3.1 하드웨어 기본소자 발전

기계식 컴퓨터 시대 (전자식 컴퓨터가 만들어지기 전)

2. relay 릴레이 / 1950년대 이전
   전자석으로 만들어짐

3. vaccum tube 진공관 / 1950년대
   진공 유리관 내부의 전자 운동을 이용함. 많은 열을 발생시켜 컴퓨터의 크기가 커졌다
   → ENIAC, IBM도 다음과 같음. 자기드럼과 천공 카드 함께 사용

4. transistor 트랜지스터 / 1950년대 말
   진공관보다 전력 소비가 작고, 발열이 적음. 자기 디스크 사용

5. IC (Integrated circuit) 직접회로 / 1960년대
   많은 개수의 트랜지스터를 하나의 칩에 집적한 것

6. VLSI (Very Large Scale Integration)
   초직접 반도체 회로 / 1970년대
   수천 개의 트랜지스터가 하나의 칩으로 직접됨

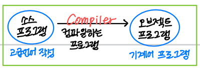
7. ULSC / SOC / 초병렬 컴퓨터 / 2000년대
   초대규모 직접회로 / 단일 칩 시스템

3.2 논리 회로 기초

1. 기계어 프로그래밍

기계어 2진수 → 고급 언어 프로그래밍

· 컴파일 과정



고급 프로그래밍 언어 → 기계어 : compile 과정

메인 메모리 → CPU 차례대로 수행 → adress가 감
기계어 밑에는 하드웨어가 있다

· 기계어 (Machine Language):
컴퓨터가 직접 이해하고 실행할 수 있는 가장 저수준의 프로그래밍 언어임
이 언어는 0과 1로 이루어진 이진수의 집합으로 구성되어 있음

· 어셈블리어 (Assembly Language):
기계어와 1:1로 대응되는 저수준 프로그래밍 언어. 사람이 읽고 쓰기 더 쉽게 만든 언어
기계어는 사람이 이해하기 어렵기 때문에, 어셈블리어는 이러한 기계어 명령에
대해 사람이 읽을 수 있는 간단한 표현을 제공함

2. 기본 게이트

게이트 : 컴퓨터에서 전기신호에 따라 가장 기본적인 연산을 수행하는 소자

디지털 회로에서 입력 신호를 받아 특정 규칙에 따라 출력 신호를 생성하는 기본적인 논리소자
이진수의 참과 거짓 상태를 처리하는데 사용된다

종류	의미	회로도 심볼(논리도)	기호(논리식)	진리표
AND	논리곱		$A \cdot B$	입력 A, B / 출력 A AND B 0 0 0 0 1 0 1 0 0 1 1 1
OR	논리합		$A + B$	입력 A, B / 출력 A or B 0 0 0 0 1 1 1 0 1 1 1 1
NOT	논리부정		$\bar{A}$ or A'	입력 A / 출력 NOT A 1 0 0 1
XOR	베타적 논리합		$A \oplus B$	입력 A, B / 출력 A XOR B 0 0 0 0 1 1 1 0 1 1 1 0
NAND			$\overline{A \cdot B}$	입력 A, B / 출력 A NAND B 0 0 1 0 1 1 1 0 1 1 1 0
NOR			$\overline{A + B}$	입력 A, B / 출력 A NOR B 0 0 1 0 1 0 1 0 0 1 1 0

XOR에 자기자신대입 → 0이 된다
XOR: 둘 중 하나만 1이어야 한다
→ 배타적 논리합

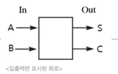
NAND: 둘 다 1이어야 0이 나온다
→ 1이 많음

NOR: 둘 중 하나라도 1이면 0이 나온다
→ 0이 많음

3.3 논리회로의 표현방식

논리도 (logic diagram)
논리식 (logic expression, boolean expression)
진리표 (truth table)
논리도와 논리식은 1대1 대응관계

반 가산기 (half-adder)



<입출력만 표시된 회로>

입력		출력	
A	B	Sum(S)	Carry(C)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

<1 bit Half-adder의 진리표>

불 대수 (Boolean Algebra)

연산자 우선 순위 : OR << AND << NOT

$a'b + a = a'b + a(1+b) = a'b + a + ab = b(1+a') + a = a + b$

$a'b + a = a'b + a + ab = a + b(a + a') = a + b$

불 대수 규칙

- Commutative (교환 법칙)
  - $a + b = b + a$
  - $a \times b = b \times a$
- Associative(결합 법칙)
  - $(a + b) + c = a + (b + c)$
  - $(a \times b) \times c = a \times (b \times c)$
- Distributive(분배 법칙) 반대로 묶는게 어렵다
  - $a + (b \times c) = (a + b) \times (a + c)$
  - $a \times (b + c) = (a \times b) + (a \times c)$
- Identity(항등 법칙)
  - $a + 0 = a$
  - $a + 1 = 1$
  - $a \times 0 = 0$
  - $a \times 1 = a$
- Complement(보수 법칙)
  - $a + a' = 1$
  - $a \times a' = 0$

- Idempotent(멱등 법칙)
  - $a + a = a$
  - $a \times a = a$
- Absorption(흡수 법칙) ☆증명
  - $a + (a \times b) = a$
  - $a \times (a + b) = a$
- De Morgan's law(드모르강의 법칙)
  - $(a \times b)' = a' + b'$
  - $(a + b)' = a' \times b'$
- Complement theorem(부정)
  - $(a')' = a$
  - $1' = 0$
  - $0' = 1$

· $a + (a \times b) = a$
$(a \times 1) + (a \times b)$ 항등법칙
$= a \times (1 + b)$ 분배법칙
$= a$ 멱등법칙

· $a \times (a + b) = a$
$(a \times a) + (a \times b)$ 분배법칙
$= a + (a \times b)$ 멱등법칙
$= a$

dual expression(짝 수식, 상대 수식)

불 대수에서는 dual expression(짝 수식, 상대 수식)이 존재한다. dual expression이란 원래의 식을 아래와 같이 변경하면 그에 대응하는 불 식(boolean expression)이 있다는 의미이다.

- +와 ×를 서로 맞바꾸고,
- 1과 0을 서로 맞바꾼다.

예를 들어서 아래의 식으로 이야기해보자.

$a + a' = 1$

위의 식을 변환 방법에 따라서 +와 ×을 서로 바꾸고, 1과 0을 서로 바꾸면, 아래와 같은 식이 된다. 이 두 개의 식을 서로 dual expression이라고 한다.

$a \times a' = 0$

이러한 것이 모든 불 식에 대해서 만족하는데, 이러한 식을 dual expression이라고 한다.

ex)

a	b	f
0	0	0
0	1	1
1	0	1
1	1	1

사실 이거도 흡수법칙임
$\rightarrow a'b + ab' + ab = a'b + a = a + b$
a=0 이면 b값에 따라서
a=1 이면 a+b

$a + (a \times b) = a$
$(a \times 1) + (a \times b)$
$= a \times (1+b) = a$

$a \times (a+b) = a$
$(a \times a) + (a \times b) =$
$= a + (a \times b)$

논리회로의 2가지 종류

① 조합 논리 회로 (combinational logic circuit)
입력값만으로 출력값이 결정되는 회로

② 순서 논리 회로 (sequential logic circuit)
입력값과 회로 내부의 상태가 출력값을 결정되는 회로, 메모리가 필요함

Q. 조합 논리 회로와 순서 논리 회로의 차이점은?
A. '회로 내부의 상태값'을 저장하는지 여부

SOP (곱의 합)' ≠ (합의 곱) POS
이거여서
And-or　　Or-and

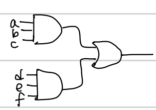
$bc + ab$　　$b(a+c)$
곱의 합 형태 → 합의 곱

$ab + ac = a(b+c)$

a	b	c
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

$A'B'C + A'BC' + AB'C' + ABC$
$= A'(B'C + BC') + A(B'C' + BC)$
$= A'(B \oplus C) + A(\overline{B \oplus C})$
$= A \oplus B \oplus C$

$(a' + b'c')$
$= (a(b+c))'$



↳ 이걸 NAND 게이트 논리식으로
$((abc)' \cdot (def)')'$
$= abc + def$