컴퓨터 산술과 논리 연산 (1)
(제 5주 차)

서울사이버대학교

오 창 환

학습 목표

- ALU의 구성 요소, 정수의 표현 등을 설명할 수 있다.
- 논리 연산을 설명할 수 있다.
- 쉬프트 연산을 설명할 수 있다.

학습 내용

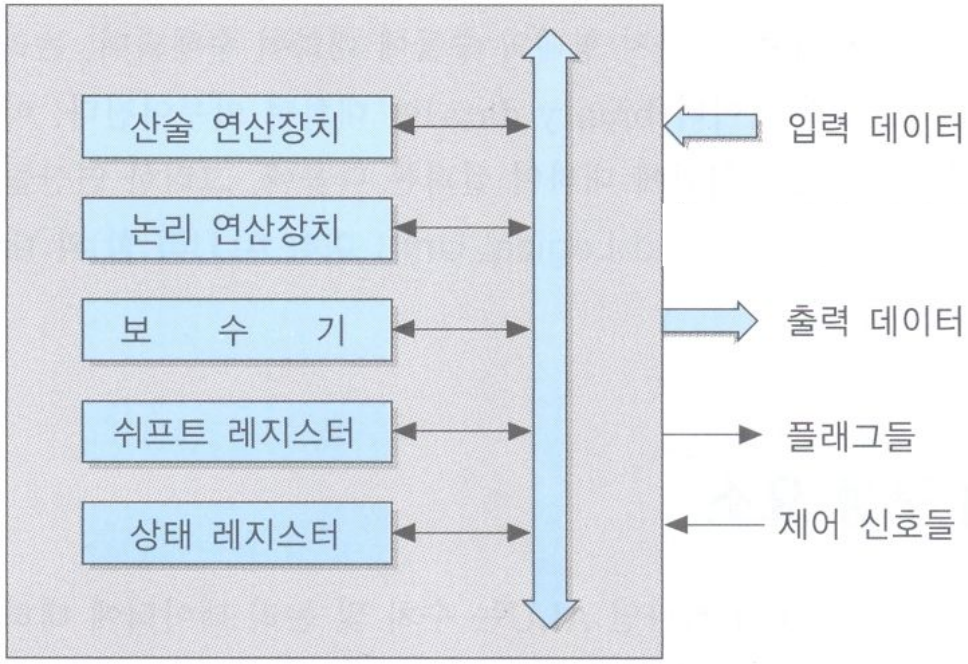
- ALU의 구성 요소, 정수의 표현
- 논리 연산
- 쉬프트 연산

ALU의 구성 요소 (1)

- ALU는 수치 및 논리 데이터에 대하여 실제적으로 연산을 수행하는 부분임.
- ALU의 내부 요소들의 주요 기능은 아래와 같음.
  * 산술 연산장치 : 산술 연산들(+, −, x, ÷)을 수행함.
  * 논리 연산장치 : 논리 연산들(AND, OR, XOR, NOT 등)을 수행함.
  * 쉬프트 레지스터(shift register) : 비트들을 좌측 혹은 우측으로 이동시키는 기능을 가진 레지스터임.
  * 보수기(complementer) : 2진 데이터에 대하여 2의 보수를 취함.
  * 상태 레지스터 : 연산 결과의 상태를 나타내는 플래그(flag)들을 저장하는 레지스터임.
- 이 요소들에 의해 처리될 데이터들은 레지스터 혹은 주기억장치로부터 ALU로 입력되고, 그 결과는 일반적으로 레지스터들 중의 하나에 저장됨.

ALU의 구성 요소 (2)

- ALU는 연산의 결과에 따라 상태 레지스터의 해당 플래그들을 세트하는데, 이 플래그들은 조건 분기 명령어 혹은 산술 명령어들에 의해 사용됨.
- 입력 데이터에 대하여 연산을 수행할 내부 요소의 선택과 ALU 내외로의 데이터 이동을 제어하는 신호들은 제어 유니트로부터 제공됨.



• ALU의 내부 구성 요소들

정수의 표현 (1)

- 2진수 체계에서는 어떤 수를 0과 1, 부호 및 소수점으로 표현함.
- 예를 들어서, 10진수 -13.625는 2진수로 -1101.101로 표현되는데, 컴퓨터가 데이터를 저장하거나 처리하는 과정에서는 부호와 소수점을 사용할 수 없으며, 0과 1만 사용할 수 있음.
- 만약 부호 없는 정수, 즉 양수만 사용한다면 표현 방법은 간단함.
  예를 들어서 8비트 단어로는 다음과 같이 0에서 255까지의 정수들을 표현할 수 있음.
  * 00000000 = 0
  * 00000001 = 1
  * 00111001 = 57
  * 10000000 = 128
  * 11111111 = 255

정수의 표현 (2)

- 만약 n-비트의 2진수 배열(a_{n-1}, a_{n-2}, …, a_1, a_0)이 부호 없는 정수 A로 해석된다면, 그 수에 대한 10진수는 아래와 같이 일반식으로 계산할 수 있음.

$$A = a_{n-1} \times 2^{n-1} + a_{n-2} \times 2^{n-2} + \cdots a_1 \times 2^1 + a_0 \times 2^0$$

- 위의 경우에는 최하위 비트인 a_0의 우측에 소수점이 있다고 가정한 경우인데, 만약 최상위 비트인 a_{n-1}의 좌측에 소수점이 있는 소수로 가정한다면, 그 수를 10진수로 변환하는 일반식은 아래와 같음.

$$A = a_{n-1} \times 2^{-1} + a_{n-2} \times 2^{-2} + \cdots a_1 \times 2^{-(n-1)} + a_0 \times 2^{-n}$$

즉, 소수점의 바로 우측 비트는 2^{-1}, 그 다음 비트는 2^{-2} , … 의 자리수(weight)를 가짐. 다음과 같은 2진수를 10진수로 계산해 보면 아래와 같음.

$2^3\ 2^2\ 2^1\ 2^0\ 2^{-1}\ 2^{-2}\ 2^{-3}$

$$1\ 1\ 0\ 1.\ 1\ 0\ 1 = 1x2^3 + 1x2^2 + 0x2^1 + 1x2^0 + 1x2^{-1} + 0x2^{-2} + 1x2^{-3}$$

$$= 8+4+1+0.5+0.125 = 13.625$$

정수의 표현 (3)

- 컴퓨터는 양수뿐만 아니라 음수도 처리하기 때문에 음수를 표현하는 방법이 필요한데
  그 방법에는 여러 가지가 있으나 모든 방법들에서 공통적인 것은
  단어의 맨 좌측 비트(leftmost bit)를 부호 비트로 사용한다는 점임.
  만약 부호 비트가 0이면 그 수는 양수이고, 1이면 음수임.
- 부호 비트를 사용하는 표현으로는 다음과 같은 세 가지 방법이 있음.
  * 부호화-크기 표현
  * 1의 보수 표현
  * 2의 보수 표현

정수의 표현 (4)

(1) 부호화-크기 표현

- 위의 세 가지 표현 방법들 중에서 가장 간단한 방법으로서, 단어의 비트 수가 n 일 때 맨 좌측 비트가 부호 비트이고, 나머지 n-1개의 비트들은 수의 크기를 나타냄.
- 아래에 예가 제시됨.

  * +9 = 0 0001001

    -9 = 1 0001001

- 부호화-크기 표현에서는 첫 째로 덧셈과 뺄셈을 수행하기 위해서 부호 비트와 크기 부분을 별도로 처리해야 한다는 결점이 있음.

  예를 들어, 두 수의 덧셈 과정은 아래와 같음.

  ① 두 수의 부호를 비교함.

  ② 부호가 같은 경우에는 크기 부분들을 더하고, 다른 경우에는 크기 부분의 차이를 구함.

  ③ 크기 부분의 절대값이 더 큰 수의 부호를 결과값의 부호로 세트 함.

정수의 표현 (5)

• 이 표현 방법의 두 번째 결점은 0에 대한 표현이 다음과 같이 두 가지라는 점임.

0 0000000 = +0

1 0000000 = −0

이러한 경우에는 데이터가 0인지를 검사하는 과정이 더복잡해지며, n비트 단어로 표현할 수 있는 수들이 2^n개가 아니라 $(2^n - 1)$개로 줄어듦.

(2) 보수 표현

• 부호화-크기 표현의 결점들을 해결하기 위해 보수 표현이 개발되었는데,

어떤 양수에 대하여 1의 보수와 2의 보수로 표현한 결과는 동일하지만,

수의 부호를 바꾸는 음수화 방법에서는 다음과 같은 차이가 있음.

* 1의 보수 표현 : 모든 비트들을 반전함(0 → 1, 1 →0).

* 2의 보수 표현 : 모든 비트들을 반전하고, 결과값에 1을 더함.

정수의 표현 (6)

- 다음과 같이 음수화를 할 수 있음.

  + 9 = 0 0001001

  – 9 = 1 1110110 (1의 보수)

  – 9 = 1 1110111 (2의 보수)

- 변환하는 과정은 1의 보수가 더 간단하지만, 일반적으로 컴퓨터들에서 2의 보수가 더 많이 사용되고 있는 이유는 동일한 비트들로 표현할 수 있는 수의 개수가 더 많기 때문임. 1의 보수의 표현 개수가 더 적은 이유는 부호화-크기 표현과 마찬가지로 0에 대한 표현이 두 가지이기 때문임.

- 2의 보수로 표현된 2진수에서 부호 비트 $a^{n-1} = 0$인 양수를 10진수로 변환하는 방법은 아래와 같음.

  $A = a_{n-2} \times 2^{n-2} + a_{n-3} \times 2^{n-3} + \cdots a_1 \times 2^1 + a_0 \times 2^0$

- 부호비트 $a^{n-1} = 1$인 음수를 10진수로 변환하면 아래와 같음.

  $A = -2^{n-1} + (a_{n-2} \times 2^{n-2} + a_{n-3} \times 2^{n-3} + \cdots a_1 \times 2^1 + a_0 \times 2^0)$

정수의 표현 (7)

10진수	1의 보수	2의 보수
127	01111111	01111111
126	01111110	01111110
⋮	⋮	⋮
1	00000001	00000001
+0	00000000	00000000
−0	11111111	−
−1	11111110	11111111
−2	11111101	11111110
⋮	⋮	⋮
−126	10000001	10000010
−127	10000000	10000001
−128	−	10000000

• 8비트 보수로 표현된 정수들

정수의 표현 (8)

- 2의 보수 10101110을 10진수로 변환하면 아래와 같음.

  $A = -128 + (1x2^5 + 1x2^3 + 1x2^2 + 1x2^1)$

  $= -128 + (32+8+4+2)$

  $= -82$

  또 다른 방법으로는 ① 2의 보수로 표현된 음수에 대응되는 양수 표현으로 변환한 다음에, ②그 표현에 대한 10진수를 계산하고, - 부호를 붙이는 방법도 있음.

  ① 10101110 → 01010010 : 2의 보수 10101110은 1의 보수로 10101101임.

  ② $A = -(1x2^6 + 1x2^4 + 1x2^1)$

  $= -(64+16+2)$

  $=-82$

정수의 표현 (9)

(3) 비트 확장

- 컴퓨터에서 어떤 수가 기억장치에 저장되어있을 때의 길이와 CPU 연산 과정에서의 길이가 서로 다른 경우가 있는데, 예를 들어서 기억장치에 8비트의 길이로 저장되어 있는 정수를 읽어와서 16비트 레지스터에 저장되어 있는 다른 정수와 더하는 경우에는 읽어온 8비트 정수를 16비트 정수로 길이를 확장한 다음에 덧셈을 해야 함.
- 부호화-크기 표현에서는 단순히 부호 비트를 새로운 맨 좌측 위치로 이동시키고, 그 외의 위치들은 0으로 채우면 됨.
- 예를 들면 아래와 같음.

  +21 = 00010101 (부호화-크기, 8비트)

  +21 = 0000000000010101 (부호화-크기, 16비트)

  -21 = 10010101 (부호화-크기, 8비트)

  -21 = 1000000000010101 (부호화-크기, 16비트)

정수의 표현 (10)

- 반면에 2의 보수인 경우에는 확장된 상위 비트를 부호 비트와 같은 값으로 세트 해야 함.
  즉, 양수의 경우에는 모든 상위 비트들을 0으로 세트하고,
  음수의 경우에는 1로 세트 하는데, 이것을 부호-비트 확장이라고 부름.
- 예를 들면 아래와 같음.
  +21 = 00010101 (2의 보수, 8비트)
  +21 = 0000000000010101 (2의 보수, 16비트)
  -21 = 11101011 (2의 보수, 8비트)
  -21 = 1111111111101011 (2의 보수, 16비트)

2 교시

논리 연산 (1)

• 두 개의 논리 데이터들 A와 B에 대하여 처리될 수 있는 기본적인 논리 연산들은 아래 표와 같음.

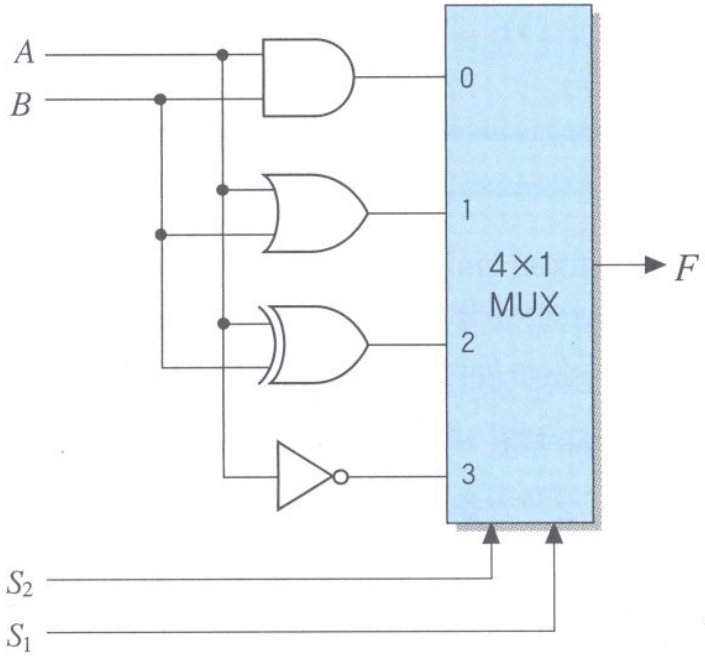
A	*B*	NOT *A*	NOT *B*	*A* AND *B*	*A* OR *B*	*A* XOR *B*
0	0	1	1	0	0	0
0	1	1	0	0	1	1
1	0	0	1	0	1	1
1	1	0	0	1	1	0

• 기본적인 논리 연산들

논리 연산 (2)

• 상기의 표와 같은 비트 단위의 논리 연산들을 위한 하드웨어 모듈은 아래와 같이 구성할 수 있음.

그림에서 입력 비트들은 일단 모든 게이트들을 통과하여 각 논리 연산의 출력을 발생하고, 두 개의 선택 신호들(S_1, S_2)에 의해 선택된 연산의 출력만 4x1 멀티플렉서의 출력으로 나가게 됨.

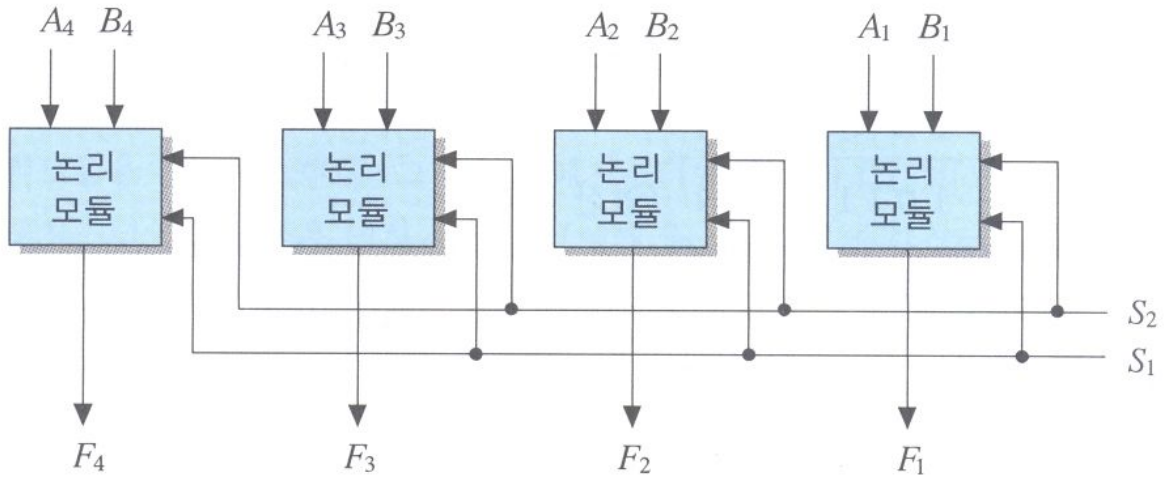


S_2	S_1	출력	연산
0	0	$F = A \wedge B$	AND
0	1	$F = A \vee B$	OR
1	0	$F = A \oplus B$	XOR
1	1	$F = \overline{A}$	NOT

• 논리 연산을 위한 하드웨어 모듈

논리 연산 (3)

- n 비트 데이터들에 대하여 논리 연산을 수행하는 장치는 상기의 하드웨어 모듈 n개를 이용하여 구성할 수 있음.
- 한 예로서, 4 비트 데이터를 위한 논리 연산장치는 아래 그림과 같음.
  각 모듈에는 두 단어들에서 동일한 위치에 있는 비트들이 입력되며, 연산 선택 신호인 S_1과 S_2는 모든 모듈들에 공통으로 가해짐.
  논리 연산에서는 대응되는 비트들 간에 독립적으로 연산이 수행되므로 모듈간의 데이터 전송 통로는 필요하지 않음.



• 4 비트 논리 연산장치

논리 연산 (4)

(1) AND 연산

- 두 데이터의 동일한 위치에 있는 비트들이 모두 1인 경우에는 결과 데이터의 해당 비트가 1이 되고, 어느 하나라도 0이면 그 비트는 0이 됨.
- 예를 들면 아래와 같음.

A = 1 0 1 1 0 1 0 1

B = 0 0 1 1 1 0 1 1

---

0 0 1 1 0 0 0 1 (연산 결과)

(2) OR 연산

- 두 데이터의 동일한 위치에 있는 비트들 중의 어느 하나만 1이면 결과 데이터의 해당 비트가 1이 되고, 두 비트들이 모두 0이면 그 비트는 0이 됨.
- 예를 들면 아래와 같음.

A = 1 0 1 1 0 1 0 1

B = 0 0 1 1 1 0 1 1

---

1 0 1 1 1 1 1 1 (연산 결과)

논리 연산 (5)

(3) XOR 연산

- 두 데이터의 동일한 위치에 있는 비트들이 서로 다른 값을 가지면 결과 데이터의 해당 비트가 1이 되고, 같은 값을 가지면 그 비트는 0이 됨.
- 예를 들면 아래와 같음.

A = 1 0 1 1 0 1 0 1

B = 0 0 1 1 1 0 1 1

1 0 0 0 1 1 1 0 (연산 결과)

(4) NOT 연산

- 이 연산을 수행하면 데이터 단어의 모든 비트들이 반전(invert)됨. 즉 비트의 값이 1이면 0으로, 0이면 1로 바뀜.
- 예를 들면 아래와 같음.

A = 1 0 1 1 0 1 0 1 (연산 전)

A = 0 1 0 0 1 0 1 0 (연산 결과)

논리 연산 (6)

(5) 선택적-세트 연산

- 이 연산은 B 레지스터의 비트들 중에서 1로 세트 된 비트들과 동일한 위치에 있는 A 레지스터의 비트들을 1로 세트 하는 것인데, 두 레지스터 사이에 OR 연산으로 구현됨.
- 예를 들면 아래와 같음.

A = 1 0 0 1 0 0 1 0 (연산 전)

B = 0 0 0 0 1 1 1 1

---

1 0 0 1 1 1 1 1 (연산 결과)

(6) 선택적-보수 연산

- 이 연산은 B 레지스터의 비트들 중에서 1로 세트 된 비트들과 동일한 위치에 있는 A 레지스터의 비트들을 보수로 바꾸는 것인데, 두 레지스터 사이에 XOR 연산으로 구현됨.
- 예를 들면 아래와 같음.

A = 1 0 0 1 0 1 0 1 (연산 전)

B = 0 0 0 0 1 1 1 1

---

1 0 0 1 1 0 1 0 (연산 결과)

논리 연산 (7)

(7) 마스크 연산

- 이 연산은 B 레지스터의 비트들 중에서 0인 비트들과 동일한 위치에 있는 A 레지스터의 비트들을 0으로 바꾸는 것인데, 두 레지스터 사이에 AND 연산으로 구현됨.
- 예를 들면 아래와 같음.

A = 1 1 0 1 0 1 0 1 (연산 전)

B = 0 0 0 0 1 1 1 1

0 0 0 0 0 1 0 1 (연산 결과)

(8) 삽입 연산

- 이 연산은 새로운 비트 값들을 데이터 단어 내의 특정 위치에 삽입하는 것임.
- 이 연산은 두 단계로 이루어지는데, 첫 번째 단계에서는 삽입하고자 하는 비트 위치들에 대해 마스크 연산(AND 연산)을 수행하고, 두 번째 단계에서는 새로이 삽입할 비트들과 OR 연산을 수행함.

논리 연산 (8)

- 8 비트 데이터의 좌측 네 비트에 새로운 비트들 (1110)을 삽입하는 예는 아래와 같음.

① A = 1 0 0 1 0 1 0 1

B = 0 0 0 0 1 1 1 1 마스크 (AND 연산)

A = 0 0 0 0 0 1 0 1 첫 단계 결과

② A = 0 0 0 0 0 1 0 1

B = 1 1 1 0 0 0 0 0 삽입 (OR 연산)

A = 1 1 1 0 0 1 0 1 최종(삽입) 결과

논리 연산 (9)

(9) 비교 연산

- 이 연산은 A와 B 레지스터의 내용을 비교하여, 만약 두 레지스터의 대응되는 비트들의 값이 같으면, A 레지스터의 해당 비트가 0으로 세트 되고, 서로 다르면 1로 세트 됨. 이 연산은 XOR 연산으로 수행되는데, 만약 두 레지스터의 모든 비트들이 동일하다면 A 레지스터의 모든 비트들이 0이 되어 상태 레지스터에 있는 Z(zero) 플래그가 세트 되므로 두 레지스터의 내용을 비교한 결과는 Z 플래그를 검사함으로써 확인 가능함.
- 예를 들면 아래와 같음.

A = 1 1 0 1 0 1 0 1

B = 1 0 0 1 0 1 1 0

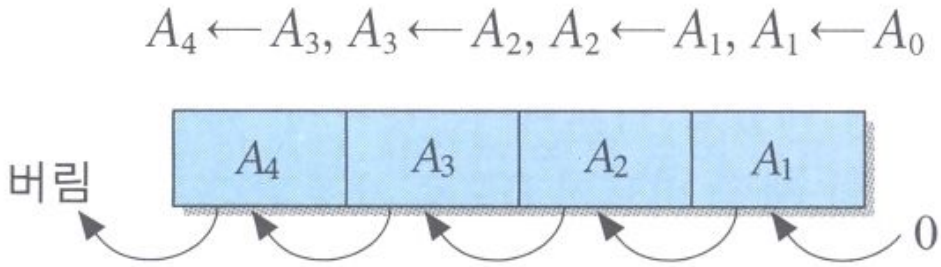
0 1 0 0 0 0 1 1 (연산 결과)

3 교시

쉬프트 연산 (1)

(1) 논리적 쉬프트(logical shift)

- 쉬프트 연산(shift operation)이란 레지스터의 데이터 비트들을 왼쪽 혹은 오른쪽으로 한 칸씩 이동시키는 것을 말함.
- 예를 들어서, 아래 그림과 같이 4 비트 레지스터에 저장된 데이터 A = A_4 A_3 A_2 A_1에 대하여 좌측-쉬프트 연산이 수행되면 비트들이 왼쪽으로 한 칸씩 이동하고, 맨 우측 비트로는 0이 들어옴.



• 4 비트 레지스터에서의 좌측 쉬프트

- 4 비트 레지스터는 네 개의 플립플롭들로 구성되므로 좌측 쉬프트는 각 플립플롭의 출력이 좌측에 위치한 플립플롭으로 입력되는 것을 의미함. 여기에서 유의할 사항은 최하위 비트(A_1)로는 0이 들어오고, 최상위 비트(A_4)는 버리게 된다는 점임.

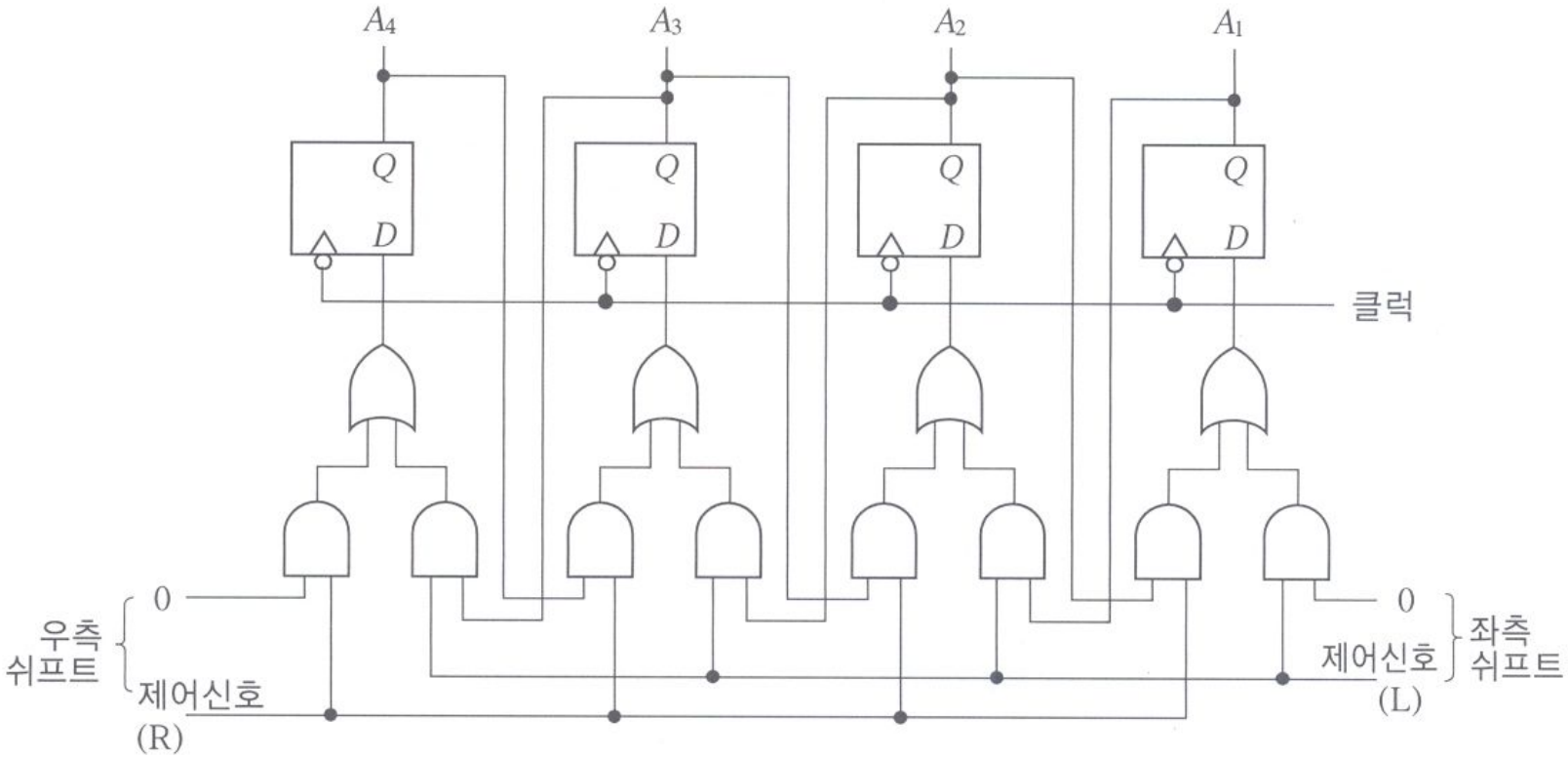
쉬프트 연산 (2)

- 반대 방향의 연산인 우측-쉬프트에서는 모든 비트들이 우측으로 한 칸씩 이동하게 되고, 최상위 비트(A_4)로 0이 들어오며, 최하위 비트(A_1)는 버림.
- 어떤 레지스터에 4 (0100)가 저장되어 있을 때, 우측-쉬프트를 하면 2 (0010)가 되어서 원래의 수를 2로 나눈 값이 되고, 반대로 좌측-쉬프트를 하면 8 (1000)이 되어 2와 곱한 결과가 되는데, 이와 같이 쉬프트는 간단한 곱셈과 나눗셈에도 사용될 수 있음.
- 아래 그림은 D 플립플롭을 이용한 쉬프트 레지스터(shift register)의 내부 구성도를 보여줌.

  좌측 쉬프트를 원하는 경우에는 제어 신호 L을 1로 세트 하면 모든 AND 게이트들의 한쪽 입력들이 1로 세트 되어서 우측 플립플롭의 출력을 D 입력으로 받을 수 있게 되고 맨 우측 AND 게이트로는 0이 입력되는데 이 입력들은 클럭 신호가 들어오는 순간에 각 플립플롭에 저장됨.

  우측 쉬프트의 경우에는 제어 신호 R을 1로 세트 하면 각 플립플롭이 좌측 플립플롭의 출력을 입력으로 받을 수 있게 되며, 맨 좌측 AND 게이트로는 0이 입력됨.

쉬프트 연산 (3)



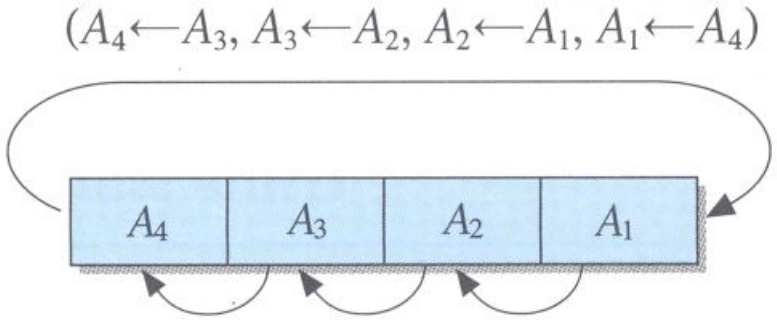
• 쉬프트 레지스터의 내부 구성도

쉬프트 연산 (4)

(2) 순환 쉬프트(circular shift)

• 순환 쉬프트는 회전(rotate)이라고도 부르는데 이 연산은 논리적 쉬프트와 근본적으로 동일하지만, 최상위 혹은 최하위에 있는 비트를 버리지 않고 반대편 끝에 있는 비트 위치로 들어가게 한다는 것이 다른 점임.

즉, 순환 좌측-쉬프트 연산은 최상위 비트인 A_4가 최하위 비트 위치인 A_1으로 이동함.



• 4 비트 레지스터에서의 순환 좌측-쉬프트

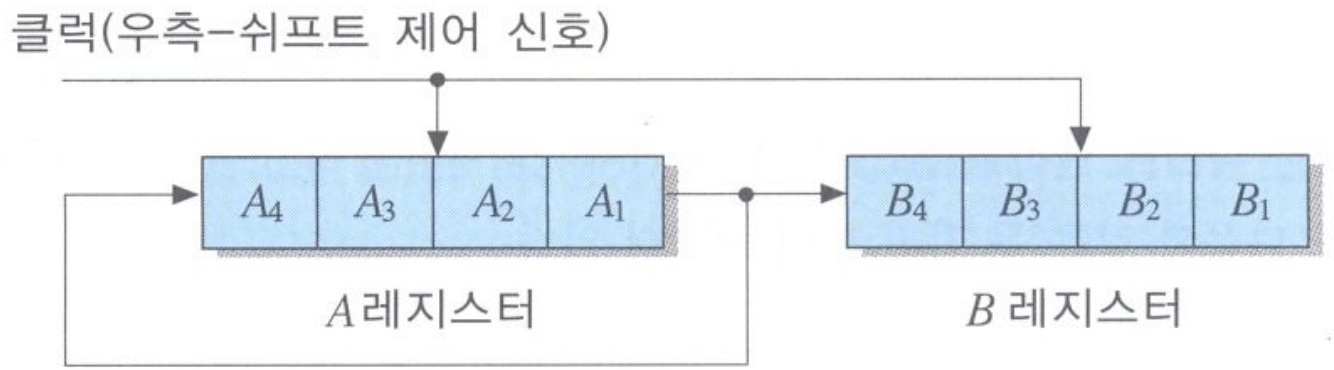
• 순환 우측-쉬프트인 경우에는 (A_4 → A_3, A_3 → A_2, A_2 → A_1, A_1 → A_4)가 되고, 논리적 쉬프트와는 달리 외부로부터 0 입력이 들어오지 않음.

쉬프트 연산 (5)

- 논리적 쉬프트와 순환 쉬프트 연산을 이용하면 레지스터들 사이에 직렬 데이터 전송이 가능함.

  이 동작은 쉬프트 연산을 데이터 수만큼 연속적으로 수행함으로써 두 레지스터들 사이에 한 개의 선을 통하여 전체 데이터를 이동시키는 것임.

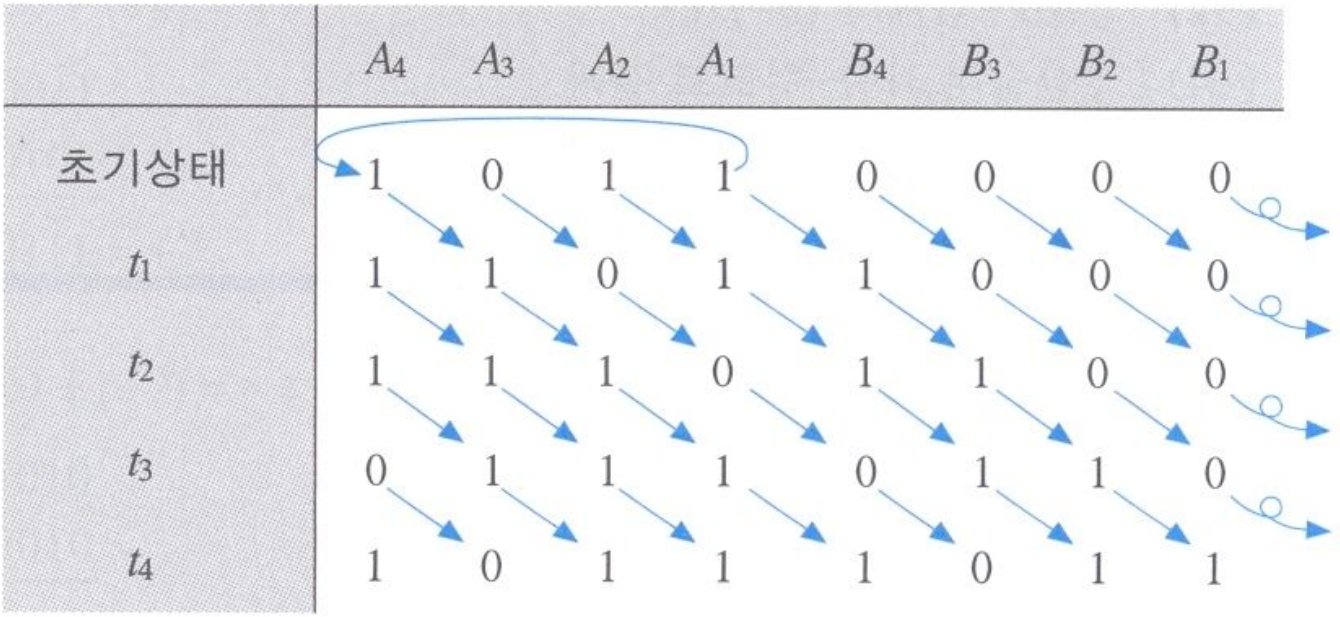
  한 예로서, 아래 그림과 같이 A 레지스터의 내용을 B 레지스터로 전송하는 경우를 살펴보고자 함.



- 쉬프트 레지스터를 이용한 직렬 데이터 전송

쉬프트 연산 (6)

- 이 전송에서 각 클럭 주기마다 한 번씩의 쉬프트 연산이 일어나므로, 4 비트 레지스터들 간의 직렬 전송을 위해서는 A 레지스터에서 네 번의 순환 쉬프트가 연속적으로 수행되고, B 레지스터에서는 네 번의 논리적 쉬프트가 수행되어야 함.
- 예를 들어서, A 레지스터에 1011이 저장된 경우에 B 레지스터로 직렬 전송이 수행되는 과정을 클럭 주기별로 보면 아래와 같음.



	A_4	A_3	A_2	A_1	B_4	B_3	B_2	B_1
초기상태	1	0	1	1	0	0	0	0
t_1	1	1	0	1	1	0	0	0
t_2	1	1	1	0	1	1	0	0
t_3	0	1	1	1	0	1	1	0
t_4	1	0	1	1	1	0	1	1

• 4 비트 레지스터들 간의 직렬 전송의 예

셀프 테스트

• -12에 대한 2의 보수는 1바이트로 표시하면 어떤 값이 되는가?

* 11110100

해설) 양수 12 크기는 1100 이고 이것에 1의 보수를 취하면 0011이 되며 2의 보수는 여기에 1을 더하므로 0100 이 됨. 그런데 -12이므로 부호 비트를 앞에 두어 10100 이 되고 8비트로 확장하기 위해 나머지 앞의 세 비트들을 부호비트 즉 1로 세트 하므로 11110100 이 됨.

• 데이터 01110101을 00111111로 마스크 연산을 수행하면 결과는 무엇인가?

* 00110101

해설) 마스크 연산은 AND 연산이므로 01110101 AND 00111111의 결과가 해답이 됨.

• 데이터 01001011에 대해 순환 좌측-쉬프트를 1회 수행하면 결과는 무엇이 되는가?

* 10010110

해설) 순환 좌측-쉬프트의 경우에는 맨 좌측 비트가 맨 우측 끝자리로 이동하고 원래 위치의 비트들은 각각 한 비트씩 좌측으로 이동함.

요점 정리

- ALU의 내부 요소들에는 산술 연산장치, 논리 연산장치, 쉬프트 레지스터, 보수기, 상태 레지스터 등이 있음.
- 2진수 체계에서는 어떤 수를 0과 1, 부호 및 소수점으로 표현함.
- 부호 비트 사용 표현에는 부호화-크기 표현, 1의 보수 표현, 2의 보수 표현 등이 있음.
- 어떤 양수에 대하여 1의 보수와 2의 보수로 표현한 결과는 동일하지만, 수의 부호를 바꾸는 음수화 방법에서 1의 보수 표현은 모든 비트들을 반전시키고 2의 보수 표현에서는 모든 비트들을 반전하고, 결과값에 1을 더함.
- 논리 연산에는 AND 연산, OR 연산, XOR 연산, NOT 연산, 선택적-세트 연산, 선택적-보수 연산, 마스크 연산, 삽입 연산, 비교 연산 등이 있음.
- 쉬프트 연산(shift operation)이란 레지스터의 데이터 비트들을 왼쪽 혹은 오른쪽으로 한 칸씩 이동시키는 것을 말함.
- 순환 쉬프트는 회전(rotate)이라고도 부르는데 이 연산은 논리적 쉬프트와 근본적으로 동일하지만, 최상위 혹은 최하위에 있는 비트를 버리지 않고 반대편 끝에 있는 비트 위치로 들어가게 한다는 것이 다른 점임.