

# Unidad de control

Estructura de Computadores  
9ª Semana

## Bibliografía:

- |               |  |
|---------------|--|
| [HAM03] Cap.7 | Organización de Computadores. Hamacher, Vranesic, Zaki. McGraw-Hill 2003 |
|               | Signatura ESIIT/ <a href="#">C.1 HAM org</a>                             |

# Guía de trabajo autónomo (4h/s)

## ■ Repaso

- Apuntes TOC
  - Tema 6. Análisis y diseño de sistemas secuenciales
  - Tema 7. Sistemas en el nivel transferencia entre registros (RTL)

## ■ Lectura

- Cap.7 Hamacher

## Bibliografía:

[TOC] Temas 6-7

Apuntes Tecnología y Organización de Computadores

[HAM03] Cap.7

Organización de Computadores. Hamacher, Vranesic, Zaki. McGraw-Hill 20  
Signatura ESIIT/[C.1 HAM org](#)

# Guía de trabajo autónomo (4h/s)

## ■ Repaso

- Apuntes TOC
  - 6. Análisis y diseño de sistemas secuenciales
    - 6.1 Concepto de sistema secuencial
    - 6.2 Elementos básicos de memoria
    - 6.3 Análisis de un sistema secuencial
    - 6.4 Diseño de un sistema secuencial
    - 6.5 Componentes secuenciales estándar
  - 7. Sistemas en el nivel transferencia entre registros (RTL)\*
    - 7.1 Introducción y definiciones generales
    - 7.2 Unidad de procesamiento o camino de datos
    - 7.3 Unidad de control
    - 7.4 Introducción a lenguajes de descripción hardware
    - 7.5 Fases de diseño

# Unidad de control

## ■ Camino de datos

- Unidad de procesamiento y unidad de control
- Unidad de procesamiento con un bus
- Unidad de procesamiento con múltiples buses

## ■ Unidades de control cableadas y microprogramadas

- Diseño de una UC cableada
- Ejemplo de UC cableada
- Concepto de UC microprogramada

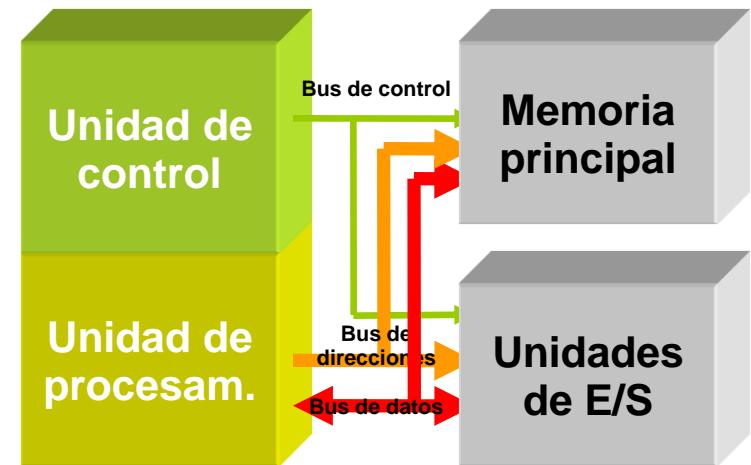
## ■ Control microprogramado

- Formato de las microinstrucciones
- Nanoprogramación
- Secuenciamiento de microinstrucciones
- Ejemplo de arquitectura microprogramada

# Introducción

- Un computador con arquitectura Von Neumann consta de tres bloques fundamentales:

- CPU o procesador
  - Memoria principal
    - (datos + instrucciones)
  - Unidades de E/S
    - (y memoria masiva)
- } unidos mediante buses



- En este tema estudiaremos la CPU, que puede entenderse como una unidad constituida por:
  - Unidad de procesamiento o camino de datos ("datapath")
  - Unidad de control

# Unidad de procesamiento

- La unidad de procesamiento comprende elementos hardware como:
  - unidades funcionales (ALU, desplazad., multiplic., etc.)
  - registros
    - registros de uso general (varios GPR)
    - registro de estado
    - contador de programa (PC)
    - puntero de pila (SP)
    - registro de instrucción (IR)
    - registro de dato de memoria (MDR / MBR)
    - registro de dirección de memoria (MAR)
  - multiplexores
  - buses internos
  - ...

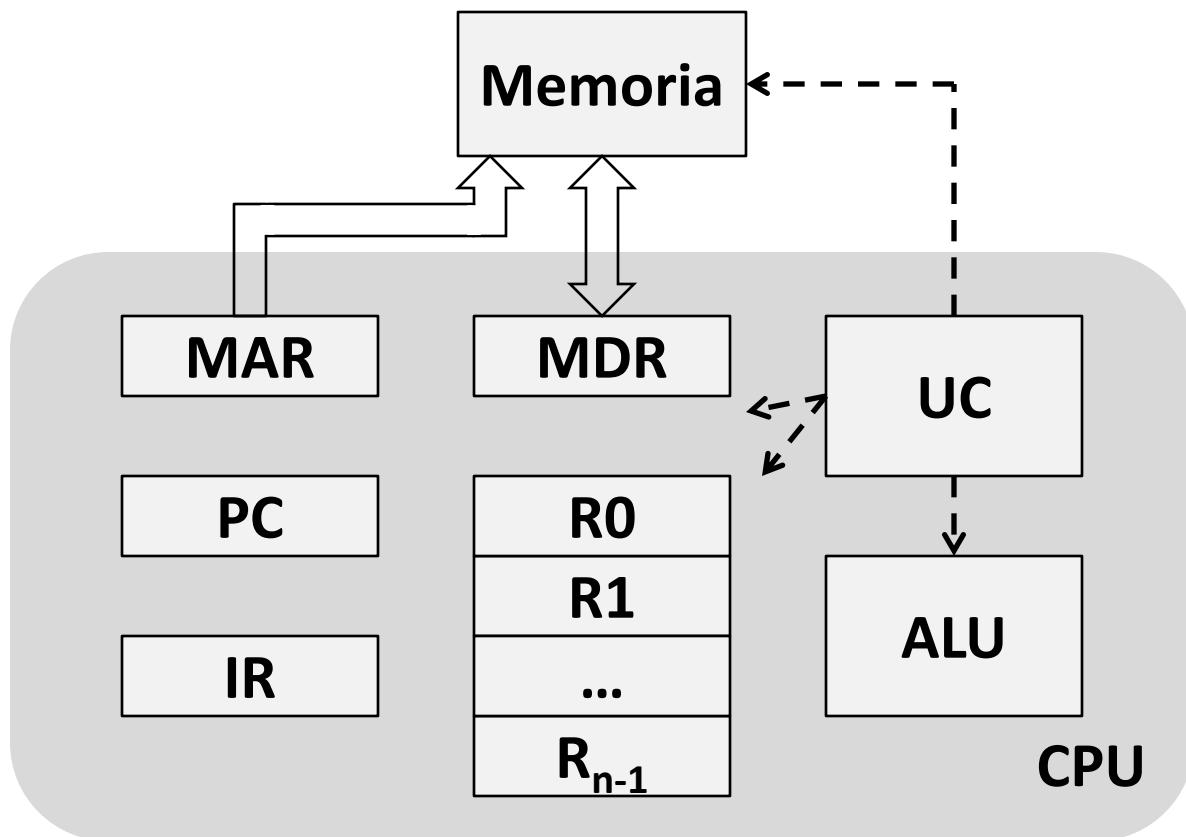
# Unidad de control

- **La unidad de control interpreta y controla la ejecución de las instrucciones leídas de la memoria principal, en dos fases:**
  - secuenciamiento de las instrucciones
    - La UC lee de MP la instrucción apuntada por PC,  $IR \leftarrow M[PC]^*$
    - determina la dirección de la instrucción siguiente y la carga en PC\*
  - ejecución/interpretación de la instrucción en IR
    - La UC reconoce el tipo de instrucción,
    - manda las señales necesarias para tomar los operandos necesarios y dirigirlos a las unidades funcionales adecuadas de la unidad de proceso,
    - manda las señales necesarias para realizar la operación,
    - manda las señales necesarias para enviar los resultados a su destino.

# Ej. de ejecución Add A, R0\*

## ■ Pensar tareas realizadas por UC para ejecutar instrucción

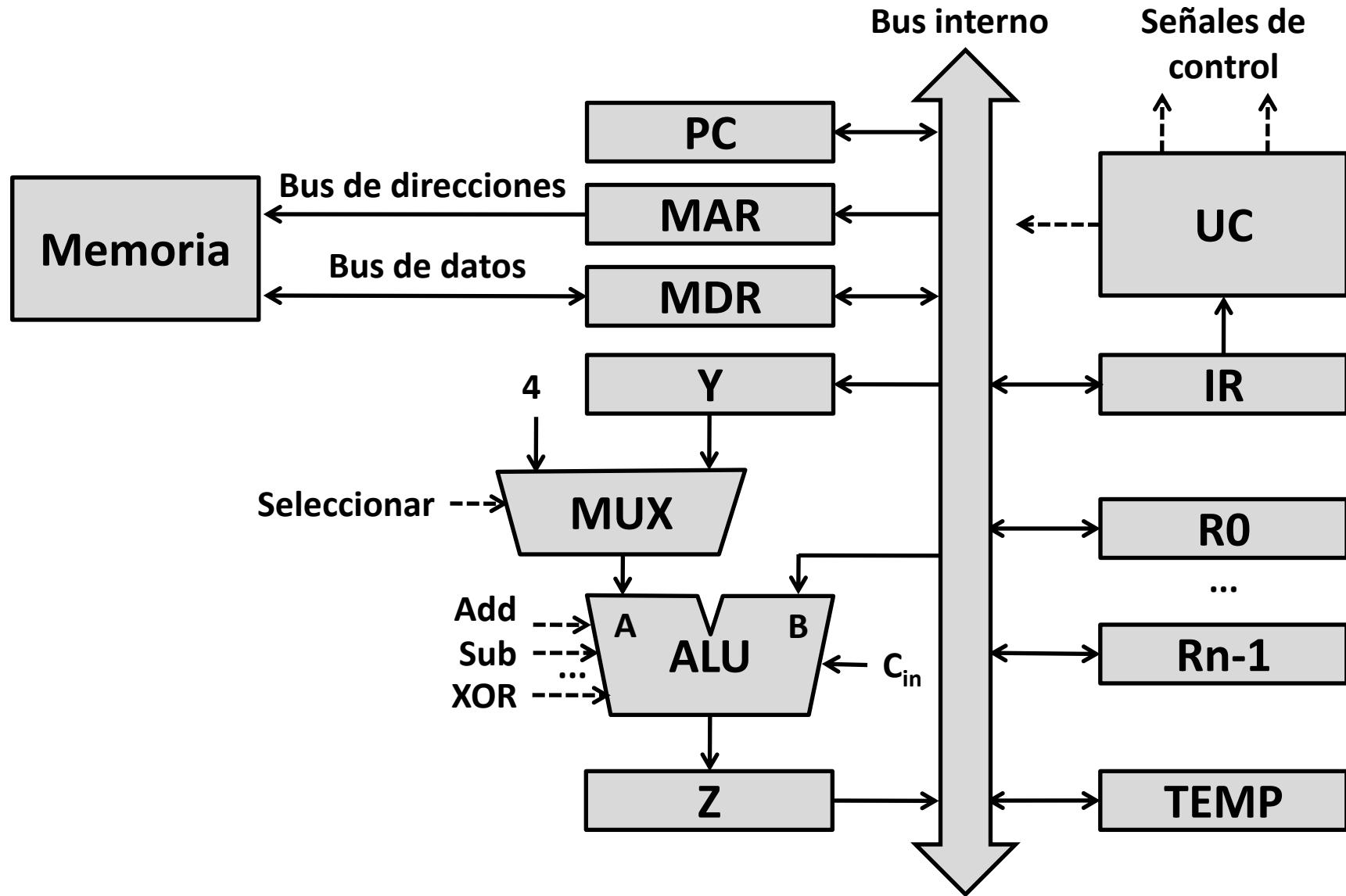
- Por ejemplo: Add A, R0
  - $M[A] + R0 \rightarrow R0$
- Detalles en [HAM03] Cap-1.3
- Ejercicios similares en TOC §2.1



# Ej. de ejecución Add A, R0\*

- $M[POS_A] + R0 \rightarrow R0$ 
  - Ensamblador traduce p.ej:  $POS_A=100$
  - Valor anterior R0 perdido, el de  $POS_A$  se conserva
  - Arquitectura R/M
- **Pasos básicos de la UC**
  - PC apunta a posición donde se almacena instrucción
  - **Captación:**  $MAR \leftarrow PC$ , Read,  $PC \leftarrow PC + 1$ ,  $T_{acc} \leftarrow$  MDR  $\leftarrow$  bus,  $IR \leftarrow MDR$
  - **Decodificación:** se separan campos instrucción
    - Codop: ADD  $mem + reg \rightarrow reg$
    - Dato1: 100 direcciónamiento directo, habrá que leer  $M[100]$
    - Dato2: 0 direcciónamiento registro, habrá que llevar R0 a ALU  
CPUs con longitud instrucción variable – dirección (100) en siguiente palabra
  - **Operando:**  $MAR \leftarrow 100$ , Read,  $T_{acc} \leftarrow$  MDR,  $ALU_{in1} \leftarrow MDR$
  - **Ejecución:**  $ALU_{in2} \leftarrow R0$ , add,  $T_{alu}$
  - **Almacenamiento:**  $R0 \leftarrow ALU_{out}$

# Unidad de procesamiento con un bus



# Unidad de procesamiento con un bus

- Componentes interconectados mediante bus común
- MDR: dos entradas, dos salidas
- MAR: unidireccional (procesador → memoria)
- R0...Rn-1: GPR, SP, índices,...
- Y, Z, TEMP:
  - registros transparentes al programador
  - almácen temporal interno en la ejecución de una instrucción
- MUX: selecciona entrada A de la ALU
  - La constante 4 se usa para incrementar el contador de programa
- La UC genera señales internas y externas (memoria)

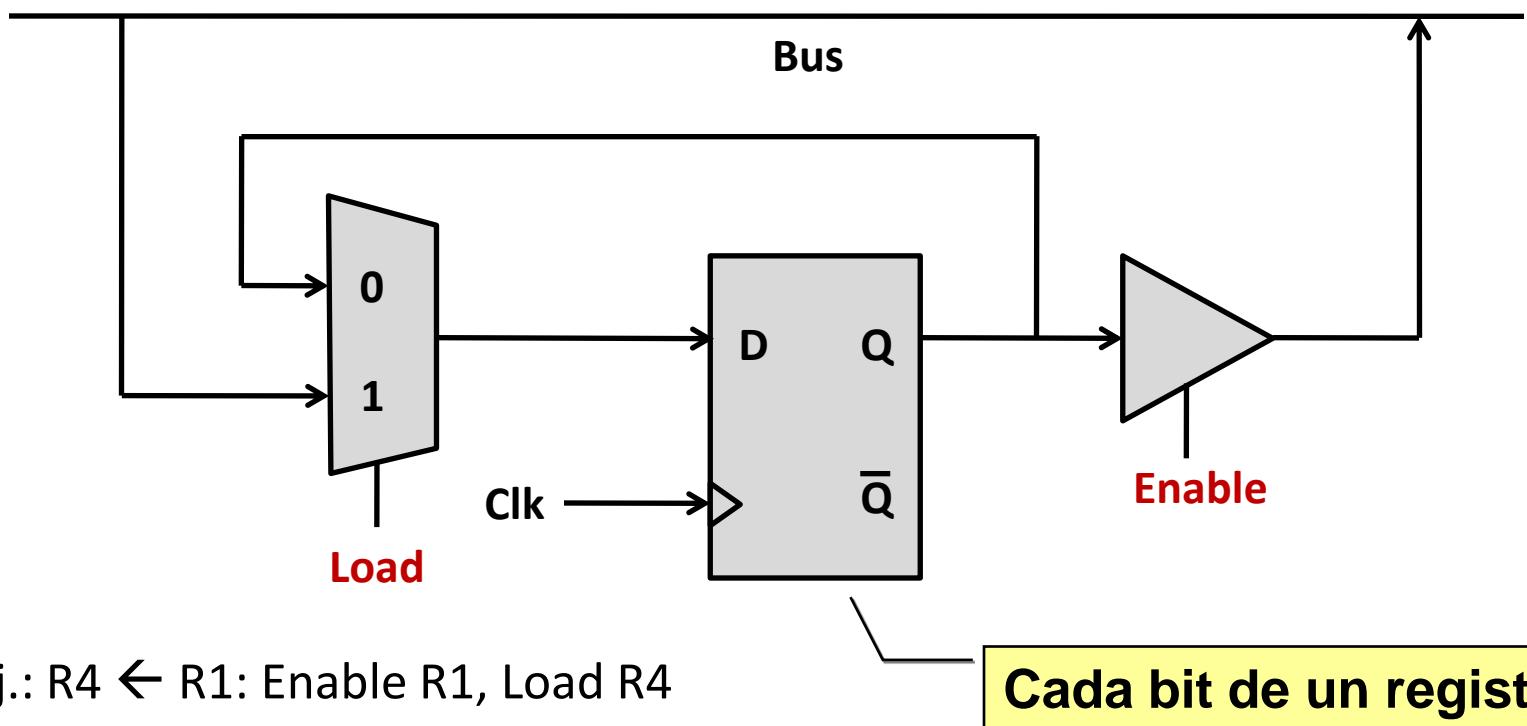
# Unidad de procesamiento con un bus

- Una instrucción puede ser ejecutada mediante una o más de las siguientes operaciones:
  - Transferir de un registro a otro
  - Realizar operación aritmética o lógica y almacenar en registro
  - Cargar posición de memoria en registro
  - Almacenar registro en posición de memoria

# Unidad de procesamiento con un bus

## ■ Transferir de un registro a otro

- Cada registro usa dos señales de control:
  - Load: Carga en paralelo
  - Enable: Habilitación de salida (buffer triestado)



# Unidad de procesamiento con un bus

## ■ Realizar operación aritmética o lógica y almacenar en registro

- ALU: circuito combinacional sin memoria
- Resultado almacenado temporalmente en Z
- Ej.:  $R3 \leftarrow R1 + R2$

1. Enable R1, Load Y
2. Enable R2, Select Y, Add, Load Z
3. Enable Z, Load R3

Cada línea: 1 ciclo de reloj

Esta transferencia no puede realizarse en el paso 2 ¿por qué?

- Las señales de control de la ALU podrían estar codificadas

# Unidad de procesamiento con un bus

## ■ Cargar posición de memoria en registro

- Transferir dirección a MAR
- Activar lectura de memoria
- Almacenar dato leído en MDR (MDR dos entradas, dos salidas)
- La temporización interna debe coordinarse con la de la memoria
  - La lectura puede requerir varios ciclos de reloj
  - El procesador debe esperar la activación de señal de finalización de ciclo de memoria
- Ej.: **Load (R1) → R2**
  1. Enable R1, Load MAR
  2. Comenzar lectura
  3. Esperar fin de ciclo de memoria, Load MDR desde memoria
  4. Enable MDR hacia bus interno, Load R2

# Unidad de procesamiento con un bus

## ■ Almacenar registro en posición de memoria

- Transferir dirección a MAR
- Transferir dato a escribir a MDR (MDR dos entradas, dos salidas)
- Activar escritura de memoria
- La temporización interna debe coordinarse con la de la memoria
  - La escritura puede requerir varios ciclos de reloj
  - El procesador debe esperar la activación de señal de finalización de ciclo de memoria
- Ej.: **Store R2 → (R1)**
  1. Enable R1, Load MAR
  2. Enable R2, Load MDR desde bus interno
  3. Comenzar escritura
  4. Esperar fin de ciclo de memoria

# Unidad de procesamiento con un bus

## ■ Ejecución de una instrucción completa

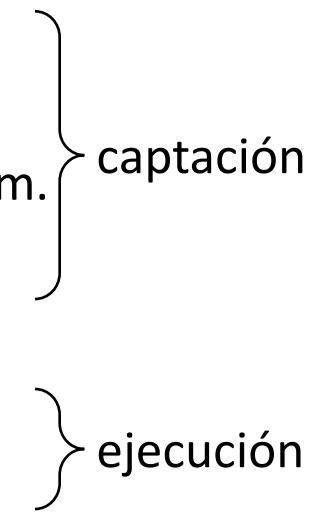
■ Ej.: **Add (R3) → R1**

1. Enable PC, Load MAR, Select 4, Sumar, Enable Z
  2. Comenzar lectura, Enable Z, Load PC, Load Y\*
  3. Esperar fin de ciclo de memoria, Load MDR desde mem.
  4. Enable MDR hacia bus interno, Load IR
  5. Decodificar instrucción
  6. Enable R3, Load MAR
  7. Comenzar lectura, Enable R1, Load Y
  8. Esperar fin de ciclo de memoria, Load MDR desde mem.
  9. Enable MDR hacia bus interno, Select Y, Sumar, Load Z
  10. Enable Z, Load R1, Saltar a captación
- 
- captación
- ejecución

# Unidad de procesamiento con un bus

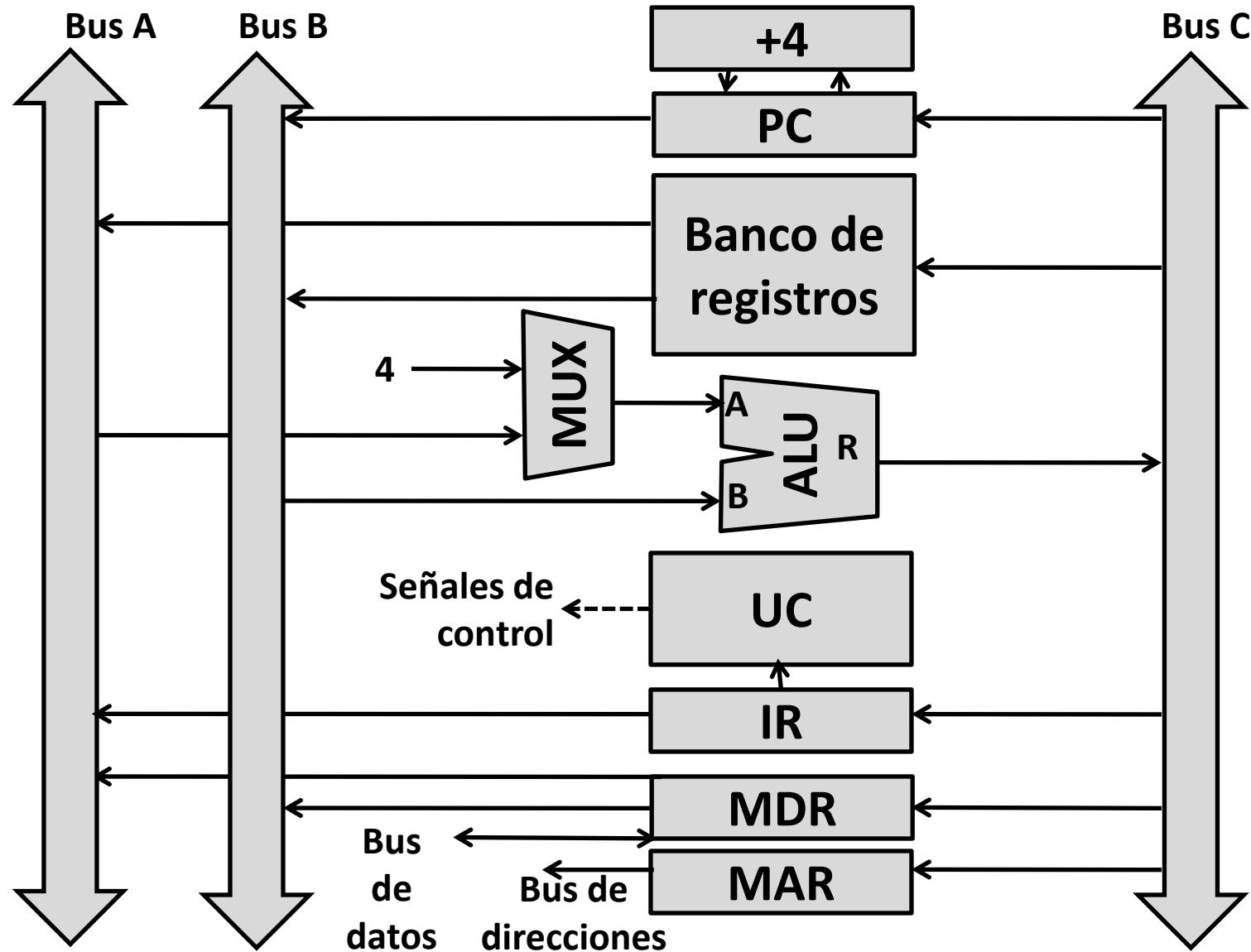
## ■ Ejecución de una instrucción de salto

### ■ Ej.: **Jmp desplazamiento**

1. Enable PC, Load MAR, Select 4, Sumar, Enable Z
  2. Comenzar lectura, Enable Z, Load PC, Load Y
  3. Esperar fin de ciclo de memoria, Load MDR desde mem.
  4. Enable MDR hacia bus interno, Load IR
  5. Decodificar instrucción
  6. Enable Campo desplazamiento en IR, Sumar, Load Z
  7. Enable Z, Load PC, Saltar a captación
- 
- captación
- ejecución

### ■ ¿Qué habría que añadir al paso 6 para **JS** (saltar si negativo)?

# Unidad de procesam. buses múltiples



# Unidad de procesam. buses múltiples

## ■ Banco de registros con tres puertos

- Dos registros pueden poner sus contenidos en los buses A y B
  - Un dato del bus C puede cargarse en un registro
- } En el mismo ciclo

## ■ ALU

- No necesita los registros Y y Z
- Puede pasar A o B directamente a R (bus C)

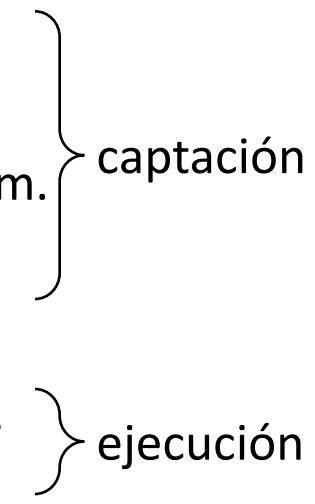
## ■ Unidad de incremento (+4)

- Pero la constante 4 como fuente de la ALU sigue siendo útil para incrementar otras direcciones en instrucciones de movimiento múltiple

# Unidad de procesam. buses múltiples

## ■ Ejecución de una instrucción completa

■ Ej.: **R6 ← R4 + R5**

1. Enable PC, R=B, Load MAR
  2. Comenzar lectura, Incrementar PC
  3. Esperar fin de ciclo de memoria, Load MDR desde mem.
  4. Enable MDR hacia B, R=B, Load IR
  5. Decodificar instrucción
  6. Enable R4 hacia A, Enable R5 hacia B, Select A, Sumar,  
Load R6, Saltar a captación
- 
- The diagram illustrates the execution of the instruction **R6 ← R4 + R5**. It shows a sequence of six steps. Steps 1 through 3 are grouped by a brace on the right labeled "captación" (capture), which corresponds to the initial phase of reading from memory and loading the MDR. Steps 4 through 6 are grouped by another brace on the right labeled "ejecución" (execution), which corresponds to the arithmetic operation and result loading.

# Unidad de control

## ■ Camino de datos

- Unidad de procesamiento y unidad de control
- Unidad de procesamiento con un bus
- Unidad de procesamiento con múltiples buses

## ■ Unidades de control cableadas y microprogramadas

- Diseño de una UC cableada
- Ejemplo de UC cableada
- Concepto de UC microprogramada

## ■ Control microprogramado

- Formato de las microinstrucciones
- Nanoprogramación
- Secuenciamiento de microinstrucciones
- Ejemplo de arquitectura microprogramada

# Unidad de control

## ■ Señales de entrada a la UC:

- Señal de reloj
- Instrucción actual (codop, campos de direccionamiento,...)
- Estado de la unidad de proceso
- Señales externas (por ej. interrupciones)

## ■ Señales de salida de la UC:

- Señales que gobiernan la unidad de procesamiento:
  - Carga de registros
  - Incremento de registros
  - Desplazamiento de registros
  - Selección de entradas de multiplexores
  - Selección de operaciones de la ALU ...
- Señales externas
  - Por ej. lectura/escritura en memoria

# Tipos de unidades de control

## ■ Existen dos formas de diseñar la UC:

- Control fijo o cableado (“hardwired”)
  - Se emplean métodos de diseño de circuitos digitales secuenciales a partir de diagramas de estados.
  - El circuito final se obtiene conectando componentes básicos como puertas y biestables, aunque más a menudo se usan PLA.
- Control microprogramado
  - Todas las señales que se pueden activar simultáneamente se agrupan para formar palabras de control, que se almacenan en una memoria de control (normalmente ROM).
  - Una instrucción de lenguaje máquina se transforma sistemáticamente en un programa (microprograma) almacenado en la memoria de control.
    - Mayor facilidad de diseño para instrucciones complejas
    - Método estándar en la mayoría de los CISC.

# Diseño de una UC cableada

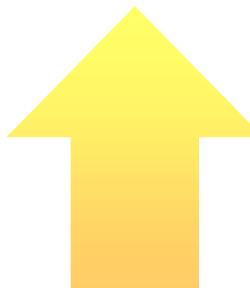
- **Se diseña mediante puertas lógicas y biestables siguiendo uno de los métodos clásicos de diseño de sistemas digitales secuenciales ya conocidos (TOC)**
  - El diseño es laborioso y difícil de modificar debido a la complejidad de los circuitos.
  - Suele ser más rápida que la misma UC microprogramada.
  - Se utilizan PLA (matrices lógicas programables) para llevar a cabo la implementación.

Debido a las modernas técnicas de diseño y a RISC ha tomado nuevo auge la realización de UC cableadas

# Diseño de una UC cableada

## ■ Técnicas de diseño por computador (CAD) para circuitos VLSI (compiladores de silicio)

- Resuelven automáticamente la mayor parte de las dificultades de diseño de lógica cableada.
- Generan directamente las máscaras de fabricación de circuitos VLSI a partir de descripciones del comportamiento funcional del circuito en un lenguaje de alto nivel.



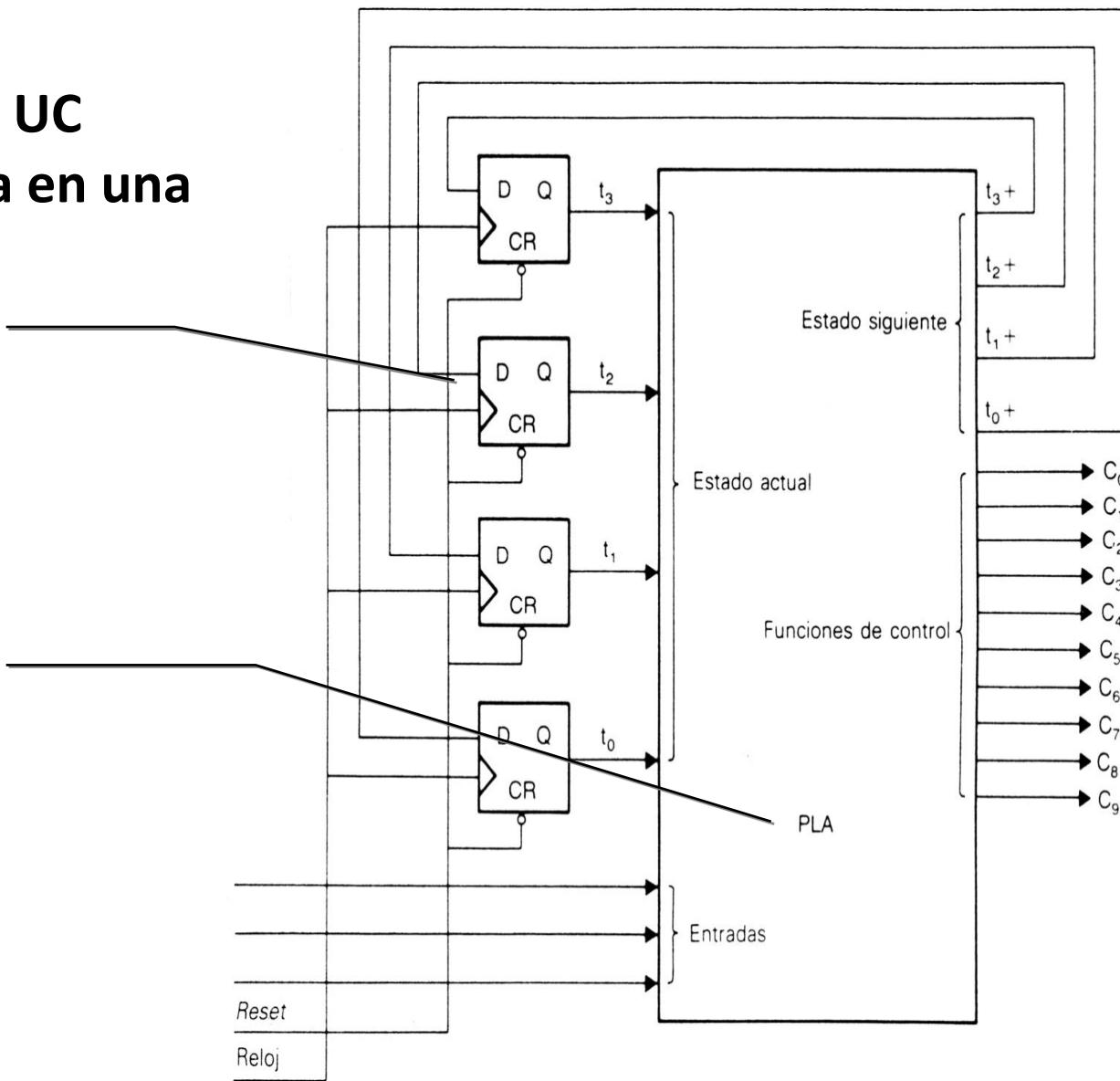
Debido a las modernas técnicas de diseño y a RISC ha tomado nuevo auge la realización de UC cableadas

# Diseño de una UC cableada

- Organización de UC cableada basada en una PLA:

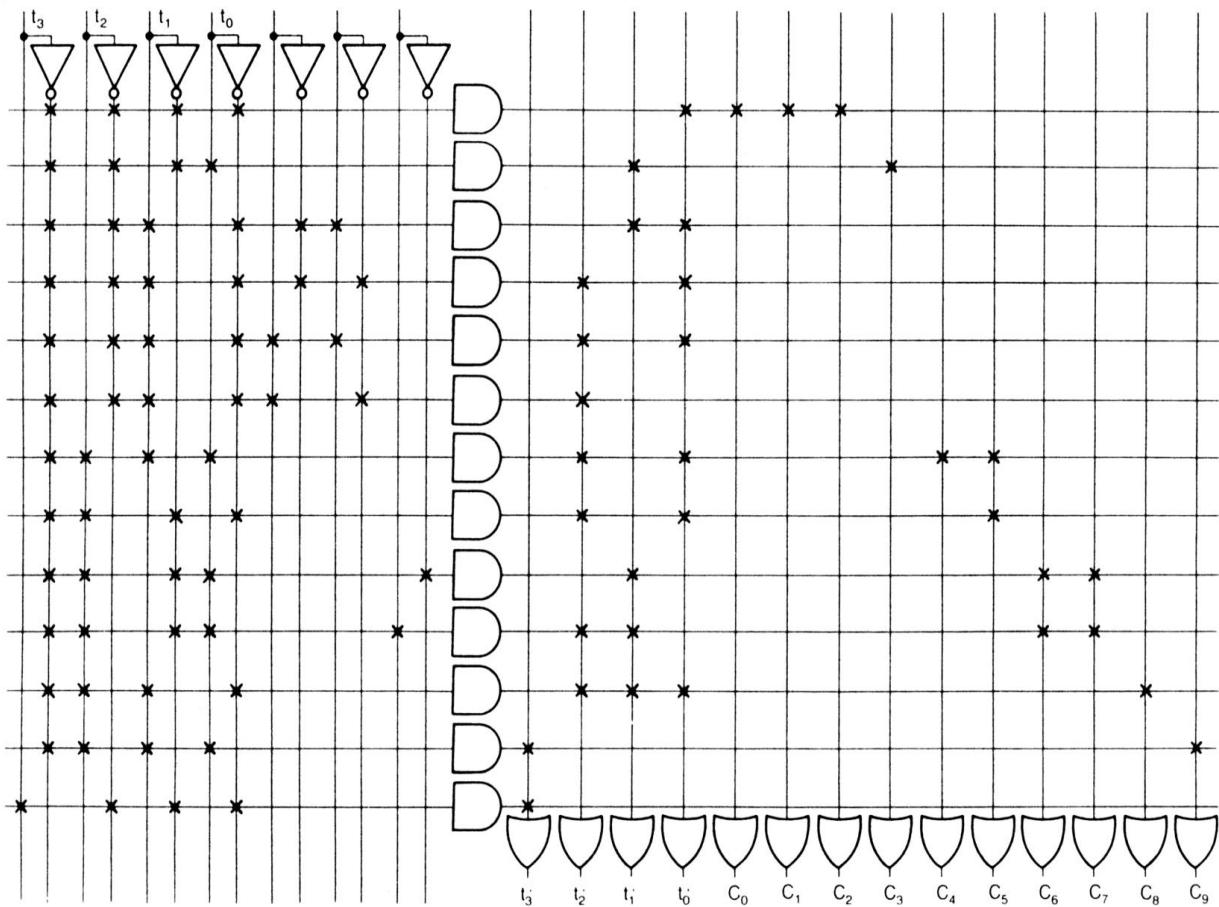
Los biestables contienen la información relativa al estado en que se encuentra el sistema

La PLA utiliza esta información de estado, junto con las entradas externas, para generar el siguiente estado



# Diseño de una UC cableada

- Organización de UC cableada basada en una PLA:



## Ventajas:

- ✓ Minimización del esfuerzo de diseño.
- ✓ Mayor flexibilidad y fiabilidad.
- ✓ Ahorro de espacio y potencia.

# Ejemplo de UC cableada

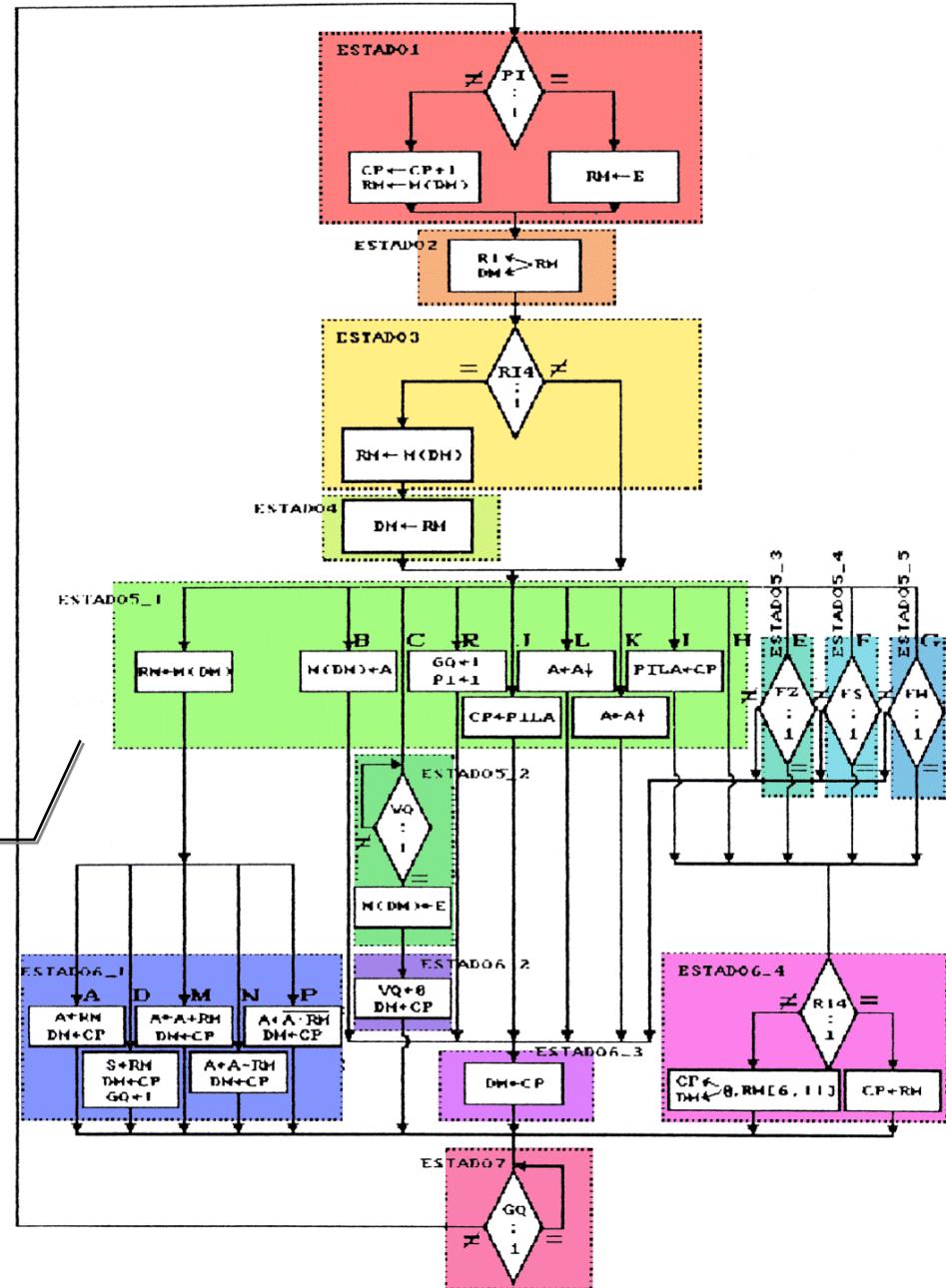
- **Implementación de una unidad de control cableada sencilla (ODE)**
- **Pasos a seguir para llegar al diseño físico:**
  1. Definir una máquina de estados finitos
  2. Describir dicha máquina en un lenguaje de alto nivel
  3. Generar la tabla de verdad para la PLA
  4. Minimizar la tabla de verdad
  5. Diseñar físicamente la PLA partiendo de la tabla de verdad

# Ej. de UC cableada

## ■ 1. Definir una máquina de estados finitos

- Dado el diagrama de flujo de la UC de ODE, detallamos éste como un conjunto finito de estados y transiciones entre ellos.

**Modelo Mealy:** salidas dependen de entradas y estado presente



# Ej. de UC cableada

## ■ 2. Describir dicha máquina en un lenguaje de alto nivel

- El lenguaje concreto depende del programa que utilicemos para “compilar” la descripción de la máquina.
- Estos lenguajes tienen sentencias para definir:
  - entradas y salidas
  - estados y transiciones condicionales e incondicionales entre estados

```

-- ***** UNIDAD DE CONTROL DE ODE *****
-- Definicion de líneas de entrada y de salida

INPUTS: RIO R11 R12 R13 R14 PI V0 G0 FS FW FZ V;
OUTPUTS: A B C D E F G H I J K L M N P R S Y T U R1 R2 R3 R4 R5 R6 INSTR_C INSTR_D INSTR_R;

-- Estado de reset

RESET ON V TO STATE ESTADO07 (A=? E=? F=? G=? I J=? L=? M=? N=? R=? S=? Y=?);

-- Definicion de estados

ESTADO01: IF PI THEN ESTADO02 (A=? E=? F=? G=? I J=0 K L=? M=? N=? R=? S=? Y=? R1)
          ELSE ESTADO02 (A=? D E=? F=? G=? I J=1 K L=? M=0 N=0 R=? S=? Y=? R1);

ESTADO02: GOTO ESTADO03 (A=? E=? F=? G=0 H I J=? L=0 M=1 N=? R=? S=? Y=? U R2);

ESTADO03: IF R14 THEN ESTADO04 (A=? E=? F=? G=? I J=1 K L=? M=? N=? R=? S=? Y=? R3)
          ELSE ESTADO05_1 (A=? E=? F=? G=? I J=? L=? M=? N=? R=? S=? Y=? R3);

ESTADO04: GOTO ESTADOS_1 (A=? E=? F=? G=0 H I J=? L=1 M=1 N=? R=? S=? Y=? R4 INSTR_C=0 INSTR_D=0 INSTR_R=0);

ESTADOS_1: CASE (RIO R11 R12 R13)
    0 0 0 1 => ESTADO06_3 (A=? E=? F=? G=? I J=? L=? M=0 N=1 R=? S=? Y=? RS);
    0 0 1 0 => ESTADO05_2 (A=? E=? F=? G=? I J=? L=? M=? N=? R=? S=? Y=? RS);
    1 1 0 1 => ESTADO06_3 (A=? E=? F=? G=? I J=? L=? M=? N=? R=? S=? Y=? RS INSTR_R);
    1 0 0 1 => ESTADO06_3 (A=? B C E=0 F=1 G=? I J=? L=? M=? N=? R=? S=? Y=? RS);
    1 0 1 1 => ESTADO06_3 (A=? E=? F=? G=? I J=? L=? M=? N=? R=1 S=0 Y=0 T RS);
    1 0 0 0 => ESTADO06_3 (A=? E=? F=? G=? I J=? L=? M=? N=? R=? S=? Y=? RS);
    0 1 0 1 => ESTADO05_3 (A=? E=? F=? G=? I J=? L=? M=? N=? R=? S=? Y=? RS);
    0 1 0 0 => ESTADO05_3 (A=? E=? F=? G=? I J=? L=? M=? N=? R=? S=? Y=? RS);
    0 1 1 0 => ESTADO05_4 (A=? E=? F=? G=? I J=? L=? M=? N=? R=? S=? Y=? RS);
    0 1 1 1 => ESTADO05_5 (A=? E=? F=? G=? I J=? L=? M=? N=? R=? S=? Y=? RS);
    0 1 0 1 => ESTADO06_4 (A=? E=? F=? G=? I J=? L=? M=? N=? R=? S=? Y=? RS);
    1 0 0 0 => ESTADO06_4 (A=? B E=1 F=0 G=? I J=? L=? M=? N=? R=? S=? Y=? RS);
    => ESTADO06_1 (A=? E=? F=? G=? I J=1 K L=? M=? N=? R=? S=? Y=? RS);

ESTADO05_2: IF V0 THEN ESTADO06_2 (A=? E=? F=? G=? I J=? L=? M=0 N=0 R=? S=? Y=? RS)
             ELSE LOOP (A=? E=? F=? G=? I J=? L=? M=? N=? R=? S=? Y=? RS);

ESTADO05_3: IF FZ THEN ESTADO06_4 (A=? E=? F=? G=? I J=? L=? M=? N=? R=? S=? Y=? RS)
             ELSE ESTADO06_3 (A=? E=? F=? G=? I J=? L=? M=? N=? R=? S=? Y=? RS);

ESTADO05_4: IF FS THEN ESTADO06_4 (A=? E=? F=? G=? I J=? L=? M=? N=? R=? S=? Y=? RS)
             ELSE ESTADO06_3 (A=? E=? F=? G=? I J=? L=? M=? N=? R=? S=? Y=? RS);

ESTADO05_5: IF FW THEN ESTADO06_4 (A=? E=? F=? G=? I J=? L=? M=? N=? R=? S=? Y=? RS)
             ELSE ESTADO06_3 (A=? E=? F=? G=? I J=? L=? M=? N=? R=? S=? Y=? RS);

ESTADO06_1: CASE (RIO R11 R12 R13)
    0 0 0 0 => ESTADO07 (A=? E=? F=? G=1 H I J=? L=1 M=1 N=? R=1 S=1 Y=0 T R6);
    0 0 1 1 => ESTADO07 (A=? E=? F=? G=1 H I J=? L=1 M=1 N=? P R=? S=? Y=? R6 INSTR_D);
    1 1 0 0 => ESTADO07 (A=? E=? F=? G=1 H I J=? L=1 M=1 N=? R=0 S=0 Y=1 T R6);
    1 1 0 1 => ESTADO07 (A=? E=? F=? G=1 H I J=? L=1 M=1 N=? R=0 S=1 Y=0 T R6);
    1 1 1 0 => ESTADO07 (A=? E=? F=? G=1 H I J=? L=1 M=1 N=? R=0 S=1 Y=1 T R6);
    ENCASE => ANY (A=? B=? C=? D=? E=? F=? G=? H=? I=? J=? K=? L=? M=? N=? P=? R=? S=? Y=? T=? U=? R1=? R2=? R3=? R4=? RS=? R6=?)

ESTADO06_2: GOTO ESTADO07 (A=? E=? F=? G=1 H I J=? L=? M=? N=? R=? S=? Y=? R6 INSTR_C);

ESTADO06_3: GOTO ESTADO07 (A=? E=? F=? G=0 H I J=? L=? M=? N=? R=? S=? Y=? R6);

ESTADO06_4: IF R14 THEN ESTADO7 (A=0 C E=? F=? G=0 I J=? L=1 M=1 N=? R=? S=? Y=? R6)
             ELSE ESTADO7 (A=0 C E=? F=? G=0 H I J=? L=0 M=1 N=? R=? S=? Y=? R6);

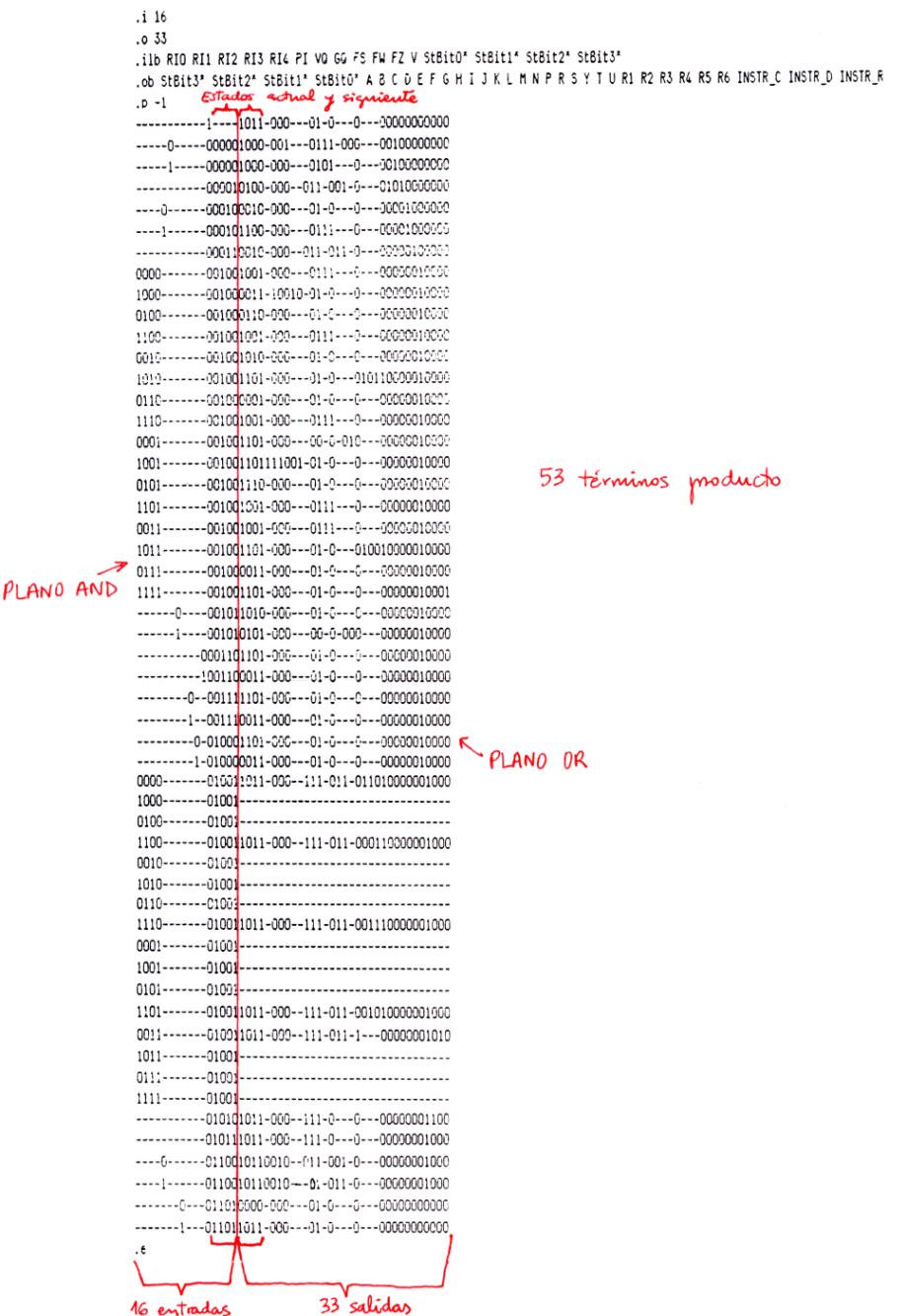
ESTADO07: IF GO THEN LOOP (A=? E=? F=? G=? I J=? L=? M=? N=? R=? S=? Y=?)
             ELSE ESTADO1 (A=? E=? F=? G=? I J=? L=? M=? N=? R=? S=? Y=?);

```

# Ej. de UC cableada

## ■ 3. Generar tabla de verdad necesaria para PLA

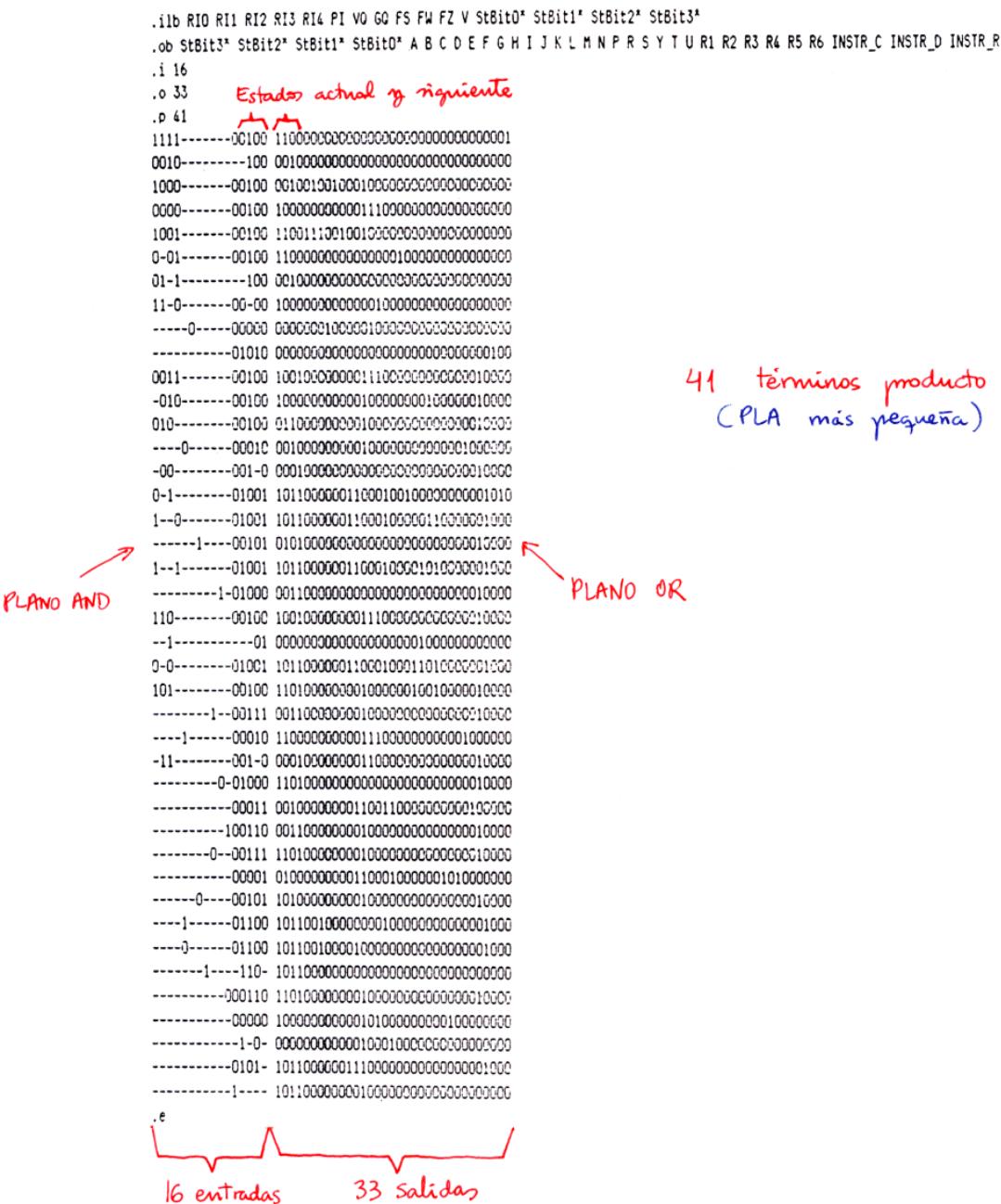
- Según la descripción que hayamos hecho de la máquina de estados...
  - podemos usar un programa que use el modelo **Mealy**
    - salidas dependen de entradas y de estado presente
  - o uno que use el modelo **Moore**
    - salidas dependen exclusivamente de estado actual



# Ej. de UC cableada

## ■ 4. Minimizar la tabla de verdad

- Mediante un programa que utiliza algoritmos heurísticos rápidos.



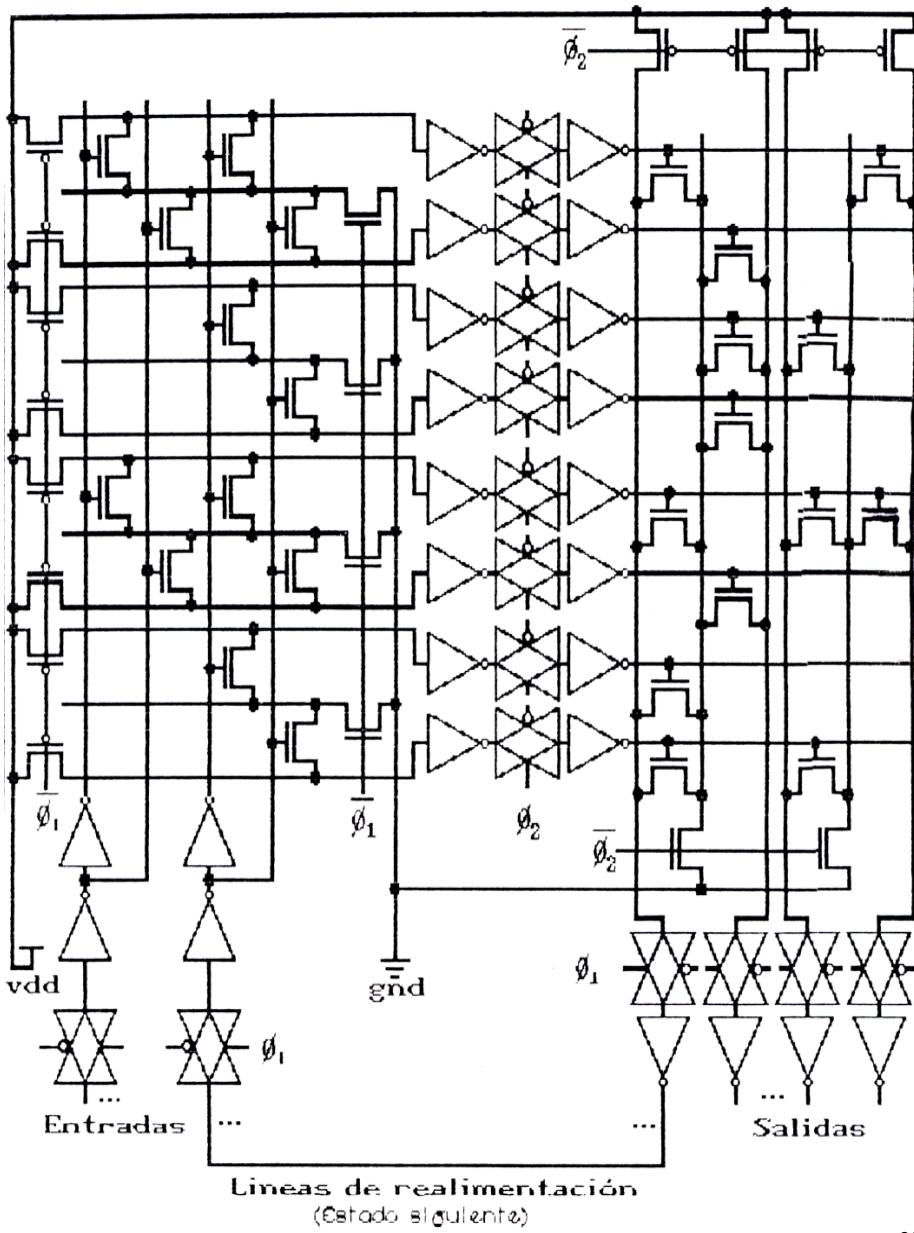
# Ej. de UC cableada

- **5. Diseñar físicamente la PLA partiendo de la tabla de verdad minimizada**
  - Automáticamente:
    - Mediante un programa especial para diseño de layouts de PLA.
  - Semiautomáticamente:
    - Diseñando mediante un programa de CAD de circuitos VLSI cada una de las celdas que, repetidas convenientemente, forman la PLA.
    - Dando una especificación de cómo han de colocarse (tabla de verdad minimizada).

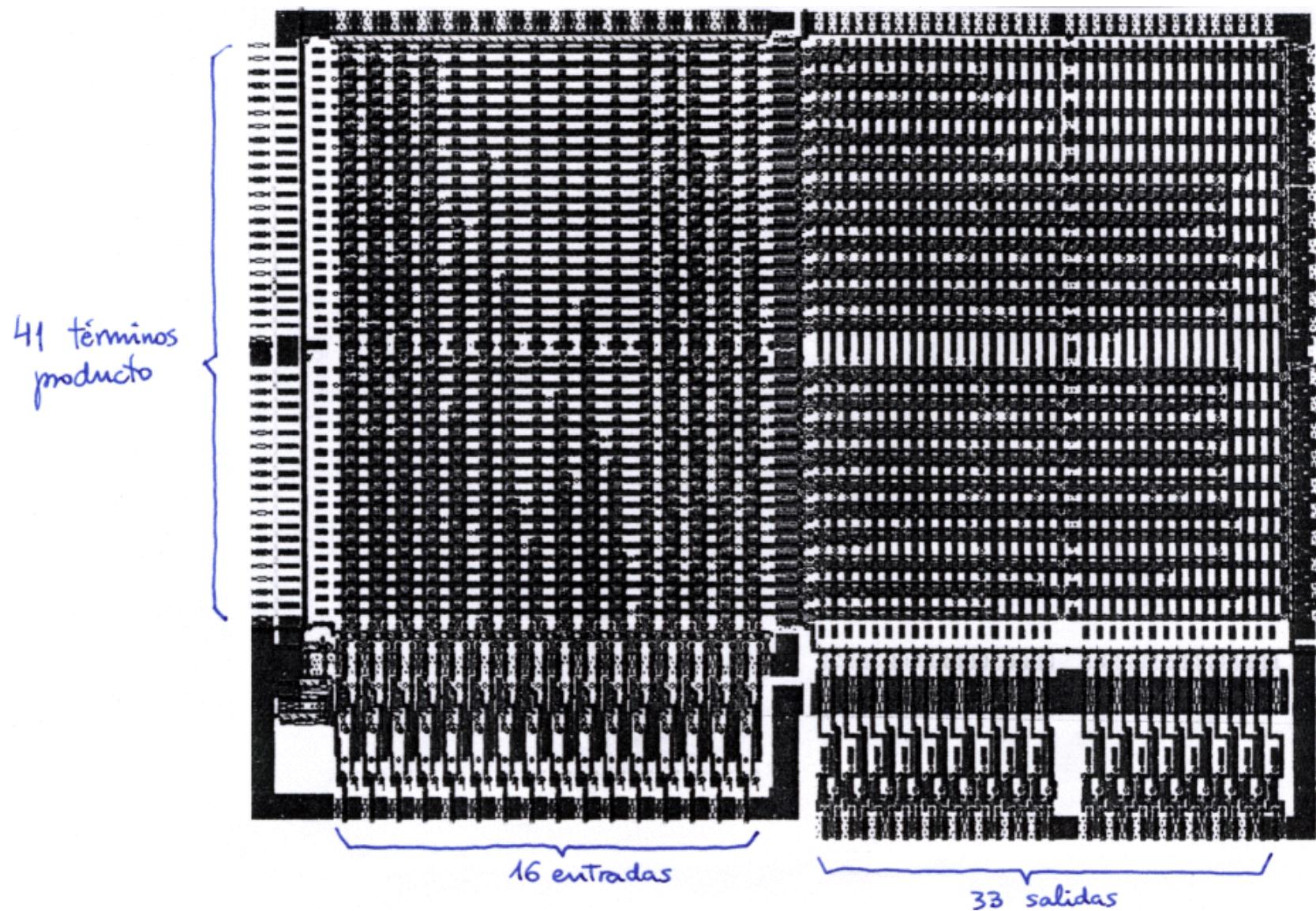
# Ej. de UC cableada

- **Esquema simplificado de la PLA usada para la UC de ODE**

- CMOS de dos fases de reloj
  - No hacen falta biestables de estado siguiente
  - $\Phi_1=1$  se leen las entradas y se precarga el plano AND
  - $\Phi_2=1$  se evalúa el plano AND y se precarga el plano OR
  - $\Phi_2=0$  se evalúa el plano OR



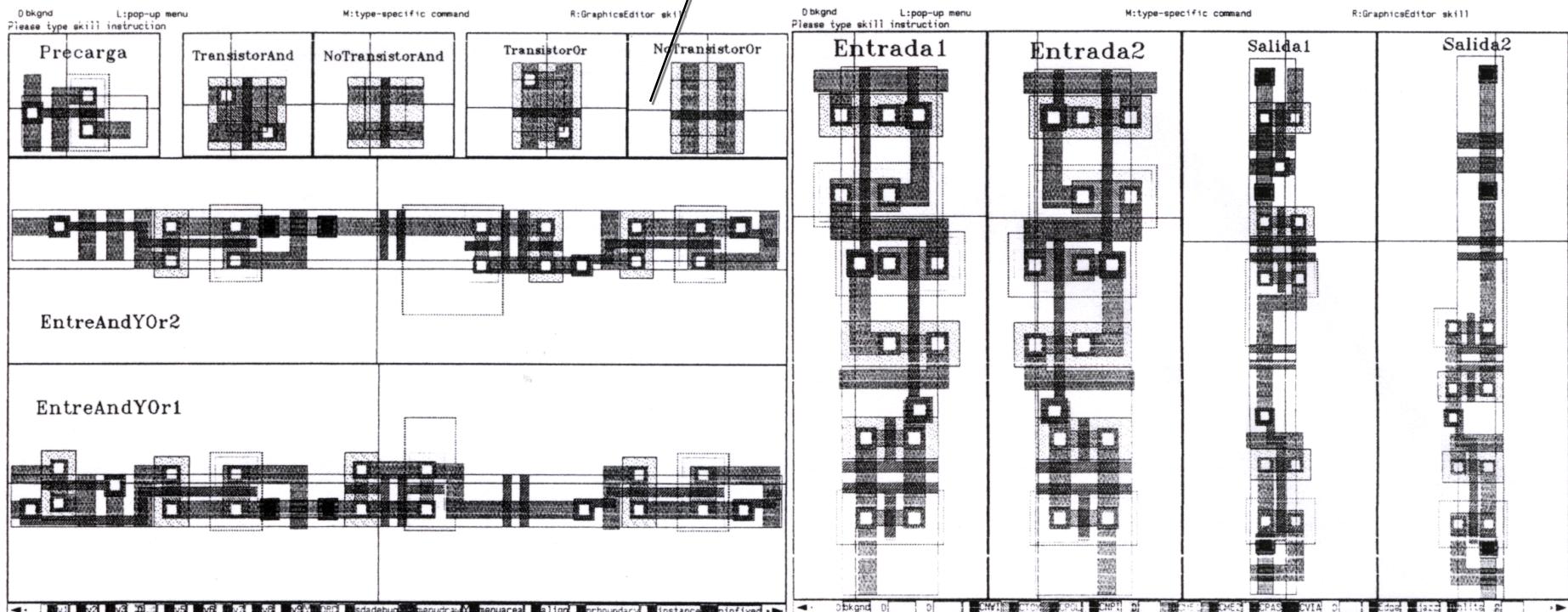
# Ej. de UC cableada



# Ej. de UC cableada

- PLA diseñada semiautomáticamente

Detalle de las celdas básicas que constituyen la PLA diseñadas con un programa de CAD.  
Un programa puede unirlas de acuerdo con el archivo de la tabla de verdad minimizada.



# Concepto de UC microprogramada

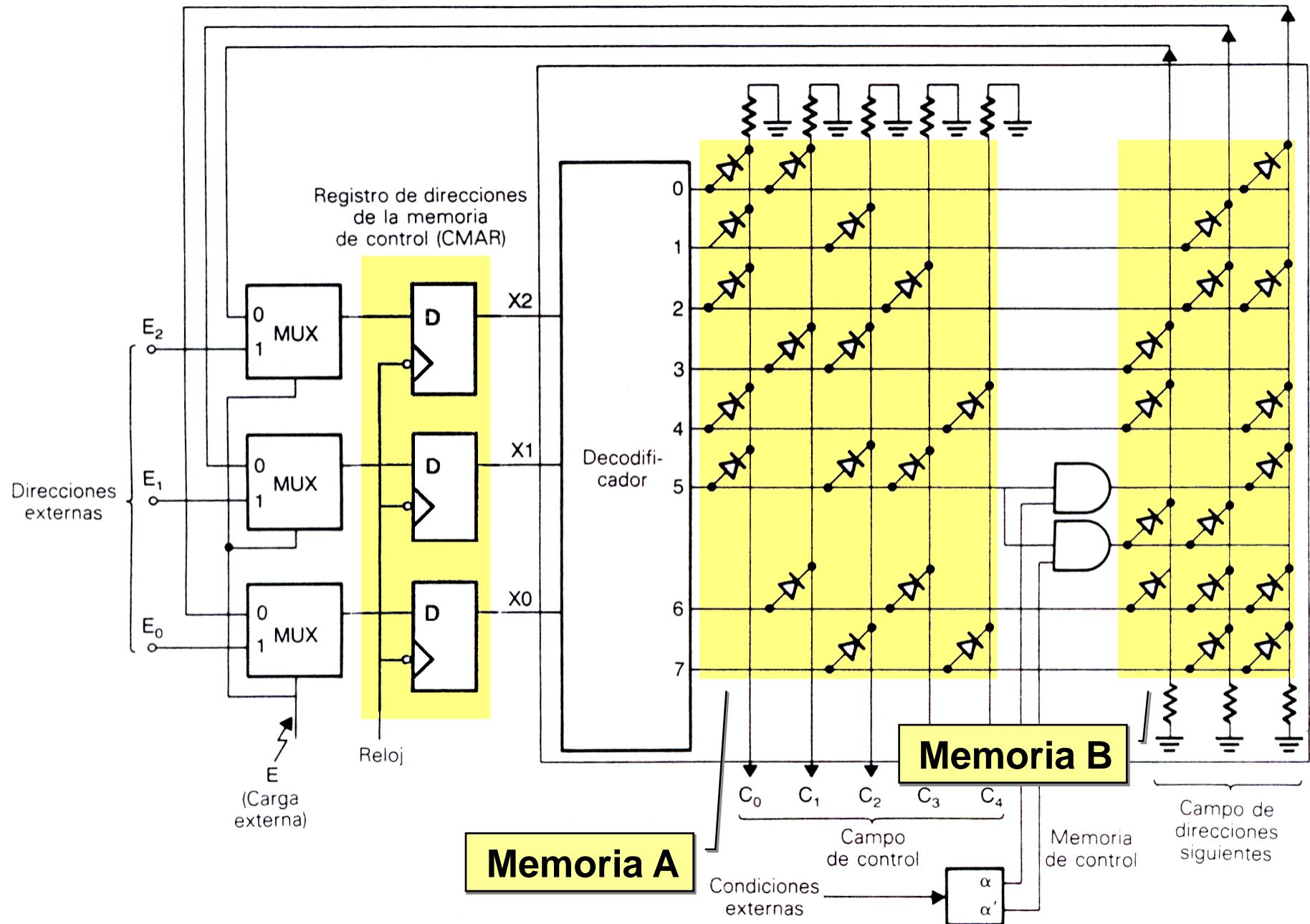
## ■ Idea básica:

- Emplear una memoria (de control) para almacenar las señales de control de los períodos de cada instrucción

## ■ Origen histórico

- Maurice V. Wilkes (1913-2010) en 1951-1953 propone el siguiente esquema:
  - Dos memorias A y B, construidas con matrices de diodos.
    - Las señales de control se encuentran almacenadas en la memoria A.
    - La memoria B contiene la dirección de la siguiente microinstrucción.
  - Se permiten microbifurcaciones condicionales, mediante un biestable y un decodificador que selecciona entre dos direcciones de la matriz B.
- Más info.: <http://www.cs.clemson.edu/~mark/uprog.html>





# Concepto de UC microprogramada

## ■ Definiciones:

- **Microinstrucción**: cada palabra de la memoria de control
- **Microprograma**: conjunto ordenado de microinstrucciones cuyas señales de control constituyen el cronograma de una (macro)instrucción del lenguaje máquina.
- Ejecución de un microprograma: lectura en cada pulso de reloj de una de las microinstrucciones que lo forman, enviando las señales leídas a la unidad de proceso como señales de control.
- **Microcódigo**: conjunto de los microprogramas de una máquina.

# Concepto de UC microprogramada

## ■ Ventajas de la microprogramación:

- Simplicidad conceptual.
  - La información de control reside en una memoria.
- Se pueden incluir, sin dificultades, instrucciones complejas, de muchos ciclos de duración.
  - El único límite es el tamaño de la memoria de control.
- Las correcciones, modificaciones y ampliaciones son mucho más fáciles de hacer que en una unidad de control cableada.
  - No hay que rediseñar toda la unidad, sino sólo cambiar el contenido de algunas posiciones de la memoria de control.
- Permite construir computadores con varios juegos de instrucciones, cambiando el contenido de la memoria de control (si es RAM permite emular otros computadores).

# Concepto de UC microprogramada

- **Desventaja de la microprogramación:**
  - Lentitud frente a cableada, debido a una menor capacidad de expresar paralelismo de las microinstrucciones.

# Unidad de control

## ■ Camino de datos

- Unidad de procesamiento y unidad de control
- Unidad de procesamiento con un bus
- Unidad de procesamiento con múltiples buses

## ■ Unidades de control cableadas y microprogramadas

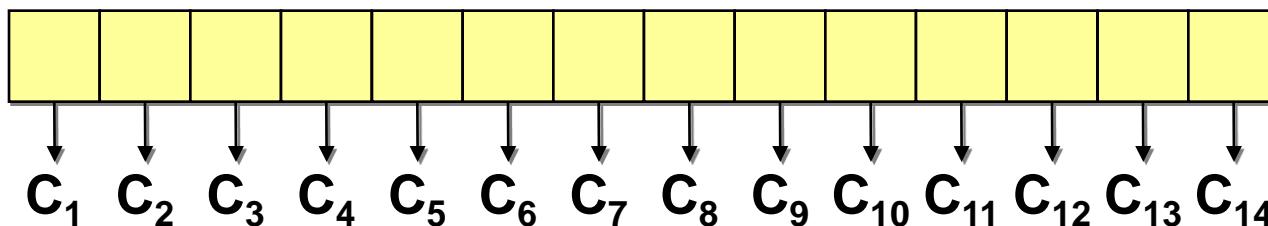
- Diseño de una UC cableada
- Ejemplo de UC cableada
- Concepto de UC microprogramada

## ■ Control microprogramado

- Formato de las microinstrucciones
- Nanoprogramación
- Secuenciamiento de microinstrucciones
- Ejemplo de arquitectura microprogramada

# Formato de las microinstrucciones

- Las señales de control que gobiernan un mismo elemento del datapath se suelen agrupar en campos.
  - Ejemplos:
    - señales triestado que controlan el acceso a un bus
    - señales de operación de la ALU
    - señales de control de la memoria
- Formato no codificado:
  - Hay un bit para cada señal de control de un campo

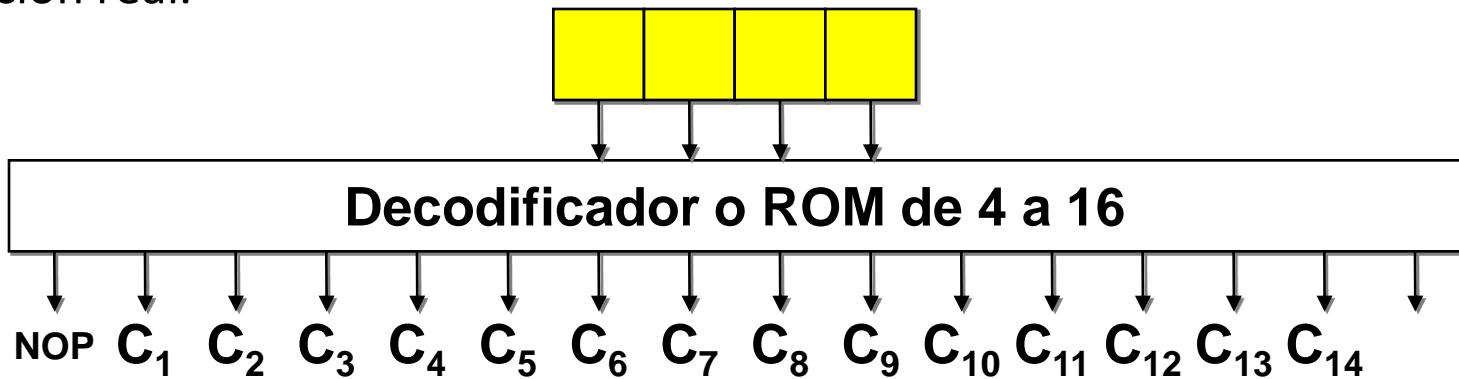


# Formato de las microinstrucciones

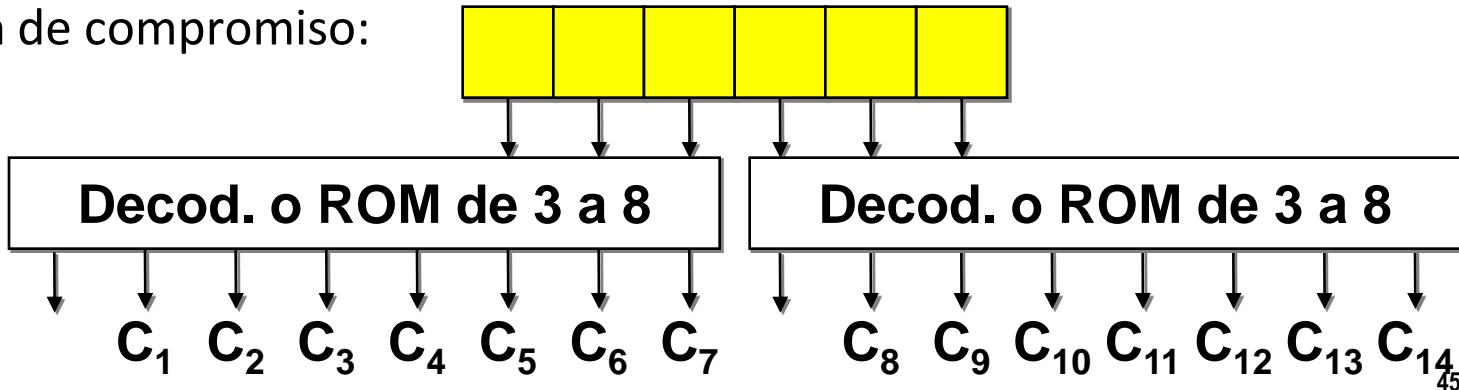
## ■ Formato codificado:

- Para acortar el tamaño de las microinstrucciones se codifican todos o alguno de sus campos.

Inconveniente: Hay que incluir decodificadores para extraer la información real.



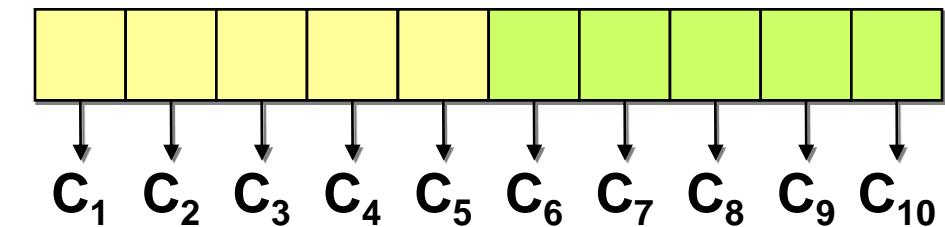
- Solución de compromiso:



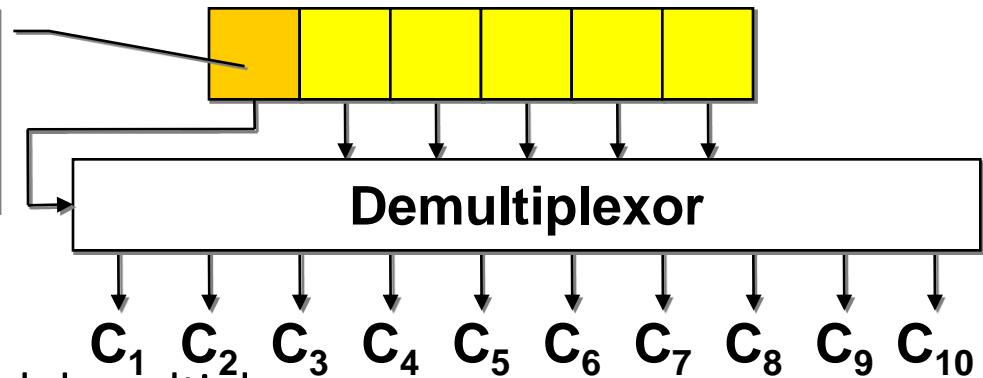
# Formato de las microinstrucciones

## ■ Solapamiento de campos:

- Si sólo unas pocas señales de control están activas en cada ciclo, o existen con frecuencia señales excluyentes, que no se pueden activar simultáneamente...
- ...se puede acortar la longitud de las microinstrucciones solapando campos.



**La señal adicional de control define si los bits corresponden al campo 1 o al campo 2**



- Inconvenientes:
  - Retardo introducido por el demultiplexor.
  - Hace incompatibles las operaciones de los campos solapados.

# Formato de las microinstrucciones

## Micro-programación vertical:

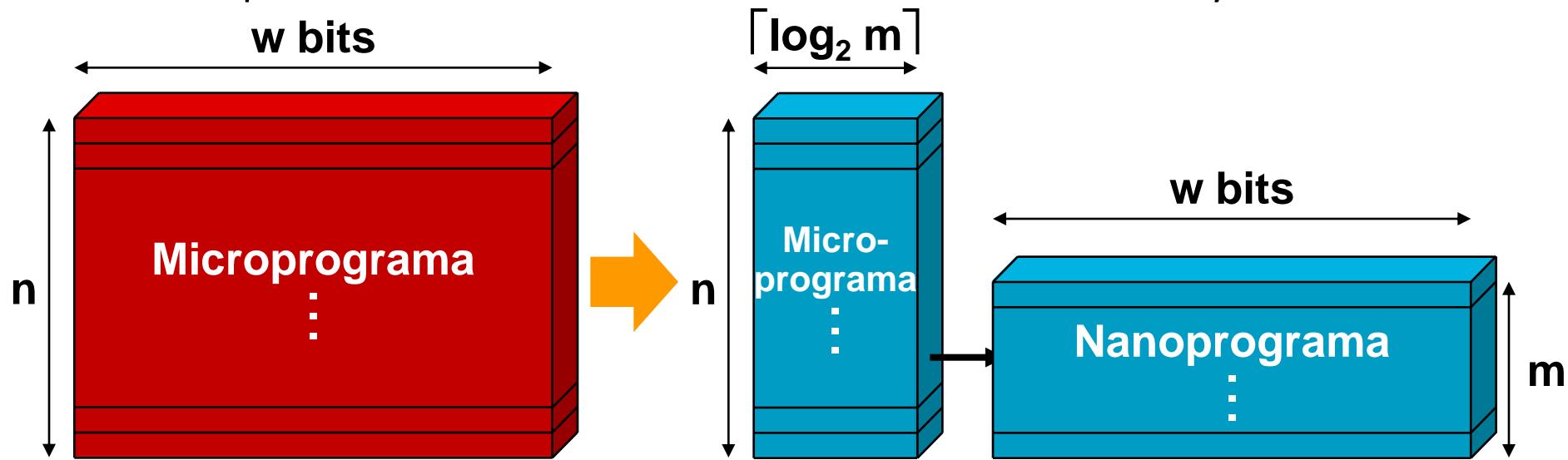
- Mucha codificación
- ✓ Microinstrucciones cortas
- ✗ Escasa capacidad para expresar paralelismo (la longitud del programa se ve incrementada)

## Micro-programación horizontal:

- Ninguna o escasa codificación
- ✓ Capacidad para expresar un alto grado de paralelismo en las microoperaciones a ejecutar (simultáneamente)
- ✗ Microinstrucciones largas

# Nanoprogramación

- **Objetivo: reducir el tamaño de la memoria de control**
  - Implica una memoria a dos niveles: memoria de control y nanomemoria.



**Microprograma original con  $n$  μinstrucciones de  $w$  bits. Tamaño =  $n \cdot w$**

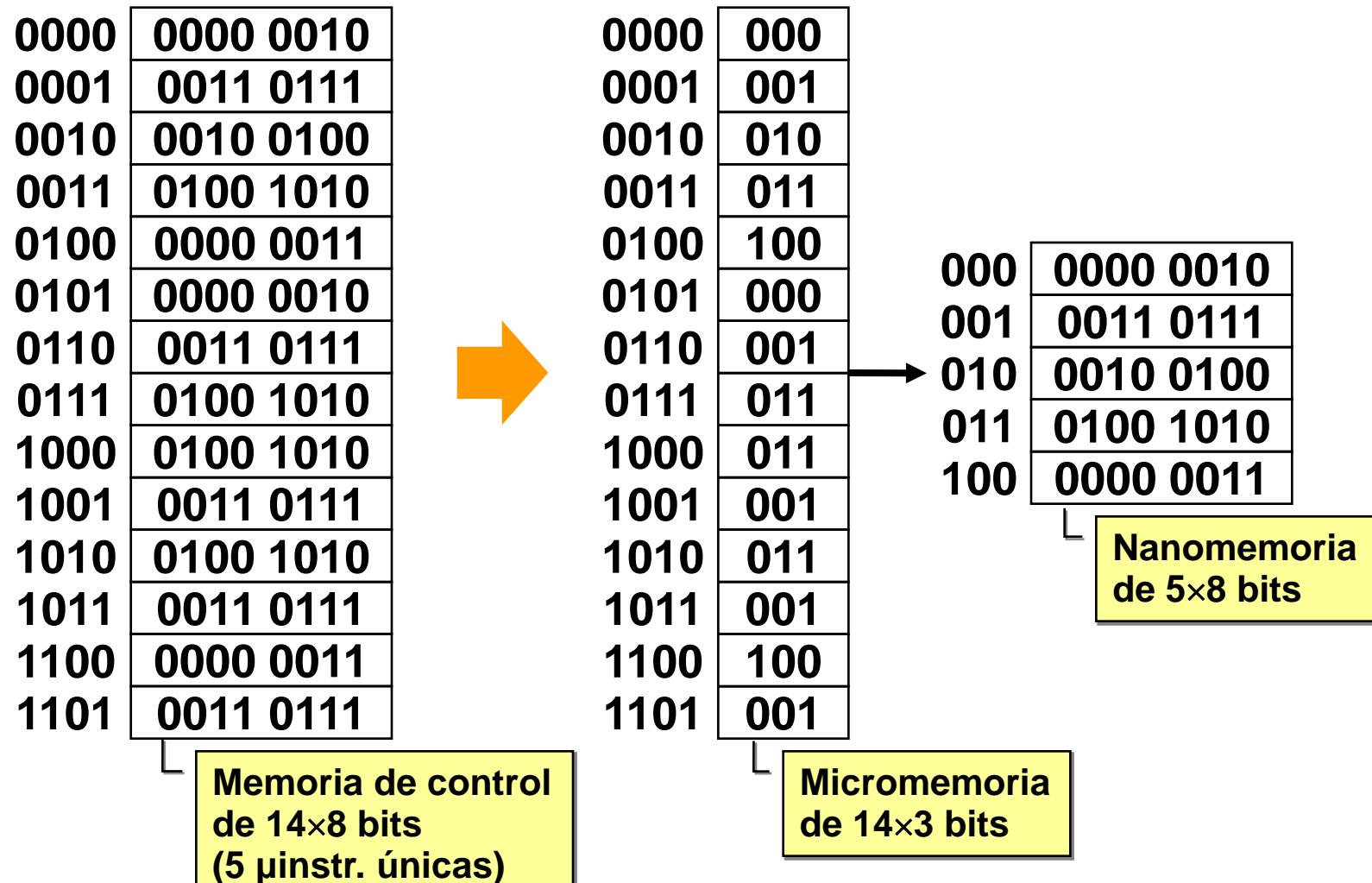
**$m << n$  μinstrucciones únicas de  $2^w$  posibles**

**Se reemplaza cada μinstrucción por su dirección en la nanomemoria  
Tamaño:  $n \cdot \lceil \log_2 m \rceil$**

**Contiene las  $m$  μinstrucciones diferentes (cada una se incluye una sola vez).  
Tamaño:  $m \cdot w$**

**Ahorro de memoria:  $n \cdot w - (n \cdot \lceil \log_2 m \rceil + m \cdot w)$**

# Nanoprogramación. Ejemplo 1

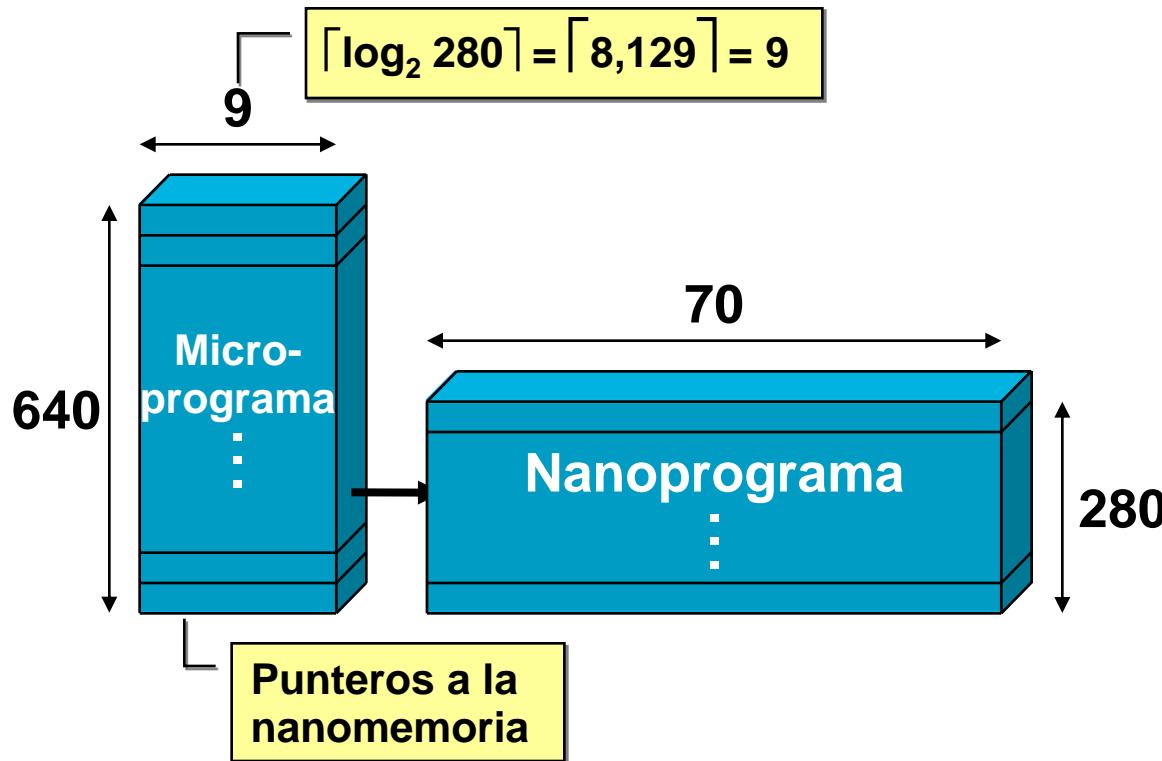


Ahorro de memoria:  $14 \cdot 8 - (14 \cdot \lceil \log_2 5 \rceil + 5 \cdot 8) = 112 - 82 = 30 \text{ bits (27\%)}$

# Nanoprogramación. Ejemplo 2

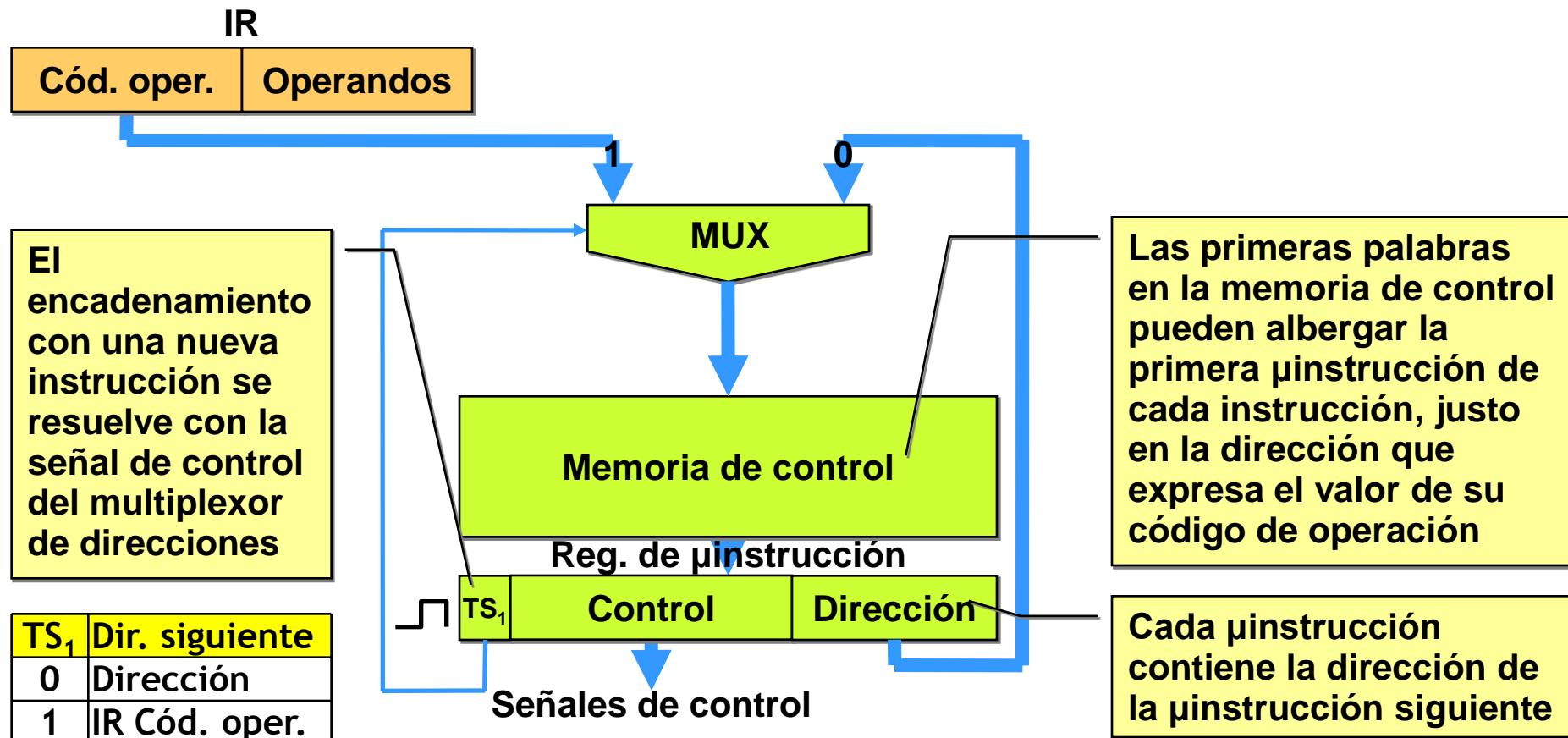
## ■ Estructura de la UC del Motorola 68000

- 640 microinstrucciones, de las cuales 280 son únicas



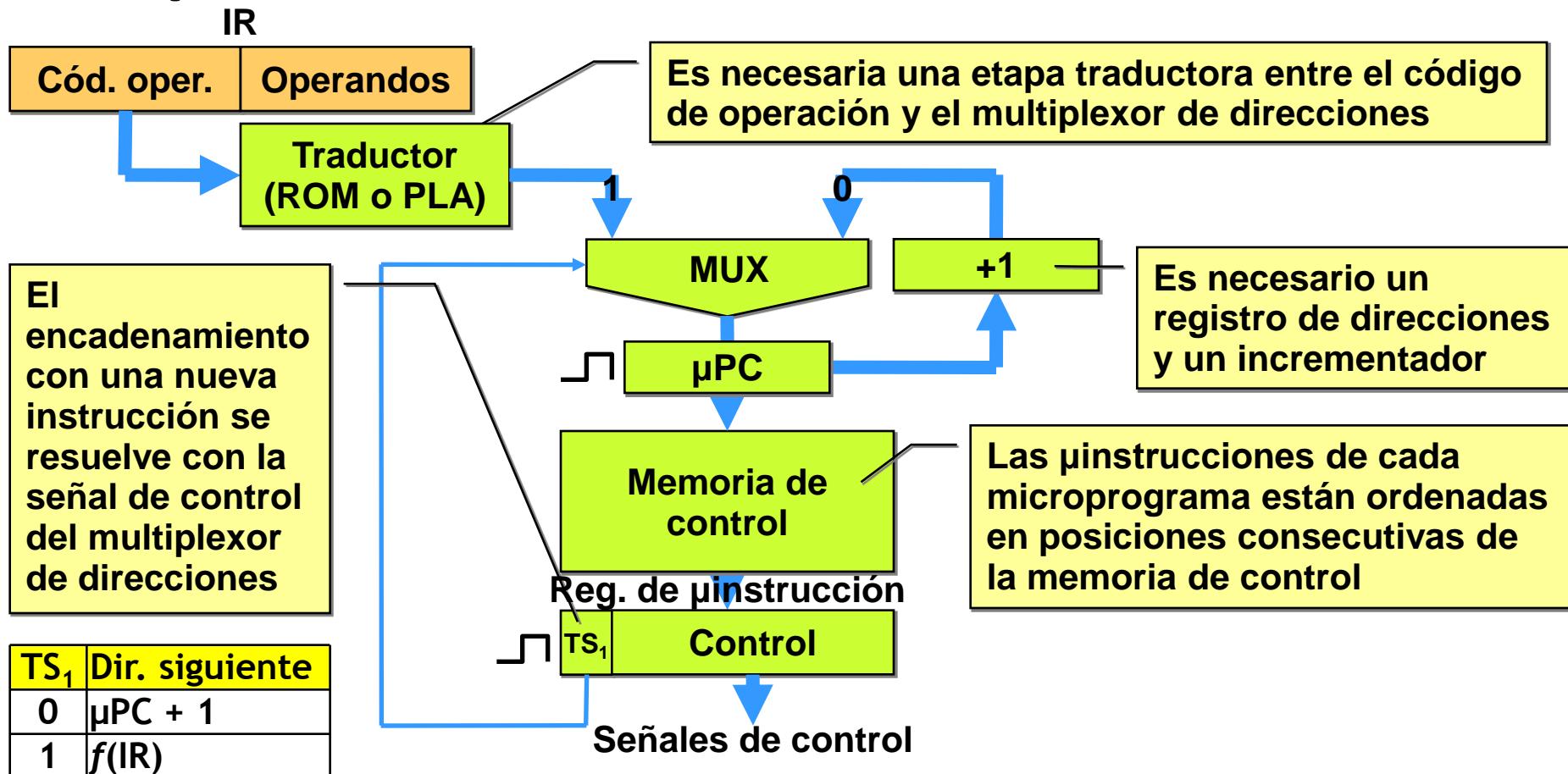
Ahorro de memoria:  $640 \cdot 70 - (640 \cdot 9 + 280 \cdot 70) = 44800 - 25360 = 19440$  bits (43%)

# Secuenciamiento de μinstrucciones explícito



- ✖ Inconveniente: gran cantidad de memoria empleada en el secuenciamiento (dirección de la siguiente microinstrucción)

# Secuenciamiento de μinstrucciones implícito



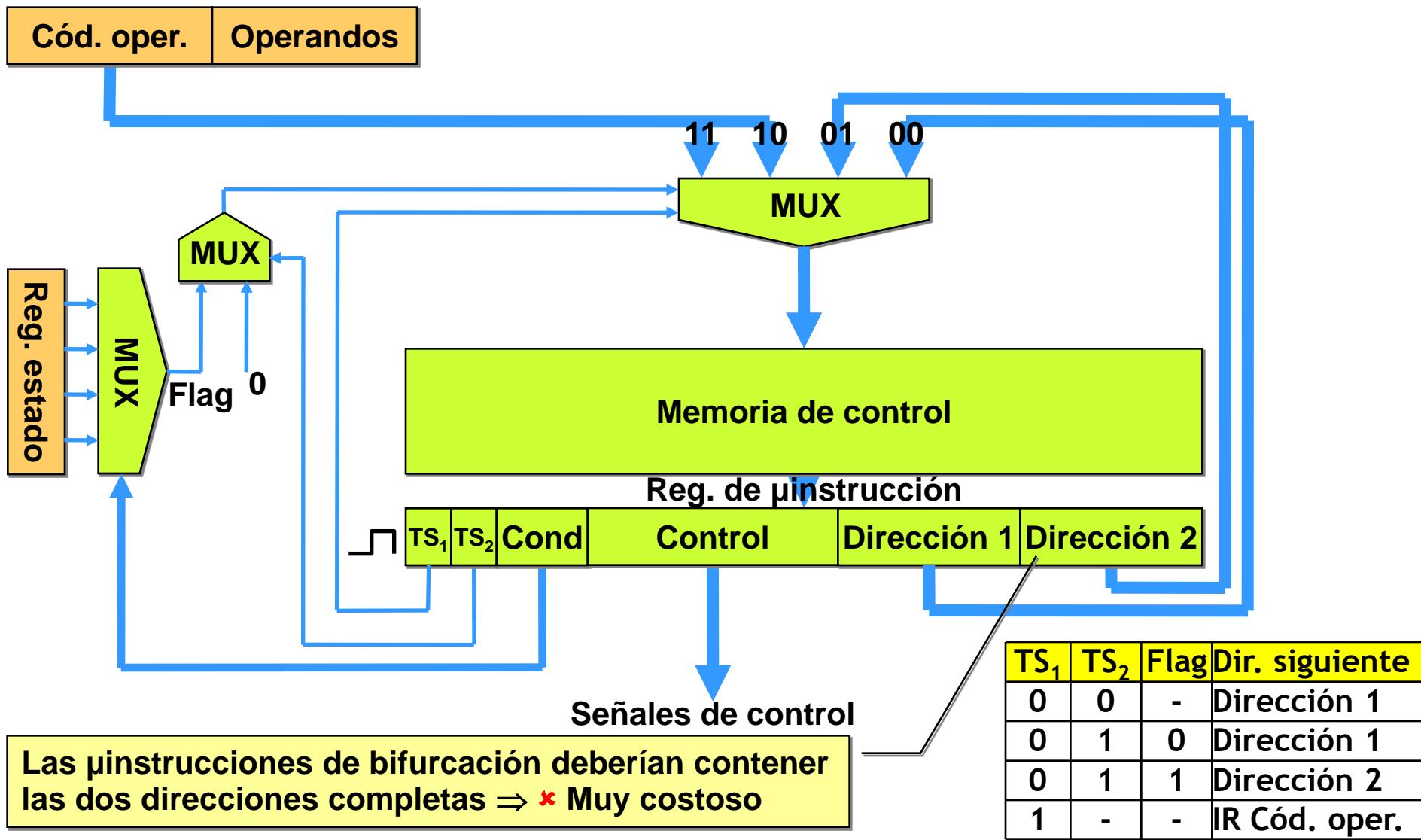
- ✗ Inconveniente: esta estructura sólo permite ejecutar programas lineales

# Secuenciamiento de μinstrucciones

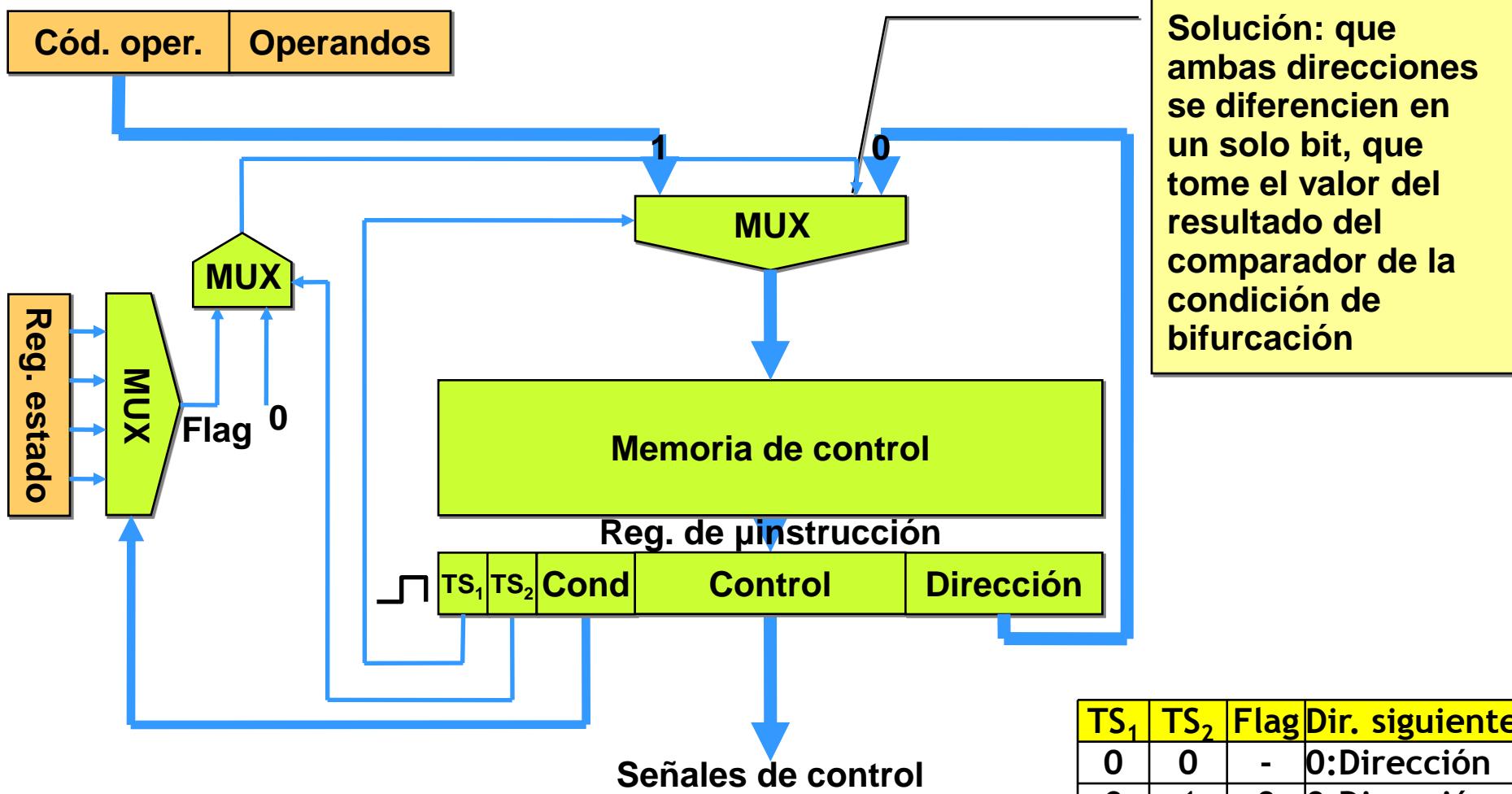
## ■ Microbifurcaciones condicionales

- Las instrucciones máquina de bifurcación condicional presentan dos cronogramas alternativos, diferentes a partir del punto en el que se hace la comprobación de la condición de bifurcación.
- Los microprogramas correspondientes han de presentar una microbifurcación condicional para seleccionar la rama deseada.
- Es necesario que la microinstrucción de bifurcación pueda elegir entre dos direcciones para poder seguir por uno de los dos caminos alternativos.

# μ bifurcaciones condicionales en secuenciamiento explícito (1)

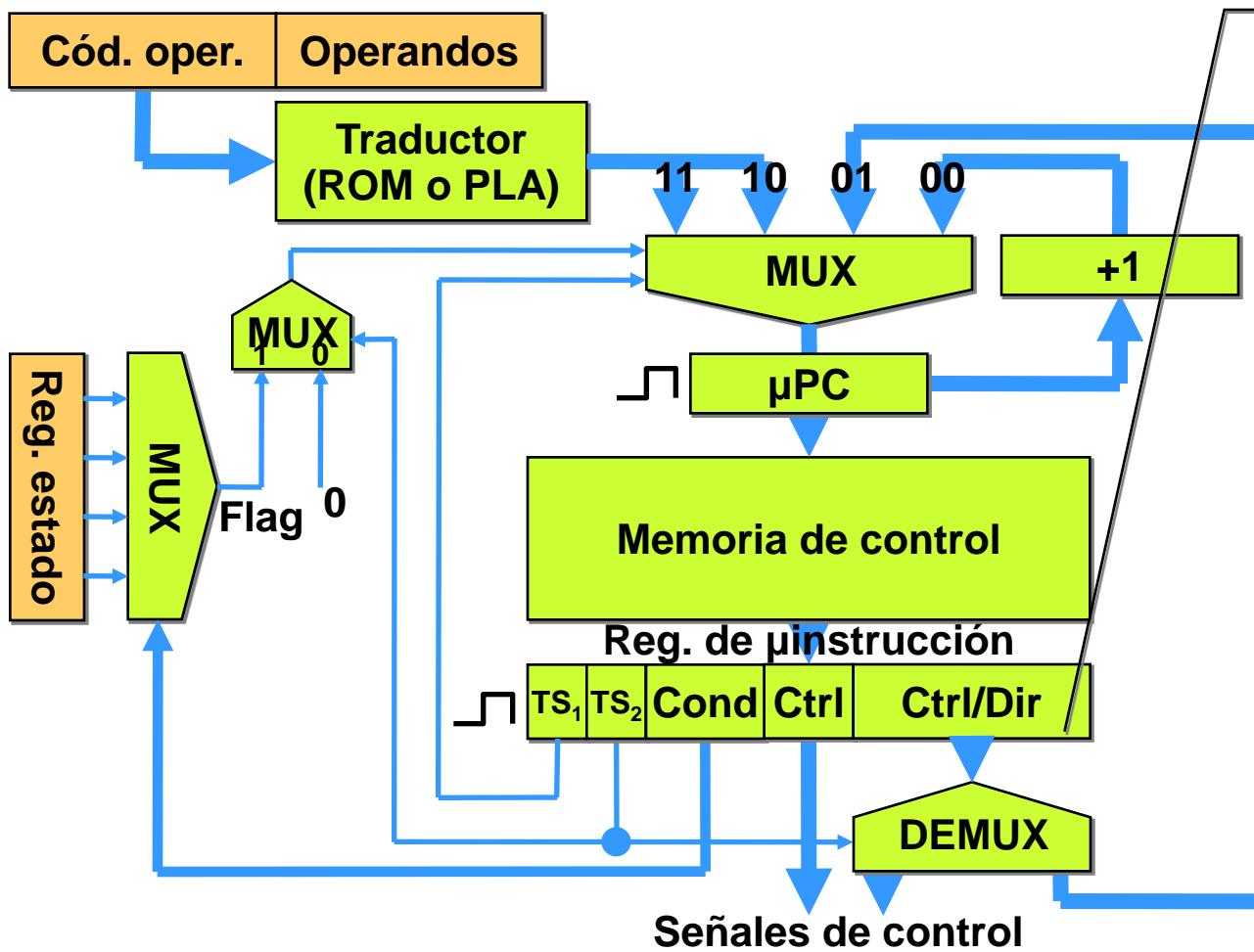


# μ bifurcaciones condicionales en secuenciamiento explícito (2)



TS <sub>1</sub>	TS <sub>2</sub>	Flag	Dir. siguiente
0	0	-	0:Dirección
0	1	0	0:Dirección
0	1	1	1:Dirección
1	-	-	IR Cód. oper.

# μ bifurcaciones condicionales en secuenciamiento implícito



Se debe poder elegir entre la μinstrucción siguiente u otra distinta  
 $\Rightarrow$  la μinstrucción de bifurcación debe contener dicha dirección.  
 Se puede solapar el campo de dirección con otros, puesto que incluir un campo específico de dirección en todas las μinstrucciones conduciría al secuenciamiento explícito

TS <sub>1</sub>	TS <sub>2</sub>	Flag	Dir. siguiente
0	0	-	μPC + 1
0	1	0	μPC + 1
0	1	1	Dirección
1	-	-	f(IR)

# Secuenciamiento de microinstrucciones

## ■ Microbucle

- Existen instrucciones máquina con operaciones repetidas
  - Ejs.:
    - desplazamiento múltiple
    - multiplicación
    - división
    - cadenas
  - Necesidad de microbucle.
- Se puede utilizar la bifurcación condicional más un contador con autodecremento
- Cuando el contador llegue a 0 se bifurca
- El contador debe poder inicializarse desde una microinstrucción

# Secuenciamiento de μinstrucciones

## ■ Microsubrutinas

- Las instrucciones máquina tienen con frecuencia partes comunes
  - Necesidad de microsubrutinas.
- La llamada a microsubrutinas exige un almacenamiento para guardar la dirección de retorno.
  - La solución usual es añadir una pila al registro de direcciones ( $\mu$ PC).

# Control residual

## ■ Control inmediato:

- Hasta aquí, todas las señales de control necesarias para manipular la microarquitectura estaban codificadas en campos de la microinstrucción actual.

## ■ Control residual:

- En ciertos casos puede ser útil que una microinstrucción pueda almacenar señales de control en un registro (de control residual) para usarlas en ciclos posteriores.
- Objetivo: optimizar el tamaño del microprograma.
- Usos:
  - En microsubrutinas o conjuntos compartidos de  $\mu$ instrucciones.
  - En caso de que parte de la información de control permanezca invariable durante muchas  $\mu$ instrucciones.

# Control residual

## ■ En microsubrutinas o conjuntos compartidos de μinstrucciones.

### ■ Ejemplo:

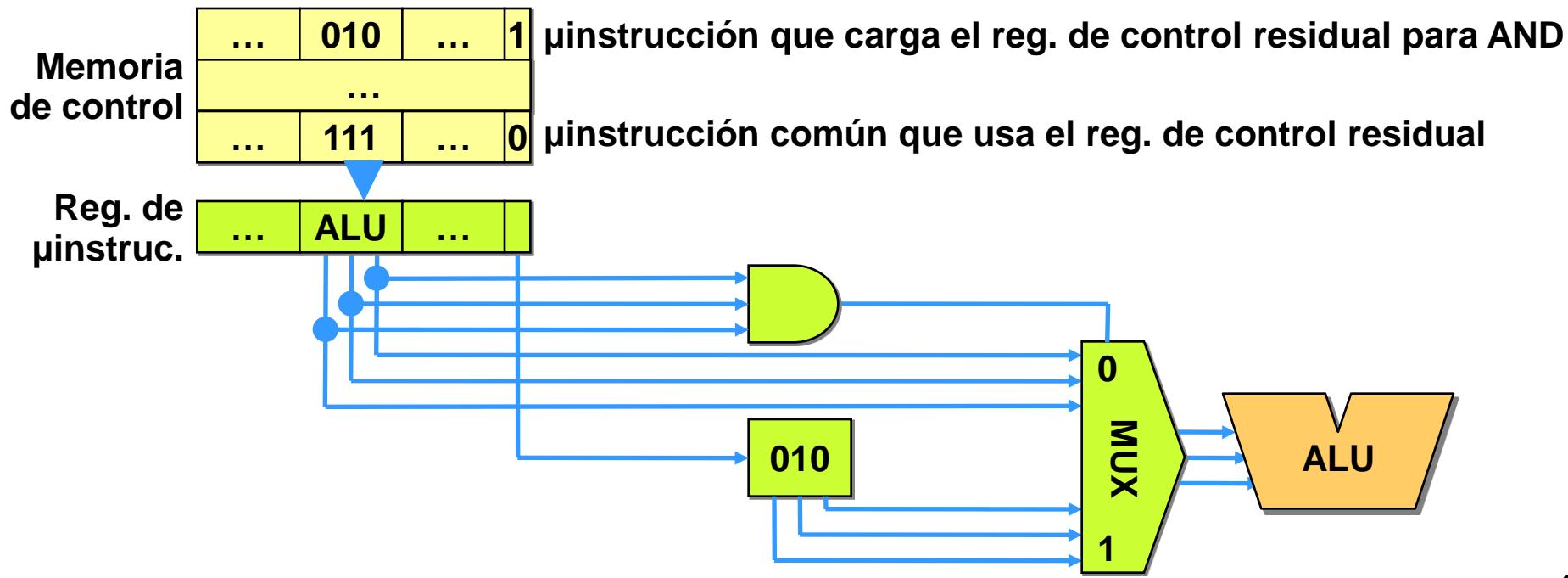
- Procesador con dos instrucciones máquina para realizar las operaciones AND y OR entre dos cadenas de bytes.
- Se puede utilizar una microsubrutina o un conjunto común de microinstrucciones para las dos instrucciones máquina, incorporando un registro de control residual en el diseño.
- Supongamos que la ALU está controlada por 3 bits:

000 Suma
111 Resta
010 AND
011 OR
100 XOR
101 NOR
110 Pasar entrada izquierda
111 No utilizada

Utilizaremos el valor 111 para especificar que las señales de control de la ALU no proceden de este campo sino del registro de control residual

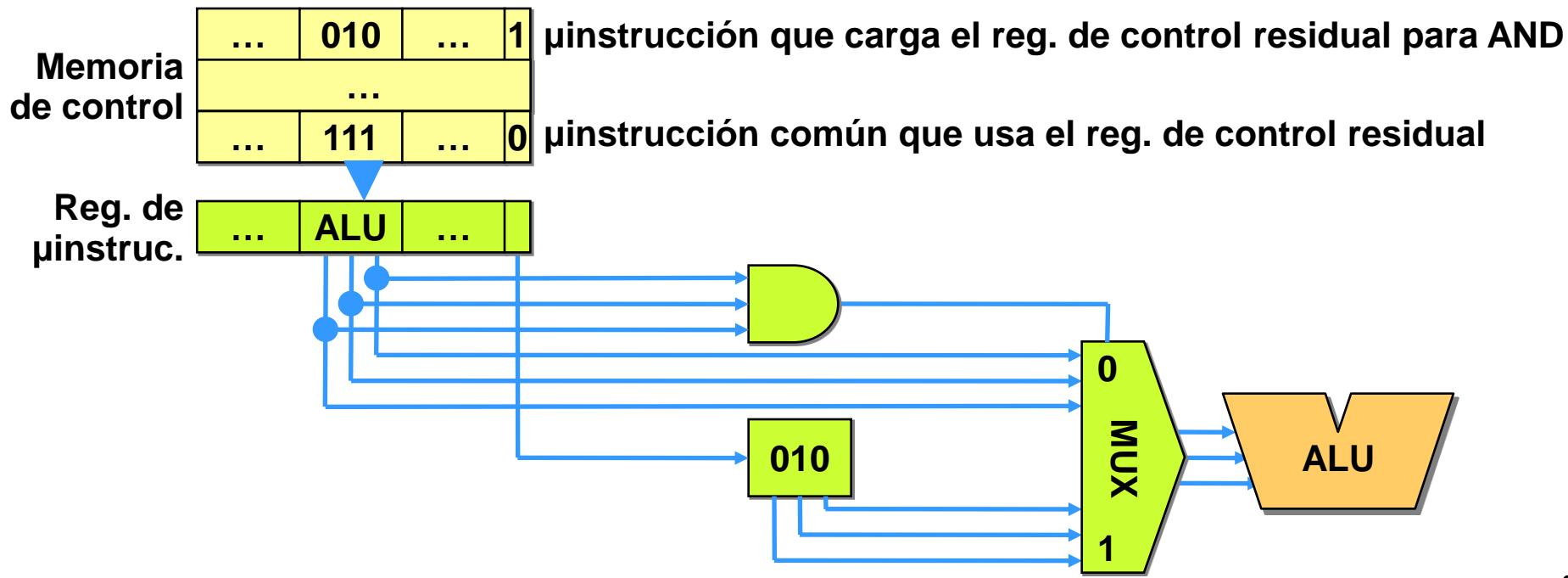
# Control residual

- Necesitamos un método para cargar información en el registro de control residual, por ej. una microinstrucción especial (puede ser la microinstrucción de llamada a microsubrutina) con un campo que indique el valor a almacenar y un bit que indique que se desea almacenar información en el registro.



# Control residual

- El microcódigo de la instrucción que realiza el AND almacenará 010 en el registro de control residual y la correspondiente al OR almacenará 011. Ambas saltarán o llamarán al microcódigo común.



# Control residual

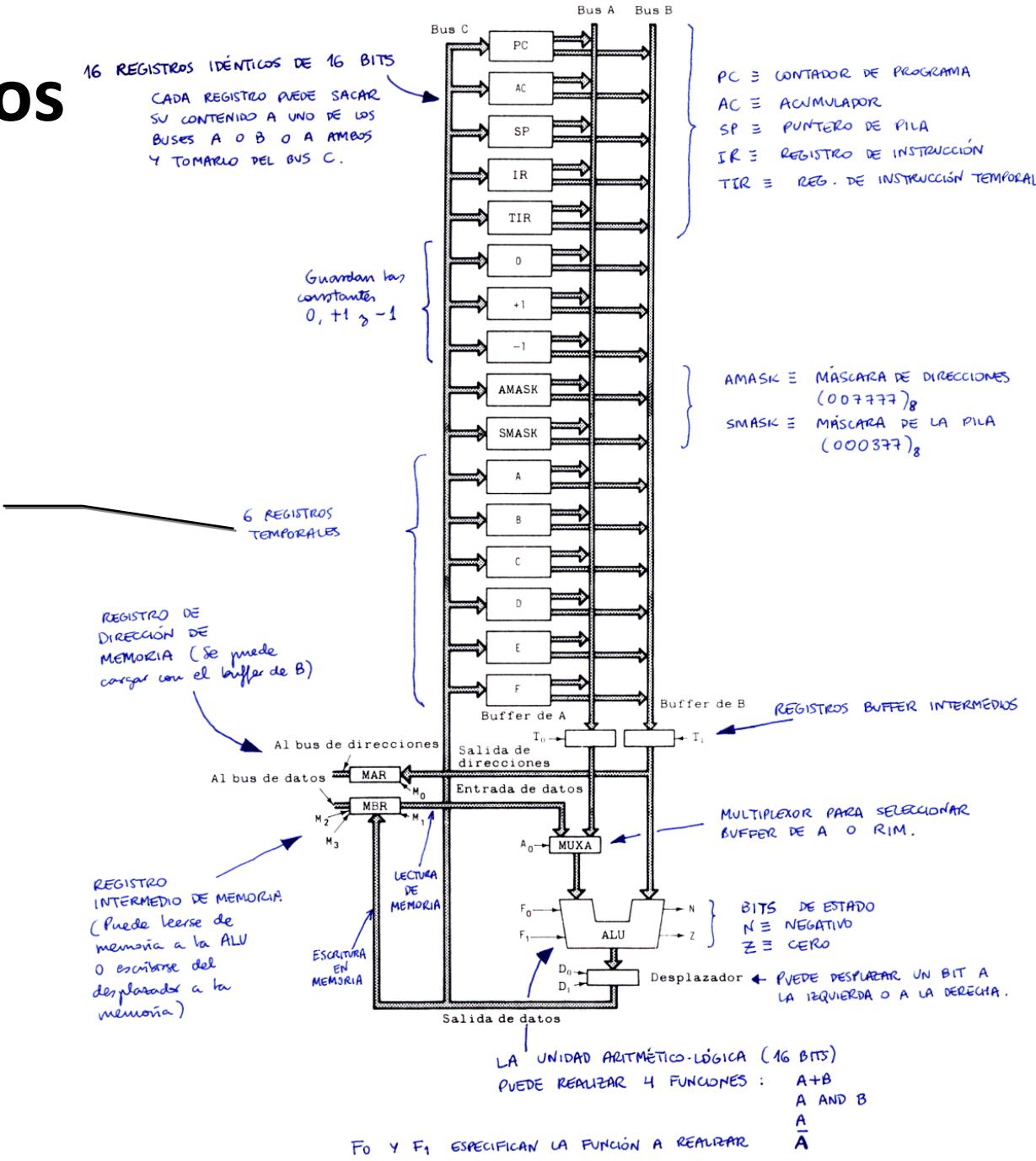
- El concepto del ej. anterior puede generalizarse para la especificación de otros elementos, por ej.:
    - registros destino
    - operaciones de memoria
    - direcciones o condiciones de salto
  - Se reduce el número de  $\mu$ instrucciones.
- 
- **En caso de que parte de la información de control no varíe durante muchas  $\mu$ instrucciones.**
    - En lugar de mantener la misma información en  $\mu$ instrucciones sucesivas, ésta se coloca en el registro de control residual durante un período deseado de tiempo. En este caso el registro de control residual se suele denominar registro de setup (configuración).
    - Se reduce la anchura de la  $\mu$ instrucciones.

# Ej. de arquitectura microprogramada

- Bibliografía: Tanenbaum: “Organización de computadoras: un enfoque estructurado”
  
- **Camino de datos**
- **Diseño horizontal**
  - Repertorio de instrucciones máquina
- **Diseño vertical**

# Camino de datos

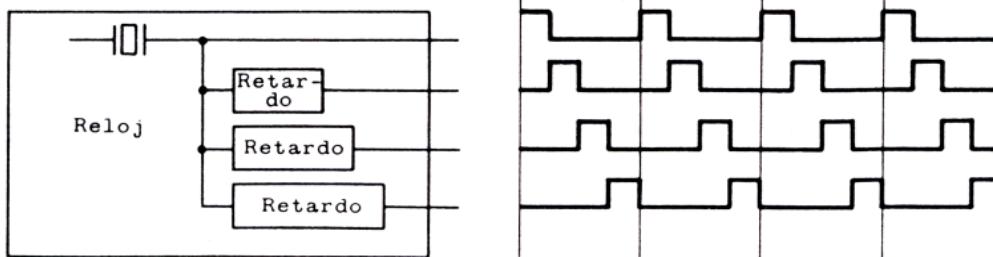
## Unidad de proceso o camino de datos ("datapath")



# Camino de datos

## ■ Temporización:

- Un ciclo básico de la UC consiste en una secuencia de 4 subciclos controlada por un reloj de 4 fases:

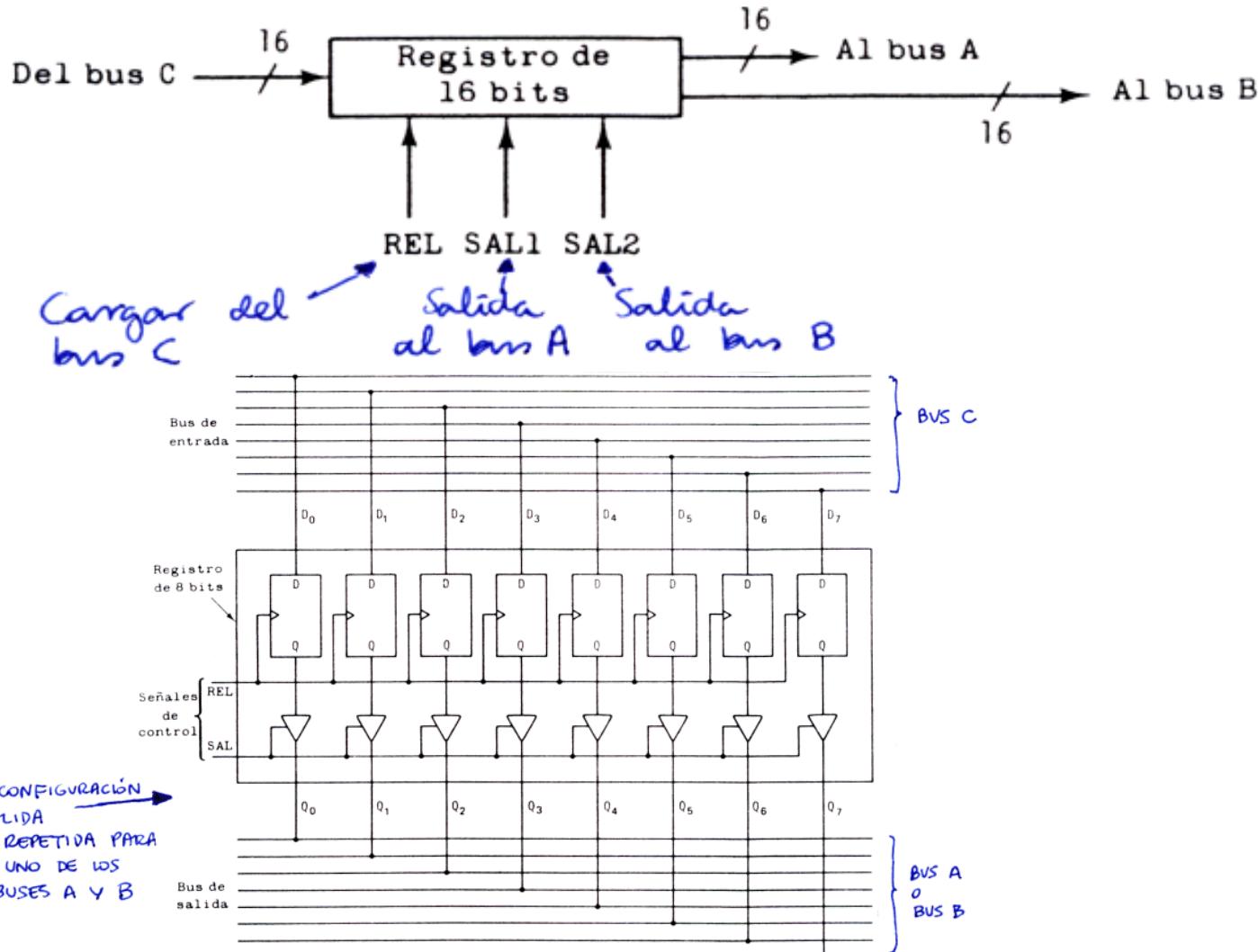


- Subciclos:

- Se lee de la memoria de control la siguiente microinstrucción a ejecutar y se carga en el registro de microinstrucción.
- Los contenidos de uno o dos registros pasan a los buses A y B y se cargan en los buffers.
- Las entradas de la ALU están estabilizadas. Se da tiempo a la ALU y al desplazador a que produzcan una salida estable y se carga MAR si es necesario.
- La salida del desplazador está estabilizada. Se almacena el bus C en un registro y en MBR si es necesario.

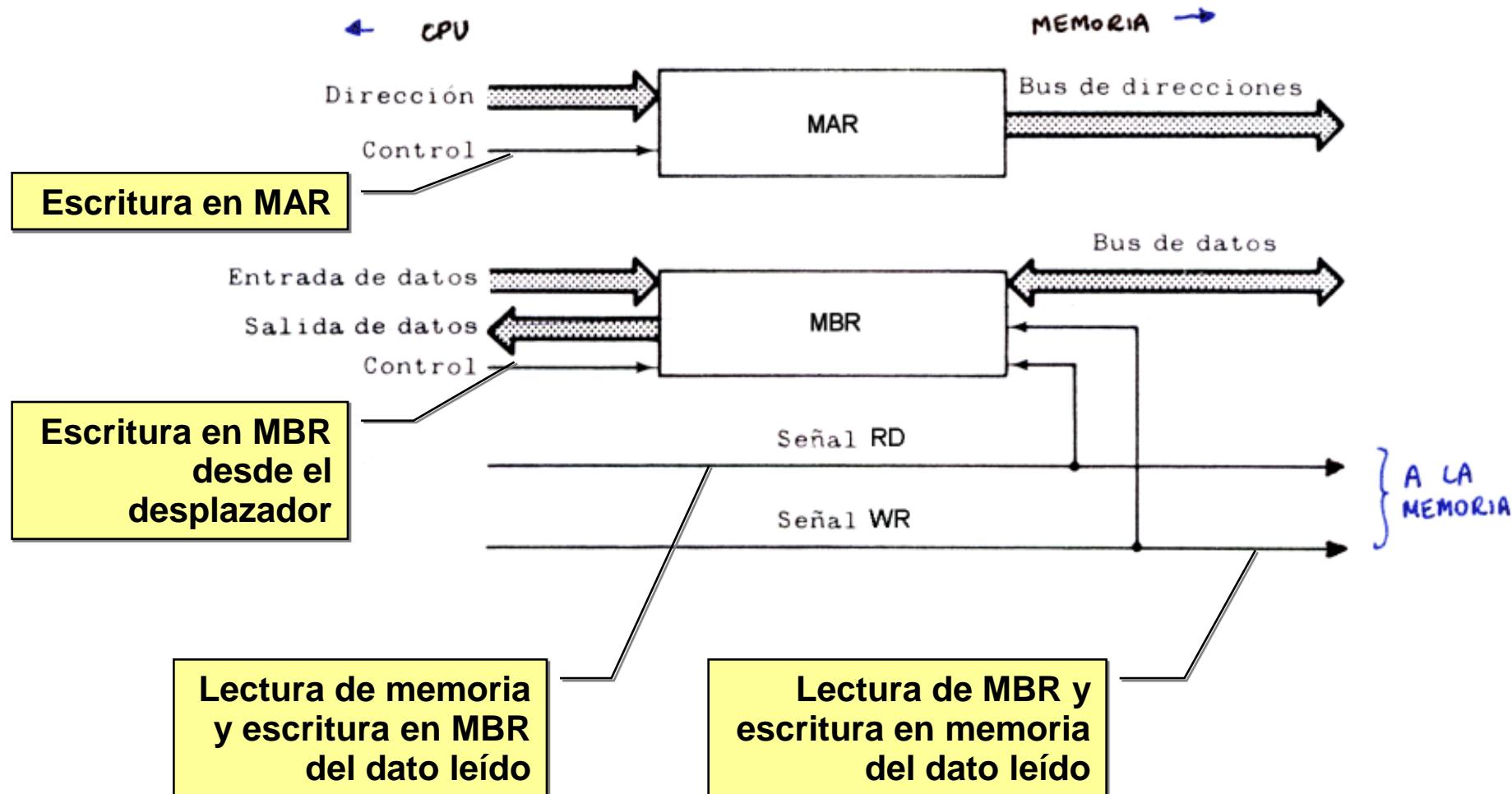
# Camino de datos

## ■ Detalle de uno de los registros de 16 bits:



# Camino de datos

## ■ Detalle de señales y buses de MAR y MBR:



# Diseño horizontal

## ■ Diseño horizontal de la unidad de control

- Señales de control necesarias:
  - 16 para carga del bus A desde los registros
  - 16 para carga del bus B desde los registros
  - 16 para carga de los registros desde el bus C
  - 2 para carga de los buffers de A y B
  - 2 para controlar la función de la ALU
  - 2 para controlar el desplazador
  - 4 para controlar MAR y MBR
  - 2 para indicar lectura o escritura en memoria
  - 1 para controlar el multiplexor MUXA
- 61 señales en total

# Diseño horizontal

## ■ Podemos reducir las señales necesarias:

- Codificando las señales de los registros (suponiendo que el contenido del bus C sólo se almacene en un registro)
  - 48 bits → 12 bits
- Eliminando los 2 bits para carga de los buffers de A y B (siempre se cargan en el mismo momento del ciclo máquina, por lo que puede activarlos el segundo subciclo del reloj)
  - 2 bits → 0 bits
- Reduciendo el nº de señales que controlan MAR y MBR (LEC puede usarse para cargar MBR desde la memoria y ESC para darle salida)
  - 4 bits → 2 bits

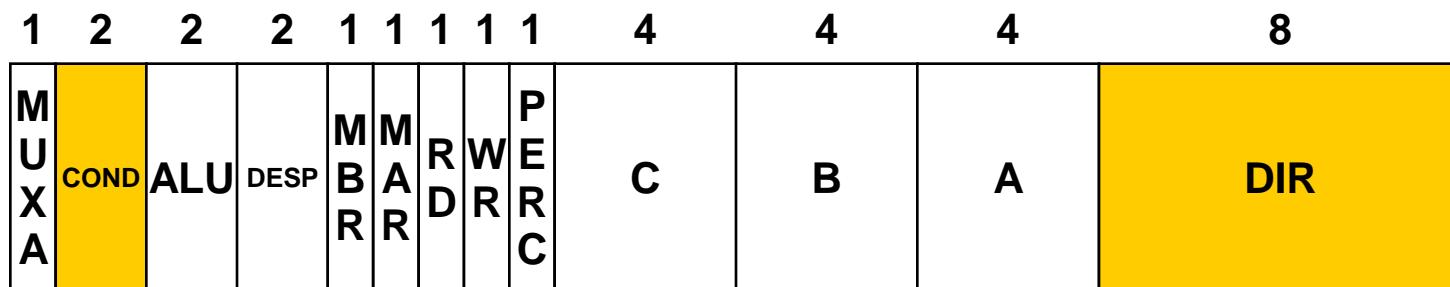
# Diseño horizontal

- **Es necesario añadir otras señales:**
  - Nos puede interesar generar N y Z sin almacenar el resultado, o almacenarlo sólo en MBR. Necesitamos un bit adicional PERC (permiso de C) para que se almacene el bus C en un registro (PERC = 1) o no (PERC = 0).
- **Nos quedan 22 bits de control.**

# Diseño horizontal

## ■ Formato de microinstrucción (32 bits):

- 22 bits de control
- 2 bits de condición de salto
- 8 bits de dirección



### MUXA

0 = Buffer de A  
1 = MBR

### COND

0 = No salta  
1 = Salta si N = 1  
2 = Salta si Z = 1  
3 = Salta siempre

### ALU

0 = A + B  
1 = A & B  
2 = A  
3 = No A

### DESP

0 = No desplaza  
1 = Desplaza 1 bit a la dcha.  
2 = Desplaza 1 bit a la izda.  
3 = (no utilizado)

### A, B, C

Selección de uno de los 16 registros

### DIR

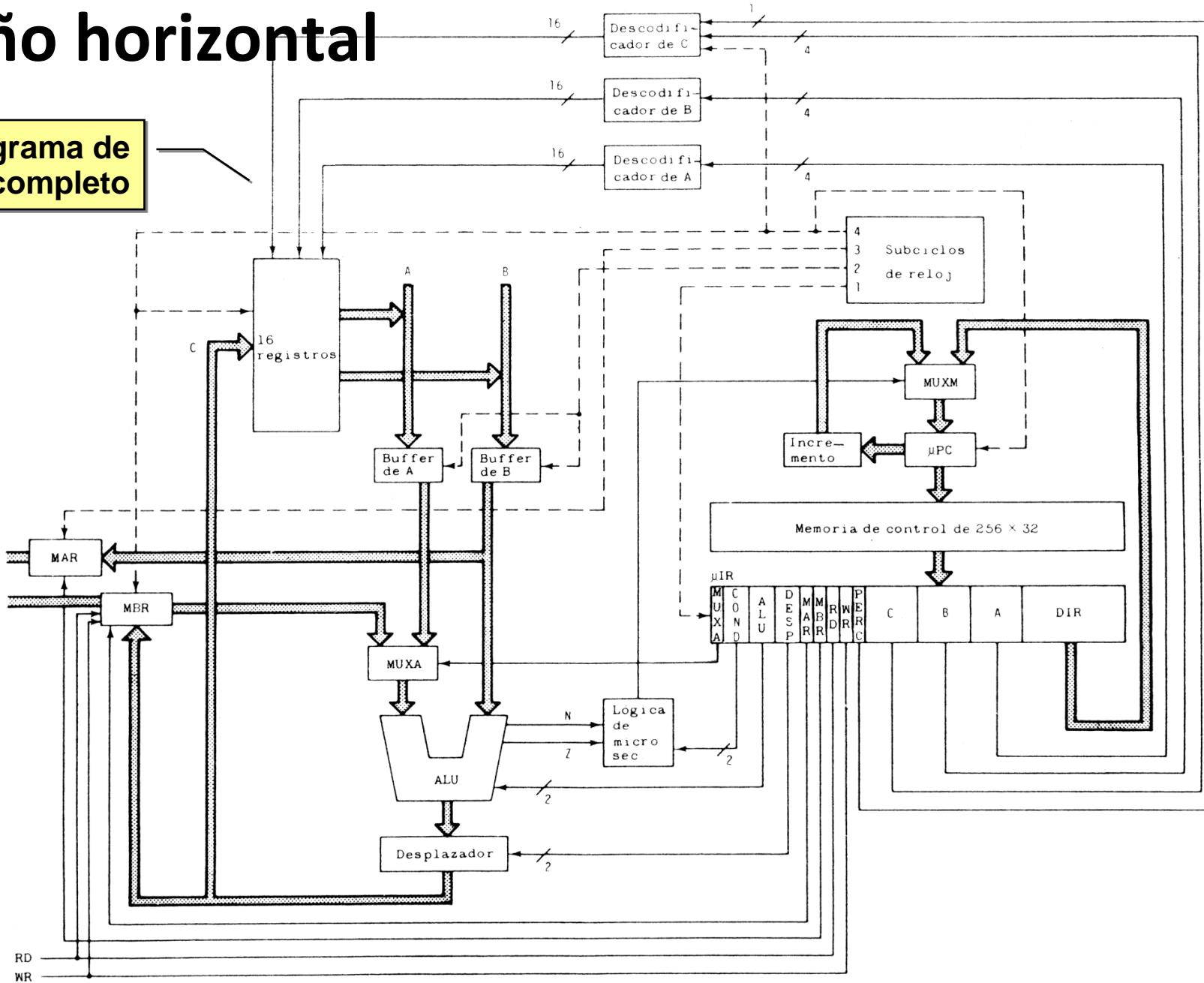
Dirección de salto en la memoria de control

### MBR, MAR, RD, WR, PERC

0 = No  
1 = Sí

# Diseño horizontal

**Diagrama de bloques completo**



# Diseño horizontal

## ■ Unidad de control

- Memoria de control: 256 palabras de 32 bits = 8192 bits
- La unidad de incremento calcula  $\mu\text{PC} + 1$
- Un ciclo de memoria principal dura dos microinstrucciones.
  - Las dos señales que controlan la memoria, RD y WR están activas mientras estén presentes en el  $\mu\text{IR}$ .
  - Si una microinstrucción comienza una lectura de memoria poniendo RD = 1, también debe ser RD = 1 en la siguiente microinstrucción.
- La elección de la siguiente microinstrucción la realiza la lógica de microsecuenciamiento durante el subciclo 4 (cuando N y Z son válidos), a partir de N, Z y los dos bits COND ( $I = \text{Izdo.}, D = \text{dcho.}$ ):
  - $\text{MUXM} = /I \cdot D \cdot N + I \cdot /D \cdot Z + I \cdot D = D \cdot N + I \cdot Z + I \cdot D$

# Diseño horizontal

## ■ Arquitectura (1)

- Memoria principal: 4096 palabras de 16 bits
- 3 registros visibles por el programador de leng. máquina:
  - PC : contador de programa
  - SP : puntero de pila
  - AC : acumulador
- 3 modos de direccionamiento:
  - Directo: los 12 bits menos significativos son una dirección de memoria
  - Indirecto: AC contiene una dirección de memoria
  - Local
    - los 12 bits menos significativos son un desplazamiento que se suma al puntero de pila
    - para direccionar variables locales de procedimientos

# Diseño horizontal

## ■ Arquitectura (2)

- La **pila** crece hacia direcciones de memoria menores
- La **E/S** es mapeada en memoria → no hay instrucciones de E/S específicas
- El registro **AMASK** ( $0FFF_{16}$ ) se utiliza para obtener el campo dirección en las instrucciones de direccionamiento directo y local
- El registro **SMASK** ( $00FF_{16}$ ) se utiliza para obtener el incremento/decremento del puntero de pila en las instrucciones INSP y DESP

# Diseño horizontal

## ■ Arquitectura (3): repertorio de 23 instrucciones máquina

Binario	Nemotécnico	Instrucción	Significado
0000xxxxxxxxxxxx	LODD	Carga directa	$ac := m[x]$
0001xxxxxxxxxxxx	STOD	Almacenamiento directo	$m[x] := ac$
0010xxxxxxxxxxxx	ADDD	Suma directa	$ac := ac + m[x]$
0011xxxxxxxxxxxx	SUBD	Resta directa	$ac := ac - m[x]$
0100xxxxxxxxxxxx	JPOS	Salto si positivo	if $ac \geq 0$ then $pc := x$
0101xxxxxxxxxxxx	JZER	Salto si cero	if $ac = 0$ then $pc := x$
0110xxxxxxxxxxxx	JUMP	Salto incondicional	$pc := x$
0111xxxxxxxxxxxx	LOCO	Carga de constante	$ac := x$ ( $0 \leq x \leq 4095$ )
1000xxxxxxxxxxxx	LODL	Carga local	$ac := m[sp + x]$
1001xxxxxxxxxxxx	STOL	Almacenamiento local	$m[sp + x] := ac$
1010xxxxxxxxxxxx	ADDL	Suma local	$ac := ac + m[sp + x]$
1011xxxxxxxxxxxx	SUBL	Resta local	$ac := ac - m[sp + x]$
1100xxxxxxxxxxxx	JNEG	Salto si negativo	if $ac < 0$ then $pc := x$
1101xxxxxxxxxxxx	JNZE	Salto si no cero	if $ac \neq 0$ then $pc := x$
1110xxxxxxxxxxxx	CALL	Llamada a subrutina	$sp := sp - 1$ ; $m[sp] := pc$ ; $pc := x$
1111000000000000	PSHI	Apilamiento indirecto	$sp := sp - 1$ ; $m[sp] := m[ac]$
1111001000000000	POPI	Desapilamiento indirecto	$m[ac] := m[sp]$ ; $sp := sp + 1$
1111010000000000	PUSH	Apilamiento	$sp := sp - 1$ ; $m[sp] := ac$
1111011000000000	POP	Desapilamiento	$ac := m[sp]$ ; $sp := sp + 1$
1111100000000000	RETN	Retorno de subrutina	$pc := m[sp]$ ; $sp := sp + 1$
1111101000000000	SWAP	Intercambio de AC y SP	$tmp := ac$ ; $ac := sp$ ; $sp := tmp$
1111110yyyyyyy	INSP	Incremento de SP	$sp := sp + y$ ( $0 \leq y \leq 255$ )
11111110yyyyyyy	DESP	Decremento de SP	$sp := sp - y$ ( $0 \leq y \leq 255$ )

# Diseño horizontal

## ■ Lenguaje para microprogramar en alto nivel:

- Los microprogramas se pueden escribir:
  - en binario: 32 bits por microinstrucción
  - nombrando cada campo distinto de 0 y su valor (una microinstrucción por línea):
    - Ej.: PERC = 1, C = 1, B = 1, A = 10
  - con instrucciones de alto nivel tipo PASCAL

- Funciones de la ALU:

`ac:=a+ac;`

`a:=band(ir,smask);`

`ac:=a;`

`a:=inv(a);`

- Desplazamiento:

`tir:=lshift(tir+tir);`

`a:=rshift (a);`

- Saltos incondicionales:

`goto 27`

- Saltos condicionales:

`if n then goto 27;`

`if z then goto 27;`

- Examen de un registro sin almacenamiento:

`alu:=tir; if n then goto 27;`

# Diseño horizontal

- Algunas sentencias y sus microinstrucciones correspondientes:

M	C	D		P		D
U	O	A	E	M	M	E
X	N	L	S	B	A	R
A	D	U	P	R	R	D
						I
				C	C	B
				B	A	R

mar:=pc; rd;	0	0	2	0	0	1	1	0	0	0	0	0	0	00
rd;	0	0	2	0	0	0	1	0	0	0	0	0	0	00
ir:=mbr;	1	0	2	0	0	0	0	0	1	3	0	0	0	00
pc:=pc+1	0	0	0	0	0	0	0	0	1	0	6	0	0	00
mar:=ir; mbr:=ac; wr;	0	0	2	0	1	1	0	1	0	0	3	1	0	00
alu:=tir; if n then goto 15;	0	1	2	0	0	0	0	0	0	0	0	4	15	
ac:=inv(mbr);	1	0	3	0	0	0	0	0	1	1	0	0	0	00
tir:=lshift(tir); if n then goto 25;	0	1	2	2	0	0	0	0	1	4	0	4	25	
alu:=ac; if z then goto 22;	0	2	2	0	0	0	0	0	0	0	0	1	22	
ac:=band(ir,amask); goto 0;	0	3	1	0	0	0	0	0	1	1	8	3	00	
sp:=sp+(-1); rd;	0	0	0	0	0	0	1	0	1	2	2	7	00	
tir:=lshift(ir+ir);if n then goto 69;	0	1	0	2	0	0	0	0	1	4	3	3	69	

# Diseño horizontal

## ■ Microprograma:

- 79 microinstrucciones de 32 bits = **2528** bits  
(diapositiva siguiente)
- La decodificación de las instrucciones se realiza examinando IR bit a bit.
  - Proporción considerable de tiempo dedicada a la decodificación
  - El CPI disminuiría si se pudiera extraer el  $\mu$ PC a partir del código de operación de la instrucción máquina (**goto f(ir)**)

## ■ Ejercicio: ¿cómo se modificaría el microprograma suponiendo que existe el tipo de salto **goto f(ir)**?

0: mar := pc ; rd;	{ciclo principal}	40: tir := lshift(tir); if n then goto 46;	{110x o 111x?}
1: pc := pc + 1; rd;	{incrementa pc}	41: alu := tir; if n then goto 44;	{1100 o 1101?}
2: ir := mbr; if n then goto 28;	{salva y descodifica mbr}	42: alu := ac; if n then goto 22;	{1100 = JNEG}
3: tir := lshift(ir + ir); if n then goto 19;	{000x o 001x?}	43: goto 0;	
4: tir := lshift(tir); if n then goto 11;	{0000 o 0001?}	44: alu := ac; if z then goto 0;	{1101 = JNZE}
5: alu := tir; if n then goto 9;	{0000 = LODD}	45: pc := band(ir, amask); goto 0;	
6: mar := ir ; rd;	{0001 = STOD}	46: tir := lshift(tir); if n then goto 50;	
7: rd;	{0010 o 0011?}	47: sp := sp + (-1);	{1110 = CALL}
8: ac := mbr; goto 0;	{0010 = ADDD}	48: mar := sp; mbr := pc ; wr;	
9: mar := ir ; mbr := ac ; wr;	{0011 = SUBD}	49: pc := band(ir , amask ); wr ; goto 0;	
10: wr ; goto 0;	{Nota: x-y = x + 1 + no y}	50: tir := lshift(tir); if n then goto 65;	{1111, examina addr}
11: alu := tir; if n then goto 15;	{010x o 011x?}	51: tir := lshift(tir); if n then goto 59;	
12: mar := ir ; rd;	{0100 o 0101?}	52: alu := tir; if n then goto 56;	
13: rd;	{0100 = JPOS}	53: mar := ac ; rd;	{1111000 = PSHI}
14: ac := mbr + ac ; goto 0;	{realiza el salto}	54: sp := sp + (-1); rd;	
15: mar := ir ; rd;	{0101 = JZER}	55: mar := sp ; wr ; goto 10;	
16: ac := ac + 1; rd;	{no se produce el salto}	56: mar := sp ; sp := sp + 1; rd;	{1111001 = POPI}
17: a := inv(mbr);	{0110 o 0111?}	57: rd;	
18: ac := ac + a; goto 0;	{0110 = JUMP}	58: mar := ac ; wr ; goto 10;	
19: tir := lshift(tir); if n then goto 25;	{0111 = LOCO}	59: alu := tir; if n then goto 62;	
20: alu := tir; if n then goto 23;	{10xx o 11xx?}	60: sp := sp + (-1);	{1111010 = PUSH}
21: alu := ac ; if n then goto 0;	{100x o 101x?}	61: mar := sp ; mbr := ac ; wr ; goto 10;	
22: pc := band(ir , amask ); goto 0;	{1000 o 1001?}	62: mar := sp ; sp := sp + 1; rd;	{1111011 = POP}
23: alu := ac ; if z then goto 22;	{1000 = LODL}	63: rd;	
24: goto 0;	{1001 = STOL}	64: ac := mbr ; goto 0;	
25: alu := tir ; if n then goto 27;	{010 o 1011?}	65: tir := lshift(tir); if n then goto 73;	
26: pc := band(ir , amask ); goto 0;	{1010 = ADDL}	66: alu := tir ; if n then goto 70;	
27: ac := band(ir , amask ); goto 0;	{1011 = SUBL}	67: mar := sp ; sp := sp + 1; rd;	{1111100 = RETN}
28: tir := lshift(ir + ir); if n then goto 40;		68: rd;	
29: tir := lshift(tir); if n then goto 35;		69: pc := mbr ; goto 0;	
30: alu := tir ; if n then goto 33;		70: a := ac ;	
31: a := ir + sp ;		71: ac := sp ;	
32: mar := a ; rd ; goto 7;		72: sp := a ; goto 0;	
33: a := ir + sp ;		73: alu := tir ; if n then goto 76;	
34: mar := a ; mbr := ac ; wr ; goto 10;		74: a := band(ir , smask );	{1111110 = INSP}
35: alu := tir ; if n then goto 38;		75: sp := sp + a ; goto 0;	
36: a := ir + sp ;		76: a := band(ir , smask );	{1111111 = DESP}
37: mar := a ; rd ; goto 13;		77: a := inv(a);	
38: a := ir + sp ;		78: a := a + 1; goto 35	
39: mar := a ; rd ; goto 16;			

# Diseño vertical

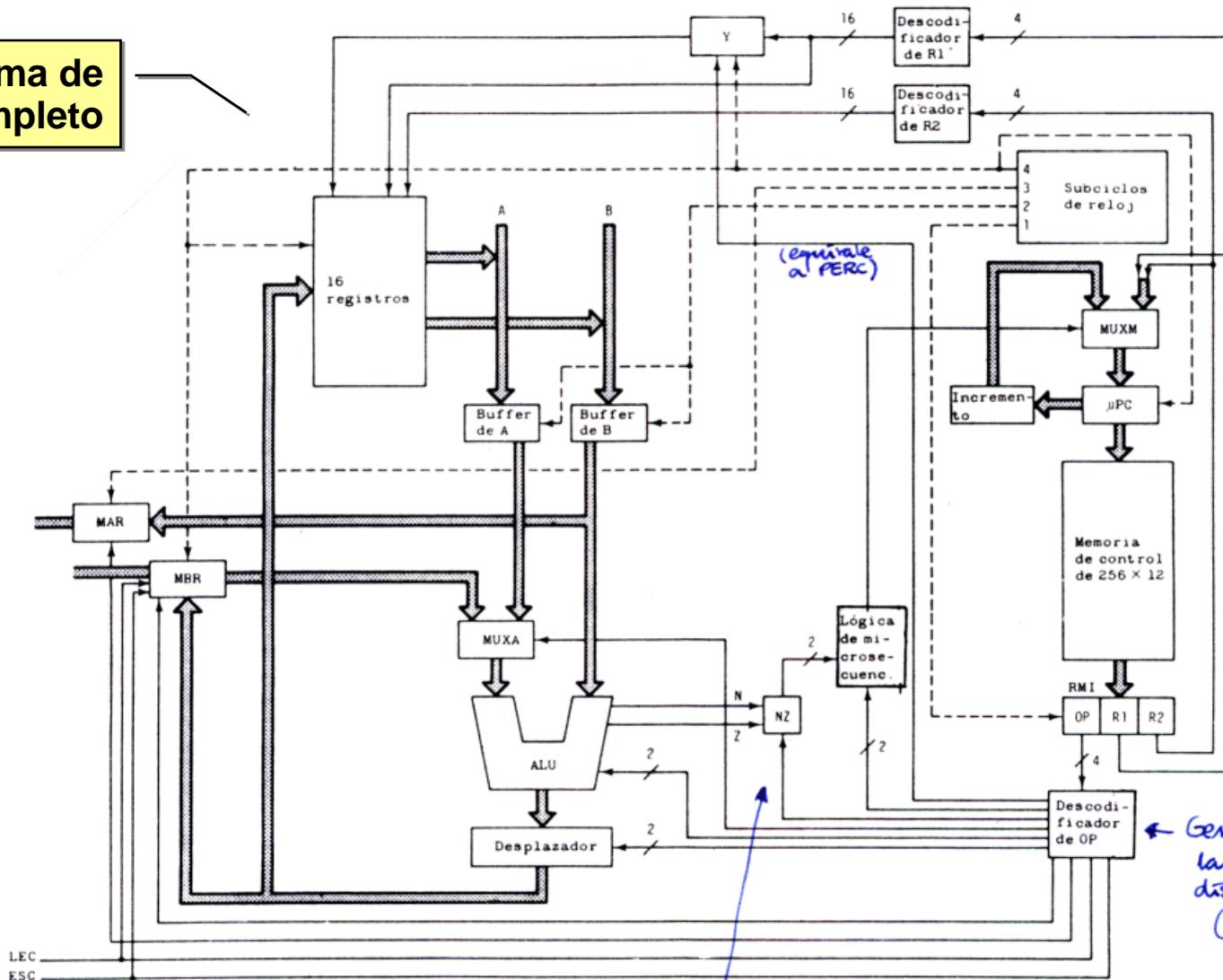
- Cada microinstrucción contiene ahora 3 campos de 4 bits:
  - OP: código de operación (dice qué hace la microinstrucción)
  - r1 y r2: dos registros (dirección en el caso de los saltos)

Binario	Nemotécnico	Instrucción	Significado
0000	ADD	Suma	$r1 := r1 + r2$
0001	AND	AND bit a bit	$r1 := r1 \& r2$
0010	MOV	Mueve reg. a reg.	$r1 := r2$
0011	NEG	Complementa	$r1 := \text{inv}(r2)$
0100	SHL	Desplaza a la izda.	$r1 := \text{lshift}(r2)$
0101	SHR	Desplaza a la dcha.	$r1 := \text{rshift}(r2)$
0110	MOV_MBR	Mueve MBR a registro	$r1 := \text{MBR}$
0111	TST	Examina registro	if $r2 < 0$ then $n := \text{true}$ ; if $r2 = 0$ then $z := \text{true}$
1000	LD_1	Comienza lectura	$\text{MAR} := r1; \text{RD}$
1001	ST_1	Comienza escritura	$\text{MAR} := r1; \text{MBR} := r2; \text{WR}$
1010	LD_2	Termina lectura	$\text{RD}$
1011	ST_2	Termina escritura	$\text{WR}$
1100	-	(no usado)	
1101	BN	Salta si $N = 1$	if $n$ then goto dir
1110	BZ	Salta si $Z = 1$	if $z$ then goto dir
1111	JMP	Salta siempre	goto dir

# Diseño vertical

Se necesita Y porque el campo R1 lleva el bits A y el C.

Diagrama de bloques completo



Registro de 2 bits para almacenar N y Z, ya que ahora el salto condicional requiere otra microinstrucción.

← Genera todas las señales del diseño horizontal  
(Ver siguiente figura)

# Diseño vertical

## ■ Señales generadas por el decodificador de OP:

- Requiere una PLA con 4 entradas, 13 salidas y 15 términos producto.

Binario Nemotécnico	ALU A	ALU B	DESP A	DESP B	NZ	MUXA	Y	MAR	MBR	RD	WR	LMS A	LMS B
0000	ADD				1		1						
0001	AND		1		1		1						
0010	MOV	1			1		1						
0011	NEG	1	1		1		1						
0100	SHL	1		1	1		1						
0101	SHR	1			1	1	1						
0110	MOV_MBR	1			1	1	1						
0111	TST	1			1								
1000	LD_1	1						1		1			
1001	ST_1	1						1	1		1		
1010	LD_2	1								1			
1011	ST_2	1									1		
1100	-												
1101	BN												1
1110	BZ											1	
1111	JMP											1	1

# Diseño vertical

## ■ Microprograma:

- 160 microinstrucciones de 12 bits = **1920** bits  
(diapositiva siguiente)
- Menos memoria de control, pero UC más lenta (¿por qué? ¿cuánto más lenta (suponer todas las instrucciones equiprobables)?)

## ■ Ejercicio: ¿cómo se modificaría el microprograma suponiendo que existe el tipo de salto **goto f(ir)**?

```

0: mar := pc ; rd;
1: rd;
pc := pc + 1;
2: ir := mbr;
tir := lshift(ir);
if n then goto 28;
3: tir := lshift(tir);
if n then goto 19;
4: tir := lshift(tir);
if n then goto 11;
5: alu := tir;
if n then goto 09;
6: mar := ir ; rd ; {LODD}
7: rd;
8: ac := mbr;
goto 0;
9: mar := ir ; mbr := ac ; wr ; {STOD}
10: wr;
goto 0;
11: alu := tir;
if n then goto 15;
12: mar := ir ; rd ; {ADDD}
13: rd;
14: a := mbr;
ac := ac + a;
goto 0;
15: mar := ir ; rd ; {SUBD}
16: rd;
99: ac := ac + 1;
17: a := mbr;
a := inv(a);
18: ac := ac + a;
goto 0;
19: tir := lshift(tir);
if n then goto 25;
20: alu := tir;
if n then goto 23;
21: alu := ac ; {JPOS}
if n then goto 0;
22: pc := ir ;
pc := band(pc , amask );
goto 0;
23: alu := ac ; {JZER}
if z then goto 22;
24: goto 0;
25: alu := tir ;
if n then goto 27;
26: pc := ir ; {JUMP}
pc := band(pc , amask );
goto 0;
27: ac := ir ; {LOCO}
ac := band(ac , amask );
goto 0;
28: tir := lshift(tir);
if n then goto 40;
29: tir := lshift(tir);
if n then goto 35;
30: alu := tir;
if n then goto 33;
31: a := ir ; {LODL}
a := a + sp;
32: mar := a ; rd ;
rd;
ac := mbr;
goto 0;
33: a := ir ; {STOL}
a := a + sp;
34: mar := a ; mbr := ac ; wr ;
wr;
goto 0;
35: alu := tir ;
if n then goto 38;
36: a := ir ; {ADDL}
a := a + sp;
37: mar := a ; rd ;
rd;
a := mbr;
ac := ac + a;
goto 0;
38: a := ir ; {SUBL}
a := a + sp ;
39: mar := a ; rd ;
rd;
goto 99;
40: tir := lshift(tir);
if n then goto 46;
41: alu := tir ;
if n then goto 44;
42: alu := ac ; {JNEG}
if n then goto 22;
43: goto 0;
44: alu := ac ; {JNZE}
if z then goto 0;
45: pc := ir ;
pc := band(pc , amask );
goto 0;
46: tir := lshift(tir);
if n then goto 50;
47: sp := sp + (-); {CALL}
48: mar := sp ; mbr := pc ; wr ;
wr;
49: pc := ir ;
pc := band(pc , amask );
goto 0;
50: tir := lshift(tir);
if n then goto 65;
51: tir := lshift(tir);
if n then goto 59;
52: alu := tir ;
if n then goto 56;
53: mar := ac ; rd ; {PSHI}
rd;
54: sp := sp + (-);
55: a := mbr ;
mar := sp ; mbr := a ; wr ;
wr;
goto 0;
56: mar := sp ; rd ; {POPI}
57: rd ;
sp := sp + 1;
58: a := mbr ;
mar := ac ; mbr := a ; wr ;
wr;
goto 0;
59: alu := tir ;
if n then goto 62;
60: sp := sp + (-); {PUSH}
61: mar := sp ; mbr := ac ; wr ;
wr;
goto 0;
62: mar := sp ; rd ; {POP}
63: rd ;
sp := sp + 1;
64: ac := mbr ;
goto 0;
65: tir := lshift(tir);
if n then goto 73;
66: alu := tir ;
if n then goto 70;
67: mar := sp ; rd ; {RETN}
68: rd ;
sp := sp + 1;
69: pc := mbr ;
goto 0;
70: a := ac ; {SWAP}
71: ac := sp ;
72: sp := a ;
goto 0;
73: alu := tir ;
if n then goto 76;
74: a := ir ; {INSP}
a := band(a , smask );
75: sp := sp + a ;
goto 0;
76: a := ir ; {DESP}
a := band(a , smask );
77: a := inv(a );
78: a := a + 1;
sp := sp + a ;
goto 0;

```