

프로그래머가 알아야 하는 2진수 기반의 컴퓨터 동작 원리

Sunny Kwak

(sunnykwak@daum.net)

목 차

- **반도체와 디지털, 그리고 CPU**
 - 디지털의 기원, 반도체
 - 진공관, 트랜지스터, 그리고 IC
 - 비트, 바이트 그리고 워드
 - 산술/논리 장치 (ALU)
- **2진수 산술 연산**
 - 십진법과 이진법
 - 진법 변환
 - 보수 (Complementary Number)
 - 고정 소수점 과 부동 소수점
 - 정수 산술 연산 (integer arithmetic operation)
 - 2진수의 음수화 및 사칙연산
 - 부동 소수점 연산
- **2진수 논리 연산**
 - 명제, 참/거짓
 - 논리연산과 진리표
 - 논리 연산의 우선 순위와 법칙
 - 논리 연산 명령
- **2진수 문자 처리**
 - 문자의 표현 방식
 - 문자 코드 체계
 - ASCII 코드 표
 - 2 byte 완성형 코드
 - 유니코드

왜 컴퓨터는 세상을 2진수로 이해할까?

반도체와 디지털 그리고 CPU

디지털의 기원, 반도체

- 도체, 부도체, 반도체

- 도체 (conductor) : 구리처럼 전류가 흐르는 물질.
- 부도체 (nonconductor) : 유리처럼 전류가 흐르지 않는 물질.
- 반도체 (semiconductor) : 전류가 흐르는 성질이 도체와 부도체의 중간 정도

半 + 導體 또는 Semi + Conductor

- 반도체의 의미

- 원래는 거의 전기가 통하지 않지만 빛이나 열, 또는 불순물을 가해주면 전기가 통하고 또한 조절도 할 수 있는 물질
- 어떻게 조작하느냐에 따라 전기 흐름을 조절할 수 있다는 점을 응용해 전류가 흐르는가 아닌가에 따라 2 가지 상태를 나타낼 수 있다.
- 숫자로 표현하면 0 혹은 1이 되며, 이를 디지털(digital)이라 말한다.

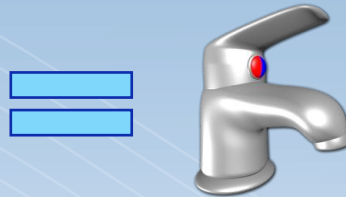
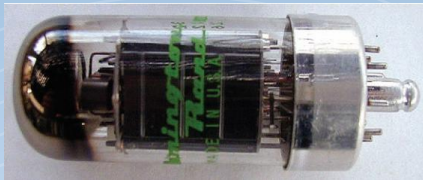
- 디지털 기기

- 반도체의 특성을 이용해 정보를 '0'과 '1'의 조합으로 기록하고 처리하는 장치.

진공관, 트랜지스터, 그리고 IC

- **진공관 (vacuum tube)**

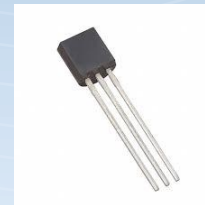
- 초기 통신기술 연구 단계에서 장거리를 이동하는 도중에 전기신호가 약해지는 현상이 나타났고, 중간에 전기 신호를 증폭시키기 위해 '반도체'를 사용할 수 있다는 것이 알려지고, 신호 증폭을 위해 발명된 것이 '진공관'이다.
- 증폭 및 감쇠 시킬 수 있다면, 아예 신호를 계속 유지하거나 막을 수도 있으므로 2가지 상태를 나타낼 수 있다! 다만, 처리 속도는 그리 빠르지 않음. (ms, 10^{-3} sec 이하)



수도꼭지에 비유하자면,
물을 틀면 1, 확실히 잠그면 0이 된다.

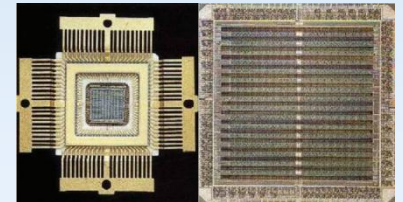
- **트랜지스터 (transister)**

- 1948년, 벨전화연구소의 과학자 윌리엄 쇼클리, 존 바딘과 월터 브래튼이 트랜지스터를 발명하고 노벨상을 수상.
- 진공관에 비해 크기가 매우 작고, 안정적이며, 처리 속도가 1,000 배 이상 빠르다. (μ s, 10^{-6} sec 이하)



- **IC (Integred Chip)**

- 미국 Texas Instrument I社の 기술자, 잭 킬비(Jack Kilby)가 집적 회로(IC) 발명.
- 손톱 수준의 크기의 칩(chip)에 수만 ~ 수십억 개의 트랜지스터나 다른 전자부품을 집합시킨 것. 처리 속도가 더욱 빨라졌다. (ns, 10^{-9} sec 이하)



비트, 바이트 그리고 워드

- 2진수 기반

- 진공관 및 트랜지스터가 두 가지 상태만을 나타낼 수 있으므로, 자연스럽게 컴퓨터는 모든 데이터를 2진수(binary numeral)로 표현하게 된다.

- 비트 (bit)

- **binary digit**의 약자이며, 산술적으로는 0 혹은 1의 값을 가질 수 있다. 논리적으로는 참(true) 혹은 거짓(false)을 나타낸다.
- 하나의 컴퓨터 소자(素子)는 하나의 bit 값을 저장할 수 있다.

- 바이트 (byte)

- 비트(bit)가 여러 개 모인 것으로, 원래는 크기가 명확히 정해져 있지 않았지만, 점차 1 옥텟(octet)인 8 비트를 1 바이트로 정의하게 되었다.
- 바이트의 실질적 의미는 ASCII 문자(character) 하나를 나타낼 수 있다는 것이다.

- 워드 (word)

- 한번의 기계어 명령어 수행을 통해 저장된 장치(메모리 혹은 I/O 장치)로부터 레지스터(register)에 옮겨 놓거나, ALU을 통해 조작할 수 있는 데이터 단위이다. 32비트 CPU라면 워드의 크기는 32비트가 된다.

산술/논리 연산 장치 (ALU)

- **ALU 의 역할**

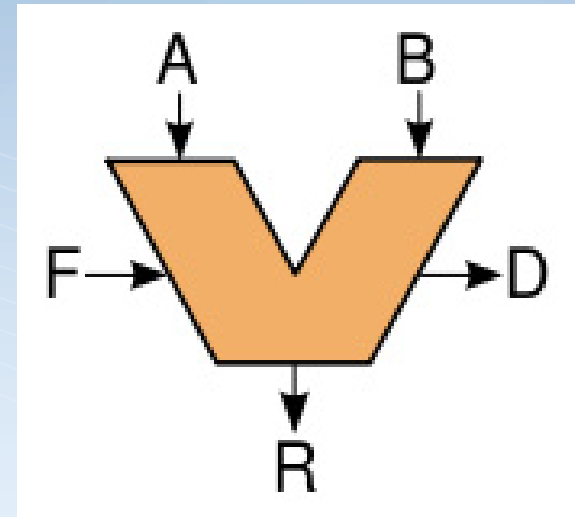
- 'Arithmetic Logic Unit' 혹은 ALU는 매우 다양한 산술(arithmetic) 및 논리(logic) 연산을 수행한다.
- ALU는 CPU의 '심장(heart)'이며, 심지어 CPU의 나머지 모든 부품은 ALU를 지원하기 위해 존재한다고 말할 수 있다.
- '산술 연산'은 더하기, 빼기, 곱하기, 나누기 등 사칙연산 등이며, '논리 연산'은 논리합(OR), 논리곱(AND), 부정(NOT) 등 부울린(boolean) 계산 명령이다.

- **ALU 기호 기호(Typical Schematic Symbol)**

- A and B : ALU 입력 (operands)
- R : 출력 혹은 결과
- F : 제어 장치에서 입력되는 코드 혹은 명령 (op-code)
- D : 출력 상태

- **ALU의 계산 방식**

- ALU 는 모든 산술/논리 연산을 2진수로 처리한다.
- 따라서 소프트웨어의 동작 원리를 깊이 이해하려면 2진법을 학습해야 한다.



컴퓨터의 진법

2진수 산술 연산

십진법과 이진법

- **십진법 (decimal system)**

- 0 ~ 9의 10가지의 기호를 이용하여 수를 표현
- 10의 제곱(10^n)으로 자릿수(n)의 값을 계산
- $(724)_{10} = 7 \times 10^2 + 2 \times 10^1 + 4 \times 10^0$

- **이진법 (binary system)**


- 0과 1만을 가지고 수를 표현
- 2의 제곱(2^n)으로 자릿수(n)의 값을 계산
- $(101101)_2 = 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = (45)_{10}$

진법 변환

- 십진수를 이진수로의 변환

- 연속적으로 2로 나눗셈을 수행하면서 얻어지는 나머지에 의해서 만들어진다.
- $(41)_{10}$ 을 이진수로 변환
 - 41을 2로 연속해서 나눗셈
 - 생성된 나머지를 역순으로 정렬
 - $(41)_{10} = (101001)_2$

2		41		
2		20	1
2		10	0
2		5	0
2		2	1
		1	0



- 이진수를 십진수로의 변환

- 이진수의 '각 자릿수(n) × 2ⁿ' 의 계산 결과를 합산.
- $(101001)_2$
 - $= 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$
 - $= 32 + 8 + 1$
 - $= (41)_{10}$

보수 (Complementary Number)

- 2의 보수 표기법 (2's complement representation)

- 컴퓨터에서 음수(minus number)를 표현하는 방법은 '부호의 절대값, '1의 보수', '2의 보수' 등이 있다.
- 2의 보수를 사용하면, 계산이 편리하기 때문에 일반적으로 '2의 보수'를 이용해 음수(minus number)를 표현한다.

- '1의 보수'와 '2의 보수' 생성

- '1의 보수'는 2진수로 표현된 양수의 0은 1로, 1은 0으로 치환해 생성한다.

+3	0	0	0	0	0	0	1	1
-3	1	1	1	1	1	1	0	0

- '2의 보수'는 1의 보수에 1을 더해 생성한다.

+3	0	0	0	0	0	0	1	1
-3	1	1	1	1	1	1	0	1

고정 소수점 과 부동 소수점 (1/2)

- 고정 소수점 (fix-point arithmetic)

- 소수점의 위치가 고정되어 있다고 가정하고 수치(number)를 표현하는 방식
- 예를 들어, 1203 이라는 수가 있을 때 소수점 위치가 2번째 라고 하면 12.03이라는 값을 의미한다.
- 컴퓨터는 소수점의 위치가 0번째에 있다고 가정하고 정수(integer) 값을 저장하기 위한 형식으로 사용한다. 소수점의 위치가 고정되어 있어 계산이 용이하나 매우 큰 값이나 작은 값을 표현할 수 없다.
- 1비트 부호부(sign bit)와 나머지 정수부(integer part)로 구성된다.

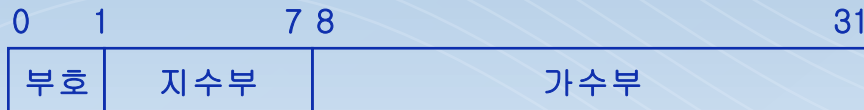


고정 소수점 과 부동 소수점 (2/2)

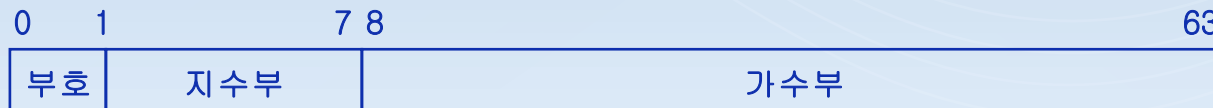
- **부동 소수점 (Floating point arithmetic)**

- 소수점의 위치를 고정하지 않고 그 위치를 나타내는 수를 따로 적어 실수를 표현하는 방식
- 고정 소수점 방식보다 매우 크거나 작은 넓은 범위의 수를 나타낼 수 있다.
- 양수와 음수를 구분하는 부호부(sign bit), 유효숫자를 나타내는 가수부(mantissa), 소수점의 위치를 나타내는 지수부(exponent)로 나누어 표현한다.

단정도 실수형 (single precision, 4 byte)

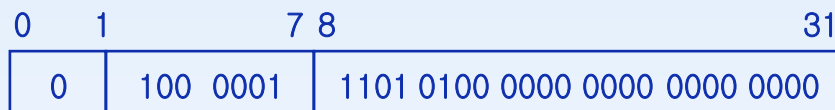


배정도 실수형 (double precision, 8 byte)



예시 : $13.25_{10} \rightarrow D.4_{16} \rightarrow 0.D4 \times 16^1$

부호 : 0, 지수부 : $1+64=65(100\ 0001)$, 소수부 : $D4(1101\ 0100)$



정수 산술 연산 (integer arithmetic operation)

- 정수 연산 = 음수 변환 그리고 사칙연산, 5 종 세트
- 음수화(negation)
 - 양의 2진수를 2의 보수로 변환.
- 덧셈
 - A와 B 2개의 2진수를 더해 C 값을 출력.
- 뺄셈
 - A에서 B (2의 보수)를 더한 값 C를 출력
 - 컴퓨터는 2의 보수를 활용해 뺄셈도 덧셈으로 한다.
- 곱셈
 - A와 B 2개의 2진수를 곱해 C 값을 출력.
- 나눗셈
 - A를 B로 나누고 몫(quotient)과 나머지(remainder)를 출력

2진수의 음수화 (negation)

- 양수 값을 '2의 보수'로 변환

$$\begin{array}{rcl} +19 : & 00010011 & \\ 1의\ 보수 : & 11101100 & \\ & + \quad 1 & \\ \hline -19 : & 11101101 & \end{array}$$

2진수 덧셈

『 양수와 양수의 덧셈 』

$$2 + 3 = 5$$

$$\begin{array}{r} 0010 \\ + 0011 \\ \hline 0101 \end{array}$$

『 음수와 양수의 덧셈 』

(자리 올림수가 발생하지 않음)

$$-6 + 3 = -3$$

$$\begin{array}{r} 1010 \\ + 0011 \\ \hline 1101 \end{array}$$

『 음수와 양수의 덧셈 』

자리 올림수가 발생

$$(-3) + 5 = 2$$

$$\begin{array}{r} 1101 \\ + 0101 \\ \hline 1\ 0010 \end{array}$$

버림

『 음수와 음수의 덧셈 』

자리 올림수가 발생

$$-2 + (-4) = -6$$

$$\begin{array}{r} 1110 \\ + 1100 \\ \hline 1\ 1010 \end{array}$$

버림

2진수 뺄셈

『 자리 올림수가 발생하지 않는 경우 』

$$2 - 5 = +2 + (-5) = -3$$

$$\begin{array}{r} 0010 \\ + 1011 \\ \hline 1101 = -3 \end{array}$$

『 자리 올림수가 발생하는 경우 』

$$5 - 2 = 5 + (-2) = 3$$

$$\begin{array}{r} 0101 \\ + 1110 \\ \hline 10011 = +3 \end{array}$$

1
버림

2진수 곱셈

- $A \times B = C$

- 곱하는 수(B)를 승수(multiplier), 곱하여지는 수(A)를 피승수(multiplicand)라고 함
- 4비트의 두 수를 곱하면, 2배인 8비트 길이의 결과를 출력한다.

				×	1	1	0	1	피승수(13)
					1	0	1	1	승수(11)
<hr/>									
					1	1	0	1	부분 합
			1		1	0	1		(partial product)
		0	0		0	0			
	1	1	0		1				
<hr/>									
1	0	0	0		1	1	1	1	곱의 결과(143)

2진수 나눗셈

- $D \div V = Q \dots R$
 - 나누어지는 수 D를 피제수(dividend), 나누는 수 V를 제수(divisor)
 - 나눗셈의 결과 몫(quotient) Q와, 나머지 수(remainder) R

		00001101	← 몫
제수 →	1011	10010011	← 피제수
		1011	

부분나머지	→	001110	
		1011	

부분나머지	→	001111	
		1011	

		100	← 나머지

부동 소수점 연산

- 부동 소수점 연산 기본 원리
 - 가수와 지수의 연산을 분리해서 수행한다.
- 덧셈과 뺄셈
 - 지수를 같은 값으로 조정한 후, 가수들에 대하여 덧셈과 뺄셈을 수행
- 곱셈과 나눗셈
 - 가수끼리는 곱셈과 나눗셈을 수행하고, 지수의 연산에서는 곱셈의 경우는 덧셈을 수행하고 나눗셈의 경우에는 뺄셈을 한다
- 정밀도
 - 부동 소수점 연산은 오차가 발생할 수 있으므로 정밀한 값을 계산하고자 할 경우, 정밀도 높은 연산을 제공하는 계산 라이브러리를 활용해야 한다.

컴퓨터의 논리

2진수 논리 연산

명제, 참/거짓

- 명제
 - 논리학적으로 뜻이 분명한 문장을 말한다.
 - 즉, 어떤 말을 딱 본 순간 '참' 혹은 '거짓'을 대번에 알 수 있는 말을 말한다.
 - 예를 들어, 2는 3보다 작다. 2는 3보다 작고, 2는 1보다 작다.
- 참, 거짓 또한 이진수(binary)로 표현
 - 컴퓨터는 0과 1만을 읽고 쓸 수 있기 때문에 '참', '거짓' 또한 2진수로 표현
 - 일반적으로 0은 '거짓', 1은 '참'을 나타낸다.
- 논리 연산
 - 참과 거짓이 피 연산자인 연산

논리연산과 진리표

- 1854년 부울(Boole)

- "사고의 논리" 발표
- 인간의 사고는 명제(참/거짓)과 명제를 연산의 대상으로 하는 '논리합', '논리곱', '논리부정'으로 이루어 진다고 설명.

- 논리연산의 요소

- 피연산자 및 연산의 결과 : 참(1), 거짓(0)
- 연산자 : 논리곱, 논리합, 논리부정
- 논리 연산의 피연산자 및 결과가 0, 1이므로 입출력 관계를 진리표로 표현 가능.

X	Y	$X+Y$
0	0	0
0	1	1
1	0	1
1	1	1

논리합 진리표

X	Y	$X * Y$
0	0	0
0	1	0
1	0	0
1	1	1

논리곱 진리표

X	Y
0	1
1	0

논리부정 진리표

논리 연산의 우선 순위와 법칙

- 논리연산 우선 순위
 - 논리부정
 - 논리곱
 - 논리합
 - 괄호를 이용하여 우선순위 변경 가능
 - $X + YZ = X + (YZ)$
 - $(X+Y) Z$
- 교환법칙
 - $X+Y = Y+X$
 - $XY = YX$
- 결합법칙
 - $(X+Y) + Z = X + (Y+Z)$
 - $(XY)Z = X(YZ)$
- 분배법칙
 - $X(Y+Z)=XY+XZ$
 - $X+(YZ)=(X+Y)(X+Z)$
- 드모르간의 법칙
 - $X+Y)=(XY)$
 - $(XY)=X+Y$

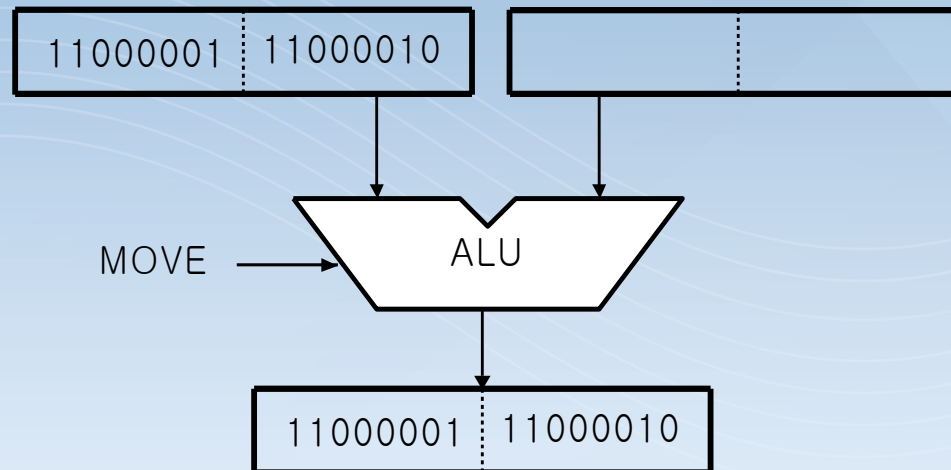
논리 연산 명령

- 컴퓨터에서 제공하는 대표적인 논리 연산 명령들은 다음과 같다.
- 단항 연산 (unary operation)
 - move : 레지스터의 데이터를 다른 레지스터로 이동
 - complement : 보수 연산
 - shift : 입력 데이터의 비트를 이웃한 비트로 자리를 옮기는 연산
 - rotate : shift 와 유사하나 한쪽에서 밀려난 데이터가 다른 쪽으로 들어옴
 - increment : 입력 데이터 값을 1 만큼 증가
 - decrement : 입력 데이터 값을 1만큼 감소
- 이항 연산 (binary operation)
 - and : 2개의 데이터를 입력 받아 비트 간의 논리곱을 연산
 - or : 2개의 데이터를 입력 받아 비트 간의 논리합을 연산
 - xor : 2개의 데이터를 입력 받아 배타적 논리합을 연산
 - compare : 2개의 데이터의 값을 비교해, '같다', '크다', '작다'를 기록

이동 연산

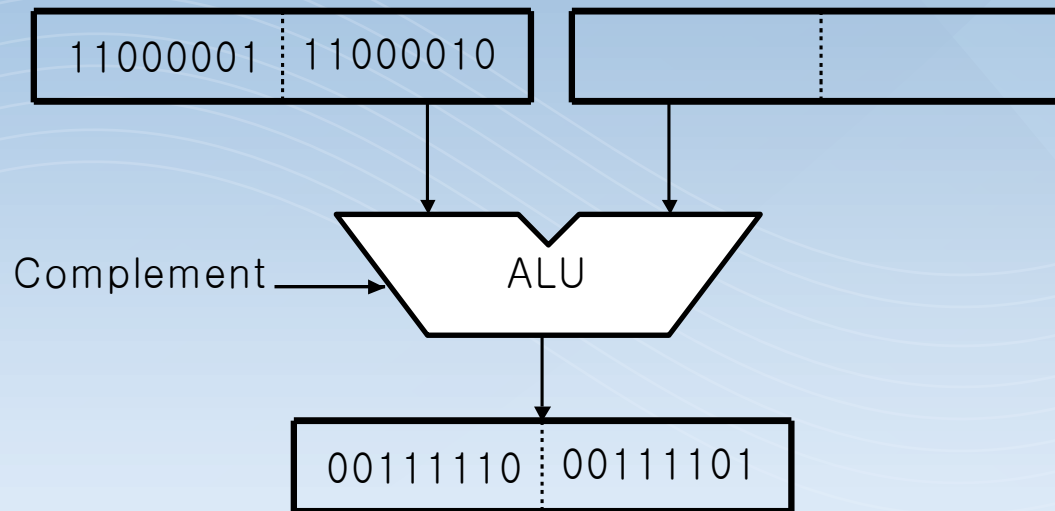
- 이동 (move) 연산

- 연산 장치의 입력 데이터를 받아들여 그대로 출력하는 단항 연산
- 레지스터에 상수 값을 설정하거나,
- 특정 레지스터의 데이터를 다른 레지스터로 옮기는 등의 작업.



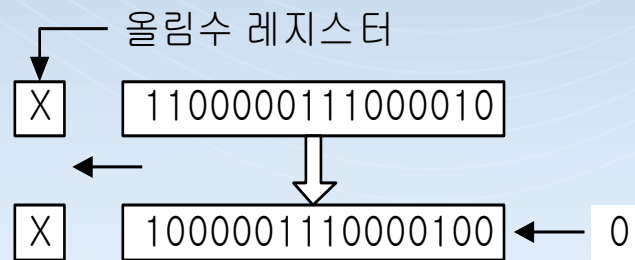
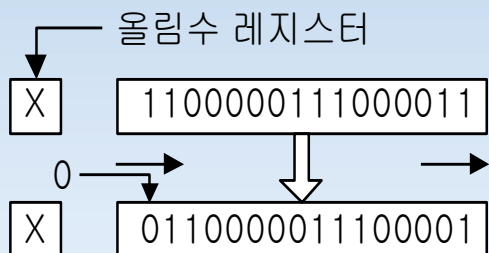
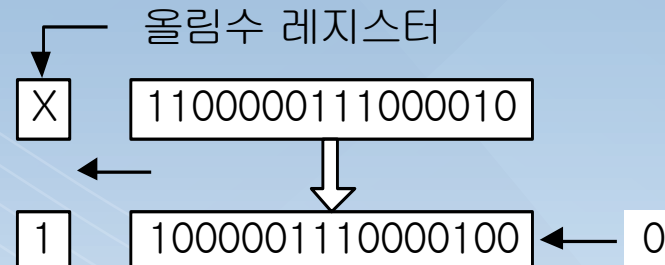
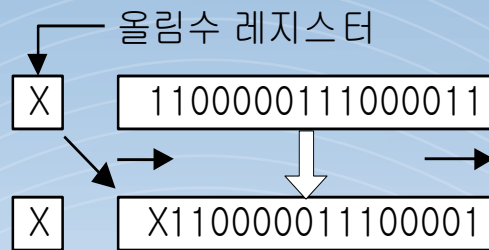
보수 연산

- 보수(COMPLEMENT) 연산
 - 입력 데이터의 1의 보수 값을 출력하는 연산.
 - 1의 보수 및 2의 보수를 생성하기 위해 사용함.



시프트 연산

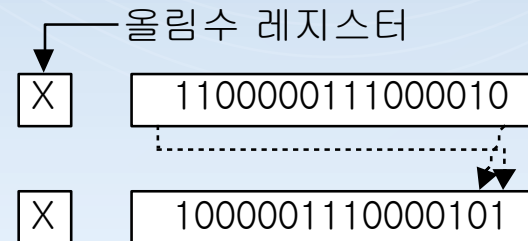
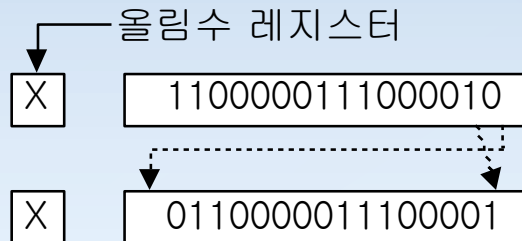
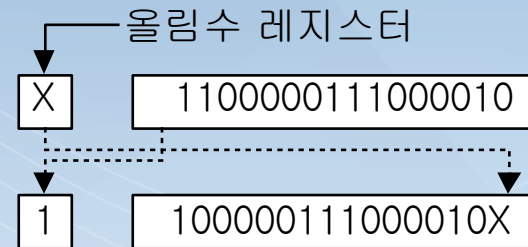
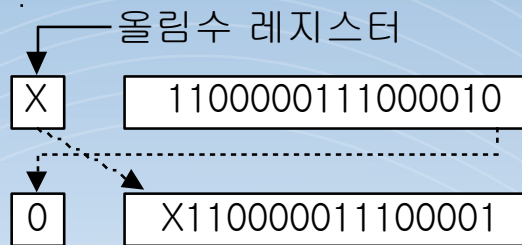
- 시프트(shift) 연산
 - 입력 데이터의 모든 비트들을 각각 서로 이웃한 비트로 자리를 옮기는 연산
 - 우측 시프트(right shift)와 좌측 시프트(left shift)가 있다.



로테이트 연산

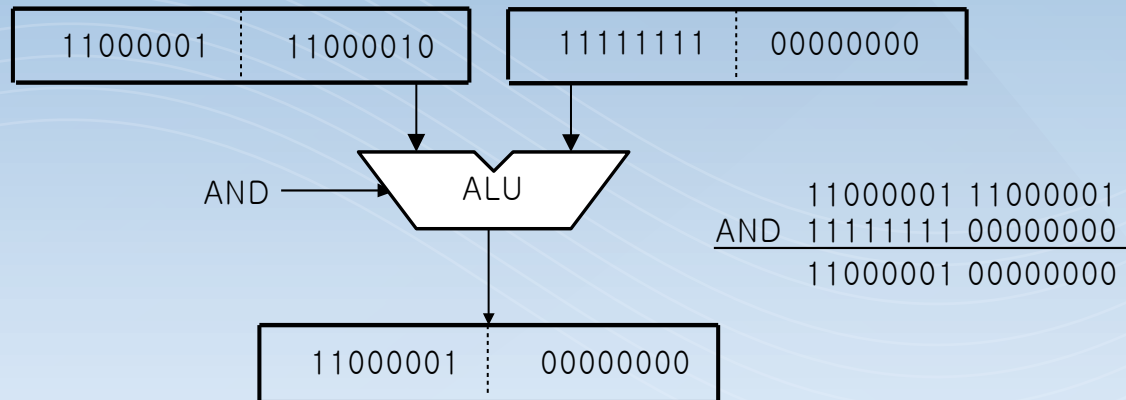
- 로테이트(rotate) 연산

- 로테이트 연산은 시프트와 유사한 연산으로 한쪽 끝에서 밀려나가는 비트가 다시 반대편 끝으로 들어오게 되는 것으로 회전을 의미.
- 데이터의 특정한 비트의 검색이나 비수치적 데이터에서 문자의 위치를 교환하는데 사용.



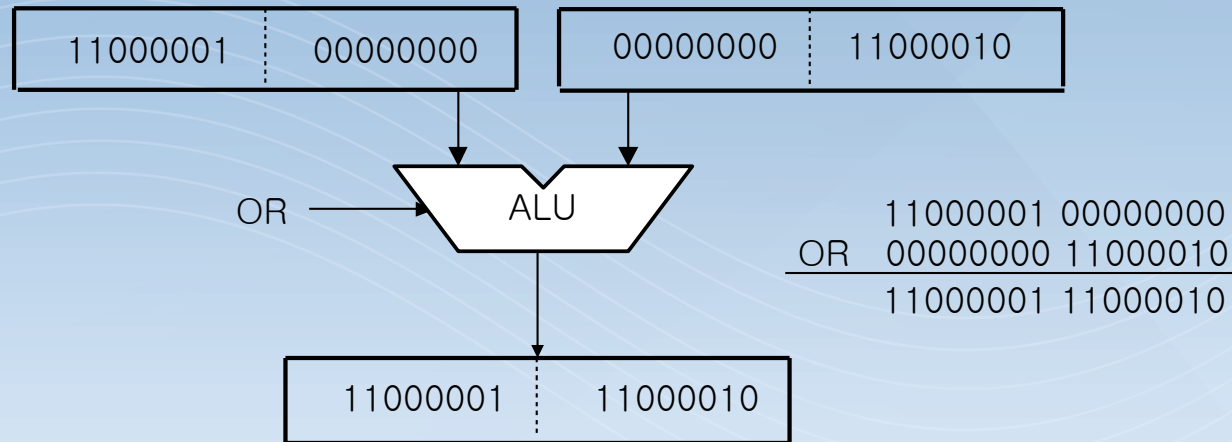
논리곱 연산

- 논리곱 (logical and) 연산
 - 2개의 데이터를 입력 받아 비트 간의 논리곱을 연산
 - 지우고 싶은 특정한 비트나 문자를 삭제하여 0으로 만들 때 사용함



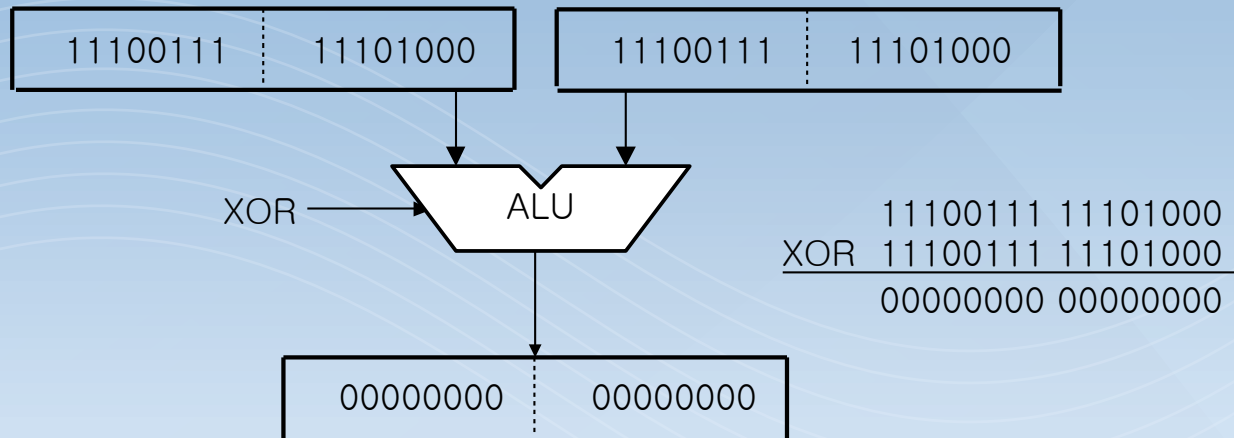
논리합 연산

- 논리합 (logical or) 연산
 - 2개의 데이터를 입력 받아 비트 간의 논리합을 연산
 - 필요한 비트나 문자를 삽입할 때 사용.



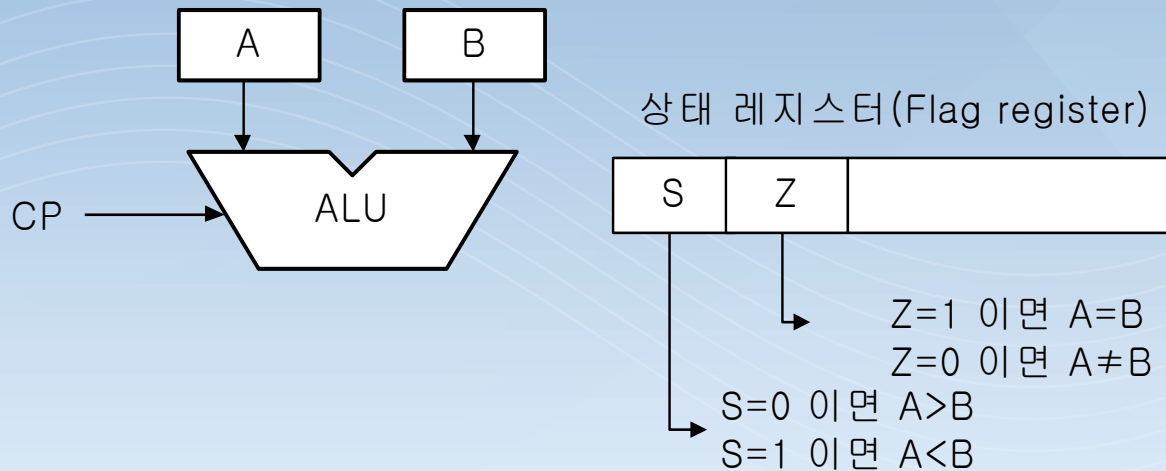
배타적 논리합 연산

- 배타적 논리 합(logical exclusive or) 연산
 - 2개의 데이터를 입력 받아 배타적 논리합을 연산



논리 비교 연산

- 논리 비교(logical compare) 연산
 - 2개의 입력이 비교되어서 출력에 나타나는 연산
 - 크고(>), 작음(<)과 같음(=)을 판단하여 flag 레지스터에 결과를 기억시킴.



컴퓨터의 문자 표현

2진수 문자 처리

문자의 표현 방식

- **문자도 2진수로 표현**
 - 문자(character)조차 2진수로 표현되어야 컴퓨터가 읽고 쓸 수 있다.
 - n 비트를 이용해 서로 다른 2^n 개의 0, 1 조합 가능
- **문자 집합의 갯수는 유한**
 - 숫자 : 10개의 문자로 구성된다.
 - 영문자 : 52개의 문자로 구성된다.
 - 한글(완성형 KSC-5601) : 2350자.
- **각각 문자에 서로 다른 2진수 코드 혹은 부호를 지정**
 - " 문자 : 0010 0010
 - 1 문자 : 0011 0001
 - b 문자 : 0110 0010
- **표준 코드**
 - 혼란을 없애기 위해 표준 문자 코드 체계를 제정.
 - ASCII, EBCDIC, KS 5601, Unicode 등이 있음.

문자 코드 체계

- **아스키코드(ASCII)**
 - American Standard Code for Information Interchange의 약어
 - 7 bit 크기의 코드이며, 영문자, 숫자, 특수문자 표현 가능
 - 본래 네트워크 통신을 위한 표준이었으나, 컴퓨터에 적용
- **EBDIC**
 - IBM에서 제정하여 사용, 8비트 크기의 코드
- **유니코드(Unicode)**
 - 16비트 코드이며, 65,536 개의 문자 표현 가능
 - 사실상 문헌 상에 존재하는 모든 문자 집합 표현 가능
- **KS 5601**
 - 완성형한글(2350자), 한자(4888자), 숫자, 특수문자 등
 - 16비트 코드이며, 유니코드 이전에 사용되던 한국 표준 문자 코드.

ASCII 코드 표

• ASCII 의 한계

- 본래 '미국(America)' 내에서 사용하기 위해 만든 표준이므로 알파벳과 숫자, 일부 문장 부호만 표현할 수 있음.
- 영문자를 표현하기 위해서는 7비트면 충분하기 때문에 나머지 128개는 그래픽 문자를 표현하기 위해 사용된다.

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	 	Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	\$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	((72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051))	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[[123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174	|	
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135]]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177		DEL

Source: www.asciitable.com

2 byte 완성형 코드

- 한글 표현을 위한 코드 크기

- 영어권의 문자를 표현하기 위해서는 최소 7비트만으로 충분하다.
- 한글 문자를 표현하기 위해서는 256 글자 이상을 표현할 공간이 필요하여 2바이트를 사용 한다.

- 2 바이트 완성형 코드

- 한글 문자에 기반을 두고 코드를 부여
- 자주 사용되는 2,350자를 추출하여 이들 글자 하나 하나를 완성된 글자로 보고, 가나다 순으로 배열

B 0	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
A0	가 각 간 감 갈 값 값 감 갑 값 갓 갓 강 갓 갓
B0	갈 값 값 개 객 갠 겔 겜 겹 겹 겹 갱 가 각 간 갈
C0	갓 강 개 갠 겔 거 격 건 겹 겹 겹 겹 겹 겹 겹 겹 겹
D0	겹 겹 겹 겹 게 겹 겹 겹 겹 겹 겹 겹 겹 겹 겹 겹 겹 겹
E0	겹 겹 겹 겹 겹 겹 겹 겹 겹 겹 겹 겹 겹 겹 겹 겹 겹 겹
F0	곶 곶 곶 곶 곶 곶 곶 곶 곶 곶 곶 곶 곶 곶 곶 곶 곶

유니코드

- 다양한 문자를 표현하기 위한 코드 필요성
 - 문자를 표현하기 위해 영어는 7비트, 비영어 (알파벳 기반 문자)는 8비트, 한글, 한자, 일본어는 16비트 이상이 필요한데, 유니코드는 이들 문자를 모두 표현할 수 있도록 만들어졌다.
 - 유니코드 체계에 따르면 한글은 기존처럼 한 음절 당 2바이트로 표현되고, 자소 분리가 용이하며 최대 11,172자까지 표현할 수 있다.
- 유니코드의 인코딩(encoding) 방식
 - 유니코드 자체는 자리수에 제약이 없는 코드 부여 체계이고, 그 부여된 코드, 즉 코드 포인트를 어떤 형식으로 표현하느냐에 따라 UTF-8, UTF-16, UTF-32등의 인코딩 규약으로 문자열을 표현한다.
 - UTF(UCS Transformation Format)-8과 UTF-16이 있다. UTF-8은 유니코드의 각 문자를 표현할 때 1바이트에서 3바이트까지로 가변적으로 표현한다.
 - 유니코드 값 0000 - 007F까지는 1 바이트를 사용하여 표현하고, 그 다음부터 07FF까지는 2 바이트, 그 다음부터 FFFF까지는 3바이트를 사용한다. 예를 들어 '가'라는 글자는 UTF-8로 인코딩하면 0xEAB080으로 3 바이트로 표현된다.
 - 이에 반해 UTF-16은 일반 유니코드 즉 Unicode 2.0/ ISO-10646 UCS-2와 같은 것으로 모든 문자를 2 바이트로 표현한다.
 - UTF-16이라고 해도 모든 국가의 모든 문자가 2바이트로 표현될 수는 없기에, UTF-16인코딩에서도 4바이트로 인코딩 되는 문자가 있다.

부연 설명

- 진공관은 '반도체'인가?
 - 진공관 자체는 반도체와 아무런 관련이 없고, 가열된 전극의 전자 방출을 통한 정류/증폭용 소자이다. (2극 진공관은 정류, 3극 진공관은 증폭) 이와 동등한 기능을 에너지가 많이 드는 가열이나 생산/관리가 힘든 유리 진공관을 대체할 방법을 찾다가 반도체를 이용할 수 있다는 걸 발견/발명 한 것이다. 진공관 자체에는 반도체가 들어가지 않는다.