Group-code Technique for Data Representation

Krishan Gupta
M. Tech Scholar, Computers Department
T.I.T&S College
Bhiwani, Haryana, India
krishan57gupta@gmail.com

Monika
M. Tech Scholar, Computers Department
T.I.T&S College
Bhiwani, Haryana, India
sharmamonika093@gmail.com

Dr Mukesh Sharma
Associate Professor, Computers Department
T.I.T&S College
Bhiwani, Haryana, India
drmukeshji@titsbhiwani.ac.in

***Abstract*— basically, computers just deal with numbers. They can store letters and other characters by assigning a number for each one. Before Unicode was designed, there were found hundreds of different encoding systems for assigning these numbers. No single encoding could contain sufficient characters: As for example, the European Union alone requires several different encodings to cover all its languages. Even for a solo language like English no solo encoding was adequate for all the punctuation, letters and technical symbols in common use. But Unicode takes 16-bits to represent a character that consumes more memory space and also increases the traffic over transmission. This paper describes the Group-code technique which can represent every character of different languages in 8-bit. Group-code contains all the characters of different languages in different groups. But every group represents a specific languages and member of particular group represents character of that particular group language.**

Keywords*— *Group-code technique; representation; different format; ASCII; Unicode.

I. INTRODUCTION

The name ASCII is an acronym for: American Standard Code for Information Interchange. Which is a character encoding standard established several decades ago to provide a standard mode for digital machines to encode characters. The ASCII codes offer a mechanism for encoding alphabetic characters, numeric digits, and punctuation marks. As initially designed, it was a seven bit code. The seven bits agree to the representation of 128 unique characters. The ASCII standard was later extended to 8 bit code (which permits 256 unique code patterns) and several additional symbols were added, including characters with diacritical marks (like accents) used in European languages, which do not appear in English. Unicode offers a unique number for every character, no matter what the program, no matter what the platform, no matter what the language. Basically, computers just deal with numbers. They can store letters and other characters by assigning a number for each one. Before Unicode was designed, there were found hundreds of different encoding systems for assigning these numbers. No single encoding could contain sufficient characters: As for example, the European Union alone requires several different encodings to cover all its languages. Even for a solo language like English no solo encoding was adequate for all the punctuation, letters and technical symbols in common use. These encoding systems also conflict to each another. That is, two encodings can use the identical number for 2 different characters, or use dissimilar numbers for the same character.

Any given computer (particularly servers) needs to support many different encodings; yet whenever data is delivered between different encodings or platforms, that data every time runs the risk of corruption. Group-code will used 8 bits to represent every character of every language. Unicode16 Contain maximum 128 languages and maximum 128 character in a specific language. Thus, to represent maximum 128 language with maximum 128 character of every language Group-code technique uses 8 bits. According to Group-code technique there is 128 groups will become and every group will contains 128 characters. Means every language has its group and every group have 128 characters means 128 characters of that language. As from 8 bits 256 different characters can be represent. 8 bits of Group-code also have 256 different combinations. 0 to 127 first combination will be used for group representation and 128 to 256 combination will be used for character representation of particular language. Hence, there is 8 bit enough to represent all characters of all languages which comes in ASCII 16 bit encoding scheme.

II. BRIEF HISTORY OF REPRESENTATION TECHNIQUE

A. Decimal and Binary Numbers

As decimal (base 10) numbers, a positional notation system will use. Every digit is multiplied by an appropriate power of 10 depending on its position in the number: For example: 853 = 8 x 10^2 + 5x 10^1 + 3 x 10^0 = 8 x 100 + 5 x 10 + 3 x 1 = 800 + 50 + 3 for whole numbers, the rightmost figure position is the one's position. The numeral in that position indicates how many ones are present in the number. The come on next position to the left is ten's, then hundred's, thousand's, and so on. Every figure position has a weight that is ten times the weight of the position to its right. In the decimal number system, there are ten expected values that can appear in each digit position, and so there are ten numerals required to represent the quantity in each digit position. The decimal numerals are from zero to nine (0, 1, 2, 3, 4, 5, 6, 7, 8, 9). In a positional notation system, the number base is called the radix. Hence, the base ten systems that we usually use have a radix of 10. The word radix and base can be used interchangeably. The binary number system is also a positional notation numbering system, but in this case, the base is not ten, but is instead two. Every digit position in a binary number represents a power of two. So, whenever we write a binary number,

978-1-4799-4910-6/14 $31.00 © 2014 IEEE
DOI 10.1109/ACCT.2014.86

every binary digit is multiplied by an appropriate power of 2 based on the position in the number: For example:101100 = 1 x 2^5 + 0 x 2^4 + 1 x 2^3 + 1 x 2^2 + 0 x 2^1 + 1 x 2^0= 1 x 32 + 0 x 16 + 1 x 8 + 1 x 4 + 0 x 2 + 0 x 1= 32 + 8 + 4 + 0. In binary number system are only two possible values 0 and 1 that can appear in each digit position rather than the ten from 0 to 9 that can appear in a decimal number. Only the numerals 0 and 1 in place of 0 to 9 are used in binary numbers.

B. Hexadecimal Numbers

In adding to binary, another number base that is usually used in digital systems is base 16 is called hexadecimal, and each digit position represents a power of 16. For every number base greater than ten, a problem arises because there are more than ten symbols needed to represent the numerals for that number base. It is expected in these cases to use the ten decimal numerals followed by the letters of the alphabet start with A to F provide the needed numerals. As the hexadecimal system is base 16 sixteen numerals required. The following are the hexadecimal numerals: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F. The motive for the common use of hexadecimal numbers is the relationship between the numbers 2 and 16. Sixteen is a power of 2 (16 = 2^4). As of this relationship, four digits in a binary number can be represented with a single hexadecimal digit. When working with large digital systems, such as computers, it is common to find binary numbers with 8, 16 and also 32 digits. By using hexadecimal, the numbers can be written with less digits and much less possibility of error. In the C and C++ and java languages, hexadecimal constants are represented with a '0x' preceding the number, as in: 0x547F, or 0x1654, or 0xCF.

C. Binary Coded Decimal Numbers

Another number system which is encountered occasionally is Binary Coded Decimal. Numbers are represented in a decimal form each decimal digit is encoded using a four bit binary number. For example: The decimal number 131 would be represented in BCD as follows: 131 = 0001 0011 0001. There are a couple of differences on the BCD representation, namely packed and unpacked. A packed BCD representation, two decimal digits are placed in each byte. Normally, the high order bits of the data byte contain the more significant decimal digit. But in the unpacked BCD number has only a single decimal digit kept in each data byte. In this case, the decimal digit will be in the low four bits and the upper 4 bits of the byte will be. The use of BCD to represent numbers isn't as common as binary in most computer systems, as it is not as space effective. In packed BCD, only 10 out of the 16 possible bit patterns in each 4 bit unit are used. In unpacked BCD, only 10 out of the 256 possible bit patterns in each byte are used. A Sixteen bits quantity can represent the range 0 to 65535 in binary, 0 to 9999 in packed BCD and only 0to 99 in unpacked BCD.

D. ASCII Character Encoding

ASCII is a character encoding standard developed several decades ago to provide a standard way for digital machines to encode characters. There are numerous non-standard extensions to ASCII giving different encoding for the upper 128 character codes than the standard. The character set encoded into the display card for the original IBM PC had a non-standard encoding for the upper character set. Which is a non-standard extension that is in very wide spread use, and could be considered a standard in itself. **ASCII Character Set** NUL DLE Space 0 @ P ` p SOH DC1! 1 A Q a q STX DC2 " 2 B R b r ETX DC3 # 3 C S c s EOT DC4 $ 4 D T d t ENQ NAK % 5 E U e u ACK SYN & 6 F V f v BEL ETB ' 7 G W g w BS CAN (8 H X h x HT EM) 9 I Y i y LF SUB * : J Z j z VT ESC + ; K [k { FF FS , < L \ l | CR GS - = M] m } SO RS . > N ^ n ~ **SI** US /? O _ o DEL

E. About the Unicode Consortium

The Unicode Consortium was founded to develop, promote and extend use of the Unicode Standard, which indicates the representation of text in modern software products and standards. The Consortium is a non-profit, 501(c)(3) charitable organization. Consortium membership represents a broad spectrum of organizations and corporations in the computer and information processing industry. The Consortium is supported financially through membership donations and dues. Membership for Unicode Consortium is open to individuals and organizations anywhere in the world who support the Unicode Standard and wish to assist in its implementation and extension. All are invited to contribute to the support of the Consortium's important work by making a donation. Unicode provides a unique number for every character, no matter what the program, no matter what the platform, no matter what the language. The Unicode Standard has been adopted by such industry leaders as Apple, HP, Microsoft, Just Systems, IBM, SAP, Oracle, Unisys, Sybase, Sun and many others. Unicode is required by modern standards such as XML, ECMAScript (JavaScript), Java LDAP, WML, CORBA 3.0, etc., and is the official way to implement ISO/IEC 10646 and supported in many operating systems, all modern browsers, and various other products. The availability of tools supporting it and emergence of the Unicode Standard, are among the most significant recent global software technology trends.

III. PROPOSED GROUP-CODE TECHNIQUE

In Group-code technique representation first of all we will make the groups of all different languages. And group size and number of groups will depend upon number of bit in which we want to represent. As Unicode16 Contains maximum 128 language and maximum 128 character in a specific language. Thus to represent 128 language with maximum 128 character of every language. Group-code technique use 8 bits. According to Group-code Technique, there is 128 group will become and every group will contains 128 characters means every language have its group and every group have maximum 128 characters. As from 8 bits 256 different characters can be represented. 8 bits of Group-code also have 256 different combinations. 0 to 127 first combination will be used for group representation and 128 to 256 combination will be used for character representation of that particular language. For identity of every character group code + character code will use. But every time there is no need group code for identity

group code as necessary one time means same group character needs only character code. For Example first of all group code come then for its all characters every time character code is needed but if an only if the character of different group comes then there is need become to take group code of that group. Hence, Group-code technique takes first of all group code than always character code to continue for all the characters of same group whenever next group character not come. If next Group character come then again take group code than always character code to continue for all the characters of same group whenever next group character not come.

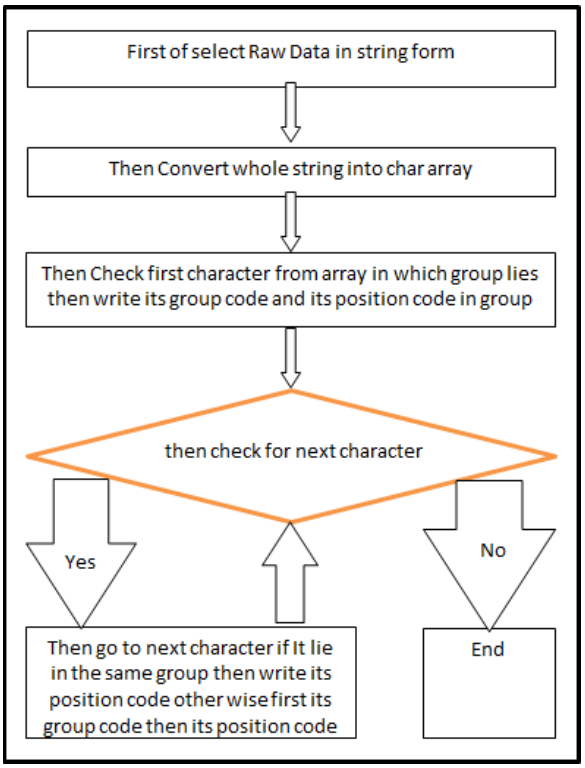
A. Initialization

First off all we initialize all the different characters in an array. We can use a 2 dimensional array to store all the different characters. Here first dimension is used for the number of groups and second dimension is used for storing different characters in groups. Hence, here two dimensional array will become of size 128*128 as in Java language representation Char[][] Group-Code=newChar[128][128]. Hence, there storage will become of 128*128=16384 will be enough to control all the characters of all the languages.

B. Conversion of Text Data into Bits Data

Here we simply represent the whole text data into bit data according to the following DFD.

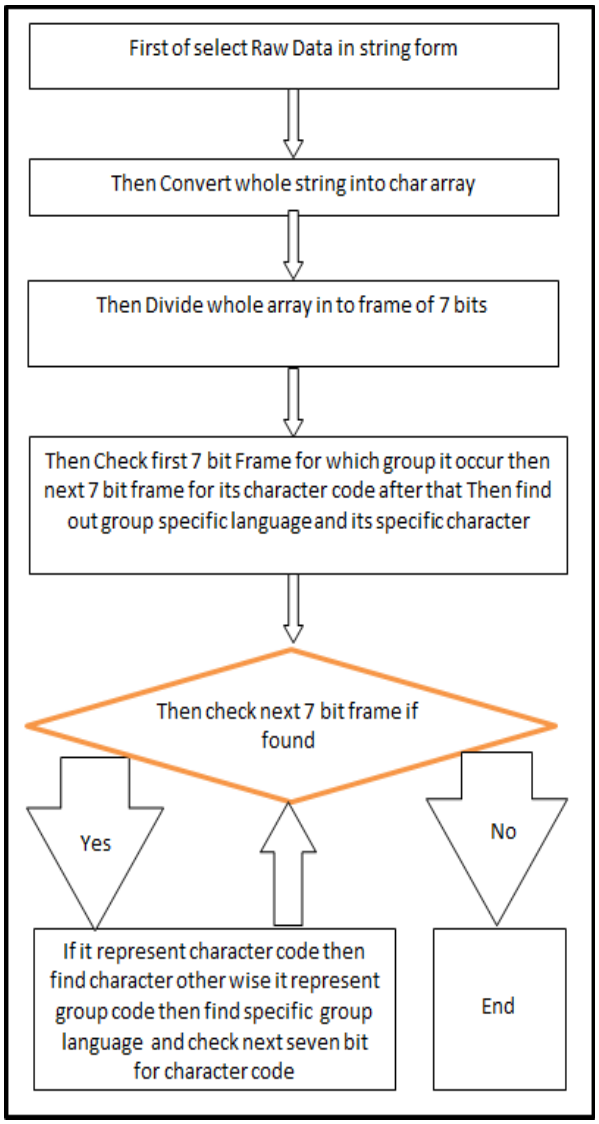
DFD (Data Flow Diagram)



C. Reverse Of This Representation is From Bits to Text Data

Here we will again do initiaization of different characters as done in previous method. As according to previous illustration, 256 different character initialization is done by init1() method we will represent its reverse from bit to text data.

DFD (Data Flow Diagram)



IV. PERFORMANCE OF GROUPCODE TECHNIQUE

Here we have shown how the bits can be reduced by the Groupcode technique representation as compared to the simple representation(ASCII UNICODE And Some Other). The performance of Groupcode technique can be observed with the help of following table:

Number of different objects	Simple representation required bits	Groupcode representation required bits
256(ASCII)	8	5 +n1 (overhead bits to change group)
128(ASCII)	7	5 +m1 (overhead bits to change group but Here m1<n1 always)
256(Unicode8)	8	5+n2(overhead bits to change group)
65536(Unicode16)	16	8+n3(overhead bits to change group)
Unicode32	32	17+n4(overhead bits to change group)
16777216(for 16 m color)	24	13+n5(overhead bits to change group)
4284967296(for neon technology)	32	17+n6(overhead bits to change group)

Table1: Bits Required in Simple And 'GROUP' Technique

The comparison between the Groupcode technique for data representation and the simple techniques for the data representations can be made and this has been tabulated in the following table:

No. of different objects	No. of objects in raw data	Simple representation required total bits	Group-code representation on required total bits	Simple representation required total bits and average bits both same in every case	Average bits required by Group-code representation
256(ASCII)	1000	1000*8 =8000	1000*5 =5000 to 1000*5+ 1000*5 =10000	8000	7500
128(ASCII)	1000	1000*7 =7000	1000*5 =5000 to 1000*5+ 1000*5 =10000	7000	<7500
256(Unicode8)	1000	1000*8 =8000	1000*5 =5000 to 1000*5+ 1000*5 =10000	8000	7500
Unicode16(Simple Case)	1000	1000*16 =16000	1000*9 =9000 to 1000*9+ 1000*9 =18000	16000	13500
Unicode32	1000	1000*32 =32000	1000*17 =17000 to 1000*17 +1000*17 =34000	32000	25500
16777216(for 16 m color)	1000	1000*24 =24000	1000*13 =13000 to 1000*13 +1000*13 =26000	24000	19500
Unicode16(Complex Case after on the bases of its character)	1000	1000*16 =16000	1000*8 =8000 to 1000*8+ 1000*8 =16000	16000	12000

Table 2 : Bits Required in Simple And Group code Technique And Average Bits Required In Group code Technique

V. CONCLUSION

Group-code technique can be used to represent data in less number of bits as compared to other representation methods available historically as Unicode. With the help of Group-code technique, we can compress the data which is very helpful to send data at fast rate over network and also consume less space in memory. As in Unicode representation as 16 bits required representing all the character of all different languages but Group code makes the group equivalent to languages and useful to represent all the character of with all different languages. And after some experiments in Unicode found that there is maximum 128 language and maximum128 characters in a specific language. By this Experiment, found that Group code technique is become more powerful because now Group code technique will represent every character in only 8 bits. Hence, Group code technique is a powerful technique to represent data in less number of bits as compare to Unicode. The technique described here, will gives us better encoding scheme without any loss in the data. It will not only reduces storage requirements but also overall execution time. Thus data can be modified and designed in such a way that it is able to decode the part of compressed transmitted data which is error free, therefore identify and either correcting or leaving the erroneous data.

VI. FUTURE PROSPECTS

For the future point of view, Group-code technique is a big technique which will help us achieve less space uses and transmission power goals very well. According to our technique results, in future it provide us higher compression ratio for any kind of data as well as image data. It provides a potential cost savings related to sending less data over switched telephone networks. It will reduces probability of transmission errors since fewer bits are transferred and will also provide level of security against unauthenticated users in future.

REFERENCES

[1] Unicode Definition
http://en.wikipedia.org/wiki/Unicode

[2] Unicode Defination
http://en.wikipedia.org/wiki/Unicode

[3] Unicode is changing all that!
www.unicode.org/standard/WhatIsUnicode.html

[4] Unicode Table
www.tamasoft.co.jp/en/general-info/unicode.html