

El Microprocesador: Unidad de Control

Sistemas con Microprocesadores

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

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

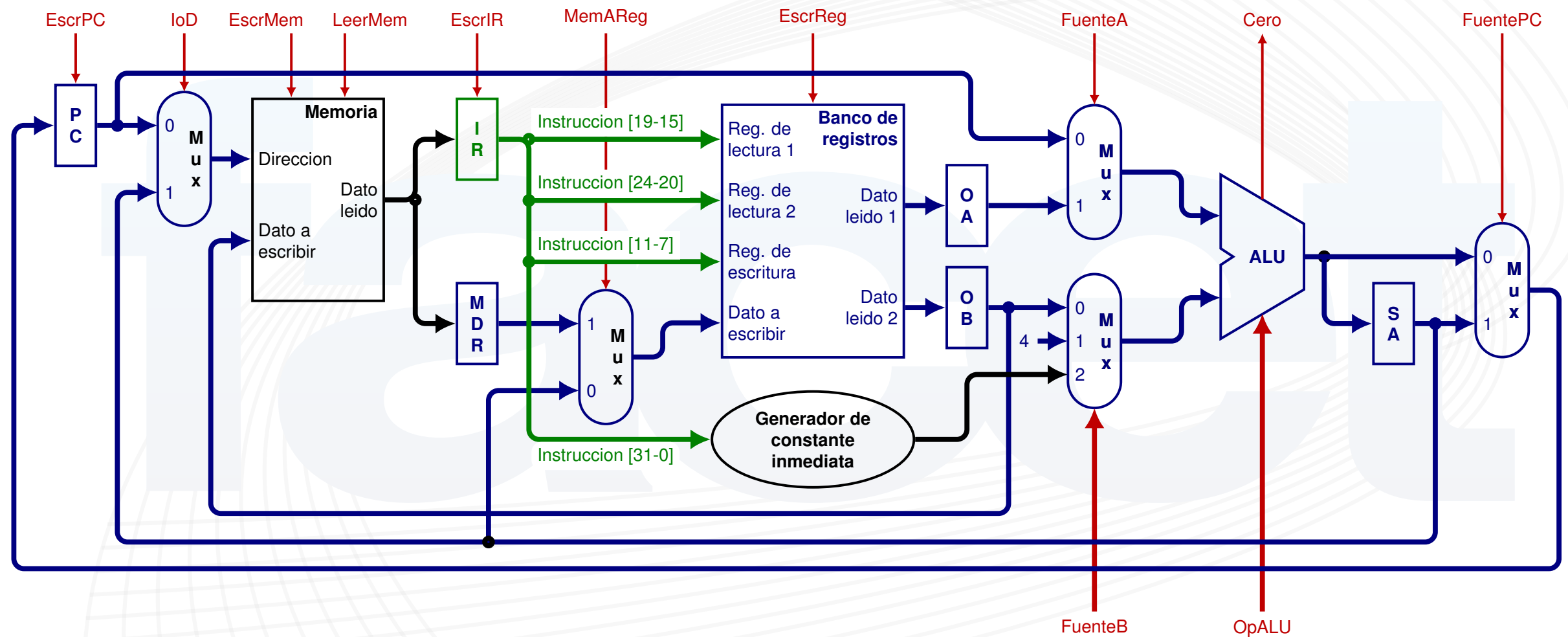
Cronograma

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

- ▶ Mejoras al camino de datos
 - ▶ Control de la ALU independiente
 - ▶ Mejoras sobre los saltos
- ▶ Diseño del Control Principal
 - ▶ Diagrama de la MEF
 - ▶ Mapeo de las señales de control
- ▶ Implementación con ROMs
 - ▶ Análisis de Transición de estados
- ▶ Implementación con Secuenciador
- ▶ Microprogramación

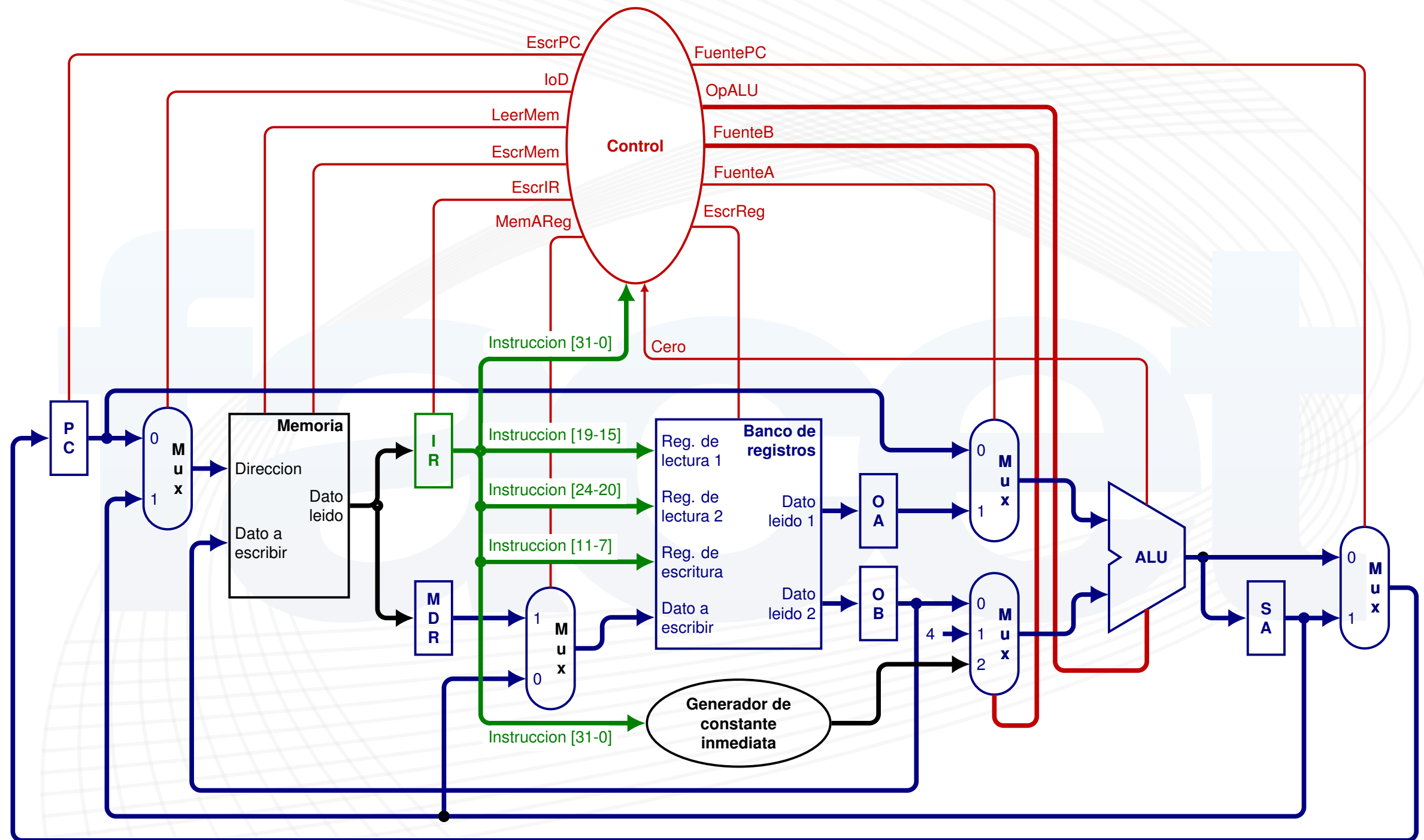
Camino de datos completo



Objetivo

- ▶ En el tema anterior quedaron pendientes las 11 señales de control
 - ▶ Son generadas por la unidad de control cuando se decodifica la instrucción
- ▶ Diseñaremos una unidad de control que se encargue de decodificar las instrucciones y generar las señales de control correspondientes

Camino de datos



Observaciones control de la ALU

- ▶ Una de las señales es la del control de la ALU
 - ▶ Típicamente, es una señal de 4 bits
- ▶ Las funciones que usa nuestra implementación son:
 - ▶ Para el fetch, sumar
 - ▶ Para **lw** y **sw**, sumar
 - ▶ Para **beq**, sumar y restar
 - ▶ Para las operaciones aritméticas (**add**, **sub**, **and** y **or**), está determinado en los campos función de dichas instrucciones (**funct7** y **funct3**, 10 bits en total)

Observaciones control de la ALU

- ▶ A su vez, la función que realiza la ALU depende de la instrucción
 - ▶ Con la excepción del fetch
 - ▶ Por lo que las señales de control de la ALU podrían generarse posteriormente a la decodificación de la instrucción
- ▶ Decisión de diseño: independizar el control de la ALU del control principal

Control de la ALU independiente

▶ Ventajas:

- ▶ Modularidad
- ▶ Reduce el tamaño del control principal al no tener que pasarle como entradas los campos función
- ▶ Reduce el retardo del control principal

▶ Desventajas:

- ▶ Diseñar dos controles

Entradas del control de la ALU

- ▶ Un campo de dos bits, denominado **OpALU**, proveniente del control principal
 - ▶ Al decodificar la instrucción, se sabe qué función debe realizar la ALU
 - ▶ Las funciones mencionadas se agrupan y codifican de la siguiente manera:
 - ▶ '00' para sumar, '01' para restar, '10' para hacer lo que especifiquen los campos función
- ▶ 10 bits correspondientes a los campos función provenientes de la instrucción (**funct7** y **funct3**)

Salidas del control de la ALU

- ▶ Los 4 bits necesarios para especificar la función de la ALU
- ▶ Estrictamente, dependen de la ALU y se toman de su hoja de datos o de su especificación
- ▶ Para implementar el set completo de RV32I, se necesitan las siguientes funciones, con la siguiente codificación

Bits de control	Función de la ALU
0000	AND
0001	OR
0010	Sumar
0110	Restar
0111	Slt
1100	NOR
Otra	Cero

Control de la ALU

- ▶ Como las salidas dependen solo de las señales de entrada, este control de la ALU será un circuito combinacional
- ▶ La tabla de verdad es la siguiente:

Instrucción	opcode	OpALU	funct7	funct3	ALUCtrl
add	tipo r	10	0000000	000	0010
sub	tipo r	10	0100000	000	0110
and	tipo r	10	0000000	111	0000
or	tipo r	10	0000000	110	0001
lw	lw	00	XXXXXXXX	XXX	0010
sw	sw	00	XXXXXXXX	XXX	0010
beq	beq	01	XXXXXXXX	XXX	0110

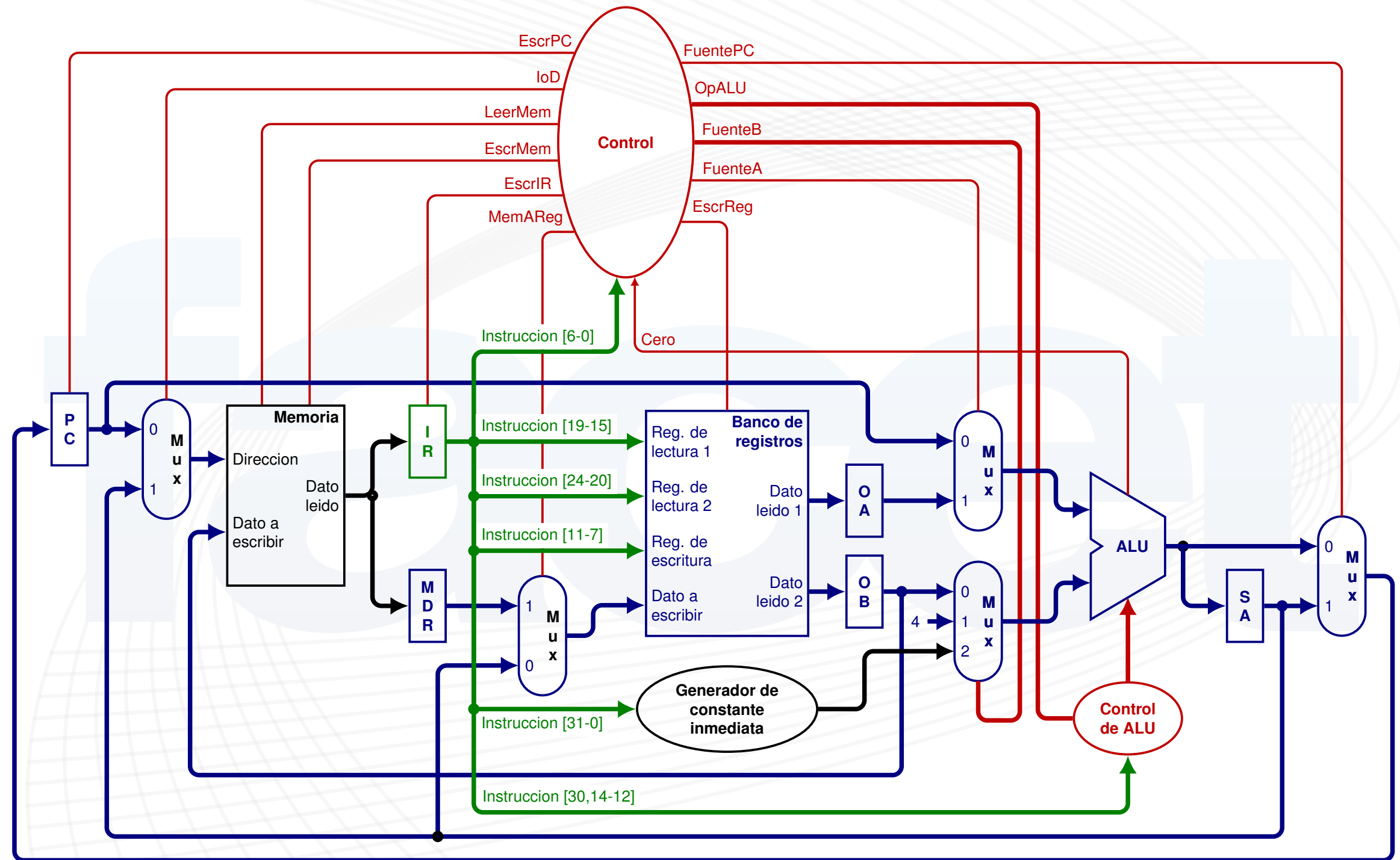
Control de la ALU

- ▶ En la tabla verdad anterior, se omitieron muchísimas entradas, porque son condiciones de indiferencia
- ▶ Para el subconjunto de instrucciones, solamente varía un bit del campo **funct7** y los tres bits del campo **funct3**
- ▶ Por lo que solamente son necesarios 4 bits desde el **IR**, y no los 10 en total

Implementación del control de la ALU

- ▶ La manera más simple y eficiente de implementar un circuito combinacional es mediante una lógica de dos niveles: como una suma de productos
- ▶ Una fila de compuertas AND conectadas a las 6 entradas, generando los minterms correspondientes
- ▶ Una columna de compuertas OR seleccionando los minterms correspondientes y generando las salidas

Camino de datos mejorado



Observaciones sobre la señal EscrPC

- ▶ En el camino de datos anterior quedan 11 señales de control por especificar
- ▶ Una de ellas es la señal de control **EscrPC**, que debe tomar un valor igual a '**1**' cuando sea necesario modificar el contenido del registro **PC**
- ▶ Esto sucede en dos ocasiones:
 - ▶ Durante el fetch de una instrucción
 - ▶ Durante la ejecución de la instrucción **beq**
 - ▶ Pero solamente si los registros que se comparan son iguales, es decir, si se activa la salida **Cero** de la ALU

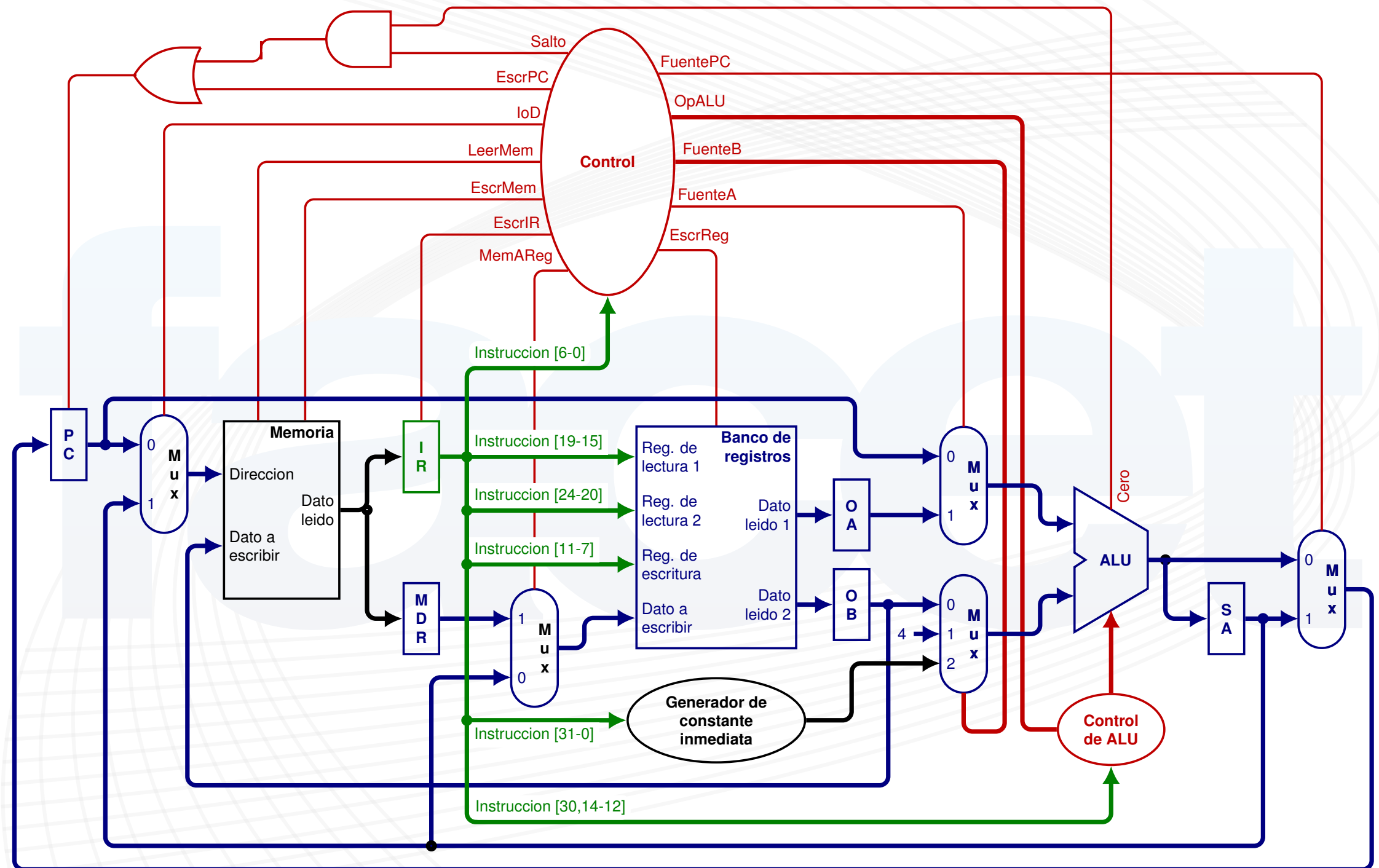
Observaciones sobre la señal EscrPC

- ▶ Esta condición representa una situación particular:
 - ▶ Una señal de control depende del resultado de una operación de la ALU
 - ▶ Es la condición “IF” expresada en el RTN concreto
- ▶ Por esto es que la señal **Cero** de la ALU es una entrada de la unidad de control
- ▶ Es posible quitar esta señal como entrada del control
 - ▶ Deja únicamente los datos de la instrucción del **IR**
 - ▶ Simplifica el diseño del control al tener menos entradas

Mejoras sobre la señal EscrPC

- ▶ Se realiza agregando una nueva señal de control al camino de datos, denominada **Salto**
 - ▶ Esta señal solamente se activará cuando la instrucción que se decodifique sea una instrucción de salto, en este caso **beq**
 - ▶ Será una nueva salida de la Unidad de Control, ahora serán 12
- ▶ Se agrega al camino de datos una compuerta AND para evaluar si la instrucción es un salto y si el resultado de la comparación de los registros es cero (**Salto AND Cero**)
- ▶ Se agrega al camino de datos una compuerta OR ya que el registro **PC** se debe escribir cuando se active la señal de control **EscrPC** o cuando sea una instrucción de salto y los registros sean iguales (**EscrPC OR (Salto AND Cero)**)

Camino de datos mejorado



Diseño del control principal

- ▶ A diferencia del control para la ALU, las señales de control principales no dependen únicamente de la instrucción
- ▶ ¿Qué debe hacer la ALU en una instrucción add?
 - ▶ La respuesta correcta es otra pregunta: ¿en qué ciclo?
- ▶ Luego, las señales de control principales dependen de la instrucción actual y del ciclo que se está ejecutando de esa instrucción (estado actual)
- ▶ El control principal será secuencial y no combinacional
- ▶ Se utilizará una Máquina de Estados Finitos (MEF)

Repaso: RTN Concreto

- Dados los 'pasos' del RTN concreto vistos en el tema anterior:

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 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;		

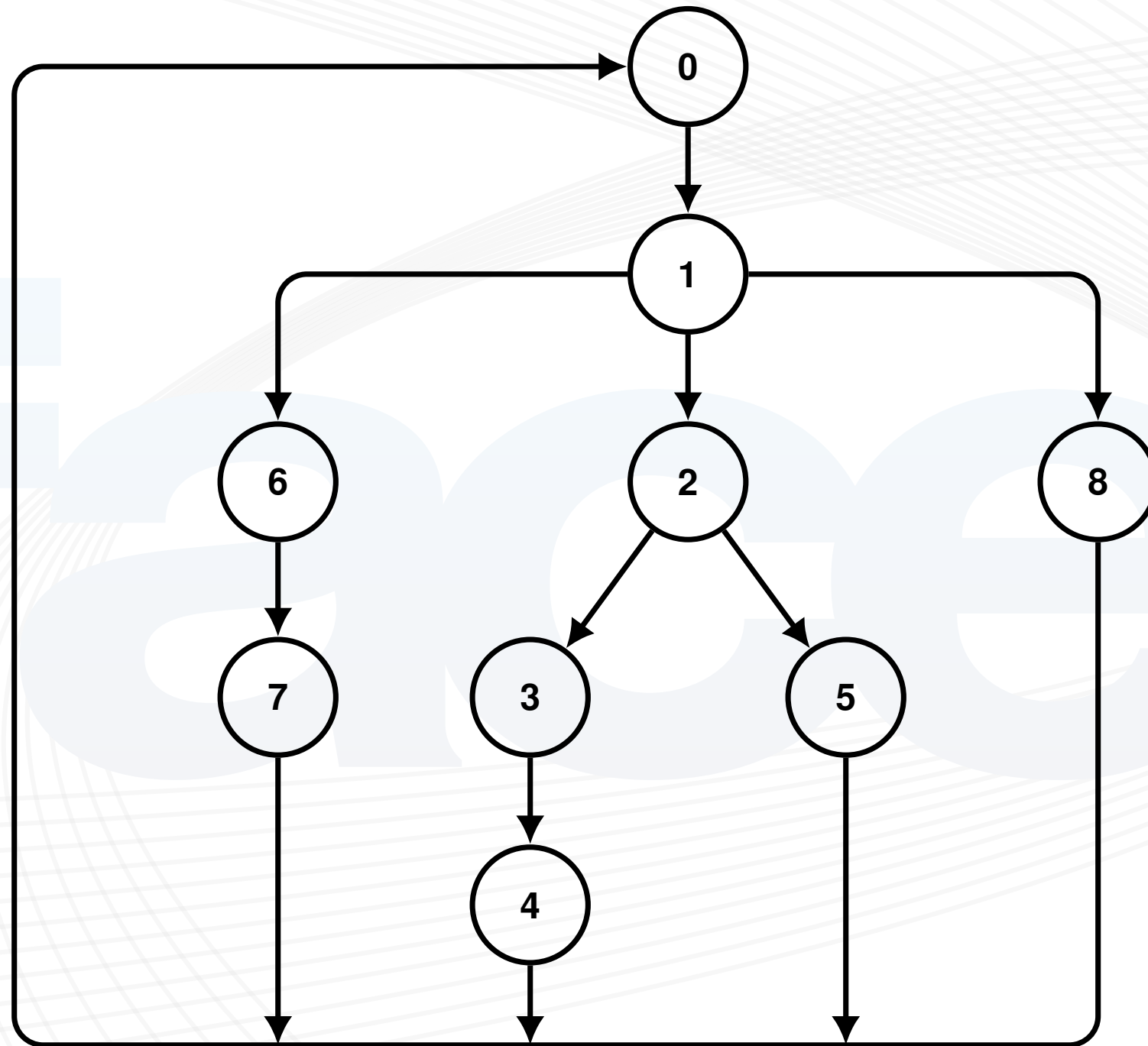
- Cada uno de estos 'pasos' será una combinación diferente de señales de control, con un mapeo 1:1
- Por lo tanto, cada uno de estos 'pasos' será un estado diferente de la MEF de control

MEF de Control (1/14)

- Se asigna un número a cada uno de los estados

Ciclo	tipo R	lw	sw	beq
1	⁰ $IR \leftarrow M[PC]; PC \leftarrow PC + 4;$			
2	¹ $OA \leftarrow R[IR<19:15>]; OB \leftarrow R[IR<24:20>]; SA \leftarrow PC + imm(IR);$			
3	⁶ $SA \leftarrow OA \text{ op } OB;$	² $SA \leftarrow OA + imm(IR);$	⁸ $if (OA == OB) PC \leftarrow SA;$	
4	⁷ $R[IR<11:7>] \leftarrow SA;$	³ $MDR \leftarrow M[SA];$	⁵ $M[SA] \leftarrow OB;$	
5		⁴ $R[IR<11:7>] \leftarrow MDR;$		

MEF de Control (2/14)



MEF de Control (3/14)

- ▶ Dos observaciones muy importantes del diagrama anterior:
- ▶ Cuando finaliza la ejecución de una instrucción, se retorna siempre al estado inicial (0, Fetch)
- ▶ Es una MEF estilo Moore, porque en cada estado las salidas dependen únicamente del mismo estado

MEF de Control (4/14)

- ▶ Si se analizan las señales de control en el estado 6 (ejecución de una instrucción tipo r)
 - ▶ RTN: **SA** $\leftarrow$ **0A** op **0B**;
- ▶ Señales de Control:
 - ▶ **FuenteA** = 1, para tomar **0A** como primer operando de la ALU
 - ▶ **FuenteB** = 00, para tomar **0B** como segundo operando de la ALU
 - ▶ **OpALU** = 10, para indicarle a la ALU que decida la operación según los campos función

MEF de Control (5/14)

- ▶ Se especifican solamente 3 de las 12 señales de control
 - ▶ Las necesarias para cumplir con el RTN concreto
 - ▶ ¿Qué se hace con las demás?
- ▶ Para cada estado, habrá señales de control que pueden tener un valor indiferente
 - ▶ Normalmente son señales que controlan multiplexores, cuyo valor seleccionado no afecta la ejecución del RTN en cuestión
 - ▶ Para el paso 6 que estamos viendo, son las señales **IoD**, **MemAReg** y **FuentePC**

MEF de Control (6/14)

- ▶ Pero hay otras señales de control que jamás pueden tomar un valor indiferente
 - ▶ Normalmente son señales que controlan la escritura (o lectura) de elementos de estado
 - ▶ Si el RTN no modifica alguno de los elementos de estado, entonces sus señales de control correspondientes deben tomar el valor '0'
 - ▶ Para el paso 6 que estamos viendo, son las señales **EscrPC, LeerMem, EscrMem, EscrIR, EscReg y Salto**

MEF de Control (7/14)

- Entonces es posible empezar a completar una tabla como la siguiente:

Estado	Escr PC	Salto	IoD	Escr Mem	Leer Mem	Escr IR	Escr Reg	Mem AReg	Fuente A	Fuente B	Op ALU	Fuente PC
0												
1												
2												
3												
4												
5												
6	0	0	X	0	0	0	0	X	1	00	10	X
7												
8												

MEF de Control (8/14)

- ▶ Para el estado 7 (escritura del resultado en tipo r)
 - ▶ RTN: **$R[IR<11:7>] \leftarrow SA;$**
- ▶ Señales de Control:
 - ▶ **MemAReg** = 0, para indicar que el dato a escribir en el banco de registros proviene de la salida de la ALU
 - ▶ **EscrReg** = 1, para indicar que debe escribirse el banco de registros
- ▶ El resto de señales no especificadas se comportan de la misma manera que para el estado anterior
 - ▶ **IoD, FuenteA, FuenteB, OpALU y FuentePC** = X
 - ▶ **EscrPC, LeerMem, EscrMem, EscrIR y Salto** = 0

MEF de Control (9/14)

- ▶ El estado 4 (escritura del resultado de una instrucción lw)
 - ▶ RTN: **$R[IR<11:7>] \leftarrow SA;$**
- ▶ Es idéntico que el estado 7, con una única diferencia:
 - ▶ **MemAReg** = 1, para indicar que el dato a escribir en el banco de registros proviene del registro MDR

MEF de Control (10/14)

► Continuamos completando la tabla anterior:

Estado	Escr PC	Salto	IoD	Escr Mem	Leer Mem	Escr IR	Escr Reg	Mem AReg	Fuente A	Fuente B	Op ALU	Fuente PC
0												
1												
2												
3												
4	0	0	X	0	0	0	1	1	X	XX	XX	X
5												
6	0	0	X	0	0	0	0	X	1	00	10	X
7	0	0	X	0	0	0	1	0	X	XX	XX	X
8												

MEF de Control (11/14)

- ▶ Para el estado 0 (Fetch)
 - ▶ RTN: **$IR \leftarrow M[PC]; PC \leftarrow PC + 4;$**
- ▶ Señales de Control:
 - ▶ **IoD** = 0, para indicar que se accederá a memoria usando a PC como dirección
 - ▶ **LeerMem** = 1, para hacer la lectura de memoria
 - ▶ **EscrIR** = 1, para escribir el registro **IR**
 - ▶ **FuenteA** = 0, para tomar **PC** como primer operando de la ALU
 - ▶ **FuenteB** = 01, para tomar la constante '4' como segundo operando de la ALU
 - ▶ **OpALU** = 00, para indicarle a la ALU que realice una suma
 - ▶ **FuentePC** = 0, para hacer que el resultado de la ALU se guarde en el **PC**
 - ▶ **EscrPC** = 1, para escribir el registro **PC**
- ▶ Al igual que antes, el resto de las señales no especificadas son indiferentes o toman el valor '0', según corresponda.

MEF de Control (12/14)

- ▶ Para el estado 1 (Decode)
 - ▶ RTN: $0A \leftarrow R[IR<19:15>]: 0B \leftarrow R[IR<24:20>]:$
 $SA \leftarrow PC + imm(IR);$
- ▶ Señales de Control:
 - ▶ **FuenteA** = 0, tomar PC como primer operando de la ALU
 - ▶ **FuenteB** = 10, tomar el valor inmediato como segundo operando de la ALU
 - ▶ **OpALU** = 00, para indicarle a la ALU que realice una suma
- ▶ Al igual que antes, el resto de las señales no especificadas son indiferentes o toman el valor '0', según corresponda

MEF de Control (13/14)

► La tabla se actualiza del siguiente modo:

Estado	Escr PC	Salto	IoD	Escr Mem	Leer Mem	Escr IR	Escr Reg	Mem AReg	Fuente A	Fuente B	Op ALU	Fuente PC
0	1	0	0	0	1	1	0	X	0	01	00	0
1	0	0	X	0	0	0	0	X	0	10	00	X
2												
3												
4	0	0	X	0	0	0	1	1	X	XX	XX	X
5												
6	0	0	X	0	0	0	0	X	1	00	10	X
7	0	0	X	0	0	0	1	0	X	XX	XX	X
8												

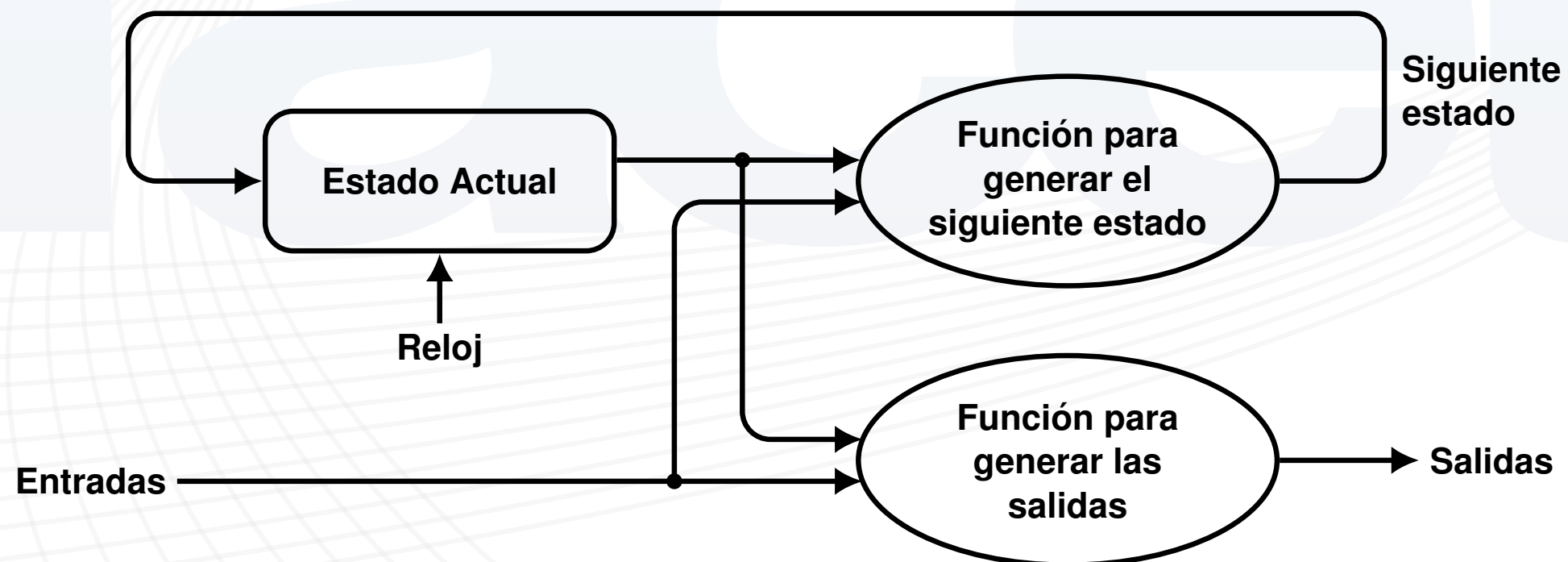
MEF de Control (14/14)

- Completando los estados restantes, la tabla quedará de la siguiente manera:

Estado	Escr PC	Salto	IoD	Escr Mem	Leer Mem	Escr IR	Escr Reg	Mem AReg	Fuente A	Fuente B	Op ALU	Fuente PC
0	1	0	0	0	1	1	0	X	0	01	00	0
1	0	0	X	0	0	0	0	X	0	10	00	X
2	0	0	X	0	0	0	0	X	1	10	00	X
3	0	0	1	0	1	0	0	X	X	XX	XX	X
4	0	0	X	0	0	0	1	1	X	XX	XX	X
5	0	0	1	1	0	0	0	X	X	XX	XX	X
6	0	0	X	0	0	0	0	X	1	00	10	X
7	0	0	X	0	0	0	1	0	X	XX	XX	X
8	1	1	X	0	0	0	0	X	1	00	01	1

Implementación de la MEF

- ▶ Hay muchas maneras de sintetizar una MEF, vistas en materias anteriores
- ▶ Pero siempre se trata de determinar dos funciones (salidas y próximo estado) en base a las entradas y el estado actual
- ▶ Un esquema muy genérico es el siguiente:



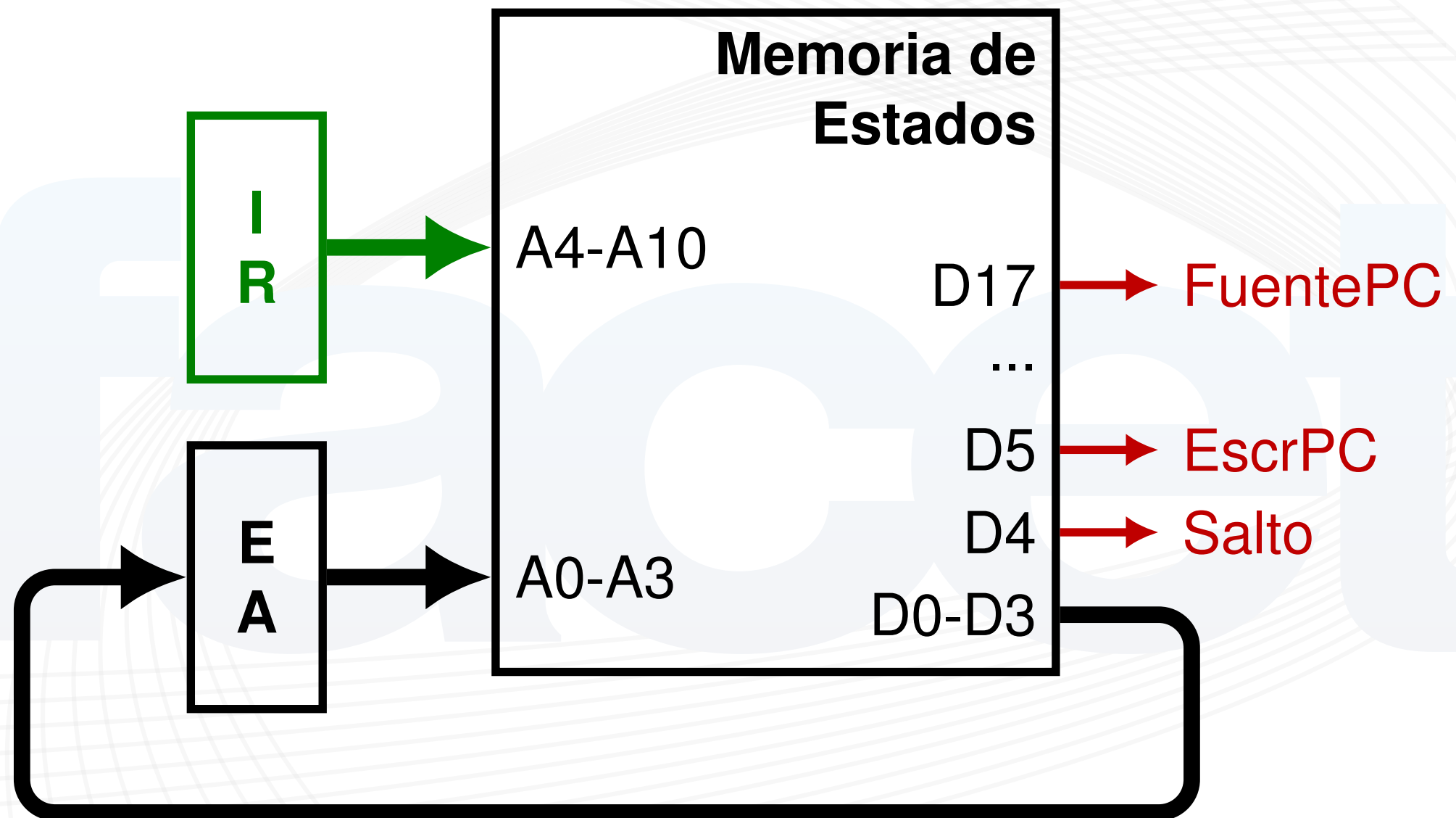
Implementación de la MEF

- ▶ La MEF es tipo Moore, se simplifica la función de salida
- ▶ Un registro en el control mantiene el estado actual
 - ▶ Como los demás, se actualiza en cada ciclo de reloj
- ▶ Para codificar los 9 estados de la MEF, se necesitan 4 bits
 - ▶ Ese será el tamaño del registro de estado actual
- ▶ Entonces, las entradas del circuito serán 11
 - ▶ 7 bits del opcode, que provienen del IR
 - ▶ 4 bits del estado actual, que proviene del nuevo registro
- ▶ Las salidas serán 18
 - ▶ 14 bits para las 12 señales de control del camino de datos
 - ▶ 4 bits para indicar el próximo estado

Implementación de la MEF con ROM

- ▶ Sintetizar una MEF de 11 entradas y 18 salidas no es sencillo utilizando métodos convencionales (como mapas de Karnaugh)
- ▶ Cuando se quiera ampliar la cantidad de instrucciones el resultado no será escalable
- ▶ Posiblemente la implementación más sencilla sea mediante una memoria ROM
- ▶ Cada dirección representará un estado

Implementación de la MEF con ROM



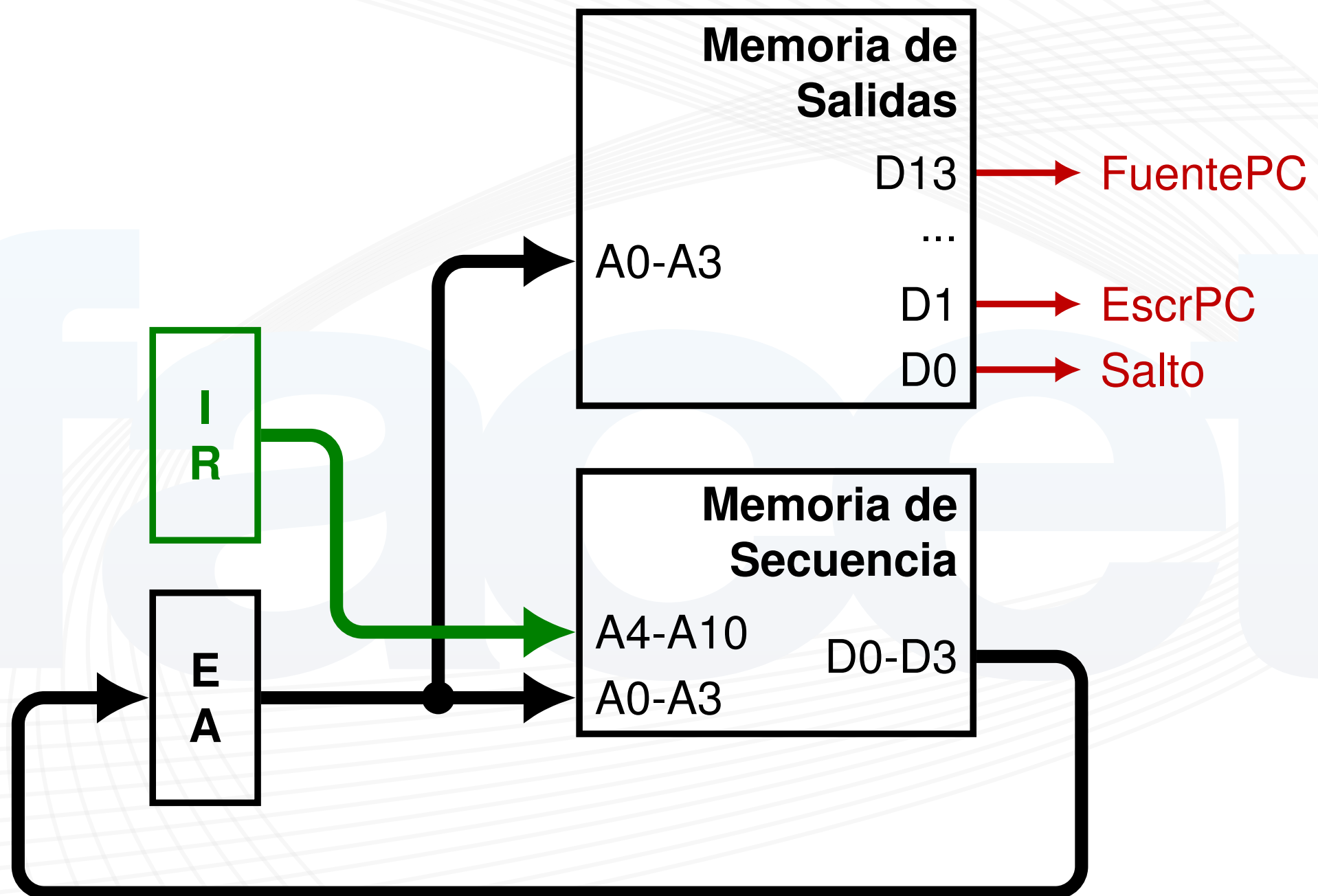
Implementación de la MEF con ROM

- ▶ Los 11 bits de entrada permiten tener $2^{11} = 2048$ direcciones diferentes
 - ▶ Se podrían implementar hasta 2048 estados diferentes
 - ▶ Pero solo se utilizan 9 estados
- ▶ Tamaño total de la ROM: $2^{11} * 18 = 36 \text{ Kb}$
 - ▶ No tiene en cuenta las condiciones de indiferencia

Implementación de la MEF con ROM

- ▶ Aprovechando que la MEF es estilo Moore, se puede mejorar la implementación, dividiendo en dos la memoria ROM:
- ▶ Una Memoria de Secuencia, que determina el estado siguiente, en base a todas las entradas
 - ▶ 11 bits de entrada, 4 bits de salida
- ▶ Una Memoria de Salidas, que genera las salidas para el camino de datos en base al estado actual
 - ▶ 4 bits de entrada, 14 bits de salida

Implementación de la MEF con ROM



Implementación de la MEF con ROM

- ▶ Tamaño Total: $2^{11} * 4 + 2^4 * 14 = 8,22 \text{ Kb}$
 - ▶ Reduce un 77% el tamaño, mucho más eficiente
- ▶ La Memoria de Secuencias es la que consume la mayor cantidad de espacio, más del 97% del total
 - ▶ Sigue sin considerar las condiciones de indiferencia
 - ▶ Posee múltiples direcciones redundantes, o que no se necesitan
 - ▶ Por ejemplo, habrá 128 direcciones que indiquen que del estado 0 se pasa al estado 1

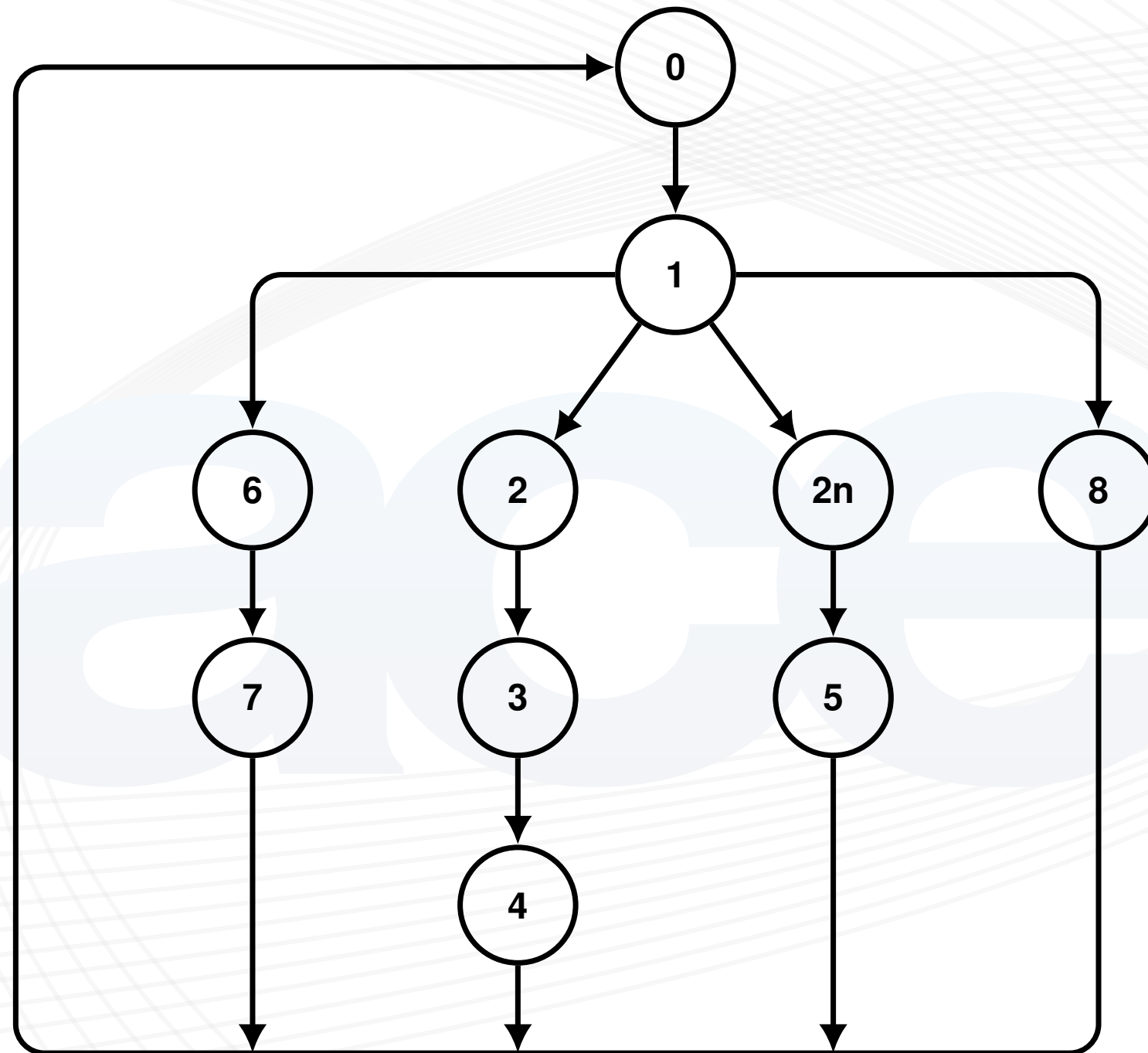
Análisis de transición de estados

- ▶ En el diagrama de estados la MEF, se observa que la mayoría de las transiciones de estado son simples
 - ▶ Se pasa al estado siguiente, o al estado inicial
- ▶ Tres estados pasan siempre al siguiente
 - ▶ 0 pasa a 1, 3 pasa a 4 y 6 pasa a 7
- ▶ Cuatro estados pasan siempre al estado inicial
 - ▶ 4, 5, 7 y 8 pasan a 0
- ▶ Solamente dos estados poseen bifurcaciones que dependen de la instrucción
 - ▶ 1 puede pasar a 2, 6 u 8
 - ▶ 2 puede pasar a 3 o 5

Simplificación de bifurcaciones

- ▶ Es posible eliminar una de las dos bifurcaciones
 - ▶ 2 puede pasar a 3 o 5
- ▶ Simplemente, duplicando el estado 2 en un nuevo estado idéntico $2n$
 - ▶ 2 pasa a 3 y $2n$ pasa a 5
- ▶ En vez de 9 estados serán 10

Simplificación de bifurcaciones



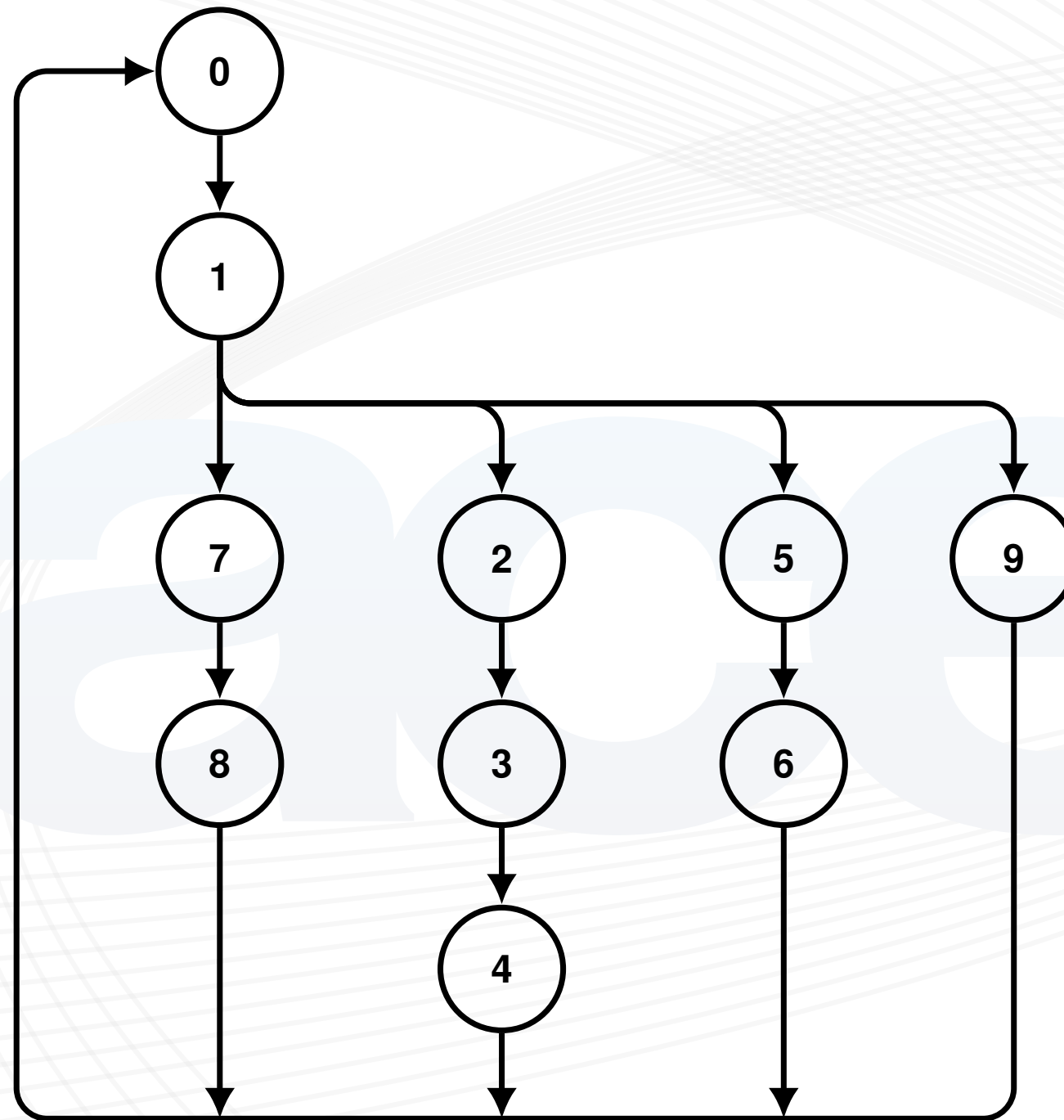
Simplificación de bifurcaciones

- ▶ Agregar el nuevo estado no produce ningún efecto perjudicial en lo diseñado hasta ahora
 - ▶ Siguen siendo necesarios 4 bits para la cantidad de estados
- ▶ Eliminar la segunda bifurcación genera múltiples ventajas:
 - ▶ Simplifica la implementación de la MEF, haciendo que los estados transicionen solamente entre una de tres opciones:
 - ▶ Al siguiente, al inicial, o en la única ramificación
 - ▶ Esa única ramificación es múltiple, y fácilmente escalable cuando se quieran agregar más instrucciones
 - ▶ Esa única ramificación es la única transición en la cual se usan los opcodes

Simplificación de bifurcaciones

- ▶ Sin embargo, genera un pequeño inconveniente:
 - ▶ Al estado duplicado, $2n$, hay que darle un número de estado “real”
 - ▶ Se debe mantener que todos los estados transicionen siempre al siguiente, o al inicial, o en la única ramificación
- ▶ Resulta necesario renombrar los estados del diagrama de la MEF, y de la tabla de control vista.
 - ▶ **IMPORTANTE:** solo cambian los números de estado

MEF de Control Final



MEF de Control Final

Estado	Salto	Escr PC	IoD	Escr Mem	Leer Mem	Escr IR	Mem AReg	Escr Reg	Fuente A	Fuente B	Op ALU	Fuente PC
0	0	1	0	0	1	1	X	0	0	01	00	0
1	0	0	X	0	0	0	X	0	0	10	00	X
2	0	0	X	0	0	0	X	0	1	10	00	X
3	0	0	1	0	1	0	X	0	X	XX	XX	X
4	0	0	X	0	0	0	1	1	X	XX	XX	X
5	0	0	X	0	0	0	X	0	1	10	00	X
6	0	0	1	1	0	0	X	0	X	XX	XX	X
7	0	0	X	0	0	0	X	0	1	00	10	X
8	0	0	X	0	0	0	0	1	X	XX	XX	X
9	1	1	X	0	0	0	X	0	1	00	01	1

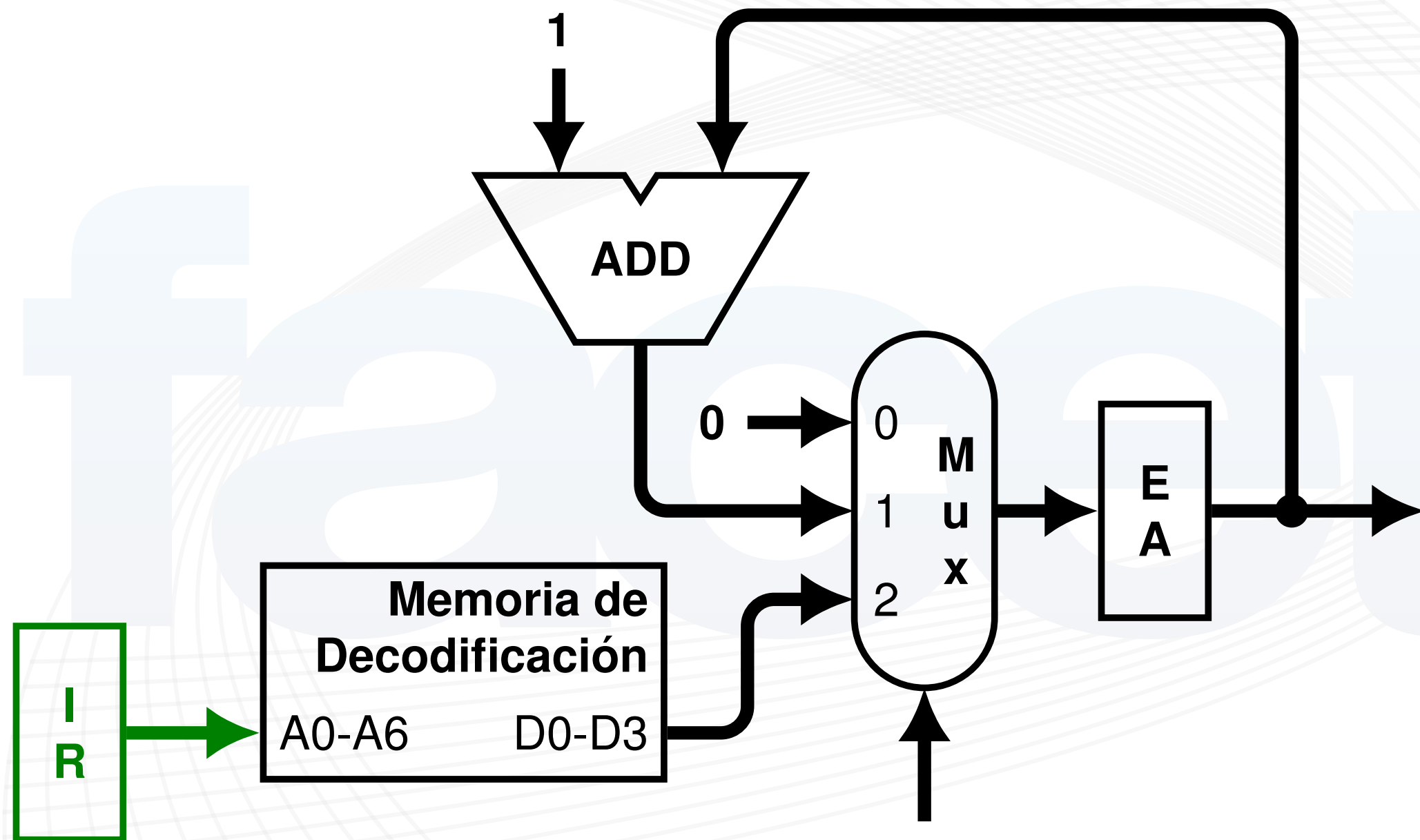
Transición de estados final

- ▶ Cuatro estados pasan siempre al estado inicial:
 - ▶ 4, 6, 8 y 9 pasan a 0
- ▶ Cinco estados pasan siempre al siguiente:
 - ▶ 0 pasa a 1, 2 pasa a 3, 3 pasa a 4, 5 pasa a 6 y 7 pasa a 8
- ▶ Solamente un estado posee ramificaciones que dependen de la instrucción:
 - ▶ 1 puede pasar a 2, 5, 7 o 9

Implementación con secuenciador

- ▶ La Memoria de Secuencias puede reemplazada por un circuito Secuenciador
- ▶ Un multiplexor que selecciona 3 posibles entradas
 - ▶ Inicial (estado 0), estado siguiente o ramificación
 - ▶ Un sumador +1, para calcular el estado siguiente
 - ▶ Una ROM pequeña para las ramificaciones, a la que llamaremos Memoria de Decodificación

Implementación con secuenciador



Implementación con secuenciador

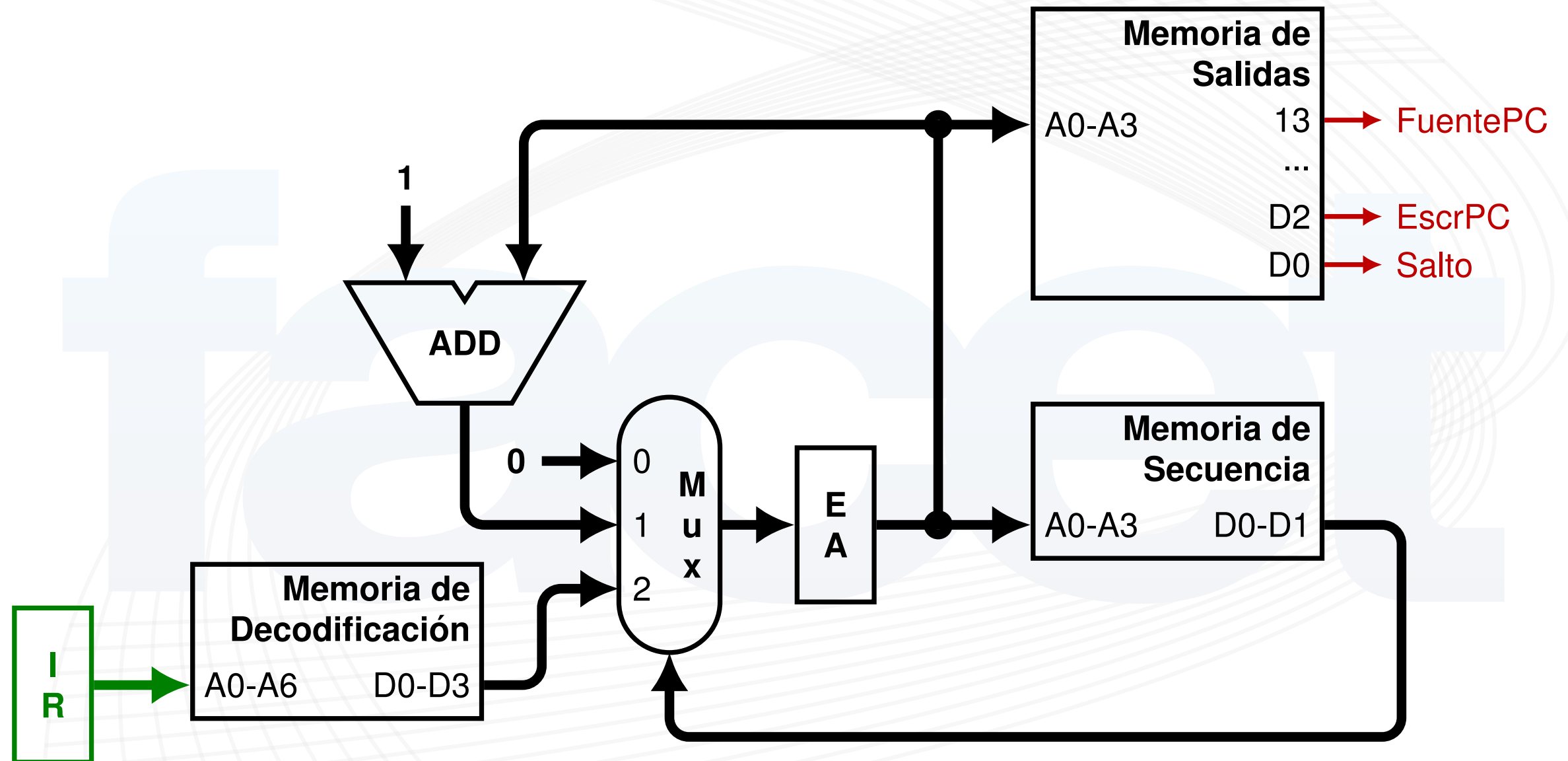
- ▶ La Memoria de Decodificación es la encargada de implementar la única ramificación
- ▶ Solamente depende del opcode
 - ▶ Tendrá 7 bits de entradas
- ▶ Su salida es el próximo estado
 - ▶ Tendrá 4 bits de salida
- ▶ Tamaño: $2^7 * 4 = 0,5 \text{ Kb}$
- ▶ El contenido simplificado se muestra en la tabla adjunta

opcode	Estado
tipo r	0111
lw	0010
sw	0101
beq	1001

Implementación con secuenciador

- ▶ La Memoria de Secuencias aún no puede ser totalmente eliminada
- ▶ Se necesitan las señales de control para el multiplexor del Secuenciador (2 bits)
- ▶ Sin embargo, su tamaño se ve notoriamente reducido, ya que ahora sus entradas dependen únicamente del estado actual
- ▶ Posee 4 bits de entrada y 2 de salida
- ▶ Tamaño: $2^4 * 2 = 32$ bits

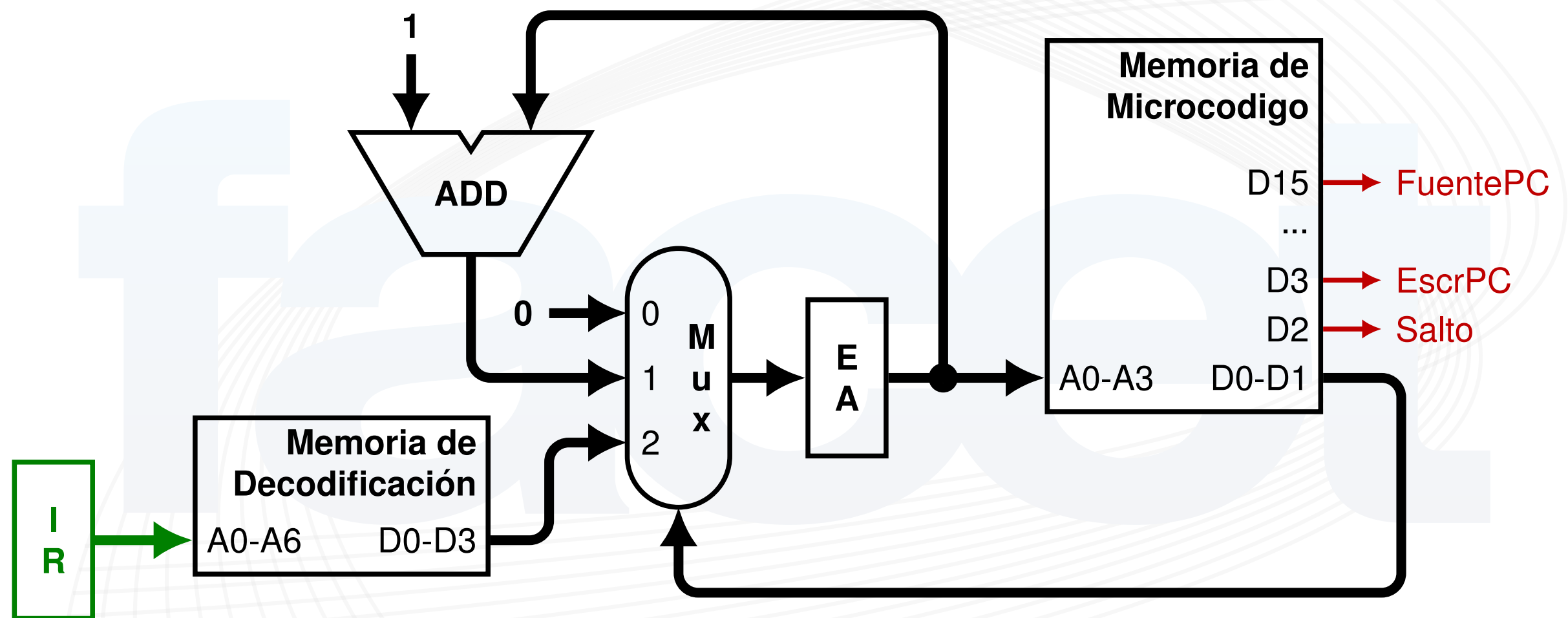
Implementación con secuenciador



Implementación final de la MEF

- ▶ En el diagrama anterior, se observa que ahora tanto la Memoria de Salidas como la nueva Memoria de Secuencias dependen del estado actual
- ▶ Por lo tanto, podrían unificarse en una única memoria, que tenga 4 entradas y 16 salidas
 - ▶ 14 para el camino de datos, y 2 para el Secuenciador
 - ▶ Tamaño: $2^4 * (14 + 2) = 0,25 \text{ Kb}$
- ▶ Esta memoria unificada suele recibir el nombre de Memoria de Microcódigo.
 - ▶ Almacena las señales de control para el camino de datos, y el control de la secuencia para el Secuenciador

Implementación final de la MEF



Implementación final de la MEF

- ▶ La implementación final posee dos memorias ROM
 - ▶ Memoria de Decodificación, de 0,5 Kb
 - ▶ Memoria de Microcódigo, de 0,25 Kb
- ▶ Tamaño Total: 0,75 Kb
 - ▶ Versus 36 Kb de la primera y 8 Kb de la segunda versión
- ▶ Pero las principales ventajas son flexibilidad y modularidad
 - ▶ Es muy simple agregar instrucciones nuevas, ampliando el contenido de la Memoria de Decodificación
 - ▶ Si se modifica el camino de datos, únicamente hay que modificar la Memoria de Microcódigo

Contenido Memoria de Microcódigo

- Igual que la tabla de diapositivas anteriores, pero con los bits adicionales para control de secuencia

Estado	Ctrl Seq	Salto	Escr PC	IoD	Escr Mem	Leer Mem	Escr IR	Mem AReg	Escr Reg	Fuente A	Fuente B	Op ALU	Fuente PC
0	01	0	1	0	0	1	1	X	0	0	01	00	0
1	10	0	0	X	0	0	0	X	0	0	10	00	X
2	01	0	0	X	0	0	0	X	0	1	10	00	X
3	01	0	0	1	0	1	0	X	0	X	XX	XX	X
4	00	0	0	X	0	0	0	1	1	X	XX	XX	X
5	01	0	0	X	0	0	0	X	0	1	10	00	X
6	00	0	0	1	1	0	0	X	0	X	XX	XX	X
7	01	0	0	X	0	0	0	X	0	1	00	10	X
8	00	0	0	X	0	0	0	0	1	X	XX	XX	X
9	00	1	1	X	0	0	0	X	0	1	00	01	1

Contenido Memoria de Microcódigo

- Comprimiendo la tabla anterior, y comparándola con el RTN concreto

Estado	Memoria de Microcódigo	RTN Concreto
0	01010011X0001000	$IR \leftarrow M[PC]; PC \leftarrow PC + 4;$
1	1000X000X001000X	$0A \leftarrow R[IR<19:15>]; 0B \leftarrow R[IR<24:20>];$ $SA \leftarrow PC + imm(IR);$
2	0100X000X011000X	$SA \leftarrow 0A + imm(IR);$
3	01001010X0XXXXXX	$MDR \leftarrow M[SA];$
4	0000X00011XXXXXX	$R[IR<11:7>] \leftarrow MDR;$
5	0100X000X011000X	$SA \leftarrow 0A + imm(IR);$
6	00001100X0XXXXXX	$M[SA] \leftarrow 0B;$
7	0100X000X010010X	$SA \leftarrow 0A \ 0p \ 0B;$
8	0000X00001XXXXXX	$R[IR<11:7>] \leftarrow SA;$
9	0011X000X0100011	$If \ (0A == 0B) \ PC \leftarrow SA;$

Microprogramación

- ▶ Un programa es una secuencia de instrucciones, que indica qué hacer con los datos y cómo sigue la secuencia
- ▶ En la Memoria de Microcódigo, las salidas indican al camino de datos qué hacer y cómo sigue la secuencia
- ▶ Entonces, se dice que cada línea de la Memoria de Microcódigo es una microinstrucción
- ▶ Y el contenido total de la Memoria de Microcódigo se denomina microprograma
- ▶ Las instrucciones manejan datos mientras que las microinstrucciones manejan señales de control

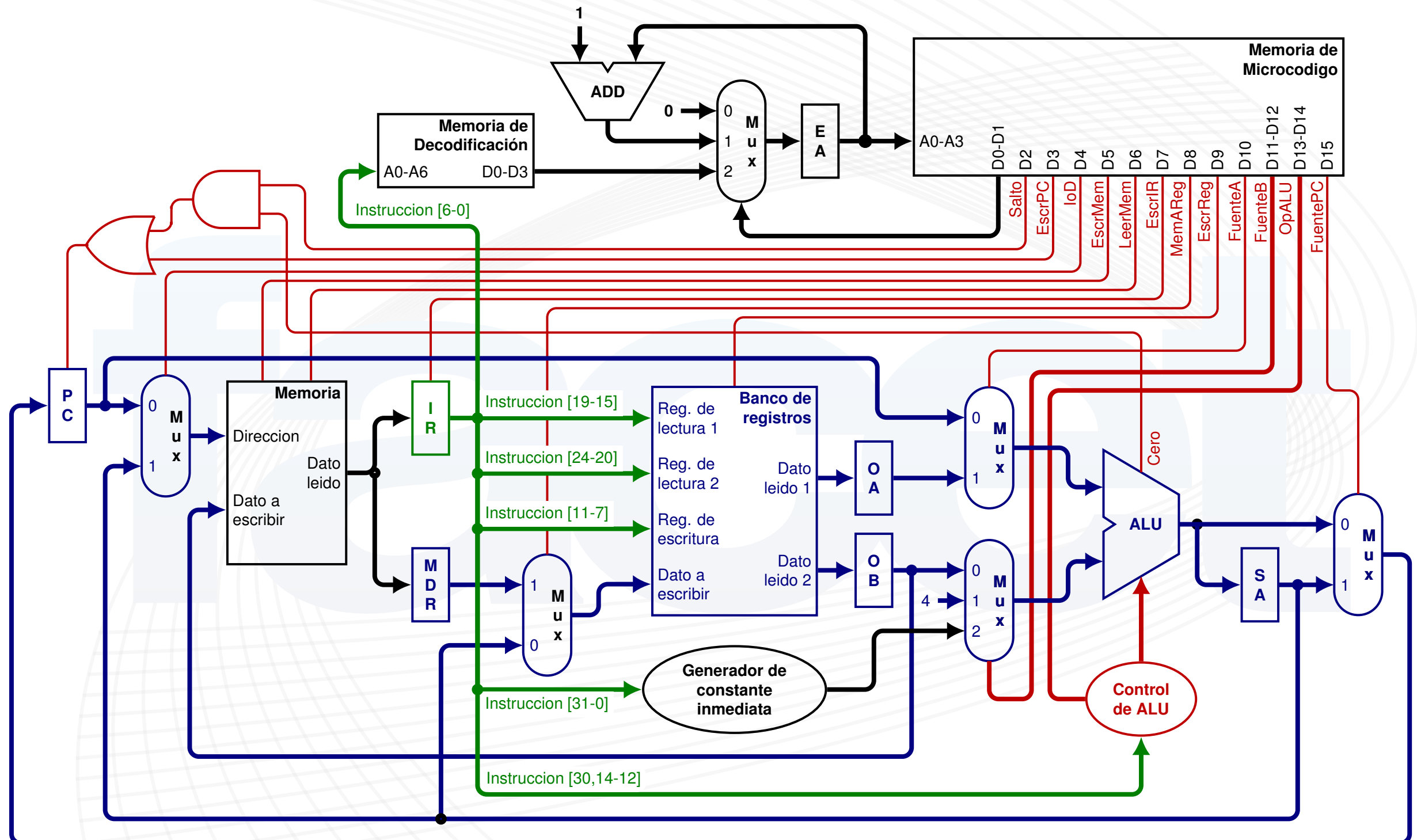
Microprogramación

- ▶ Cada instrucción del ISA se mapea en una secuencia de microinstrucciones
 - ▶ Las microinstrucciones son fijas, son las instrucciones las que pueden cambiar al ejecutar otro programa
- ▶ Cambiar las microinstrucciones implica cambiar la Memoria de Microcódigo
 - ▶ Se hace usualmente, para corregir bugs o para ampliar el ISA con nuevas instrucciones
 - ▶ O para hacer que una máquina se comporte como otra (emulación)
- ▶ Es muy sencillo agregar instrucciones que sean combinaciones de microinstrucciones ya existentes

Conclusiones

- ▶ El control implementado con microprogramación es:
 - ▶ Relativamente sencillo y muy flexible
 - ▶ Permite hacer actualizaciones en el tiempo
- ▶ Estilo usado desde el inicio de la computación hasta mediados de la década de 1980
 - ▶ Aunque de algún modo sigue presente en la actualidad
- ▶ Su principal inconveniente, es que no es compatible con diseños de alta performance
 - ▶ Sirve de inspiración para otros diseños actuales
 - ▶ Que verán en la materia que sigue

Camino de Datos completo



SRC-1: una arquitectura alternativa

