

El Microprocesador: Camino de Datos

Sistemas con Microprocesadores

Ing. Esteban Volentini (evolentini@herrera.unt.edu.ar)

<http://www.microprocesadores.unt.edu.ar/procesadores>

Cronograma

Actividad	Inicio	Descripción	Completa
Presentación	24/08	Presentación y reglamento	✓
Tema 1	24/08	Estructura de las computadoras	✓
Tema 2	26/08	Proyecto con un microcontrolador	✓
Tema 3	31/08	Descripción funcional del ISA RV32IM	✓
Tema 4	07/09	Programación en lenguaje ensamblador	✓
Tema 5	18/09	Diseño del camino de datos de un procesador	←
Tema 6	30/09	Diseño de la unidad de control del procesador	
Tema 7	05/10	Tecnologías y sistema de memoria	
Tema 8	09/10	Dispositivos de entrada/salida	
Tema 9	19/10	Entradas/salidas digitales (GPIO)	
Tema 10	28/10	Generalidades de las interrupciones	
Tema 11	04/11	Interrupciones y excepciones en RISC-V	
Tema 12	13/11	Manejo de señales con temporizadores	
Tema 13	18/11	Transferencias de datos	
Proyectos	30/11	Seminarios de proyectos	

Temas a tratar

- ▶ Componentes combinacionales y secuenciales
- ▶ Metodología de clocking
- ▶ Pasos para diseñar un procesador
- ▶ RTN Abstracto
- ▶ Construcción paso a paso de un camino de datos
- ▶ RTN Concreto

Objetivo

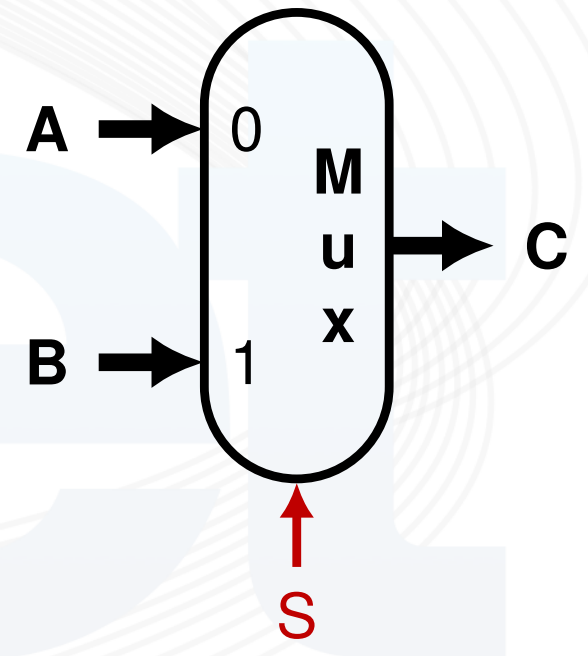
- ▶ Diseñar un camino de datos para el ISA RV32I
- ▶ Sin duplicar recursos innecesarios
- ▶ Que ejecute las instrucciones en múltiples ciclos de reloj
- ▶ Con un ciclo de reloj razonablemente pequeño

Componentes Combinacionales (1/3)

- ▶ Las salidas son función únicamente de las entradas
- ▶ Tanto las entradas como las salidas pueden ser de uno o más bits
- ▶ Cada salida se puede escribir como una función lógica a partir de las entradas

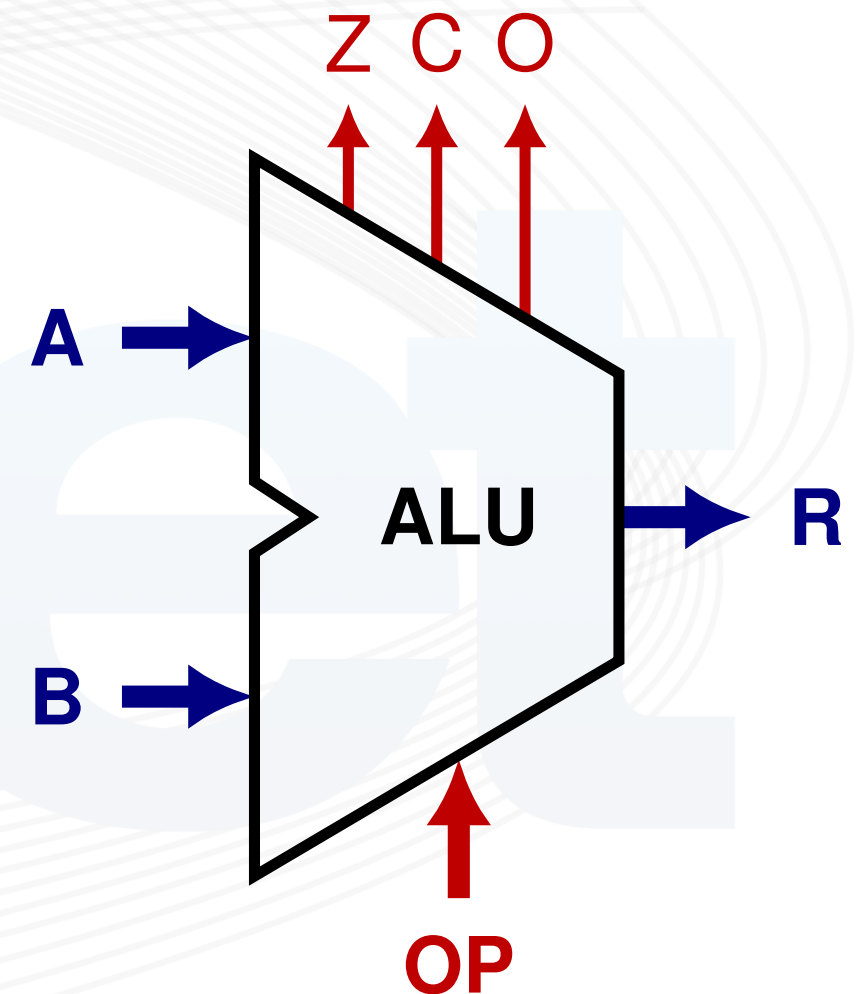
Componentes Combinacionales (2/3)

- ▶ Un multiplexor permite seleccionar un destino entre dos o más fuentes
- ▶ También se puede seleccionar un número de n bits:
 - ▶ Se requieren n multiplexores de 1 bit en paralelo
 - ▶ Todas las señales de control de estos toman el mismo valor



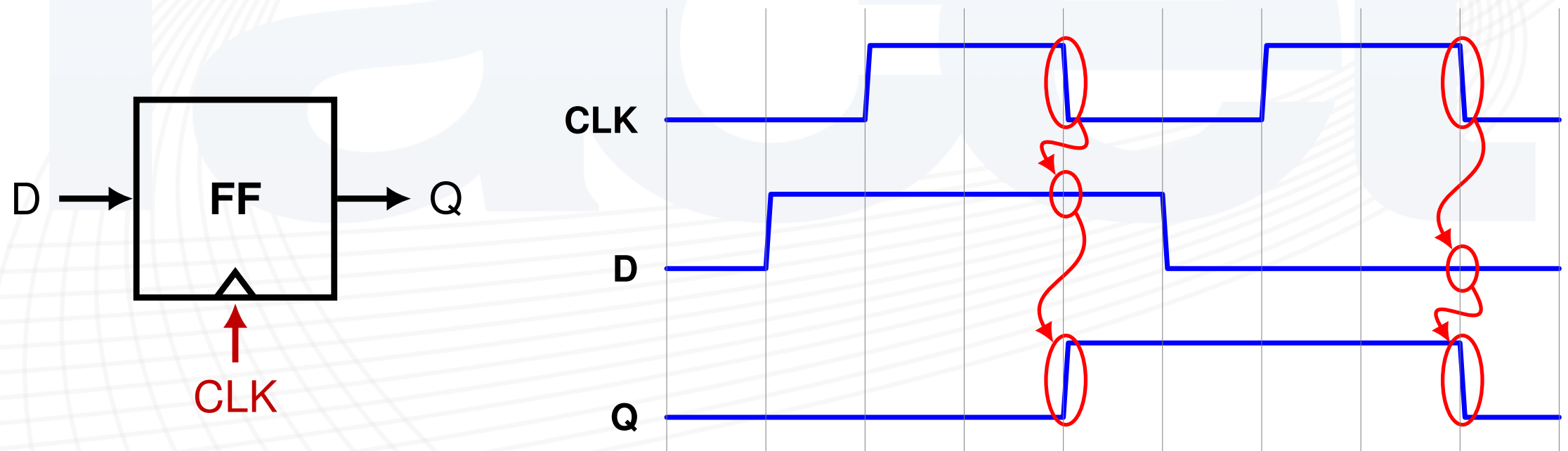
Componentes Combinacionales (3/3)

- ▶ Una ALU permite calcular el resultado de una operación matemática o lógica entre dos números de n bits
 - ▶ Entradas A y B para los operandos, números de n bits
 - ▶ Entrada OP que permite elegir la operación: sumar, restar, etc
 - ▶ Salida R para el resultado, un número de n bits (en general)
 - ▶ Salidas con información del resultado: Zero, Carry, Overflow



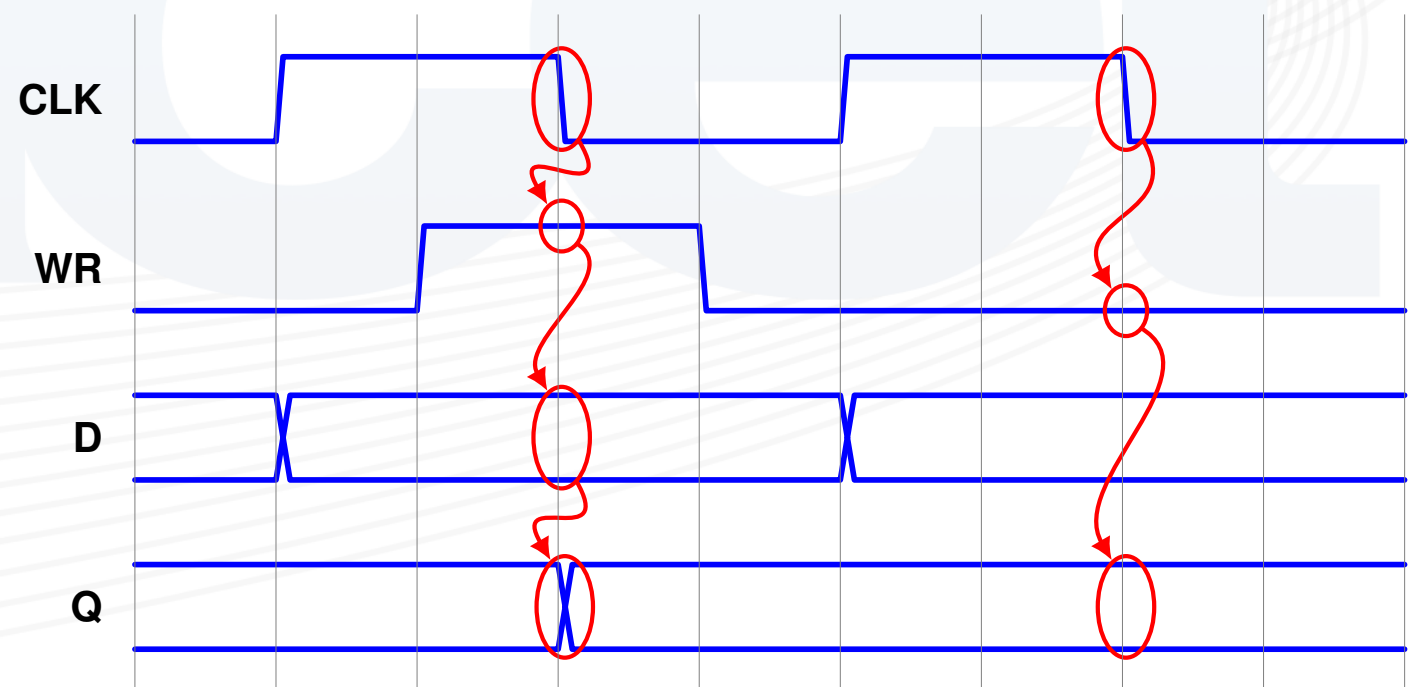
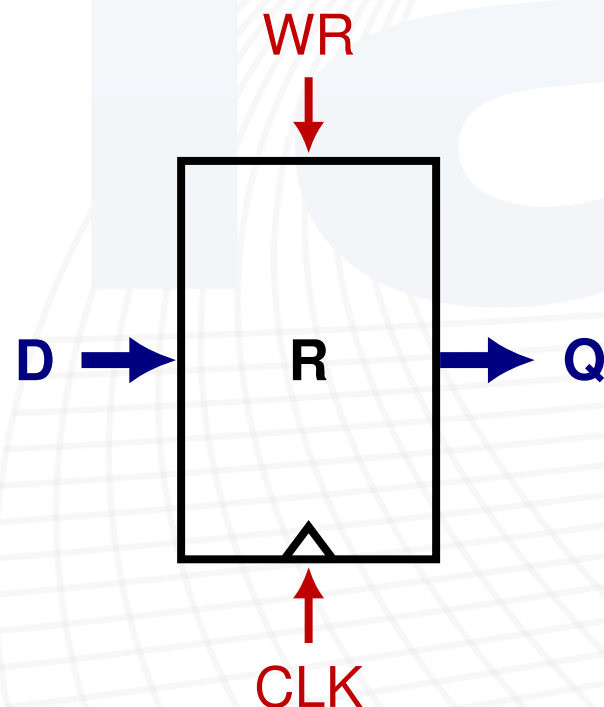
Componentes secuenciales (1/7)

- ▶ Las salidas son función no solo de las entradas sino también del estado actual, es decir que almacenan información
- ▶ Se utilizan, principalmente, flip-flops tipo D
 - ▶ Una señal de reloj determina actualiza el valor almacenado
 - ▶ Normalmente, se actualiza en un flanco de la señal de reloj



Componentes secuenciales (2/7)

- ▶ Para almacenar números se utiliza registros tipo D
 - ▶ Un conjunto de n flip-flops con la misma señal de reloj
 - ▶ Pueden tener una señal de entrada adicional para escritura
 - ▶ Sólo actualiza el valor almacenado si, cuando llega el flanco de reloj, la señal de escritura está activa



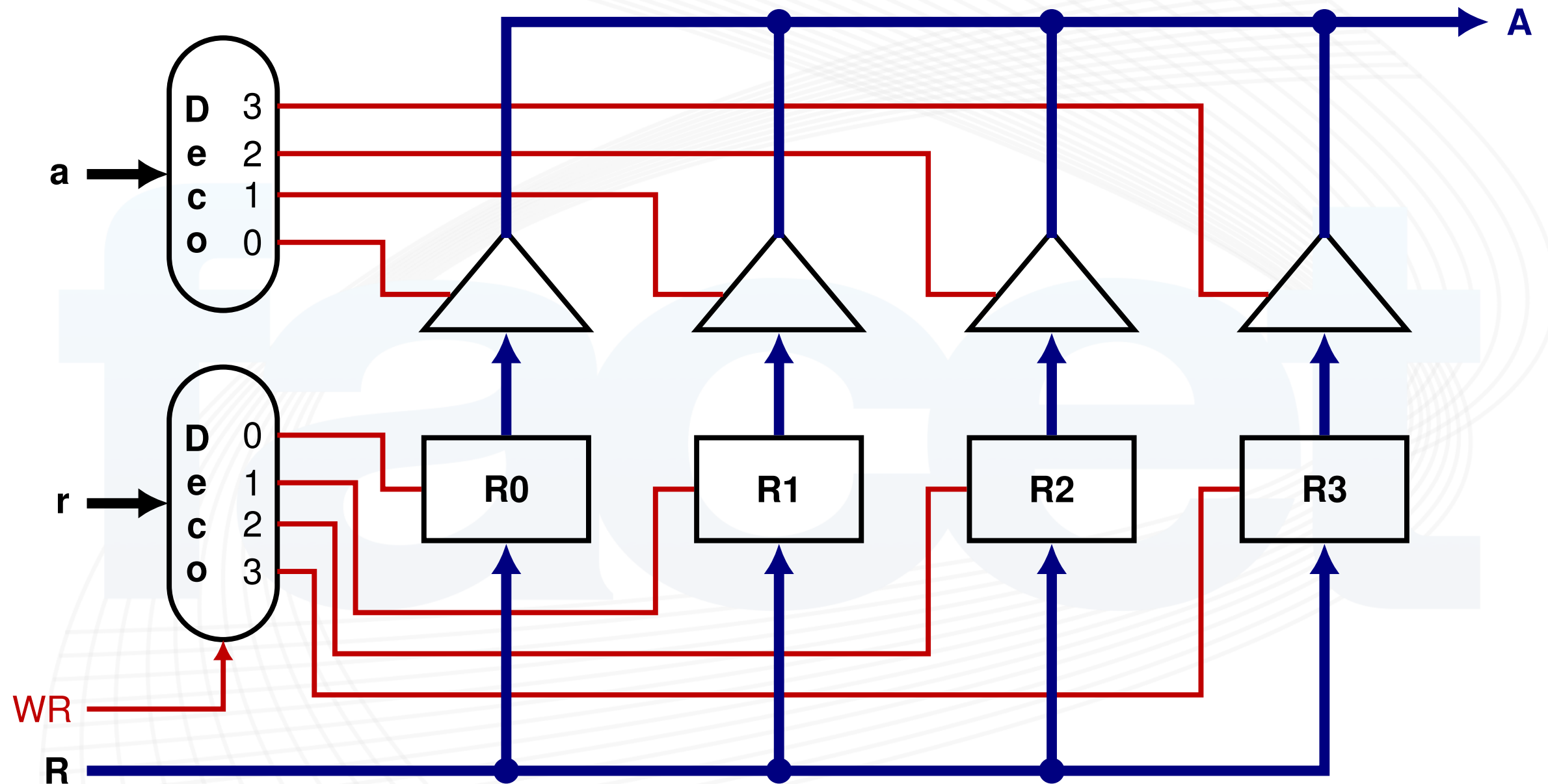
Componentes secuenciales (3/7)

- ▶ Los registros pueden tener su salida tri-state
 - ▶ Manejada mediante una entrada de control (OE, Output Enable)
 - ▶ OE = 1, muestra por la salida el contenido del registro
 - ▶ OE = 0, la salida se mantiene en un estado de alta impedancia
- ▶ Esto permite conectar varios registros mediante un conjunto único de conectores (cables)
 - ▶ Normalmente denominado bus
 - ▶ Restricción: solamente un registro puede habilitar su salida al bus en un instante determinado

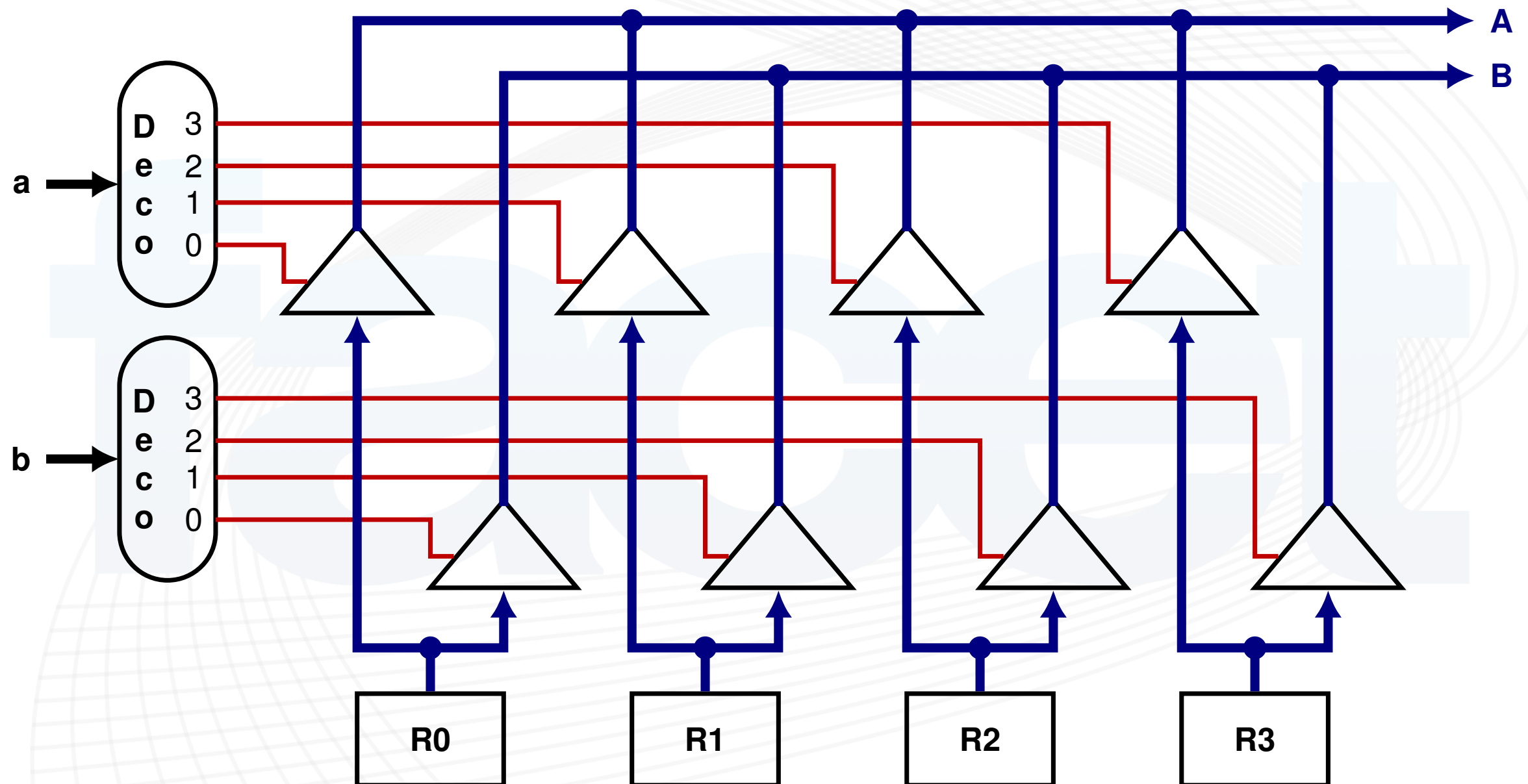
Componentes secuenciales (4/7)

- ▶ También pueden agruparse en una estructura llamada Banco de Registros
 - ▶ Usualmente agrupan N registros, con N una potencia de 2
- ▶ Para seleccionar un registro en particular, para lectura, posee una entrada de control
 - ▶ Esta entrada tendrá $\log_2(N)$ bits
 - ▶ La selección se realiza mediante un decodificador interno
 - ▶ Una vez que una entrada de lectura está estable, después de un retardo, se obtiene el contenido del registro seleccionado por la salida correspondiente
 - ▶ Pueden tener una o más entradas de este tipo, para seleccionar varios registros al mismo tiempo
- ▶ También pueden tener una o más entradas de escritura, con su correspondiente entrada de selección, y con su correspondiente señal de control para escritura
 - ▶ Para modificar el valor almacenado en un registro

Un banco de registros por dentro

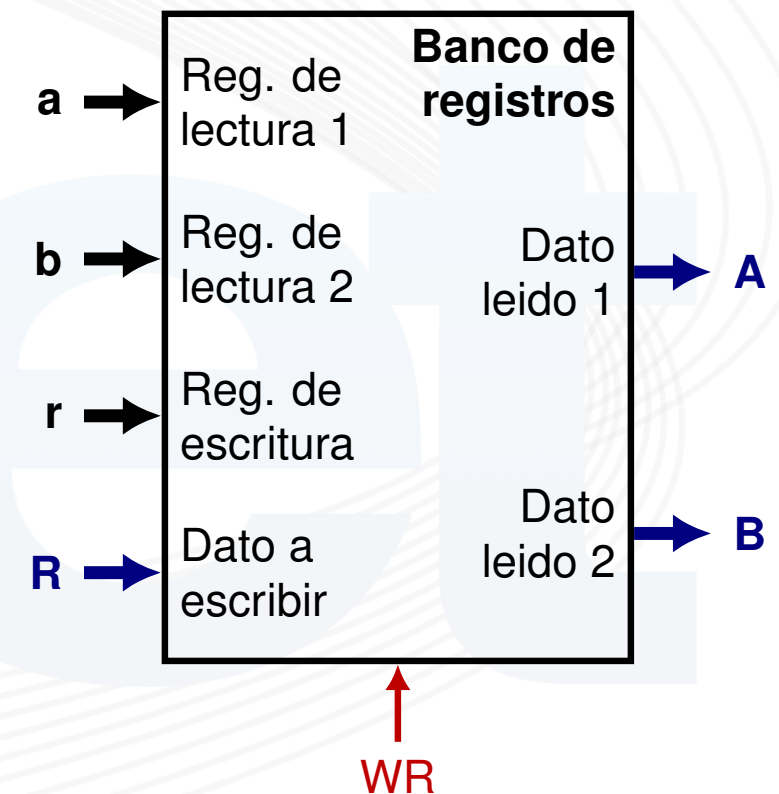


Un banco de registros por dentro



Componentes secuenciales (5/7)

- ▶ Ejemplo de Banco de Registros para RV32I:
 - ▶ 32 registros agrupados
 - ▶ 2 puertos de salida (de lectura) de 32 bits cada uno
 - ▶ 2 entradas de selección de lectura de 5 bits cada una
 - ▶ 1 puerto de escritura de 32 bits
 - ▶ 1 entrada de selección de escritura de 5 bits
 - ▶ 1 señal de control de escritura

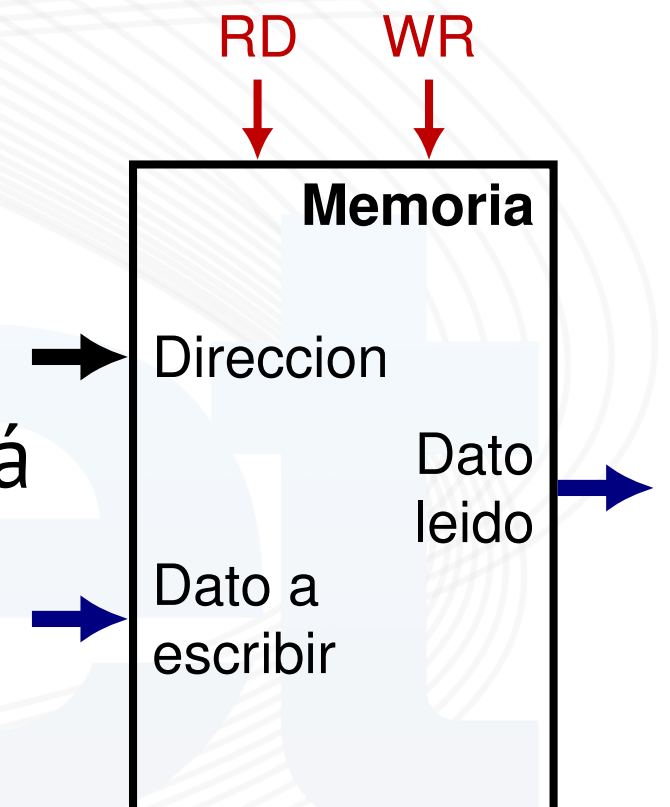


Componentes secuenciales (6/7)

- ▶ Memoria
 - ▶ Componente bastante más complejo, merece su propio tema
- ▶ Por ahora, utilizaremos una memoria “ideal”, que funciona como una tabla unidimensional de datos:
 - ▶ Una entrada de direcciones, que funciona como índice en la tabla y permite elegir una dirección de memoria específica
 - ▶ Una entrada de datos, que permite tomar un valor y escribirlo en una dirección dada
 - ▶ Una salida de datos, para recuperar el valor contenido en una dirección de memoria dada
 - ▶ Una señal de control de lectura, que se debe activar para leer un dato de una dirección de memoria
 - ▶ Una señal de control de escritura, que se debe activar para escribir un dato en una dirección de memoria

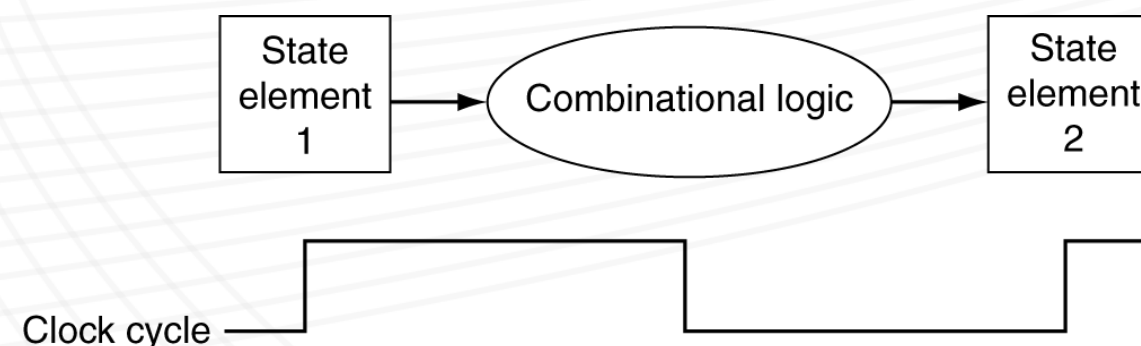
Componentes secuenciales (7/7)

- ▶ Funcionamiento de la Memoria “Ideal”
 - ▶ Cuando la señal de control de lectura está activa, se muestra en la salida el contenido apuntado por la dirección de entrada, después de un cierto tiempo de acceso
 - ▶ Cuando la señal de control de escritura está activa, se guarda en el contenido apuntado por la dirección el valor en la entrada de datos, después de un cierto tiempo de acceso
 - ▶ Se podría considerar como un banco de registros externo más grande y más lento



Metodología de Clocking

- ▶ Define cuándo las señales pueden ser leídas y cuándo escritas.
- ▶ La lógica combinatorial modifica los datos en un ciclo de reloj (entre dos flancos iguales).
- ▶ Siempre toma los datos de un elemento de estado, y guarda los resultados en un elemento de estado.
 - ▶ Es posible que un elemento de estado sea leído y escrito en el mismo ciclo de reloj, sin ambigüedades.
- ▶ El mayor retardo combinatorial entre dos elementos de estado determina la duración del ciclo de reloj.



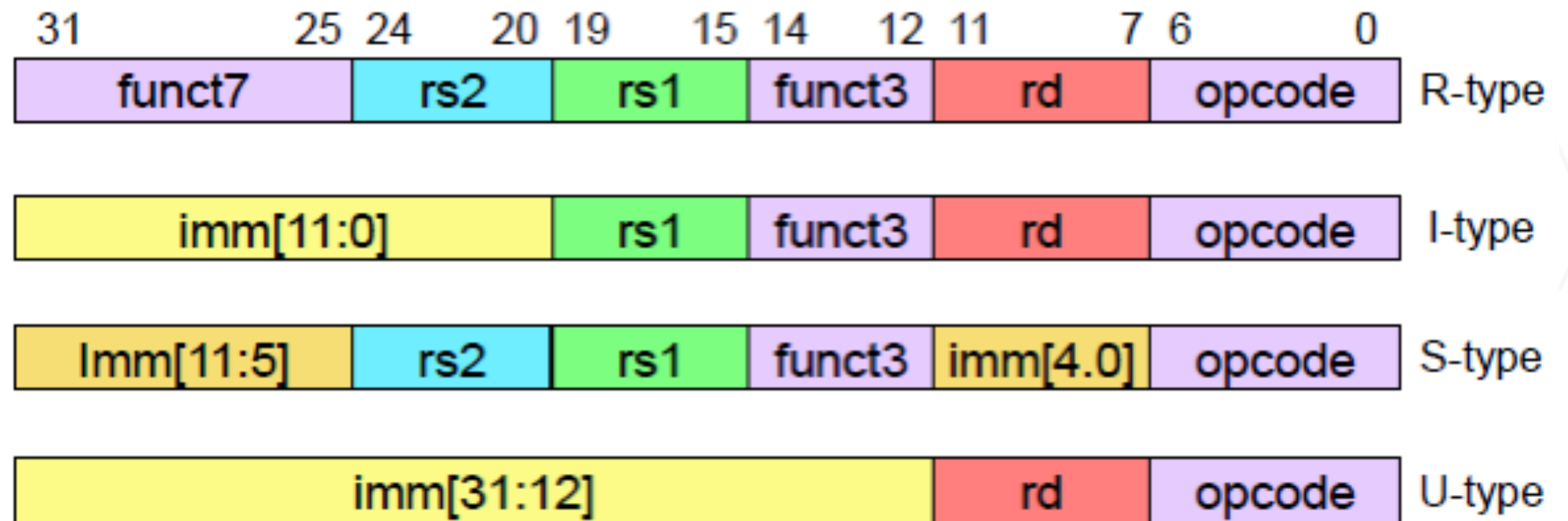
Pasos para el diseño de un procesador

- ▶ Se parte desde el ISA, porque es la interfaz entre hardware y software
 - ▶ El hardware que se diseñe debe ejecutar correctamente las instrucciones
- ▶ Primero, vamos a escribir las instrucciones con una notación específica
 - ▶ RTN (Register Transfer Notation)
- ▶ Esto nos dará una pauta de los componentes necesarios para el camino de datos y de cómo interconectarlos entre sí

Subconjunto de RV32I

- ▶ Diseñaremos un camino de datos para algunas de las instrucciones de RV32I, para simplificar
- ▶ La técnica usada es la misma, así que las restantes quedan para los prácticos
- ▶ Las que contemplaremos son:
 - ▶ De procesamiento: **add, sub, and y or**
 - ▶ De acceso a memoria: **lw y sw**
 - ▶ De control de flujo: **beq**

Repaso: Formato de las instr. RV32I



RTN Abstracto (1/3)

- ▶ Es lo que vimos en los temas anteriores, al momento de describir las instrucciones
- ▶ Indica cómo se utilizan los operandos para producir el resultado de la instrucción
- ▶ Por ejemplo, para la instrucción:
 - ▶ **add rd, rs1, rs2**
- ▶ Su RTN abstracto sería:
 - ▶ **$R[rd] \leftarrow R[rs1] + R[rs2]$**

RTN Abstracto (2/3)

- ▶ ¡No olvidarse del ciclo de la instrucción!
- ▶ La búsqueda de la instrucción (Fetch) es parte de la misma instrucción
 - ▶ Por el momento, vamos a dejar de lado la decodificación de la instrucción
- ▶ Entonces, para la instrucción:
 - ▶ **add rd, rs1, rs2**
- ▶ Su RTN abstracto realmente sería:
 - ▶ **M[PC]** # Se busca en memoria la instrucción apuntada por el registro PC
 - ▶ **PC ← PC + 4** # Se apunta el PC a la instrucción siguiente
 - ▶ Se suma 4 al PC porque las instrucciones son de 32 bits y porque el mismo ISA define que el direccionamiento a memoria es de a byte
 - ▶ **R[rd] ← R[rs1] + R[rs2]** # Realizo la suma

RTN Abstracto (3/3)

Instrucción	RTN Abstracto
add rd, rs1, rs2	M[PC] PC $\leftarrow$ PC + 4 R[rd] $\leftarrow$ R[rs1] + R[rs2]
lw rd, rs1, imm	M[PC] PC $\leftarrow$ PC + 4 R[rd] $\leftarrow$ M[R[rs1] + SignExt(imm)]
sw rs1, rs2, imm	M[PC] PC $\leftarrow$ PC + 4 M[R[rs1] + SignExt(imm)] $\leftarrow$ R[rs2]
beq rs1, rs2, imm	M[PC] PC $\leftarrow$ PC + 4 Si (R[rs1] == R[rs2]) PC $\leftarrow$ PC + SignExt(imm)

- ▶ Todos los Fetch son idénticos, inclusive para los saltos
- ▶ Las instrucciones **lw** y **sw** requieren el cálculo de la dirección a la que accederán en memoria
- ▶ La instrucción **beq** requiere una comparación

Camino de Datos para Fetch (1/5)

- ▶ Como el Fetch es común a todas las instrucciones, empezamos por allí
- ▶ Consta de dos acciones distintas
- ▶ ¿Qué componentes se necesitan?
- ▶ Para hacer **M[PC]** se necesita:
 - ▶ Un registro, **PC**
 - ▶ Mantiene su valor durante todo el ciclo de reloj, y se actualiza al final
 - ▶ Una memoria (ideal)
 - ▶ Por el momento, no son necesarias las señales de control, ni la entrada de datos
 - ▶ La salida de la memoria será la instrucción a decodificar y ejecutar
- ▶ Para hacer **PC ← PC + 4** se necesita:
 - ▶ El mismo registro **PC**
 - ▶ Una ALU
 - ▶ No es necesario un registro que almacene el valor 4. Simplemente lo cableamos

Camino de Datos para Fetch (2/5)

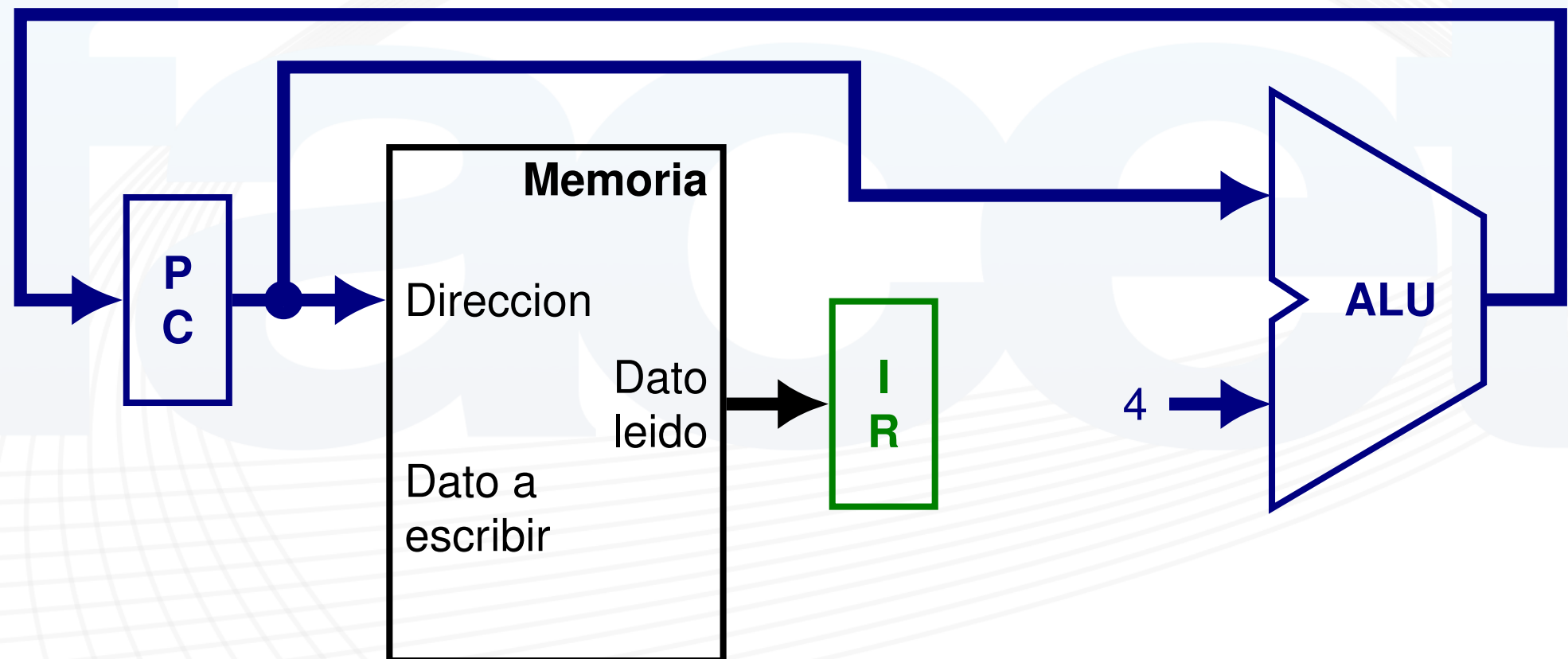
- ▶ Los resultados de las operaciones deben escribirse en elementos de estado, para mantener su valor
- ▶ El resultado de **PC + 4** se escribe en el mismo **PC**
- ▶ Es necesario tener un lugar donde almacenar temporalmente la salida de la memoria
 - ▶ Que sería la instrucción a decodificar y ejecutar
- ▶ Se agrega un registro adicional, denominado Instrucción Register (**IR**)
 - ▶ El retardo de lectura y escritura del registro será despreciable
- ▶ Este registro adicional es necesario para la implementación, pero no forma parte del ISA en sí
 - ▶ En la materia que sigue verán otras implementaciones que no lo tienen

Camino de Datos para Fetch (3/5)

- ▶ Por otra parte, las dos acciones del Fetch...
 - ▶ ...¿deberían hacerse en dos ciclos diferentes?
 - ▶ ...¿o podrían hacerse en el mismo ciclo de reloj?
- ▶ Ambas opciones son posibles, con sus ventajas y desventajas
- ▶ En nuestra implementación, se harán en el mismo ciclo de reloj
 - ▶ Porque no hay una secuencialidad entre ambas, entonces es posible hacerlas en simultáneo
 - ▶ Porque ambas necesitan un único valor, contenido en el mismo registro de origen (**PC**)
 - ▶ Asumimos también que los retardos de la memoria ideal y de la ALU son similares (200 ps cada uno, por ejemplo)

Camino de Datos para Fetch (4/5)

- ▶ Con las consideraciones anteriores, interconectamos los elementos definidos de la siguiente manera:



Camino de Datos para Fetch (5/5)

- ▶ El dibujo anterior muestra el camino de datos necesario para realizar el fetch definido en el RTN Abstracto:

$$M[PC] \text{ y } PC \leftarrow PC + 4$$

- ▶ Pero el camino de datos es solamente la conexión de los distintos componentes, no hace nada por sí solo
- ▶ Lo que realmente hará que en ese camino de datos se produzca un fetch son los valores de las señales de control de los componentes
 - ▶ Hay que activar la señal de lectura de la memoria
 - ▶ Hay que indicarle a la ALU que sume
 - ▶ Hay que habilitar la señal de escritura del registro **PC**, para que se escriba al finalizar el ciclo
 - ▶ Hay que habilitar la señal de escritura del registro auxiliar **IR**
- ▶ Volveremos sobre esto más adelante

Camino de Datos para Tipo R (1/6)

- ▶ Continuamos con el RTN de la instrucción **add**
$$\mathbf{R[rd] \leftarrow R[rs1] + R[rs2]}$$
- ▶ Consta de cuatro acciones distintas
- ▶ ¿Qué componentes se necesitan?
- ▶ Para buscar **R[rs1]** y **R[rs2]** se necesita un banco de registros
- ▶ Para sumarlos, se necesita una ALU
- ▶ Para guardar el resultado en **R[rd]**, se necesita el mismo banco de registros

Camino de Datos para Tipo R (2/6)

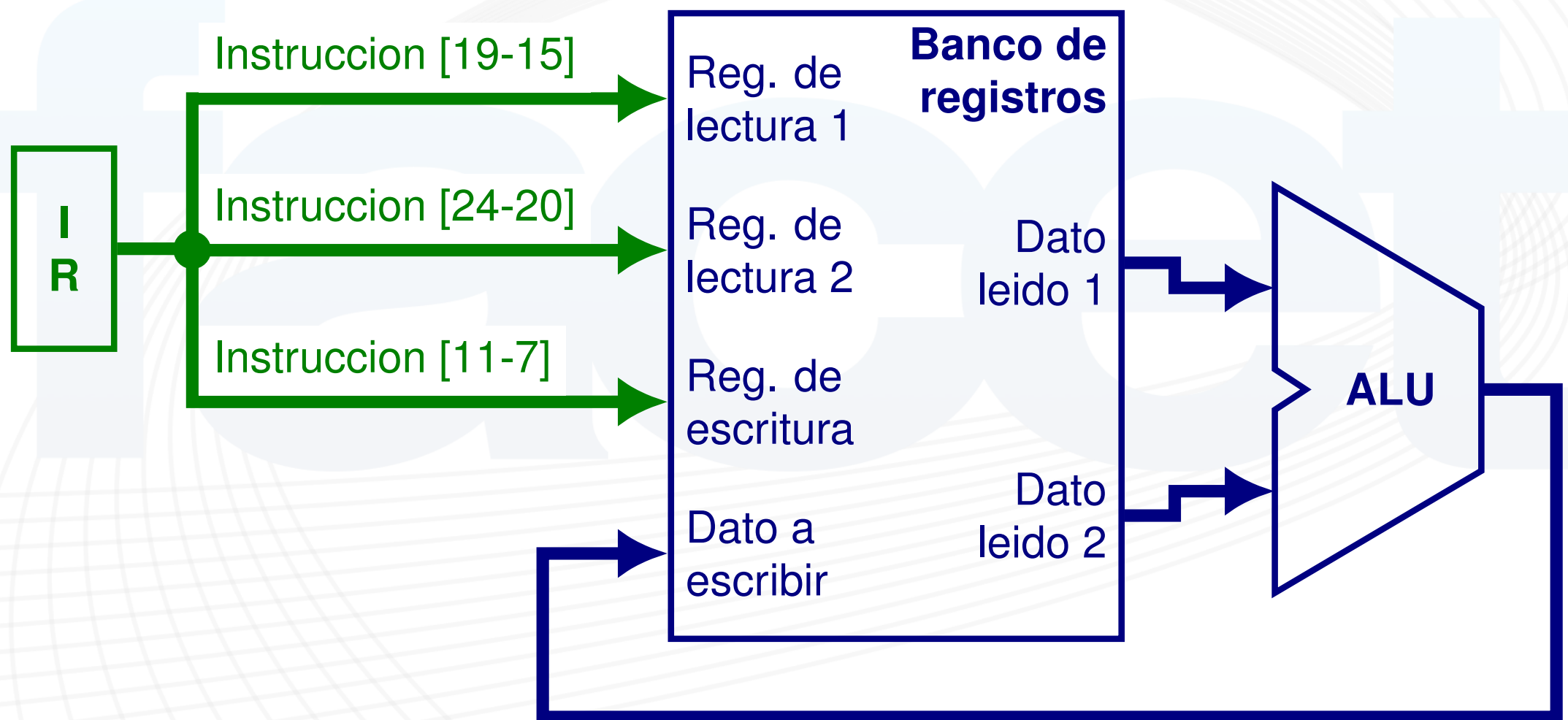
- ▶ Las entradas rs1, rs2 y rd del banco de registros provienen del registro IR
 - ▶ Están especificadas en campos fijos, según el formato de la instrucción
 - ▶ Siempre deben provenir de un elemento de estado
- ▶ Usaremos un banco de registros con dos puertos de lectura y uno de escritura, como el visto
 - ▶ Esto permite simplificar las conexiones y no agregar registros temporales adicionales
 - ▶ Supondremos que el banco de registros tiene un retardo de 100 ps, tanto para lectura como para escritura
- ▶ La ALU utilizada será la misma que ya tenemos para el Fetch
 - ▶ Se reutilizan los recursos
 - ▶ En distintos ciclos la misma ALU se utilizará para distintas partes de la misma instrucción
 - ▶ Habrá que tener cuidado con las conexiones

Camino de Datos para Tipo R (3/6)

- ▶ Las cuatro acciones necesarias, ¿se pueden hacer en un solo ciclo? ¿O necesitaremos más?
- ▶ Nuevamente, hay varias opciones posibles, con sus ventajas y desventajas
- ▶ Primer intento: hacer las cuatro acciones en un solo ciclo de reloj

Camino de Datos para Tipo R (4/6)

- Interconectamos los elementos definidos de la siguiente manera, manteniendo su secuencialidad:

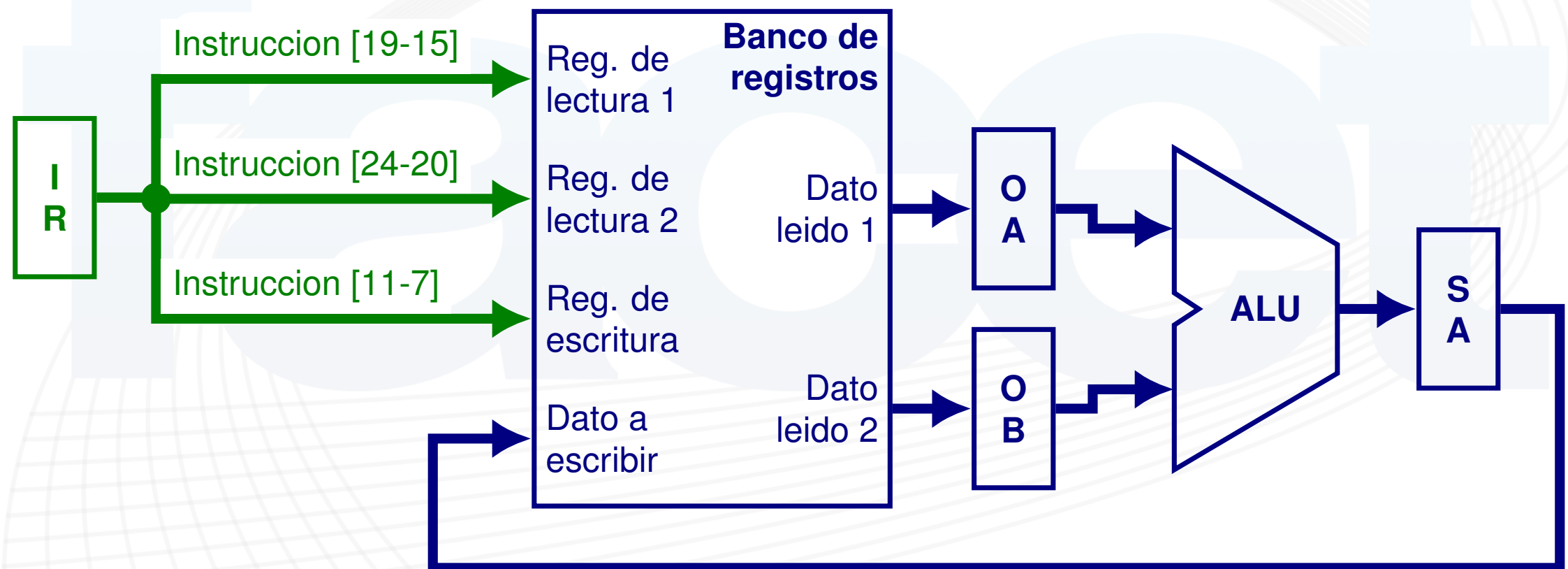


Camino de Datos para Tipo R (5/6)

- ▶ La principal desventaja de esta implementación es que la duración del ciclo de reloj debe incluir todos los retardos
 - ▶ Acceso al banco de registros para lectura de los dos operandos + retardo en la operación de la ALU + acceso al banco de registros para escritura del resultado
 - ▶ Retardo de la implementación en un solo ciclo: $100 + 200 + 100 = 400$ ps
 - ▶ Otra desventaja es que no es posible reutilizar los resultados intermedios
- ▶ Un mejor intento: hacerlas en tres ciclos de reloj
 - ▶ Aprovechando la secuencialidad entre ellas
 - ▶ Permite tener un ciclo de reloj más pequeño
 - ▶ Retardo de la implementación en tres ciclos: $\max(100, 200, 100) = 200$ ps
- ▶ Se necesitan agregar tres registros temporales para almacenar los valores entre los distintos ciclos
 - ▶ Dos a la salida del banco de registros, **0A** y **0B**, uno para cada valor leído
 - ▶ Uno a la salida de la ALU, denominado **SA**

Camino de Datos para Tipo R (6/6)

- ▶ Con las consideraciones anteriores, interconectamos los elementos definidos de la siguiente manera:

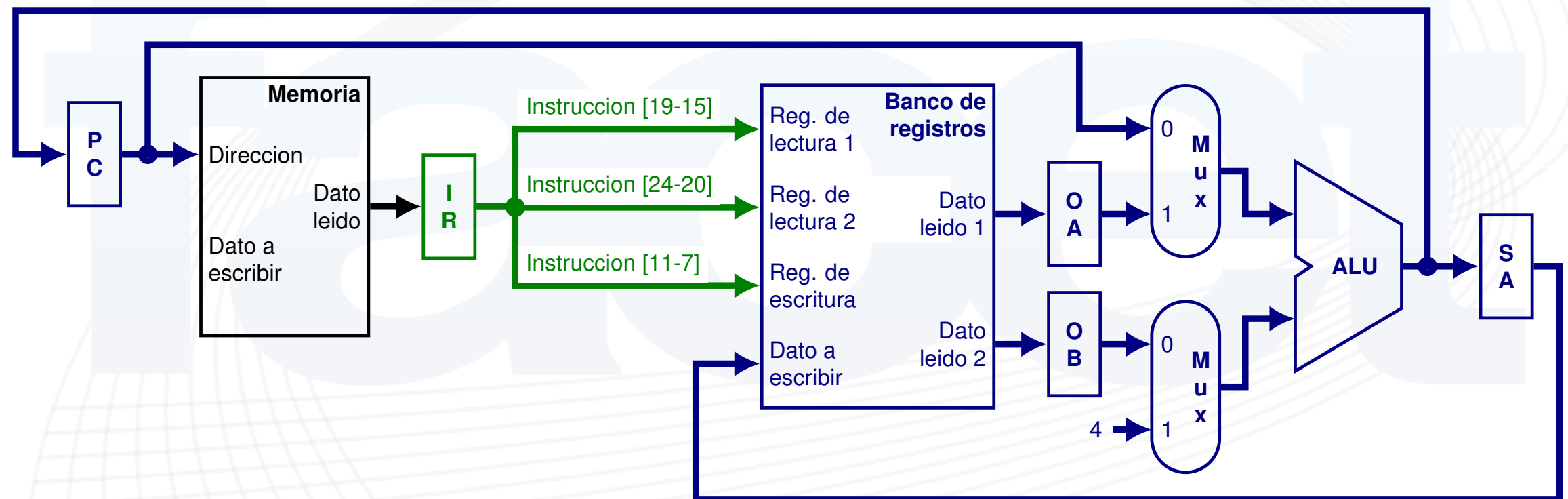


Camino de Datos Fetch + Tipo R (1/2)

- ▶ La reutilización de la ALU produce detalles adicionales
 - ▶ Porque hay múltiples fuentes para las entradas y múltiples destinos para la salida
- ▶ La entrada superior de la ALU puede recibir valores desde **PC** (en el Fetch) y desde el registro temporal **OA** (para el add)
- ▶ La entrada inferior de la ALU puede recibir valores desde un '4' cableado y desde el registro temporal **OB**
- ▶ Ambos casos se solucionan agregando dos multiplexores, cada uno antes de cada entrada de la ALU, denominados **FuenteA** y **FuenteB**
 - ▶ Asumiremos que los multiplexores tienen un retardo despreciable
- ▶ La salida de la ALU va al registro **PC** (en el Fetch) y al registro temporal **SA** (para el add)
 - ▶ No es un inconveniente, ya que ambos son elementos de estado, y no hay ningún problema que la misma señal llegue a ambos
 - ▶ No implica que necesariamente se escriben ambos. Dependerá de las señales de control

Camino de Datos Fetch + Tipo R (2/2)

- ▶ Con las consideraciones anteriores, interconectamos todos los elementos definidos de la siguiente manera:



Camino de Datos para sw (1/5)

- ▶ Continuamos con el RTN abstracto de la instrucción sw
$$\mathbf{M[R[rs1] + SignExt(imm)] \leftarrow R[rs2]}$$
- ▶ Consta de cuatro acciones distintas, y secuenciales
- ▶ ¿Qué componentes se necesitan?
- ▶ Para buscar **R[rs1]** y **R[rs2]** se necesita el banco de registros
- ▶ Necesitamos algo para generar la constante
- ▶ Para generar la dirección de memoria, se necesita la ALU para que sume
- ▶ Se necesita una memoria para guardar el dato

Camino de Datos para sw (2/5)

- ▶ La constante proviene de la instrucción, y está especificada según el formato
 - ▶ El campo **imm** se toma del registro **IR**, al igual que los campos **rs1** y **rs2**
- ▶ Es necesario agregar un bloque combinacional que se encargue de conformar la constante, para que pueda ingresar a la ALU
 - ▶ En este caso, extenderla en signo, desde 11 bits hasta los 32 necesarios
 - ▶ Para otros formatos, la conformación será más compleja, pero siempre combinacional
 - ▶ El retardo de este bloque es menor que el de otros componentes como la ALU o el banco de registros, y lo supondremos en 30 ps

Camino de Datos para sw (3/5)

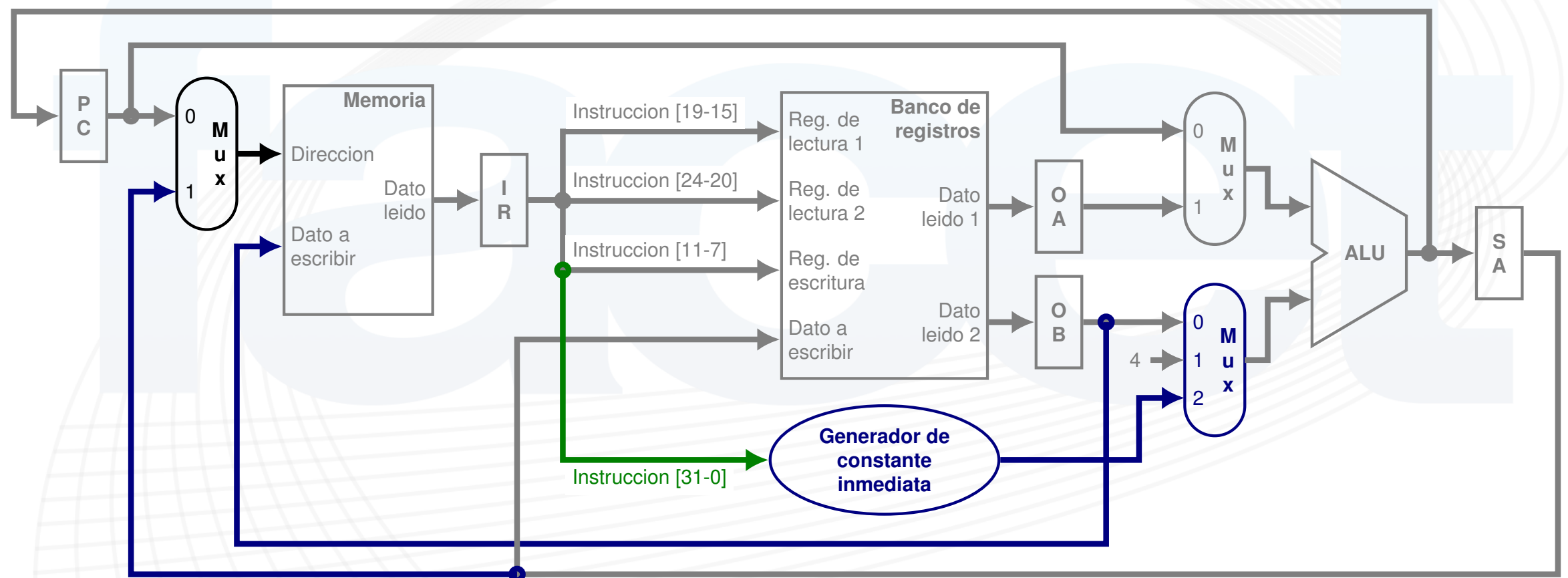
- ▶ El agregado del generador de constantes implica que es necesario agrandar el multiplexor **FuenteB**, ya que ahora hay tres posibles fuentes:
 - ▶ El valor '4' (en el Fetch), el registro temporal **OB** (en la instrucción **add**) y ahora el generador de constantes (en la instrucción **sw**)
- ▶ La memoria donde se guardará el dato, ¿será la misma que se utilizó para guardar las instrucciones?
- ▶ ¿Arquitectura Von Neumann o Harvard?

Camino de Datos para sw (4/5)

- ▶ En nuestra implementación, usaremos la misma memoria tanto para datos como para instrucciones
- ▶ Implica que es necesario agregar un nuevo multiplexor antes de la entrada de direcciones de la memoria, denominado **IoD**
 - ▶ La dirección de memoria puede provenir del registro **PC** (en el fetch) o del registro **SA** (en la instrucción **sw**)
- ▶ El dato a escribir en la memoria será el proveniente del registro temporal **OB**, por lo que simplemente se agrega esta conexión

Camino de Datos para sw (5/5)

- ▶ Con las consideraciones anteriores, interconectamos todos los elementos definidos de la siguiente manera:



Camino de Datos para lw (1/3)

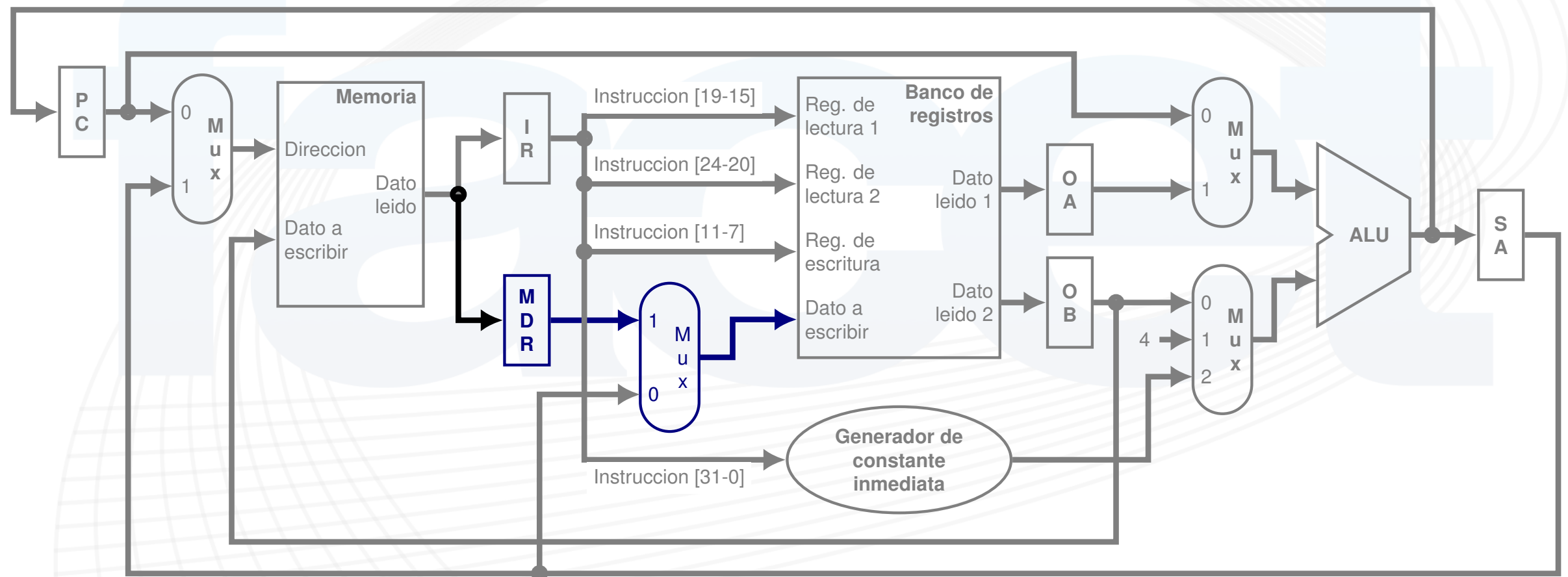
- ▶ Continuamos con el RTN abstracto de la instrucción lw
$$\mathbf{R[rd] \leftarrow M[R[rs1] + \text{SignExt}(imm)]}$$
- ▶ Consta de cuatro acciones distintas, y secuenciales
- ▶ ¿Qué componentes se necesitan?
- ▶ Para buscar **R[rs1]** se necesita el banco de registros, y el generador de constante para la constante inmediata
- ▶ Para generar la dirección de memoria, se necesita la ALU para que sume
- ▶ Se necesita la memoria para leer el dato buscado
- ▶ El dato leído se debe guardar en el banco de registros

Camino de Datos para lw (2/3)

- ▶ Al igual que se hizo anteriormente, para evitar acceder a memoria y escribir el dato leído en el banco de registros en el mismo ciclo, se agrega un registro temporal intermedio
 - ▶ Denominado Memory Data Register (**MDR**)
 - ▶ Almacena el valor leído de memoria, para ser utilizado en el ciclo siguiente
 - ▶ ¿No se podría simplemente reutilizar el **IR**? ¡NO! Porque el **IR** debe mantener su valor durante toda la ejecución de la instrucción
- ▶ Implica que es necesario agregar un nuevo multiplexor antes de la entrada de escritura del banco de registros, denominado **MemAReg**
 - ▶ El dato a escribir en el banco de registros puede provenir del registro **SA** (en la instrucción **add**) o del registro **MDR** (en la instrucción **lw**)

Camino de Datos para lw (3/3)

- ▶ Con las consideraciones anteriores, interconectamos todos los elementos definidos de la siguiente manera:



Camino de Datos para beq (1/3)

- ▶ Finalmente, queda el RTN abstracto de la instrucción beq
Si ($R[rs1] == R[rs2]$) $PC \leftarrow PC + \text{SignExt}(imm)$
- ▶ Consta de varias acciones distintas, algunas secuenciales y algunas que dependen de una decisión
- ▶ ¿Qué componentes se necesitan?
- ▶ Para buscar **$R[rs1]$** y **$R[rs2]$** se necesita el banco de registros, y el generador de constante para la constante inmediata
- ▶ Para generar la dirección de la próxima instrucción (salte o no), se necesita la ALU para que sume
- ▶ Necesitaremos algo para hacer la comparación entre los registros

Camino de Datos para beq (2/3)

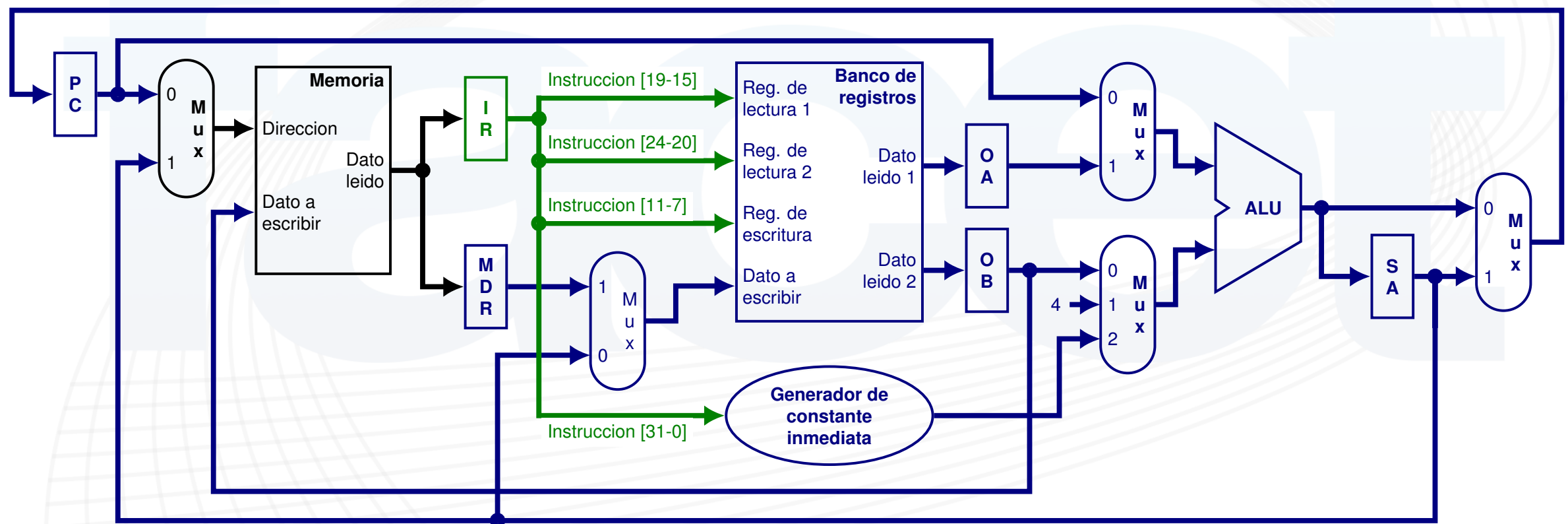
- ▶ Los registros a comparar son leídos del banco de registros y almacenados en los registros temporales **A** y **B**
- ▶ Esto permite que en el ciclo siguiente, la ALU pueda ser utilizada para compararlos
 - ▶ Se los resta, y la ALU posee una salida adicional para indicar si el resultado de una operación es igual a cero.
 - ▶ No es necesario agregar componentes adicionales para la comparación

Camino de Datos para beq (3/3)

- ▶ Mientras se buscan los registros en el banco de registros, la ALU estaría desocupada
 - ▶ La constante inmediata sí está disponible, ya que al igual que el banco de registros toma los datos necesarios del registro **IR**
 - ▶ Es posible hacer que en simultáneo la ALU calcule la dirección de salto, porque tiene los datos disponibles, y la almacene en el registro temporal **SA**
- ▶ Luego, en el ciclo siguiente, según el resultado de la comparación, se decidirá si la dirección calculada que está en el registro temporal **SA** será escrita en el registro **PC** o no
- ▶ Entonces, solamente hace falta agregar al camino de datos un multiplexor antes del registro **PC**, llamado **FuentePC**
 - ▶ Permite seleccionar si el registro **PC** toma el siguiente valor desde la ALU (en el Fetch) o desde el registro **SA** (en la instrucción **beq**)

Camino de Datos completo

- ▶ Con las consideraciones anteriores, interconectamos todos los elementos definidos de la siguiente manera:



RTN Concreto (1/9)

- ▶ El RTN abstracto nos detalla qué es lo que hace una instrucción, pero no cómo lo hace, porque eso es dependiente de la implementación
- ▶ Una vez definido un camino de datos, es posible especificar realmente cómo se ejecuta una instrucción en ese camino
- ▶ Esto se denomina RTN concreto
- ▶ Es simplemente formalizar un poco las secuencias de acciones vistas

RTN Concreto (2/9)

- ▶ Por ejemplo, utilizando como ejemplo la instrucción add:

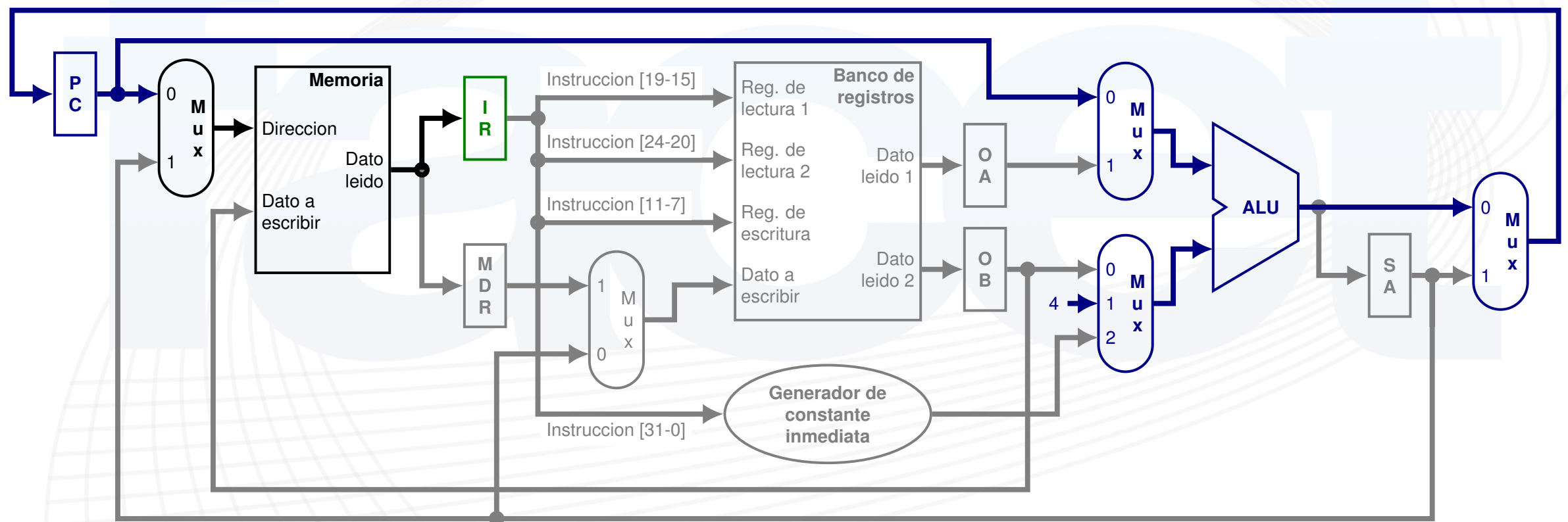
Instrucción	RTN Abstracto
add rd, rs1, rs2	M[PC]: PC $\leftarrow$ PC + 4: R[rd] $\leftarrow$ R[rs1] + R[rs2];

Instrucción	RTN concreto (Multiciclo)
add rd, rs1, rs2	IR $\leftarrow$ M[PC]: PC $\leftarrow$ PC + 4; 0A $\leftarrow$ R[IR<19:15>]: 0B $\leftarrow$ R[IR<24:20>]; SA $\leftarrow$ 0A + 0B; R[IR<11:7>] $\leftarrow$ SA;

- ▶ Cada renglón del RTN concreto representa un paso de la ejecución, y se ejecuta en un ciclo de reloj
- ▶ Se usan los registros temporales y se especifican los bits de los campos
- ▶ Una línea de RTN abstracto puede convertirse en varias líneas de RTN concreto: la instrucción **add** se ejecuta en 4 ciclos de reloj

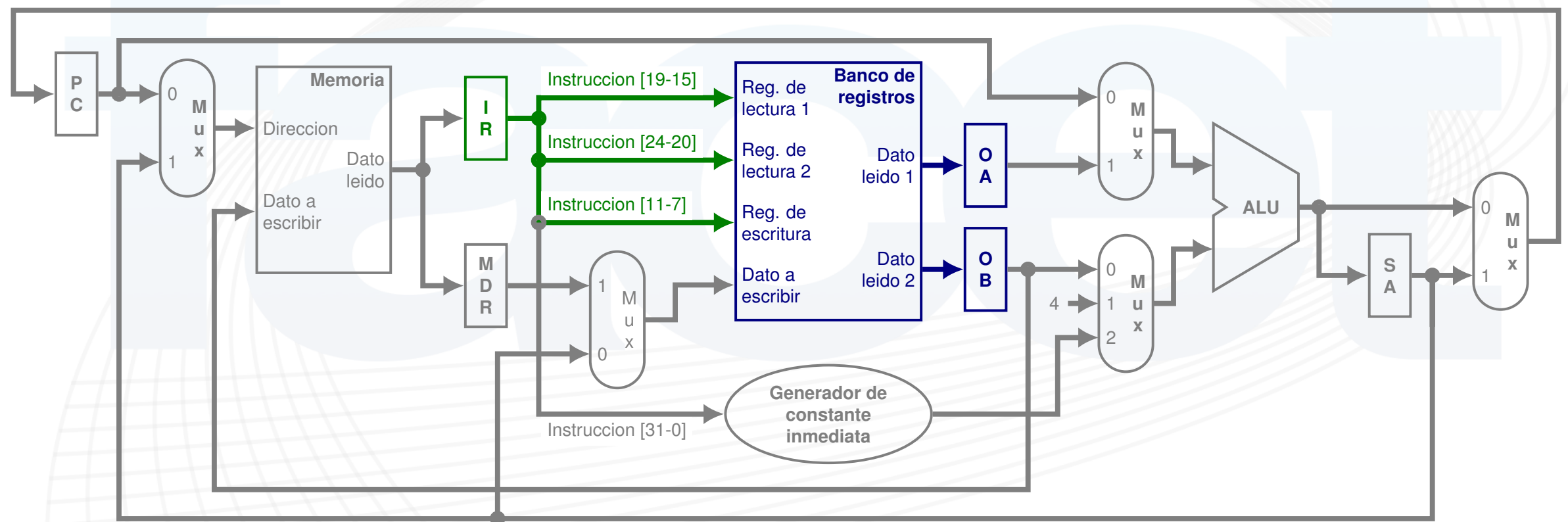
RTN Concreto (3/9)

- Primer paso para la instrucción **add**:



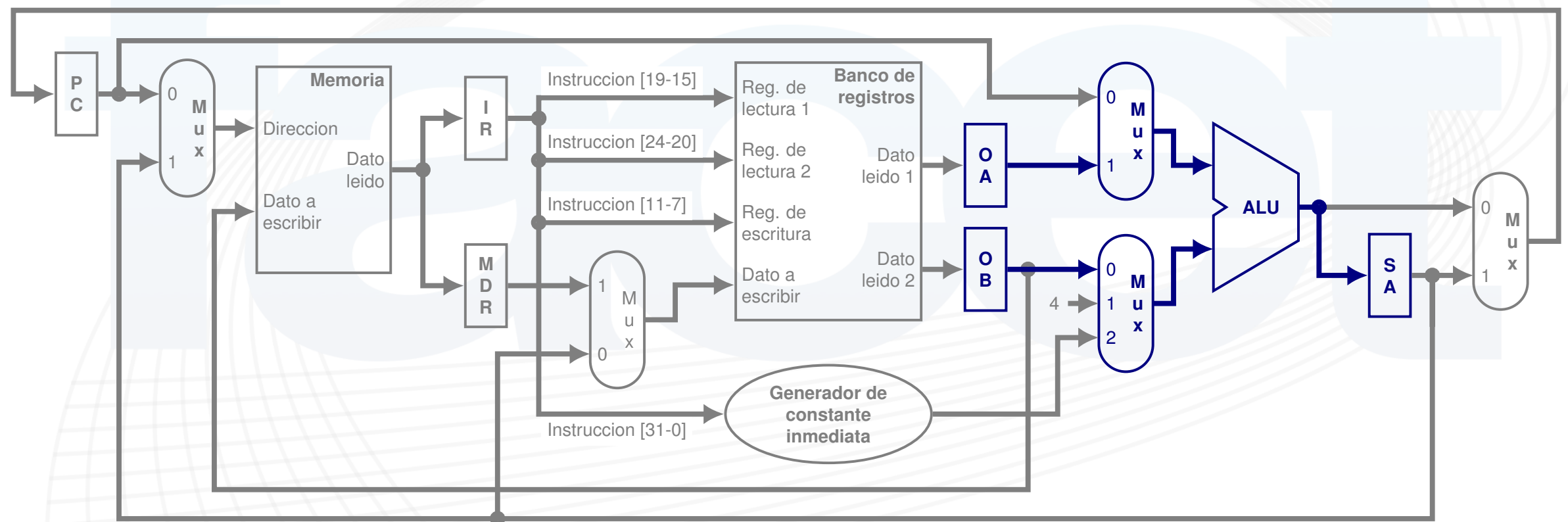
RTN Concreto (4/9)

- Segundo paso para la instrucción **add**:



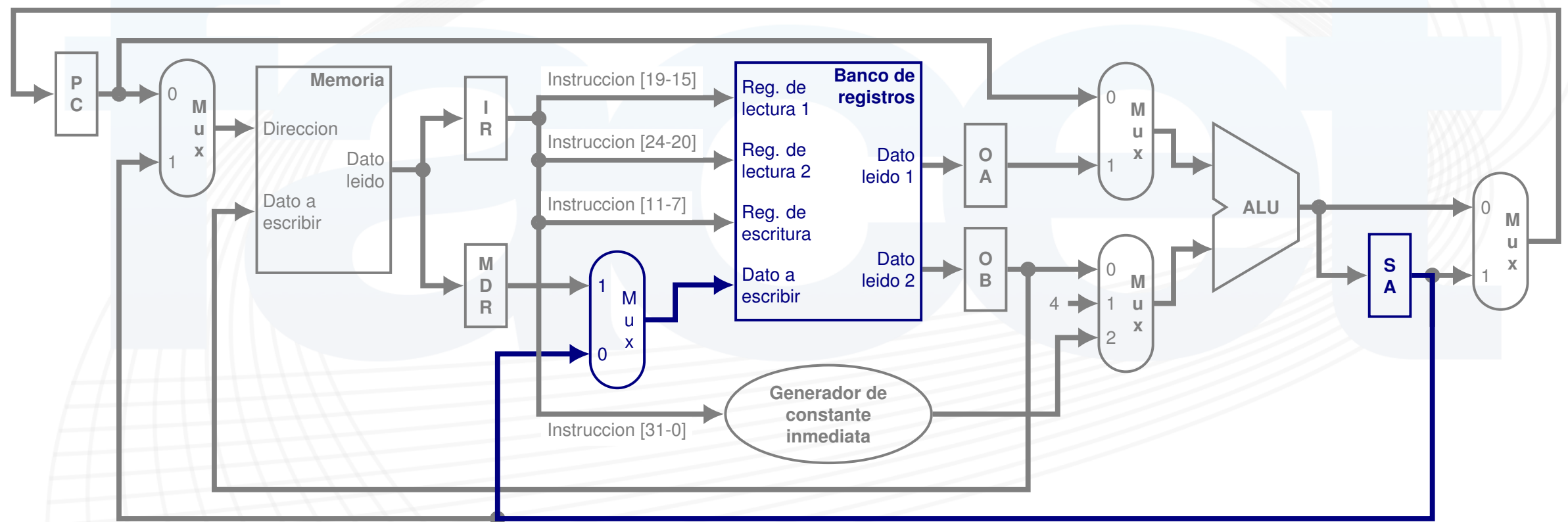
RTN Concreto (5/9)

- ▶ Tercer paso para la instrucción **add**:



RTN Concreto (6/9)

- Último paso para la instrucción **add**:



RTN Concreto (7/9)

- ▶ Repitiendo para las demás instrucciones, **sw** también demora 4 ciclos, mientras que **beq** demora 3 ciclos y **lw** demora 5 ciclos
- ▶ Los dos primeros ciclos son comunes para todas las instrucciones
- ▶ No importa que se calculen valores que después no se utilicen o se sobrescriban

RTN Concreto (8/9)

- ▶ Si agrupamos los RTN concretos de todas las instrucciones, podemos escribir la siguiente tabla

Ciclo	tipo R	lw	sw	beq
1	$IR \leftarrow M[PC]; PC \leftarrow PC + 4;$			
2	$OA \leftarrow R[IR<19:15>]; OB \leftarrow R[IR<24:20>]; SA \leftarrow PC + imm(IR);$			
3	$SA \leftarrow OA \text{ op } OB;$	$SA \leftarrow OA + imm(IR);$	$if (OA == OB) PC \leftarrow SA;$	
4	$R[IR<11:7>] \leftarrow SA;$	$MDR \leftarrow M[SA];$	$M[SA] \leftarrow OB;$	
5		$R[IR<11:7>] \leftarrow MDR;$		

- ▶ Como vimos cuando hicimos el fetch, cada uno de estos ‘pasos’ debe convertirse en las señales de control de cada componente para que efectivamente se realice la acción deseada
- ▶ En el tema siguiente, haremos un mapeo 1:1 entre cada uno de estos pasos y las señales de control correspondientes

RTN Concreto (9/9)

- ▶ El ciclo de reloj de nuestra implementación estará determinado por el más lento de los 'pasos' de la tabla anterior
- ▶ Recordando las suposiciones realizadas para los retardos de los componentes, ¿cuál de todos los 'pasos' es el que más demora?
 - ▶ Memoria y ALU, 200 ps. Banco de registros, 100 ps
- ▶ La implementación propuesta lleva a que la duración de los 'pasos' sea bastante balanceada
 - ▶ Y además, mínima.
- ▶ ¿Cuánto demorará una instrucción **lw**?

Ejemplo de un programa

- ▶ Programa

```
0x0000.0000 0x1000.2083 lw x1, 0x0100(x0)
0x0000.0004 0x1040.2183 lw x2, 0x0104(x0)
0x0000.0008 0x0020.80B3 add x1, x1, x2
0x0000.000C 0x1010.2423 sw x1, 0x0108(x0)
```

- ▶ Datos

```
0x0000.0100 0x0000.0002 .word 2
0x0000.0104 0x0000.0003 .word 3
0x0000.0108 0x????.???? .space 4
```

- ▶ El programa y los datos están en la memoria

Ejemplo de un programa

► **lw** **x1**, **0x0100(x0)**

T0	IR $\leftarrow$ M[PC]:	IR = 0x1000.2083
	PC $\leftarrow$ PC + 4;	PC = 0x0000.0004
T1	0A $\leftarrow$ R[IR<19:15>]:	0A = 0x0000.0000
	0B $\leftarrow$ R[IR<24:20>]:	0B = 0x????.????
	SA $\leftarrow$ PC + imm(IR);	SA = 0x0000.0904
T2	SA $\leftarrow$ 0A + imm(IR);	SA = 0x0000.0100
T3	MDR $\leftarrow$ M[SA];	MDR = 0x0000.0002
T4	R[IR<11:7>] $\leftarrow$ MDR;	R1 = 0x0000.0002

Ejemplo de un programa

► **lw** **x1**, **0x0104(x0)**

T5	IR $\leftarrow$ M[PC] ;	IR = 0x1040.2183
	PC $\leftarrow$ PC + 4;	PC = 0x0000.0008
T6	0A $\leftarrow$ R[IR<19:15>] ;	0A = 0x0000.0000
	0B $\leftarrow$ R[IR<24:20>] ;	0B = 0x????.????
	SA $\leftarrow$ PC + imm(IR) ;	SA = 0x0000.094A
T7	SA $\leftarrow$ 0A + imm(IR) ;	SA = 0x0000.0104
T8	MDR $\leftarrow$ M[SA] ;	MDR = 0x0000.0003
T9	R[IR<11:7>] $\leftarrow$ MDR ;	R1 = 0x0000.0003

Ejemplo de un programa

► **add x1, x1, x2**

T10 $IR \leftarrow M[PC]:$

IR = 0x0020.80B3

$PC \leftarrow PC + 4;$

PC = 0x0000.000C

T11 $0A \leftarrow R[IR<19:15>]:$

0A = 0x0000.0002

$0B \leftarrow R[IR<24:20>]:$

0B = 0x0000.0003

$SA \leftarrow PC + imm(IR);$

SA = 0x0000.080C

T12 $SA \leftarrow 0A \text{ op } 0B;$

SA = 0x0000.0005

T13 $R[IR<11:7>] \leftarrow SA;$

R1 = 0x0000.0005

Ejemplo de un programa

► **SW** **x1**, **0x0108(x0)**

T14 **IR** $\leftarrow$ **M[PC]** : **IR** = **0x1010.2423**

PC $\leftarrow$ **PC** + 4; **PC** = **0x0000.0010**

T15 **0A** $\leftarrow$ **R[IR<19:15>]** : **0A** = **0x0000.0000**

0B $\leftarrow$ **R[IR<24:20>]** : **0B** = **0x0000.0005**

SA $\leftarrow$ **PC** + **imm(IR)**; **SA** = **0x0000.094A**

T16 **SA** $\leftarrow$ **0A** + **imm(IR)**; **SA** = **0x0000.0108**

T17 **M[SA]** $\leftarrow$ **0B**; **M[0x0000.0108]** = **0x0000.0005**

Ejemplo de un programa

- ▶ En el programa de ejemplo hay 4 instrucciones del ISA
- ▶ Para ejecutar ese programa en la arquitectura desarrollada se requieren 18 ciclos de reloj
- ▶ Se puede definir una metrica de cantidad de ciclos promedio por instrucción (CPI)
- ▶ En nuestro caso $CPI = 18 / 4 = 4.5$

Consideraciones finales

- ▶ El camino de datos diseñado posee todos los componentes necesarios, y solamente quedan pendientes las señales de control
 - ▶ Estas son generadas por la unidad de control cuando se decodifica la instrucción
 - ▶ Esto es el tema siguiente
- ▶ Siempre que fue posible, se trató de reutilizar componentes
 - ▶ Objetivo: minimizar costos
- ▶ Siempre que fue posible, se trató de usar solamente un componente “lento” en cada ciclo de reloj
 - ▶ Si se usó más de uno, es porque no hay secuencialidad entre ellos
 - ▶ Objetivo: minimizar duración del ciclo de reloj

Consideraciones finales

- ▶ Se agregaron cinco registros temporales, que no forman parte del ISA
 - ▶ Son particulares de esta implementación
- ▶ Que los formatos de instrucción sean pocos, y que sus campos sean regulares y ubicados siempre en los mismos lugares, ayuda mucho en el diseño del camino de datos
- ▶ Se necesitarán varios ciclos de reloj para ejecutar una sola instrucción
 - ▶ Por ello, una implementación como la realizada se conoce como Multiciclo
 - ▶ Distintos tipos de instrucciones necesitan distinta cantidad de ciclos para ejecutarse

Comentario extra

- ▶ La definición de los objetivos de diseño, la toma de decisiones entre distintas alternativas, la construcción del camino de datos y la evaluación de performance son conceptos que conforman lo que se conoce como “Arquitectura de Computadoras”
- ▶ Serán vistos con mucha mayor profundidad en una materia posterior