🐱

Ch.4 Combinational Logic - part B

Subtraction

negative numbers → A - B = A + (-B)

음수화해서 add

One’s complement

모든 bit 반대

representation

$1101_2 = 13_{10}$ (a 4-bit unsigned number)

$0\ 1101 = +13_{10}$ (a positive number in 5-bit one 's complement)

$1\ 0010 = -13_{10}$ (a negative number in 5-bit one 's complement)

sign도 포함하여, 보수로 바꿔줌

- sign을 포함하여 1의 보수로 바꿔줌 → 모든 bit 반대로

addition

0111	(+7)	0011	(+3)
+ 1011	+ (-4)	+ 0010	+ (+2)
1 0010		0 0101	
0010		0101	
+ 1		+ 0	
0011	(+3)	0101	(+5)

⇒ 1의보수, + signbit

→ carry

1. 부호 없이 덧셈 (sign bit 포함)
2. sum += carry out

⇒ 항상 음수는 어떻게 처리할 것인지 약속해야 함.

Two's complement

1의 보수 + 1

OR 2의 n승 - (구하고자 하는 수)

OR 첫번째 1 나온 이후로만 1의 보수 적용 ex) 0101 ⇒(2의 보수) 1011

representation

$0100_2 = 4_{10}$ (a 4-bit unsigned number)

$0\,0100 = +4_{10}$ (a positive number in 5-bit two 's complement)

$1\,1011 = -4_{10}$ (a negative number in 5-bit *ones'* complement)

$1\,1100 = -4_{10}$ (a negative number in 5-bit two 's complement)

+1

addition

For example, to find 0111 + 1100, or (+7) + (-4):

- First add 0111 + 1100 as unsigned numbers:

$$\begin{array}{r} 0111 \\ +\ 1100 \\ \hline 10011 \end{array}$$

- Discard the carry out (1).
- The answer is 0011 (+3).

1. 부호 없이 덧셈 ⇒ 이미 including sign bit
2. carry out 무시

→ **why?**

1. n- bit number에서, 2의 보수로의 B의 음수 = 2의 ^n - B
2.

$$A - B = A + (-B)$$ ☆

$$\Rightarrow A + (2^n - B) = (A - B) + 2^n = 2^n - [-(A-B)]$$

3. A ≥ B

   ⇒ A-B가 양수가 되기에 2의 n승은 carry out이 1임.

   ⇒ carry out을 무시하는 것은 2의 n승에서 빼는 거랑 마찬가지

   ⇒ 무시해도 됨!

4. A < B

   ⇒ A-B가 양수가 되기에 → 2의 n승 - [-(A-B)]

   ⇒ -(A-B)가 2의 보수 형태임 (이미 포함)

   ⇒ 무시해도 됨!

Subtraction Circuit

making

부호 없는 adders 만들었던 것과 비슷

→ 2의 보수를 사용하기 위해서는 모든 뺄셈 문제를 덧셈 문제로 바꿔야 함

→ A-B = A +(-B)

A two's complement subtraction circuit

ch.3에서 배웠던 adder를 이용하여 제작

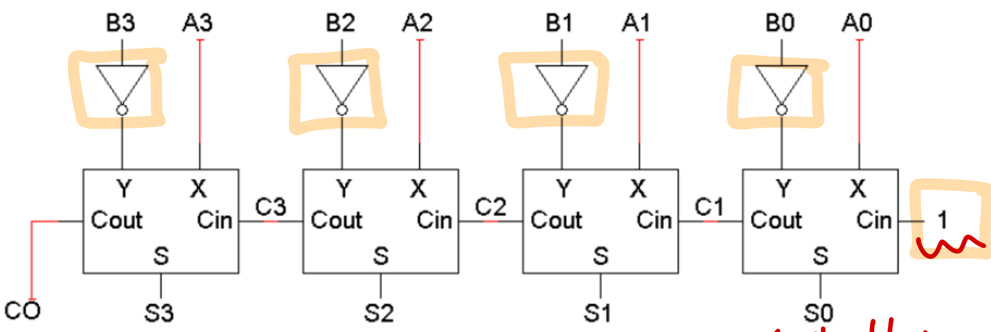
- need

  1. B(빼는 수)의 1의 보수(Complement each bit)
  2. carry in input = 1

  ⇒ A + B' + 1 = A + B''(2의 보수)

  1의 보수 2의 보수

1의보수 ⇒ inverter



Fulladder

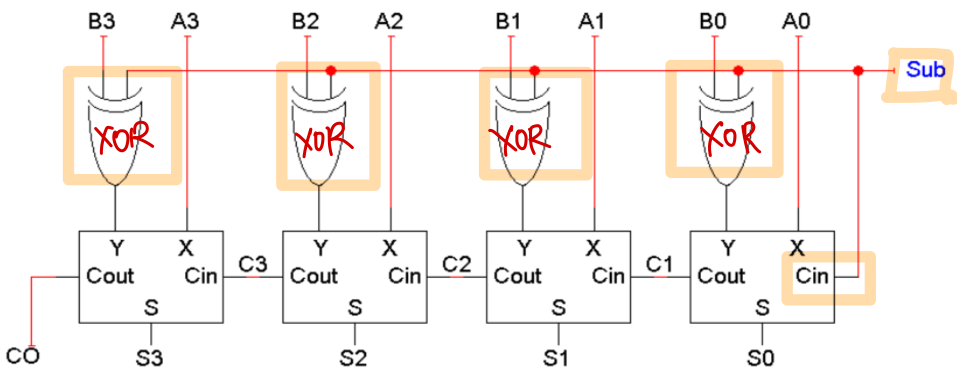
- small differences
  1. negate B3, B2, B1, B0.(빼는 수) inverter!
  2. sets the initial carry in to 1, instead of 0

[inverter → XOR
cin=1 → cin=sub.

An adder-subtractor circuit(가감산기)

sub input 추가 후 B(빼는 수)에 inverter 대신 XOR gate를 사용하여 sub에 따라 뺄셈, 덧셈 수행

1. **sub = 0** ⇒ cin = 0으로 덧셈 수행
2. **sub = 1** ⇒ cin = 1으로 B가 2의 보수가 되어 뺄셈 수행 → B0~B3 + 1
   ↳ inverter랑 같은 결과



Detecting signed overflow

(ch.3에 나오는 overflow랑 다른 이야기임)

- 가장 쉬운 방법 : 모든 부호 비트 체크하기

Carry 01

  0100 (+4)
+ 0101 (+5)
~~0~~1001 (-7)

2의보수 ⇒ 1111

Carry 10

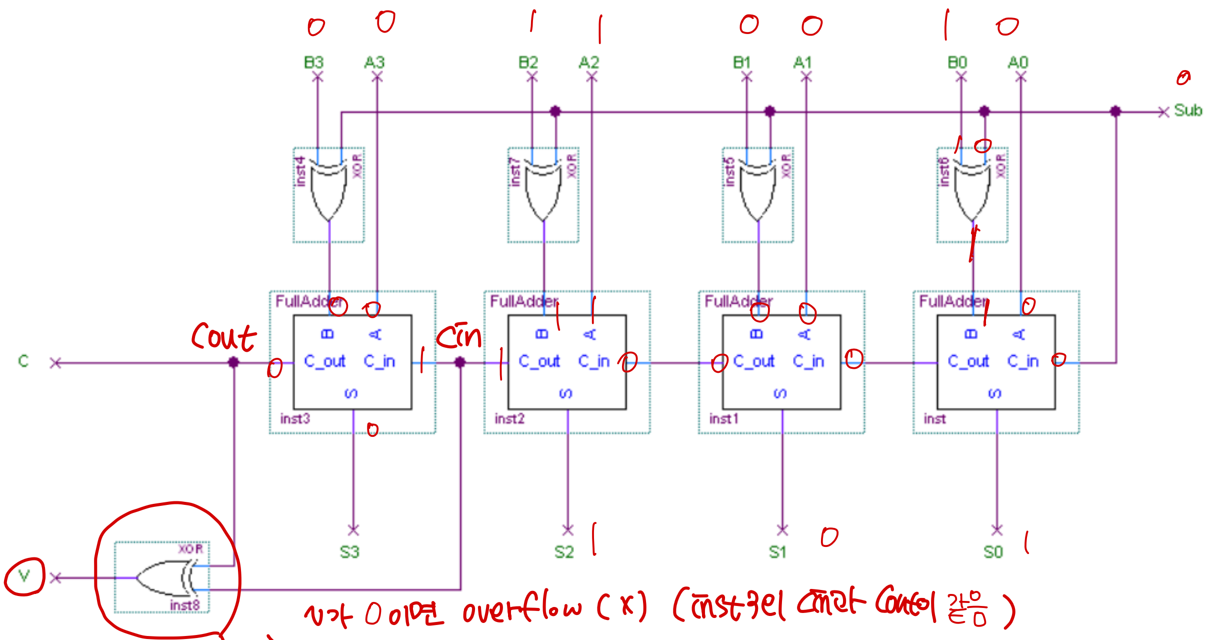
  1100 (-4)
+ 1011 (-5)
~~1~~0111 (+7)

- over flow occurs only in the two situations(연산에서 절댓값이 7 이상이면 overflow)

1. 두 양수를 더했는데 음수 결과가 나왔을 때
2. 두 음수를 더했는데 양수 결과가 나왔을 때

⇒ 양수 + 음수 했을 때는 안 일어남

- how to detect → cin과 cout of the sign bit position이 같으면 overflow 일어나지 않음.



V가 0이면 overflow (X) (inst3의 cin과 cout이 같음)

V가 1이면 overflow (O) (〃 다름)

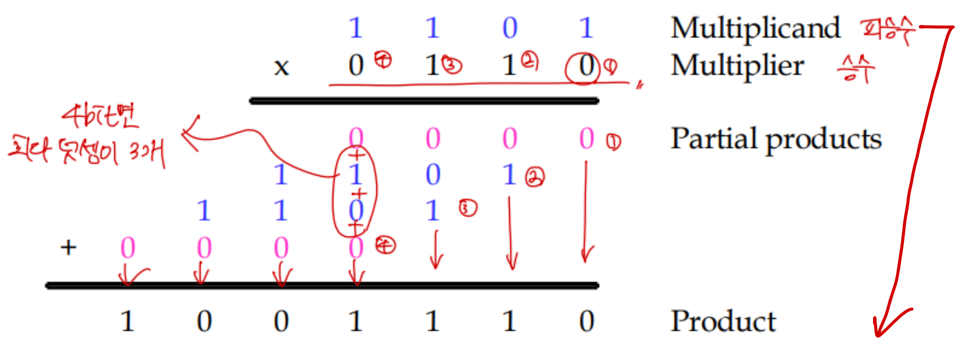
Multiplication

repeated addition

→ adders가 있으면 할 수 있음

- multipliers are very complex circuits
    - n by n multiplier → n개의 partitial products와 n-1 adders(m bits)
    - 32 bit, 64 bit가 되면 엄청 커짐

Binary multiplication example



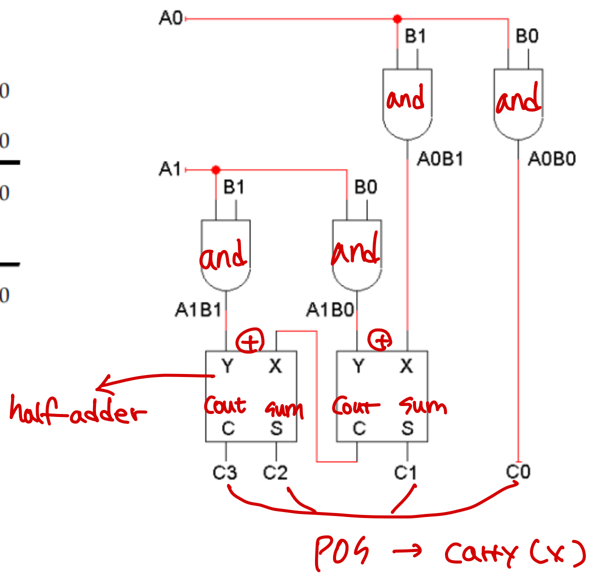
1101

- 0과 1의 multiply에서 partial products(부분합)은 항상 0000 or multiplicand(피승수)
- 이 example에서는 총 4개의 partial products가 필요함 → 즉 3개의 “ + “

a 2 x 2 binary multiplier

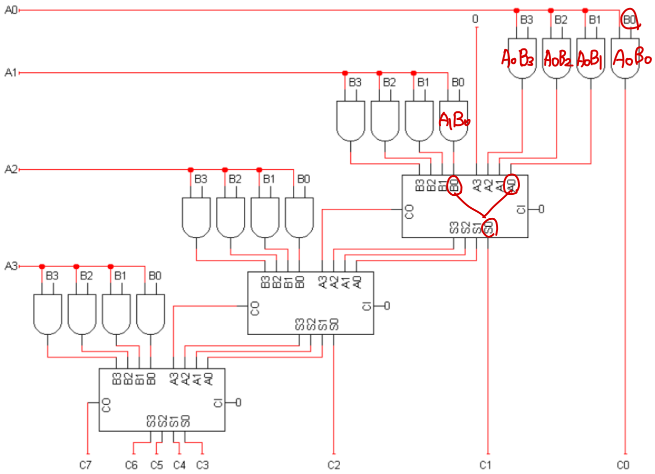
- partial products ⇒ AND gates
- 2 bit + 2bit ⇒ two half adders(full adders)로 구현

		B_1	B_0
	x	A_1	A_0
	Carry	A_0B_1	A_0B_0
+ Carry	A_1B_1	A_1B_0	
C_3	C_2	C_1	C_0



a 4 x 4 binary multiplier

				B_3	B_2	B_1	B_0
			X	A_3	A_2	A_1	A_0
				A_0B_3	A_0B_2	A_0B_1	A_0B_0
			A_1B_3	A_1B_2	A_1B_1	A_1B_0	
		A_2B_3	A_2B_2	A_2B_1	A_2B_0		
	A_3B_3	A_3B_2	A_3B_1	A_3B_0			
C_7	C_6	C_5	C_4	C_3	C_2	C_1	C_0



- 8 bit result
- 만약 4by4 multiplier에서 4bit 결과값이 필요하다면
  - C4 - C7을 무시하고 4bit보다 길어진 결과면 overflow 조건이 된다는 것을 고려할 것

a special case

- in decimal(10의 n승)
  - 128 X 10 = 1280

- 왼쪽 끝에 0을 하나 더 넣으면 결과값

- in binary(2의 n승)
  - 11 X 10 = 110
  - 11 X 100 = 1100
  - 110 / 10 = 11

Magnitude Comparator

크기를 비교하는 회로

MSB LSB

$A = A_3A_2A_1A_0$

$B = B_3B_2B_1B_0$

모든 i에 대해서 x가 1이 나오면 A=B

$x_i = A_iB_i + A'_iB'_i$

$= \overline{A \oplus B}$

↳ 같으면 1, 다르면 0 (equivalance 회로)

① $(A=B) = x_3x_2x_1x_0$ → 전부 다 1이면

② $(A>B) = A_3B_3' + x_3A_2B_2' + x_3x_2A_1B_1' + x_3x_2x_1A_0B_0'$

A₃>B₃ 이면 A>B

③ $(A<B) = A_3'B_3 + x_3A_2'B_2 + x_3x_2A_1'B_1 + x_3x_2x_1A_0'B_0$