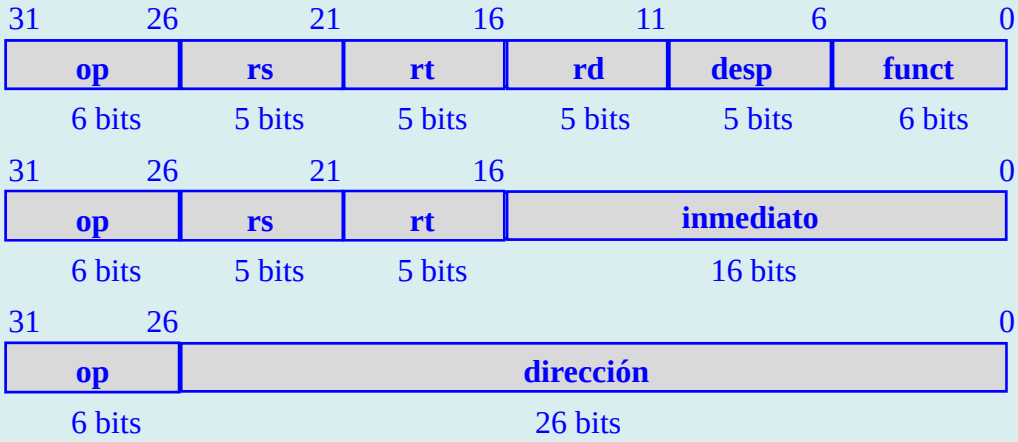
Arquitectura MIPS: Formato de la instrucción máquina

• La ruta de datos la diseñaremos para un subconjunto de instrucciones del procesador MIPS, que dispone de sólo 3 formatos de diferentes de longitud fija de 32 bits:

- **Tipo R:**
  - Aritmético-lógicas
- **Tipo I:**
  - Referencia a memoria
  - Aritméticas (inmediato)
  - Salto condicional
- **Tipo J:**
  - Salto incondicional



• El significado de los campos es el siguiente:

- **op**: código de operación (identificador de cada instrucción)
- **rs**, **rt**, **rd**: identificadores de los registros fuentes y destino de las operaciones
- **desp**: cantidad a desplazar en operaciones de desplazamiento
- **funct**: identificador de la operación aritmética a realizar
- **inmediato**: operando inmediato, o desplazamiento en direccionamiento base+desplazamiento
- **dirección**: dirección destino del salto

Arquitectura MIPS: Semántica de las instrucciones seleccionadas

- **Instrucciones de referencia a memoria** (formato tipo I):
  - lw rt, inmed(rs) rt ← Memoria(rs + SignExt(inmed)) , PC ← PC + 4
  - sw rt, inmed(rs) Memoria(rs + SignExt(inmed)) ← rt , PC ← PC + 4

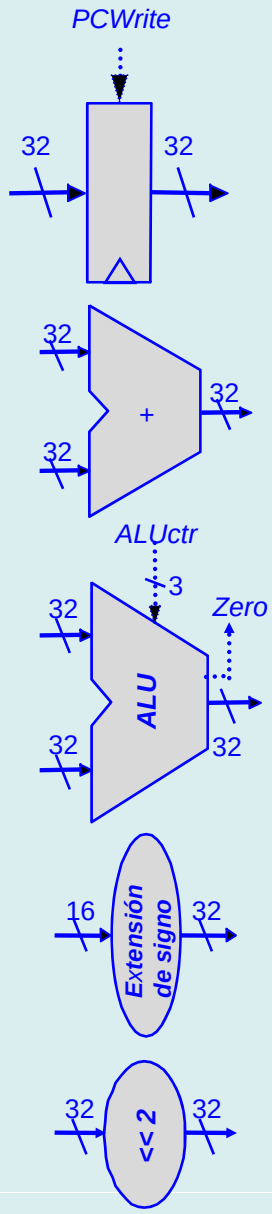
- **Instrucciones aritmético-lógicas con operandos en registros** (formato tipo R)
  - add rd, rs, rt rd ← rs + rt, PC ← PC + 4
  - sub rd, rs, rt rd ← rs - rt , PC ← PC + 4
  - and rd, rs, rt rd ← rs and rt , PC ← PC + 4
  - or rd, rs, rt rd ← rs or rt , PC ← PC + 4
  - slt rd, rs, rt (si (rs < rt) entonces (rd ← 1)
    en otro caso (rd ← 0)), PC ← PC+4

- **Instrucciones de salto condicional** (formato tipo I)
  - beq rs, rt, inmed si (rs = rt) entonces (PC ← PC + 4 + 4·SignExt(inmed))
    caso contrario PC ← PC + 4

Diseño de la ruta de datos monociclo (1)

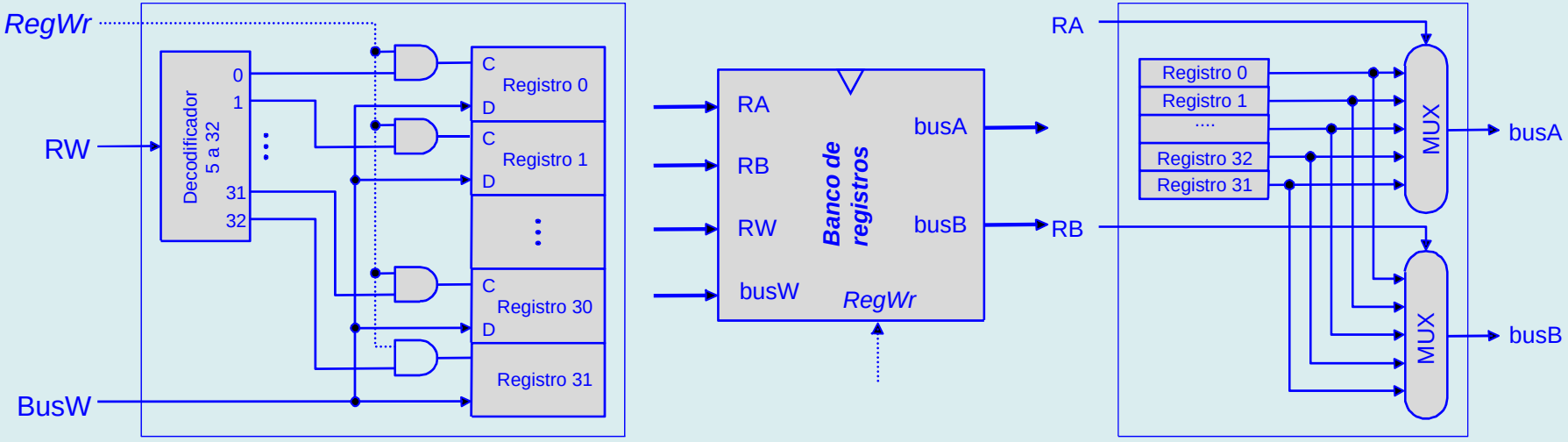
- **Componentes de la ruta de datos** (1):
  - Contador de programa
  - **2** Sumadores**:**
    - El primero para sumar 4 al PC
    - El segundo para sumar al PC el valor inmediato de salto.
  - ALU**:** capaz de realizar:
    - Suma
    - Resta
    - And
    - Or
    - Comparación de igualdad mediante resta
  - Extensor de signo:
  - Desplazador a la izquierda **(**para la multiplicar por 4):

ALUctr	*función*
000	A and B
001	A or B
010	A + B
110	A – B
111	1 si (A<B), sino 0



Diseño de la ruta de datos monociclo (2)

- **Componentes de la ruta de datos (2): Banco de registros**
  - Los 32 registros conforman un banco de registros. Dado que las instrucciones de tipo R requieren acceso simultáneo a 3 registros, el banco dispondrá de los siguientes elementos:
    - 2 salidas de datos de 32 bits (*busA y busB*)
    - 1 entradas de datos de 32 bits (*busW)*
    - 3 entradas de 5 bits para la identificación de los registros (*RA, RB y RW*)
    - 1 entrada de control para habilitar la escritura sobre uno de los registros (*RegWr*)
    - 1 reloj que sólo actúa durante las operaciones de escritura, las de lectura son combinacionales
  - La lógica para estas operaciones de lectura y escritura es la siguiente:

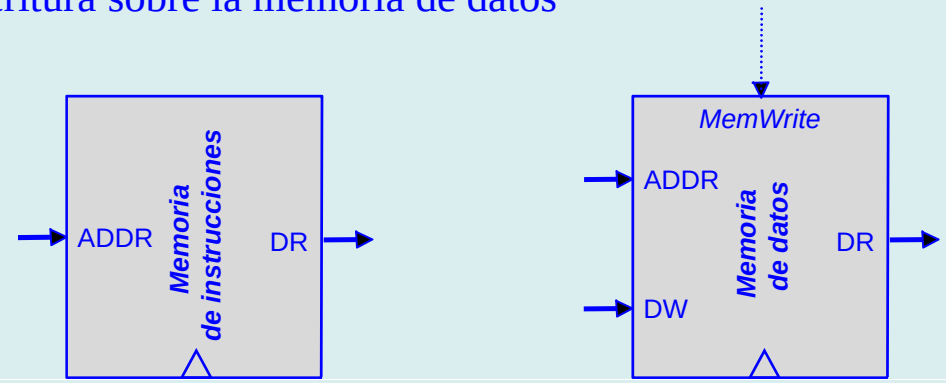


Mecanismo de escritura

Mecanismo de lectura

Diseño de la ruta de datos monociclo (3)

- **Componentes de la ruta de datos (3): Memoria**
  - Se supondrá dividida en dos para poder hacer dos accesos a memoria en el mismo ciclo:
    - Memoria de instrucciones
    - Memoria de datos
  - Será direccionable por bytes, pero capaz de aceptar/ofrecer 4 bytes por acceso
    - 1 entrada de dirección
    - 1 salida de datos de 32 bits
    - 1 entrada de datos de 32 bits (sólo en la de datos)
  - Se supondrá que se comporta temporalmente como el banco de registros (síncronamente) y que tiene un tiempo de acceso menor que el tiempo de ciclo
  - Existirá una entrada de control, *MemWrite* para seleccionar la operación de lectura/escritura sobre la memoria de datos

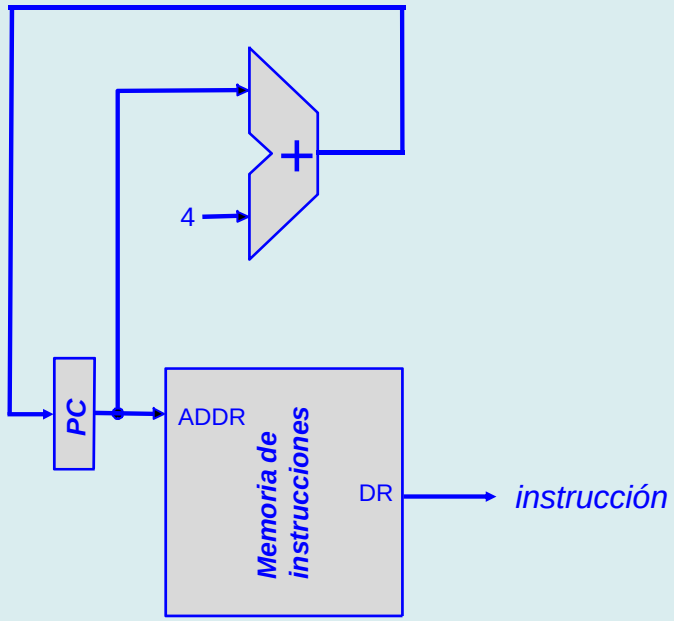


Diseño de la ruta de datos monociclo (4)

- **Interconexión de la ruta de datos (1)**
  - La **búsqueda de instrucciones** implica leer la instrucción ubicada en la dirección de la **memoria de instrucciones** indicada por el **contador de programa (*PC*)**.
  - La **ejecución secuencial** de programas implica actualizar el **contador de programa** para que apunte a la siguiente instrucción (sumando 4 por ser una memoria direccionable por bytes y una arquitectura con tamaño de palabra de 32 bits):

$$PC \leftarrow PC + 4$$

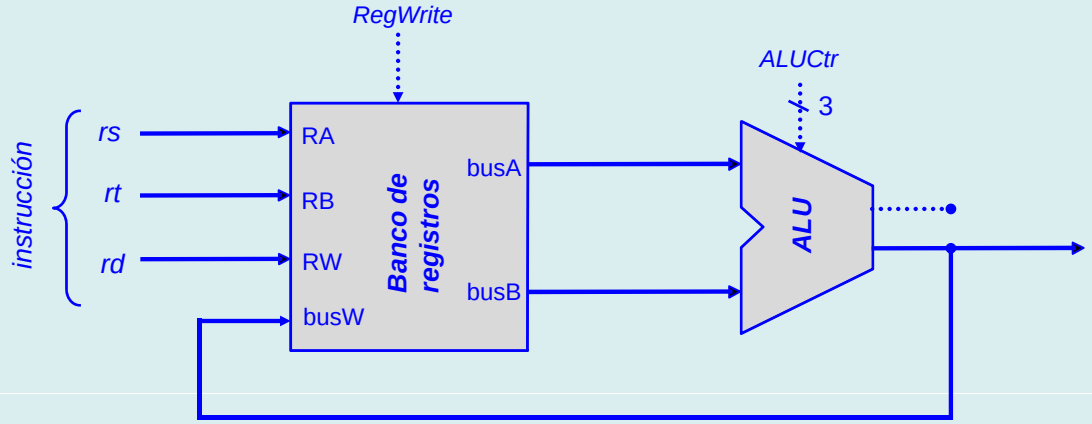
  - Luego la estructura necesaria para la búsqueda secuencial de instrucciones será:



Diseño de la ruta de datos monociclo (5)

- **Interconexión de la ruta de datos (2)**
  - Las instrucciones **aritmético lógicas** (tipo-R) implican operar sobre el banco de registros **BR** y la **ALU** de la siguiente manera:

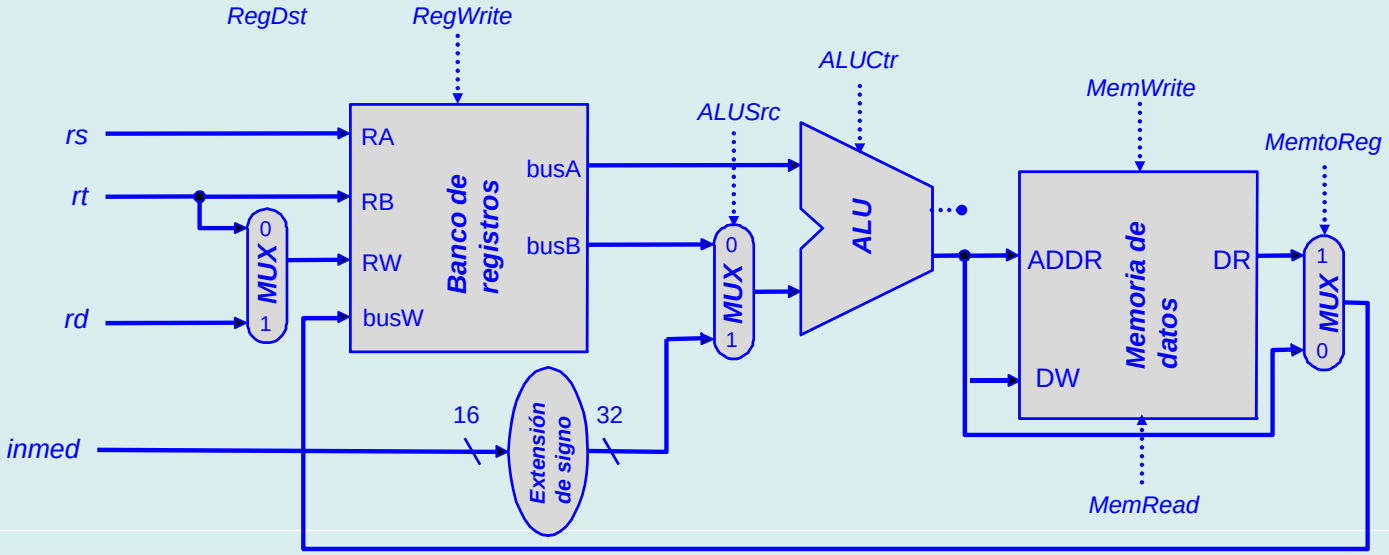
    BR(rd) ← BR(rs) funct BR(rt)
  - Por tanto será necesario:
    - Leer dos registros cuyos identificadores se ubican en los campos **rs** y **rt** de la instrucción
    - Operar sobre ellos según el contenido del campo de código de operación aritmética (**funct**)
    - Almacenar el resultado en otro registro cuyo identificador se localiza en el campo **rd** de la instrucción



Diseño de la ruta de datos monociclo (6)

- **Interconexión de la ruta de datos (3)**
  - La **instrucción de carga** (lw) requiere la siguiente operación de transferencia:

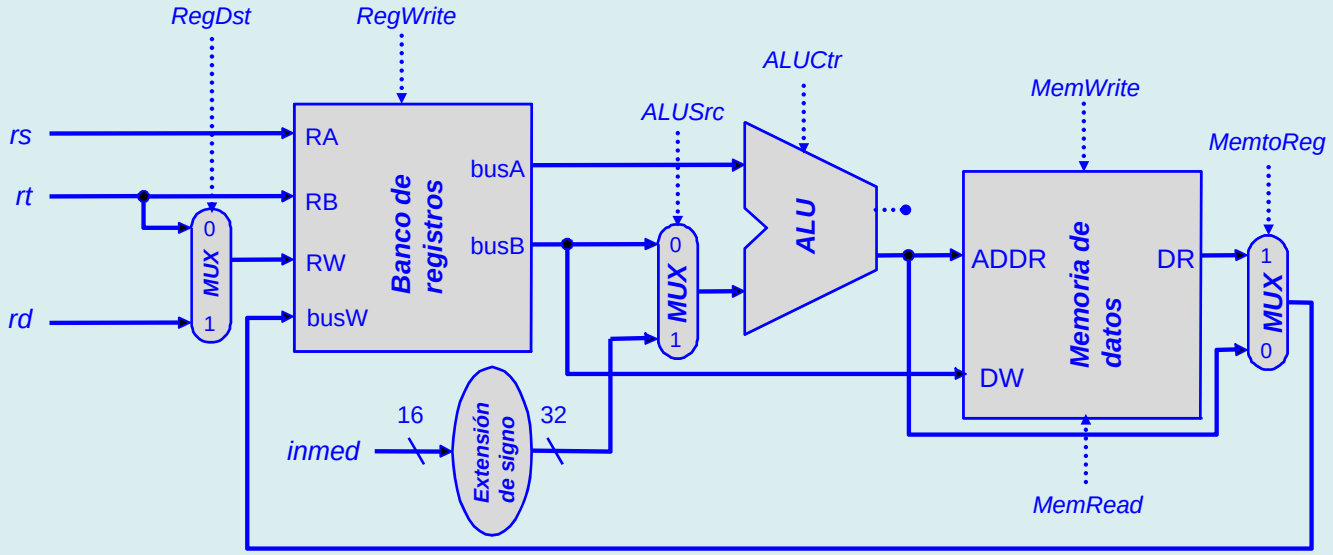
    BR(rt) ← Memoria(BR(rs) + SignExt(inmed))
  - Lo que implica las siguientes operaciones elementales:
    - Calcular la dirección efectiva de memoria:
      - Leyendo el *registro base* cuyo identificador se ubica en el campo **rs**
      - Un *desplazamiento* de 32 bits de la **extensión** del operando inmediato (**inmed**)
      - **Sumando** base y desplazamiento.
    - Leer el dato de la **memoria de datos** cuya dirección es la anteriormente calculada
    - Almacenar el dato leído en el registro cuyo especificado en el campo **rt**



Diseño de la ruta de datos monociclo (7)

- **Interconexión de la ruta de datos (4)**
  - La **instrucción de almacenamiento** (sw) requiere la siguiente operación:

    Memoria(BR(rs) + SignExt(inmed)) ← BR(rt)
  - Lo que implica las siguientes operaciones elementales:
    - Leer el dato almacenado en el registro cuyo identificador se especifica en **rt**
    - Calcular la dirección efectiva de memoria:
      - Leyendo el *registro base* cuyo identificador se ubica en el campo **rs**
      - Un *desplazamiento* de 32 bits a partir de la **extensión** del campo **inmed**
      - **Sumando** base y desplazamiento.
    - Almacenar el dato leído de la **memoria de datos** en la dirección calculada

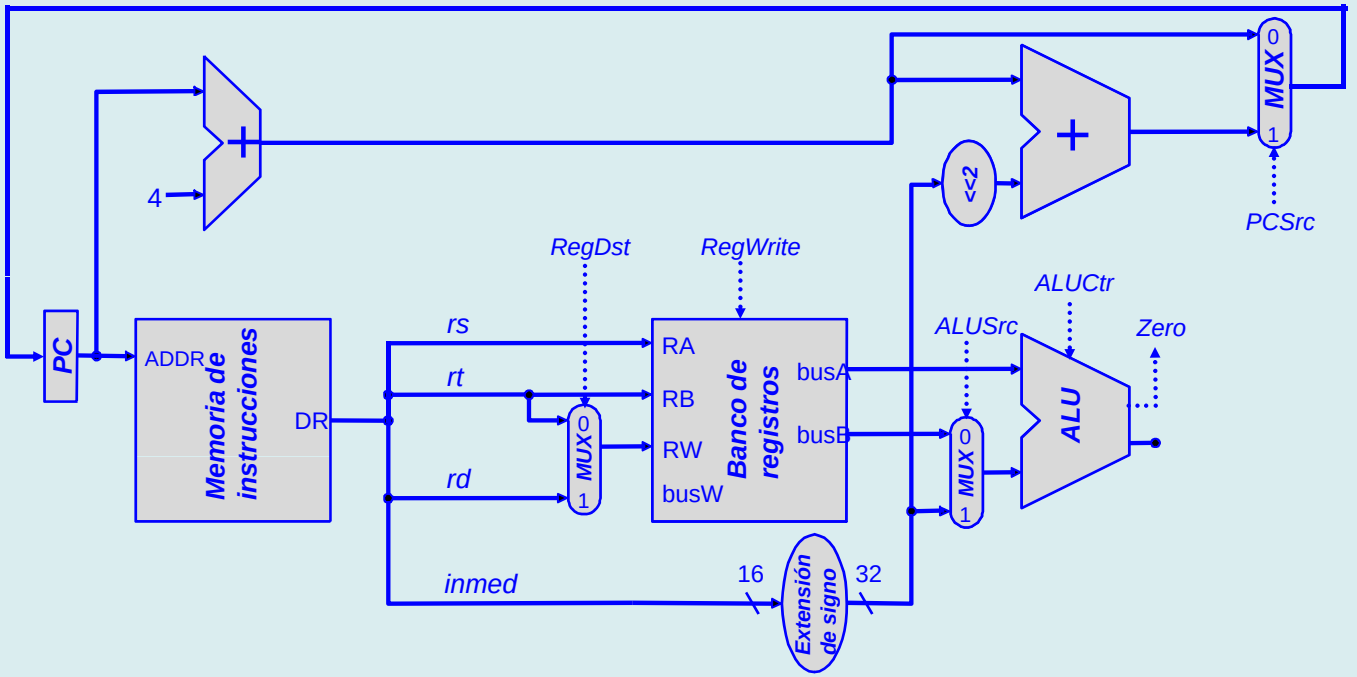


Diseño de la ruta de datos monociclo (8)

- **Interconexión de la ruta de datos (5)**
  - La **instrucción de salto condicional** (beq) requiere la siguiente operación:

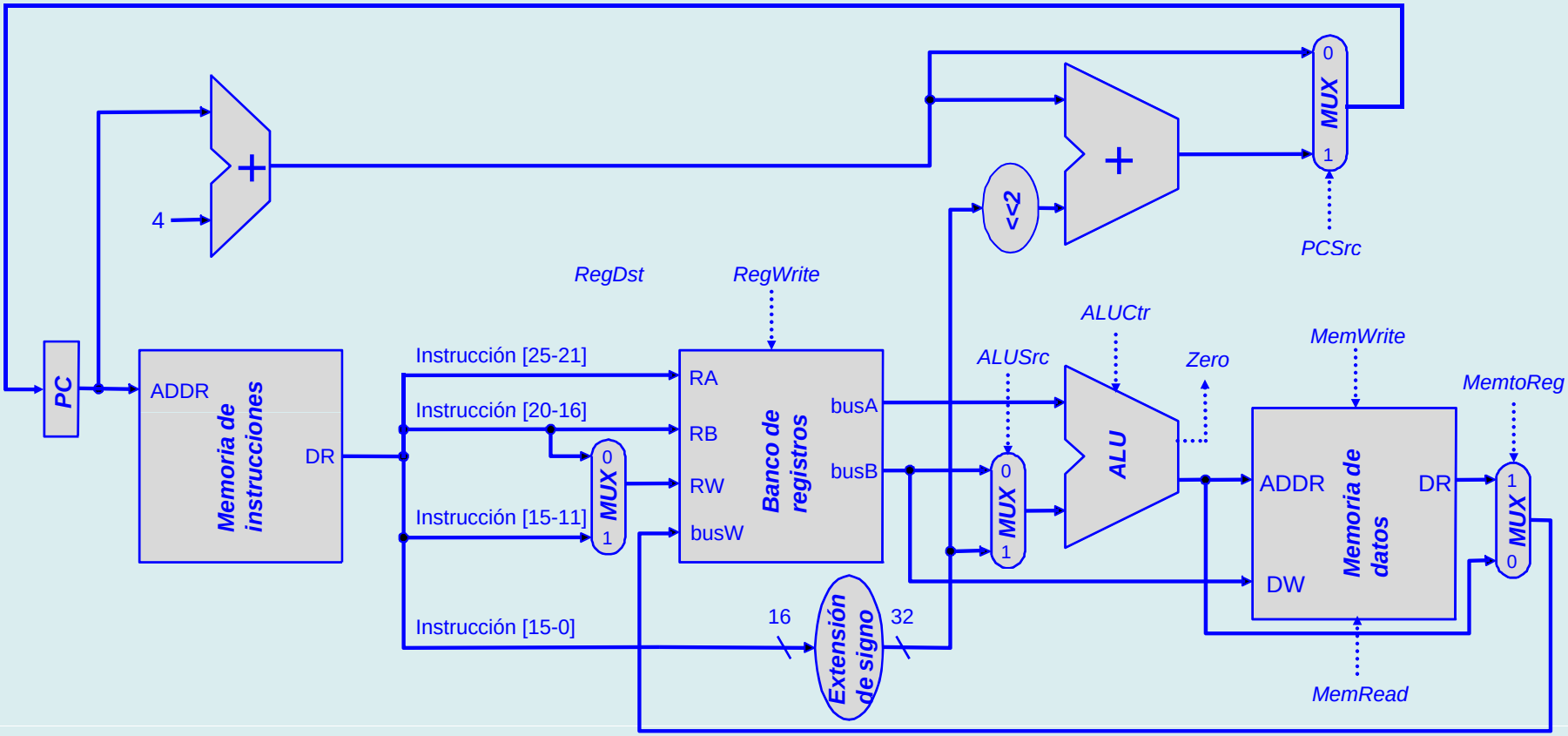
    Si (BR(rs) = BR(rt)) entonces (PC ← PC + 4·SignExt(inmed))
  - Lo que implica las siguientes operaciones elementales:
    - Leer dos registros cuyos identificadores se ubican en los campos **rs** y **rt**
    - Comparar la igualdad de sus contenidos y en función del resultado:

      - **Sumar** al *PC* un *desplazamiento* de 32 bits obtenido a partir de la **extensión** del campo de operando inmediato (**inmed**) o
      - No hacer nada



Diseño de la ruta de datos monociclo (9)

- **Ruta de datos monociclo**
  - La ejecución monociclo ha obligado a:
    - No usar más de una vez por instrucción cada recurso. Duplicarlo si es necesario
    - Memoria de instrucciones y datos separadas
    - Añadir multiplexores cuando un valor pueda provenir de varias fuentes



Diseño del controlador de la ruta de datos monociclo (1)

- La tarea del **controlador** es:
  - Seleccionar las operaciones a realizar por los módulos multifunción (ALU, etc.)
  - Controlar el flujo de datos, activando la entrada de selección de los multiplexores y la señal de carga de los registros
- Para ello hay que determinar los valores de los puntos de control para cada instrucción:

Instrucción de carga (lw)

rt ← Memoria(rs + SignExt(inmed)), PC ← PC + 4

RegDest ← 0, RegWrite ← 1, ALUsrc ← 1, ALUctr ← 010, PCSrc ← 0, MemWrite ← 0, MemRead ← 1, MemtoReg ← 1

Instrucción de almacenamiento (sw)

Memoria(rs + SignExt(inmed)) ← rs, PC ← PC + 4

RegDest ←X, RegWrite ← 0, ALUsrc ← 1, ALUctr ← 010, PCSrc ← 0, MemWrite ← 1, MemRead ← 0, MemtoReg ← X

Instrucción *and*

rd ← rs and rt, PC ← PC + 4

RegDest ← 1, RegWrite ← 1, ALUsrc ← 0, ALUctr ← 000, PCSrc ← 0, MemWrite ← 0, MemRead ← 0, MemtoReg ← 0

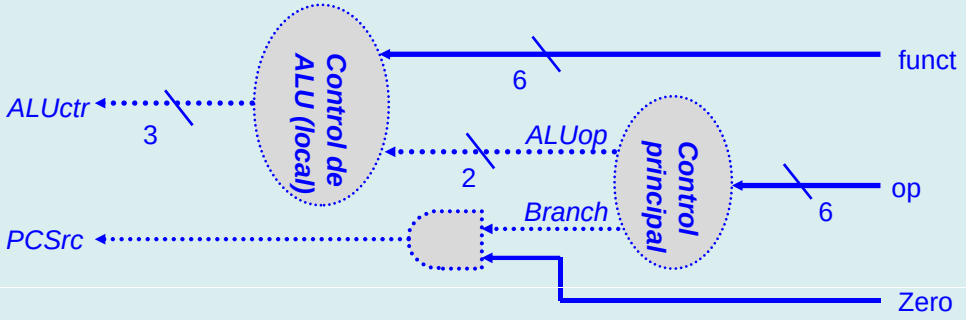
Instrucción de salto condicional (beq)

si (rs = rt) entonces (PC ← PC + 4 + 4·SignExp(inmed)) en otro caso PC ← PC + 4

RegDest ←X, RegWrite ← 0, ALUsrc ← 0, ALUctr ← 110, PCSrc ← Zero, MemWrite ← 0, MemRead ← 0, MemtoReg ← X

Diseño del controlador de la ruta de datos monociclo (2)

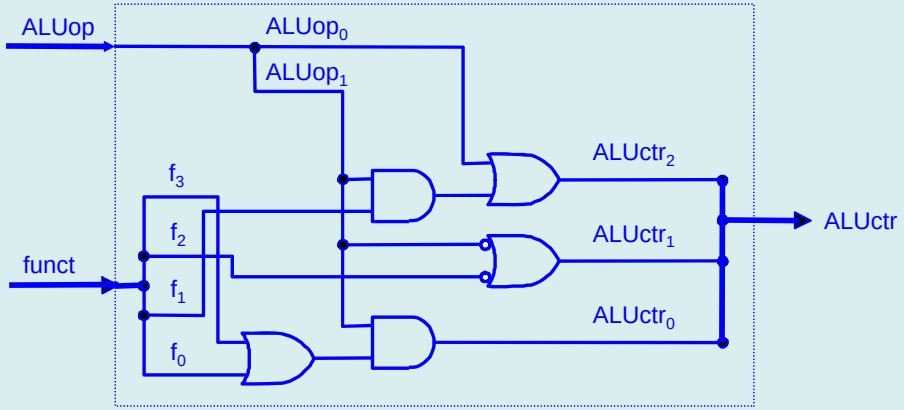
- Todas las operaciones aritméticas comparten el mismo código de operación y durante su ejecución todas las señales generales de la ruta de datos son iguales, por ello utilizaremos:
  - Un **control principal** para decodificar el campo de código de operación (**op**) y configurar globalmente la ruta de datos
  - Un **control local** a la ALU que decodifique el campo de operación aritmética (**funct**) y seleccione la operación que debe realizar
- Adicionalmente, en operaciones no aritméticas (lw, sw y beq) el control principal puede ordenar alguna operación a la ALU para calcular las DE o realizar comparaciones.
- Utilizaremos la señal intermedia ***ALUop*** cuyo valor será:
  - 00 en operaciones con acceso a memoria
  - 01 en operaciones de salto
  - 10 en operaciones aritméticas
- Del mismo modo para controlar qué dirección debe cargar el PC se utilizará una señal intermedia ***Branch*** (activada durante la instrucción beq) a la que se hará la y-lógica con la señal ***Zero*** que genera la ALU.



Diseño del controlador de la ruta de datos monociclo (3)

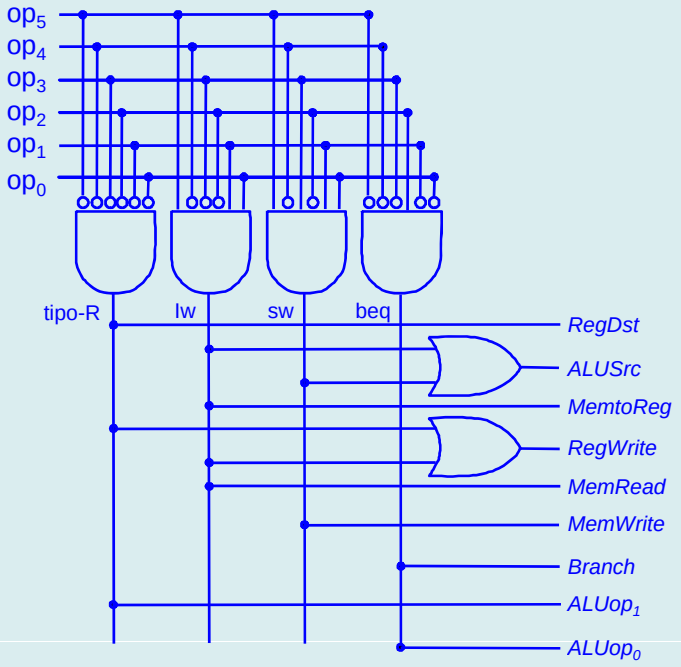
- Control de la ALU

op	funct	ALUop	ALUctr
100011 (lw)	XXXXXX	00	010
101011 (sw)		00	010
000100 (beq)		01	110
000000 (tipo-R)	100000 (add)	10	010
	100010 (sub)	10	110
	100100 (and)	10	000
	100101 (or)	10	001
	101010 (slt)	10	111



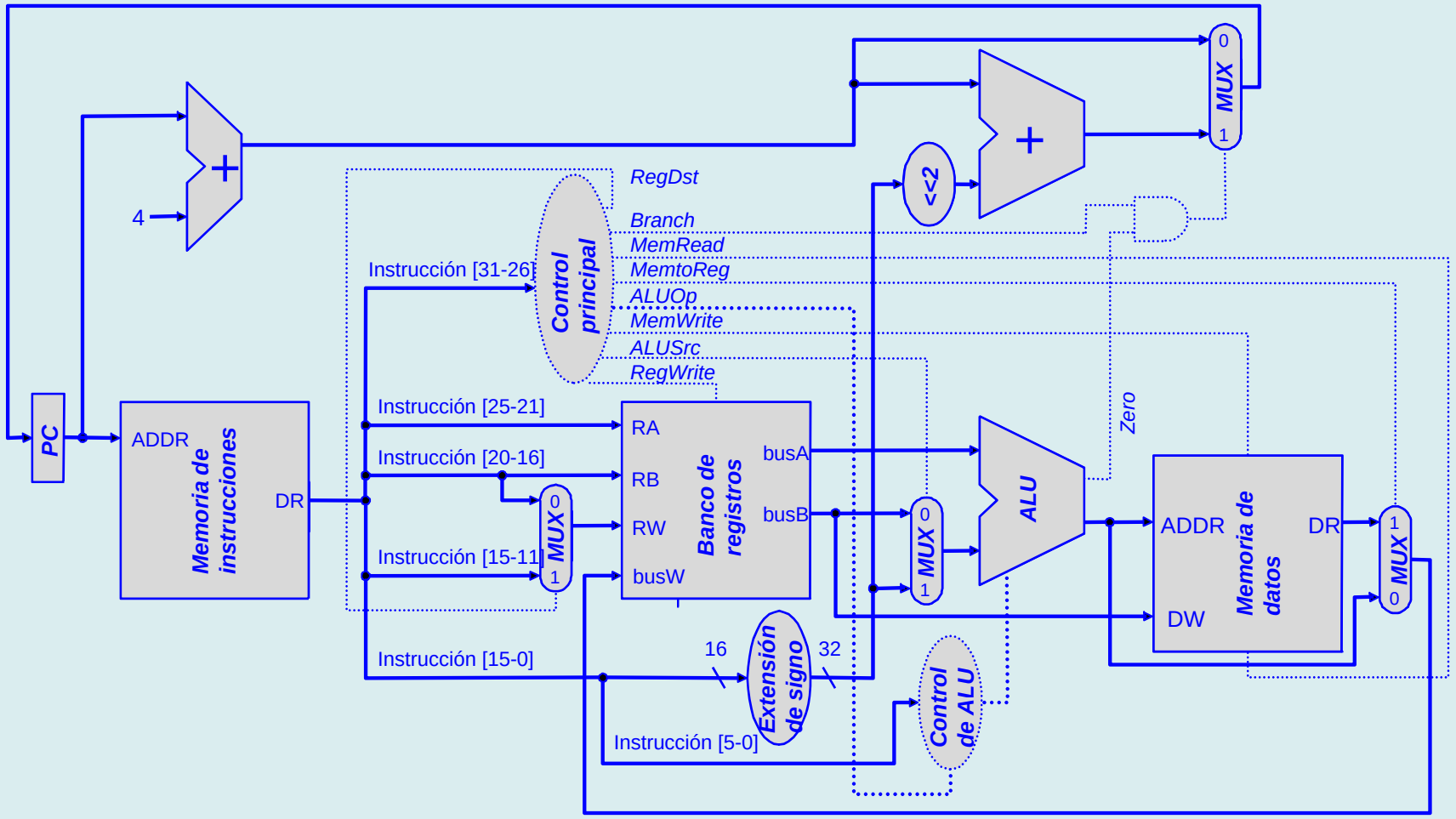
- Control principal

op	RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Brach	ALUop
100011 (lw)	0	1	1	1	1	0	0	00
101011 (sw)	X	1	X	0	0	1	0	00
000100 (beq)	X	0	X	0	0	0	1	01
000000 (tipo -R)	1	0	0	1	0	0	0	10



Diseño del procesador

- **Ruta de datos monociclo + controlador**

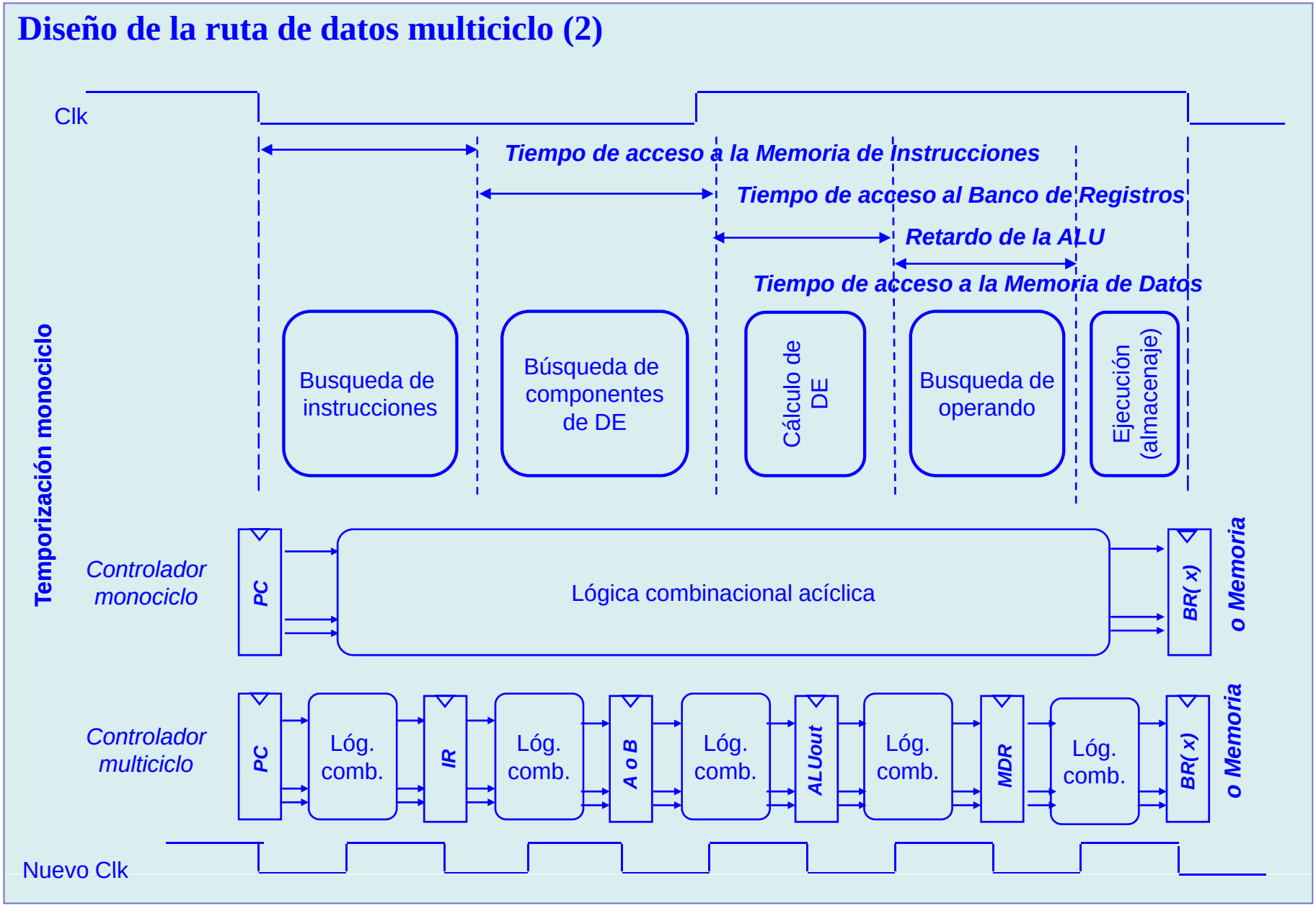


Diseño de la ruta de datos multiciclo (1)

- **Inconvenientes de la ruta de datos monociclo**
  - El reloj debe tener igual periodo que la instrucción más lenta
    - Con un periodo fijo las instrucciones rápidas desaprovechan tiempo.
    - En repertorios reales existen coexisten instrucciones cortas con otras muy largas: aritmética en punto flotante, modos de direccionamiento complejos, etc.
  - No se puede reutilizar hardware
    - Si en una instrucción se necesitara hacer 4 sumas (resolver los 3 modos de direccionamiento de los operandos y sumarlos) se necesitarían 4 sumadores.

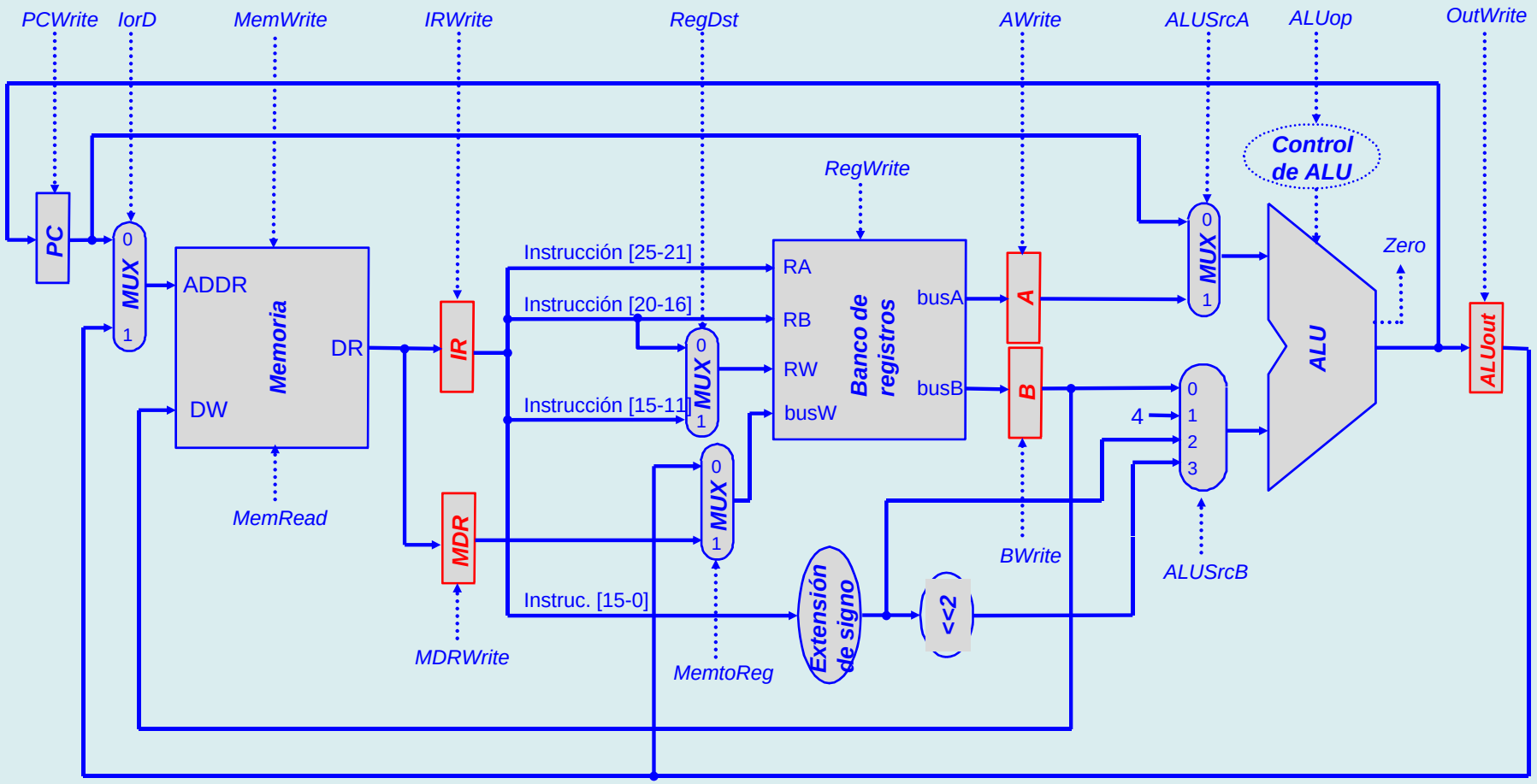
- **Solución: dividir la ejecución de la instrucción en varios ciclos más pequeños**
  - Cada instrucción usará el número de ciclos que necesite.
  - Un mismo elemento hardware puede ser utilizado varias veces en la ejecución de una instrucción si se hace en ciclos diferentes.
  - Se requieren elementos adicionales para almacenar valores desde el ciclo en que se calculan hasta el ciclo en que se usan.

Diseño de la ruta de datos multiciclo (2)



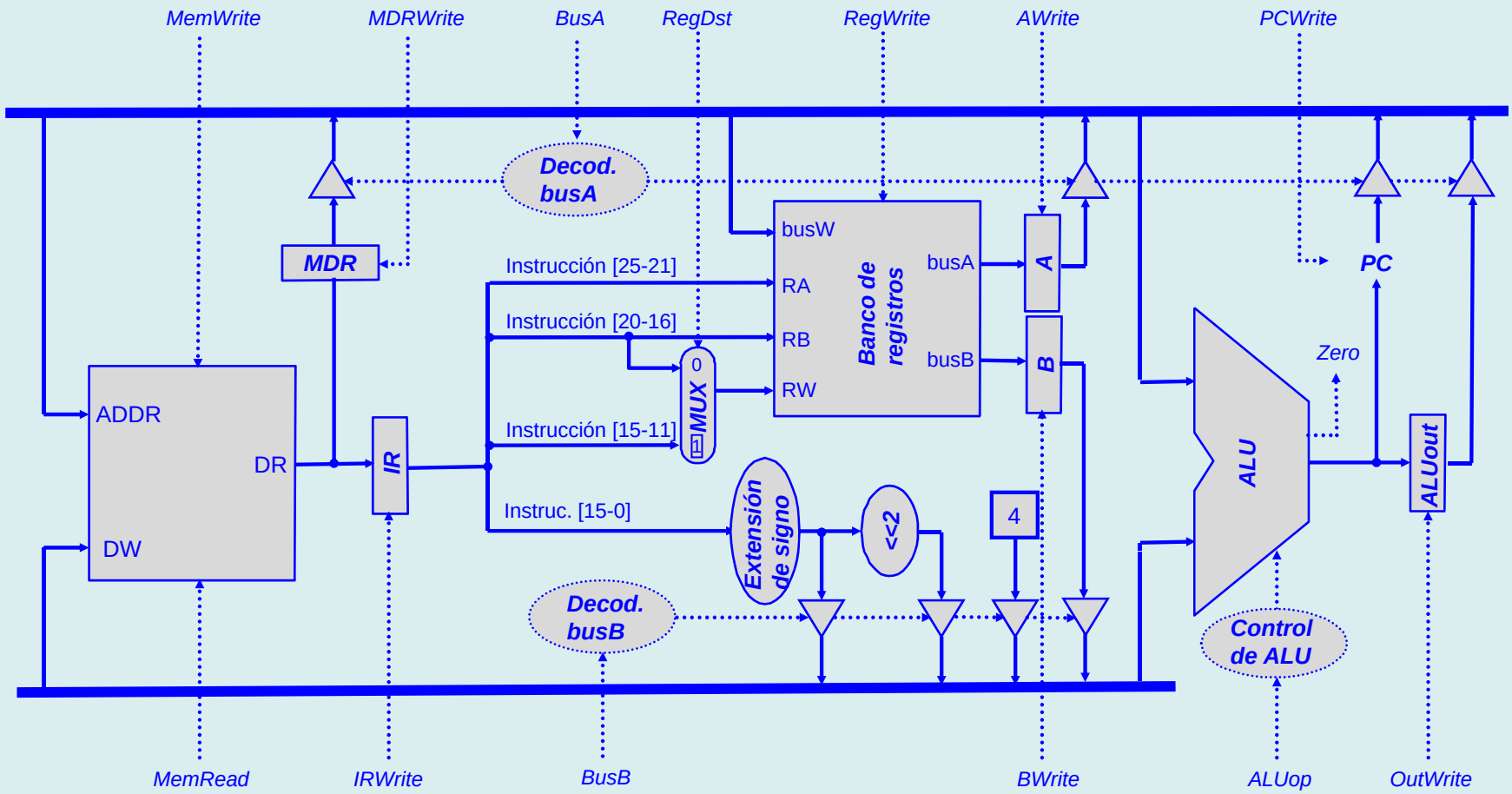
Diseño de la ruta de datos multiciclo (3)

- Introducción de registros para almacenar datos intermedios entre ciclos



Diseño de la ruta de datos multiciclo (4)

- **Ruta de datos multiciclo con buses**



Diseño de la ruta de datos multiciclo (5)

Instrucción de carga (lw)

Transferencias entre registros “lógicas”

BR(rt) ← Memoria(BR(rs) + SignExt(inmed)),

PC ← PC + 4

Transferencias entre registros “físicas”

1. IR ← Memoria(PC), PC ← PC + 4
2. A ← BR(rs)
3. ALUout ← A + SignExt(inmed)
4. MDR ← Memoria(ALUout)
5. BR(rt) ← MDR

Instrucción de almacenaje (sw)

Transferencias entre registros “lógicas”

Memoria(BR(rs) + SignExt(inmed)) ← BR(rt),

PC ← PC + 4

Transferencias entre registros “físicas”

1. IR ← Memoria(PC), PC ← PC + 4
2. A ← BR(rs), B ← BR(rt)
3. ALUout ← A + SignExt(inmed)
4. Memoria(ALUout) ← B

Instrucción aritmética (tipo-R)

Transferencias entre registros “lógicas”

BR(rd) ← BR(rs) funct BR(rt),

PC ← PC + 4

Transferencias entre registros “físicas”

1. IR ← Memoria(PC), PC ← PC + 4
2. A ← BR(rs), B ← BR(rt)
3. ALUout ← A funct B
4. BR(rd) ← ALUout

Instrucción de salto condicional (beq)

Transferencias entre registros “lógicas”

si (BR(rs) = BR(rt))

entonces PC ← PC + 4 + 4·SignExt(inmed)

sino PC ← PC + 4

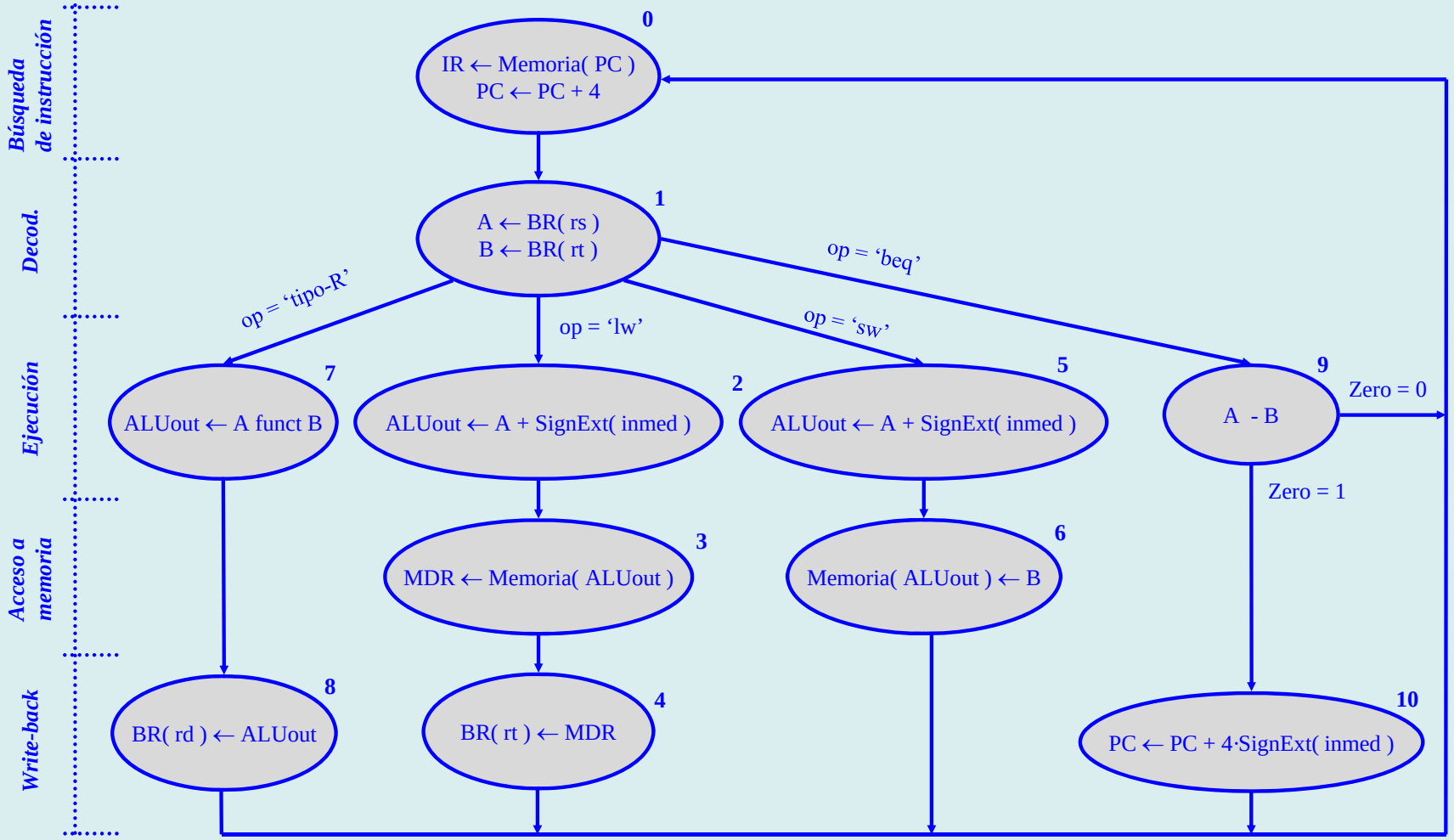
Transferencias entre registros “físicas”

1. IR ← Memoria(PC), PC ← PC + 4
2. A ← BR(rs), B ← BR(rt),
3. A - B
4. si Zero entonces PC ← PC + 4·SignExt(inmed)

Observaciones: en todas las instrucciones las acciones 1. y 2. son iguales (excepto en lw, pero no habría problema en modificarla)

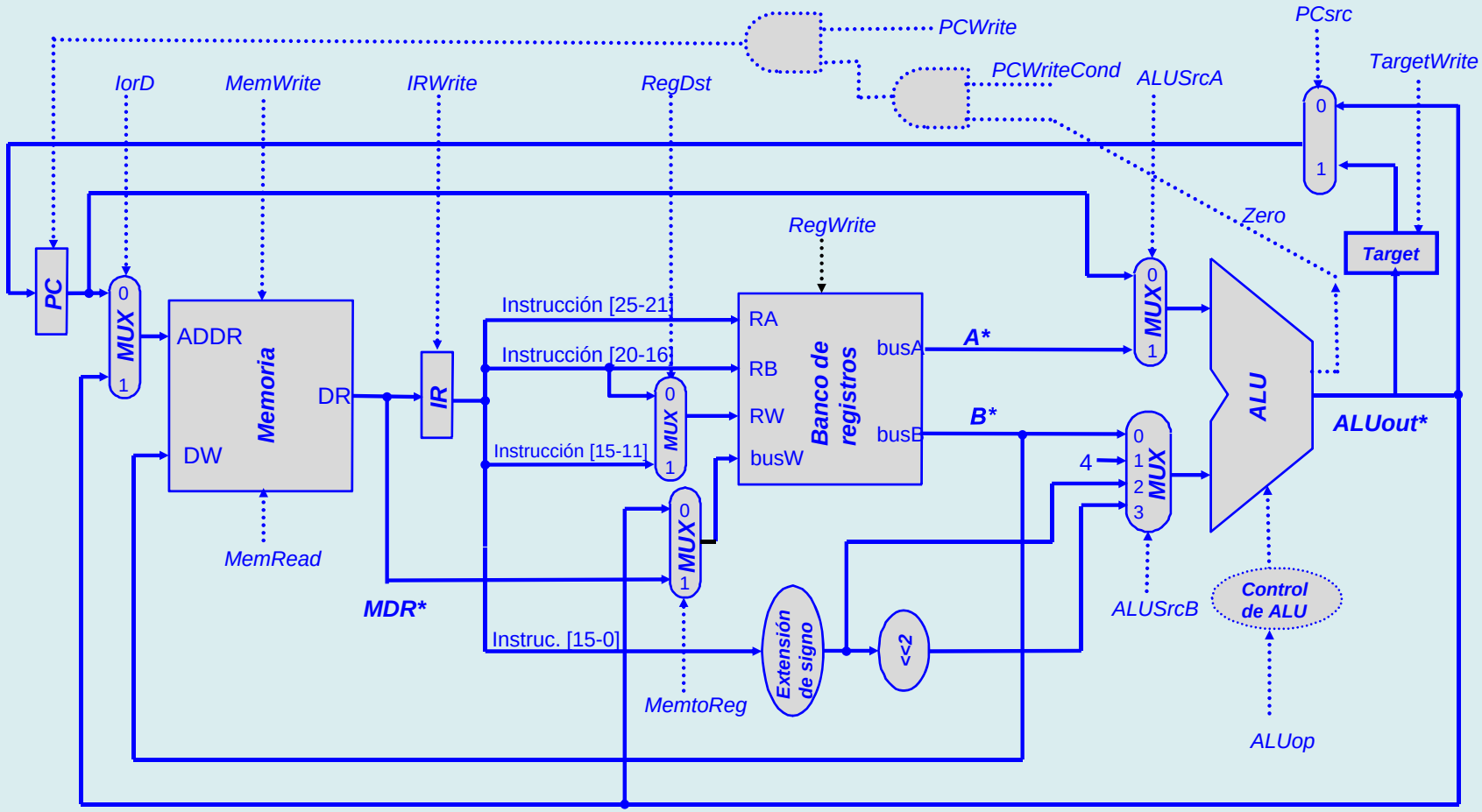
Diseño del controlador multiciclo

- **Diagrama de estados del controlador multiciclo**



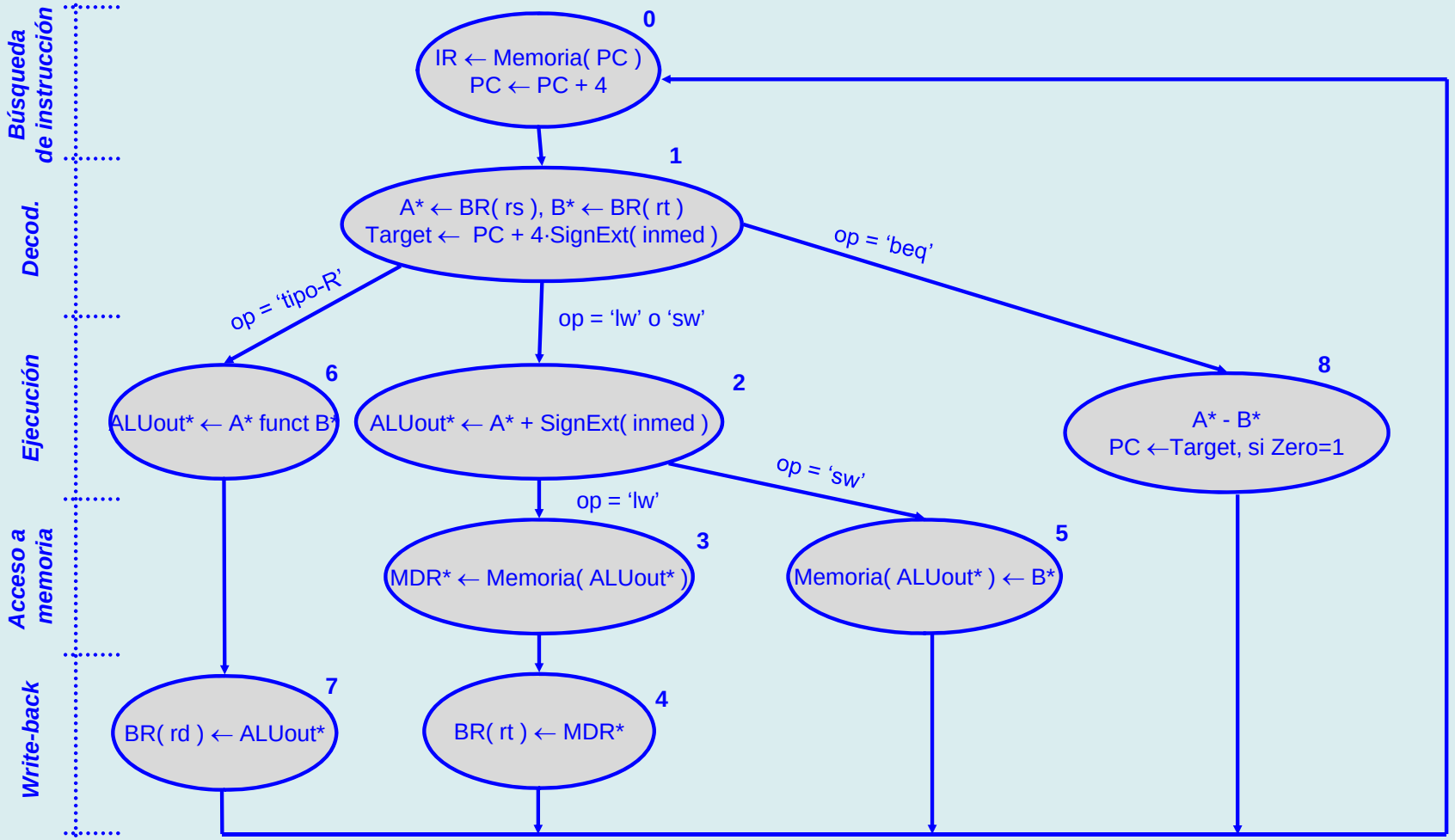
Optimización de la ruta de datos multiciclo

- Ahorro de registros temporales y tiempo de ejecución



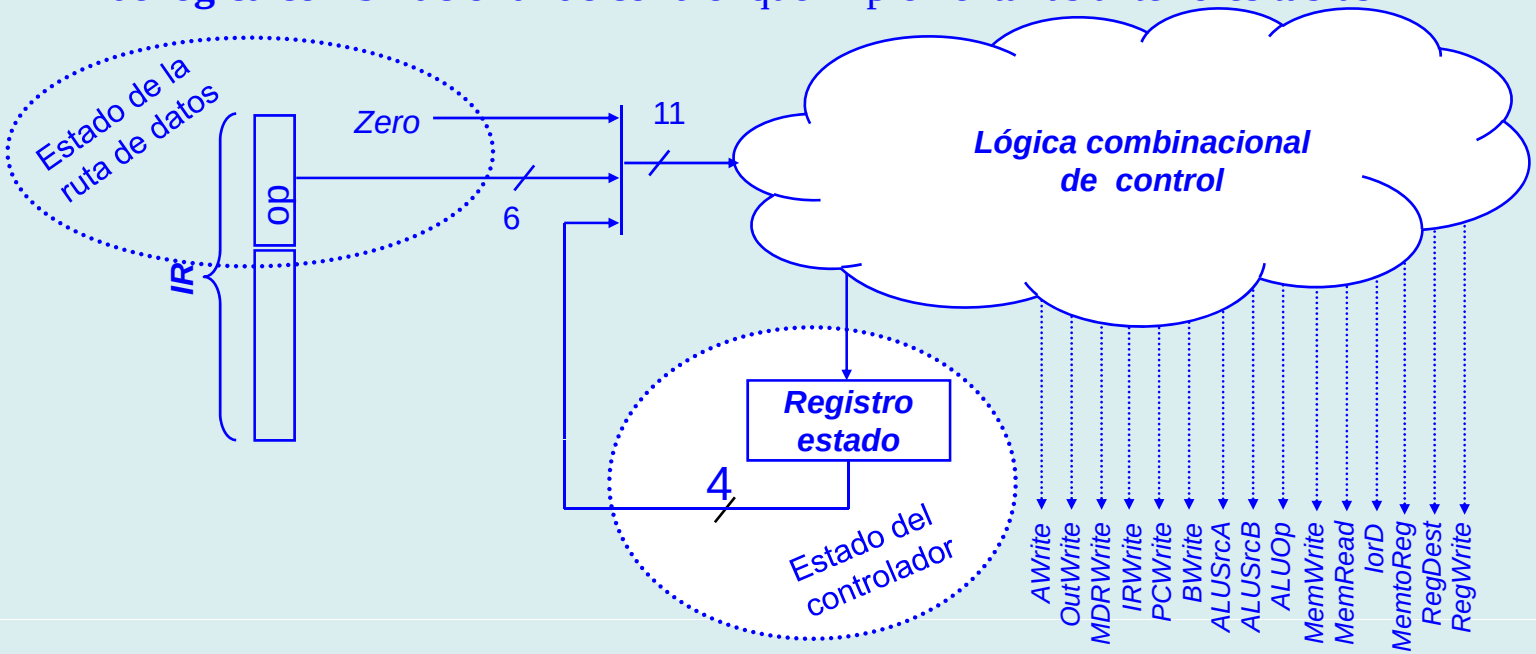
Controlador multiciclo para la ruta de datos optimizada

- Diagrama de estados del controlador multiciclo
  - Se ahorran los estados antiguos 5 y 10 y registros



Diseño del controlador multiciclo (1)

- El controlador se diseña como una máquina de estados finitos (FSM):
  1. Se **traducen** las transferencias entre registros como **conjuntos de activaciones** de los puntos de control de la ruta de datos
  2. Se **codifican** los estados
  3. Mediante **tablas de verdad** se describen las **transiciones de estado** en función del código de operación y del estado actual de la ruta de datos
  4. Mediante tablas de verdad se describe el **valor de las señales** de control en función del estado (máquina Moore) y adicionalmente en función del estado de la ruta de datos (máquina Mealy).
  5. La **estructura del controlador** estará formada por **registro de estado** y el conjunto de **lógica combinacional de control** que implementa las anteriores tablas



Diseño del controlador multiciclo (2)

- Tabla de verdad del controlador

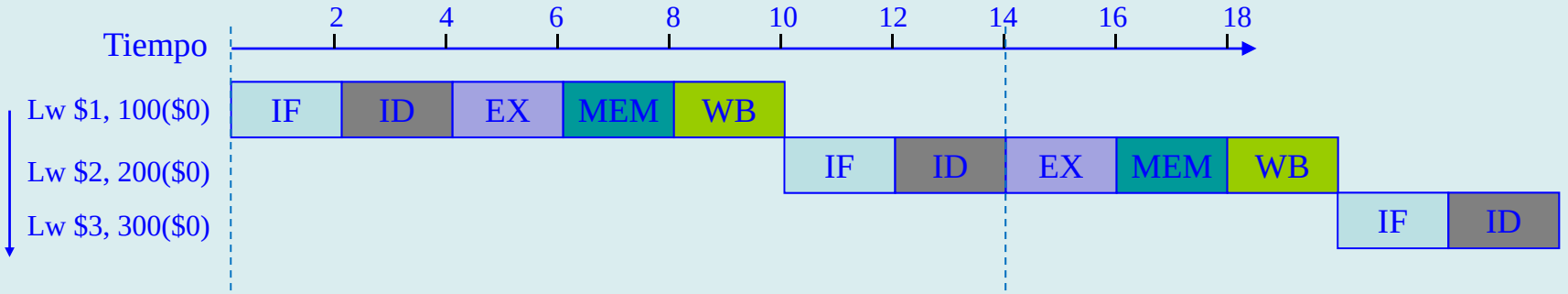
Estado actual	op	Zero	Estado siguiente	IRWrite	PCWrite	AWrite	BWrite	ALUSrcA	ALUScrB	ALUOp	OutWrite	MemWrite	MemRead	IorD	MDRWrite	MemtoReg	RegDest	RegWrite
0000	XXXXXX	X	0001	1	1			0	01	00 (add)		0	1	0				0
0001	100011 (lw)	X	0010															
0001	101011 (sw)	X	0101															
0001	000000 (tipo-R)	X	0111	0	0	1	1					0	0					0
0001	000100 (beq)	X	1001															
0010	XXXXXX	X	0011	0	0			1	10	00 (add)	1	0	0					0
0011	XXXXXX	X	0100	0	0							0	1	1	1			0
0100	XXXXXX	X	0000	0	0							0	0			1	0	1
0101	XXXXXX	X	0110	0	0		0	1	10	00 (add)	1	0	0					0
0110	XXXXXX	X	0000	0	0							1	0	1				0
0111	XXXXXX	X	1000	0	0			1	00	10 (funct)	1	0	0					0
1000	XXXXXX	X	0000	0	0							0	0			0	1	1
1001	XXXXXX	0	0000	0	0			1	00	01 (sub)		0	0					0
1001	XXXXXX	1	1010															
1010	XXXXXX	X	0000	0	1			0	11	00 (add)		0	0					0

Segmentación (1)

• La segmentación es una técnica que permite optimizar (minimizar) el tiempo de ejecución de los programas.

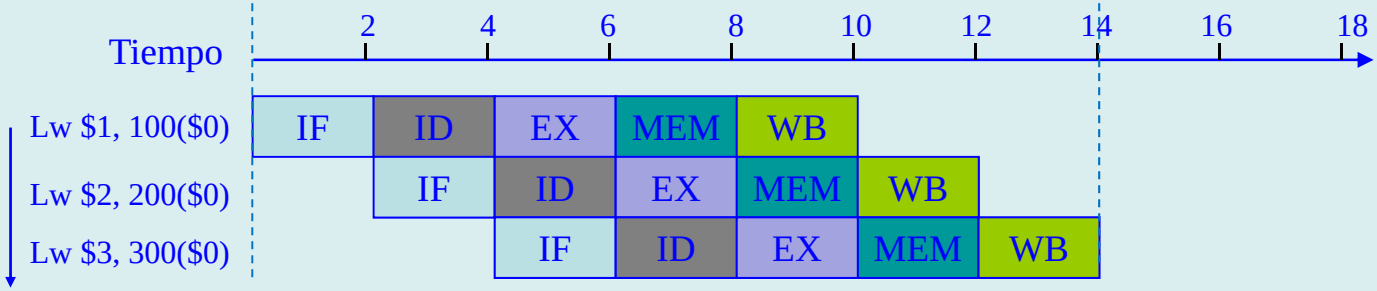
• Consiste en aplicar el principio de *fabricación en cadena* al proceso de ejecución de las instrucciones, solapando en el tiempo la ejecución de fases diferentes de diferentes instrucciones:

Ejecución secuencial del programa



30ns

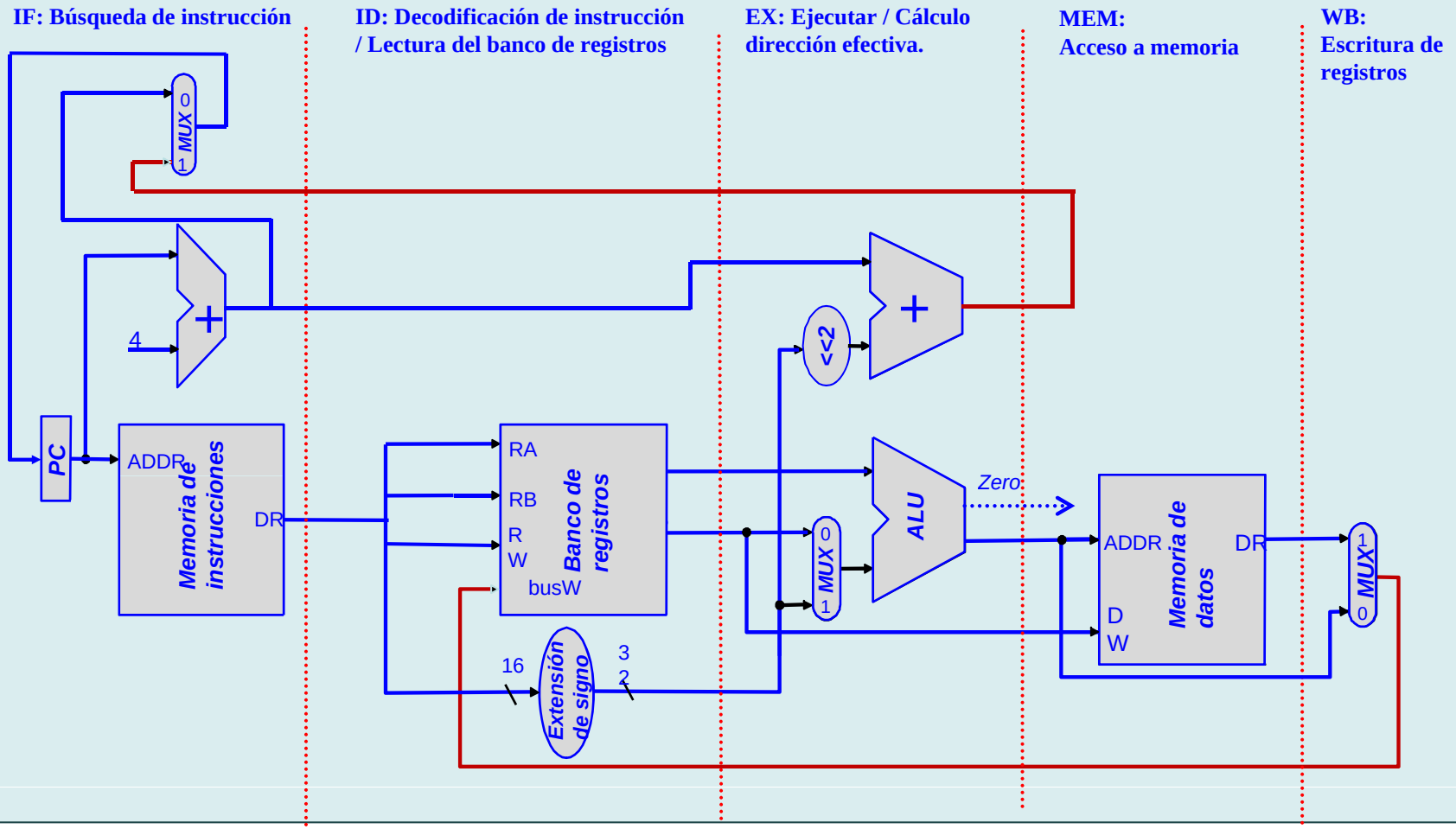
Ejecución segmentada del programa



14ns

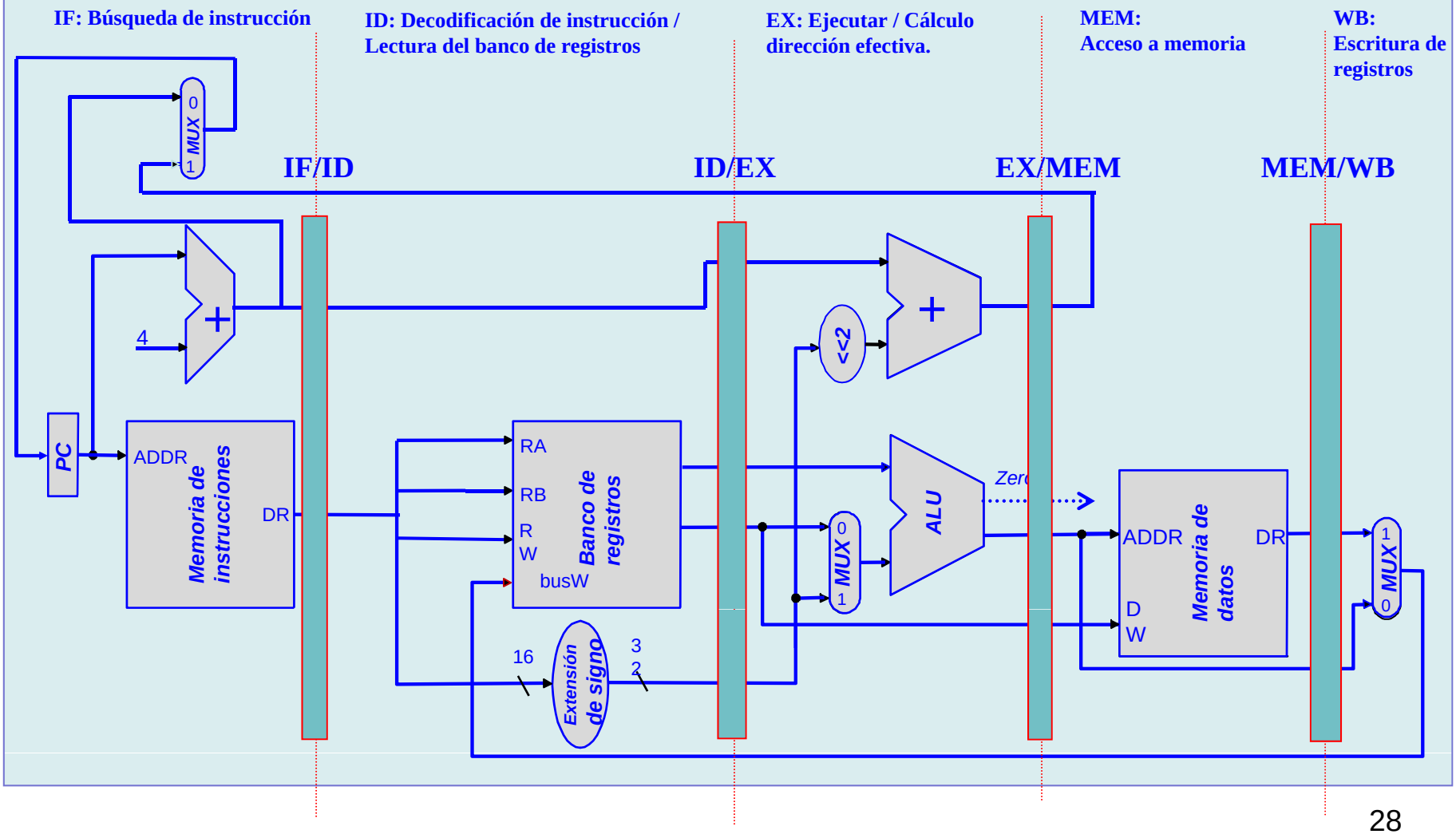
Segmentación

- Modificación de la ruta de datos para segmentar la ejecución de instrucciones
  - Se replican los sumadores para actualizar el PC y calcular la dirección de memoria
  - Se replica la memoria para datos e instrucciones



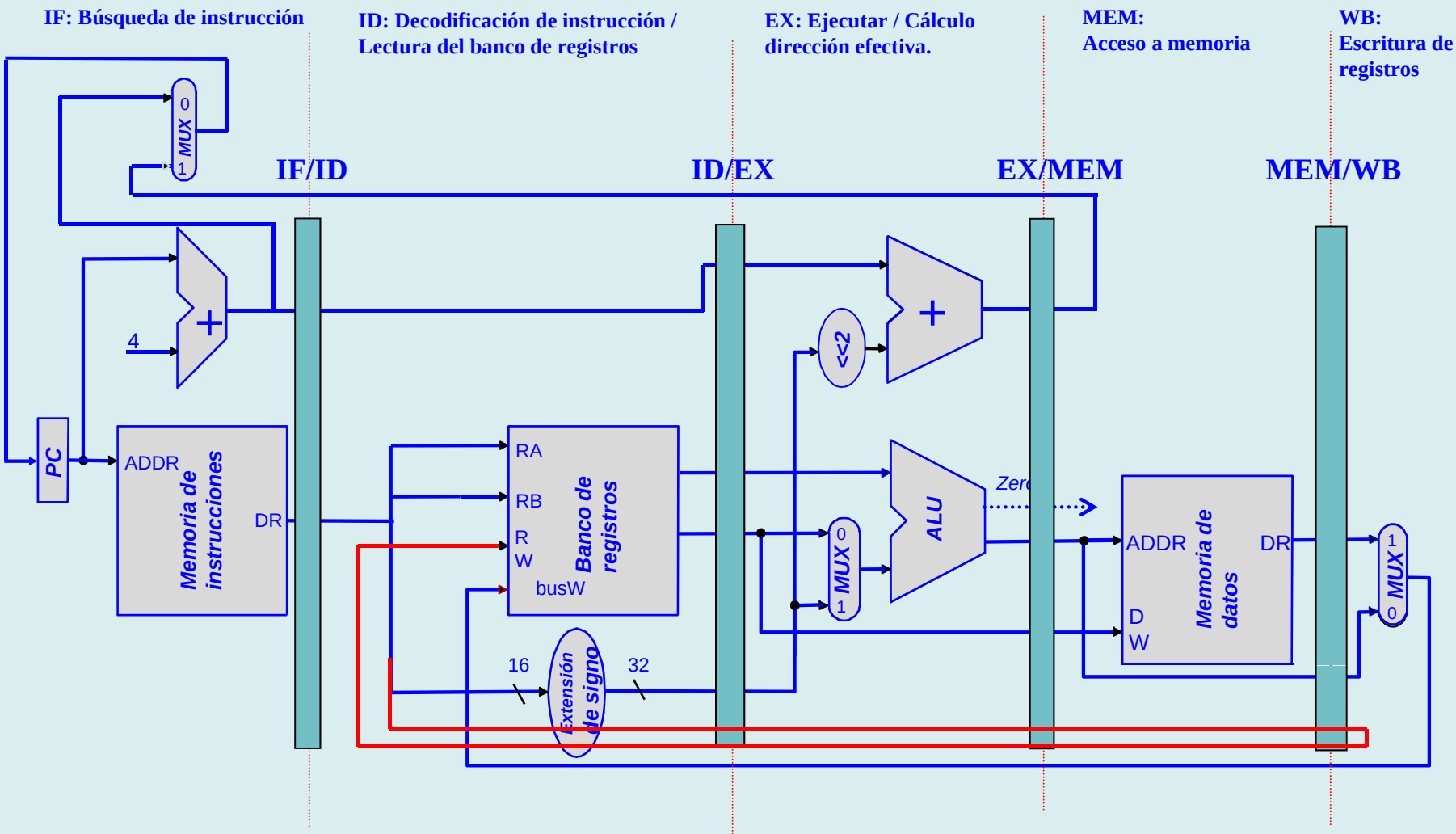
Ruta de datos segmentada (1)

• Se necesitan registros entre etapas para pasar la información de datos y control a la siguiente etapa y la actual quede libre para recibir una nueva instrucción



Ruta de datos segmentada (2)

- Modificación de algunas señales



Ruta de datos segmentada

- Nombramiento de los puntos de control