Number Representation and Arithmetic Circuits

VOLKAN KURSUN

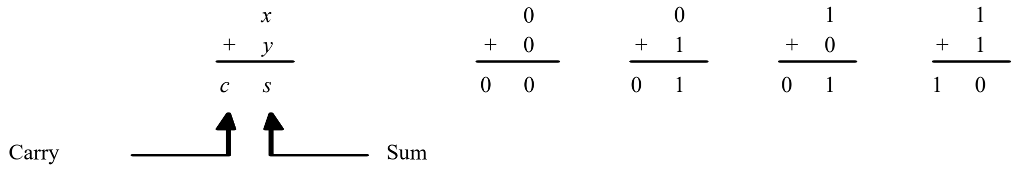
Some material from McGraw Hill

Outline

- Addition of Unsigned Numbers
- Arithmetic Overflow (Unsigned)
- Signed Numbers
- Addition of Signed Numbers
- Subtraction of Signed Numbers
- Arithmetic Overflow (Signed)

One-Bit Addition

- Four possible input combinations and three possible results



(a) The four possible cases

x	y	Carry c	Sum s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

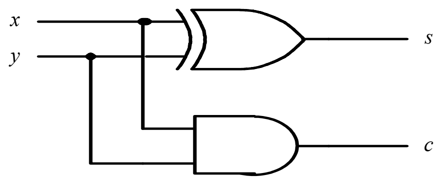
(b) Truth table

Half Adder

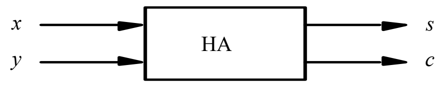
- Half adder: the circuit used for adding two bits

x	y	Carry c	Sum s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

(b) Truth table



(c) Circuit



(d) Graphical symbol

Multi-Bit Addition

- Example: add two 5-bit unsigned numbers
- With 5-bits, unsigned numbers in the range of 0 to 2^5-1 (31) can be represented

Generated carries $\longrightarrow$ 1 1 1 0

$$\begin{array}{rrr} X = x_4x_3x_2x_1x_0 & 01111 & (15)_{10} \\ +Y = y_4y_3y_2y_1y_0 & +01010 & +(10)_{10} \\ \hline S = s_4s_3s_2s_1s_0 & 11001 & (25)_{10} \end{array}
\qquad
\begin{array}{cccc} \dots & c_{i+1} & c_i & \dots \\ \dots & \dots & x_i & \dots \\ \dots & \dots & y_i & \dots \\ \hline \dots & \dots & s_i & \dots \end{array}$$

- At each bit position i, there may be a carry-in coming from the previous bit position i-1

Multi-Bit Addition

- At each bit position i, addition operation requires addition of three bits: x_i, y_i, and c_i
- **For multi-bit addition**, design a logic circuit that has 3 inputs (c_i, x_i, and y_i) and 2 outputs (c_{i+1} and s_i)

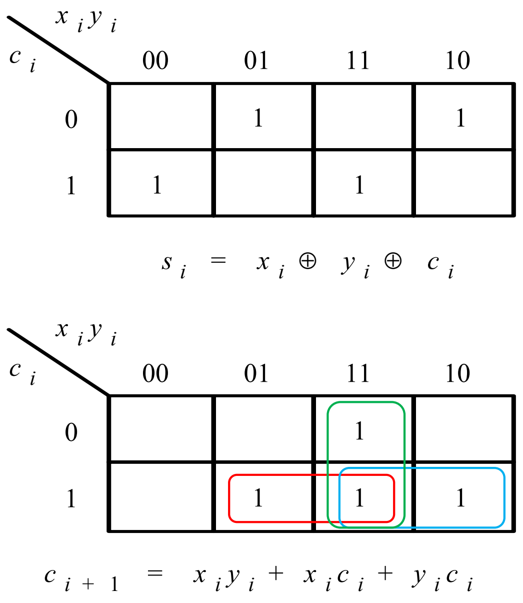
Generated carries $\longrightarrow$ 1 1 1 0

$$\begin{array}{rrr} X = x_4x_3x_2x_1x_0 & 01111 & (15)_{10} \\ +Y = y_4y_3y_2y_1y_0 & +01010 & +(10)_{10} \\ \hline S = s_4s_3s_2s_1s_0 & 11001 & (25)_{10} \end{array}
\qquad
\begin{array}{cccc} \dots & c_{i+1} & c_i & \dots \\ \dots & \dots & x_i & \dots \\ \dots & \dots & y_i & \dots \\ \hline \dots & \dots & s_i & \dots \end{array}$$

Full Adder

- 3 inputs (c_i, x_i, and y_i) and 2 outputs (c_{i+1} and s_i)

c_i	x_i	y_i	c_{i+1}	s_i
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



$c_i \backslash x_iy_i$	00	01	11	10
0		1		1
1	1		1	

$s_i = x_i \oplus y_i \oplus c_i$

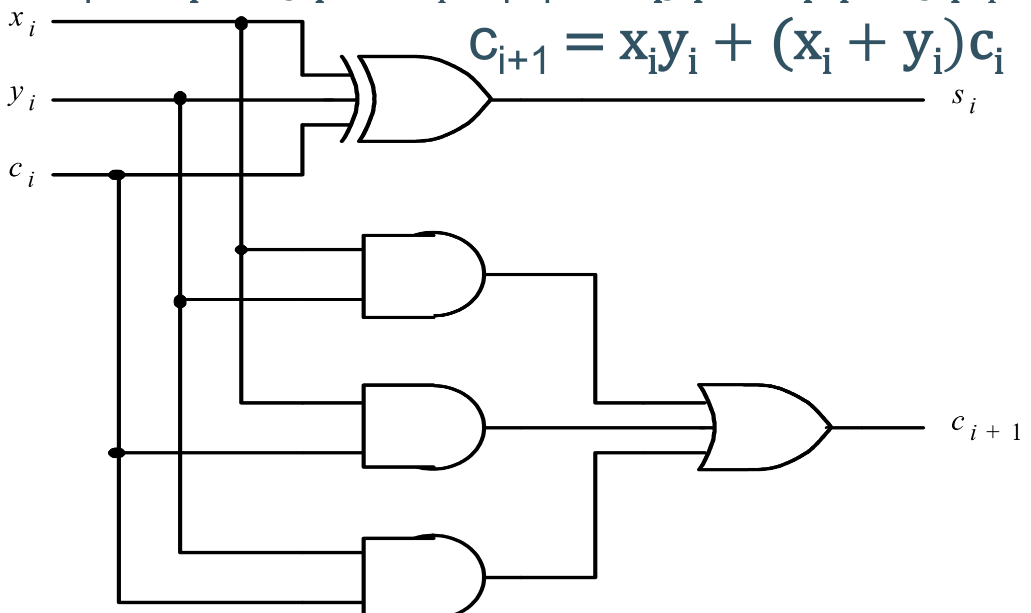
$c_i \backslash x_iy_i$	00	01	11	10
0			1	
1		1	1	1

$c_{i+1} = x_iy_i + x_ic_i + y_ic_i$

Full Adder Implementation

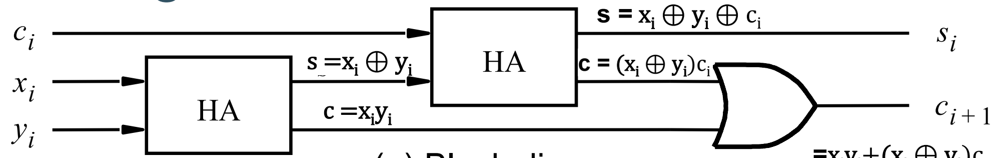
- $s_i = x_i \oplus y_i \oplus c_i$, $c_{i+1} = x_iy_i + x_ic_i + y_ic_i$

$c_{i+1} = x_iy_i + (x_i + y_i)c_i$

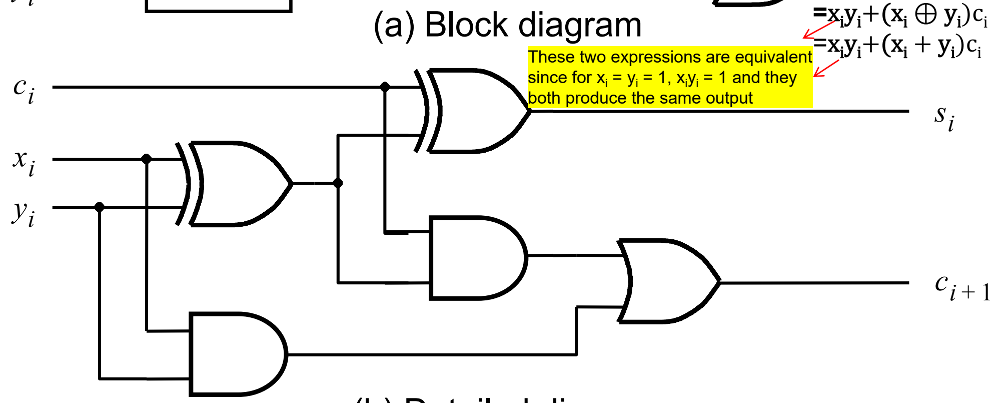


Decomposed Full Adder Implementation

❑A full adder can be constructed using half adders



(a) Block diagram

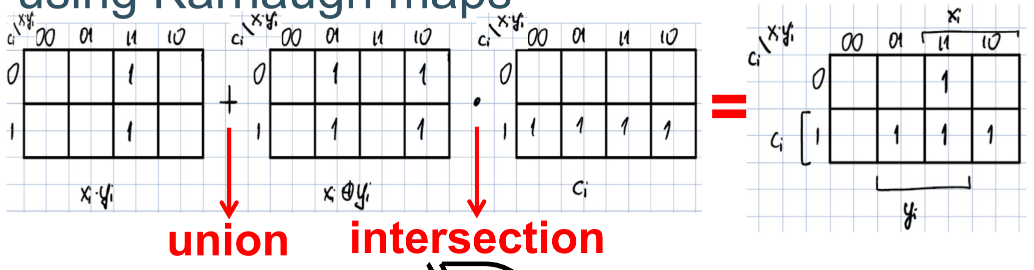


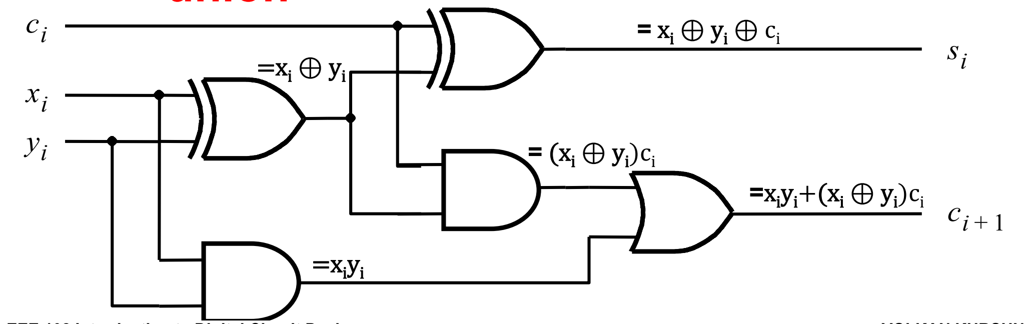
These two expressions are equivalent since for $x_i = y_i = 1$, $x_iy_i = 1$ and they both produce the same output

(b) Detailed diagram

Decomposed Full Adder Implementation

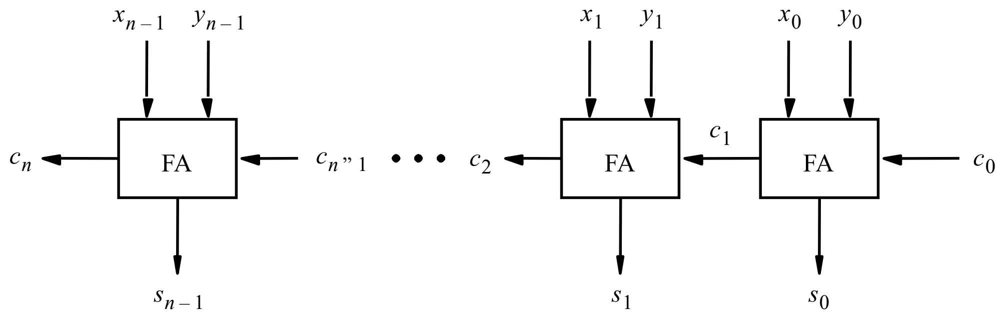
Prove $c_{i+1} = x_iy_i + x_ic_i + y_ic_i = x_iy_i+(x_i \oplus y_i)c_i$ using Karnaugh maps





Ripple Carry Adder

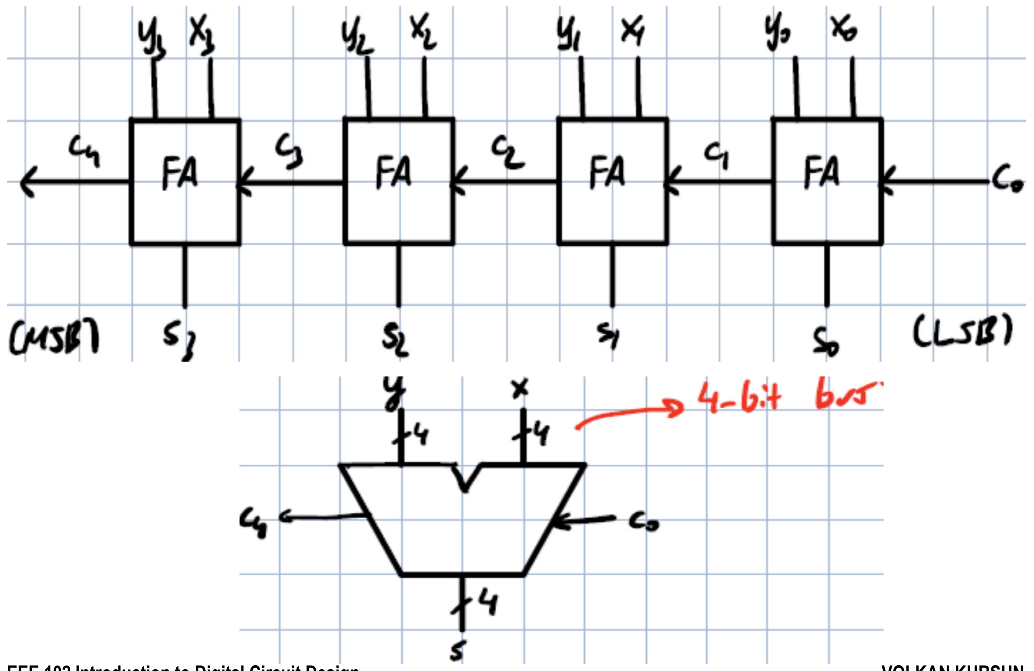
❑An n-bit adder can be formed by using n full adders: the carry signals ripple through the full adder stages from right to left



MSB position LSB position

❑Quite slow if n is high but a ripple carry adder is easy to design and compact

4-Bit Ripple Carry Adder



4-Bit Adder VHDL Code

```
library ieee;
use ieee.std_logic_signed.all

entity addnibble is
   port(a,b: in std_logic_vector(3 downto 0);
        y: out std_logic_vector(3 downto 0));
end addnibble;

architecture addarch of addnibble is
begin
   y <= a+ b;
end addarch;
```

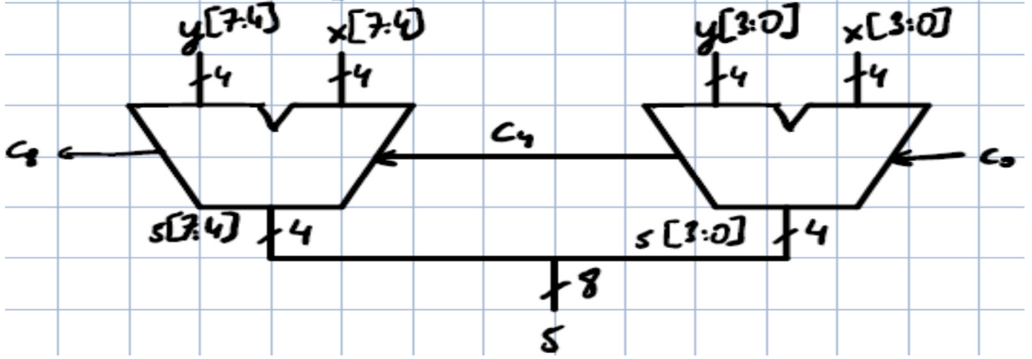
std_logic_signed package: allows STD_LOGIC signals to be used with arithmetic operators

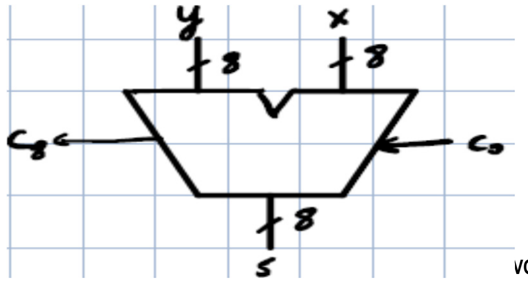
std_logic_vector type: **array of STD_LOGIC data objects**

std_logic_vector size: **range of array indices**

8-Bit Ripple Carry Adder

- Can be designed with two 4-bit adders





Outline

- Addition of Unsigned Numbers
- Arithmetic Overflow (Unsigned)
- Signed Numbers
- Addition of Signed Numbers
- Subtraction of Signed Numbers
- Arithmetic Overflow (Signed)

Unsigned Integers: Addition

- Addition is just like in elementary school. Add bits in each position and carry 1s to more significant bit positions

```
   00111      7
 + 00110    + 6
 --------------
   01101     13
```

- Assume an n-bit architecture with n-bit source operands provided by n-bit registers and the result will be stored in an n-bit register
- **Range of unsigned integers** that can be represented with n-bits: 0 to $+2^n-1$
- **OVERFLOW:** if an addition operation produces a result that cannot be correctly represented with n-bits (**result > $+2^n$-1**), an overflow occurs

Unsigned Integers: Addition

- Examples: **assume an architecture with 5-bit registers** and operands. **Range of unsigned integers** that can be represented with 5-bits: 0 to $+2^5-1$ = **0 to +31**
- **OVERFLOW:** if an unsigned addition operation produces a result that cannot be correctly represented with 5-bits (**result > +31**) an overflow occurs

Cout_4	Cin_4	Cin_3	Cin_2	Cin_1	Cin_0	Decimal	
0	0	1	0	0	0	⟵ Carry bits	
	0	0	1	1	1	7	
	+ 1	0	1	0	0	+ 20	
0	1	1	0	1	1	27	Correct

Cout_4	Cin_4	Cin_3	Cin_2	Cin_1	Cin_0	Decimal	
1	1	1	1	1	0	⟵ Carry bits	
	0	1	1	1	1	15	
	+ 1	0	0	0	1	+ 17	
1	0	0	0	0	0	$\cancel{0}$	Overflow

Cout_4	Cin_4	Cin_3	Cin_2	Cin_1	Cin_0	Decimal	
0	0	0	0	0	0	⟵ Carry bits	
	0	1	1	1	1	15	
	+ 1	0	0	0	0	+ 16	
0	1	1	1	1	1	31	Correct

Cout_4	Cin_4	Cin_3	Cin_2	Cin_1	Cin_0	Decimal	
1	1	1	1	0	0	⟵ Carry bits	
	1	1	0	1	1	27	
	+ 0	1	1	1	0	+ 14	
1	0	1	0	0	1	$\cancel{9}$	Overflow

Unsigned Integers: Addition

- **OVERFLOW:** if an addition operation produces a result that cannot be correctly represented with 5-bits (**result > +31**) an overflow occurs
- **OBSERVATION:** In unsigned addition overflow is captured by the carry output from the most significant bit position. If Cout_4 = 1, **there is an overflow**. If Cout_4 = 0, there is **NO overflow**
- **Carry output from the most significant bit position** (Cout_(n-1)) can be used as the **overflow flag** for unsigned addition

Cout_4	Cin_4	Cin_3	Cin_2	Cin_1	Cin_0	Decimal	
0	0	1	0	0	0	⟵ Carry bits	
	0	0	1	1	1	7	
	+ 1	0	1	0	0	+ 20	
0	1	1	0	1	1	27	Correct

Cout_4	Cin_4	Cin_3	Cin_2	Cin_1	Cin_0	Decimal	
1	0	1	1	0	0	⟵ Carry bits	
	1	0	0	1	1	19	
	+ 1	0	1	1	0	+ 22	
1	0	1	0	0	1	$\cancel{9}$	Overflow

Cout_4	Cin_4	Cin_3	Cin_2	Cin_1	Cin_0	Decimal	
0	1	1	0	0	0	⟵ Carry bits	
	0	1	1	1	1	15	
	+ 0	1	1	0	0	+ 12	
0	1	1	0	1	1	27	Correct

Cout_4	Cin_4	Cin_3	Cin_2	Cin_1	Cin_0	Decimal	
1	1	1	1	0	0	⟵ Carry bits	
	1	1	0	1	1	27	
	+ 0	1	1	1	0	+ 14	
1	0	1	0	0	1	$\cancel{9}$	Overflow

Outline

- Addition of Unsigned Numbers
- Arithmetic Overflow (Unsigned)
- Signed Numbers
- Addition of Signed Numbers
- Subtraction of Signed Numbers
- Arithmetic Overflow (Signed)

Signed Integers (2's-Complement)

Most significant bit is the sign bit:

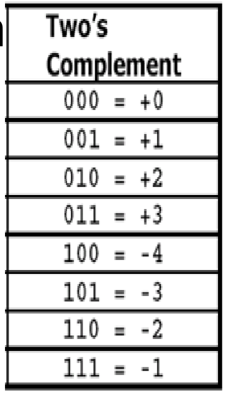
MSB = 0: positive integer

MSB = 1: negative integer

A number is negated by inverting all the bits and adding 1 to the inverted number: 2's complement representation

Signed Number Representation

- Two's complement representation
  - Used by modern computers
  - Example: 0b 101 = -3
- Properties
  - Only one zero
  - One more negative number than positive number
  - All negative numbers have a 1 in the most significant bit (MSB)



Two's Complement
000 = +0
001 = +1
010 = +2
011 = +3
100 = -4
101 = -3
110 = -2
111 = -1

Signed Integers (2s-Complement)

Decimal equivalent of an n-bit signed binary number:

$$x = -x_{n-1}2^{n-1} + x_{n-2}2^{n-2} + \cdots + x_1 2^1 + x_0 2^0$$

Range: -2^{n-1} to $+2^{n-1} - 1$

- Range with 32 bits: -2^{31} to $+2^{31} - 1$

  –2,147,483,648 to +2,147,483,647
- Example:

  0b 1111 1111 1111 1111 1111 1111 1111 1100

  $= -1\times2^{31} + 1\times2^{30} + \ldots + 1\times2^{2} + 0\times2^{1} + 0\times2^{0}$

  $= -2{,}147{,}483{,}648 + 2{,}147{,}483{,}644 = (-4)_{10}$

Signed Negation

- To negate a number: complement all digits and add 1

  Complement (or invert) means 1 → 0, 0 → 1

$$x + \overline{x} = 0b1111\ldots111 = -1$$

$$\overline{x} + 1 = -x$$

- Example: negate +2 (write -2 in binary)

  +2 = 0b 0000 0000 … 0010

  –2 = 0b 1111 1111 … 1101 + 0b 0000 0000 … 0001

  = 0b 1111 1111 … 1110

Signed Binary ↔ Decimal Conversion

Signed Binary → Decimal

$0b1001010 = ?_{ten}$

Binary Digit	Decimal Value
0	$0 \times 2^0 = 0$
1	$1 \times 2^1 = 2$
0	$0 \times 2^2 = 0$
1	$1 \times 2^3 = 8$
0	$0 \times 2^4 = 0$
0	$0 \times 2^5 = 0$
1	$-1 \times 2^6 = -64$
	$\Sigma = -54_{ten}$

Decimal → Signed Binary

$-54 = 0b?$

Decimal	Binary Digit
-54/2 = -27	0
-27/2 = -14	1
-14/2 = -7	0
-7/2 = -4	1
-4/2 = -2	0
-2/2 = -1	0
-1/2 = -1	1
-1/2 = -1	1: repeats
Collect → remainder bits	0b$\boxed{1}$001010

Repeating sign bit

Decimal → Binary Conversion Notes

Positive Decimal → Binary

Continue division until quotient and remainder both become 0 and repeat (sign bit 0 repeats)

Decimal	Binary Digit
+15	
+15/2 = +7	1
+7/2 = +3	1
+3/2 = +1	1
+1/2 = 0	1
0/2 = 0	0
0/2 = 0	0
Sign bit repeats	
Collect → remainder bits	0b01111

Repeating sign bit

Negative Decimal → Binary

Continue division until quotient becomes -1, remainder becomes 1, and they repeat (sign bit 1 repeats)

Decimal	Binary Digit
-15	
-15/2 = -8	1
-8/2 = -4	0
-4/2 = -2	0
-2/2 = -1	0
-1/2 = -1	1
-1/2 = -1	1
Sign bit repeats	
Collect → remainder bits	0b10001

Repeating sign bit

Outline

- Addition of Unsigned Numbers
- Arithmetic Overflow (Unsigned)
- Signed Numbers
- Addition of Signed Numbers
- Subtraction of Signed Numbers
- Arithmetic Overflow (Signed)

2's Complement Addition

- The carry output from the sign-bit position (MSB) is ignored

$$\begin{array}{rr} (+5) & 0101 \\ +\ (+2) & +\ 0010 \\ \hline (+7) & 0111 \end{array} \qquad \begin{array}{rr} (-5) & 1011 \\ +\ (+2) & +\ 0010 \\ \hline (-3) & 1101 \end{array}$$

$$\begin{array}{rr} (+5) & 0101 \\ +\ (-2) & +\ 1110 \\ \hline (+3) & 10011 \end{array} \qquad \begin{array}{rr} (-5) & 1011 \\ +\ (-2) & +\ 1110 \\ \hline (-7) & 11001 \end{array}$$

Cin and Cout of the MSB position are identical (both are 1): therefore, Cout can be ignored

ignore

Cin and Cout of the MSB position are identical (both are 1): therefore, Cout can be ignored

ignore

Outline

- Addition of Unsigned Numbers
- Arithmetic Overflow (Unsigned)
- Signed Numbers
- Addition of Signed Numbers
- Subtraction of Signed Numbers
- Arithmetic Overflow (Signed)

2's Complement Subtraction

- Take the 2's complement of the subtrahend (the second operator) and add it to the minuend (the first operator)

$$\begin{array}{r} (+5) \\ -\ (-2) \\ \hline (+7) \end{array} \qquad \begin{array}{r} 0\,1\,0\,1 \\ -\ 1\,1\,1\,0 \\ \hline \end{array} \qquad \Longrightarrow \qquad \begin{array}{r} 0\,1\,0\,1 \\ +\ 0\,0\,1\,0 \\ \hline 0\,1\,1\,1 \end{array}$$

$$\begin{array}{r} (-5) \\ -\ (-2) \\ \hline (-3) \end{array} \qquad \begin{array}{r} 1\,0\,1\,1 \\ -\ 1\,1\,1\,0 \\ \hline \end{array} \qquad \Longrightarrow \qquad \begin{array}{r} 1\,0\,1\,1 \\ +\ 0\,0\,1\,0 \\ \hline 1\,1\,0\,1 \end{array}$$

2's Complement Subtraction

- Take the 2's complement of the subtrahend (the second operator) and add it to the minuend (the first operator)
- The carry output from the sign-bit position (MSB) is **ignored**

$$\begin{array}{r} (+5) \\ -\ (+2) \\ \hline (+3) \end{array} \qquad \begin{array}{r} 0\,1\,0\,1 \\ -\ 0\,0\,1\,0 \\ \hline \end{array} \qquad \Longrightarrow \qquad \begin{array}{r} 0\,1\,0\,1 \\ +\ 1\,1\,1\,0 \\ \hline 1\,0\,0\,1\,1 \end{array}$$

Cin and Cout of the MSB position are identical (both are 1): therefore, Cout can be ignored

$\uparrow$ ignore

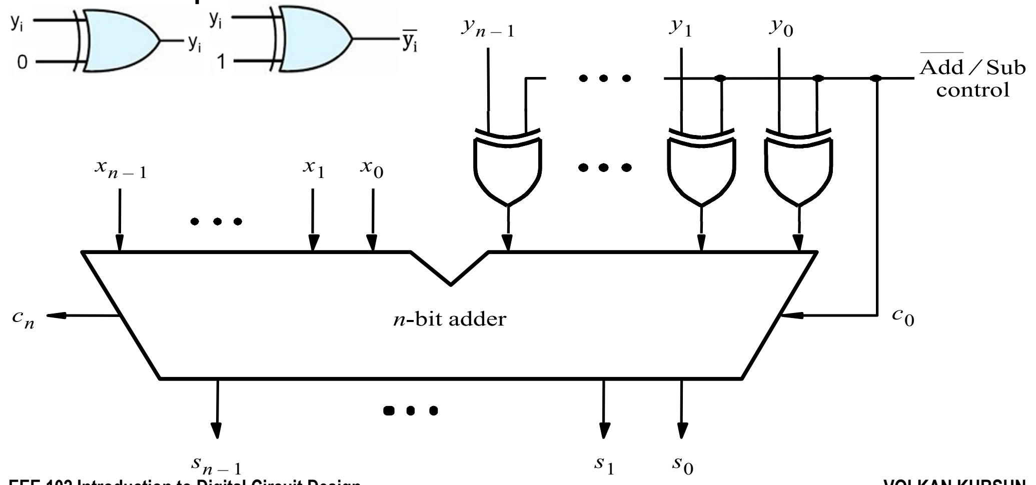
$$\begin{array}{r} (-5) \\ -\ (+2) \\ \hline (-7) \end{array} \qquad \begin{array}{r} 1\,0\,1\,1 \\ -\ 0\,0\,1\,0 \\ \hline \end{array} \qquad \Longrightarrow \qquad \begin{array}{r} 1\,0\,1\,1 \\ +\ 1\,1\,1\,0 \\ \hline 1\,1\,0\,0\,1 \end{array}$$

Cin and Cout of the MSB position are identical (both are 1): therefore, Cout can be ignored

$\uparrow$ ignore

Adder/Subtractor

- Only difference between addition and subtraction: 2's complement of the second operand is added to the first operand in case of subtraction
- 2's complement: invert all bits and add 1



Outline

- Addition of Unsigned Numbers
- Arithmetic Overflow (Unsigned)
- Signed Numbers
- Addition of Signed Numbers
- Subtraction of Signed Numbers
- Arithmetic Overflow (Signed)

Signed Integers: Addition

Assume an architecture with n-bit signed source operands provided by n-bit registers and the signed result will be stored in an n-bit register

Range of signed integers that can be represented with n-bits: -2^{n-1} to $+2^{n-1}-1$

1. **POSITIVE OVERFLOW:** if addition of **two positive numbers produces** a large positive result that cannot be correctly represented with n-bits (**result > $+2^{n-1}$-1**), a **positive overflow** occurs

2. **NEGATIVE OVERFLOW:** if addition of **two negative numbers produces** a small negative result that cannot be correctly represented with n-bits (**result < -2^{n-1}**), a negative **overflow** occurs

Signed Integers: Addition

- Examples: assume an architecture with 5-bit registers and operands: **range of signed integers** that can be represented with 5-bits: -2^4 to $+2^4-1$ = **-16 to +15**
- **OVERFLOW:** if an addition operation produces a result that cannot be correctly represented with 5-bits (**result > +15 or result < -16**) a **positive** or **negative overflow** occurs

Cout_4	Cin_4	Cin_3	Cin_2	Cin_1	Cin_0	Decimal	
0	0	1	0	0	0	← Carry bits	
	0	0	1	1	1	+7	
+	1	0	1	0	0	+ -12	
0	1	1	0	1	1	-5	Correct

Cout_4	Cin_4	Cin_3	Cin_2	Cin_1	Cin_0	Decimal	
1	0	1	1	0	0	← Carry bits	
	1	0	0	1	1	-13	
+	1	0	1	1	0	+ -10	
1	0	1	0	0	1	$\cancel{+9}$	Negative Overflow

Cout_4	Cin_4	Cin_3	Cin_2	Cin_1	Cin_0	Decimal	
0	1	1	0	0	0	← Carry bits	
	0	1	1	1	1	+15	
+	0	1	1	0	0	+ +12	
0	1	1	0	1	1	$\cancel{-5}$	Positive Overflow

Cout_4	Cin_4	Cin_3	Cin_2	Cin_1	Cin_0	Decimal	
1	1	1	1	0	0	← Carry bits	
	1	1	0	1	1	-5	
+	0	1	1	1	0	+ +14	
1	0	1	0	0	1	+9	Correct

Signed Integers: Addition

- OBSERVATION: In signed addition overflow is captured by the carry output from and carry in to the most significant bit position. If Cout_4Cin_4 = 0b01, **there is positive overflow**. If Cout_4Cin_4 = 0b10, **there is negative overflow**. If Cout_4Cin_4 = 0b00 or 0b11, there is **NO overflow**
- **Carry output from and carry in to the most significant bit position** (Cout_(n-1) and Cin_(n-1)) are used to identify the overflow conditions in signed addition

Cout_4	Cin_4	Cin_3	Cin_2	Cin_1	Cin_0	Decimal	
0	**0**	1	0	0	0	← Carry bits	
	0	0	1	1	1	+7	
+	1	0	1	0	0	+ -12	
0	1	1	0	1	1	-5	Correct

Cout_4	Cin_4	Cin_3	Cin_2	Cin_1	Cin_0	Decimal	
1	**0**	1	1	0	0	← Carry bits	
	1	0	0	1	1	-13	
+	1	0	1	1	0	+ -10	
1	0	1	0	0	1	$\cancel{+9}$	Negative Overflow

Cout_4	Cin_4	Cin_3	Cin_2	Cin_1	Cin_0	Decimal	
0	**1**	1	0	0	0	← Carry bits	
	0	1	1	1	1	+15	
+	0	1	1	0	0	+ +12	
0	1	1	0	1	1	$\cancel{-5}$	Positive Overflow

Cout_4	Cin_4	Cin_3	Cin_2	Cin_1	Cin_0	Decimal	
1	**1**	1	1	0	0	← Carry bits	
	1	1	0	1	1	-5	
+	0	1	1	1	0	+ +14	
1	0	1	0	0	1	+9	Correct

Signed Overflow Conditions

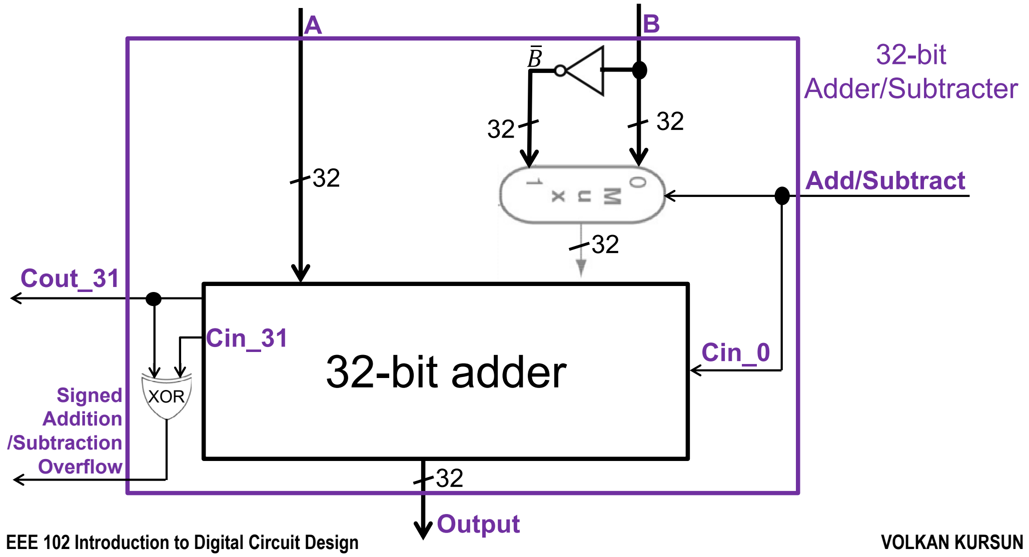
- **Signed overflow** conditions can be captured by checking the carry output from and carry in to the most significant (sign) bit position during addition

$$\boxed{Overflow = Cout\_(n-1) \oplus Cin\_(n-1)}$$

Cout_(n-1)	Cin_(n-1)	Overflow	
0	0	**0**	→ NO overflow
0	1	**1**	→ Positive overflow
1	0	**1**	→ Negative overflow
1	1	**0**	→ NO overflow

Signed Integers: Subtraction

- Subtraction (M-S): the second operand (subtrahend) is negated and added to the first operand (minuend)
  - M - S = M + (-S) = M + (S' + 1): take the 2's complement of the subtrahend and add it to the minuend
  - Add/subtract = 0: control signal for addition. Output = A + B
  - Add/subtract = 1: control signal for subtraction. Output = A + $\bar{B}$ + 1 = A - B


A
B
$\bar{B}$
32
32
32
Mux
1
0
32
32-bit Adder/Subtracter
Add/Subtract
Cout_31
Cin_31
32-bit adder
Cin_0
XOR
Signed Addition /Subtraction Overflow
32
Output