Signed numbers

- Non-negative (unsigned) numbers are well understood
- An n-bit number represents numbers from 0 to 2^n-1
- However, negative numbers cause another problem
- In all solutions, one bit is needed to represent the sign, + or -
- MSB (Most Significant Bit) can be used for that purpose, i.e., represent sign (0: +ve 1: -ve)
- Remaining bits can be interpreted differently
  - They can represent magnitude as a positive number
  - They can be complemented (represent 0 by 1 and 1 by 0)
  - Or manipulate in some other way

Interpretation

- Sign and Magnitude
  - Out of n bits, one is reserved for sign
  - Remaining bits represent the value of number as positive
  - It is equivalent of representing it as $(1-2x_{n-1})\sum_{i=0}^{n-2} 2^i x_i$
- 1's Complement
  - Convert the magnitude of number as a binary string
  - Then complement every bit (replace 1 by 0 and 0 by 1)
  - This is equivalent of having the weight of MSB as -(2^{n-1}-1)
- 2's Complement
  - Convert the magnitude of number as a binary string
  - Complement every bit (replace 1 by 0 and 0 by 1) and add 1
  - This is equivalent of having the weight of MSB as -2^{n-1}

Example

Consider the bit string 1010:

- Sign and Magnitude
  - $\underline{1}$ $\underline{010}$
    -ve 2
  - So it represents -2
- 1's Complement
  - $\underline{1}$ $\underline{010}$ → $\underline{1}$ $\underline{101}$
    -ve 5
  - So it represents -5
- 2's Complement
  - $\underline{1}$ $\underline{010}$ → $\underline{1}$ $\underline{101}$ + 1
    -ve 6
  - So it represents -6

Sign Magnitude, 1's, and 2's complement

Binary	Sign Magnitude	1's Complement	2's Complement
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	-0	-7	-8
1001	-1	-6	-7
1010	-2	-5	-6
1011	-3	-4	-5
1100	-4	-3	-4
1101	-5	-2	-3
1110	-6	-1	-2
1111	-7	-0	-1

Maximum and Minimum values in n-bits

- We use 2's complement as it makes arithmetic (add/sub) simple
- n-bits uses only n-1 bits to store the value
- Largest positive value is 2^{n-1}-1
- Largest negative value is -2^{n-1}
- For n=4, these values are +7 and - 8
- For n=8, these values are +127 and - 128
- If we need larger or smaller values to be stored, we have problem -- leads to overflow and underflow

- For MULT/DIV, sign and magnitude is better
  - But we cannot keep switching

Negation

- To change sign of a number
- In Sign and Magnitude
  - Just complement the sign
- 1's Complement
  - Complement all bits
- 2's Complement
  - Complement all bits and add 1
- Adding 1 is expensive operation (Example: Add 1 to 0111)
- Alternate 2's complement method
  - Scan the string from right
  - Retain all bits up to the first 1
  - Then complement the remaining bits

Example:
$6 = 01\underline{10}$
$-6 = 10\underline{10}$

Negation Examples

- Negate the following 4-bit 2's Complement Binary Values:

0011	1111	0111	1010
1100+1 → 1101	0000+1 → 0001	1000+1 → 1001	0101+1 → 0110

- What is the negation of 1000 in 4-bit 2's complement?

Converting negative number to Binary

- Convert a negative decimal number to binary in 2's complement
- Method 1:
  - Convert the magnitude to an n-bit string
  - Negate the number
  - Example: -5 Magnitude in binary: 0101 Negation: 1011
- Method 2:
  - The magnitude of number must be less than or equal to 2^{n-1}
  - Add 2^n to the number
  - Convert this number as an n-bit unsigned integer
  - Example: -4 + (16) = 12 (decimal) = 1100 (binary)
    -7 + (16) = 9 (decimal) = 1001 (binary)

Computer Arithmetic for one bit

- ADD and SUB are fundamental
- Adding one digit to another gives result(R) and carry(C) bit
- Subtracting a digit from another gives result(R) and borrow(B)
- Examples of adding/subtracting two digits

X	0	0	1	1	X	0	0	1	1
Y	+0	+1	+0	+1	Y	-0	-1	-0	-1
R	0	1	1	0	R	0	1	1	0
C	0	0	0	1	B	0	1	0	0

- Add/sub of two digits with carry/borrow also gives two digits
- That is adding/subtracting two digits with carry/borrow

Previous →	C	1	1	1	1	B	1	1	1	1
	X	0	0	1	1	X	0	0	1	1
	Y	+0	+1	+0	+1	Y	-0	-1	-0	-1
	R	1	0	0	1	R	1	0	0	1
Current →	C	0	1	1	1	B	1	1	0	1

ADD/SUB with more than one bit

- Follow rules of decimal arithmetic
- Add carry to/sub borrow from the next digit
- In 2's complement, if we simply add or subtract without regard to sign, we get correct result if there is no overflow/underflow
- Overflow/Underflow occurs when the carry into and the carry out of the sign bit position are different.
- Examples

C/B	00010		01000		11010		10000	
X	0101	0101	0101	1001	0010	1011	0101	1011
Y	+0001	+1011	+0100	+1010	-0101	-1001	-1101	-0100
Res	0110		1001		1101		1000	
	Corr	Corr	Over	Under	Corr	Corr	Over	Under

ADD/SUB revisited

- Understand the examples again
- Overflow
  - When two positive numbers added together or a negative number subtracted from a positive number yields negative
- Underflow
  - When two negative numbers added together or a positive number subtracted from a negative number yields positive

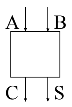
C/B	00010	11110	01000	10000	11010	00000	10000	01000
X	0101	0101	0101	1001	0010	1011	0101	1011
Y	+0001	+1011	+0100	+1010	-0101	-1001	-1101	-0100
Res	0110	0000	1001	0011	1101	0010	1000	0111
	Corr	Corr	Over	Under	Corr	Corr	Over	Under

Using 1-bit building blocks to make n-bit circuit

- Design a 1-bit circuit with proper "glue logic" to use it for n-bits
  - It is called a bit slice
  - The basic idea of bit slicing is to design a 1-bit circuit and then piece together n of these to get an n-bit component

- Example:
- A half-adder adds two 1-bit inputs
- Two half adders can be used to add 3 bits
- A 3-bit adder is a full adder
- A full adder can be a bit slice to construct an n-bit adder

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

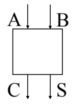


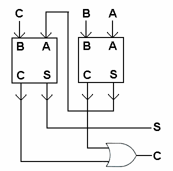
Half Adder

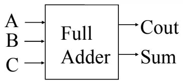
Full adder and multi-bit ripple-carry adder

- Two half adders can be used to add 3 bits
- n-bit adder can be built by full adders
- n can be arbitrary large

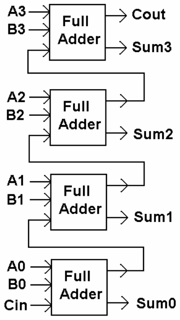
Half Adder







4-bit ripple-carry adder



Three Representations of Logic Functions

	AND	OR	NOT
1. Logic Expression	X.Y	X+Y	X’ $\overline{X}$ ~X !X

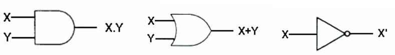
2. Truth Table

X	Y	X.Y
0	0	0
0	1	0
1	0	0
1	1	1

X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1

X	X'
0	1
1	0

3. Circuit Diagram / Schematic



Logic Functions of 2 Variables

X Y	F0	F1	F2	F3	F4	F5	F6	F7
0 0	0	0	0	0	0	0	0	0
0 1	0	0	0	0	1	1	1	1
1 0	0	0	1	1	0	0	1	1
1 1	0	1	0	1	0	1	0	1

X Y	F8	F9	F10	F11	F12	F13	F14	F15
0 0	1	1	1	1	1	1	1	1
0 1	0	0	0	0	1	1	1	1
1 0	0	0	1	1	0	0	1	1
1 1	0	1	0	1	0	1	0	1

- F1 is called a logical AND, denoted by X.Y
- F6 is called an XOR (Exclusive-OR), denoted by $X \oplus Y$
- F7 is called OR, denoted by X + Y
- F8 is NOR, denoted by $\overline{X + Y}$
- F9 is called an XNOR (Exclusive-NOR), denoted by $\overline{X \oplus Y}$
- F14 is NAND, denoted by $\overline{X.Y}$

Truth Tables for 2 Variable Functions

X	Y	AND
0	0	0
0	1	0
1	0	0
1	1	1

X	Y	OR
0	0	0
0	1	1
1	0	1
1	1	1

X	Y	XOR
0	0	0
0	1	1
1	0	1
1	1	0

X	Y	NAND
0	0	1
0	1	1
1	0	1
1	1	0

X	Y	NOR
0	0	1
0	1	0
1	0	0
1	1	0

X	Y	XNOR
0	0	1
0	1	0
1	0	0
1	1	1

Half Adder Truth Tables

1)

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

2)

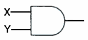
A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

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

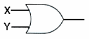
C	B	A	C
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Logic Gate Symbols

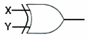
- AND denoted by X.Y



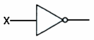
- OR denoted by X + Y



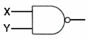
- XOR denoted by $X \oplus Y$



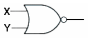
- NOT denoted by X’ or X



- NAND denoted by $\overline{X.Y}$



- NOR denoted by $\overline{X + Y}$



- XNOR denoted by $\overline{X \oplus Y}$