레지스터의 길이는
워드와 연관O

# 4-1. Register Transfer Language



## 4-1 Register Transfer Language

- Microoperation : 레지스터에 저장되어있는 데이터 간 연산을 하는 것

  The operations executed on data stored in registers*(shift, clear, load, count)*

- Internal H/W Organization*(best defined by specifying)*

  1. The set of registers(register의 개수, 종류, 기능)
  2. The sequence of microoperations
  3. The sequence control of microoperations

칩
CPU마다 다르다

## 4-2 Register Transfer

- Registers : ***Fig. 4-1*** 종류

  주소저장

  Designated by Capital Letter(*sometimes followed by numerals*) : MAR(Memory Address Register), PC(Program Counter), IR(Instruction Register), R1(Processor Register)

프로그램의 명령어에서
현재 수행의 다음에
수행될 명령어를 저장

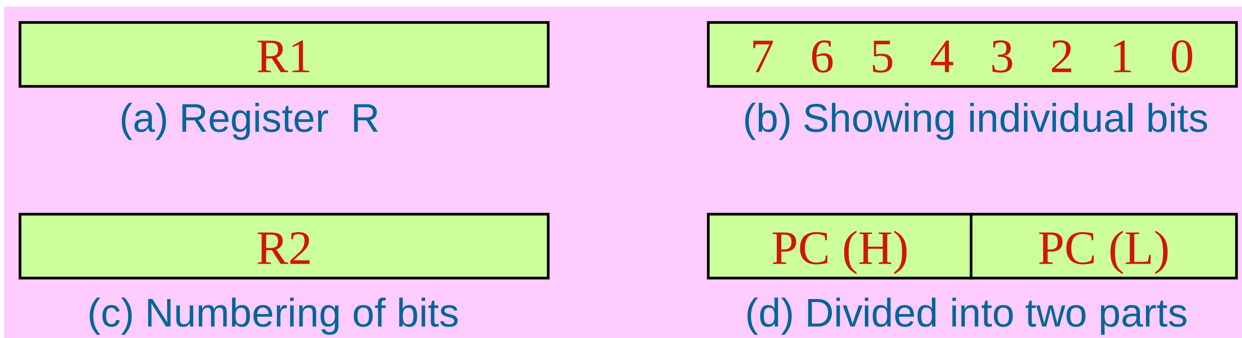
다음에 수행될
명령어의 주소를
MAR로 부터 받아
명령어 수행

# 4-2. Register Transfer

The individual F/Fs in an n-bit register : numbered in sequence from 0(*rightmost position*) through n-1

The numbering of bits in a 16-bit register : marked on top of the box

A 16-bit register partitioned into two parts : bit 0-7(*symbol "L" Low byte*), bit 8-15(*symbol "H" High byte*)




레지스터 나누기가능

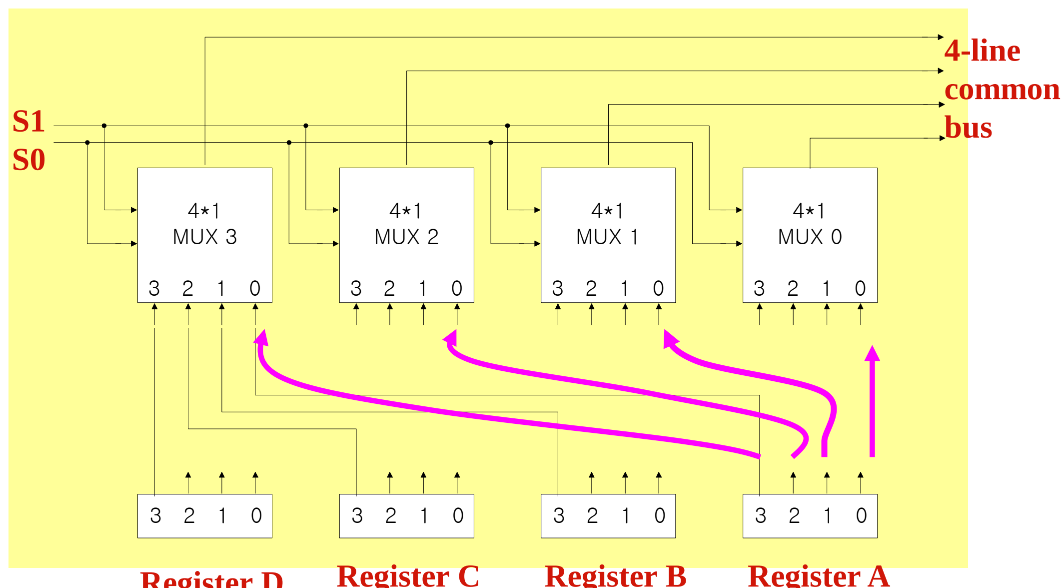
Figure 4-1. Block diagram of register

여러 형태로 레지스터 표시 가능

# 4-3 Bus and Memory Transfers

- **Common Bus** : 데이터가 이동하는 통로
  - A more efficient scheme for transferring information between registers in ***a multiple-register configuration***
  - A bus structure = a set of common lines 1비트당 버스 1개 필요
  - ***Control signals*** determine which register is selected
    - » One way of constructing a common bus system is with ***multiplexers***
    - » The *multiplexers* select the source register whose binary information is place on the bus
  - The construction of a bus system for four registers : Fig. 4-3
    - » 4 bit register X 4
    - » Four 4 X 1 Multiplexers
    - » Bus Selection : S0, S1

| S1 | S0 | Register selected |
|---|---|---|
| 0 | 0 | A |
| 0 | 1 | B |
| 1 | 0 | C |
| 1 | 1 | D |




  - 8 Registers with 16 bit
    - » 16 X 1 mux 8 개 필요

산술연산

# 4-4. Arithmetic Microoperation

◆ 4-bit Binary Adder : ***Fig. 4-6***

전가산기 Full adder = 2-bits sum + previous carry

Binary adder = the arithmetic sum of two binary numbers of any length

$c_0$(input carry), $c_4$(output carry)

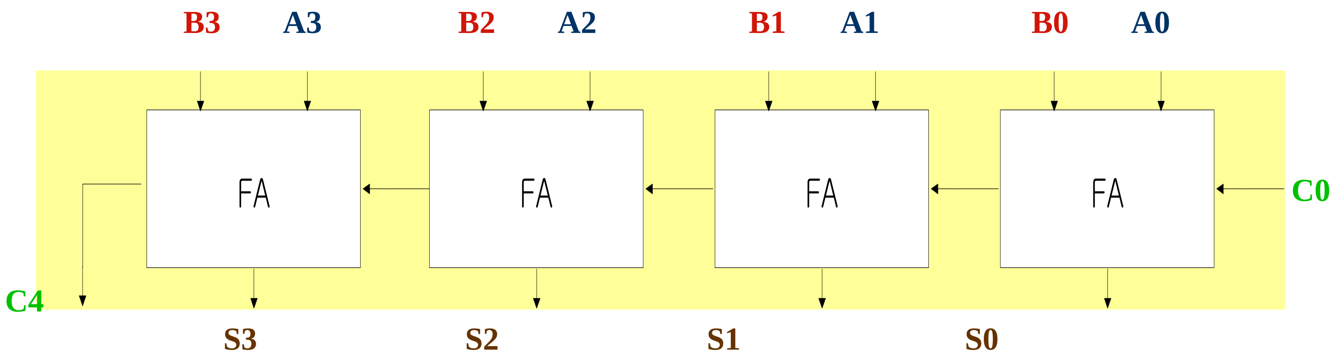



Figure 4-6. 4-bit binary adder

FA가 4개

# 4-4. Arithmetic Microoperation

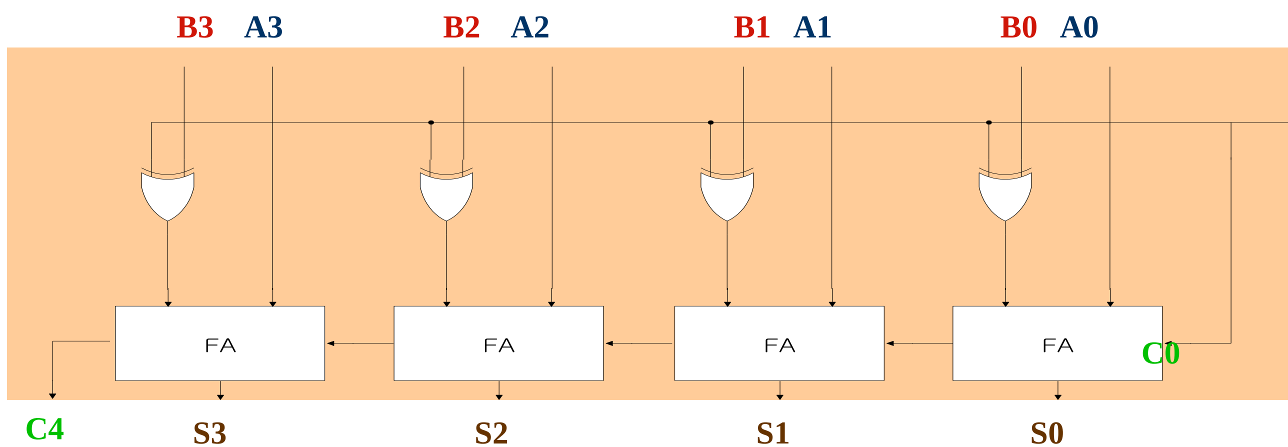
- **4-bit Binary Adder-Subtractor :** ***Fig. 4-7*** 덧셈기로 뺄셈기 구현 가능

  회로 회로

  One common circuit by including an exclusive-OR gate with each full-adder

  M =0 : Adder $B \oplus M + C = B \oplus 0 + 0 = B$, ∴ ***A + B***

  M =1 : Subtractor $B \oplus M + C = B \oplus 1 + 1 = B' + 1 = -B$(2's comp), ∴***A - B***




M 셀렉션 비트

# 4-4. Arithmetic Microoperation

## Arithmetic Circuit

One composite arithmetic circuit in ***Tab. 4-4*** : ***Fig. 4-9***

$D = A_0(X_0) + B_0(Y_0) + C_{in}$

- » $B_0$ : $S_0$, $S_1$에 따라 ***B, $\overline{B}$, 0, 1***
- » Tab. 4-4의 Input Y = B

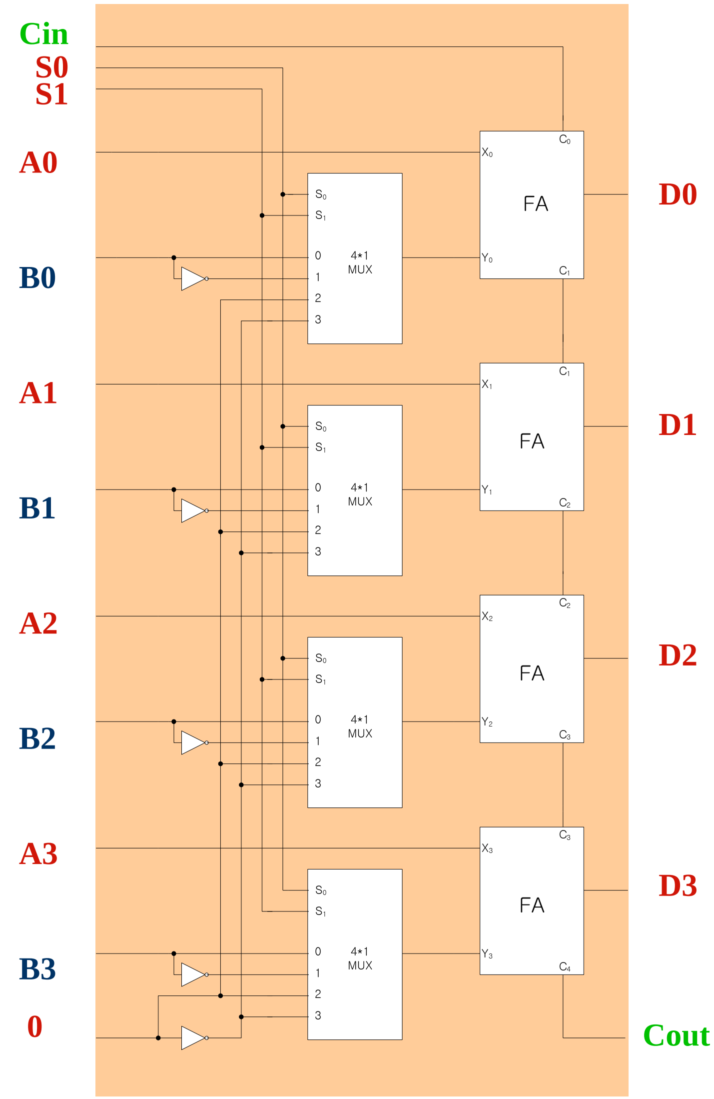
설계 예시

| Select | | | Input | Output | Microoperation |
|---|---|---|---|---|---|
| S1 | S0 | $C_{in}$ | Y | $D=A+Y+C_{in}$ | |
| 0 | 0 | 0 | B | D=A+B | Add |
| 0 | 0 | 1 | B | D=A+B+1 | Add with carry |
| 0 | 1 | 0 | B' | D=A+B' | Subtract with borrow |
| 0 | 1 | 1 | B' | D=A+B'+1 | Subtract |
| 1 | 0 | 0 | 0 | D=A | Transfer A |
| 1 | 0 | 1 | 0 | D=A+1 | Increment A |
| 1 | 1 | 0 | 1 | D=A−1 | Decrement A |
| 1 | 1 | 1 | 1 | D=A | Transfer A |

A+1111=A-1

A-1+1=A

A+B'=A+B'+1-1 = A-B-1



레지스터에 저장되어있는 데이터를 대상으로 논리 연산

# 4-5 Logic Microoperation

## ◆ Logic microoperation

Logic microoperations consider ***each bit of the register separately*** and treat them as binary variables

» ***exam)***

$P: R1 \leftarrow R1 \oplus R2$

| | |
|---|---|
| 1010 | Content of R1 |
| + 1100 | Content of R2 |
| 0110 | Content of R1 after P=1 |

Special Symbols

» Special symbols will be adopted for the logic microoperations ***OR(∨), AND(∧),*** and ***complement(a bar on top)***, to distinguish them from the corresponding symbols used to express Boolean functions

Arithmetic에서 1's Complement와 동일

» ***exam)***

$P+Q: R1 \leftarrow R2+R3,\ \ R4 \leftarrow R5 \vee R6$

Logic OR

Arithmetic ADD

## ◆ List of Logic Microoperation

Truth Table for 16 functions for 2 variables : ***Tab. 4-5 (뒷면에...)***

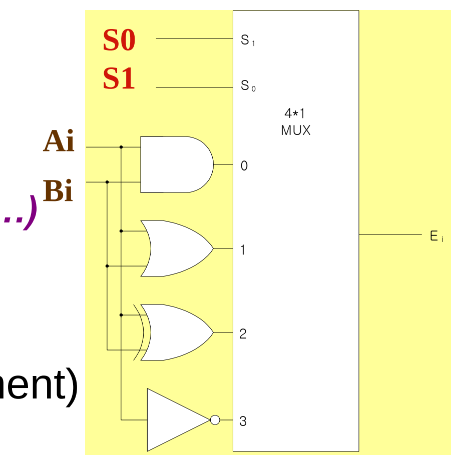
16 Logic Microoperation : ***Tab. 4-6***

## ◆ Hardware Implementation

16 microoperation → Use only 4(AND, OR, XOR, Complement)

∵ All other Operation can be derived

One stage of logic circuit : ***Fig. 4-10***



# 4-5. Logic Microoperation

4비트의 디코더로 16개 중 1개를 택할 수 있는 예시

| X | Y | $F_0$ | $F_1$ | $F_2$ | $F_3$ | $F_4$ | $F_5$ | $F_6$ | $F_7$ | $F_8$ | $F_9$ | $F_{10}$ | $F_{11}$ | $F_{12}$ | $F_{13}$ | $F_{14}$ | $F_{15}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

**TABLE 4-5. Truth Table for 16 Functions of Two Variables**

식을 외우지 말고, 식을 보고 구분할 줄 알기

| 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 |
| $F_6 = x \oplus y$ | $F \leftarrow A \oplus B$ | Ex-OR |
| $F_7 = x+y$ | $F \leftarrow A \vee B$ | OR |

| Boolean function | Microoperation | Name |
|---|---|---|
| $F_8 = (x+y)'$ | $F \leftarrow \overline{A \vee B}$ | NOR |
| $F_9 = (x \oplus y)'$ | $F \leftarrow \overline{A \oplus B}$ | Ex-NOR |
| $F_{10} = y'$ | $F \leftarrow \overline{B}$ | Compl-B |
| $F_{11} = x+y'$ | $F \leftarrow A \vee \overline{B}$ | |
| $F_{12} = x'$ | $F \leftarrow \overline{A}$ | Compl-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 |

**TABLE 4-6. Sixteen Logic Microoperations**

&& : 일반적 AND 연산자
& : 비트연산자(0,1)

# 4-5. Logic Microoperation



## Some Applications

Logic microoperations are very useful for ***manipulating individual bits*** or ***a portion of a word*** stored in a register

Used to change bit values, delete a group of bits, or insert new bit values

Selective-set $A \leftarrow A \vee B$

» The selective-set operation sets to 1 the bits in register A where there are corresponding 1's in register B. It does not effect bit positions that have 0's in B

Selective-complement $A \leftarrow A \oplus B$

» The selective-complement operation complements bits in A where there are corresponding 1's in B. It does not effect bit positions that have 0's in B

1 → 0

Selective-clear $A \leftarrow A \wedge \overline{B}$

» The selective-clear operation clears to 0 the bits in A only where there are corresponding 1's in B

0 → 0

Selective-mask $A \leftarrow A \wedge B$

» The mask operation is similar to the selective-clear operation except that the bits of A are cleared only where there are corresponding 0's in B

여러 비트 중에서, 선택된 비트에 대해서만 비트 논리 연산을 수행하라

| | | | |
|---|---|---|---|
| 1010 A before | 1010 A before | 1010 A before | 1010 A before |
| 1100 B(Logic Operand) | 1100 B(Logic Operand) | 1100 B(Logic Operand) | 1100 B(Logic Operand) |
| 1110 A After | 0110 A After | 0010 A After | 1000 A After |
| *Selective-set* | *Selective-complement* | *Selective-clear* | *Selective-mask* |

Insert

» The insert operation inserts a new value into a group of bits

» This is done by first masking the bits and then ORing them with the required value

| 1) Mask | | |
|---|---|---|
| **0110** | 1010 | A before |
| **0000** | 1111 | B mask |
| **0000** | 1010 | A after mask |

| 2) OR | | |
|---|---|---|
| **0000** | 1010 | A before |
| **1001** | 0000 | B insert |
| **1001** | 1010 | A after insert |

| Clear | |
|---|---|
| **1010** | A |
| **1010** | B |
| **0000** | A after clear |

Clear $A \leftarrow A \oplus B$

» The clear operation compares the words in A and B and produces an all 0's result if the two numbers are equal

옮기다

## 4-6 Shift Microoperations

### ◆ Shift Microoperations : ***Tab. 4-7***

Shift microoperations are used for serial transfer of data

Three types of shift microoperation : ***Logical, Circular,*** and ***Arithmetic***

논리

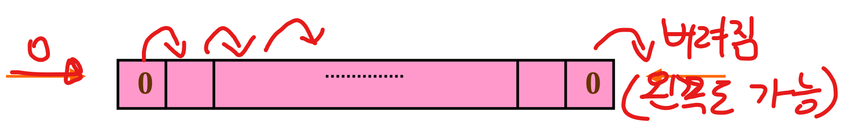
### ◆ Logical Shift

A ***logical shift*** transfers 0 through the serial input

The bit transferred to the end position through the serial input is assumed to be 0 during a logical shift(***Zero inserted***)

$R1 \leftarrow shl\ \ R1$

$R2 \leftarrow shr\ R2$



# 4-6. Shift Microoperation 3종류 있다



logical

| Symbolic designation | Description |
|---|---|
| 1 R ← shl R 왼 | Shift-left register R |
| R ← shr R 오 | Shift-right register R |
| 2 R ← cil R 왼 | Circular shift-left register R |
| R ← cir R 오 | Circular shift-right register R |
| 3 R ← ashl R 왼 | Arithmetic shift-left R |
| R ← ashr R 오 | Arithmetic shift-right R |

TABLE 4-7. Shift Microoperations

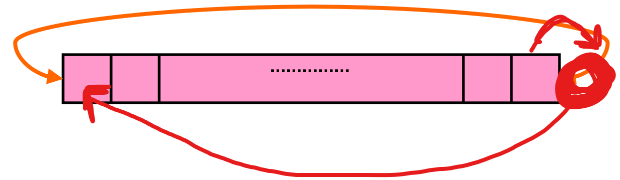
# 4-6. Shift Microoperation

- **Circular Shift(Rotate)**

  The ***circular shift*** circulates the bits of the register around the two ends without loss of information

  $R1 \leftarrow cil\ R1$

  $R2 \leftarrow cir\ R2$



나가는 비트가 다시 처음으로 삽입

- **Arithmetic Shift**

  An ***arithmetic shift*** shifts a ***signed*** binary number to the left or right

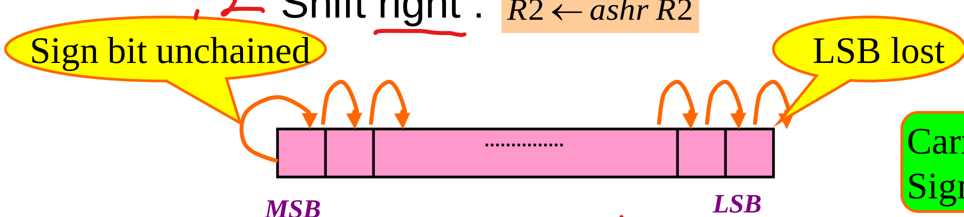
  An arithmetic ***shift-left multiplies*** a signed binary number by 2

  An arithmetic ***shift-right divides*** the number by 2

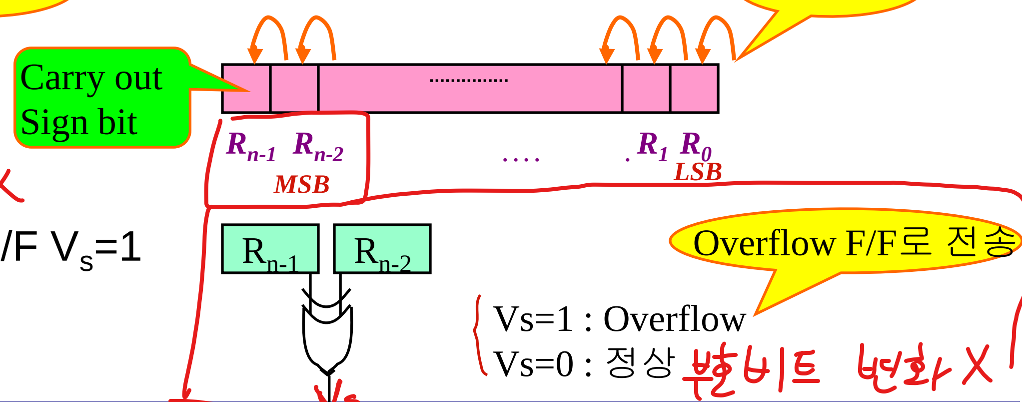
  Arithmetic shifts must leave the sign bit unchanged because the sign of the number remains the same

÷2 Shift right : $R2 \leftarrow ashr\ R2$

×2 Shift left : $R2 \leftarrow ashl\ R2$




부호비트를 고려해 부호바뀔 위험X




부호비트 변화X

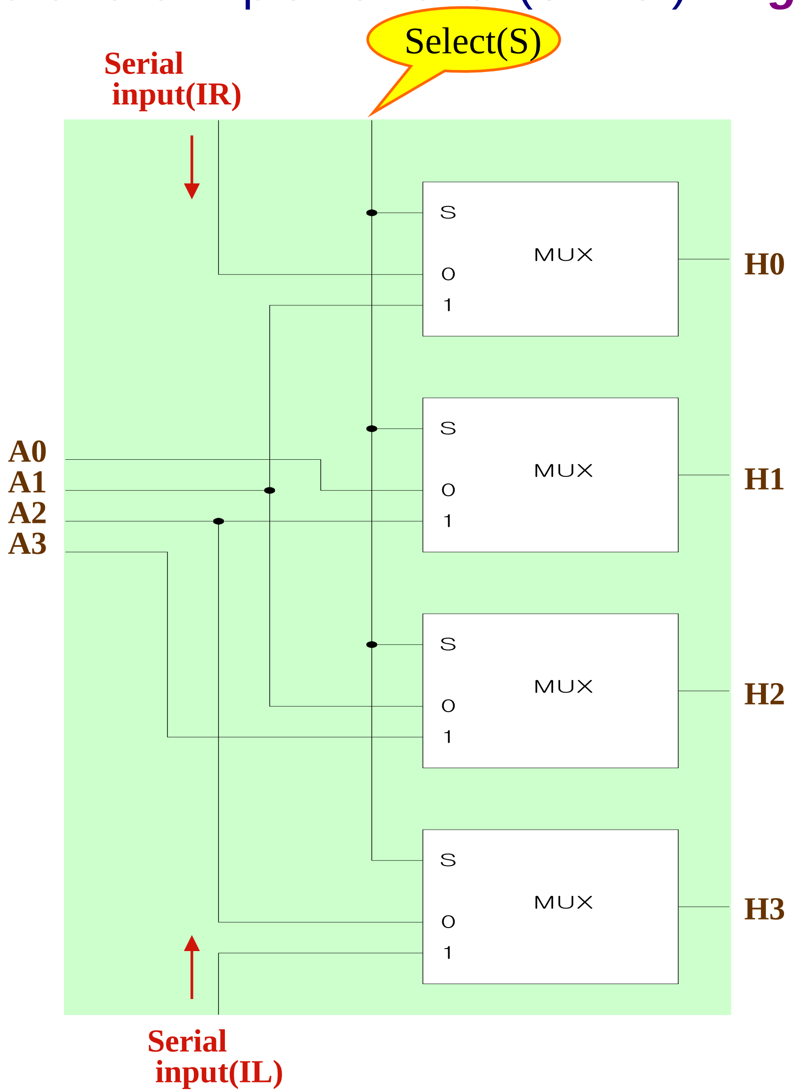
Sign reversal occur : Overflow F/F $V_s$=1

$$V_s = R_{n-1} \oplus R_{n-2}$$

# 4-7. Arithmetic Logic Shift Unit

1 2 3 한꺼번에

- Hardware Implementation(Shifter) : *Fig. 4-12*




Function Table

| Select | output | | | |
|---|---|---|---|---|
| S | H0 | H1 | H2 | H3 |
| 0 | IR | A0 | A1 | A2 |
| 1 | A1 | A2 | A3 | IL |

# 4 -7. Arithmetic Logic Shift Unit

## 4-7 Arithmetic Logic Shift Unit

- One stage of arithmetic logic shift unit : ***Fig. 4-13***

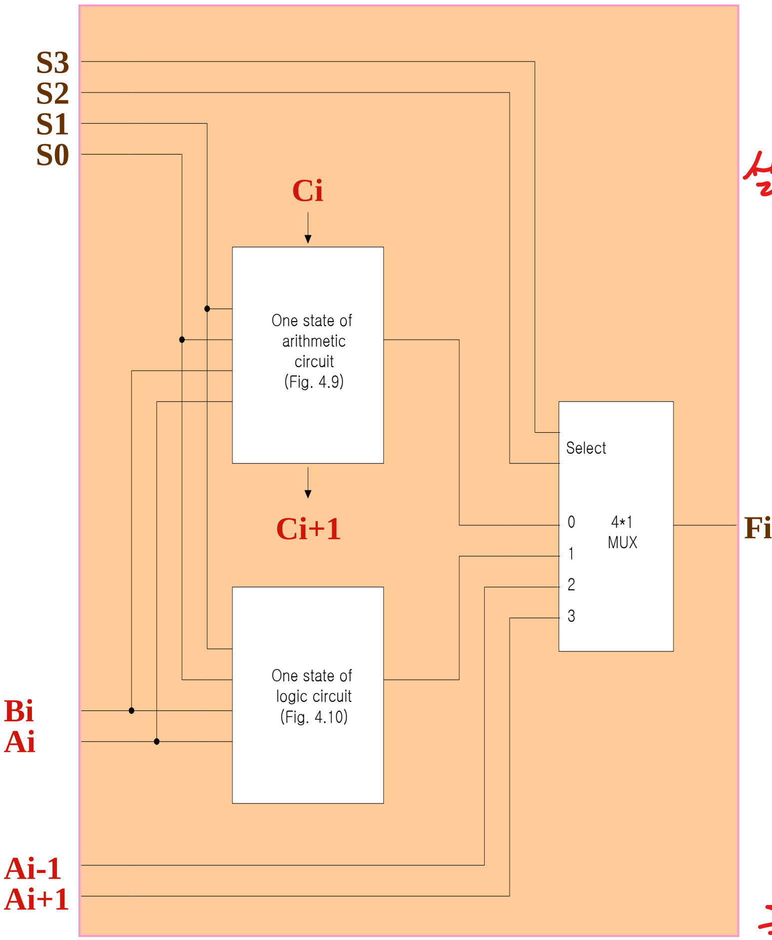



Table 4-8. Function Table for Arithmetic Logic Shift Unit

선택신호 캐리

| Operation select $S_3$ | $S_2$ | $S_1$ | $S_0$ | $C_{in}$ | 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 | Addition |
| 0 | 0 | 0 | 1 | 1 | F = A+B+1 | Add with carry |
| 0 | 0 | 1 | 0 | 0 | F = A+B | Subtract with borrow |
| 0 | 0 | 1 | 0 | 1 | F = A+B+1 | Subtraction |
| 0 | 0 | 1 | 1 | 0 | F = A − 1 | Decrement A |
| 0 | 0 | 1 | 1 | 1 | F = A | Transfer A |
| 0 | 1 | 0 | 0 | × | $F = A \wedge B$ | AND |
| 0 | 1 | 0 | 1 | × | $F = A \vee B$ | OR |
| 0 | 1 | 1 | 0 | × | $F = A \oplus B$ | XOR |
| 0 | 1 | 1 | 1 | × | F = A | Complement A |
| 1 | 0 | × | × | × | F = shr A | Shift right A into F |
| 1 | 1 | × | × | × | F = shl A | Shift left A into F |

즉, 여러 종류의 Microoperation 이 혼재가능