1 Basic Computing Concepts (4) Data Representations

The Binary System

The **Binary** System is a way of writing numbers using only the digits 0 and 1. This is the method used by the (**digital**) computer.

The system we use to write numbers is called **decimal** (or **denary**).

In the DECIMAL system “fifty two” is written as 52.
In the BINARY system “fifty two” is written as 110100.

Converting from Binary to Decimal

How can we convert numbers from the binary system to the decimal system? Follow this example. We have 10110011. Note that this number is a sequence of 8 **bits**, so it is a **byte**. Add the weights as seen in the diagram. Add all weights that are associated with the bits that are equal to 1.

Weights	128	64	32	16	8	4	2	1
Number	1	0	0	1	0	1	1	0

Add 128 + 16 + 4 + 2. This amounts to 150.

Sometimes we use **subscripts** to indicate whether a number is binary of decimal as shown here:

A binary number: 10010110_2
A decimal number: 150_{10}

The method we have seen here is called the **positional notation method**.

Exercise

Convert the following binary numbers to decimal.

a. 11001
b. 11011101
c. 1110010

Converting from Decimal to Binary

Suppose we want to convert the decimal number 149 to binary. One way we can do this is by repeatedly dividing the number by 2 and then taking the remainder digits to get the solution. An example is shown here:

2	149		
2	74	remainder	1
2	37	remainder	0
2	18	remainder	1
2	9	remainder	0
2	4	remainder	1
2	2	remainder	0
2	1	remainder	0
	0	remainder	1
	Quotient after dividing by 2		Remainder after dividing by 2

After performing this process you have to read the remainders FROM BOTTOM TO TOP. This gives us that the decimal number 149 is equal to the binary number 10010101.

Exercise

Convert the following decimal numbers to binary.

a. 108
b. 53
c. 74

Number Bases

Positional number systems depend on a special number called a **base**. A **positional** number system (like the decimal and binary systems) is such that a digit represents a number according to the position where it is e.g. in the number 2725 the first ‘2’ (from the left) represents 2000 while the second ‘2’ represents 20.

Note that:
2725 = 2x1000 + 7x100 + 2x10 + 5.
2725 = 2x$\mathbf{10^3}$ + 7x$\mathbf{10^2}$ + 2x$\mathbf{10^1}$ + 5x$\mathbf{10^0}$.

The base number of the decimal system is 10.

Working on the same principles we can show that the base number of the binary system is 2.

Let us investigate the number 10010110_2.
10010110 = 1x128 + 0x64 + 0x32 + 1x16 + 0x8 + 1x4 + 1x2 + 0x1.
10010110 = 1x2^7 + 0x2^6 + 0x2^5 + 1x2^4 + 0x2^3 + 1x2^2 + 1x2^1 + 0x2^0.

Note also that the base number shows us how many different symbols are required to express a number i.e. in the decimal system we need 10 symbols (0, 1, 2, 3, 4, 5, 6, 7, 8 and 9) while in the binary system we need 2 (0 and 1).

Adding and Subtracting Binary Numbers

The following example shows two numbers being added:

Carry							
First number	0	0	1	1	1	1	0
Second number	1	0	0	1	0	1	1
Result of addition							1

Step 1:
0 + 1 gives 1

Carry					1		
First number	0	0	1	1	1	1	0
Second number	1	0	0	1	0	1	1
Result of addition						0	1

Step 2:
1 + 1 gives 0 carry 1

Carry				1	1		
First number	0	0	1	1	1	1	0
Second number	1	0	0	1	0	1	1
Result of addition					0	0	1

Step 3:
1 + 0 + 1 gives 0
carry 1

Carry			1	1	1		
First number	0	0	1	1	1	1	0
Second number	1	0	0	1	0	1	1
Result of addition				1	0	0	1

Step 4:
1 + 1 + 1 gives 1
carry 1

Proceeding in this way we get:

Carry		1	1	1	1		
First number	0	0	1	1	1	1	0
Second number	1	0	0	1	0	1	1
Result of addition	1	1	0	1	0	0	1

30

75

105

Exercise in Addition

Add the following binary numbers. Then check your result by using decimal numbers.

a. 1010001 and 0101110
b. 1011011 and 1011100

The following example shows subtraction:

Borrow							
First number	1	0	1	1	0	1	1
Second number	0	1	0	1	1	1	0
Result of subtraction							1

Step 1:
1 – 0 gives 1

Borrow							
First number	1	0	1	1	0	1	1
Second number	0	1	0	1	1	1	0
Result of subtraction						0	1

Step 2:
1 – 1 gives 0

Borrow					2		
First number	1	0	1	~~1~~	0	1	1
Second number	0	1	0	1	1	1	0
Result of subtraction					1	0	1

Step 3:
0 - 1 means that we have to borrow

Proceeding in this way we get

Borrow		2		2	2		
First number	~~1~~	0	~~1~~	~~1~~	0	1	1
Second number	0	1	0	1	1	1	0
Result of subtraction	0	1	0	1	1	0	1

91

46

45

Exercise in Subtraction

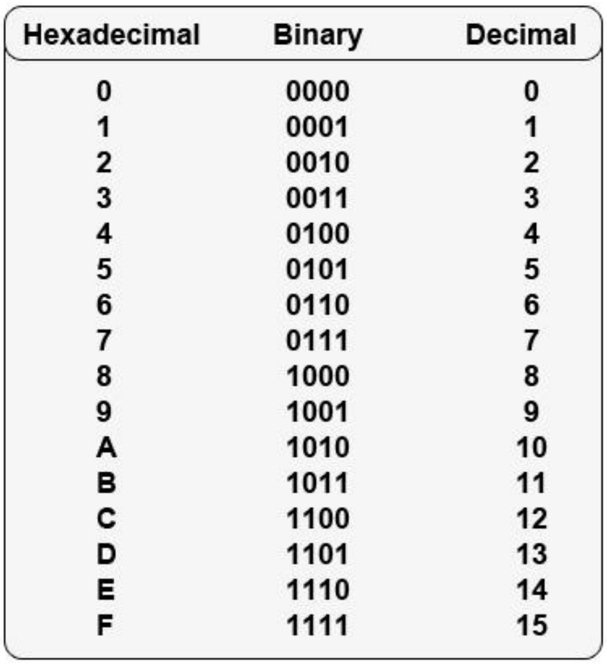
Perform the following binary subtractions. Then check your results by using decimal numbers:

a. 10001101 – 00101100
b. 10000101 – 01010111

Hexadecimal Numbers

Hexadecimal numbers are based on the number 16. A hexadecimal number uses 16 symbols which are 0, 1, 2 … 8, 9, A, B, C, D, E and F. A stands for ten, B stands for eleven, C stands for twelve etc. The term 'hexadecimal' is shortened to 'hex'. Hex numbers are used as a shortcut notation for binary numbers. The conversions from binary to hex and vice-versa are very simple and straightforward.

In the following table we can see the representations of the numbers from zero to fifteen in all three bases.

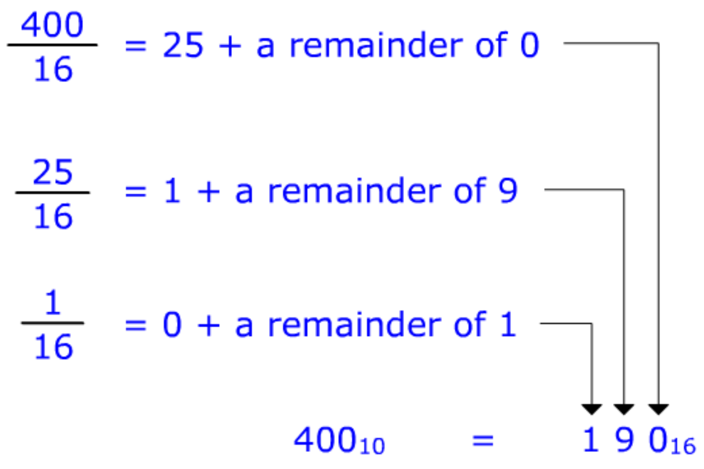


Hexadecimal	Binary	Decimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

An **example** of a hex (hexadecimal) number: 3C90B.

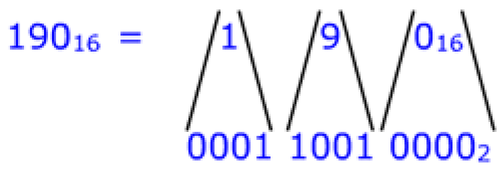
Conversion from decimal to hexadecimal

The following example shows how the decimal number 400 is converted to hex.


$\frac{400}{16}$ = 25 + a remainder of 0
$\frac{25}{16}$ = 1 + a remainder of 9
$\frac{1}{16}$ = 0 + a remainder of 1
400_{10} = $1\,9\,0_{16}$

Conversion from Hex to Binary

In the following example the hex number 190 is converted to binary.



Leading zeros, in binary numbers, may be omitted:

$$000110010000_2 = 110010000_2$$

Conversion from binary to hex

It is easy to convert from an integer binary number to hex. This is accomplished by the following two steps:

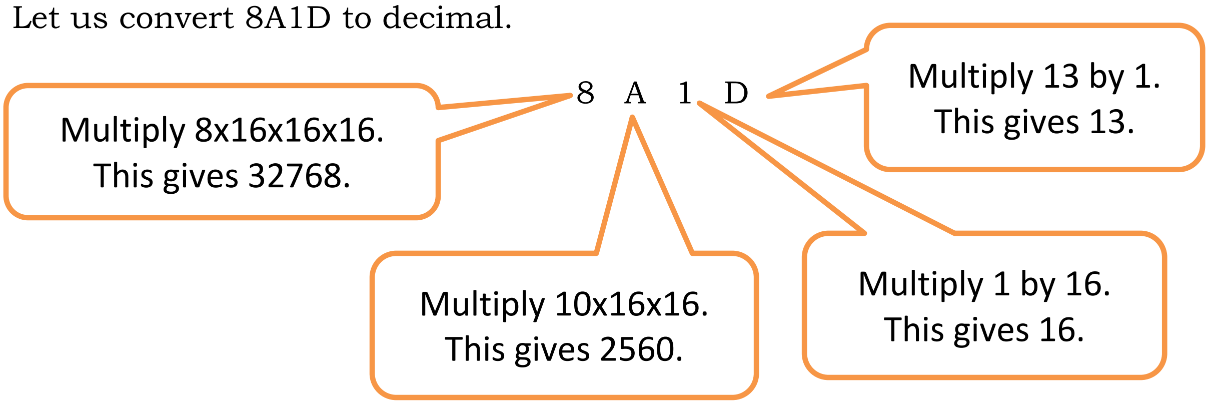
1. Break the binary number into 4-bit sections from the LSB (least significant bit) to the MSB (most significant bit).
2. Convert the 4-bit binary number to its Hex equivalent.

The following example shows how the binary value 1010111110110010 is converted to hex.

1010	**1111**	**1011**	**0010**
A	**F**	**B**	**2**

Conversion from hex to decimal

Let us convert 8A1D to decimal.



8 A 1 D
Multiply 8x16x16x16. This gives 32768.
Multiply 10x16x16. This gives 2560.
Multiply 1 by 16. This gives 16.
Multiply 13 by 1. This gives 13.

Finally add 32768 + 2560 + 16 + 13. This gives 35357.

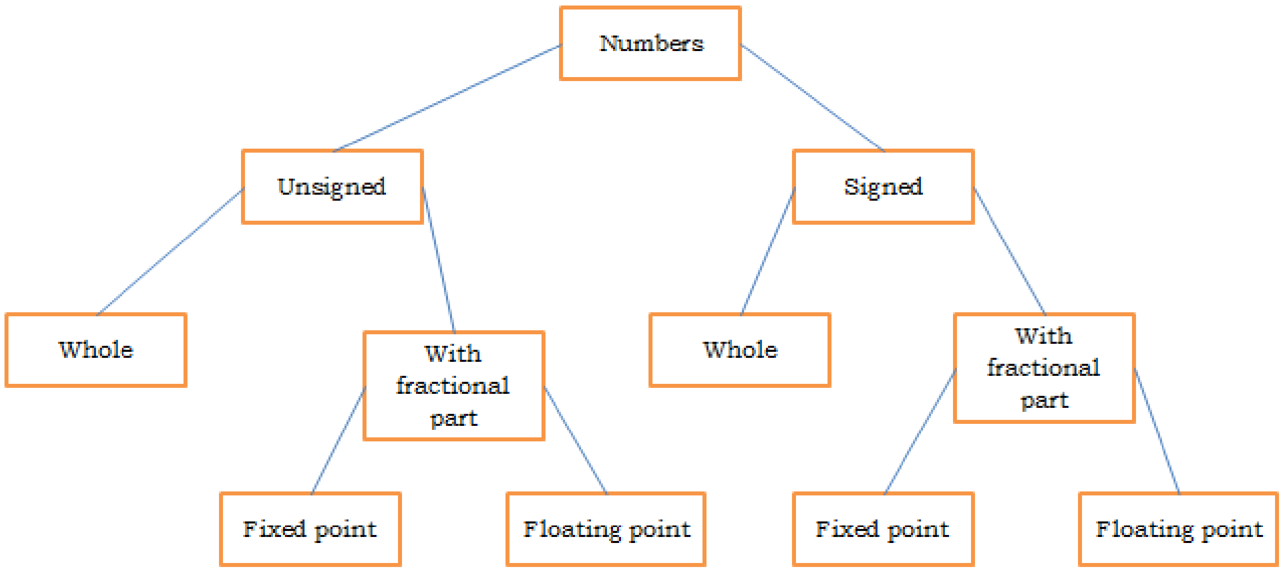
Exercise

Perform the following conversions:

a. Convert 10001110101_2 to hex.
b. Convert 20154_{10} to hex.
c. Convert $3F7A_{16}$ to decimal.
d. Convert $5BA_{16}$ to binary.

Classification of numbers

Numbers can be classified as shown in the following tree structure. Unsigned numbers are always positive while signed numbers can be positive or negative.



Consider three bits. There are eight different combinations of ones and zeros (these are 000, 001, 010, 011, 100, 101, 110 and 111). These eight combinations can be made to represent the range of numbers from 0 to 7 or the numbers from -4 to 3. We say that the range “from 0 to 7” is **unsigned** and the range “from -4 to 3” is **signed**.

Sign and Magnitude

In the Sign and Magnitude representation the leftmost bit is reserved to represent the sign. 1 indicates negative and 0 indicates positive. Let us take an example with 8 bits.

sign	**64**	**32**	**16**	**8**	**4**	**2**	**1**	
0	1	0	0	1	1	0	1	77_{10}
1	1	0	0	1	1	0	1	-77_{10}

Now let us consider 4 bits. All the numbers are represented in the following table.

Binary Sign & Magnitude	Decimal
1111	-7
1110	-6
1101	-5
1100	-4
1011	-3
1010	-2
1001	-1

1000	-0
0000	+0
0001	+1
0010	+2
0011	+3
0100	+4
0101	+5
0110	+6
0111	+7

Note that the value zero is repeated twice. The range of values is between -7 and 7. If we had one byte the minimum value would be 11111111 (-127_{10}) and the maximum would be 01111111 (127_{10}).

Apart from the repeated zero another problem with sign and magnitude representation is that it is not good for computation e.g. making an addition of two numbers etc.

Two's Complement

Another representation scheme to represent both positive and negative numbers is called two's complement. It is now the nearly universal way of doing this. Integers are represented in a fixed number of bits. Both positive and negative integers can be represented.

How to Construct the Negative of an Integer in Two's Complement:

Start with an N-bit representation of an integer.

To calculate the N-bit representation of the negative integer:

1. Change every 0 to 1 and every 1 to 0).
2. Add one.

Suppose we have one byte to represent a number, say -87_{10}, in two's complement. This would be done in the following steps:

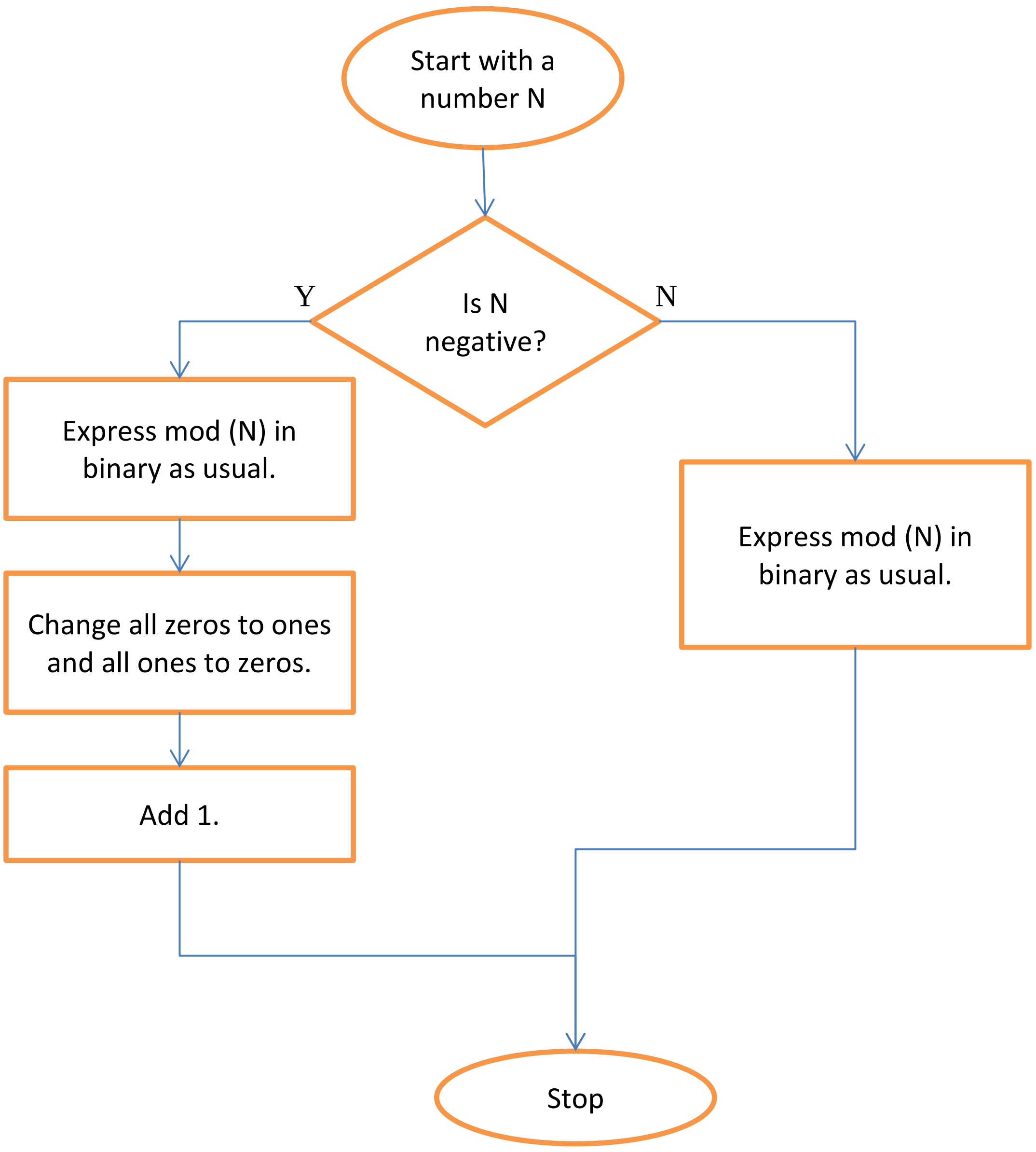
(1) Write 87_{10} in binary. This gives 01010111.

(2) Change all ones to zeros and all zeros to ones. This gives 10101000 (this number is called one's complement)

(3) Add 1. 10101000 + 1 gives 10101001.

Therefore $-87_{10} = 10101001_2$ (in two's complement).

To express a decimal number is two's complement follow the algorithm:



Start with a number N
Is N negative?
Y
N
Express mod (N) in binary as usual.
Change all zeros to ones and all ones to zeros.
Add 1.
Express mod (N) in binary as usual.
Stop

Exercise

Express the following decimal numbers both in sign-and-magnitude and in two's complement (assume one byte of space): (a) 43 (b) -99 (c) -106

Faster Computation of Two's Complement

To work out the two's complement of a number do the following:

1. Start from the right hand side and move to the left hand side by considering all the bits.

2. If a bit is equal to zero write a zero, move to the next bit on the left and repeat step 2. If it is equal to one write a 1 and go to step 3.
3. From the next digit on the left to the leftmost digit convert all bits from 0 to 1 and from 1 to 0.

Example 1

Express 57_{10} in two's complement.

$57_{10} = 00111001_2$

Example 2

Express -104_{10} in two's complement.

$104_{10} = 01101000_2$
$-104_{10} = 10011000_2$

Subtraction using Two's Complement and Addition

Suppose we want to calculate 5-3. We follow these steps:

1. 5 is converted to binary. Call this f.
2. 3 is converted to binary (two's complement). Call this t.
3. Add f and t but ignore the overflow bit.
4. You have the result.

```
5 + (-3) = 2      0000 0101  =  +5
                + 1111 1101  =  -3
                ___________
                  0000 0010  =  +2
```

Another example:

```
7 - 12 = (-5)     0000 0111  =  +7
                + 1111 0100  =  -12
                ___________
                  1111 1011  =  -5
```

Exercise

Perform the following calculations using two's complement arithmetic. All numbers are memorized in one byte.

(a) 99 – 56, (b) 85 – 106, (c) -25 – 30

Fractions in Fixed-Point Number Representation

Suppose we know that the number 1011.0110_2 is unsigned, what is its conversion in decimal?

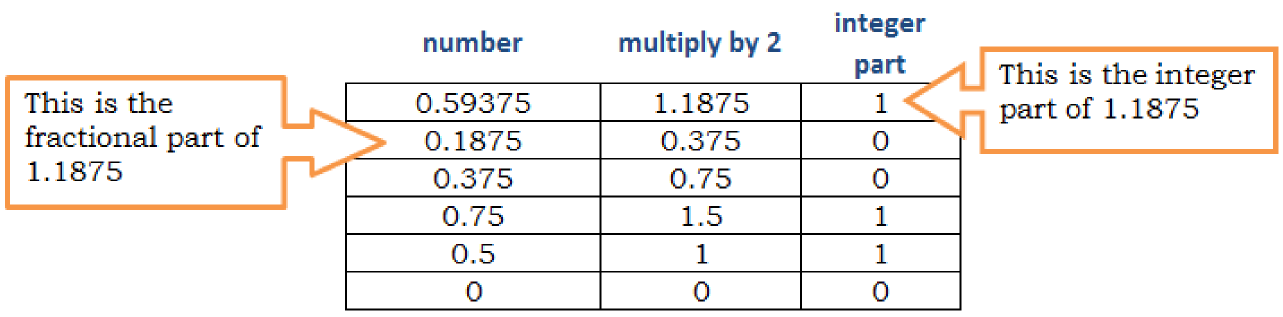
$$\begin{aligned} 1011.0110_2 &= 1\text{x}2^3 + 0\text{x}2^2 + 1\text{x}2^1 + 0\text{x}2^0 + 0\text{x}2^{-1} + 1\text{x}2^{-2} + 1\text{x}2^{-3} + 0\text{x}2^{-4} \\ &= 8 + 0 + 2 + 1 + 0 + 0.25 + 0.125 + 0 \\ &= 11.375_{10} \end{aligned}$$

In this representation (i.e. we have 8 bits and the decimal point is fixed after the fourth bit; this is called **fixed-point representation**):

- the maximum number is 1111.1111 (15.9375), and
- the minimum number is 0000.0000

What we have seen above is the conversion of a binary number (with a fractional part) into decimal representation. How do we do the opposite? Suppose we want to convert 13.59375 to binary we execute the following steps:

1. Convert the whole part to binary i.e. $13_{10} = 1011_2$
2. Convert the decimal part to binary. This is done using the algorithm shown in the diagram below. So $0.59375_{10} = 0.10011_2$ (read the last column from top to bottom)


This is the fractional part of 1.1875
This is the integer part of 1.1875

number	multiply by 2	integer part
0.59375	1.1875	1
0.1875	0.375	0
0.375	0.75	0
0.75	1.5	1
0.5	1	1
0	0	0

3. Join the parts i.e. 1011.10011

Exercise

(a) Convert the following binary numbers to decimal (i) 11001.011_2, (ii) 10011.10011_2

(b) Convert the following decimal numbers to binary (i) 23.6875_{10}, (ii) 87.15625_{10}

Maximum and Minimum Values and Ranges

The following table shows some maximum and minimum values (on one byte).

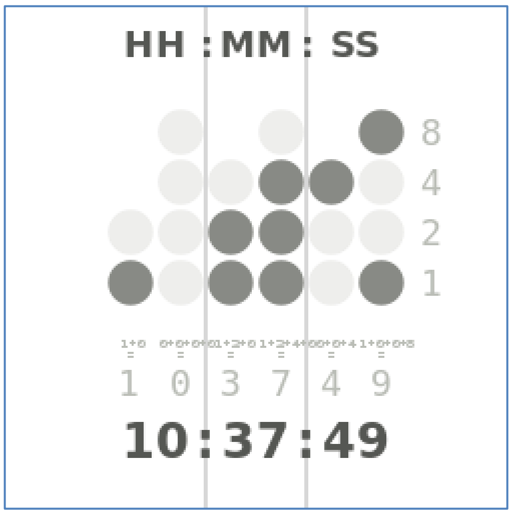
	Minimum		Maximum	
	Decimal	Binary	Decimal	Binary
Unsigned integer (i.e. whole number)	0	00000000	255	11111111
Signed integer (sign and magnitude)	-127	11111111	127	01111111
Signed integer (two's complement)	-128	10000000	127	01111111
Unsigned fractional number with fixed point representation (binary point is three positions from the right)	0	00000.000	31.875	11111.111
Signed fractional number with fixed point representation (binary point is three positions from the right, sign and magnitude)	-15.875	11111.111	15.875	01111.111
Unsigned fractional number with fixed point representation (binary point is three positions from the right, two's complement)	-16	10000.000	15.875	01111.111

Exercise

In each of the following questions find the maximum, minimum and the range of the numbers. Assume that the numbers are made up of 6 bits.

(a) An unsigned binary integer.
(b) Signed integer (sign and magnitude).
(c) Signed integer (two's complement).
(d) Unsigned fractional number with fixed point representation (binary point is two positions from the right).
(e) Signed fractional number with fixed point representation (binary point is two positions from the right, sign and magnitude)
(f) Unsigned fractional number with fixed point representation (binary point is two positions from the right, two's complement)

8421 Binary-Coded Decimal



Abbreviated as **8421 BCD**, **binary-coded decimal** is a format for representing decimal numbers (integers) in which each digit is represented by four bits (a nibble). For example, the number 375 would be represented as: 0011 0111 0101.

One advantage of BCD over binary representations is that there is no limit to the size of a number. To add another digit, you just need to add a new 4-bit sequence. In contrast, numbers represented in binary format are generally limited to the largest number that can be represented by 8, 16, 32 or 64 bits.

Another virtue of BCD is that it is a more accurate representation (no rounding of decimal quantities). As compared to binary positional systems, BCD's principal drawbacks are a small increase in the complexity of the circuits needed to implement basic arithmetic and a slightly less dense storage.

BCD is very common in electronic systems where a numeric value is to be displayed. The BIOS in many personal computers stores the date and time in BCD.

Exercise

(a) Convert the following decimal numbers in 8421 BCD: (i) 38_{10}, (ii) 109_{10}.
(b) Convert the following BCD number in decimal: 0111 1001 0001.

ASCII and Unicode

Both ASCII and Unicode are codes that convert characters (like 'n', 'N', '@', '+' etc) and control codes (like 'backspace' and 'line feed' which causes a printer to advance its paper). ASCII is an acronym for the 'American Standard Code for Information Interchange'. ASCII is a code for representing English characters as numbers, with each letter assigned a number from 0 to 127 (using 7 bits). For example, the ASCII code for uppercase M is 77. Most computers use ASCII codes to represent text, which makes it possible to transfer data from one computer to another.

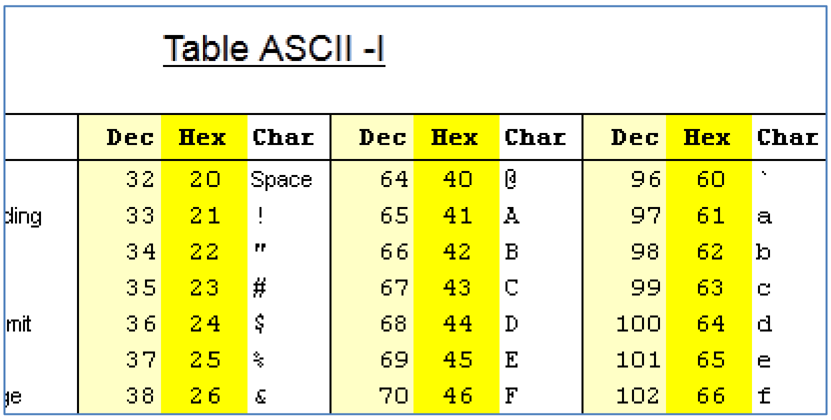
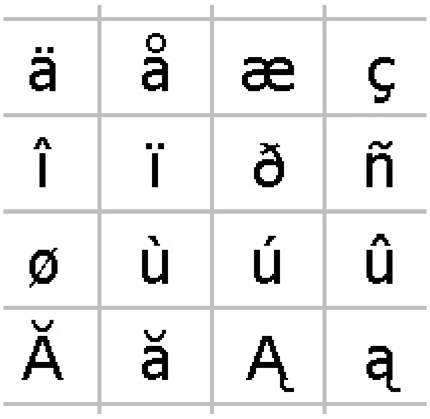


Table ASCII -I

	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
	32	20	Space	64	40	@	96	60	`
ding	33	21	!	65	41	A	97	61	a
	34	22	"	66	42	B	98	62	b
	35	23	#	67	43	C	99	63	c
mit	36	24	$	68	44	D	100	64	d
	37	25	%	69	45	E	101	65	e
je	38	26	&	70	46	F	102	66	f

Text files stored in ASCII format are sometimes called ASCII files.

The standard ASCII character set uses just 7 bits for each character. There are several larger character sets that use 8 bits, which gives them 128 additional characters. The extra characters are used to represent non-English characters (like ö, ū, etc), graphics symbols (like □, ◊, ●, etc), and mathematical symbols (like ⅛, √, ∫, etc). The DOS operating system uses a superset of ASCII called 'extended ASCII' or 'high ASCII'. Extended ASCII uses 8 bits to represent a character. Another ASCII derivative is 'ISO Latin 1'.



ä	å	æ	ç
î	ï	ð	ñ
ø	ù	ú	û
Ă	ă	Ą	ą

Another set of codes that is used on large IBM computers is EBCDIC (Extended Binary-Coded Decimal Interchange Code). The EBCDIC uses 8-bit representations.

Unicode uses 16 bits, which means that it can represent more than 65,000 unique characters. This is necessary so that other languages, such as Greek, Chinese and Japanese can have their symbols represented.