**Mini-TP**

Alumno: Facundo Martín Sosa

Comisión: 2

DNI: 44750205

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 00 | 2D | 00 | 11 | 00 |
| 15 | FF | F1 | 01 | 13 |
| EF | E3 | 3B | 44 | 9B |
| A6 | 53 | 69 | 20 | 73 |
| 65 | 20 | 70 | 75 | 65 |
| 64 | 65 | 20 | 69 | 6D |
| 61 | 67 | 69 | 6E | 61 |
| 72 | 2C | 20 | 73 | 65 |
| 20 | 70 | 75 | 65 | 64 |
| 65 | 20 | 70 | 72 | 6F |
| 67 | 72 | 61 | 6D | 61 |
| 72 | 2E | 00 | 00 | 00 |

Los últimos dos casilleros marcados con morado son memoria libre, ya que el string termina con 0 en la convención de C, y como no hay otra variable que ocupe ese espacio entonces queda libre.

Valor1 = 002D = 0000,0000,0010,1101 = 1x2^5 + 1x2^3 + 1x2^2 + 1x2^0 = 32 + 8 + 4 + 1 = 45

Valor2 = 0011 = 0000,0000,0001,0001 = 1x2^4 + 1x2^0 = 16 + 1 = 17

Valor3 = 0015 = 0000,0000,0001,0101 = 1x2^4 + 1x2^2 + 1x2^0 = 16 + 4 + 1 = 21

Dato1 = FFF1 = 1111,1111,1111,0001 = 0000,0000,0000,1111 = -15

(Como Dato1 está en complemento A2, lo pasé a binario puro para saber que número negativo es. En este caso es 15 en BP, por ende, en complemento A2 es -15.)

Dato2 = 0113 = 0000,0001,0001,0011 = 1x2^8 + 1x2^4 + 1x2^1 + 1x2^0 = 256 + 16 + 2 + 1 = 275

1111

X 1,0001,0011

1111

1111

0000

0000

1111

0000

0000

0000

1111

0001000000011101

0001000000011101 = 1x2^12 1x2^4 + 1x2^3 + 1x2^2 + 1x2^0 = 4096 + 16 + 8 + 4 + 1 = 4125

Como sé que Dato1 es un factor negativo, y por ende, el resultado es negativo, me tomo la libertad de cambiarle el signo al producto. Quedando entonces:

1001000000011101 = -4125 (en signo magnitud)

Producto = -4125

Pero como la computadora trabaja en complemento A2, antes de pasar el producto a hexadecimal y agregarlo a la memoria, lo pasé a complemento A2.

0001000000011101 🡪1110111111100010 🡪1110111111100011 = -4125 en C2

Y el numero convertido a hexadecimal es: EFE3

En la siguiente tabla están los valores hexadecimales del string traducidos a su respectivo valor decimal y el carácter que representan:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Hexadecimal | Decimal | Carácter | Hexadecimal | Decimal | Carácter |
| 53 | 83 | S | 73 | 115 | s |
| 69 | 105 | i | 65 | 101 | e |
| 20 | 32 | ‘espacio’ | 20 | 32 | ‘espacio’ |
| 73 | 115 | s | 70 | 112 | p |
| 65 | 101 | e | 75 | 117 | u |
| 20 | 32 | ‘espacio’ | 65 | 101 | e |
| 70 | 112 | p | 64 | 100 | d |
| 75 | 117 | u | 65 | 101 | e |
| 65 | 101 | e | 20 | 32 | ‘espacio’ |
| 64 | 100 | d | 70 | 112 | p |
| 65 | 101 | e | 72 | 114 | r |
| 20 | 32 | ‘espacio’ | 6F | 111 | o |
| 69 | 105 | i | 67 | 103 | g |
| 6D | 109 | m | 72 | 114 | r |
| 61 | 97 | a | 61 | 97 | a |
| 67 | 103 | g | 6D | 109 | m |
| 69 | 105 | i | 61 | 97 | a |
| 6E | 110 | n | 72 | 114 | r |
| 61 | 97 | a | 2E | 46 | . |
| 72 | 114 | r | 00 | NULL | Fin del string |
| 2C | 44 | , | 00 | NULL | Vacío |
| 20 | 32 | ‘espacio’ | 00 | NULL | Vacío |

|  |  |
| --- | --- |
| Variable | Valor |
| Valor1 | 45 |
| Valor2 | 17 |
| Valor3 | 21 |
| Dato1 | -15 |
| Dato2 | 275 |
| Mensaje | “Si se puede imaginar, se puede programar.” |

1. Al convertir el resultado de la operación a decimal, no da el mismo número. Según mi criterio esto pasa porque al convertir un número mediante el esquema IEEE-754 y luego a hexadecimal, estamos representando el número en un formato específico para números de punto flotante. Y este proceso implica redondeos y aproximaciones que pueden generar errores como este en la representación, ya que el número está en otro formato.

Pase de coeficiente decimal a binario mediante el esquema IEEE-754:

Multiplicaciones sucesivas:

0.003 x 2 = 0.006 0.006 x 2 = 0.012 0.012 x 2 = 0.024

0.024 x 2 = 0.048 0.048 x 2 = 0.096 0.096 x 2 = 0.192

0.192 x 2 = 0.384 0.384 x 2 = 0.768 0.768 x 2 = 1.536

0.536 x 2 = 1.072 0.072 x 2 = 0.144 0.144 x 2 = 0.288

0.288 x 2 = 0.576 0.576 x 2 = 1.152 0.152 x 2 = 0.304

0.304 x 2 = 0.608 0.608 x 2 = 1.216 0.216 x 2 = 0.432

0.432 x 2 = 0.864 0.864 x 2 = 1.728 0.728 x 2 = 1.456

0.456 x 2 = 0.912 0.912 x 2 = 1.824 0.824 x 2 = 1.648

0.648 x 2 = 1.296 0.296 x 2 = 0.592 0.592 x 2 = 1.184

0.184 x 2 = 0.368 0.368 x 2 = 0.736 0.736 x 2 = 1.472

0.472 x 2 = 0.944 0.944 x 2 = 1.888

Resultado = 0.00000000110001001001101110100101

Resultado normalizado = 1.10001001001101110100101 x 2^-9

Exponente = -9 +127 = 118 = 01110110

|  |  |  |
| --- | --- | --- |
| 0 | 01110110 | 10001001001101110100101 |

Resultado en hexadecimal = 3B449BA6

3x16^7 + 11x16^6 + 4x16^5 + 4x16^4 + 9x16^3 + 11x16^2 + 10x16^1 + 6x16^0 =

805.306.368 + 184.549.376 + 4.194.304 + 262.144 + 36.864 + 2.816 + 160 + 6 = 994.352.038

Resultado en decimal = 994.352.038

1. Aparte de ASCII y ASCII-E, al menos en este caso podría usarse ISO\_8859\_1 o UTF-8, ya que ocuparían la misma cantidad de bytes para armar el mismo string. Si hubiera caracteres especiales como acentos y demás, solo podría usarse ASCII e ISO\_8859\_1, porque UTF-8 necesitaría ocupar más bytes.