El sistema de tipos de C

- Enteros sin signo
- Enteros con signo
- Representación de enteros en base 2
- Representación de enteros negativos en complemento de 2
- Números reales y su representación
- Representación de caracteres en ASCII

Tipos de datos en C

- El sistema de tipos de un lenguaje incluye:
  - Tipos de datos primitivos
  - Expresiones y operadores
  - Reglas de inferencia para el tipo de una expresión
  - Mecanismos para definir nuevos tipos

Tipos de datos en C

- El sistema de tipos de un lenguaje incluye:
  - Tipos de datos primitivos
    - Enteros, Reales, Punteros.

Tipos de datos en C

- El sistema de tipos de un lenguaje incluye:
  - Tipos de datos primitivos
    - Enteros, Reales, Punteros.
  - Expresiones y operadores
    - a + b * c ⇔ a + (b * c) ; por precedencia
    - a – b + c ⇔ (a – b) + c ; asociatividad de izquierda a derecha

Tipos de datos en C

- El sistema de tipos de un lenguaje incluye:
  - Tipos de datos primitivos
    - Enteros, Reales, Punteros.
  - Expresiones y operadores
    - a + b * c ⇔ a + (b * c) ; por precedencia
    - a – b + c ⇔ (a – b) + c ; asociatividad de izquierda a derecha
  - Reglas de inferencia para el tipo de una expresión
    - int a = 1; int b = 2; double c = a/b; (c = 0 por ser una división de enteros)

Tipos de datos en C

- El sistema de tipos de un lenguaje incluye:
  - Tipos de datos primitivos
    - Enteros, Reales, Punteros.
  - Expresiones y operadores
    - a + b * c ⇔ a + (b * c) ; por precedencia
    - a – b + c ⇔ (a – b) + c ; asociatividad de izquierda a derecha
  - Reglas de inferencia para el tipo de una expresión
    - int a = 1; int b = 2; double c = a/b; (c = 0 por ser una división de enteros)
    - Para tener división de reales con enteros:
      double c = (double) a / b ; (a pasa a ser tipo double, b tipo entero, c será división real)

Tipos de datos en C

- El sistema de tipos de un lenguaje incluye:
  - Tipos de datos primitivos
    - Enteros, Reales, Punteros.
  - Expresiones y operadores
    - a + b * c ⇔ a + (b * c) ; por precedencia
    - a – b + c ⇔ (a – b) + c ; asociatividad de izquierda a derecha
  - Reglas de inferencia para el tipo de una expresión
    - int a = 1; int b = 2; double c = a/b; (c = 0 por ser una división de enteros)
    - Para tener división de reales con enteros:

      double c = (double) a / b ; (a pasa a ser tipo double, b tipo entero, c será división real)
  - Mecanismos para definir nuevos tipos

Tipos de datos en C

- Tipos primitivos en C:

Entero con signo	Entero sin signo	Abreviación	Tamaño
char			1 byte = 8 bits
short int			2 bytes
int			4 bytes
long int			8 bytes
long long int			8 bytes

Tipos de datos en C

- Tipos primitivos en C:

Entero con signo	Entero sin signo	Abreviación	Tamaño
char		char	1 byte = 8 bits
short int		short	2 bytes
int		int	4 bytes
long int		long	8 bytes
long long int		long long	8 bytes

Tipos de datos en C

- Tipos primitivos en C:

Entero con signo	Entero sin signo	Abreviación	Tamaño
char	**unsigned** char	char	1 byte = 8 bits
short int	**unsigned** short	short	2 bytes
int	**unsigned** int	int	4 bytes
long int	**unsigned** long	long	8 bytes
long long int	**unsigned** long long	long long	8 bytes

Tipos de datos en C

- Tipos primitivos en C:

Entero con signo	Entero sin signo	Abreviación	Tamaño
char	**unsigned** char	char	1 byte = 8 bits
short int	**unsigned** short	short	2 bytes
int	**unsigned** int	int	4 bytes
long int	**unsigned** long	long	8 bytes
long long int	**unsigned** long long	long long	8 bytes

Un bit más para representaciones pero se pierde el signo (solo se pueden representar números positivos)

Tipos de datos en C

- Tipos primitivos en C:

Entero con signo	Entero sin signo	Abreviación	Tamaño
char	**unsigned** char	char	1 byte = 8 bits
short int	**unsigned** short	short	2 bytes
int	**unsigned** int	int	4 bytes
long int	**unsigned** long	long	8 bytes
long long int	**unsigned** long long	long long	8 bytes

Un bit más para representaciones pero se pierde el signo (solo se pueden representar números positivos)

```
unsigned int a = 140;
unsigned int b = 1;
unsigned int c = b – a;
```

c != - 139

Tipos de datos en C

- Tipos primitivos en C:

Entero con signo	Entero sin signo	Abreviación	Tamaño
char	unsigned char	char	1 byte = 8 bits
short int	unsigned short	short	2 bytes
int	unsigned int	int	4 bytes
long int	unsigned long	long	8 bytes
long long int	unsigned long long	long long	8 bytes

  - **Reales**:
    - float: 4 bytes
    - double: 8 bytes

Tipos de datos en C

- Tipos primitivos en C:

Entero con signo	Entero sin signo	Abreviación	Tamaño
char	unsigned char	char	1 byte = 8 bits
short int	unsigned short	short	2 bytes
int	unsigned int	int	4 bytes
long int	unsigned long	long	8 bytes
long long int	unsigned long long	long long	8 bytes

  - **Reales**:
    - float: 4 bytes
    - double: 8 bytes
  - **Punteros:** Son enteros pero corresponden a espacios de memoria, no a un entero en particular

Tipos de datos en C

- Tipos primitivos en C:

Entero con signo	Entero sin signo	Abreviación	Tamaño
char	unsigned char	char	1 byte = 8 bits
short int	unsigned short	short	2 bytes
int	unsigned int	int	4 bytes
long int	unsigned long	long	8 bytes
long long int	unsigned long long	long long	8 bytes

- **Reales**:
    - float: 4 bytes
    - double: 8 bytes
- **Punteros:** Son enteros pero corresponden a espacios de memoria, no a un entero en particular

Ordenamiento de tipos:

char < short <= int <= long <= long long < float < double

Tipos de datos en C

- C no define un tipo especial para los valores de verdad o para strings (Java sí define los tipos boolean y String)

Tipos de datos en C

- C no define un tipo especial para los valores de verdad o para strings (Java sí define los tipos boolean y String)
  - En C, 0 es falso y cualquier valor diferente a 0 es verdadero.
  - Entonces si quiero hacer un while(true) en C, debiese hacer: while(1) { … }

Tipos de datos en C

- C no define un tipo especial para los valores de verdad o para strings (Java sí define los tipos boolean y String)
  - En C, 0 es falso y cualquier valor diferente a 0 es verdadero.
  - Entonces si quiero hacer un while(true) en C, debiese hacer: while(1) { … }
  - Ejemplo: Obtener el factorial de un entero n

Tipos de datos en C

- C no define un tipo especial para los valores de verdad o para strings (Java sí define los tipos boolean y String)
  - En C, 0 es falso y cualquier valor diferente a 0 es verdadero.
  - Entonces si quiero hacer un while(true) en C, debiese hacer: while(1) { … }
  - Ejemplo: Obtener el factorial de un entero n

```
int fact(int n) {
        int res = 1;
        while (n > 0) {
                res = res * n;
                n = n – 1;
        }
        return res;
}
```

Tipos de datos en C

- C no define un tipo especial para los valores de verdad o para strings (Java sí define los tipos boolean y String)
  - En C, 0 es falso y cualquier valor diferente a 0 es verdadero.
  - Entonces si quiero hacer un while(true) en C, debiese hacer: while(1) { … }
  - Ejemplo: Obtener el factorial de un entero n

```
int fact(int n) {
    int res = 1;
    while (n > 0) {
        res = res * n;
        n = n – 1;
    }
    return res;
}
```

```
int fact(int n) {
    int res = 1;
    while (1) {
        if (n == 0) {
            break;
        }
        res = res * n;
        n = n – 1;
    }
    return res;
}
```

Tipos de datos en C

- C no define un tipo especial para los valores de verdad o para strings (Java sí define los tipos boolean y String)
  - En C, 0 es falso y cualquier valor diferente a 0 es verdadero.
  - Entonces si quiero hacer un while(true) en C, debiese hacer: while(1) { … }
  - Ejemplo: Obtener el factorial de un entero n

```
int fact(int n) {
        int res = 1;
        while (n > 0) {
                res = res * n;
                n = n – 1;
        }
        return res;
}
```

```
res *= n;
n -= 1;
```

Enteros sin signo

- Los enteros sin signo se representan internamente en binario (base 2)
- Usaremos la notación $(x)_b$ para indicar que la constante x esta representada en base b. Si no se indica la base, se supone base 10
- Ejemplo:

  $(13)_{10} = (1101)_2 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 8+4+0+1 = 13$

Enteros sin signo

- Los enteros sin signo se representan internamente en binario (base 2)
- Usaremos la notación $(x)_b$ para indicar que la constante *x* esta representada en base *b*. Si no se indica la base, se supone base 10
- Ejemplo:

$$(13)_{10} = (\mathbf{1}101)_2 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 8+4+0+1 = 13$$

Bits MÁS significativo

Enteros sin signo

- Aunque la representación interna es base 2, en el lenguaje las constantes se anotan en base 10

Enteros sin signo

- Aunque la representación interna es base 2, en el lenguaje las constantes se anotan en base 10
- ¡También se puede usar base 8 anteponiendo el prefijo 0! El 013 no es el mismo número que 13 porque está en octal:

  $013 = (13)_8 = (001\ 011)_2 = 8+2+1 = 11$
- Un cero a la izquierda sí vale en C

Enteros sin signo

- Aunque la representación interna es base 2, en el lenguaje las constantes se anotan en base 10
- ¡También se puede usar base 8 anteponiendo el prefijo 0! El 013 no es el mismo número que 13 porque está en octal:

  $013= (13)_8 = (001\ 011)_2 = 8+2+1= 11$
- Un cero a la izquierda sí vale en C
- Se puede usar base 16 anteponiendo el prefijo 0x:

  $0x13 = (13)_{16} = (0001\ 0011)_2 = 16+2+1 = 19$

Enteros sin signo

- Aunque la representación interna es base 2, en el lenguaje las constantes se anotan en base 10
- ¡También se puede usar base 8 anteponiendo el prefijo 0! El 013 no es el mismo número que 13 porque está en octal:

  $013 = (13)_8 = (001\ 011)_2 = 8+2+1 = 11$
- Un cero a la izquierda sí vale en C
- Se puede usar base 16 anteponiendo el prefijo 0x:

  $0x13 = (13)_{16} = (0001\ 0011)_2 = 16+2+1 = 19$

Entonces en C se puede hacer:

```
int a = 13; int b = 013; int c = 0x13;
```

Enteros sin signo

- Aunque la representación interna es base 2, en el lenguaje las constantes se anotan en base 10
- ¡También se puede usar base 8 anteponiendo el prefijo 0! El 013 no es el mismo número que 13 porque está en octal:

  013= $(13)_8$ = $(001\ 011)_2$ = 8+2+1= 11
- Un cero a la izquierda sí vale en C
- Se puede usar base 16 anteponiendo el prefijo 0x:

  0x13 = $(13)_{16}$ = $(0001\ 0011)_2$ = 16+2+1 = 19

Entonces en C se puede hacer:

```
int a = 13; int b = 013; int c = 0x13;
```

- No existe 0b0110 para binarios, ¡pero lo usaremos en las explicaciones!

Conversiones

- Para convertir un número binario $x_{n-1} \dots x_0$ a base 10 hay que calcular:

$$\sum_{i=0}^{n-1} x_i 2^i$$

- Ejemplo:

  $(1100101)_2 =$

Conversiones

- Para convertir un número binario x_{n-1} … x_0 a base 10 hay que calcular:

$$\sum_{i=0}^{n-1} x_i 2^i$$

- Ejemplo:

  $(110010\mathbf{1})_2$ = **1**$\cdot 2^0$ +

Conversiones

- Para convertir un número binario $x_{n-1} \ldots x_0$ a base 10 hay que calcular:

$$\sum_{i=0}^{n-1} x_i 2^i$$

- Ejemplo:

$(1100\mathbf{1}01)_2 = 1 \cdot 2^0 + \mathbf{1} \cdot 2^2 +$

Conversiones

- Para convertir un número binario x_{n-1} ... x_0 a base 10 hay que calcular:

$$\sum_{i=0}^{n-1} x_i 2^i$$

- Ejemplo:

  $(1\mathbf{1}00101)_2 = 1 \cdot 2^0 + 1 \cdot 2^2 + \mathbf{1} \cdot 2^5$

Conversiones

- Para convertir un número binario x_{n-1} ... x_0 a base 10 hay que calcular:

$$\sum_{i=0}^{n-1} x_i 2^i$$

- Ejemplo:

  $(\mathbf{1}100101)_2 = 1 \cdot 2^0 + 1 \cdot 2^2 + 1 \cdot 2^5 + \mathbf{1} \cdot 2^6$

Conversiones

- Para convertir un número binario $x_{n-1} \dots x_0$ a base 10 hay que calcular:

$$\sum_{i=0}^{n-1} x_i 2^i$$

- Ejemplo:

$(1100101)_2 = 1 \cdot 2^0 + 1 \cdot 2^2 + 1 \cdot 2^5 + 1 \cdot 2^6 = 101$

Conversiones

- Para convertir un número en base 10 a base 2, dividir repetidamente por 2, anotando el resto de la división, hasta llegar a 0. El número en base 2 se obtiene leyendo los restos en orden inverso.
- Ejemplo: $25 = (?)_2$

Conversiones

- Para convertir un número en base 10 a base 2, dividir repetidamente por 2, anotando el resto de la división, hasta llegar a 0. El número en base 2 se obtiene leyendo los restos en orden inverso.
- Ejemplo: $25 = (?)_2$

división	resultado	resto
25 / 2		

Conversiones

- Para convertir un número en base 10 a base 2, dividir repetidamente por 2, anotando el resto de la división, hasta llegar a 0. El número en base 2 se obtiene leyendo los restos en orden inverso.
- Ejemplo: $25 = (?)_2$

división	resultado	resto
25 / 2	12	1

Conversiones

- Para convertir un número en base 10 a base 2, dividir repetidamente por 2, anotando el resto de la división, hasta llegar a 0. El número en base 2 se obtiene leyendo los restos en orden inverso.
- Ejemplo: 25 = $(?)_2$

división	resultado	resto
25 / 2	12	1
12 / 2		

Conversiones

- Para convertir un número en base 10 a base 2, dividir repetidamente por 2, anotando el resto de la división, hasta llegar a 0. El número en base 2 se obtiene leyendo los restos en orden inverso.
- Ejemplo: $25 = (?)_2$

división	resultado	resto
25 / 2	12	1
12 / 2	6	0

Conversiones

- Para convertir un número en base 10 a base 2, dividir repetidamente por 2, anotando el resto de la división, hasta llegar a 0. El número en base 2 se obtiene leyendo los restos en orden inverso.
- Ejemplo: $25 = (?)_2$

división	resultado	resto
25 / 2	12	1
12 / 2	6	0
6 / 2	3	0

Conversiones

- Para convertir un número en base 10 a base 2, dividir repetidamente por 2, anotando el resto de la división, hasta llegar a 0. El número en base 2 se obtiene leyendo los restos en orden inverso.
- Ejemplo: $25 = (?)_2$

división	resultado	resto
25 / 2	12	1
12 / 2	6	0
6 / 2	3	0
3 / 2	1	1

Conversiones

- Para convertir un número en base 10 a base 2, dividir repetidamente por 2, anotando el resto de la división, hasta llegar a 0. El número en base 2 se obtiene leyendo los restos en orden inverso.
- Ejemplo: 25 = $(?)_2$

división	resultado	resto
25 / 2	12	1
12 / 2	6	0
6 / 2	3	0
3 / 2	1	1
1 / 2	0	1

Conversiones

- Para convertir un número en base 10 a base 2, dividir repetidamente por 2, anotando el resto de la división, hasta llegar a 0. El número en base 2 se obtiene leyendo los restos en orden inverso.
- Ejemplo: $25 = (?)_2$

división	resultado	resto
25 / 2	12	1
12 / 2	6	0
6 / 2	3	0
3 / 2	1	1
1 / 2	0	1

Conversiones

- Para convertir un número en base 10 a base 2, dividir repetidamente por 2, anotando el resto de la división, hasta llegar a 0. El número en base 2 se obtiene leyendo los restos en orden inverso.
- Ejemplo: $25 = (?)_2$

división	resultado	resto
25 / 2	12	1
12 / 2	6	0
6 / 2	3	0
3 / 2	1	1
1 / 2	0	1

Respuesta: $25 = (11001)_2$

Rango de representación de enteros sin signo

tipo	Espacio en bytes Rango	Espacio en bytes Rango	Espacio en bytes Rango
	Máquinas de 64 bits	Máquinas de 32 bits	Máquinas de 16 bits
unsigned char	1	1	1
unsigned short	2	2	2
unsigned int	4	4	**2**
unsigned long	8	**4**	**4**
unsigned long long	8	8	8

- Que una máquina sea de *n* bits significa que los punteros son de *n* bits y por lo tanto puede direccionar hasta 2^n bytes de memoria

Rango de representación de enteros sin signo

tipo	Espacio en bytes Rango	Espacio en bytes Rango	Espacio en bytes Rango
	Máquinas de 64 bits	Máquinas de 32 bits	Máquinas de 16 bits
unsigned char	1	1	1
	$[0, 2^8[\equiv [0, 255]$	$[0, 2^8[$	$[0, 2^8[$
unsigned short	2	2	2
	$[0, 2^{16}[\equiv [0, 65535]$	$[0, 2^{16}[$	$[0, 2^{16}[$
unsigned int	4	4	**2**
	$[0, 2^{32}[\equiv [0, 4.294.967.295]$	$[0, 2^{32}[$	**$[0, 2^{16}[$**
unsigned long	8	**4**	**4**
	$[0, 2^{64}[$	**$[0, 2^{32}[$**	**$[0, 2^{32}[$**
unsigned long long	8	8	8
	$[0, 2^{64}[$	$[0, 2^{64}[$	$[0, 2^{64}[$

- Que una máquina sea de *n* bits significa que los punteros son de *n* bits y por lo tanto puede direccionar hasta 2^n bytes de memoria

Rango de representación de enteros sin signo

tipo	Espacio en bytes Rango	Espacio en bytes Rango	Espacio en bytes Rango
	Máquinas de 64 bits	Máquinas de 32 bits	Máquinas de 16 bits
unsigned char	1	1	1
	$[0, 2^8[\equiv [0, 255]$	$[0, 2^8[$	$[0, 2^8[$
unsigned short	2	2	2
	$[0, 2^{16}[\equiv [0, 65535]$	$[0, 2^{16}[$	$[0, 2^{16}[$
unsigned int	4	4	**2**
	$[0, 2^{32}[\equiv [0,$ 4.294.967.295$]$	$[0, 2^{32}[$	**$[0, 2^{16}[$**
unsigned long	8	**4**	**4**
	$[0, 2^{64}[$	**$[0, 2^{32}[$**	**$[0, 2^{32}[$**
unsigned long long	8	8	8
	$[0, 2^{64}[$	$[0, 2^{64}[$	$[0, 2^{64}[$

- Que una máquina sea de *n* bits significa que los punteros son de *n* bits y por lo tanto puede direccionar hasta 2^n bytes de memoria
- En Windows de 64 bits el tipo long es de 32 bits

Rango de representación de enteros sin signo

tipo	Espacio en bytes Rango	Espacio en bytes Rango	Espacio en bytes Rango
	Máquinas de 64 bits	Máquinas de 32 bits	Máquinas de 16 bits
unsigned char	1	1	1
	$[0, 2^8[\equiv [0, 255]$	$[0, 2^8[$	$[0, 2^8[$
unsigned short	2	2	2
	$[0, 2^{16}[\equiv [0, 65535]$	$[0, 2^{16}[$	$[0, 2^{16}[$
unsigned int	4	4	**2**
	$[0, 2^{32}[\equiv [0,$ 4.294.967.295]	$[0, 2^{32}[$	**$[0, 2^{16}[$**
unsigned long	8	**4**	**4**
	$[0, 2^{64}[$	**$[0, 2^{32}[$**	**$[0, 2^{32}[$**
unsigned long long	8	8	8
	$[0, 2^{64}[$	$[0, 2^{64}[$	$[0, 2^{64}[$

- Que una máquina sea de *n* bits significa que los punteros son de *n* bits y por lo tanto puede direccionar hasta 2^n bytes de memoria
- En Windows de 64 bits el tipo long es de 32 bits

Rango de representación de enteros sin signo

tipo	Espacio en bytes Rango	Espacio en bytes Rango	Espacio en bytes Rango
	Máquinas de 64 bits	Máquinas de 32 bits	Máquinas de 16 bits
unsigned char	1	1	1
	$[0, 2^8[\equiv [0, 255]$	$[0, 2^8[$	$[0, 2^8[$
unsigned short	2	2	2
	$[0, 2^{16}[\equiv [0, 65535]$	$[0, 2^{16}[$	$[0, 2^{16}[$
unsigned int	4	4	**2**
	$[0, 2^{32}[\equiv [0, 4.294.967.295]$	$[0, 2^{32}[$	**$[0, 2^{16}[$**
unsigned long	**4**	**4**	**4**
	$[0, 2^{32}[$	**$[0, 2^{32}[$**	**$[0, 2^{32}[$**
unsigned long long	8	8	8
	$[0, 2^{64}[$	$[0, 2^{64}[$	$[0, 2^{64}[$

- Que una máquina sea de *n* bits significa que los punteros son de *n* bits y por lo tanto puede direccionar hasta 2^n bytes de memoria
- En Windows de 64 bits el tipo long es de 32 bits

Rango de representación de enteros sin signo

tipo	Espacio en bytes Rango	Espacio en bytes Rango	Espacio en bytes Rango
	Máquinas de 64 bits	Máquinas de 32 bits	Máquinas de 16 bits
unsigned char	1	1	1
	$[0, 2^8[\equiv [0, 255]$	$[0, 2^8[$	$[0, 2^8[$
unsigned short	2	2	2
	$[0, 2^{16}[\equiv [0, 65535]$	$[0, 2^{16}[$	$[0, 2^{16}[$
unsigned int	4	4	**2**
	$[0, 2^{32}[\equiv [0,$ 4.294.967.295$]$	$[0, 2^{32}[$	**$[0, 2^{16}[$**
unsigned long	8	**4**	**4**
	$[0, 2^{64}[$	**$[0, 2^{32}[$**	**$[0, 2^{32}[$**
unsigned long long	8	8	8
	$[0, 2^{64}[$	$[0, 2^{64}[$	$[0, 2^{64}[$

- Que una máquina sea de *n* bits significa que los punteros son de *n* bits y por lo tanto puede direccionar hasta 2^n bytes de memoria
- En Windows de 64 bits el tipo long es de 32 bits
- Las máquinas de 64 bits pueden correr los programas de las máquinas de 32 bits

Rango de representación de enteros sin signo

tipo	Espacio en bytes Rango	Espacio en bytes Rango	Espacio en bytes Rango
	Máquinas de 64 bits	Máquinas de 32 bits	Máquinas de 16 bits
unsigned char	1	1	1
	$[0, 2^8[\equiv [0, 255]$	$[0, 2^8[$	$[0, 2^8[$
unsigned short	2	2	2
	$[0, 2^{16}[\equiv [0, 65535]$	$[0, 2^{16}[$	$[0, 2^{16}[$
unsigned int	4	4	**2**
	$[0, 2^{32}[\equiv [0, 4.294.967.295]$	$[0, 2^{32}[$	**$[0, 2^{16}[$**
unsigned long	8	**4**	**4**
	$[0, 2^{64}[$	**$[0, 2^{32}[$**	**$[0, 2^{32}[$**
unsigned long long	8	8	8
	$[0, 2^{64}[$	$[0, 2^{64}[$	$[0, 2^{64}[$

- Que una máquina sea de *n* bits significa que los punteros son de *n* bits y por lo tanto puede direccionar hasta 2^n bytes de memoria
- En Windows de 64 bits el tipo long es de 32 bits
- Las máquinas de 64 bits pueden correr los programas de las máquinas de 32 bits
- Muchos sistemas embebidos y dispositivos usan procesadores de 16 bits por razones de costo (mouse y teclado por ejemplo)

Rango de representación de enteros con signo

tipo	Espacio en bytes Rango	Espacio en bytes Rango	Espacio en bytes Rango
	Máquinas de 64 bits	Máquinas de 32 bits	Máquinas de 16 bits
signed char	1	1	1
	$[-2^7, 2^7[\equiv [-128, 127]$	$[-2^7, 2^7[\equiv [-128, 127]$	$[-2^7, 2^7[\equiv [-128, 127]$
short	2	2	2
	$[-2^{15}, 2^{15}[\equiv$ $[-32768, 32767]$	$[-2^{15}, 2^{15}[\equiv$ $[-32768, 32767]$	$[-2^{15}, 2^{15}[\equiv$ $[-32768, 32767]$
int	4	4	2
	$[-2^{31}, 2^{31}[\equiv$ [-2.147.483.648, 2.147.483.647]	$[-2^{31}, 2^{31}[\equiv$ [-2.147.483.648, 2.147.483.647]	$[-2^{15}, 2^{15}[$
long	8	4	4
	$[-2^{63}, 2^{63}[$	$[-2^{31}, 2^{31}[$	$[-2^{31}, 2^{31}[$
long long	8	8	8
	$[-2^{63}, 2^{63}[$	$[-2^{63}, 2^{63}[$	$[-2^{63}, 2^{63}[$

- Observe que el rango de representación no es simétrico: se puede representar el -128 en un char pero no el +128

Rango de representación de enteros con signo

tipo	Espacio en bytes Rango	Espacio en bytes Rango	Espacio en bytes Rango
	Máquinas de 64 bits	Máquinas de 32 bits	Máquinas de 16 bits
signed char	1	1	1
	$[-2^7, 2^7[$ ≡ [-128, 127]	$[-2^7, 2^7[$ ≡ [-128, 127]	$[-2^7, 2^7[$ ≡ [-128, 127]
short	2	2	2
	$[-2^{15}, 2^{15}[$ ≡ [-32768, 32767]	$[-2^{15}, 2^{15}[$ ≡ [-32768, 32767]	$[-2^{15}, 2^{15}[$ ≡ [-32768, 32767]
int	4	4	2
	$[-2^{31}, 2^{31}[$ ≡ [-2.147.483.648, 2.147.483.647]	$[-2^{31}, 2^{31}[$ ≡ [-2.147.483.648, 2.147.483.647]	$[-2^{15}, 2^{15}[$
long	8	4	4
	$[-2^{63}, 2^{63}[$	$[-2^{31}, 2^{31}[$	$[-2^{31}, 2^{31}[$
long long	8	8	8
	$[-2^{63}, 2^{63}[$	$[-2^{63}, 2^{63}[$	$[-2^{63}, 2^{63}[$

- Observe que el rango de representación no es simétrico: se puede representar el -128 en un char pero no el +128

```
char a = 127;

char b = 1;

char c = a + b;
```

Rango de representación de enteros con signo

tipo	Espacio en bytes Rango	Espacio en bytes Rango	Espacio en bytes Rango
	Máquinas de 64 bits	Máquinas de 32 bits	Máquinas de 16 bits
signed char	1	1	1
	$[-2^7, 2^7[\equiv [-128, 127]$	$[-2^7, 2^7[\equiv [-128, 127]$	$[-2^7, 2^7[\equiv [-128, 127]$
short	2	2	2
	$[-2^{15}, 2^{15}[\equiv$ [-32768, 32767]	$[-2^{15}, 2^{15}[\equiv$ [-32768, 32767]	$[-2^{15}, 2^{15}[\equiv$ [-32768, 32767]
int	4	4	2
	$[-2^{31}, 2^{31}[\equiv$ [-2.147.483.648, 2.147.483.647]	$[-2^{31}, 2^{31}[\equiv$ [-2.147.483.648, 2.147.483.647]	$[-2^{15}, 2^{15}[$
long	8	4	4
	$[-2^{63}, 2^{63}[$	$[-2^{31}, 2^{31}[$	$[-2^{31}, 2^{31}[$
long long	8	8	8
	$[-2^{63}, 2^{63}[$	$[-2^{63}, 2^{63}[$	$[-2^{63}, 2^{63}[$

- Observe que el rango de representación no es simétrico: se puede representar el -128 en un char pero no el +128

```
char a = 127;

char b = 1;

char c = a + b;
```

c será una suma de char, por lo que será una suma de 8 bits. Pero 128 no es representable en 8 bits, entonces c = -128 (overflow)

Rango de representación de enteros con signo

tipo	Espacio en bytes Rango	Espacio en bytes Rango	Espacio en bytes Rango
	Máquinas de 64 bits	Máquinas de 32 bits	Máquinas de 16 bits
signed char	1	1	1
	$[-2^7, 2^7[\equiv [-128, 127]$	$[-2^7, 2^7[\equiv [-128, 127]$	$[-2^7, 2^7[\equiv [-128, 127]$
short	2	2	2
	$[-2^{15}, 2^{15}[\equiv$ $[-32768, 32767]$	$[-2^{15}, 2^{15}[\equiv$ $[-32768, 32767]$	$[-2^{15}, 2^{15}[\equiv$ $[-32768, 32767]$
int	4	4	2
	$[-2^{31}, 2^{31}[\equiv$ [-2.147.483.648, 2.147.483.647]	$[-2^{31}, 2^{31}[\equiv$ [-2.147.483.648, 2.147.483.647]	$[-2^{15}, 2^{15}[$
long	8	4	4
	$[-2^{63}, 2^{63}[$	$[-2^{31}, 2^{31}[$	$[-2^{31}, 2^{31}[$
long long	8	8	8
	$[-2^{63}, 2^{63}[$	$[-2^{63}, 2^{63}[$	$[-2^{63}, 2^{63}[$

- Observe que el rango de representación no es simétrico: se puede representar el -128 en un char pero no el +128

```
char a = 127;

char b = 1;

char c = a + b;
```

0 1 1 1 1 1 1 1

0 0 0 0 0 0 0 1

1 0 0 0 0 0 0 0

1 0 0 0 0 0 0 0 = 2^7 – 256 = -128

Rango de representación de enteros con signo

tipo	Espacio en bytes Rango	Espacio en bytes Rango	Espacio en bytes Rango
	Máquinas de 64 bits	Máquinas de 32 bits	Máquinas de 16 bits
signed char	1	1	1
	$[-2^7, 2^7[$ ≡ [-128, 127]	$[-2^7, 2^7[$ ≡ [-128, 127]	$[-2^7, 2^7[$ ≡ [-128, 127]
short	2	2	2
	$[-2^{15}, 2^{15}[$ ≡ [-32768, 32767]	$[-2^{15}, 2^{15}[$ ≡ [-32768, 32767]	$[-2^{15}, 2^{15}[$ ≡ [-32768, 32767]
int	4	4	2
	$[-2^{31}, 2^{31}[$ ≡ [-2.147.483.648, 2.147.483.647]	$[-2^{31}, 2^{31}[$ ≡ [-2.147.483.648, 2.147.483.647]	$[-2^{15}, 2^{15}[$
long	8	4	4
	$[-2^{63}, 2^{63}[$	$[-2^{31}, 2^{31}[$	$[-2^{31}, 2^{31}[$
long long	8	8	8
	$[-2^{63}, 2^{63}[$	$[-2^{63}, 2^{63}[$	$[-2^{63}, 2^{63}[$

- Observe que el rango de representación no es simétrico: se puede representar el -128 en un char pero no el +128
- Al operar con enteros, si se produce un desborde en la representación no se genera ningún tipo de error, ¡pero el resultado es incorrecto!

Rango de representación de enteros con signo

tipo	Espacio en bytes Rango	Espacio en bytes Rango	Espacio en bytes Rango
	Máquinas de 64 bits	Máquinas de 32 bits	Máquinas de 16 bits
signed char	1	1	1
	$[-2^7, 2^7[\equiv [-128, 127]$	$[-2^7, 2^7[\equiv [-128, 127]$	$[-2^7, 2^7[\equiv [-128, 127]$
short	2	2	2
	$[-2^{15}, 2^{15}[\equiv$ $[-32768, 32767]$	$[-2^{15}, 2^{15}[\equiv$ $[-32768, 32767]$	$[-2^{15}, 2^{15}[\equiv$ $[-32768, 32767]$
int	4	4	2
	$[-2^{31}, 2^{31}[\equiv$ $[-2.147.483.648, 2.147.483.647]$	$[-2^{31}, 2^{31}[\equiv$ $[-2.147.483.648, 2.147.483.647]$	$[-2^{15}, 2^{15}[$
long	8	4	4
	$[-2^{63}, 2^{63}[$	$[-2^{31}, 2^{31}[$	$[-2^{31}, 2^{31}[$
long long	8	8	8
	$[-2^{63}, 2^{63}[$	$[-2^{63}, 2^{63}[$	$[-2^{63}, 2^{63}[$

- Observe que el rango de representación no es simétrico: se puede representar el -128 en un char pero no el +128
- Al operar con enteros, si se produce un desborde en la representación no se genera ningún tipo de error, ¡pero el resultado es incorrecto!
- Los enteros positivos se representan en base 2 como si fuesen enteros sin signo

Rango de representación de enteros con signo

tipo	Espacio en bytes Rango	Espacio en bytes Rango	Espacio en bytes Rango
	Máquinas de 64 bits	Máquinas de 32 bits	Máquinas de 16 bits
signed char	1	1	1
	$[-2^7, 2^7[\equiv [-128, 127]$	$[-2^7, 2^7[\equiv [-128, 127]$	$[-2^7, 2^7[\equiv [-128, 127]$
short	2	2	2
	$[-2^{15}, 2^{15}[\equiv [-32768, 32767]$	$[-2^{15}, 2^{15}[\equiv [-32768, 32767]$	$[-2^{15}, 2^{15}[\equiv [-32768, 32767]$
int	4	4	2
	$[-2^{31}, 2^{31}[\equiv$ [-2.147.483.648, 2.147.483.647]	$[-2^{31}, 2^{31}[\equiv$ [-2.147.483.648, 2.147.483.647]	$[-2^{15}, 2^{15}[$
long	8	4	4
	$[-2^{63}, 2^{63}[$	$[-2^{31}, 2^{31}[$	$[-2^{31}, 2^{31}[$
long long	8	8	8
	$[-2^{63}, 2^{63}[$	$[-2^{63}, 2^{63}[$	$[-2^{63}, 2^{63}[$

- Observe que el rango de representación no es simétrico: se puede representar el -128 en un char pero no el +128
- Al operar con enteros, si se produce un desborde en la representación no se genera ningún tipo de error, ¡pero el resultado es incorrecto!
- Los enteros positivos se representan en base 2 como si fuesen enteros sin signo
- Los enteros negativos se representan en *complemento de 2*

Rango de representación de enteros con signo

tipo	Espacio en bytes Rango	Espacio en bytes Rango	Espacio en bytes Rango
	Máquinas de 64 bits	Máquinas de 32 bits	Máquinas de 16 bits
signed char	1	1	1
	$[-2^{7}, 2^{7}[\equiv [-128, 127]$	$[-2^{7}, 2^{7}[\equiv [-128, 127]$	$[-2^{7}, 2^{7}[\equiv [-128, 127]$
short	2	2	2
	$[-2^{15}, 2^{15}[\equiv$ $[-32768, 32767]$	$[-2^{15}, 2^{15}[\equiv$ $[-32768, 32767]$	$[-2^{15}, 2^{15}[\equiv$ $[-32768, 32767]$
int	4	4	2
	$[-2^{31}, 2^{31}[\equiv$ [-2.147.483.648, 2.147.483.647]	$[-2^{31}, 2^{31}[\equiv$ [-2.147.483.648, 2.147.483.647]	$[-2^{15}, 2^{15}[$
long	8	4	4
	$[-2^{63}, 2^{63}[$	$[-2^{31}, 2^{31}[$	$[-2^{31}, 2^{31}[$
long long	8	8	8
	$[-2^{63}, 2^{63}[$	$[-2^{63}, 2^{63}[$	$[-2^{63}, 2^{63}[$

- Observe que el rango de representación no es simétrico: se puede representar el -128 en un char pero no el +128
- Al operar con enteros, si se produce un desborde en la representación no se genera ningún tipo de error, ¡pero el resultado es incorrecto!
- Los enteros positivos se representan en base 2 como si fuesen enteros sin signo
- Los enteros negativos se representan en *complemento de 2*
- El tipo char es unsigned en arquitecturas Arm y Risc-V, y es signed en arquitecturas Intel/Amd x86

Representación de negativos en complemento de 2

- ¿Cómo se representa el -28 en binario en un **signed char**?

Representación de negativos en complemento de 2

- ¿Cómo se representa el -28 en binario en un signed char?
- Método:
  - Tomar valor absoluto: 28

Representación de negativos en complemento de 2

- ¿Cómo se representa el -28 en binario en un signed char?
- Método:
  - Tomar valor absoluto: 28
  - Representar en binario (dividir por 2 repetidamente hasta llegar a 0): 11100

Representación de negativos en complemento de 2

- ¿Cómo se representa el -28 en binario en un signed char?
- Método:
  - Tomar valor absoluto: 28
  - Representar en binario (dividir por 2 repetidamente hasta llegar a 0): 11100
  - Extender a 8 bits: 00011100

Representación de negativos en complemento de 2

- ¿Cómo se representa el -28 en binario en un signed char?
- Método:
  - Tomar valor absoluto: 28
  - Representar en binario (dividir por 2 repetidamente hasta llegar a 0): 11100
  - Extender a 8 bits: 00011100
  - Calcular el complemento de 1 (convertir 0s a 1 y 1s a 0): 11100011

Representación de negativos en complemento de 2

- ¿Cómo se representa el -28 en binario en un signed char?
- Método:
  - Tomar valor absoluto: 28
  - Representar en binario (dividir por 2 repetidamente hasta llegar a 0): 11100
  - Extender a 8 bits: 00011100
  - Calcular el complemento de 1 (convertir 0s a 1 y 1s a 0): 11100011

~(00011100)

Representación de negativos en complemento de 2

- ¿Cómo se representa el -28 en binario en un signed char?
- Método:
  - Tomar valor absoluto: 28
  - Representar en binario (dividir por 2 repetidamente hasta llegar a 0): 11100
  - Extender a 8 bits: 00011100
  - Calcular el complemento de 1 (convertir 0s a 1 y 1s a 0): 11100011
  - Sumar 1 en binario: 11100100

1 1 1 0 0 0 1 1
0 0 0 0 0 0 0 1

Representación de negativos en complemento de 2

- ¿Cómo se representa el -28 en binario en un signed char?
- Método:
  - Tomar valor absoluto: 28
  - Representar en binario (dividir por 2 repetidamente hasta llegar a 0): 11100
  - Extender a 8 bits: 00011100
  - Calcular el complemento de 1 (convertir 0s a 1 y 1s a 0): 11100011
  - Sumar 1 en binario: 11100100

1 1 1 0 0 0 1 1
0 0 0 0 0 0 0 1

Representación de negativos en complemento de 2

- ¿Cómo se representa el -28 en binario en un signed char?
- Método:
  - Tomar valor absoluto: 28
  - Representar en binario (dividir por 2 repetidamente hasta llegar a 0): 11100
  - Extender a 8 bits: 00011100
  - Calcular el complemento de 1 (convertir 0s a 1 y 1s a 0): 11100011
  - Sumar 1 en binario: 11100100

1
1 1 1 0 0 0 1 1
0 0 0 0 0 0 0 1
0

Representación de negativos en complemento de 2

- ¿Cómo se representa el -28 en binario en un signed char?
- Método:
  - Tomar valor absoluto: 28
  - Representar en binario (dividir por 2 repetidamente hasta llegar a 0): 11100
  - Extender a 8 bits: 00011100
  - Calcular el complemento de 1 (convertir 0s a 1 y 1s a 0): 11100011
  - Sumar 1 en binario: 11100100

						1	
1	**1**	**1**	**0**	**0**	**0**	**1**	**1**
0	**0**	**0**	**0**	**0**	**0**	**0**	**1**
							0

Representación de negativos en complemento de 2

- ¿Cómo se representa el -28 en binario en un signed char?
- Método:
  - Tomar valor absoluto: 28
  - Representar en binario (dividir por 2 repetidamente hasta llegar a 0): 11100
  - Extender a 8 bits: 00011100
  - Calcular el complemento de 1 (convertir 0s a 1 y 1s a 0): 11100011
  - Sumar 1 en binario: 11100100

					1	1	
1	**1**	**1**	**0**	**0**	**0**	**1**	**1**
0	**0**	**0**	**0**	**0**	**0**	**0**	**1**
						0	0

Representación de negativos en complemento de 2

- ¿Cómo se representa el -28 en binario en un signed char?
- Método:
  - Tomar valor absoluto: 28
  - Representar en binario (dividir por 2 repetidamente hasta llegar a 0): 11100
  - Extender a 8 bits: 00011100
  - Calcular el complemento de 1 (convertir 0s a 1 y 1s a 0): 11100011
  - Sumar 1 en binario: 11100100

					1	1	
1	**1**	**1**	**0**	**0**	**0**	**1**	**1**
0	**0**	**0**	**0**	**0**	**0**	**0**	**1**
						0	0

Representación de negativos en complemento de 2

- ¿Cómo se representa el -28 en binario en un signed char?
- Método:
    - Tomar valor absoluto: 28
    - Representar en binario (dividir por 2 repetidamente hasta llegar a 0): 11100
    - Extender a 8 bits: 00011100
    - Calcular el complemento de 1 (convertir 0s a 1 y 1s a 0): 11100011
    - Sumar 1 en binario: 11100100

0 1 1
1 1 1 0 0 0 1 1
0 0 0 0 0 0 0 1
1 0 0

Representación de negativos en complemento de 2

- ¿Cómo se representa el -28 en binario en un signed char?
- Método:
  - Tomar valor absoluto: 28
  - Representar en binario (dividir por 2 repetidamente hasta llegar a 0): 11100
  - Extender a 8 bits: 00011100
  - Calcular el complemento de 1 (convertir 0s a 1 y 1s a 0): 11100011
  - Sumar 1 en binario: 11100100

0 0 0 0 0 1 1
1 1 1 0 0 0 1 1
0 0 0 0 0 0 0 1
1 1 1 0 0 1 0 0

Representación de negativos en complemento de 2

- ¿Cómo se representa el -28 en binario en un signed char?
- Método:
  - Tomar valor absoluto: 28
  - Representar en binario (dividir por 2 repetidamente hasta llegar a 0): 11100
  - Extender a 8 bits: 00011100
  - Calcular el complemento de 1 (convertir 0s a 1 y 1s a 0): 11100011
  - Sumar 1 en binario: 11100100
- En resumen para representar un entero negativo, calcular complemento de 1 + 1 del valor positivo
- En C: -x ≡ ~x + 1 ...

Representación de negativos en complemento de 2

- ¿Cómo se representa el -28 en binario en un signed char?
- Método:
  - Tomar valor absoluto: 28
  - Representar en binario (dividir por 2 repetidamente hasta llegar a 0): 11100
  - Extender a 8 bits: 00011100
  - Calcular el complemento de 1 (convertir 0s a 1 y 1s a 0): 11100011
  - Sumar 1 en binario: 11100100
- En resumen para representar un entero negativo, calcular complemento de 1 + 1 del valor positivo
- En C: $-x \equiv \sim x + 1 \quad \Rightarrow \quad x - y \equiv x + (\sim y + 1)$

Tabla

Número binario	Valor sin signo	Valor con signo
00000000	0	0
00000001	1	1
00000010	2	2
00000011	3	3
00000100	4	4
…		
01111111	127	127
10000000	128	-128
10000001	129	-127
…	…	…
11111100	252	-4
11111101	253	-3
11111110	254	-2
11111111	255	-1

- El primer bit en un signed indica el signo

Tabla

Número binario	Valor sin signo	Valor con signo
00000000	0	0
00000001	1	1
00000010	2	2
00000011	3	3
00000100	4	4
…		
01111111	127	127
10000000	128	-128
10000001	129	-127
…	…	…
11111100	252	-4
11111101	253	-3
11111110	254	-2
11111111	255	-1

- El primer bit en un signed indica el signo
- Si es un número positivo el primer bit es 0, si es negativo el primer bit es 1

Tabla

Número binario	Valor sin signo	Valor con signo
00000000	0	0
00000001	1	1
00000010	2	2
00000011	3	3
00000100	4	4
…		
01111111	127	127
10000000	128	-128
10000001	129	-127
…	…	…
11111100	252	-4
11111101	253	-3
11111110	254	-2
11111111	255	-1

- El primer bit en un signed indica el signo
- Si es un número positivo el primer bit es 0, si es negativo el primer bit es 1
- Se llama bit de signo y es el bit más significativo

Tabla

Número binario	Valor sin signo	Valor con signo
00000000	0	0
00000001	1	1
00000010	2	2
00000011	3	3
00000100	4	4
…		
01111111	127	127
10000000	128	-128
10000001	129	-127
…	…	…
11111100	252	-4
11111101	253	-3
11111110	254	-2
11111111	255	-1

- El primer bit en un signed indica el signo
- Si es un número positivo el primer bit es 0, si es negativo el primer bit es 1
- Se llama bit de signo y es el bit más significativo
- El -1 es 111…..111 en todos los tamaños

Conversión a base 10 de enteros con signo

- ¿Qué valor con signo representa el 11101101?

Conversión a base 10 de enteros con signo

- ¿Qué valor con signo representa el 11101101?
- Para convertir un número binario de *n* bits $x_{n-1} \ldots x_0$ a base 10:
  - Si bit de signo $x_{n-1} \equiv 0$: calcular $\sum_{i=0}^{n-1} x_i 2^i$
  - Si bit de signo $x_{n-1} \equiv 1$: calcular $\sum_{i=0}^{n-1} x_i 2^i - 2^n$

Conversión a base 10 de enteros con signo

- ¿Qué valor con signo representa el 11101101?
- Para convertir un número binario de *n* bits $x_{n-1} \ldots x_0$ a base 10:
  - Si bit de signo $x_{n-1} \equiv 0$: calcular $\sum_{i=0}^{n-1} x_i 2^i$
  - Si bit de signo $x_{n-1} \equiv 1$: calcular $\sum_{i=0}^{n-1} x_i 2^i - 2^n$
- Para 11101101, $n \equiv 8$: $\sum \equiv$ 128+64+32+8+4+1= 237

Conversión a base 10 de enteros con signo

- ¿Qué valor con signo representa el 11101101?
- Para convertir un número binario de *n* bits $x_{n-1} \ldots x_0$ a base 10:
  - Si bit de signo $x_{n-1} \equiv 0$: calcular $\sum_{i=0}^{n-1} x_i 2^i$
  - Si bit de signo $x_{n-1} \equiv 1$: calcular $\sum_{i=0}^{n-1} x_i 2^i - 2^n$
- Para 11101101, $n \equiv 8$: $\sum \equiv$ 128+64+32+8+4+1= 237
- Como el bit de signo es 1 : 237 – 256 = **-19**

Conversión a base 10 de enteros con signo

- ¿Qué valor con signo representa el 11101101?
- Para convertir un número binario de *n* bits $x_{n-1} \ldots x_0$ a base 10:
  - Si bit de signo $x_{n-1} \equiv 0$: calcular $\sum_{i=0}^{n-1} x_i 2^i$
  - Si bit de signo $x_{n-1} \equiv 1$: calcular $\sum_{i=0}^{n-1} x_i 2^i - 2^n$
- Para 11101101, $n \equiv 8$: $\sum \equiv 128+64+32+8+4+1= 237$
- Como el bit de signo es 1 : 237 – 256 = **-19**
- ¿Por qué se eligió el complemento de 2?
- ¡Porque la suma de los enteros con signo es la misma que la suma de los enteros sin signo!
- Porque $x - y \equiv x + \sim y + 1$
- ¡El mismo sumador sirve para sumar o restar enteros con o sin signo!

Representación de números reales en *punto fijo*

- Se destina una cantidad *fija* de bits para la parte fraccional

Representación de números reales en *punto fijo*

- Se destina una cantidad *fija* de bits para la parte fraccional
- Por ejemplo 6.25 en punto fijo de 32 bits con una fracción de 16 bits sería:

  00000000 00000**110** . **01**000000 00000000

  porque su valor es $1 \cdot 2^{2} + 1 \cdot 2^{1} + 0 \cdot 2^{0} + 0 \cdot 2^{-1} + 1 \,.\, 2^{-2}$
- El punto va en un lugar fijo

Representación de números reales en *punto fijo*

- Se destina una cantidad *fija* de bits para la parte fraccional
- Por ejemplo 6.25 en punto fijo de 32 bits con una fracción de 16 bits sería:

  00000000 00000**110** . **01**000000 00000000

  porque su valor es $1 \cdot 2^{2} + 1 \cdot 2^{1} + 0 \cdot 2^{0} + 0 \cdot 2^{-1} + 1 . 2^{-2}$
- El punto va en un lugar fijo

Multiplicando por 2 hasta llegar exactamente a 1.00

Representación de números reales en *punto fijo*

- Se destina una cantidad *fija* de bits para la parte fraccional
- Por ejemplo 6.25 en punto fijo de 32 bits con una fracción de 16 bits sería:

  00000000 00000**110** . **01**000000 00000000

  porque su valor es $1 \cdot 2^{2} + 1 \cdot 2^{1}$ $0 \cdot 2^{0} + 0 \cdot 2^{-1} + 1 \,.\, 2^{-2}$
- El punto va en un lugar fijo

Multiplicando por 2 hasta llegar exactamente a 1.00

Operación	Resultado	Parte entera
0.25 * 2	0.50	0
0.50 * 2	1.00	1
0.00 * 2	0	0

Representación de números reales en *punto fijo*

- Se destina una cantidad *fija* de bits para la parte fraccional
- Por ejemplo 6.25 en punto fijo de 32 bits con una fracción de 16 bits sería:

  00000000 00000**110** . **01**000000 00000000

  porque su valor es $1 \cdot 2^{2} + 1 \cdot 2^{1} + 0 \cdot 2^{0} + 0 \cdot 2^{-1} + 1 . 2^{-2}$
- El punto va en un lugar fijo

Multiplicando por 2 hasta llegar exactamente a 1.00

Operación	Resultado	Parte entera
0.25 * 2	0.50	0
0.50 * 2	1.00	1
0.00 * 2	0	0

Representación de números reales en *punto fijo*

- Se destina una cantidad *fija* de bits para la parte fraccional
- Por ejemplo 6.25 en punto fijo de 32 bits con una fracción de 16 bits sería:

  00000000 00000**110** . **01**000000 00000000

  porque su valor es $1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 + 0 \cdot 2^{-1} + 1 . 2^{-2}$
- El punto va en un lugar fijo
- En general si:

  $x = x_{15} \ldots x_0\, x_{-1} \ldots x_{-16}$ el valor sería $\sum_{i=-16}^{15} x_i 2^i$

Representación de números reales en *punto fijo*

- Se destina una cantidad *fija* de bits para la parte fraccional
- Por ejemplo 6.25 en punto fijo de 32 bits con una fracción de 16 bits sería:

  00000000 00000**110** . **01**000000 00000000

  porque su valor es $1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 + 0 \cdot 2^{-1} + 1 . 2^{-2}$
- El punto va en un lugar fijo
- En general si:

  $x = x_{15} \ldots x_0 \, x_{-1} \ldots x_{-16}$ el valor sería $\sum_{i=-16}^{15} x_i 2^i$
- Habría que destinar otro bit para indicar el signo

Representación de números reales en *punto fijo*

- Se destina una cantidad *fija* de bits para la parte fraccional
- Por ejemplo 6.25 en punto fijo de 32 bits con una fracción de 16 bits sería:

  00000000 00000**110** . **01**000000 00000000

  porque su valor es $1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 + 0 \cdot 2^{-1} + 1 . 2^{-2}$
- El punto va en un lugar fijo
- En general si:

  $x = x_{15} \ldots x_0\ x_{-1} \ldots x_{-16}$ el valor sería $\sum_{i=-16}^{15} x_i 2^i$
- Habría que destinar otro bit para indicar el signo

¡No se usa!

Representación de números reales en *punto flotante* de 32 bits

- Sea $x \neq 0$ un número real expresado en base 2
- Ejemplo: $6.25 = (110.01)_2$

Representación de números reales en *punto flotante* de 32 bits

- Sea $x \neq 0$ un número real expresado en base 2
- Ejemplo: $6.25 = (110.01)_2$
- Hay que *normalizar* el número: se reescribe de manera que esté en el formato $1.bbbb\ldots \cdot 2^e$

Representación de números reales en *punto flotante* de 32 bits

- Sea $x \neq 0$ un número real expresado en base 2
- Ejemplo: $6.25 = (110.01)_2$
- Hay que *normalizar* el número: se reescribe de manera que esté en el formato $1.bbbb... \cdot 2^e$

$$(1\,1\,0\,.\,0\,1)_2$$

Representación de números reales en *punto flotante* de 32 bits

- Sea $x \neq 0$ un número real expresado en base 2
- Ejemplo: $6.25 = (110.01)_2$
- Hay que *normalizar* el número: se reescribe de manera que esté en el formato $1.bbbb\ldots \cdot 2^e$

$$(1\,1\,0\,.\,0\,1)_2$$

$$(1\,.\,1\,0\,0\,1)_2 \cdot 2^2$$

OJO, dado que descartamos el 0, entonces siempre habrá un bit en 1 que sea el más significativo

Representación de números reales en *punto flotante* de 32 bits

- Sea $x \neq 0$ un número real expresado en base 2
- Ejemplo: $6.25 = (\ 110.01\ )_2$
- Hay que *normalizar* el número: se reescribe de manera que esté en el formato $1.bbbb\ldots \cdot 2^e$
- Ejemplo: $6.25 = (\ 1.1001\ )_2 \cdot 2^2$

Representación de números reales en *punto flotante* de 32 bits

- Sea $x \neq 0$ un número real expresado en base 2
- Ejemplo: $6.25 = (\ 110.01\ )_2$
- Hay que *normalizar* el número: se reescribe de manera que esté en el formato $1.bbbb\ldots \cdot 2^e$
- Ejemplo: $6.25 = (\ 1.1001\ )_2 \cdot 2^2$
- En general x estará en el formato:

  $signo \cdot 1\,.\,m_{-1}\,m_{-2}\ldots m_{-23} \cdot 2^{e}$

Representación de números reales en *punto flotante* de 32 bits

- Sea $x \neq 0$ un número real expresado en base 2
- Ejemplo: $6.25 = (\ 110.01\ )_2$
- Hay que *normalizar* el número: se reescribe de manera que esté en el formato $1.bbbb\ldots \cdot 2^e$
- Ejemplo: $6.25 = (\ 1.1001\ )_2 \cdot 2^2$
- En general x estará en el formato:

  $signo \cdot 1\,.\,m_{-1}\, m_{-2} \ldots m_{-23} \cdot 2^{e}$
- En donde signo puede ser 1 o -1

Representación de números reales en *punto flotante* de 32 bits

- Sea $x \neq 0$ un número real expresado en base 2
- Ejemplo: $6.25 = (\ 110.01\ )_2$
- Hay que *normalizar* el número: se reescribe de manera que esté en el formato $1.bbbb\ldots \cdot 2^e$
- Ejemplo: $6.25 = (\ 1.1001\ )_2 \cdot 2^2$
- En general x estará en el formato:

  $signo \cdot 1\,.\,m_{-1}\, m_{-2} \ldots m_{-23} \cdot 2^{e}$
- En donde signo puede ser 1 o -1
- Los bits $m_{-1}\, m_{-2} \ldots m_{-23}$ se llaman la mantisa

Representación de números reales en *punto flotante* de 32 bits

- Sea $x \neq 0$ un número real expresado en base 2
- Ejemplo: $6.25 = (\ 110.01\ )_2$
- Hay que *normalizar* el número: se reescribe de manera que esté en el formato $1.bbbb... \cdot 2^e$
- Ejemplo: $6.25 = (\ 1.1001\ )_2 \cdot 2^2$
- En general x estará en el formato:

  $signo \cdot 1\ .\ m_{-1}\ m_{-2} \ldots m_{-23} \cdot 2^e$
- En donde signo puede ser 1 o -1
- Los bits $m_{-1}\ m_{-2} \ldots m_{-23}$ se llaman la mantisa

Representado en 23 bits $\boxed{\mathbf{1001}0000000000000000000}$

Representación de números reales en *punto flotante* de 32 bits

- Sea $x \neq 0$ un número real expresado en base 2
- Ejemplo: $6.25 = (\ 110.01\ )_2$
- Hay que *normalizar* el número: se reescribe de manera que esté en el formato $1.bbbb\ldots \cdot 2^e$
- Ejemplo: $6.25 = (\ 1.1001\ )_2 \cdot 2^2$
- En general x estará en el formato:

  $signo \cdot 1\,.\,m_{-1}\, m_{-2} \ldots m_{-23} \cdot 2^{e}$
- En donde signo puede ser 1 o -1
- Los bits $m_{-1}\, m_{-2} \ldots m_{-23}$ se llaman la mantisa
- El número x se representa como: $s\ e_7 \ldots e_0\ m_{-1} \ldots m_{-23}$

Representación de números reales en *punto flotante* de 32 bits

- Sea $x \neq 0$ un número real expresado en base 2
- Ejemplo: $6.25 = (\ 110.01\ )_2$
- Hay que *normalizar* el número: se reescribe de manera que esté en el formato $1.bbbb\ldots \cdot 2^e$
- Ejemplo: $6.25 = (\ 1.1001\ )_2 \cdot 2^2$
- En general *x* estará en el formato:

  $signo \cdot 1\,.\,m_{-1}\, m_{-2} \ldots m_{-23} \cdot 2^e$
- En donde signo puede ser 1 o -1
- Los bits $m_{-1}\, m_{-2} \ldots m_{-23}$ se llaman la mantisa
- El número *x* se representa como: $s\ e_7 \ldots e_0\ m_{-1} \ldots m_{-23}$
- Con s=0 si signo es 1 o s=1 si signo es -1

$\boxed{6.25 \text{ es positivo} \Rightarrow s = 0}$

Representación de números reales en *punto flotante* de 32 bits

- Sea $x \neq 0$ un número real expresado en base 2
- Ejemplo: $6.25 = (\ 110.01\ )_2$
- Hay que *normalizar* el número: se reescribe de manera que esté en el formato $1.bbbb\ldots \cdot 2^e$
- Ejemplo: $6.25 = (\ 1.1001\ )_2 \cdot 2^2$
- En general x estará en el formato:

  $signo \cdot 1\,.\,m_{-1}\,m_{-2} \ldots m_{-23} \cdot 2^{e}$
- En donde signo puede ser 1 o -1
- Los bits $m_{-1}\,m_{-2} \ldots m_{-23}$ se llaman la mantisa
- El número x se representa como: $s\ e_7 \ldots e_0\ m_{-1} \ldots m_{-23}$
- Con s=0 si signo es 1 o s=1 si signo es -1
- La parte 1. no se incluye porque es siempre lo mismo

Representación de números reales en *punto flotante* de 32 bits

- Sea $x \neq 0$ un número real expresado en base 2
- Ejemplo: $6.25 = (110.01)_2$
- Hay que *normalizar* el número: se reescribe de manera que esté en el formato $1.bbbb... \cdot 2^e$
- Ejemplo: $6.25 = (1.1001)_2 \cdot 2^2$
- En general x estará en el formato:

  $signo \cdot 1 . m_{-1}\, m_{-2} \dots m_{-23} \cdot 2^{e}$
- En donde signo puede ser 1 o -1
- Los bits $m_{-1}\, m_{-2} \dots m_{-23}$ se llaman la mantisa
- El número x se representa como: $s\ e_7 \dots e_0\ m_{-1} \dots m_{-23}$
- Con s=0 si signo es 1 o s=1 si signo es -1
- La parte 1. no se incluye porque es siempre lo mismo
- El valor sin signo de $e_7 \dots e_0$ es e + 127 (129 para 6.25)

Representado en 8 bits $\boxed{10000001}$

Representación de números reales en *punto flotante* de 32 bits

- Sea $x \neq 0$ un número real expresado en base 2
- Ejemplo: $6.25 = (\ 110.01\ )_2$
- Hay que *normalizar* el número: se reescribe de manera que esté en el formato $1.bbbb\ldots \cdot 2^e$
- Ejemplo: $6.25 = (\ 1.1001\ )_2 \cdot 2^2$
- En general *x* estará en el formato:

  $signo \cdot 1\,.\,m_{-1}\, m_{-2} \ldots m_{-23} \cdot 2^{\mathrm{e}}$
- En donde signo puede ser 1 o -1
- Los bits $m_{-1}\, m_{-2} \ldots m_{-23}$ se llaman la mantisa
- El número *x* se representa como: $s\ e_7 \ldots e_0\ m_{-1} \ldots m_{-23}$
- Con s=0 si signo es 1 o s=1 si signo es -1
- La parte 1. no se incluye porque es siempre lo mismo
- El valor sin signo de $e_7 \ldots e_0$ es e + 127 (129 para 6.25)
- Por lo tanto 6.25 = 0

Representación de números reales en *punto flotante* de 32 bits

- Sea $x \neq 0$ un número real expresado en base 2
- Ejemplo: $6.25 = (\ 110.01\ )_2$
- Hay que *normalizar* el número: se reescribe de manera que esté en el formato $1.bbbb\ldots \cdot 2^e$
- Ejemplo: $6.25 = (\ 1.1001\ )_2 \cdot 2^2$
- En general x estará en el formato:

  $signo \cdot 1\,.\,m_{-1}\, m_{-2} \ldots m_{-23} \cdot 2^{e}$
- En donde signo puede ser 1 o -1
- Los bits $m_{-1}\, m_{-2} \ldots m_{-23}$ se llaman la mantisa
- El número x se representa como: $s\ e_7 \ldots e_0\ m_{-1} \ldots m_{-23}$
- Con s=0 si signo es 1 o s=1 si signo es -1
- La parte 1. no se incluye porque es siempre lo mismo
- El valor sin signo de $e_7 \ldots e_0$ es e + 127 (129 para 6.25)
- Por lo tanto 6.25 = 0100 0000 1

Representación de números reales en *punto flotante* de 32 bits

- Sea $x \neq 0$ un número real expresado en base 2
- Ejemplo: $6.25 = (\ 110.01\ )_2$
- Hay que *normalizar* el número: se reescribe de manera que esté en el formato $1.bbbb\ldots \cdot 2^e$
- Ejemplo: $6.25 = (\ 1.1001\ )_2 \cdot 2^2$
- En general x estará en el formato:

  $signo \cdot 1\ .\ m_{-1}\ m_{-2} \ldots m_{-23} \cdot 2^{e}$
- En donde signo puede ser 1 o -1
- Los bits $m_{-1}\ m_{-2} \ldots m_{-23}$ se llaman la mantisa
- El número x se representa como: $s\ e_7 \ldots e_0\ m_{-1} \ldots m_{-23}$
- Con s=0 si signo es 1 o s=1 si signo es -1
- La parte 1. no se incluye porque es siempre lo mismo
- El valor sin signo de $e_7 \ldots e_0$ es e + 127 (129 para 6.25)
- Por lo tanto 6.25 = 0100 0000 1100 1000 00000000 00000000

Representación de números reales en *punto flotante*

- Los casos $e_7 \ldots e_0$ = 0 o 255 son especiales
- El 0 se representa como 000…0 (solo ceros)

Representación de números reales en *punto flotante*

- Los casos $e_7 \ldots e_0$ = 0 o 255 son especiales
- El 0 se representa como 000…0 (solo ceros)
- Hay una representación para el NaN: *not a number*

Representación de números reales en *punto flotante*

- Los casos $e_7 \dots e_0$ = 0 o 255 son especiales
- El 0 se representa como 000…0 (solo ceros)
- Hay una representación para el NaN: *not a number*

Tipo	Espacio	Precisión	Min magnitude	Máx magnitud	Mantiza	Exp
float	32 bits (4 bytes)	7 dígitos	$1.18 \cdot 10^{-38}$	$3.4 \cdot 10^{38}$	23 bits	8 bits
double	64 bits (8 bytes)	15 dígitos	$2.23 \cdot 10^{-3}$	$1.79 \cdot 10^{308}$	52 bits	11 bits

Representación de números reales en *punto flotante*

- Los casos $e_7 \ldots e_0$ = 0 o 255 son especiales
- El 0 se representa como 000…0 (solo ceros)
- Hay una representación para el NaN: *not a number*

Tipo	Espacio	Precisión	Min magnitud	Máx magnitud	Mantiza	Exp
float	32 bits (4 bytes)	7 dígitos	$1.18 \cdot 10^{-38}$	$3.4 \cdot 10^{38}$	23 bits	8 bits
double	64 bits (8 bytes)	15 dígitos	$2.23 \cdot 10^{-3}$	$1.79 \cdot 10^{308}$	52 bits	11 bits

- Cuidado: 0.1 no es representable de manera exacta

Números periódicos en base 10 o base 2 no son representables en binarios

Representación de números reales en *punto flotante*

- Los casos $e_7 \ldots e_0$ = 0 o 255 son especiales
- El 0 se representa como 000…0 (solo ceros)
- Hay una representación para el NaN: *not a number*

Tipo	Espacio	Precisión	Min magnitud	Máx magnitud	Mantiza	Exp
float	32 bits (4 bytes)	7 dígitos	$1.18 \cdot 10^{-38}$	$3.4 \cdot 10^{38}$	23 bits	8 bits
double	64 bits (8 bytes)	15 dígitos	$2.23 \cdot 10^{-3}$	$1.79 \cdot 10^{308}$	52 bits	11 bits

- Cuidado: 0.1 no es representable de manera exacta
- Cuidado: ¡Nunca escriba x==y! Use $|x\text{-}y| < \varepsilon$

Representación de números reales en *punto flotante*

- Los casos $e_7 \ldots e_0$ = 0 o 255 son especiales
- El 0 se representa como 000…0 (solo ceros)
- Hay una representación para el NaN: *not a number*

Tipo	Espacio	Precisión	Min magnitud	Máx magnitud	Mantiza	Exp
float	32 bits (4 bytes)	7 dígitos	$1.18 \cdot 10^{-38}$	$3.4 \cdot 10^{38}$	23 bits	8 bits
double	64 bits (8 bytes)	15 dígitos	$2.23 \cdot 10^{-3}$	$1.79 \cdot 10^{308}$	52 bits	11 bits

- Cuidado: 0.1 no es representable de manera exacta
- Cuidado: ¡Nunca escriba x==y! Use |x-y| < ε

Debido a las imprecisiones del cálculo x será aproximadamente y, no igual en caso de números reales

Representación de números reales en *punto flotante*

- Los casos $e_7 \ldots e_0$ = 0 o 255 son especiales
- El 0 se representa como 000…0 (solo ceros)
- Hay una representación para el NaN: *not a number*

Tipo	Espacio	Precisión	Min magnitud	Máx magnitud	Mantiza	Exp
float	32 bits (4 bytes)	7 dígitos	$1.18 \cdot 10^{-38}$	$3.4 \cdot 10^{38}$	23 bits	8 bits
double	64 bits (8 bytes)	15 dígitos	$2.23 \cdot 10^{-3}$	$1.79 \cdot 10^{308}$	52 bits	11 bits

- Cuidado: 0.1 no es representable de manera exacta
- Cuidado: ¡Nunca escriba x==y! Use |x-y| < ε
- Ud. encontrará más detalles en la Wikipedia

Representación de caracteres

- Se usa la codificación ASCII
- Las constantes 'a' 'b' 'c' ... 'z' son 97 98 99 ... 122
- 'A' 'B' 'C' ... 'Z' son 65 66 67 ... 90
- '0' '1' '2' ... '9' son 48 49 50 ... 57
- '!' es 33 '"' es 34 ... etc.
- '\n' es 10
- Note que 'A'+1 es 'B' y que '0'+4 es '4'

Un char se escribe entre comillas simples (‘’), por ejemplo ‘A’. El computador interpreta esto internamente en código ASCII

Representación de caracteres

- Se usa la codificación ASCII
- Las constantes 'a' 'b' 'c' ... 'z' son 97 98 99 ... 122
- 'A' 'B' 'C' ... 'Z' son 65 66 67 ... 90
- '0' '1' '2' ... '9' son 48 49 50 ... 57
- '!' es 33 '"' es 34 ... etc.
- '\n' es 10
- Note que 'A'+1 es 'B' y que '0'+4 es '4'

ASCII - Hex - Binary
Conversion Chart

ASCII	Hex	Binary	ASCII	Hex	Binary	ASCII	Hex	Binary	ASCII	Hex	Binary
NUL	00	00000000	SP	20	00100000	@	40	01000000	`	60	01100000
SOH	01	00000001	!	21	00100001	A	41	01000001	a	61	01100001
STX	02	00000010	"	22	00100010	B	42	01000010	b	62	01100010
ETX	03	00000011	#	23	00100011	C	43	01000011	c	63	01100011
EOT	04	00000100	$	24	00100100	D	44	01000100	d	64	01100100
ENQ	05	00000101	%	25	00100101	E	45	01000101	e	65	01100101
ACK	06	00000110	&	26	00100110	F	46	01000110	f	66	01100110
BEL	07	00000111	'	27	00100111	G	47	01000111	g	67	01100111
BS	08	00001000	(	28	00101000	H	48	01001000	h	68	01101000
HT	09	00001001	)	29	00101001	I	49	01001001	i	69	01101001
LF	0A	00001010	*	2A	00101010	J	4A	01001010	j	6A	01101010
VT	0B	00001011	+	2B	00101011	K	4B	01001011	k	6B	01101011
FF	0C	00001100	,	2C	00101100	L	4C	01001100	l	6C	01101100
CR	0D	00001101	-	2D	00101101	M	4D	01001101	m	6D	01101101
SO	0E	00001110	.	2E	00101110	N	4E	01001110	n	6E	01101110
SI	0F	00001111	/	2F	00101111	O	4F	01001111	o	6F	01101111
DLE	10	00010000	0	30	00110000	P	50	01010000	p	70	01110000
DC1	11	00010001	1	31	00110001	Q	51	01010001	q	71	01110001
DC2	12	00010010	2	32	00110010	R	52	01010010	r	72	01110010
DC3	13	00010011	3	33	00110011	S	53	01010011	s	73	01110011
DC4	14	00010100	4	34	00110100	T	54	01010100	t	74	01110100
NAK	15	00010101	5	35	00110101	U	55	01010101	u	75	01110101
SYN	16	00010110	6	36	00110110	V	56	01010110	v	76	01110110
ETB	17	00010111	7	37	00110111	W	57	01010111	w	77	01110111
CAN	18	00011000	8	38	00111000	X	58	01011000	x	78	01111000
EM	19	00011001	9	39	00111001	Y	59	01011001	y	79	01111001
SUB	1A	00011010	:	3A	00111010	Z	5A	01011010	z	7A	01111010
ESC	1B	00011011	;	3B	00111011	[	5B	01011011	{	7B	01111011
FS	1C	00011100	<	3C	00111100	\	5C	01011100	|	7C	01111100
GS	1D	00011101	=	3D	00111101	]	5D	01011101	}	7D	01111101
RS	1E	00011110	>	3E	00111110	^	5E	01011110	~	7E	01111110
US	1F	00011111	?	3F	00111111	_	5F	01011111	DEL	7F	01111111

Representación de caracteres

- Se usa la codificación ASCII
- Las constantes 'a' 'b' 'c' ... 'z' son 97 98 99 ... 122
- 'A' 'B' 'C' ... 'Z' son 65 66 67 ... 90
- '0' '1' '2' ... '9' son 48 49 50 ... 57
- '!' es 33 '"' es 34 ... etc.
- '\n' es 10
- Note que 'A'+1 es 'B' y que '0'+4 es '4'

ASCII - Hex - Binary
Conversion Chart

ASCII	Hex	Binary	ASCII	Hex	Binary	ASCII	Hex	Binary	ASCII	Hex	Binary
NUL	00	00000000	SP	20	00100000	@	40	01000000	`	60	01100000
SOH	01	00000001	!	21	00100001	A	41	01000001	a	61	01100001
STX	02	00000010	"	22	00100010	B	42	01000010	b	62	01100010
ETX	03	00000011	#	23	00100011	C	43	01000011	c	63	01100011
EOT	04	00000100	$	24	00100100	D	44	01000100	d	64	01100100
ENQ	05	00000101	%	25	00100101	E	45	01000101	e	65	01100101
ACK	06	00000110	&	26	00100110	F	46	01000110	f	66	01100110
BEL	07	00000111	'	27	00100111	G	47	01000111	g	67	01100111
BS	08	00001000	(	28	00101000	H	48	01001000	h	68	01101000
HT	09	00001001	)	29	00101001	I	49	01001001	i	69	01101001
LF	0A	00001010	*	2A	00101010	J	4A				
VT	0B	00001011	+	2B	00101011	K	4B				
FF	0C	00001100	,	2C	00101100	L	4C				
CR	0D	00001101	-	2D	00101101	M	4D				
SO	0E	00001110	.	2E	00101110	N	4E				
SI	0F	00001111	/	2F	00101111	O	4F				
DLE	10	00010000	0	30	00110000	P	50				
DC1	11	00010001	1	31	00110001	Q	51				
DC2	12	00010010	2	32	00110010	R	52				
DC3	13	00010011	3	33	00110011	S	53				
DC4	14	00010100	4	34	00110100	T	54				
NAK	15	00010101	5	35	00110101	U	55				
SYN	16	00010110	6	36	00110110	V	56				
ETB	17	00010111	7	37	00110111	W	57	01010111	w	77	01110111
CAN	18	00011000	8	38	00111000	X	58	01011000	x	78	01111000
EM	19	00011001	9	39	00111001	Y	59	01011001	y	79	01111001
SUB	1A	00011010	:	3A	00111010	Z	5A	01011010	z	7A	01111010
ESC	1B	00011011	;	3B	00111011	[	5B	01011011	{	7B	01111011
FS	1C	00011100	<	3C	00111100	\	5C	01011100	|	7C	01111100
GS	1D	00011101	=	3D	00111101	]	5D	01011101	}	7D	01111101
RS	1E	00011110	>	3E	00111110	^	5E	01011110	~	7E	01111110
US	1F	00011111	?	3F	00111111	_	5F	01011111	DEL	7F	01111111

El primer bit siempre es 0. Utiliza una representación binaria de 7 bits

Contacto

- Ucursos
- Correo: alexandra.ibarra.c@gmail.com
- Telegram / Whatsapp: +569 74418244