– Typeset by GMNI & FoilTEX **–**

Lenguajes de programación en ingeniería:
CONCEPTOS GENERALES
SOBRE ALMACENAMIENTO DE INFORMACIÓN

F. Navarrina, J. París & GMNI

GMNI — GRUPO DE MÉTODOS NUMÉRICOS EN INGENIERÍA

Escuela Técnica Superior de Ingenieros de Caminos, Canales y Puertos
Universidad de A Coruña, España

e-mail: {fermin.navarrina,jose.paris}@udc.es
página web: http://caminos.udc.es/gmni

ÍNDICE

- Bit
- Byte
- Código ASCII
- Códigos ASCII extendidos
- Unidades de medida de información
- Archivo
- Palabra
- Tipos de Variables
- Almacenamiento en coma flotante: advertencias

Bit (I)

BIT

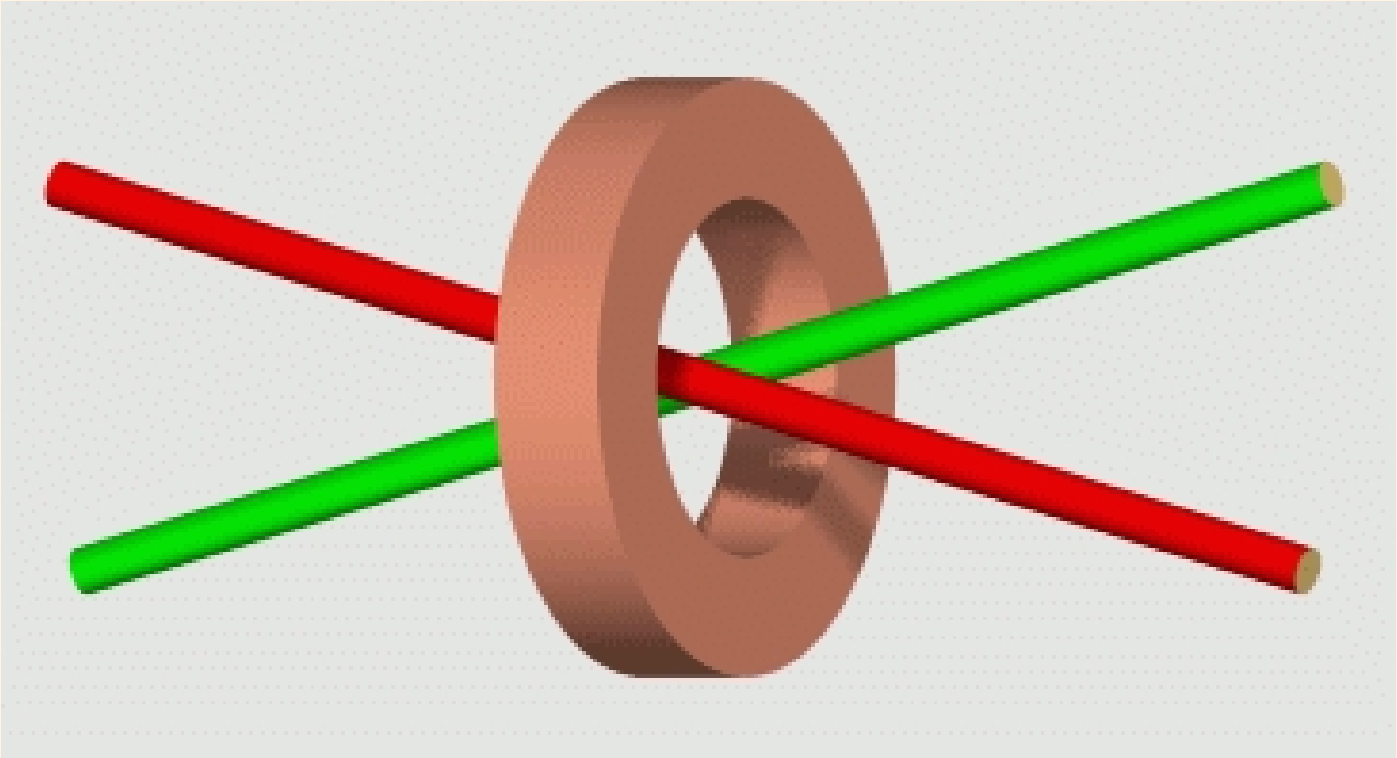
- Componente de memoria con 2 estados posibles.
- Elemento básico de la codificación binaria (de números, caracteres, etc.)
- Variable capaz de almacenar 2 números enteros

$$bit \in \{0, 1\}.$$

- Unidad de medida de información.

Bit (II)

- Origen: **NÚCLEOS DE FERRITA**



Esquema de un componente elemental de memoria de ferrita:
Núcleo de ferrita, hilo de escritura e hilo de lectura.
(Fuente: <http://www.lsi.us.es/~rovayo/ferrita/ferrita_pf.html>, M. Rovayo, Univ. de Sevilla)

Byte (I)

BYTE

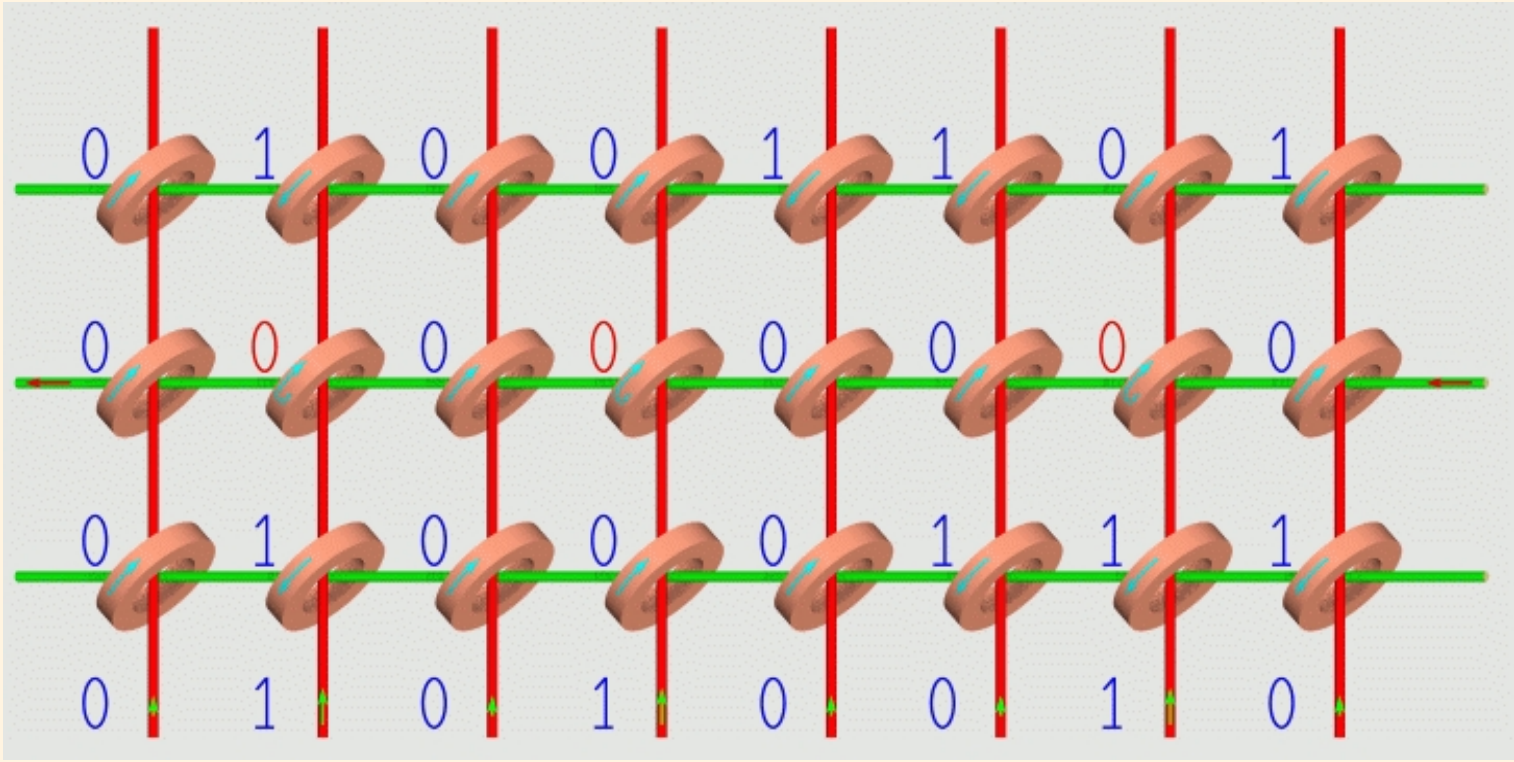
- Conjunto de 8 bits con $2^8 = 256$ estados posibles.
- Memoria necesaria para codificar el conjunto habitual de caracteres (letras mayúsculas y minúsculas, signos de puntuación, símbolos matemáticos más habituales, caracteres de control de impresión (CR, LF), caracteres latinos, etc.
- Variable capaz de almacenar $2^8 = 256$ números enteros

$$byte \in \{0, \ldots, 255\}.$$

- Unidad de medida de información.

Byte (II)

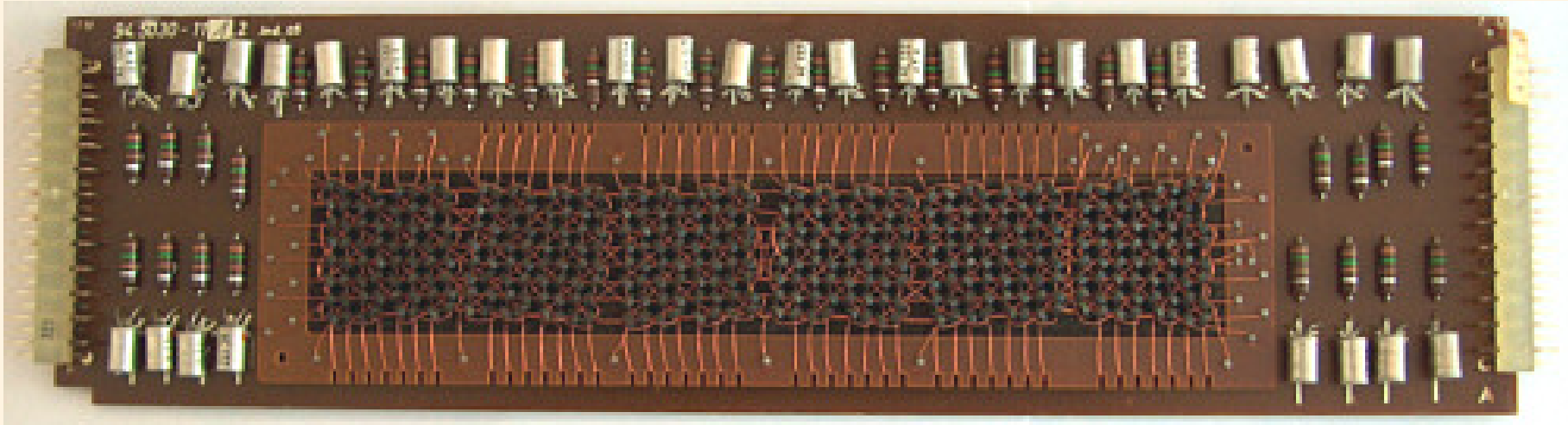
- Origen: **MEMORIAS DE FERRITA**



Esquema de un componente elemental de memoria de ferrita:
Núcleo de ferrita, hilo de escritura e hilo de lectura.
(Fuente: <http://www.lsi.us.es/~rovayo/ferrita/ferrita_pf.html>, M. Rovayo, Univ. de Sevilla)

Byte (IIIa)

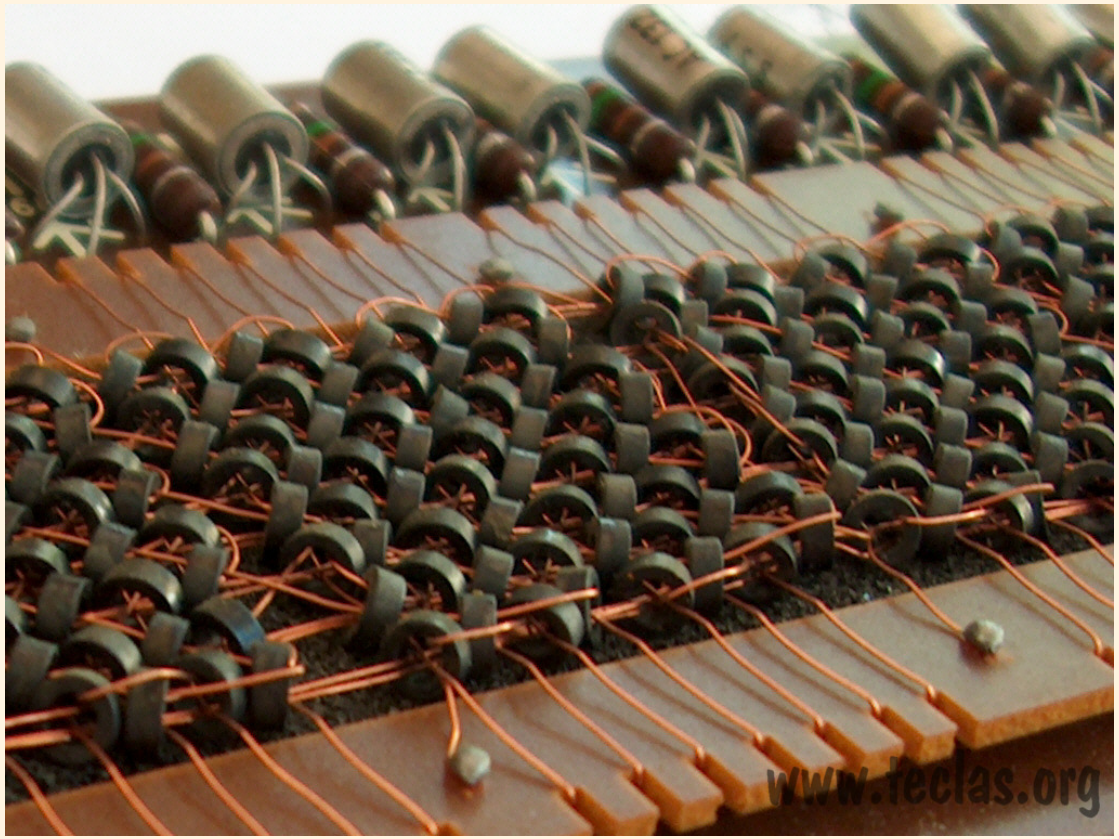
- Ejemplo:



Memoria de una calculadora Olympia RAE 4/30-3 (1966 aprox.).
(Fuente: <http://www.teclas.org>)

Byte (IIIb)

- Ejemplo:



Memoria de una calculadora Olympia RAE 4/30-3 (1966 aprox.). Detalle.
(Fuente: <http://www.teclas.org>)

Byte (IIIc)

- Ejemplo:

Calculadora Olympia RAE 4/30-3 (1966 aprox.).
(Fuente: <http://www.teclas.org>)

Código ASCII (I)

CÓDIGO ASCII (*)

- Creado en 1963 por **ASA** (**), actualmente **ANSI** (***).
  - Desarrollado a partir de los códigos telegráficos usados en los Estados Unidos.
  - 7 bits para codificar $2^7 = 128$ caracteres (caracteres imprimibles del alfabeto inglés y códigos de control).
  - 1 bit adicional se utilizaba inicialmente para control de paridad (detección de errores).
- Revisado en 1967 (inclusión de minúsculas) y en 1986.
- Base de códigos posteriores, más completos (letras acentuadas, otros alfabetos).

(*) ASCII $\Longrightarrow$ American Standard Code for Information Interchange.

(**) ASA $\Longrightarrow$ American Standards Association.

(***) ANSI $\Longrightarrow$ American National Standards Institute.

Código ASCII (IIa)

▶ **CÓDIGO ASCII**: (caracteres 0—31)

BIN.	HEX.	DEC.	REPRESENTACIÓN
0000 0000	0 0	0	NUL (null character)
0000 0001	0 1	1	SOH (start of header)
0000 0010	0 2	2	STX (start of text)
0000 0011	0 3	3	ETX (end of text)
0000 0100	0 4	4	EOT (end of transm.)
0000 0101	0 5	5	ENQ (enquiry)
0000 0110	0 6	6	ACK (acknowledgement)
0000 0111	0 7	7	BEL (bell)
0000 1000	0 8	8	BS (back space)
0000 1001	0 9	9	HT (horizontal tab.)
0000 1010	0 A	10	LF (line feed)
0000 1011	0 B	11	VT (vertical tab.)
0000 1100	0 C	12	FF (form feed)
0000 1101	0 D	13	CR (carriage return)
0000 1110	0 E	14	SO (shift out)
0000 1111	0 F	15	SI (shift in)

BIN.	HEX.	DEC.	REPRESENTACIÓN
0001 0000	1 0	16	DLE (data link escape)
0001 0001	1 1	17	DC1 (device control 1)
0001 0010	1 2	18	DC2 (device control 2)
0001 0011	1 3	19	DC3 (device control 3)
0001 0100	1 4	20	DC4 (device control 4)
0001 0101	1 5	21	NAK (negative acknowl.)
0001 0110	1 6	22	SYN (synchronous idle)
0001 0111	1 7	23	ETB (end of tr. block)
0001 1000	1 8	24	CAN (cancel)
0001 1001	1 9	25	EM (end of medium)
0001 1010	1 A	26	SUB (substitute)
0001 1011	1 B	27	ESC (escape)
0001 1100	1 C	28	FS (file separator)
0001 1101	1 D	29	GS (group separator)
0001 1110	1 E	30	RS (record separator)
0001 1111	1 F	31	US (unit separator)

Código ASCII (IIb)

▶ **CÓDIGO ASCII**: (caracteres 32—63)

BIN.	HEX.	DEC.	REPRESENTACIÓN
0010 0000	20	32	SP (espacio)
0010 0001	21	33	!
0010 0010	22	34	”
0010 0011	23	35	#
0010 0100	24	36	$
0010 0101	25	37	%
0010 0110	26	38	&
0010 0111	27	39	’
0010 1000	28	40	(
0010 1001	29	41	)
0010 1010	2A	42	*
0010 1011	2B	43	+
0010 1100	2C	44	,
0010 1101	2D	45	−
0010 1110	2E	46	.
0010 1111	2F	47	/

BIN.	HEX.	DEC.	REPRESENTACIÓN
0011 0000	30	48	0
0011 0001	31	49	1
0011 0010	32	50	2
0011 0011	33	51	3
0011 0100	34	52	4
0011 0101	35	53	5
0011 0110	36	54	6
0011 0111	37	55	7
0011 1000	38	56	8
0011 1001	39	57	9
0011 1010	3A	58	:
0011 1011	3B	59	;
0011 1100	3C	60	<
0011 1101	3D	61	=
0011 1110	3E	62	>
0011 1111	3F	63	?

Código ASCII (IIc)

▶ **CÓDIGO ASCII**: (caracteres 64—95)

BIN.	HEX.	DEC.	REPRESENTACIÓN
0100 0000	40	64	@
0100 0001	41	65	A
0100 0010	42	66	B
0100 0011	43	67	C
0100 0100	44	68	D
0100 0101	45	69	E
0100 0110	46	70	F
0100 0111	47	71	G
0100 1000	48	72	H
0100 1001	49	73	I
0100 1010	4A	74	J
0100 1011	4B	75	K
0100 1100	4C	76	L
0100 1101	4D	77	M
0100 1110	4E	78	N
0100 1111	4F	79	O

BIN.	HEX.	DEC.	REPRESENTACIÓN
0101 0000	50	80	P
0101 0001	51	81	Q
0101 0010	52	82	R
0101 0011	53	83	S
0101 0100	54	84	T
0101 0101	55	85	U
0101 0110	56	86	V
0101 0111	57	87	W
0101 1000	58	88	X
0101 1001	59	89	Y
0101 1010	5A	90	Z
0101 1011	5B	91	[
0101 1100	5C	92	\
0101 1101	5D	93	]
0101 1110	5E	94	^
0101 1111	5F	95	_

Código ASCII (IId)

▶ **CÓDIGO ASCII**: (caracteres 96—127)

BIN.	HEX.	DEC.	REPRESENTACIÓN
0110 0000	60	96	‘
0110 0001	61	97	a
0110 0010	62	98	b
0110 0011	63	99	c
0110 0100	64	100	d
0110 0101	65	101	e
0110 0110	66	102	f
0110 0111	67	103	g
0110 1000	68	104	h
0110 1001	69	105	i
0110 1010	6A	106	j
0110 1011	6B	107	k
0110 1100	6C	108	l
0110 1101	6D	109	m
0110 1110	6E	110	n
0110 1111	6F	111	o

BIN.	HEX.	DEC.	REPRESENTACIÓN
0111 0000	70	112	p
0111 0001	71	113	q
0111 0010	72	114	r
0111 0011	73	115	s
0111 0100	74	116	t
0111 0101	75	117	u
0111 0110	76	118	v
0111 0111	77	119	w
0111 1000	78	120	x
0111 1001	79	121	y
0111 1010	7A	122	z
0111 1011	7B	123	{
0111 1100	7C	124	|
0111 1101	7D	125	}
0111 1110	7E	126	~
0111 1111	7F	127	DEL (delete)

Código ASCII (Ile)

- **CÓDIGO ASCII**: (caracteres 128—255)
  - Su primer bit es igual a 1.
  - En realidad, las correspondientes secuencias de bits **NO forman parte del Código ASCII**.
  - Dan lugar a extensiones compatibles del Código ASCII.

Códigos ASCII extendidos (I)

CÓDIGOS ASCII EXTENDIDOS

- **CÓDIGO ISO-8859-1 (o ISO Latin 1)**: (*)
  - 8 bits para codificar $2^8 = 256$ caracteres (caracteres imprimibles del alfabeto latino y códigos de control).
  - Los primeros 128 códigos (primer bit = 0) coinciden con los ASCII.
  - Los restantes 128 códigos (primer bit = 1) permiten codificar caracteres adicionales:
    - letras de otros alfabetos (ñ, Ñ, ç, Ç,...)
    - letras acentudadas (á, à, â,...,ü,...)
    - otros caracteres (©, ¡, ¿,...)
- **CÓDIGOS UNICODE**:
  - Utilizan más de 1 byte para codificar (casi) todos los alfabetos.
  - Los primeros 256 códigos coinciden con los ISO-8859-1.

(*) ISO $\Longrightarrow$ International Organization for Standardization.

Unidades de medida de información (I)

UNIDADES DE MEDIDA DE INFORMACIÓN

- **Unidades Básicas**:
  - 1 b $\Longrightarrow 1$ bit
  - 1 B $\Longrightarrow 1$ byte
- **Otras Unidades**:
  - **PREFIJO** + Unidad Básica

Unidades de medida de información (IIa)

- **Prefijos del Sistema Internacional**: (*)

SÍMBOLO	NOMBRE	SIGNIFICADO	EQUIVALENCIA	
k	kilo	10^3	1.000	$= 10^3$
M	mega	10^6	1.000.000	$= 10^6$
G	giga	10^9	1.000.000.000	$= 10^9$
T	tera	10^{12}	1.000.000.000.000	$= 10^{12}$
P	peta	10^{15}	1.000.000.000.000.000	$= 10^{15}$
E	exa	10^{18}	1.000.000.000.000.000.000	$= 10^{18}$
Z	zetta	10^{21}	1.000.000.000.000.000.000.000	$= 10^{21}$
Y	yotta	10^{24}	1.000.000.000.000.000.000.000.000	$= 10^{24}$

(*) Uso habitual en telecomunicaciones (velocidad de transmisión: bps/Bps=bits/bytes por segundo).

Unidades de medida de información (IIb)

- **Prefijos Tradicionales**: (*)

SÍMBOLO	NOMBRE	SIGNIFICADO	EQUIVALENCIA	
k,K	kilo	2^{10}	1.024	$\approx 10^3$
M	mega	2^{20}	1.048.576	$\approx 10^6$
G	giga	2^{30}	1.073.741.824	$\approx 10^9$
T	tera	2^{40}	1.099.511.627.776	$\approx 10^{12}$
P	peta	2^{50}	1.125.899.906.842.624	$\approx 10^{15}$
E	exa	2^{60}	1.152.921.504.606.846.976	$\approx 10^{18}$
Z	zetta	2^{70}	1.180.591.620.717.411.303.424	$\approx 10^{21}$
Y	yotta	2^{80}	1.208.925.819.614.629.174.706.176	$\approx 10^{24}$

  - Problemas:
    - Los errores respecto a los prefijos del Sistema Internacional son cada vez mayores.
    - Ambigüedad: ¿cuánto cabe en un disco de 320 GB?

(*) Uso habitual en informática (capacidad de almacenamiento en memoria, direccionamiento).

Unidades de medida de información (IIc)

- **Nuevos prefijos binarios, IEEE Standard 1541-2002**: (*)

SÍMBOLO	NOMBRE	SIGNIFICADO	EQUIVALENCIA	
Ki	kibi	2^{10}	1.024	$\approx 10^3$
Mi	mebi	2^{20}	1.048.576	$\approx 10^6$
Gi	gibi	2^{30}	1.073.741.824	$\approx 10^9$
Ti	tebi	2^{40}	1.099.511.627.776	$\approx 10^{12}$
Pi	pebi	2^{50}	1.125.899.906.842.624	$\approx 10^{15}$
Ei	exbi	2^{60}	1.152.921.504.606.846.976	$\approx 10^{18}$
Zi	zebi	2^{70}	1.180.591.620.717.411.303.424	$\approx 10^{21}$
Yi	yobi	2^{80}	1.208.925.819.614.629.174.706.176	$\approx 10^{24}$

(*) IEEE $\Longrightarrow$ Institute of Electrical and Electronics Engineers.

Archivo (I)

ARCHIVO

- Ristra de BYTES
  almacenada en una unidad de memoria de cualquier tipo.
- Tipos:
  - **ARCHIVOS DE TEXTO / ASCII**

programas FORTRAN:	(***.f**, ***.for**),
programas C:	(***.c**),
archivos TXT (O CON FORMATO):	(***.txt**),
programas de comandos:	(***.bat**),
...	

  - **ARCHIVOS BINARIOS / DE TIPO IMAGEN**

programas OBJETO:	(***.o**), ***.obj**
programas EJECUTABLES:	(***.exe**, sin extensión),
archivos SIN FORMATO:	(***.xls**, ***.dwg**, ***.jpg**)...
...	

Archivo (II)

▶ ARCHIVOS DE TEXTO /ASCII

- Los bytes que contienen representan (exclusivamente) códigos ASCII de
  - ▷ caracteres imprimibles (letras, números, símbolos) y de
  - ▷ códigos de control de "impresión" (CR, LF, FF, . . .).
- Los "controles de carro" (CR LF) crean una organización **por líneas**.
- Ejemplos: (véase la carpeta EjemplosDeArchivosDeTextoYBinarios)
  - ▷ El archivo hello.c es un archivo de texto que contiene un programa muy sencillo escrito en C.
  - ▷ El archivo hello.c.txt muestra los bytes que forman el archivo hello.c,
  - ▷ y el archivo hello.c.b.txt muestra los correspondientes bits.

Archivo (III)

▶ ARCHIVOS BINARIOS / DE TIPO IMAGEN

- Los bytes que contienen pueden tomar cualquier valor (entre 0 y 255):
  - Algunos pueden representar códigos ASCII de caracteres no de códigos de impresión (texto de títulos, menús de opciones, preguntas o respuestas, mensajes de error o advertencia, etc.)
  - Con carácter general tendrán otro significado (instrucciones, datos, . . .) que depende del sistema operativo o de la aplicación que use el archivo.
- No están organizados por líneas.
- Ejemplos: (véase la carpeta EjemplosDeArchivosDeTextoYBinarios)
  - El archivo hello.o es un archivo binario que contiene el programa objeto que se obtiene al compilar hello.c.
  - El archivo hello.o.txt muestra los bytes que forman el archivo hello.o,
  - y el archivo hello.o.b.txt muestra los correspondientes bits.
  - El archivo hello.exe es un archivo binario que contiene el programa ejecutable que se obtiene al linkar hello.o.
  - El archivo hello.exe.txt muestra los bytes que forman el archivo hello.exe,
  - y el archivo hello.exe.b.txt muestra los correspondientes bits.

Palabra (I)

WORD (palabra)

- Grupos de n bits (con 2^n estados posibles)
  con los que el **procesador** es capaz de realizar una operación elemental
  (comparación, suma, etc.)
  en un ciclo de reloj.
- Típicamente:
  - $n = 8$ [70's] $\Longrightarrow$ Intel 8008, Z80 (Spectrum), ...
  - $n = 16$ [80's] $\Longrightarrow$ Intel 8086–80286 (IBM-PC/XT/AT), AMD Am386, PDP-11, ...
  - $n = 32$ [90's] $\Longrightarrow$ Intel 80386–80486–Pentium I/II/III/4, AMD K5–K6, VAX,...
  - $n = 64$ [00's] $\Longrightarrow$ Intel Core 2, AMD Athlon 64, AXP,...
  - $n = 128$ [??'s] $\Longrightarrow$ Videojuegos (de momento).
- La longitud de palabra determina el formato preferente de almacenamiento de datos numéricos (enteros y reales).

Palabra (II)

- Variable capaz de almacenar 2^n números enteros (sin signo)

$$word \in \{0, \ldots, 2^n - 1\}.$$

- La longitud de palabra (n bits) determina la capacidad de direccionamiento directo en memoria... (*)

  Típicamente:

  - $n = 8$ [70's] $\Longrightarrow$ 256 B
  - $n = 16$ [80's] $\Longrightarrow$ 64 KiB
  - $n = 32$ [90's] $\Longrightarrow$ 4 GiB
  - $n = 64$ [00's] $\Longrightarrow$ 16 EiB $\approx 16 \cdot 10^9$ GiB
  - $n = 128$ [??'s] $\Longrightarrow$ $256 \cdot 10^{12}$ YiB $\approx 256 \cdot 10^{27}$ GiB

(*) Algunos procesadores utilizan varias palabras lo que permite extender su capacidad de direccionamiento.

Palabra (III)

- Ejemplo:

Detalle del Panel frontal de un ordenador HP 2116-B (1966 aprox.):
procesador de 16 bits, circuitos integrados, 16 KiB de memoria (expandible a 64 KiB), programable en Ensamblador y FORTRAN.
Obsérvense los interruptores que permiten introducir palabras en binario. (Fuente: <http://www.hpmuseum.net>)

Tipos de Variables (I)

TIPOS ELEMENTALES

- **VARIABLES ENTERAS: CON SIGNO ($+$, $-$) Y SIN SIGNO ($+$)**
  - **Enteros de 8 bits** $\longrightarrow$ 1 byte
  - **Enteros de 16 bits** $\longrightarrow$ 2 bytes
  - **Enteros de 32 bits** $\longrightarrow$ 4 bytes
  - **Enteros de 64 bits** $\longrightarrow$ 8 bytes
- **VARIABLES REALES (en coma flotante)**
  - **Reales de 32 bits** $\longrightarrow$ 4 bytes
  - **Reales de 64 bits** $\longrightarrow$ 8 bytes
  - **Reales de 128 bits** $\longrightarrow$ 16 bytes
- **VARIABLES NO NUMÉRICAS**
  - **Lógicas** $\longrightarrow$ 1–8 bytes
  - **Alfanuméricas** $\longrightarrow$ ”n” bytes

Tipos de Variables (II)

RANGO DE LAS VARIABLES NUMÉRICAS: (*)

TIPO	BITS	MÍNIMO	MÁXIMO
SIN SIGNO	8	0	$2^8 - 1 = 255$
(idem.)	16	0	$2^{16} - 1 = 65.535$
(idem.)	32	0	$2^{32} - 1 = 4.294.967.295$
(idem.)	64	0	$2^{64} - 1 = 18.446.744.073.709.551.615$
ENTERO	8	$-2^7 = -128$	$2^7 - 1 = 127$
(idem.)	16	$-2^{15} = -32.768$	$2^{15} - 1 = 32.767$
(idem.)	32	$-2^{31} = -2.147.483.648$	$2^{31} - 1 = 2.147.483.647$
(idem.)	64	$-2^{63} = 9.223.372.036.854.775.808$	$2^{63} - 1 = 9.223.372.036.854.775.807$
REAL	32	$\sim -2^{128} \approx -3.403 \cdot 10^{38}$	$\sim 2^{128} \approx 3.403 \cdot 10^{38}$
(idem.)	64	$\sim -2^{1024} \approx -1.798 \cdot 10^{308}$	$\sim 2^{1024} \approx 1.798 \cdot 10^{308}$
(idem.)	128	$\sim -2^{16384} \approx -1.190 \cdot 10^{4932}$	$\sim 2^{16384} \approx 1.190 \cdot 10^{4932}$

(*) Según las normas de coma flotante **IEEE 754–2019** e **ISO/IEC 60559:2020**.
Los números reales se codifican en base 2 y se almacenan en coma flotante (formato mantisa–exponente).
El formato más común de 32 bits destina 24 bits a la mantisa y 8 al exponente, incluidos sus correspondientes signos.
El formato más común de 64 bits destina 53 bits a la mantisa y 11 al exponente, incluidos sus correspondientes signos.

Tipos de Variables (III)

PRECISIÓN DE LAS VARIABLES NUMÉRICAS: (*)

TIPO	BITS	$\lvert$MÍNIMO$\rvert \neq 0$	ERROR DE MÁQUINA
SIN SIGNO	8	1	0
(idem.)	16	1	0
(idem.)	32	1	0
(idem.)	64	1	0
ENTERO	8	1	0
(idem.)	16	1	0
(idem.)	32	1	0
(idem.)	64	1	0
REAL	32	$\sim 2^{-126} \approx 1.175 \cdot 10^{-38}$	$2^{-24} \approx 5.960 \cdot 10^{-8}$
(idem.)	64	$\sim 2^{-1022} \approx 2.225 \cdot 10^{-308}$	$2^{-53} \approx 1,110 \cdot 10^{-16}$
(idem.)	128	$\sim 2^{-16382} \approx 2.362 \cdot 10^{-4932}$	$2^{-113} \approx 9.630 \cdot 10^{-35}$

(*) Según las normas de coma flotante **IEEE 754–2019** e **ISO/IEC 60559:2020**.
Los números reales se codifican en base 2 y se almacenan en coma flotante (formato mantisa–exponente).
El formato más común de 32 bits destina 24 bits a la mantisa y 8 al exponente, incluidos sus correspondientes signos.
El formato más común de 64 bits destina 53 bits a la mantisa y 11 al exponente, incluidos sus correspondientes signos.

Tipos de Variables (IV)

DECLARACIÓN DE TIPOS ELEMENTALES (en máquinas de 32 bits):

TIPO	BITS	FORTRAN	LENGUAJE C
SIN SIGNO	8		unsigned char
(idem.)	16		unsigned short int
(idem.)	32		unsigned int, unsigned long int
(idem.)	64		unsigned long long
ENTERO	8	byte,integer*1	char
(idem.)	16	integer*2	short int
(idem.)	32	integer*4, integer	int, long int
(idem.)	64	integer*8	long long
REAL	32	real*4, real	float
(idem.)	64	real*8, double precision	double
(idem.)	128	REAL*16, quadruple precision	long double
COMPLEJO	64/128/256	complex*4/*8/*16, complex	
LÓGICO	8/16/32/64	logical*1/*2/*4/*8, logical	
ALFANUM.	$8 \cdot n$	character*(n)	

Almacenamiento en coma flotante: advertencias (I)

Según las normas IEEE 754–2019 e ISO/IEC 60559:2020...

- El número $x \neq 0$ se codifica como $x = \pm\, M\, 2^E,\ M \in [0.5, 1),\ E \in \mathbb{Z}$.
- El signo se codifica en un bit ($0 \Rightarrow +,\ 1 \Rightarrow -$).
- La mantisa M se codifica en binario y se redondea por aproximación a m bits significativos. El primer bit no se almacena (siempre es 1).
- El exponente E se codifica en binario y se almacena en e bits.
- Se reservan ciertos valores del exponente para casos especiales: ± 0, $\pm\infty$ y resultados indeterminados $\pm$ NaN. (*)
- En general, el valor almacenado $\mathcal{A}(x)$ será distinto de x, pero

$$\left|\frac{x - \mathcal{A}(x)}{x}\right| \leq r_m, \qquad \text{con} \quad r_m = 2^{-m} \text{(error de máquina)}.$$

(*) Las codificaciones sobrantes también se utilizan y se denominan valores **no normalizados**.

Almacenamiento en coma flotante: advertencias (II)

Debido al redondeo de la mantisa...

- Para algunos valores de x muy frecuentes, $\mathcal{A}(x) \neq x$. (*)
- La aritmética es imprecisa: ¿`0.1+0.2 = 0.3`? (**)
- El orden de las operaciones importa:
  `(0.1+0.2)-0.3` $\neq$ `(0.1-0.3)+0.2`.
- El resultado de una serie de operaciones puede depender del compilador, de las opciones de compilación y del hardware, por lo que puede variar de una instalación a otra. (***)
- Un valor grande no se altera al sumarle un valor pequeño.
- El almacenamiento de enteros grandes como reales no es exacto.

(*) Por ejemplo, si se utilizan variables reales de 32 bits $\mathcal{A}(0.1) \approx 0.10000000149011612$

(**) Depende de la precisión: ¡puede ser cierto con valores de 32 bits pero no con valores de 64 bits!

(***) ¿Se realizan los cálculos intermedios con más precisión? ¿Se redondean estos resultados? ¿Se optimiza el código?

Almacenamiento en coma flotante: advertencias (III)

¿Por qué no se deben utilizar números reales para almacenar enteros?

- $x > 0$ y $(x + 1)$ son indistinguibles cuando $1/x \leq r_m$. (*)

¡Ojo con las comparaciones de valores reales!

- ¿El resultado de `z=(1.0/x)*x-1.0` es igual a `0.0`? (**)

(*) El error de máquina es $r_m = 2^{-m}$, donde m es el número de bits disponibles para la mantisa (incluido el signo).
Por tanto $x > 0$ y $(x + 1)$ son indistinguibles para $x > 2^m$.
En consecuencia, el máximo número entero positivo que se puede almacenar con todas sus cifras exactas en una variable real de 32 bits es $2^{24} = 16.777.216$, que es mucho menor que $2^{32} - 1 = 4.294.967.295$.
El máximo número entero positivo que se puede almacenar con todas sus cifras exactas en una variable real de 64 bits es $2^{53} = 9.007.199.254.740.992$, que es mucho menor que $2^{64} - 1 = 18.446.744.073.709.551.615$.

(**) El resultado de la operación estará afectado por errores de redondeo, por lo que el valor de `z` puede ser distinto de `0.0`.
Es posible que se obtengan resultados diferentes dependiendo del compilador y de las opciones de compilación que se utilicen. Por ejemplo, un compilador podría detectar que el resultado exacto de la operación `z=(1.0/x)*x-1.0` es `z=0.0`, así que si se permite un cierto nivel de optimización el compilador podría modificar el programa para evitar operaciones innecesarias.
Pero también es posible que las mismas instrucciones den lugar a resultados diferentes según donde se encuentren situadas en el programa. Por ejemplo, si la operación anterior se escribe como `q=1.0/x` seguida de `p=q*x` y de `z=p-1.0`, el compilador podría detectar que el resultado exacto es `z=0.0` y modificar el código en consecuencia, de forma que el valor de `z` será exacto. Pero si los cálculo de `q`, `p` y `z` se realizan en subrutinas que se compilan separadamente, el compilador no podrá detectar que el resultado exacto es `z=0.0` ni optimizar esta parte del código, de forma que el valor de `z` será inexacto.
Si el procesador puede realizar cálculos intermedios con precisión extendida, el resultado puede ser diferente según se redondee sólo el valor final de `z` o también los valores intermedios de `q` y `p`.

Almacenamiento en coma flotante: advertencias (IV)

Conviene indicar explícitamente la conversión de los tipos de variable.

¡OJO CON LAS CONSTANTES! (*)

¡CUIDADO CON LA PROPAGACIÓN DE LOS ERRORES!

¿ES CORRECTO EL RESULTADO FINAL? (**)

(*) En Fortran normalmente `0.1` es un valor de 32 bits, mientras que `0.1d+00` es un valor de 64 bits.
Si `x` es una variable de tipo `real*8`, el resultado de `x=0.1` es distinto del de `x=0.1d+00`.
La opción de compilación `-fdefault-real-8` fuerza que todas las constantes sean valores de 64 bits.

(**) Si se utilizan variables de 32 bits, la serie armónica es aparentemente convergente.