임베디드 기초 프로그래밍 (마이크로프로세서 실습) - 기본 학습

정보네트워크공학전공
이훈재

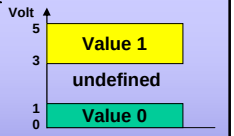
hjlee@dongseo.ac.kr
http://crypto.dongseo.ac.kr
http://kowon.dongseo.ac.kr/~hjlee

컴퓨터의 논리 회로

- 강의 내용
  - 디지탈 논리 회로(Digital Logic Circuit)
  - Boolean Algebr
  - 게이트(Gate)의 표시와 진위표
  - 조합 논리 회로(Combinational Logic)
  - 순서 논리 회로(Sequential Logic)
  - 집적 회로(Integrated Circuit)
  - 범용 대규모 집적 회로(LSI)
    - ROM - ROM, PROM, EPROM
    - PLA
    - 마이크로프로세서(Microprocessor)

디지탈 논리 회로

- 회로
  - 아날로그(Analog) 회로와 디지탈(Digital) 회로
- 디지탈 논리 회로(Digital Logic Circuit)
  - 디지탈 신호(불연속적 신호)로 나타낸 정보를 처리하는 회로
    - 예: 2진 디지탈 회로 - 정보는 0과 1 두 가지로 표현
  - 2 종류의 디지탈 회로
    - 조합 (Combinational) 논리 회로
      - 출력 값은 입력 값에 의하여만 결정됨
      - 예: 게이트(AND, OR, Inverter, NAND, NOR, XOR, 등)
    - 순서 (Sequential) 논리 회로
      - 출력 값은 회로의 상태(기억된 정보)와 입력 값에 의하여 결정됨
      - 플립플롭(Flip Flop: FF),



Boolean Algebra's Axiom

- 수학자 George Boole(1815~1864)은 논리를 두개의 값을 지닌 "binary notation"으로 나타냄
- 기본 논리 연산 : 논리 덧셈(+, OR), 곱셈(x, AND), 부정(', NOT)

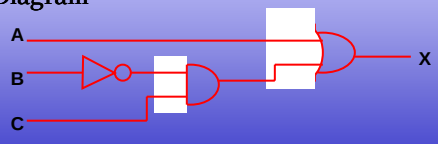
x + 0 = x	x + x = x
x . 0 = 0	x . x = x
x + 1 = 1	x + x' = 1
x . 1 = x	x . x' = 0
x + y = y + x(교환법칙)	x . y = y . x
x + (y +z) = (x+y) + z(결합법칙)	x . (y . z) = (x . y) . z
x . (y +z) = x . y + x . z(분배법칙)	x + y . z = (x + y) . (x + z)
(x + y)' = x' . y'(De Morgan 정리)	(x . y)' = x' . y'
(x')' = x	

논리 회로의 기능 표시방법

- Boolean Function : $f:\{0,1\}^n \rightarrow \{0,1\}$, *n is no of variables*
  - X = A + B'C, (B' = $\overline{B}$)
- Truth Table

A	B	C	X
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

- Logic Diagram

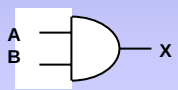


조합 논리 회로:

게이트 도형과 진위표

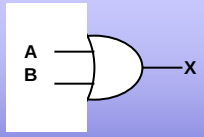
- Gate : 트랜지스터나 다이오드 등의 능동 소자를 포함하고 있는 회로

AND 게이트



A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

OR 게이트



A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

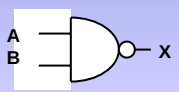
버퍼(Buffer)

A	X
0	0
1	1

조합 논리 회로:

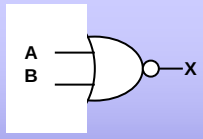
게이트 도형과 진위표

NAND 게이트



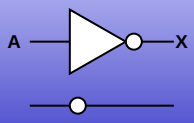
A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

NOR 게이트



A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

인버터(Inverter)



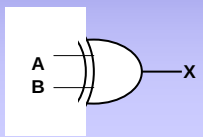
A	X
0	1
1	0

조합 논리 회로:

게이트 도형과 진위표

배타적 OR(Exclusive OR)

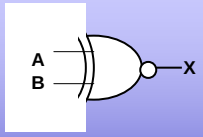
$X = A \oplus B = A.B' + A'.B$



A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

배타적 NOR(Exclusive NOR)

$X = A \odot B = A'.B' + A.B$



A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

조합 논리 회로:

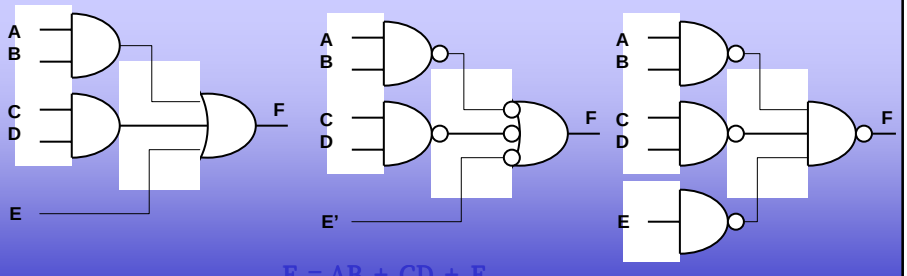
완전 집합

- 사용되는 게이트의 종류가 적을 수록 운영 및 유지 보수에 있어서 경제적이다.
- 게이트의 완전 집합(Complete Set)
  - 하나의 완전 집합에 속하는 게이트들만 가지고 임의의 논리 기능을 구현할 수 있음
  - {I, AND}, {I, OR}, {NAND}, {NOR}, {AND, XOR}, {OR, XOR}, …
  - {I, AND} => OR
    - A + B = (A')' + (B')' = (A' . B')'

조합 논리 회로:

완전 집합

- 완전 집합과 반도체 장치 기술과의 관계
  - 제 2 세대 컴퓨터 : AND, OR, NOT 등의 게이트를 이용
  - 제 3 세대 컴퓨터 : {NAND}, {NOR}의 완전 집합을 이용
- 1970년대 초에 개발된 bipolar semiconductor device 기술
  - TTL 회로 기능 : NAND 게이트
  - ECL 회로 기능 : NOR 게이트



F = AB + CD + E

조합 논리 회로

- 조합 논리 회로 : Gate와 결선 회로만으로 만든 회로
  - 반 가산기(Half Adder)
  - 전 가산기(Full Adder)
  - 연산기(ALU)
  - 부호 해독기(Decoder)
  - 멀티플렉서(Multiplexer)
  - 디멀티플렉서(Demultiplexer)

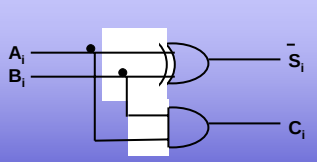
반 가산기(Half Adder)

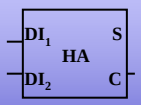
- 2 비트를 합하여 그 결과로 1 비트의 합과 올림수(carry)를 발생하는 회로

$S = A' \cdot B + A \cdot B' = A \oplus B$

$C = A \cdot B$

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



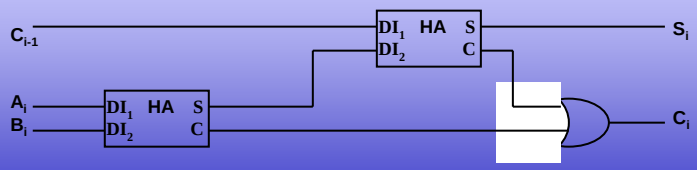


전 가산기(Full Adder)

$S_i = (A_i \oplus B_i) \oplus c_{i-1}$

$C_i = (A_i \oplus B_i) \cdot c_{i-1} + A_i \cdot B_i$

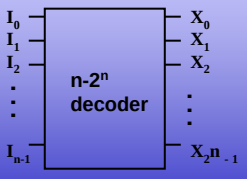
A_i	B_i	c_{i-1}	S_i	C_i
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

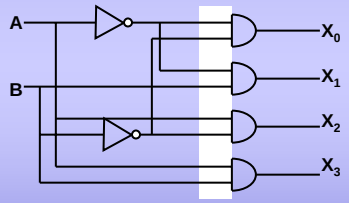


부호 해독기(Decoder)

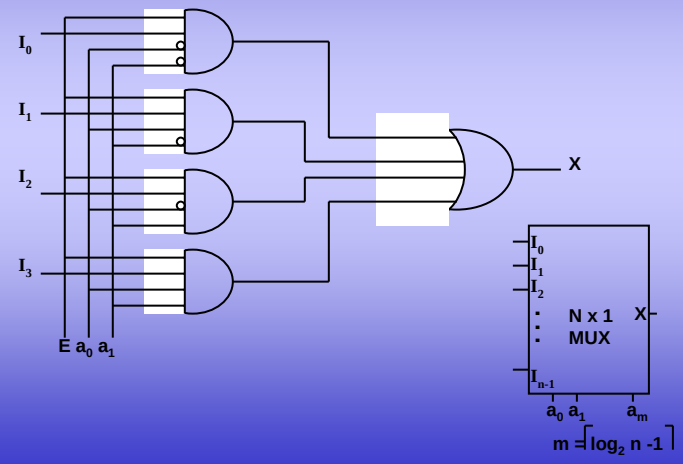
- n 가지의 정보를 부호화하여 2 진수 코드로 나타내려면 $\lceil \log_2 n \rceil$ 비트가 필요

A	B	X_0	X_1	X_2	X_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

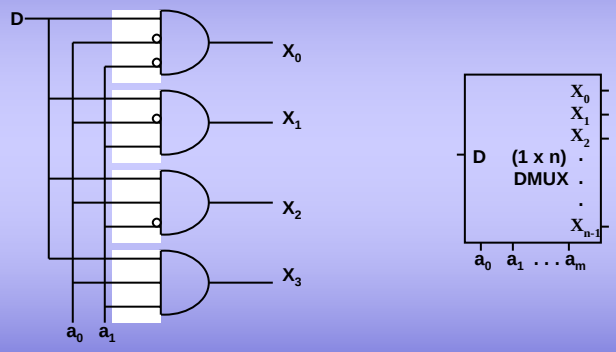




멀티플렉서(Multiplexer)

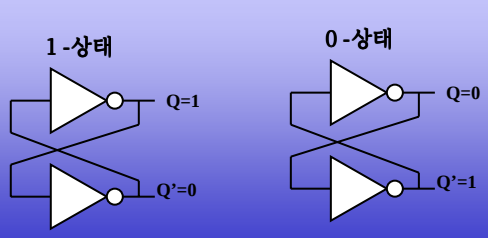


디멀티플렉서(Demultiplexer)

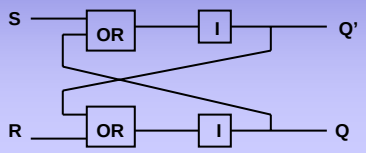


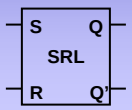
순서 논리 회로 요소

- 플립플롭(Flip Flop: FF)과 래치(latch)
  - 1 비트 기억 장소로 2 가지의 안정된 상태(회로의 외부로 부터 입력을 가하지 않는한 본래의 상태를 유지)를 가진 장치
  - 플립플롭 : 펄스의 전이에 민감
  - 래치 : 펄스의 지속 시간에 민감
- 플립플롭의 2 가지 안정된 상태



SR-래치(SR-Latch)





특성표(characteristic table)

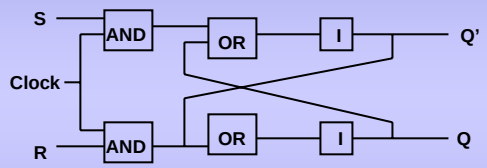
S	R	Q	Comment
0	0	Q_0	불변
0	1	0	리세트(reset)
1	0	1	세트(set)
1	1	-	금지(Q=Q'=0)

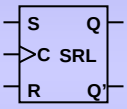
비동기적(Asynchronous) 동작
- 래치의 동작 싯점이 정해진 것이 아니라 입력 변화에 반응
- 많은 수의 FF이 있는 회로의 경우에는 신호 전달의 지연시간이 일정하지 않으므로 어느 FF의 상태가 언제 변하는지 알 수 없기 때문에 대단히 위험한 동작임

☞ 동기적(Synchronous) 동작이 안전

동기(Synchronous) SR-래치

동기 SR-래치(Synchronous SR-Latch)



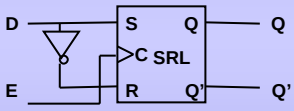


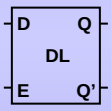
특성표

S	R	C	Q	Q'	Comment
d	d	0	Q_0	Q_0'	불변
0	0	1	Q_0	Q_0'	불변
0	1	1	0	1	리세트(reset)
1	0	1	1	0	세트(set)
1	1	1	-	-	금지

D-래치(Data-Latch)

동기 SR-래치와 같으나, 입력 단자에 입력 신호가 S=R=1이 되지 못하게 하는 외부 회로가 있음





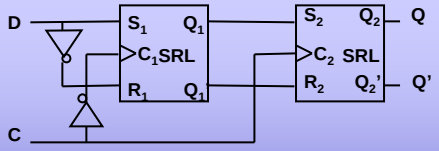
특성표

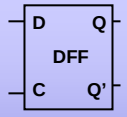
D	E	Q	Q'
d	0	Q_0	Q_0'
0	1	0	1
1	1	1	0

D-플립플롭(D-Flip Flop)

D-플립플롭은 클록 펄스의 유효한 펄스 전환이 일어나기 (클럭 펄스가 1이 되기) 직전에 D 입력 단자에 가해진 입력 정보를 기억시킨다.

2 개의 동기 SR-래치를 이용하여 다음과 같이 합성할 수 있다



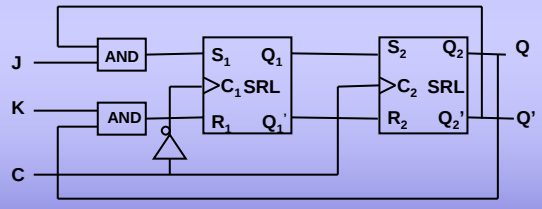


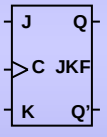
특성표

D	C	Q	Q'
d	0	Q_0	Q_0'
0	0->1	0	1
1	0->1	1	0

JK-플립플롭(JK-Flip Flop)

JK-플립플롭은 SR-래치의 입력 신호의 값이 S=R=1 인 것을 유효한 입력이 되게 한다. JK-플립플롭은 2 개의 동기 SR-래치를 사용하여 다음과 같이 합성할 수 있다



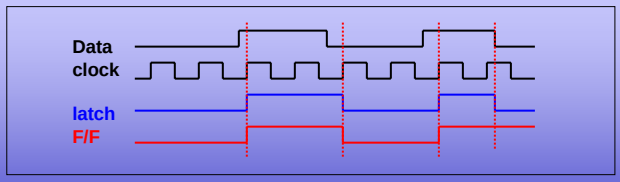


특성표

J	K	C	Q	Q'
0	0	d	Q_0	Q_0'
d	d	0	Q_0	Q_0'
0	1	0->1	0	1
1	0	0->1	1	0
1	1	0->1	Q_0'	Q_0

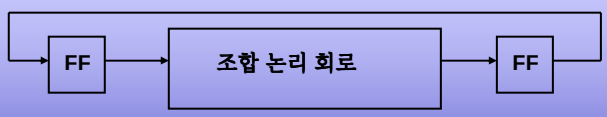
래치와 플립플롭 비교

- 래치
  - 클록 펄스의 유효 레벨에서 데이터 입력 단자에 가해지는 신호의 값을 클럭 펄스가 0으로 되기 직전에 기억시킴
- 플립플롭
  - 클록 펄스의 유효 전환(클럭 펄스가 1로 되기)직전 에 입력 단자에 가해지는 신호의 값을 기억시킴

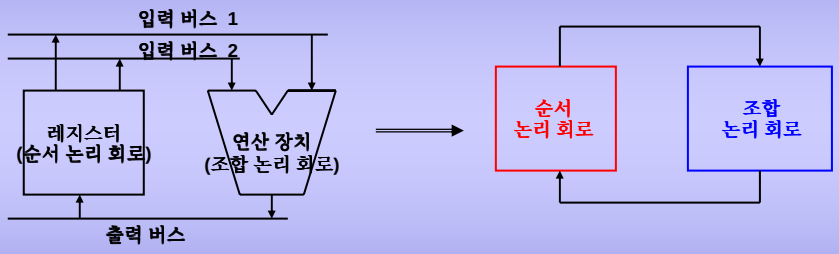


순서 논리회로

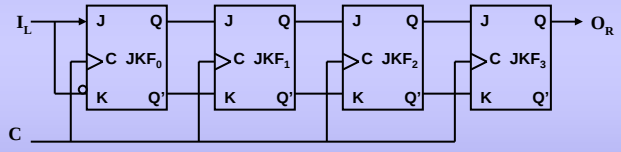
- 순서 논리 회로 - FF나 래치가 포함된 논리 회로
  - 레지스터(Register)
  - 쉬프트 레지스터(Shift Register)
  - 계수기(Counter)
  - CPU, RAM, ROM . . .



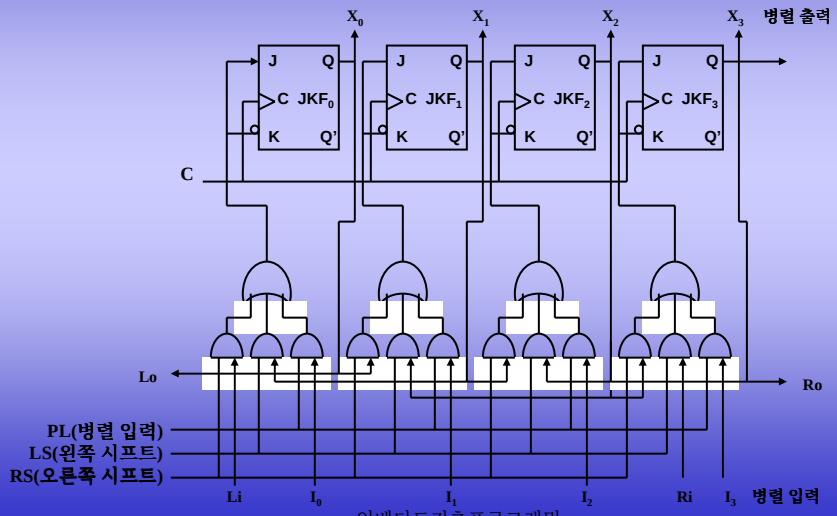
CPU


입력 버스 1
입력 버스 2
레지스터
(순서 논리 회로)
연산 장치
(조합 논리 회로)
출력 버스
순서
논리 회로
조합
논리 회로

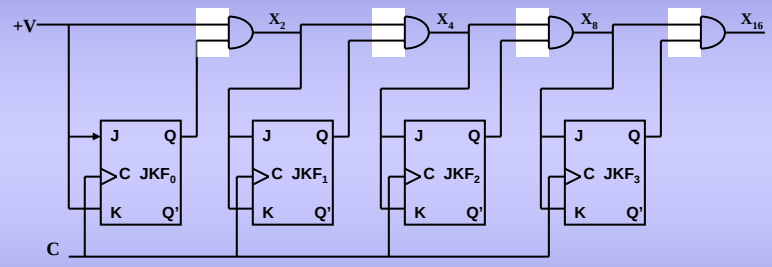
쉬프트 레지스터


I_L
J
Q
C
JKF_0
K
Q'
JKF_1
JKF_2
JKF_3
O_R
C

다기능 쉬프트 레지스터



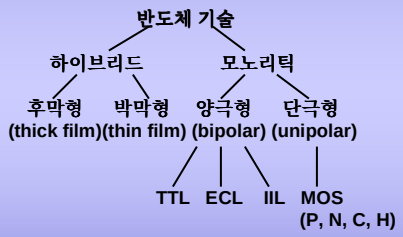
컴퓨터 논리 회로:
계수기(Counter)



모듈로 16 2진 계수기(Modulo -16 Binary Counter)

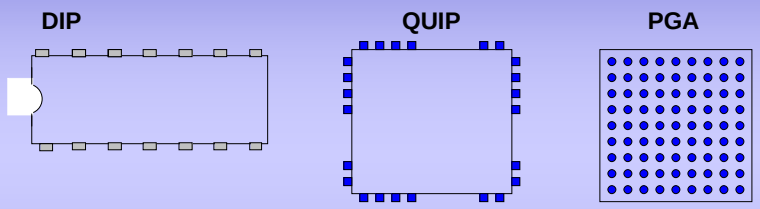
집적 회로(Integrated Circuit)

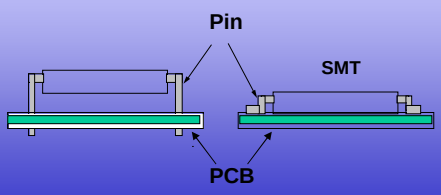
- 하이브리드(Hybrid) IC와 모노리틱(Monolithic) IC
- 특성
  - 크기가 작음
  - 속도가 빠름
  - 전력 소모가 적음
  - 수명이 길고, 신뢰도가 높음
  - 외부 회로와의 접속이 간결함
  - 경제성이 있음



- 집적도에 따른 분류
  - SSI, MSI, LSI, VLSI, WSI
- 포장 방법에 따른 분류
  - DIP(dual inline package), QUIP(quad inline package), PGA(pin grid array), SMT(surface mount technology) ...

집적 회로의 종류





범용 LSI

- 다용도 논리 회로 부품
  - ROM(Read Only Memory)
  - PLA(Programmable Logic Array)
  - 마이크로프로세서(Microprocessors)
- 범용 부품의 장점
  - 대규모 생산(mass production)으로 인한 경제성
  - 유지보수의 경제성
    - 유지해야 할 부품(spare parts)의 종류가 적음
    - 고장 수리가 편리

범용 LSI:

ROM

- 특성
  - 기억시킨 내용을 변경(write)시킬 수는 없으나, 빠르게 읽을 수 있음
  - 쓰기(Writing)
    - 마스크(mask)를 제작하여야 함
    - 마스크 제작을 위하여 논리 기능을 캐노니칼 곱의 덧셈 (sum of canonical product terms) 형식으로 나타내야 함

캐노니칼 곱(Canonical Products):
n 개의 변수로 나타낸 부울 함수의 경우, 모든 곱셈항을 n 개의 변수로 나타낸 것

예: 3 개의 변수 A, B, C로 나타낸 곱셈항

B ⟶ ABC + A'BC + ABC' + A'BC'
AB ⟶ ABC + ABC'

범용 LSI:

ROM에 의한 논리기능 구현

X1 = A'B' + AB'C → A'B'C + A'B'C' + AB'C
X2 = A'C' → A'BC' + A'B'C'
X3 = BC + AB'C' → ABC + A'BC + AB'C'
X4 = AB' + A'BC' → AB'C + AB'C' + A'BC'
X5 = AB → ABC + ABC'

주소	단어
ABC	X1 X2 X3 X4 X5
000	1 0 0 0 0
001	1 1 0 0 0
010	0 1 0 1 0
011	0 0 1 0 0
100	0 0 1 1 0
101	1 0 0 1 0
110	0 0 0 0 1
111	0 0 1 0 1

문젯점
변수가 하나 증가하면
ROM의 기억 용량을 두
배로 증가 시켜야 함

범용 LSI:

ROM의 종류

- ROM
  - 기억시킨 내용을 변경시킬 수 없음
  - 공장에서 마스크를 사용하여 제조과정의 마지막 부분에서 정보를 기억시킴
  - 정보를 기억시키지 않은 ROM은 가격이 싸지만 정보를 기억시키는 비용은 큼
  - 대량 생산에 적합함
- PROM
  - 기억시킨 내용을 변경시킬 수 없음
  - PROM 프로그래머라는 장치를 사용하여 정보를 기억시킴, 실험실 환경에서 가능
  - 정보를 기억시키지 않은 PROM은 ROM보다 비싸지만 정보를 기억시키는 비용은 ROM보다 저렴함
  - 적은 수의 시스템 생산에 적합함
- EPROM
  - 기억시킨 내용을 변경시킬 수 있음 - 자외선이나 전기적으로 기억된 내용을 지운 다음에 다시 사용할 수 있음)
  - PROM 프로그래머를 사용하여 정보를 기억시킴
  - 정보를 기억시키지 않은 EPROM은 PROM보다 비싸지만정보를 기억시키는 비용은 저렴함
  - 개발 단계에 사용하기 적합함

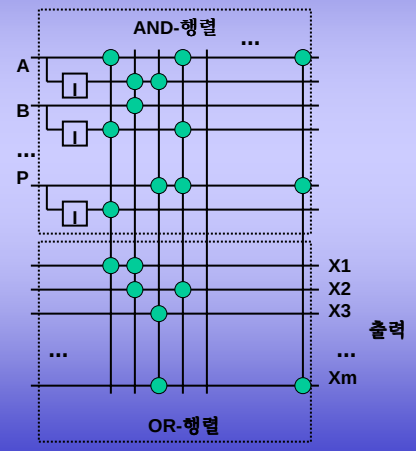
범용 LSI:

Programmable Logic Array

PLA의 구조

- AND-배열
  - PLA의 입력을 받음
  - 수직선은 AND 게이트를 나타냄
- OR-배열
  - PLA의 출력을 발생
  - 수평선은 OR 게이트를 나타냄

논리 기능의 구현;
- 논리 기능을 최소 곱셈항의 덧셈 (Minimal Sum of Products) 형식으로 나타낸 후에 구현



범용 LSI:

마이크로프로세서

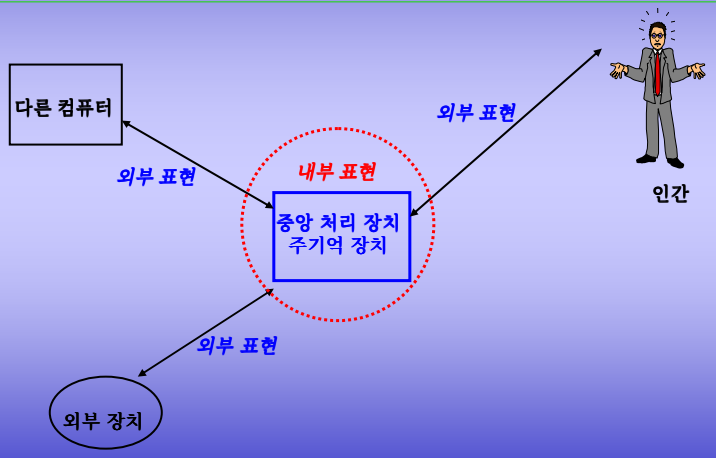
- 하나의 집적회로 칩에 구현한 중앙 처리 장치(CPU on a Chip)
- 마이크로프로세서는 프로그램을 통하여 다양한 기능을 구현시킬 수 있는 범용 LSI칩이다.
  - 8, 16, 32, 64-비트 마이크로프로세서가 있음
  - 다양한 보조 칩이 있음
  - 마이크로프로세서 중에는 마이크로프로그램이 가능한 것도 있음
- 현대의 고성능 마이크로프로세서에는 고성능 컴퓨터 구조 설계 기법이 포함됨
  - 캐쉬(Cache)
  - 가상 기억 체제(Virtual Storage System)
  - 파이프라인(Pipeline)
  - 복수 연산 장치(Multiple ALUs)

정보의 표현

강의 내용

- 컴퓨터 시스템에서 정보를 나타낼 때 고려 사항
- 정보의 내부 표현(Internal Representation)과 외부 표현(External Representation)
- 정보의 내부 표현
  - 수의 표현
    - 고정 소수점 수의 표현(Fixed Point Representation)
      - 부호와 절대치(Sign Plus Magnitude)
      - 1의 보수(1's Complement)
      - 2의 보수(2's Complement)
    - 부동 소수점 수의 표현(Floating Point Representation)
  - 비수치 정보의 표현(Representation of Nonnumeric Information)
- 정보의 외부 표현(External Representation)
  - ASCII, EBCDIC, 6-bit BCD

정보의 내부 및 외부 표현



내부 표현과 외부 표현

- 외부 표현
  - ASCII, EBCDIC, 6-bit BCD
- 내부 표현
  - 비수치 데이터
    - 데이터의 변경이 발생하지 않는 처리가 행해짐 -워드프로세싱
    - 같은 내부 표현과 외부 표현을 사용
      - 정보 표현의 변환이 불필요
  - 수치 데이터
    - 과학기술 응용
      - 수의 범위가 대단히 큼 -높은 정밀도를 위하여 별도의 내부 표현을 사용
      - 신속한 계산을 위하여 외부 표현과 다른 내부 표현을 사용
      - 대부분의 계산은 중앙 처리 장치 내에서 행해짐(CPU bound) - 희소한 입출력을 위하여 코드 변환을 감수
    - 업무 전산화 응용
      - 대량의 입출력이 필요 - 입출력을 위하여 부호 변환이 필요하면 비효율적
      - 복잡한 계산은 거의 불필요 - 컴퓨터 내부에서 외부 표현을 그대로 사용

데이터 구조의 표현

- 데이터들 사이의 관계
  - 앞뒤 관계(Predecessor and Successor)
  - 상하 관계(Superior, Peer, Subordinate)
  - 종속 관계(Membership), 등
- 데이터들의 관계를 기억 장치의 1 차원 기억 공간에 기억시켜야 함
  - 리스트(List)
  - 스트링(String)
  - n -차원 배열(Array)
  - 트리(Tree)
  - 큐(FIFO)
  - 스택(LIFO)
  - 기타

수치 정보의 표현

10진수(인간) ⟷ 부호(기계)

- 수치 정보의 컴퓨터 내부에서의 표현 시의 고려 사항
  - 기억 공간을 적게 차지 (부호의 경우 부호의 비트수)
  - CPU 내에서 정보 처리의 효율성 및 정보 교환의 용이성
  - 10진수와 상호 변환이 용이
  - 한정된 비트 수에 대한 정보의 정밀도

수의 내부 표현 - 수의 체계

로마 수(Roman Number)
- 비 자릿(Non -positional) 수 체계

+	I	II	III	IIII	V	...
I	II	III	IIII	V	VI	...
II	III	IIII	V	VI	VII	...
III	IIII	V	VI	VII	VIII	...
IIII	V	VI	VII	VIII	VIIII	...
V	VI	VII	VIII	VIIII	X	...
VI	VII	VIII	VIIII	X	XI	...
...	...	...	...	...	...	...

무한한 크기의 덧셈표
- 비효율적 덧셈
- 덧셈기의 구현이 불가능

10 진수(Decimal Number)
- 자릿(Positional) 수 체계

+	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	10
2	2	3	4	5	6	7	8	9	10	11
3	3	4	5	6	7	8	9	10	11	12
4	4	5	6	7	8	9	10	11	12	13
5	5	6	7	8	9	10	11	12	13	14
6	6	7	8	9	10	11	12	13	14	15
7	7	8	9	10	11	12	13	14	15	16
8	8	9	10	11	12	13	14	15	16	17
9	9	10	11	12	13	14	15	16	17	18

유한한 크기의 덧셈표
- 10진수와의 변환이 용이
- 덧셈기의 구현 가능
- 덧셈표가 작으면 덧셈기를 저렴하게 구현 가능

수의 내부 표현 - 베이스 R

MSB LSB

수의 표현: $A_R = a_{n-1} \ldots a_1 a_0 . a_{-1} \ldots a_{-m}$

$0 < a_i < R$이며, R 은 기저(base 혹은 radix)라 부름

값 : $V(A_R) = \sum_{i=-m}^{n-1} a_i . R^i$

- 진법(number system)
  - 16 진법
  - 10 진법
  - 8 진법
  - 2 진법
    - 고정 소수점 수(fixed point number)
    - 부동 소수점 수(floating point number)

고정 소수점 수의 표현

- 소수점의 위치를 일정한 곳에 고정시킨 표현 방법
- 범람(overflow)
  주어진 비트수를 가지고 정확하게 나타내기에는 너무 큰 수(overflow) 이거나 너무 작은 수(underflow)임을 나타냄.
- A의 보수
  - (R-1)의 보수((R-1)'s Complement)
    - 수의 모든 자리에서 (R-1)에서 그 자리에 있는 기호를 뺀 수
    - $R^n - R^{-m} - |A|$의 2진수 표현
  - R의 보수(R's Complement)
    - (R-1)의 보수에 1을 더한 수
    - A=0이면 0이고, A=1이면 $(R^n - |A|)$의 2진수 표현

2 진수 표현

$B_2 = b_n\ b_{n-1} \dots b_1\ b_0 \cdot b_{-1} \dots b_{-m}$

b_n: 부호 비트

$b_{n-1} \dots b_1\ b_0 \cdot b_{-1} \dots b_{-m}$: 절대치 비트

- 부호와 절대치(Sign Plus Magnitude) 표현
- (부호와) 1의 보수((Sign plus) 1's Complement) 표현
- (부호와) 2의 보수((Sign plus) 2's Complement) 표현

부호와 절대치

$B = b_n\ b_{n-1} \ldots b_1\ b_0 . b_{-1} \ldots b_{-m}$

b_n: 부호 비트

$b_{n-1} \ldots b_1\ b_0 . b_{-1} \ldots b_{-m}$: 절대치 비트

$$V_S(B) = (-1)^{b_n} \sum_{i=-m}^{n-1} b_i \cdot 2^i$$

음수로의 변환이 용이

표현의 범위;

최대: 011 … 11 . 11 … 1 ☛ $(2^n - 2^{-m})$

최소: 111 … 11 . 11 … 1 ☛ $-(2^n - 2^{-m})$

0의 2가지 표현;

+0: 000 … 00 . 00 … 0

- 0: 100 … 00 . 00 … 0

1의 보수

$B = b_n\ b_{n-1} \ldots b_1\ b_0 . b_{-1} \ldots b_{-m}$

b_n: 부호 비트

$b_{n-1} \ldots b_1\ b_0 . b_{-1} \ldots b_{-m}$: 크기 비트

$$V_1(B) = b_n(2^{-m} - 2^n) + \sum_{i=-m}^{n-1} b_i \cdot 2^i$$

- 표현 범위;

  최대: 011 … 11 . 11 … 1 ☛ $(2^n - 2^{-m})$

  최소: 100 … 00 . 00 … 0 ☛ $-(2^n - 2^{-m})$

- 0의 2가지 표현;

  +0: 000 … 00 . 00 … 0

  -0: 111 … 11 . 11 … 1

2의 보수

$B = b_n\ b_{n-1} \ldots b_1\ b_0 . b_{-1} \ldots b_{-m}$

b_n: 부호 비트

$b_{n-1} \ldots b_1\ b_0 . b_{-1} \ldots b_{-m}$: 크기 비트

$$V_2(B) = -b_n 2^n + \sum_{i=-m}^{n-1} b_i . 2^i$$

- 표현의 범위;
  - 최대: 011 … 11 . 11 … 1 $(2^n - 2^{-m})$
  - 최소: 100 … 00 . 00 … 0 -2^n
- 0의 표현;
  - 000 … 00 . 00 … 0

비교

- 음수 변환의 용이성
  - 부호+절대치 > 1의 보수 > 2의 보수
- 연산 장치 하드웨어
  - 부호+절대치: 덧셈기와 뺄셈기가 모두 필요
  - 1의 보수, 2의 보수: 덧셈기만 필요
- 연산 속도
  - 2의 보수 > 1의 보수(올림수 되돌려 더하기(end-around carry))
- 0의 인식
  - 2의 보수가 가장 빠름

부동 소수점 수의 표현

고정 소수점 수의 표현은 표현할 수 있는 수의 범위가 작은 것이 문제임

19,850,000,000,000
.000,000,000,034,82

대단히 큰 수와 대단히 작은 수의 다양한 표현;
1,985 x 10^{10}
0.1985 x 10^{14}, 1.985 x 10^{13}, 19.85 x 10^{12}, 198.5 x 10^{11}, ...
0.3482 x 10^{-10}
3.482 x 10^{-11}, 34.82 x 10^{-12}, 348.2 x 10^{-13}, 3,482 x 10^{-14}, ...

부동 소수점 수 : F
F = AB

b_n	$a_k\ a_{k-1} \ldots a_0$	$b_{n-1} \ldots b_1\ b_0 . b_{-1} \ldots b_{-m}$

부호 지수(exponent) 가수(mantissa)

$V(F) = V(B) \times 2^{V(A)}$

부동 소수점 수:

정규화(Normalization)

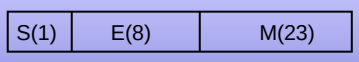
- 부동 소수점 수의 표현에서는 한 수에 대하여 여러 개의 표현들이 존재
  - 예) 0.1101 x 2^5, 110.1 x 2^2, 0.01101 x 2^6, . . .
- 혼란을 막기 위하여 한 가지 표준 형식을 정하여 사용
- 정규화된 표현(normalized representation)
  - 소수점의 바로 오른 편에 있는 비트가 항상 1이 되도록 위치를 조정
  - ±0.1bbbb….b x 2^E , where b ∈ {0, 1}
  - 정규화된 표현에서 소수점의 바로 오른 편에 있는 비트가 항상 1이므로, 이 비트를 반드시 저장할 필요는 없슴

부동 소수점 수 : 0의 표현

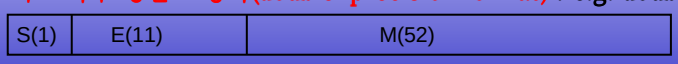
- 0×2^E : 지수(E)는 어떠한 수가 되어도 상관 없이 0
  - 프로그램 실행 과정에서 빈번히 발생하는 zero -test 에서 가수 및 지수의 모든 비트들이 0이 아닌 경우에, 검사 과정이 복잡
    - 예) : 0+F, 0/F ...
  - 가장 합리적인 표현 방법은 하드웨어적으로 쉽게 인식할 수 있도록 가수 및 지수의 모든 비트가 0이 되도록 함
- 바이어스된 지수(biased exponent)
  - E의 값이 아주 큰 음수이면 2^E 의 절대값이 0에 가까움
  - 지수의 값에 일정한 값의 바이어스(Bias)를 더하여 지수 부분의 모든 비트가 0이 되도록 함
    - 예) : 바이어스 127(excess -127 코드) . . .

IEEE 754:표준 부동 소수점 형식

- IEEE 754 : 컴퓨터 회사 및 기종 마다 다른 부동 소수점의 표준화된 표현 방법
  - 가수 :
    - 부호 비트와 함께 부호화 크기로 표현
    - 정규화된 표현을 사용하며, 첫번째 1은 포함 시키지 않음(hidden bit)
  - 지수 :
    - 바이어스된 2 진수로 표현(excess -127 코드)
- 32 비트 단일 정밀도 형식(single -precision format) : e.g. float



- 64 비트 복수 정밀도 형식(double -precision format) : e.g. double



IEEE 754:표준 부동 소수점 형식

- 32 비트 단일 정밀도 형식(single -precision format)
  - $N = (-1)^S 2^{E-127}(1.M)$
  - 예) $N = -13.625_{10}$ 에 대한 IEEE 754 표현
    - $13.625_{10} = 1101.101_2 = 1.101101 \times 2^3$
    - S = 1 (-1)
    - E = 00000011 + 01111111 = 1000010 (바이어스 127)
    - M = 1011010000 … 0 (소수점 좌측의 1은 제외)
- 64 비트 복수 정밀도 형식(double -precision format)
  - $N = (-1)^S 2^{E-1023}(1.M)$

10 진수의 내부 표현

- 10 진수의 계산
  - 외부 표현이 10 진수라면 컴퓨터 내부에서 다른 표현으로 변환이 불필요
- 업무 전산화 등에서 사용
- 1 자리 → $\lceil \log_2 10 \rceil = 4$ → 4 -비트 BCD(Binary Coded Decimal)
  - 4비트로는 16가지 코드를 나타낼 수 있으므로 6 개의 사용되지 않는 코드가 있음(unused code space)
  - 계산 속도가 느림

10진수의 표현:

BCD

BCD(Binary Coded Decimal)

- 4-비트 코드로 각 자리는 8, 4, 2, 1의 가중치를 가짐 - 8421 코드
- 음수 표현을 위하여 9의 보수로 변환이 어려움
- 2421 코드, 84-2-1 코드, 3 초과 코드 등은 이를 개선

10진수	BCD	2421	84-2-1	excess 3
0	0000	0000	0000	0011
1	0001	0001	0111	0100
2	0010	0010	0110	0101
3	0011	0011	0101	0110
4	0100	0100	0100	0111
5	0101	1011	1011	1000
6	0110	1100	1010	1001
7	0111	1101	1001	1010
8	1000	1110	1000	1011
9	1001	1111	1111	1100

Biquinary 코드

9의 보수 표현

비수치 정보의 표현

- 6-비트 BCD
  - 4-비트 BCD + 2 비트
  - 수와 비수치 기호를 모두 나타낼 수 있음
  - 영문자의 대문자와 소문자 구별을 할 수 없음
  - 단어의 크기가 6의 배수인 컴퓨터에 적합 - CDC 6000, Cybre 계열
- ASCII(America Standard Code for Information Interchange)
  - 128 개의 문자를 7비트로 나타냄
  - 8-비트 ASCII ←7-비트 ASCII + 1 패리티 비트
- EBCDIC(Extended BCD Interchange Code)
  - 8-비트 코드
  - 주로 IBM 기종에서 많이 사용