Tema 3: Representación de los Datos

HISTORIAL DE REVISIONES

NÚMERO	FECHA	MODIFICACIONES	NOMBRE
v1.0.0	2018 September 12		CA

Índice

1. Temario

1. Representación de datos
    a. Bit, Byte y Palabra
    b. Caracteres, enteros y reales

2. Objetivo

- Representación de los datos alfanuméricos en el lenguaje máquina, es decir, código binario.
- Libro de texto W.Stalling
    - Parte 3ª, Capítulo 9 : Sistemas Numéricos

3. Introducción

- Un programa almacenado en la memoria principal se representa en *lenguaje máquina* y está compuesto por datos e instrucciones . El lenguaje de la máquina es el lenguaje binario formado por los símbolos *0* y *1*. Por lo tanto los datos e instrucciones de un programa almacenado en la memoria principal debe de codificarse y representarse mediante estos dos símbolos.
- Las instrucciones son órdenes que debe capturar, interpretar y ejecutar (ciclo de instrucción) la Unidad Central de Proceso (CPU): sumar, restar, transferir, parar, etc
- Los datos tienen un valor númerico que pueden ser procesados por la Unidad Aritmetico Lógica (ALU) para realizar operaciones aritméticas como la suma, resta, etc ó lógicas como las operaciones not,or,etc

4. Bit, Byte, Palabra

- BInary digiT (bit) : los dígitos binarios son el *0* y el *1*. Son los símbolos que se utilizan para codificar tanto las instrucciones como los datos de un programa almacenado en memoria.
- Byte: Es una secuencia de 8 dígitos binarios. Ejemplo: 00110101
- Palabra: Es una secuencia de dígitos binarios múltiplo de 8, es decir, múltiplo de un byte. En el entorno de arquitectura de computadores se define como el número de bits del bus de datos que conecta la unidad central de proceso (CPU) a la Memoria principal y también suele ser la anchura de los registros de propósito general de memoria interna de la CPU.
    - En linux `sudo lshw -C system` → anchura: **64 bits**

```
    lur
    descripción: Notebook
    producto: 20F1S0H400 (LENOVO_MT_20F1_BU_Think_FM_ThinkPad L560)
    fabricante: LENOVO
    versión: ThinkPad L560
    serie: MP15YSW7
    anchura: 64 bits
    capacidades: smbios-2.8 dmi-2.8 smp vsyscall32
    configuración: administrator_password=disabled chassis=notebook family=ThinkPad L560 ↩
         power-on_password=disabled sku=LENOVO_MT_20F1_BU_Think_FM_ThinkPad L560 uuid=4 ↩
        C2F45AA-0A2C-B211-A85C-B5C56EB5BBAC
```

5. Números Enteros

5.1. introducción

- Sistema Posicional:
  - Número en Base Decimal
    - Dígitos, Valor?
    - Centenas, decenas, unidades
    - Posición → índice
    - Posición → pesos → Potencias $base^{posición}$
    - Valor= sumatorio de digitos ponderados con su peso posicional
    - Ejemplo: 1197

5.2. Base binaria

- Base 2 . Digitos : 0,1.
- Pesos 2^0,2^1,2^2,2^3,2^4 → 1,2,4,8,16
- Conversión Decimal-Binaria:
  - Divisiones sucesivas / 2 → Dividendo1 = 2*Cociente1 + Resto1
  - Cociente1 = 2*Cociente2 + Resto2 → Dividendo1 = *2 * (2*Cociente2 + Resto2) + Resto1 = Resto1*2^0 + Resto2*2^1 +Cociente*2^2*
  - Resto1 es el dígito binario de la posición 0, Resto2 es el digito binario de la posición 1, Cociente es el dígito binario de la posición 2.
  - Regla: los digitos binarios son todos los restos y el último cociente.
  - La división se termina cuando un cociente no es divisible por 2, es decir, el cociente es 1. Este cociente es el MSB.
  - Ejemplo: decimal 1197 → binario 10010101101

Cuadro 1: Conversión decimal binario

Número	**1ª Div**		**2ª Div**		**3ª Div**		**4ª Div**		**5ª Div**		**6ª Div**	
	Coc	Resto	Coc	Resto	Coc	Resto	Coc	Resto	Coc	Resto	Coc	Resto
1197	598	1	299	0	149	1	74	1	37	0	18	1

Número	**7ª Div**		**8ª Div**		**9ª Div**		**10º Div**	
	Coc	Resto	Coc	Resto	Coc	Resto	Coc	Resto
1197	9	0	4	1	2	0	1	0

5.3. Base Octal

- Base 8
- Digitos: 0-7
- Pesos: 8 elevado a la posición
- En C se específica la base con el prefijo *0* → `int 077;`

- Conversión Octal ←→ Binario y viceversa → cada digito octal se descompone en un binario de 3 bits
- decimal 1197 → binario 10010101101 → octal 02255

5.3.1. Base Hexadecimal

- Base 16
- Digitos: 0-1-2-3-4-5-6-7-8-9-A-B-C-D-E-F
- Pesos: 16 elevado a a la posición
- En C se específica la base con el prefijo *0x* → `int 0xAF;`
- Hexadecimal ←→ Binario y viceversa → cada digito hexadecimal se descompone en un binario de 4 bits
- decimal 1197 → binario 10010101101 → 0x4AD

5.4. Calculadora

- Calculadora en el sistema Linux
  - `candido@lur:~$ echo "obase=2 ; ibase=16; 80AA010F" | bc`
    - 10000000101010100000000100001111
  - `echo "obase=10 ; ibase=16; 80AA010F" | bc` → es obligado poner primero la base del formato de salida
    - 2.158.625.039
  - Intérprete `$ bc`

5.5. Python

- https://docs.python.org/3/tutorial/index.html
  - help(builtins)

```
bin(1197) -> '0b10010101101'
oct(1197) -> '02255'
hex(1197) -> '0x4ad'
int(0x4ad) -> 1197
```

5.6. Enteros con Signo

5.6.1. Signo-Magnitud

- Formato Signo-Magnitud
  - El bit más significativo no tiene valor, indica el signo: el cero para los números positivos y el uno para los negativos.
  - El resto de bits representa el módulo del número entero
  - Ejemplo :
  - *+1197 : 010010101101*
  - -1197 : 110010101101
  - ¿Cómo se representa el cero?

5.6.2. Complemento a 2

- Formato Complemento a 2.
  - Positivos: Igual que el formato signo-magnitud: Bit MSB= 0. Pesos: potencia $2^{\text{posición}}$.
  - Negativos: Transformación del número con la misma magnitud pero positivo mediante la función Complemento a 2.

- La Rueda

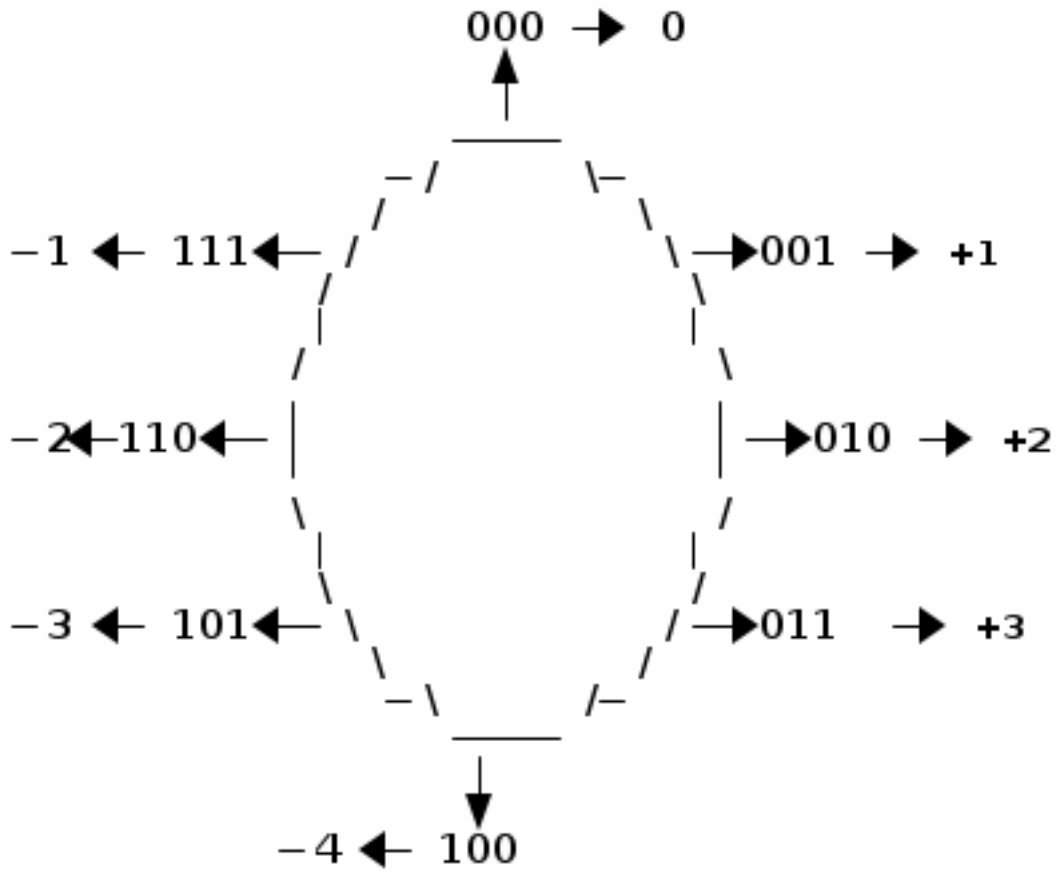

Figura 1: Complemento a 2. Longitud 3 bits.

  - N: cantidad de bits del número : 3 bits
  - Dividir la circunferencia en el número de combinaciones binarias posibles: $2^N : 2^3$
  - Pinto todas las combinaciones binarias en sentido agujas reloj de forma secuencial: 000,001,010,011,
  - Pinto los valores de forma alternante: *0, +1, -1, +2, -2,...*.
  - Representar el número positivo *+4* en complemento a 2

- Conclusiones:

  - Asimetría entre el rango positivo y negativo
  - El cero tiene una única representación
  - Los números negativos comienzan por *1*
  - El valor -1 se codifica con todos los dígitos unos *11111111111111*
  - Extensión de Signo: un 1 por la izda es como en los positivos un cero por la izda: no tiene valor y se puede elimar la repetición de 1 por la izda dejando el último de los más significativos. *11110111* es equivalente a *10111*.

- **Función Complemento a 2**: El complemento a 2 de un número entero equivale a cambiar su signo. La conversión entre números enteros positivos y negativos en complemento a 2 se puede realizar mediante distintos métodos (ejemplos de complemento a 2 del número entero con código binario X):

a. Realizar la operación lógica complemento (negación) del código X y sumar 1 → ~X+1
   - X=0101 tiene valor *+5* en complemento a 2
   - ¿el código del valor -5? *~0101 + 1 = 1010 + 1 = 1011 = -5*
   - X=1111 tiene valor -1 en complemento a 2
   - ¿el código del valor *+1*? *~1111 + 1 = 0000 + 1 = 0001 = +1*
   - X=01100111000101010100000 → positivo por tener el bit más significativo (MSB) cero
   - C2(X)=10011000111010101011111+1=10011000111010101100000 → negativo por tener el bit más significativo (MSB) uno

b. Empezando por la posición 0 del código X (bit X_0) copiar todos los dígitos hasta llegar al primer dígito 1 y a partir de ahí negar todos los dígitos hasta el bit más significativo (MSB).
   - X = 01100111000101010100000 → en total 22 bits
   - El primer dígito 1 de X está en la posición 4 → 01100111000101010-10000 → copio los 5 primeros digitos e invierto los 17 restantes
   - C2 (X) = 10011000111010101-10000

c. Realizar la operación aritmética 0-X
   - X = 01100111000101010100000
   - 0-X=00000000000000000000 - 01100111000101010100000 = 01100111000101010100000

- Ejemplos
  - Representar el número entero negativo -1197 en signo-magnitud y en complemento a 2
    - *+1197* = 010010101101 tanto en signo-magnitud como complemento a 2
    - *-1197* = 101101010011
  - Calcular el rango de los números enteros con 8 bits en complemento a 2
    - Código máximo positivo: 01111111 → Valor = 2^7-1
    - Código mínimo negativo: 10000000
      - C2(n=10000000) = 010000000 = 2^7 ,luego n=10000000 tiene el valor -2^7
    - Rango [-2^7,+2^7-1]

6. Números Reales

6.1. Coma Fija

- Números Reales
  - Coma Fija
    - 1234.56789
    - Sistema Posicional → pesos de los dígitos fracción

6.2. Coma Flotante

6.2.1. Formato

- Coma Flotante → Notación científica
  - -23.4567E-34 ó -23.4567*10^{-34}
  - La **mantisa** o **significando** es el número que múltiplica a la potencia → -23.4567
  - Mantisa **normalizada** : La mantisa tiene como parte entero un número entero de un dígito distinto de cero.→ -2.34567*10^{-33}
    - parte entera de la mantisa normalizada : 2
    - parte fracción de la mantisa normalizada : 0.34567
  - El **exponente** es el número entero al que se eleva la base de la potencia. Depende del lugar de la coma en la mantisa.
  - La **base** es la base de la potencia.
- Codificación Binaria
  - Ejemplo: 1234.56789
    - Parte Entera: 1234 → 10011010010
    - Parte Fracción: 0.56789

```
0.56789 * 2 = 1.13578 = 1 + 0.13578 -> 1, bit de la posición -1
0.13578 * 2 = 0.27156 -> 0, bit de la posición -2
0.27156 * 2 = 0.54312 -> 0,  bit de la posición -3
0.54312 * 2 = 1.08624 = 1 + 0.08624 -> 1, bit de la posición -4
```

      - ◇ fracción redondeada 0.1001
      - ◇ fracción sin redondear 0.10010001011000010
      - ◇ fracción redondeada 0.100100011
    - Código Binario: 10011010010.10010001011000010
    - Notación científica: 1.00110100101001000101100001 0*2^{+10} → coma flotante

6.2.2. Precisión

- Es el número de digitos significantes
- Se dice que el número q es una aproximación del número p != 0 con una precisión de, al menos, **m** cifras signficativas en la base b, siempre que el error relativo |p-q|/p $\leq$ 0.5*b^{-m+1}
  - Cuando m es el mayor entero para el que se cumple la desigualdad anterior, se dice que q aproxima a p con m cifras signficativas.
- Ejemplo

a. p = .1E0 y q = .9999E0 → Error relativo=0.1E-3<0.5E(-4+1) → precisión de 4 cifras significativas
b. Una calculadora, A, trabaja en base 2 con mantisa de 22 bits y otra, B, trabaja en base 16 con 6 dígitos de precision (24 bits). ¿Cuál de las dos es más precisa?

6.2.3. Norma IEEE-Standard 754

- Float
  - Norma IEEE-Standard 754
    - Precisión simple → formato de longitud 32 bits en 3 campos *Signo/Exponente/Fraccción* de longitudes 1/8/23 bits

    - Precisión doble → formato de longitud 64 bits en 3 campos *Signo/Exponente/Fraccción* de longitudes 1/11/52 bits

    - conversion manual
    - wiki
    - Binario: Tres campos

Signo	Exponente en Exceso	Fracción de la Mantisa Normalizada

- Valor $(-1)^{Signo}$ x 1.Fracción_Mantisa_Normalizada x $2^{Exponente}$
  - Signo: positivo -→ bit *0* , negativo --→bit *1*
  - Exponente en exceso: Es el Exponente al que se añade 127 (precisión simple) ó 1023 (precisión doble)
  - Mantisa Normalizada: Es la mantisa tal que su parte entera es *1*
    - Fracción de la Mantisa Normalizada: Es la fracción de la mantisa normalizada.

6.2.4. Conversores de Código

- Conversores online:
  - binary converter: tipos char,short,int,float,double
  - conversor ieee754
  - IEEE 754 single precision: decimal → binario/hexadecimal y viceversa
  - http://www.rsu.edu/faculty/pmacpherson/programs/iee754.html

6.2.5. Float Point: Representaciión del Cero,Infinito e Indeterminado

- Cuando el campo del exponente son todo ceros o unos, no se sigue la regla general de un número normalizado

Cuadro 2: Single precision

Números	Exp	Fracción
Ceros	0x00	0
Números desnormalizados	0x00	distinto de 0
Números normalizados	0x01-0xFE	cualquiera
Infinitos	0xFF	0
NaN (Not a Number)	0xFF	distinto de 0

- Cero
  - Por qué el cero se representa en single precision como una secuencia de 32 ceros
  - Por qué cuando el campo del exponente es cero la potencia es 2^{-126} en lugar de 2^{-127} y la mantisa se considera NO normalizada, es decir, 0.fracción en lugar de 1.fracción.
- Notas Maryland
- Infinito

6.2.6. Referencia

- numerical analysis: programas ejemplo sencillos
- IEEE
- William Kahan
- Yale: C programming. float.c.
- Bruce Dawson blog
  - https://randomascii.wordpress.com/2012/01/11/tricks-with-the-floating-point-format/
- wikipedia
  - http://www.validlab.com/goldberg/paper.pdf
    - The pitfalls of verifying oating-point computations
    - el mismo?
- cprogramming
- code tips
- c review: practicas

7. Character Type

7.1. ASCII

- Codificación ASCII
  - American Standard Code International Intechange: codificación con 7 bits : rango 0x00-0x7F
  - Tabla de conversión carácter-código_hexadecimal-código binario
    - `man ascii`
    - K.N. King,Apéndice E, pg801

Caracter	ASCII hex	Control (Secuencia de Escape)
0	0x30	
1	0x31	
a	0x61	
A	0x41	
+	0x2B	
^J	0x0A	**nueva línea (\n)**
^M	0x0D	**retorno de carro (\r)**

C program '\X' escapes are noted.

Oct	Dec	Hex	Char	Oct	Dec	Hex	Char
────────────────							
000	0	00	NUL '\0'	100	64	40	@
001	1	01	SOH (start of heading)	101	65	41	A
002	2	02	STX (start of text)	102	66	42	B
003	3	03	ETX (end of text)	103	67	43	C
004	4	04	EOT (end of transmission)	104	68	44	D
005	5	05	ENQ (enquiry)	105	69	45	E
006	6	06	ACK (acknowledge)	106	70	46	F
007	7	07	BEL '\a' (bell)	107	71	47	G
010	8	08	BS '\b' (backspace)	110	72	48	H
011	9	09	HT '\t' (horizontal tab)	111	73	49	I
012	10	0A	LF '\n' (new line)	112	74	4A	J
013	11	0B	VT '\v' (vertical tab)	113	75	4B	K
014	12	0C	FF '\f' (form feed)	114	76	4C	L
015	13	0D	CR '\r' (carriage ret)	115	77	4D	M
016	14	0E	SO (shift out)	116	78	4E	N
017	15	0F	SI (shift in)	117	79	4F	O
020	16	10	DLE (data link escape)	120	80	50	P
021	17	11	DC1 (device control 1)	121	81	51	Q
022	18	12	DC2 (device control 2)	122	82	52	R
023	19	13	DC3 (device control 3)	123	83	53	S
024	20	14	DC4 (device control 4)	124	84	54	T
025	21	15	NAK (negative ack.)	125	85	55	U
026	22	16	SYN (synchronous idle)	126	86	56	V
027	23	17	ETB (end of trans. blk)	127	87	57	W
030	24	18	CAN (cancel)	130	88	58	X
031	25	19	EM (end of medium)	131	89	59	Y
032	26	1A	SUB (substitute)	132	90	5A	Z

033	27	1B	ESC (escape)	133	91	5B	[
034	28	1C	FS (file separator)	134	92	5C	\ '\\'
035	29	1D	GS (group separator)	135	93	5D	]
036	30	1E	RS (record separator)	136	94	5E	^
037	31	1F	US (unit separator)	137	95	5F	_
040	32	20	SPACE	140	96	60	`
041	33	21	!	141	97	61	a
042	34	22	"	142	98	62	b
043	35	23	#	143	99	63	c
044	36	24	$	144	100	64	d
045	37	25	%	145	101	65	e
046	38	26	&	146	102	66	f
047	39	27	´	147	103	67	g
050	40	28	(	150	104	68	h
051	41	29	)	151	105	69	i
052	42	2A	*	152	106	6A	j
053	43	2B	+	153	107	6B	k
054	44	2C	,	154	108	6C	l
055	45	2D	-	155	109	6D	m
056	46	2E	.	156	110	6E	n
057	47	2F	/	157	111	6F	o
060	48	30	0	160	112	70	p
061	49	31	1	161	113	71	q
062	50	32	2	162	114	72	r
063	51	33	3	163	115	73	s
064	52	34	4	164	116	74	t
065	53	35	5	165	117	75	u
066	54	36	6	166	118	76	v
067	55	37	7	167	119	77	w
070	56	38	8	170	120	78	x
071	57	39	9	171	121	79	y
072	58	3A	:	172	122	7A	z
073	59	3B	;	173	123	7B	{
074	60	3C	<	174	124	7C	|
075	61	3D	=	175	125	7D	}
076	62	3E	>	176	126	7E	~
077	63	3F	?	177	127	7F	DEL

- `showkey -a` : espera a pulsar una letra y visualiza el código ASCII de la letra pulsada
  - útil para descubrir el código de cada carácter en ascii standard y el de caracteres ñ, á, é, í, ó, ú si en el código en que el sistema esté configurado.
  - útil para descubri el código de control de combinaciones Ctrl-C, CR, Ctrl-CR, Ctrl-D

```
\         92 0134 0x5c   -> tecla ESC: escape
^J        10 0012 0x0a   -> teclas Ctrl-CR: Salto de línea
^M        13 0015 0x0d   -> tecla CR: Retorno de Carro
^C         3 0003 0x03   -> teclas Ctrl-c
^D         4 0004 0x04   -> teclas Ctrl-d
ñ        195 0303 0xc3   -> MSB: More Significand Byte.
         177 0261 0xb1   -> LSB: Less Significand Byte
```

  - UPNA → 0x55-0x50-0x4e-0x41-0x00 donde 0x00 es el caracter NUL de fin de cadena.
- ASCII Extendido
  - https://en.wikipedia.org/wiki/Extended_ASCII#ISO_8859_and_proprietary_adaptations

- `man iso_8859_1`: latin-1: ascii extendido: 0x80-0xFF
- http://www.theasciicode.com.ar/ascii-printable-characters/vertical-bar-vbar-vertical-line-vertical-slash-ascii-code-124.html
  - El linux pulsar ctrl-Shift-u-ascii_code Enter
  - Ejemplo: el código extendido de la *ñ* es 0xF1 → C-S-u-f1 Enter→ C-S-u simultáneo y aparece la u esperando al código, F-1-enter
- ascii code finder
- 0
- ~
- ¬
- ñ

7.2. Python

- ejemplos de conversión
  - `python`

```
ord('A')
hex(ord('A'))
chr(65)
chr(0x41)
[hex(ord(c)) for c in "Hola"]
[chr(c) for c in [0x48, 0x6f, 0x6c, 0x61, 0x20, 0x4d, 0x75, 0x6e, 0x64, 0x6f]]
```

7.3. Unicode UTF-8

- Unicode Main
- Unicode: Unicode can be implemented by different character encodings. The Unicode standard defines Unicode Transformation Formats (UTF): UTF-8, UTF-16, and UTF-32, and several other encodings. The most commonly used encodings are UTF-8, UTF-16, and the obsolete UCS-2 (a precursor of UTF-16 without full support for Unicode)
- Unicode encoded: https://www.unicode.org/versions/Unicode14.0.0/ch02.pdf#G25564
  - Se describe con el prefijo U+ seguido de un número entero (integers from 0 to 0x10FFFF). Al código se le llama "code point".
- UTF-8:
  - The dominant encoding on the World Wide Web and on most Unix-like operating systems
  - Uses one byte[note 1] (8 bits) for the first 128 code points, and up to 4 bytes for other characters. The first 128 Unicode code points represent the ASCII characters, which means that any ASCII text is also a UTF-8 text.
  - La *ñ* da como salida 0xc3b1 . El terminal está configurado con salida Unicode UTF8 según la variable de entorno local. Mediante el comando `locale charmap` volcamos con que codificación tenemos la salida del terminal. Mediante `locale -m` los posibles. Podría haber sido iso-8859-1 en lugar de utf8.
- utf8:
  - 8-bit Unicode Transformation Format
  - Usa símbolos de longitud variable (de 1 a 4 bytes por carácter Unicode).
  - Esta orientado a la transmisión de palabras de 1 byte.
  - unicode ñ
  - la ñ tiene unicode point *U+00F1* ó hex_code_utf8 *0xC3B1*

  - en la wikipedia utf-8 explica cómo pasare de unicode point a hex code.
  - https://unicode-table.com/es/00F1/
- Problema Para copiar los caracteres no US-ASCII de la barra URL de firefox: https://es.wikipedia.org/wiki/Conmutaci%C3%B3n_d → C3B3 es el código hexadecimal del código utf-8 del carácter ó.
- wikipedia utf-8:
  - desglose de códigos según 1byte,2byte,3 byte, 4 bytes.
  - cómo se mapea el unicode code point del utf-8 a hexadecimal
- `man utf-8`

- Unicode chart
  - Colocando el puntero sobre la categoría se visualiza el rango hexadecimal del charset
  - Symbols Punctuation:
    - Punctuation: ASCII Punctuation: U0000.pdf
    - Find chart by hex code: 278a
    - Pictographs: Dingbats: ❶ → U2700.pdf
    - Mathematical symbols: Mathematical Operators:
      - wikipedia
      - U2200
      - ∨
      - ⊼
      - ⊽
      - ⊦
  - otros
    - 😋
    - ñ
    - ñ
    - ←
    - →
- https://wiki.mozilla.org/Help:Special_characters#Unicode
- info detallada sobre un caracter unicode: Pej U+0305
- Microsoft Office
- Overline o suprarayado
  - LibreOffice has direct support for several styles of overline in its `Format / Character / Font Effects" dialog: suprarayado`

8. ISO-8859-1

- Alternativa a UTF-8 para el alfabeto latino
- https://es.wikipedia.org/wiki/ISO/IEC_8859-1
  - Sólo utiliza 1 byte , por lo tanto es equivalente al ascii extended.
  - La norma ISO/IEC 8859-15 consistió en una revisión de la ISO 8859-1, incorporando el símbolo del Euro
- `man iso_8859-1`
- La "ñ" tiene el código 0xF1

8.1. Programación en C

- Convertir un carácter numérico en su valor entero
  - Mediante una operación aritmética
- Convertir un carácter minúscula en mayúscula
  - Mediante una operación aritmética

8.2. Otros

- Lenguaje C: printf
  - `locale -a` → LC_CTYPE
  - ñ → `env printf \u00f1 \n` : incluir las simples comillas
    - printf invocation