**Representación de la información**

**Clasificación de la información**

* **Números:** Enteros, racionales, reales…
  + Su representación se define por su rango y su precisión
  + Representación por punto fijo es para enteros, punto flotante para racionales y reales.
* **Caracteres:**
  + Utilizan códigos especiales (ASCII, UTF-8)
  + Problemas con diferentes alfabetos
* **Instrucciones:**
  + No tienen signo ni valor
  + Formadas por varios campos independientes y codificados por separado
    - Algunos campos serán números (registro) y otros secuencias de bits con significados particulares (código de operación)

**Números en punto fijo**

* Se utiliza un sistema de representación **posicional:** El número es una cadena de dígitos, cada uno multiplicado por su peso según su posición en la cadena.
  + El factor de peso es una potencia de la base del sistema de representación (en binario es 2)
* Se pueden representar decimales utilizando las **potencias negativas** de la base, con precisión limitada.
* El prefijo **0x** indica un número hexadecimal. 0b indica número en binario.
* **Limitaciones:** Rango limitado[[1]](#footnote-0), y se debe decidir previamente cuantos bits se dedican a la parte entera y cuantos a la parte decimal.

**Codificación de números binarios con signo**

* **Signo-magnitud:** El bit más a la izquierda (MSB) es el bit del signo, 1 negativo y 0 positivo.
  + Los bits de magnitud son el número sin signo.
  + Con n bits se pueden representar números en el intervalo {-(2n-1 -1), (2n-1 -1)}
  + Ej.: -16 = 110000
* **Complemento a 1:** El bit más a la izquierda es el bit del signo. Además, los números negativos son el complemento a 1 del correspondiente número positivo: intercambia los 0 y los 1.
  + Mismo intervalo que signo-magnitud
  + Dos representaciones posibles para el 0: 0000(+0) o 1111(-0)
  + Ej.: -16 = 101111
* **Complemento a 2:** El bit a la izquierda es el signo, los números negativos son el complemento a 2 del positivo:
  + Para calcular el complemento a 2 de un binario N de n bits, se calcula la diferencia 2^n-N y se toman los n bits más a la derecha.
    - Se realiza el complemento a 1 y suma 1 bit. Si el número está en un extremo, se suprime un bit a la izquierda. Ej.: -16 = ~~1~~**10000**
  + Intervalo: {-(2n-1), (2n-1 -1)} (**1 número negativo más** que signomagnitud)
  + 0 sólo tiene una representación, 0. Non existe o -0.
* **NOTA:** Para C1 y C2, los positivos son iguales que en signo magnitud.

**Tipos básicos de enteros en C**

| Tipo | Bits mínimos | Rango | Especificador |
| --- | --- | --- | --- |
| bool | 1 | [0,1] | %hhu |
| char | 8 | Depende | %c |
| signed char | 8 | [-127,127] | %c |
| unsigned char | 8 | [0,255] | %c |
| short | 16 | [-32767,32767] | %hi |
| unsigned short | 16 | [0,65535] | %hu |
| int | 16 | [-32767,32767] | %i o %d |
| unsigned int | 16 | [0,65535] | %u |
| long | 32 | [−(231 − 1), 231 − 1] | %li |
| unsigned long | 32 | [0, 232 − 1] | %lu |
| long long | 64 | [−(263 − 1), 263 − 1] | %lli |
| unsigned long long | 64 | [0, 264 − 1] | %llu |

**Punto flotante**

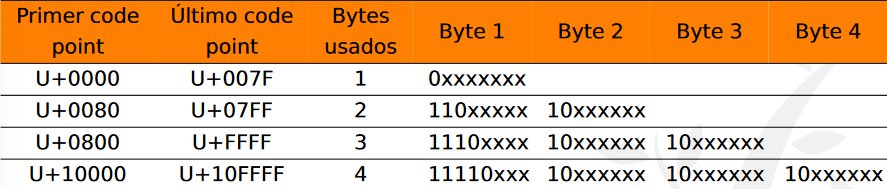
* Representación basada en: **Signo (s), exponente (e) y mantisa (M)**
  + Valor : (-1)S x M x be [siendo b la base]
* Se utilizará el estándar **IEEE 754.**
* Formatos (solo se estudiarán los 2 primeros):
  + Simple precisión: 32 bits, float en C
    - 1 bit de signo, 8 de exponente, 23 de mantisa normalizada
  + Doble precisión: 64 bits, double en C
    - 1 bit de signo, 11 de exponente, 52 de mantisa normalizada
  + Precisión extendida: 80 bits, long double en C. No siempre implementado
  + Cuádruple precisión: 128 bits, tipo \_\_float128 en gcc v>=4.6
* Al leer un número (ej: 11,011x20011) cambiamos el exponente para que en la parte entera quede **solo un 1**.
  + El ejemplo pasa a ser 1,1011x20100
  + De esta forma, sabemos que siempre habrá un 1 en la parte entera, y no es necesario almacenar este bit.[[2]](#footnote-1)
* Se codifica en **exceso:** el exponente que se guarda es expreal + exceso
  + **exceso** es 127 en simple precisión y 1023 si doble (**2n-1-1**, siendo n los bits de exponente)
  + De esta forma, el exponente representado será siempre positivo, y estará en [1,254] o [1,2048] según precisión. [1,2n]
* Tras estos cambios, el valor del número será **(-1)s x 1.M x 2e-exceso**
* **Ejemplo:** Valor decimal de 0x55F20000 en precisión simple.
  + Se pasa a binario: 0101 0101 1101 0010 0000 0000 0000 0000
    - Signo: 0 → positivo
    - Exponente: 101010112 = 17110
      * Exponente real: 171-127 = 44
    - Mantisa: 1,1010012 = 1,64062510
    - Valor: (-1)0 x 1.640625 x 244 **= 2.88621602 x 1013**
* Los exponentes 0 y 255/2047[[3]](#footnote-2) se reservan para **números especiales:**
  + Exp 0 y mantisa 0 representa el **0**.
    - Si la mantisa es 0 y el exponente no, el número será 2-126 y no 0, pues se debe añadir el 1 asumido
  + Exp 0 y mantisa ≠ 0 representa un valor **denormal**
  + Exp 255 y mantisa 0 representa
  + Exp 255 y mantisa ≠ 0 representa Not a Number (+-NaN)

**Números denormales**

* Números no normalizados próximos a 0.
* Su valor es **(-1)s x 0.M x 21-exceso**
* El exponente será siempre -126 o -1022 según representación.
* **Ejemplo:** 0 00000000 00010100000000000000000
  + Valor = (-1)0 x 0.000101 x 2-126 = 1.01 x 2-130 = 9.18 x 10-40

**Representación de caracteres**

* **ASCII**: 7 bits → 128 caracteres. Sólo incluye alfabeto inglés
  + ISO/IEC 646: Similar a ASCII, pero incluye variantes para algunos países
  + **Extended ASCII**: 8 bits → 256 caracteres. ASCII + otros 128 caracteres
    - ISO/IEC 8859: 15 estándares para diferentes idiomas
    - Windows-1252, Mac OS Roman: lol
* **Unicode:** Cada carácter está representado por un **code-point** en hexadecimal, desde U+000016 hasta U+10FFFF16 → **1.114.112 códigos**
  + Los code-points se organizan en 17 planos de 216 code-points cada uno. Los planos se organizan en charts.
  + **UTF-8:** Símbolos de longitud variable, utiliza ASCII de 7 bits.
    - El byte 1, además de tener datos, indica el **nº de bits de longitud**:



* + - Ejemplo: El símbolo € es U+20AC (3 bytes). Su UTF-8 será: 11100010 10000010 10101100 → 0xE282AC
  + **UTF-16**: Símbolos de longitud variable: 1 o 2 palabras de 16 bits (2 o 4 bytes), siendo la mayoría de caracteres de 16 bits.
  + **UTF-32:** Exactamente 32 bits por carácter.
    - Más sencillo pero menos eficiente en almacenamiento

1. [0, 2n-1] sin signo. Si se usa C2, [-2n-1, +(2n-1-1] [↑](#footnote-ref-0)
2. Por eso los bits en los casos previos suman a 63 y no 64 [↑](#footnote-ref-1)
3. 255 si precisión simple, 2047 si precisión doble (basicamente todo 1s no exponente) [↑](#footnote-ref-2)