INTRODUCTION TO COMPUTING

Representation of Numbers and Number Base Conversion

Representation of numbers

- Definition: A number x expressed in base r is written using digits $0,1,2,\ldots,r-1$ in the form: $x=(x_{n-1}x_{n-2}\cdots x_2x_1x_0.x_{-1}x_{-2}\cdots x_{-m})_r$.
- The decimal value of a number $x=(x_{n-1}x_{n-2}\cdots x_2x_1x_0.x_{-1}x_{-2}\cdots x_{-m})_r$ is
  $(x)_{10}=x_{n-1}\times r^{n-1}+x_{n-2}\times r^{n-2}+x_{n-3}\times r^{n-3}+\cdots+x_1\times r+x_0+x_{-1}\times r^{-1}+x_{-2}\times r^{-2}x_{-2}+\cdots+x_{-m}\times r^{-m}$
- Example 1: Decimal Numbers (Base 10 numbers)
  - Written using the digits $0,1,2,\ldots,9$.
  - Example: $(25.31)_{10}=2\times10+5+3\times10^{-1}+1\times10^{-2}$

- Example 2: Binary Numbers (Base 2 numbers)
  - Written using the digits 0 and 1.
  - Example:
    $(1011.11)_2=(1\times2^3+0\times2^2+1\times2+1+1\times2^{-1}+1\times2^{-2})_{10}$
    $=(7.75)_{10}$

- Example 3: Hexadecimal Numbers (Base 16 numbers)
  - Written using $0,1,2,3,4,5,6,7,8,9,A(10),B(11),C(12),D(13),E(14),F(15)$.
  - Example:
    $(AF93.ED5)_{16}=(10\times16^3+15\times16^2+9\times16+3+14\times16^{-1}+13\times16^{-2}+5\times16^{-3})_{10}$
    $=(40960+3840+144+3+0.875+0.05+0.00122)_{10}$
    $=(44947.92622)_{10}$

- Example 4: Octal Numbers (Base 8 numbers)
  - Written using $0,1,2,\ldots,8$.
  - Example:
    $(256.4)_8=(2\times8^2+5\times8+6+4\times8^{-1})_{10}$
    $=(128+40+6+0.5)_{10}$
    $=(174.5)_{10}$

Number Base Conversion

- Conversion to Decimal: See previous examples.
- Conversion from Decimal to Binary
  - Example: Convert $(39.53)_{10}$ to binary.
  - Step 1. $(39.53)_{10} = (39)_{10} + (0.53)_{10}$.
  - Step 2. Convert the integer part to binary.
    - Divide the integer part by 2, and write the quotient and remainder.

		Integer		**Remainder**	**Coefficient**
39/2	=	19	+	1	1
19/2	=	9	+	1	1
9/2	=	4	+	1	1
4/2	=	2	+	0	0
2/2	=	1	+	0	0
1/2	=	0	+	1	1
					↑ Answer: $(39)_{10} = (100111)_2$

  - Step 3. Convert the decimal fraction part to binary.

		Integer		**Fraction**	**Coefficient**
					↓ Answer: $(0.53)_{10} = (0.1000011...)_2$
0.53x2	=	1	+	0.06	1
0.06x2	=	0	+	0.12	0
0.12x2	=	0	+	0.24	0
0.24x2	=	0	+	0.48	0
0.48x2	=	0	+	0.96	0
0.96x2	=	1	+	0.92	1
0.92x2	=	1	+	0.84	1

  - Therefore:

$$(39.53)_{10} = (39)_{10} + (0.53)_{10}$$
$$= (100111)_2 + (0.1000011...)_2$$
$$= (100111.1000011...)_2$$

- Decimal to Hexadecimal
  - Example: Convert $(39.53)_{10}$ to hexadecimal.
  - Step 1. $(39.53)_{10} = (39)_{10} + (0.53)_{10}$.
  - Step 2. Convert the integer part to hexadecimal.
    - Divide the integer part by 16, and write the quotient and remainder.

Integer		Quotient		Remainder	Coefficient
39					
39/16	=	2	+	7	7
2/16	=	0	+	2	2
					↑ Answer: $(39)_{10} = (27)_{16}$

  - Step 3. Convert the decimal fraction part to hexadecimal.

		Integer		Fraction	Coefficient
					↓ Answer: $(0.53)_{10} = (0.87AE1...)_{16}$
0.53x16	=	8	+	0.48	8
0.48x16	=	7	+	0.68	7
0.68x16	=	10	+	0.88	A
0.88x16	=	14	+	0.08	E
0.08x16	=	1	+	0.28	1

  - Therefore:

$$(39.53)_{10} = (39)_{10} + (0.53)_{10}$$
$$= (27)_{16} + (0.87AE1...)_{16}$$
$$= (27.87AE1...)_{16}$$

- Binary to Hexadecimal
  - Partition the binary number into groups of 4 digits each, starting from the binary point.
  - Assign the corresponding hexadecimal code to each group of 4 digits.
  - Example:

$$\left(\underbrace{1}_{1}\underbrace{1101}_{D}\underbrace{0011}_{3}\underbrace{0111}_{7}.\underbrace{1010}_{A}\underbrace{0111}_{7}\right)_2 = (1D37.A7)_{16}$$

Arithmetic Operations on Unsigned Binary Integer Numbers

Addition

Addition of 1-bit numbers:

Augend		Addend	Carry	Sum
0	+	0	0	0
0	+	1	0	1
1	+	0	0	1
1	+	1	1	0

Addition of n-bit numbers:

- The decimal value for an unsigned n-bit number ranges from 0 to $2^n - 1$.
- The sum of two unsigned n-bit numbers ranges from 0 to $2^{n+1} - 2 = 2(2^n - 1)$.
- Addition starts with the least significant bits of the augend and addend, and continues with the addition of successively higher order bits and the carry from the previous lower order bits addition.
  - Example: n=8

Carries		**0**	**0**	**1**	**0**	**0**	**0**	**0**	**1**		
Augend	X		**1**	**0**	**1**	**1**	**0**	**0**	**0**	**1**	177
Addend	Y		**0**	**0**	**1**	**0**	**0**	**1**	**0**	**1**	+ 37
Sum	X+Y	0]	**1**	**1**	**0**	**1**	**0**	**1**	**1**	**0**	214

- As long as the sum of two n-bit unsigned numbers is less than $2^n - 1$, it can be represented as an n-bit number.
- When the sum of two n-bit unsigned numbers exceeds $2^n - 1$, it produces an ***arithmetic overflow***: the sum cannot be represented with n bits, and a carry bit is required.
  - Example: n=3.

Carries		**1**	**0**	**0**			
Augend	X		**1**	**1**	**1**		7
Addend	Y		**1**	**0**	**0**	+	4
Sum	X+Y	1]	**0**	**1**	**1**		11

Subtraction

Subtraction of 1-bit numbers:

Minuend		Subtrahend	Borrow	Difference
0	-	0	0	0
0	-	1	1	1
1	-	0	0	1
1	-	1	0	0

Subtraction of n-bit numbers:

- Subtraction starts with the least significant bits of the minuend and subtrahend and continues with the subtraction of successively higher order bits, including a borrow from the previous lower order bit.
  - Example:

Borrows			**0**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	
Minuend	X			**1**	**0**	**1**	**1**	**0**	**1**	**0**	**1**	181
Subtrahend	Y			**0**	**0**	**0**	**1**	**1**	**1**	**0**	**1**	- 29
Difference	X-Y	0]	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	152

- If the subtrahend is greater than the minuend, an ***arithmetic underflow*** condition exists and results in a borrow of the higher order bit.

  - Example:

Borrows			**1**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	
Minuend	X			**0**	**0**	**1**	**1**	**0**	**1**	**0**	**1**	53
Subtrahend	Y			**1**	**0**	**0**	**1**	**1**	**1**	**0**	**1**	- 157
Difference	X-Y	1]	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	- 104

- In unsigned binary representations, although a borrow flag indicates a negative difference, this difference cannot be represented for subsequent storage.
- Unsigned binary is inappropriate and cannot be used as a method for handling subtraction requiring borrow.

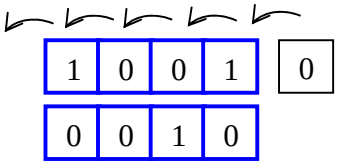
Logical Shifts

Logical shifts multiply and divide unsigned binary integer numbers.

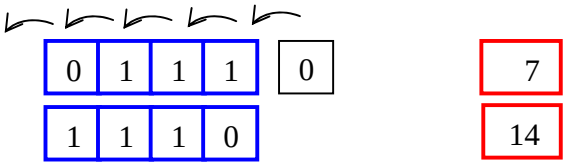
Logical Left Shift

- Logical left shift moves each bit in a register one position to the left.
  - The bit shifted out of the left end of the register is lost.
  - A 0 bit is shifted into the right end.

- Example: 4-bit register.



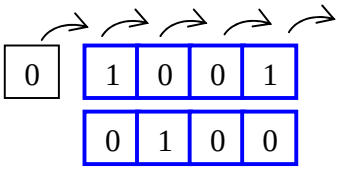
- If the data is an unsigned binary integer, then:
  - If the bit shifted in is 0 and the bit shifted out is zero, then the number is multiplied by 2.
  - Example: 4-bit register



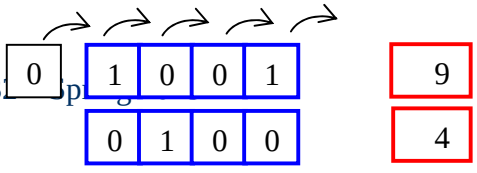
  - Why?

Logical Right Shift

- A logical right shift moves each bit in a register one position to the right:
  - The bit shifted out of the right end of the register is lost.
  - A 0 bit is shifted into the left end.
- Example: 4-bit register



- If the data is an unsigned binary integer, then:
  - Since the bit shifted in is 0, then the number is divided by 2, and the register shows only the integer part of the result.
  - Example: 4-bit register



- Why?

Two's Complement Numbers

- Two's complement numbers provide a binary representation for both positive and negative values.
  - Can be added and subtracted by the same addition and subtraction procedure for unsigned binary numbers.
  - In an n-bit two's complement number $x = (x_{n-1}x_{n-2}\cdots x_2x_1x_0)_2$, the most significant bit x_{n-1} is the sign.
    - $x_{n-1} = 0 \Rightarrow$ positive number
    - $x_{n-1} = 1 \Rightarrow$ negative number
- For $x \geq 0 \Rightarrow x = \left(\underbrace{x_{n-1}}_{\substack{Sign \\ bit=0}} \underbrace{x_{n-2}\cdots x_2x_1x_0}_{magnitude} \right)_2$; $0 \leq \text{magnitude} \leq 2^{n-1}-1$.
  - Example: 3-bit numbers

    $x = \left(\underbrace{x_2}_{\substack{Sign \\ bit=0}} \underbrace{x_1x_0}_{magnitude} \right)_2$; $0 \leq \text{magnitude} \leq 2^{3-1}-1 = 3$

    $x = (011)_2 = (0\times 2^2 + 1\times 2^1 + 1\times 2^0)_{10} = (3)_{10}$
- The decimal value of an n-bit positive two's complement number $x = (0x_{n-2}\cdots x_2x_1x_0)_2$ is:

$$x = \sum_{i=0}^{n-2} x_i 2^i$$
$$= x_{n-2}\times 2^{n-2} + x_{n-3}\times 2^{n-3} + \cdots + x_1\times 2 + x_0$$

- The decimal value of an n-bit negative two's complement number $x = (1x_{n-2}\cdots x_2x_1x_0)_2$ ranges from -1 to -2^{n-1} and is given by:

$$x = -1\times 2^{n-1} + \sum_{i=0}^{n-2} x_i 2^i$$
$$= -2^{n-1} + x_{n-2}\times 2^{n-2} + x_{n-3}\times 2^{n-3} + \cdots + x_1\times 2 + x_0$$

- Example: 8-bit numbers in two's complement form:

Decimal	Binary	Hex
+127	01111111	7F
+126	01111110	7E
...	...	...
+2	00000010	02
+1	00000001	01
0	00000000	00
-1	11111111	FF
-2	11111110	FE
...	...	
-127	10000001	81
-128	10000000	80

- Consider an n-bit number in two's complement form

$$(x)_2 = \left(\underbrace{x_{n-1}}_{\substack{Sign \\ bit}} \underbrace{x_{n-2} \cdots x_2 x_1 x_0}_{magnitude} \right)_2$$

$$(x)_{10} = -x_{n-1} \times 2^{n-1} + \sum_{i=0}^{n-2} x_i 2^i$$

$$= -x_{n-1} \times 2^{n-1} + x_{n-2} \times 2^{n-2} + x_{n-3} \times 2^{n-3} + \cdots + x_1 \times 2 + x_0$$

Then, its negative equivalent $(-x)_{10}$ is obtained by computing $2^n - x$, i.e. computing $\left(1\underbrace{00\cdots0}_{n\ zeros}\right)_2 - \left(x_{n-1}x_{n-2}\cdots x_2x_1x_0\right)_2$ where $x = \left(x_{n-1}x_{n-2}\cdots x_2x_1x_0\right)_2$.

- The computation of $2^n - x$ is referred to as "*taking the two's complement of* x ".
- The computation of $2^n - x$ is as follows:
  - $x = \left(\underbrace{x_{n-1}}_{\substack{Sign \\ bit}} \underbrace{x_{n-2} \cdots x_2 x_1 x_0}_{magnitude} \right)_2$
  - The two's complement of x is:

$$2^n - x = \left(1\underbrace{00\cdots0}_{n\ zeros}\right)_2 - \left(x_{n-1}x_{n-2}\cdots x_2x_1x_0\right)_2$$

$$= \left(\underbrace{11\cdots1}_{n\ 1s}\right)_2 + 1 - \left(x_{n-1}x_{n-2}\cdots x_2x_1x_0\right)_2$$

$$= \left(\underbrace{11\cdots1}_{n\ 1s}\right)_2 - \left(x_{n-1}x_{n-2}\cdots x_2x_1x_0\right)_2 + 1$$

$$= \left(\underbrace{(1-x_{n-1})}_{\bar{x}_{n-1}} \underbrace{(1-x_{n-2})}_{\bar{x}_{n-2}} \cdots \underbrace{(1-x_1)}_{\bar{x}_1} \underbrace{(1-x_0)}_{\bar{x}_0} \right)_2 + 1$$

$$= \left(\bar{x}_{n-1}\bar{x}_{n-2}\cdots\bar{x}_2\bar{x}_1\bar{x}_0\right)_2 + 1$$

where $\bar{x}_i$ is the logical complement of x_i.

- The two's complement of x is obtained by complementing each bit of x and adding 1 to the result. Any carry out of the high order bit position resulting from the addition is ignored. The resulting n-bit number represents the two's complement of x.
- Taking the two's complement of a number refers to computing the negative of a number that is already in two's complement form, regardless of whether the number was originally positive or negative.
- When the two's complement procedure is applied to a number (positive or negative) that is initially represented in two's complement form, the negative of that number is obtained.
- Example:

$(x)_2$	=	**1**	**1**	**1**	**0**	**0**	**1**	**1**	**0**	-26
Complement	=	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	
Add 1	+								**1**	
Two's complement representation	0]	**0**	**0**	**0**	**1**	**1**	**0**	**1**	**0**	26

- The above procedure cannot be applied to compute $(-x)_{10}$ when $x = \left(1\underbrace{00\cdots0}_{n-1\ 0s}\right)_2$.

Addition of Two's Complement Numbers

- Two's complement numbers are added without paying special attention to the sign bit.
- The sum is correct if it is within the allowed range -2^{n-1} to $2^{n-1}-1$.
- Errors arise if the result exceeds the allowed range, producing an arithmetic overflow.

- Example:

										When interpreted as unsigned integers		When interpreted as two's complement numbers	
Carries			**0**	**1**	**1**	**1**	**0**	**1**	**1**	**1**				
Augend	X			**0**	**0**	**0**	**1**	**0**	**1**	**0**	**1**	21		21
Addend	Y			**0**	**1**	**1**	**1**	**0**	**1**	**1**	**1**	+ 119	+	119
Sum	X+Y	0]	**1**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	140			-116

										When interpreted as unsigned integers		When interpreted as two's complement numbers	
Carries			**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**				
Augend	X			**1**	**1**	**0**	**0**	**1**	**0**	**1**	**0**	202		-54
Addend	Y	+	**1**	**0**	**1**	**0**	**0**	**0**	**1**	**1**	+ 163	+	-93	
Sum	X+Y	1]	**0**	**1**	**1**	**0**	**1**	**1**	**0**	**1**	109			-147
		Carry lost									8 bit result			

- If the last two carry bits (the ones on the far left of the top row) are 11 or 00, the result is valid. If the last two carry bits are 10 or 01, an overflow has occurred.