Activities:

(1)

a) 1001(2=>1\*2(3)+0\*2(2)+0\*2(1)+1\*2(0)=9(10

b) 110010(2=>1\*2(5)+1\*2(4)+0\*2(3)+0\*2(2)+1\*2(1)+0\*2(0)=50(10

c) 1010(2=>1\*2(3)+0\*2(2)+1\*2(1)+0\*2(0)=10(2

d) 100101,101(2=>1\*2(5)+0\*2(4)+0\*2(3)+1\*2(2)+0\*2(1)+1\*2(0)+1\*2(-1)+0\*2(-2)+1\*2(-3)=37,625(10

e) 1011(2=>1\*2(3)+0\*2(2)+1\*2(1)+1\*2(0)=11(10

(2)

a) 8(10=>1\*2(3)+0\*2(2)+0\*2(1)+0\*2(0)=1000(2

b) 512(10=>1\*2(9)+0\*2(8)+0\*2(7)+0\*2(6)+0\*2(5)+0\*2(4)+0\*2(3)+0\*2(2)+0\*2(1)+0\*2(0)=1000000000(2

c) 20,625(10=>1\*2(4)0\*2(3)+1\*2(2)+0\*2(1)+0\*2(0) )+1\*2(-1)+0\*2(-2)+1\*2(-3)=10100,101(2

d) 255(10=>1\*2(7)+1\*2(6)+1\*2(5)+1\*2(4)+1\*2(3)+1\*2(2)+1\*2(1)+1\*2(0)=11111111(2

e) 3560,75(10 = 1\*2(11)+1\*2(10)+>0\*2(9)+1\*2(8)+1\*2(7)+1\*2(6)+1\*2(5)+0\*2(4)+1\*2(3)+0\*2(2)+0\*2(1)+0\*2(0)+1\*2(-1)+1\*2(-2) = 110111101000,11(2

(3)

a) 100100101(2 = 125(16

b) 1000000000(2 = 200(16

c) 1001001(2 = 49(16

d) 11111(2 = 1F(16

(4)

a) 5A43(16 = 101101001000011(2

b) BEA(16 = 101111101010(2

c) 23A(16 = 1000111010(2

d) 100(16 = 100000000(2

e) F410(16 = 1111010000010000(2

(5)

a) 100101(2 = 45(8

b) 11101(2 = 35(8

c) 110011(2 = 63(8

d) 100(2 = 4(8

e) 11010101(2 = 325(8

(6)

a) 521(8 = 101010001(2

b) 1234(8 = 1010011100(2

c) 100(8 = 1000000(2

d) 7543(8 = 111101100011(2

e) 111(8 = 1001001(2

(7)

a) F2A3(16 => 1111001010100011(2 => 62115(10

b) 4227(16 => 100001000100111(2 => 16935(10

c) 4227(8 => 100010010111(2 => 2199(10

d) AAFF(16 => 1010101011111111(2 => 43775(10

(8)

a) 16(10 => 10000(2 => 10(16

b) 427(10 => 110101011(2 => 1AB(16

c) 255(10 => 11111111(2 => FF(16

d) 534(10 => 1000010110(2 => 216(16

(9)

a) 16(10 => 10000(2 => 20(8

b) 427(10 => 110101011(2 => 653(8

c) 255(10 => 11111111(2 => 377(8

d) 534(10 => 1000010110(2 => 1026(8

(10)

45(10 => 1\*2(5)+0\*2(4)+1\*2(3)+1\*2(2)+0\*2(1)+1\*2(0)=101101(2

31(10 => 1\*2(4)+1\*2(3)+1\*2(2)+1\*2(1)+1\*2(0)=11111(2

111111  
 101101  
 +11111  
 1001100

10011002 => 1\*2(6)+0\*2(5)+0\*2(4)+1\*2(3)+1\*2(2)+0\*2(1)+0\*2(0)=**76(10**

45+31=**76**

(11)

80(10 => 1\*2(6)+0\*2(5)+1\*2(4)+0\*2(3)+0\*2(2)+0\*2(1)+0\*2(0)=1010000(2

46(10 => 1\*2(5)+0\*2(4)+1\*2(3)+1\*2(2)+1\*2(1)+0\*2(0)=101110(2

1010000  
1011100  
 -101110  
 100010

100010(2 => 1\*2(5)+0\*2(4)+0\*2(3)+0\*2(2)+1\*2(1)+0\*2(0)=**34(10**

80-46=**34**

(12)

109(10 => 1\*2(6)+1\*2(5)+0\*2(4)+1\*2(3)+1\*2(2)+0\*2(1)+1\*2(0)=1101101(2

23(10 => 1\*2(4)+0\*2(3)+1\*2(2)+1\*2(1)+1\*2(0)=10111(2

1101101  
101100  
 -10111  
1010110

1010110(2 => 1\*2(6)+0\*2(5)+1\*2(4)+0\*2(3)+1\*2(2)+1\*2(1)+0\*2(0)=**86(10**

109-23=**86**

(13)

30(10 => 1\*2(4)+1\*2(3)+1\*2(2)+1\*2(1)+0\*2(0)=11110**(2**

6(10 => 1\*2(2)+1\*2(1)+0\*2(0)=110**(2**

11110  
 \*110  
11110000  
00000  
111100  
 +1111 .  
10110100

10110100(2 => 1\*2(7)+0\*2(6)+1\*2(5)+1\*2(4)+0\*2(3)+1\*2(2)+0\*2(1)+0\*2(0)=**180**(10

30\*6=**180**

(14)

58(10  => 1\*2(5)+1\*2(4)+1\*2(3)+0\*2(2)+1\*2(1)+0\*2(0)= 111010(2

00111010 =>10111010(2 SM  
00111010 =>11000101(2 C1  
00111010 =>11000110(2 C2  
00111010 => K=28-1 = 128 => 58 – 128 = -70

(15)

10101010 => K=28-1= 128 => 170-128= 42

(16)

a) NOT (10001001 OR 10111001) = NOT 10111001 = 1000110

b) 11011011 XOR 10111001 = 1100010

c) 00000111 AND 11111111 = 00000111

d) 00000111 XOR 11111111 = 11111000

(17)

Siempre  
62(10 = 111110=> 6bits

(18)

12bits => 212 = 4096(10

(19)

Unicode es un estándar que puede usar mas de byte para cada carácter, por lo que permite un rango de representación mucho mas amplio.

Sus tres elementos mas importantes son:

Punto de código, glifo y codificación.

Según su codificación puede ocupar mas o menos

UTF8 – de 1 a 6 Bytes – 1 a 256bits

UTF16 – un mínimo 2 Bytes – un mínimo de 16bits por carácter

UTF32 – un valor fijo de 4 Bytes – un valor fijo de 64bits por carácter

(20)

83 105 115 116 101 109 97 115 32 100 101 32 114 101 112 114 101 115 101 110 116 97 99 105 162 110

53 69 73 74 65 6D 61 73 20 64 65 20 72 65 70 72 65 73 65 6E 74 61 63 69 A2 6E

123 151 163 164 145 155 141 163 040 144 145 040 162 145 160 162 145 163 145 156 164 141 143 151 242 156

(21)

C19E0000(16) => 11000001100111100000000000000000(2)Signo = 1(2) = -   
Exponente = 10000011 = 131 – 127 = 4  
Mantisa = 00111100000…000 => 1,001111000…000

-10011,11 = -19,75(10)

(22)

34TB \*1000=34000GB\*1000=34000000MB

1200GB/1000=1,2TB/1000=0,0012PB/1024=0,0000012EB

100Mb\*1000=100000kB

6MB/s = 3628800MB/Week / 1000=3628,8GB/Week

(23)

105(10 => 0\*2(6)+1\*2(5)+1\*2(4)+0\*2(3)+1\*2(2)+0\*2(1)+1\*2(0)=1101001**(2**

5(10 => 1\*2(2)+0\*2(1)+1\*2(0)=101**(2**

1101001|10100  
-1010000010101  
00110 000000 0  
-10100000000  
000010100000 .  
-101000000  
000000000

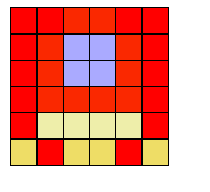
10101(2 => 1\*2(4)+0\*2(3)+1\*2(2)+0\*2(1)+1\*2(0)=**21**(10

105/5=**21**

(24)

3,5GB = 3500Mb/100Mb/s=35seg  
35/100\*5=1,75seg (de error) \*100Mb/s = 175Mb(Perdidos)

(25)



(26)

Podría hacer una suma o una resta de potencias.

Por ejemplo con la palabra 0101:

Para activar el bit 1 para obtener 0111: 0101 = 5(10 bastaría con sumarle 2 siendo 7 = 0111  
Para deshabilitar el bit 0 para obtener 0110: 0111 = 7 bastaría con restarle 1 siendo 6 = 0110  
Para obtener el estado del bit 3 bastaría saber si el valor decimal es >= 8 (activo) o <= 8 (deshabilitado)