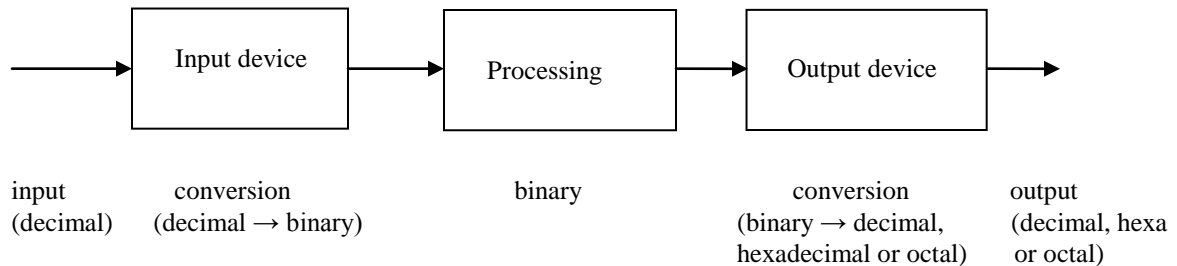


CIS-350
INFRASTRUCTURE TECHNOLOGIES

NUMBER SYSTEMS – Chapter 3 (Summary)

You do not have to read chapter 3. You can just rely on these lecture notes that follow.

We humans use a decimal or base 10 number system, presumably because people have 10 fingers. Early computers were designed around a decimal number system. This approach was somehow cumbersome, low reliable, complex and inefficient. In 1945, the computer pioneers were struggling to change it. John von Neumann suggested that the number system used by computers should take advantage of the physical characteristics of electronic circuitry. Instead of dealing with 10 states and using one discrete voltage value for each state let's use two states only: "on" or "off", "1" or "0". In other words, von Neumann suggested using a binary number system. Today's computers operate in binary numbers and communicate to us mainly in decimal numbers, but sometimes in octal or hexadecimal numbers as well.



Therefore, we have to cover the following topics:

1. Number systems.
2. Translation of integer and real numbers from one system to another.

Integer (Whole) Numbers

All number systems are positional systems.

Decimal number system:

base or radix - 10

digits used - 0 through 9

254 is a different number from 245 in spite of the fact that they contain the same digits.

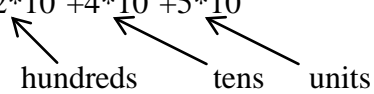
A relative position of a digit in a number is important.

	Integer Quotient (IQ)	Remainder (R)
254 : 10 =	25	4
25 : 10 =	2	5
2 : 10 =	0	2

$(254)_{10} = 2 \cdot 100 + 5 \cdot 10 + 4 \cdot 1 = 2 \cdot 10^2 + 5 \cdot 10^1 + 4 \cdot 10^0$

↖ ↖ ↖
most significant position hundreds tens units least significant position

$$(245)_{10} = 2*100 + 4*10 + 5*1 = 2*10^2 + 4*10^1 + 5*10^0$$

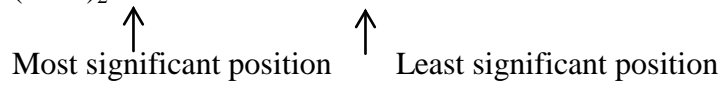


In a decimal system, powers of 10 are used.

Binary number system

- base or radix - 2
- digits used - 0 or 1

$$(1011)_2 = 1*2^3 + 0*2^2 + 1*2^1 + 1*2^0$$



Again, the position of a digit matters. Powers of 2 are used.

$$(1101)_2 = 1*2^3 + 1*2^2 + 0*2^1 + 1*2^0$$

Octal

- base or radix - 8
- digits used - 0 through 7

$$(361)_8 = 3*8^2 + 6*8^1 + 1*8^0$$

Powers of 8 are used.

Hexadecimal (Hex)

- base or radix - 16
- digits and letters used: 0 through 9, and A, B, C, D, E, and F
10,11,12,13,14, and 15.

$$(A6E2)_{16} = A*16^3 + 6*16^2 + E*16^1 + 2*16^0 = 10*16^3 + 6*16^2 + 14*16^1 + 2*16^0$$

Again, both octal and hex system are positional systems. Powers of 16 are used.

Conversions from one system to another

Decimal to Binary

$$(57)_{10} \rightarrow (\quad)_2$$

Divide the number by base 2 until the integer quotient (IQ) is 0 and record the remainder (R).

	IQ	R	
$57 : 2 =$	28	1	<----- <u>least</u> significant position
$28 : 2 =$	14	0	
$14 : 2 =$	7	0	
$7 : 2 =$	3	1	
$3 : 2 =$	1	1	
$1 : 2 =$	0	1	<----- <u>most</u> significant position

$$(57)_{10} \rightarrow (111001)_2$$

Binary to Decimal

$$\begin{aligned}(111001)_2 &= 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 \\ &= \underset{2^5}{32} + 16 + 8 + 0 + 0 + \underset{2^0}{1} = (57)_{10}\end{aligned}$$

Decimal to Hex

$$(185)_{10} = (\quad)_{16}$$

Divide the number by base 16 until the integer quotient (IQ) is 0 and record the remainder (R).

	IQ	R	
$185 : 16 =$	11	9	least significant position
$11 : 16 =$	0	11 (B)	most significant position

$$(185)_{10} = (B9)_{16}$$

Hex to Decimal

$$(B9)_{16} = 11 \cdot 16^1 + 9 \cdot 16^0 = 176 + 9 = (185)_{10}$$

Decimal to Octal

$$(185)_{10} = (\quad)_8$$

	IQ	R
$185 : 8 =$	23	1
$23 : 8 =$	2	7
$2 : 8 =$	0	2

$$(185)_{10} = (271)_8$$

Octal to Decimal

$$(271)_8 = 2 \cdot 8^2 + 7 \cdot 8^1 + 1 \cdot 8^0 = 128 + 56 + 1 = (185)_{10}$$

Binary <-----> Octal

In these conversions, the following table will be useful.

Octal	Binary	Decimal	Octal	Binary	Decimal
0	000	0	4	100	4
1	001	1	5	101	5
2	010	2	6	110	6
3	011	3	7	111	7

Each octal digit is equivalent to 3 binary digits. The maximum unsigned decimal number that can be stored in 3 bits is 7.

Octal ----> Binary

Using the table above, it is easy to perform the conversion.

$$(673)_8 = (110\ 111\ 011)_2$$

6 7 3

Binary -----> Octal

$$(101|011|110)_2 \text{--->} (536)_8$$

Starting from the right, we form groups containing 3 binary digits each. Each group of 3 binary digits represents 1 octal digit.

$$(10|111|101)_2 \text{--->} (010|111|101)_2 = (275)_8$$

Pad with 0's to the left to create a 3-tuple, if the count of binary digits is not the multiplicity of 3.

Binary <-----> Hex

In these conversions, the following table will be handy.

Hex	Binary	Decimal	Hex	Binary	Decimal
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	A	1010	10
3	0011	3	B	1011	11
4	0100	4	C	1100	12
5	0101	5	D	1101	13
6	0110	6	E	1110	14
7	0111	7	F	1111	15

Each hexadecimal symbol is equivalent to precisely 4 binary digits.

The maximum positive unsigned decimal number that can be stored in 4 bits is 15.

Hex ----> Binary

Using the handy table above, the conversion becomes easy.

$$(EAF8)_{16} = (1110\ 1010\ 1111\ 1000)_2$$

E A F 8

Binary ----> Hex


$$(1011|1100|0110)_2 = (BC6)_{16}$$

B C 6

Starting from the right, we form groups of 4 binary digits. Each group represents 1 hex digit or symbol.

$$(1|0110|1110)_2 = (0001|0110|1110)_2 = (16E)_{16}$$

1 6 E



Pad with 0s to the left to create a 4-tuple, if the count of binary digits is not the multiplicity of 4.

Hex <-----> Octal

$$(AB)_{16} = (\quad)_8$$

Method 1: hex ---> decimal ---> octal (less convenient)
Method 2: hex ---> binary ---> octal (more convenient)

Method 1

$$(AB)_{16} = 10 * 16^1 + 11 * 16^0 = 160 + 11 = (171)_{10}$$

	IQ	R
171 : 8 =	21	3
21 : 8 =	2	5
2 : 8 =	0	2

$$(AB)_{16} = (253)_8$$

Method 2

$$(AB)_{16} = (10|10\ 1|011)_2 = (253)_8$$

So far, we have been dealing with unsigned, positive integer numbers in four different systems: decimal, binary, octal, and hexadecimal. By now, we should know how to represent numbers in different systems and how to convert them from one system to another. We could also see that the octal or hexadecimal system is more convenient to use than the binary one because less digits or symbols are required to represent a number.

$$(0101|1101)_2 = (5D)_{16}$$

$$(01|011|101)_2=(135)_8$$

The maximum decimal unsigned number that can be stored in 4 bits is $(1111)_2=(15)_{10}$.

The minimum decimal unsigned number that can be stored in 4 bits is $(0000)_2=(0)_{10}$.

4 bits can represent $2^4=16$ different patterns. The range is $[0,2^4-1]=[0,15]$

The maximum decimal unsigned number that can be stored in 8 bits is $(11111111)_2=(255)_{10}$.

The minimum decimal unsigned number that can be stored in 8 bits is $(00000000)_2=(0)_{10}$.

8 bits can represent $2^8=256$ unique patterns. The range is $[0,2^8-1]=[0,255]$

How shall one represent larger numbers than 255?

Combine bytes to form:

- half-words (2 bytes - 16 bits, size limit: [0-65535],
16 bits can represent $2^{16}=65536$ unique patterns

- words (4 bytes - 32 bits)
32 bits can represent 2^{32} unique patterns

- double words (8 bytes - 64 bits)
64 bits can represent 2^{64} unique patterns

Ex. Program in C++/Java/C#

```
unsigned int counter;
counter=0;
while (counter <= 70000) {
    //do something
    counter=counter+1;
}
```

Knowledge of the size limits of numbers used for calculations in a particular language such as C++/Java/C# is very important, since some calculations (see the program above) can cause a numerical result that falls out of range. In some implementations of C++/Java/C# language, the declaration *unsigned int counter* will allocate to the variable *counter* 2 bytes only. Therefore, the variable can store numbers within the range [0, 65535] only. The condition *counter <= 70000*, assumes that the variable *counter* will eventually reach 70001, and the loop will end. Actually, it will never happen. As a result, the program will work incorrectly. It will be caught in an endless loop because when the variable *counter* reaches 65535, it will be initialized back to 0. To fix the problem, change the declaration statement to *unsigned long int counter*. Now, 4 bytes have been allocated to the variable *counter* allowing the variable to store the larger number. As a result, the program will work correctly.

Real/Floating Point Numbers (with Fraction)

How do we deal with real numbers?. How are they represented and converted from one base to another?

$$(20.75)_{10} = \underline{2*10^1 + 0*10^0} + \underline{7*10^{-1} + 5*10^{-2}}$$

integer part fraction

Decimal ----> Binary

$$(20.75)_{10} = (\quad)_2$$

We deal with the integer part and fraction separately.

Integer part:

	IQ	R	
20 : 2 =	10	0	least significant position
10 : 2 =	5	0	
5 : 2 =	2	1	
2 : 2 =	1	0	
1 : 2 =	0	1	most significant position

We have the partial answer as $(20.75)_{10} = (10100. ?)_2$

Now we deal with the fraction.

	Integer Product (IP)	Fraction	
$0.75 * 2 = 1.5$	1	0.5	Most significant position 2^{-1}
$0.5 * 2 = 1.0$	1	0.0	Least significant position 2^{-2}

We multiply by base 2 until the fraction is 0, or the desired accuracy is achieved.

The answer is: $(20.75)_{10} = (10100.11)_2$

Conversions of real numbers from one base to another are also possible.

Decimal ---> Hex

$$(20.75)_{10} = (\quad)_{16}$$

Method 1 (more convenient?) - through binary (the binary result is above)

$$(20.75)_{10} = (10100.11)_2 = (14.C)_{16}$$

Start with the integer part (to the left of the decimal point). Starting from the right create groups of four binary digits. Each group of four binary digits corresponds to one hex digit. Pad with three insignificant 0s to the left of 1.

Now deal with the fraction (to the right of the decimal point). Starting from the left create groups of four binary digits. Pad with two 0s.

Method 2 (less convenient?)

$$(20.75)_{10} = (\quad)_{16}$$

Integer part:

	IQ	R
20 : 16 =	1	4
1 : 16 =	0	1

Fraction:

	Integer Product (IP)	Fraction
0.75*16=12.0	12 (C)	0

$$(20.75)_{10} = (14.C)_{16}$$

Some real numbers have finite binary representation. They are stored exactly in memory. For example,

$$\begin{aligned} (0.5)_{10} &= (0.1)_2 \\ (0.75)_{10} &= (0.11)_2 \\ (0.875)_{10} &= (0.111)_2 \\ (10.625)_{10} &= (1010.101)_2 \end{aligned}$$

However, very many real numbers do not have a finite representation/expansion and they are not stored exactly in memory:

$$(0.51)_{10} = (\quad ? \quad)_2$$

	IP	Fraction
0.51*2=1.02	1	0.02
0.02*2=0.04	0	0.04
0.04*2=0.08	0	0.08
0.08*2=0.16	0	0.16
0.16*2=0.32	0	0.32
0.32*2=0.64	0	0.64
0.64*2=1.28	1	0.28
0.28*2=0.56	0	0.56
0.56*2=1.12	1	0.12
0.12*2=0.24	0	0.24
0.24*2=0.48	0	0.48
0.48*2=0.96	0	0.96
0.96*2=1.92	1	0.92
0.92*2=1.84	1	0.84
0.84*2=1.68	1	0.68

.....
.....

$$(0.51)_{10} = (0.100000101000111.....)_2$$

The fraction does not “want” to become 0, so we continue multiplying by 2 until the desired accuracy is achieved.

The computer word, which stores a real number, has a limited number of bits. For example, when you declare a variable x as *float x*, *double x*, or *long double x*, compiler will typically assign 4 bytes (32 bits), 8 bytes (64 bits), or 10 bytes (80 bits), respectively, to the variable x . Regardless the declaration, however, 0.51 will not be stored exactly as 0.51 in the computer’s memory; it will be very close to 0.51. Therefore, very many real numbers are stored in the computer in their approximate form.