# Review of Number Systems

The study of number systems is important from the viewpoint of understanding how data are represented before they can be processed by any digital system including a computer. Different characteristics that define a number system include the number of independent digits used in the number system, the place values of the different digits constituting the number and the maximum numbers that can be written with the given number of digits. The base or radix of a number system is defined as the maximum number of digits or symbols that can be used in any position. The radix of the decimal number system is 10 as it has 10 independent digits, i.e. 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9. Similarly, the binary number system with only two independent digits, 0 and 1, is a radix-2 number system. The octal and hexadecimal number systems have a radix (or base) of 8 and 16 respectively.

**Decimal Number System:**

The decimal number system is mainly suitable for human beings. As it uses 10 digits in any position of a given number, the base or radix is 10.

Example: $(152)_{10}, (82.45)_{10}, (98.79)_{10}$

It obeys positional notation rule. The value or magnitude of a given decimal number can be expressed as the sum of the various digits multiplied by their place values or weights.

Example: $(1245)_{10} = 1 \times 10^3 + 2 \times 10^2 + 4 \times 10^1 + 5 \times 10^0$

**Binary Number System:**

Most of the electronic components will have two state or binary operation ON and OFF. The decimal number system is not suitable to digital systems which contain two state operation. The number system used for digital systems is binary number system. The binary number system is a radix-2 number system with ‘0’ and ‘1’ as the two independent digits. All larger binary numbers are represented in terms of '0' and '1'. These symbols are known as binary digits or simply bits. It also obeys positional notation rule.

**Example**: $(101001)_2, (11.01101)_2$

$$(1001.11)_2 = 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2}$$

**Octal Number System:**

The octal number system has a radix of 8 and therefore has eight distinct digits. The independent digits are 0, 1, 2, 3, 4, 5, 6 and 7. It also obeys positional notation rule.

**Example**: $(754.16)_8, (557.63)_8$

$$(714.77)_8 = 7 \times 8^2 + 1 \times 8^1 + 4 \times 8^0 + 7 \times 8^{-1} + 7 \times 8^{-2}$$

**Hexadecimal Number System:**

To represent large numbers in digital systems like computers, hexadecimal number system is used. Here the base is 16 and hence 16 digits or symbols (0-9) and (A-F) can be placed in any given number. This number system obeys positional notation rule.

**Example**: $(1AC.BD)_{16}, (2BC)_H$

$$(3CD.4E)_{16} = 3 \times 16^2 + C \times 16^1 + D \times 16^0 + 4 \times 16^{-1} + E \times 16^{-2}$$

| Number | Decimal Number System | Binary Number System | Octal Number System | Hexadecimal Number System |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |
| 2 | 2 | 10 | 2 | 2 |
| 3 | 3 | 11 | 3 | 3 |
| 4 | 4 | 100 | 4 | 4 |
| 5 | 5 | 101 | 5 | 5 |
| 6 | 6 | 110 | 6 | 6 |

| | | | | |
|---|---|---|---|---|
| 7 | 7 | 111 | 7 | 7 |
| 8 | 8 | 1000 | 10 | 8 |
| 9 | 9 | 1001 | 11 | 9 |
| 10 | 10 | 1010 | 12 | A |
| 11 | 11 | 1011 | 13 | B |
| 12 | 12 | 1100 | 14 | C |
| 13 | 13 | 1101 | 15 | D |
| 14 | 14 | 1110 | 16 | E |
| 15 | 15 | 1111 | 17 | F |
| 16 | 16 | 10000 | 20 | 10 |

**Conversion of numbers:**

**Decimal to Binary Conversion:**

A decimal number can be converted into a binary number by using Divide-by-2 or double dabble method. For the integer part, the binary equivalent can be found by successively dividing the integer part of the number by 2 and recording the remainders until the quotient becomes '0'. The remainders written in reverse order constitute the binary equivalent. For the fractional part, it is found by successively multiplying the fractional part of the decimal number by 2 and recording the carry until the result of multiplication is '0'. The carry sequence written in forward order constitutes the binary equivalent of the fractional part of the decimal number.

Example: Convert $(152.49)_{10}$ to binary number.

$$152 \div 2 = 76 \; with \; remainder \; 0$$

$$76 \div 2 = 38 \; with \; remainder \; 0$$

$$38 \div 2 = 19 \; with \; remainder \; 0$$

$$19 \div 2 = 9 \; with \; remainder \; 1$$

$$9 \div 2 = 4 \; with \; remainder \; 1$$

$$4 \div 2 = 2 \; with \; remainder \; 0$$

$$2 \div 2 = 1 \; with \; remainder \; 0$$

$$1 \div 2 = 0 \; with \; remainder \; 1$$

The binary representation of integer part is $(10011000)_2$

$$0.49 \times 2 = 0.98 \; with \; carry \; 0$$

$$0.98 \times 2 = 0.96 \; with \; carry \; 1$$

$$0.96 \times 2 = 0.92 \; with \; carry \; 1$$

The binary representation of fractional part is $(.011)_2$

$$\therefore (152.49)_{10} = (10011000.011)_2$$

**Decimal to Octal Conversion:**

The process of decimal-to-octal conversion is similar to that of decimal-to-binary conversion. The progressive division in the case of the integer part and the progressive multiplication while working on the fractional part here are by '8' which is the radix of the octal number system. Again, the integer and fractional parts of the decimal number are treated separately. This process of conversion is known as octal-dabble method.

**Example**: Convert $(152.49)_{10}$ to octal number.

$$152 \div 8 = 19 \; with \; remainder \; 0$$

$$19 \div 8 = 2 \; with \; remainder \; 3$$

$$2 \div 8 = 0 \; with \; remainder \; 2$$

The octal representation of integer part is $(230)_8$

$$0.49 \times 8 = 0.92 \; with \; carry \; 3$$

$$0.92 \times 8 = 0.36 \; with \; carry \; 7$$

$$0.36 \times 8 = 0.88 \; with \; carry \; 2$$

The octal representation of fractional part is $(.372)_8$

$$\therefore (152.49)_{10} = (230.372)_8$$

**Decimal-to-Hexadecimal Conversion:**

The process of decimal-to-hexadecimal conversion is known as hex-dabble method. Since the hexadecimal number system has a base of 16, the progressive division and multiplication factor in this case is 16.

**Example**: Convert $(1542.89)_{10}$ to hexadecimal number.

$$1542 \div 16 = 96 \; with \; remainder \; 6$$

$$96 \div 16 = 6 \; with \; remainder \; 0$$

$$6 \div 16 = 0 \; with \; remainder \; 6$$

The hexadecimal representation of integer part is $(606)_{16}$

$$0.89 \times 16 = 0.24 \; with \; carry \; E$$

$$0.24 \times 16 = 0.84 \; with \; carry \; 3$$

$$0.84 \times 16 = 0.44 \; with \; carry \; D$$

The hexadecimal representation of fractional part is $(.E3D)_{16}$

$$\therefore (152.49)_{10} = (606.E3D)_{16}$$

**Binary to Decimal Conversion:**

A binary number is converted to a decimal number by expressing the number using positional notation rule.

**Example:** $(11.101)_2 = (X)_{10}$ , find $X$.

$$11.101 = 1 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}$$
$$= 2 + 1 + 0.5 + 0.125 = 3.625$$

$$\therefore X = 3.625$$

**Binary to Octal Conversion:**

The octal number system contains 8 digits which can be expressed as a third power of the base of binary system i.e., 2. Hence a binary number can be converted into an equivalent octal number by splitting the integer and fractional parts into groups of three bits, starting from the binary point on both sides. The 0s can be added to complete the outside groups if needed.

**Example:** Convert $(101001.001011)_2$ into octal number.

$$101001.001011 = 101 \quad 001 \quad . \quad 001 \quad 011 \quad = 51.13$$

$$\therefore \text{The octal equivalent is } (51.13)_8$$

**Binary to Hexadecimal Conversion:**

The hexadecimal number system contains 16 digits which can be expressed as a fourth power of the base of binary system i.e., 2. Hence a binary number can be converted into an equivalent octal number by splitting the integer and fractional parts into groups of four bits, starting from the binary point on both sides. The 0s can be added to complete the outside groups if needed.

**Example**: Convert $(101001.001011)_2$ into hexadecimal number.

$$101001.001011 = 0010 \quad 1001 \quad . \quad 0010 \quad 1100 \quad = 29.2C$$

$$\therefore \text{The hexadecimal equivalent is } (29.2C)_{16}$$

**Octal to Decimal Conversion:**

An octal number is converted to a decimal number by expressing the number using positional notation rule.

**Example**: $(754.65)_8 = (X)_{10}$ , find $X$.

$$(754.65)_8 = 7 \times 8^2 + 5 \times 8^1 + 4 \times 8^0 + 6 \times 8^{-1} + 5 \times 8^{-2} = 492.828$$

$$\therefore X = 492.828$$

**Octal to Binary Conversion:**

An octal number can be converted into its binary equivalent by replacing each octal digit with its three-bit binary equivalent.

**Example**: Find the binary equivalent for $(557.63)_8$ .

$$557.63 = 101101111.110011$$

**Octal to Hexadecimal Conversion:**

For octal to hexadecimal conversion, the octal number may first be converted into an equivalent binary number and then the binary number transformed into its hex equivalent.

Example: Convert $(557.63)_8$ into hexadecimal number.

$$557.63 = 101101111.110011 = 0001 \quad 0110 \quad 1111 \;.\; 1100 \quad 1100$$
$$= 16F.CC$$

$$\therefore \text{The hexadecimal equivalent is } (16F.CC)_{16}$$

**Hexadecimal to Decimal Conversion:**

A hexadecimal number is converted to a decimal number by expressing the number using positional notation rule.

Example: Find the decimal equivalent for $(3CD.4E)_{16}$.

$$(3CD.4E)_{16} = 3 \times 16^2 + C \times 16^1 + D \times 16^0 + 4 \times 16^{-1} + E \times 16^{-2}$$
$$= 973.304$$

$$\therefore \text{The decimal equivalent is } (973.304)_{10}$$

**Hexadecimal to Binary Conversion:**

A hexadecimal number can be converted into its binary equivalent by replacing each digit with its four-bit binary equivalent.

Example: Find the binary equivalent for $(3CD.4E)_{16}$.

$$(3CD.4E)_{16} = 001111001101.01001110$$

**Hexadecimal to Octal Conversion:**

For hexadecimal–octal conversion, the given hex number is firstly converted into its binary equivalent which is further converted into its octal equivalent.

Example: Find the octal equivalent for $(3CD.4E)_{16}$.

$$(3CD.4E)_{16} = 001111001101.01001110$$

$$= 001\ \ 111\ \ 001\ \ 101\ .\ \ 010\ \ 011\ \ 100 = 1715.234$$

$\therefore$ The octal equivalent is $(1715.234)_8$

**Representation of Negative numbers:**

**Signed magnitude Representation:**

In this representation, a sign bit is used to represent the sign of the number; if sign bit is 0, the number is positive and if the sign bit is 1, the number is negative. So in this form, the MSB represents the sign bit and the remaining bits represent the magnitude. For example, -7 and +5 is represented as 1111 and 0101 respectively. But it is not suitable for arithmetic operations as it requires frequent conversion from signed magnitude to normal form.

**Complements:**

The method of complements is the efficient method for representing negative numbers which eliminates the drawbacks of signed magnitude representation. There are two types of complements namely *r*'s and *(r-1)*'s complement where *r* is the radix of the number system.

In binary number system, we have 1's and 2's complements. The 1's complement of an unsigned binary number can be achieved by inverting each bit from 0 to 1 or 1 to 0.

Example: 1's complement of 1011011 is 0100100

2's complement is obtained just by adding 1 to 1's complement.

Example: 2's complement of 1011011 is 0100101

## BINARY CODES

Digital circuits use binary signals but are required to handle data which may be alphabetic, numeric, or special characters. Hence the signals available in some other form other than binary have to be converted into suitable binary form before they can be processed further by digital circuits. This means, in whatever format the information may be available it must be converted into binary format. To achieve this, a process of coding is required where each letter, special character, or numeral is coded in a unique combination of 0s and 1s using a coding scheme known as *code*. In digital systems a variety of codes are used to serve different purposes, such as data entry, arithmetic operation, error detection and correction, etc. Selection of a particular code depends on the requirement. Even in a single digital system a number of different codes may be used for different operations and it may even be necessary to convert data from one type of code to another.

Codes can be broadly classified into five groups:

(*i*) Weighted Binary Codes (*ii*) Non-weighted Codes (*iii*) Error-detection Codes

(*iv*) Error-correcting Codes and (*v*) Alphanumeric Codes.

**Weighted Codes:**

If each position of a number represents a specific weight then the coding scheme is called weighted binary code. In such coding the bits are multiplied by their corresponding individual weight, and then the sum of these weighted bits gives the equivalent decimal digit.

Examples: Straight Binary code, Binary Coded Decimal code (BCD)

**Straight Binary Code:**

This code is used to represent numbers using natural binary form.

Example: Straight binary code for 17 is 10001

**BCD Code:**

It is difficult or time taking process to represent large numbers using straight binary code. A BCD code is one in which the digits of a decimal number are encoded one at a time into groups of four binary digits or bits. Hence decimal digits (0-9) are represented using 4 bits each from 0000 to 1001. But using 4 bits 16 combinations are possible from 0000-1111. Hence in BCD code, the combinations (1010-1111) are invalid. It is also known as 8421 code.

Example: BCD code for 214 is 001000010100

The disadvantage of BCD code is it is not suitable to perform subtraction using 9's complement. This drawback can be avoided in Excess-3 code.

**Excess-3 Code:**

The excess-3 code is another important BCD code which is used to overcome the problem of 8421 BCD code. The Excess-3 code can be formed by adding decimal 3 to the BCD code.

| Decimal | 8421 BCD code | Excess-3 code |
|---|---|---|
| 0 | 0000 | 0011 |
| 1 | 0001 | 0100 |
| 2 | 0010 | 0101 |
| 3 | 0011 | 0110 |
| 4 | 0100 | 0111 |
| 5 | 0101 | 1000 |
| 6 | 0110 | 1001 |
| 7 | 0111 | 1010 |
| 8 | 1000 | 1011 |
| 9 | 1001 | 1100 |

In Excess-3 code, the combinations 0000, 0001, 0010, 1101, 1110 and 1111 are invalid or unused states. The complement of the excess-3 code of a given decimal number yields the excess-3 code for 9's complement of the decimal number and hence the excess-3 code can be used effectively for both addition and subtraction of decimal numbers, thus avoiding the drawback of 8421 code. It is a self-complementing code. A code is said to be self-complementing, if the code word of the 9's complement of $N$, can be obtained from the code word of $N$ by interchanging all the 0s and 1s. In self-complementing codes, the sum of the weights is equal to 9. The 2421, 5211, 642-3, 84-2-1 and Excess-3 codes are self-complementing codes.

**Gray Code:**

Gray Code is a unit distance code in which two successive values differ only by 1 bit. It is a non-weighted code. The Gray code is used in applications where the normal sequence or binary numbers may produce an error or ambiguity during the transition from one number to the next. It is also known as reflective code or unit distance code or cyclic code. These codes find great applications in Boolean function minimization using Karnaugh map.

| Decimal | Gray | Decimal | Gray |
|---|---|---|---|
| 0 | 0000 | 8 | 1100 |
| 1 | 0001 | 9 | 1101 |
| 2 | 0011 | 10 | 1111 |
| 3 | 0010 | 11 | 1110 |
| 4 | 0110 | 12 | 1010 |
| 5 | 0111 | 13 | 1011 |
| 6 | 0101 | 14 | 1001 |
| 7 | 0100 | 15 | 1000 |

**Error detecting Codes:**

Binary information may be transmitted through some form of communication medium such as wires or radio waves or fiber optic cables, etc. Any external noise introduced into a physical communication medium changes bit values from 0 to 1 or vice versa. An error detection code can be used to detect errors during transmission.

In general, to obtain an $n$-bit error-detecting code, no more than half of the possible $2^n$ combinations of digits can be used. The code words are chosen in such a manner that, in order to change one valid code word into another valid code word, at least two digits must be complemented. Thus, to obtain an error detecting code for the 10 decimal digits, at least 5 binary digits are needed. The distance between two code words is defined as the number of digits that must change in one word so that the other word results. For example, the distance between 1010 and 0100 is three, since the two code words differ in three bit positions. The minimum distance of a code is the smallest number of bits in which any two code words differ. Thus the minimum distance of the BCD or Excess-3 codes is one and a BCD with parity bit is two. Therefore, a code is an error-detecting code if and only if its minimum distance is two or more.

**Parity:** A parity bit is an extra bit included with a message to make the total number of 1s either odd or even. In an odd parity, P is chosen so that the sum of all 1s is odd (including the parity bit). In an even parity, P is chosen so that the sum of all 1s is even (including the parity bit). In the sending end, the message (in this case the first four bits) is applied to a "parity generation" circuit where the required P bit is generated. The message, including the parity bit, is transferred to its destination. In the receiving end, all the incoming bits (in this case five) are applied to a "parity check" circuit to check the proper parity adopted. An error is detected if the checked parity does not correspond to the adopted one. The parity method detects the presence of one, three, five, or any odd combination of errors. An even combination of errors is undetectable since an even number of errors will not change the parity of the bits. This is an error detecting code and it cannot correct the errors.

| Message | P (odd) | P (even) |
|---|---|---|
| 0000 | 1 | 0 |
| 0001 | 0 | 1 |
| 0010 | 0 | 1 |
| 0011 | 1 | 0 |
| 0100 | 0 | 1 |
| 0101 | 1 | 0 |
| 0110 | 1 | 0 |
| 0111 | 0 | 1 |
| 1000 | 0 | 1 |
| 1001 | 1 | 0 |
| 1010 | 1 | 0 |
| 1011 | 0 | 1 |
| 1100 | 1 | 0 |
| 1101 | 0 | 1 |
| 1110 | 0 | 1 |
| 1111 | 1 | 0 |

**2-out-of-5 code:**

The 2-out-of-5 code consists of all 10 possible combinations of two 1's in a five-bit code. With the exception of the code word for decimal 0, the 2-out-of-5 code is a weighted code and can be derived from the (1, 2, 4, 7) code. If a single error occurs, it transforms the valid code word into an invalid one, thus making the detection of the error straightforward.

| Decimal digit | 2-out-of-5 code 0 | 1 | 2 | 4 | 7 |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 |
| 2 | 1 | 0 | 1 | 0 | 0 |
| 3 | 0 | 1 | 1 | 0 | 0 |
| 4 | 1 | 0 | 0 | 1 | 0 |
| 5 | 0 | 1 | 0 | 1 | 0 |
| 6 | 0 | 0 | 1 | 1 | 0 |
| 7 | 1 | 0 | 0 | 0 | 1 |
| 8 | 0 | 1 | 0 | 0 | 1 |
| 9 | 0 | 0 | 1 | 0 | 1 |

**Error correcting codes:**

For a code to be error-correcting, its minimum distance must be further increased. In general, a code is said to be an error-correcting code if the correct code can always be deduced from the erroneous word. The important properties are

(*i*) For detection of a single error $d_{min}$ should be at least two.

(*ii*) For single error correction, $d_{min}$ should be at least three, since the number of errors, $E \leq [(d_{min} - 1)/2]$.

(*iii*) Greater values of $d_{min}$ will provide detection and/or correction of more number of errors.

**Hamming code:**

Hamming code is a single error correcting code and the basic principles in constructing the code are as follows. To each group of $m$ information, or message, digits, $k$ parity checking digits, denoted $p_1, p_2, \dots, p_k$, are added to form an $(m + k)$-digit code. The location of each of the $m + k$ digits within a code word is assigned a decimal value, starting by assigning a 1 to the most significant digit and $m + k$ to the least significant digit. $k$ parity checks are performed on selected digits of each code word. The result of each parity check is recorded as 1 or 0, depending, respectively, on whether an error has or has not been detected. These parity checks make possible the development of a binary number, $c_1 c_2 \dots c_k$, whose value when an error occurs is equal to the decimal value assigned to the location of the erroneous digit, and is equal to zero if no error occurs.

The number $k$ of digits in the position number must be large enough to describe the location of any of the $m + k$ possible single errors, and must in addition take on the value zero to describe the 'no error' condition.

Consequently, $k$ must satisfy the inequality $2^k \geq m + k + 1$. Therefore if $m = 4$, then $k = 3$ and at least three parity checking digits must be added to the BCD code.

Thus the error correcting code consists of seven digits. If the position number is equal to 101, it means that an error has occurred in position 5.If the position number is equal to 000, the message is correct.

The Hamming code with $m = 4\ and\ k = 3$ can be constructed as follows:

$p_1$ is selected so as to establish even parity in positions 1, 3, 5, 7.

$p_2$ is selected so as to establish even parity in positions 2, 3, 6, 7.

$p_3$ is selected so as to establish even parity in positions 4, 5, 6, 7.

The code can be constructed by adding appropriate checking digits to the message digits.

Example: Let the message be 0100.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Position : | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| | $p_1$ | $p_2$ | $m_1$ | $p_3$ | $m_2$ | $m_3$ | $m_4$ |
| | | | 0 | | 1 | 0 | 0 |

$p_1 = 1; p_2 = 0; p_3 = 1$

Therefore, the coded message is 1001100. The error location and correction is performed in the following manner. If 1101001 is transmitted, but due to an error in the fifth position, the sequence 1101101 is received. The location of the error can be determined by performing three parity checks as follows:

| Message received: | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| | 1 | 1 | 0 | 1 | 1 | 0 | 1 |

4-5-6-7 parity check: $c_1 = 1$

2-3-6-7 parity check: $c_2 = 0$

1-3-5-7 parity check: $c_3 = 1$

Thus the position number formed of $c_1c_2c_3$ is 101, which means that the location of the error is in position 5. To correct the error, the digit in position 5 is complemented, and the correct message 1101001 is obtained.

**Alphanumeric Codes**

Alphanumeric codes, also called character codes, are binary codes used to represent alphanumeric data. The codes write alphanumeric data, including letters of the alphabet, numbers, mathematical symbols and punctuation marks, in a form that is understandable and processable by a computer. These codes enable us to interface input–output devices such as keyboards, printers, VDUs, etc., with the computer. Two widely used alphanumeric codes include the ASCII and the EBCDIC codes.

**Internal code:**

It is a six bit code which is used to represent alphanumeric characters. As it represent few characters, it is not used in any applications nowadays.

| Example: | A | -- | 010 001 | B | -- | 010 010 |
|---|---|---|---|---|---|---|
| | 0 | -- | 000 000 | 1 | -- | 000 001 |

**ASCII code:**

The ASCII (American Standard Code for Information Interchange), pronounced 'ask-ee', is a seven-bit code based on the English alphabet. ASCII codes are used to represent alphanumeric data in computers, communications equipment and other related devices. Since it is a seven-bit code, it can at the most represent 128 characters. It currently defines 95 printable characters including 26 upper-case letters (A to Z), 26 lower-case letters (a to z), 10 numerals (0 to 9) and 33 special characters including mathematical symbols, punctuation marks and space character.

| | | | | | | |
|---|---|---|---|---|---|---|
| Example: | A | -- | 100 0001 | B | -- | 100 0010 |
| | 0 | -- | 011 0000 | 1 | -- | 011 0001 |
| | NUL | -- | 000 0000 | DEL | -- | 111 1111 |
| | + | -- | 010 1011 | ? | -- | 011 1111 |

**EBCDIC code:**

The EBCDIC (Extended Binary Coded Decimal Interchange Code), pronounced 'eb-si-dik', is another widely used alphanumeric code, mainly popular with larger systems. The code was created by IBM to extend the binary coded decimal that existed at that time. It is an eight-bit code and thus can accommodate up to 256 characters.

| | | | | | | |
|---|---|---|---|---|---|---|
| Example: | A | -- | 1100 0001 | B | -- | 1100 0010 |
| | 0 | -- | 1111 0000 | 1 | -- | 1111 0001 |
| | NUL | -- | 0000 0000 | DEL | -- | 0000 0111 |
| | + | -- | 0100 1110 | ? | -- | 0110 1111 |