original author: Feisal Mohammed; updated: January 30, 2007– CLR

Part IV
Numbers & Data

There are several issues connected with the manipulation of real numbers, strings and other forms of data by computers.

The first is how such data should be represented; the second is the means by which the data should be manipulated; and the third is the bit/byte/word/record-ordering of data within memory to accommodate the representation.

In this section, we will address these issues for two's complement, binary-coded decimal, and floating point representations of numbers, ASCII representation of strings, and storage of composite records.

11 Real numbers

At the end of this unit the student will be able to:

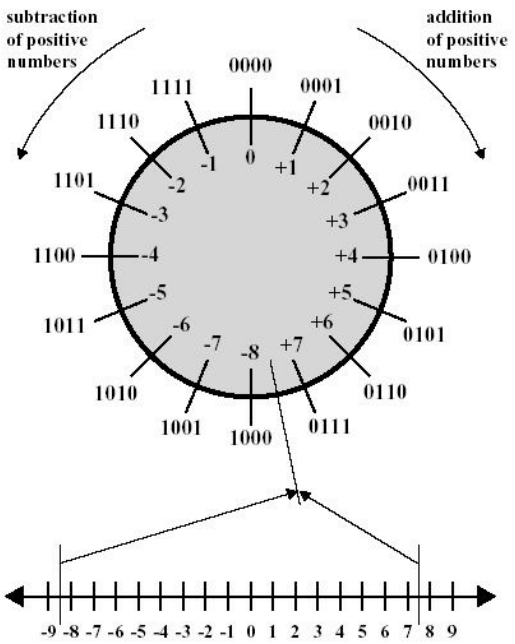
> *represent real numbers using fixed and floating point binary representations.*

Two's Complement The most commonly found representation of integer numbers in computers is the two's complement representation. In this representation, the full range of binary numbers (2^n for n bits) is split between positive and negative numbers (-2^{n-1} to $2^{n-1}-1$). Positive numbers are the same as their unsigned representations. To find the representation of a negative number, invert the bits of the unsigned binary representation and then add 1. Using this representation, all subtractive operations become additions with a negative number. Because the uppermost bit of a two's complement number is always 0 for positive numbers[22], and 1 for negative numbers, this bit is often referred to as the sign bit.

Changing the bit width of a two's complement number can be done by sign reduction or extension i.e. removing locations which are the same as the sign bit, or copying the sign bits to any additional bit locations. In practice, representation of a number using more bits involves the use of multiple data words.

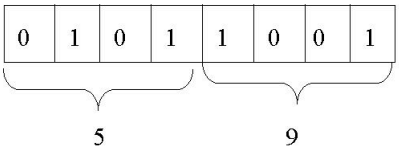
Binary Coded Decimal Binary coded decimal representation of numbers, uses a 4 bit binary number to represent the decimal digits 0-9. It is clearly an inefficient representation, as 6 of the numbers which could be represented by 4 bits are never used. The advantage of using such a scheme lies in the fact that most human I/O involves the use of decimal numbers. The use of BCD reduces the overhead of conversion at the interface-level. The disadvantage is that mathematical operations become more complex, as the manipulation of individual binary digits may yield an invalid 4 bit BCD code. Further there is no accommodation for signed quantities. Subsequently BCD arithmetic

[22] Zero is assumed to be a positive number.



BCD

- Decimal: Facilitates easy conversion to/from human input
- BCD: Each digit represented by an n-bit binary number
- Packed BCD: 2 digits represented in a single byte



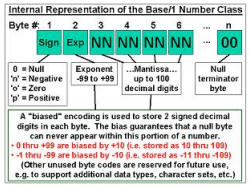
From: http://www.mindspring.com/~jc1/serial/Resources/ASCII.html

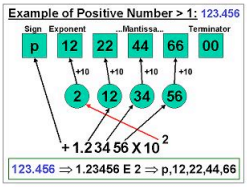
ASCII

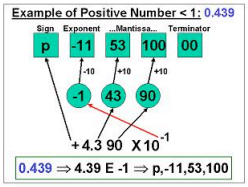
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2	SPC	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

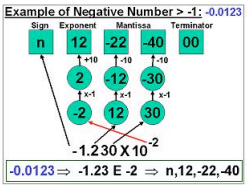
- Visible Character Codes
- Control Codes
  - Physical Communication Control
  - Logical Communication Control
  - Physical Device Control
  - Physical Device Format Effects
  - Code Extenders
  - Information Separators

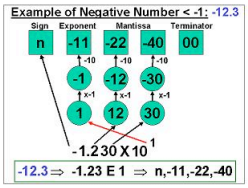
http://www.boic.com/numrep.htm











The Base/1 Number Class uses a proprietary internal representation to support arithmetic on very large numbers and exact fractions of up to 100 decimal digits. Base One's design makes it possible to store high precision numbers compactly without sacrificing computational efficiency. (U.S. Patent Number 6,384,748)

operations are generally restricted to addition and subtraction of unsigned quantities. In *packed BCD*, two digits are continued within a single byte. Other representations *pad* the extra bits in the byte/word (i.e. leave the upper bits of the data word either unused or filled with a particular prefix) e.g. the ASCII[23] character set defines a representation of characters as 7 bit numbers. The digit characters 0 through 9 are represented as $011\ 0000_2$ thru $011\ 1001_2$.

Fixed point representations Integer representations are specific cases of fixed point representations i.e. numeric representations in which the position of the decimal point is assumed to be fixed; the digits above and below the decimal point are specified.

Floating point representations represent numbers using an assumed numeric *base* i.e. all numbers in a particular system, are presumed to have the same base. Numbers are represented by an *exponent* of the base, multiplied by a normalised number called the *mantissa* or *significand*. i.e. $\pm S \times B^{\pm E}$. Normalised numbers are justified so that the first significant digit appears at a fixed place e.g. 0.00001 and 0.1 and 100 may be expressed as 1×10^{-5}, 1×10^{-1} and 1×10^{2}.

The presence of sign in the significand or exponent may be represented separately, or by use of a *biased* numeric representation. A biased numeric representation is one which includes a fixed offset e.g. for an 8 bit number with a bias of 0x7F, the number -4 would be represented as 0x7B and the number 4 would be represented by 0x83.

The base used, the use of biased numbers vs. explicit sign representation, the number of bits used to represent the mantissa and exponent, and the ordering of bits within the representation depends on the particular system/designer/programmer. In computer systems the conventions are:

- the base is usually 2,
- the exponent is biased, and
- the sign of the significand is explicitly represented.

All fixed/floating point representations, regardless of the details, share the following flaws:

Representation error The precision with which a floating point number can be represented is determined by the number of bits used in the exponent and significand. Where the number of bits is insufficient, the number be be approximated by roundoff, or truncation. In either case there is some discrepancy between the number and it's representation.

Error propagation Arithmetic operations often magnify the effects of errors in the input quantities. e.g. 2.51 * 2.32 = 5.8232 BUT 2.5 * 2.3 = 5.75.

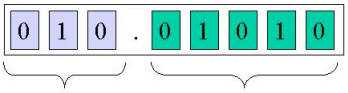
Non-unique 0 The floating point representation scheme is such that there is a choice of representations for zero.

References

Stallings, William. 2000. *Computer architecture and organization.* 5th ed. Prentice-Hall, Inc.

[23] American Standard Code for Information Interchange

Fixed point



- "Artificial" "binary point"
  - within the data-word
  - placement MUST be specified
- Bits on either side of "binary point" may be interpreted as integers or BCD
- Right side
  - BCD lists decimal fraction numerals
  - Integer uses fraction of maximum integer; in effect listing the binary fraction numerals e.g. .01 is 1 of 2; .010 is 2 of 4

Fixed point translation

Signed fixed point		Two's complement representation of signed fixed point
$+01001.001_2$	⟷	01001001_2
-01001.001_2	⟷	10110111_2
$+9.2_{16}$	⟷	49_{16}
-9.2_{16}	⟷	$B7_{16}$
$+9.125_{10}$	⟷	73_{10}
-9.125_{10}	⟷	183_{10}

When translating a negative fixed point number, keep in mind that the fraction is always positive e.g. 10110111_2 where we know we have 3 bits after the point.
The integer portion is 10110_2 in a 5 bit two's complement representation scheme i.e. -01010_2 which can be written as -10_{10}
The fractional portion is 111_2, i.e. $^7/_8 = 0.875_{10}$
$-10_{10} + 0.875_{10} = -9.125_{10}$

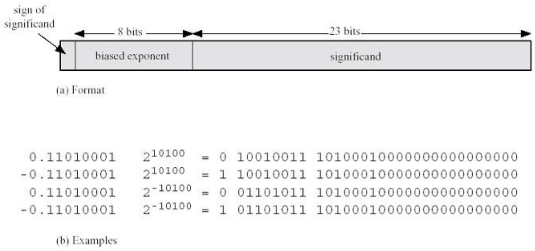


Figure 8.18 Typical 32-Bit Floating-Point Format

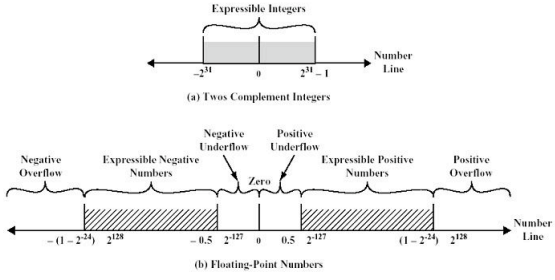
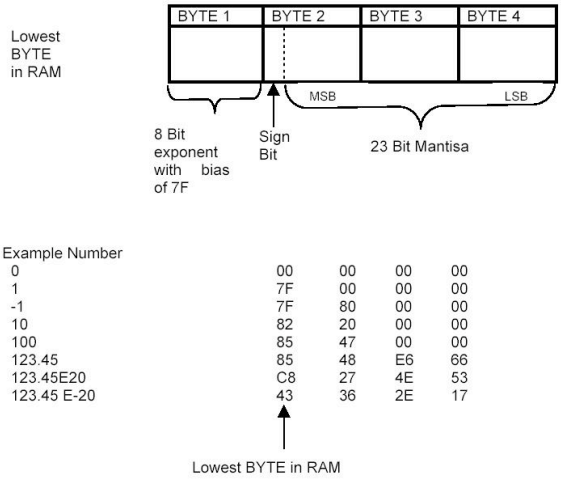


Figure 8.19 Expressible Numbers in Typical 32-Bit Formats



Changing bit-widths

- 2's complement
  - extend/reduce the sign bit
  - arithmetic shift in context
- BCD
  - fill with 0's at front
- ASCII
  - defined as 7-bit
- Floating point
  - fill significand (mantissa) at end
  - add the change in offset to exponent

Interpretation

1010 0011 1101 0101 0011 0111 0011 1010

- Unsigned binary number:
  $A3D5373A_{16}=2748659514_{10}$
- Signed binary number:
  $-5C2AC8C6_{16}= -1546307782_{10}$
- Unsigned fixed point(16/16):
  $A3D5.373A_{16}=41941.2157..._{10}$
- Floating point number(IEEE):
  $-0.11010101001101110011101_{2} \times 2^{0100\ 0111_{2}}$
- Visual C representation:
  2.74866e+009 [A3D53700]
- Questions:
  - How can we tell what type the data is?
  - How can we convert from one data type to another?
  - How can we interpret the data as another type?

Representation

Unsigned integer
- 01001001_2 written as a binary #
- 49_{16} written as a hexadecimal #
- 73_{10} written as a decimal #

Signed integer
- $+01001001_2$ written as a signed binary #
- $+49_{16}$ written as a signed hexadecimal #
- $+73_{10}$ written as a signed decimal #
- -01001001_2 written as a signed binary #
- -49_{16} written as a signed hexadecimal #
- -73_{10} written as a signed decimal #

Two's complement of an unsigned integer is also an unsigned integer

Two's complement
- 10110111_2 written as a binary #
- $B7_{16}$ written as a hexadecimal #
- 183_{10} written as a decimal #

Two's complement representation of a signed integer is not the same as the two's complement of an unsigned integer

Two's complement representation of signed integers and the Sign bit

In some representations there is a sign bit which is used to represent the plus + or minus - signs. The two's complement representation scheme does NOT have an EXPLICIT sign bit, but a feature of the scheme is that the first bit will IMPLICIT-ly reflect the sign of the number.

0000	0	positive sign bit
1001	-7	negative sign bit
0111	7	positive sign bit

Signed Integer		Two's complement Representation
$+01001001_2$	⟷	01001001_2
-01001001_2	⟷	10110111_2
$+49_{16}$	⟷	49_{16}
-49_{16}	⟷	$B7_{16}$
73_{10}	⟷	73_{10}
-73_{10}	⟷	183_{10}

Review Questions

1. What range of unsigned integer numbers can be represented in 6 bits?
   (a) -2^5 to $2^5 - 1$ ☐
   (b) $-2^5 + 1$ to $2^5 - 1$ ☐
   (c) -2^6 to $2^6 - 1$ ☐
   (d) $-2^6 + 1$ to $2^6 - 1$ ☐
   (e) 0 to $2^6 - 1$ ☐

2. If the byte 11010110_2 represented a packed BCD number, the BCD number would be:
   (a) $-2A_{16}$ ☐
   (b) -56_{10} ☐
   (c) $D6_{16}$ ☐
   (d) invalid ☐
   (e) none of the above ☐

3. If the dataword 11100010 represented a signed integer (two's complement representation scheme), the number would be:
   (a) $+1E_{16}$ ☐
   (b) $+E2_{16}$ ☐
   (c) -17_{16} ☐
   (d) $-1E_{16}$ ☐
   (e) none of the above ☐

4. Convert the following numbers (show your working):
   (a) 25_{10} to 2's complement representation, written as a hexadecimal number.
   (b) -73_{10} to 2's complement representation, written as a binary number.
   (c) -37_{16} to 2's complement representation, written as a binary number.
   (d) -12_{16} to 2's complement representation, written as a hexadecimal number.
   (e) -121_{10} to 2's complement representation, written as a hexadecimal number.
   (f) 97_{16} read as 2's complement representation, to a signed hexadecimal number.
   (g) 5.6_{10} to a binary fixed point representation with 3 bits (integer), and 5 bits (fraction).
   (h) $2.A_{16}$ to a binary fixed point representation with 3 bits (integer), and 5 bits (fraction).
   (i) 97_{16} read as signed binary fixed point representation with 2 bits (integer), and 6 bits (fraction); to a signed decimal number.
   (j) 97_{16} read as unsigned binary fixed point representation with 2 bits (integer), and 6 bits (fraction); to a signed hexadecimal number.

$$1101010111110010_2$$

5. Show how you would derive the base 10 equivalent of the number shown above, using diagrams, and/or calculations, if we interpret the value as:
   (a) an unsigned 16 bit integer number
   (b) a signed 16 bit integer number
   (c) a signed 16 bit fixed point number (9,13)
   (d) a 16 bit floating point number (explicit sign, biased exponent(base 2), significand normalised just after numeric point)

6. A PIC16F877 program needs to use certain non-integer numbers. For each number, suggest an appropriate way in which the number can be represented in a single file register, write down your representation, and determine what error there is between the actual and represented values.
   (a) 2.5
   (b) 0.333
   (c) 10000.345

7. You need to sort a list of signed integer numbers on a PIC16F877. Suggest an appropriate numeric representation for numbers in the list, and justify your answer if:
   (a) we wish to sort numbers by magnitude. e.g. (-5,4,3,-2,1,0)
   (b) we wish to sort from most negative to most positive. e.g.(-5,-2,0,1,3,4)

8. Any fixed/floating point representation used in a computer can represent only certain real numbers exactly; all others must be approximated. If A' is the stored value approximating the real value A then the relative error r is expressed as: $r = \frac{A-A'}{A}$.(Stallings 2000; 8.21–8.24)
   (a) Represent the decimal quantity +0.7 in the following formats, and determine the relative error for each representation.
      i. floating point format: base = 2; exponent: biased 4 bits; significand 7 bits.
      ii. fixed point format: 2 bits integer; 6 bits fraction.
   (b) If $A = 1.427$, find the relative error if A is truncated to 1.42 and if it is rounded to 1.43.
   (c) Numerical values A and B are stored in the computer as approximations A' and B'. Neglecting any further truncation or roundoff errors, show that the relative error of the product is approximately the sum of the relative errors in the factors.
   (d) One of the most serious errors in computer calculations occurs when two nearly equal numbers are subtracted. Consider $A = 0.22288$ and $B = 0.2221$. The computer truncates all values to four decimal digits. Thus $A' = 0.2228$ and $B' = 0.2221$.
      i. What are the relative errors for A' and B'?
      ii. What is the relative error for $C' = A' - B'$?

9. Typically floating point numbers are represented using a minimum of 32 bits.

   The IEEE standard 754 standard defines a "single" format 32 bit floating point number as a sign bit followed by an 8 bit biased exponent, followed by a 23 bit significand (representing the normalised 24 bit binary fraction 0.1xxx xxxx xxxx xxxx xxxx xxxx).

   Floating point numbers in the CCS compiler for the PIC16F877, place a biased 8 bit exponent first, followed by the sign bit and then a 23 bit significand (representing the normalised 24 bit binary fraction 0.1xxx xxxx xxxx xxxx xxxx xxxx).

   The difference is in the location of the sign bit. Suggest at least TWO possible advantages/disadvantages of placing the sign bit in either position.

10. Role Play/Challenge: Prove that

    (a) it is possible to write down the binary/octal equivalent of a hexadecimal number by individually translating each digit.

    (b) the n's complement can be calculated for any base-n number (Hint: 2's complement is for base 2 numbers).

    (c) it is possible to write down the 2's/8's complement of a hexadecimal number by individually translating each digit of the 16's complement.

    (d) (Stallings 2000; 8.29) every real number with a terminating binary representation (finite number of digits to the right of the binary point) also has a terminating decimal representation (finite number of digits to the right of the decimal point)