ECE 27000 S22 P 2

Chapter 2. Number Systems and Codes

A digital circuit processes binary digits (bits, or 0s and 1s).
Practical problems are rarely specified in terms of bits.
To design a digital circuit it is necessary to estb-lish a correspondence between binary digits and practical quantities.

2.1. Positional Number Systems

A number is represented as a string of digits.
Each digit position is associated with a weight.
The value of a number is the weighted sum of its digits.

Example:

$1734 = 1 \cdot 1000 + 7 \cdot 100 + 3 \cdot 10 + 4 \cdot 1 =$
$= 1 \cdot 10^3 + 7 \cdot 10^2 + 3 \cdot 10^1 + 4 \cdot 10^0$

The base or radix is 10 in this case.

In general, for radix r,

$d_{p-1} d_{p-2} \cdots d_1 d_0 =$
$= d_{p-1} \cdot r^{p-1} + d_{p-2} \cdot r^{p-2} + \cdots + d_1 \cdot r^1 + d_0 \cdot r^0$

$0 \leq d_i \leq r - 1$ for $0 \leq i \leq p - 1$.

The leftmost digit is called the most significant digit.
The rightmost digit is called the least significant digit.

With binary digits (bits), a number is given in base 2 or binary radix.

In general, for binary radix,
$b_{p-1} b_{p-2} \cdots b_1 b_0 =$
$= b_{p-1} \cdot 2^{p-1} + b_{p-2} \cdot 2^{p-2} + \cdots + b_1 \cdot 2^1 + b_0 \cdot 2^0$

We will indicate the base using the notation
$(b_{p-1} b_{p-2} \cdots b_1 b_0)_2$.

Example:
$10011_2 = 1 \cdot 2^4 + 1 \cdot 2^1 + 1 \cdot 1^0 =$
$= 16 + 2 + 1 = 19_{10}$

2.2. Binary, Octal and Hexadecimal Numbers

base 2 binary	base 8 octal	base 16 hexadecimal
0	0	0
1	1	1
10	2	2
11	3	3
100	4	4
101	5	5
110	6	6
111	7	7
1000	10	8
1001	11	9
1010	12	A
1011	13	B
1100	14	C
1101	15	D
1110	16	E
1111	17	F

$11101110_2 =$

$$\begin{array}{cccccccc}
2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\
1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \\
\hline
 & 8^2 & & & 8^1 & & & 8^0 \\
1 & 1 & 1 & 0 & 1 & 1 & 1 & 0
\end{array}$$

$= 3 \cdot 64 + 5 \cdot 8 + 6 \cdot 1 = 356_8$

To convert from binary to octal, partition the number into groups of three bits starting from the least significant bit (LBS).
Convert every three bits into an octal digit.

Converting to hexadecimal is similar except that the partition is into groups of four bits.

To convert from octal to binary, replace every octal digit with three bits.
To convert from hexadecimal to binary, replace every hexadecimal digit with four bits.

Examples:
$1316_8 = 001\ 011\ 001\ 110_2 = 1011001110_2$.
$2CE_{16} = 0010\ 1100\ 1110_{16} = 1011001110_2$.

A byte (eight bits) is a basic quantity in many computer systems.
Hexadecimal is a convenient way to represent bytes - every two hexadecimal digits represent one byte.
For example, $AB34_{16}$ represents two bytes with values AB_{16} and 34_{16}.

2.3. Binary-Decimal Conversions

Conversion of a decimal number D to base r:
In base r we will obtain the number
$R = d_{p-1} d_{p-2} \cdots d_0$
with decimal value
$D = d_{p-1} \cdot r^{p-1} + d_{p-2} \cdot r^{p-2} +$
$+ \cdots + d_2 \cdot r^2 + d_1 \cdot r + d_0.$

We can obtain d_0 if we divide D by r.
The remainder is d_0.
The quotient is

$$d_{p-1} \cdot r^{p-2} + d_{p-2} \cdot r^{p-3} + \cdots + d_2 \cdot r + d_1.$$

$$d_{p-1} \cdot r^{p-2} + d_{p-2} \cdot r^{p-3} + \cdots + d_2 \cdot r + d_1.$$

We can obtain d_1 if we divide the quotient by r.
The remainder is d_1.
The new quotient is

$$d_{p-1} \cdot r^{p-3} + d_{p-2} \cdot r^{p-4} + \cdots + d_3 \cdot r + d_2.$$

This can continue until the quotient is zero.

Example: 57_{10} to base 4.
57/4=14(1)
14/4=3(2)
3/4=0(3)
$57_{10} = 321_4$.

Example: 57_{10} to base 2.
57/2=28(1)
28/2=14(0)
14/2=7(0)
7/2=3(1)
3/2=1(1)
1/2=0(1)
$57_{10} = 111001_2$.

It is now possible to check that
$57_{10} = 321_4 = 111001_2$.
11 10 01
3 2 1

Addition and Subtraction

Example: Binary addition.

```
      1  0  1  1  1  1  0  0   +
      1  1  1  0  1  0  1  0
   ----------------------------
   1  1  0  1  0  0  1  1  0
```

Example: Binary subtraction.

```
   1  1  0  1  0  0  1  1  0   -
      1  0  1  1  1  1  0  0
   ----------------------------
      1  1  1  0  1  0  1  0
```

2.5. Representations of Negative Numbers

In the $signed-magnituted$ system, a number consists of a sign (+/-) and a magnitude.
For binary numbers, the sign is given by the left-most bit.
When the leftmost bit is 0 the number is positive.
When the leftmost bit is 1 the number is negative.

Example:
$01001_2 = 9_{10}$
$11001_2 = -9_{10}$

An n-bit signed-magnitude integer can be in the range between $11 \cdots 1$ and $01 \cdots 1$.
$11 \cdots 1$ is $-(2^{n-1} - 1)$.
$01 \cdots 1$ is $+(2^{n-1} - 1)$.
There is an equal number of positive and negative numbers.

Adding two numbers requires a comparison of their signs, and possibly magnitudes.
If both are positive, add the magnitudes, and make sure the sign bit is 0.
If both are negative, add the magnitudes, and make sure the sign bit is 1.
If one is positive and one is negative, compare the magnitudes to find the large number; subtract the smaller from the larger; use the sign bit of the larger.

This results in complex digital adders.

In a *complement number* system, numbers can be added or subtracted directly without first checking their signs and comparing their magnitudes.

In these systems a number is negated by applying an algorithm that complements it.
Complementation is more complex than changing one sign bit but addition is simpler.

In a $two's - complement$ system, the complement of an n-bit number is obtained by subtracting it from 2^n.

Let $B = b_{n-1}b_{n-2} \cdots b_1 b_0$
To compute $-B$ we need to compute $2^n - B$.
This is simplified by noting that
$2^n - B = (2^n - 1) - B + 1$.
The number $2^n - 1$ is represented by n 1s.
Therefore, the subtraction complements the bits of B.
It is then necessary to add 1.

Example: $n = 8$
To complement 01110100, we compute

$$\begin{array}{ll} 11111111 & - \\ \underline{01110100} & \\ 10001011 & \end{array}$$

and then

$$\begin{array}{ll} 10001011 & + \\ \underline{00000001} & \\ 10001100 & \end{array}$$

Complementing again:

```
11111111     -
10001100
__________
01110011
```

and then

```
01110011     +
00000001
__________
01110100
```

Example: $n = 4$
Complement 0000.

$$\begin{array}{lll} & 1111 & - \\ & 0000 & \\ \hline & 1111 & + \\ & 0001 & \\ \hline 1 & 0000 & \end{array}$$

The carry out is ignored.
There is only one representation of 0.

With $n = 4$, the largest positive number is $7_{10} = 0111$.

Two's complement:

$$\begin{array}{ll} 1111 & - \\ 0111 & \\ \hline 1000 & + \\ 0001 & \\ \hline 1001 & \end{array}$$

The smallest negative number is $-8_{10} = 1000$.
Two's complement:

$$\begin{array}{lr}
1111 & - \\
1000 & \\
\hline
0111 & + \\
0001 & \\
\hline
1000 &
\end{array}$$

There is no $+8_{10}$ with $n = 4$.

In general with n bits, the range of possible numbers is -2^{n-1} to $2^{n-1} - 1$.

All the numbers with $n = 4$:

number		complement	
0	0000	0000	0
1	0001	1111	-1 = -8+7
2	0010	1110	-2 = -8+6
3	0011	1101	-3 = -8+5
4	0100	1100	-4 = -8+4
5	0101	1011	-5 = -8+3
6	0110	1010	-6 = -8+2
7	0111	1001	-7 = -8+1
-	-	1000	-8 = -8+0

The most significant bit (MSB) of a two's-complement number acts as a sign bit.
It is 0 for positive numbers and 1 for negative numbers.

The magnitude of a number can be computed as for an unsigned number, except that the weight of the MSB is -2^{n-1} instead of 2^{n-1}.

Converting an n-bit two's-complement number X into an m-bit one:

If $m > n$, add $m - n$ copies of the sign bit of X to the left of X.
This is called *sign extension*.
Examples:
$X = 0010 = 00000010 = 2_{10}$
$X = 1110 = 11111110 = -2_{10}$

Showing that $1d_2d_1d_0 = 11d_2d_1d_0$:

Approach 1: compute the magnitude by computing the two's complement.
$1d_2d_1d_0 \rightarrow 0d_2'd_1'd_0' + 1$.
$11d_2d_1d_0 \rightarrow 00d_2'd_1'd_0' + 1$.
If there is no overflow the numbers are the same.

Approach 2: compute the decimal value.
$1d_2d_1d_0 = -8 + d_2 \cdot 4 + d_1 \cdot 2 + d_0$.
$11d_2d_1d_0 = -16 + 8 + d_2 \cdot 4 + d_1 \cdot 2 + d_0$.
The two numbers are equal.

If $m < n$, remove $n - m$ leftmost bits of X.
The result is valid only if all the discarded bits are the same as the sign bit of the result.
Otherwise the number will be changed.

Examples:
$X = 00000010 = 0010 = 2_{10}$
$X = 11111110 = 1110 = -2_{10}$

Examples where the number will change:

$X = 11110010 \rightarrow 0010$

$X = 00001010 \rightarrow 1010$

$X = 01000010 \rightarrow 0010$

These numbers are too large (or too small) to represent in four bits using two's-complement.

Other representations of negative numbers:

Diminished radix-complement and one's-complement.

Excess representations.

Two’s Complement Addition and Subtraction

Based on the previous table, a negative number is equal to the magnitude of the $n-1$ rightmost bits minus 2^{n-1}.
Therefore, it is possible to add positive and negative numbers without distniguishing between them.
If there is no carry into the MSB, the result is positive.
If there is a carry into the MSB, the result is negative.
Carry beyond the MSB is ignored.

Examples:

```
+3        0011   +
+4        0100
-----------------
+7        0111

-2         1110   +
-6         1010
------------------
-8     1   1000

+6          0110   +
 -3         1101
-------------------
  3     1   0011

+4         0100   +
 -7        1001
------------------
 -3        1101
```

Overflow occurs when a number exceeds the range of representable numbers.
Overflow can occur only when adding numbers with the same sign.
Examples of overflow:

```
+4          0100     +
+7          0111
--------------------
 -5          1011

 -4          1100     +
 -7          1001
--------------------
+5     1    0101
```

Overflow is identified when the sign of the result is different from the sign of the two numbers added.

Equivalently, when the carry in and out of the MSB is different.

Subtraction

Subtraction of two's-complement numbers is typically implemented as follows:
$X - Y = X + (-Y)$
Substitute the two's-complement of Y for $-Y$.
The two's-complement of Y is obtained by complementing each bit of Y to obtain Y', and adding 1.
$X - Y = X + (-Y) = X + Y' + 1$
Addition of 1 is implemented by using a carry of 1 into the least significant bit position.

Examples:

$$\begin{array}{llll}
6 & & 0110 & - \\
2 & & 0010 & \\
\hline
 & & 0001 & \\
6 & & 0110 & + \\
2' & & 1101 & \\
\hline
 & 1 & 0100 & \\
\hline
4 & & 0100 &
\end{array}$$

$$\begin{array}{lll}
2 & 0010 & - \\
6 & 0110 & \\
\hline
 & 0001 & \\
2 & 0010 & + \\
6' & 1001 & \\
\hline
-4 & 1100 &
\end{array}$$

Overflow can be detected similar to addition.

Example:

```
-6          1010     -
 3          0011
----------------------
            0001
-6          1010     +
3’          1100
----------------------
        1   0111
```

2.10. Binary Codes for Decimal Numbers

A set of n-bit strings in which different bit strings represent different numbers (or different elements of a set) is called a *code*.

A combination of n bits is called a *code word*.

To represent the ten decimal digits $0 \cdots 9$ requires code words with at least four bits.
There are many options for selecting these code words.

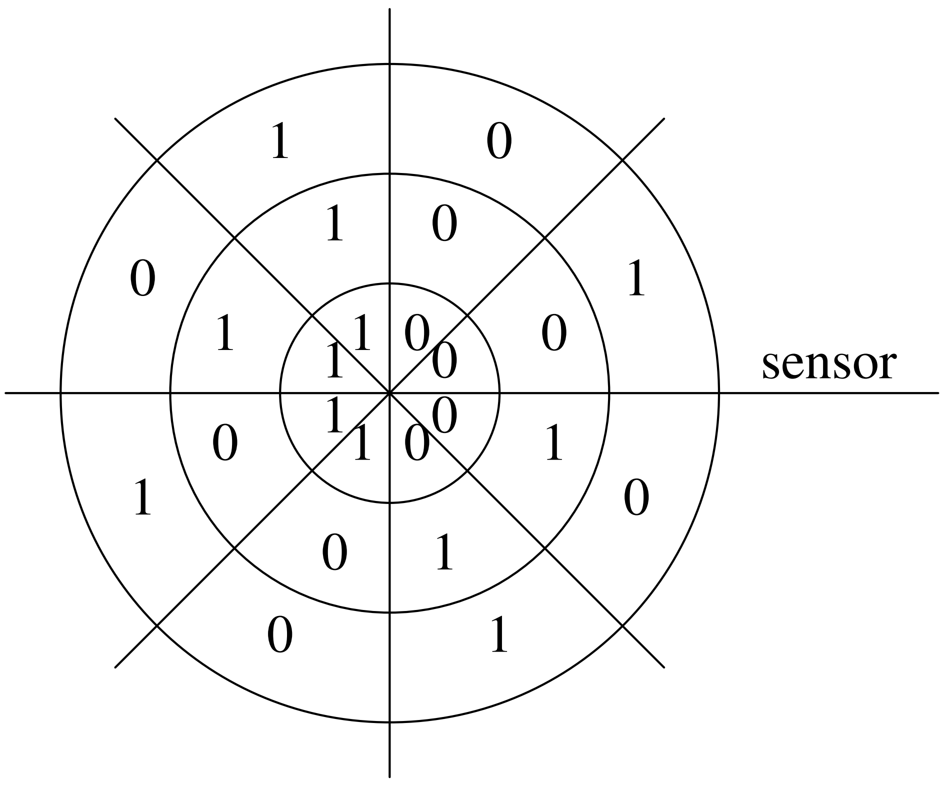
Some common codes for decimal digits:

	(8421)			
digit	BCD	2421	excess-3	1-out-of-10
0	0000	0000	0011	1000000000
1	0001	0001	0100	0100000000
2	0010	0010	0101	0010000000
3	0011	0011	0110	0001000000
4	0100	0100	0111	0000100000
5	0101	1011	1000	0000010000
6	0110	1100	1001	0000001000
7	0111	1101	1010	0000000100
8	1000	1110	1011	0000000010
9	1001	1111	1100	0000000001
	1010			
	1011			
	1100			
	1101			
	1110			
	1111			

A 4-bit code has $2^4 - 10 = 6$ unused code words. Below the line are unused code words for BCD. The 1-out-of-10 code has $2^{10} - 10 = 1014$ unused code words.

Gray Code

Consider a disk partitioned into eight regions, each one encoded using three bits.
A sensor indicates which area of the disk is above it.



If the region above the sensor is 010, the sensor provides an unambigous indication of 010.

If the boundary between 000 and 001 is above the sensor, the sensor provides an indication that the area above it is either 000 or 001. Both are acceptable.

If the boundary between 001 and 010 is above the sensor, the sensor may produce 001 or 010, which are acceptable.
However, it may also produce 000 or 011, which is unacceptable.

For the boundary between 000 and 111 there are two acceptable indications, 000 or 111, and six unacceptable indications, 001, 010, 011, 100, 101 and 110.

Solution: A code where only one bit changes between consecutive code words.

3-bit Gray code:

0	0	0
0	0	1
0	1	1
0	1	0
1	1	0
1	1	1
1	0	1
1	0	0

Constructing the n-bit Gray code:

The 1-bit Gray code has code words 0 and 1.

The first 2^n code words of an $(n + 1)$-bit Gray code are equal to the code words of the n-bit Gray code, with a leading 0 added to every code word.

The last 2^n code words of an $(n + 1)$-bit Gray code are equal to the code words of the n-bit Gray code written in reverse order, with a leading 1 added to every code word.

$n = 1$	$n = 2$	$n = 3$	$n = 4$
0	00	000	0000
1	01	001	0001
	11	011	0011
	10	010	0010
		110	0110
		111	0111
		101	0101
		100	0100
			1100
			1101
			1111
			1110
			1010
			1011
			1001
			1000

2.12. Character Codes

Codes are also used for representing text.
In ASCII code, each character is represented by a 7-bit code word.
Some of the ASCII code words:

code word	character	code word	character
0000 000	NULl	0001 010	!
0000 010	SPace	0010 010	"
0111 000	BELl		
0000 011	0	0001 100	A
0001 011	1	0010 100	B
0010 011	2	0011 100	C
0011 011	3		
0100 011	4		
0000 101	P	0001 110	a
0001 101	Q	0010 110	b
0010 101	R	0011 110	c
0000 111	p		
0001 111	q		
0010 111	r		

2.13 Codes for Actions, Conditions, and States

Example: Traffic light controller for an intersection of North-South (NS) and East-West (EW) streets.

State	NS	WE	code word
NS go	Green	Red	000
NS wait	Yellow	Red	001
NS delay	Red	Red	010
WE go	Red	Green	100
WE wait	Red	Yellow	101
WE delay	Red	Red	110

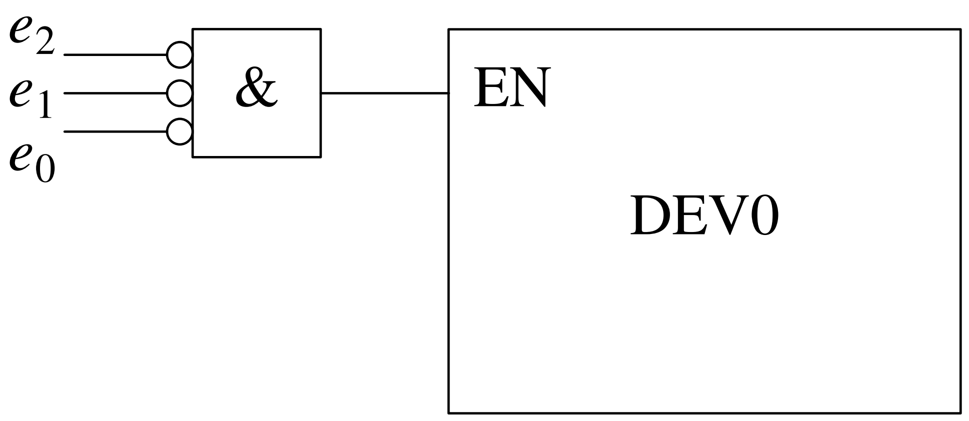
There are many options for selecting the code. They affect the cost and performance of the digital circuit.

Example: A system with n devices and a controller that selects one device at a time to operate.

A code with $\lceil \log_2(n) \rceil$ bits is sufficient.

For $n = 5$, $\lceil \log_2(n) \rceil = 3$, and the devices are encoded using three bits.

	code word	
device	$e_2 e_1 e_0$	device enable
0	000	$e_2' e_1' e_0'$
1	001	$e_2' e_1' e_0$
2	010	$e_2' e_1 e_0'$
3	011	$e_2' e_1 e_0$
4	100	$e_2 e_1' e_0'$



Using a 1-out-of-5 code:

device	code word $e_4e_3e_2e_1e_0$	device enable
0	00001	e_0
1	00010	e_1
2	00100	e_2
3	01000	e_3
4	10000	e_4
none	00000	-