EMI

- MATERIA : ANALISIS DISENO DE SISTEMAS BASADOS EN MICROPROCESADORES
- CODIGO : SIS – 03 2 12
- DOCENTE : CESAR MARTIN SUAREZ SUAREZ

Unidad de Control

Contenido

- Desarrollo interno de las instrucciones
  - Operaciones elementales
  - Ejecución de instrucciones: Cronogramas
  - Señales de control
- Diseño de la Unidad de Control
  - Cableada
  - Microprogramada
- Interrupciones y excepciones

Funciones de la U.C.

1) **Ejecutar la siguiente secuencia**:
   - Leer de la memoria principal la instrucción apuntada por el CP.
   - Incrementar el contador de programa (CP).
   - Decodificar la instrucción leída.
   - Hacer que sea ejecutada.
2) **Resolver situaciones anómalas o de conflicto.**
3) **Controlar la comunicación con los periféricos.**

Información de que se vale la UC

- Emplea y modifica el contador de programa
- Averigua qué hacer del código de operación y busca los operandos
- El registro de ESTADO informa de resultados de operaciones
  - Puede haber interrupciones de secuencia
- El contador de periodos (accionado por el reloj, CLK)
- Las señales de control y estado externas a la CPU

Operaciones elementales

- La ejecución de una instrucción necesita realizar una serie de pequeños pasos llamados *operaciones elementales*
- Le ejecución de cada operación elemental requiere la activación de las correspondientes señales de control
- Las operaciones elementales pueden ser de dos tipos:
  - Operaciones de transferencia
  - Operaciones de proceso

El objetivo de la UC es generar las secuencias de señales de control precisas para la realización de las operaciones elementales de cada instrucción

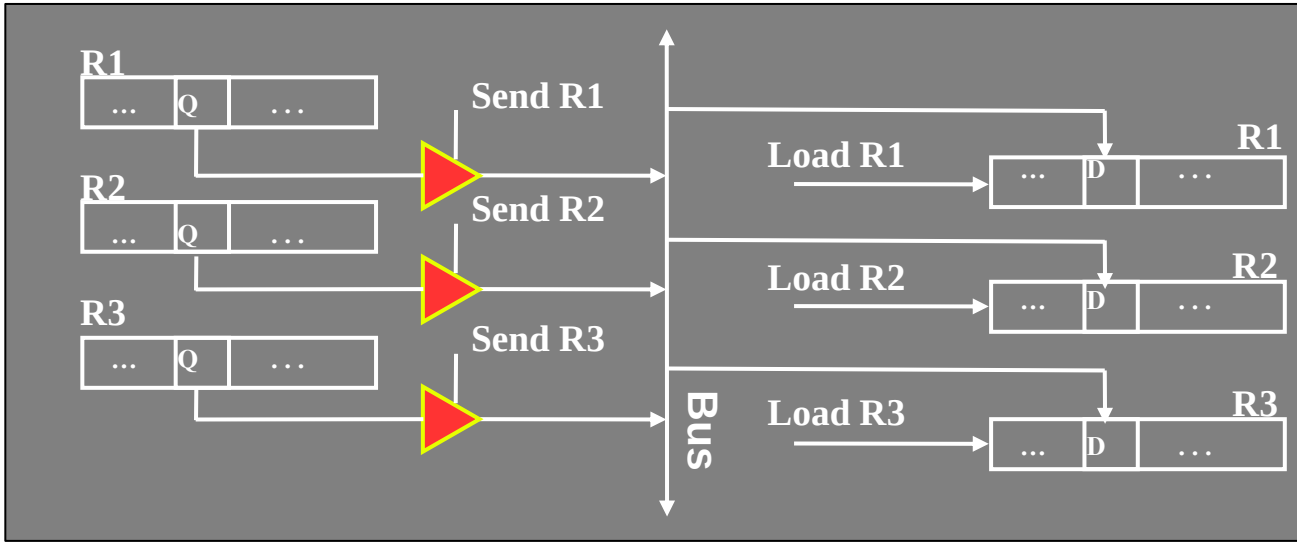
Operaciones de transferencia

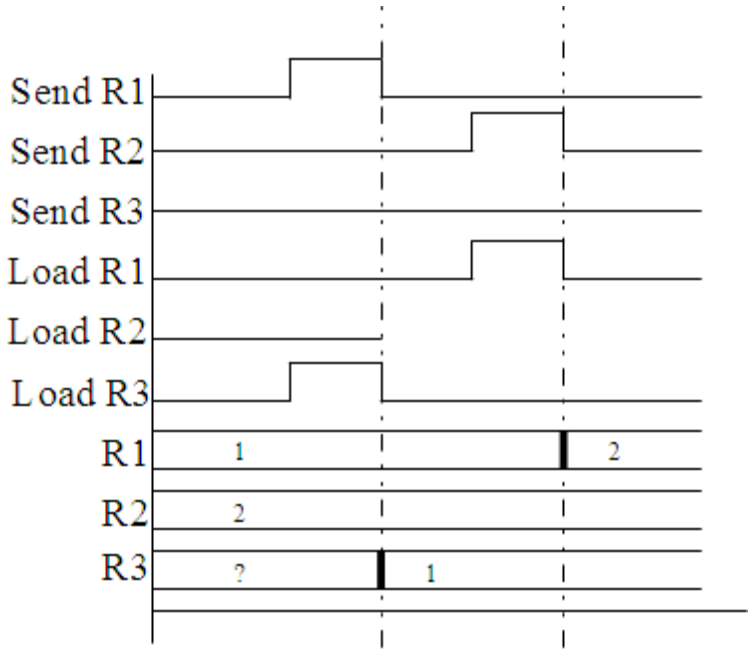
- Se precisan dos elementos de almacenamiento (origen y destino)
- Se ha de establecer previamente un camino físico entre origen y destino
- Enviar una señal al destino para que se “cargue” con lo que tiene a su entrada

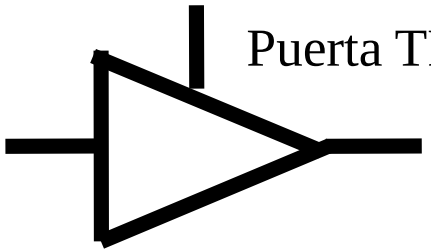
¡La información del origen queda sin modificar!

Operaciones de transferencia (II)

Ejemplo de transferencia elemental por bus







Puerta TRIESTADO

Puertas Triestado y Buses

- Puerta Triestado
  - Los tres estados de salida son: alto, bajo y alta impedancia (alta Z).
  - Cuando se selecciona el funcionamiento lógico normal, mediante la entrada de habilitación, el circuito triestado funciona de la misma forma que una puerta normal.
  - Cuando el modo de funcionamiento es de alta impedancia, la salida se desconecta del resto del circuito.
- Bus:
  - El bus es un sistema digital que transfiere datos entre los componentes de un ordenador o entre ordenadores.
  - Está formado por cables o pistas en un circuito impreso.
  - De datos, de direcciones y de control.

Datos al y desde el bus

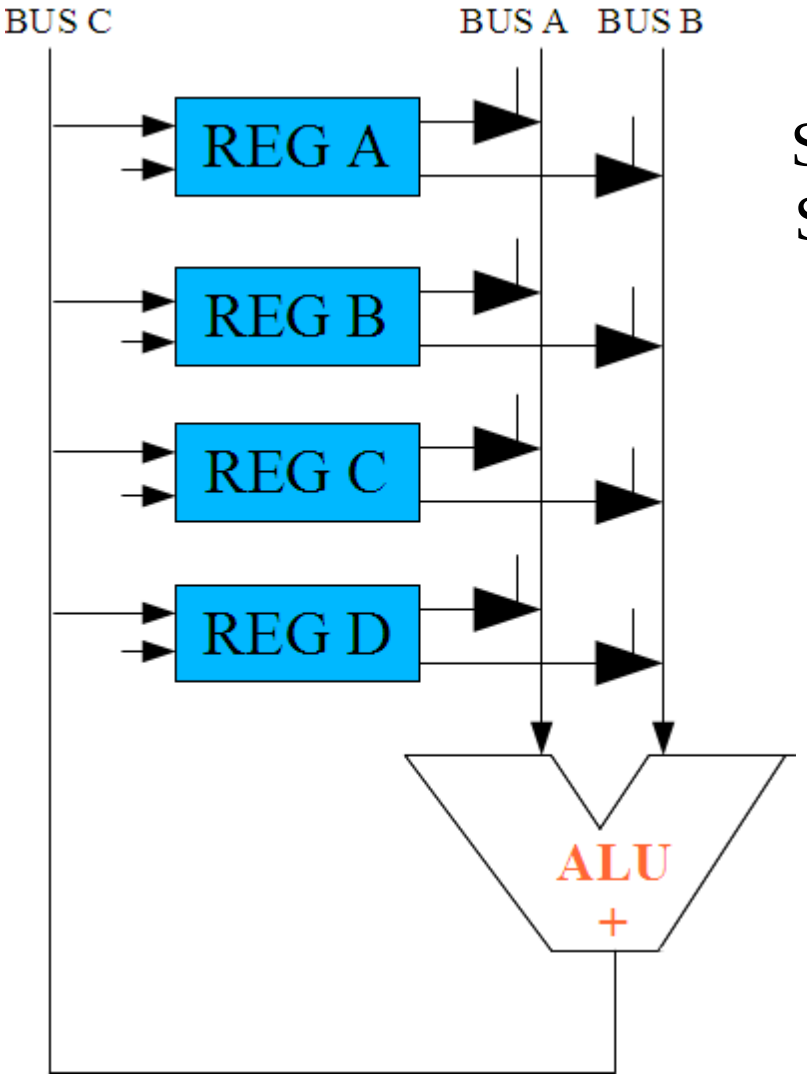
- Volcar datos al bus
  - Se activa la señal de habilitación (SendR1) de la puerta triestado: ***son más de una al mismo tiempo***
  - Se mantiene un tiempo suficiente para que se estabilice: ***es una señal activa por nivel***
- Recoger datos del bus
  - Se activa la señal de carga del registro (LoadR1)
  - Se carga el contenido del bus en ese momento: ***es una señal activa por flanco***

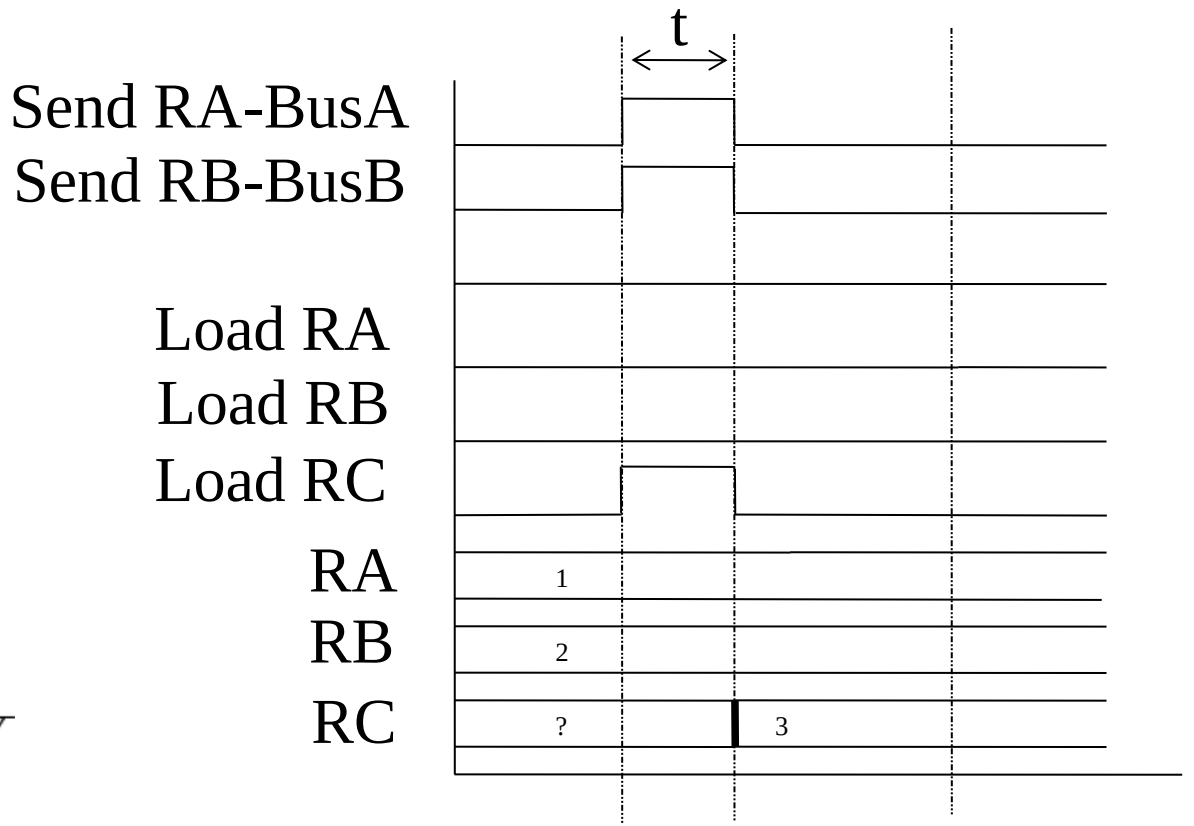
Operación de proceso

- Parecida a la operación de transferencia, pero la información origen se transforma al pasar por un operador combinacional en su camino hacia el destino
- Operaciones de proceso:
  - Diádicas, dos operadores (ejemplo suma)
  - Monódicas, un operador (ejemplo desplazamiento)
  - El operador genera, además del resultado, los bits o flags del registro ESTADO que pueden o no cambiar

Operaciones de proceso (II)

Ejemplo de suma



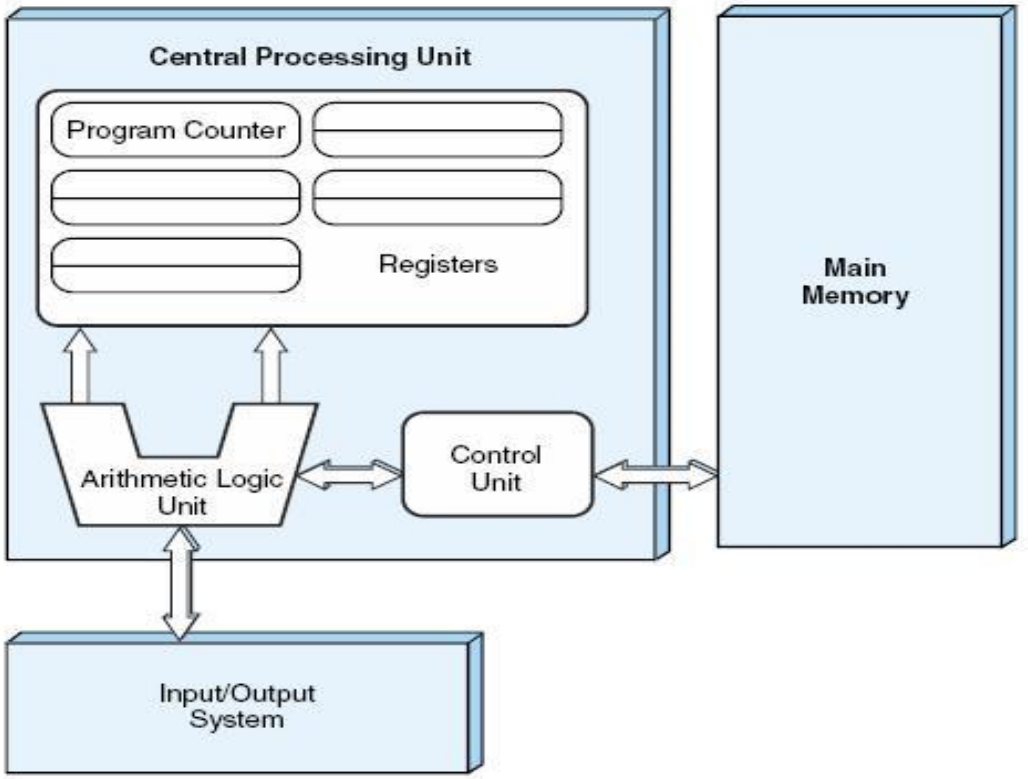


El valor de **t** tiene que ser tal que dé tiempo a la realización de la suma en la ALU

Estructura de un Computador elemental

- Estructura von Neumann elemental
- Ancho de palabra de 16 bits
- Direccionamiento a nivel de byte
- Elementos considerados
  - Memoria principal
  - Unidad aritmética
  - Banco de registros
  - Órganos de control:

    CP, RI, SP, ESTADO, RF...



Registros auxiliares para la gestión de Memoria

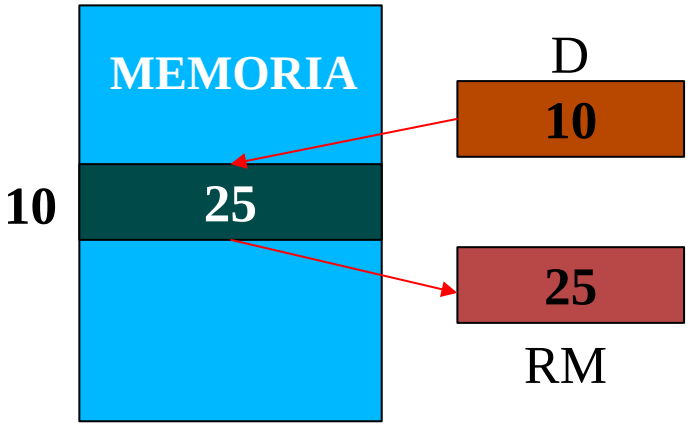
- El Registro de Direcciones (D): almacenará la dirección de memoria a la que se desea acceder.

- El Registro de Datos (RM): almacenará la información que se desea escribir (o leer) en (de) la memoria direccionada.

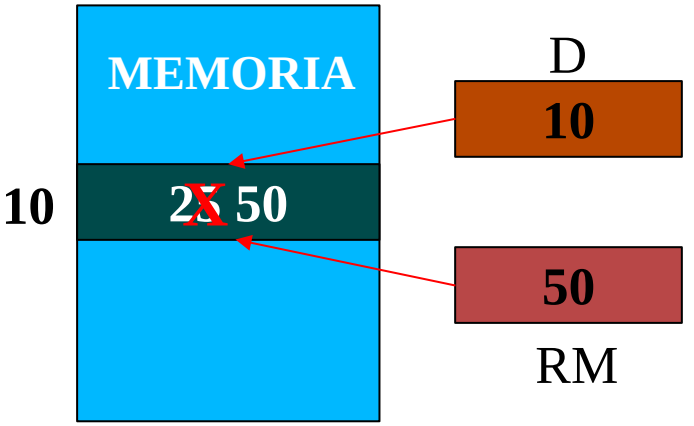
Operaciones básicas con Memoria

Operaciones:

- Lectura:



- Escritura



Ejecución de una instrucción

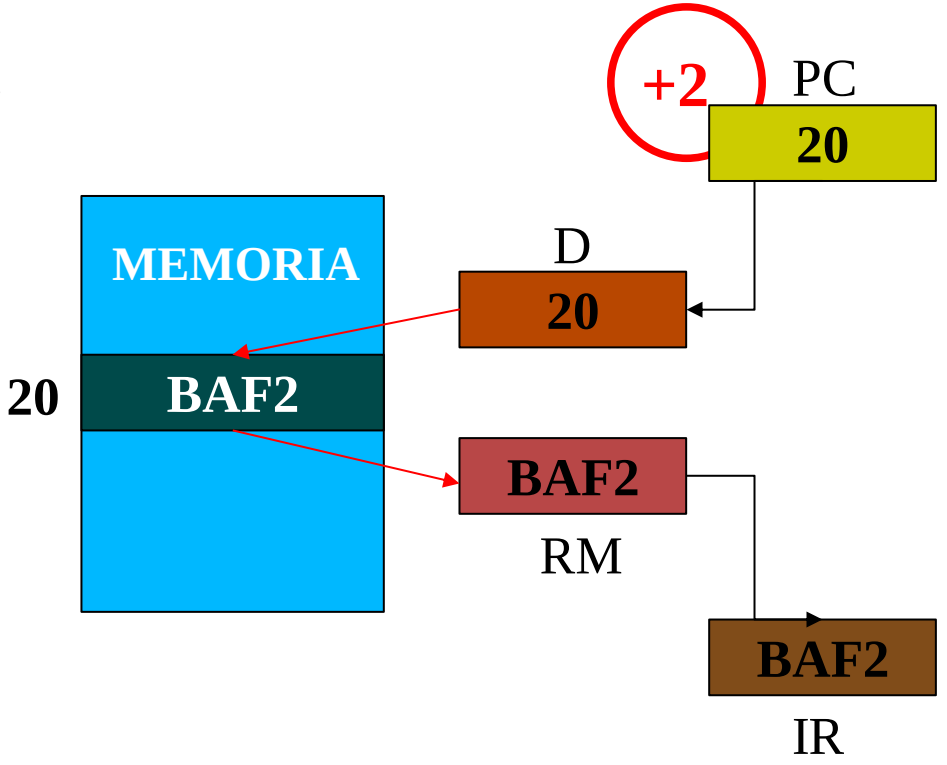
1.- Secuenciador activa circuitos para cargar instrucción en IR.

PC → D
(D) →RM
RM → IR

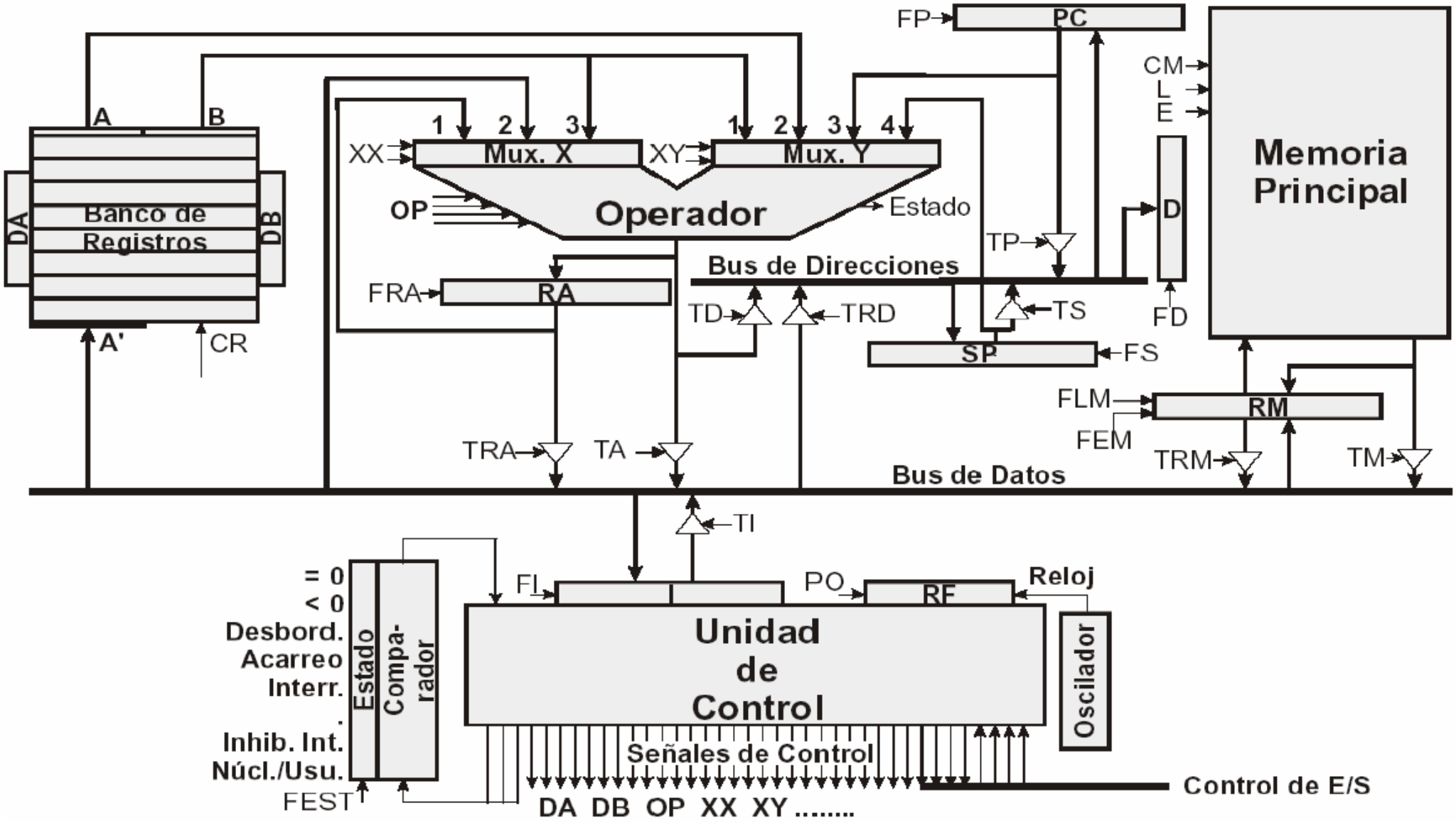
2.- PC + 2 → PC

3.- Secuenciador interpreta esta instrucción, transfiriendo a la ALU datos y operador si es necesario, y almacena el resultado.

4.- Repetición del ciclo.



Caso de estudio: Ruta de datos



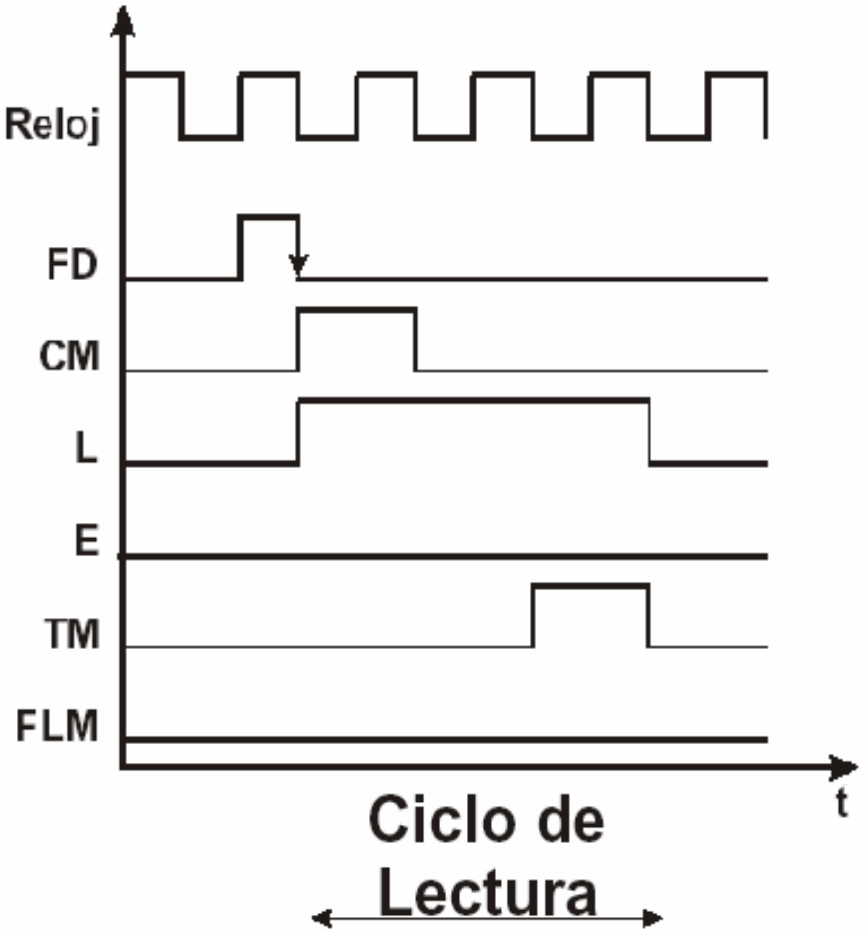
Señales de control de memoria

- **CM:** sirve para iniciar un Ciclo de Memoria
- **L:** especifica ciclo de lectura
- **E:** especifica ciclo de escritura
- **FD:** carga en D la información disponible en el bus interno de direcciones. Señal de flanco.
- **TM:** Señal triestado que conecta la salida de la memoria con el bus de datos. ¡Es generada por la memoria!
- **FLM:** carga en RM la salida de la memoria. Se emplea para mantener el valor leído en RM tras un ciclo de memoria.
- **FEM:** carga en RM la información del bus de datos. Necesaria para realizar operaciones de escritura.

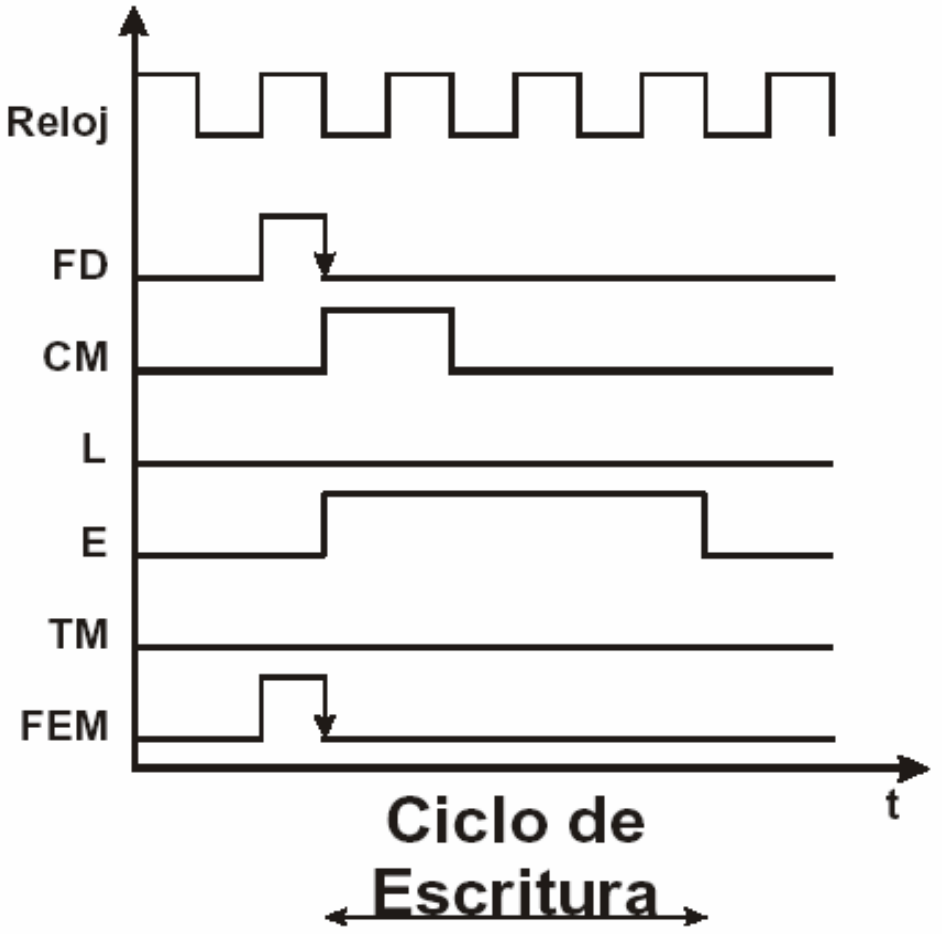
Consideraciones temporales

- Periodo o ciclo de reloj: Tiempo que dura un ciclo de un reloj maestro. Tiempo entre dos flancos sucesivos de subida. Es el tiempo mínimo que dura una operación elemental
- Frecuencia: número de ciclos de reloj por segundo. Se mide en Hz o sus múltiplos
- Frecuencia y periodos son inversos entre sí
- Los ciclos de lectura/escritura duran 30ns
- El periodo básico de esta máquina será de 10ns, lo que equivale a trabajar a 100 MHz

Ciclos de Lectura y Escritura



Con volcado al bus
(no se almacena en RM)



Unidad aritmética

- Selecciona los operandos X, Y a través de dos multiplexores Mux X y Mux Y, de cuatro entradas cada uno
- Señales de operación (OP): OP1, OP2, OP3 y OP4, con las que se selecciona una de las 16 operaciones disponibles en esta ALU (por ejemplo, podría ser el circuito integrado 74181)
- RA: registro auxiliar
  - Conectado a la ALU mediante la entrada 1 del Mux X. Podría ser entrada a la ALU
  - Conectado al bus mediante la señal TRA.
  - Es transparente al usuario. No tiene acceso a él
  - Almacena resultados intermedios

Unidad aritmética (II)

- La salida de la ALU está conectada a:
  - RA
  - Al bus de datos (señal TA, triestado)
  - Al bus de direcciones (TD)
- Los bits de estado aritmético se cargan en el registro ESTADO (STATUS) mediante la señal FEST
- La ALU: establece un camino entre el origen (uno o dos operandos) y el destino (resultado): las señales XX, XY, OP, TA y TD son activas por nivel.

Banco de registros

- 16 registros de propósito general
- Direcciones de 4 bits
- DA y DB, permiten leer simultáneamente 2 registros cuya información se almacena en A y B (puertas de salida)
- DA, también está conectado a A´ (puerta de entrada del banco de registro)
- La señal CR permite que carguemos en la dirección apuntada por DA la información presente en el bus de datos

Órganos de control

Se necesitan unos registros auxiliares de propósito específico, que son:

- **PC – contador de programa**
- **SP – puntero de pila**
- **I – registro de instrucción (RI)**
- **Estado – registro de estado (STATUS)**
- **RF – registro de fases y periodos**

Contador de programa

- **Contador de Programa (PC)**
  - FP: señal de carga
  - TP: señal triestado que lo comunica con el bus de direcciones
  - Conectado a la entrada 3 del multiplexor Y
  - El contador deberá incrementarse de 2 en 2 para acceder a la siguiente instrucción

Puntero de pila

- **SP – Puntero de Pila (Stack Pointer)**
  - FS: señal de carga
  - TS: señal triestado que lo comunica con el bus de direcciones
  - Conectado a la entrada 4 del multiplexor Y: para facilitar su incremento y decremento

Registro de instrucción

- **I – Registro de Instrucción (RI)**
  - FI: señal de carga
  - TI: señal triestado que permite el paso de operandos inmediatos o desplazamientos para direccionamientos relativos hacia la ALU.
  - TI efectúa la necesaria extensión de signo para ajustar los tamaños de palabra.
  - Los campos del registro I que contienen direcciones están conectados a DA y DB en el Banco de Registros (no está dibujado)

Registro de Estado

- **Estado – Registro de Estado (STATUS)**
  - FEST: señales de carga
  - Actualización selectiva de los bits de este registro en función de la instrucción ejecutada por la ALU.

Registro de Fases y Periodos

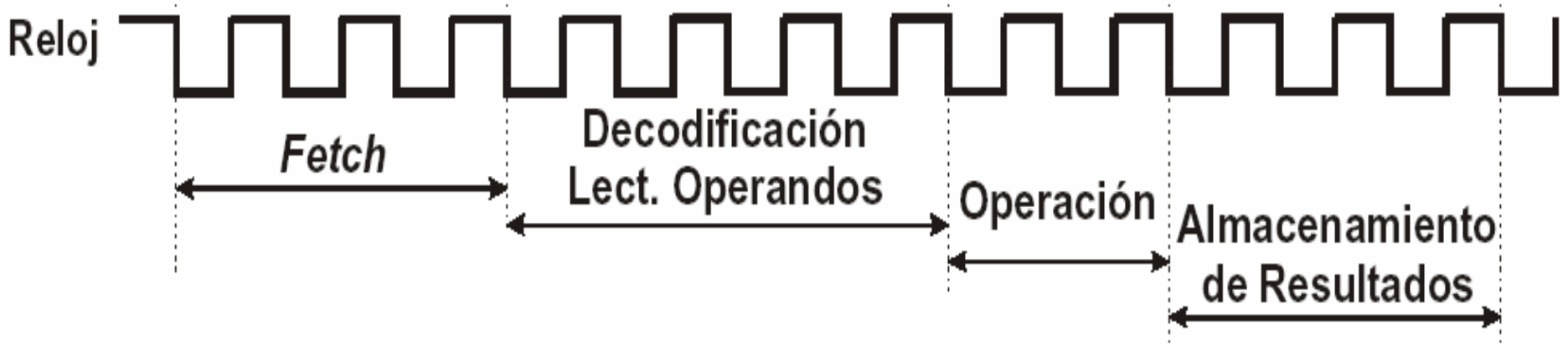
- **RF – Registro de Fases y Periodos**
  - Contador que se va incrementando con los pulsos de un oscilador (reloj) maestro
  - Permite diferenciar entre las diferentes fases y periodos de las instrucciones, NECESARIOS para realizar las operaciones elementales
  - PO: lo pone a cero

Temporización de las señales de control: fases de ejecución de una instrucción

1) **Captura de la instrucción (fetch)**
2) **Decodificación y lectura de operandos**
3) **Ejecución de la operación**
4) **Almacenamiento del resultado**

El objetivo de dividir las instrucciones en fases es sistematizar su tratamiento y simplificar el diseño de la unidad de control.

Fases de una instrucción



Instrucciones elementales

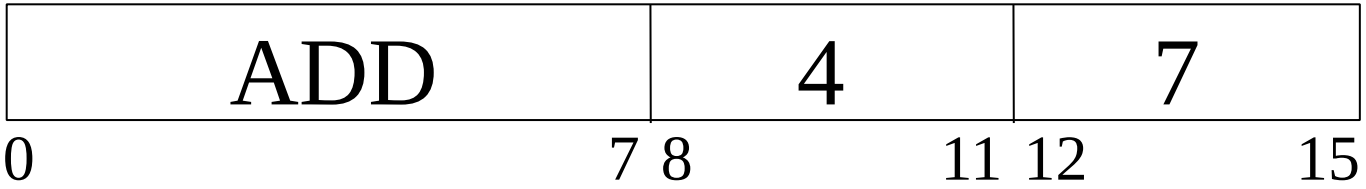
- Las microoperaciones dependen de los elementos que las realizan:
  - multiplexores, buses, registros, operadores...
  - no todos precisan del mismo tiempo para hacer sus operaciones, unos tardan más que otros.
- **Camino crítico:** es el recorrido de la señal eléctrica que provoca el máximo retardo en una operación elemental.
- El ciclo de reloj debe ser lo suficientemente grande como para permitir que se complete sin problema la operación de camino crítico más largo, y lo suficientemente pequeño como para que el computador sea rápido
- Interesa que los caminos críticos de todas las operaciones sean parecidos, para evitar “cuellos de botella”

Ejemplos de ejecución de instrucciones. Cronogramas

- Identificación de microoperaciones
- Identificación de fases de la instrucción
- Temporización:
  - creación del cronograma correspondiente

ADD .4,.7

- SUMA registro, registro
- Se suma el registro R4 y el R7 y el resultado se deja en R4
- Formato de la instrucción:



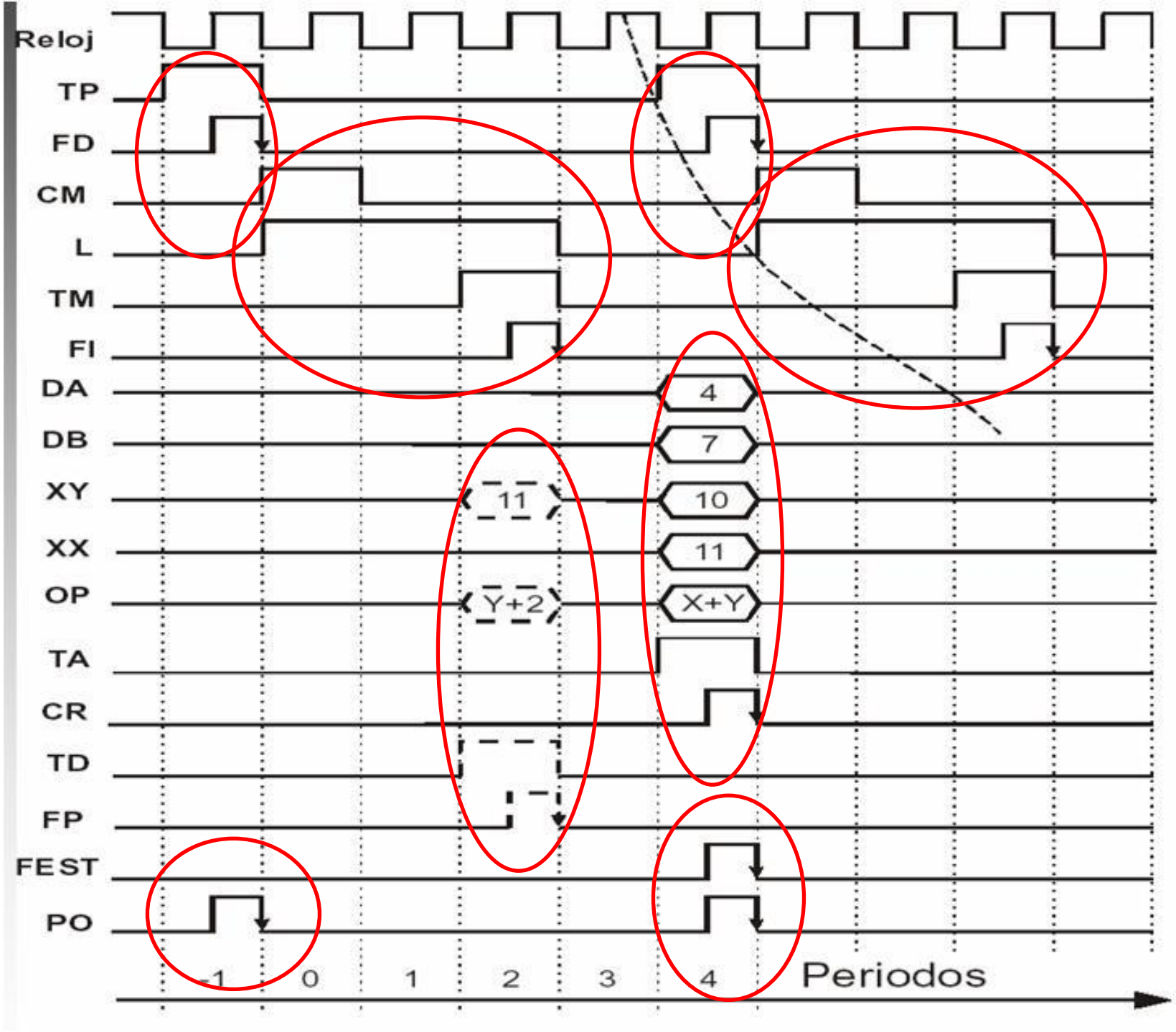
- Fases de la instrucción:
  1) D ← PC
  2) I ← M(D)
  3) Decodificación de la instrucción leída
  4) R4 ← R4 + R7 ; realización de la suma
  5) ESTADO ← bits de estado aritmético, si procede

ADD .4,.7 (II)

- **Fases detalladas de la instrucción:**

1) D ← PC ; transferencia del contador de programa al Registro de Dirección de la memoria e inicialización del registro de fases (ciclo de reloj -1)
2) I ← M(D) ; ciclo de lectura en memoria principal y carga del Registro de Instrucción. Se puede aprovechar este "tiempo muerto" para incrementar el PC en 2 (ciclos de reloj de 0 a 2)
3) PC ← PC + 2 (ciclo de reloj 2)
4) Decodificación de la instrucción leída ; requiere un periodo (ciclo de reloj 3)
5) R4 ← R4 + R7 ; realización de la suma (ciclo de reloj 4)
    1) Conexión de R4 a la puerta Y de la ALU (a través de A)
    2) Conexión de R7 a la puerta X de la ALU (a través de B)
    3) Activación de OP con el código correspondiente a X+Y
    4) Conexión de la salida de la ALU a la entrada A´ del banco de registros
    5) Orden de entrada al banco de registros con CR
6) ESTADO ← bits de estado aritmético; se cargan en ESTADO al mismo tiempo que se guarda el resultado de la suma en R4 y pone a 0 el contador del registro de fases (ciclo de reloj 4)
7) En el ciclo de reloj 4, al igual que en el -1, queda el registro D preparado para la próxima instrucción

ADD .4,.7


Reloj
TP
FD
CM
L
TM
FI
DA
DB
XY
XX
OP
TA
CR
TD
FP
FEST
PO
4
7
11
10
11
Y+2
X+Y
-1
0
1
2
3
4
Periodos

LD .3, #734[.4++]

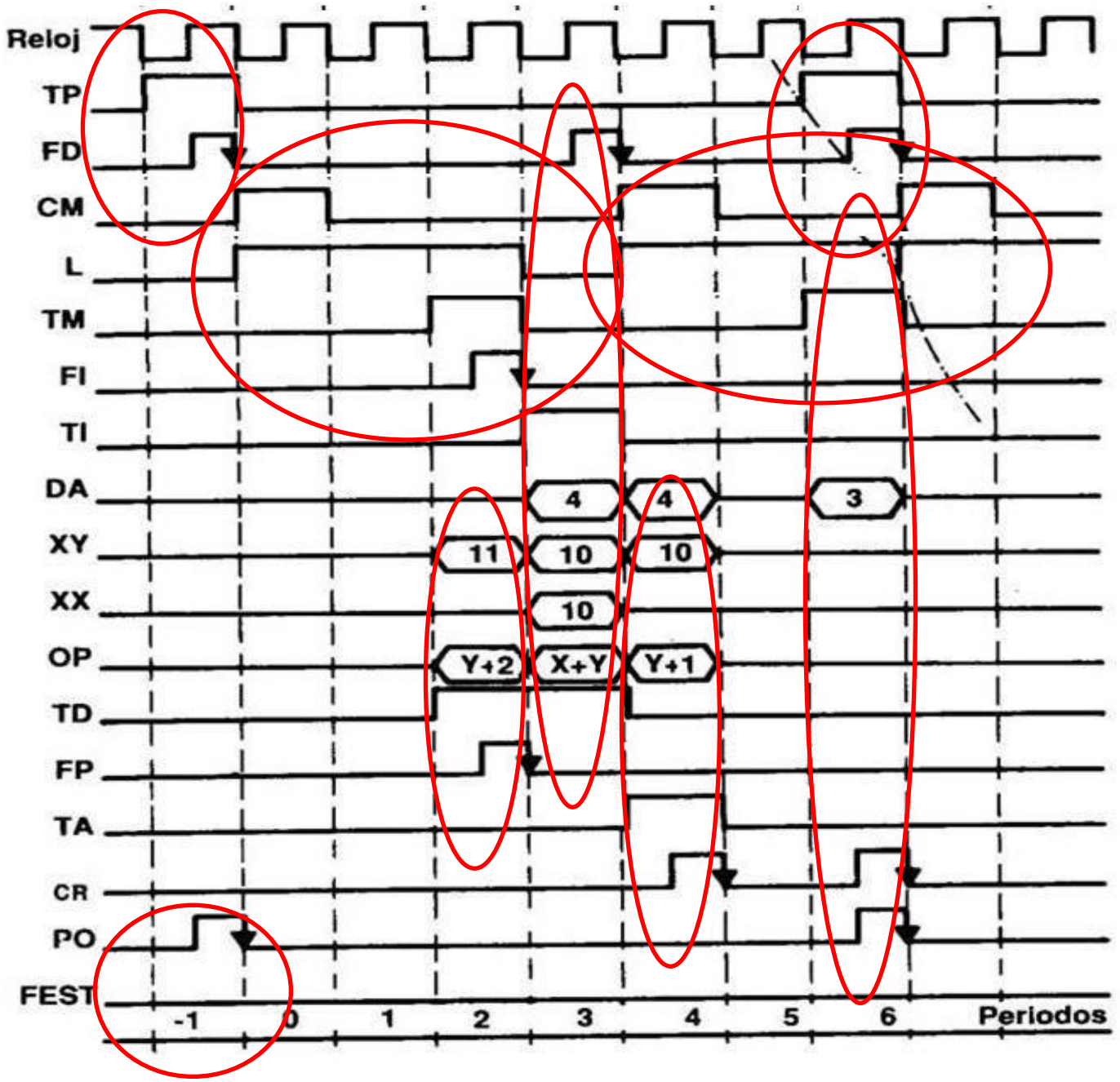
- Operación de transferencia entre una posición de memoria principal y un registro:
  - registro: .3
  - posición de memoria principal: #734[.4++]
- Se carga en el registro 3 la dirección de memoria que se calcula sumando el contenido del registro 4 y la constante 734
- La dirección de memoria se indica con direccionamiento relativo autopostincrementado. El incremento se produce DESPUÉS de obtener la dirección

LD .3, #734[.4++] (II)

- **Fases detalladas de la instrucción:**

1) D ← PC ; transferencia del contador de programa al Registro de Dirección de la memoria e inicialización del registro de fases (ciclo de reloj -1)
2) I ← M(D) ; ciclo de lectura en memoria principal y carga del Registro de Instrucción. Se puede aprovechar este “tiempo muerto” para incrementar el PC en 2 (ciclos de reloj de 0 a 2)
3) PC ← PC + 2 (ciclo de reloj 2)
4) Decodificación de la instrucción leída ; no se considera que necesite tiempo
5) D ← 734 + R4 ; cálculo de la dirección de origen y almacenamiento en D (ciclo de reloj 3). La constante 734 está como operando en el RI, y se vuelca al bus de datos mediante TI. Una vez sumada, se deja en el bus de direcciones para meterla en el registro D
6) RA ← M(D) ; nuevo ciclo de lectura del dato y vuelco al bus de datos (ciclos de reloj de 4 a 6)
7) R4 ← R4 + 1 (ciclo de reloj 4)
8) Carga del registro 3 con el dato seleccionado y actualiza los flags de estado si procede. Pone a 0 el contador del registro de fases (ciclo de reloj 6)
9) En el ciclo de reloj 6, al igual que en el -1, queda el registro D preparado para la próxima instrucción

LD .3,
#734[.4++]


Reloj
TP
FD
CM
L
TM
FI
TI
DA
XY
XX
OP
TD
FP
TA
CR
PO
FEST
4
4
3
11
10
10
10
Y+2
X+Y
Y+1
-1
0
1
2
3
4
5
6
Periodos

SUB .12,[#1734[.13]]

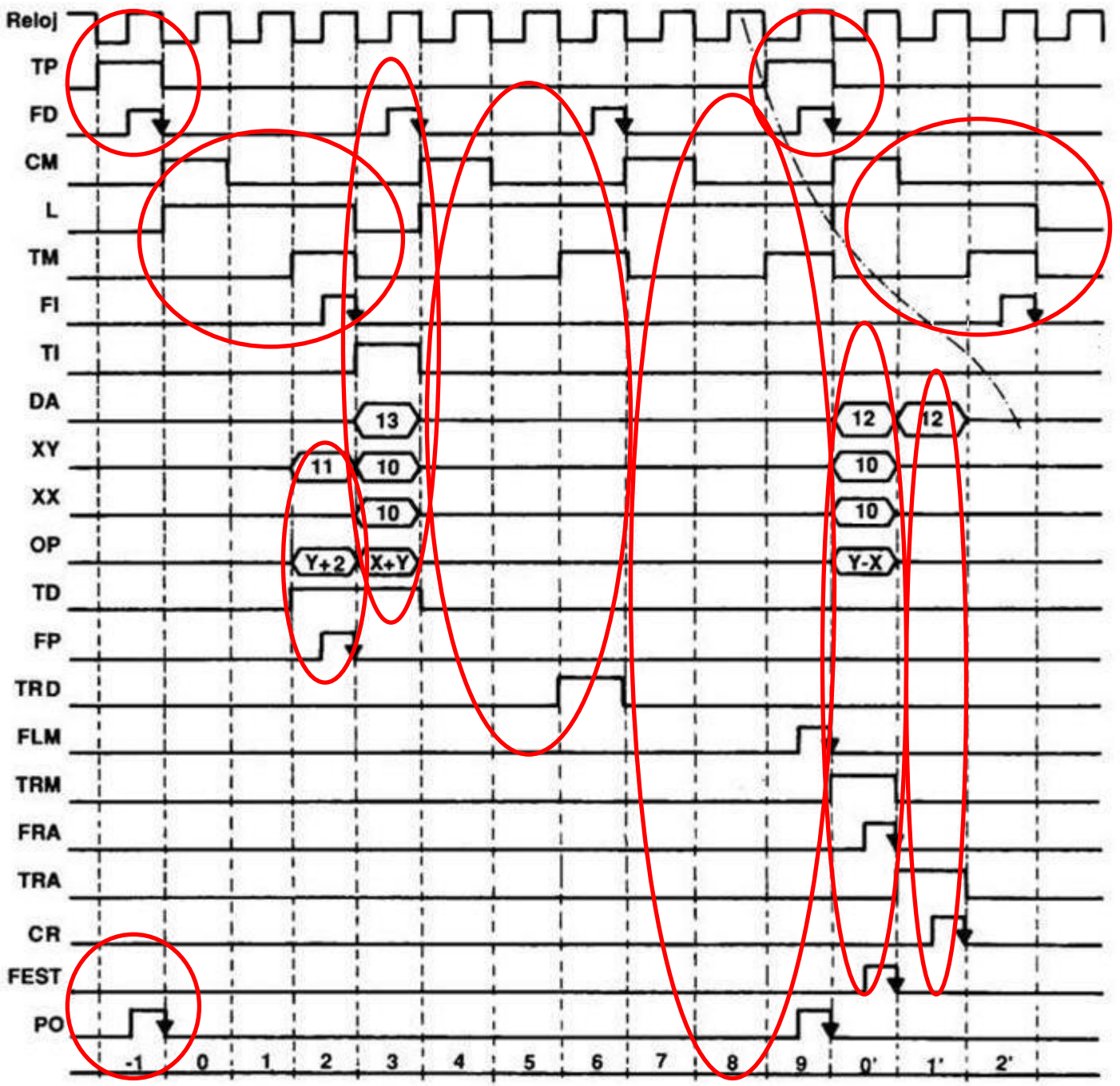
- Operación de resta entre una posición de memoria principal y un registro:
  - registro: .12
  - posición de memoria principal: [#1734[.13]]
- Se resta del registro 12 el contenido de la dirección de memoria obtenida sumando la constante 1734 y el contenido del registro 13. El resultado se guarda en el registro R12
- La dirección de memoria se indica con direccionamiento indirecto relativo, por lo tanto se precisará de dos accesos a memoria principal para obtener el segundo operando.

SUB .12,[#1734[.13]] (II)

- **Fases detalladas de la instrucción:**

1) D ← PC ; transferencia del contador de programa al Registro de Dirección de la memoria e inicialización del registro de fases (ciclo de reloj -1)
2) I ← M(D) ; ciclo de lectura en memoria principal y carga del Registro de Instrucción. Se puede aprovechar este "tiempo muerto" para incrementar el PC en 2 (ciclos de reloj de 0 a 2)
3) PC ← PC + 2 (ciclo de reloj 2)
4) Decodificación de la instrucción leída ; no se considera que necesite tiempo
5) D ← 1734 + R13 ; cálculo de la dirección de origen y almacenamiento en D (ciclo de reloj 3). La constante 1734 está como operando en el RI, y se vuelca al bus de datos mediante TI. Una vez sumada, se deja en el bus de direcciones para meterla en el registro D
6) D ← M(D) ; como es direccionamiento indirecto, hay que llevar el contenido de D de nuevo a D abriendo otra vez un ciclo de lectura que acaba poniendo el dato extraído (dirección definitiva del dato) en el bus de direcciones, e introduciéndolo en el registro D con las señales TRD y FD (ciclos de reloj 4 a 6)
7) RM ← M(D) ; se inicia el último ciclo de lectura para obtener el dato, donde además de sacarlo al bus con TM, se queda almacenado en RM, al usar FLM (ciclos de reloj 7 a 9)
8) R12 ← R12-RM ; se inicia la resta de dos registros (ciclo de reloj 0'), quedando el resultado en RA. Para ello, se activa TRM para que el dato almacenado en RM se vierta al bus de datos y llegue a la ALU. Una vez restados, se introduce la resta en RA, activando FRA. Finalmente se actualizan los bits de estado, si procede (ciclo de reloj 0')
9) El proceso termina vertiendo al bus de datos la resta, almacenada en RA, a través de TRA, y metiendo el resultado en el registro 12 gracias a la señal CR (ciclo de reloj 1')

SUB .12,[#1734[.13]



BZ #1342[.6]

- Bifurcación condicional “si cero”
- Si el bit de estado Z (procedente de una instrucción anterior que no conocemos ni nos interesa) es 1, resultado 0, se produce un salto a la dirección que tiene el registro R6 incrementada en 1342
- En caso contrario, se ignora y el programa sigue su curso natural
- Presenta dos cronogramas diferentes:
  - si la condición se cumple = bifurcación
  - si la condición no se cumple = no operación

BZ #1342[.6] (II)

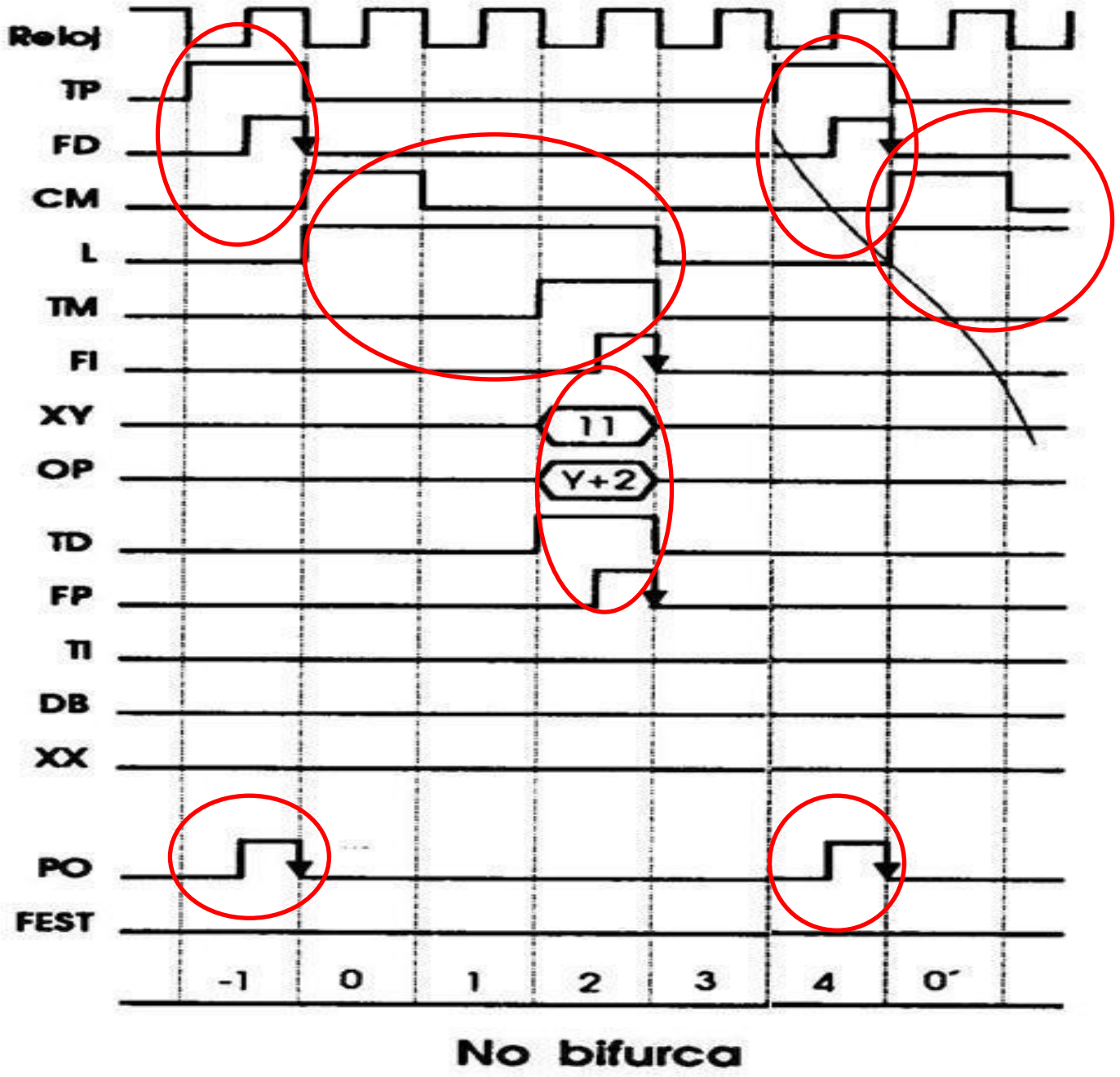
- **Fases detalladas de la instrucción:**
  1) D ← PC ; transferencia del contador de programa al Registro de Dirección de la memoria e inicialización del registro de fases (ciclo de reloj -1)
  2) I ← M(D) ; ciclo de lectura en memoria principal y carga del Registro de Instrucción. Se puede aprovechar este "tiempo muerto" para incrementar el PC en 2 (ciclos de reloj de 0 a 2)
  3) PC ← PC + 2 (ciclo de reloj 2)
  4) Decodificación de la instrucción leída y comprobación de la condición de comparación; requiere un periodo (ciclo de reloj 3)
  5) Si no se cumple la condición de comparación se continúa con
     1) Se prepara para leerla siguiente instrucción. Ya se incrementó el Contador de Programa en la parte común. Dicho incremento no sería necesario en caso de haber bifurcación, pero es más fácil hacerlo en todo caso. Se acaba con el ciclo de reloj 4 sin hacer nada, salvo inicializar el RF con PO.
  6) Si se cumple la condición de comparación se continúa con
     1) D ← PC ← Despl + R6 (ciclo de reloj 4, parte sí bifurca)
     2) Leer la siguiente instrucción

BZ #1342[.6] (III)

NOTA:

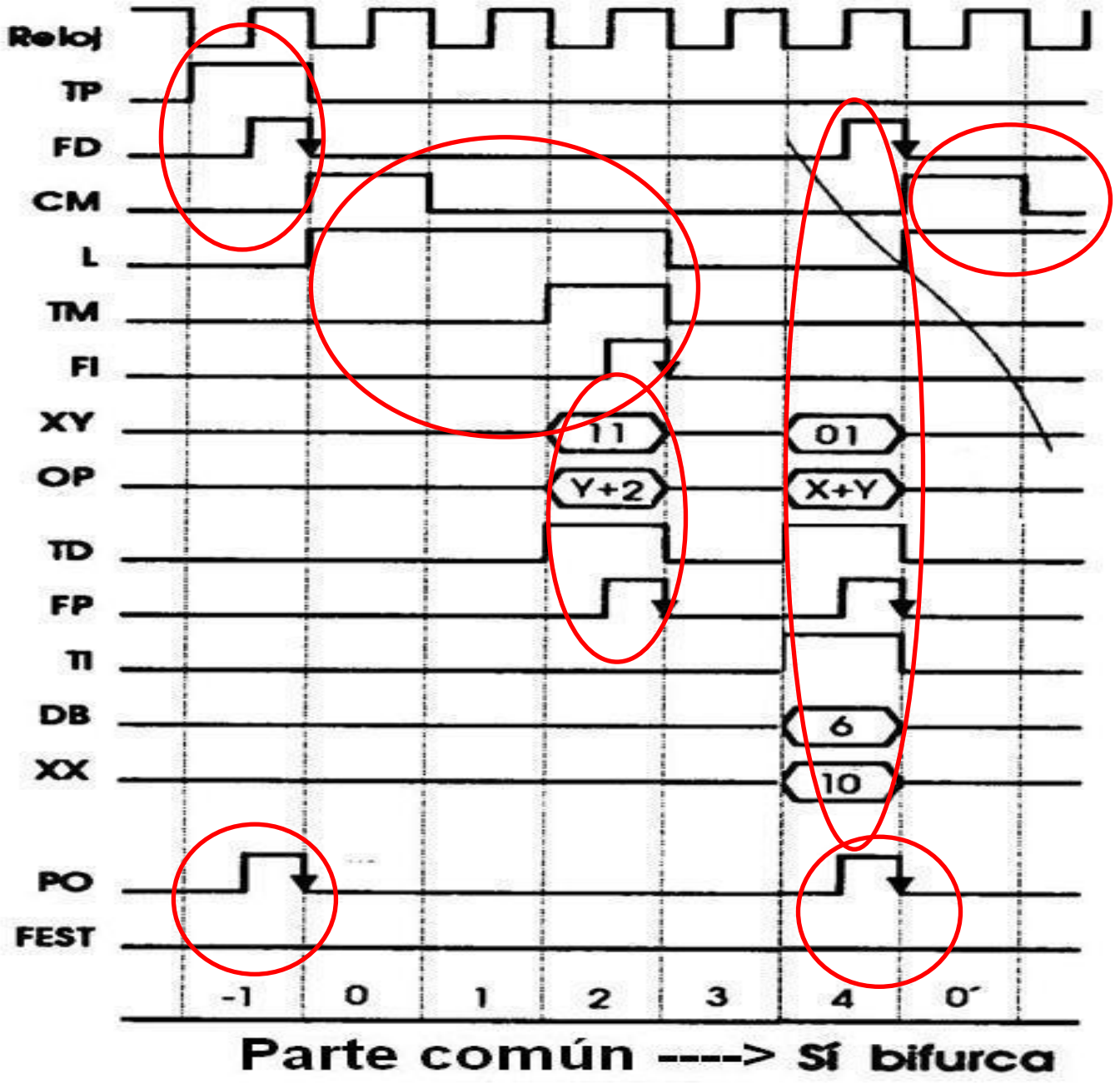
En la operación elemental 4’, el resultado de la suma se carga al mismo tiempo en D y en PC, al activarse a la vez FD y FP, respectivamente. De esa forma no es necesario hacer D ← PC, que en principio se necesitaba al tratarse de direccionamiento indirecto

BZ #1342[.6]



No bifurca

BZ #1342[.6]


Reloj
TP
FD
CM
L
TM
FI
XY
OP
TD
FP
TI
DB
XX
PO
FEST
11
Y+2
01
X+Y
6
10
-1
0
1
2
3
4
0'
Parte común ----> Sí bifurca

BZ #1342[.6] (III)

ACLARACIÓN:

En la operación elemental 4’ (caso sí bifurca), el resultado de la suma se carga al mismo tiempo en D y en PC, al activarse a la vez FD y FP, respectivamente. De esa forma no es necesario hacer D ← PC, que en principio se necesitaba al tratarse de direccionamiento indirecto

Tamaño de las Instrucciones

- En los ejemplos vistos, hemos supuesto que en un único acceso a memoria principal tenemos la instrucción en el RI y que cabe en dicho registro, pero puede no ser así:

- Instrucciones que ocupan **menos** que el tamaño de palabra
  - Almacenar los bits sobrantes (corresponden a la siguiente instrucción) para no tener que repetir otro acceso a memoria
- Instrucciones que ocupan **más** de una palabra de memoria
  - Se repetirán las lecturas de memoria que sean precisas (fetch) almacenándose el código de operación en el RI (habitualmente obtenido en el primer acceso), y las siguientes palabras (códigos de dirección, operandos, etc) en registros temporales

Asignación de periodos a cada instrucción

- Por claridad conceptual, se ha supuesto que la instrucción comienza en 0
- Evidentemente, las señales de control se generan cuando leídas e interpretadas las instrucciones
- O sea, los periodos van desde que se carga en RI e interpreta la instrucción, hasta que se carga en RI e interpreta la instrucción siguiente
- Cada instrucción termina con la carga de la siguiente
- En los ejemplos anteriores, comienzan en 3 y acaban en 2

Diseño de la Unidad de Control

- UC = circuito combinacional que convierte la combinación [código de operación + periodo + estado] en las señales de control precisas para la ejecución de la instrucción.
- Hay que definir previamente las señales de control que actúan en cada instrucción máquina que debe interpretar la U.C.
- Hay que definir, no sólo las operaciones, si no el orden en que han de realizarse. Cada instrucción llama a la siguiente. Es parte de su "trabajo"
- Supongamos un modelo sencillo:
  - COP de 8 bits
  - 32 ciclos por instrucción como máximo, 16 como término medio
  - Registro de fases de 5 bits
  - 150 señales de control

Diseño de la U.C. (II)

- 8 (cop) + 5 (RF) + 1 (comparador) = 14 señales binarias de entrada
- 150 funciones booleanas de salida
- Alrededor de 4000 posibles entradas
- Además, hay que considerar los retardos
- MUY COMPLICADO

- Métodos de construir y diseñar una unidad de control:
  - Mediante lógica cableada
  - Mediante lógica almacenada (memoria)

U.C. mediante lógica cableada

- Se construye con puertas lógica y se diseña mediante alguno de los métodos clásicos de diseños lógicos
- Ventajas:
  - En igualdad de condiciones, una U.C. construida mediante lógica cableada es mucho más rápida que la construida basada en lógica almacenada. Los computadores rápidos la usan
- Inconvenientes:
  - Diseño complicado, circuitos muy complejos
  - Difícil de modificar (requiere un rediseño completo)

U.C. mediante lógica almacenada: Microprogramación

- Se emplea una memoria para almacenar el estado de las señales de control en cada periodo de cada instrucción.
- **Memoria de control o micromemoria:** bastará con ir leyendo una a una las posiciones adecuadas de esta memoria para ejecutar una determinada instrucción máquina
- A cada palabra se la denomina **microinstrucción**.
- Cada microinstrucción tiene un bit por cada señal de control.

U.C. mediante lógica almacenada: Microprogramación

- La microprogramación nace con M.V.Wilkes hacia el 1950:
  - Dos memorias A y B construidas como matrices de diodos
  - Las microinstrucciones están almacenadas en A de donde son leídas mediante un árbol de decodificación
  - La matriz B contiene la dirección de la siguiente instrucción

Concepto de Microprograma

- **Microinstrucción**: cadena de “unos” y “ceros” que representan el estado de cada señal de control durante un ciclo de reloj
- **Microprograma:** conjunto ordenado de microinstrucciones que representan el cronograma de una instrucción máquina
- **Ejecutar una instrucción** consiste en leer un microprograma
- ***Firmware:*** conjunto de los microprogramas de una máquina (se podría traducir como “microcódigo”)
- El tamaño de la memoria de control dependerá de:
  - el número de instrucciones máquina
  - el número de ciclos por instrucción
  - del número de bits de cada microinstrucción
  - Ejemplo, un minicomputador de los 70 tenía 4K microinstrucciones de 24 bits

Microinstrucciones de ADD .4,.7

	TP	FD	CM	L	E	TM	FI	DAO	DA1	DA2	DA3	DB0	DB1	DB2	DB3	XY1	XY2	XX1	XX2	OP0	OP1	OP2	OP3	TA	CR	TD	FP	FEM	FLM	FEST	PO
I_1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
I_2	1	1	0	0	0	0	0	0	0	1	0	1	1	1	0	0	1	1	1	1	1	0	1	1	1	0	0	0	0	1	1
I_3	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
I_4	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
I_5	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	1	1	0	0	0	1	1	0	0	0	1	1	0	0	0	0

Aclaración

- Las señales con flancos de bajada se han considerado señales a 1, cuando sólo están a 1 una pequeña parte de tiempo al final del ciclo
- Para generar un flanco de bajada basta con realizar la AND lógica de un 1 con una señal adecuada

Características de la microprogramación

- Facilidad a la hora de corregir, modificar o ampliar (no precisa rediseño)
- Se pueden incluir instrucciones complejas (basta con tener una memoria de control lo suficientemente grande)
- Se puede incluir en la memoria algunas funciones del Sistema Operativo
- Se pueden construir computadores que ejecuten distintos juegos de instrucciones sin más que cambiar el contenido de la memoria de control
- Permite **emular** otros computadores:
  - ejecutando diversos juegos de instrucciones
- Permite hacer rutinas de diagnóstico del computador

La elaboración del microcódigo es una tarea laboriosa, por lo que la simplificación en el diseño hardware se paga, en cierta medida con el diseño del firmware.

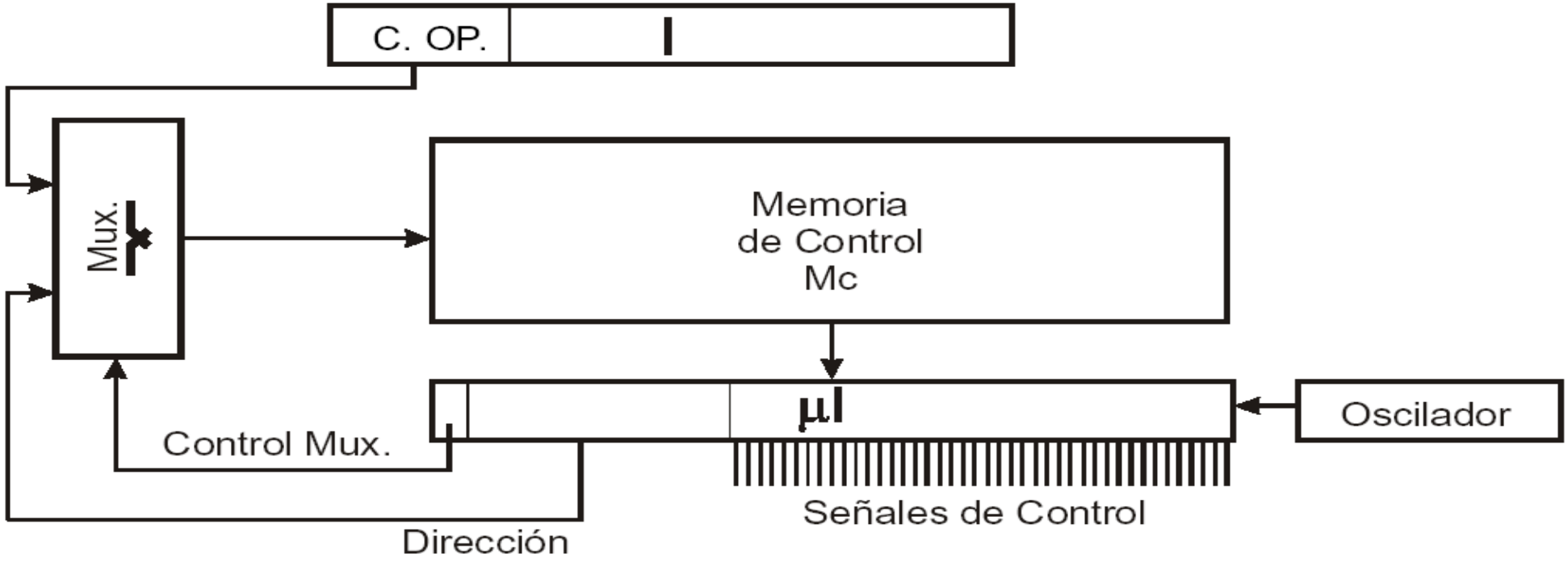
Estructura básica U.C. microprogramada

Toda U.C. microprogramada ha de tener:

- Memoria de control con capacidad para almacenar todos los microprogramas correspondientes a todas las instrucciones máquina del computador
- Procedimiento para hacer corresponder a cada instrucción máquina su microprograma, esto es:
  - convertir el código de operación en la dirección de comienzo del correspondiente microprograma
- Mecanismo para ir leyendo las sucesivas microinstrucciones y para bifurcar a un nuevo microprograma cuando termine el actual
- Mecanismo de microbifurcación condicional

Secuenciamiento explícito

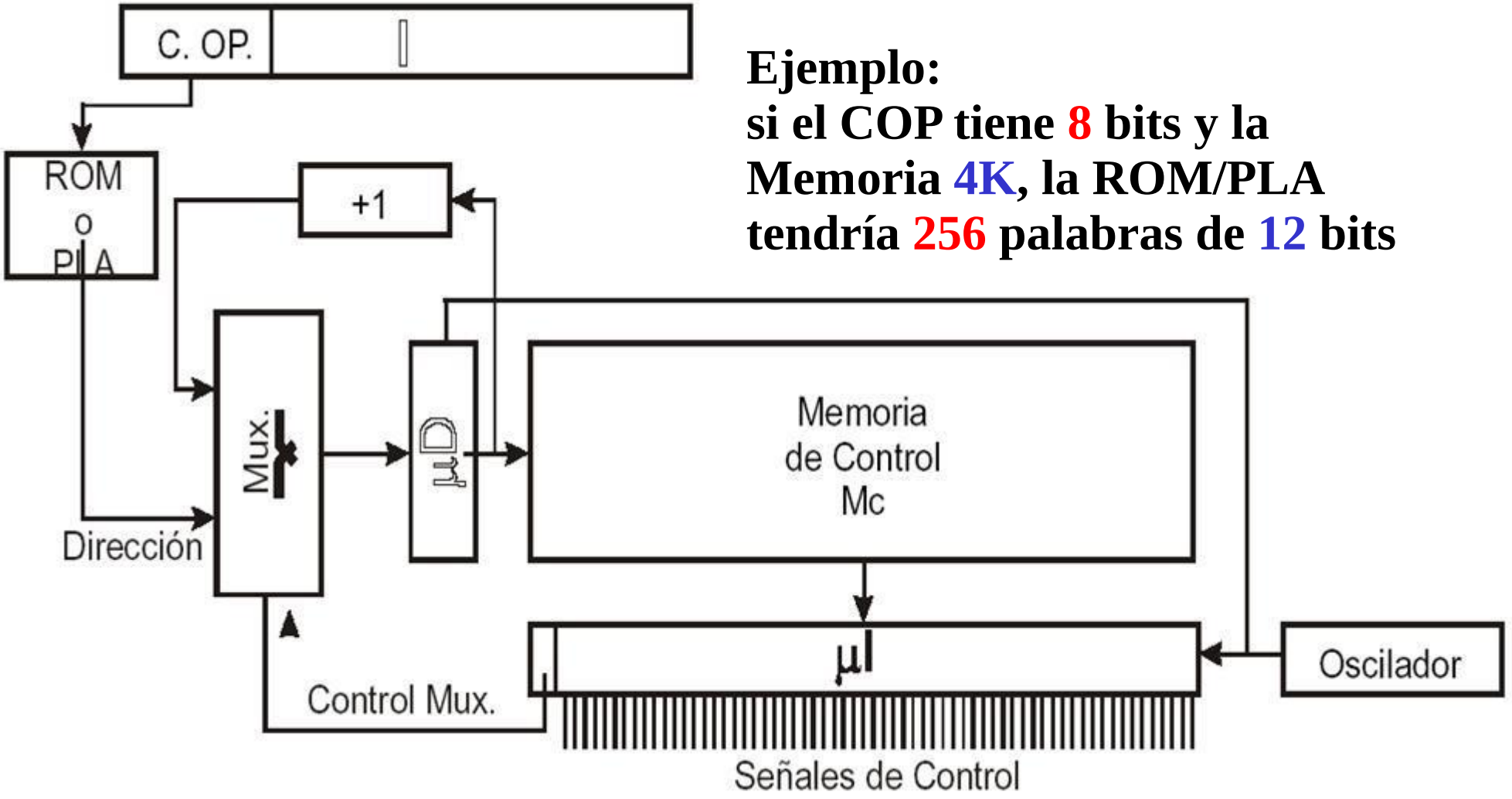
- Cada microinstrucción contiene la dirección de la siguiente microinstrucción a ejecutar
- <u>Inconveniente:</u> consumo de memoria de control para implementarlo



Secuenciamiento implícito

- Obliga a tener ordenadas las microinstrucciones de un microprograma, en posiciones consecutivas de la memoria de control
- ¿Cómo hacemos corresponder una instrucción máquina con su correspondiente microprograma?
  - Etapa traductora entre el código de operación y el multiplexor de direcciones de la memoria de control (ROM o PLA)
- ¿Cómo leemos consecutivamente las microinstrucciones?
  - Se dota a la memoria de control de un registro de microinstrucción y un incrementador

Secuenciamiento implícito (II)



Ejemplo:
si el COP tiene 8 bits y la Memoria 4K, la ROM/PLA tendría 256 palabras de 12 bits

Formato y codificación de las microinstrucciones

Cada microinstrucción tiene un bit por cada señal de control y dura 1 periodo

- Conceptualmente es sencillo, pero es muy caro
- Para reducir su tamaño:
  - simultanear operaciones
  - codificar las microinstrucciones
  - asignar una duración mayor que un periodo a la microinstrucción

Formato y codificación de las microinstrucciones (II)

El formato de las microinstrucciones especifica el número de bits que tienen y el significado de cada uno de ellos

- Hay “demasiados ceros” y señales incompatibles (p.e. L y E)
- Las señales de control que sirven para accionar un mismo órgano se agrupan formando un campo del formato de la microinstrucción.
- Formarán campos las siguientes señales:
  - señales triestado que gobiernan el acceso al bus
  - señales de gobierno de la unidad aritmética
  - señales de gobierno del banco de registros
  - señales de gobierno de la memoria

Formato y codificación de las microinstrucciones (III)

Microprogramación horizontal

- Las microinstrucciones no están codificadas
- Microinstrucciones con formatos muy largos
- Uso en paralelo de los componentes del computador

Microprogramación vertical

- Las microinstrucciones están codificadas
- Formatos de microinstrucción más cortos
- Utiliza menor cantidad de memoria de control

Codificación de campos

- Se codifican todos o algunos de los campos
- Ejemplo: si al bus de datos se conectan 13 elementos distintos, se puede:
  - Utilizar 13 señales distintas sabiendo que sólo una está activa en cada periodo de reloj
  - Usar un campo de 4 bits para codificar las 13 señales (codificación de campos)
  - Se necesita un DEC 4x16 (sobran líneas)

Solapamiento de campos

- Sólo unas pocas señales están activas en cada ciclo de reloj
- A veces son excluyentes
- Se puede utilizar un único grupo de bits para 2 campos distintos C1 y C2
- Un único bit adicional permite saber si se trata de C1 de C2
- En lugar de 2 campos, se usa 1 campo y 1 bit
- Dicho bit se puede usar como líneas de selección de un DEMUX para encaminar el campo común hacia un lugar u otro

Codificación total

- Se definen **TODAS** las microinstrucciones diferentes
- Se codifican **TODAS** con el menor número posible de bits
- Máxima verticalidad
- Se necesita un decodificador complejo
- Se ahorra mucha memoria

Microbifurcaciones condicionales

- *Las instrucciones de bifurcación condicional exigen que la microinstrucción correspondiente pueda elegir entre dos direcciones.*
- Secuenciamiento explícito:
  - Dos direcciones que difieren en 1 bit, que se obtiene del resultado de la comparación
- Secuenciamiento implícito:
  - Se elige entre la microinstrucción siguiente y otra dirección que irá contenida en la propia microinstrucción

Microbucles y microsubrutinas

Microbucles

Precisan de microbifurcación condicional junto con un contador con autodecremento, que se emplea como condición de bifurcación, y accesible desde la microinstrucción para poder ponerlo a 0

Microsubrutinas

Es necesario salvaguardar la dirección de retorno: añadimos una pila al registro µD

Interrupciones y excepciones

Situaciones en las que se rompe el flujo secuencial de ejecución de un programa:

- Error de ejecución (desbordamiento en la ALU)
- Error de paridad en una lectura de memoria
- Intento de ejecutar un código de operación prohibido o inexistente
- Intento de acceso a una zona de memoria protegida
- A petición de un periférico

Ruptura de secuencia no programada:

similar a una bifurcación condicional

Ruptura de secuencia no programada

Interrupción (interrupción externa)

- Cuando la ruptura de secuencia se produce por un elemento externo a la unidad de control
- El propio dispositivo externo proporciona la dirección de bifurcación

Excepción o trap (interrupción interna)

- Cuando la ruptura de secuencia se produce en el interior de la unidad de control
- Se utiliza una posición prefijada como dirección de bifurcación

Excepciones (I)

- Son interrupciones producidas por la propia CPU cuando se producen ciertas situaciones como división por cero, desbordamiento de la pila, fallo de página, etc.

- ¿Qué sucede si no se puede abrir un fichero? ¿Qué pasa si un puerto o punto de conexión se cierra de manera inesperada? ¿Y si dividimos por cero?

Excepciones (II)

Como todas las interrupciones el núcleo realiza los siguientes pasos:

1. Guardar el estado del proceso en la pila.
   push
2. Llamar la función manejadora de la excepción.
   call
3. Recuperar el estado de la función.
   pop

Algunos lenguajes de programación permiten tratar excepciones.
Por ejemplo, JAVA

```
try{
   //Código que puede provocar el error
}catch(IOException ioe){
   //Código para tratar la IOException
}catch(Exception e){
   //Código para tratar la Exception
}
```