EEE 204 Introduction to Embedded Systems

Asst. Prof. Dr Mahmut AYKAÇ

NUMBER SYSTEMS AND DATA FORMATS

Number Systems and Data Formats

- Computers and digital systems understand logic 1 and 0 and process the information that consists of them.
- For some devices (Figure 1), Logic 1 corresponds to 5 volts, while logic 0 corresponds to about 0 volt. On the other hand, for another device(Figure 2), Logic 1 can be 3.3V or even may be less while Logic 0 is 0V. There are also different devices to represent Logic 1 with lower voltages to save power (Figure 3)
- By combining these two symbols in any consecutive combination or in different combinations; data, commands and different symbols are produced in a format that digital systems can understand.

Figure 1. PIC16877A

Figure 2. MSP430F5529

Figure 3. Intel Core i7

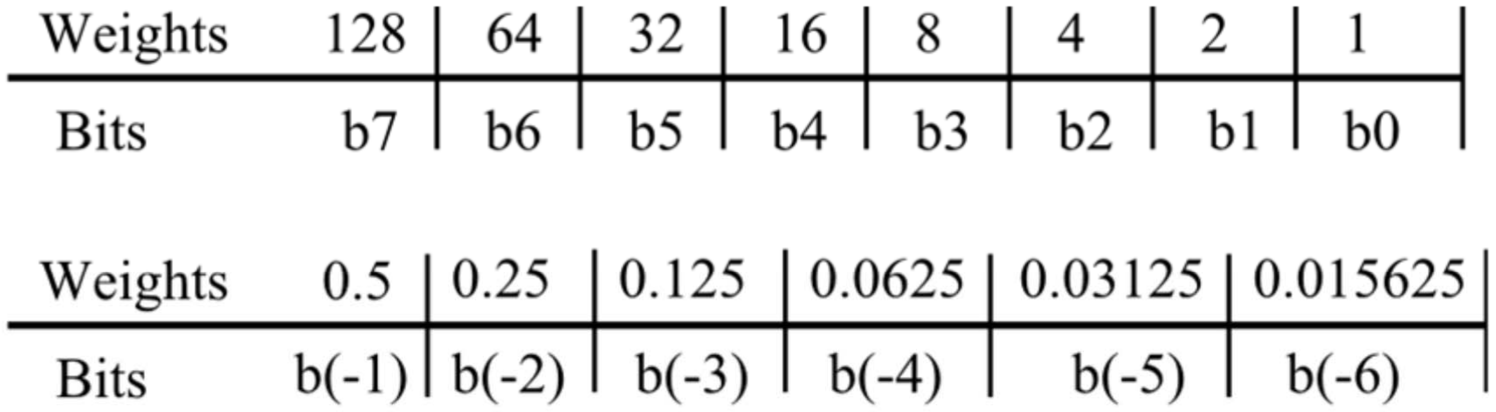
Bits, Bytes, and Words

- **Nibble:** 4-bit width
- **Byte:** 8-bit width
- **Word:** 16-bit width
- **Double Word:** 32-bit width
- **Quad:** 64-bit width

These lengths are associated to usual hardware. In the MSP430, for example, registers are 16 bits wide. Notice the specific use of the term "**word**" for 16 bits. Individual bits in a word are named after their position, starting from the right, bit 0 (b0), bit 1 (b1), and so on. Symbolically, an *n*-bit word is denoted as

$$b_{n-1}b_{n-2}\,.\,.\,.\,b_1b_0$$

Bits, Bytes, and Words



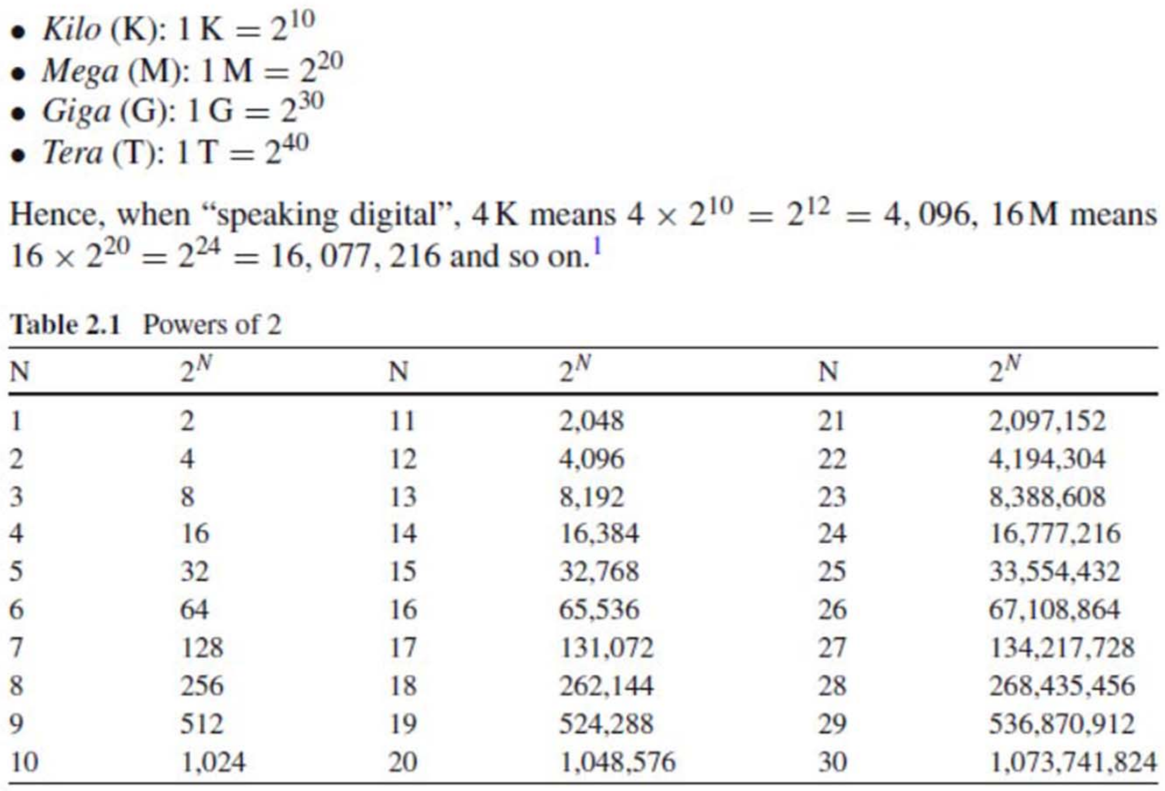
Weights	128	64	32	16	8	4	2	1
Bits	b7	b6	b5	b4	b3	b2	b1	b0

Weights	0.5	0.25	0.125	0.0625	0.03125	0.015625
Bits	b(-1)	b(-2)	b(-3)	b(-4)	b(-5)	b(-6)

Fig. 2.1 Weights for binary number $b_7b_6b_5b_4b_3b_2b_1b_0.b_{-1}b_{-2}b_{-3}b_{-4}b_{-5}b_{-6}$

The rightmost bit, *b*0, is the *least significant bit* (**lsb**), while the leftmost one, *bn*−1, is the *most significant bit* (**msb**).

Bits, Bytes, and Words



- *Kilo* (K): 1 K = 2^{10}
- *Mega* (M): 1 M = 2^{20}
- *Giga* (G): 1 G = 2^{30}
- *Tera* (T): 1 T = 2^{40}

Hence, when "speaking digital", 4 K means $4 \times 2^{10} = 2^{12} = 4,096$, 16 M means $16 \times 2^{20} = 2^{24} = 16,077,216$ and so on.[1]

Table 2.1 Powers of 2

N	2^N	N	2^N	N	2^N
1	2	11	2,048	21	2,097,152
2	4	12	4,096	22	4,194,304
3	8	13	8,192	23	8,388,608
4	16	14	16,384	24	16,777,216
5	32	15	32,768	25	33,554,432
6	64	16	65,536	26	67,108,864
7	128	17	131,072	27	134,217,728
8	256	18	262,144	28	268,435,456
9	512	19	524,288	29	536,870,912
10	1,024	20	1,048,576	30	1,073,741,824

Number Systems

Numbers can be represented in different ways using 0’s and 1’s. In this section we will talk about the most common conventions, starting with the normal binary notation, which is a positional numerical system. Our decimal system is positional, which means that any number is expressed by a permutation of digits and can be expanded as a weighted sum of powers of ten, the base of the system. Each digit contributes to the sum according to its position in the string. Thus, for example…

$$32.23 = 3 \times 10^1 + 2 \times 10^0 + 2 \times 10^{-1} + 3 \times 10^{-2},$$

$$578 = 8 \times 10^0 + 7 \times 10^1 + 5 \times 10^2$$

Number Systems

This concept can be generalized to any base. A fixed-radix, or fixed-point positional system of base r has r ordered digits 0, 1, 2, . . . "r−1". Number notations are composed of permutations of these r digits.

Base 2 : 0, 1, 10, 11, 100, 101, 110, . . . ;

Base 8 : 0, 1, 2, 3, 4, 5, 6, 7, 10, 11, . . . , 17, 20, . . . , 77, 100, 101, . . . ;

Base 12 : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, *A*, *B*, 10, 11, . . . , *BB*, 100, . . . , *BBB*, 1000, . . . ;

To avoid unnecessary complications, the same symbols are used for the digits 0, . . . , 9 in any base whenever are required, and letters *A, B, . . .* are digits with values of ten, eleven, etc. To distinguish between different bases, the radix is placed as a subscript to the string, as in 28_9, $1A_{16}$.

Number Systems

Numbers are written as a sequence of digits and a point, called radix point, which separates the integer from the fractional part of the number to the left and right side of the point, respectively.

$$a_{n-1}a_{n-2}\ldots a_1a_0.a_{-1}a_{-2}\ldots a_m$$

Here, each subscript stands for the exponent of the weight associated to the digit in the sum. The leftmost digit is referred to as the most significant digit (msd) and the rightmost one is the least significant digit (lsd). If there were no fractional part in equation the radix point would be omitted and the number would be called simply an integer. If it has no integer part, it is customary to include a “0” as the integer part. The number denoted by the equation represents a power series in r of the form.

Number Systems

$$\underbrace{a_{n-1}r^{n-1} + a_{n-2}r^{n-2} + \cdots + a_1 r^1 + a_0 r^0}_{\text{integer part}}$$

$$+ \underbrace{a_{-1}r^{-1} + \cdots + a_{-m}r^{-m}}_{\text{fractional part}} = \sum_{i=-m}^{n-1} a_i r^i$$

- Binary numbering system, base 2. For binary numbers, use suffix ‘B’ or ‘b’
- Octal numbering system, base 8. For octal numbers, use suffix ‘Q’ or ‘q’
- Hexadecimal numbering system, base 16. For hex numbers use suffix ‘H’ or ‘h’, or else prefix 0x. Numbers may not begin with a letter
- Base ten numbers have no suffix
- Hence, we write 1011B or 1011b instead of 10112, 25Q or 25q for 258, 0A5H or 0A5h or 0xA5 for A5$_{16}$.

Number Systems

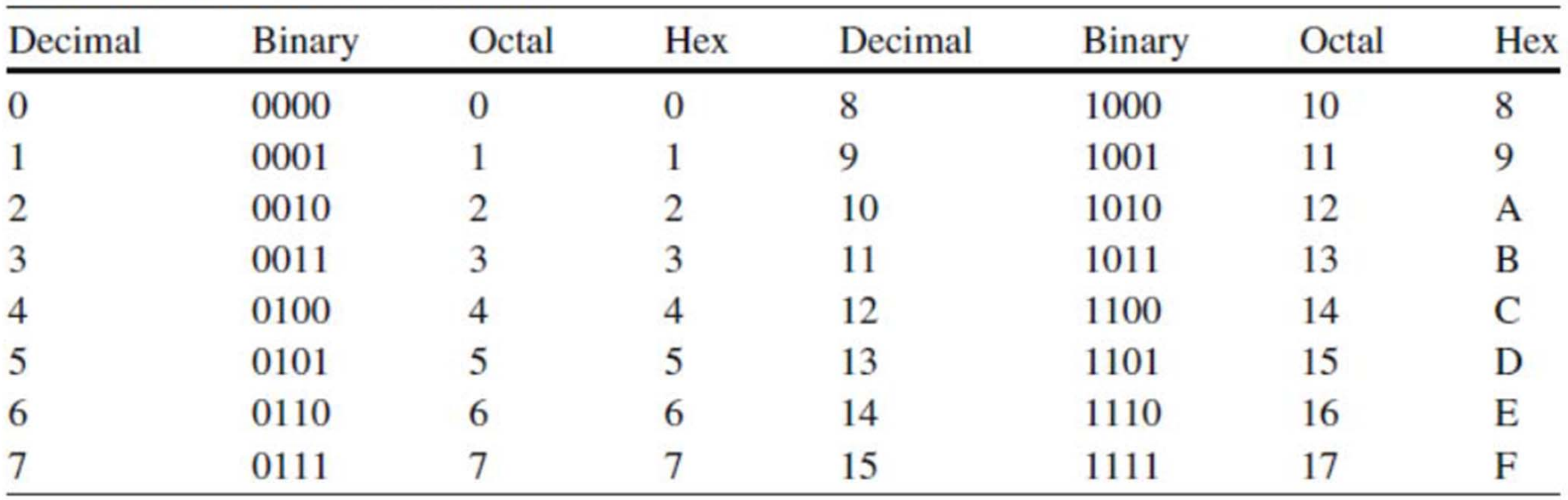


Table 2.2 Basic numeric systems

Decimal	Binary	Octal	Hex	Decimal	Binary	Octal	Hex
0	0000	0	0	8	1000	10	8
1	0001	1	1	9	1001	11	9
2	0010	2	2	10	1010	12	A
3	0011	3	3	11	1011	13	B
4	0100	4	4	12	1100	14	C
5	0101	5	5	13	1101	15	D
6	0110	6	6	14	1110	16	E
7	0111	7	7	15	1111	17	F

Conversion Between Different Bases: *Conversion from Base r to Decimal*

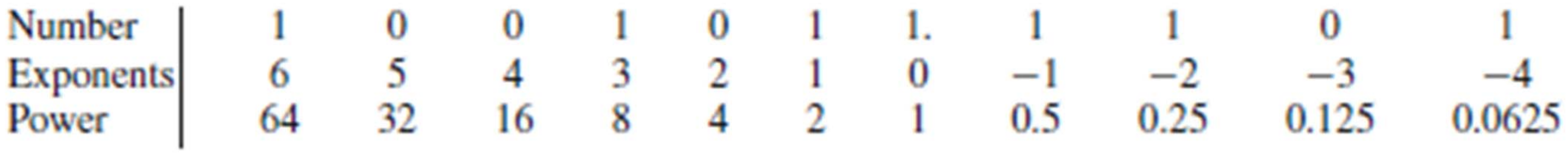
Ex: The following cases illustrate conversions to decimals:

$$
\begin{aligned}
214.23_5 &= 2 \times 5^2 + 1 \times 5^1 + 4 \times 5^0 + 2 \times 5^{-1} + 3 \times 5^{-2} = 59.52 \\
10110.01B &= 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 \\
&\quad + 0 \times 2^{-1} + 1 \times 2^{-2} = 22.25 \\
B65FH &= 11 \times 16^3 + 6 \times 16^2 + 5 \times 16^1 + 15 \times 16^0 = 46687
\end{aligned}
$$

!! Notice that for hexadecimal conversion, all hex digits are interpreted in their decimal values for the sum.

Conversion Between Different Bases: *Conversion from Base r to Decimal*

Ex: Let us convert 1001011.1101B to decimal using the powers as shown in the table.



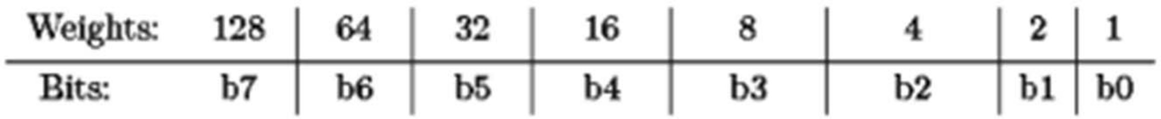
Number	1	0	0	1	0	1	1.	1	1	0	1
Exponents	6	5	4	3	2	1	0	−1	−2	−3	−4
Power	64	32	16	8	4	2	1	0.5	0.25	0.125	0.0625

Therefore, 1001011.1101B = 64 +8+2 +1+0.5+0.25+0.0625 = **75.8125**

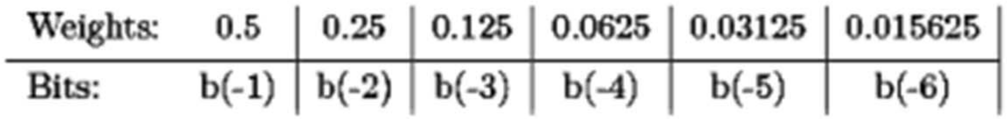
Conversion Between Different Bases: *Conversion from Decimal to Base r*

Integer Conversion

One popular procedure for converting decimal integers into base r is the repeated division method. This method is based on the division algorithm, and consists in successively dividing the number and quotients by the target radix r until the quotient is 0. The successive remainders of the divisions are the digits of the number in base r , starting from the least significant digit: divide the number by r and take the remainder as a_0; divide the quotient by r , and the remainder as a_1, and so on. Let us illustrate with a pair of examples.



Weights:	128	64	32	16	8	4	2	1
Bits:	b7	b6	b5	b4	b3	b2	b1	b0

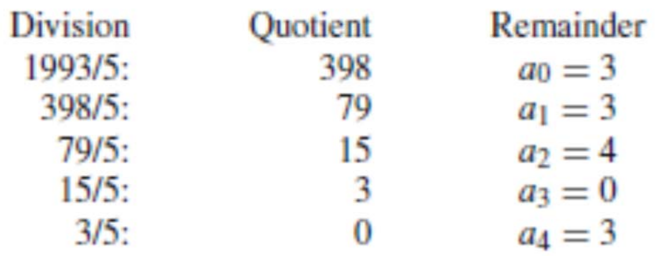


Weights:	0.5	0.25	0.125	0.0625	0.03125	0.015625
Bits:	b(-1)	b(-2)	b(-3)	b(-4)	b(-5)	b(-6)

Conversion Between Different Bases: *Conversion from Decimal to Base r*

Ex: Convert decimal 1993 to expressions in bases 5 and 16. To convert to base 5, divide by 5. The remainders are the digits of the number we are looking for, going from lsd to msd:

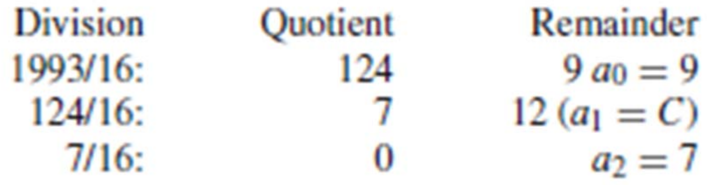
- To convert to base 5, divide by 5. The remainders are the digits of the number we are looking for, going from lsd to msd:



Division	Quotient	Remainder
1993/5:	398	$a_0 = 3$
398/5:	79	$a_1 = 3$
79/5:	15	$a_2 = 4$
15/5:	3	$a_3 = 0$
3/5:	0	$a_4 = 3$

Hence, 1993 = $\mathbf{30433_5}$

- To convert to base 16, repeat the procedure using 16 as divisor. If the remainder is greater than or equal to 10, convert to hex equivalent digit.



Division	Quotient	Remainder
1993/16:	124	9 $a_0 = 9$
124/16:	7	12 $(a_1 = C)$
7/16:	0	$a_2 = 7$

The result is then 1993 = $7C9_{16}$ = **7C9h.**

Conversion Between Different Bases: *Conversion from Decimal to Base r*

Fractional Part

Conversion of decimal fractions can be done by the repeated multiplication method: Multiplying the number by r , the integer part of the first product becomes most significant digit a_{-1}. Discard then the integer part and multiply again the new fractional part by r to get the next digit a_{-2}. The process continues until one of the following conditions is met.

- A zero decimal fraction is obtained, yielding a finite representation in radix r; or
- A previous fractional part is again obtained, having then found the periodic representation in radix r; or
- The expression in base r has the number of digits allocated for the process.

Ex: Convert the following decimal fractions to binary, limited to 8 digits if no periodicity appears before:

(a) 0.375, (b) 0.05, (c) 0.23. **(a) Let us begin with 0.375:**

$$\begin{aligned} 2 \times 0.375 &= 0.750 && \rightarrow a_{-1} = 0 \\ 2 \times 0.750 &= 1.50 && \rightarrow a_{-2} = 1 \\ 2 \times 0.5 &= 1.00 && \rightarrow a_{-3} = 1 \quad \text{Zero fractional part. Stop} \end{aligned}$$

Therefore, since the decimal fractional part in the last product is 0.00, the equivalent expression in binary is finite, specifically, **$0.375 = 0.011_2$ = 0.011B.**

Conversion Between Different Bases: *Conversion from Decimal to Base r*

(b) Converting 0.05:

$$
\begin{aligned}
2 \times 0.05 &= 0.1 \rightarrow a_{-1} = 0 \\
2 \times 0.10 &= 0.2 \rightarrow a_{-2} = 0 \\
2 \times 0.2 &= 0.4 \rightarrow a_{-3} = 0 \\
2 \times 0.4 &= 0.8 \rightarrow a_{-4} = 0 \\
2 \times 0.8 &= 1.6 \rightarrow a_{-5} = 1 \\
2 \times 0.6 &= 1.2 \rightarrow a_{-6} = 1 \quad \text{Repeatingfractionalpart0.2.Stop}
\end{aligned}
$$

The decimal fraction 0.2 in the last line has appeared before (third line), so the pattern 0011 will be periodic. Therefore, 0.05 = $0.00\overline{0011}_2$ = 0.0000110011 . . . B.

Conversion Between Different Bases: *Conversion from Decimal to Base r*

(c) Converting 0.23:

$$
\begin{aligned}
2 \times 0.23 &= 0.46 \rightarrow a_{-1} = 0 \\
2 \times 0.46 &= 0.92 \rightarrow a_{-2} = 0 \\
2 \times 0.92 &= 1.84 \rightarrow a_{-3} = 1 \\
2 \times 0.84 &= 1.68 \rightarrow a_{-4} = 1 \\
2 \times 0.68 &= 1.36 \rightarrow a_{-5} = 1 \\
2 \times 0.36 &= 0.72 \rightarrow a_{-6} = 0 \\
2 \times 0.72 &= 1.44 \rightarrow a_{-7} = 1 \\
2 \times 0.44 &= 0.88 \rightarrow a_{-8} = 0 \quad \text{Stop because of predefined limit.}
\end{aligned}
$$

We have reached 8 digits without finding the decimal fraction that will repeat. Within this approximation, 0.23 ≈ 0.00111010B. Predefined limit doesn’t have to be 8 digits. It can vary according to the memory limit.

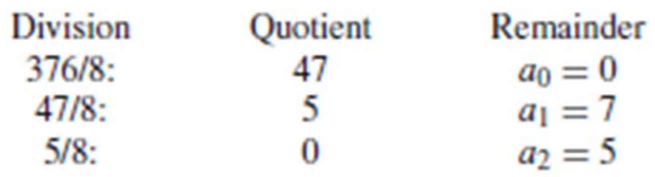
Conversion Between Different Bases: *Conversion from Decimal to Base r*

Mixed Numbers with Integer and Fractional Parts

In this case, the conversion is realized separately for each part. Let us consider an example.

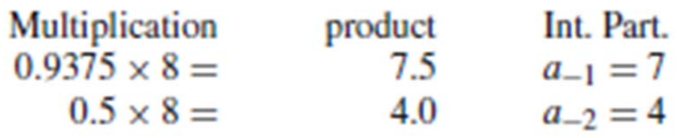
Ex: Convert 376.9375_{10} to base 8.

** First convert the integer part by successive divisions by 8 **



Division	Quotient	Remainder
376/8:	47	$a_0 = 0$
47/8:	5	$a_1 = 7$
5/8:	0	$a_2 = 5$

** We now convert the fractional part by successive multiplications **



Multiplication	product	Int. Part.
$0.9375 \times 8 =$	7.5	$a_{-1} = 7$
$0.5 \times 8 =$	4.0	$a_{-2} = 4$

which yields 3.9375 = 0.74Q. Therefore the final result is 376.9375 = **570.74Q**

Binary and Hexadecimal Systems

The lower the base, the more digits required to represent a number. The binary numerical system, the "natural" one for digital systems, is quite inconvenient for people. Associate each hex digit to four binary digits, from right to left in the integer part and from left to right in the fractional part.

$(12345)_{10}=(11\ 0000\ 0011\ 1001)_2=(3039)_H$

$(2748)_{10}=(1010\ 1011\ 1100)_2=(ABC)_H$

Ex: Convert (a) 0x4AD.16 to binary expression, convert (b) 00011111.0101 0100 to hexadecimal expression

(a) $\overbrace{0100}^{4}\overbrace{1010}^{A}\overbrace{1101}^{D}.\overbrace{0001}^{1}\overbrace{0110}^{6} \Rightarrow 10010101101.0001011B$

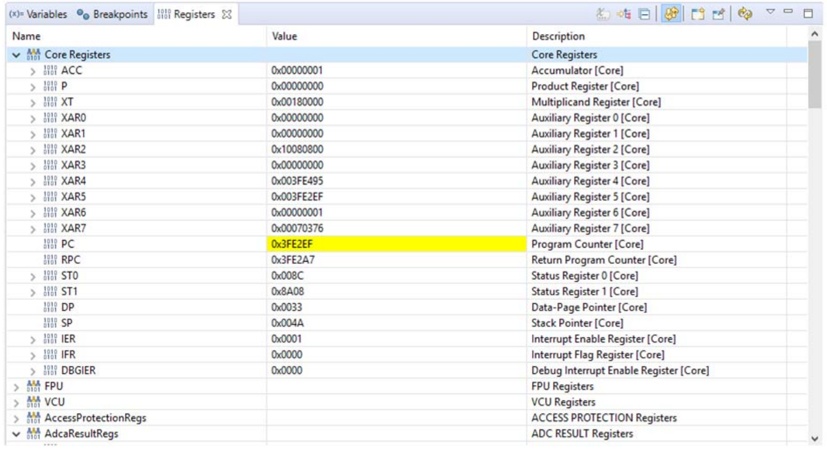
(b) $\overbrace{0001}^{1}\overbrace{1111}^{F}.\overbrace{0101}^{5}\overbrace{0100}^{4} \Rightarrow 1F.54H$

Binary and Hexadecimal Systems

There is a one to one correspondence between each hex digit and a group of four bits, a nibble.

- 1010011= 53h or 0x53
- 1101100110000010 =0xD982

Base 16 number system or hexadecimal number system has been universally adopted in the embedded systems literature as well as in debuggers. Thus, memory addresses, register contents, and pretty much everything else are expressed in hexadecimal integers without implying that they are a number.



Name	Value	Description
Core Registers		Core Registers
ACC	0x00000001	Accumulator [Core]
P	0x00000000	Product Register [Core]
XT	0x00180000	Multiplicand Register [Core]
XAR0	0x00000000	Auxiliary Register 0 [Core]
XAR1	0x00000000	Auxiliary Register 1 [Core]
XAR2	0x10080800	Auxiliary Register 2 [Core]
XAR3	0x00000000	Auxiliary Register 3 [Core]
XAR4	0x003FE495	Auxiliary Register 4 [Core]
XAR5	0x003FE2EF	Auxiliary Register 5 [Core]
XAR6	0x00000001	Auxiliary Register 6 [Core]
XAR7	0x00070376	Auxiliary Register 7 [Core]
PC	0x3FE2EF	Program Counter [Core]
RPC	0x3FE2A7	Return Program Counter [Core]
ST0	0x008C	Status Register 0 [Core]
ST1	0x8A08	Status Register 1 [Core]
DP	0x0033	Data-Page Pointer [Core]
SP	0x004A	Stack Pointer [Core]
IER	0x0001	Interrupt Enable Register [Core]
IFR	0x0000	Interrupt Flag Register [Core]
DBGIER	0x0000	Debug Interrupt Enable Register [Core]
FPU		FPU Registers
VCU		VCU Registers
AccessProtectionRegs		ACCESS PROTECTION Registers
AdcaResultRegs		ADC RESULT Registers

Figure. Example window to see the content of the registers

Unsigned Binary Arithmetic Operations

Addition

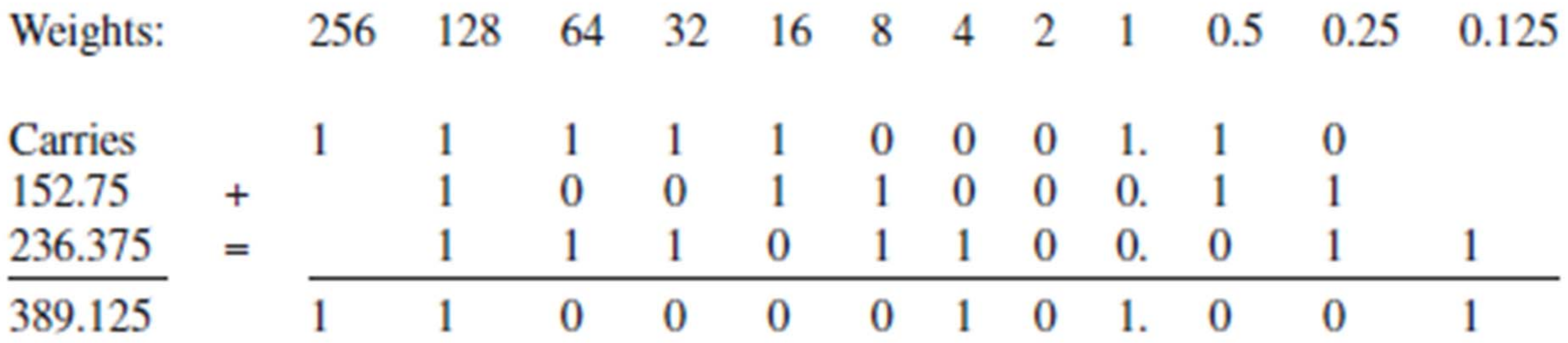
- 0 + 0 = 0
- 0 + 1 =1 + 0 = 1
- 1 + 1 = 10

Ex: Add the binary equivalents of 27 and 18

carries		→	1	0	0	1	0		
27	+	→		1	1	0	1	1	+
18	=	→		1	0	0	1	0	=
45		→	1	0	1	1	0	1	

Unsigned Binary Arithmetic Operations

Ex: Add the binary equivalents of 152.75 + 236.375.



Weights:		256	128	64	32	16	8	4	2	1	0.5	0.25	0.125
Carries		1	1	1	1	1	0	0	0	1.	1	0	
152.75	+		1	0	0	1	1	0	0	0.	1	1	
236.375	=		1	1	1	0	1	1	0	0.	0	1	1
389.125		1	1	0	0	0	0	1	0	1.	0	0	1

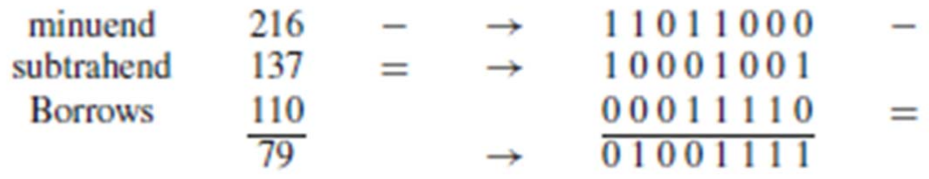
Unsigned Binary Arithmetic Operations

Subtraction

- 0 − 0 = 1 − 1 = 0
- 1 − 0 = 1
- 0 − 1 = 1 with a borrow

When a borrow is needed, it is taken from the next more significant digit of the minuend, from which the borrow should be subtracted. Hence, actual subtraction can be considered to be carried out using three digits: the minuend, the subtrahend, and the borrowed digit.

Ex: Subtract 137 from 216 using binary subtraction

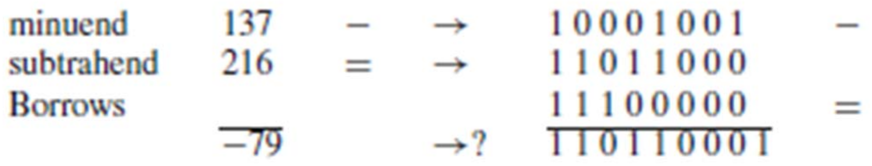


minuend	216	−	→	1 1 0 1 1 0 0 0	−
subtrahend	137	=	→	1 0 0 0 1 0 0 1	
Borrows	110			0 0 0 1 1 1 1 0	=
	79		→	0 1 0 0 1 1 1 1	

Unsigned Binary Arithmetic Operations

Subtraction

Ex: Subtract 216 from 137 using binary subtraction



minuend	137	−	→	1 0 0 0 1 0 0 1	−
subtrahend	216	=	→	1 1 0 1 1 0 0 0	
Borrows				1 1 1 0 0 0 0 0	=
	−79		→?	1 1 0 1 1 0 0 0 1	

Unsigned Binary Arithmetic Operations

Subtraction by Complement's Addition

Let us consider two decimal numbers A and B with the same number of digits, say N. If necessary, left zeros can be appended to one of them. From elementary Algebra, it follows that

$$A - B = [A + (10^N - B)] - 10^N$$

Ex: 127 − 31 = (127 + 969) − 1000 = 1096 − 1000 = 96

31 − 127 = (31 + 873) − 1000 = 904 − 1000 = −(1000 − 904) = −96

Unsigned Binary Arithmetic Operations

Two’s Complement Concept and Application

When considering two’s complements, we need to specify the number of bits. For example, with four bits, the two’s complement of 1010B is 10000B − 1010B = 0110B, but with 6 bits this becomes 1000000B − 1010B = 110110B. If you feel comfortable using decimal equivalents for powers of 2, these two examples can be thought as follows: 1010B is equivalent to decimal 10.

For four bits, 2^4 = 16 so the two’s complement becomes 16 − 10 = 6 = 0110B; with 6 bits, 2^6 = 64 and the two’s complement is 64 − 10 = 54 = 110110B.

Ex: The operations (a) 216 – 137 and (b) 137 – 216 using binary numbers and the two’s complement addition, and interpret accordingly, expressing your result in decimal numbers. The binary equivalents for the data are 216 = 11011000B and 137 = 10001001B, respectively. For the two’s complement, since both numbers have 8 bits, consider 2^8 = 256 as the reference, so the two’s complement of 216 is 256 − 216 = 40 = 00101000B, and that of 137 is 256 − 137 = 119 = 01110111B. With this information, we have then:

Unsigned Binary Arithmetic Operations

Two’s Complement Concept and Application

(A) For 216 – 137:

$$\begin{array}{r} 11011000\ + \\ 01110111\ = \\ \hline 101001111 \end{array}$$

Since the answer has a carry (9 bits), the result is positive. Dropping this carry, we have the solution to this subtraction $1\underbrace{01001111}$B = 79.

(B) (B) For 137 – 216:

$$\begin{array}{r} 10001001\ + \\ 00101000\ = \\ \hline 10110001 \end{array}$$

Since the sum does not yield a carry (8 bits), the result is negative, with an absolute value equal to its two’s complement. 10110001B = 177, *and the two’s complement is* 256 − 177 = 79. *Therefore, the solution is* −79, *as expected*

Unsigned Binary Arithmetic Operations

Calculating two’s Complements

The operand’s binary two’s complement was calculated in the previous example through the interpretation of the algorithm itself. Two common methods to find the two’s complement of a number expressed in binary form are mentioned next. The first one is practical for hardware realization and the second one is easy for fast hand conversion.

- **Invert-plus-sum**: Invert all bits (0–1 and vice versa) and add arithmetically 1.
- **Right-to-left-scan:** Invert only all the bits to the left of the rightmost ‘1’.

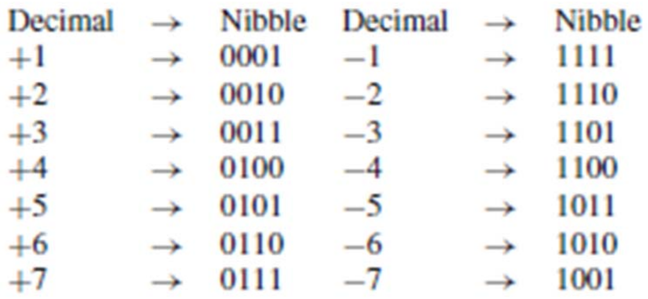
To illustrate with an example, assume that the six least significant bits of the original number are a 1 followed by five zeros, *****100000. After inverting the bits, all the bits to the right of the original rightmost 1 (now0)will be 1’s, and all those to the left will be inverted of the original bits, that is, after inversion we have xxxxxxx011111.... When we add 1, we will get xxxxxxx100000...., where the ’x’ are the inverted bits of original number.

Two’s Complement Signed Integers Representation:

Sign bit: If the most significant bit is 0, the number is non-negative, if it is 1 the number is negative. The most significant bit is called sign bit.

Range of the total set of 2^n words, one half correspond to negative number representations. With *n* bits, the interval of integers is between -2^{n-1} and $2^{n-1}-1$.

Backward compatibility: Addition and subtraction follow the same rules as in the unsigned case.



Decimal	$\rightarrow$	Nibble	Decimal	$\rightarrow$	Nibble
+1	$\rightarrow$	0001	−1	$\rightarrow$	1111
+2	$\rightarrow$	0010	−2	$\rightarrow$	1110
+3	$\rightarrow$	0011	−3	$\rightarrow$	1101
+4	$\rightarrow$	0100	−4	$\rightarrow$	1100
+5	$\rightarrow$	0101	−5	$\rightarrow$	1011
+6	$\rightarrow$	0110	−6	$\rightarrow$	1010
+7	$\rightarrow$	0111	−7	$\rightarrow$	1001

On the other hand, 1000 represents $-2^3 = -8$. Notice that 0000 and 1000 are, respectively, two’s complements of themselves.

Two's Complement Signed Integers Representation:

Ex: What is the decimal representation of signed 10110111?

Let's call **10110111=x**

$$
\begin{array}{rl}
 & 11111111 \\
- & 10110111 = x \\
\hline
 & 01001000 \\
+ & 1 \\
\hline
 & \underbrace{01001001}_{64+8+1=73} = -x \longrightarrow 73 = -x \longrightarrow \mathbf{x=-73}
\end{array}
$$

1's complement

2' s complement

Two’s Complement Signed Integers Representation:

Ex: Find the signed decimal integer represented by the 12-bit word FD7h.

FD7h=111111010111

$$\left.\begin{array}{r}\left.\begin{array}{r}111111111111\\ -\;111111010111\\ \hline 000000101000\end{array}\right\}\text{1's complement}\\ +\;1\\ \hline\end{array}\right\}\text{2's complement}$$

$$\underbrace{000000101001}_{32+8+1=41}=-x \longrightarrow 41=-x \longrightarrow \mathbf{x=-41}$$

Two’s Complement Signed Integers Representation:

Ex: Find the two’s complement representation for −104 with 16 bits and express it in hex notation.

104= 64+32+8= 0000000001101000

$$\left.\begin{array}{rl}\left.\begin{array}{rl} & 1111111111111111 \\ - & 0000000001101000\textbf{(104)} \\ \hline & 1111111110010111 \end{array}\right\}\text{1's complement} & \\ + & 1 \\ \hline \end{array}\right\}\text{2' s complement}$$

$$\underbrace{1111}_{F}\underbrace{1111}_{F}\underbrace{1001}_{9}\underbrace{1000}_{8}\textbf{(-104)} \longrightarrow \textbf{-104=FF98h}$$

Two’s Complement Signed Integers Representation:

Arithmetic Operations with Signed Numbers and Overflow

When adding and subtracting signed number representations, a carry or borrow may appear. This one is discarded if we are to keep the fixed length representation. Overflow will occur, however, if in this fixed length the result is nonsense. In particular;

- Addition of two numbers of the same sign should yield a result with the same sign;
- a positive number minus a negative number should yield a positive result and
- a negative number minus a positive number should yield a negative result.

When the results do not comply with any of these conditions, **there is an overflow**.

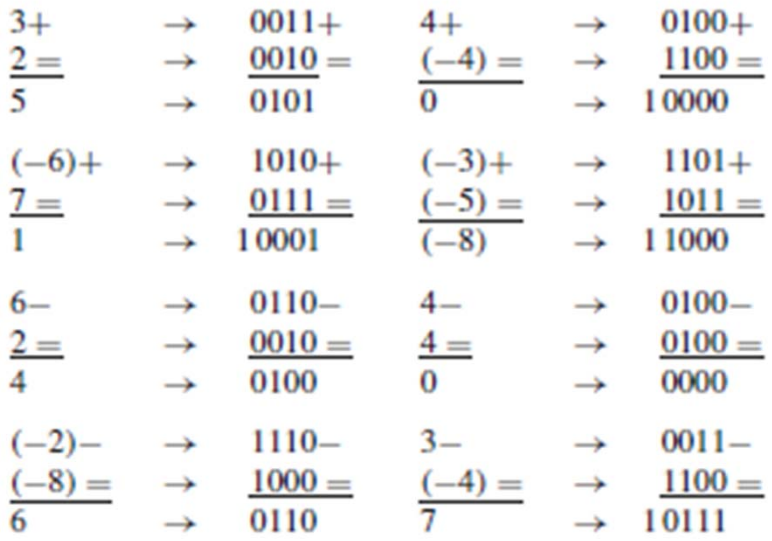
More explicitly:

Overflow occurs if (a) addition of two numbers of the same sign yields a number of opposite sign or (b) subtraction involving different signed numbers yields a difference with the sign of the subtrahend.

Two’s Complement Signed Integers Representation:

Ex: (a) Check the validity of the following operations for signed numbers using two’s complement convention with four bits: 3 + 2, 4 + (−4), (−6) + 7, (−3)+(−5), 6−2, 4−4, (−2)−(−8), 3−(−4).

All the following operations yield valid results discarding any carry or borrow when present. For example, 3 − (−4) in binary yields 10111, but only 0111 is considered yielding +7, as expected. Notice that (+4) + (−4) and 4−4 both yield 0 when only four bits are taken, but the former yields a carry.



3+	→	0011+	4+	→	0100+
2 =	→	0010 =	(−4) =	→	1100 =
5	→	0101	0	→	10000
(−6)+	→	1010+	(−3)+	→	1101+
7 =	→	0111 =	(−5) =	→	1011 =
1	→	10001	(−8)	→	11000
6−	→	0110−	4−	→	0100−
2 =	→	0010 =	4 =	→	0100 =
4	→	0100	0	→	0000
(−2)−	→	1110−	3−	→	0011−
(−8) =	→	1000 =	(−4) =	→	1100 =
6	→	0110	7	→	10111

Two’s Complement Signed Integers Representation:

Ex: (b) Verify overflow in the following cases: 3 + 5, (−5) + (−8), 4 − (−6), (−6) − (+3).

Overflow now occurs when the result of operation falls outside the range covered by the set of strings and is mainly shown by a result with a sign bit different from what was expected. Since 4-bit words cover from $-2^3 = -8$ to $2^3 - 1 = 7$, the operations 3 + 5 = 8, (−5) + (−8) = (−13) and 4 − (−6) = 10 do not make sense in this set. Let us look at the results in binary form interpreted from the standpoint of the two’s complement convention:

$$\begin{array}{lcr} 0011+ & \rightarrow & +3 \\ \underline{0101 =} & \rightarrow & +5 \\ 1000 & \rightarrow & -8 \end{array} \qquad \begin{array}{lcr} 1011+ & \rightarrow & -5 \\ \underline{1000 =} & \rightarrow & -8 \\ 10011 & \rightarrow & +3 \end{array}$$

$$\begin{array}{rl} & 0100\ (4) \\ - & \underline{1010\ (-6)} \\ & 11010\ (-6) \end{array} \qquad \begin{array}{rl} & 1010\ (-6) \\ - & \underline{0011\ (3)} \\ & 0111\ (7) \end{array}$$

We see in the first case an addition of two numbers with leading bit 0 (non negative) yielding a number with a sign bit 1. In the second case, two negative numbers add up to a positive result. In the third case, we subtract a negative number from a positive one, but the result is negative instead of positive. All these cases are deduced after analyzing the sign bits of the operands and results.