레지스터 전송과 마이크로 연산

4.1 레지스터 전송 언어

◈ 마이크로 연산 : 레지스터에 저장된 데이터를 가지고 실행되는 동작으로 한 클럭 펄스 동안에 실행되는 기본적인 동작 (시프트, 카운트, 클리어, 로드 등)

- 디지털 컴퓨터의 구조 정의를 위한 규정

  레지스터의 종류와 기능

  레지스터에 저장된 이진 정보를 가지고 수행하는 일련의 마이크로 연산

  일련의 마이크로 연산을 시작시키는 제어 기능

- 레지스터 전송 언어 : 레지스터간의 마이크로 연산 전송을 보다 간단하고 명료하게 표시하기 위해 사용하는 기호들

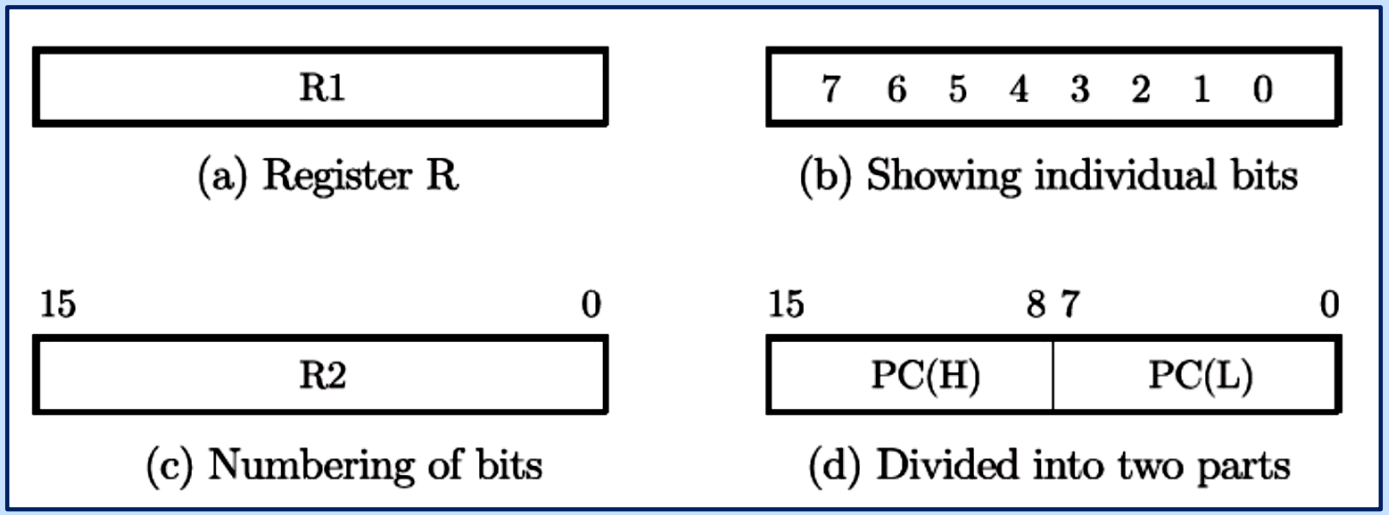
4.2 레지스터 전송

◈ 레지스터는 그 기능을 나타내기 위하여 머릿 글자들을 대문자로 표시

- 예 : MAR (메모리 주소 레지스터), PC (프로그램 카운터)

  IR (명령어 레지스터), R1 (프로세서 레지스터)

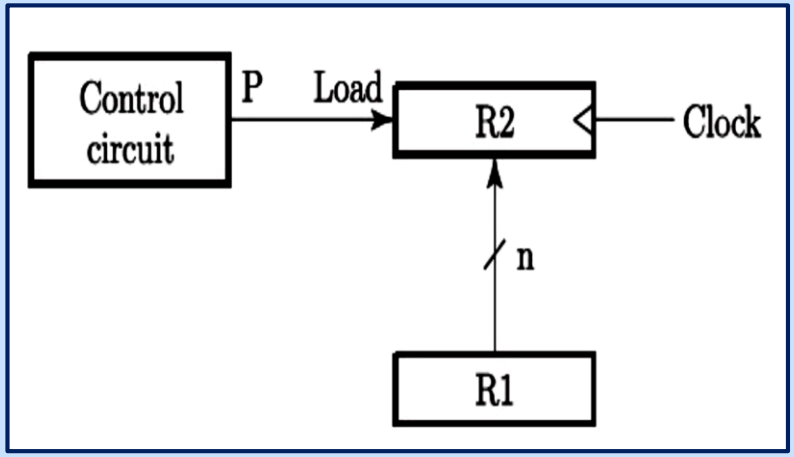
- 레지스터를 나타내는 일반적인 방법

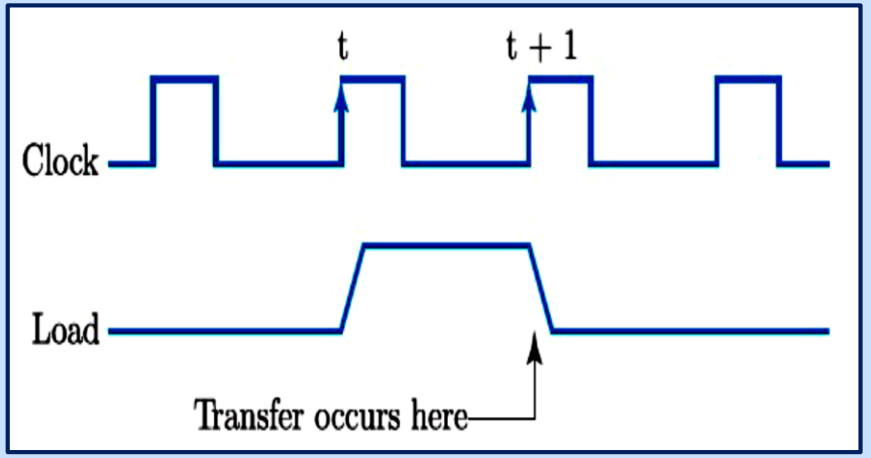


◈ 레지스터들 사이의 정보 전송은 치환 연산자를 이용하여 나타냄

- 일반적인 형태 : R2 ← R1 (R1에서 R2로 전송)
- 제어 함수가 포함된 전송 P : R2 ← R1 (P=1이면 R1에서 R2로 전송)

- 여러 개의 레지스터 전송 T : R2 ← R1, R1 ← R2 (T=1이면 R1과 R2 교환)
- 레지스터 전송을 나타내는 블록도와 레지스터 전송의 기본 기호

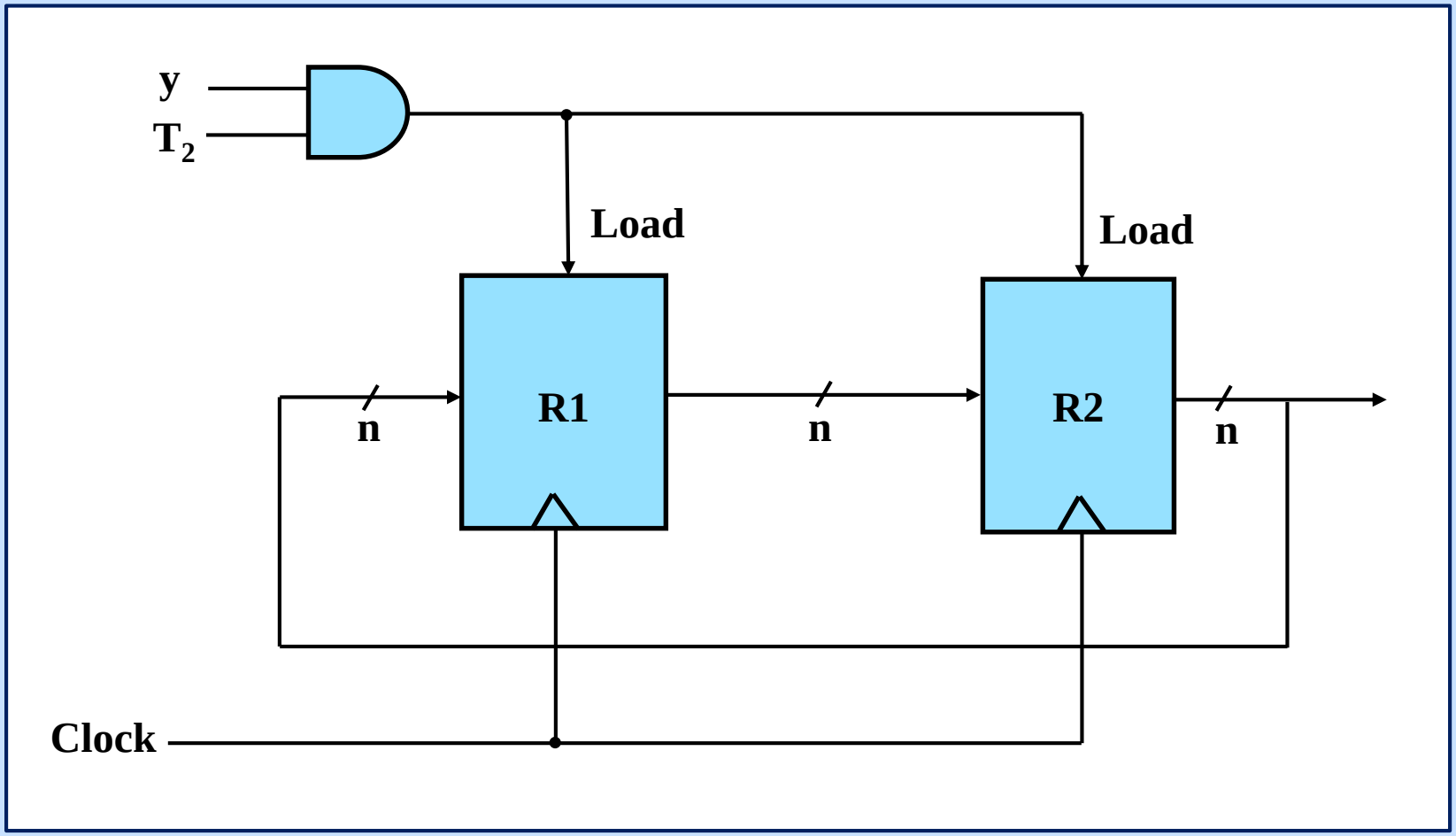




Symbol	Description	Examples
Letters(and numerals)	Denotes a register	MAR, R2
Parentheses( )	Denotes a part of a register	R2(0-7), R2(L)
Arrow←	Denotes transfer of information	R2 ← R1
Comma,	Separates two microoperations	R2 ← R1, R1 ← R2

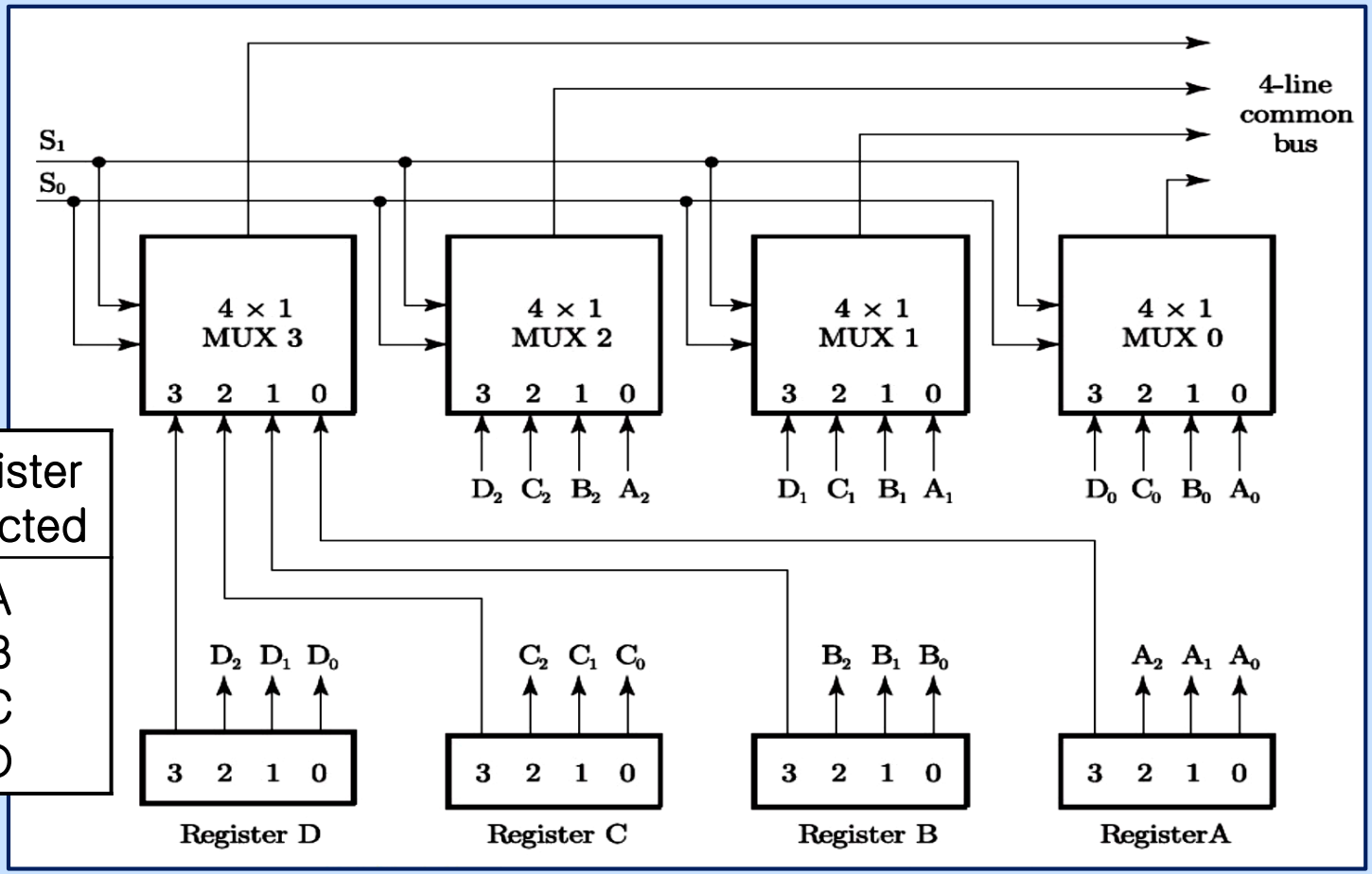
◈ 연습문제 4-1 : 다음 레지스터 전송문을 구현하는 하드웨어에 대한 블록도를 그려라.

yT_2 : R2 ← R1, R1 ← R2



4.3 버스와 메모리 전송

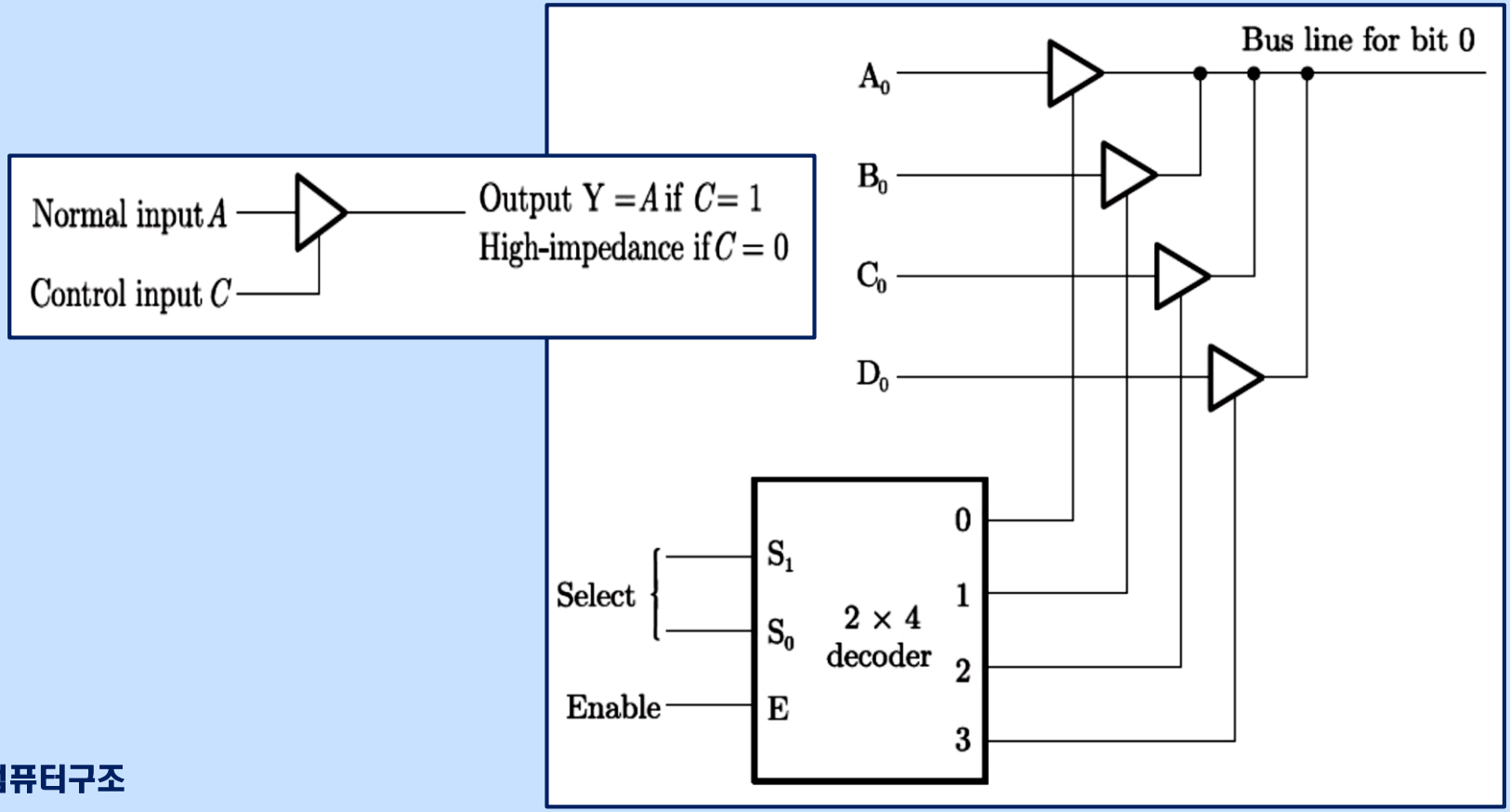
- 레지스터 각각의 전송을 위해 독립적인 전송 라인을 사용한다면 그 숫자가 많아지기 때문에 공통의 버스 시스템을 사용
- 버스에서는 제어 신호를 이용하여 하나의 레지스터 전송만이 일어나게 함



S_1	S_0	Register Selected
0	0	A
0	1	B
1	0	C
1	1	D

◈ 3-상태 버스 버퍼

- 버스 시스템은 멀티플렉서 대신 3-상태 게이트를 이용하여 구성할 수 있음
- 3-상태 게이트는 논리 0, 1, 고저항 상태가 출력
- 3-상태 버퍼를 이용한 버스 시스템



- 제어 입력 (S_1S_0)을 디코딩하여 한 순간에 하나의 버퍼만이 활성화되고 나머지는 고저항 상태에 있도록 제어
- 인에이블 입력(E)가 0이면 버스 라인은 고저항 상태가 됨

◈ 메모리 전송

- 메모리 워드로부터 외부 세계로의 정보 전송은 읽기 동작을 통해 이루어지고 메모리로 새로운 정보를 저장하는 것은 쓰기 동작에 의해 이루어짐
- 메모리 읽기 Read : DR ← M[AR] (AR (주소) 레지스터에 해당하는 메모리 위치의 내용을 읽어 데이터 레지스터에 저장)
- 메모리 쓰기 Write : M[AR] ← R1 (레지스터 R1의 데이터를 AR (주소) 레지스터에 해당하는 메모리 위치에 저장)

4.4 산술 마이크로 연산

◈ 컴퓨터에서 사용되는 마이크로 연산

- 레지스터 사이에서 이진 정보를 전송하는 레지스터 전송 마이크로 연산
- 레지스터의 수치 데이터에 대해 산술 연산을 수행하는 산술 마이크로 연산
- 레지스터의 비수치 데이터에 대해 비트 연산 수행하는 논리 마이크로 연산
- 레지스터의 데이터에 대해 시프트 연산을 수행하는 시프트 마이크로 연산

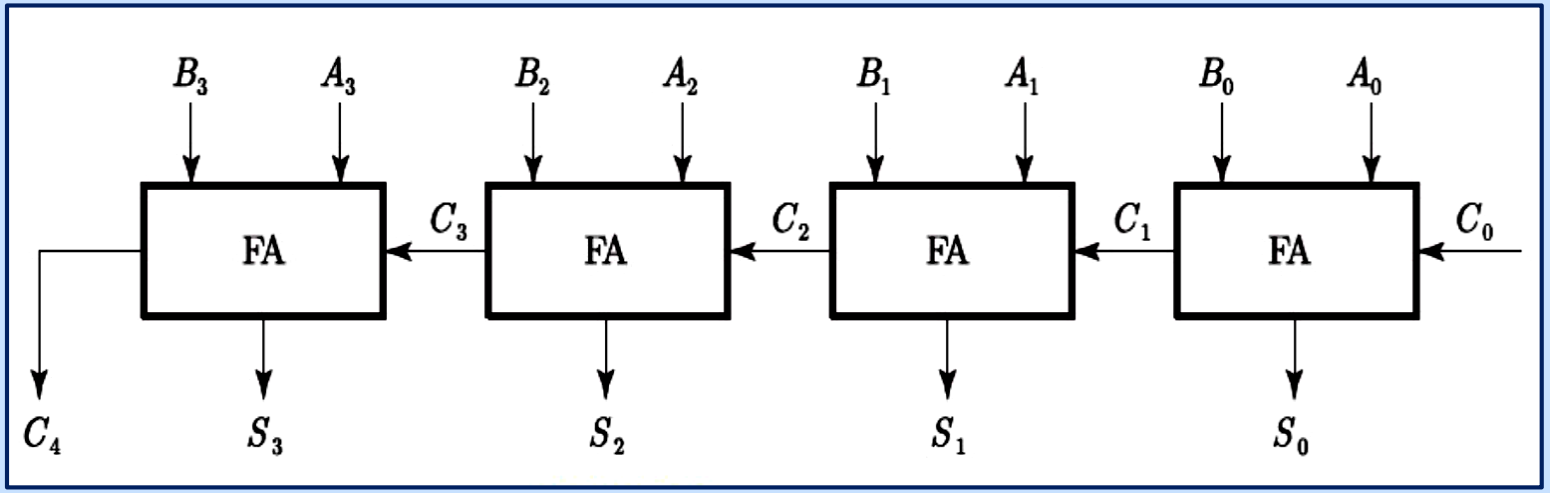
◈ 기본 산술 마이크로 연산 : 덧셈, 뺄셈, 인크리멘트, 디크리멘트 등

- 덧셈 연산 : $R3 \leftarrow R1 + R2$, 뺄셈 연산 : $R3 \leftarrow R1 + \overline{R2} + 1$ $(R1 - R2)$
- 곱셈 연산은 여러 개의 덧셈과 시프트 마이크로 연산으로 구현되고 나눗셈 연산은 여러 개의 뺄셈과 시프트 마이크로 연산으로 구현

Symbolic designation	Description
$R3 \leftarrow R1 + R2$	Contents of R1 plus R2 transferred to R3
$R3 \leftarrow R1 - R2$	Contents of R1 minus R2 transferred to R3
$R2 \leftarrow \overline{R2}$	Complement the contents of R2 (1's complement)
$R2 \leftarrow \overline{R2} + 1$	2's complement the contents of R2 (negate)
$R3 \leftarrow R1 + \overline{R2} + 1$	R1 plus the 2's complement of R2 (subtraction)
$R1 \leftarrow R1 + 1$	Increment the contents of R1 by one
$R1 \leftarrow R1 - 1$	Decrement the contents of R1 by one

◈ 이진 가산기

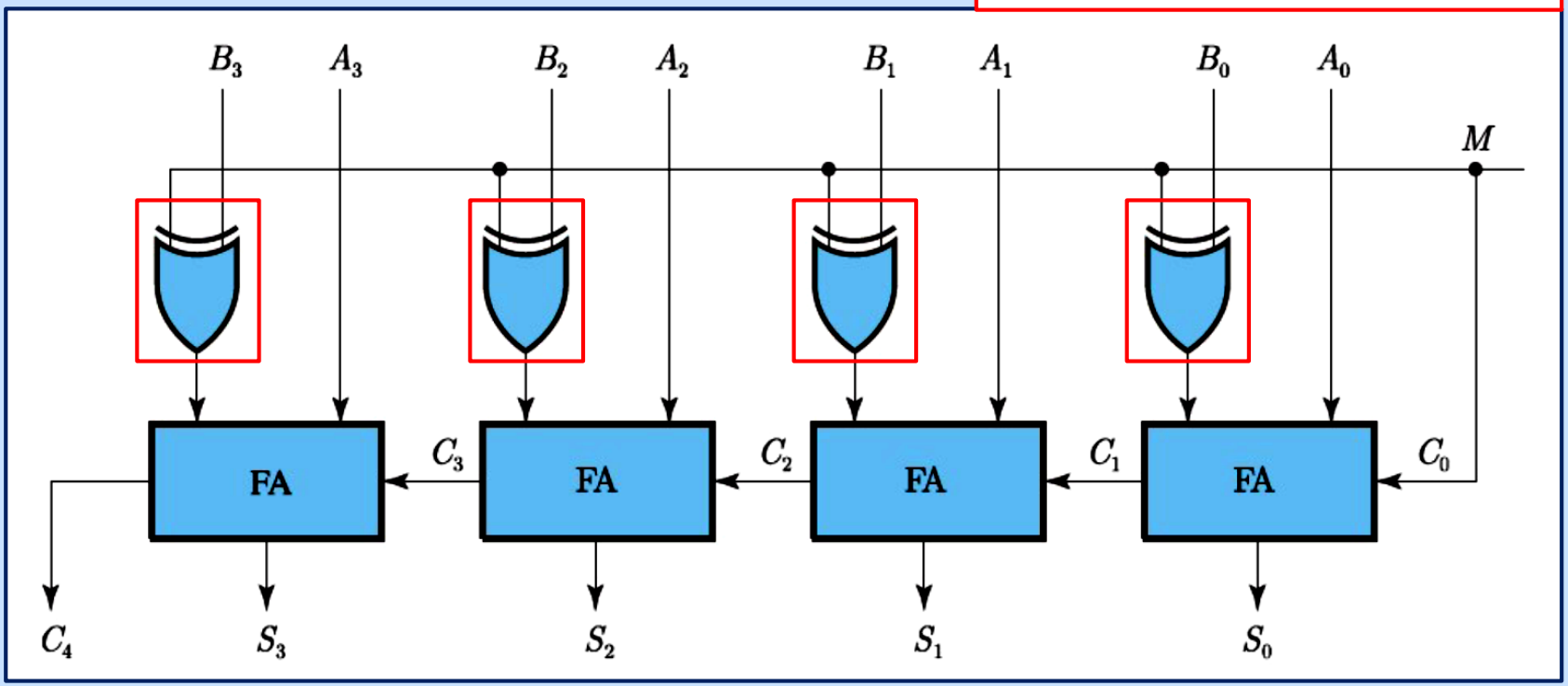
- n개의 전가산기를 직렬 연결하여 두 이진수에 대한 덧셈을 수행하는 회로
- 각 전가산기의 캐리 출력은 다음 상위 전가산기의 캐리 입력에 연결
- 이진 가산기의 입력 A, B는 소스 레지스터로부터 나오고 이진 가산기의 출력은 목적지 레지스터로 전송



◈ 이진 가감산기

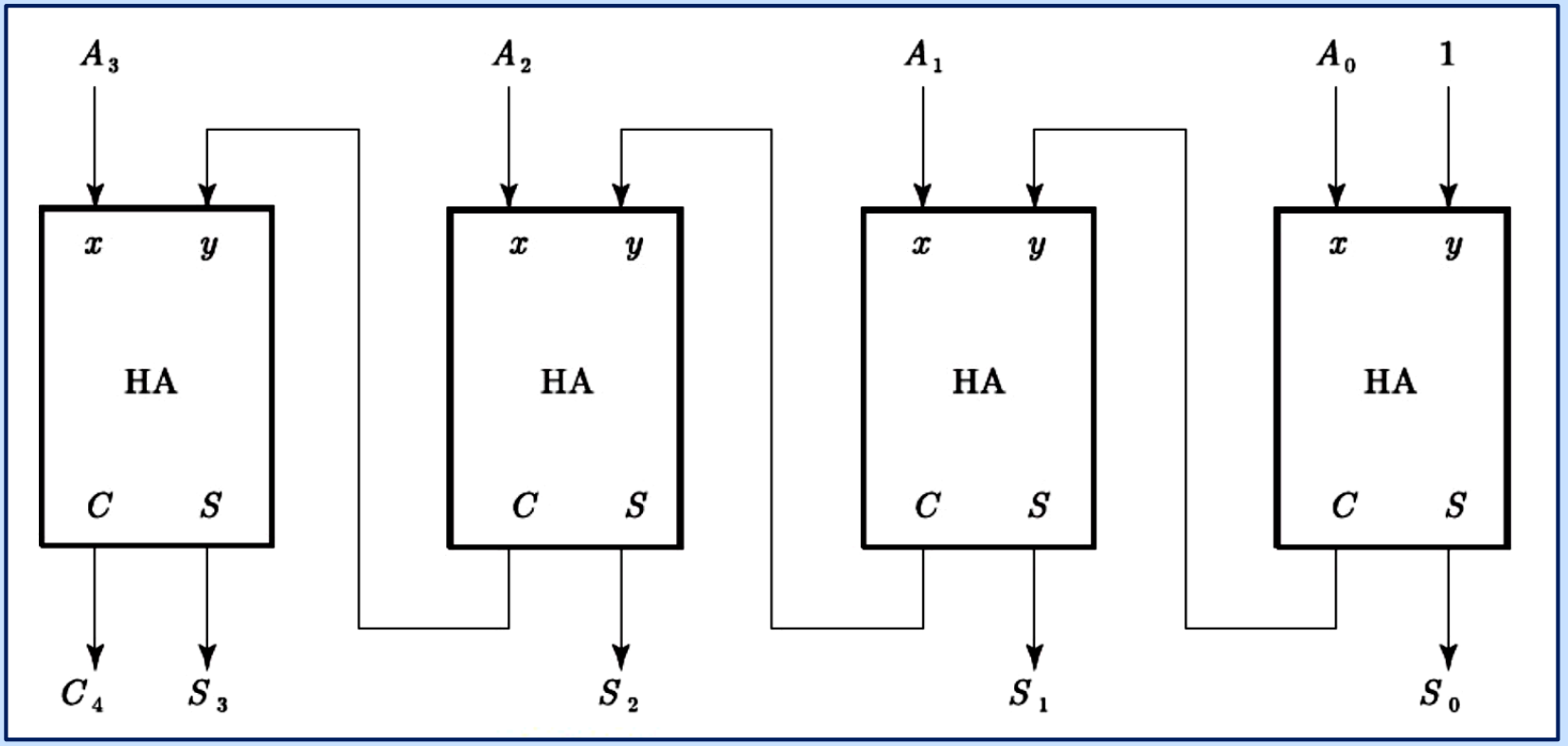
- 이진수의 뺄셈은 2의 보수를 이용하면 덧셈으로 계산될 수 있음
- 각 전가산기에 XOR 게이트를 추가하면 덧셈과 뺄셈 연산은 동일한 회로로 구현 가능

M = 0이면 B, M = 1이면 $\overline{B}$



◈ 이진 인크리멘터

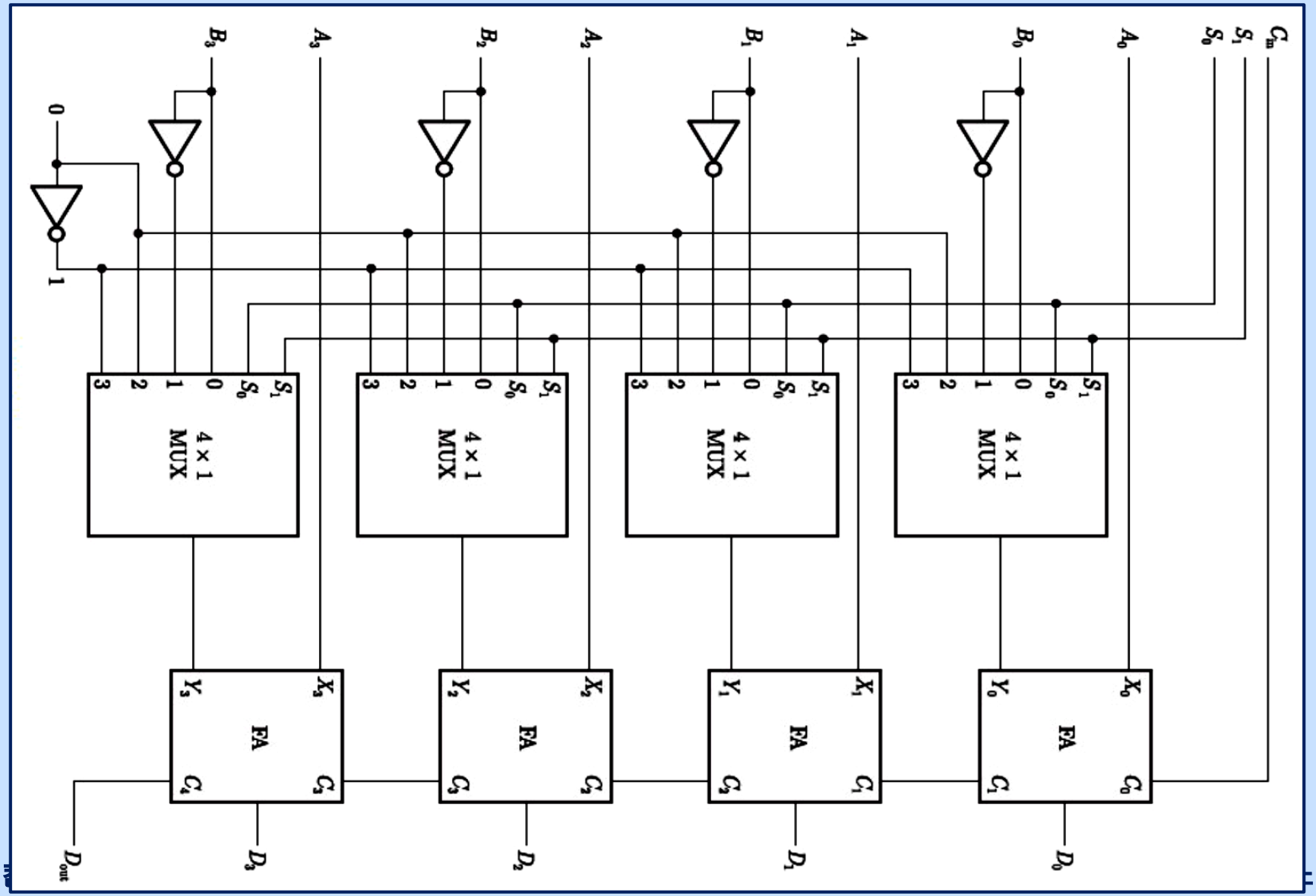
- 레지스터에 1을 더하는 연산으로 반가산기를 직렬로 연결하여 구현
- 4비트 이진 인크리멘터



◈ 산술 회로

- 산술 마이크로 연산들은 병렬 가산기의 입력을 제어함으로써 하나의 회로에 의해 모두 구현이 가능
- 산술 회로는 두 개의 입력 A, B와 출력 D로 구성되며 입력 A는 이진 가산기에 직접 연결되고 입력 B는 MUX에 연결
- 산술 마이크로 연산에 따라 MUX 출력을 (B, B의 보수, 0, 1) 중 하나 선택

– 4비트 산술 회로 (D = A + Y + C_{in})



– 산술 회로의 함수표

Select S_1	Select S_0	Select C_{in}	Input Y	Output $D = A + Y + C_{in}$	Microoperation
0	0	0	B	$D = A + B + 0 = A + B$	Add
0	0	1	B	$D = A + B + 1$	Add with carry
0	1	0	$\overline{B}$	$D = A + \overline{B} = A - B - 1$	Subtract with borrow
0	1	1	$\overline{B}$	$D = A + \overline{B} + 1 = A - B$	Subtract
1	0	0	0(0000)	$D = A + 0 + 0 = A$	Transfer A
1	0	1	0(0000)	$D = A + 0 + 1 = A + 1$	Increment A
1	1	0	−1(1111)	$D = A - 1 + 0 = A - 1$	Decrement A
1	1	1	−1(1111)	$D = A - 1 + 1 = A$	Transfer A

S_1, S_0, C_{in}을 제어하여 서로 다른 7개의 산술 마이크로 연산을 수행

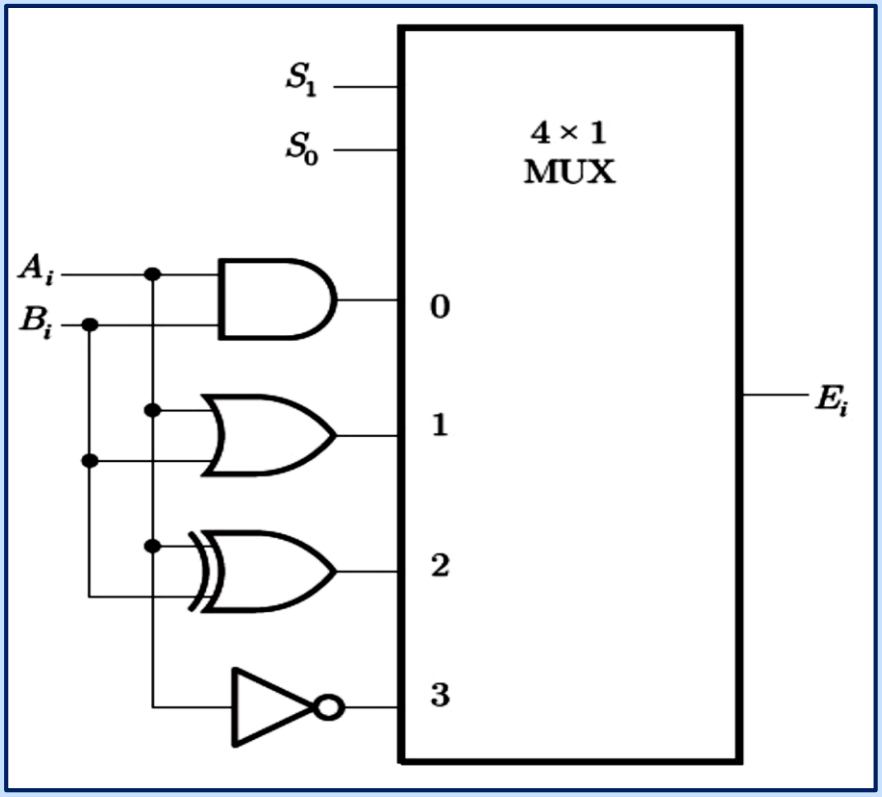
4.5 논리 마이크로 연산

Boolean function	Microoperation	Name
$F_0 = 0$	$F \leftarrow 0$	Clear
$F_1 = xy$	$F \leftarrow A \wedge B$	AND
$F_2 = xy'$	$F \leftarrow A \wedge \overline{B}$	
$F_3 = x$	$F \leftarrow A$	Transfer A
$F_4 = x'y$	$F \leftarrow \overline{A} \wedge B$	
$F_5 = y$	$F \leftarrow B$	Transfer B
$F_6 = x \oplus y$	$F \leftarrow A \oplus B$	Exclusive-OR
$F_7 = x + y$	$F \leftarrow A \vee B$	OR
$F_8 = (x + y)'$	$F \leftarrow \overline{A \vee B}$	NOR
$F_9 = (x \oplus y)'$	$F \leftarrow \overline{A \oplus B}$	Exclusive-NOR
$F_{10} = y'$	$F \leftarrow \overline{B}$	Complement B
$F_{11} = x + y'$	$F \leftarrow A \vee \overline{B}$	
$F_{12} = x'$	$F \leftarrow \overline{A}$	Complement A
$F_{13} = x' + y$	$F \leftarrow \overline{A} \vee B$	
$F_{14} = (xy)'$	$F \leftarrow \overline{A \wedge B}$	NAND
$F_{15} = 1$	$F \leftarrow$ all 1's	Set to all 1's

◈ OR 연산은 ∨, AND 연산은 ∧, 1의 보수 연산은 bar로 표시

- 제어 함수에서의 +는 OR 연산자를 나타냄
- 예) P+Q : R1 ← R2 + R3, R4 ← R5 ∨ R6

  (P 또는 Q가 1이면 덧셈 연산과 OR 연산을 수행)
- 대부분의 컴퓨터에서는 4개의 (AND, OR, XOR, NOT) 연산만을 구현



S_1	S_0	Output	Operation
0	0	$E = A \wedge B$	AND
0	1	$E = A \vee B$	OR
1	0	$E = A \oplus B$	XOR
1	1	$E = \overline{A}$	Complement

◈ 응용

- selective-set 연산 (OR 연산) : 비트의 일부를 1로 만드는 연산
- selective-보수 연산 (XOR 연산) : 비트의 일부를 보수화하는데 사용
- selective-clear 연산 : B의 1에 해당하는 A의 비트만 0으로 만드는 연산으로 $A \leftarrow A \wedge \overline{B}$ 연산과 같음
- mask 연산 (AND 연산) : B의 0에 해당하는 A의 비트만 0으로 만드는 연산
- insert 연산 : 비트 묶음 속에 새로운 값을 삽입하는 연산으로 우선 원하는 위치 비트를 마스크시킨 후 원하는 값을 OR시킴
- 클리어 연산 : A와 B를 비교하여 두 개의 워드가 일치하면 A를 0으로 만드는 연산으로 XOR 연산과 같음

4.6 시프트 마이크로 연산

◈ 논리 시프트 : 직렬 입력으로 0이 전송되는 것으로 shl, shr로 표시

- 왼쪽 논리 시프트 : R1 ← shl R1

R_{15}	R_{14}	R_{13}	R_{12}	R_{11}	R_{10}	R_9	R_8	R_7	R_6	R_5	R_4	R_3	R_2	R_1	R_0

R_{14}	R_{13}	R_{12}	R_{11}	R_{10}	R_9	R_8	R_7	R_6	R_5	R_4	R_3	R_2	R_1	R_0	0

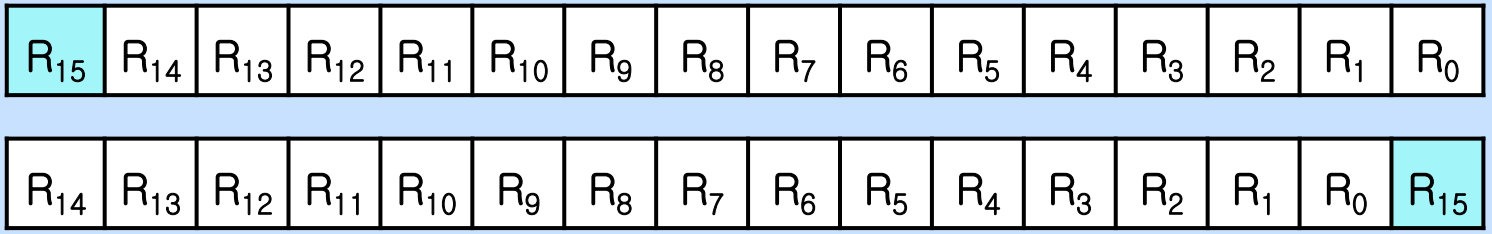
- 오른쪽 논리 시프트 : R2 ← shr R2

R_{15}	R_{14}	R_{13}	R_{12}	R_{11}	R_{10}	R_9	R_8	R_7	R_6	R_5	R_4	R_3	R_2	R_1	R_0

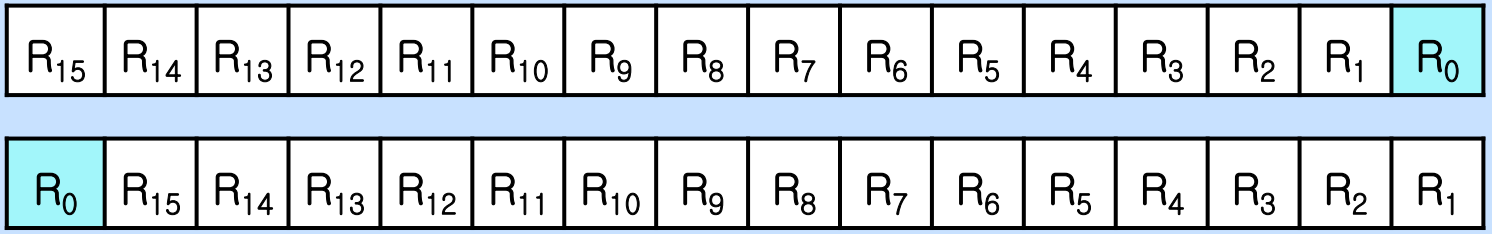
0	R_{15}	R_{14}	R_{13}	R_{12}	R_{11}	R_{10}	R_9	R_8	R_7	R_6	R_5	R_4	R_3	R_2	R_1

◈ 순환 시프트 : 시프트 레지스터의 직렬 출력을 직렬 입력에 연결하여 저장되어 있던 정보의 손실 없이 비트들을 순환시키는 연산

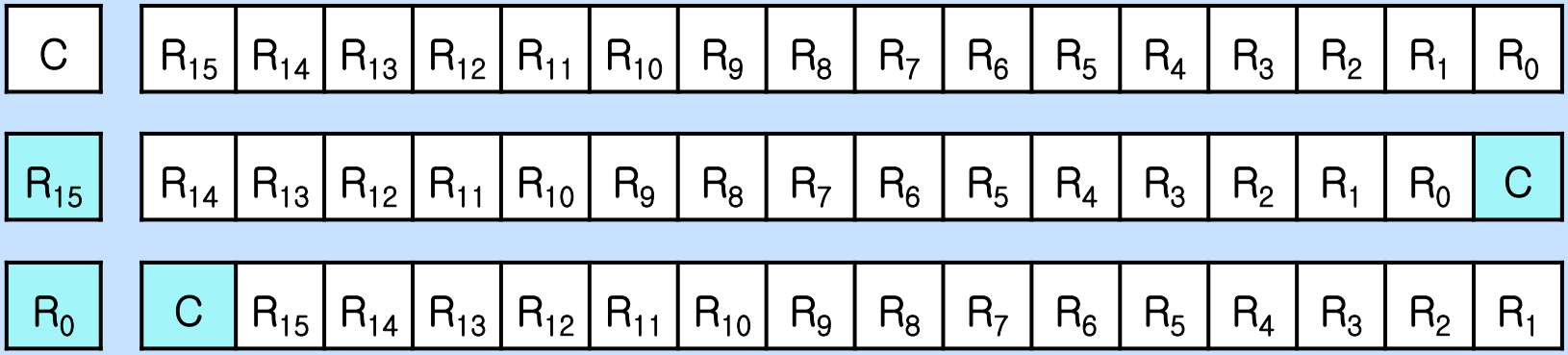
– 왼쪽 순환 시프트 : R1 ← cil R1



– 오른쪽 순환 시프트 : R2 ← cir R2



– 캐리 비트까지 포함한 왼쪽 순환 시프트, 오른쪽 순환 시프트



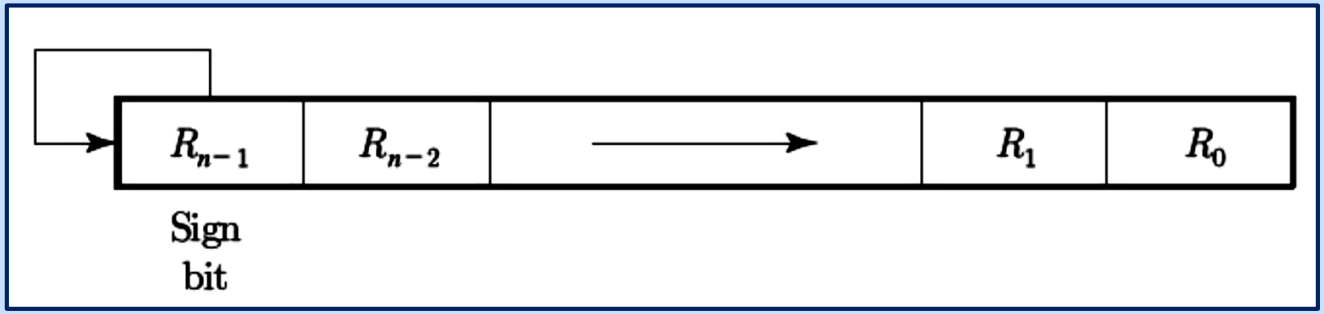
◈ 산술 시프트 : 부호가 있는 이진수를 시프트하는 것

– 왼쪽 산술 시프트 : R1 ← ashl R1 (부호 유지 불가능)

R_{15}	R_{14}	R_{13}	R_{12}	R_{11}	R_{10}	R_9	R_8	R_7	R_6	R_5	R_4	R_3	R_2	R_1	R_0

R_{14}	R_{13}	R_{12}	R_{11}	R_{10}	R_9	R_8	R_7	R_6	R_5	R_4	R_3	R_2	R_1	R_0	0

– 오른쪽 산술 시프트 : R2 ← shr R2 (부호 유지 가능)



R_{15}	R_{14}	R_{13}	R_{12}	R_{11}	R_{10}	R_9	R_8	R_7	R_6	R_5	R_4	R_3	R_2	R_1	R_0

R_{15}	R_{15}	R_{14}	R_{13}	R_{12}	R_{11}	R_{10}	R_9	R_8	R_7	R_6	R_5	R_4	R_3	R_2	R_1

- 왼쪽 산술 시프트는 2를 곱한 것과 같고 오른쪽 산술 시프트는 2로 나눈 것과 같음
- 왼쪽 산술 시프트에 의한 오버플로우 탐지 : $V_s = R_{n-1} \oplus R_{n-2}$

Symbolic designation	Description
R ← shl R	Shift-left register R
R ← shr R	Shift-right register R
R ← cil R	Circular shift-left register R
R ← cir R	Circular shift-right register R
R ← ashl R	Arithmetic shift-left register R
R ← ashr R	Arithmetic shift-right register R

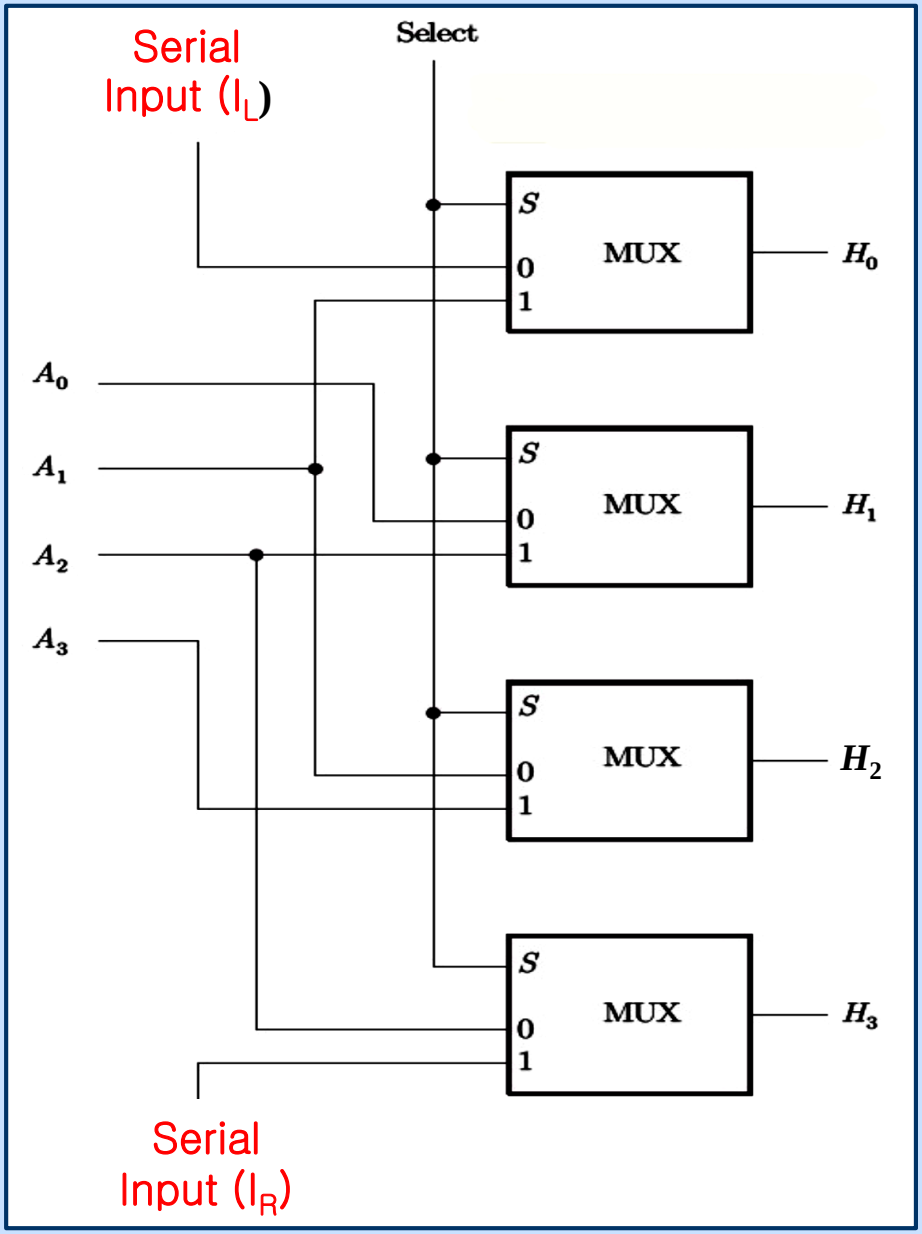
- 4비트 조합 회로 시프터

S=00이면 shift left

(H_3 H_2 H_1 H_0 = A_2 A_1 A_0 I_L)

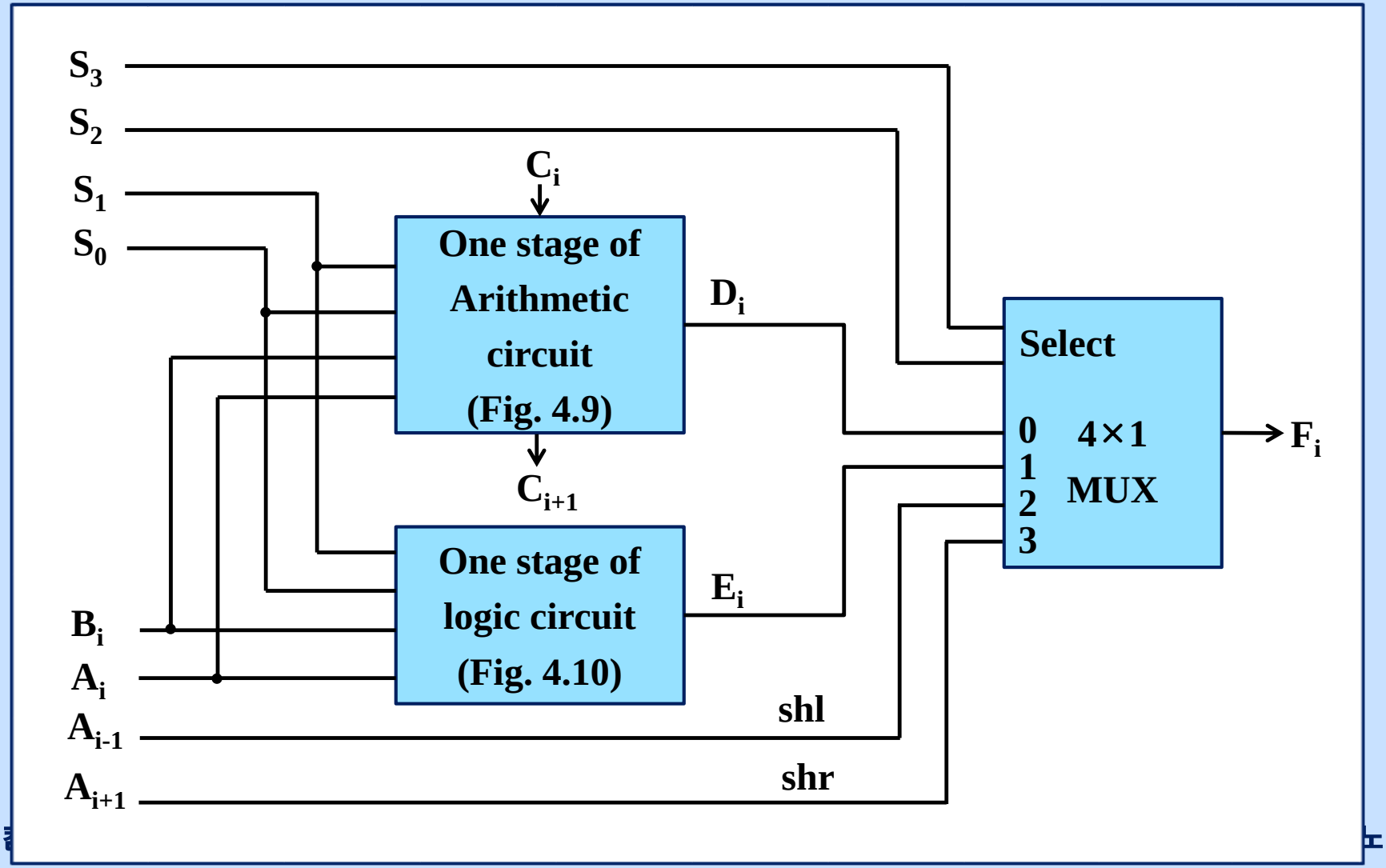
S=10이면 shift right

(H_3 H_2 H_1 H_0 = I_R A_3 A_2 A_1)



4.7 산술 논리 시프트 장치

◈ 8개 산술 연산, 4개 논리 연산, 2개 시프트 연산을 위한 산술 논리 장치



◈ 산술 논리 시프트 장치에 대한 함수표

– S_3, S_2는 산술, 논리, 시프트 연산 구분, S_1, S_0은 특정 마이크로 연산 선택

S_3	S_2	S_1	S_0	C_{in}	Operation	Function
0	0	0	0	0	F = A	Transfer A
0	0	0	0	1	F = A + 1	Increment A
0	0	0	1	0	F = A + B	Add
0	0	0	1	1	F = A + B + 1	Add with carry
0	0	1	0	0	F = A + $\overline{B}$	Subtract with borrow
0	0	1	0	1	F = A + $\overline{B}$ + 1	Subtract
0	0	1	1	0	F = A − 1	Decrement A
0	0	1	1	1	F = A	Transfer A
0	1	0	0	×	F = A ∧ B	AND
0	1	0	1	×	F = A ∨ B	OR
0	1	1	0	×	F = A ⊕ B	XOR
0	1	1	1	×	F = $\overline{A}$	Complement A
1	0	×	×	×	F = shl A	Shift left A into F
1	1	×	×	×	F = shr A	Shift right A into F