Plancha 1

Representación Computacional de datos

2024 - Arquitectura del Computador

Licenciatura en Ciencias de la Computación

1

Introducción

El objetivo de esta serie de ejercicios es familiarizarse con la representación de datos orientada a la computación, sin depender de la implementación computacional propiamente dicha. Muchos de los ejercicios están diseñados para ser resueltos con papel y lápiz, aunque aunque posteriormente se pueden utilizar herramientas inform´aticas para verificar los resultados.

2

Ejercicios

1. Determinar la representación binaria de los siguientes números:

   a) 29

   b) 0,625

   c) 5,75

   d) -138

   e) -15,125

   f) 0,1

Soluciones

a)

- $29 = 2 \cdot 14 + \boxed{1} \Rightarrow a_0 = \boxed{1}$.
- $14 = 2 \cdot 7 + \boxed{0} \Rightarrow a_1 = \boxed{0}$.
- $7 = 2 \cdot 3 + \boxed{1} \Rightarrow a_2 = \boxed{1}$.
- $3 = 2 \cdot 1 + \boxed{1} \Rightarrow a_3 = \boxed{1}$.
- $1 = 2 \cdot 0 + \boxed{1} \Rightarrow a_4 = \boxed{1}$.

$\therefore 29 = (a_4 a_3 a_2 a_1 a_0)_2 = (11101)_2$.

b)

- $0,625 \cdot 2 = \boxed{1},25 \Rightarrow a_{-1} = \boxed{1}$.
- $0,25 \cdot 2 = \boxed{0},5 \Rightarrow a_{-2} = \boxed{0}$.
- $0,5 \cdot 2 = \boxed{1} \Rightarrow a_{-3} = \boxed{1}$.

$\therefore 0,625 = (0, a_{-1} a_{-2} a_{-3})_2 = (0,101)_2$.

c)

- $0,75 \cdot 2 = \boxed{1},5 \Rightarrow a_{-1} = \boxed{1}$.
- $0,5 \cdot 2 = \boxed{1} \Rightarrow a_{-2} = \boxed{1}$.

- $5 = 2 \cdot 2 + \boxed{1} \Rightarrow a_0 = \boxed{1}$.
- $2 = 2 \cdot 1 + \boxed{0} \Rightarrow a_1 = \boxed{0}$.
- $1 = 2 \cdot 0 + \boxed{1} \Rightarrow a_2 = \boxed{1}$.

$\therefore 5,75 = (a_2 a_1 a_0, a_{-1} a_{-2})_2 = (101,11)_2$.

d)

- $138 = 2 \cdot 69 + \boxed{0} \Rightarrow a_0 = \boxed{0}$.
- $69 = 2 \cdot 34 + \boxed{1} \Rightarrow a_1 = \boxed{1}$.
- $34 = 2 \cdot 17 + \boxed{0} \Rightarrow a_2 = \boxed{0}$.
- $17 = 2 \cdot 8 + \boxed{1} \Rightarrow a_3 = \boxed{1}$.
- $8 = 2 \cdot 4 + \boxed{0} \Rightarrow a_4 = \boxed{0}$.
- $4 = 2 \cdot 2 + \boxed{0} \Rightarrow a_5 = \boxed{0}$.
- $2 = 2 \cdot 1 + \boxed{0} \Rightarrow a_6 = \boxed{0}$.
- $1 = 2 \cdot 0 + \boxed{1} \Rightarrow a_7 = \boxed{1}$.

$\therefore -138 = (-a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0)_2 = (-10001010)_2$.

e)

- $0,125 \cdot 2 = \boxed{0},25 \Rightarrow a_{-1} = \boxed{0}$.
- $0,25 \cdot 2 = \boxed{0},5 \Rightarrow a_{-2} = \boxed{0}$.
- $0,5 \cdot 2 = \boxed{1} \Rightarrow a_{-3} = \boxed{1}$.
- $15 = 2 \cdot 7 + \boxed{1} \Rightarrow a_0 = \boxed{1}$.
- $7 = 2 \cdot 3 + \boxed{1} \Rightarrow a_1 = \boxed{1}$.
- $3 = 2 \cdot 1 + \boxed{1} \Rightarrow a_2 = \boxed{1}$.
- $1 = 2 \cdot 0 + \boxed{1} \Rightarrow a_3 = \boxed{1}$.

$\therefore -15,125 = (-a_3 a_2 a_1 a_0, a_{-1} a_{-2} a_{-3})_2 = (-1111,001)_2$.

f)

- $0,1 \cdot 2 = \boxed{0},2 \Rightarrow a_{-1} = \boxed{0}$.
- $0,2 \cdot 2 = \boxed{0},4 \Rightarrow a_{-2} = \boxed{0}$.
- $0,4 \cdot 2 = \boxed{0},8 \Rightarrow a_{-3} = \boxed{0}$.
- $0,8 \cdot 2 = \boxed{1},6 \Rightarrow a_{-4} = \boxed{1}$.
- $0,6 \cdot 2 = \boxed{1},2 \Rightarrow a_{-5} = \boxed{1}$.

$\therefore 0,1 = (a_0, a_{-1}\overline{a_{-2}a_{-3}a_{-4}a_{-5}})_2 = (0,0\overline{0011})_2$.

2. Repita el ejercicio anterior utilizando el sistema de numeración posicional denominado *magnitud y signo* con punto fijo:

$$(-1)^s \, (a_n a_{n-1} \ldots a_1 a_0, a_{-1} a_{-2} \ldots)_2$$

Analizar en cada caso cuántos dígitos son necesarios para poder representar cada uno de los números.

a)

- $29 = (-1)^{0}\,(11101)_2 = (011101)_{|\pm 2|}$.
- Son necesarios 6 bits.

b)

- $0,625 = (-1)^{0}\,(0,101)_2 = (00,101)_{|\pm 2|}$.
- Son necesarios 5 bits.

c)

- $5,75 = (-1)^{0}\,(101,11)_2 = (0101,11)_{|\pm 2|}$.
- Son necesarios 6 bits.

d)

- $-138 = (-1)^{1}\,(10001010)_2 = (110001010)_{|\pm 2|}$.
- Son necesarios 9 bits.

e)

- $-15,125 = (-1)^{1}\,(1111,001)_2 = (11111,001)_{|\pm 2|}$.
- Son necesarios 8 bits.

f)

- $0,1 = (-1)^{0}\,(0,\overline{00011})_2 = (00,\overline{00011})_{|\pm 2|}$.
- Son necesarios infinitos bits.

3. Convertir los siguientes números decimales a binario utilizando la representación en complemento a dos con seis bits:

   a) -16
   b) 13
   c) -1
   d) -10
   e) 16
   f) -31

   ¿Qué tienen en común todos los números negativos y todos los números positivos al utilizar esta representación?

Soluciones

a) $-16 = (-010000)_2 = (110000)_{C_2^6}$.
b) $13 = (001101)_2 = (001101)_{C_2^6}$.
c) $-1 = (-000001)_2 = (111111)_{C_2^6}$.
d) $-10 = (-001010)_2 = (110110)_{C_2^6}$.

e) $16 = (010000)_2 = (010000)_{C_2^6}$.

f) $31 = (-011111)_2 = (100001)_{C_2^6}$.

Los números negativos en complemento a 2, tienen el bit mas significativo en 1 mientras que los positivos lo tienen en 0.

4. Repetir el ejercicio anterior pero ahora utilizando 8 bits. ¿Qué conclusión se puede sacar comparando los resultados con los del ejercicio anterior?

Soluciones

a) $-16 = (110000)_{C_2^6} = (11110000)_{C_2^8}$.

b) $13 = (001101)_{C_2^6} = (00001101)_{C_2^8}$.

c) $-1 = (111111)_{C_2^6} = (11111111)_{C_2^8}$.

d) $-10 = (110110)_{C_2^6} = (11110110)_{C_2^8}$.

e) $16 = (010000)_{C_2^6} = (00010000)_{C_2^8}$.

f) $31 = (100001)_{C_2^6} = (11100001)_{C_2^8}$.

Basta con extender el bit mas significativo para realizar la conversión.

5. Dadas las siguientes secuencias de bits, indicar a qué números corresponden en sistema decimal utilizando la representación en complemento a dos:

a) $(00001101)_{C_2^8}$

b) $(01001101)_{C_2^8}$

c) $(11100001)_{C_2^8}$

d) $(11111001)_{C_2^8}$

e) $(11111111)_{C_2^8}$

f) $(00000000)_{C_2^8}$

Soluciones

a) $(00001101)_{C_2^8} = 2^0 + 2^2 + 2^3 = 1 + 4 + 8 = 13$.

b) $(01001101)_{C_2^8} = 2^0 + 2^2 + 2^3 + 2^6 = 1 + 4 + 8 + 64 = 77$.

c) $(11100001)_{C_2^8} = 2^0 + 2^5 + 2^6 - 2^7 = 1 + 32 + 64 - 128 = -31$.

d) $(11111001)_{C_2^8} = 2^0 + 2^3 + 2^4 + 2^5 + 2^6 - 2^7 = -7$.

e) $(11111111)_{C_2^8} = 2^0 + 2^1 + 2^2 + 2^3 + 2^4 + 2^5 + 2^6 - 2^7 = -1$.

f) $(00000000)_{C_2^8} = 0$.

6. Mostrar que $(13,25)_{10} = (1101,01)_2 = (15,2)_8 = (D,4)_{16}$.

   Solución

   - $(13,25)_{10} = (1101,01)_2$:
     - $0,25 \cdot 2 = \boxed{0},5 \Rightarrow a_{-1} = \boxed{0}$.
     - $0,5 \cdot 2 = \boxed{1} \Rightarrow a_{-2} = \boxed{1}$.
     - $13 = 2 \cdot 6 + \boxed{1} \Rightarrow a_0 = \boxed{1}$.
     - $6 = 2 \cdot 3 + \boxed{0} \Rightarrow a_1 = \boxed{0}$.
     - $3 = 2 \cdot 1 + \boxed{1} \Rightarrow a_2 = \boxed{1}$.
     - $1 = 2 \cdot 0 + \boxed{1} \Rightarrow a_3 = \boxed{1}$.
   - $(13,25)_{10} = (15,2)_8$:
     - $0,25 \cdot 8 = \boxed{2} \Rightarrow a_{-1} = \boxed{2}$.
     - $13 = 8 \cdot 1 + \boxed{5} \Rightarrow a_0 = \boxed{5}$.
     - $1 = 8 \cdot 0 + \boxed{1} \Rightarrow a_1 = \boxed{1}$.
   - $(13,25)_{10} = (D,4)_{16}$:
     - $0,25 \cdot 16 = \boxed{4} \Rightarrow a_{-1} = \boxed{4}$.
     - $13 = 16 \cdot 0 + 13 = \boxed{13} \Rightarrow a_0 = \boxed{D}$.

7. Rellenar la siguiente tabla:

Binario	Octal	Hexadecimal	Decimal
$1101100,110$			
	$362,23$		
		$A1,03$	
			$74,3$

   En cada fila encontrarán un valor numérico expresado en la base indicada en la casilla superior de la columna correspondiente. Completen el resto de las casillas con la representación correspondiente según la base indicada en la parte superior, de manera que se mantengan las equivalencias en cada fila. Asuman que los números son sin signo.

SOLUCIÓN

Binario	Octal	Hexadecimal	Decimal
$1101100,110$	$154,6$	$6C,C$	$108,75$
$11110010,010011$	$362,23$	$F2,4C$	$242,296875$
$10100001,00000011$	$241,006$	$A1,03$	$161,01171875$
$1001010,0\overline{1001}$	$112,2\overline{3146}$	$4A,4\overline{C}$	$74,3$

8. Determinar el formato hexadecimal que use el mínimo número de dígitos y que permita representar el número $(16,25)_{10}$ de manera exacta. ¿Cuál es el rango y la precisión del formato? Asumir números sin signo.

   Ayuda: Tener en cuenta que la precisión se puede interpretar como la diferencia entre un número y el siguiente número representable.

   SOLUCIÓN Observemos que $16,25_{10} = (10,4)_{16}$, luego nos bastará un formato hexadecimal de punto fijo con dos dígitos para la parte entera y uno para la parte fraccionaria.

   Este formato permite expresar numeros entre $0 \leq N \leq (FF,F)_{16} = (255,9375)_{10}$ y tiene una presición de $16^{-1} = 0,0625$.

9. Realicen cada una de las siguientes operaciones en complemento a dos, utilizando 8 bits para representar cada número.

   a) $10 - 3$
   b) $-39 + 92$
   c) $-19 - 7$
   d) $44 + 45$
   e) $104 + 45$
   f) $-75 + 59$
   g) $-103 - 69$
   h) $127 + 1$
   i) $-1 + 1$
   j) $-1 - 1$
   k) $-8 - 127$
   l) $127 + 127$

   Para cada operación, analicen si los resultados obtenidos son correctos o incorrectos, explicando las razones. Ubique los resultados en la recta real, incluyendo los valores mínimos y máximos para esta representación. Además, indicar el estado de las banderas *Carry* y *Overflow* luego de realizar cada una de las operaciones anteriores.

Soluciones

a) Observemos que $(10)_{10} = (00001010)_{C_2^8}$ y $(3)_{10} = (00000011)_2 = (11111101)_{C_2^8}$, luego:

$$\begin{array}{cccccccccc} & \mathbf{0} & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ + & \mathbf{1} & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ \hline 1 & \mathbf{0} & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{array}$$

- El resultado es correcto pues $(00000111)_{C_2^8} = (7)_{10} = 10 - 3$.
- La bandera *Carry Flag* se enciende y la *Overflow Flag* se **apaga**.

b) Observemos que $(-39)_{10} = (-00100111)_2 = (11011001)_{C_2^8}$ y $(92)_{10} = (01011100)_{C_2^8}$, luego:

$$\begin{array}{cccccccccc} & \mathbf{1} & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ + & \mathbf{0} & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ \hline 1 & \mathbf{0} & 0 & 1 & 1 & 0 & 1 & 0 & 1 \end{array}$$

- El resultado es correcto pues $(00110101)_{C_2^8} = (53)_{10} = -39 + 92$.
- La bandera *Carry Flag* se enciende y la *Overflow Flag* se **apaga**.

c) Observemos que $(-19)_{10} = (-00010011)_2 = (11101101)_{C_2^8}$ y $(-7)_{10} = (-00000111)_2 = (11111001)_{C_2^8}$, luego:

$$\begin{array}{cccccccccc} & \mathbf{1} & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ + & \mathbf{1} & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ \hline 1 & \mathbf{1} & 1 & 1 & 0 & 0 & 1 & 1 & 0 \end{array}$$

- El resultado es correcto pues $(11100110)_{C_2^8} = (-26)_{10} = -19 - 7$.
- La bandera *Carry Flag* se enciende y la *Overflow Flag* se **apaga**.

d) Observemos que $(44)_{10} = (00101100)_2$ y $(45)_{10} = (00101101)_2$, luego:

$$\begin{array}{cccccccccc} & \mathbf{0} & 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ + & \mathbf{0} & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ \hline 0 & \mathbf{0} & 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{array}$$

- El resultado es correcto pues $(01011001)_{C_2^8} = (89)_{10} = 44 + 45$.
- La bandera *Carry Flag* se apaga y la *Overflow Flag* se **apaga**.

e) Observemos que $(104)_{10} = (01101000)_{C_2^8}$ y $(45)_{10} = (00101101)_{C_2^8}$, luego:

$$\begin{array}{cccccccccc} & \mathbf{0} & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ + & \mathbf{0} & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ \hline 0 & \mathbf{1} & 0 & 0 & 1 & 0 & 1 & 0 & 1 \end{array}$$

- El resultado es correcto pues $(10010101)_2 = (-107)_{10} \neq (149)_{10} = 104 + 45$.
- La bandera *Carry Flag* se apaga y la *Overflow Flag* se ***enciende***.

f) Observemos que $(-75)_{10} = (-01001011)_2 = (10110101)_{C_2^8}$ y $(59)_{10} = (00111011)_{C_2^8}$, luego:

$$\begin{array}{cccccccc} & \mathbf{1} & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ + & \mathbf{0} & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ \hline 0 & \mathbf{1} & 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{array}$$

- El resultado es correcto pues $(11110000)_{C_2^8} = (-16)_{10} = -75 + 59$.
- La bandera *Carry Flag* se apaga y la *Overflow Flag* se **apaga**.

g) Observemos que $(-103)_{10} = (-01100111)_2 = (10011001)_{C_2^8}$ y $(-69)_{10} = (-01000101)_2 = (10111011)_{C_2^8}$, luego:

$$\begin{array}{cccccccc} & \mathbf{1} & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ + & \mathbf{1} & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ \hline 1 & \mathbf{0} & 1 & 0 & 1 & 0 & 1 & 0 & 0 \end{array}$$

- El resultado es incorrecto pues $(01010100)_{C_2^8} = (84)_{10} \neq -172 = -19 - 7$.
- La bandera *Carry Flag* se enciende y la *Overflow Flag* se **enciende**.

h) Observemos que $(127)_{10} = (01111111)_{C_2^8}$ y $(1)_{10} = (00000001)_{C_2^8}$, luego:

$$\begin{array}{cccccccc} & \mathbf{0} & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ + & \mathbf{0} & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \hline 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

- El resultado es incorrecto pues $(10000000)_{C_2^8} = (-1)_{10} \neq 128 = 127 + 1$.
- La bandera *Carry Flag* se apaga y la *Overflow Flag* se **enciende**.

i) Observemos que $(-1)_{10} = (11111111)_{C_2^8}$ y $(1)_{10} = (00000001)_{C_2^8}$, luego:

$$\begin{array}{cccccccc} & \mathbf{1} & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ + & \mathbf{0} & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \hline 1 & \mathbf{0} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

- El resultado es correcto pues $(00000000)_{C_2^8} = (0)_{10} = -1 + 1$.
- La bandera *Carry Flag* se enciende y la *Overflow Flag* se **apaga**.

j) Observemos que $(-1)_{10} = (11111111)_{C_2^8}$, luego:

$$\begin{array}{cccccccccc} & \mathbf{1} & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ + & \mathbf{1} & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \hline 1 & \mathbf{1} & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{array}$$

- El resultado es correcto pues $(11111110)_{C_2^8} = (-2)_{10} = -1 - 1$.
- La bandera *Carry Flag* se enciende y la *Overflow Flag* se **apaga**.

k) COMPLETAR.

l) COMPLETAR.

10. Dados $A = 200$, $B = 300$, $C = 500$ y $D = 400$, calcular $S = A \cdot B \cdot C \cdot D$, interpretando los datos como enteros con signo. Imprimir el resultado por pantalla haciendo un programa en lenguaje C con datos tipo `int` y analizar el resultado. Repetir para datos tipo `long int`.

    Ayuda: Para poder imprimir un `long int` se debe usar el especificador `%ld`.

    Solución

    - $S = (10110010110100000101111000000000000)_2 = 2^{11} + 2^{12} + 2^{13} + 2^{14} + 2^{16} + 2^{22} + 2^{24} + 2^{25} + 2^{27} + 2^{30} + 2^{31} + 2^{33} = 12000000000$.
    - $T = (11001011010000010111100000000000)_{C_2^8} = 2^{11} + 2^{12} + 2^{13} + 2^{14} + 2^{16} + 2^{22} + 2^{24} + 2^{25} + 2^{27} + 2^{30} - 2^{31} = -884901888$.

```c
#include <stdio.h>

int main() {
    int a = 200, b = 300, c = 500, d = 400;

    int s = a * b * c * d;
    printf("s = %d\n", s);

    long int t = (long int) a * b * c * d;
    printf("t = %ld\n", t);
}
```

11. Ejecutar el siguiente programa y analizar lo que imprime por pantalla. Conviene usar GDB para observar el contenido de las variables.

```c
#include <stdio.h>

int main() {
    char a = 127;
    printf("%hhd %hhu\n", a, a);

    a = ++a;
    printf("%hhd %hhu\n", a, a);

    unsigned char b = 128;
    printf("%hhd %hhu\n", b, b);

    b = ++b;
    printf("%hhd %hhu\n", b, b);
}
```

Solución

```
127 127
-128 128
-128 128
-127 129
```

- Puesto que $127 = (1111111)_2 = (01111111)_{C_2^8}$, el resultado impreso no presenta ninguna particularidad.
- Ahora bien $127 + 1 = (10000000)_2 = 128$, pero $(10000000)_{C_2^8} = -128$.
- A partir del punto anterior, es análogo interpretar este.
- Finalmente $128 + 1 = (10000001)_2 = 129$, pero $(10000001)_{C_2^8} = -127$.

12. Dados los siguientes números $A = 0xc4$ y $B = 0x4e$:

    a) Realizar la suma $C = A + B$ en hexadecimal.

    b) Escriba el valor decimal de C, interpretando el resultado como `signed char` y luego como `unsigned char`.

    c) Indicar el rango y la cantidad de números representables en ambos casos.

    d) Indicar el valor de las banderas *CF*, *OF* y *SF* luego de realizarse la operación suma.

SOLUCIÓN

a)

$$\begin{array}{cccc} & & c & 4 \\ + & & 4 & e \\ \hline & 1 & 1 & 2 \end{array}$$

b) Puesto que $(112)_{16} = 274 = (100010010)_2$, luego $(00010010)_2 = (00010010)_{C_2^8} = 18$.

c)

$$\begin{array}{cccccccccc} & & \mathbf{1} & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ + & & \mathbf{0} & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ \hline & 1 & \mathbf{0} & 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{array}$$

La bandera *Carry Flag* se enciende y la *Overflow Flag* se **apaga**.

13. Supongamos que en un servidor de almacenamiento los tamaños de los archivos se gestionan usando el sistema octal. La capacidad total del servidor es de 010000 unidades de almacenamiento. Actualmente, el servidor contiene los siguientes archivos:

Archivo	Unidades
A	0345
B	0672
C	01250
D	0507
E	0710

a) Convertir los tamalos de los archivos a su equivalente en decimal.

b) Convertir la capacidad total del servidor a su equivalente en decimal.

c) Calcular el espacio total usado en el servidor en octal y decimal.

d) Calcular el espacio libre en el servidor en octal y decimal.

e) Si se desea almacenar un nuevo archivo de tamaño 0500 unidades, determinar si hay suficiente espacio disponible. Si el archivo se puede almacenar, actualizar la lista de archivos y recalcular el espacio usado y libre en el servidor.

SOLUCIÓN COMPLETAR.

14. Una plataforma de criptomonedas gestiona las transacciones de sus usuarios utilizando identificadores de transacción en sistema hexadecimal. Cada transacción tiene un identificador único y un valor asociado en una criptomoneda llamada CryptoCoin (CC). Actualmente, la plataforma tiene las siguientes transacciones registradas:

Transacción	Identificador	Valor
A	$0x1a2f$	$0x3b$ CC
B	$0x2c4d$	$0x4e$ CC
C	$0x3d5a$	$0x25$ CC
D	$0x4e7c$	$0x6a$ CC
E	$0x5f8e$	$0x72$ CC

   a) Convierte los identificadores y valores de las transacciones a su equivalente en decimal.

   b) Calcula el valor total de todas las transacciones en CryptoCoin en decimal y hexadecimal.

   c) La plataforma añade dos nuevas transacciones con los siguientes detalles:

   - Transacción F: Identificador $0x6a7b$, valor $0x3c$ CC.
   - Transacción G: Identificador $0x7b9d$, valor $0x4a$ CC.

   Actualiza la lista de transacciones y recalcula el valor total de todas las transacciones.

SOLUCIÓN COMPLETAR.

15. A continuación se presentan ciertos números enteros expresados en binario utilizando 32 bits y a su derecha, expresiones en lenguaje C incompletas. Complete estas expresiones de forma que la igualdad sea cierta. Utilice operadores de bits, operadores enteros y constantes de enteros literales según considere necesario.

   a) `10000000 00000000 00000000 00000000 == ... << ...`

   b) `10000000 00000000 10000000 00000000 == (1 << ...) | (1 << ...)`

   c) `11111111 11111111 11111111 00000000 == -1 & ...`

   d) `10101010 00000000 00000000 10101010 == 0xAA ... (0xAA << ...)`

   e) `00000000 00000000 00000101 00000000 == 5 ... 8`

   f) `11111111 11111111 11111110 11111111 == -1 & (... (1 << 8))`

   g) `11111111 11111111 11111111 11111111 == 0 ... 1`

   h) `00000000 00000000 00000000 00000000 == 0x80000000 + ............`

Soluciones

a) `10000000 00000000 00000000 00000000 == 1 << 31`

b) `10000000 00000000 10000000 00000000 == (1 << 31) | (1 << 15)`

c) `11111111 11111111 11111111 00000000 == -1 & (-1 << 8)`

d) `10101010 00000000 00000000 10101010 == 0xAA | (0xAA << 24)`

e) `00000000 00000000 00000101 00000000 == 5 << 8`

f) `11111111 11111111 11111110 11111111 == -1 & (~ (1 << 8))`

g) `11111111 11111111 11111111 11111111 == 0 - 1`

h) `00000000 00000000 00000000 00000000 == 0x80000000 + 0x80000000`

16. Implemente una función `int is_one(int n, int b);` que indique si el bit `b` del entero `n` es 1 o 0.

    Ayuda: Pensar en las propiedades de los operadores de bits.

    Solución

    ```
    int is_one(int n, int b) {
        return n & (1 << b);
    }
    ```

17. Implemente una función `void printbin(int n);` que tome un entero de 32 bits y lo imprima en binario. Utilizar esta función para mostrar en binario los números del ejercicio 2.

    Ayuda: Se puede utilizar la función realizada en el ejercicio anterior.

    Solución

    ```
    void printbin(int n) {
        for (int i = 31; i >= 0; i--)
            printf("%c", is_one(n, i) ? '1' : '0');
    }
    ```

18. Implemente una función en lenguaje C que tome tres parámetros *a*, *b* y *c* y que rote los valores de las variables de manera que al finalizar la función el valor de *a* se encuentre en *b*, el valor de *b* en *c* y el de *c* en *a*. Evitar utilizar variables auxiliares.

    Ayuda: Tener en cuenta las propiedades del operador `XOR`. Se puede pensar primero en intercambiar dos variables.

Solución

```
void swap(unsigned long* a, unsigned long* b, unsigned long* c) {
    *a = *a ^ *b ^ *c;
    *b = *a ^ *b ^ *c;
    *c = *a ^ *b ^ *c;
    *a = *a ^ *b ^ *c;
}
```

19. Escriba en lenguaje C un programa que tome la entrada estándar, la codifique e imprima el resultado en salida estándar. La codificación deberá ser hecha carácter a carácter utilizando el operador `XOR` y un código que se pase al programa como argumento de línea de comando.

    El código adicional para el operador `XOR` también se debe pasar como argumento de línea de comandos al programa. Es decir, suponiendo que el ejecutable se llame `prog`, la línea de comando para ejecutar el programa tendría el formato: `./prog <código> <cadena a codificar>`.

    Por ejemplo, se podría hacer `./prog 12 Mensaje` para codificar la cadena «Mensaje» con el código 12. Pruebe el programa codificando con diferentes códigos, por ejemplo, utilizando el código -98.

    ¿Qué modificaciones se tendrían que hacer al programa para que decodifique? ¿Se gana algo codificando más de una vez?

    Solución

```
#include <stdio.h>  // putchar
#include <stdlib.h> // atoi

int main(int argc, char** argv) {
    for (int i = 0; argv[2][i]; i++)
        putchar(atoi(argv[1]) ^ argv[2][i]);
}
```

    No es necesario modificar el programa para decodificar pues:

$$(m \oplus a) \oplus a = m \oplus (a \oplus a) = m \oplus 0 = m$$

    No se gana nada codificando varias veces pues:

$$(m \oplus a) \oplus b = m \oplus (a \oplus b)$$

20. *Algoritmo del campesino ruso*: La multiplicación de enteros positivos puede implementarse con sumas, el operador `AND` y desplazamientos de bits usando las siguientes identidades:

$$a \cdot b \begin{cases} 0 & \text{si } b = 0 \\ a & \text{si } b = 1 \\ 2a \cdot k & \text{si } b = 2k \\ 2a \cdot k + a & \text{si } b = 2k + 1 \end{cases}$$

Úselas para implementar una función en lenguaje C `unsigned mult(unsigned a, unsigned b);`.

Solución

```c
unsigned mult(unsigned a, unsigned b) {
    if (!b) return 0;                                  // b == 0
    if (b == 1) return a;                              // b == 1
    if (!(b & 1)) return mult((a << 1), (b >> 1)); // b par
    return mult((a << 1), (b >> 1)) + a;               // b impar
}
```

21. (*Opcional*) Muchas arquitecturas de CPU restringen los enteros a un máximo de 64 bits. ¿Qué sucede si ese rango no nos alcanza? Una solución es extender el rango utilizando más de un entero (en este caso enteros de 16 bits) para representar un valor. Así podemos pensar que:

```c
typedef struct {
    unsigned short n[16];
} nro;
```

representa el valor:

$$\begin{aligned} N = \ & \texttt{nro.n[0]} & + \\ & \texttt{nro.n[1] * (2\^{}sizeof(short) * 8)} & + \\ & \texttt{nro.n[1] * (2\^{}(2 * sizeof(short) * 8))} & + \\ & & \vdots \\ & \texttt{nro.n[15] * (2\^{}(15 * sizeof(short) * 8))} \end{aligned}$$

Podemos pensar en la estructura `nro` como un entero de 256 bits. Lamentablemente la arquitectura no soporta operaciones entre valores de este tipo, por lo cual debemos realizarlas en software.

a) Implemente funciones que comparen con 0 y con 1 y determinen paridad para valores de este tipo.

b) Realice funciones que corran a izquierda y derecha los valores del tipo `nro`.

c) Implemente la suma de valores del tipo `nro`.

Nota: en el repositorio Subversion de la materia hay una función para imprimir valores de este tipo. Esta función utiliza la biblioteca GMP (*GNU Multiple Precision Arithmetic Library*), por lo cual deberá compilar el código agregando la opción `-lgmp`. Puede encontrar la función en el archivo `código/enteros/grandes/gmp1.c`: https://svn.dcc.fceia.unr.edu.ar/svn/lcc/R-222/Public/código/enteros_grandes/gmp1.c.

Soluciones

a)

```
int es_cero(nro n) {
    for (int i = 0; i < 16; i++)
        if (n.n[i]) return 0;

    return 1;
}
int es_uno(nro n) {
    for (int i = 1; i < 16; i++)
        if (n.n[i]) return 0;

    return n.n[0] == 1;
}
int es_par(nro n) {
    return (!(n.n[0] & 1));
}
```

b) COMPLETAR.

c)

```
nro suc(nro n) {
    nro s = n;

    for (int i = 0; i < 16; i++) {
        s.n[i] += 1;
        if ((int) n.n[i] + 1 == s.n[i]) // No hay acarreo
            return s;
    }

    return s;
}

nro pred(nro n) {
    nro p = n;

    for (int i = 0; i < 16; i++) {
        p.n[i] -= 1;
```

```
            if ((int) n.n[i] - 1 == p.n[i]) return p;
        }

        return p;
    }

    nro add(nro a, nro b) {
        if (es_cero(b)) return a;
        return add(suc(a), pred(b));
    }
```

22. (*Opcional*) Implemente el algoritmo del campesino ruso para los números anteriores.

SOLUCIÓN

```
nro mult(nro a, nro b) {
        if (es_cero(b)) return b;
        if (es_uno(b)) return a;
        if (es_par(b)) return mult(izq(a, 1), der(b, 1));
        return add(mult(izq(a), der(b)), a);
}
```