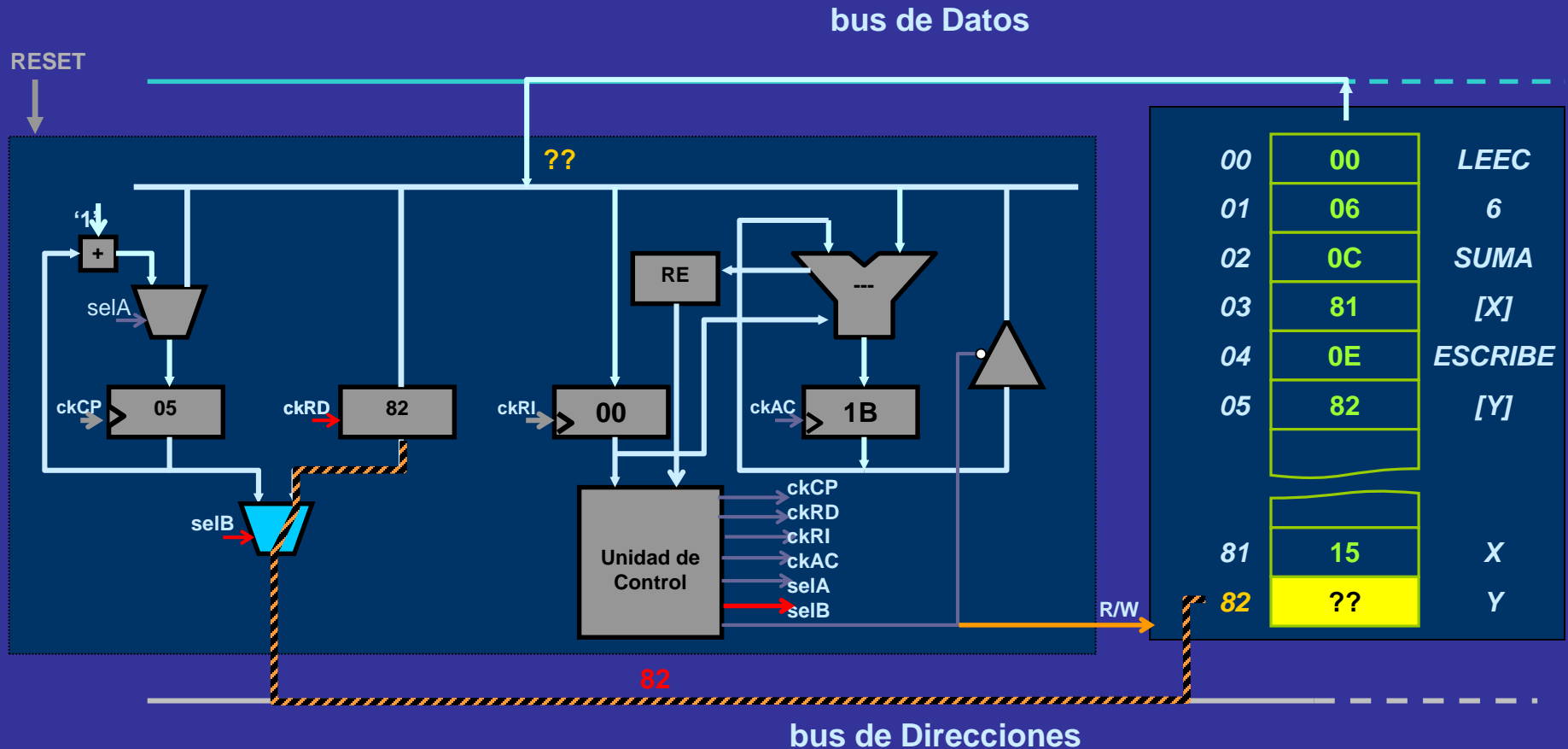
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
LEEC	ANDC	ORC	REST AC	SUMA C	SALT A	SALT AZ	SALT AN	LEE	AND	OR	REST A	SUMA	NOT	ESCR IBE	NOP

12 – Dicha dirección se almacena en el Registro de Direcciones RD para realizar el mismo mecanismo de direccionamiento utilizado en la lectura de X.



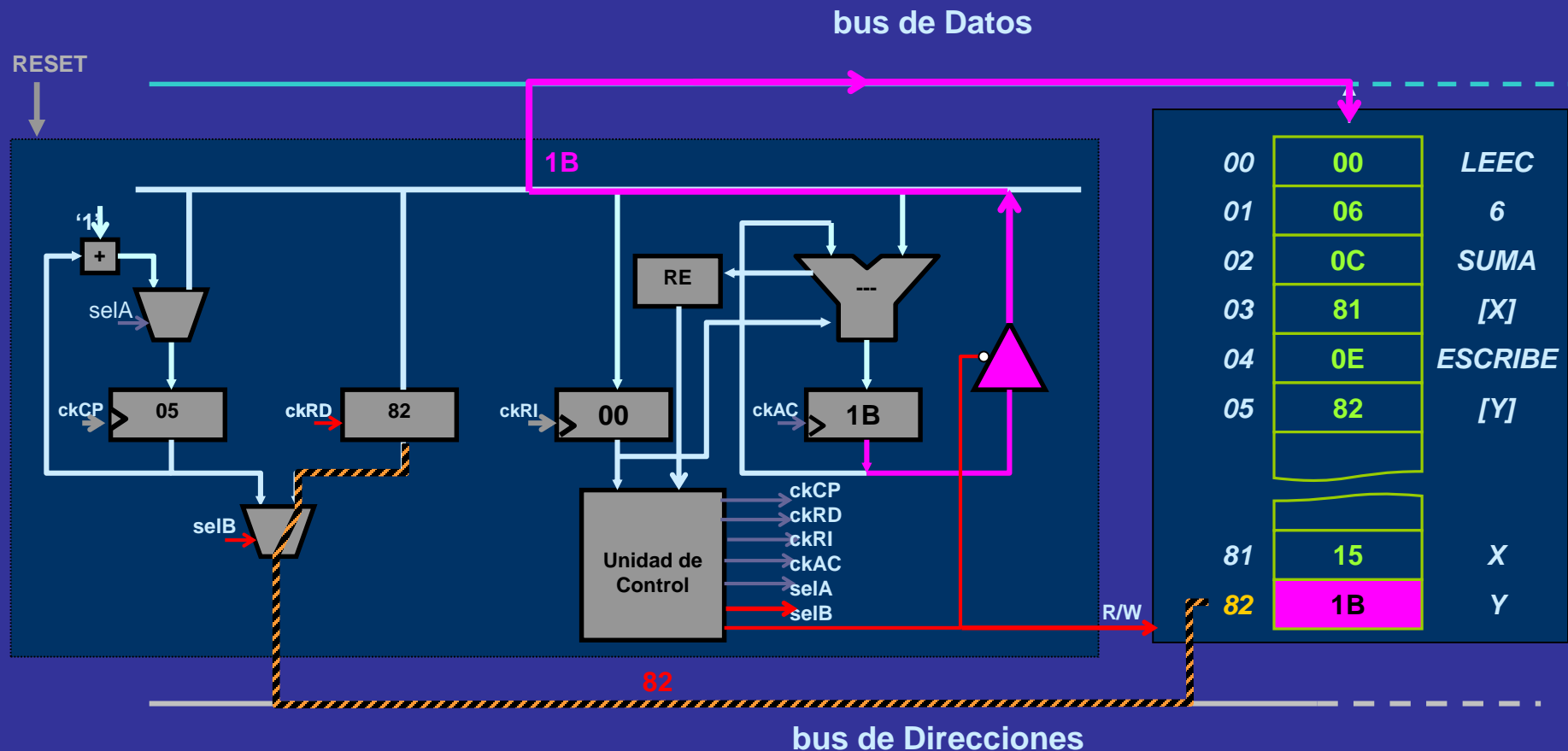
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
LEEC	ANDC	ORC	REST AC	SUMA C	SALT A	SALT AZ	SALT AN	LEE	AND	OR	REST A	SUMA	NOT	ESCR IBE	NOP

13 – Moviendo selB el multiplexor inferior deja pasar el valor de RD (82) al bus de direcciones, quedando así apuntada la variable Y.



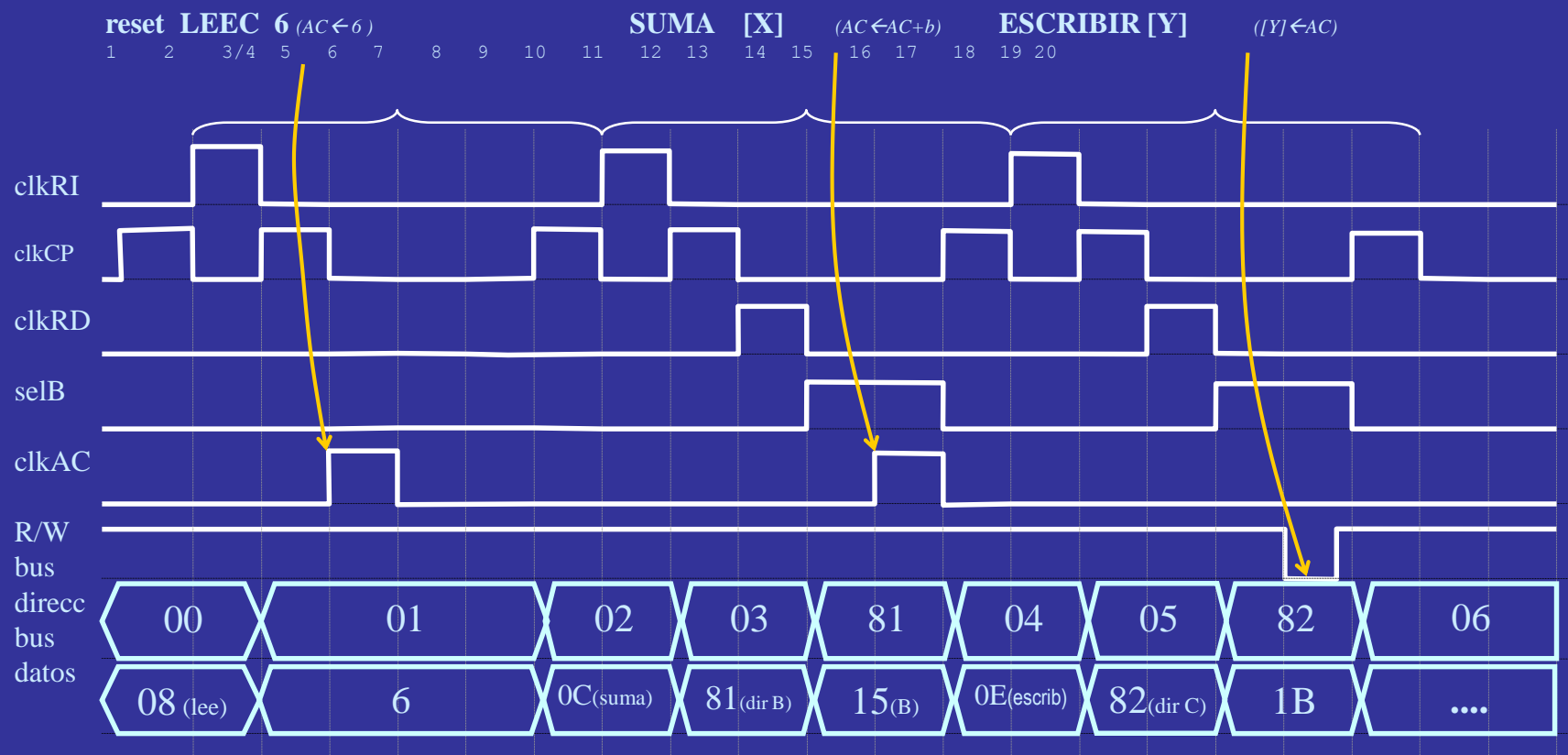
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
LEEC	ANDC	ORC	REST AC	SUMA C	SALT A	SALT AZ	SALT AN	LEE	AND	OR	REST A	SUMA	NOT	ESCRIBE	NOP

14 – Al poner la señal R/-W en '0' el buffer Triestate deja pasar el valor del acumulador (1B) al Bus de Datos. Al mismo tiempo pone a la RAM en modo escritura. El dato (1B) se escribe en la posición 82

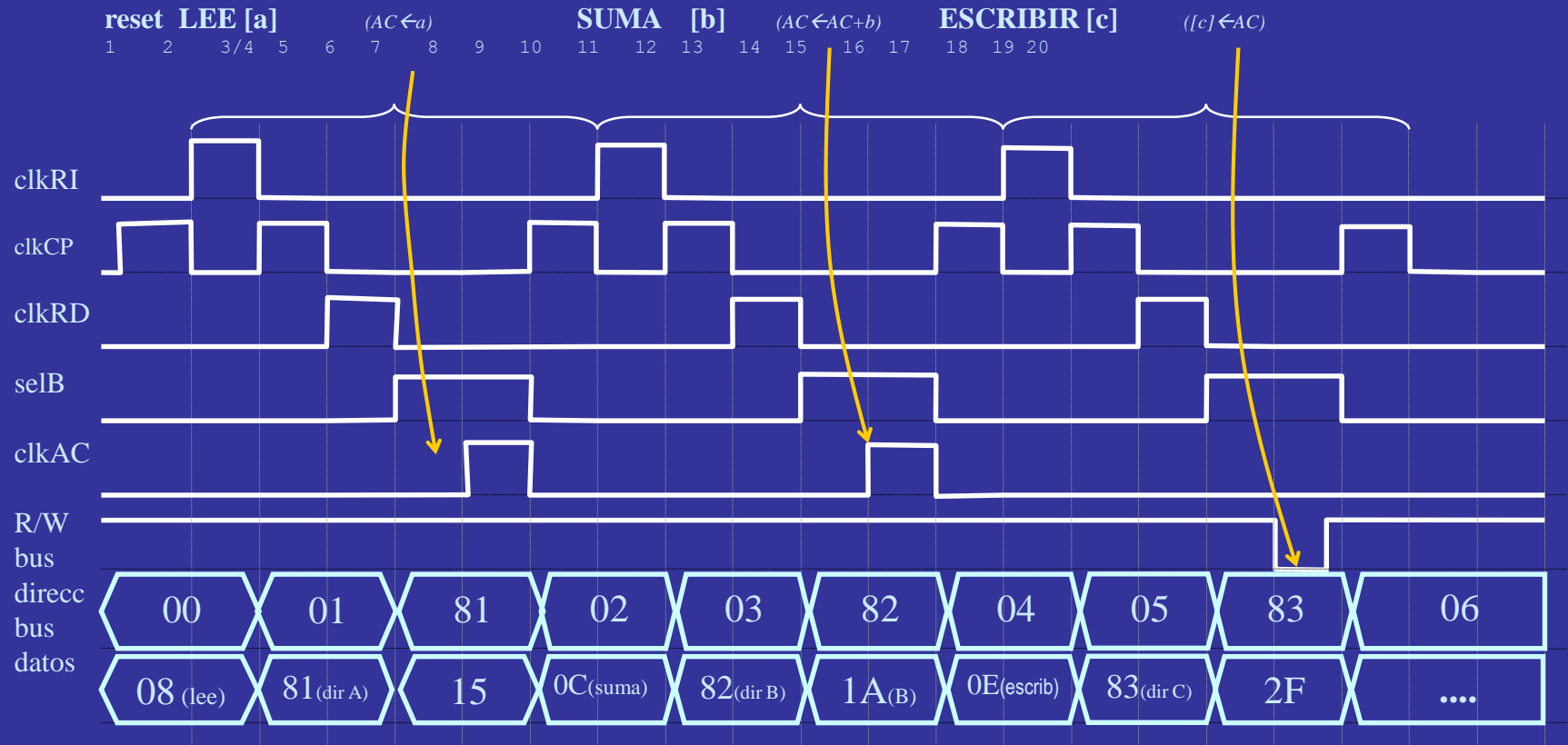


00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
LEEC	ANDC	ORC	REST AC	SUMA C	SALT A	SALT AZ	SALT AN	LEE	AND	OR	REST A	SUMA	NOT	ESCR IBE	NOP

Señales de control para la operación $Y=6+X$

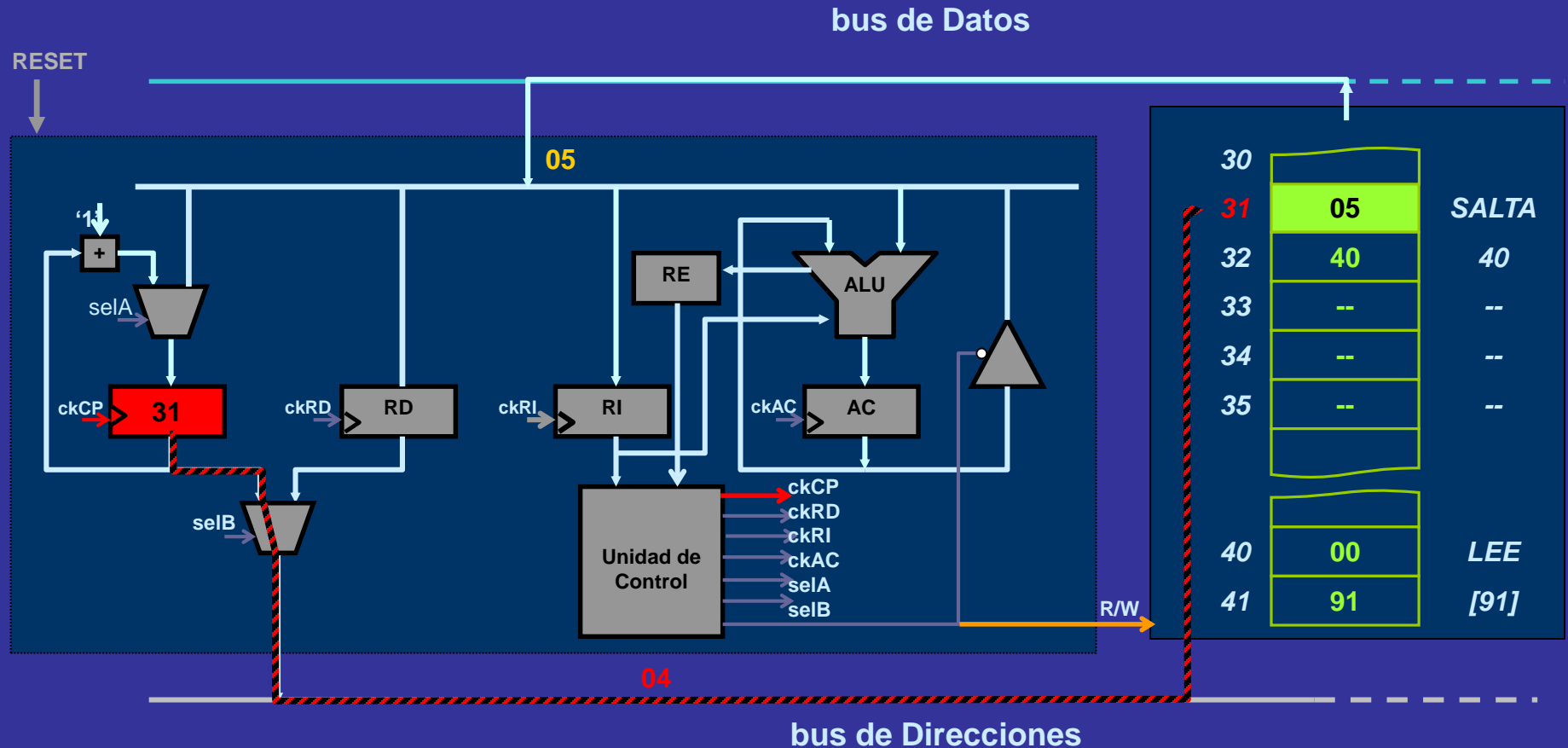


Señales de control para la operación $C=A+B$



Operación de Salto

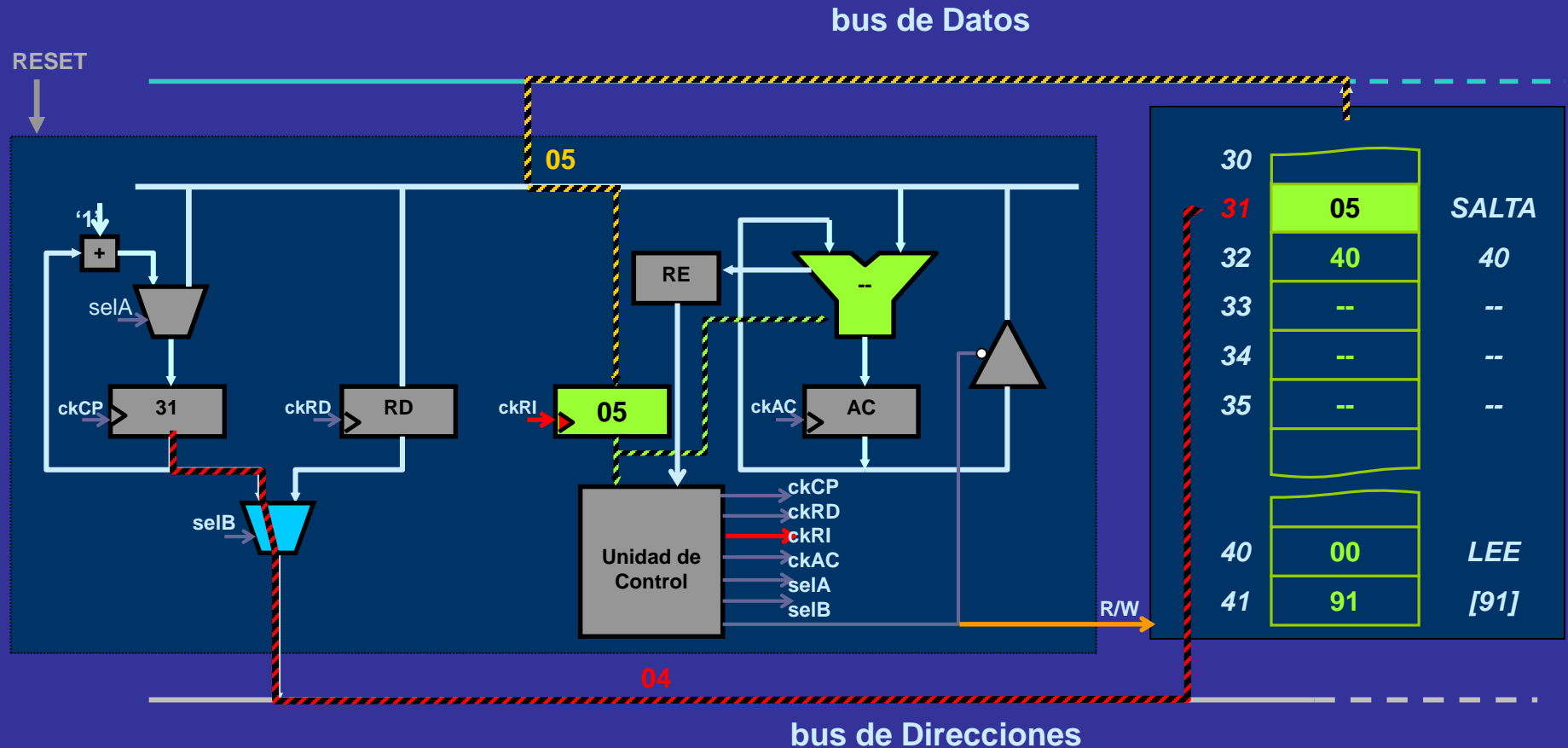
1-El CP apunta a la dirección 31 donde hay una instrucción de salto. La instrucción de salto se presenta en el Bus de Datos



00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
LEEC	ANDC	ORC	REST AC	SUMAC	SALTA	SALT AZ	SALT AN	LEE	AND	OR	REST A	SUMA	NOT	ESCRIBE	NOP

Operación de Salto

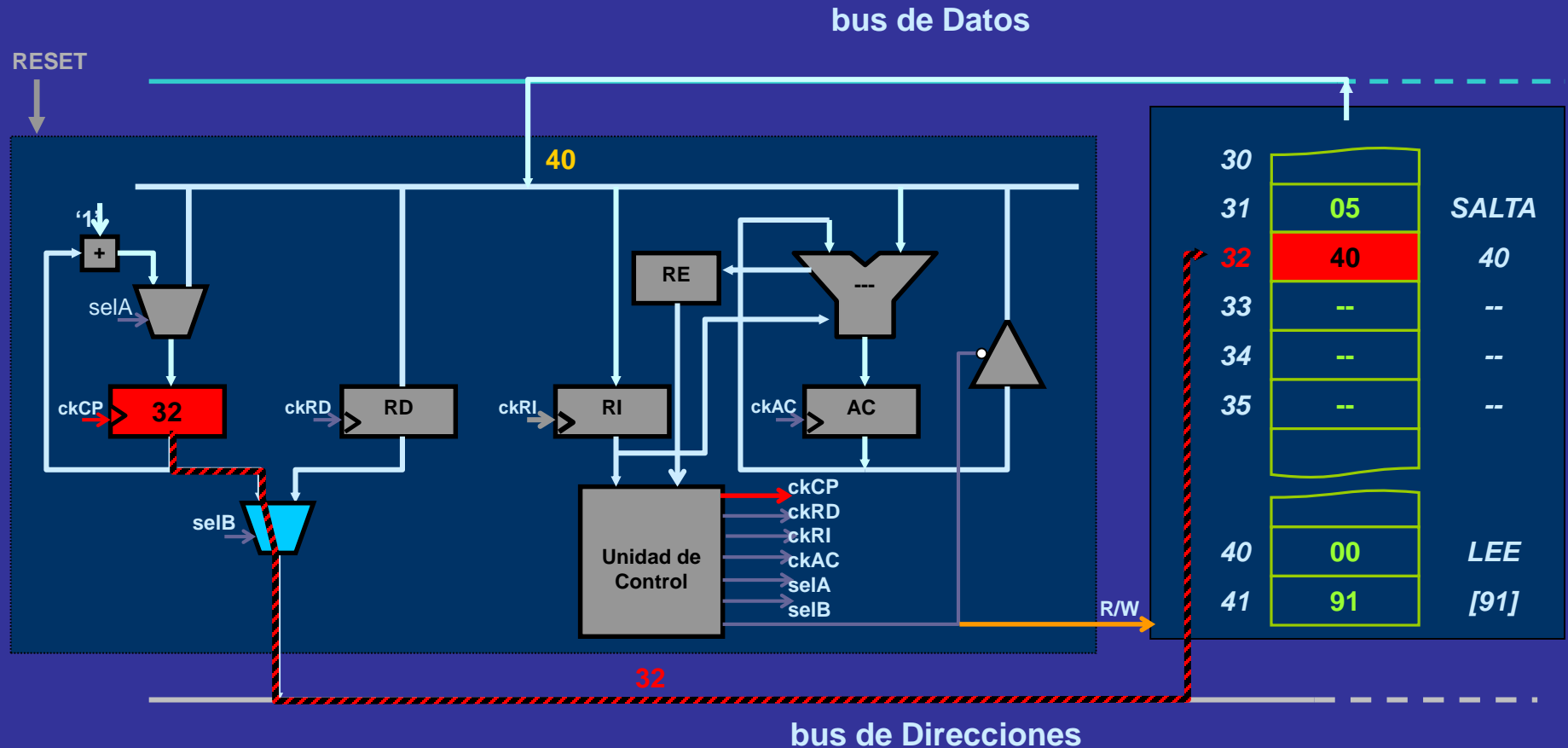
2-La instrucción de salto se carga en el RI – La ALU es indiferente



00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
LEEC	ANDC	ORC	REST AC	SUMA C	SALT A	SALT AZ	SALT AN	LEE	AND	OR	REST A	SUMA	NOT	ESCR IBE	NOP

Operación de Salto

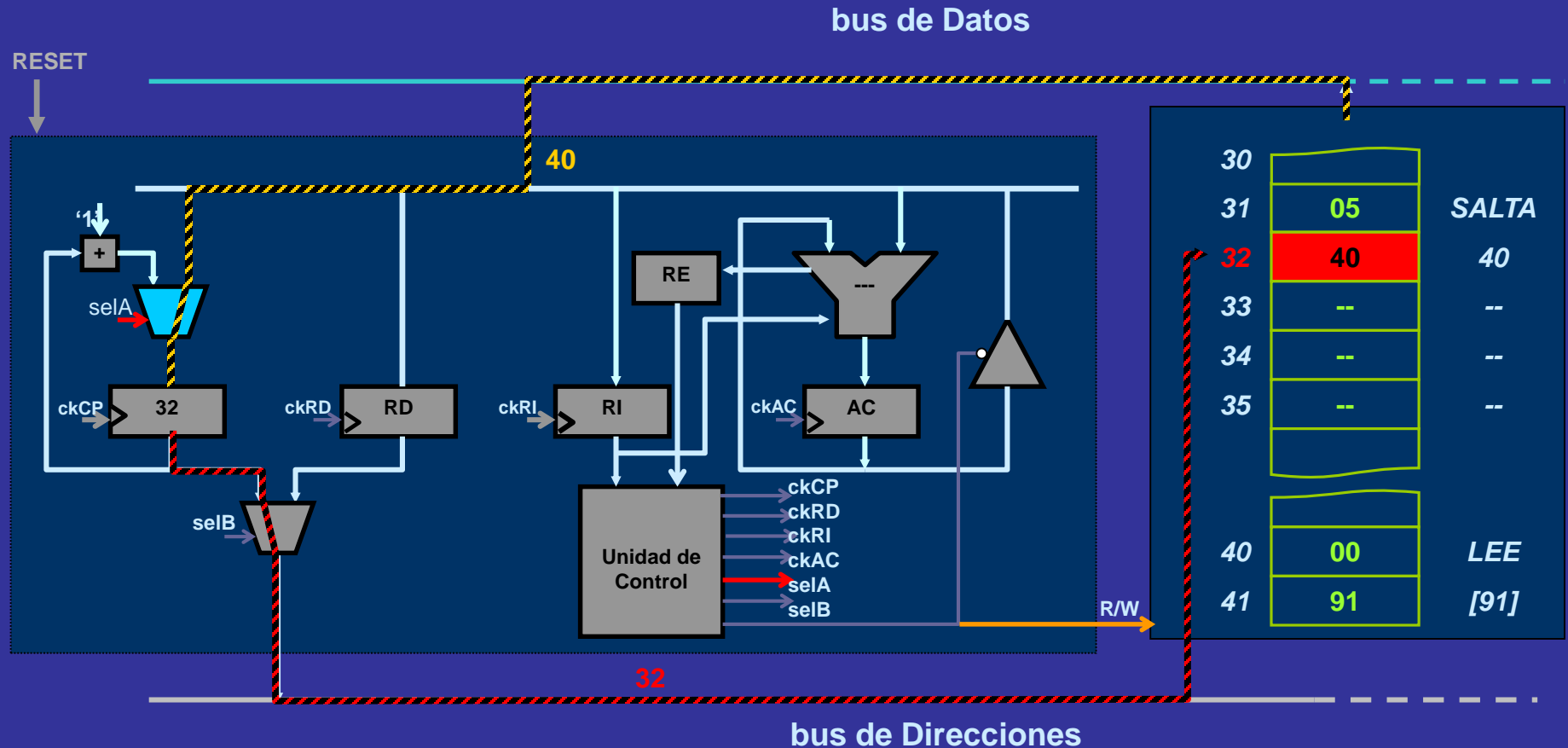
4-EI CP se incrementa y apunta a 32, donde se encuentra la dirección de destino del salto (en este ejemplo 40)



00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
LEEC	ANDC	ORC	REST AC	SUMA C	SALT A	SALT AZ	SALT AN	LEE	AND	OR	REST A	SUMA	NOT	ESCR IBE	NOP

Operación de Salto

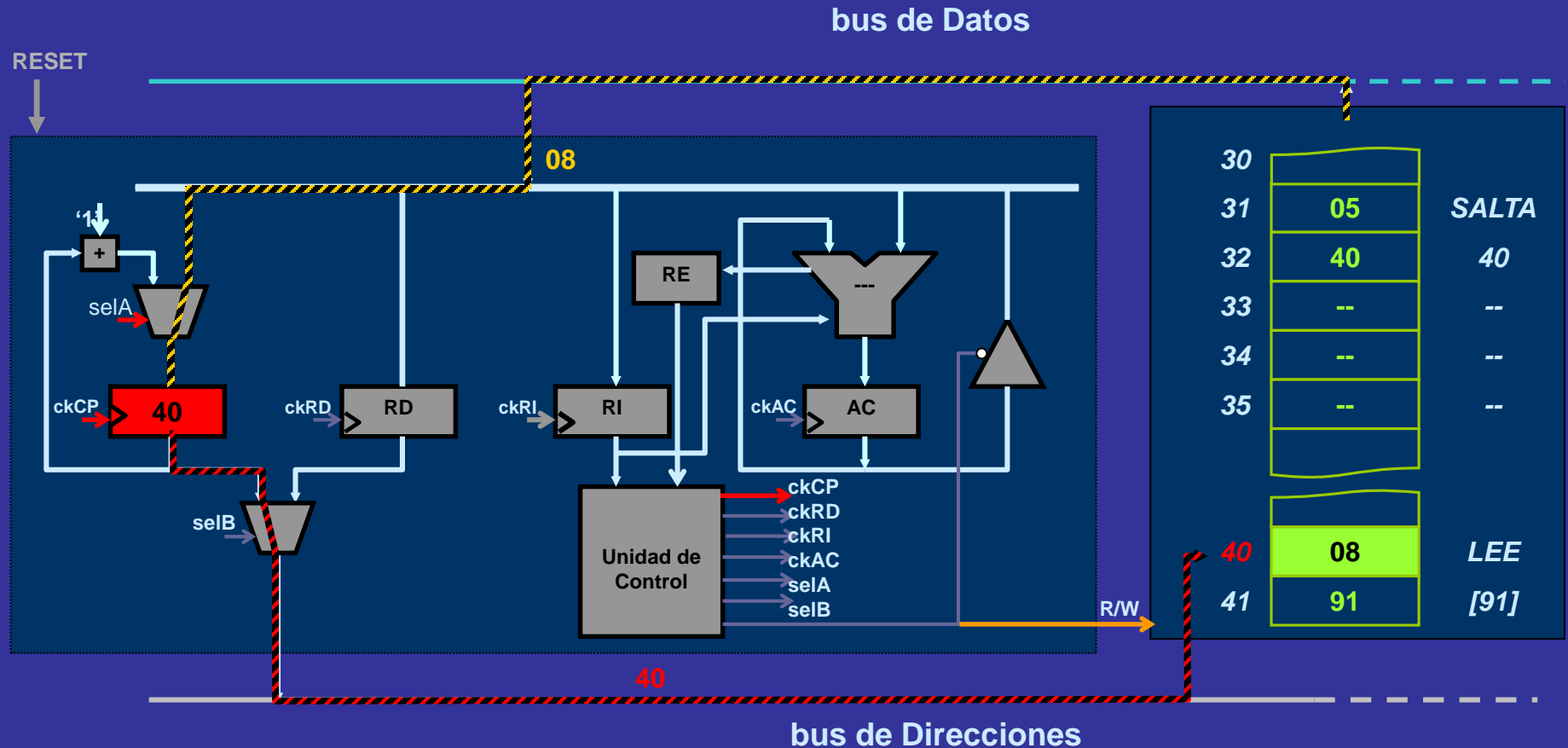
6-El multiplexor A selecciona el bus de datos (en vez de CP+1)



00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
LEEC	ANDC	ORC	REST AC	SUMA C	SALT A	SALT AZ	SALT AN	LEE	AND	OR	REST A	SUMA	NOT	ESCR IBE	NOP

Operación de Salto

7- Al dar clock a CP, éste adopta el valor 40.
El programa continúa en dicha dirección.

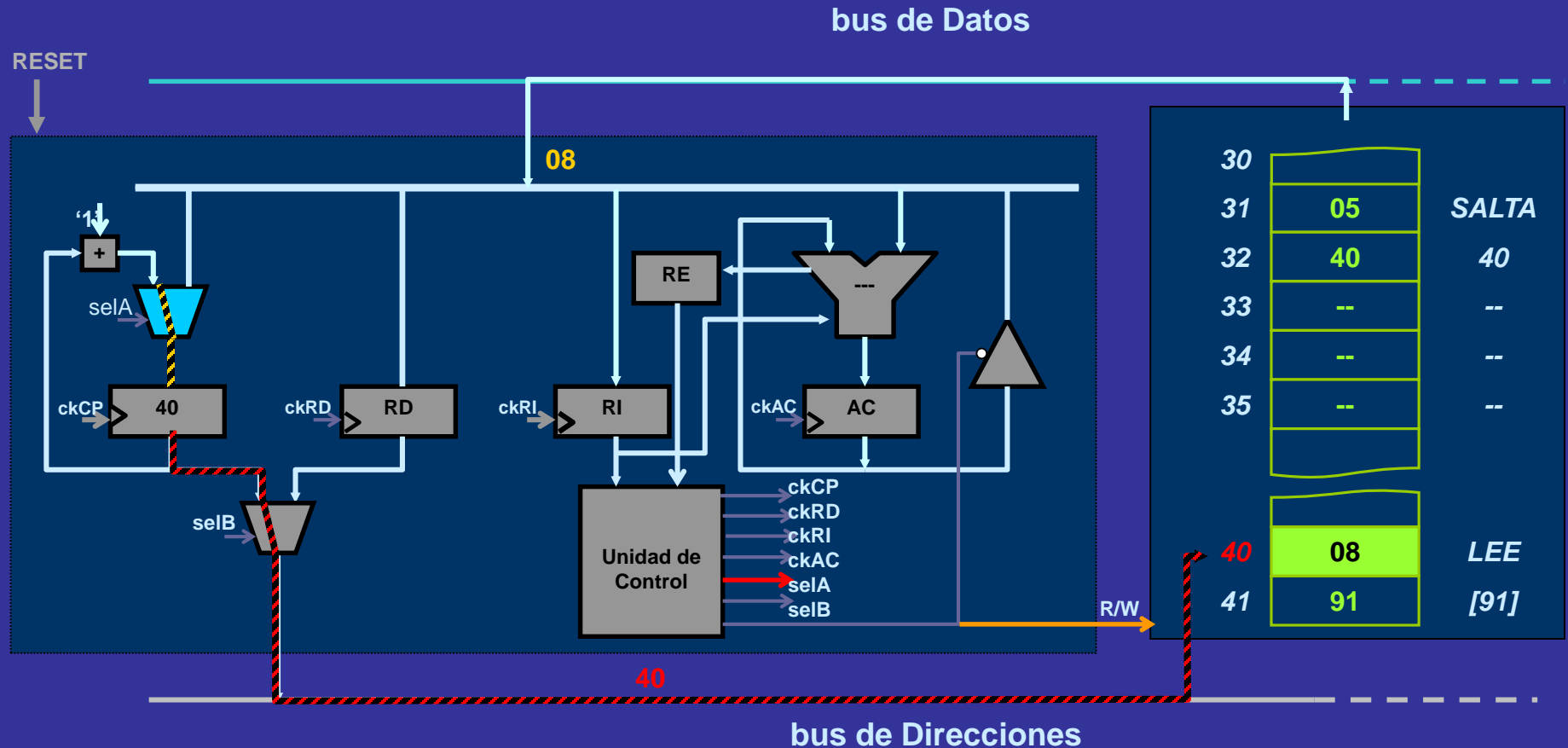


00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
LEEC	ANDC	ORC	REST AC	SUMA C	SALT A	SALT AZ	SALT AN	LEE	AND	OR	REST A	SUMA	NOT	ESCR IBE	NOP

Operación de Salto

8-El multiplexor A vuelve a la posición habitual (CP+1).

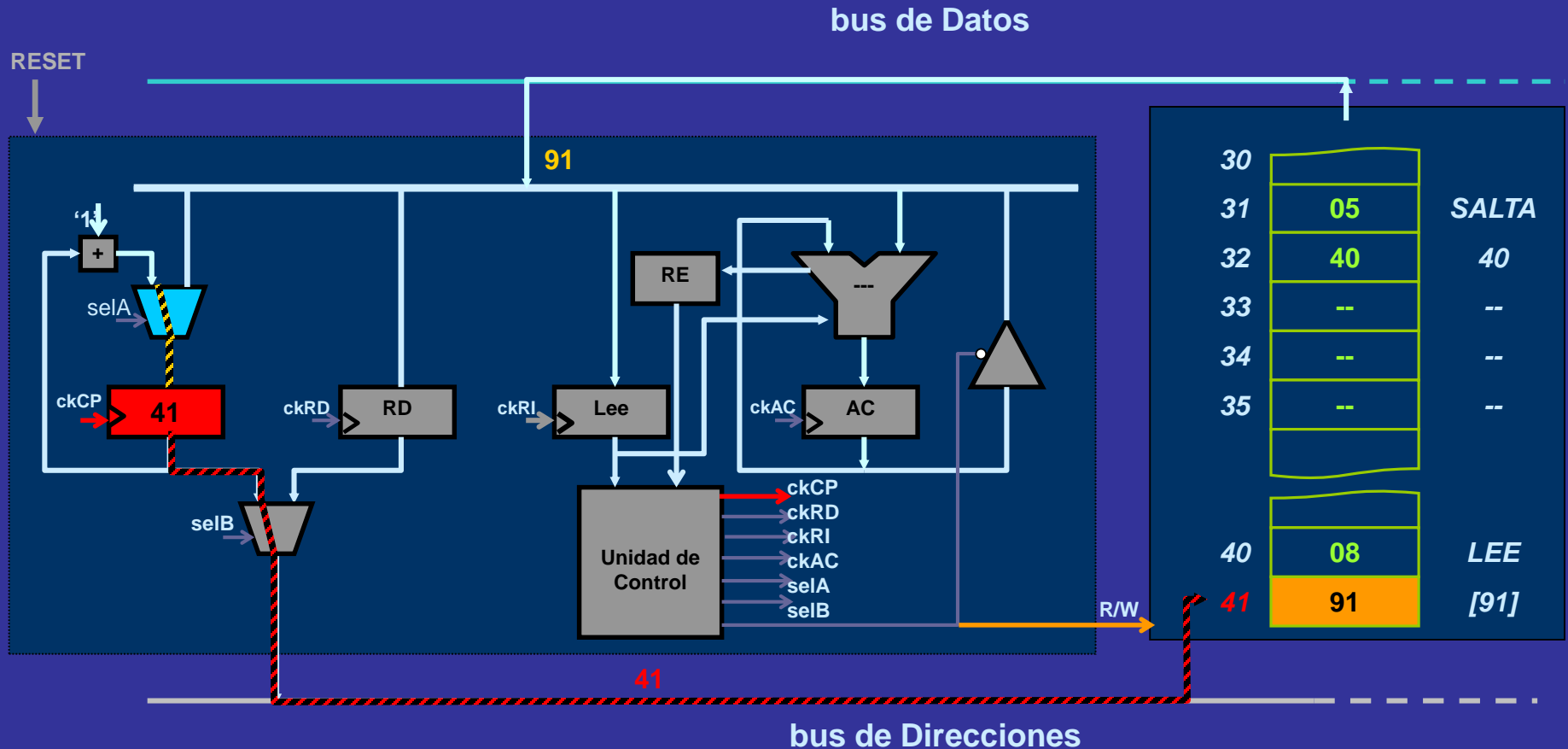
Se apunta a la instrucción Lee



00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
LEEC	ANDC	ORC	REST AC	SUMA C	SALT A	SALT AZ	SALT AN	LEE	AND	OR	REST A	SUMA	NOT	ESCR IBE	NOP

Operación de Salto

(...) – Cuando se produce un nuevo clock en CP
éste se incrementa al valor 41



00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
LEEC	ANDC	ORC	REST AC	SUMA C	SALT A	SALT AZ	SALT AN	LEE	AND	OR	REST A	SUMA	NOT	ESCR IBE	NOP

Llamados a subrutina

En el programa puede repetirse un conjunto de instrucciones más o menos complejo, por ejemplo la división entre dos números mediante el mecanismo de restas sucesivas, la aproximación de una función trascendente mediante polinomios etc. En tal caso es útil, para ahorrar memoria y simplificar la comprensión del programa, agrupar este conjunto de instrucciones en una “subrutina”, que pueda utilizarse desde distintos puntos del programa “principal”.

Programa Principal

LEE	[A]
ESCRIBE	[dividendo]
LEE	[B]
ESCRIBE	[divisor]
LLAMA	<i>dividir</i>
LEE	[cociente]

LEE	[C]
ESCRIBE	[dividendo]
LEE	[D]
ESCRIBE	[divisor]
LLAMA	<i>dividir</i>
LEE	[cociente]

1) El programa principal, antes de llamar a la subrutina, escribe los operandos **dividendo** y **divisor** en posiciones de la RAM o en registros auxiliares internos (acumuladores auxiliares)

2) Luego “llama” a la subrutina (es decir el contador de programa salta a la posición donde comienza la subrutina)

Subrutina *dividir*

LEE	[dividendo]
RESTA	[divisor]
ESCRIBE	[cociente]
RETORNA	

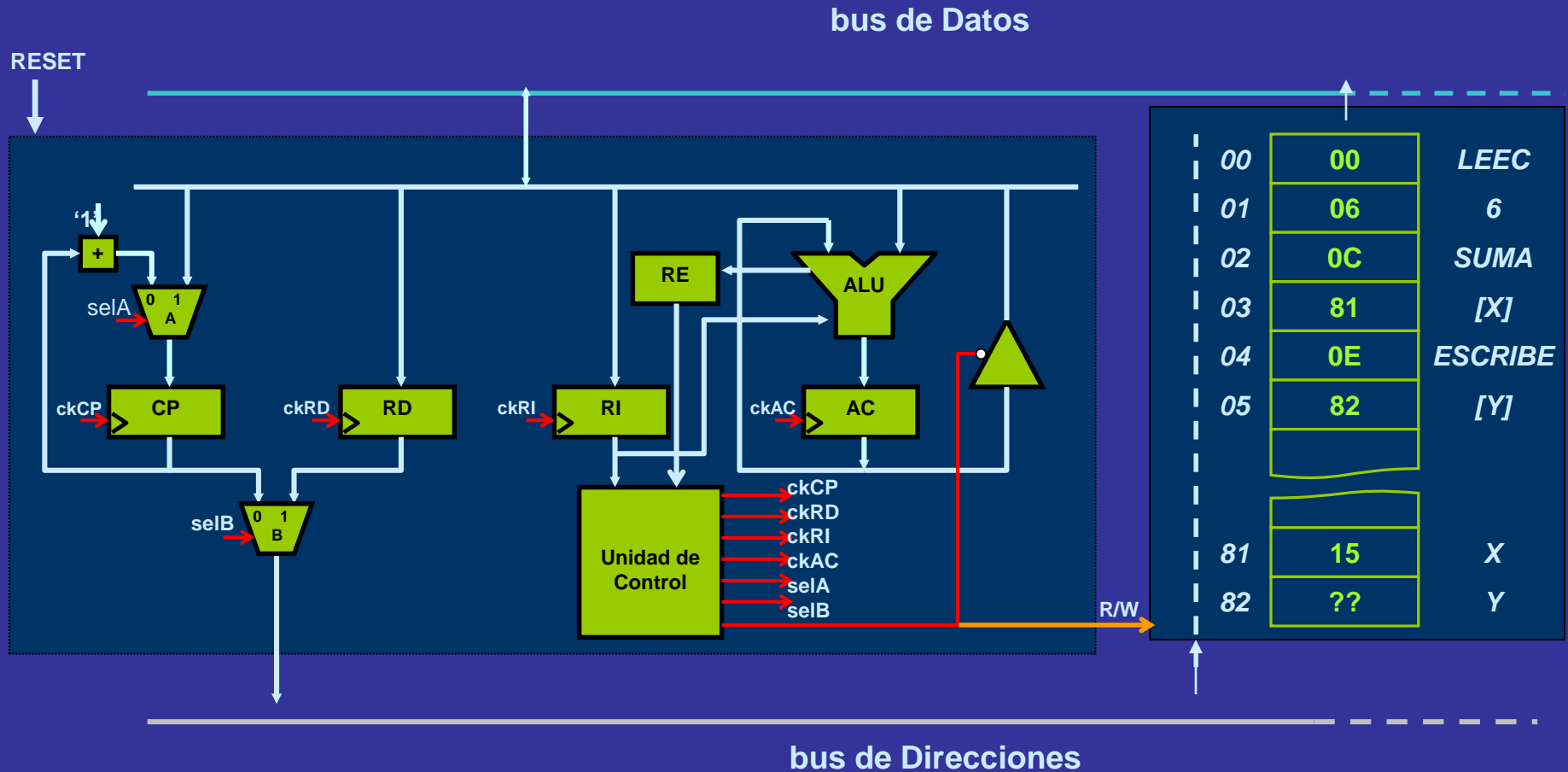
3) La subrutina realiza la operación y escribe el resultado en **cociente** (RAM o registro auxiliar del microprocesador).

4) La subrutina **retorna** a la **instrucción siguiente** a la de la llamada.

5) El programa lee el resultado en cociente

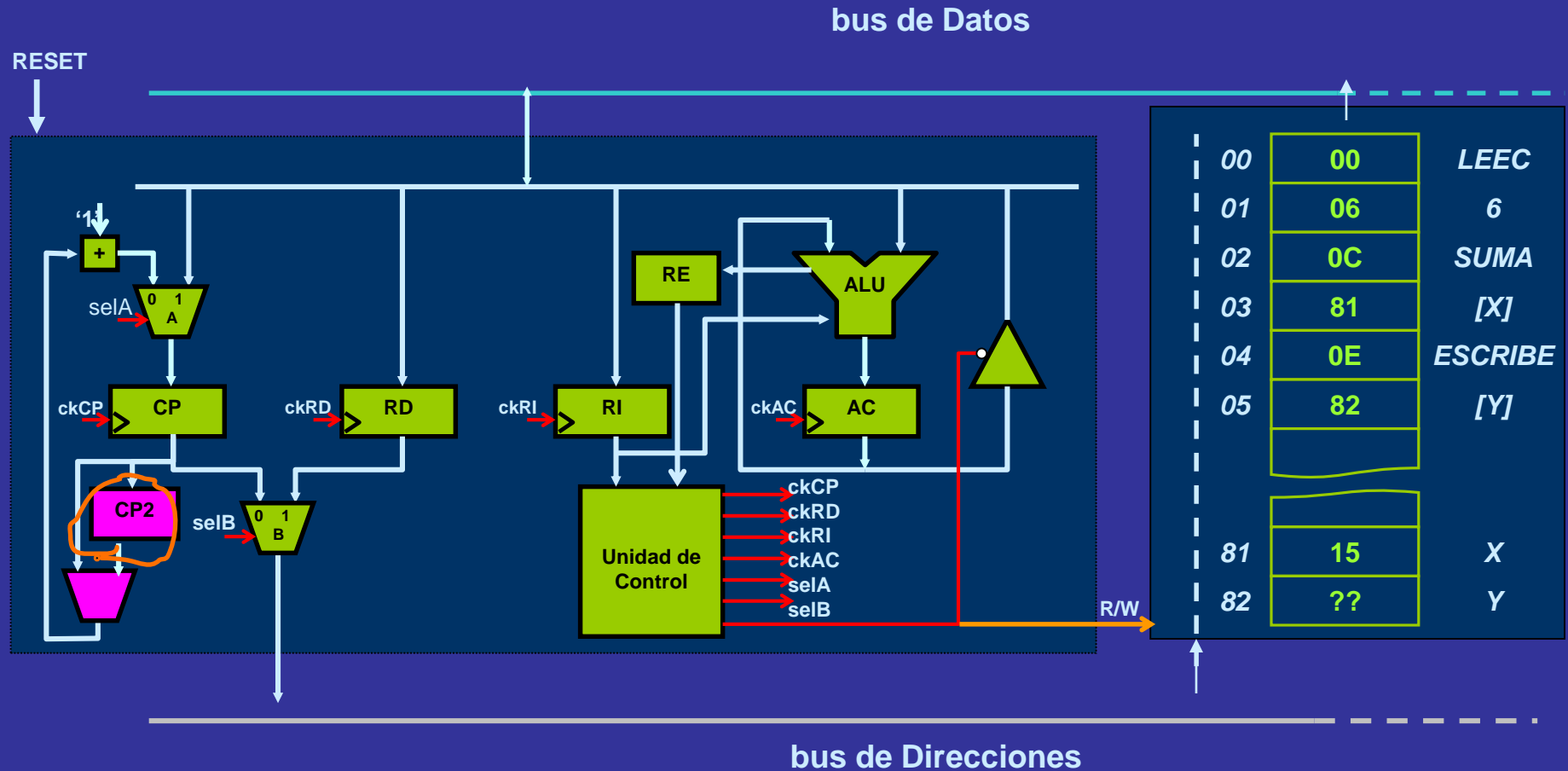
Llamados a subrutina:

Hardware que no soporta llamado a subrutina

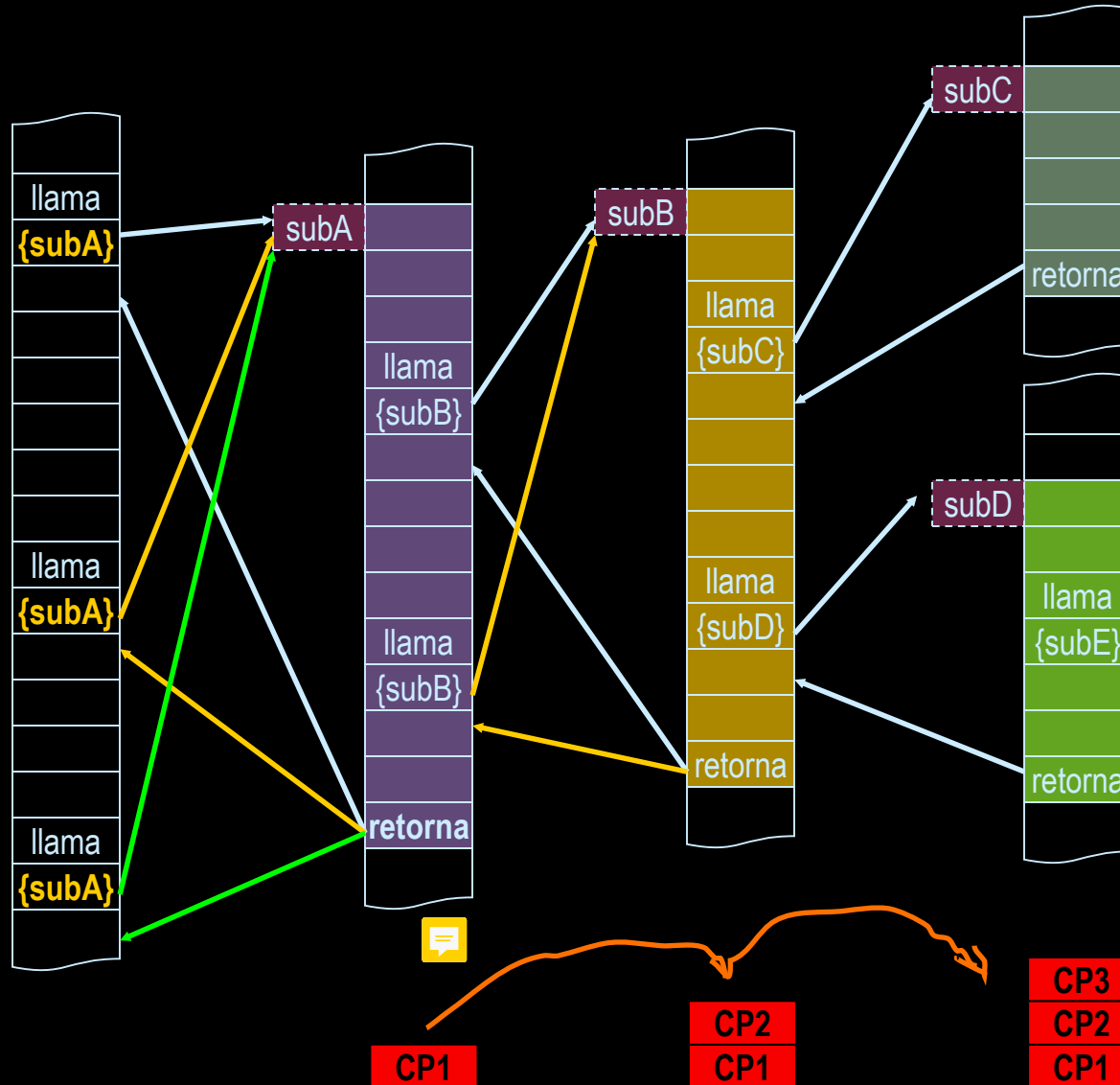


Llamados a subrutina:

Hardware que soporta 1 nivel de llamado a subrutina



Llamados a subrutina: Subrutinas anidadas

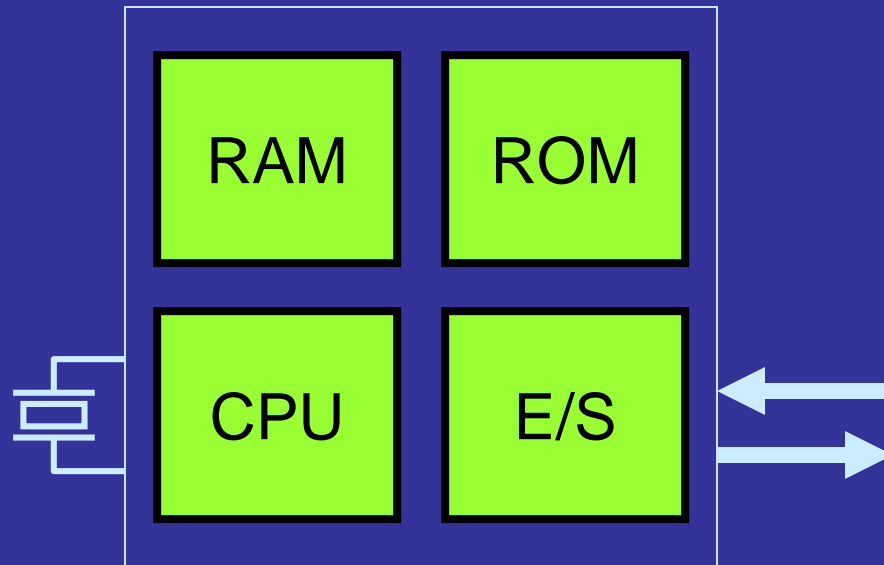


Resumen

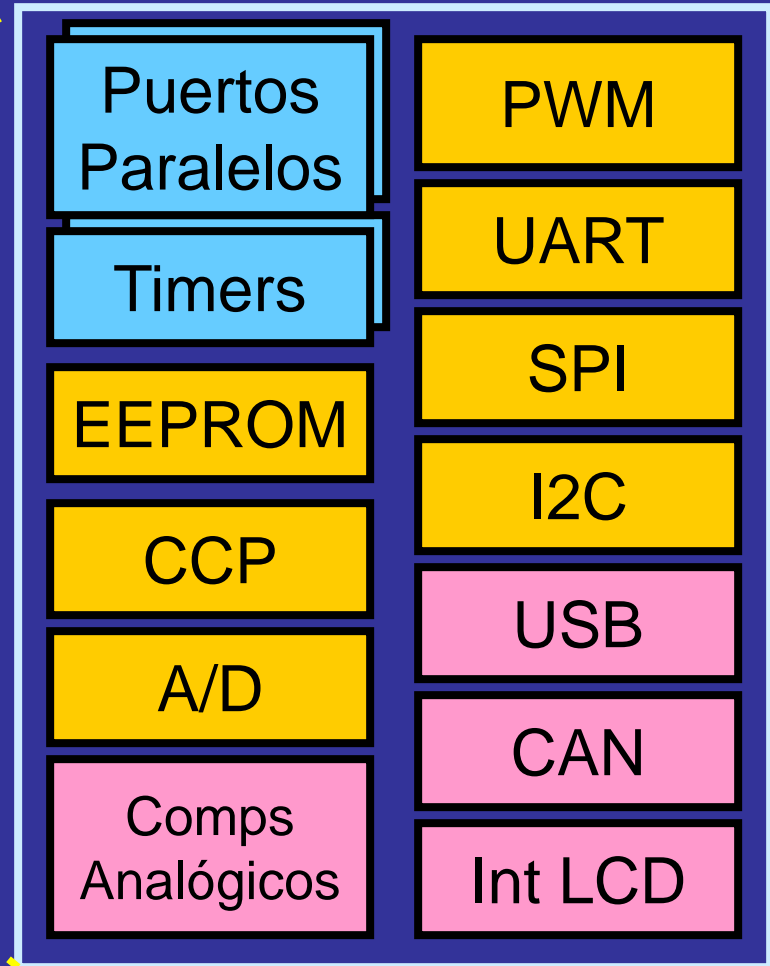
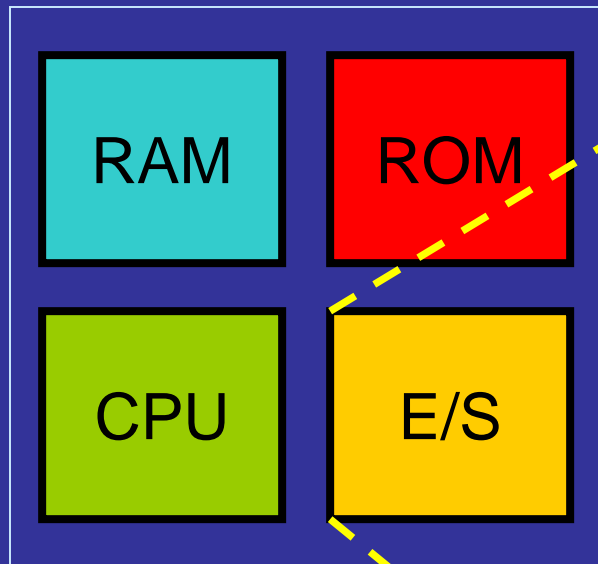
- Un sistema de cómputo programable está constituido por una unidad de Procesamiento, una unidad de Memoria y una interfaz de E/S.
- El modo en que se implementa este sistema da lugar a las arquitecturas Harvard o Von Neumann. Esta última es la que se ha analizado porque es la más utilizada.
- Un sistema con arquitectura Von Neumann tiene una única memoria para instrucciones y datos, una unidad de procesamiento y una unidad de E/S, conectados por 3 buses: de datos, de direcciones y de control. El bus de datos transfiere tanto instrucciones de programa como datos.
- Un μP es básicamente un conjunto de registros, dispositivos de selección, una ALU y una Unidad de Control UC, que es un circuito generador de secuencias, que se denomina también Máquina de Estados.
- La UC de un μP elemental con un reducido repertorio de instrucciones –16 en nuestro ejemplo- debe realizar para cada una de ellas alguno de estos cuatro tipos de secuencia: de lectura inmediata (LEEC ANDC ..) , de lectura direccionada (LEE AND SUMA ...), de escritura o de salto.
- Una instrucción típica está compuesta por dos campos, un campo código de operación (qué hay que hacer) y un campo operando (dato para ejecutar dicha operación). El campo operando puede ser el valor con el cual operar (instrucciones de lectura inmediata), o la dirección de memoria de datos donde se encuentra el valor a operar (instrucciones de lectura direccionada o de escritura), o la dirección de memoria donde debe continuar el programa (instrucciones de salto).
- El uso de llamado a subrutinas (y el correspondiente retorno) permite escribir programas más cortos, escribiendo una sola vez las rutinas más utilizadas. Para el retorno se requiere resguardar el contador de programa. En un esquema de subrutinas anidadas es necesario contar con una pila y un puntero de pila.




Microcontroladores

Bloques de un microcontrolador (μC)

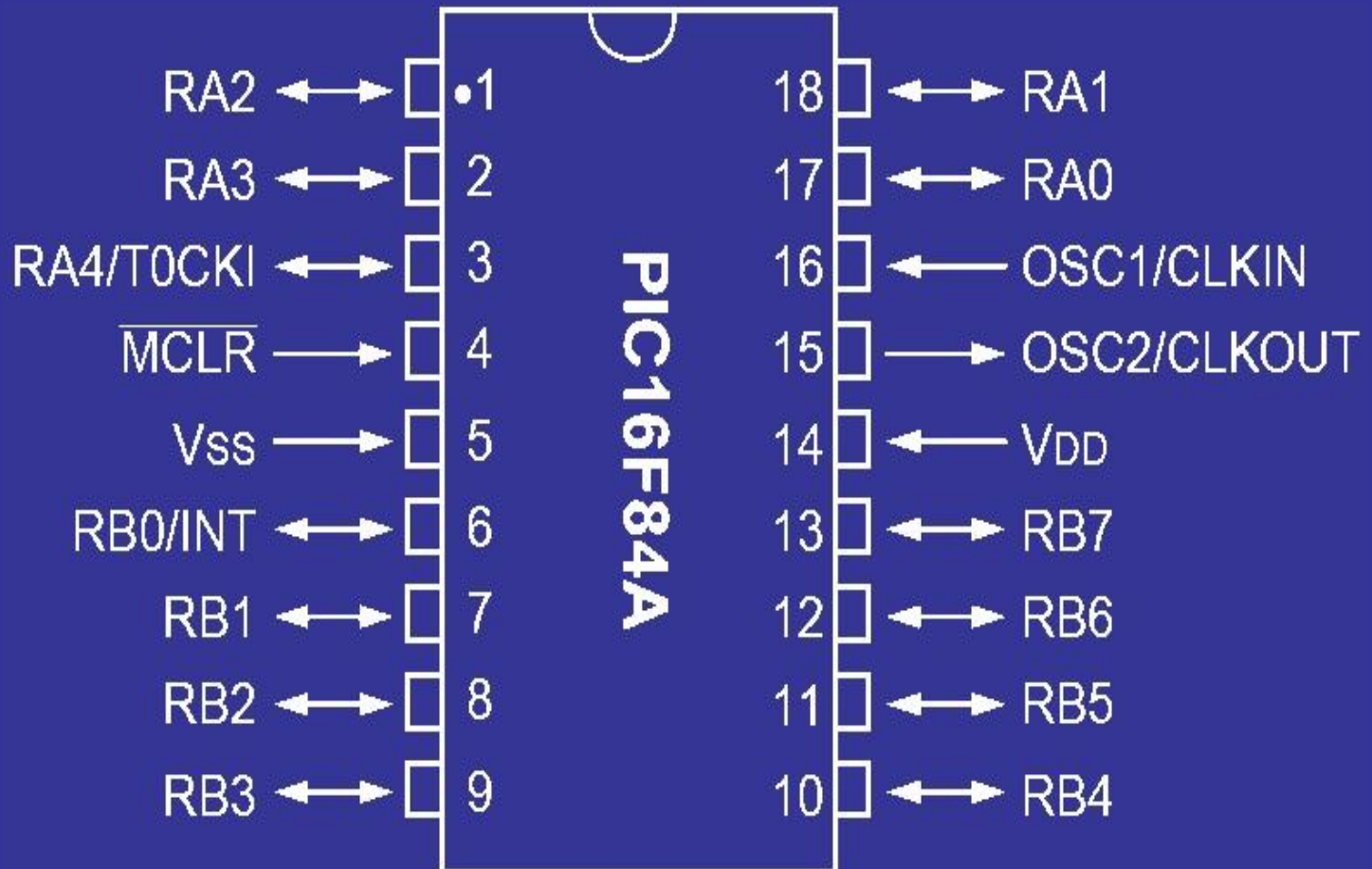


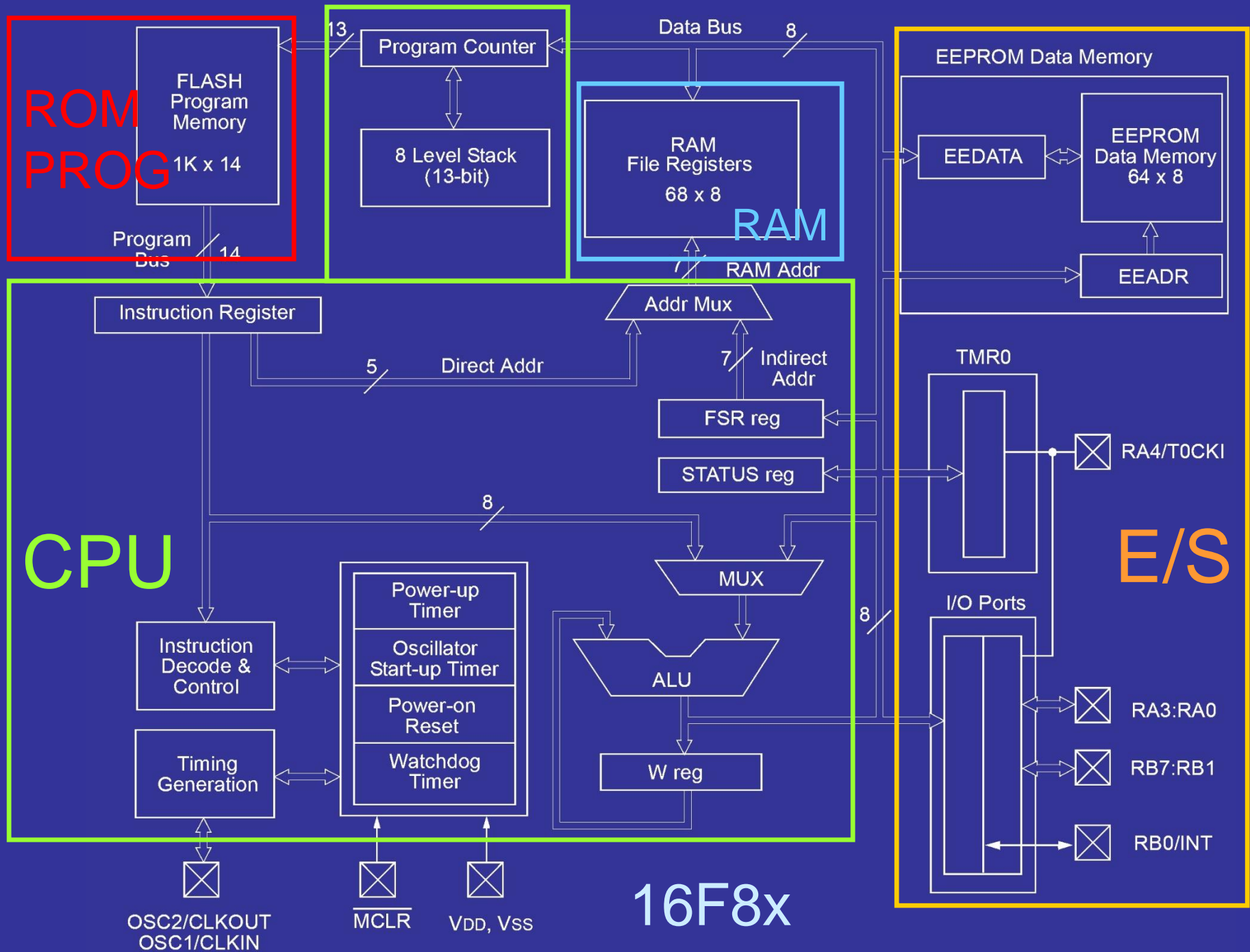
Bloques de un microcontrolador (2)



-  Periféricos básicos
-  Periféricos comunes
-  Periféricos especiales

PIC 16F83/84

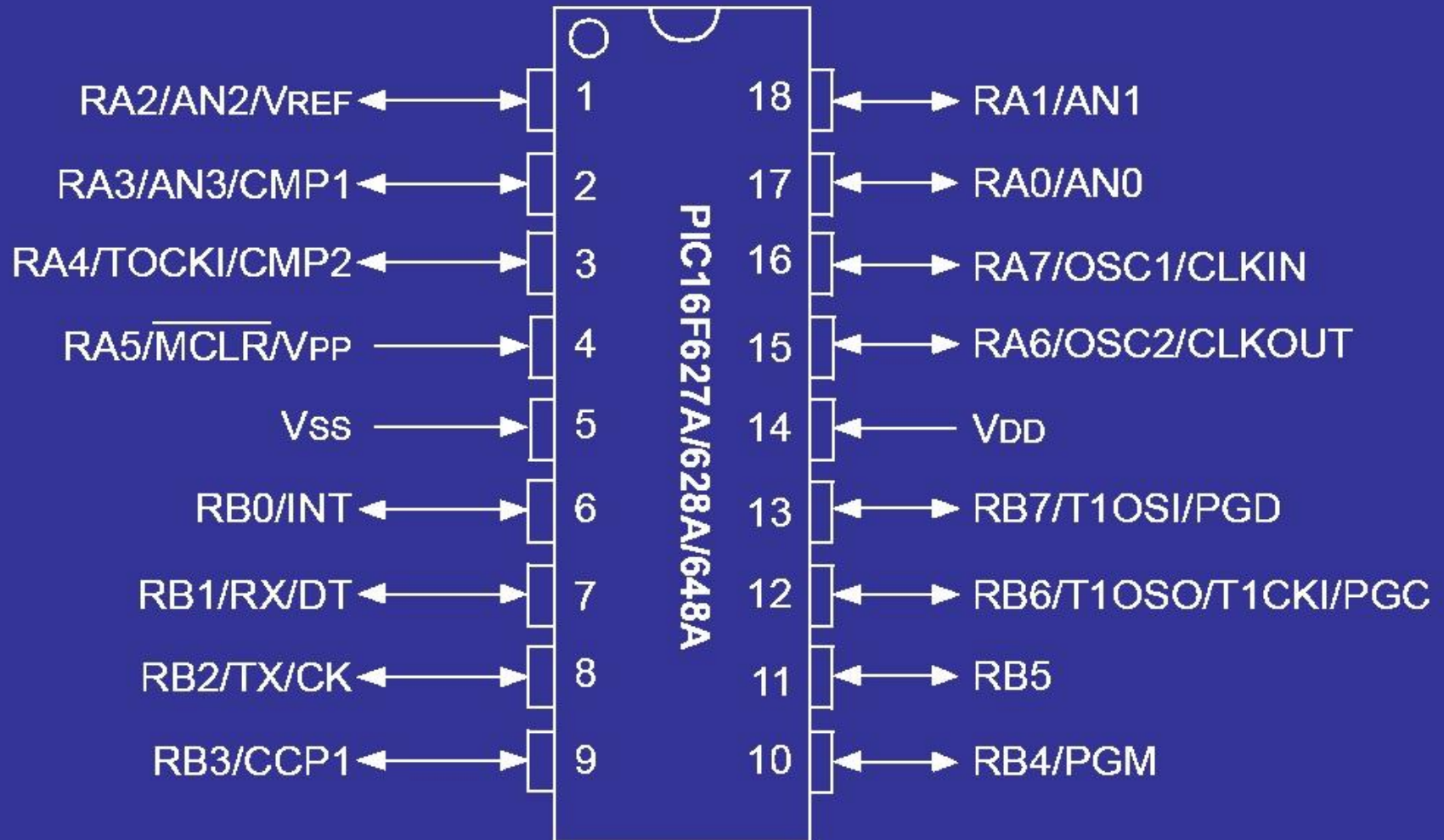




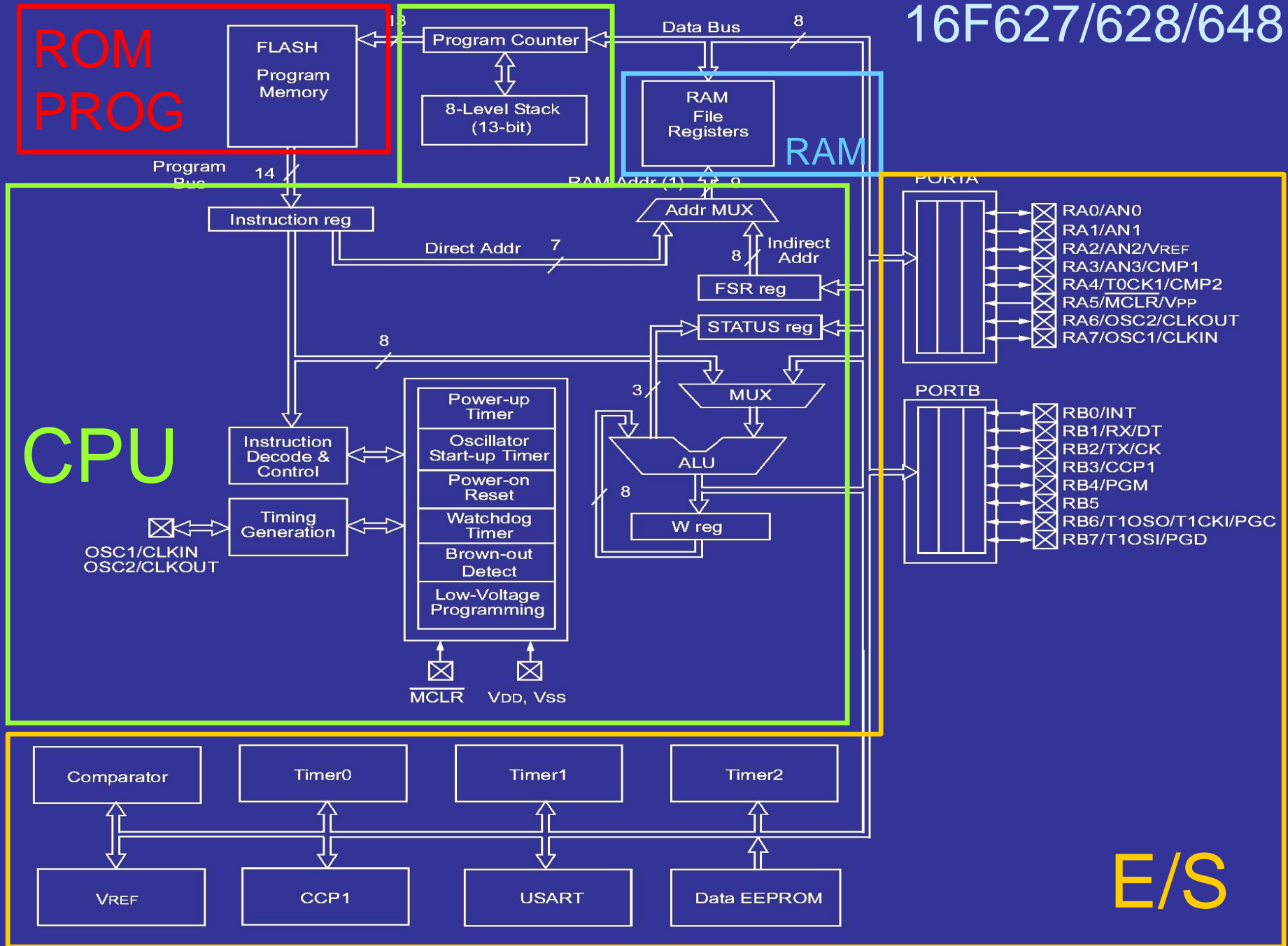
Características de los modelos 16F8x

Device	Program Memory (words)	Data RAM (bytes)	Data EEPROM (bytes)	Max. Freq (MHz)
PIC16F83	512 Flash	36	64	10
PIC16F84	1 K Flash	68	64	10
PIC16CR83	512 ROM	36	64	10
PIC16CR84	1 K ROM	68	64	10

PIC 16F627/628/648



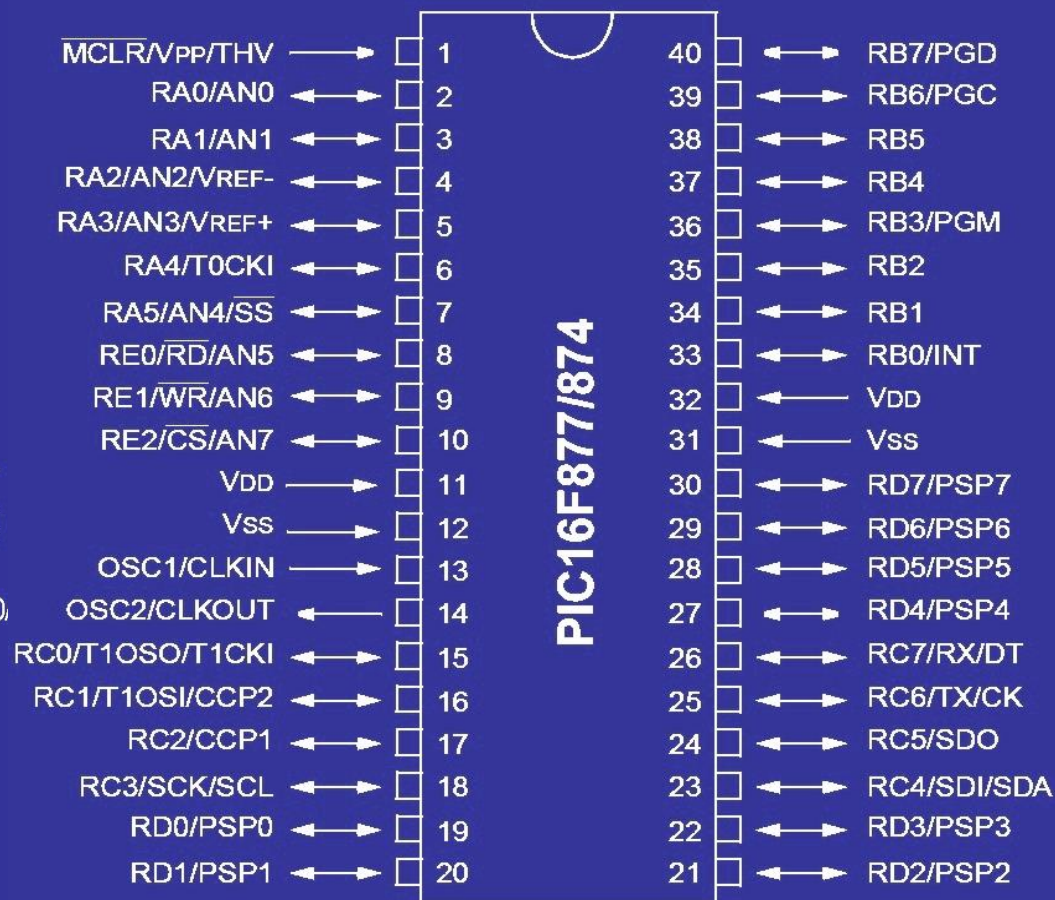
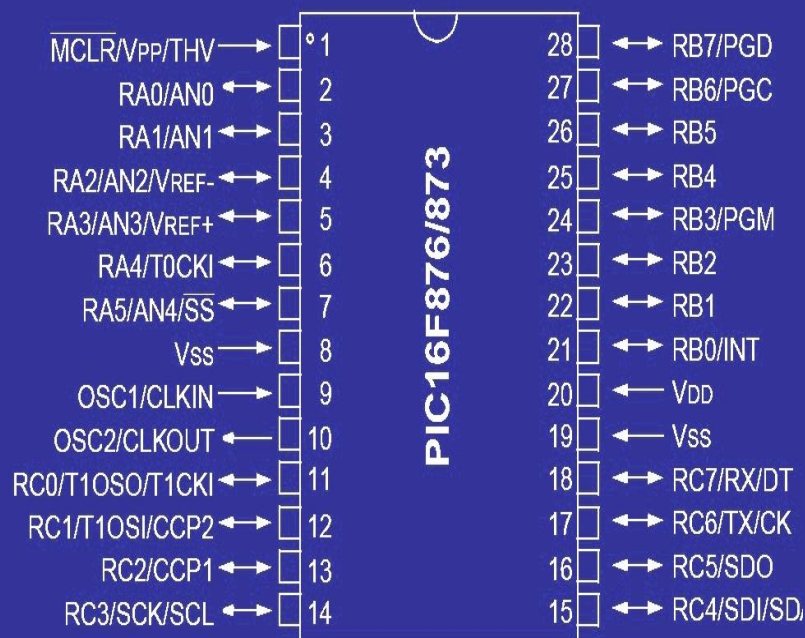
16F627/628/648

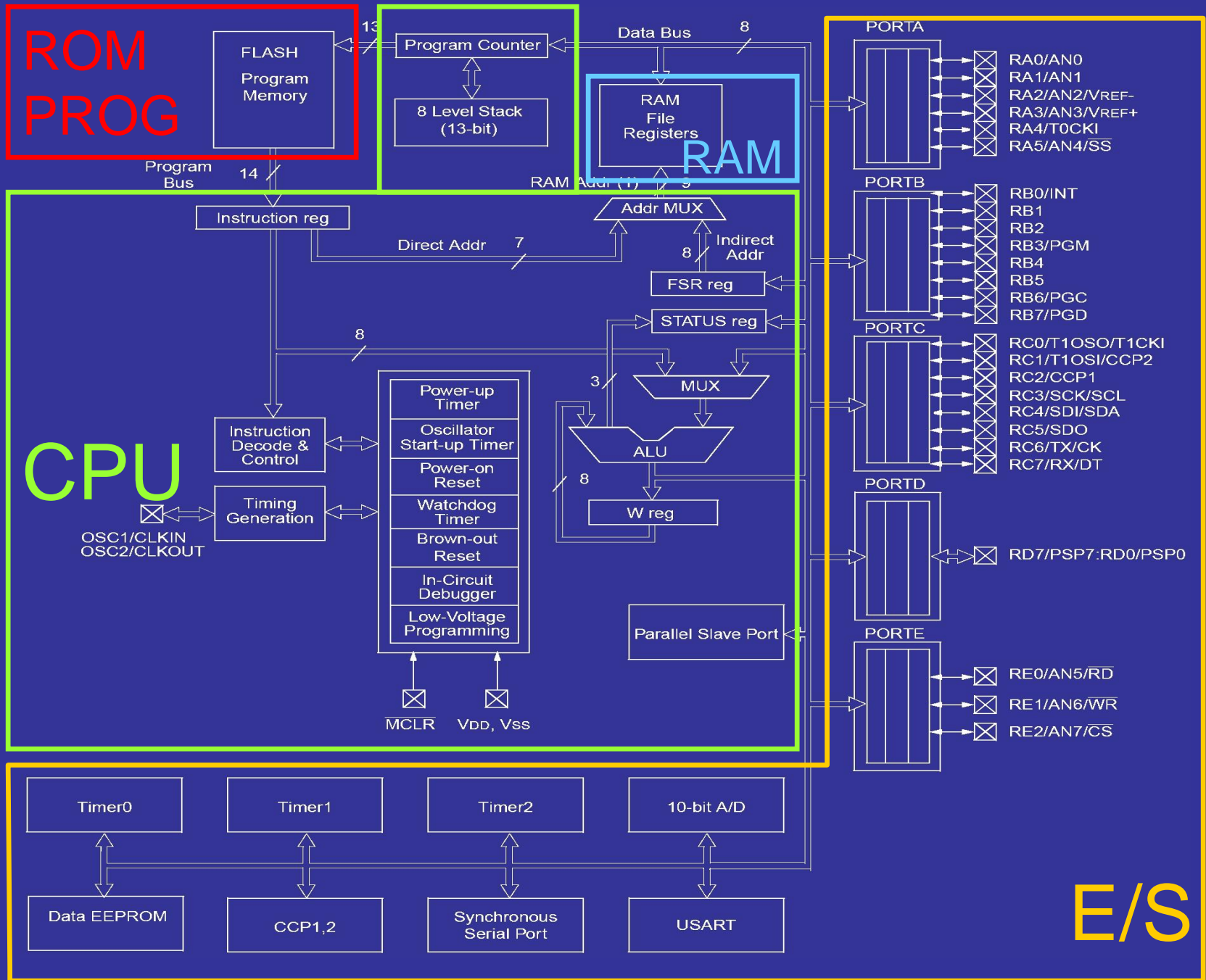


Características de los modelos 16F6xx

[illegible]

PIC 16F873/876/874/877

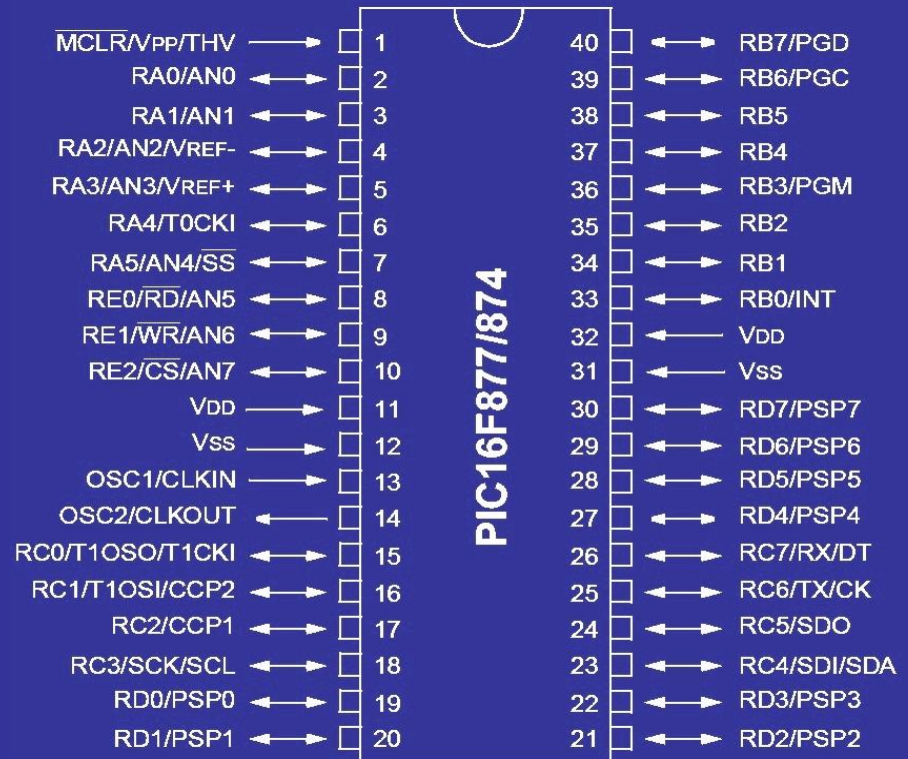
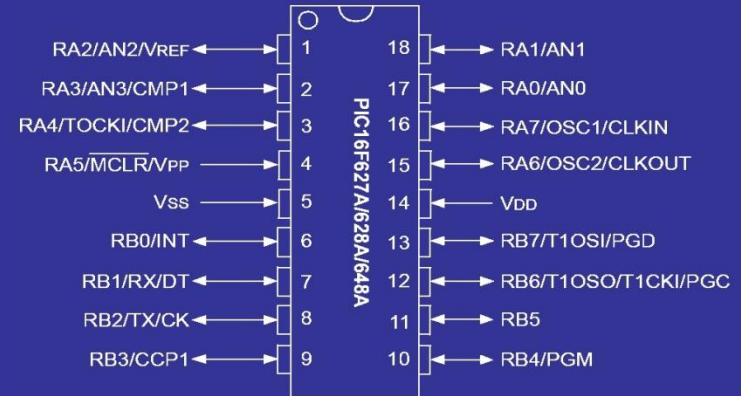
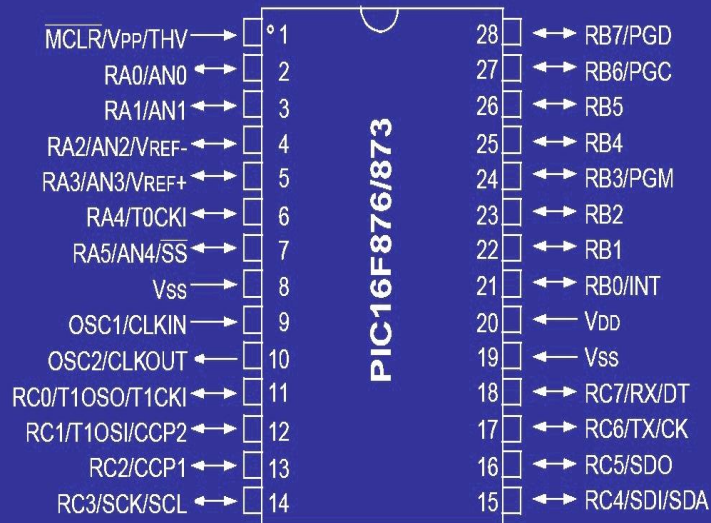
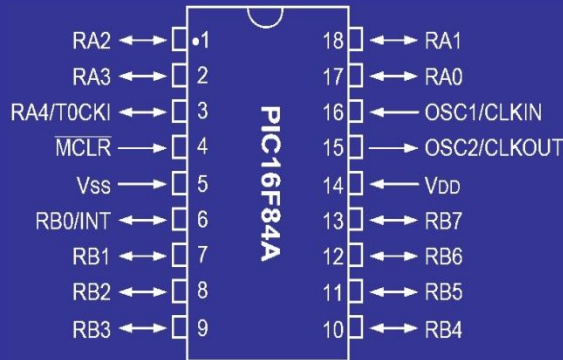
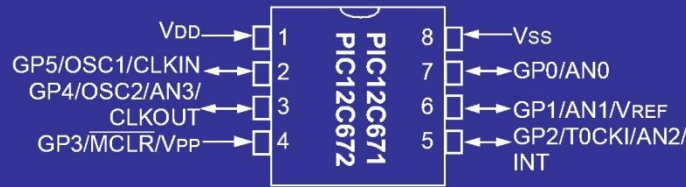




Características de los modelos 16F87x

Key Features PICmicro™ Mid-Range Reference Manual (DS33023)	PIC16F873	PIC16F874	PIC16F876	PIC16F877
Operating Frequency	DC - 20 MHz	DC - 20 MHz	DC - 20 MHz	DC - 20 MHz
RESETS (and Delays)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)
FLASH Program Memory (14-bit words)	4K	4K	8K	8K
Data Memory (bytes)	192	192	368	368
EEPROM Data Memory	128	128	256	256
Interrupts	13	14	13	14
I/O Ports	Ports A,B,C	Ports A,B,C,D,E	Ports A,B,C	Ports A,B,C,D,E
Timers	3	3	3	3
Capture/Compare/PWM Modules	2	2	2	2
Serial Communications	MSSP, USART	MSSP, USART	MSSP, USART	MSSP, USART
Parallel Communications	—	PSP	—	PSP
10-bit Analog-to-Digital Module	5 input channels	8 input channels	5 input channels	8 input channels
Instruction Set	35 instructions	35 instructions	35 instructions	35 instructions

Algunos μ C de 14 bits (12F y 16F)



Estructura general de un programa

CONFIGURACIÓN

Elegir los pines E/S de acuerdo a un esquema y los periféricos a utilizar en la aplicación (timers, puerto serie etc).

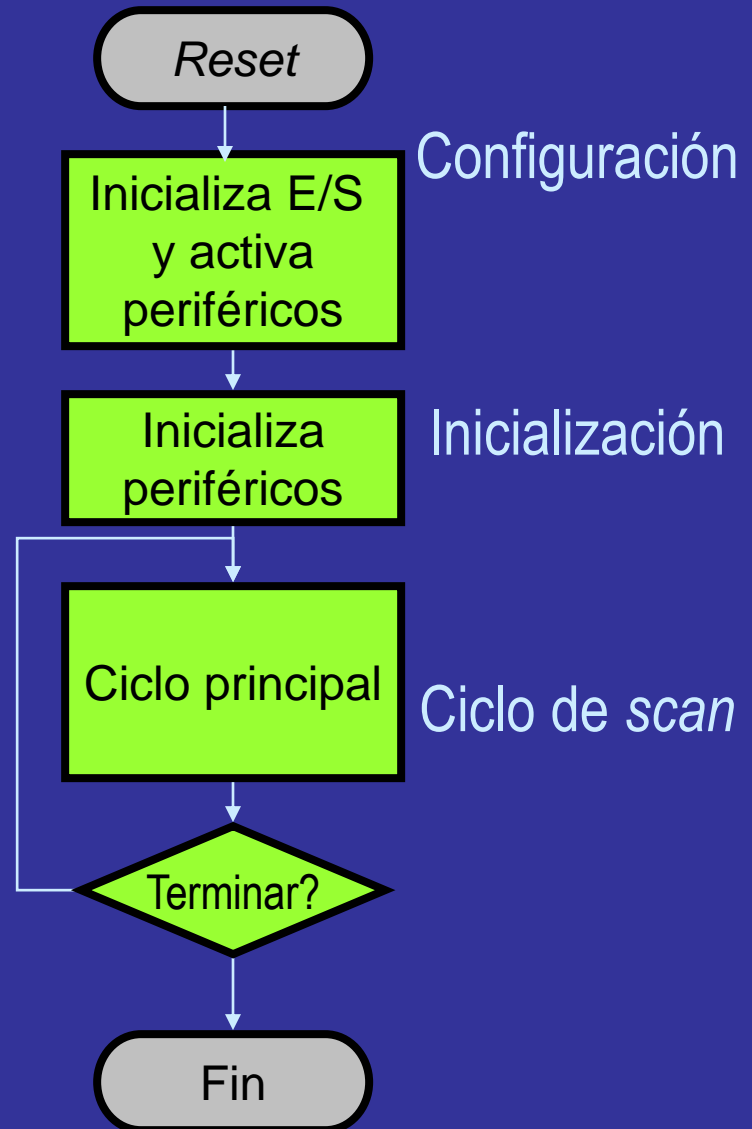
Escribir los registros de activación y configuración de E/S y periféricos

INICIALIZACIÓN

Poner los periféricos y las E/S en un estado inicial adecuado (ej. las salidas en '0')

CICLO DE SCAN

El programa principal, que por lo general se ejecuta cíclicamente.



Lenguajes de Programación de μ C

ALTO NIVEL (C, Basic, Pascal)	Ensamblador (assembler)
<ul style="list-style-type: none">• Requiere compilador	<ul style="list-style-type: none">• Traducción directa a lenguaje de máquina.
<ul style="list-style-type: none">• Código portable	<ul style="list-style-type: none">• Específico del Hardware
<ul style="list-style-type: none">• Comprensible. Facilidad para trabajar programas grandes, operaciones aritméticas y de formateo de datos.	<ul style="list-style-type: none">• Difícil seguimiento y concepción de programas grandes
<ul style="list-style-type: none">• Los compiladores incluyen funciones para manejo de periféricos.	<ul style="list-style-type: none">• Debe manipularse los registros que controlan cada periférico.
<ul style="list-style-type: none">• Menos eficiente en velocidad y en tamaño del código.	<ul style="list-style-type: none">• Puede optimizarse en velocidad y tamaño.

Se suele utilizar lenguaje de alto nivel,
con rutinas críticas en *assembler*

PC

Lenguaje C Ej01.C

```
do
{
    // llave abr
    A=atoi(getc()); //recibe dat
    B=atoi(getc()); //recibe dat
    C=A+B;           //suma A y B
    printf("La suma es: %d",C);
}while(C!=0);       // el ciclo
```

Lenguaje Assembler Ej01.ASM

```
01DD: MOUWF 0x2A
01DE: CALL 0068
01DF: MOVF 0x78,W
01E0: MOUWF 0x27
01E1: MOVF 0x27,W
01E2: ADDWF 0x26,W
01E3: MOUWF 0x28
01E4: CLRF 0x29
01E5: MOVF 0x29,W
01E6: CALL 0068
```

compilar

Lenguaje Máquina Ej01.HEX

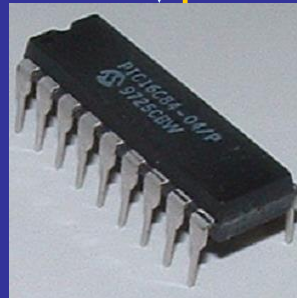
```
0030307303F00831083111
0A1010130A200A301A4017
0C03908388100831290010
0003083169200073083129
01030F700F70BCA2900008
083169F018312172078087
07808A60017207808A9007
0A70027082607A800A9017
0F700AC0027210C3029020
0AA001830AB006229A8080
0D22D6200A6
```

ensamblar

desensamblar

grabar

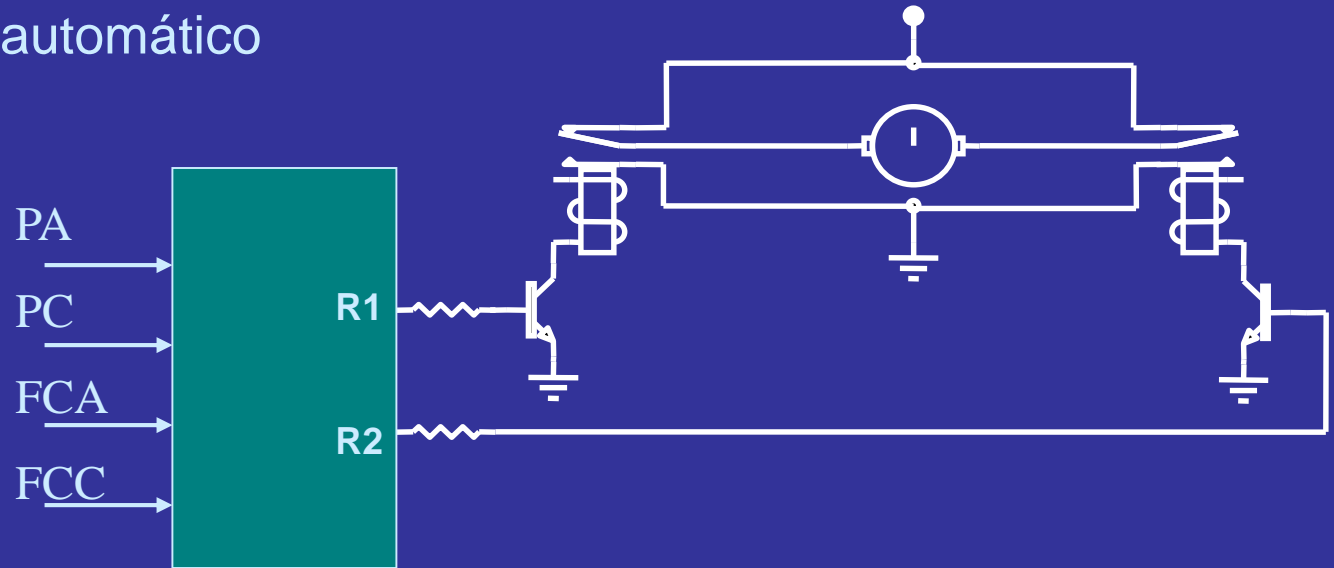
leer



Grabador
ó ISP

Programación de un automatismo

Ejemplo: Portón automático



PA: Pulsador para abrir

PC: Pulsador para cerrar

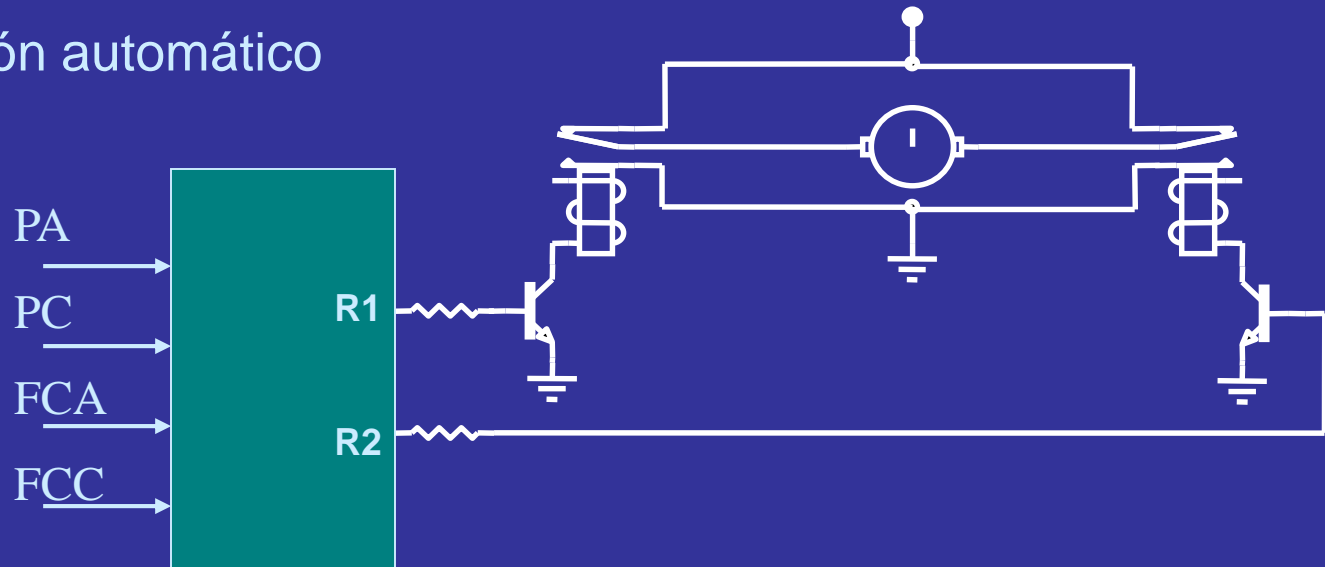
FCA: Final de carrera al abrir

FCC: Final de carrera al cerrar

R1, R2: Relés para giro directo-inverso de motor

Programación de un automatismo

Ejemplo: Portón automático



Comportamiento del automatismo:

El portón se encuentra en Reposo.

Si se pulsa PA el portón se debe abrir, y parar cuando se detecta FCA.

Si se pulsa PC el portón se debe cerrar, y parar cuando se detecta FCC.

Opciones:

¿Se supone el portón inicialmente cerrado? ¿Se cierra inicialmente?

¿Mientras el portón se está moviendo ignora los pulsadores? (SI/NO)

Diagrama de Estados

Leyendo la descripción del funcionamiento se reconocen 3 estados.

Reposo: El motor está detenido (portón abierto o cerrado). $R1=R2=0$

Abre: El motor gira en sentido de abrir portón $R1=1$ $R2=0$

Cierra: El motor gira en sentido de cerrar portón $R1=0$ $R2=1$

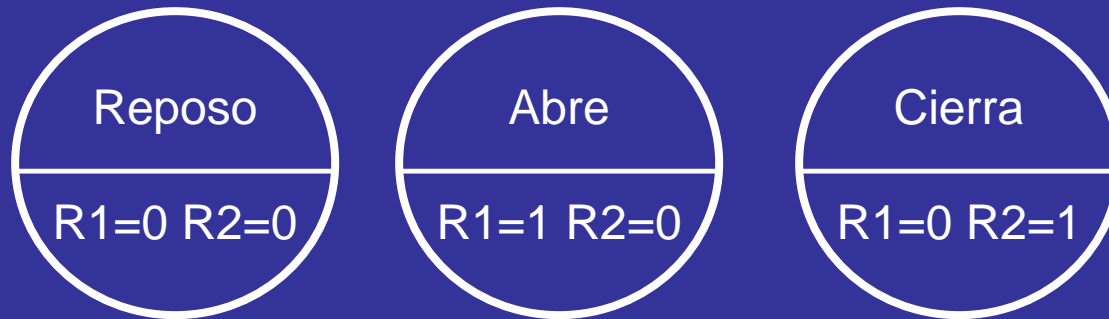


Diagrama de Estados (2)

Ejemplo: Portón automático

Opción: Supone portón inicialmente cerrado

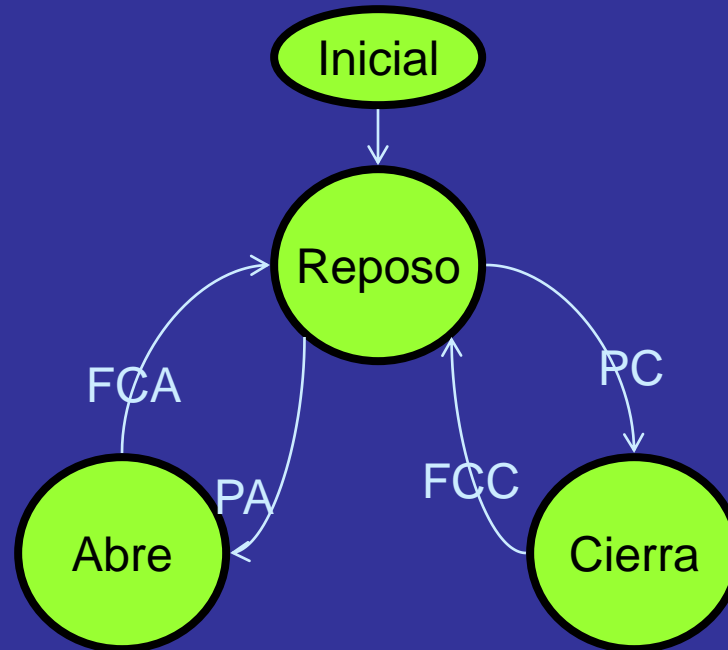


Diagrama de Estados (3)

Ejemplo: Portón automático

Opción: Inicialmente se cierra

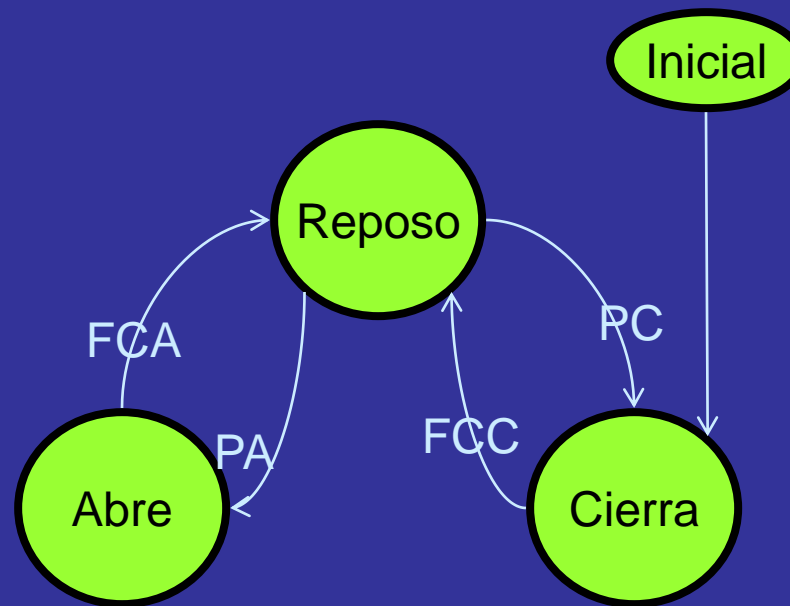


Diagrama de Estados (4)

Ejemplo: Portón automático

Opción: Atiende pulsadores para invertir marcha

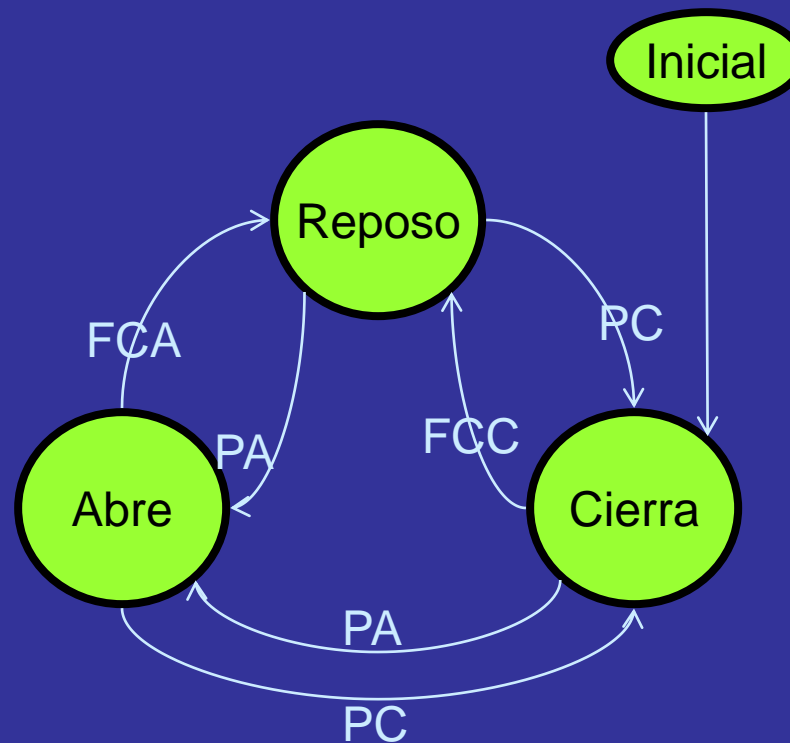


Diagrama de Estados (5)

Ejemplo: Portón automático

Opción: Atiende pulsadores para invertir marcha pero previamente pasa por Reposo

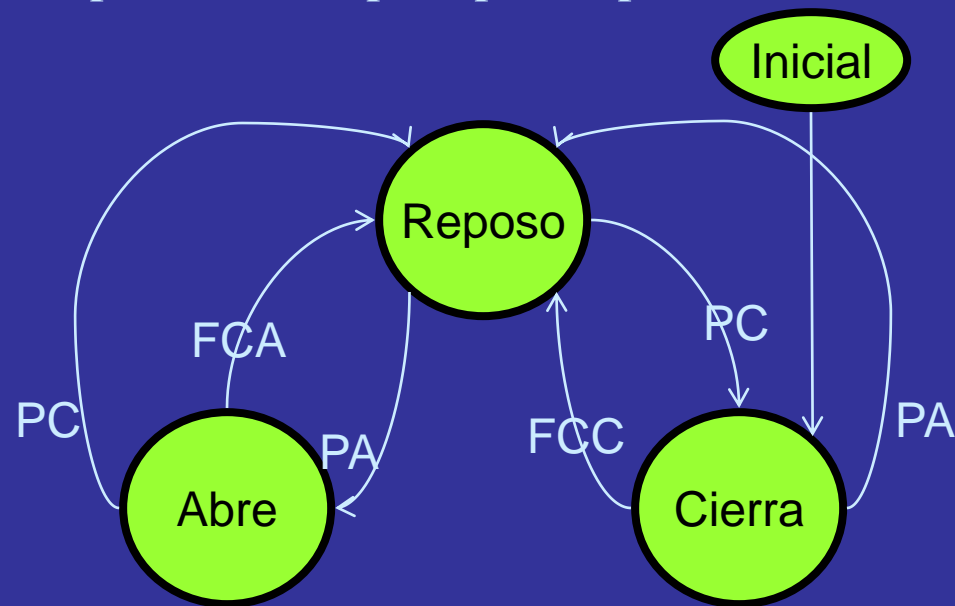
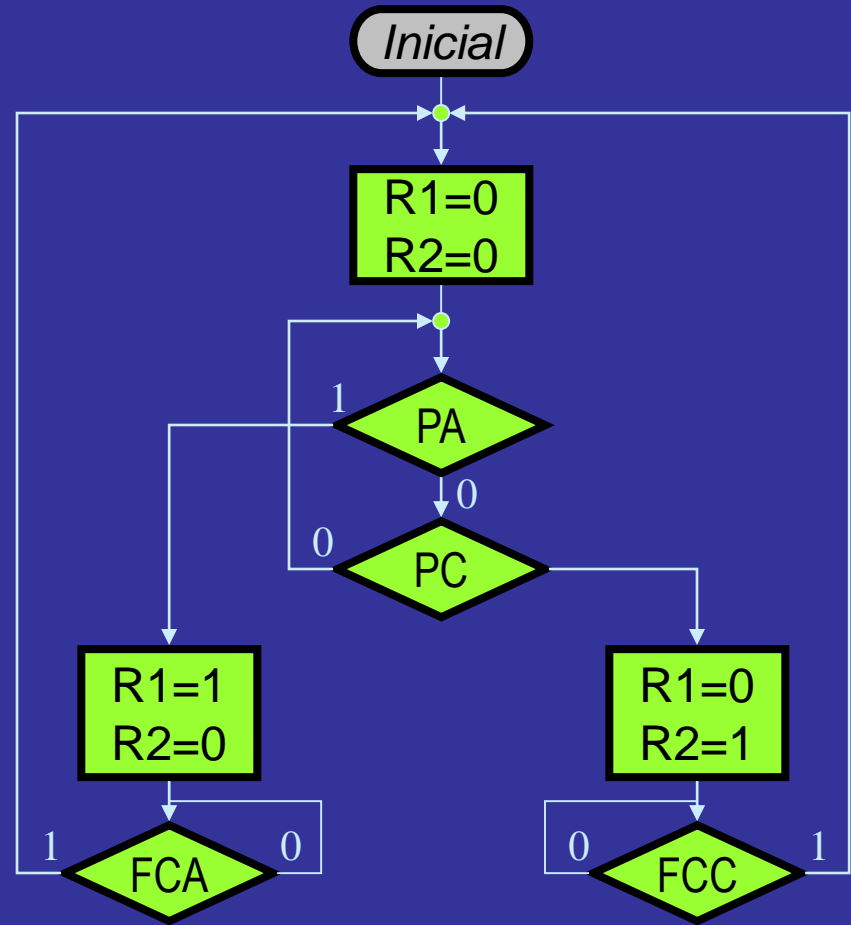
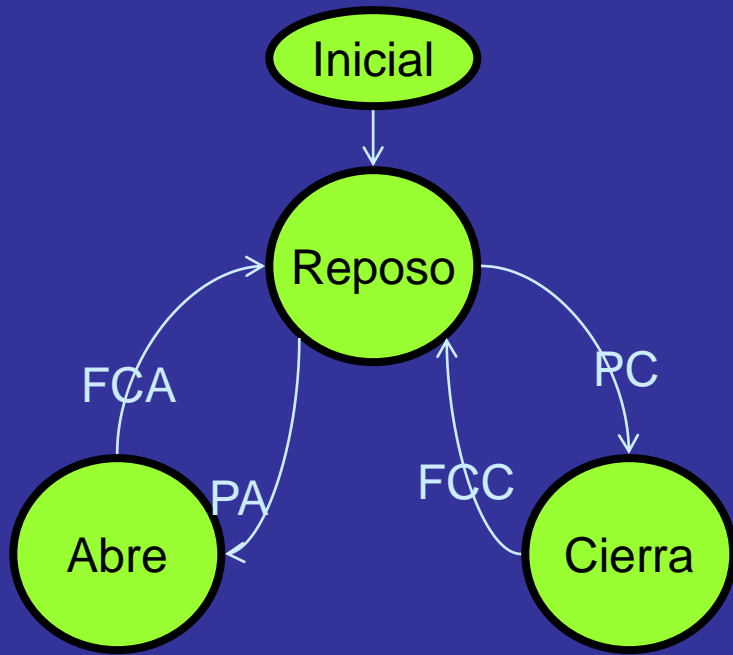
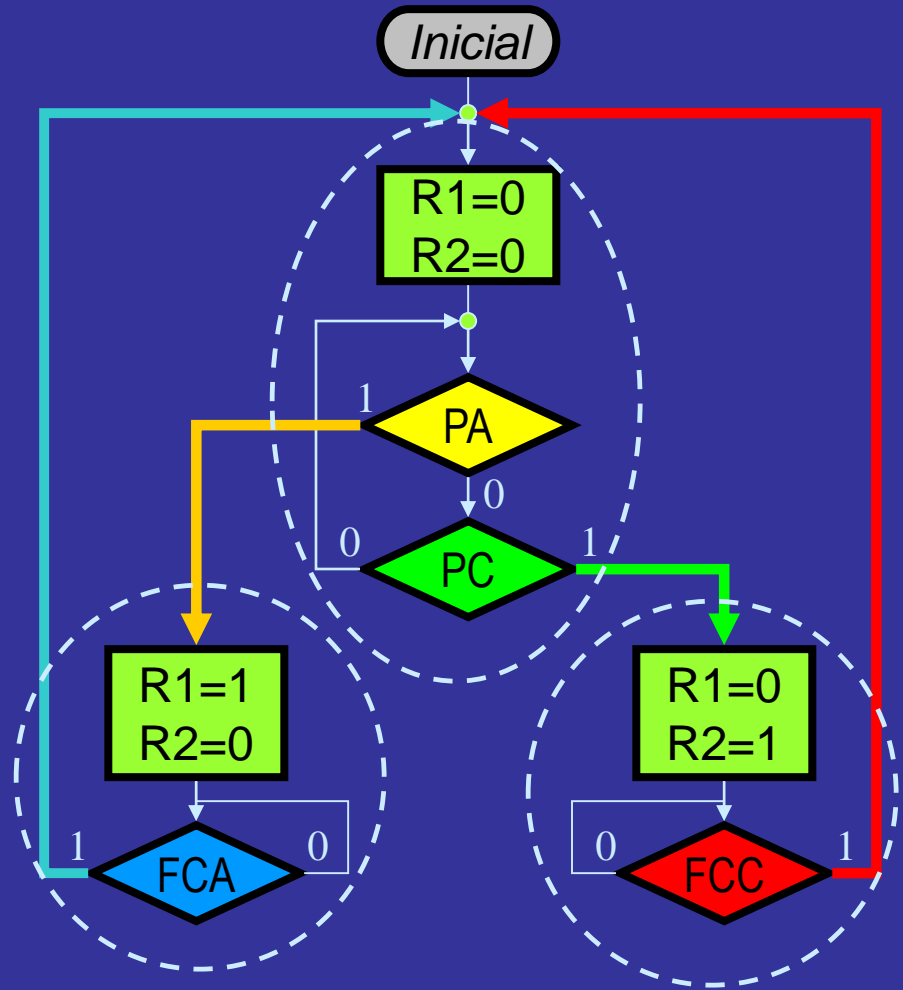
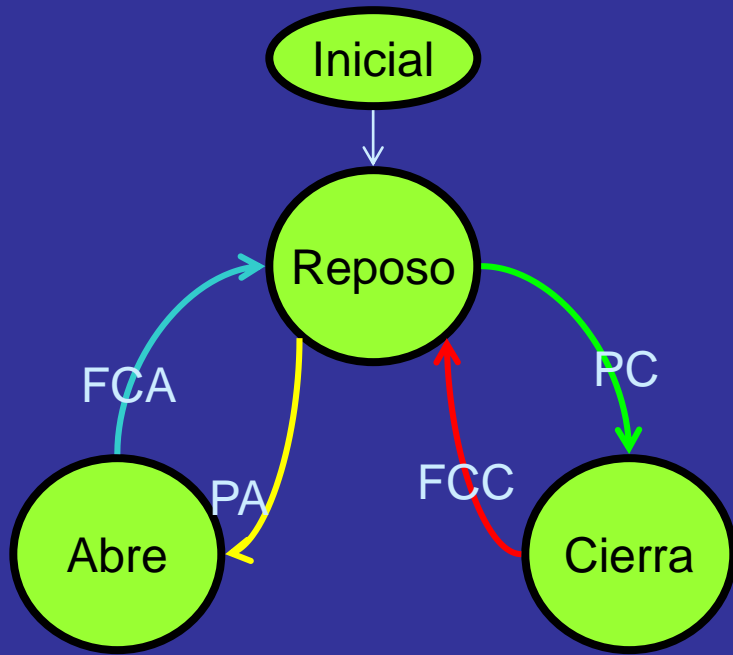


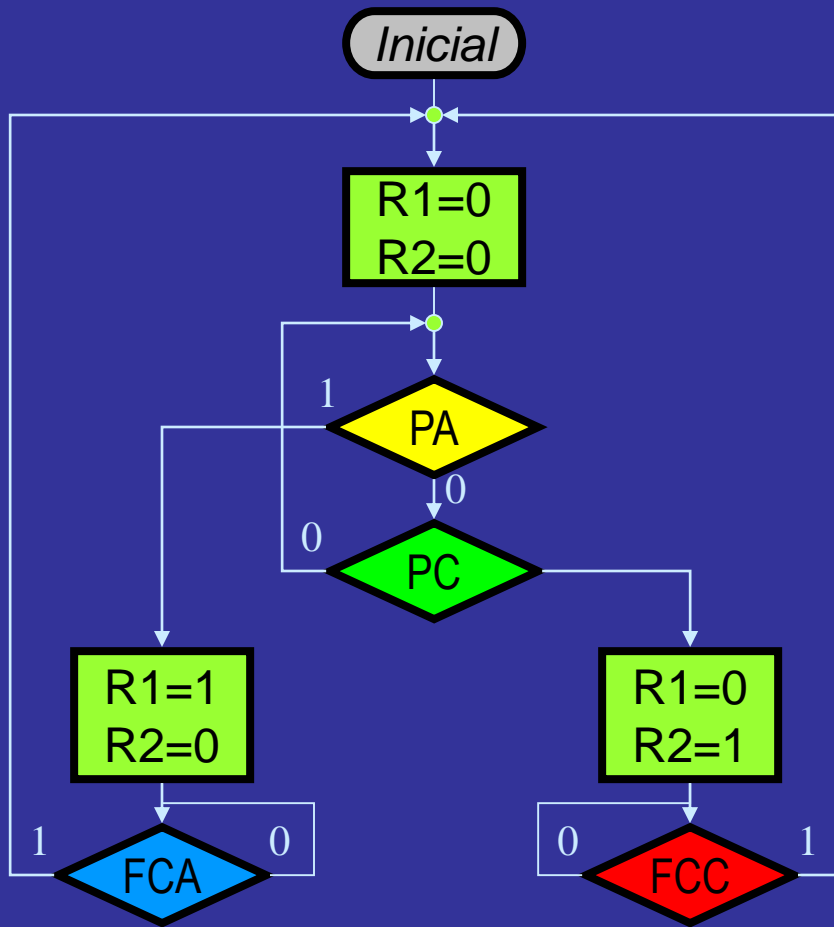
Diagrama de Flujo



Relación entre Diagramas



Codificación en BASIC



REPOSO: R1=0
R2=0

REPOSO2: IF PA=1 THEN
GOTO ABRE
END IF
IF PC=1 THEN
GOTO CIERRA
END IF
GOTO REPOSO2

ABRE: R1=1
R2=0

ABRE2: IF FCA=1 THEN
GOTO REPOSO
END IF
GOTO ABRE2

CIERRA: R1=0
R2=1

CIERRA2: IF FCC=1 THEN
GOTO REPOSO
END IF
GOTO CIERRA2

Conceptos básicos de desarrollo en entorno PICC

PC

Lenguaje C
Ej01.C

Código
Fuente

```
do
{
    // llave abr
    A=atoi(getc()); //recibe dat
    B=atoi(getc()); //recibe dat
    C=A+B;           //suma A y B
    printf("La suma es: %d",C);
}while(C!=0);       // el ciclo
```

Lenguaje Assembler
Ej01.ASM

Código
Fuente

```
01DD: MOUWF 0x2A
01DE: CALL 0068
01DF: MOVF 0x78,W
01E0: MOUWF 0x27
01E1: MOVF 0x27,W
01E2: ADDWF 0x26,W
01E3: MOUWF 0x28
01E4: CLRF 0x29
01E5: MOVF 0x29,W
01E6: CALL 0068
```

compilar

Lenguaje Máquina
Ej01.HEX

Código Objeto

```
0030307303F00831083111
0A1010130A200A301A4017
0C03908388100831290010
0003083169200073083129
01030F700F70BCA2900008
083169F018312172078087
07808A60017207808A9007
0A70027082607A800A9017
0F700AC0027210C3029020
0AA001830AB006229A8080
0D22D6200000
```

ensamblar

desensamblar

grabar

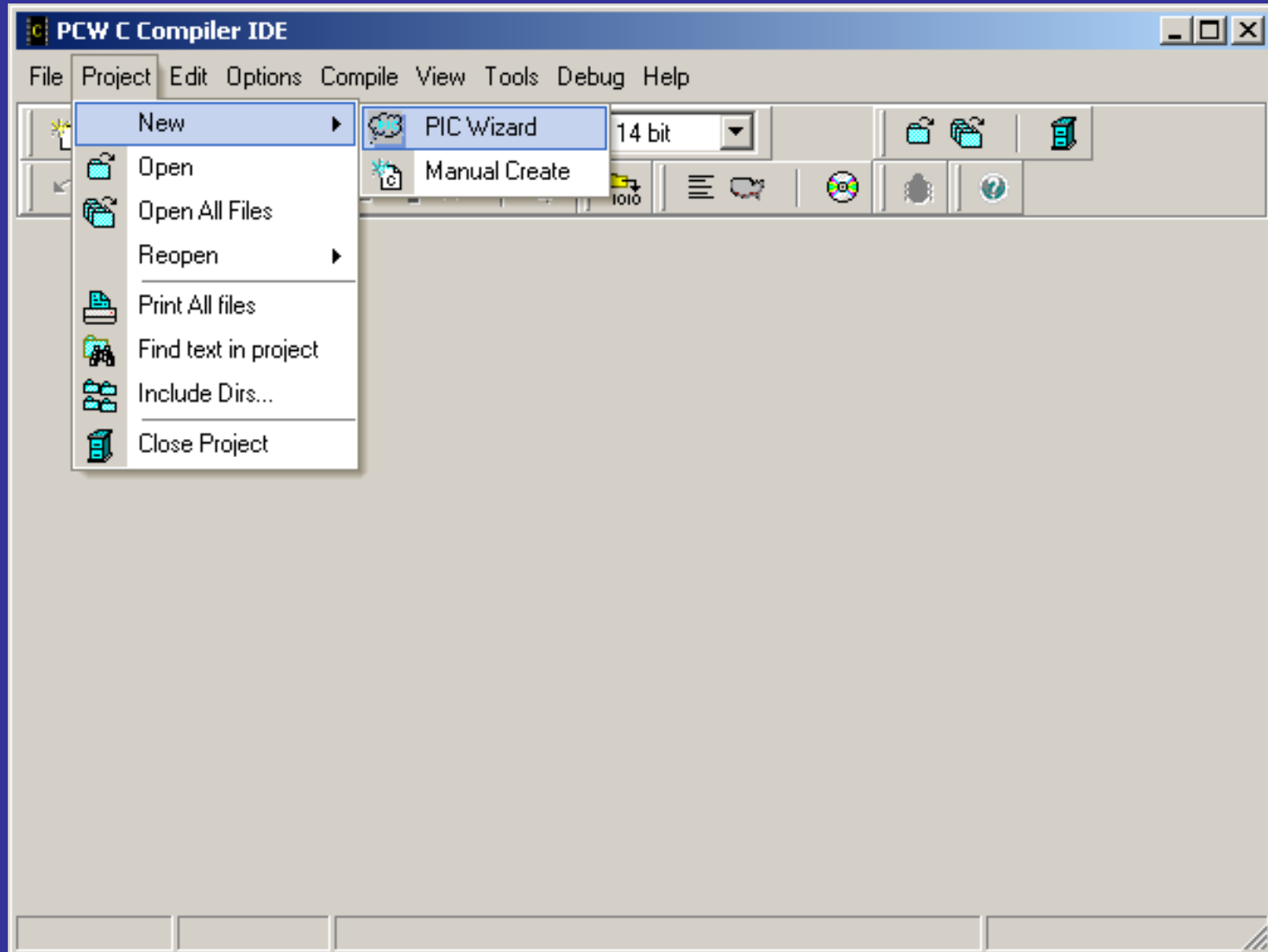
leer

Grabador
ó ISP



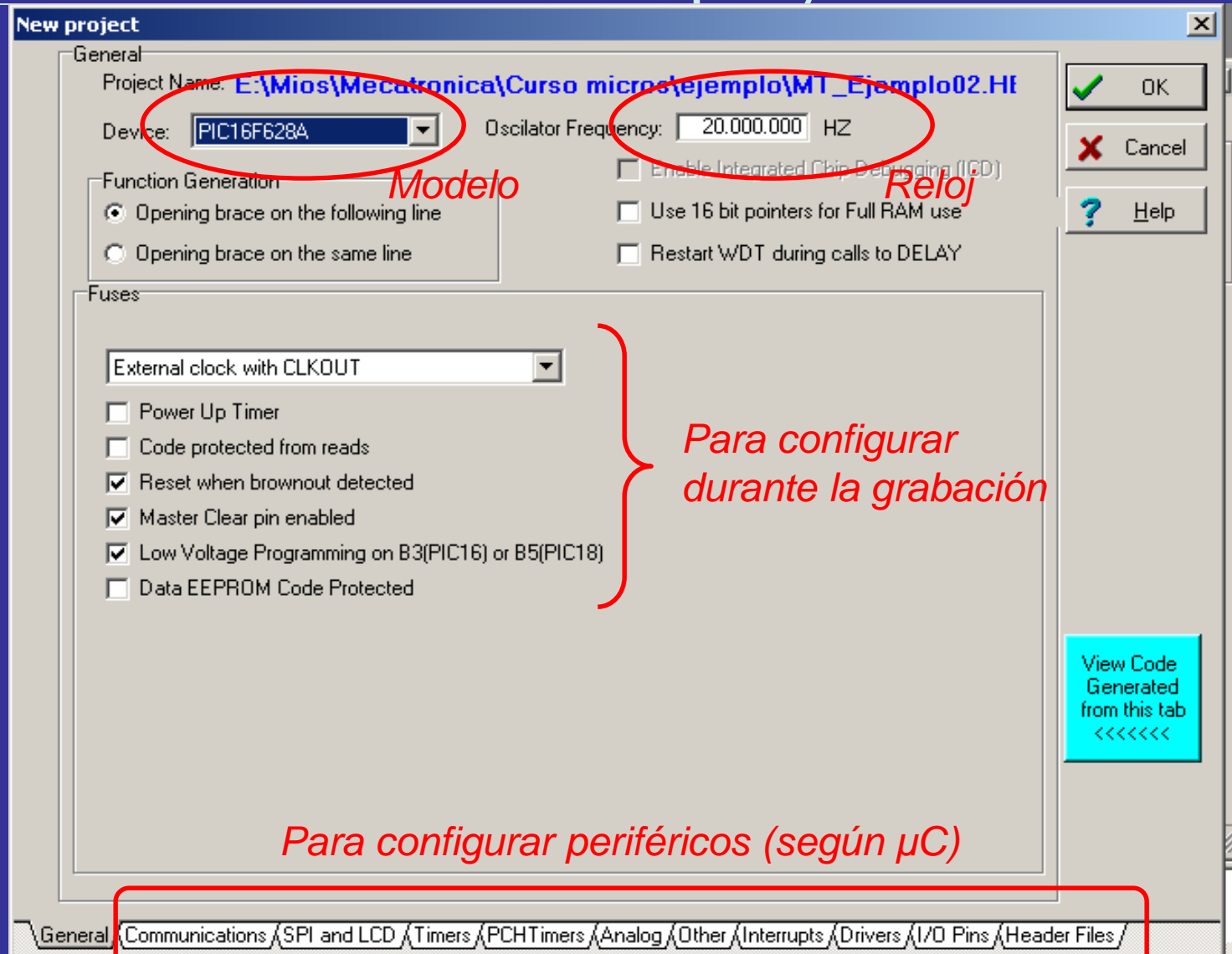
Entorno de Desarrollo (IDE) del PICC

Creación de proyecto



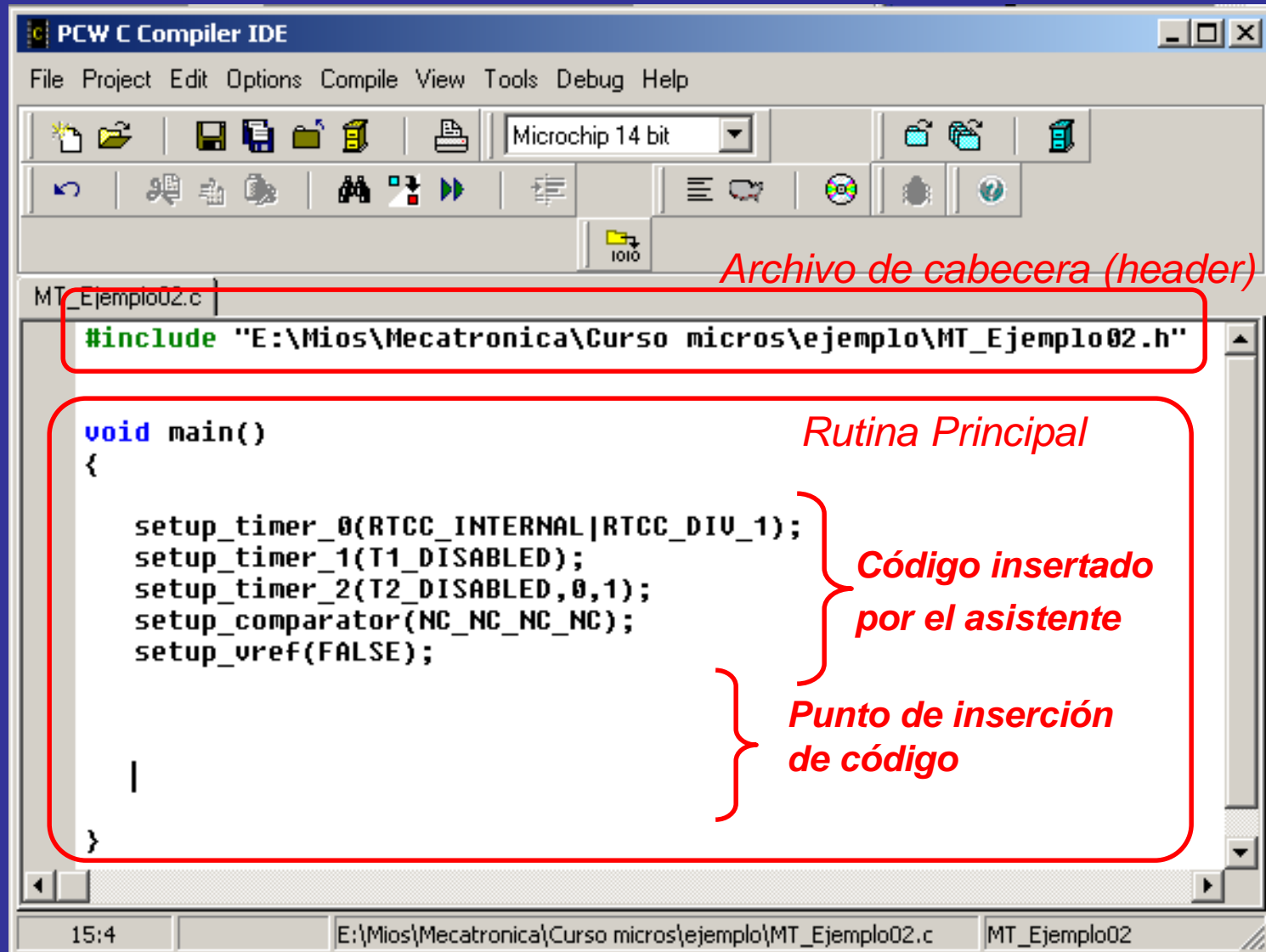
Entorno de Desarrollo (IDE) del PICC

Creación de proyecto



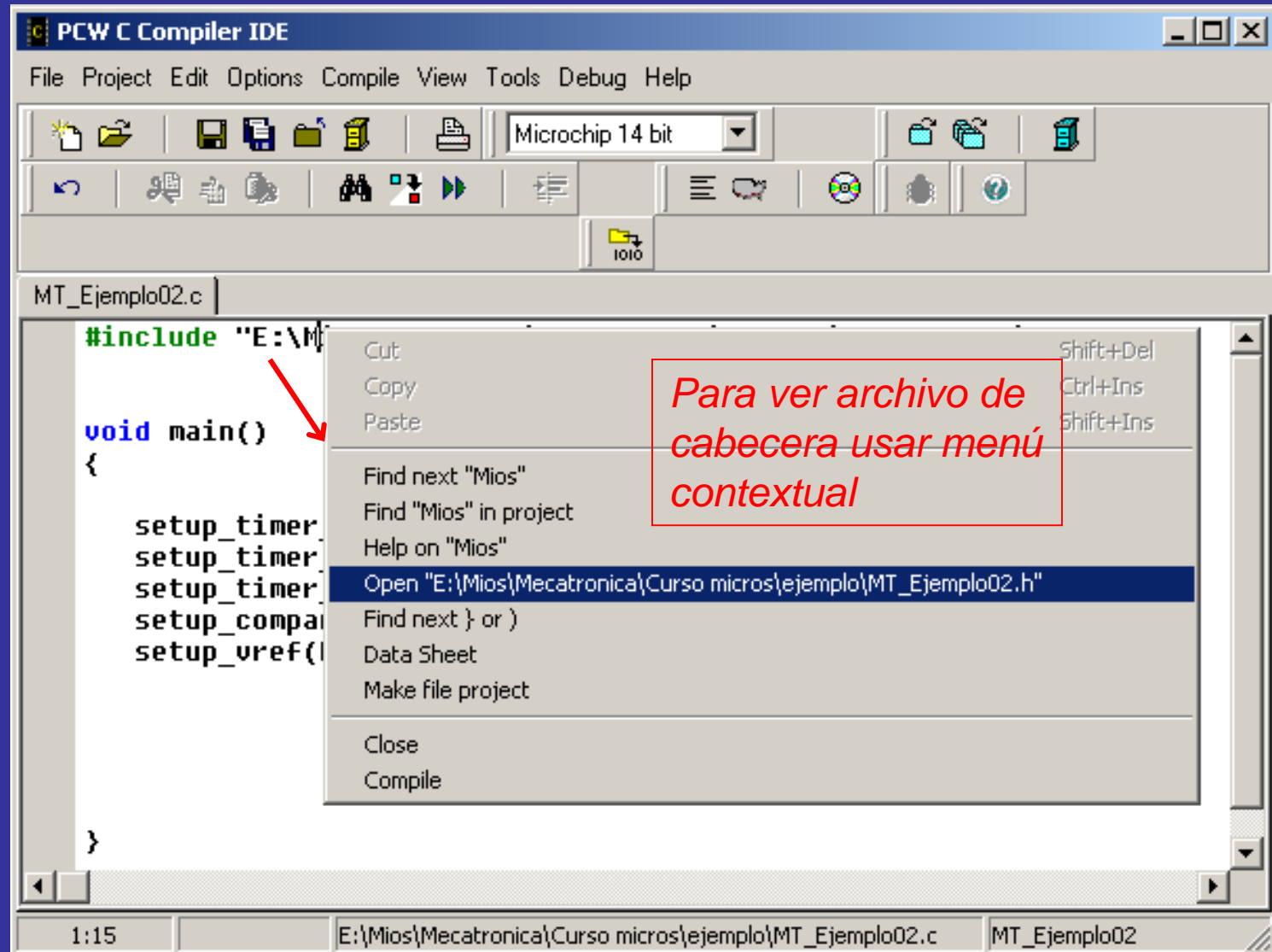
Entorno de Desarrollo (IDE) del PICC

Archivo principal



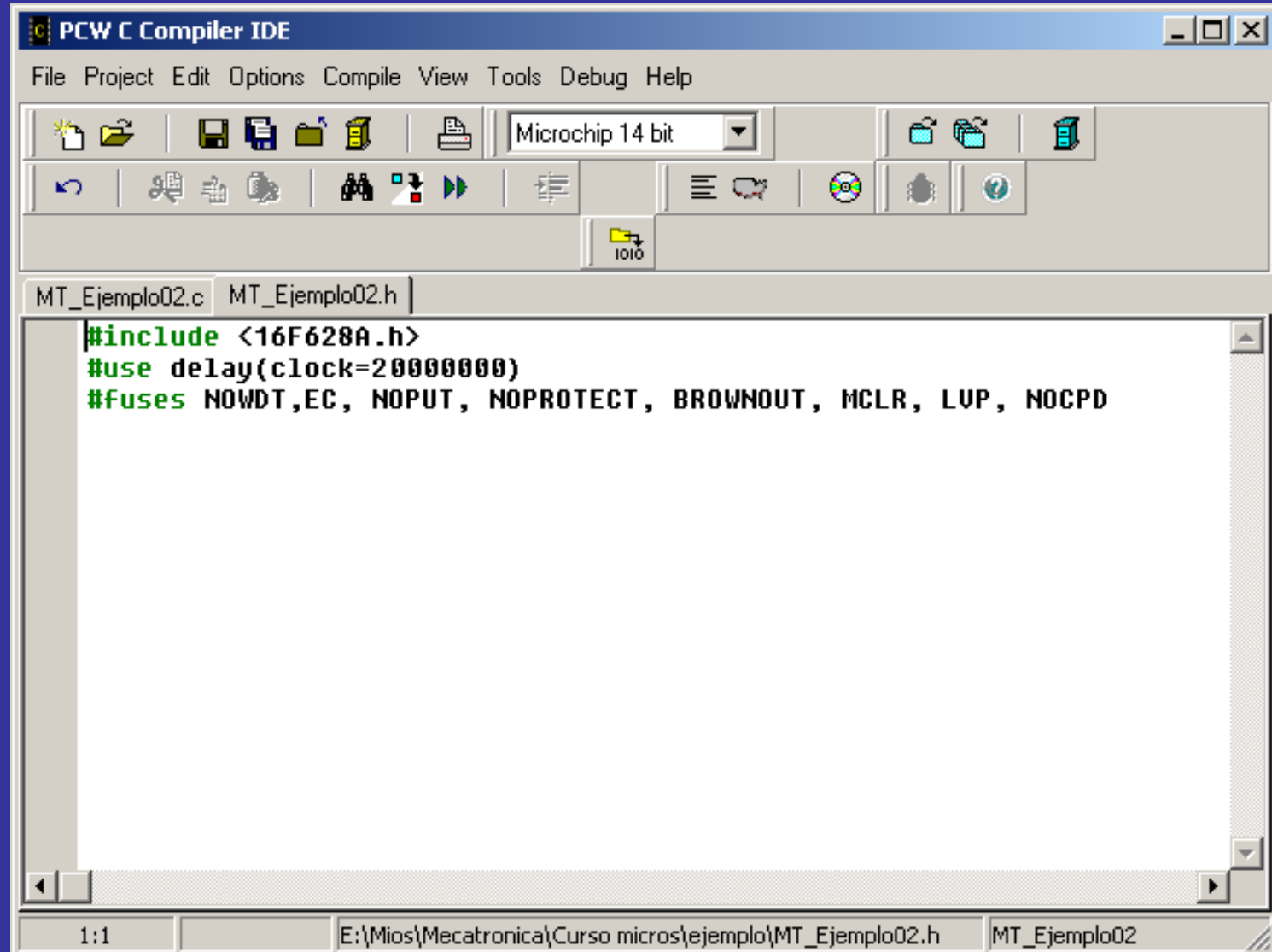
Entorno de Desarrollo (IDE) del PICC

Archivo cabecera

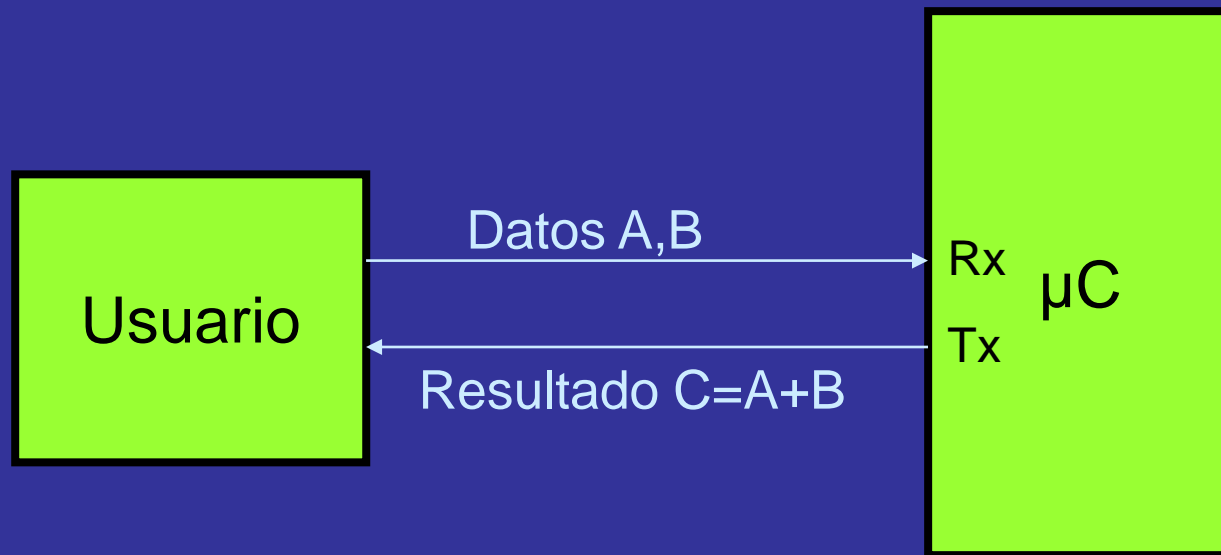


Entorno de Desarrollo (IDE) del PICC

Archivo cabecera



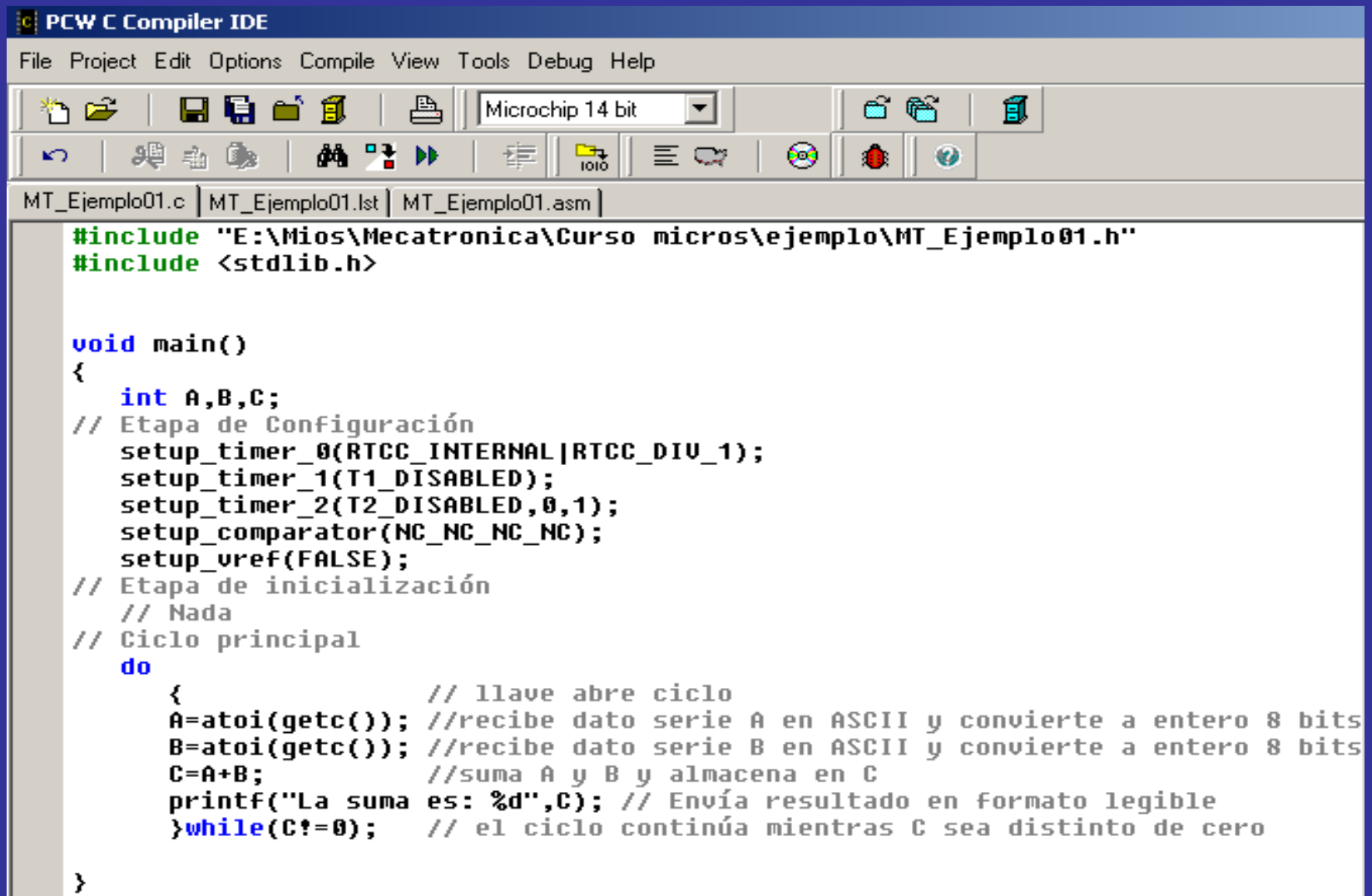
Ejemplo: Sumador por puerto serie



A través de Rx (receptor serie) el microcontrolador recibe los datos A y B, los suma y transmite el resultado por Tx (transmisor serie)

Ejemplo: Sumador por puerto serie

Programa en C



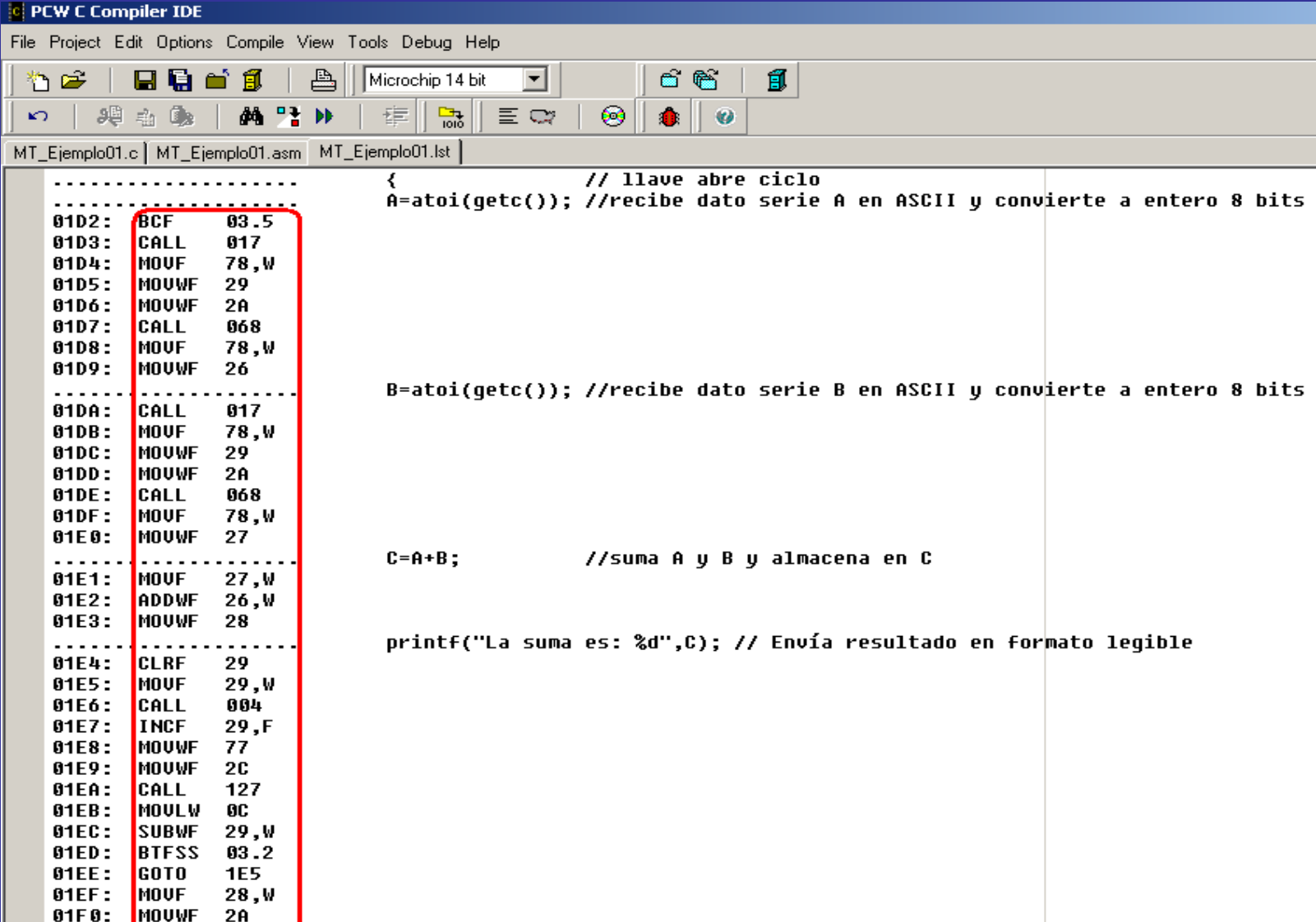
The image shows a screenshot of the PCW C Compiler IDE. The window title is "PCW C Compiler IDE". The menu bar includes "File", "Project", "Edit", "Options", "Compile", "View", "Tools", "Debug", and "Help". The toolbar contains various icons for file operations, compilation, and debugging. The status bar at the bottom shows the current file is "MT_Ejemplo01.c". The main editor area displays the following C code:

```
#include "E:\Mios\Mecatronica\Curso micros\ejemplo\MT_Ejemplo01.h"
#include <stdlib.h>

void main()
{
    int A,B,C;
    // Etapa de Configuración
    setup_timer_0(RTCC_INTERNAL|RTCC_DIV_1);
    setup_timer_1(T1_DISABLED);
    setup_timer_2(T2_DISABLED,0,1);
    setup_comparator(NC_NC_NC_NC);
    setup_vref(FALSE);
    // Etapa de inicialización
    // Nada
    // Ciclo principal
    do
    {
        // llave abre ciclo
        A=atoi(getc()); //recibe dato serie A en ASCII y convierte a entero 8 bits
        B=atoi(getc()); //recibe dato serie B en ASCII y convierte a entero 8 bits
        C=A+B;           //suma A y B y almacena en C
        printf("La suma es: %d",C); // Envía resultado en formato legible
    }while(C!=0);        // el ciclo continúa mientras C sea distinto de cero
}
```

Ejemplo: Sumador por puerto serie

Programa en *assembler* (fragmento)



```
PCW C Compiler IDE
File Project Edit Options Compile View Tools Debug Help
Microchip 14 bit
MT_Ejemplo01.c MT_Ejemplo01.asm MT_Ejemplo01.lst

{ // llave abre ciclo
A=atoi(getc()); //recibe dato serie A en ASCII y convierte a entero 8 bits

01D2: BCF 03.5
01D3: CALL 017
01D4: MOVF 78,W
01D5: MOUWF 29
01D6: MOUWF 2A
01D7: CALL 068
01D8: MOVF 78,W
01D9: MOUWF 26

B=atoi(getc()); //recibe dato serie B en ASCII y convierte a entero 8 bits

01DA: CALL 017
01DB: MOVF 78,W
01DC: MOUWF 29
01DD: MOUWF 2A
01DE: CALL 068
01DF: MOVF 78,W
01E0: MOUWF 27

C=A+B; //suma A y B y almacena en C

01E1: MOVF 27,W
01E2: ADDWF 26,W
01E3: MOUWF 28

printf("La suma es: %d",C); // Envía resultado en formato legible

01E4: CLRF 29
01E5: MOVF 29,W
01E6: CALL 004
01E7: INCF 29,F
01E8: MOUWF 77
01E9: MOUWF 2C
01EA: CALL 127
01EB: MOULW 0C
01EC: SUBWF 29,W
01ED: BTFSS 03.2
01EE: GOTO 1E5
01EF: MOVF 28,W
01F0: MOUWF 2A
```

Ejemplo: Sumador por puerto serie

Código de máquina

PCW C Compiler IDE

File Project Edit Options Compile View Tools Debug Help

Microchip 14 bit

MT_Ejemplo01.c MT_Ejemplo01.asm MT_Ejemplo01.lst

..... { // llave abre ciclo

01D2: BCF 03.5
 01D3: CALL 017
 01D4: MOVF 78,W
 01D5: MOUWF 29
 01D6: MOUWF 2A
 01D7: CALL 068
 01D8: MOVF 78,W
 01D9: MOUWF 26

.....

01DA: CALL 017
 01DB: MOVF 78,W
 01DC: MOUWF 29
 01DD: MOUWF 2A
 01DE: CALL 068
 01DF: MOVF 78,W
 01E0: MOUWF 27

.....

01E1: MOVF 27,W
 01E2: ADDWF 26,W
 01E3: MOUWF 28

.....

01E4: CLRF 29
 01E5: MOVF 29,W
 01E6: CALL 004
 01E7: INCF 29,F
 01E8: MOUWF 77
 01E9: MOUWF 2C
 01EA: CALL 127
 01EB: MOVLW 0C
 01EC: SUBWF 29,W
 01ED: BTFSS 03.2
 01EE: GOTO 1E5
 01EF: MOVF 28,W
 01F0: MOUWF 2A

01DA: 2017 CALL 0017
 01DB: 0878 MOVF 0x78,W
 01DC: 00A9 MOUWF 0x29
 01DD: 00AA MOUWF 0x2A
 01DE: 2068 CALL 0068
 01DF: 0878 MOVF 0x78,W
 01E0: 00A7 MOUWF 0x27
 01E1: 0827 MOVF 0x27,W
 01E2: 0726 ADDWF 0x26,W
 01E3: 0000 MOUWF 0x00

convierte a entero 8 bits

convierte a entero 8 bits

01E1:	0827	MOVF	0x27,W
Dirección en ROM	Código de máquina	Equivalente <i>Assembler</i>	

00 1000 0010 0111
 0 8 2 7

Ejemplo: Sumador por puerto serie

Archivo .HEX para grabar microcontrolador

```
01DA: 2017 CALL 0017
01DB: 0878 MOVF 0x78,W
01DC: 00A9 MOUWF 0x29
01DD: 00AA MOUWF 0x2A
01DE: 2068 CALL 0068
01DF: 0878 MOVF 0x78,W
01E0: 00A7 MOUWF 0x27
01E1: 0827 MOVF 0x27,W
01E2: 0726 ADDWF 0x26,W
01E3: 00A8 MOUWF 0x28
01E4: 0100 CLRF 0x20
```

```
:100380000003083169200073083129F008316050801
:100390001030F700F70BCA29000083121F080C1356
:1003A00083169F01831217207808A900AA006820ED
:1003B0007808A60017207808A900AA006820780805
:1003C000A70027082607A800A90129080420A90AD0
:1003D000F700AC0027210C302902031DE52928086D
:1003E000AA001830AB006229A8080319F92983165E
:0402F000002206200AF
```