Operaciones de bajo nivel

Sofía Beatriz Pérez
Daniel Agustín Rosso

sperez@iua.edu.ar
drosso@iua.edu.ar

Centro Regional Univesitario Córdoba
Instituto Univeristario Areonáutico

Clase número 11 - Ciclo lectivo 2023

Agenda

Disclaimer

Los siguientes slides tienen el objetivo de dar soporte al dictado de la asignatura. De ninguna manera pueden sustituir los apuntes tomados en clases y/o la asistencia a las mismas.
Es importante mencionar que todos este material se encuentra en un proceso de mejora continua.
Si encuentra bugs, errores de ortografía o redacción, por favor repórtelo a sperez@iua.edu.ar y/o drosso@iua.edu.ar. También puede abrir issues en el repositorio de este link: ▸ infoI_IUA_GitLab

Sistemas de numeración I

Definición de Decimal

El sistema decimal es un sistema de numeración que utiliza diez símbolos diferentes, del 0 al 9, para representar cualquier número. Cada posición en un número decimal tiene un valor diez veces mayor que la posición anterior.

Definición de Binario

El sistema binario es un sistema de numeración que utiliza solo dos símbolos, 0 y 1, para representar cualquier número. Es ampliamente utilizado en computadoras y sistemas digitales debido a su simplicidad en la implementación electrónica.

Sistemas de numeración II

Definición de Octal

El sistema octal es un sistema de numeración que utiliza ocho símbolos, del 0 al 7, para representar cualquier número. Cada dígito en el sistema octal representa tres bits en el sistema binario, lo que lo hace útil en computación.

Definición de Hexadecimal

El sistema hexadecimal es un sistema de numeración que utiliza dieciséis símbolos, del 0 al 9 y las letras A a F, para representar cualquier número. Es comúnmente utilizado en informática para representar valores binarios de manera más compacta y legible.

Sistemas de numeración III

Conversión de Decimal a Binario

Para convertir un número decimal a binario, dividimos el número entre 2 y tomamos los residuos como dígitos binarios, de abajo hacia arriba.

Ejemplo: Convertir 13_{10} a binario.

$$13_{10} = 1101_2$$

Sistemas de numeración IV

Conversión de Binario a Decimal

Para convertir un número binario a decimal, multiplicamos cada dígito binario por la potencia de 2 correspondiente y sumamos los resultados.

Ejemplo: Convertir 1101_2 a decimal.

$$1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 13_{10}$$

Sistemas de numeración V

Extensión a Números con Parte Fraccional

Para números con parte fraccional, extendemos el mismo proceso. Por ejemplo:

$$0.75_{10} = ?_2$$

$$0.75_{10} \times 2 = 1.50_{10} = 1_2$$

$$0.50_{10} \times 2 = 1.00_{10} = 1_2$$

$$0.00_{10} \times 2 = 0_{10}$$

Por lo tanto, $0.75_{10} = 0.11_2$.

Sistemas de numeración VI

Conversión de Binario a Hexadecimal

Para convertir un número binario a hexadecimal, agrupamos los dígitos binarios de cuatro en cuatro, desde el punto decimal hacia ambos lados, y los convertimos a dígitos hexadecimales.

Ejemplo: Convertir 1101.11_2 a hexadecimal.

$$1101.11_2 = D.C_{16}$$

Sistemas de numeración VII

Decimal	Binario	Octal	Hexadecimal
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A

Complemento a dos I

- Método utilizado para representar números enteros tanto positivos como negativos en sistemas binarios.
- Importante en aritmética computacional

Example

- 0110 representa +6 en binario.
- 1010 representa -6 en binario usando complemento a dos.

Complemento a dos II

Números Binarios Signados

- Bit más significativo (MSB) como el bit de signo (0 para positivo, 1 para negativo).
- Resto de los bits representan el valor absoluto del número en binario.

Obteniendo el Complemento a Dos

- Invertir todos los bits del número.
- Sumar 1 al número invertido.

Complemento a dos III

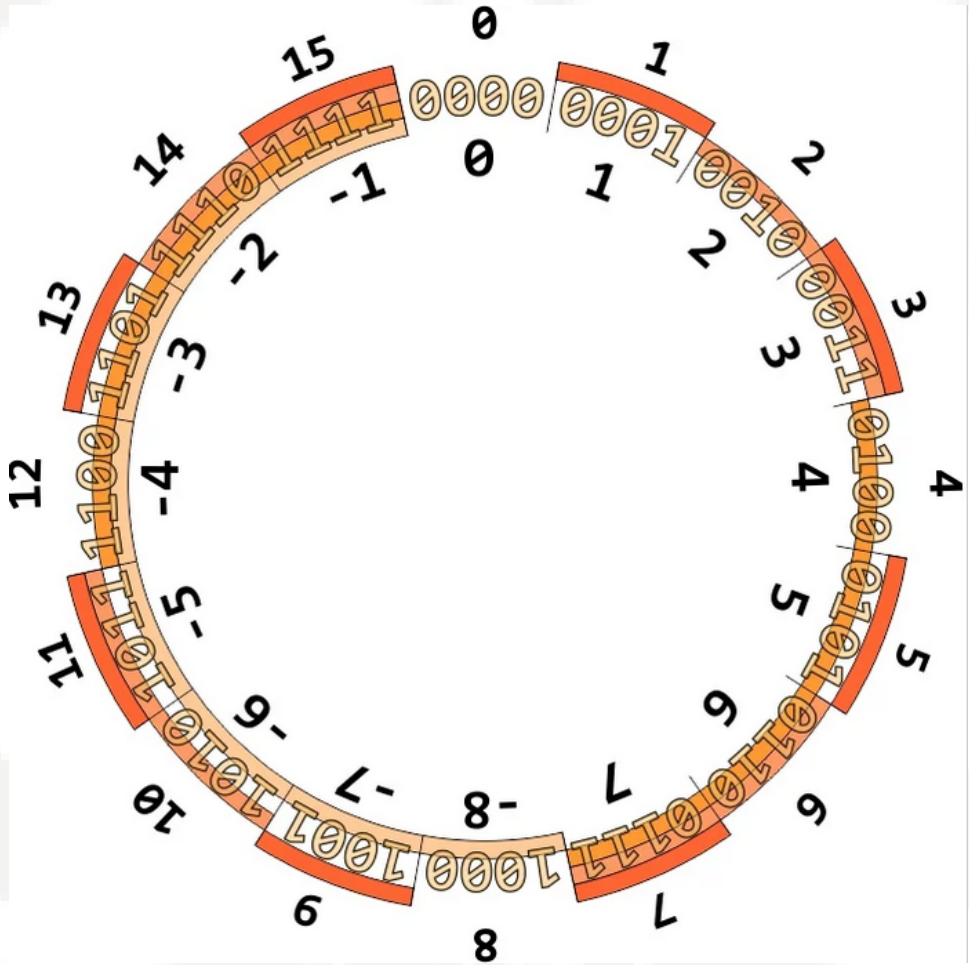
Example

- Para representar -6 en 4 bits:
  6 en binario: 0110
  Invertir bits: 1001
  Sumar 1: 1010
- El resultado 1010 representa -6 en complemento a dos con 4 bits.

Desbordamiento y rango:

- Desbordamiento:
  - Ocurre cuando el resultado de una operación supera el rango representable en el número de bits disponibles.
- Rango:
  - Para n bits, el rango es de $-2^{(n-1)}$ a $2^{(n-1)} - 1$.

Complemento a dos IV



Campos de bits I

Definición

C y C++ traen una característica intrínseca que permite conocer y operar con cada bit de un tipo de dato en particular.

Algunas aplicaciones:

- Cuando el almacenamiento es limitado, se pueden almacenar múltiples valores booleanos en un byte
- Muchos dispositivos transmiten la información codificada en uno o más bits de un byte
- Algoritmos de encriptación

Campos de bits II

Definición general:

```
1  struct nombre
2  {
3      tipo nombre1 : lenght;
4      tipo nombre2 : lenght;
5      tipo nombre3 : lenght;
6      tipo nombre4 : lenght;
7
8  }
```

- Tipo: indica el tipo del campo de bit
- Lenght: indica el ancho del campo (1, 2, 3, n bits)

Los campos que tengan un solo bit de ancho, deben ser **obligatoriamente** declarados como unsigned.

Campos de bits III

```c
#include <stdio.h>


struct byte_bit{
    unsigned char bit_0 : 1;
    unsigned char bit_1 : 1;
    unsigned char bit_2 : 1;
    unsigned char bit_3 : 1;
    unsigned char bit_4 : 1;
    unsigned char bit_5 : 1;
    unsigned char bit_6 : 1;
    unsigned char bit_7 : 1;
};

typedef struct byte_bit byte_bit_t;
```

Campos de bits IV

```
16
17  int main()
18  {
19      byte_bit_t example;
20      example.bit_0 = 1;
21      example.bit_1 = 0;
22      example.bit_2 = 1;
23
24      if(example.bit_0)
25          printf("0");
26      else
27          printf("1");
28
29  }
```

Uniones I

Definición

Es un tipo de dato derivado cuyos miembros **comparten** el mismo espacio de memoria. En general las uniones estan compuestas por dos o más miembros. Todos ellos están físicamente almacenados en **la misma memoria**.

```
1  union nombre
2  {
3      tipo nombre1;
4      tipo nombre2;
5      tipo nombre3;
6      tipo nombre4;
7  }
```

Uniones II

La unión ocupará en memoria lo necesario para almacenar al dato más grande.

```
1  union datos
2  {
3      int x;
4      float y;
5  }
```

Uniones III

```c
#include <stdio.h>


union char_int
{
    int ascii;
    char val;
};

typedef union char_int char_int;



```

Uniones IV

```
16  int main()
17  {
18    char_int example;
19    example.val='a';
20    printf("ascii vale %d\n",example.ascii);
21    printf("val vale %c\n",example.val);
22    return(0);
23  }
```

Uniones y campo de bits I

Una interesante aplicación surge cuando se combinan las uniones junto a los campos de bits.

```
1  #include<stdio.h>
2
3  struct ocho_bits
4  {
5    unsigned bit7:1;
6    unsigned bit6:1;
7    unsigned bit5:1;
8    unsigned bit4:1;
9    unsigned bit3:1;
10   unsigned bit2:1;
11   unsigned bit1:1;
12   unsigned bit0:1;
13 };
```

Uniones y campo de bits II

```
14
15
16
17
18
19
20  union char_and_bits
21  {
22    unsigned char     data_char;
23    struct ocho_bits  bits_union;
24  };
25
26
27
28
```

Uniones y campo de bits III

```
29  int main(void)
30  {
31      union char_and_bits test;
32      test.data_char = 'a';
33      printf("%d",test.bits_union.bit0);
34      printf("%d",test.bits_union.bit1);
35      printf("%d",test.bits_union.bit2);
36      printf("%d",test.bits_union.bit3);
37      printf("%d",test.bits_union.bit4);
38      printf("%d",test.bits_union.bit5);
39      printf("%d",test.bits_union.bit6);
40      printf("%d",test.bits_union.bit7);
41      return(0);
42  }
```

Enumeraciones I

Son un tipo de dato definido por el usuario. Las enumeraciones son generadas por la palabra reservada **enum** y consisten en un conjunto de constantes enteras representadas por identificadores.

Las *constantes de enumeración* son constantes simbólicas cuyos valores pueden ser definidos automáticamente.

```
1 typedef enum
2 {   ENE,
3     FEB,
4     MAR
5 }meses_t;
```

Los valores dentro de una enumeración se inicializan con cero a menos de que se defina otra manera y se incrementan en uno.

Enumeraciones II

```
1  typedef enum
2  {    ENE = 1,
3       FEB,
4       MAR,
5       ABR
6  }meses_t;
```

Dado que el primer valor de la enumeración anterior se define explícitamente en uno, los valores subsiguientes se incrementan en uno dando como resultado valores desde el 1 hasta el 4.
Los identificadores de enumeración deben ser únicos, pero varios miembros pueden tener el mismo valor entero.

Enumeraciones III

```
1  #include <stdio.h>
2  typedef enum
3  {    LUN = 1,
4       MAR,
5       MIE,
6       JUE,
7       VIE,
8       SAB,
9       DOM
10 }dias_t;
11
12
13
14
15
```

Enumeraciones IV

```
16
17  int main()
18  {
19      dias_t hoy;
20      hoy = LUN;
21      printf("%d",hoy);
22      hoy = MAR;
23      printf("%d",hoy);
24      hoy = DICIEMBRE; /*COMPILATION ERROR*/
25      return(0);
26  }
```

Operaciones a nivel de bits

C y C++ operan con entidades de datos que se almacenan como uno o mas bytes, según la naturaleza de la variable.
Afortunadamente para los desarrolladores de bajo nivel, también proveen la posibilidad de manipular bits de forma individual.

Operador	Descripción
&	AND bit por bit
|	OR inclusivo bit por bit
^	OR exclusivo bit por bit
~	Complemento a uno bit por bit
<<	Desplazamiento a la izquierda
>>	Desplazamiento a la derecha

El operador AND

La operación AND realiza una comparación bit por bit. El resultado de dicha comparación sólo es 1 cuando ambos bits comparados son 1. De lo contrario, la operación resultará cero.

```
1 #include <stdio.h>
2 int main()
3 {
4       int op1=10; //0b000001010
5       int op2=3;  //0b000000011
6       int op3= op1&op2;
7       printf("op1 & op2 es= %d",op3);
8       return(0);
9 }
```

El operador OR inclusivo

El operador or inclusivo | ejecuta una comparación bit por bit de sus dos operandos. El resultado de la comparación es 1 si cualquier bit comparado es 1, de lo contrario es cero.

```
1 #include <stdio.h>
2 int main()
3 {
4     int op1=10; //0b000001010
5     int op2= op1|op2;
6     printf("op1 | op2 es= %d",op3);
7     return(0);
8 }
```

El operador OR exclusivo (XOR)

El resultado de una operación XOR es 1 si uno y sólo uno de los bits que se estan comparando es un uno. De lo contrario, el resultado es cero.

```
1 #include <stdio.h>
2 int main()
3 {
4       int op1=10; //0b000001010
5       int op2=3;  //0b000000011
6       int op3= op1^op2;
7       printf("op1 ^ op2 es= %d",op3);
8       return(0);
9 }
```

El operador de complemento

Cambia cada bit 1 de su operando a 0 y cada bit 0 a 1. Este operador es particularmente útil cuando se desea forzar cualquier bit de un operando a cero.

```c
#include <stdio.h>
#include <stdint.h>
int main()
{
    uint8_t op1=10; //0b00001010
    uint8_t op2= ~op1;
    printf("op2=~op1 %d",op2);
    return(0);
}
```

¿Por qué definimos como uint8_t?

▸ Stdint

El operador de desplazamiento a la izquierda

El operador << causa que los bits en un operando sean desplazados a la izquierda por una cantidad de terminada. Para los números enteros sin signo cada desplazamiento a la izquierda corresponde a una multiplicación por dos.

```
1 #include <stdio.h>
2 #include <stdint.h>
3 int main()
4 {
5     uint8_t op1=10; //0b00001010
6     uint8_t op2 =  op1<<3;
7     printf("op2 =  op1<<3 %d",op2);
8     return(0);
9 }
```

El operador de desplazamiento a la derecha

El operador >> causa que los bits en un operando sean desplazados a la derecha por una cantidad de terminada. Los bits son desplazados mas allá del final y son descartados. Para números sin signo, el bit de la extrema izquierda no se utiliza como bit de signo, por lo cual los bit vacantes de la izquierda son completados con ceros.

```
1  #include <stdio.h>
2  #include <stdint.h>
3  int main()
4  {
5      uint8_t op1=10; //0b00001010
6      uint8_t op2 =  op1>>3;
7      printf("op2 =  op1>>3 %d",op2);
8      return(0);
9  }
```

Gestión de Máscaras I

Uso de Máscaras

- Creación de máscaras con bits específicos en 1 o 0.
- `1 << n` crea una máscara con el n-ésimo bit en 1.
- `~(1 << n)` crea una máscara con el n-ésimo bit en 0.

Example

Ejemplo `máscara = (1 << 3) | (1 << 5);`
Crea una máscara con el tercer y quinto bits en 1.

Gestión de Máscaras II

Conteo de Bits en un Número

```
int contarBits(int n) {
    int count = 0;
    while (n) {
        count++;
        n &= (n - 1);
    }
    return count;
}
```

Example

`contarBits(13);`

Devuelve 3, ya que el número binario 13 tiene tres bits en 1.

Ejemplo I

Implementar un algoritmo que permita encriptar y desencriptar un mensaje utilizando la operación XOR.

- Encriptación:
  - Se toma el mensaje original y la clave
  - Se realiza una operación XOR bit a bit entre el mensaje y la clave
  - El resultado de esta operación es el mensaje encriptado
- Desencriptación:
  - Se toma el mensaje encriptado y la misma clave que se usó para encriptar
  - Se realiza una operación XOR bit a bit entre el mensaje encriptado y la clave
  - El resultado de esta operación es el mensaje original

Ejemplo II

- Criterios de diseño:
  - El mensaje y la llave tienen que estar almacenados en la misma estructura
  - Debe haber una función para encriptar y desencriptar
  - Se debe imprimir el mensaje original, luego el encriptado y finalmente el desencriptado.

▸ Link a la solución propuesta

¡Muchas gracias!

Consultas:

sperez@iua.edu.ar
drosso@iua.edu.ar