Digital Logic and Digital Systems

Introduction

A digital computer stores data in terms of digits (numbers) and proceeds in discrete steps from one state to the next. The states of a digital computer typically involve binary digits which may take the form of the presence or absence of magnetic markers in a storage medium , on-off switches or relays.

In digital computers, even letters, words and whole texts are represented digitally. Digital Logic is the basis of electronic systems, such as computers and cell phones. Digital Logic is rooted in binary code, a series of zeroes and ones each having an opposite value. This system facilitates the design of electronic circuits that convey information, including logic gates. Digital Logic gate functions include and, or and not. The value system translates input signals into specific output.

Digital Logic facilitates computing, robotics and other electronic applications. Digital Logic Design is foundational to the fields of electrical engineering and computer engineering. Digital Logic designers build complex electronic components that use both electrical and computational characteristics. These characteristics may involve power, current, logical function, protocol and user input. Digital Logic Design is used to develop hardware, such as circuit boards and microchip processors. This hardware processes user input, system protocol and other data in computers, navigational systems, cell phones or other high-tech systems.

Machine Level Representation of Data

Numeric systems

The numeric system we use daily is the decimal system, but this system is not convenient for machines since the information is handled codified in the shape of on or off bits; this way of codifying takes us to the necessity of knowing the positional calculation which will allow us to express a number in any base where we need it.

Radix number systems

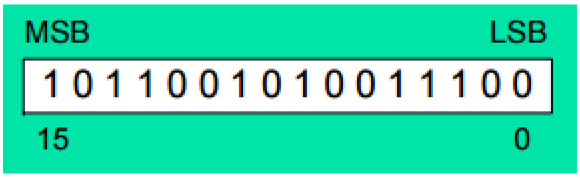
The numeric system we use daily is the decimal system, but this system is not convenient for machines since the information is handled codified in the shape of on or off bits; this way of codifying takes us to the necessity of knowing the positional calculation which will allow us to express a number in any base where we need it.

A base of a number system or radix defines the range of values that a digit may have.

In the binary system or base 2, there can be only two values for each digit of a number, either a "0" or a "1".

□ MSB – most significant bit □

LSB – least significant bit



In the octal system or base 8, there can be eight choices for each digit of a number:

"0", "1", "2", "3", "4", "5", "6", "7".

In the decimal system or base 10, there are ten different values for each digit of a number:

In the hexadecimal system, we allow 16 values for each digit of a number:

"0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "A", "B", "C", "D", "E", and "F".

Where “A” stands for 10, “B” for 11 and so on

Conversion among radices

- Convert from Decimal to Any Base

Let’s think about what you do to obtain each digit. As an example, let's start with a decimal number 1234 and convert it to decimal notation. To extract the last digit, you move the decimal point left by one digit, which means that you divide the given number by its base 10

1234/10 = 123 + 4/10

The remainder of 4 is the last digit. To extract the next last digit, you again move the decimal point left by one digit and see what drops out.

123/10 = 12 + 3/10

The remainder of 3 is the next last digit. You repeat this process until there is nothing left. Then you stop. In summary, you do the following:

1234/10 = 123 ……….. 4

123/10 = 12 …………. 3

12/10 = 1 ………… 2

1/10 = 0 …………. 1 (Stop when the quotient is 0)

1 2 3 4 (Base 10)

Now, let's try a nontrivial example. Let's express a decimal number 1341 in binary notation. Note that the desired base is 2, so we repeatedly divide the given decimal number by 2.

1341/2 = 670 …………. 1

670/2 = 335 …………. 0

335/2 = 167 …………. 1

167/2 = 83 ………… 1

83/2 = 41 …………. 1

41/2 = 20 …………. 1

20/2 = 10 …………. 0

10/2 = 5 …………. 0

5/2 = 2 …………. 1

2/2 = 1 …………. 0

1/2 = 0 …………. 1

1 0 1 0 0 1 1 1 1 0 1 (BIN; Base 2)

Let's express the same decimal number 1341 in octal notation.

1341/8 = 167 …………. 5

167/8 = 20 …………. 7

20/8 = 2 …………. 4

2/8 = 0 …………. 2

2 4 7 5 (OCT; Base 8)

Let's express the same decimal number 1341 in hexadecimal notation.

1341/16 = 83 ………….13

83/16 = 5 …………. 3

5/16 = 0 …………. 5

5 3 D (HEX; Base 16)

Example. Convert the decimal number 3315 to hexadecimal notation. What about the hexadecimal equivalent of the decimal number 3315.3?

Solution:

3315/16 = 207 ……………3

207/16 = 12 ……………..15

12/16 = 0 ………………..12 (Stop when the quotient is 0)

C F 3 (HEX; Base 16)

0.3

0.3*16 = 4.8 …………….4

0.8*16 = 12.8 ………….12

0.8*16 = 12.8 ………….12

0.8*16 = 12.8 ………….12

(HEX; Base 16) 0.4 C C C ...

Thus, 3315.3 (DEC) --> CF3.4CCC... (HEX)

Example. Convert 234.14 expressed in an octal notation to decimal.

$2*8^{2} + 3*8^{1} + 4*8^{0} + 1*8^{-1} + 4*8^{-2}$

= 2*64 +3*8 +4*1 +1/8 +4/64 =156.1875

Example. Convert the hexadecimal number 4B3 to decimal notation. What about the decimal equivalent of the hexadecimal number 4B3.3?

Solution:

Original Number:	4	B	3	.	3
How Many Tokens:	4	11	3		3
Digit/Token Value:	256	16	1	0.0625	
Value:	1024 +	176 +	3 +	0.1875	

= 1203.1875

- Relationship between Binary - Octal and Binary-hexadecimal

As demonstrated by the table bellow, there is a direct correspondence between the binary system and the octal system, with three binary digits corresponding to one octal digit. Likewise, four binary digits translate directly into one hexadecimal digit.

BIN	OCT	HEX	DEC
0000	00	0	0
0001	01	1	1
0010	02	2	2
0011	03	3	3
0100	04	4	4
0101	05	5	5
0110	06	6	6
0111	07	7	7
1000	10	8	8
1001	11	9	9
1010	12	A	10
1011	13	B	11
1100	14	C	12
1101	15	D	13
1110	16	E	14
1111	17	F	15

For conversion from base 2 to base 16, we use groups of four.

Consider converting 101102 to base 8

$101102 = 010_2 \quad 110_2 = 2_8\, 6_8 = 268$

Notice that the leftmost two bits are padded with a 0 on the left in order to create a full triplet

Now consider converting 101101102 to base 16:

$10110110_2 = 1011_2 \quad 0110_2 = B_{16} \quad 6_{16} = B6_{16}$

(Note that 'B' is a base 16 digit corresponding to 1110. B is not a variable.)

The conversion methods can be used to convert a number from any base to any other base, but it may not be very intuitive to convert something like 513.03 to base 7. As an aid in performing an unnatural conversion, we can convert to the more familiar base 10 form as an intermediate step, and then continue the conversion from base 10 to the target base. As a general rule, we use the polynomial method when converting into base 10, and we use the remainder and multiplication methods when converting out of base 10.

Mathematical Operations

SUM and SUB

The method of complements is especially useful in binary (radix 2) since the ones' complement is very easily obtained by inverting each bit (changing '0' to '1' and vice versa). And adding 1 to get the two's complement can be done by simulating a carry into the least significant bit. For example:

01100100 (x, equals decimal 100)

–

00010110 (y, equals decimal 22)

becomes the sum:

01100100 (x)

+ 11101001 (ones' complement of y)

+ 1 (to get the two's complement)

==========

101001110

Using Binary to Store Text

Storing numbers using binary is easy as binary is a counting system for numbers. To store text characters we have to come up with a different solution.

Task 4 - A System for Storing Text Split into pairs and collect a piece of scrap paper from your teacher. Your task is as follows: 1. Design a method of storing a single character (A, v, Z etc) using a pattern of 1s and 0s. 2. Once you've decided how to store your characters, use your method to write a three letter binary message for your partner. Give you partner the coded binary message. 3. Now try to decode each others binary messages. Could you decode the other person's message?

Unless you are extremely good at decoding messages (and very lucky) you will have discovered that it is nearly impossible to decode the message without knowing the method your partner used. Task 4 simulates what happened in the early days of computing when methods of storing text were developed. The problem with everyone deciding how each character will be stored is that nobody can understand anybody else's codes. Any text you save can't be viewed by anyone using a different code. As with many developments in technology, eventually most of the methods died out leaving only a few. From those few, one method now rules

ASCII (American Standard Code for Information Interchange)

ASCII uses 8 bit binary numbers to represent text characters.

An 8 bit code allows 256 different characters to be stored:

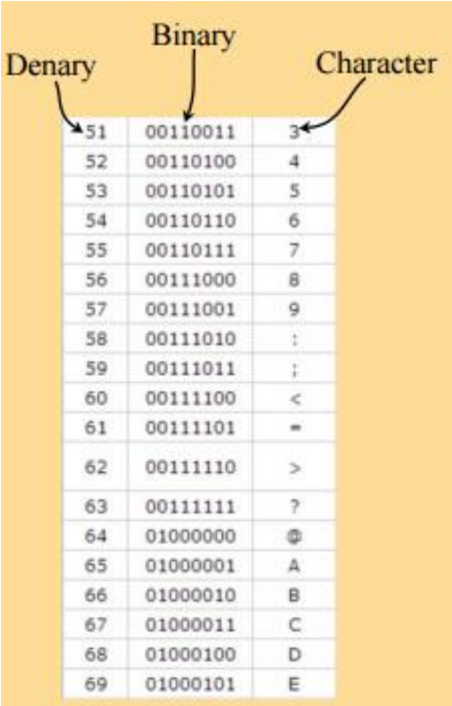
A-Z - 26 characters

a-z - 26 characters

Control Characters (return, tab etc) - 32 characters ü 0-9 - 10 characters

Punctuation - approximately 20 characters

Mathematical Symbols - approximately 50 characters



Denary	Binary	Character
51	00110011	3
52	00110100	4
53	00110101	5
54	00110110	6
55	00110111	7
56	00111000	8
57	00111001	9
58	00111010	:
59	00111011	;
60	00111100	<
61	00111101	=
62	00111110	>
63	00111111	?
64	01000000	@
65	01000001	A
66	01000010	B
67	01000011	C
68	01000100	D
69	01000101	E

Machine Code

We have learned that numbers, text and graphics are all stored as binary. To process all this data requires a computer program to provide instructions on how to calculate, move, store or display the binary values.

As everything processed in a computer system has to be in binary form it should come as no surprise now that program code is also stored as binary.

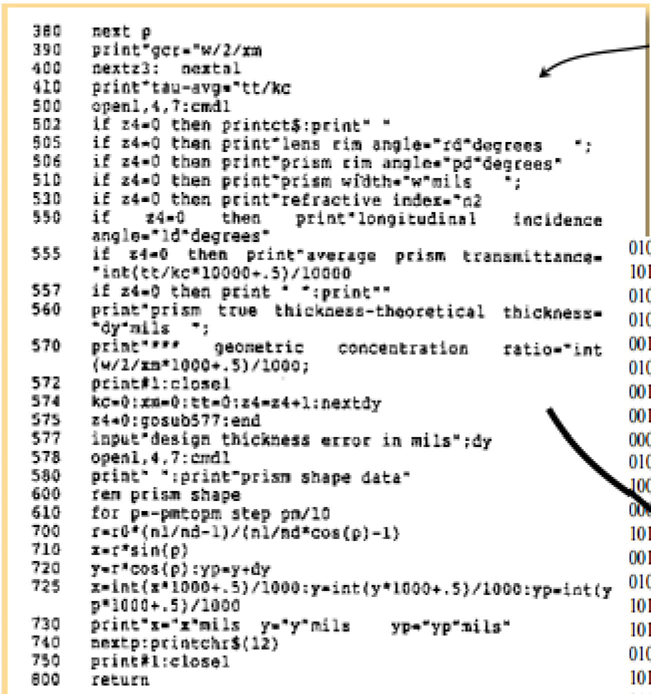
When programmers sit and write programs, they do not however write instructions in binary. Imagine how difficult it would be to understand, edit and find mistakes in long sequences of 1's and 0's.

010101010001010001001010100100100001011010101010
101001010101010001010001001010100100100001011010
001110101001010101000101000100101010010010000101111

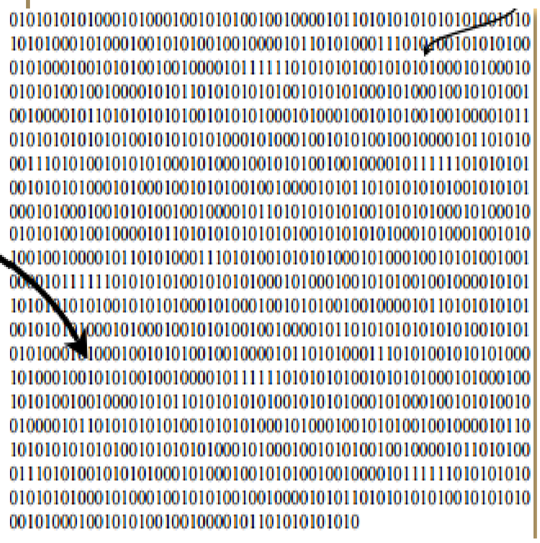
111010101010010101010001010001001010100100100000101
011010101010100101010100010100010010101001001000001

It’s much easier to write a program using an English based programming language and then translate it into binary so that the computer can then understand and process the code.

High Level Language (program written in English)



Low Level Language (translated binary version)



Computer program code in binary form is called machine code.

Variables - How Computer Programs Store Data

Computer programs are written to input, process and output data. To achieve this, the data being used or created by a program has to be stored in memory. When declaring memory locations to store the data (called variables in programming) the program will allocate a different a numbers of locations depending on the type of data being stored.

Integers 32 or 64 bits may be allocated to storing a single integer. Depending on the size of each memory location this will usually equate to only 1 or 2 locations.

Real Numbers Again 1 or 2 locations may be allocated to storing a real number. The memory locations may be split with part of the location being used to store the mantissa and part being used to store the exponent.

Characters Using ASCII code only 8 bits are required to store a single character. One character will easily fit into a single 32 or 64 bit memory location in a modern computer. Strings A string is a list of characters (word or sentence) and will require multiple memory locations to store the data.

Arrays An array is a structure that stores multiple values. The number of locations will depend on the type of data being stored and how many examples of that data. For example, an array of 1000 integers may require 1000 memory locations to be set aside.