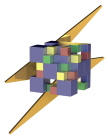
Lecture 02

이진 정수의 연산

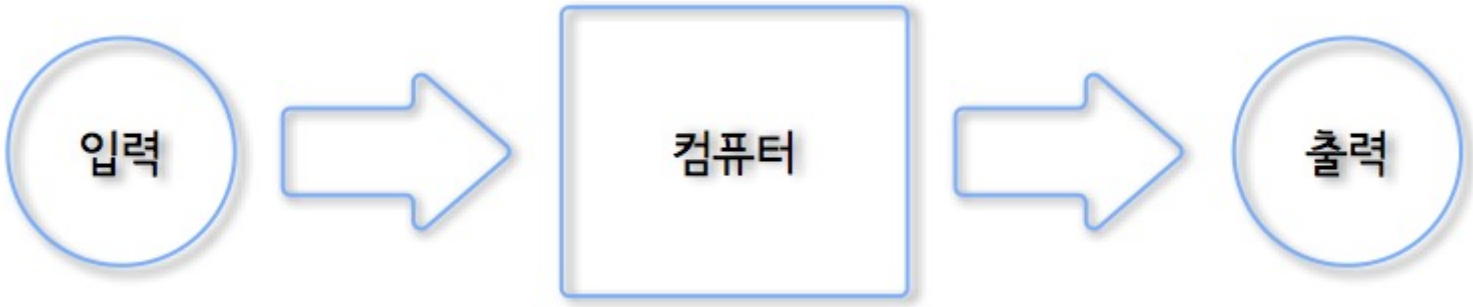
이재진
서울대학교 데이터사이언스대학원
서울대학교 공과대학 컴퓨터공학부
http://aces.snu.ac.kr/~jlee



THUNDER Research Group
Seoul National University
서울대학교 천둥 연구실

컴퓨터

- 입력으로 데이터를 받아들여 저장하고, 저장한 데이터를 내장된 프로그램(program)의 지시에 따라 처리하여 그 결과를 원하는 형식으로 출력하는 전자기기
- 블랙박스로 추상화
  - 내부의 구조 및 동작방식에 관계없이 입력과 출력, 입력과 출력의 관계, 그 기기가 제공하는 기능만을 관심의 대상으로 삼음



입력, 출력, 데이터

- 입력(input)
  - 컴퓨터에 사람이나 다른 컴퓨터, 또는 그 환경이 제공하는 정보
- 출력(output)
  - 컴퓨터가 생산한 결과물로서 텍스트, 오디오, 그래프, 그림, 이미지 등의 형식으로 표현됨
- 데이터
  - 사전적 정의
    - 추론, 논의, 계산의 기반이 되는 측정이나 통계에 의해 얻어진 사실적 정보
  - 컴퓨터공학에서 데이터의 정의
    - 컴퓨터로 처리할 수 있는 형태로 구성된 사실적 정보
    - 두 개의 상태(0과 1)로 표현되는 전기 신호로 컴퓨터에 저장됨
    - 디지털 정보

비트와 바이트

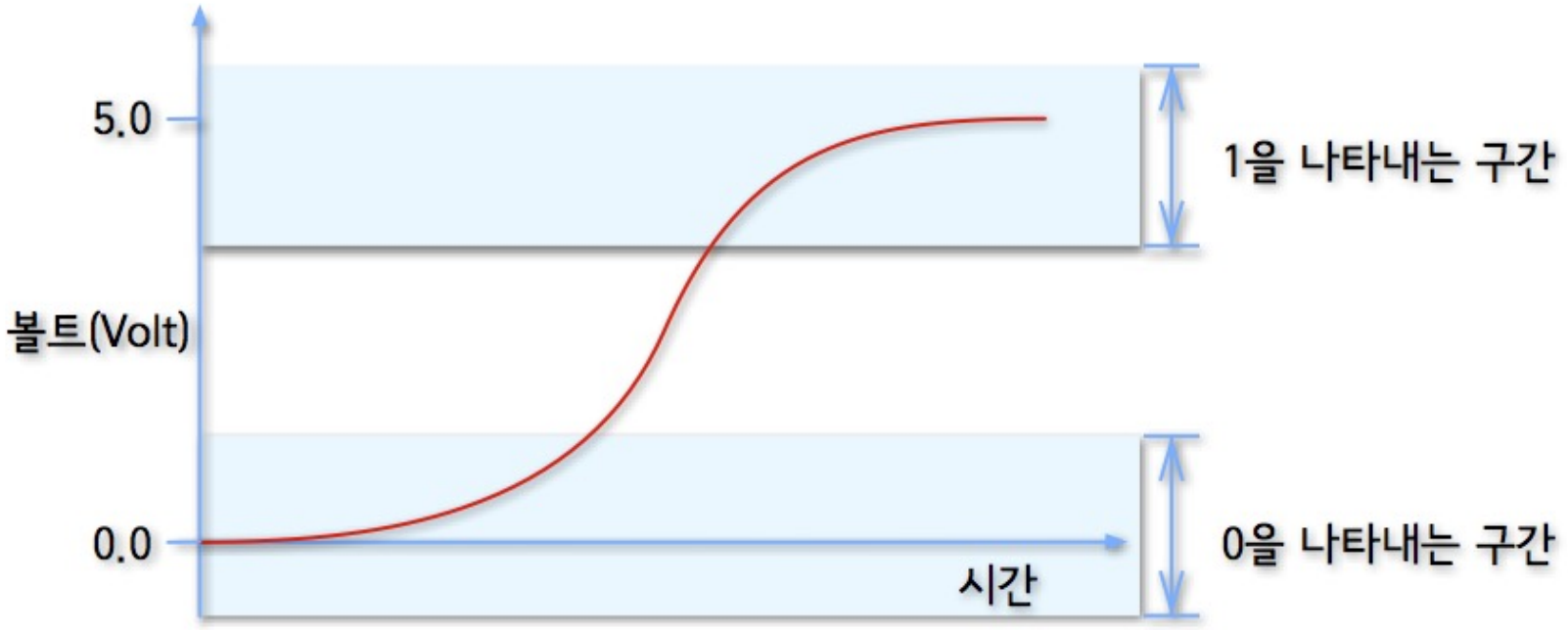
- 비트(bit)는 컴퓨터공학에서 가장 근본적인 정보의 표현 단위
  - Binary digit 의 준말
  - 두 개의 값, 0 과 1
  - 컴퓨터 내부에서 정보는 비트들의 패턴으로 표현됨

- 비트 패턴(pattern)
  - 비트로 구성된 열(sequence) 또는 스트링(string)
  - 비트 패턴의 의미는 해석에 따라 달라짐
  - 같은 비트 패턴이라도 수치, 기호, 이미지, 오디오 등으로 다르게 해석될 수 있음

- 바이트(byte) 정보 표현의 다른 단위
  - 8 개의 비트를 묶어 한 개의 바이트라고 함
  - 역사적으로 한 개의 알파벳 문자를 나타내는 비트 열을 바이트라고 함

0과 1을 나타내는 전기 신호

- 데이터는 전기신호의 형태로 저장되고 처리됨
  - 명확히 구분되는 두 가지의 서로 다른 상태 : 0과 1

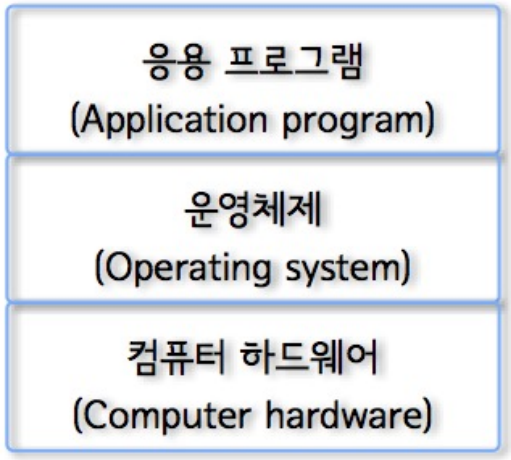


하드웨어와 소프트웨어

- 하드웨어(hardware)
  - 컴퓨터 시스템의 물리적 구성요소
- 소프트웨어(software)
  - 여러 개의 프로그램으로 구성된 집합
- 프로그램(program)
  - 주어진 입력을 가지고 원하는 출력을 얻기 위해 무엇을 해야 하는지 컴퓨터에게 지시하여 컴퓨터를 동작시키는 역할을 함

순차 컴퓨터 시스템

- 순차(順次) 컴퓨터 시스템(sequential computer system)은 우리가 흔히 알고 있는 컴퓨터
- 병렬(竝列) 컴퓨터 시스템(parallel computer system)과 비교하여 이야기할 때 사용하는 용어
- 추상화된 요소인 하드웨어, 운영체제, 응용 프로그램



응용 소프트웨어(application software)

- 줄여서 응용(application)으로 부르기도 함
- 사용자가 특별한 작업을 수행할 때 도움을 주는 응용 프로그램의 집합
  - Spreadsheet

시스템 소프트웨어

- 컴퓨터 하드웨어를 운용하고 응용 소프트웨어를 실행하기 위한 플랫폼(platform)
  - 운영체제, 커맨드라인 인터프리터(command-line interpreter), 윈도우 시스템(window system), 컴파일러(compiler) 및 디버거(debugger)

유틸리티 소프트웨어(utility software)

- 시스템 소프트웨어의 일종
- 컴퓨터 하드웨어와 소프트웨어를 관리하고 튜닝(tuning)할 때 이용
  - 바이러스 스캐너, 데이터 압축 유틸리티, 디스크 파티션 유틸리티, 알카이브 유틸리티(archive utility), 시스템 모니터, 텍스트 에디터, 어셈블러(assembler)

기수법(記數法)

- Number Systems

- 수를 시각적으로 표현하는 방법
  - 정해놓은 기본적인 심볼(숫자, digit)들의 조합을 사용하여 나타냄

- 정수, 소수 혹은 실수를 표현할 수 있음

- 실수 = 정수 부분 + 소수 부분
  - 소수점(radix point, '.')으로 정수부와 소수부를 구분
  - 소수부는 0 이상 1 미만의 실수
  - 예) 3.14159265
    - 정수부 : 3
    - 소수부 : 14159265

위치기반 기수법 (Positional Number System)

- 수는 숫자들을 나열한 스트링(string)을 통하여 표현
  - 각 숫자가 가지는 값은 숫자의 위치에 따른 무게값에 따라 결정됨

- 십진법 (Decimal Number System)
  - 사용하는 숫자 : $0, 1, 2, 3, 4, 5, 6, 7, 8, 9$
  - 십진수 $d_{p-1}d_{p-2}\cdots d_1\ d_0\ .\ d_{-1}\ d_{-2}\cdots d_{-n}$ 의 값은 아래와 같이 주어짐

$$\sum_{i=-n}^{p-1} d_i \cdot 10^i$$

  - 기수(base, radix) : 10
  - 가장 왼쪽의 숫자(d_{p-1}) : Most Significant Digit (MSD)
  - 가장 오른쪽의 숫자(d_{-n}) : Least Significant Digit (LSD)
  - 예) $754.82 = 7\times100 + 5\times10 + 4\times1 + 8\times0.1 + 2\times0.01$

기수가 r인 기수법

- 앞에서 다룬 10진법에서 기수를 10이 아닌 다른 수 r로 대치하면 r-진법 체계가 됨
  - r 개의 서로 다른 숫자들을 사용
  - i 번째 위치한 숫자의 무게값은 r^i
  - r 진수 $x_{p-1}x_{p-2}\cdots x_1\ x_0\ .\ x_{-1}\ x_{-2}\cdots x_{-n}$ 는 아래와 같은 값을 가짐

$$\sum_{i=-n}^{p-1} x_i \cdot r^i$$

  - $r \leq 10$ 일 때, 10진수에서 쓰는 숫자들 중 처음 r 개의 숫자들을 사용
  - $r > 10$ 일 때, 10진수에서 쓰는 숫자들과 처음 $r-10$ 개의 알파벳을 사용
    - 대소문자 구별 없이 사용 가능

고정소수점 표현과 부동소수점 표현

- 고정소수점 표현(fixed-point representation)
  - 소수점이 어떤 위치에 고정되어 있다고 가정
    - 정수일 경우 소수점은 LSD의 오른쪽에 비명시적으로 존재 $(3.,\ \ 145.)$
    - 소수일 경우 소수점은 MSD의 왼쪽에 비명시적으로 존재 $(.176,\ \ .997)$
  - 소수점을 컴퓨터 내부에서 표시할 필요가 없음
  - r-진법은 왼쪽 p 개의 숫자와 오른쪽 n 개의 숫자 사이에 소수점의 위치를 고정
    - $x_{p-1}x_{p-2}\cdots x_0\,.\,x_{-1}x_{-2}\cdots x_{-n}$
- 부동소수점(浮動小數點, floating-point numbers) 표현
  - 소수점의 위치가 유효숫자의 개수에 따라 상대적으로 어디든 위치할 수 있음

흔히 쓰는 기수법

이름	기수	숫자
이진법	2	$0, 1$
팔진법	8	$0, 1, 2, 3, 4, 5, 6, 7$
십진법	10	$0, 1, 2, 3, 4, 5, 6, 7, 8, 9$
십육진법	16	$0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F$

- 혼동을 피하기 위하여 나타낸 수의 오른쪽 끝에 붙은 첨자를 이용하여 사용한 기수법을 표현
  - 예) 101
    - 2진수 : 101_2
    - 8진수 : 101_8
    - 10진수 : 101_{10}
    - 16진수 : 101_{16}

이진법

- Boole 논리, 컴퓨터와 밀접한 연관이 있음
- 컴퓨터는 내부적으로 수치 데이터를 이진수로 나타냄
- 사용하는 숫자(binary digits, bits) : $0, 1$
- 이진수 $b_{p-1}b_{p-2}\cdots b_1 b_0 \,.\, b_{-1}\, b_{-2}\cdots b_{-n}$ 의 값은 아래와 같이 주어짐

$$\sum_{i=-n}^{p-1} b_i \cdot 2^i$$

- 기수 : 2
- 가장 왼쪽의 숫자(b_{p-1}) : Most Significant Bit (MSB)
- 가장 오른쪽의 숫자(b_{-n}) : Least Significant Bt (LSB)
- 예)
  - $10011 = 1\cdot 2^4 + 0\cdot 2^3 + 0\cdot 2^2 + 1\cdot 2^1 + 1\cdot 2^0 = 19$
  - $101.001 = 1\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0 + 0\cdot 2^{-1} + 0\cdot 2^{-2} + 1\cdot 2^{-3} = 5.125$

팔진법

- Boolean 논리, 컴퓨터와 밀접한 연관이 있음
- 사용하는 숫자(octal digits) : $0, 1, 2, 3, 4, 5, 6, 7$
- 팔진수 (octal number) $x_{p-1}x_{p-2}\cdots x_1\ x_0\ .\ x_{-1}\ x_{-2}\cdots x_{-n}$ 는 아래와 같은 값을 가짐

$$\sum_{i=-n}^{p-1} x_i \cdot 8^i$$

- 기수 : 8
- 가장 왼쪽의 숫자(x_{p-1}) : Most Significant Digit (MSD)
- 가장 오른쪽의 숫자(x_{-n}) : Least Significant Digit (LSD)
- 예)
  - $731 = 7 \cdot 8^2 + 3 \cdot 8^1 + 1 \cdot 8^0 = 473$
  - $731.462 = 7 \cdot 8^2 + 3 \cdot 8^1 + 1 \cdot 8^0 + 4 \cdot 8^{-1} + 6 \cdot 8^{-2} + 2 \cdot 8^{-3} = 473.59765625$

십육진법

- 사용하는 숫자(hexadecimal digits) : $0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F$
- 십육진수(hexadecimal number) $x_{p-1}x_{p-2}\cdots x_1\, x_0\, .\, x_{-1}\, x_{-2}\cdots x_{-n}$ 는 아래와 같은 값을 가짐

$$\sum_{i=-n}^{p-1} x_i \cdot 16^i$$

- 기수 : 16
- 가장 왼쪽의 숫자(x_{p-1}) : Most Significant Digit (MSD)
- 가장 오른쪽의 숫자(x_{-n}) : Least Significant Digit (LSD)
- C 언어 등에서 십육진법으로 표현한 수 앞에 $0\times$를 붙여 십육진수임을 나타내기도 함
- 예)
  - $0\times 3AE = 3\cdot 16^2 + 10\cdot 16^1 + 14\cdot 16^0 = 798$
  - $3\mathrm{AE}.\,4\mathrm{C} = 3\cdot 16^2 + 10\cdot 16^1 + 14\cdot 16^0 + 4\cdot 16^{-1} + 12\cdot 16^{-2} = 798.296875$

4-bit 이진수, 팔진수, 십진수, 십육진수

Binary	Octal	Hexadecimal	Decimal
0000	00	0	0
0001	01	1	1
0010	02	2	2
0011	03	3	3
0100	04	4	4
0101	05	5	5
0110	06	6	6
0111	07	7	7
1000	10	8	8
1001	11	9	9
1010	12	A	10
1011	13	B	11
1100	14	C	12
1101	15	D	13
1110	16	E	14
1111	17	F	15

이진수에서 팔진수로

$$100011001110_2 = 100\ \ 011\ \ 001\ \ 110_2 = 4316_8$$

$$10.1011001011_2 = 010\ .\ 101\ \ 100\ \ 101\ \ 100_2 = 2.5454_8$$

이진수에서 십육진수로

$$100011001110_2 = 1000\ \ 1100\ \ 1110_2 = 8CE_{16}$$

$$10.1011001011_2 = 0010\ .\ 1011\ \ 0010\ \ 1100_2 = 2.B2C_{16}$$

십진 정수를 이진수로 변환하는 방법

- 십진 정수 N 을 이진수로 변환하여 이진 정수 $b_{p-1}b_{p-2}\cdots b_1b_0$를 얻는 알고리즘
  1. i 를 0으로 설정
  2. N 을 2로 나누어 몫과 나머지를 얻음
  3. b_i 를 나머지 값으로, N 을 몫의 값으로 설정
  4. N 이 0 이 아니면 i 를 1 증가 시킨 후 2 단계부터 다시 반복
- 예) 십진수 108을 이진수로 변환

i	N	b_i
0	108/2 = 54	0 (LSB)
1	54/2 = 27	0
2	27/2 = 13	1
3	13/2 = 6	1
4	6/2 = 3	0
5	3/2 = 1	1
6	1/2 = 0	1 (MSB)

십진 소수를 이진수로 변환하는 방법

- 십진 소수 N 을 이진수로 변환하여 이진 소수 $b_{-1}b_{-2}\cdots b_{-n}$ 를 얻는 알고리즘
  1. i 를 1 로 설정
  2. N 에 2를 곱하여 그 결과 값을 N 의 값으로 설정
  3. N 이 1보다 작으면 b_{-i} 를 0 으로 설정하고, 아니라면 b_{-i} 를 1 로 설정한 후 N 을 1 감소시킴
  4. i 가 전체 자릿수 n 보다 작으면 i 를 1 증가시킨 후 2 단계부터 다시 반복
- 예) 십진수 0.735를 네자리 이진소수 0.1011로 변환

i	N	b_{-i}
1	0.735 X 2 = 1.47	1
2	0.47 X 2 = 0.94	0
3	0.94 X 2 = 1.88	1
4	0.88 X 2 = 1.76	1

십진수를 이진수로 변환하기

- 정수부와 소수부를 구분하여 이진수로 변환
- 변환 이후, 정수부 변환 결과와 소수부 변환 결과를 소수점과 조합하여 표기
- 예) 108.735_{10}는 십진정수의 변환과 십진소수의 변환 결과를 조합하여 1101100.1011_2로 변환됨

십진수를 팔진수로 변환하기

- 십진수를 이진수로 변환하는 과정과 거의 같음
- 예) 십진수 108.735를 팔진수로 변환

i	N	b_i
0	$108/8 = 13$	$4 \quad (LSB)$
1	$13/8 = 1$	5
2	$1/8 = 0$	$1 \quad (MSB)$

i	N	b_{-i}
1	$0.735 \times 8 = 5.88$	5
2	$0.88 \times 8 = 7.04$	7
3	$0.04 \times 8 = 0.32$	0
4	$0.32 \times 8 = 2.56$	2

십진수를 십육진수로 변환하기

- 십진수를 이진수로 변환하는 과정과 거의 같음
- 예) 십진수 108.735를 십육진수로 변환

i	N	b_i	
0	$108/16 = 6$	$12(C)$	(LSB)
1	$6/16 = 0$	6	(MSB)

i	N	b_{-i}
1	$0.735 \times 16 = 11.76$	$11(B)$
2	$0.76 \times 16 = 12.16$	$12(C)$
3	$0.16 \times 16 = 2.56$	2
4	$0.56 \times 16 = 8.96$	8

십진수를 변환하는 다른 방법

- 주어진 십진수를 이진수로 변환

- 이진수로 변환된 수를 다시 팔진수, 혹은 십육진수로 변환

- 예)
  - $108.731_{10} = 001\ 101\ 100\ .\ 101\ 111\ 000\ 010_2 = 154.5702_8$
  - $108.731_{10} = 0110\ 1100\ .\ 1011\ 1100\ 0010\ 1000_2 = 6C.BC28_8$

부호가 있거나 없는 이진수

- Unsigned/signed binary numbers
- n 비트로 2^n 개의 서로 다른 이진수를 표현 가능
- 부호가 없는(unsigned) 이진수
  - 0을 포함한 2^n 개의 양수
- 부호가 있는(signed) 이진수
  - 음수와 0, 양수를 모두 나타낼 수 있음
  - MSB를 부호(sign) 비트로 사용하고, 나머지 비트들은 수의 크기를 나타냄
- 컴퓨터 하드웨어를 최대한 간단히 구현하기 위하여 2의 보수 표현을 많이 사용함
  - 거의 모든 컴퓨터시스템에서 2의 보수 표현을 사용

부호 없는 수(unsigned number)

- 이진수가 0이나 양의 값을 나타낼 때
- n-비트 부호 없는 이진 정수의 값 x의 범위
  - $0 \leq x \leq 2^n - 1$

부호 붙은 수(signed number)

- n 개의 비트를 이용하여 0과 양수뿐만 아니라 음수도 인코딩할 수 있음
- 부호 붙은 수를 n 개의 비트로 인코딩하는 방법
  - 부호 붙은 크기 표현(signed magnitude representation)
    - 0과 양수의 경우 MSB를 0으로, 음수의 경우 MSB를 1로 설정하고 나머지 비트가 그 수의 크기를 나타냄
  - 1의 보수 표현
  - 2의 보수 표현
    - 하드웨어의 구현이 간단함
    - 대부분의 컴퓨터가 사용 중

보수

- r-진법으로 표현된 n 개의 자리를 가진 수 x의 r의 보수(補數, radix complement, complement) $\bar{x}$는,
  - $\bar{x} = \begin{cases} r^n - x, & x \neq 0 \\ 0, & x = 0 \end{cases}$

- x의 $(r-1)$의 보수(diminished radix complement) x'은,
  - $x' = (r^n - 1) - x$

보수 구하기

- x의 $(r-1)$의 보수는 x의 각 자리에 있는 숫자의 $(r-1)$의 보수를 취하면 됨
  - 각 자리의 숫자를 $(r-1)$에서 빼는 것
  - $(r^n-1)-x = (r^n-1^n)-x = ((r-1)\cdot r^{n-1}+(r-1)\cdot r^{n-2}+\cdots+(r-1)\cdot r^0)-x$
  - 예) 십진수 836의 $(10-1)$의 보수는 $(10^3-1)-836 = 999-836 = 163$

- x가 0 이 아닌 경우, x의 $(r-1)$의 보수에 1을 더하면 x의 r의 보수를 얻을
  - 예) 세자리 십진수 836의 $(10-1)$의 보수는 163 이고 10의 보수는 $10^3 - 836 = 1000-836 = 164$

1의 보수 표현(1's complement representation)

- $r = 2$ 일 경우에 $(r - 1)$의 보수를 사용하여 음수를 표현
- 예) 6_{10}의 1의 보수는 $(2^4 - 1) - 6 = 15 - 6 = 9_{10}$
  - 9_{10}는 1001_2이고 1의 보수 표현에서 -6_{10}을 나타냄
- n-비트 1의 보수 표현으로 나타낼 수 있는 가장 큰 정수는 $2^{n-1} - 1$ 이고 가장 작은 정수는 $-2^{n-1} + 1$
- MSB는 부호 비트(sign bit)
  - 1 이면 음수, 0이면 양수
- 0을 표현하는 방법이 두 가지인 단점
  - $0000\ (+0)$
  - $1111\ (-0)$

2의 보수 표현(2's complement representation)

- 현재 대부분의 컴퓨터가 사용 중
- n-비트 이진수로 표현되는 정수 x의 2의 보수 $\bar{x}$
  - $\bar{x} = \begin{cases} 2^n - x, & x \neq 0 \\ 0, & x = 0 \end{cases}$
  - 가장 큰 수는 $2^{n-1} - 1$ 이고 가장 작은 수는 -2^{n-1}
- 먼저 1의 보수를 구하고 1을 더하면 됨
  - 가장 오른쪽에 나오는 1과 뒤따라 나오는 0을 제외한 모든 비트들을 뒤집는(flip) 것과 같음
  - 예) 6_{10}의 2의 보수는 $10_{10}(1010_2)$이고, 1010_2이 2의 보수 표현에서 -6_{10}을 나타냄
- MSB는 부호 비트

모듈라 연산

- Modular arithmetic 또는 clock arithmetic
- 모듈로 연산(modulo operation)을 이용
  - 유클리드 나눗셈에서 나머지를 구하는 연산
  - mod로 표기
- 유클리드 알고리즘(Euclidean algorithm)
  - 유클리드 나눗셈(Euclidean division)이라고도 불림
  - 두 정수 m과 $n(\neq 0)$이 주어졌을 때 $m = qn + r$과 $0 \leq r < |n|$을 만족하는 유일한 정수 q와 r이 항상 존재
    - q는 m을 n으로 나눈 몫이고 r은 나머지

우리가 잘 알고 있는 나눗셈

- 음수의 경우 유클리드 나눗셈과 다름
- 우리가 알고 있는 나눗셈에서 -7을 3으로 나누면 몫이 -2 이고 나머지가 -1
- 유클리드 나눗셈은 -7을 3으로 나누면 몫이 -3이고 나머지가 1

mod

- 두 정수 x와 $m(\neq 0)$이 주어질 때 x/m의 몫 q 는 $\lfloor x/m \rfloor$으로 나타낼 수 있고, 그 나머지 r 을 나타내는 연산이 mod
  - $q = \left\lfloor \frac{x}{m} \right\rfloor,\ m \neq 0$
  - $r = x \bmod m = x - mq,\ m \neq 0$
  - m은 modulus라고 불림

합동관계(合同關係, congruence relation)

- $x \bmod m = y \bmod m$
  - x와 y의 관계를 $x \equiv y \pmod m$이라 표기
  - x와 y는 모듈로 m에 대하여 합동
  - x와 y가 m으로 나누었을 때 유클리드 나눗셈의 나머지가 같음

모듈로-m 연산(modulo-m operation)

- 모듈러스가 m인 합동관계를 이용한 연산
- m 개의 수 $0, 1, 2, \cdots, m-1$을 연산에 이용
- 예) 모듈로-8 덧셈
  - $1 + 4 \equiv 5 \pmod 8$
  - $8 + 5 \equiv 5 \pmod 8$
  - $7 + 3 \equiv 2 \pmod 8$
  - $0 + 8 \equiv 0 \pmod 8$

OR, AND, and NOT 연산의 정의

- $OR : X \vee Y$ 의 값은 X 또는 Y가 1이거나 모두가 1일 때 1, 그렇지 않으면 0
- $AND : X \wedge Y$ 의 값은 X 와 Y 모두가 1 일 때 1, 그렇지 않으면 0
- NOT(negation) : X'의 값은 X 가 1 이면 0, 그렇지 않으면 1
- 진리표(truth table)
  - 인자의 모든 가능한 값들을 나열하고 각 경우에 대해 연산의 결과 값을 보인 것

OR		
x	y	x ∨ y
0	0	0
0	1	1
1	0	1
1	1	1

AND		
x	y	x ∧ y
0	0	0
0	1	0
1	0	0
1	1	1

NOT	
x	x′
0	1
1	0

XOR, XNOR, NOR, NAND

- XOR : Exclusive OR $x \oplus y = (x \wedge y') \vee (x' \wedge y) = \begin{cases} 1 & \text{when } x \neq y \\ 0 & \text{otherwise} \end{cases}$
- XNOR : Exclusive NOT-OR $x \odot y = (x \wedge y) \vee (x' \wedge y') = \begin{cases} 1 & \text{when } x = y \\ 0 & \text{otherwise} \end{cases}$
- NOR : NOT-OR

  $(x \vee y)'$
- NAND : NOT-AND

  $(x \wedge y)'$

XOR		
x	y	x⊕y
0	0	0
0	1	1
1	0	1
1	1	0

XNOR		
x	y	x⊙y
0	0	1
0	1	0
1	0	0
1	1	1

NOR		
x	y	(x∨y)′
0	0	1
0	1	0
1	0	0
1	1	0

NAND		
x	y	(x∧y)′
0	0	1
0	1	1
1	0	1
1	1	0

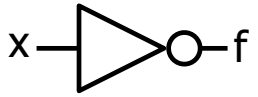
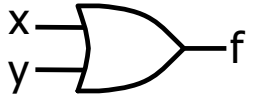
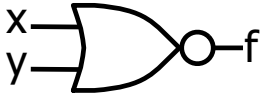
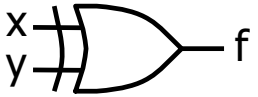
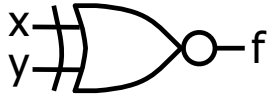
논리 게이트(Logic Gate)

- 논리 게이트는 개념적인 또는 물리적인 장치로 한 개 이상의 Boole 연산을 수행

- 논리 게이트로 Boole 함수를 구현할 수 있음

- 논리 게이트는 블랙박스로 볼 수 있음(추상화)
  - $f: B^n \rightarrow B^m$
    - n개의 입력 변수와 m개의 출력 변수
    - n개의 입력 핀과 m개의 출력 핀
  - 어떤 식으로 내부가 구성되어 있는지 신경 쓰지 않고 입력과 출력, 수행하는 연산만 알고 있으면 됨
    - 논리 게이트를 구성하는 전자회로에서 트랜지스터가 어떻게 조합되어 있고 어떤 동작을 하는지 신경 쓰지 않아도 됨

기본적인 논리 게이트

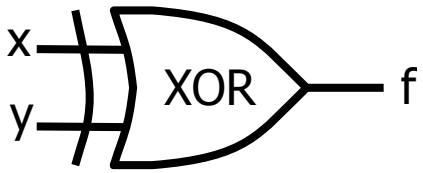
Name	Symbol	Function
NOT		f = x'
OR		f = x ∨ y
AND		f = x ∧ y
NOR		f = (x ∨ y)'
NAND		f = (x ∧ y)'
XOR		f = x ⊕ y
XNOR		f = x ⊙ y

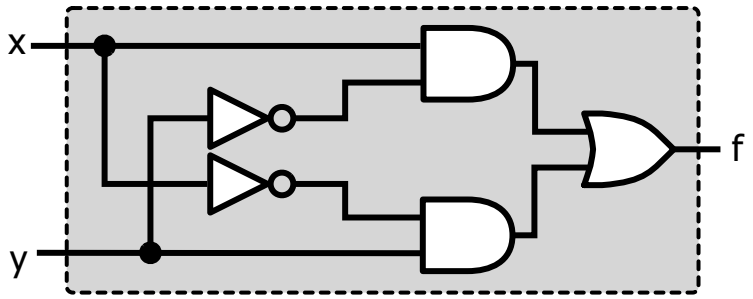
논리 게이트의 조합

- 논리 다이어그램(logic diagram)은 논리 회로를 그림으로 표현한 것으로 논리 게이트 간의 연결을 보여줌

- 기본 논리게이트를 조합하고 연결해서 좀 더 복잡한 기능의 논리 게이트를 만들 수 있음

- 예) AND, OR, NOT 게이트로 XOR 을 구현





이진 정수의 논리 연산

- n-비트 이진수의 논리연산은 비트 단위로 주어진 논리연산을 적용함
  - NOT, AND, OR, NOR, NAND, XOR, XNOR
- C 언어는 NOT, AND, OR, XOR를 지원함
  - ~, &, |, ^
  - 피연산자는 정수 타입이어야 함

이진 정수의 시프트(shift) 연산

- 주어진 n-비트 이진수의 모든 비트를 주어진 회수만큼 주어진 방향(오른쪽 또는 왼쪽)으로 이동시키는 연산
  - $x \gg m$: n-비트 이진수 x를 m 비트만큼 왼쪽으로 시프트하는 연산
  - $x \ll m$: n-비트 이진수 x를 m 비트만큼 오른쪽으로 시프트하는 연산
  - 논리 시프트(logical shift)와 산술 시프트(arithmetic shift)가 있음
    - 논리 시프트와 산술 시프트의 구분은 첨자 L과 A를 붙여 표시

$$10011101_2 \gg_L 3 = 00010011_2$$

$$10011101_2 \ll_L 3 = 11101000_2$$

$$10011101_2 \gg_A 3 = 11110011_2$$

$$10011101_2 \ll_A 3 = 11101000_2$$

시프트 연산과 곱셈

- 부호 없는 수나 2의 보수 표현에서 어떤 수 x를 왼쪽으로 m 비트 시프트 하는 것은 x에 2^m을 곱하는 것과 동일
- 부호 없는 수

$$00001101_2(13_{10}) \ll 3 = 01101000_2(104_{10})$$

- 2의 보수 표현

$$11111101_2(-3_{10}) \ll 3 = 11101000_2(-24_{10})$$

시프트 연산과 나눗셈

- n-비트 부호 없는 수의 표현에서 어떤 수 x를 오른쪽으로 m 비트 논리 시프트나 산술 시프트 하는 것은 x를 2^m으로 나누어 몫을 취하는 것과 동일

$$1111101_2(29_{10}) \gg 3 = 00000011_2(3_{10})$$

- 2의 보수 표현에서는 $x \geq 0$ 일 경우, x를 오른쪽으로 m 비트 산술 시프트 하는 것은 부호 없는 수와 마찬가지로 x를 2^m으로 나누어 몫을 취하는 것과 같음
  - 하지만, $x < 0$일 경우는 x를 2^m으로 나누어 몫을 취하는 것과 다름

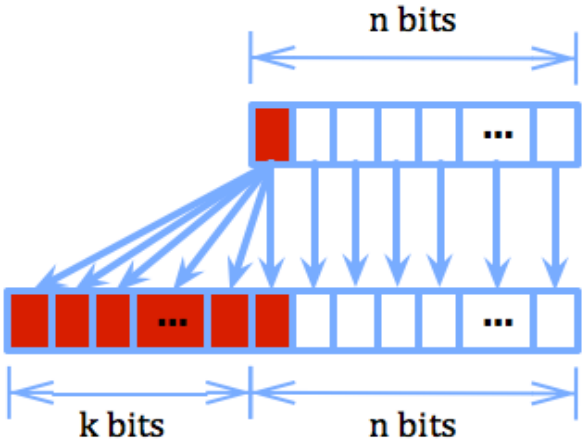
$$11111001_2(-7_{10}) \gg 2 = 11111110_2(-2_{10})$$

C 언어의 시프트 연산

- 오른쪽 시프트와 왼쪽 시프트 연산을 지원
- n-비트 수 x의 오른쪽 m-비트 시프트는 $x \gg m$으로 표현하고, 왼쪽 m-비트 시프트는 $x \ll m$으로 표현
- C99 표준은 음수에 대한 오른쪽 시프트를 정의하지 않음
  - 동작은 C 컴파일러의 구현에 따라 다름
  - 즉, 오른쪽 시프트가 논리 시프트인지 산술 시프트인지는 컴파일러의 구현에 따라 다름

부호 확장(sign extension)

- n-비트 2의 보수 표현을 같은 값을 가지는 $(n + k)$-비트 2의 보수 표현으로 변환할 때 필요
- 하위 n 비트는 원래의 n-비트 2의 보수 표현
- 원래의 부호 비트로 나머지 k 개의 비트를 모두 채움



부호 없는 이진 정수의 덧셈

- 십진수의 덧셈과 비슷
- 어떤 연산의 결과로 나온 값이 표현할 수 있는 값의 범위를 넘어가서 표현할 수 없을 경우를 오버플로우(overflow)라고 함
  - n-비트 부호 없는 수의 덧셈을 수행할 때 MSB에서 나오는 받아올림이 1 일 경우

받아올림	0	0	0	1	
		1	0	0	1 (9_{10})
+		0	1	0	1 (5_{10})
	0	1	1	1	0 (14_{10})

받아올림	1	1	1	1	
		1	0	1	1 (11_{10})
+		0	1	1	1 (7_{10})
	1	0	0	1	0 (18_{10})

부호 없는 이진 정수의 덧셈

- modulo 2^n 덧셈을 수행하는 것과 같음
  - 덧셈 결과로 나온 $n+1$개의 비트 중 하위 n 개의 비트를 취하고 MSB를 무시
- $0 \le x + y < 2^n$
  - $(x + y) \bmod 2^n = x + y$
- $2^n \le x + y \le 2^{n+1} - 2$: 오버플로우
  - $(x + y) \bmod 2^n = (x + y) - 2^n$

$$x +_U^n y = (x + y) \bmod 2^n$$
$$= \begin{cases} x + y, & 0 \le x + y < 2^n \\ x + y - 2^n, & 2^n \le x + y \le 2^{n+1} - 2 \text{ (오버플로우)} \end{cases}$$

2의 보수 표현의 덧셈

- 두 정수 x와 y를 n-비트 2의 보수 표현으로 나타내었을 때
  - $b_x = x_{n-1}x_{n-2}\cdots x_0$와 $b_y = y_{n-1}y_{n-2}\cdots y_0$
- $x +_C^n y$
  - b_x와 b_y를 부호 없는 이진 표현으로 취급하여 모듈로-2^n 덧셈을 수행한 다음, 그 결과를 다시 2의 보수 표현으로 해석하는 것

2의 보수 표현의 덧셈

- 오버플로우가 일어날 수 있음
  - 더하는 과정에서 MSB로 들어가는 받아올림이 MSB에서 나오는 받아올림과 다르면 오버플로우

MSB

받아올림	0	1	1	1	
		0	1	1	1	(7_{10})
+		0	1	1	1	(7_{10})
		1	1	1	0	

MSB로 들어가는 받아올림	x의 부호 비트	y의 부호비트	결과의 부호 비트	MSB에서 나오는 받아올림
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

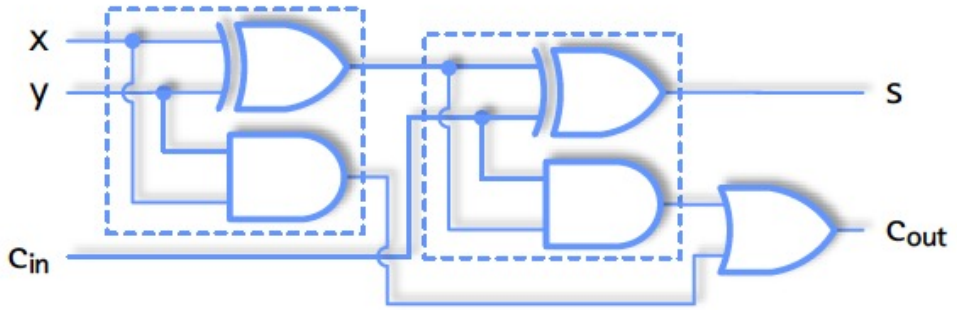
2의 보수 표현의 덧셈

$$x +_C^n y = \begin{cases} x + y - 2^n, & 2^{n-1} \leq x + y \text{ (오버플로우)} \\ x + y, & -2^{n-1} \leq x + y < 2^{n-1} \\ x + y + 2^n, & x + y < -2^{n-1} \text{ (오버플로우)} \end{cases}$$

전가산기와 반가산기

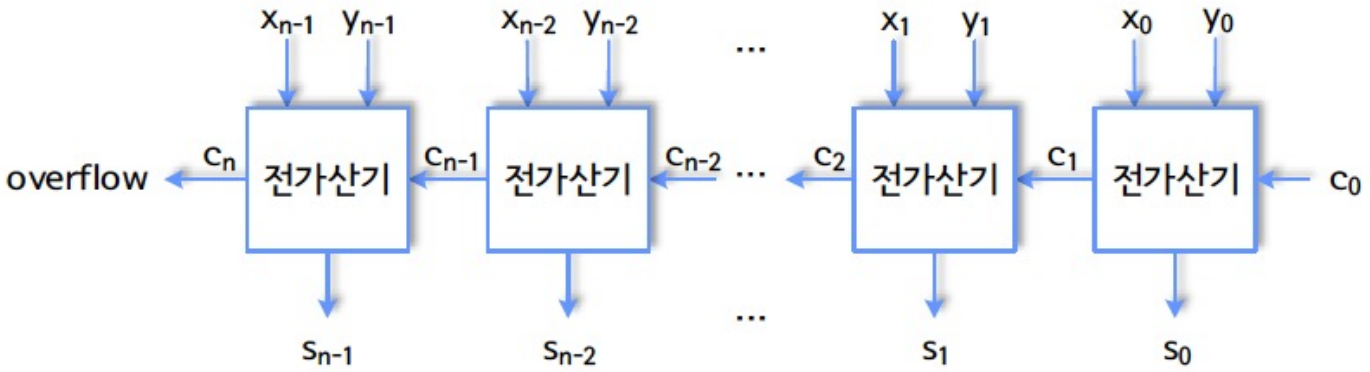
- 반가산기
  - 두 개의 비트 x와 y를 더하여 그 합 s와 받아올림(carry) c를 출력
- 전가산기
  - 세 개의 비트 x, y, c_{in}을 입력으로 받아들여 이들의 합을 한 개의 비트 s와 받아올림 c_{out}으로 출력

x	y	c_{in}	s	c_{out}
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



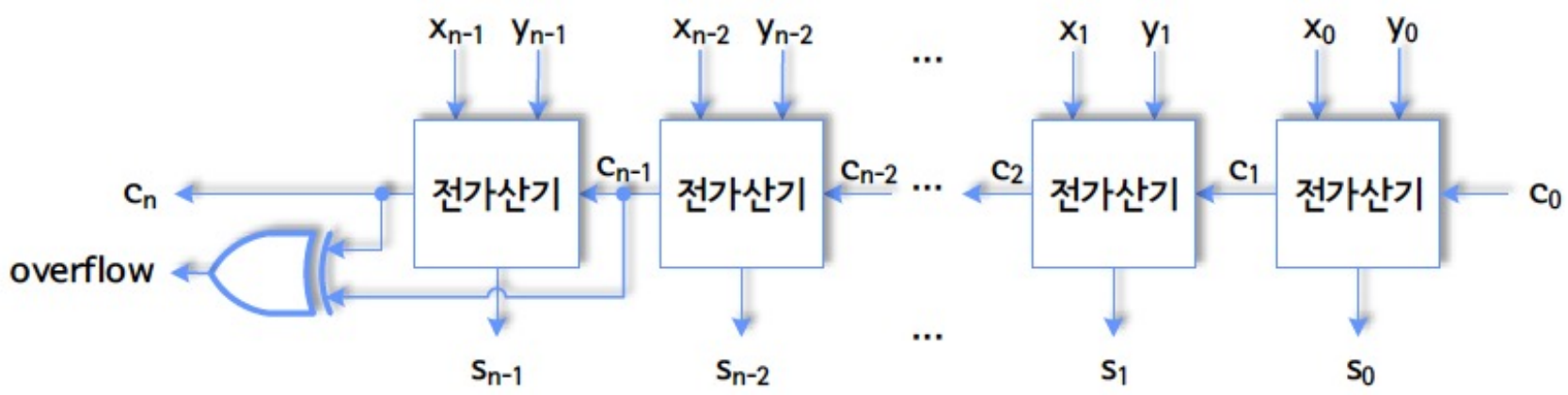
부호 없는 수의 덧셈 하드웨어

- n-비트 가산기(adder)
- 전가산기 $n-1$ 개와 한 개의 반가산기를 연결하여 두 개의 n-비트 부호 없는 이진수 $x = x_{n-1}x_{n-2}\cdots x_0$와 $y = y_{n-1}y_{n-2}\cdots y_0$의 덧셈 $x +_U^n y$을 수행하여 결과로 $s = s_{n-1}s_{n-2}\cdots s_0$를 출력
  - c_n이 1 이면 오버플로우



2의 보수 표현의 덧셈 하드웨어

- 부호 없는 이진수의 덧셈 $+_U^n$을 이용
  - 오버플로우 감지만 다름



이진 정수의 뺄셈

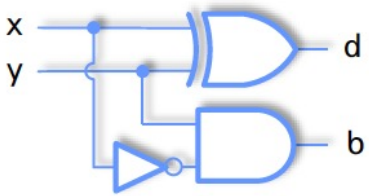
- 부호 없는 수
  - 십진수의 뺄셈과 그 방식이 같음
- 2의 보수 표현
  - 2의 보수 표현으로 표현된 두 정수 x와 y 뺄셈 $x - y$는 $x - y = x + (-y)$ 이므로 감수(減數) y의 2의 보수를 취하여 덧셈으로 변경
  - $x + (-y) = x + y' + 1$

받아내림	0	1	0	0	
		1	0	0	1 (9_{10})
−		0	1	0	1 (5_{10})
		0	1	0	0 (4_{10})

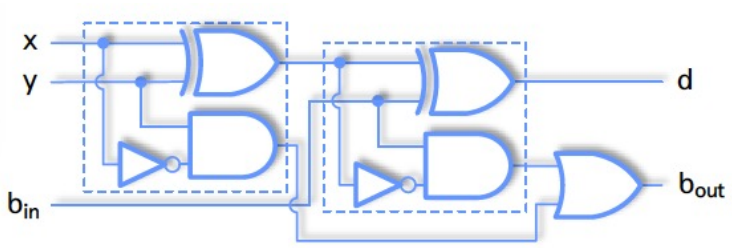
반가산기와 전감산기

- 반감산기(半減算器, half subtractor)
  - 두 개의 비트 x와 y를 입력으로 받아들여 $x - y$을 계산한 결과를 차 d와 받아내림(borrow) b로 출력
- 전감산기(全減算器, full subtractor)
  - 세 개의 비트 x, y, b_{in}을 입력으로 받아들여 $x - y - b_{in}$을 계산한 결과를 한 개의 비트 d와 받아내림 b_{out}으로 출력

x	y	d	b
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

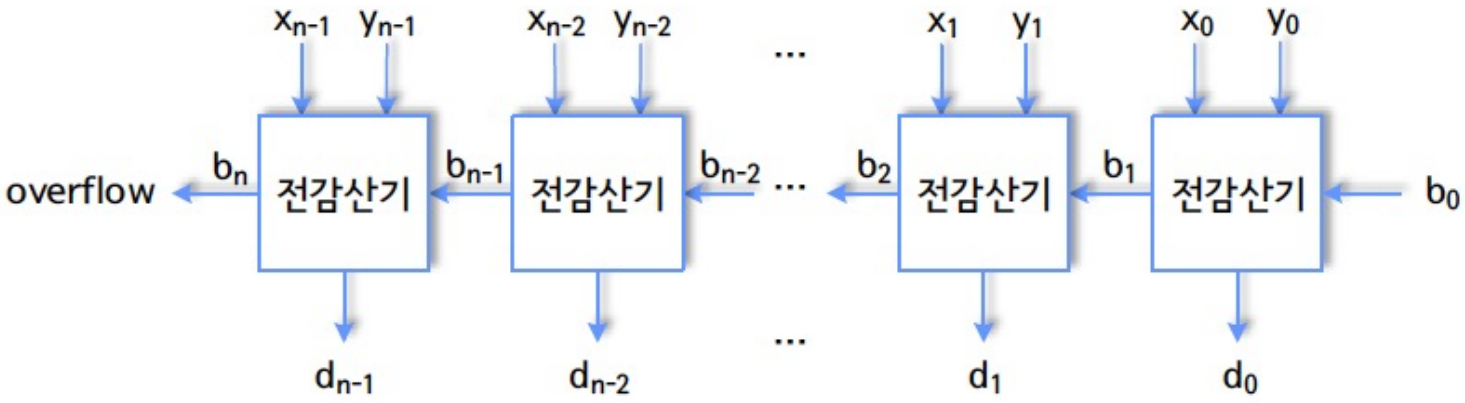


x	y	b_{in}	d	b_{out}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1



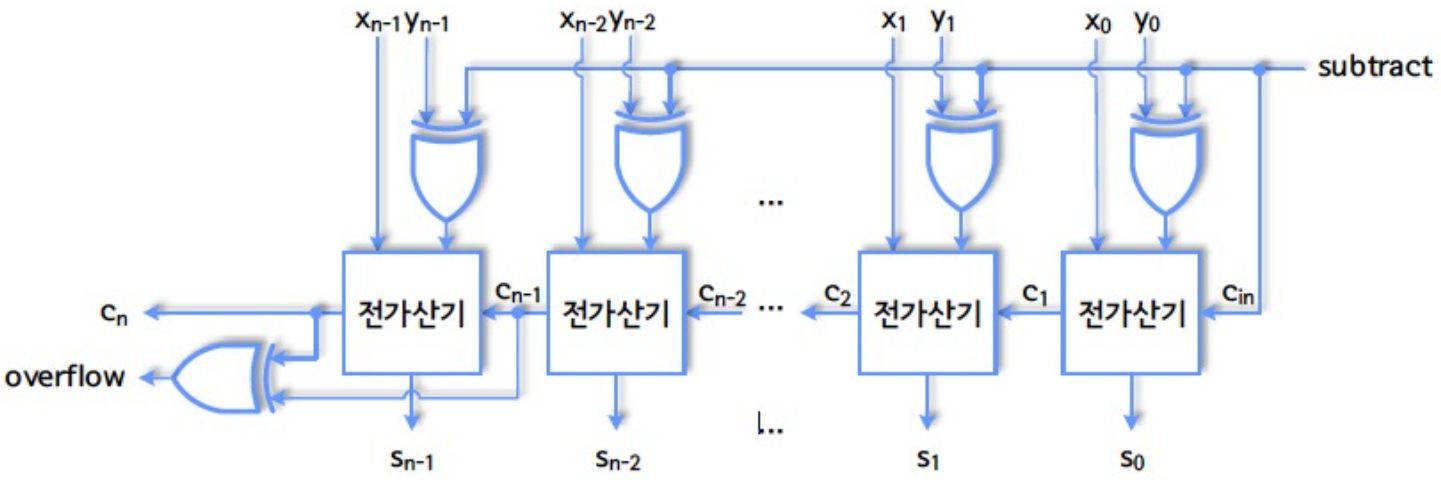
부호 없는 수의 뺄셈 하드웨어

- $n-1$ 개의 전감산기와 한 개의 반감산기를 이용하여 구현
- $x < y$ 이면 b_{n-1}이 1 이 되고, 결과가 n-비트 부호 없는 이진수로 표현할 수 없는 음수가 되어 오버플로우가 일어남



2의 보수 표현의 뺄셈 하드웨어

- Subtract=0, 덧셈 수행
- Subtract=1, 뺄셈 수행



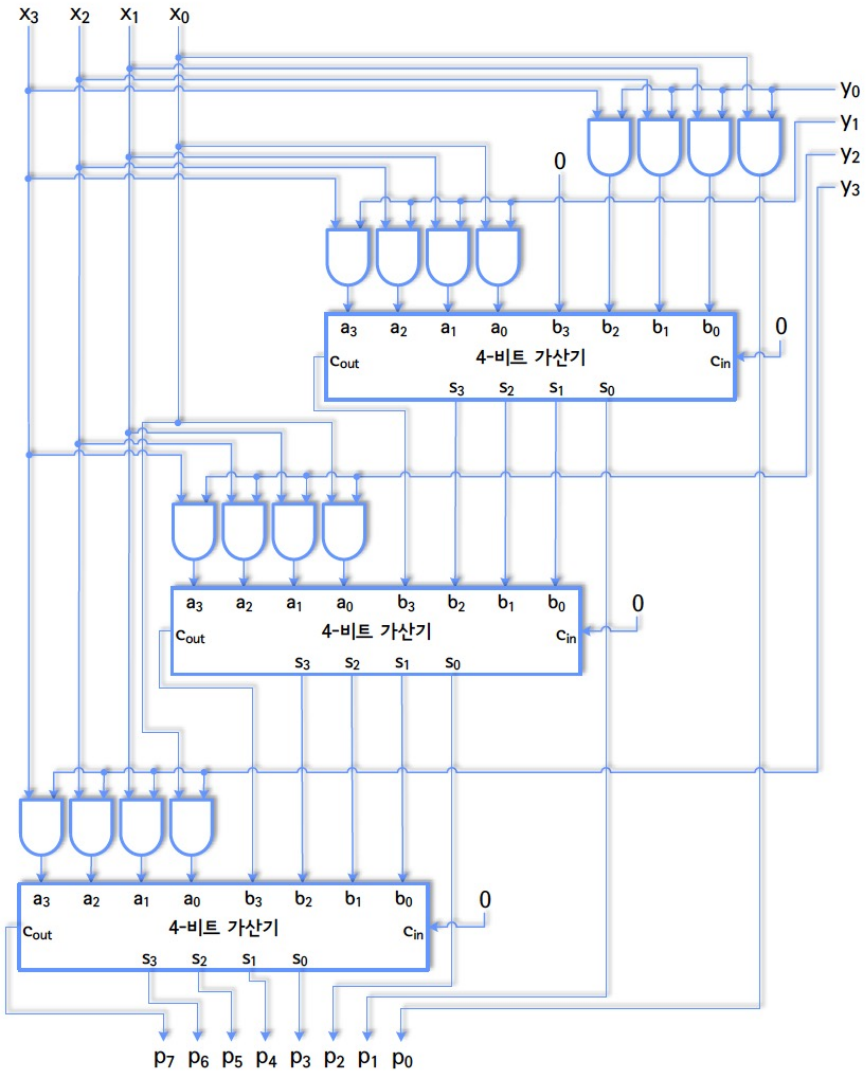
부호 없는 수의 곱셈

- 십진수의 곱셈과 그 원리가 같음

$$
\begin{array}{cccccccc}
 & & & & x_3 & x_2 & x_1 & x_0 \\
\times & & & & y_3 & y_2 & y_1 & y_0 \\
 & & & & x_3 \cdot y_0 & x_2 \cdot y_0 & x_1 \cdot y_0 & x_0 \cdot y_0 \\
+ & & & x_3 \cdot y_1 & x_2 \cdot y_1 & x_1 \cdot y_1 & x_0 \cdot y_1 & \\
+ & & x_3 \cdot y_2 & x_2 \cdot y_2 & x_1 \cdot y_2 & x_0 \cdot y_2 & & \\
+ & x_3 \cdot y_3 & x_2 \cdot y_3 & x_1 \cdot y_3 & x_0 \cdot y_3 & & & \\
\hline
p_7 & p_6 & p_5 & p_4 & p_3 & p_2 & p_1 & p_0
\end{array}
$$

부호 없는 수의 곱셈

- 하드웨어는 $(n-1)^2$ 개의 전가산기(FA)와 $n-1$ 개의 반가산기(HA)를 이용



2의 보수 표현의 곱셈

$$x \cdot y = \left(\left(-x_3 2^3 + \sum_{i=0}^{2} x_i \cdot 2^i\right) \times \left(-y_3 2^3 + \sum_{i=0}^{2} y_i \cdot 2^i\right)\right)$$

$$\begin{aligned}
&(-x_3 2^3 + x_2 2^2 + x_1 2^1 + x_0 2^0) \times (-y_3 2^3 + y_2 2^2 + y_1 2^1 + y_0 2^0) \\
&= (-x_3 \cdot y_0 \cdot 2^3 + x_2 \cdot y_0 \cdot 2^2 + x_1 \cdot y_0 \cdot 2^1 + x_0 \cdot y_0 \cdot 2^0) \\
&+(-x_3 \cdot y_1 \cdot 2^4 + x_2 \cdot y_1 \cdot 2^3 + x_1 \cdot y_1 \cdot 2^2 + x_0 \cdot y_1 \cdot 2^1) \\
&+(-x_3 \cdot y_2 \cdot 2^5 + x_2 \cdot y_2 \cdot 2^4 + x_1 \cdot y_2 \cdot 2^3 + x_0 \cdot y_2 \cdot 2^2) \\
&-(-x_3 \cdot y_3 \cdot 2^6 + x_2 \cdot y_3 \cdot 2^5 + x_1 \cdot y_3 \cdot 2^4 + x_0 \cdot y_3 \cdot 2^3)
\end{aligned}$$

2의 보수 표현의 곱셈

- 부분적인 곱셈 결과를 나타내기 위해 부호확장이 필요

1						x_3	x_2	x_1	x_0
2	×					y_3	y_2	y_1	y_0
3		$x_3 \cdot y_0$	$x_3 \cdot y_0$	$x_3 \cdot y_0$	$x_3 \cdot y_0$	$x_3 \cdot y_0$	$x_2 \cdot y_0$	$x_1 \cdot y_0$	$x_0 \cdot y_0$
4	+	$x_3 \cdot y_1$	$x_3 \cdot y_1$	$x_3 \cdot y_1$	$x_3 \cdot y_1$	$x_2 \cdot y_1$	$x_1 \cdot y_1$	$x_0 \cdot y_1$	0
5	+	$x_3 \cdot y_2$	$x_3 \cdot y_2$	$x_3 \cdot y_2$	$x_2 \cdot y_2$	$x_1 \cdot y_2$	$x_0 \cdot y_2$	0	0
6	−	$x_3 \cdot y_3$	$x_3 \cdot y_3$	$x_2 \cdot y_3$	$x_1 \cdot y_3$	$x_0 \cdot y_3$	0	0	0
7		p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

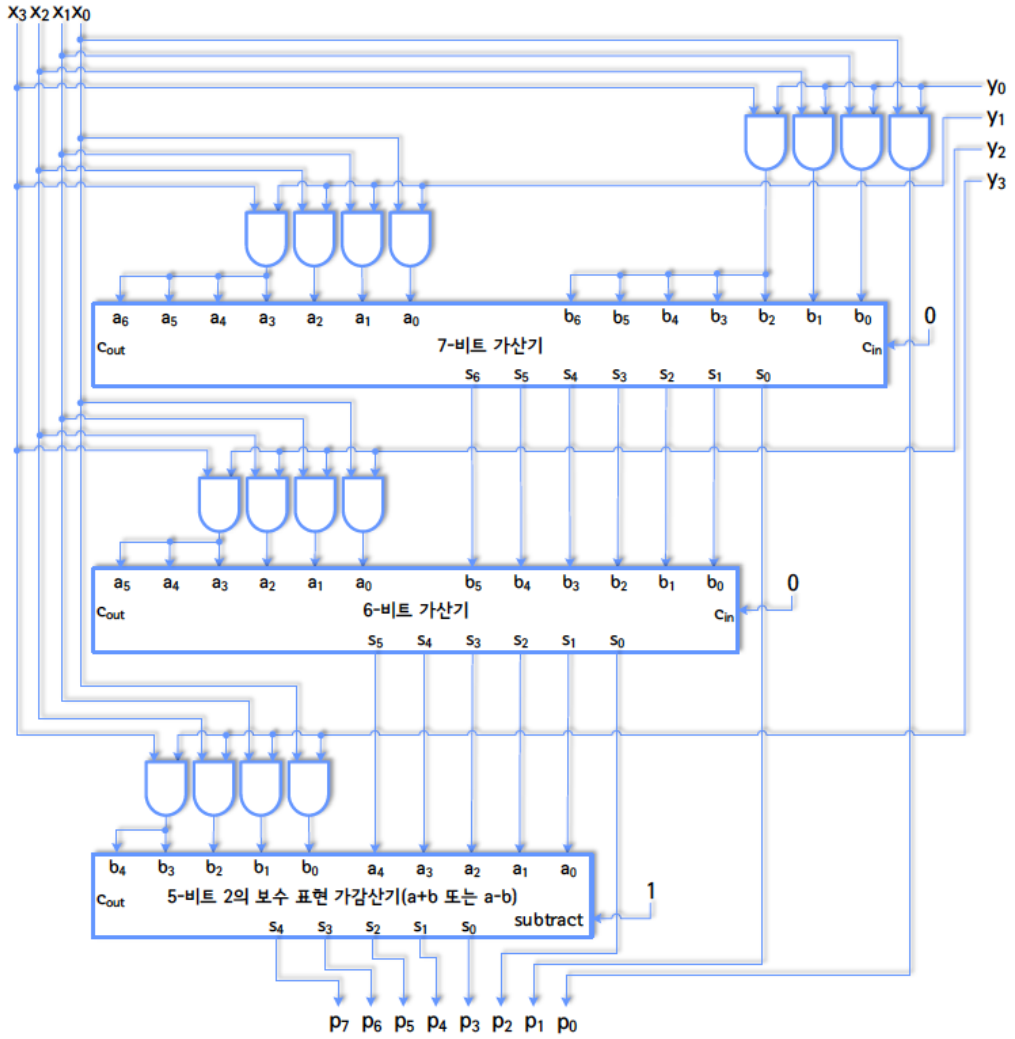
1						x_3	x_2	x_1	x_0
2	×					y_3	y_2	y_1	y_0
3		$x_3 \cdot y_0$	$x_3 \cdot y_0$	$x_3 \cdot y_0$	$x_3 \cdot y_0$	$x_3 \cdot y_0$	$x_2 \cdot y_0$	$x_1 \cdot y_0$	$x_0 \cdot y_0$
4	+	$x_3 \cdot y_1$	$x_3 \cdot y_1$	$x_3 \cdot y_1$	$x_3 \cdot y_1$	$x_2 \cdot y_1$	$x_1 \cdot y_1$	$x_0 \cdot y_1$	0
5	+	$x_3 \cdot y_2$	$x_3 \cdot y_2$	$x_3 \cdot y_2$	$x_2 \cdot y_2$	$x_1 \cdot y_2$	$x_0 \cdot y_2$	0	0
6	+	$\sim(x_3 \cdot y_3)$	$\sim(x_3 \cdot y_3)$	$\sim(x_2 \cdot y_3)$	$\sim(x_1 \cdot y_3)$	$\sim(x_0 \cdot y_3)$	1	1	1
7	+	0	0	0	0	0	0	0	1
8		p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

2의 보수 표현의 곱셈 하드웨어

1						x_3	x_2	x_1	x_0
2	×					y_3	y_2	y_1	y_0
3		$x_3 \cdot y_0$	$x_3 \cdot y_0$	$x_3 \cdot y_0$	$x_3 \cdot y_0$	$x_3 \cdot y_0$	$x_2 \cdot y_0$	$x_1 \cdot y_0$	$x_0 \cdot y_0$
4	+	$x_3 \cdot y_1$	$x_3 \cdot y_1$	$x_3 \cdot y_1$	$x_3 \cdot y_1$	$x_2 \cdot y_1$	$x_1 \cdot y_1$	$x_0 \cdot y_1$	0
5	+	$x_3 \cdot y_2$	$x_3 \cdot y_2$	$x_3 \cdot y_2$	$x_2 \cdot y_2$	$x_1 \cdot y_2$	$x_0 \cdot y_2$	0	0
6	+	$\sim(x_3 \cdot y_3)$	$\sim(x_3 \cdot y_3)$	$\sim(x_2 \cdot y_3)$	$\sim(x_1 \cdot y_3)$	$\sim(x_0 \cdot y_3)$	1	1	1
7	+	0	0	0	0	0	0	0	1
8		p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

1						x_3	x_2	x_1	x_0
2	×					y_3	y_2	y_1	y_0
3		$x_3 \cdot y_0$	$x_3 \cdot y_0$	$x_3 \cdot y_0$	$x_3 \cdot y_0$	$x_3 \cdot y_0$	$x_2 \cdot y_0$	$x_1 \cdot y_0$	$x_0 \cdot y_0$
4	+	$x_3 \cdot y_1$	$x_3 \cdot y_1$	$x_3 \cdot y_1$	$x_3 \cdot y_1$	$x_2 \cdot y_1$	$x_1 \cdot y_1$	$x_0 \cdot y_1$	0
5	+	$x_3 \cdot y_2$	$x_3 \cdot y_2$	$x_3 \cdot y_2$	$x_2 \cdot y_2$	$x_1 \cdot y_2$	$x_0 \cdot y_2$	0	0
6	+	$\sim(x_3 \cdot y_3)$	$\sim(x_3 \cdot y_3)$	$\sim(x_2 \cdot y_3)$	$\sim(x_1 \cdot y_3)$	$\sim(x_0 \cdot y_3)$	0	0	0
7	+	0	0	0	0	1	0	0	0
8		p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0

2의 보수 표현의 곱셈 하드웨어



부호 없는 이진수의 나눗셈

- 십진수의 나눗셈과 원리가 같음
- 기본적으로 연속적인 뺄셈을 수행하여 제수가 피제수에 몇 개(몫) 들어 있는가를 알아내는 과정

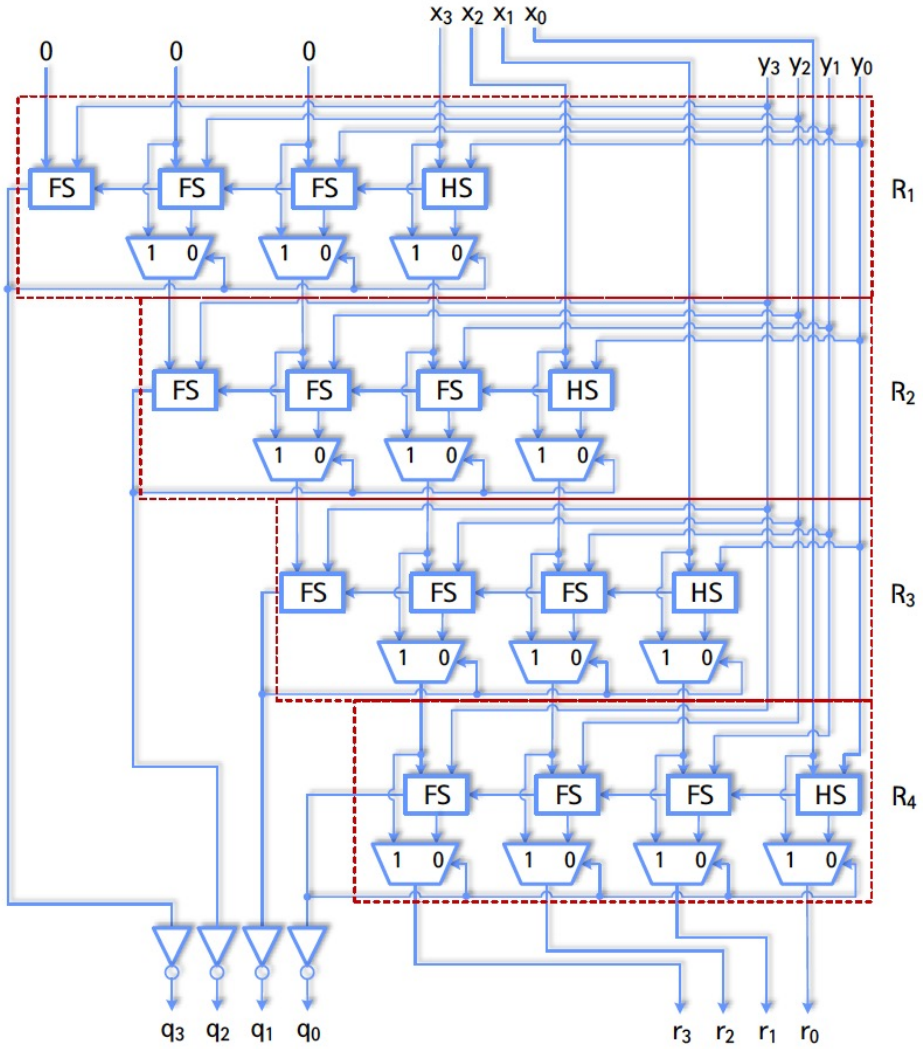
							1	1	(3_{10})
(4_{10})	1	0	0		1	1	0	1	(13_{10})
				−	1	0	0		
					0	1	0	1	
				−		1	0	0	
						0	0	1	

부호 없는 이진수의 나눗셈

- n-비트 부호 없는 이진수 x를 y로 나누어 몫 q와 나머지 r을 구하는 과정은 $R_0 = x$로 놓은 후 매 단계 마다 부분적인 나머지(partial remainder) R_i $(1 \le i \le n)$를 계산하는 과정
  - $R_i = R_{i-1} - q_{n-i} \cdot y \cdot 2^{n-i}$
- $q_{n-i} = 1$을 가정하고 R_i를 계산하여 $R_i \ge 0$ 이면 $q_{n-i} = 1$로 확정하고, 그렇지 않으면 $q_{n-i} = 0$ 과 $R_i = R_{i-1}$ 로 놓음

								q_3	q_2	q_1	q_0	
	y_3	y_2	y_1	y_0				0	0	1	1	$q = 3_{10}$
$y = 4_{10}$	0	1	0	0		0	0	0	1	1	0	1	$R_0 = x = 13_{10}$
					−	$q_3 0$	$q_3 1$	$q_3 0$	$q_3 0$	0	0	0	$q_3 \cdot y \cdot 2^3$
						~~1~~	~~1~~	~~0~~	~~0~~				~~$R_1 = R - y \cdot 2^3$~~
						0	0	0	1	1	0	1	$R_1 = R_0$
					−		$q_2 0$	$q_2 1$	$q_2 0$	$q_2 0$	0	0	$q_2 \cdot y \cdot 2^2$
							~~1~~	~~1~~	~~1~~	~~1~~			~~$R_2 = R_1 - y \cdot 2^2$~~
							0	0	1	1	0	1	$R_2 = R_1$
					−			$q_1 0$	$q_1 1$	$q_1 0$	$q_1 0$	0	$q_1 \cdot y \cdot 2^1$
								0	0	1	0	1	$R_3 = R_2 - y \cdot 2^1$
					−				$q_0 0$	$q_0 1$	$q_0 0$	$q_0 0$	$q_0 \cdot y \cdot 2^0$
									0	0	0	1	$R_4 = R_3 - y \cdot 2^0$
									r_3	r_2	r_1	r_0	

부호 없는 이진수의 나눗셈 하드웨어



2의 보수 표현의 나눗셈

- 2의 보수 표현에서 정수 x를 $y(\neq 0)$로 나눈 몫 q와 나머지 r을 구하려면
  - 우선 $|x|$를 $|y|$로 나눈 몫 q'과 나머지 r'을 부호 없는 수의 나눗셈을 이용하여 구함
  - 그런 다음 x와 y의 부호가 다르면 $q = -q'$로 놓고, 같으면 $q = q'$로 놓음
  - $x < 0$ 이면 $r = -r'$로, $x \geq 0$ 이면 $r = r'$로 놓음

2의 보수 표현의 나눗셈

- $r = x$로 놓고 피제수 x와 제수 y의 부호에 따라 $|y|$가 $|x|$에 들어 있는 개수 q'을 계산
  - $x \geq 0,\ y \geq 0 : r \geq 0$ 일 동안 y를 r에서 연속적으로 빼서
  - $x \geq 0,\ y < 0 : r \geq 0$ 일 동안 y를 x에 연속적으로 더하여
  - $x < 0,\ y \geq 0 : r \leq 0$ 일 동안 y를 x에 연속적으로 더하여
  - $x < 0,\ y < 0 : r \leq 0$ 일 동안 y를 x에서 연속적으로 빼서
- x와 y의 부호가 다를 때는 $q = -q'$, 그렇지 않으면 $q = q'$

2의 보수 표현의 나눗셈

- n-비트 2의 보수 표현인 $x_{n-1}x_{n-2}\cdots x_0$와 $y_{n-1}y_{n-2}\cdots y_0$로 표현된 이진수 x와 y가 주어졌을 때, x를 y로 나누어 몫 q와 나머지 r을 구하는 과정은, $R_0 = x$로 놓은 다음 매 단계 마다 부분적인 나머지 R_i $(1 \le i \le n)$를 계산하는 과정
  - $R_i = R_{i-1} - (-1)^{x_{n-1}} \cdot (-1)^{y_{n-1}} \cdot {q'}_{n-i} \cdot y \cdot 2^{n-i}$
- $q_{n-i} = 1$로 가정하여 R_i를 계산한 다음, R_{i-1}과 R_i의 부호가 같거나 $R_i = 0$ 이면 $q_{n-i} = 1$ 로 확정하고, 그렇지 않으면 $q_{n-i} = 0$ 과 $R_i = R_{i-1}$ 로 놓음
  - y가 R_{i-1}에 2^{n-i} 개 들어 있는가를 확인하는 과정

2의 보수 표현의 나눗셈

								q'_3	q'_2	q'_1	q'_0	
	y_3	y_2	y_1	y_0				0	0	1	0	$q = 2_{10}$
$y = 3_{10}$	0	0	1	1		1	1	1	1	0	0	1	$R_0 = x = -7_{10}$
					+	$q'_3 0$	$q'_3 0$	$q'_3 1$	$q'_3 1$	0	0	0	$q'_3 \cdot y \cdot 2^3$
						~~0~~	~~0~~	~~1~~	~~0~~	~~0~~	~~0~~	~~1~~	~~$R_1 = R_0 + y \cdot 2^3$~~
						1	1	1	1	0	0	1	$R_1 = R_0$
					+	$q'_2 0$	$q'_2 0$	$q'_2 0$	$q'_2 1$	$q'_2 1$	0	0	$q'_2 \cdot y \cdot 2^2$
						~~0~~	~~0~~	~~0~~	~~0~~	~~1~~	~~0~~	~~1~~	~~$R_2 = R_1 + y \cdot 2^2$~~
						1	1	1	1	0	0	1	$R_2 = R_1$
					+	$q'_1 0$	$q'_1 0$	$q'_1 0$	$q'_1 0$	$q'_1 1$	$q'_1 1$	0	$q'_1 \cdot y \cdot 2^1$
						1	1	1	1	1	1	1	$R_3 = R_2 + 2^1$
					+	$q'_0 0$	$q'_0 0$	$q'_0 0$	$q'_0 0$	$q'_0 0$	$q'_0 1$	$q'_0 1$	$q'_0 \cdot y \cdot 2^0$
						~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~1~~	~~0~~	~~$R_4 = R_3 + y \cdot 2^0$~~
						1	1	1	1	1	1	1	$R_4 = R_3$
								r_3	r_2	r_1	r_0	

2의 보수 표현의 나눗셈 하드웨어