Capítulo VI
Introducción al Diseño de Computadores

Cristián Tejos
Primer Semestre 2019

Basado en apuntes de Marcelo Guarini
y el libro
Digital Design and Computer Architecture (2nd edition),
D. M. Harris & S. L. Harris, Elsevier 2013

Programa

- Ira Parte - Unidad de Flujo de Datos (*Data Path*)
  - Introducción
  - Ejemplos de Unidades de Flujo de Datos
    - Unidad Aritmético Lógica
    - Desplazador
    - Representación del Flujo de Datos y Palabra de Control
- 2a Parte - Un Computador Simple
- 3a Parte - Control Alambrado de Múltiples Ciclos

Introducción

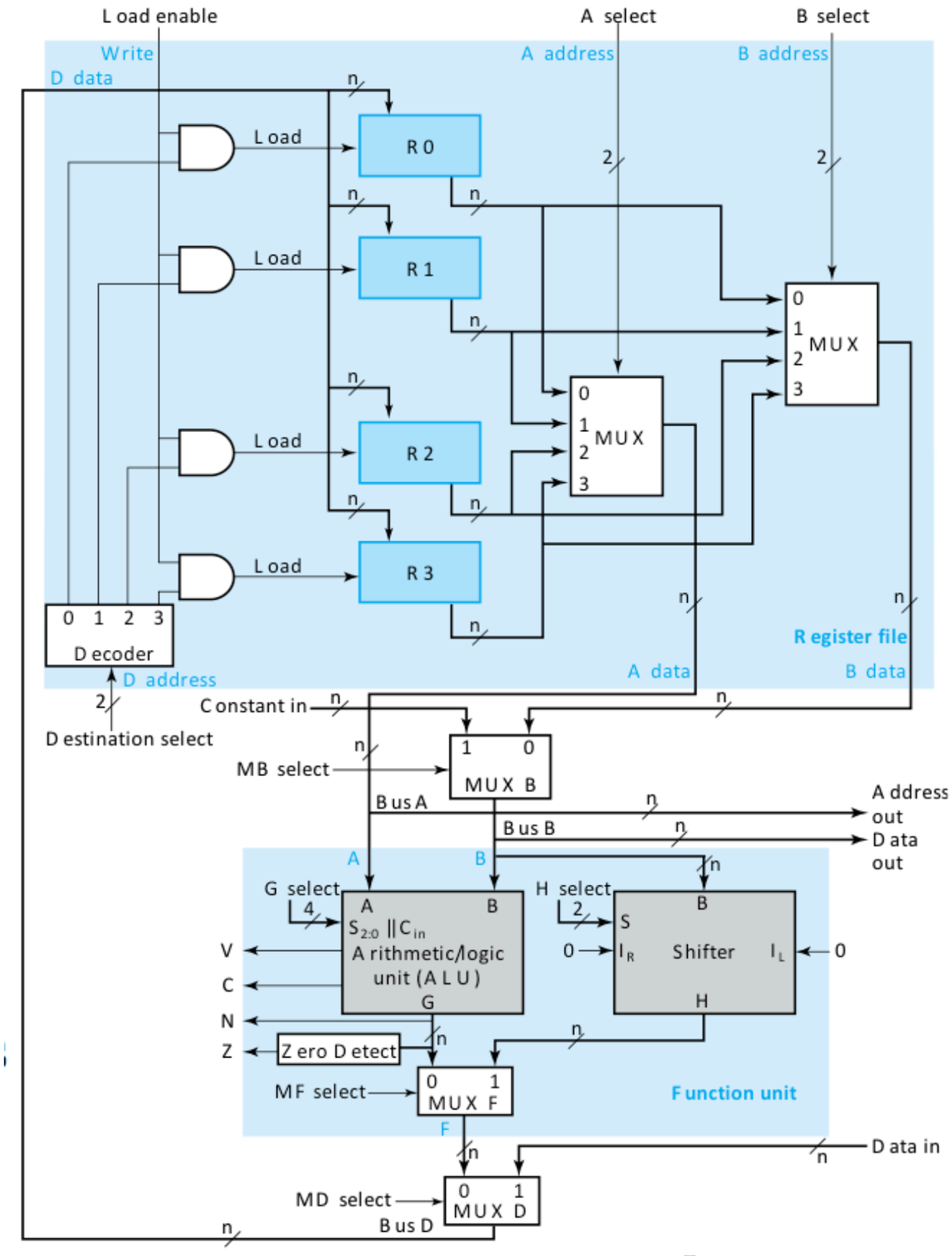
- En esencia, la especificación de un computador consiste de una descripción de su apariencia a un programador en el nivel más bajo, *su arquitectura del conjunto de instrucciones*.
- A partir de ella se formula una descripción de hardware de alto nivel para implementar el computador, llamada *arquitectura del computador*.
- Para un computador pequeño típico, esta arquitectura normalmente se divide en unidad de flujo de datos (*data path*) y una unidad de control.
- La unidad de flujo de datos se define por tres componentes básicos.
    - Un conjunto de registros.
    - Las microoperaciones realizadas sobre datos almacenados en los registros.
    - La interfaz de control.

Unidad de Flujo de Datos

- Guía de principios para el diseño de unidades de flujo de datos básicas
  - Conjunto de registros
    - Una colección de registros individuales
    - Un colección de registros con recursos de acceso común llamado *conjunto de registros*.
    - Una combinación de ambos
  - Implementación de las microoperaciones
    - Uno o más recursos compartidos para implementar las microoperaciones
    - Buses - caminos de transferencia compartidos
    - Unidad aritmético-lógica (ALU) - Recurso compartido para implementar microoperaciones aritméticas y lógicas
    - Desplazador - Recurso compartido para implementar microoperaciones de desplazamiento

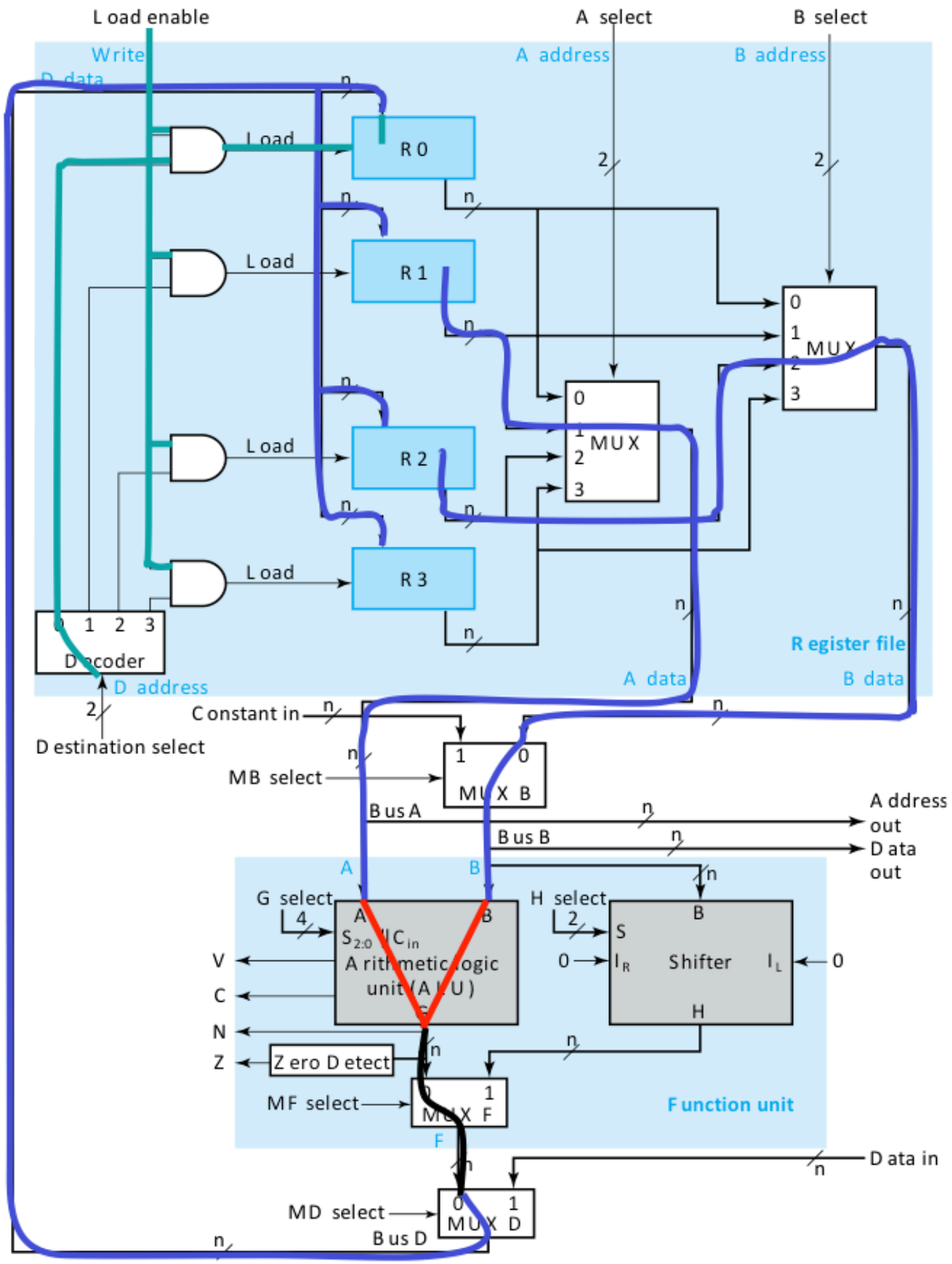
Ejemplo de Unidad de Flujo de Datos

- Cuatro registros de carga paralela
- Dos selectores de registros basados en multiplexores
- Decodificador de registro destino
- Multilexor B para ingreso de constante externa
- Buses A y B con salida de direccionamiento y de datos hacia el exterior.
- ALU y desplazador con multiplexor F para seleccionar la salida
- Multiplexor D para ingresos de datos externos
- Lógica para la generación de los bits de status V, C, N, Z.



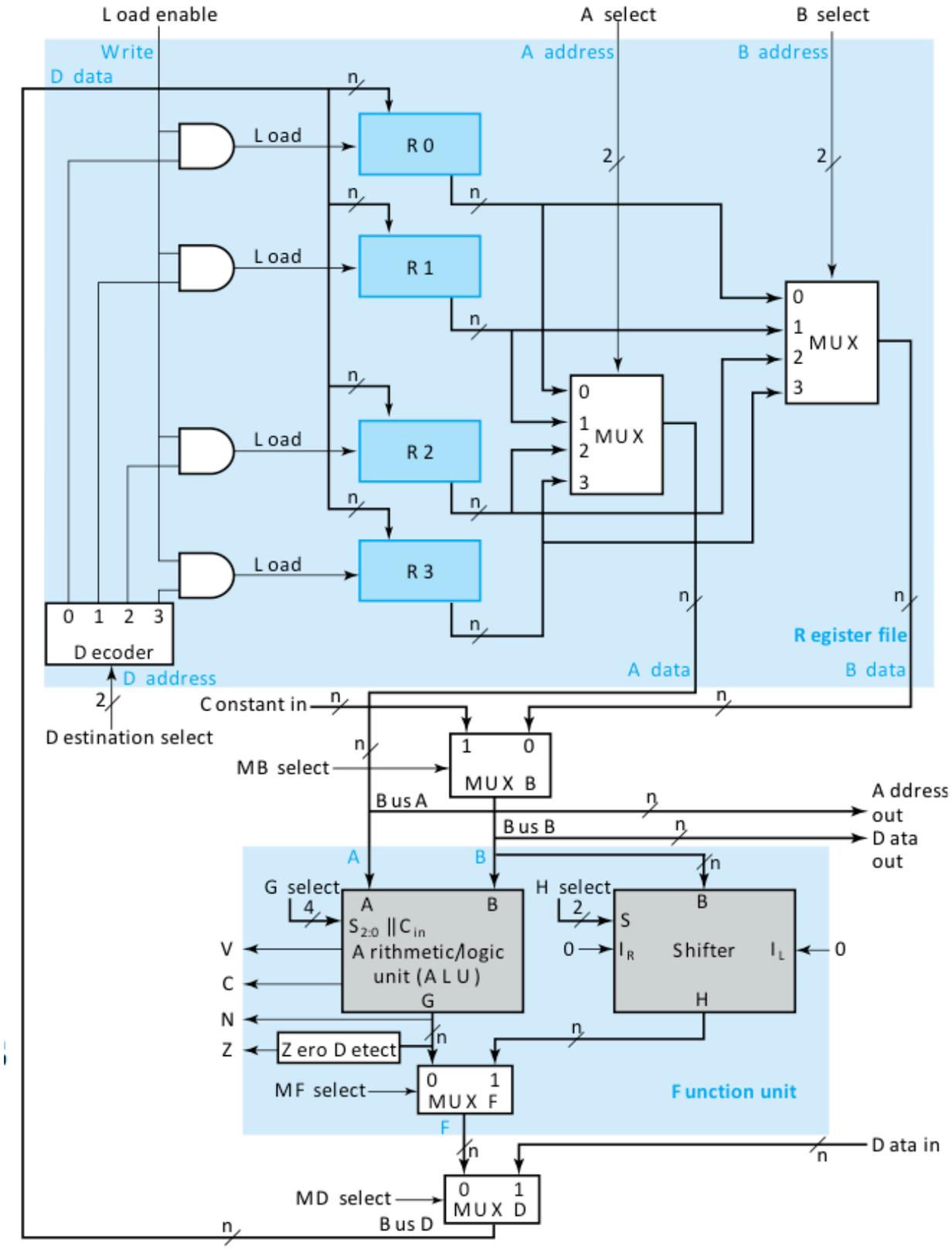
Ejemplo: Ejecución de una Microoperación

- Microoperación: R0 <== R1 + R2
- Aplicar 01 a A select para poner el contenido de R1 en el Bus A
- Aplicar 10 a B select para poner el contenido de R2 en B data y aplicar 0 en MB select para poner B data en el Bus B
- Aplicar 0010 en G select para realizar la suma G = Bus A + Bus B
- Aplicar 0 a MF select y 0 a MD select para poner el valor de G en el Bus D
- Aplicar 00 a Destination select para habilitar la entrada de Load a R0
- Aplicar 1 a Load Enable para llevar el Load Input de R0 a 1 para que R0 se carga en al flanco del clock
- Toda la microinstrucción requiere un sólo ciclo de reloj



Ejemplo: Acciones de control

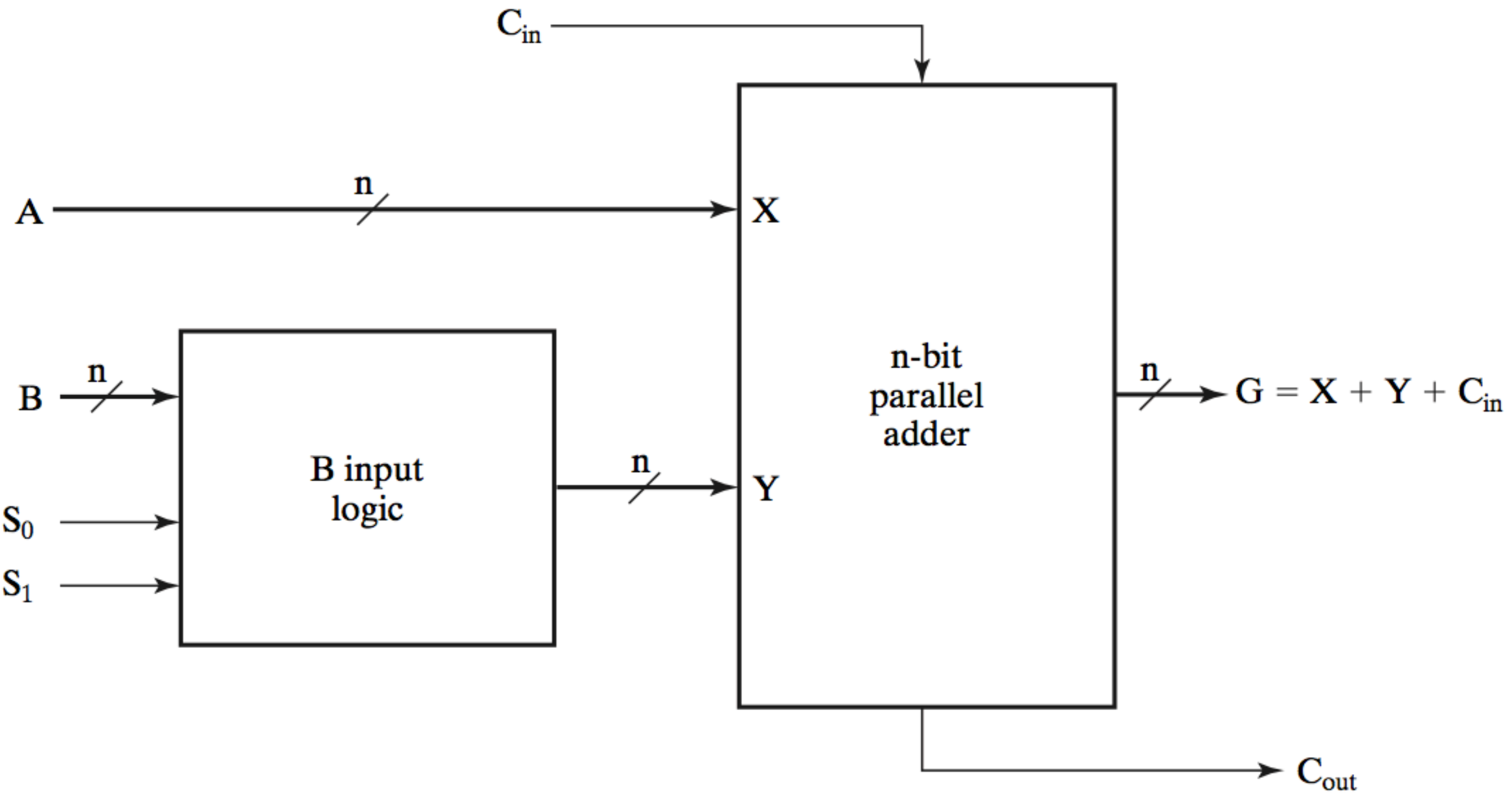
- Realizar una microoperación de desplazamiento - Aplicar 1 a MF select
- Utilizar una Cte - aplicar 1 a MB select
- Proveer una dirección y datos para una escritura en memoria externa - aplicar 0 a Load Enable para prevenir la carga de un registro
- Proveer una dirección y obtener un dato para una lectura de memoria -aplicar 1 a MD select.
- Para alguna de las microinstruciones descritas, las otras entradas de control son *don't cares*.



Unidad Aritmético Lógica (ALU)

- En las siguientes dos sección veremos el diseño detallado de una ALU típica y del desplazador.
- Descomponemos la ALU en:
  - Un circuito aritmético
  - Un circuito lógico
  - Un selector para escoger uno de los dos circuitos
- Diseño del circuito aritmético
  - Descomponemos el circuito aritmético en:
    - Un sumador paralelo de N bits
    - Un bloque lógico que selecciona 4 posibilidades para la entrada B del sumador

Diseño del circuito Aritmético



Diseño del circuito Aritmético

- Hay cuatro funciones de B para seleccionar como Y en G = A + Y

$\mathbf{Y}$	$C_{in} = 0$	$C_{in} = 1$
0	$\mathbf{G = A}$	$\mathbf{G = A + 1}$
$\mathbf{B}$	$\mathbf{G = A + B}$	$\mathbf{G = A + B + 1}$
$\bar{\mathbf{B}}$	$\mathbf{G = A + \bar{B}}$	$\mathbf{G = A + \bar{B} + 1}$
1	$\mathbf{G = A - 1}$	$\mathbf{G = A}$

Diseño del circuito Aritmético

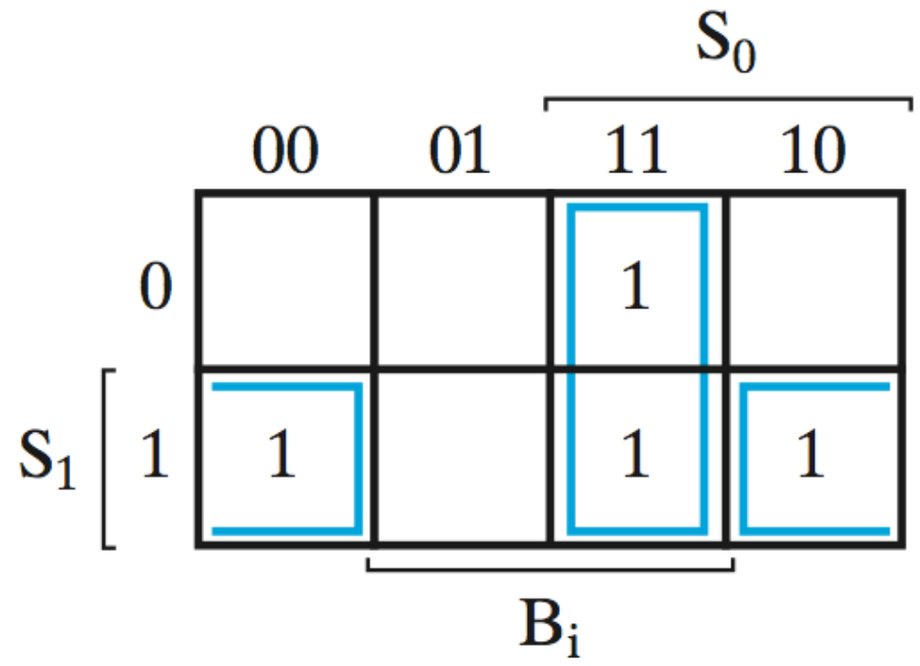
- Agregando los códigos de selección se tiene:

Select		Input	$G = (A \,1\, Y \,1\, C_{in})$	
S_1	S_0	Y	$C_{in} = 0$	$C_{in} = 1$
0	0	all 0s	$G = A$ (transfer)	$G = A + 1$ (increment)
0	1	B	$G = A + B$ (add)	$G = A + B + 1$
1	0	$\overline{B}$	$G = A + \overline{B}$	$G = A + \overline{B} + 1$ (subtract)
1	1	all 1s	$G = A - 1$ (decrement)	$G = A$ (transfer)

- Las funciones útiles están indicadas con una etiqueta (*label*)
- Notar que las cuatro funciones producen al menos una función útil.

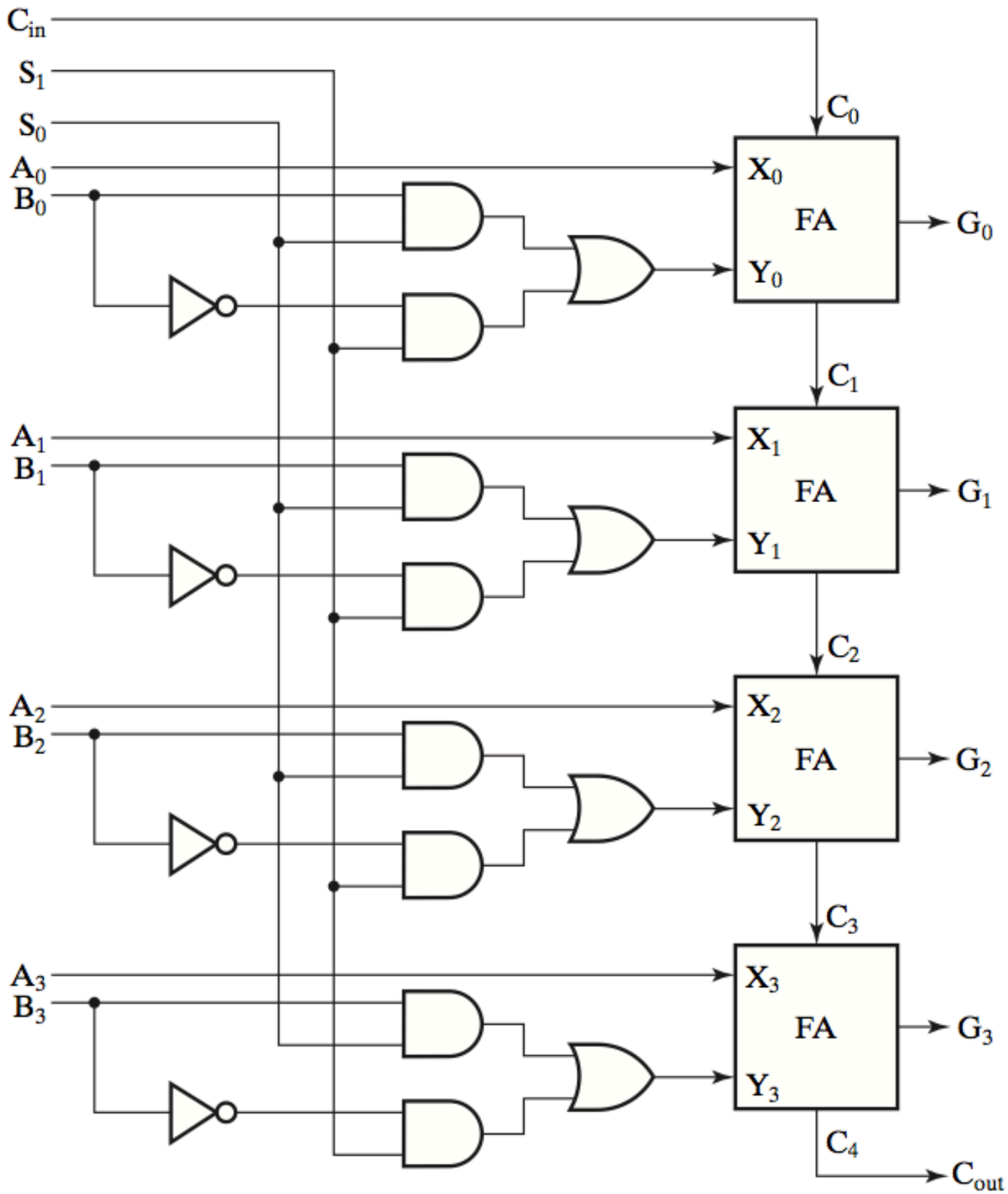
Diseño del circuito Aritmético

Inputs			Output	
S_1	S_0	B_i	Y_i	
0	0	0	0	$Y_i = 0$
0	0	1	0	
0	1	0	0	$Y_i = B_i$
0	1	1	1	
1	0	0	1	$Y_i = \overline{B}_i$
1	0	1	0	
1	1	0	1	$Y_i = 1$
1	1	1	1	



$$Y_i = B_i S_0 + \overline{B}_i S_1$$

Diseño del circuito Aritmético



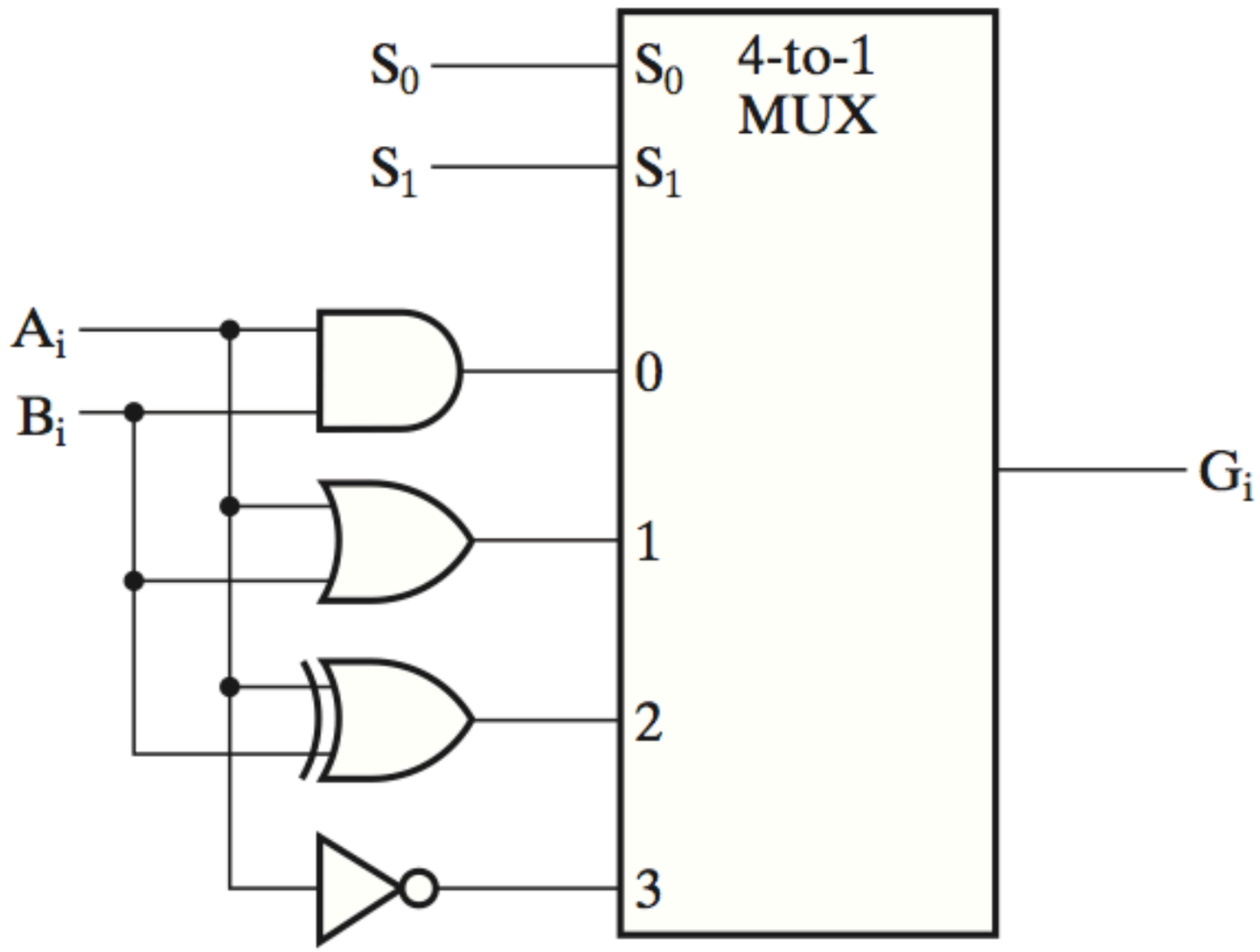
Diseño del Circuito Lógico

- Tabla de las funciones requeridas

S_1	S_0	Output	Operation
0	0	$G = A \wedge B$	AND
0	1	$G = A \vee B$	OR
1	0	$G = A \oplus B$	XOR
1	1	$G = \overline{A}$	NOT

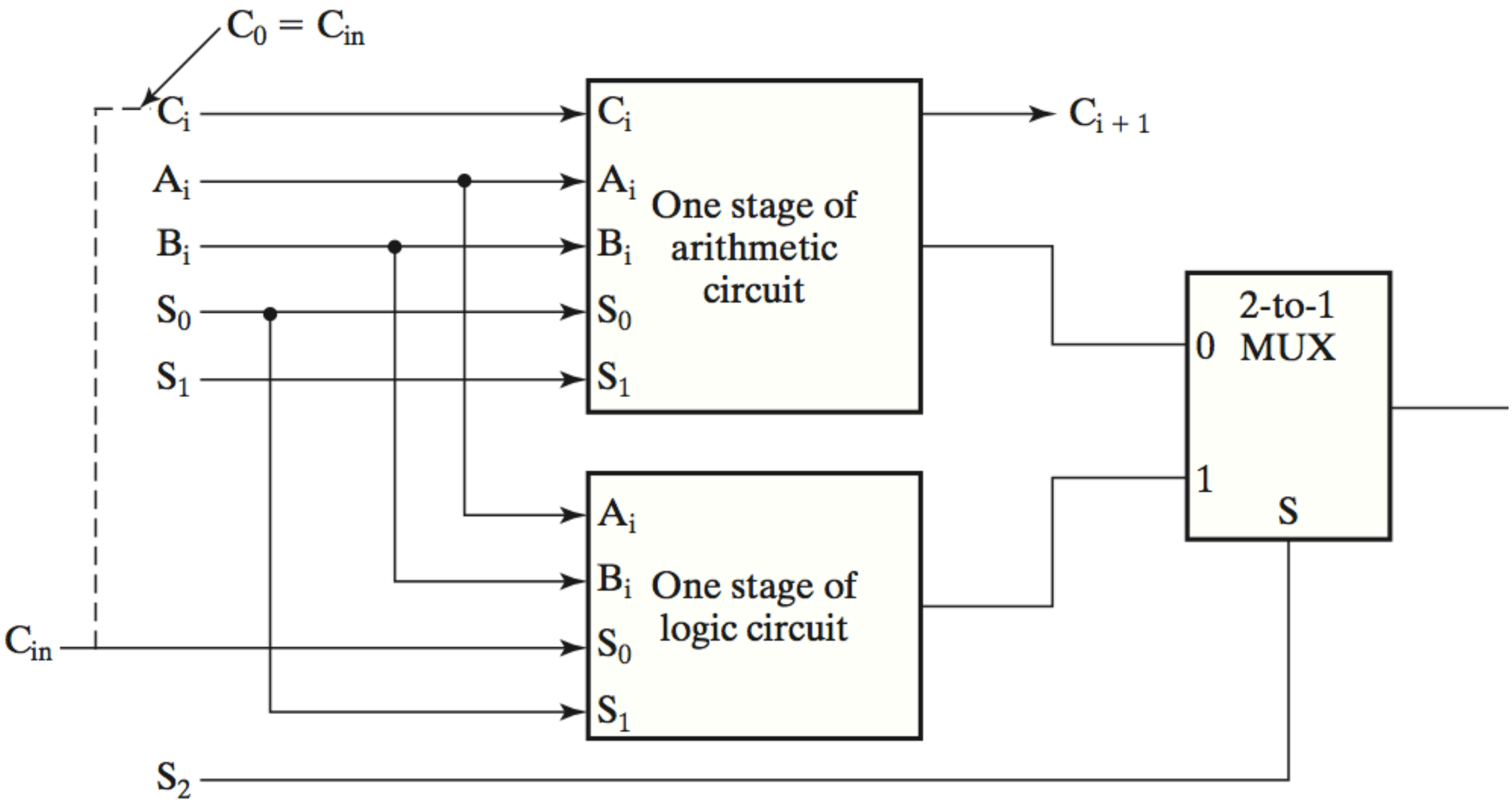
Diseño del Circuito Lógico

- Circuito que implementa las funciones requeridas (se muestra un sólo bit)



Unidad Aritmético Logica (ALU)

- Bloque correspondiente a un bit de la ALU



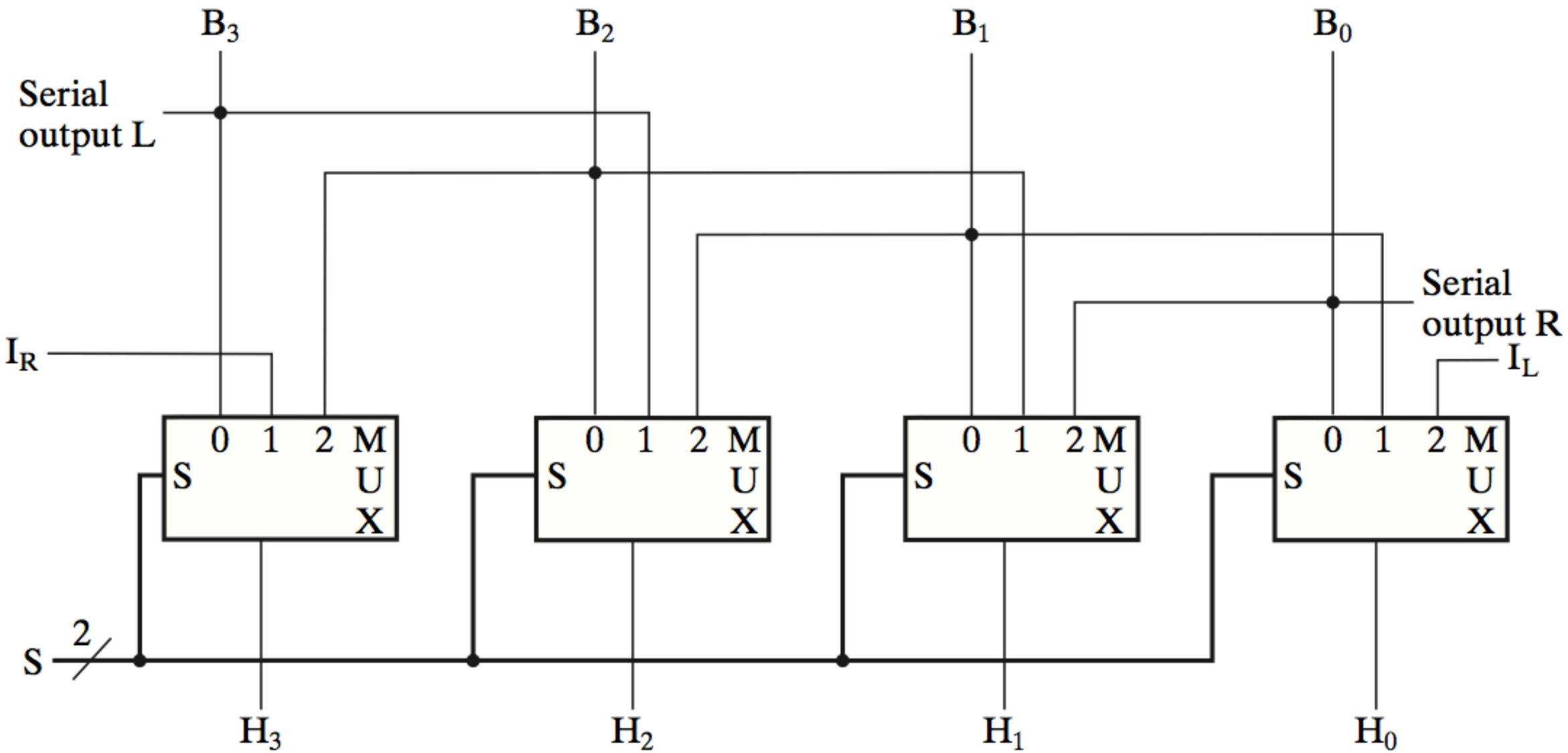
Unidad Aritmético Logica (ALU)

- Tabla de funciones de la ALU

Operation Select					
S_2	S_1	S_0	C_{in}	Operation	Function
0	0	0	0	$G = A$	Transfer A
0	0	0	1	$G = A + 1$	Increment A
0	0	1	0	$G = A + B$	Addition
0	0	1	1	$G = A + B + 1$	Add with carry input of 1
0	1	0	0	$G = A + \overline{B}$	A plus 1s complement of B
0	1	0	1	$G = A + \overline{B} + 1$	Subtraction
0	1	1	0	$G = A - 1$	Decrement A
0	1	1	1	$G = A$	Transfer A
1	X	0	0	$G = A \wedge B$	AND
1	X	0	1	$G = A \vee B$	OR
1	X	1	0	$G = A \oplus B$	XOR
1	X	1	1	$G = \overline{A}$	NOT (1s complement)

Unidad de Desplazamiento Combinacional

- Ejemplo



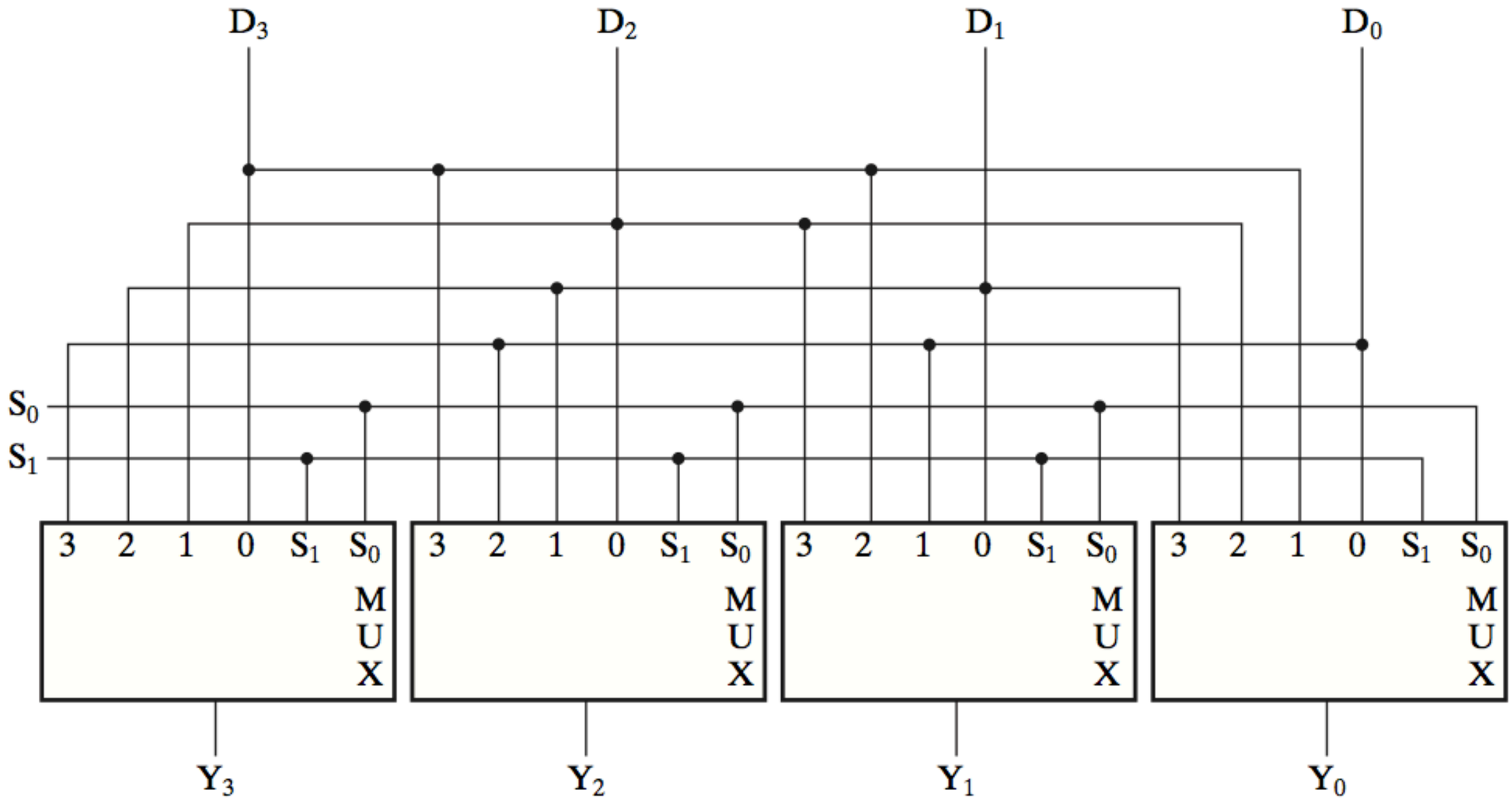
Unidad de Desplazamiento Combinacional

- Tabla de funciones de un rotador de 4 bits (4 bit barrel shifter)

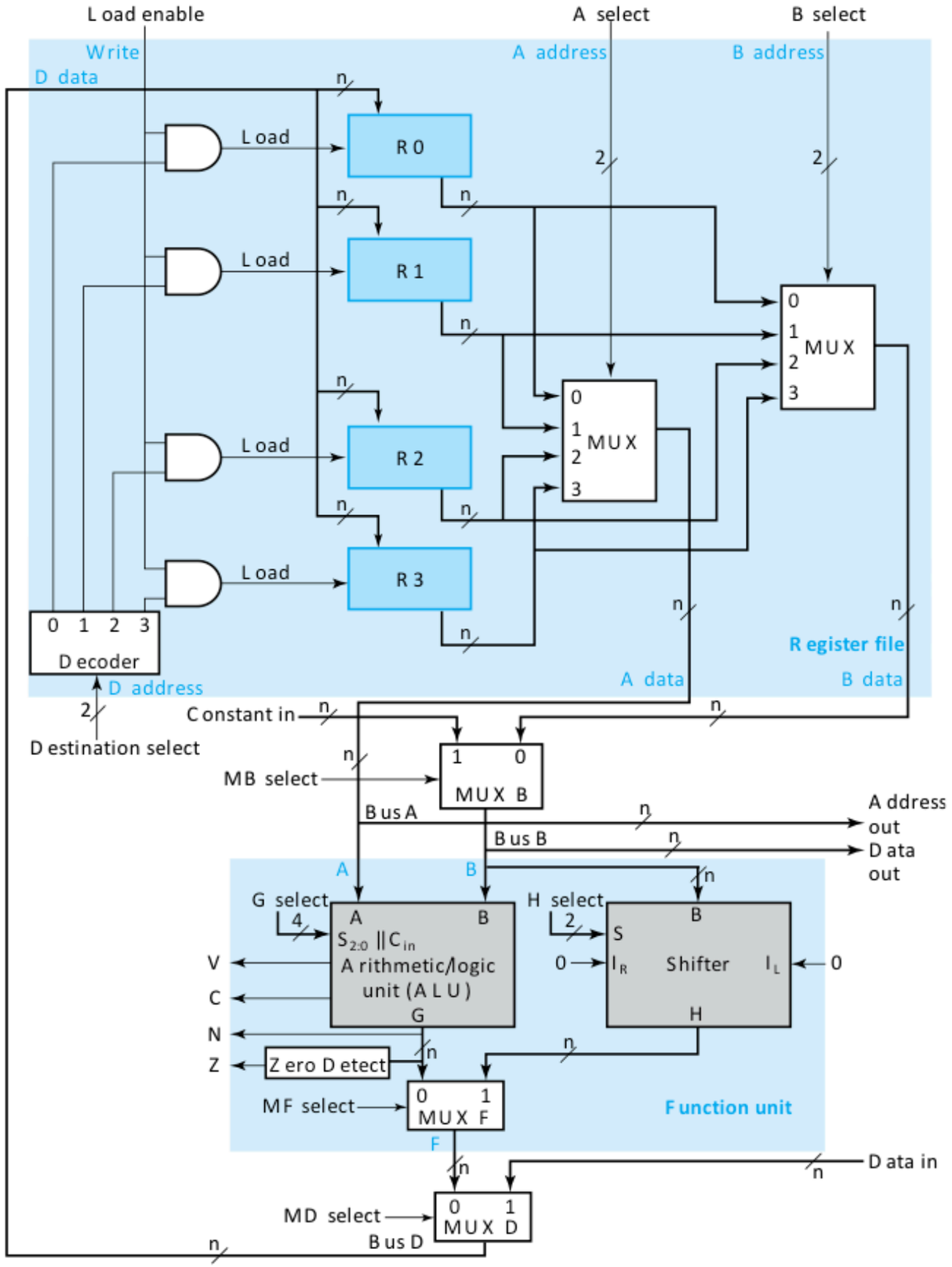
Select		Output				
S_1	S_0	Y_3	Y_2	Y_1	Y_0	Operation
0	0	D_3	D_2	D_1	D_0	No rotation
0	1	D_2	D_1	D_0	D_3	Rotate one position
1	0	D_1	D_0	D_3	D_2	Rotate two positions
1	1	D_0	D_3	D_2	D_1	Rotate three positions

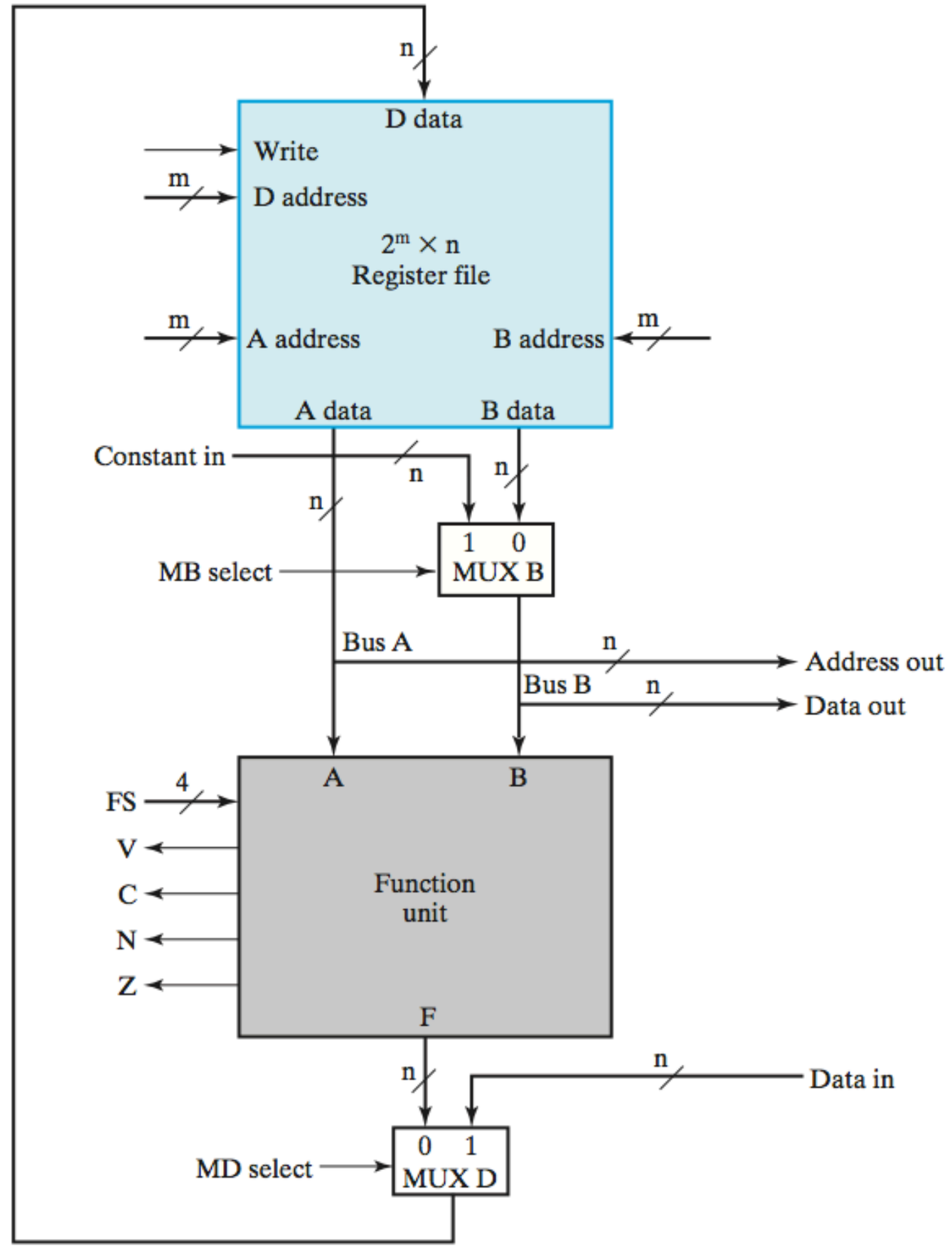
Unidad de Desplazamiento Combinacional

- Circuito rotador de 4 bits



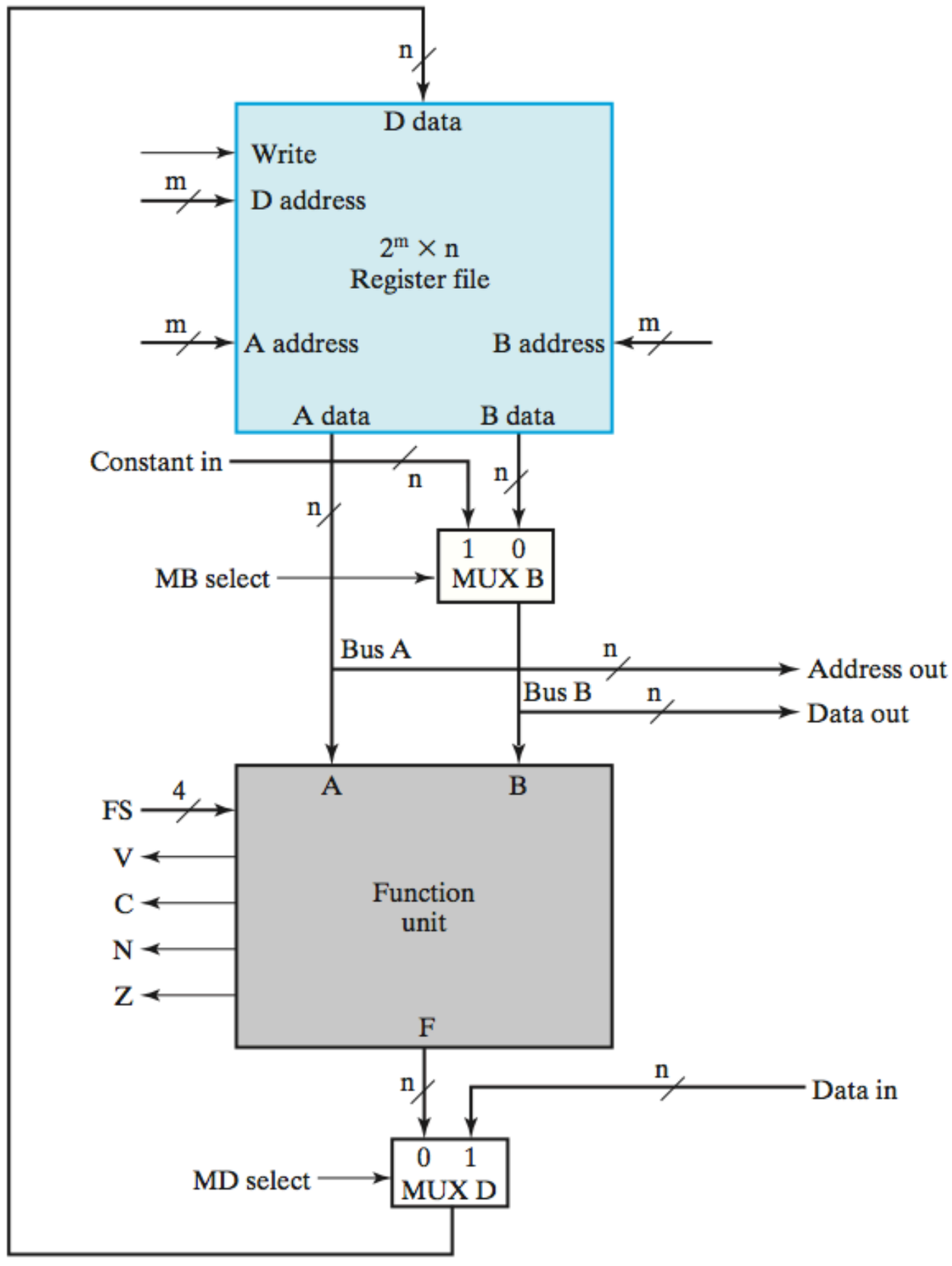
Representación del *Datapath*





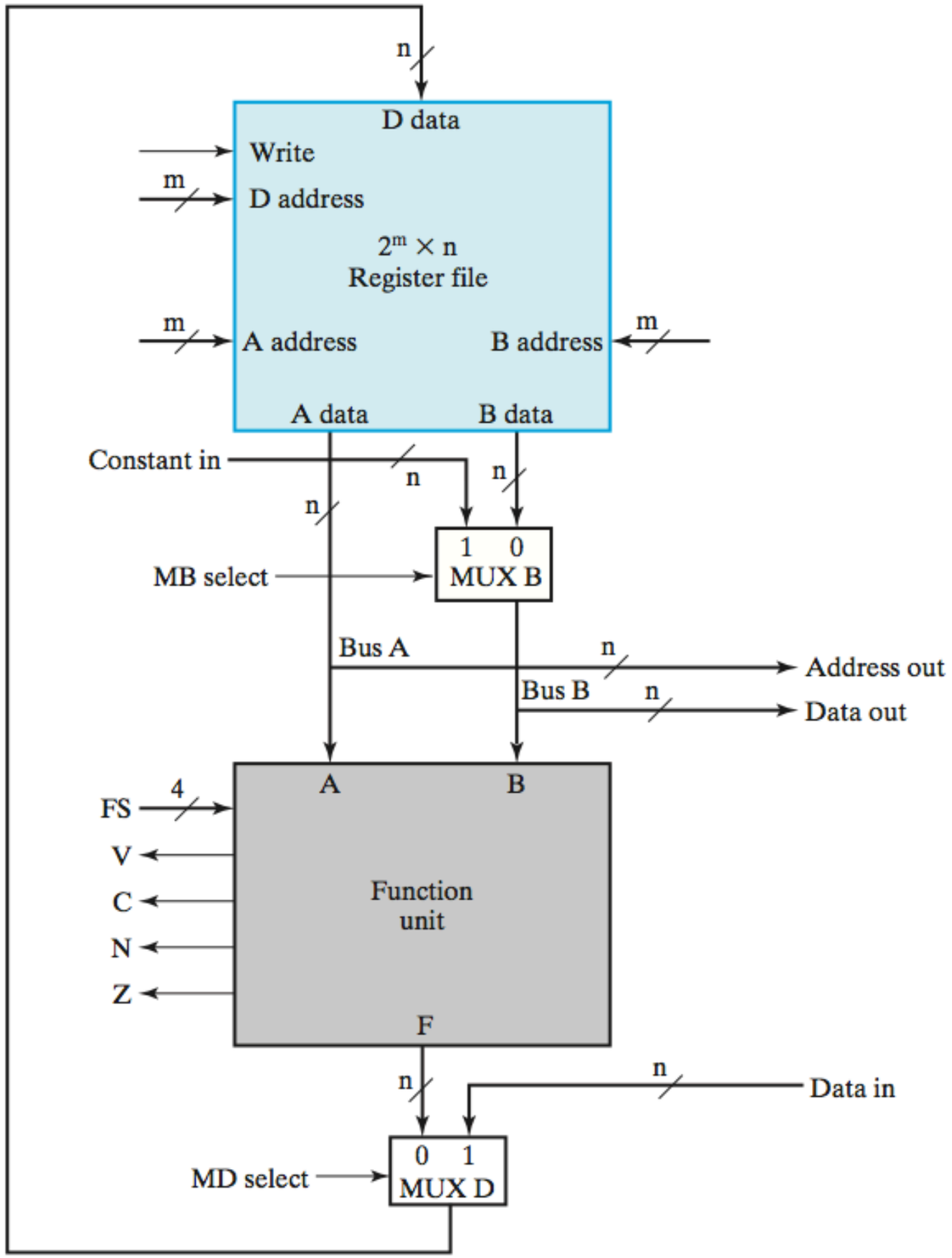
Representación del *Datapath*

- Los registros y los multiplexores, decodificadores y habilitadores para accesarlos son ahora el conjunto de registros (*register file*)
- La ALU, desplazador, Mux F y hardware de status son ahora la unidad funcional.
- Los restantes multiplexores y buses que conducen la transferencia de datos están ahora en este nuevo nivel de jerarquía.



Representación del *Datapath*

- En el conjunto de registros:
  - A address y B address son las líneas de selección de los Muxes
  - D address es la entrada del decodificador
  - Las salidas de los Muxes son A data y B data
  - La entrada de datos a los registros es D data
  - write es load enable
- El conjunto de registros aparece entonces como una memoria en bases a flip-flops.
- Los *labels* de la unidad funcional son bastante obvios excepto por FS.



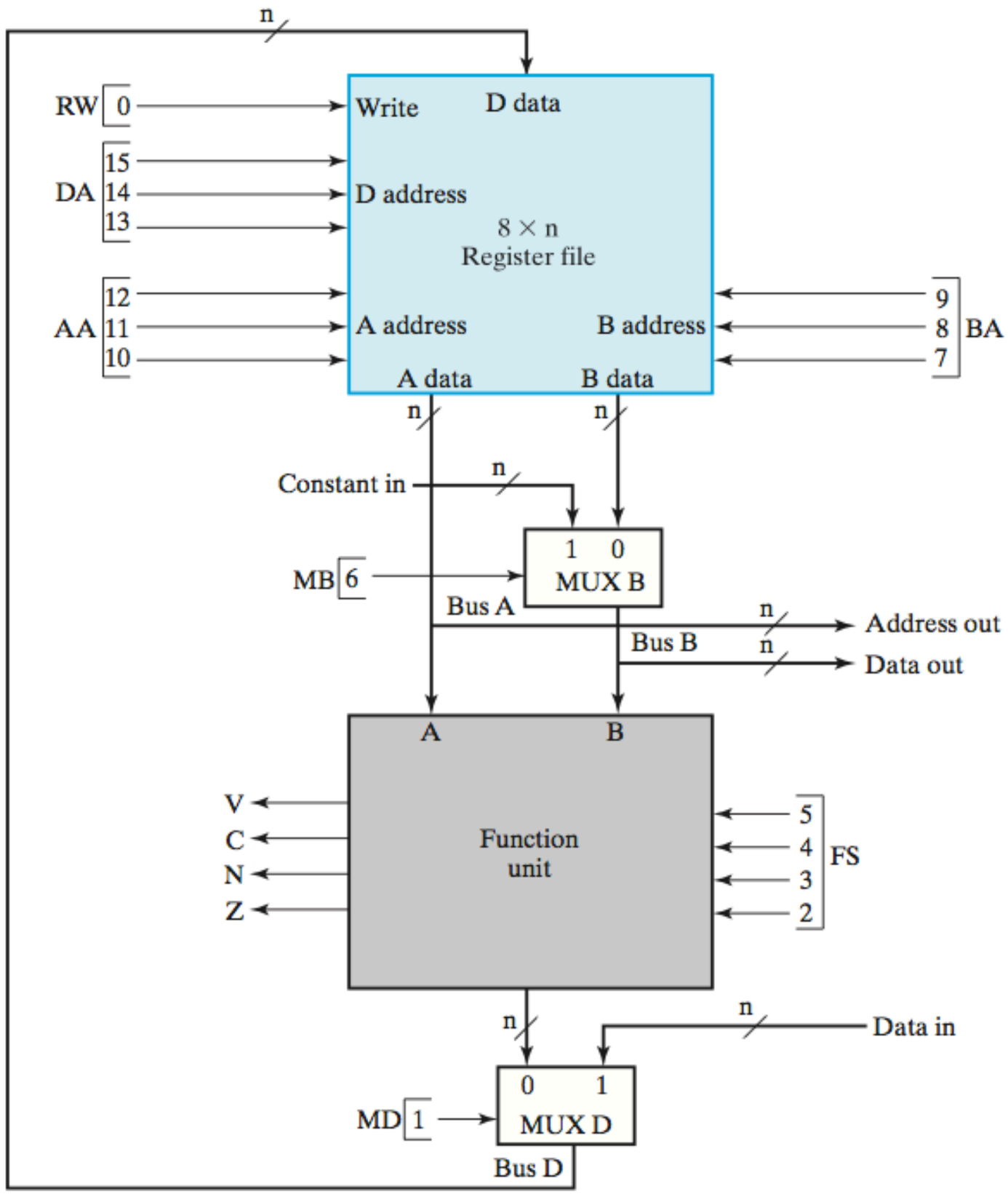
Definición de FS

FS(3:0)	MF Select	G Select(3:0)	H Select(3:0)	Microoperation
0000	0	0000	XX	$F = A$
0001	0	0001	XX	$F = A + 1$
0010	0	0010	XX	$F = A + B$
0011	0	0011	XX	$F = A + \overline{B} + 1$
0100	0	0100	XX	$F = A + B$
0101	0	0101	XX	$F = A + \overline{B} + 1$
0110	0	0110	XX	$F = A - 1$
0111	0	0111	XX	$F = A$
1000	0	1X00	XX	$F = A \wedge B$
1001	0	1X01	XX	$F = A \vee B$
1010	0	1X10	XX	$F = A \oplus B$
1011	0	1X11	XX	$F = \overline{A}$
1100	1	XXXX	00	$F = B$
1101	1	XXXX	01	$F = \text{sr } B$
1110	1	XXXX	10	$F = \text{sl } B$

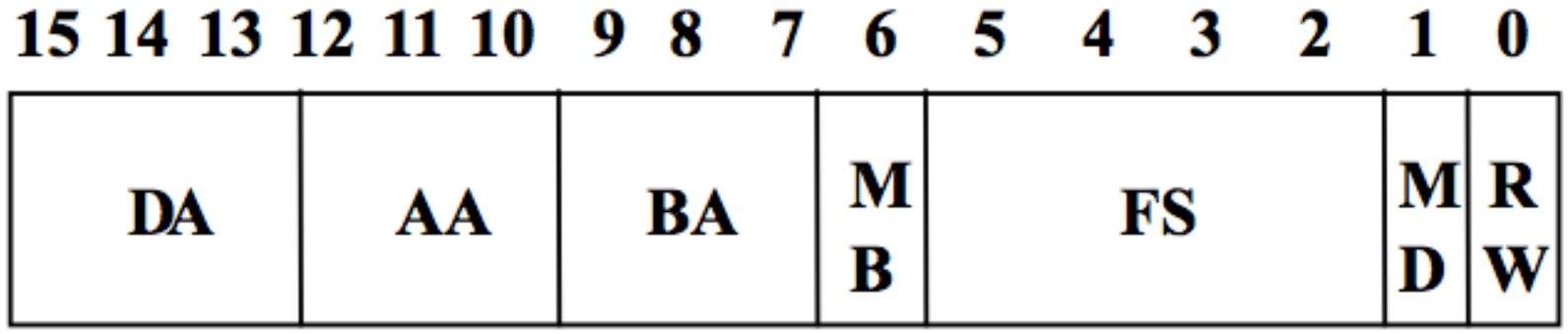
Palabra de Control

- La unidad de datos (*data path*) tiene un gran número de líneas de control
- Las señales que controlan estas entradas pueden ser definidas y organizadas en una palabra de control
- Para ejecutar una micro-instrucción aplicamos valores a la palabra de control durante un período de reloj. Para la mayor parte de las micro-operaciones, el flanco positivo del *clock* permite la carga del resultado en el registro especificado.

Palabra de Control



Campos de la Palabra de Control



- Campos
  - DA - D *Address*
  - AA - A *Address*
  - BA - B *Address*
  - MB - *Mux* B
  - FS - *Function Select*
  - MD - *Mux* D
  - RW - *Register Write*

Codificación de la Palabra de Control

DA, AA, BA		MB		FS		MD		RW	
Function	**Code**	**Function**	**Code**	**Function**	**Code**	**Function**	**Code**	**Function**	**Code**
$R0$	000	Register	0	$F = A$	0000	Function	0	No Write	0
$R1$	001	Constant	1	$F = A + 1$	0001	Data in	1	Write	1
$R2$	010			$F = A + B$	0010				
$R3$	011			$F = A + B + 1$	0011				
$R4$	100			$F = A + \overline{B}$	0100				
$R5$	101			$F = A + \overline{B} + 1$	0101				
$R6$	110			$F = A - 1$	0110				
$R7$	111			$F = A$	0111				
				$F = A \wedge B$	1000				
				$F = A \vee B$	1001				
				$F = A \oplus B$	1010				
				$F = \overline{A}$	1011				
				$F = B$	1100				
				$F = \text{sr } B$	1101				
				$F = \text{sl } B$	1110				

Ejemplo de Microoperaciones

Representación Simbólica

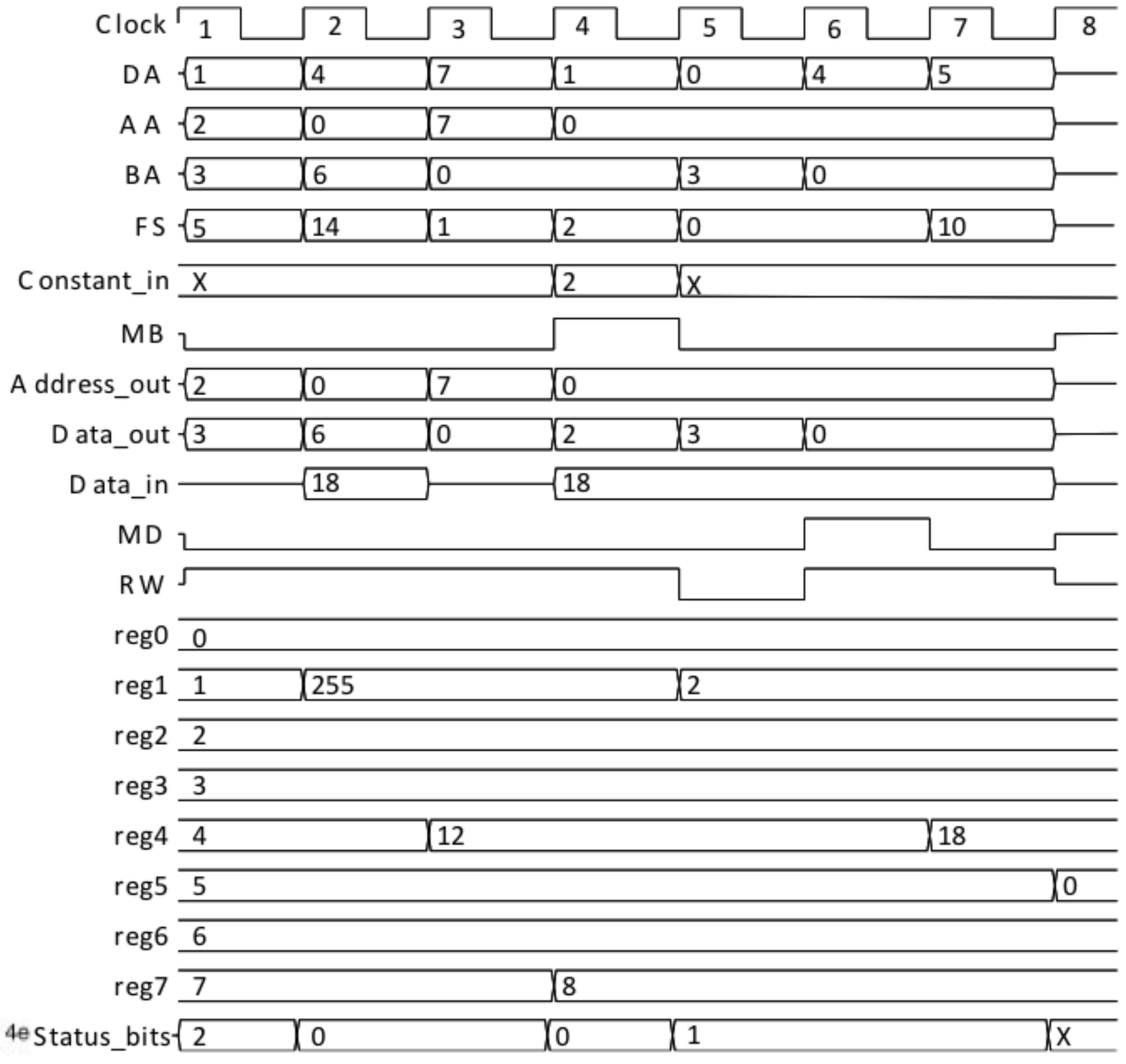
Micro-operation	DA	AA	BA	MB	FS	MD	RW
$R1 \leftarrow R2 - R3$	$R1$	$R2$	$R3$	Register	$F = A + \overline{B} + 1$	Function	Write
$R4 \leftarrow$ sl R6	$R4$	—	$R6$	Register	$F =$ sl B	Function	Write
$R7 \leftarrow R7 + 1$	$R7$	$R7$	—	Register	$F = A + 1$	Function	Write
$R1 \leftarrow R0 + 2$	$R1$	$R0$	—	Constant	$F = A + B$	Function	Write
Data out $\leftarrow R3$	——		$R3$	Register	—	—	No Write
$R4 \leftarrow$ Data in	$R4$	——		—	—	Data in	Write
$R5 \leftarrow 0$	$R5$	$R0$	$R0$	Register	$F = A \oplus B$	Function	Write

Ejemplo de Microoperaciones

Representación Binaria

Micro-operation	DA	AA	BA	MB	FS	MD	RW
$R1 \leftarrow R2 - R3$	001	010	011	0	0101	0	1
$R4 \leftarrow$ sl R6	100	XXX	110	0	1110	0	1
$R7 \leftarrow R7 + 1$	111	111	XXX	0	0001	0	1
$R1 \leftarrow R0 + 2$	001	000	XXX	1	0010	0	1
Data out $\leftarrow R3$	XXX	XXX	011	0	XXXX	X	0
$R4 \leftarrow$ Data in	100	XXX	XXX	X	XXXX	1	1
$R5 \leftarrow 0$	101	000	000	0	1010	0	1

Simulación de la Secuencia de Micro-operaciones



Programa

- 1ra Parte - Unidad de Flujo de Datos (*Data Path*)
- 2a Parte - Un Computador Simple (CS)
  - Arquitectura del conjunto de instrucciones
  - Control alambrado de un ciclo
  - Función del *Program Counter* (PC)
  - Decodificador de instrucciones
  - Ejemplo de ejecución de instrucciones
- 3a Parte - Control Alambrado de Múltiples Cíclos

Conjunto de Instrucciones

- Un sistema programable utiliza una secuencia de instrucciones para controlar su operación.
- Una instrucción típica especifica:
  - Operación a realizar
  - Operandos a utilizar, y
  - Donde poner el resultado, o
  - Qué instrucción realizar a continuación.
- Las instrucciones se almacenan en RAM o ROM como un programa.
- Las direcciones para las instrucciones en un computador las provee el PC (program counter), que puede:
  - Contar en forma ascendente
  - Cargar una nueva dirección en base a una instrucción y, opcionalmente, a la información de los bits de status.

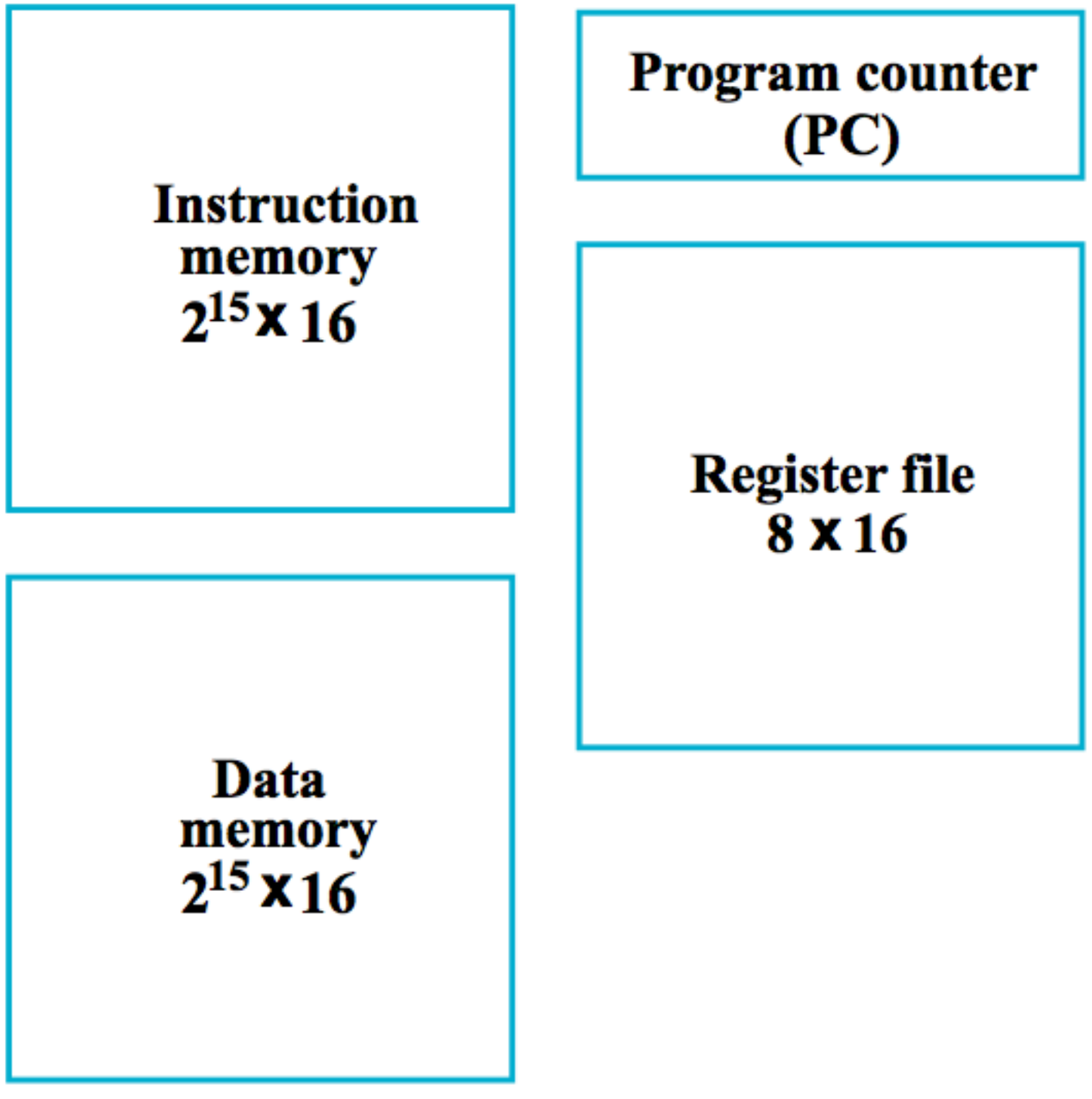
Conjunto de Instrucciones

- El PC y la lógica de control asociada son parte de la unidad de control.
- Ejecutar una instrucción consiste en activar la secuencia de operaciones necesarias especificada por la instrucción
- La ejecución es controlada por la unidad de control y realizada:

- En la unidad de datos (*data path*)
- En la unidad de control
- En hardware externo como memoria o *input/output*

Recursos de Almacenamiento

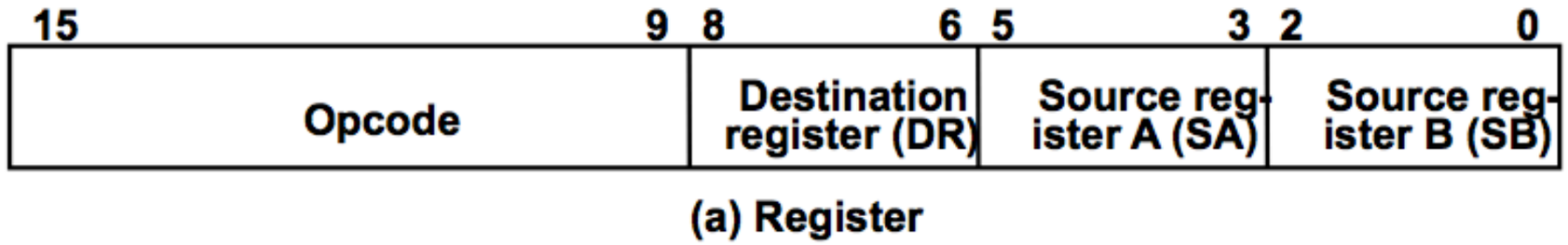
- Los recursos de almacenamiento son visibles al programador al menor nivel de software (lenguaje de máquina / assembler)
- Recursos del CS ==>
- La arquitectura Harvard especifica memorias separadas para instrucciones y datos
- Esto permite un sólo ciclo de reloj por instrucción
- Debido al uso de "cache" en los computadores modernos, es un modelo bien realista.



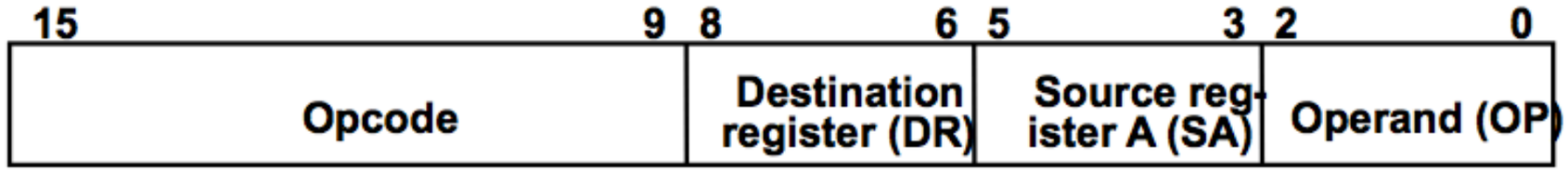
Formato de las Instrucciones

- Una instrucción es un vector de bits
- Los campos de una instrucción son subvectores que representan funciones específicas y corresponden a códigos binarios específicos
- El formato de una instrucción define los subvectores y su función.
- Un conjunto de instrucciones contiene múltiples formatos.
- Nuestro CS contiene sólo tres formatos.

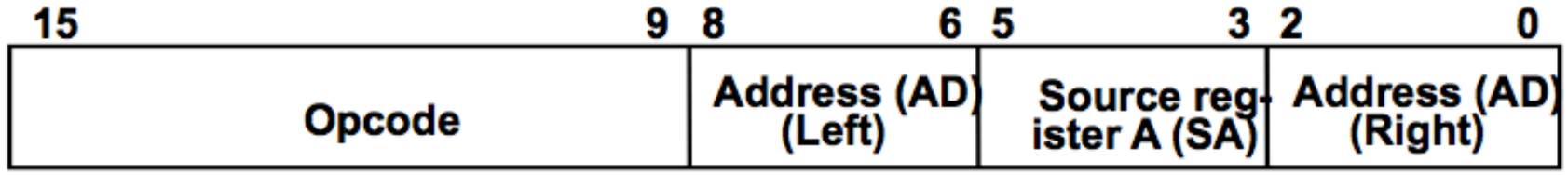
Formato de las Instrucciones



(a) Register



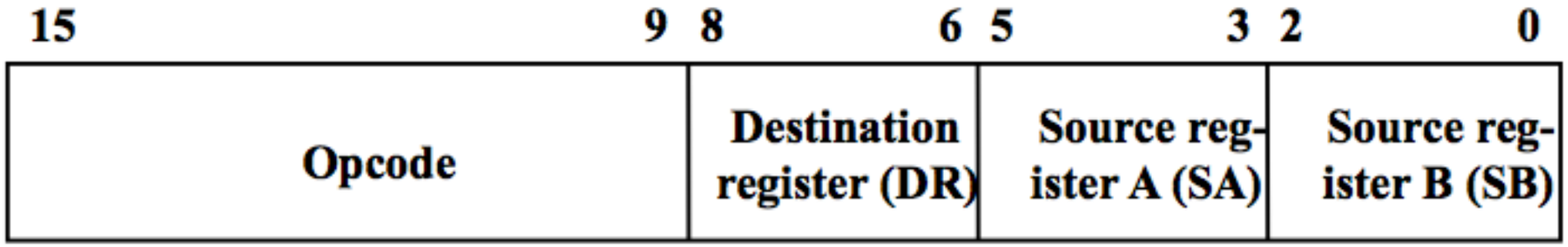
(b) Immediate



(c) Jump and Branch

- Los tres formatos son: Registro, Inmediata, y *jump* y *Branch*
- Todos los formatos contienen un campo *Opcode* en los bits 9 a 15
- El *Opcode* especifica la operación a realizar

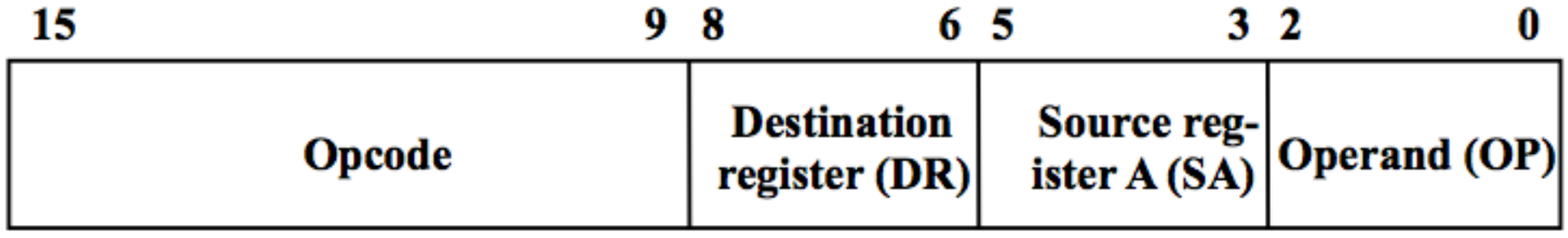
Formato de las Instrucciones



Registro

- Este formato soporta instrucciones del tipo:
  - R1 <== R2 + R3
  - R1 <== SI R2
- Hay tres campos de 3 bits c/u:
  - DR especifica el destino (R1 en los ejemplos)
  - SA especifica la fuente A (R2 en el primer ejemplo)
  - SB especifica la fuente B (R3 en primer ejemplo y R2 en el segundo)
- Porqué R2 en el segundo ejemplo es SB y no SA ?

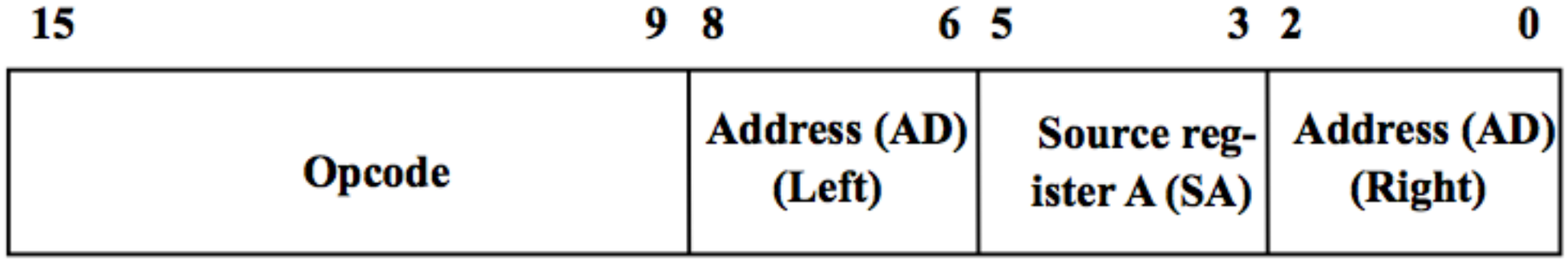
Formato de las Instrucciones



Inmediata

- Este formato soporta instrucciones del tipo:
  - R1 <== R2 + 3
- El campo SB se reemplaza por un campo OP que especifica una constante
- El operando
- Es una constante de 3 bits
- Puede tomar valores de 0 a 7
- La constante se llena de ceros a la izquierda para formar una palabra de 16 bits

Formato de las Instrucciones



Jump y Branch

- Este formato soporta instrucciones de salto agregando un *offset* de 6 bits con signo que se suma al contenido del PC
  - PC <== PC + 3
- El *offset* reemplaza los campos DR y SB
- El campo SA se mantiene para permitir saltos condicionales utilizando Z o N basados en el contenido del registro fuente A.

Especificación de la Instrucción

- La especificación de la instrucción provee:
  - El nombre de la instrucción
  - El código de operación (*opcode*)
  - Un nombre corto para la operación llamado *mnemonic*
  - Una especificación para el formato de la instrucción
  - Una descripción de transferencia de registros de la instrucción
  - Un listado de los bit de estado (*status bits*) que tienen significado durante la ejecución de una instrucción (no utilizado en la arquitectura que hemos definido aquí)

Especificaciones de las Instrucciones

Instruction	Opcode	Mne-monic	Format	Description	Status Bits
Move A	0000000	MOVA	RD, RA	R[DR] ← R[SA]*	N, Z
Increment	0000001	INC	RD, RA	R[DR] ← R[SA] + 1*	N, Z
Add	0000010	ADD	RD, RA, RB	R[DR] ← R[SA] + R[SB]*	N, Z
Subtract	0000101	SUB	RD, RA, RB	R[DR] ← R[SA] – R[SB]*	N, Z
Decrement	0000110	DEC	RD, RA	R[DR] ← R[SA] – 1*	N, Z
AND	0001000	AND	RD, RA, RB	R[DR] ← R[SA] ∧ R[SB]*	N, Z
OR	0001001	OR	RD, RA, RB	R[DR] ← R[SA] ∨ R[SB]*	N, Z
Exclusive OR	0001010	XOR	RD, RA, RB	R[DR] ← R[SA] ⊕ R[SB]*	N, Z
NOT	0001011	NOT	RD, RA	R[DR] ← $\overline{\text{R[SA]}}$ *	N, Z
Move B	0001100	MOVB	RD, RB	R[DR] ← R[SB]*	
Shift Right	0001101	SHR	RD, RB	R[DR] ← sr R[SB]*	
Shift Left	0001110	SHL	RD, RB	R[DR] ← sl R[SB]*	
Load Immediate	1001100	LDI	RD, OP	R[DR] ← zf OP*	
Add Immediate	1000010	ADI	RD, RA, OP	R[DR] ← R[SA] + zf OP*	N, Z
Load	0010000	LD	RD, RA	R[DR] ← M[SA]*	
Store	0100000	ST	RA, RB	M[SA] ← R[SB]*	
Branch on Zero	1100000	BRZ	RA, AD	if (R[SA] = 0) PC ← PC + se AD, if (R[SA] ≠ 0) PC ← PC + 1	N, Z
Branch on Negative	1100001	BRN	RA, AD	if (R[SA] < 0) PC ← PC + se AD, if (R[SA] ≥ 0) PC ← PC + 1	N, Z
Jump	1110000	JMP	RA	PC ← R[SA]	

* For all of these instructions, PC ← PC + 1 is also executed to prepare for the next cycle.

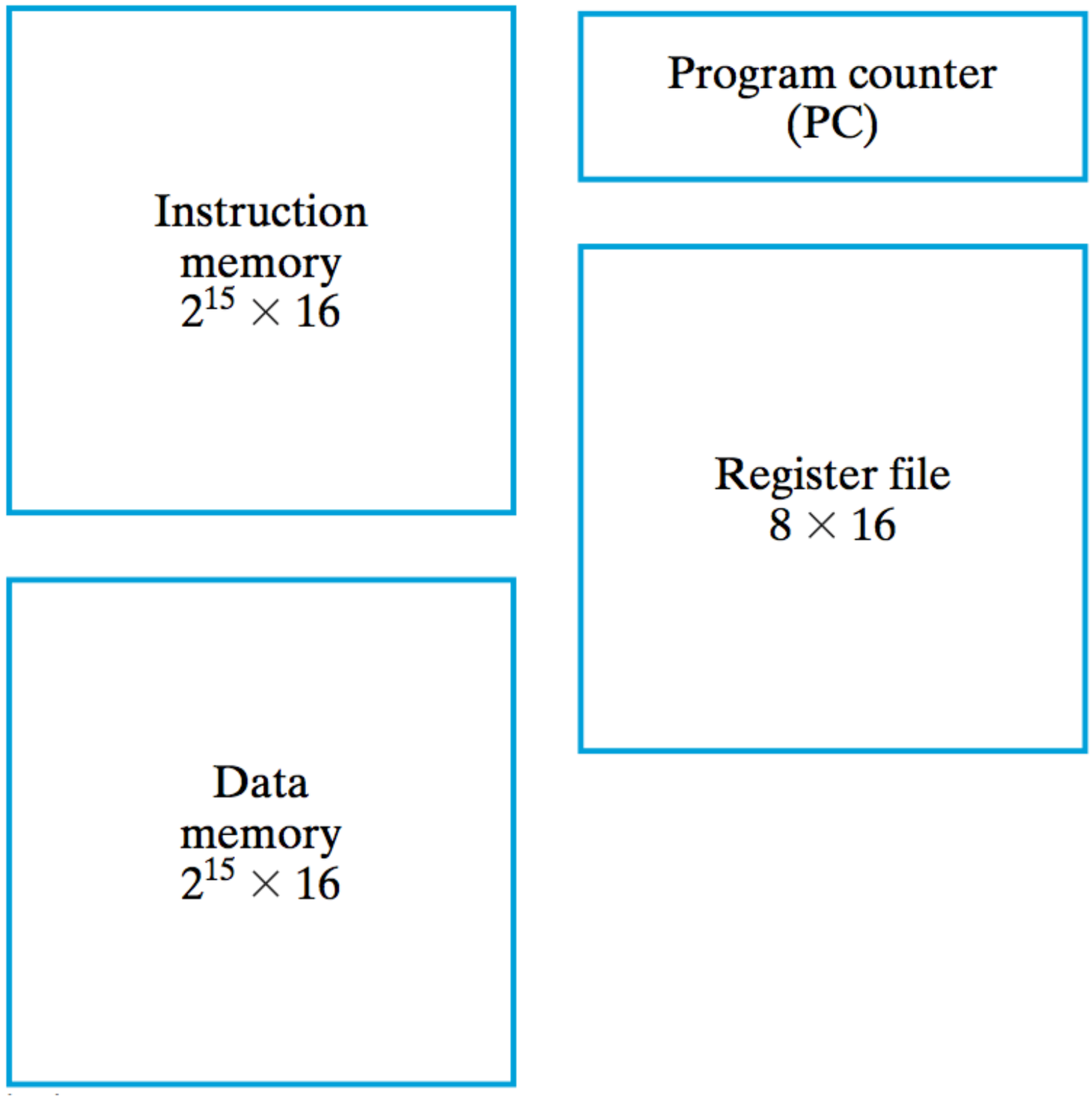
Representación en Memoria de Instrucciones y Datos

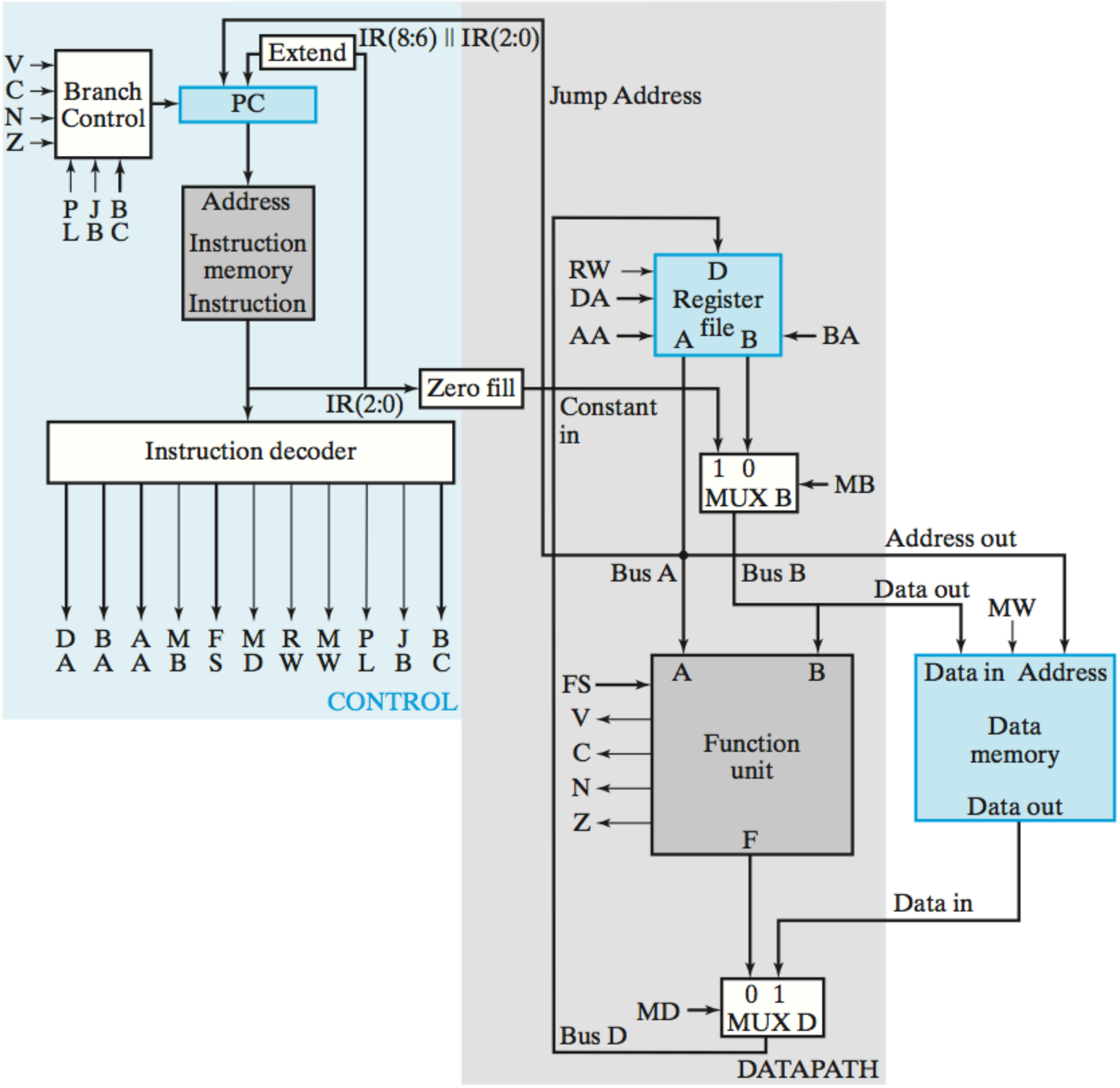
Decimal Address	Memory Contents	Decimal Opcode	Other Fields	Operation
25	0000101 001 010 011	5 (Subtract)	DR:1, SA:2, SB:3	R1 ← R2 – R3
35	0100000 000 100 101	32 (Store)	SA:4, SB:5	M[R4] ← R5
45	1000010 010 111 011	66 (Add Immediate)	DR:2, SA:7, OP:3	R2 ← R7 + 3
55	1100000 101 110 100	96 (Branch on Zero)	AD: 44, SA:6	If R6 = 0, PC ← PC – 20
70	0000000011000000	Data = 192. After execution of instruction in 35, Data = 80.		

Control Alambrado de un Ciclo

- En base a la arquitectura de instrucciones definida, diseñemos un arquitectura de hardware (computador) que la soporte.
- La arquitectura debe buscar y ejecutar cada instrucción en un solo ciclo de reloj
- Utilizaremos la unidad de datos (*datapath*) vista
- La unidad de control será definida como parte del diseño.

Recordemos los Recursos de Almacenamiento




V
C
N
Z
Branch Control
P L
J B
B C
PC
Extend
IR(8:6) || IR(2:0)
Jump Address
Address
Instruction memory
Instruction
IR(2:0)
Zero fill
Constant in
Instruction decoder
RW
DA
AA
D Register file
A
B
BA
1 0
MUX B
MB
Address out
Bus A
Bus B
Data out
MW
FS
V
C
N
Z
A
B
Function unit
F
Data in
Address
Data memory
Data out
Data in
0 1
MUX D
MD
Bus D
D A
B A
A A
M B
F S
M D
R W
M W
P L
J B
B C
CONTROL
DATAPATH

Unidad de Control

- La memoria de datos ha sido conectada al bus *Address Out* y a los buses *Data Out* y *Data In* de la unidad de datos.
- La entrada MW a la memoria de datos es la señal *memory write* de la unidad de control
- Por conveniencia, la memoria de instrucción, que normalmente no forma parte de la unidad de control, se muestra dentro de ella.
- La entrada de direccionamiento de la memoria de instrucciones es entregada por el PC y la salida de esta alimenta al decodificador de instrucciones
- *Constant In* es IR(2:0) con 13 ceros a la izquierda.
- IR(8:6)||IR(2:0) y el bus A son entradas de direccionamiento al PC
- El PC es controlado por la lógica de control de *Branch*

Función del PC

- La función del PC se basa en las instrucciones de *Branch* y *Jump*.

Branch on zero	BRZ	if(R[SA]=0) PC <== PC + seAD
Branch on negative	BRN	if(R[SA] < 0) PC <== PC + seAD
Jump	JMP	PC <== R[SA]

- Además de estas transferencias el PC debe implementar también PC <== PC + 1
- Las primeras dos transferencias requieren sumarle al PC el *offset* = IR(8:6)||IR(2:0)
- La tercera transferencia requiere que el PC sea cargado con la dirección de salto Bus A = R[SA]
- La función de cuenta del PC requiere sumar 1 al PC.

Función del PC

- La unidad de control de *Branch* determina las transferencias del PC en base a 5 de sus entradas:

  N , Z - bits de status negativo y cero
  PL - Habilitador de carga del PC
  JB - *Jump/Branch* select: if JB = 1, *Jump*, else *Branch*
  BC - Selector de condición para *Branch* If BC = 1, *Branch* si N = 1, else *Branch* si Z = 1.

- Todo esto se resume en esta tabla:

PC Operation	PL	JB	BC
Count Up	0	X	X
Jump	1	1	X
Branch on Negative (else Count Up)	1	0	1
Branch on Zero (else Count Up)	1	0	0

- Con esta información se puede diseñar el PC

Decodificador de Instrucciones

- El decodificador de instrucciones combinacional convierte las instrucciones en señales necesarias para controlar todas las partes del computador durante la ejecución de un solo ciclo.
- La entrada es la instrucción de 16 bits
- Las salidas son las señales de control
  - Direcciones del conjunto de registros: DA, AA y BA
  - Selector de la unidad funcional FS
  - Selectores de control de multiplexor MB y MD
  - Controles de escritura del conjunto de registros y de la memoria RW y MW
  - Controles del PC: PL, JB y BC
- Las salidas correspondientes al conjunto de registros pasan directamente
- La generación de las otras señales es más compleja

Decodificador de Instrucciones

- Las restantes señales de control no dependen de direcciones, son función de IR(13:9)
- Debemos examinar la relación entre las salidas y los códigos de operación
- Aparte de las instrucciones de *Jump* y *Branch*, FS = IR(12:9)
- Las otras señales de control deben depender tanto como sea posible en IR(15:13) (las cuales fueron asignadas teniendo en mente la decodificación)
- Para hacer sentido de esto, dividiremos las instrucciones en tipos como se muestra en la siguiente tabla:

Decodificador de Instrucciones

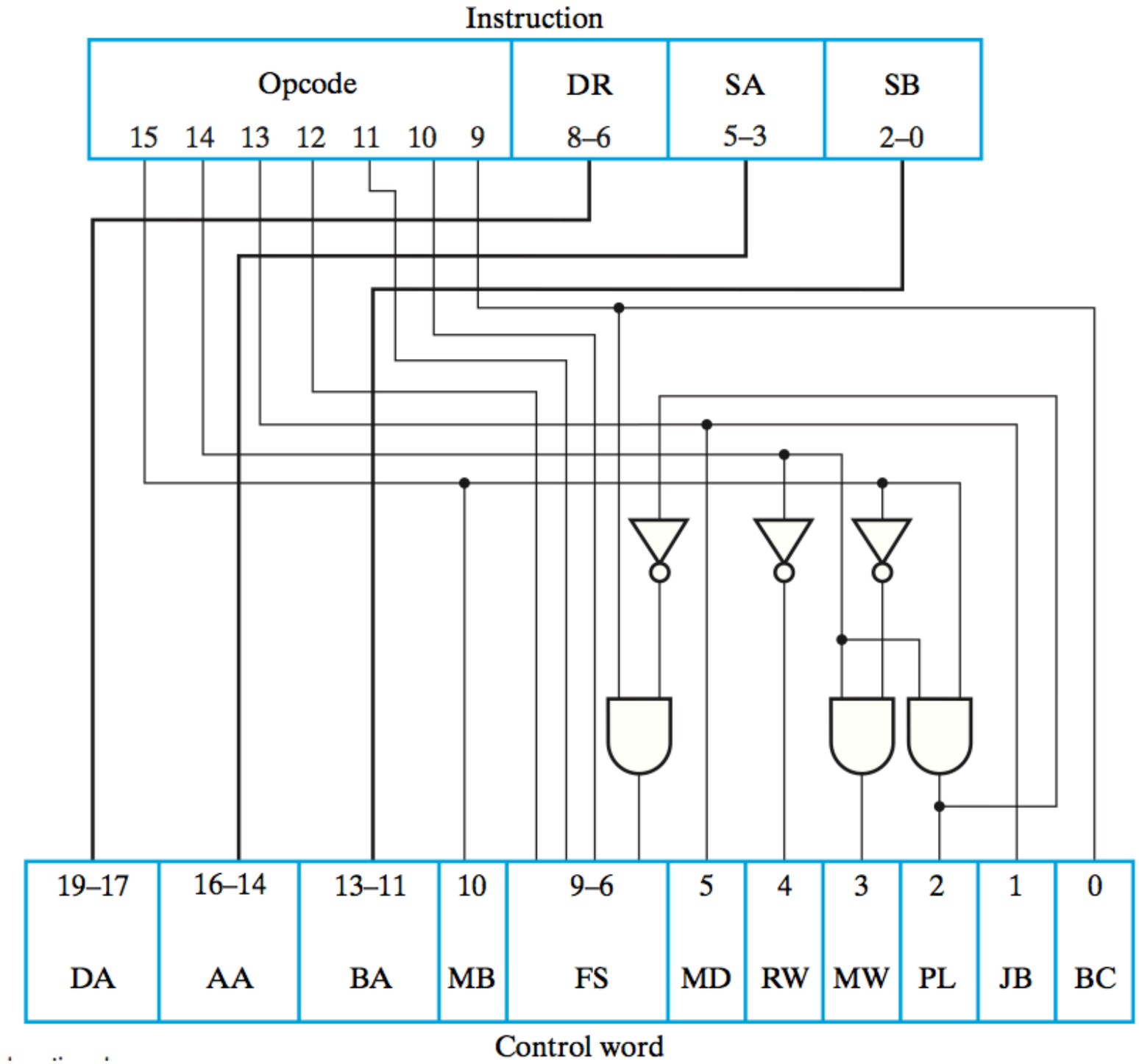
	Instruction Bits				Control Word Bits						
Instruction Function Type	**15**	**14**	**13**	**9**	**MB**	**MD**	**RW**	**MW**	**PL**	**JB**	**BC**
Function unit operations using registers	0	0	0	X	0	0	1	0	0	X	X
Memory read	0	0	1	X	0	1	1	0	0	X	X
Memory write	0	1	0	X	0	X	0	1	0	X	X
Function unit operations using register and constant	1	0	0	X	1	0	1	0	0	X	X
Conditional branch on zero (Z)	1	1	0	0	X	X	0	0	1	0	0
Conditional branch on negative (N)	1	1	0	1	X	X	0	0	1	0	1
Unconditional Jump	1	1	1	X	X	X	0	0	1	1	X

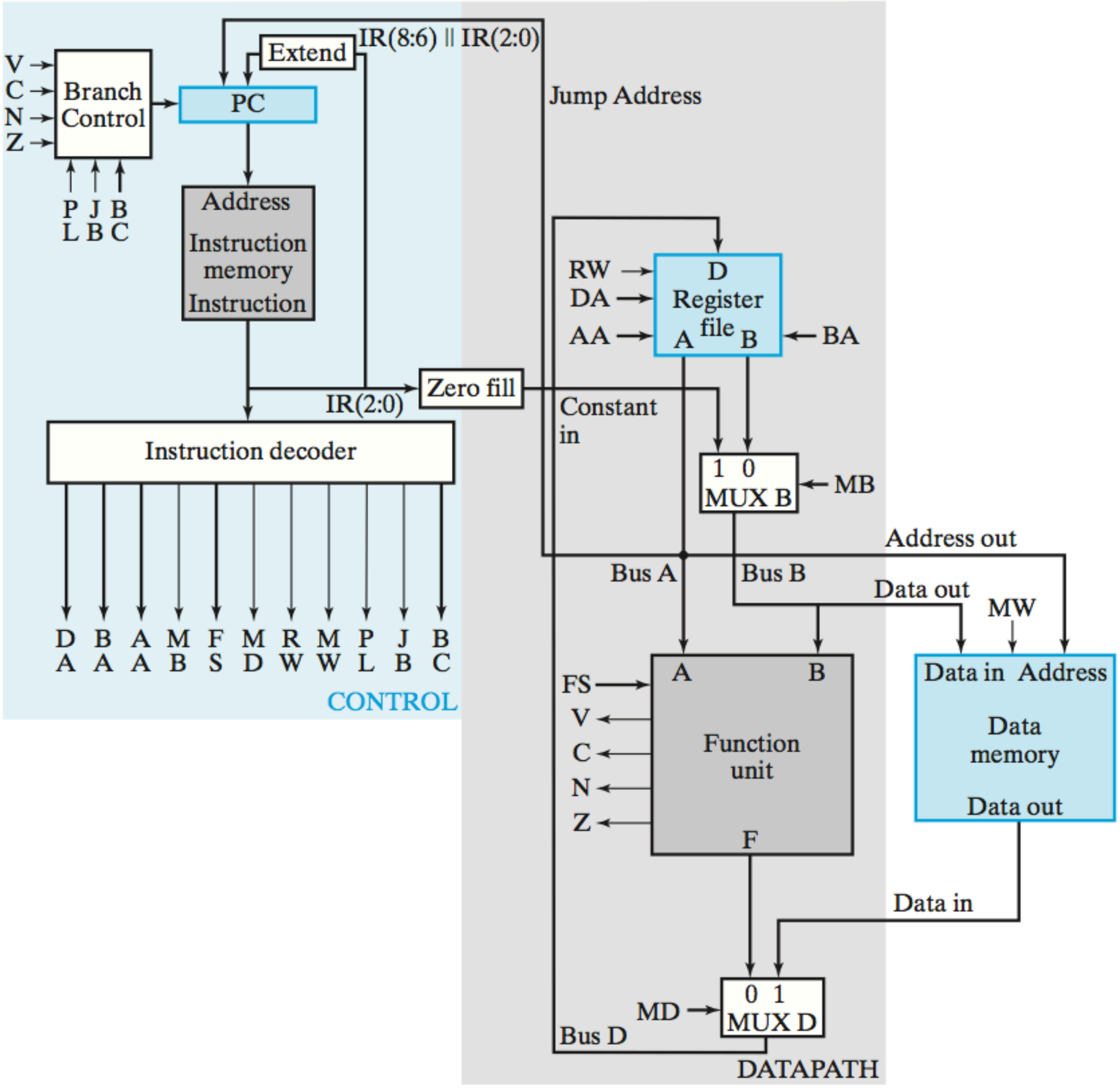
Decodificador de Instrucciones

- Los tipos se basan en los bloques que controlan y las 7 señales que hay que generar; los tipos de instrucción pueden dividirse en dos:
  - Control de la unidad de datos y de la memoria (primeros 4 tipos)
  - Control del PC (últimos tres tipos)
- En el control de la unidad de datos y de la memoria:
  - Mux B (primer y cuarto tipo)
  - Memoria y Mux D (segundo y tercer tipo)
  - Asignando códigos con ningún o un solo 1 a estos, la implementación de MB, MD, RW y MW se simplifica
- En la unidad de control del PC se asignan unos a las combinaciones
  - Bit 15 = Bit 14 = 1 se asignan para generar PL
  - Bit 13 se utiliza para generar JB
  - Bit 9 se utiliza como BC, lo que se contradice con FS = 0000 requerido para *Branches*. Para forzar FS(6) a 0 para *Branch* Bit 9 en FS(6) se deshabilita.

Decodificador de Instrucciones

- El resultado final, separando por tipos, asignando cuidadosamente los códigos y el uso de *don´t cares* conduce a una lógica muy simple
- Con esto completamos la mayoría de las partes esenciales del simple computador de un solo ciclo



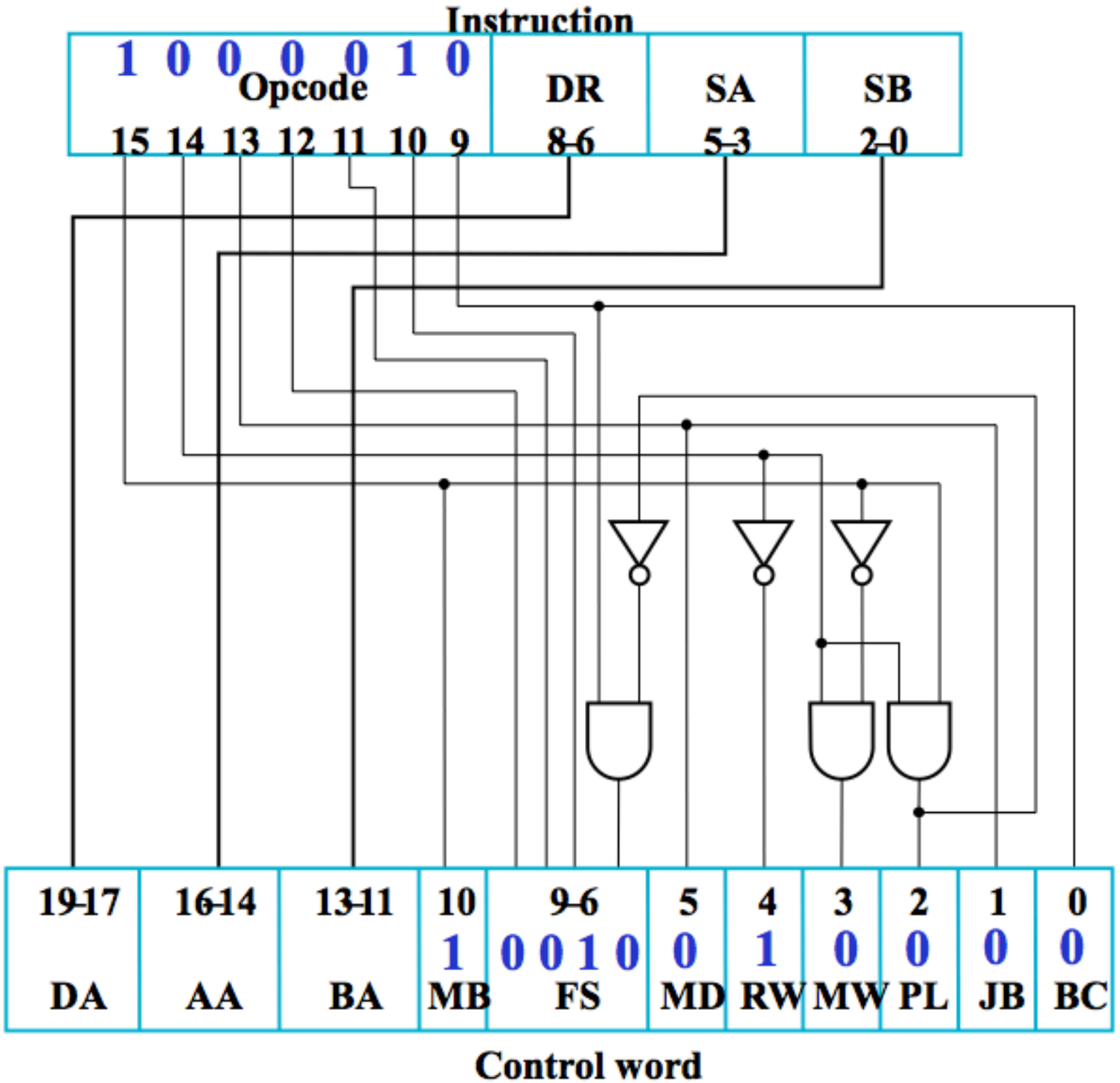

V
C
N
Z
Branch Control
P L
J B
B C
PC
Extend
IR(8:6) || IR(2:0)
Jump Address
Address
Instruction memory
Instruction
IR(2:0)
Zero fill
Constant in
Instruction decoder
RW
DA
AA
D Register file
A
B
BA
1 0
MUX B
MB
Address out
Bus A
Bus B
Data out
MW
FS
V
C
N
Z
A
B
Function unit
F
Data in
Address
Data memory
Data out
Data in
0 1
MUX D
MD
Bus D
D A
B A
A A
M B
F S
M D
R W
M W
P L
J B
B C
CONTROL
DATAPATH

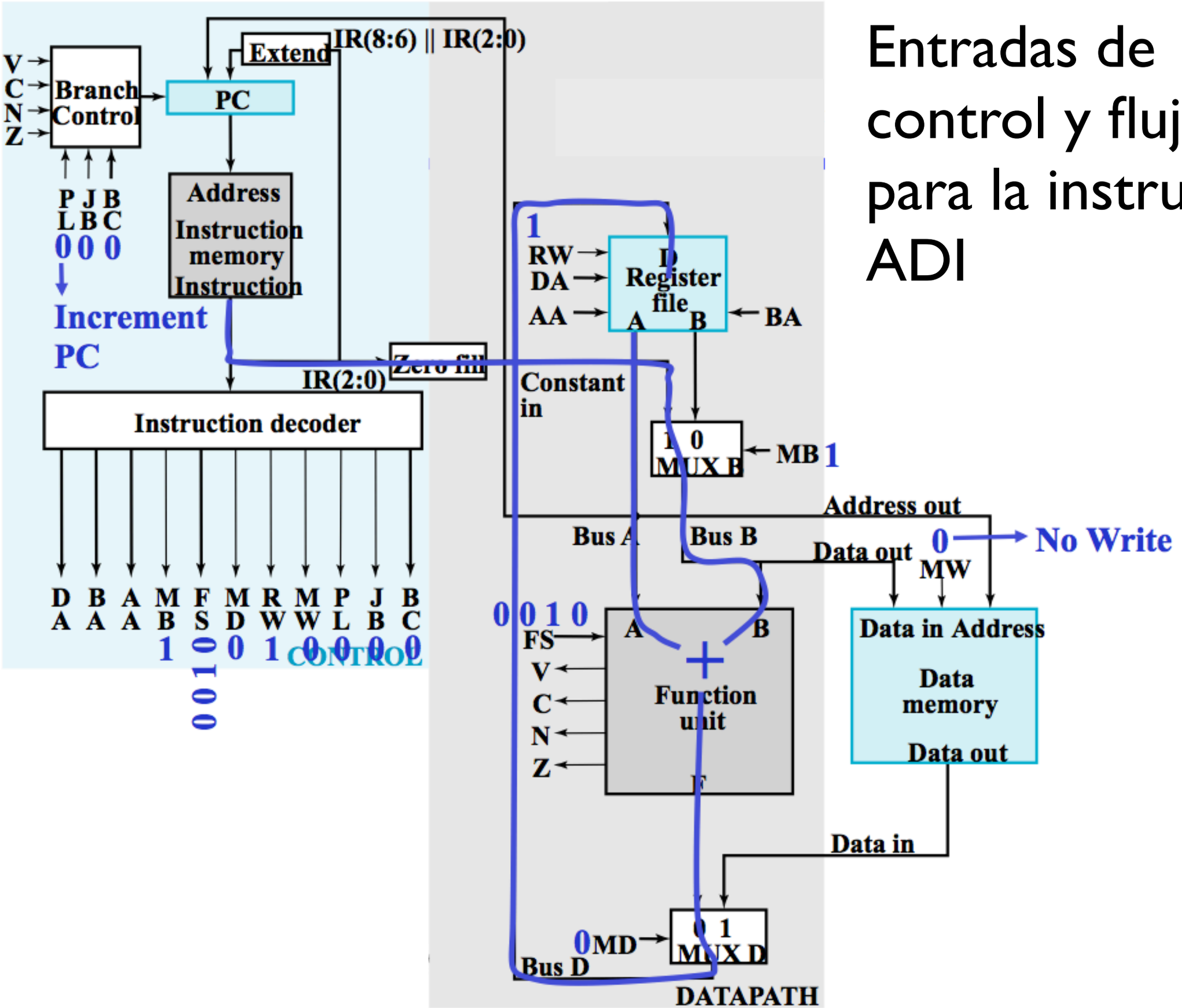
Ejemplos de Ejecución de Instrucciones

Operation Code	Symbolic Name	Format	Description	Function	MB	MD	RW	MW	PL	JB	BC
1000010	ADI	Immediate	Add immediate operand	$R[\text{DR}] \leftarrow R[\text{SA}] + \text{zf}\ I(2{:}0)$	1	0	1	0	0	0	0
0010000	LD	Register	Load memory content into register	$R[\text{DR}] \leftarrow M[R[\text{SA}]]$	0	1	1	0	0	1	0
0100000	ST	Register	Store register content in memory	$M[R[\text{SA}]] \leftarrow R[\text{SB}]$	0	1	0	1	0	0	0
0001110	SL	Register	Shift left	$R[\text{DR}] \leftarrow \text{sl}\ R[\text{SB}]$	0	0	1	0	0	1	0
0001011	NOT	Register	Complement register	$R[\text{DR}] \leftarrow \overline{R[\text{SA}]}$	0	0	1	0	0	0	1
1100000	BRZ	Jump/Branch	If $R[\text{SA}] = 0$, branch to $PC + \text{se}\ AD$	If $R[\text{SA}] = 0$, $PC \leftarrow PC + \text{se AD}$ If $R[\text{SA}] \neq 0$, $PC \leftarrow PC + 1$	1	0	0	0	1	0	0

Ejemplos de Ejecución de Instrucciones

Decodificación de la instrucción ADI

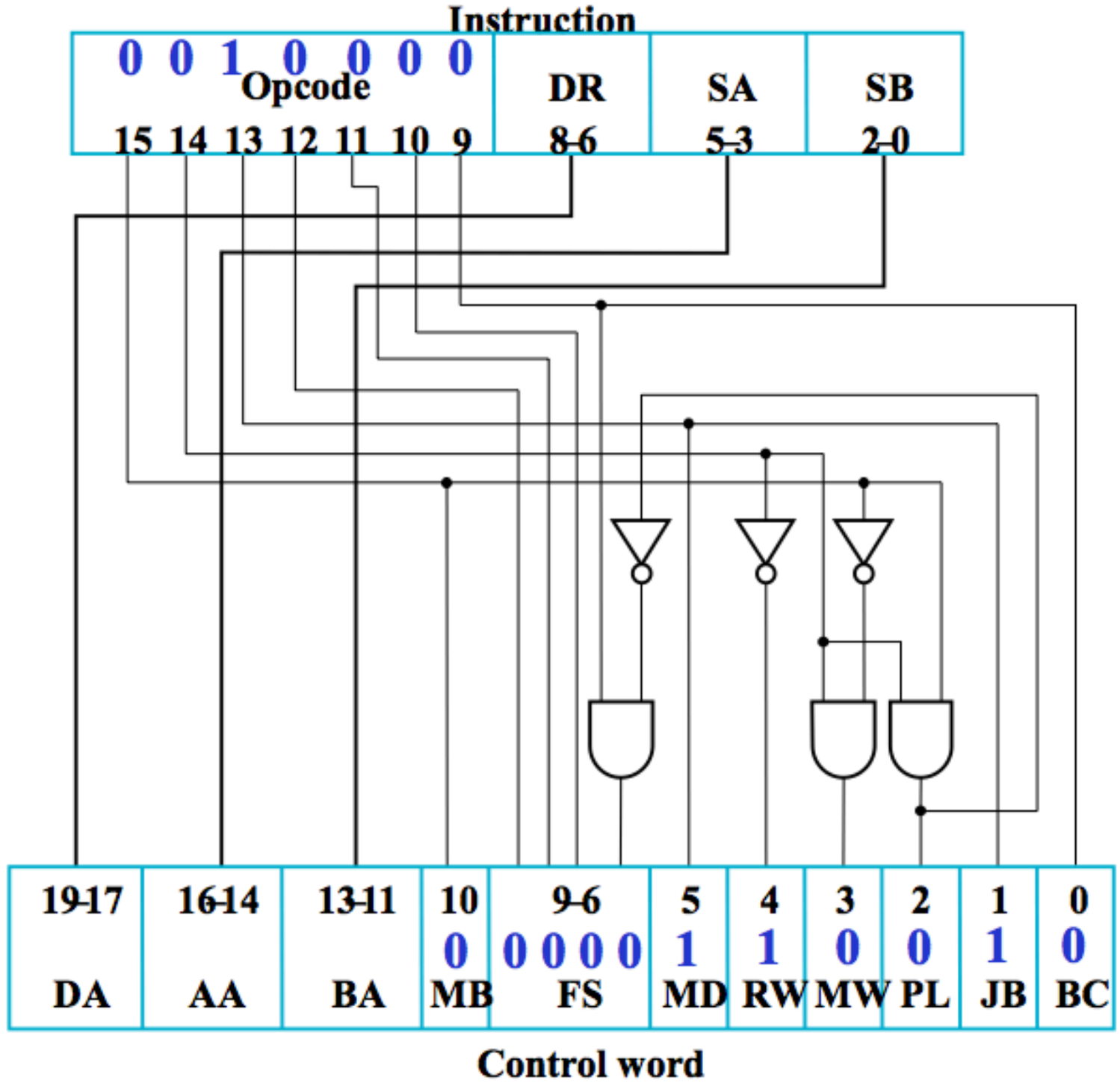




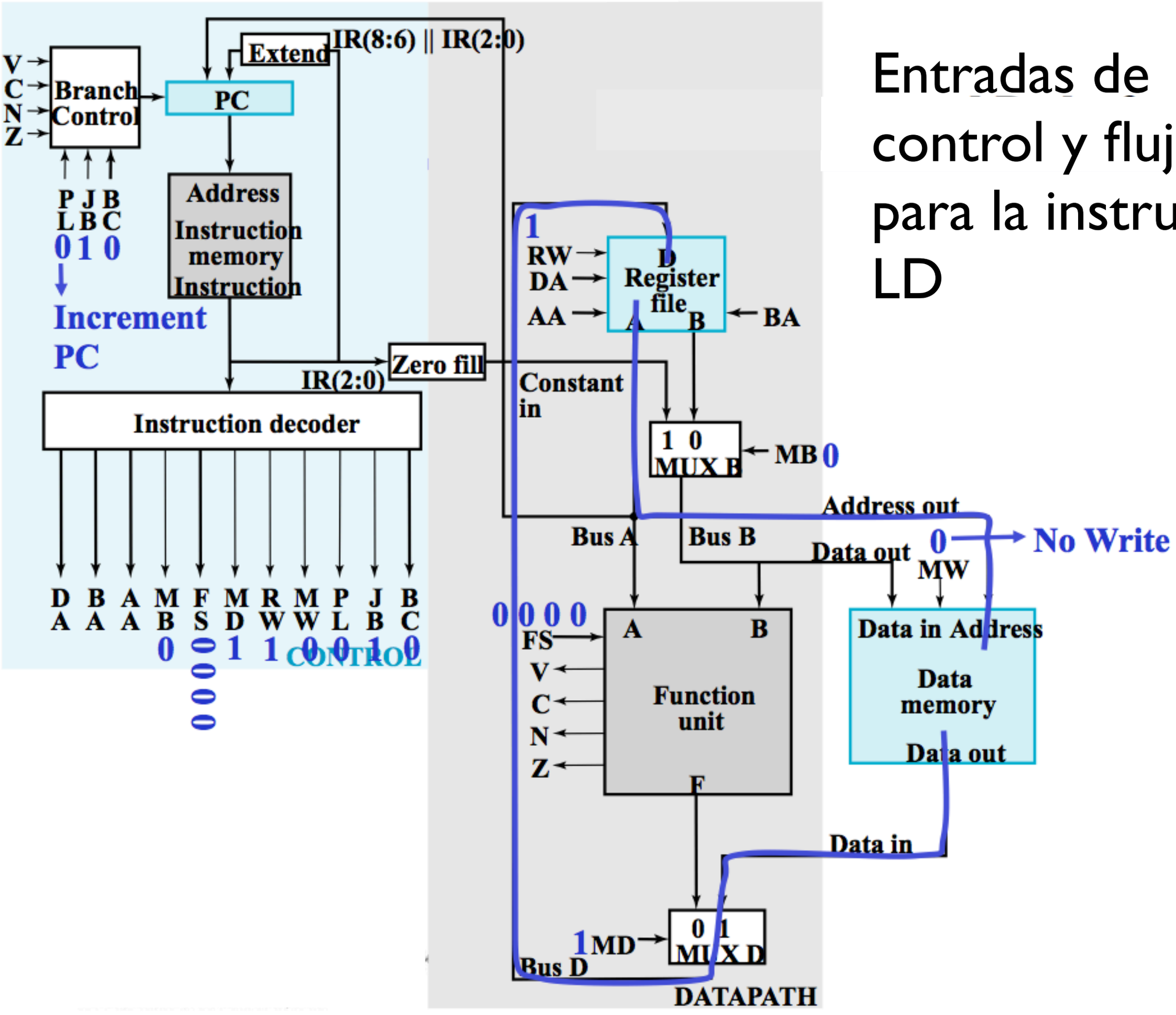
Entradas de control y flujos para la instrucción ADI

Ejemplos de Ejecución de Instrucciones

Decodificación de la instrucción LD

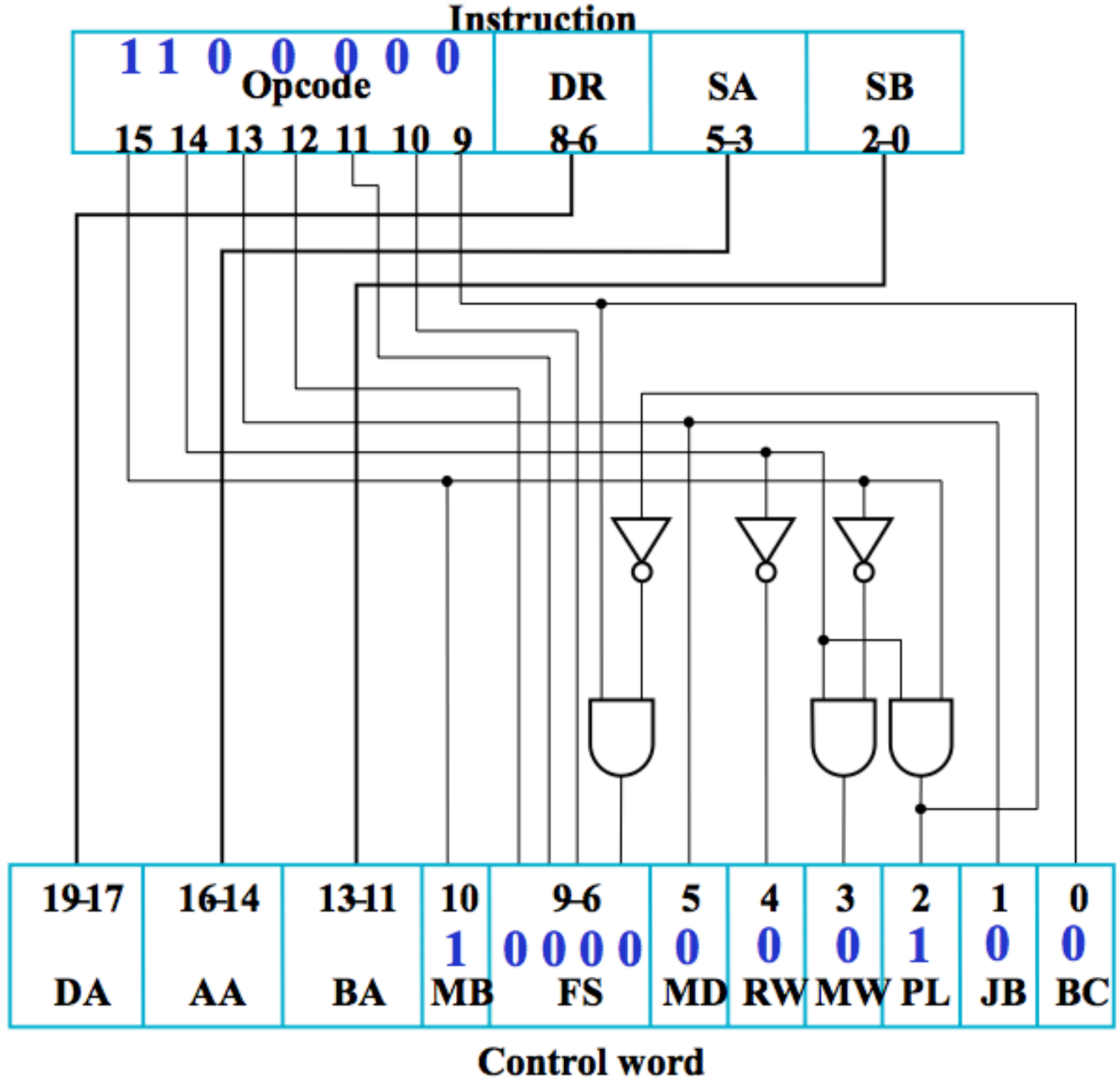


Entradas de control y flujos para la instrucción LD



Ejemplos de Ejecución de Instrucciones

Decodificación de la instrucción BRZ



Entradas de control y flujos para la instrucción BRZ