Data Representation

Some binary representations of data:

1.BCD code

→ Each decimal number is represented by 4 bit number

2.Gray Code

→ Each bit is x-or’d to the front bit

3.Self complimenting code

→ The number is complimented

4.Weighted code

→ Each bit in the number has a weight asigned to it like the 1 in 100 has weight 100

5.Access 3 code

→ 3 is added to the number

6.EBCDIC code

→ Standard before ASCII

7.ASCII code

→ Binary representation for charecters and some special symbols where each character takes 7 bits

8.Unicode

→ Modern standard for representing symbols and charecters where each bit take 64 bits

Grey code:

|  |  |  |
| --- | --- | --- |
| Decimal | Binary | Gray code |
| 1 | 1 | 1 |
| 2 | 10 | 11 |
| 3 | 11 | 10 |
| 4 | 100 | 110 |
| 5 | 101 | 111 |
| 6 | 110 | 101 |
| 7 | 111 | 100 |
| 8 | 1000 | 1100 |
| 9 | 1001 | 1101 |

ASCII code :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Uppercase | code | Lowercase | code | Digits | code |
| A | 65 | a | 97 | 0 | 48 |
| B | 66 | b | 98 | 1 | 49 |
| C | 67 | c | 99 | 2 | 50 |
| D | 68 | d | 100 | 3 | 51 |
| ... | ... | ... | …. | ... | …. |
| Z | 90 | z | 122 | 9 | 87 |

Unicode:

some of the standars are:

1.UTF8

2.UTF16

1’s compliment → flip the bits

2’s compliment → flip the bits and add one

9’s compliment → subtract each digit from 9

10’s compliment → subtract each digit from 9 and add one

**Floating point representation:**

any number in decimal form (ex 1.21) in floating point representation will be

0.121x101 where the red number is exponent , blue number is base, yellow number is significand and green number is fraction part.

|  |  |  |
| --- | --- | --- |
| Sign | Exponent | Significand |
| 1 bit | 8 bit | 23 bit |

The exponent is biased with 127 to represent negetive numbers.

Example:

3.1 = 11.000110…

Changing it in normalized form:

1.10001100011...x21

floating point format(no biasing)

0 00000001 10001100011000110001100

Green = signed bit

Red = exponent

Yellow = Mantissa/significand

**Error Checking using parity:**

Parity is the number of 1 bit in a data. If checking for even parity there must be even number of 1 bit and if checking for odd parity there must be odd number of 1 bit. The parity is checked using a extra parity bit.

Parity generator:

P = (x xor y xor z)’

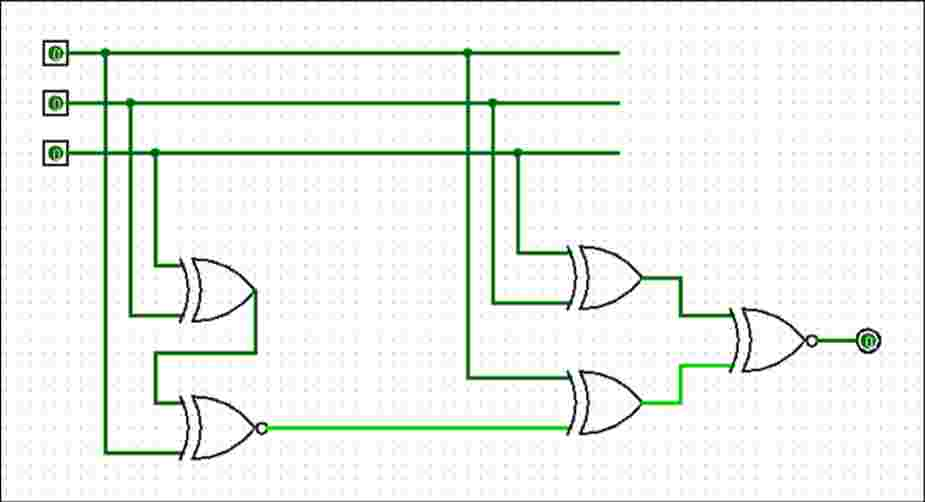
messege p-bit

001 0

011 1

100 0

101 1



Parity Checker:

e = (p xor x xor y xor z)’

**Multiplication:**

Signed magnitude multiplication:

Flowchart of multiplication:

Multiplicand in B

Multiplier in Q

As ← Qs xor Bs

Qs ← Qs xor Bs

A ← 0 , E ← 0

SC ← n-1

0 1

EA ← A+B

Qn

Shr EAQ

SC ← SC -1

SC

1

0

END

Example:

multiplicand (B) = 10111

multiplier (Q) = 10011

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SC | E | A | Q | operation |
| 5 | 0 | 00000 | 10011 | initialization |
|  | 0 | 10111 | 10011 | Qn is 1 so EA ← A+B |
| 4 | 0 | 01011 | 11001 | Shr EAQ , SC ← CS-1 |
|  | 1 | 00010 | 11001 | Qn =1 so EA ← A+B |
| 3 | 0 | 10001 | 01100 | Shr EAQ , dcr SC |
| 2 | 0 | 01000 | 10110 | Shr EAQ , dcr SC |
| 1 | 0 | 00100 | 01011 | Shr EAQ , dcr SC |
|  | 0 | 11011 | 01011 | Qn = 1 EA ← A+B |
| 0 | 0 | 01101 | 10101 | Shr EAQ , dcr SC |

Product = 0110110101

**Booth’s Multiplication(Signed multiplication):**

**flowchart in phone**

Multiply -9 x -13

here,

multiplicand(B) = 01001 → 10111 (2’s compliment represent -ve )

multiplier(Q) = 01101 → 10011 (2’s compliment represent -ve )

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SC | A | Q | Qn+1 | operation |
| 5 | 00000 | 10011 | 0 | init |
| 4 | 01001 | 10011 | 0 | Qn Qn+1 = 10, A+B’+1→ A |
| 00100 | 11001 | 1 | Ashr AQQn+1, dcr SC |
| 3 | 00010 | 01100 | 1 | Ashr AQQn+1, dcr SC |
|  | 11001 | 01100 | 1 | Qn Qn+1 = 01 , A+B→ A |
| 2 | 11100 | 10110 | 0 | AshrAQQn+1,dcrsc |
| 1 | 11110 | 01011 | 0 | Ashr AQQn+1, dcr SC |
|  | 00111(discard carry) | 01011 | 0 | Qn Qn+1 = 10, A+B’+1→ A |
| 0 | 00011 | 10101 | 1 | Ashr AQQn+1, dcr SC |

Product = 00011101011 → 117