



# Lecture 03

## 이진 정수의 연산



# 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
  - $\sim$ ,  $\&$ ,  $|$ ,  $\wedge$
  - 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  $x$ 를  $m$  비트만큼 왼쪽으로 시프트하는 연산
  - $x \ll m$  :  $n$ -비트 이진수  $x$ 를  $m$  비트만큼 오른쪽으로 시프트하는 연산
  - 논리 시프트(logical shift)와 산술 시프트(arithmetic shift)가 있음
    - 논리 시프트와 산술 시프트의 구분은 첨자  $L$ 과  $A$ 를 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  $x$ 를 왼쪽으로  $m$  비트 시프트 하는 것은  $x$ 에  $2^m$ 을 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  $x$ 를 오른쪽으로  $m$  비트 논리 시프트나 산술 시프트 하는 것은  $x$ 를  $2^m$ 으로 나누어 몫을 취하는 것과 동일

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

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

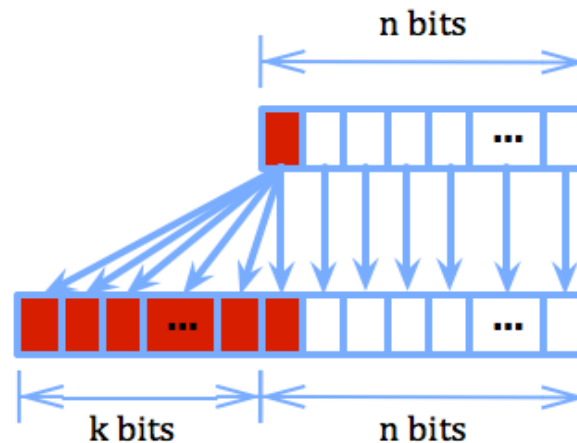
$$11111001_2(-7_{10}) \gg 2 = 11111110_2(-2_{10})$$

# C 언어의 시프트 연산

- 오른쪽 시프트와 왼쪽 시프트 연산을 지원
- $n$ -비트 수  $x$ 의 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 <sub>10</sub> )
+		0	1	0	1 (5 <sub>10</sub> )
	0	1	1	1	0 (14 <sub>10</sub> )

받아올림	1	1	1	1	
		1	0	1	1 (11 <sub>10</sub> )
+		0	1	1	1 (7 <sub>10</sub> )
	1	0	0	1	0 (18 <sub>10</sub> )



# 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
- $0 \leq x + y < 2^n$ 
  - $(x + y) \bmod 2^n = x + y$
- $2^n \leq x + y \leq 2^{n+1} - 2$  : 오버플로우
  - $(x + y) \bmod 2^n = (x + y) - 2^n$

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

## 2의 보수 표현의 덧셈

- 두 정수  $x$ 와  $y$ 를  $n$ -비트 2의 보수 표현으로 나타내었을 때
  - $b_x = x_{n-1}x_{n-2} \cdots x_0$ 와  $b_y = y_{n-1}y_{n-2} \cdots y_0$
- $x +_C^n y$ 
  - $b_x$ 와  $b_y$ 를 부호 없는 이진 표현으로 취급하여 모듈로- $2^n$  덧셈을 수행한 다음, 그 결과를 다시 2의 보수 표현으로 해석하는 것
- $f_{U2C}^n(x)$ 
  - $x$ 를 부호 없는 이진수로 나타내었을 때의 이진 표현  $x_{n-1}x_{n-2} \cdots x_0$ 을 2의 보수 표현으로 해석하였을 때의 값

$$x +_C^n y = f_{U2C}^n \left( f_{B2U}^n(b_x) +_U^n f_{B2U}^n(b_y) \right)$$

## 2의 보수 표현의 덧셈

$$f_{U2C}^n(x) = x - x_{n-1} \cdot 2^n = \begin{cases} x, & 0 \leq x \leq 2^{n-1} - 1 \\ x - 2^n, & 2^{n-1} \leq x \leq 2^n - 1 \end{cases}$$

$$\begin{aligned} x +_C^n y &= f_{U2C}^n \left( f_{B2U}^n(b_x) +_U^n f_{B2U}^n(b_y) \right) \\ &= f_{U2C}^n \left( \left( f_{B2U}^n(b_x) + f_{B2U}^n(b_y) \right) \bmod 2^n \right) \\ &= f_{U2C}^n \left( \left( f_{B2C}^n(b_x) + x_{n-1} \cdot 2^n + f_{B2C}^n(b_y) + y_{n-1} \cdot 2^n \right) \bmod 2^n \right) \\ &= f_{U2C}^n \left( \left( f_{B2C}^n(b_x) + f_{B2C}^n(b_y) \right) \bmod 2^n \right) \\ &= f_{U2C}^n \left( (x + y) \bmod 2^n \right) \end{aligned}$$

## 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	$(7_{10})$
+		0	1	1	$(7_{10})$
		1	1	1	0

MSB로 들어가는 받아올림	$x$ 의 부호 비트	$y$ 의 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

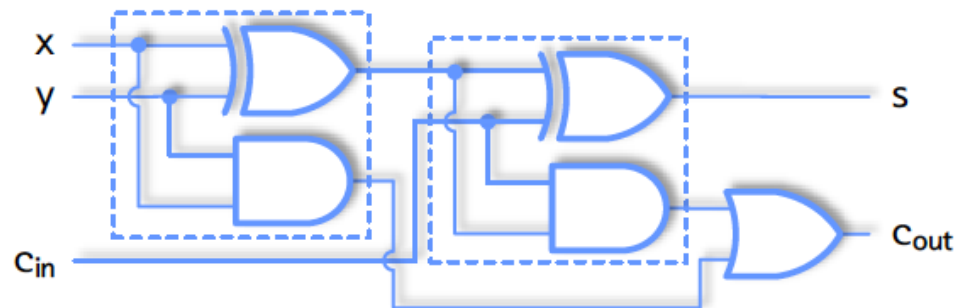
## 2의 보수 표현의 덧셈

$$\begin{aligned}
 x +_C^n y &= f_{U2C}^n(f_{B2U}^n(b_x) +_U^n f_{B2U}^n(b_y)) \\
 &= \begin{cases} x + y - 2^n, & 2^{n-1} \leq x + y \text{ (오버플로우)} \\ x + y, & -2^{n-1} \leq x + y < 2^{n-1} \\ x + y + 2^n, & x + y < -2^{n-1} \text{ (오버플로우)} \end{cases}
 \end{aligned}$$

# 전가산기와 반가산기

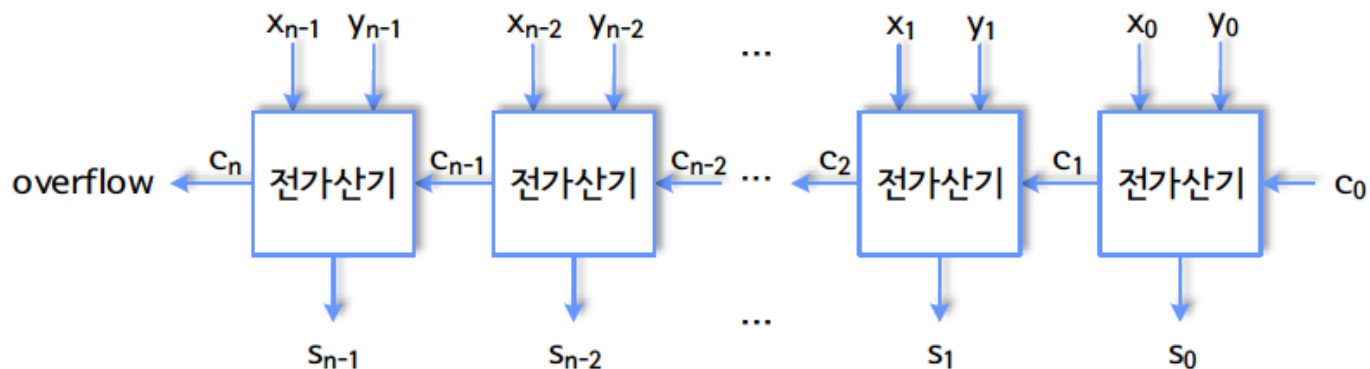
- 반가산기
  - 두 개의 비트  $x$ 와  $y$ 를 더하여 그 합  $s$ 와 받아올림(carry)  $c$ 를 출력
- 전가산기
  - 세 개의 비트  $x$ ,  $y$ ,  $c_{in}$ 을 입력으로 받아들이며 이들의 합을 한 개의 비트  $s$ 와 받아올림  $c_{out}$ 으로 출력

$x$	$y$	$c_{in}$	$s$	$c_{out}$
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



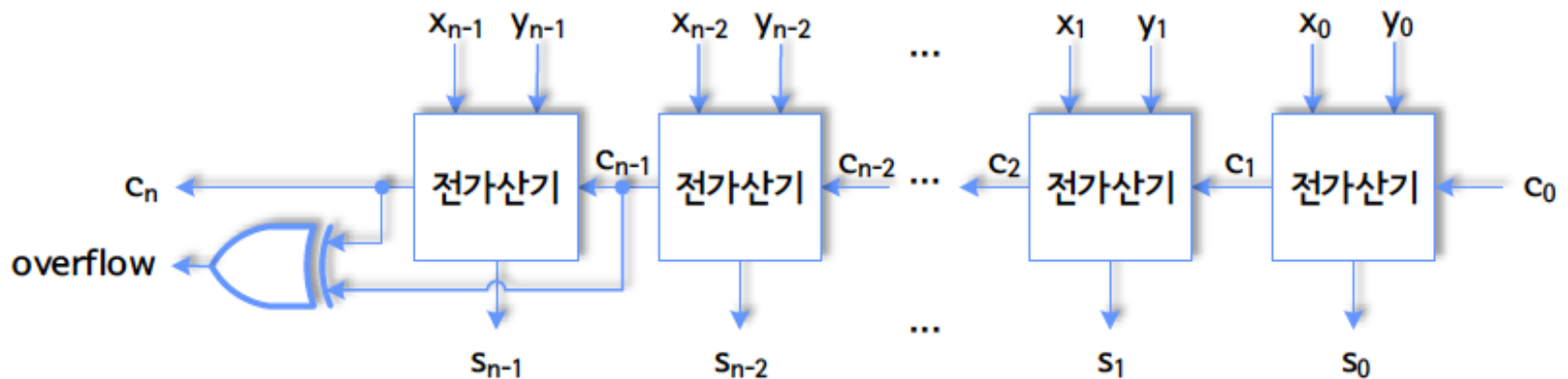
# 부호 없는 수의 덧셈 하드웨어

- $n$ -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
  - $c_n$ 이 1 이면 오버플로우



## 2의 보수 표현의 덧셈 하드웨어

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





# 이진 정수의 뺄셈

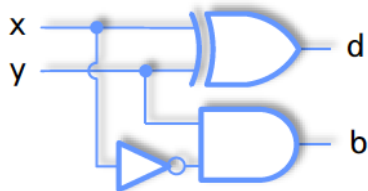
- 부호 없는 수
  - 십진수의 뺄셈과 그 방식이 같음
- 2의 보수 표현
  - 2의 보수 표현으로 표현된 두 정수  $x$ 와  $y$  뺄셈  $x - y$ 는  $x - y = x + (-y)$  이므로 감수(減數)  $y$ 의 2의 보수를 취하여 덧셈으로 변경
  - $x + (-y) = x + y' + 1$

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

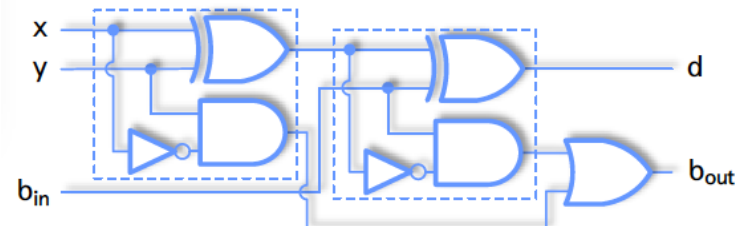
# 반가산기와 전감산기

- 반감산기(半減算器, half subtractor)
  - 두 개의 비트  $x$ 와  $y$ 를 입력으로 받아들이며  $x - y$ 을 계산한 결과를 차  $d$ 와 받아내림(borrow)  $b$ 로 출력
- 전감산기(全減算器, full subtractor)
  - 세 개의 비트  $x$ ,  $y$ ,  $b_{in}$ 을 입력으로 받아들이며  $x - y - b_{in}$ 을 계산한 결과를 한 개의 비트  $d$ 와 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

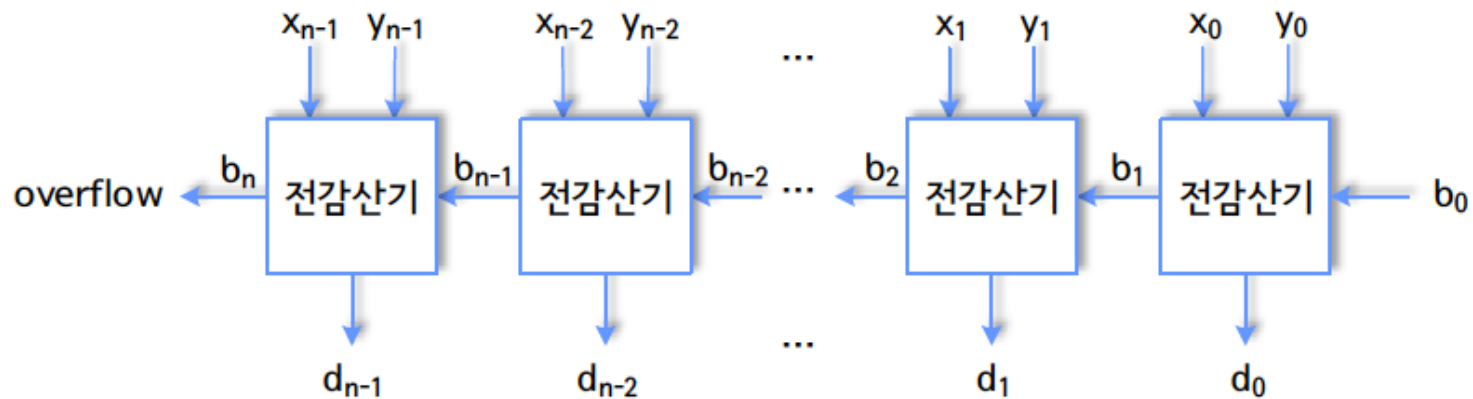


x	y	$b_{in}$	d	$b_{out}$
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1



# 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 뱀셈 하드웨어

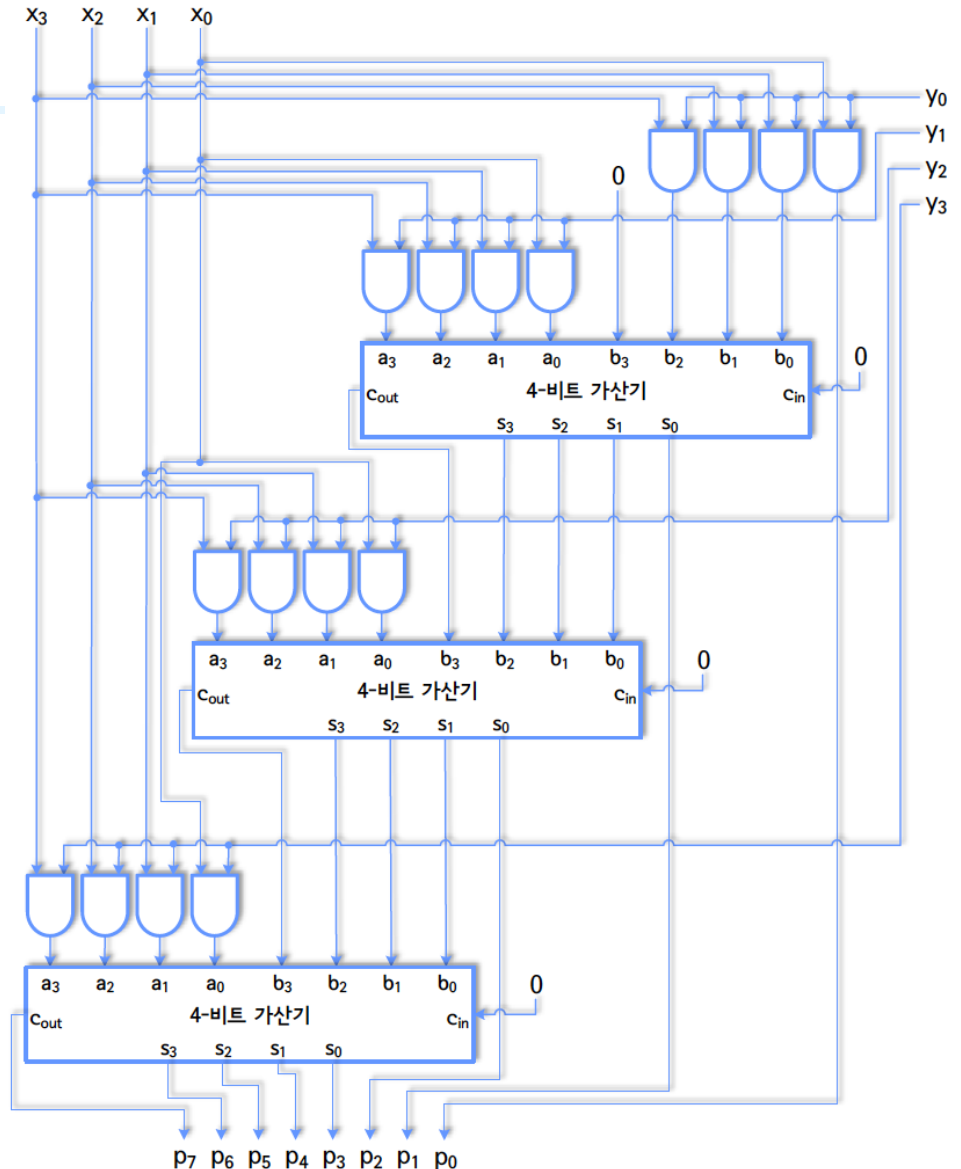
# 부호 없는 수의 곱셈

- 십진수의 곱셈과 그 원리가 같음

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

# 부호 없는 수의 곱셈

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



## 2의 보수 표현의 곱셈

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

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

## 2의 보수 표현의 곱셈

- 부분적인 곱셈 결과를 나타내기 위해 부호확장이 필요

1					$x_3$	$x_2$	$x_1$	$x_0$
2	$\times$				$y_3$	$y_2$	$y_1$	$y_0$
3		$x_3 \cdot y_0$	$x_3 \cdot y_0$	$x_3 \cdot y_0$	$x_3 \cdot y_0$	$x_2 \cdot y_0$	$x_1 \cdot y_0$	$x_0 \cdot y_0$
4	+	$x_3 \cdot y_1$	$x_3 \cdot y_1$	$x_3 \cdot y_1$	$x_3 \cdot y_1$	$x_2 \cdot y_1$	$x_1 \cdot y_1$	$x_0 \cdot y_1$
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
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
7		$p_7$	$p_6$	$p_5$	$p_4$	$p_3$	$p_2$	$p_1$

1					$x_3$	$x_2$	$x_1$	$x_0$
2	$\times$				$y_3$	$y_2$	$y_1$	$y_0$
3		$x_3 \cdot y_0$	$x_3 \cdot y_0$	$x_3 \cdot y_0$	$x_3 \cdot y_0$	$x_2 \cdot y_0$	$x_1 \cdot y_0$	$x_0 \cdot y_0$
4	+	$x_3 \cdot y_1$	$x_3 \cdot y_1$	$x_3 \cdot y_1$	$x_3 \cdot y_1$	$x_2 \cdot y_1$	$x_1 \cdot y_1$	$x_0 \cdot y_1$
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
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
7	+	0	0	0	0	0	0	1
8		$p_7$	$p_6$	$p_5$	$p_4$	$p_3$	$p_2$	$p_1$

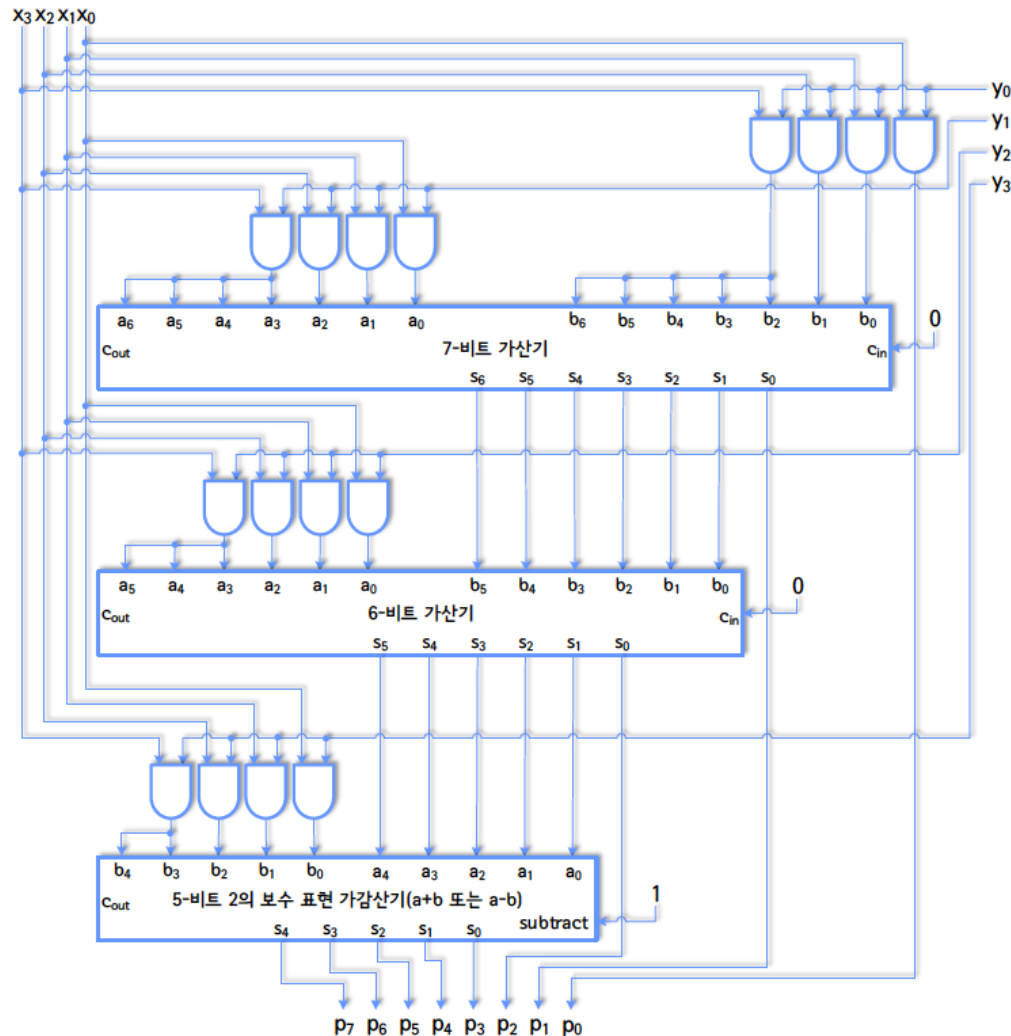


# 2의 보수 표현의 곱셈 하드웨어

1					$x_3$	$x_2$	$x_1$	$x_0$	
2	×				$y_3$	$y_2$	$y_1$	$y_0$	
3		$x_3 \cdot y_0$	$x_3 \cdot y_0$	$x_3 \cdot y_0$	$x_3 \cdot y_0$	$x_2 \cdot y_0$	$x_1 \cdot y_0$	$x_0 \cdot y_0$	
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		$x_3 \cdot y_0$	$x_3 \cdot y_0$	$x_3 \cdot y_0$	$x_3 \cdot y_0$	$x_2 \cdot y_0$	$x_1 \cdot y_0$	$x_0 \cdot y_0$	
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

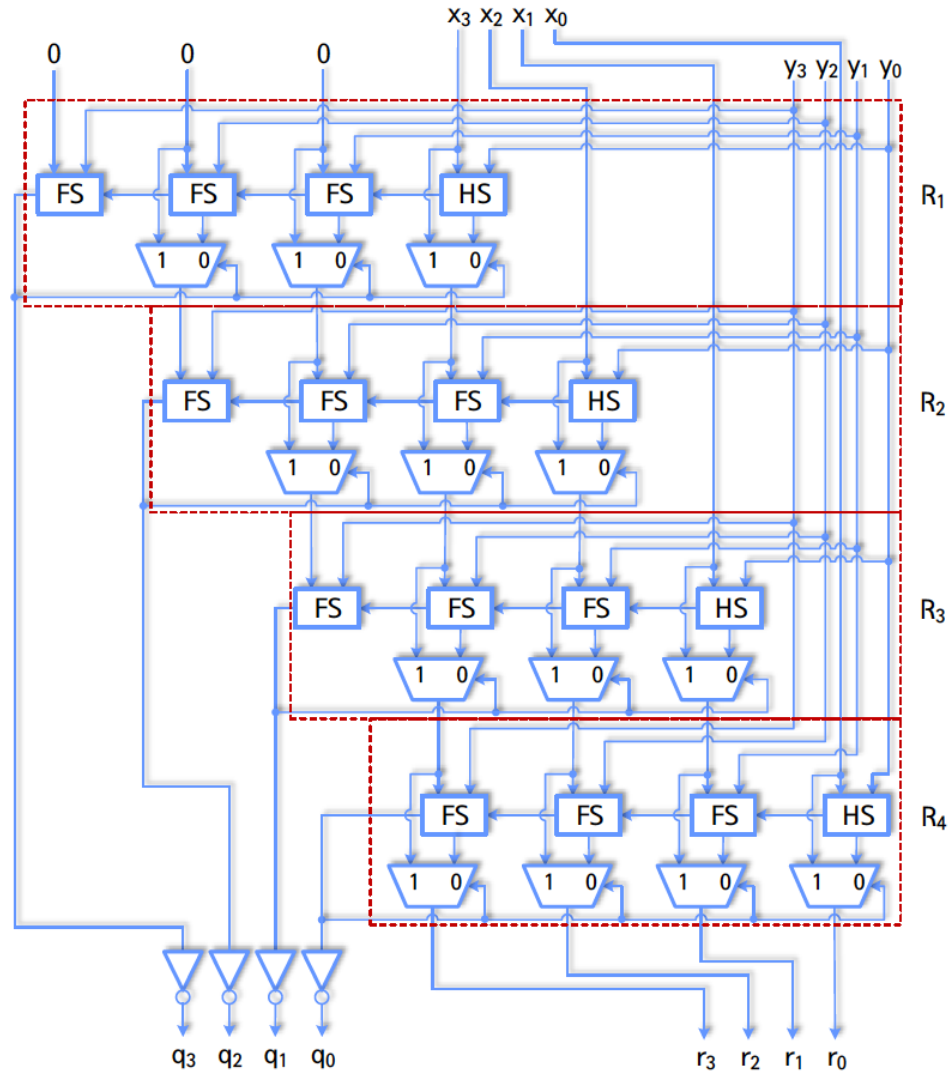
$$\begin{array}{cccccccccc}
 & & & & & & 1 & 1 & (3_{10}) \\
 (4_{10}) & 1 & 0 & 0 & & 1 & 1 & 0 & 1 & (13_{10}) \\
 & & & & - & 1 & 0 & 0 & & \\
 & & & & & 0 & 1 & 0 & 1 & \\
 & & & & - & & 1 & 0 & 0 & \\
 & & & & & & 0 & 0 & 1 & 
 \end{array}$$

# 부호 없는 이진수의 나눗셈

- $n$ -비트 부호 없는 이진수  $x$ 를  $y$ 로 나누어 몫  $q$ 와 나머지  $r$ 을 구하는 과정은  $R_0 = x$ 로 놓은 후 매 단계마다 부분적인 나머지(partial remainder)  $R_i$  ( $1 \leq i \leq n$ )를 계산하는 과정
  - $R_i = R_{i-1} - q_{n-i} \cdot y \cdot 2^{n-i}$
- $q_{n-i} = 1$ 을 가정하고  $R_i$ 를 계산하여  $R_i \geq 0$ 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	$R_0 = x = 13_{10}$		
					−	$q_3 0$	$q_3 1$	$q_3 0$	$q_3 0$	0	0	$q_3 \cdot y \cdot 2^3$	
						1	1	0	0			$R_1 = R_0 - y \cdot 2^3$	
						0	0	0	1	1	0	$R_1 = R_0$	
					−		$q_2 0$	$q_2 1$	$q_2 0$	$q_2 0$	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	$R_2 = R_1$	
					−			$q_1 0$	$q_1 1$	$q_1 0$	$q_1 0$	$q_1 \cdot y \cdot 2^1$	
								0	0	1	0	$R_3 = R_2 - y \cdot 2^1$	
					−				$q_0 0$	$q_0 1$	$q_0 0$	$q_0 0$	$q_0 \cdot y \cdot 2^0$
									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  $x$ 를  $y(\neq 0)$ 로 나눈 몫  $q$ 와 나머지  $r$ 을 구하려면
  - 우선  $|x|$ 를  $|y|$ 로 나눈 몫  $q'$ 과 나머지  $r'$ 을 부호 없는 수의 나눗셈을 이용하여 구함
  - 그런 다음  $x$ 와  $y$ 의 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  $x$ 와 제수  $y$ 의 부호에 따라  $|y|$ 가  $|x|$ 에 들어 있는 개수  $q'$ 을 계산
  - $x \geq 0, y \geq 0 : r \geq 0$  일 동안  $y$ 를  $r$ 에서 연속적으로 빼서
  - $x \geq 0, y < 0 : r \geq 0$  일 동안  $y$ 를  $x$ 에 연속적으로 더하여
  - $x < 0, y \geq 0 : r \leq 0$  일 동안  $y$ 를  $x$ 에 연속적으로 더하여
  - $x < 0, y < 0 : r \leq 0$  일 동안  $y$ 를  $x$ 에서 연속적으로 빼서
- $x$ 와  $y$ 의 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  $x$ 와  $y$ 가 주어졌을 때,  $x$ 를  $y$ 로 나누어 몫  $q$ 와 나머지  $r$ 을 구하는 과정은,  $R_0 = x$ 로 놓은 다음 매 단계마다 부분적인 나머지  $R_i$  ( $1 \leq i \leq n$ )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  $R_i$ 를 계산한 다음,  $R_{i-1}$ 과  $R_i$ 의 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
  - $y$ 가  $R_{i-1}$ 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	$q'_3 \cdot y \cdot 2^3$
						$\theta$	$\theta$	1	$\theta$	$\theta$	1	$R_1 = R_0 + y \cdot 2^3$
						1	1	1	1	0	0	$R_1 = R_0$
					+	$q'_2 0$	$q'_2 0$	$q'_2 0$	$q'_2 1$	$q'_2 1$	0	$q'_2 \cdot y \cdot 2^2$
						$\theta$	$\theta$	$\theta$	$\theta$	1	$\theta$	$R_2 = R_1 + y \cdot 2^2$
						1	1	1	1	0	0	$R_2 = R_1$
					+	$q'_1 0$	$q'_1 0$	$q'_1 0$	$q'_1 0$	$q'_1 1$	$q'_1 1$	$q'_1 \cdot y \cdot 2^1$
						1	1	1	1	1	1	$R_3 = R_2 + 2^1$
					+	$q'_0 0$	$q'_0 0$	$q'_0 0$	$q'_0 0$	$q'_0 0$	$q'_0 1$	$q'_0 \cdot y \cdot 2^0$
						$\theta$	$\theta$	$\theta$	$\theta$	$\theta$	1	$R_4 = R_3 + y \cdot 2^0$
						1	1	1	1	1	1	$R_4 = R_3$
								$r_3$	$r_2$	$r_1$	$r_0$	

# 2의 보수 표현의 나눗셈 하드웨어

