Técnicas Digitales II

Acceso a Memoria

Dr.Ing.Steiner Guillermo

Acceso a memoria

- Una característica de RISC, es que posee una modelo Load/Store, este modelo divide al set de instrucciones en dos conjuntos.
  - Acceso a memoria (carga y almacenamiento entre memoria y registro).
  - Operaciones con la ALU (siempre entre registros).

Acceso a memoria

Además ARM posee dos subconjuntos de instrucciones para acceder a memoria

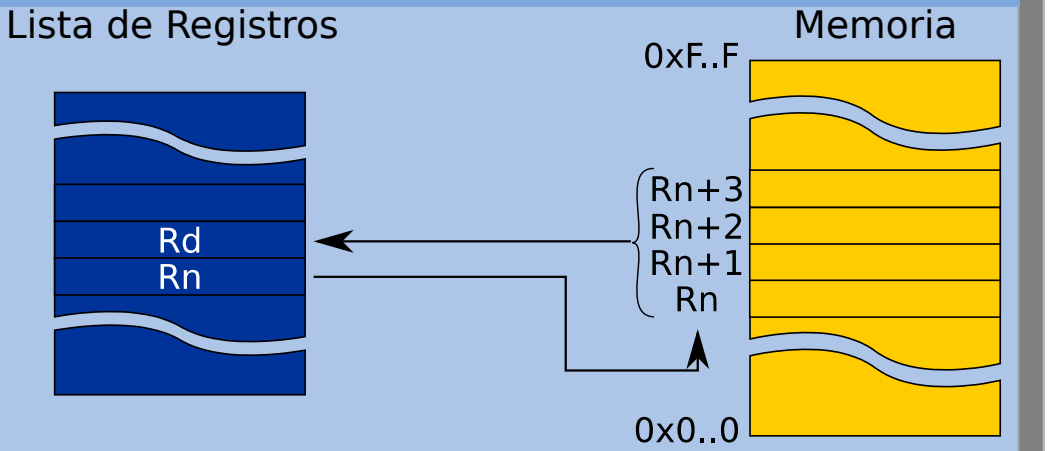
Instrucciones de Carga y Almacenamiento de UN REGISTRO
Permite la carga y almacenamiento de un solo registro desde y hacia la memoria.

Instrucciones de Carga y Almacenamiento de múltiples registros
Permite la carga y almacenamiento de un listado de registros desde y hacia la memoria, comúnmente usada en manejo de STACK.

Instrucciones de Carga y Almacenamiento

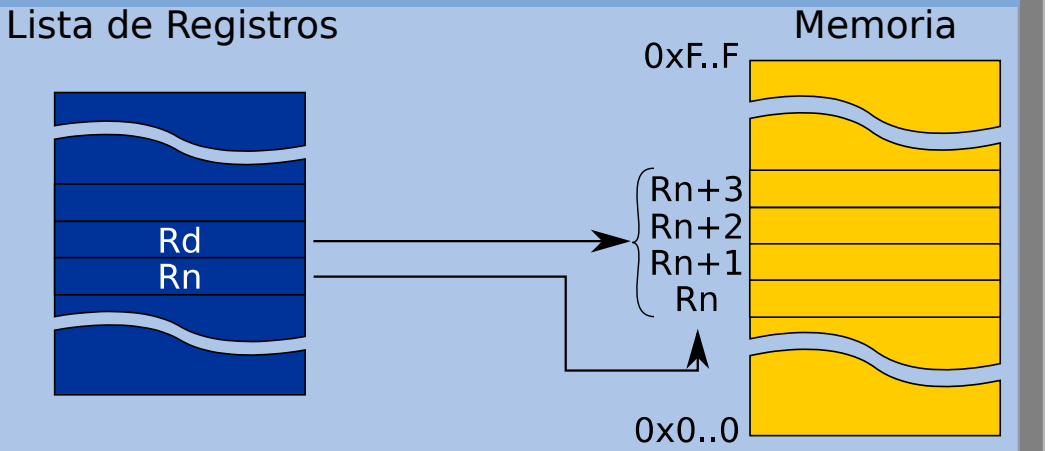
Carga

LDR Rd , [Rn]
Leer desde la posición indicada en Rn un word y lo guarda en el registro Rd.



Almacenamiento

STR Rd , [Rn]
Escribe el contenido de Rd a partir de la posición indicada por el registro Rn.



Modos de Direccionamiento

- Las instrucciones de carga y almacenamiento de un registro, requieren para definir la dirección de memoria donde se leerá o escribirá el dato, de por lo menos de un registro base.
- A este registro se le sumará o restara otro valor (offset), el tipo de este valor definirá el modo de direccionamiento:
  - Direccionamiento Inmediato. **LDR Rd , [Rn , #offset]**
  - Offset por registro. **LDR Rd , [Rn , Rm]**
  - Offset por registro escalado. **LDR Rd , [Rn , Rm lsl #2]**

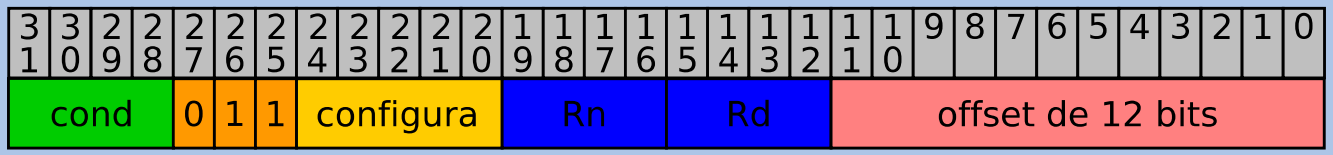
Modos de Direccionamiento

Inmediato

Al registro base se le suma o resta un valor numérico contenido en la propia instruc.

LDR Rd , [Rn , #offset]

Rd = [Rn + #offset]

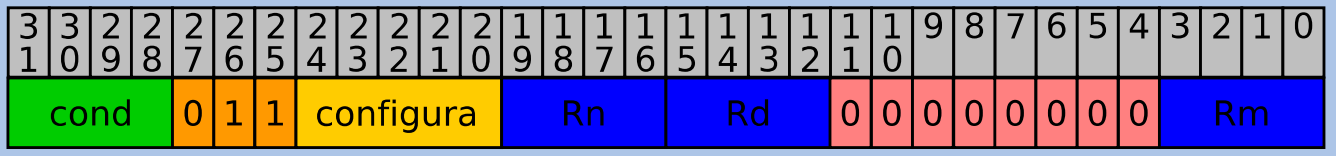


Offset por registro

Al registro base se le suma o resta otro registro.

LDR Rd , [Rn , Rm]

Rd = [Rn + Rm]

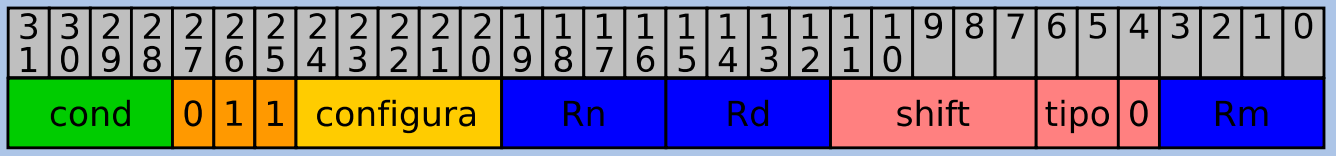


Offset por registro escalado

Al registro base se le suma o resta un registro escalado (barrel shifter).

LDR Rd , [Rn , Rm LSL #2]

Rd = [Rn + Rm · 4]



Tipo de Direccionamiento

- El registro base y el offset pueden vincularse de distintas maneras, esto define tipo de direccionamiento.
  - Direccionamiento pre-indexado. **LDR Rd,[Rn,#offset]!**
  - Direccionamiento post-indexado. **LDR Rd,[Rn],#offset**
  - Direccionamiento por corrimiento (también llamado pre-indexado sin actualizar). **LDR Rd,[Rn,#offset]**

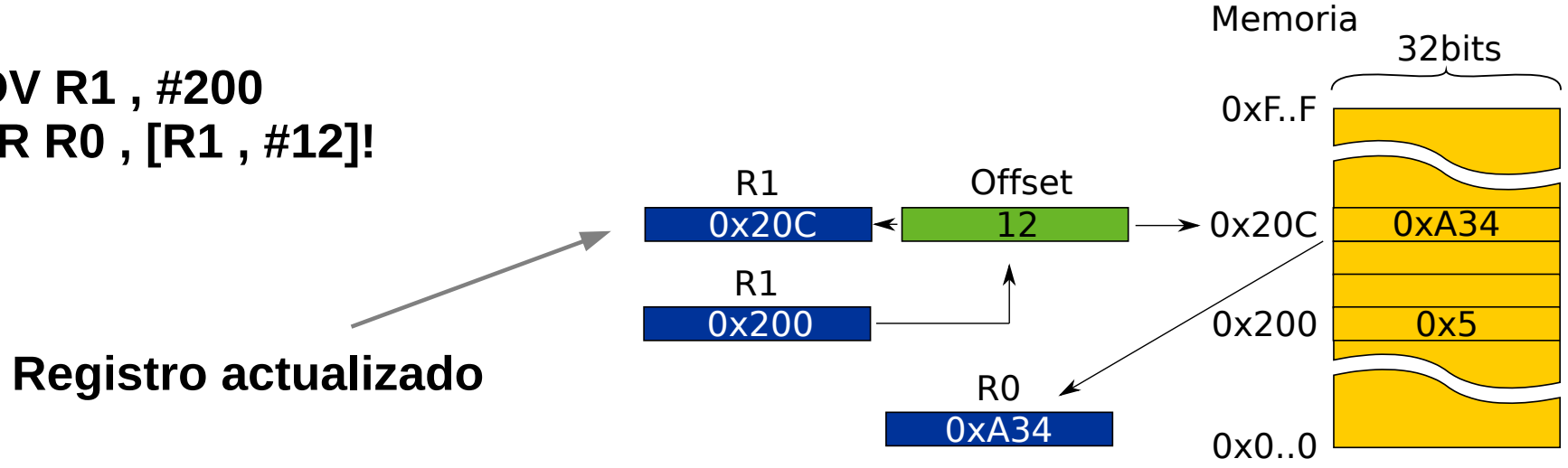
Modos de Direccionamiento

Pre-indexado
La dirección de memoria está formada por la suma o resta del offset con el registro base, esta dirección de memoria calculada, es escrita luego en el registro base.

LDR Rd,[Rn , #offset]!

Luego de la operación los registros quedan:
Rd = [Rn + #offset]
Rn = Rn + #offset

MOV R1 , #200
LDR R0 , [R1 , #12]!



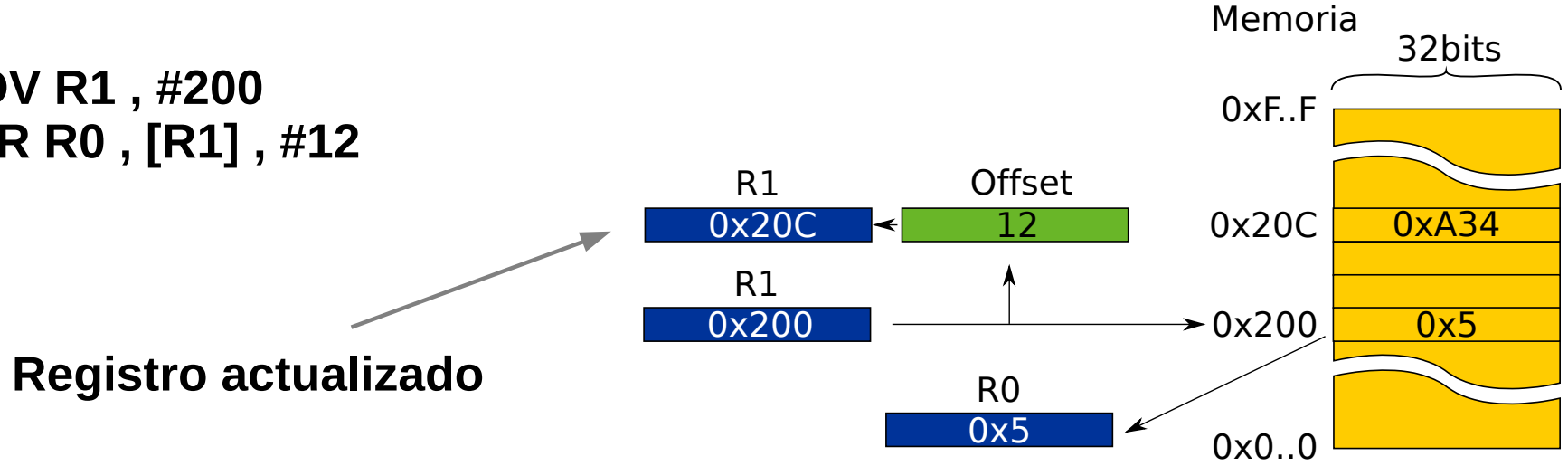
Modos de Direccionamiento

Post-indexado
Se toma el dato de la dirección apuntada por el registro base y luego se actualiza este sumándole o restándole el corrimiento.

LDR Rd , [Rn] , #offset

Luego de la operación los registros quedan:
Rd = [Rn]
Rn = Rn + #offset

MOV R1 , #200
LDR R0 , [R1] , #12



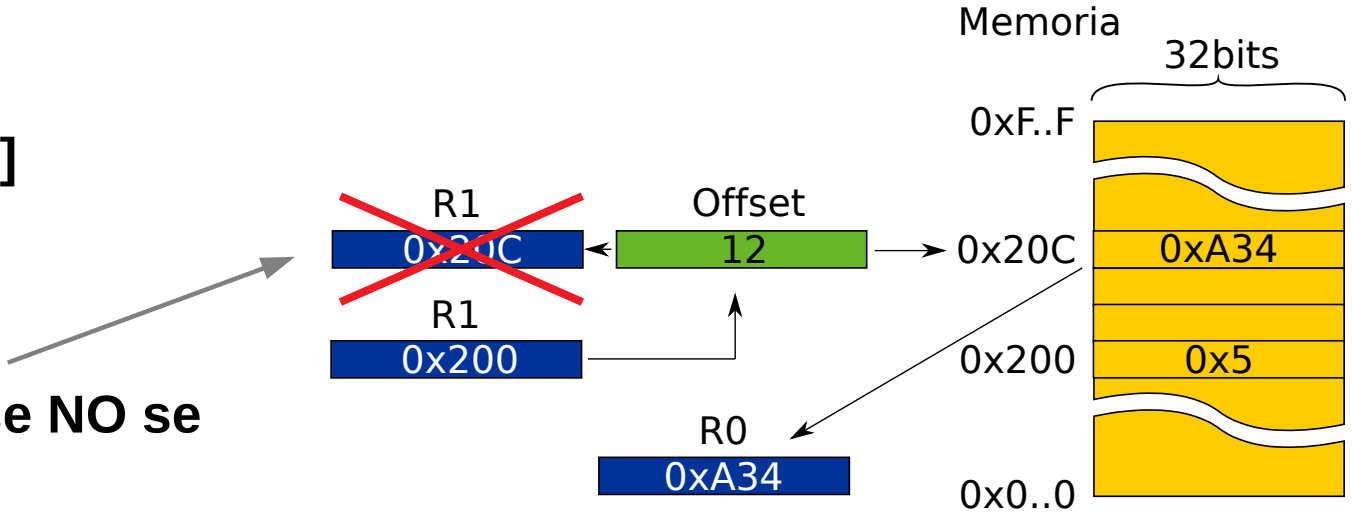
Modos de Direccionamiento

Por corrimiento o Pre-indexado sin actualizar
La dirección de memoria está formada por la suma o resta del offset con el registro base.

LDR Rd,[Rn , #offset]

Luego de la operación los registros quedan:
Rd = [Rn + #offset]
Rn = Rn

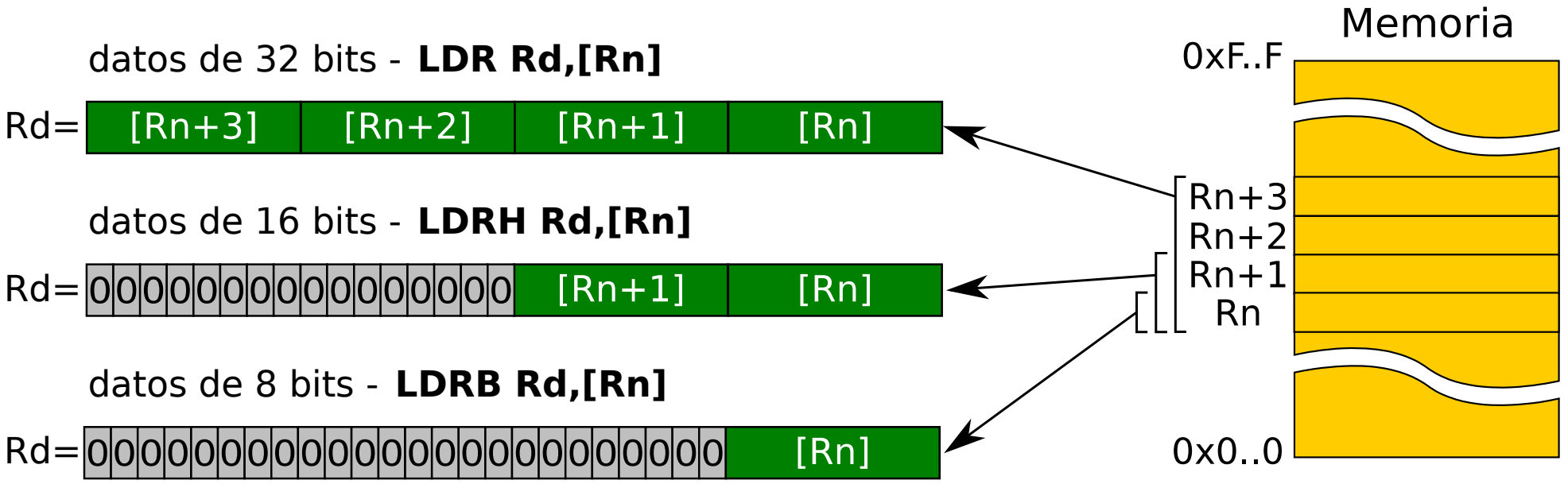
MOV R1 , #200
LDR R0 , [R1 , #12]



El registro base NO se actualiza

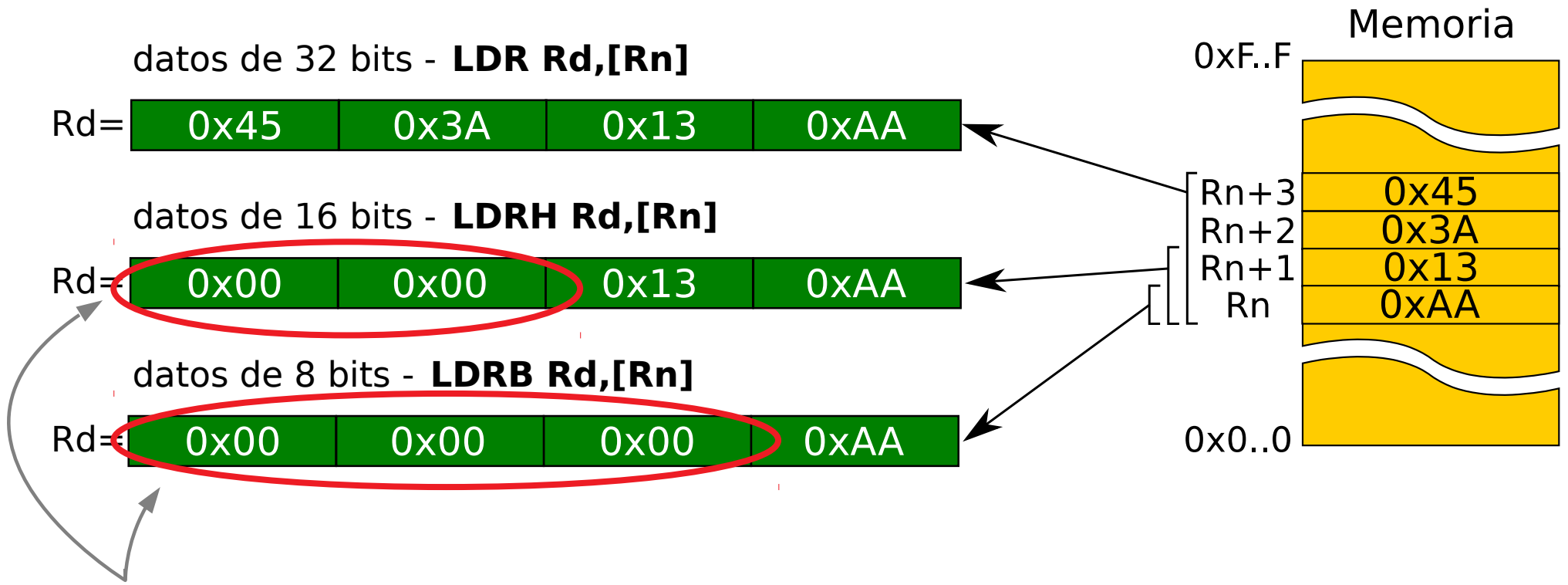
LDR tipo de datos

- La instrucción de carga desde memoria a registro, admite tres tipos distinto de tamaño de dato.
- De ser necesario el registro se completa con 0.



LDR tipo de datos

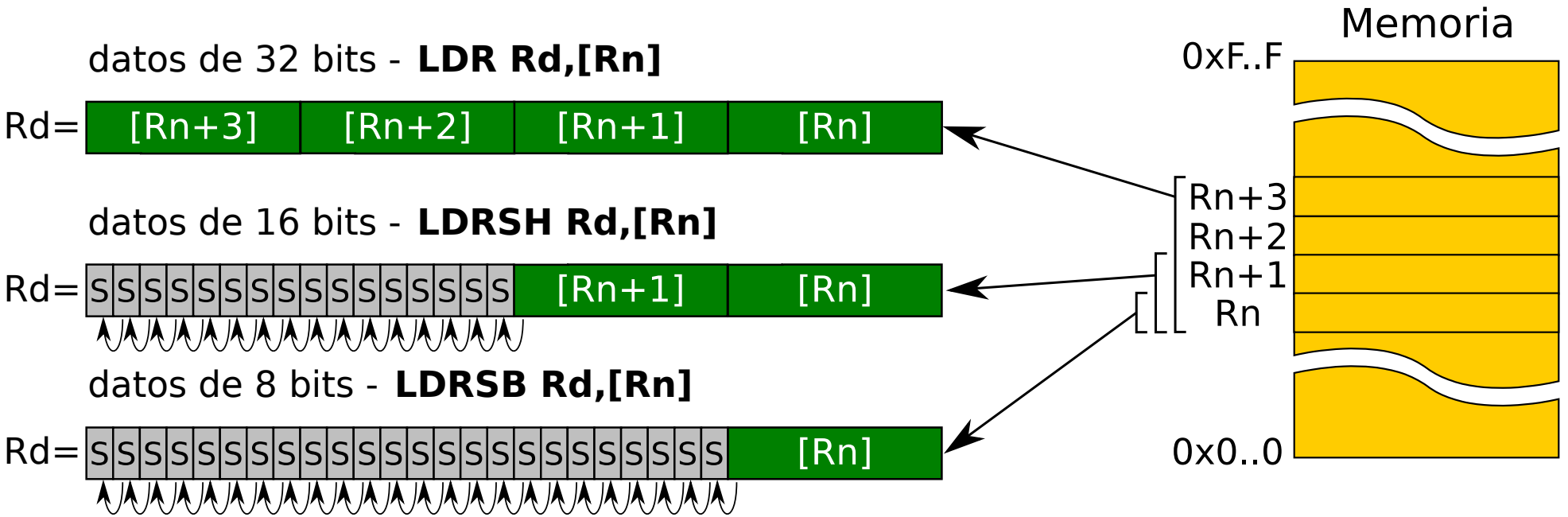
- Ejemplo de carga desde memoria.



Completado con 0 por la Instrucción LDR

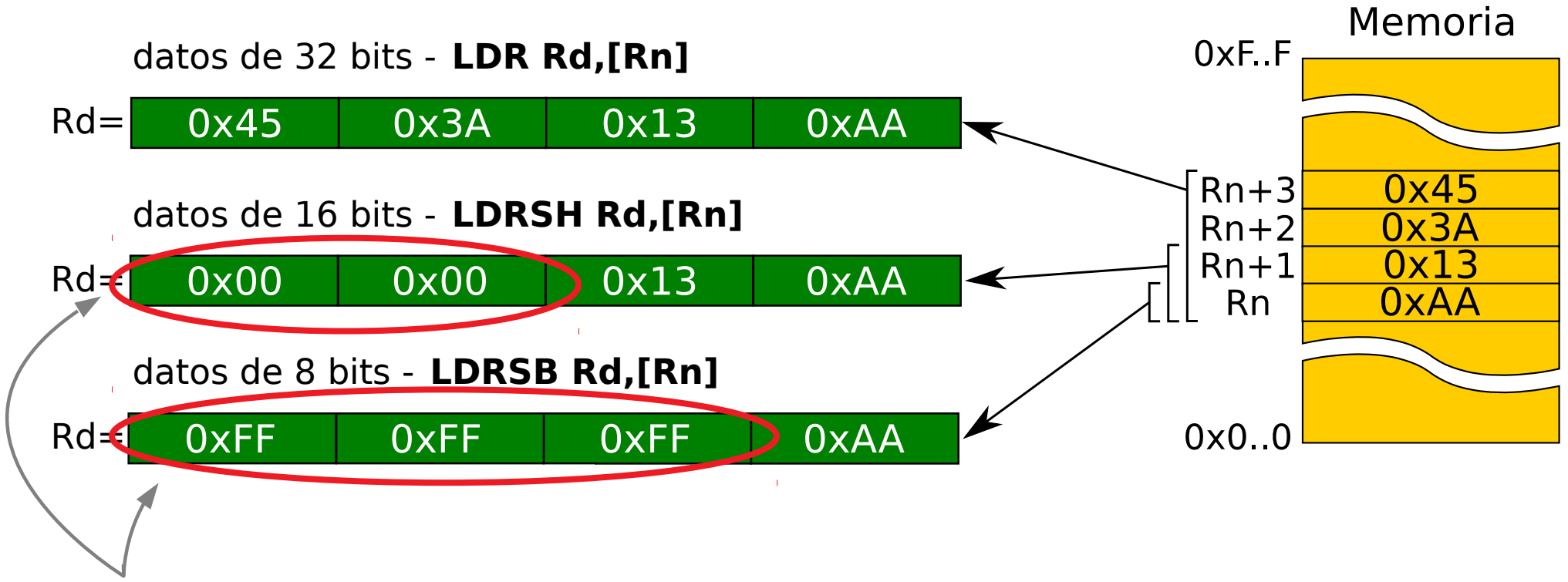
LDR tipo de datos

- El mismo conjunto de tipos de datos, será tratado de distinta forma si son números CON SIGNO.
- De ser necesario el registro se completa con el signo.



LDR tipo de datos

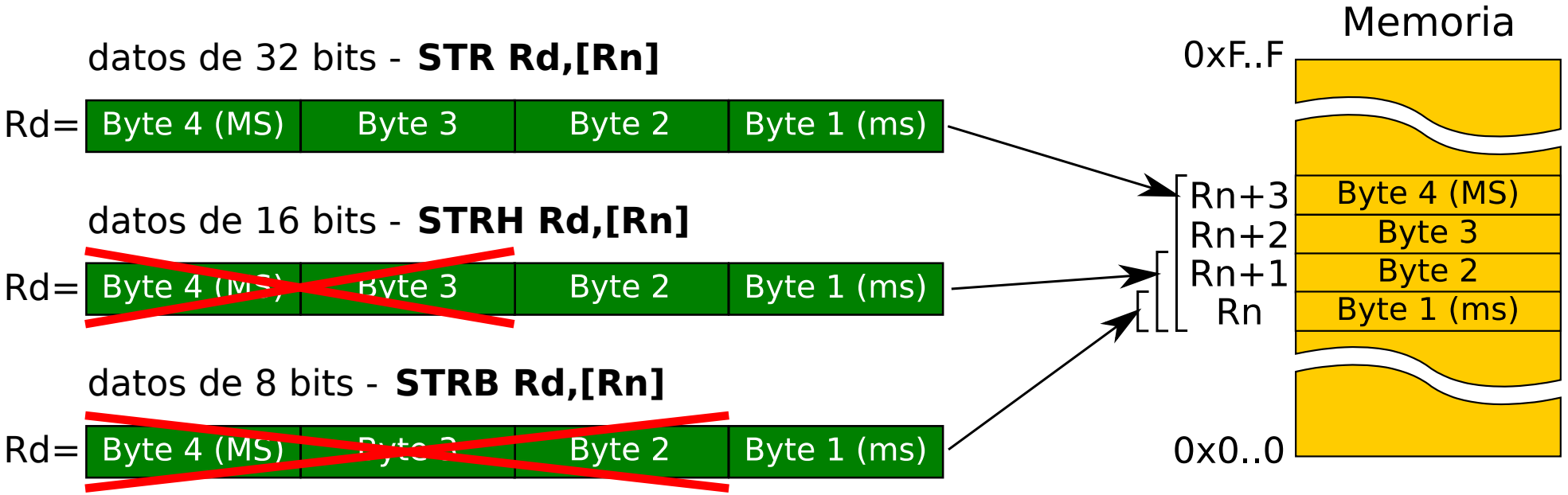
- Ejemplo de carga desde memoria CON SIGNO.



Completado con el signo (bit 15 o 7 según corresponda) por la Instrucción

STR tipo de datos

- La instrucción para guardar de un registro a memoria, admite los mismos tipos de datos que para la carga.
- En formatos menores a 32 bits, el dato es recortado al tamaño adecuado, este procedimiento no requiere conocer si el número es con signo o sin signo.



LDR tipo de datos

- Ejemplo de los datos guardados para un Rd = 0x453A83AA.

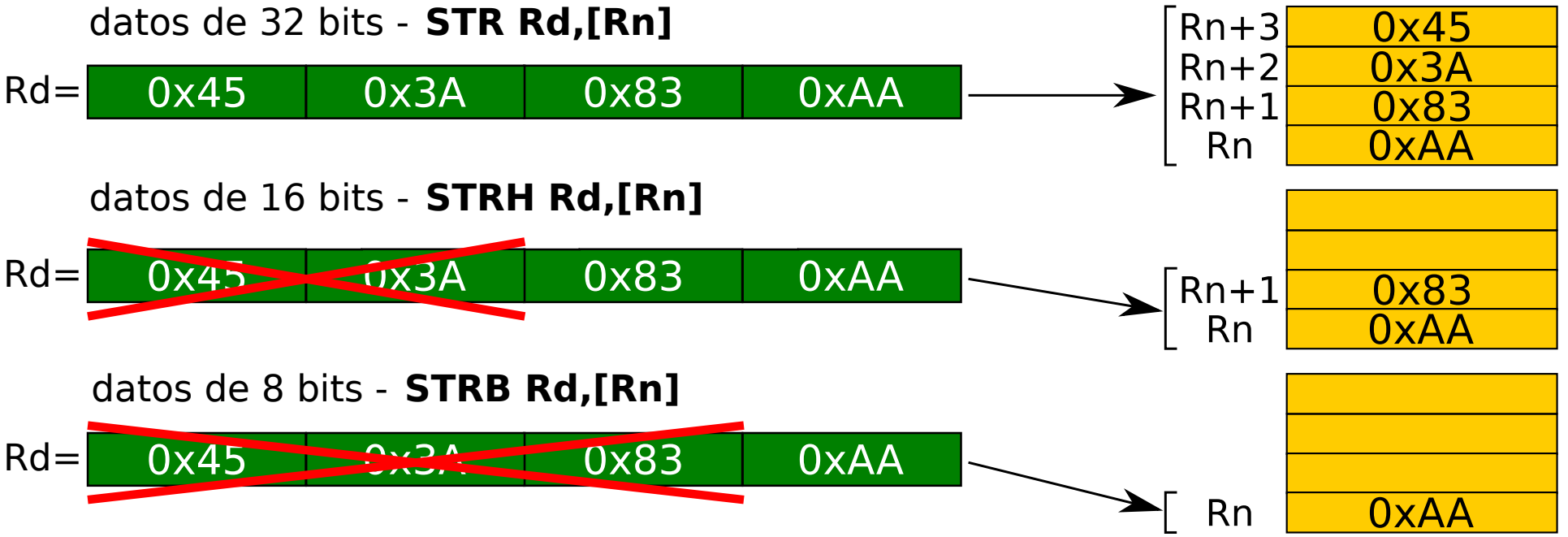


Tabla de Direccionamiento

- De acuerdo a los modos de direccionamiento y a los tipos de direccionamientos, se define la siguiente tabla.

  WB – Word o Byte sin signo / HB – Halfword o Byte con signo

TDato	Modo	Descripción	Ejemplo
WB	[<Rn>, #+/-<o_12>]	inmediato	LDR R1,[R2, #+1024]
HB	[<Rn>, #+/-<o_8>]	inmediato	LDRH R1,[R2, #+100]
todos	[<Rn>, +/-<Rm>]	por registro	LDR R1,[R2, R3]
WB	[<Rn>, +/-<Rm>,<sh>#<imm>]	por registro escalado	LDR R1,[R2, -R4, LSL #3]
WB	[<Rn>, #+/-<o_12>]	inmediato pre indexado	LDR R1,[R2, #-1024]!
HB	[<Rn>, #+/-<o_8>]	inmediato pre indexado	LDRH R1,[R2, #+100]!
todos	[<Rn>, +/-<Rm>]	por registro pre indexado	LDR R1,[R2, -R4]!
WB	[<Rn>, +/-<Rm>,<sh>#<imm>]	por reg. escalado pre-ind.	LDR R2,[R4, R3, LSL #3]!
WB	[<Rn>], #+/-<o_12>	inmediato post-index.	LDR R1,[R2],#+1024
HB	[<Rn>], #+/-<o_8>	inmediato post-index.	LDRSB R1,[R2], #+100
todos	[<Rn>], +/-<Rm>	por registro post-index.	LDR R1,[R2],R3
WB	[<Rn>], +/-<Rm>,<sh>#<imm>	por reg. escalado post-ind.	LDR R1,[R2],R3, LSL #2

Tipo de datos en ensablador

- La definición de datos es algo mas compleja que en lenguajes de Alto Nivel.
- Aquí la definición no queda supeditada solo a cuando se declara la variables, sino es necesaria tenerla en cuenta en cada lugar que se haga uso de la variable.
- Tamaño del tipo de dato y si está almacenada con signo o sin signo, deberá estar presente cada vez que la variable sea utilizada.

Variables

- Para declarar una variable se utiliza etiquetas.
- Su ubicación debe estar fuera del flujo del programa (El ensamblador no distingue código de dato).
- Los tipo más comunes se definen con palabras reservadas de la siguiente manera :
  - **.word** palabra de 32 bits
  - **.hword** half word 16 bits
  - **.byte** para byte de 8 bits
  - **.ascii** cadena de caracteres
  - **.asciz** cadena de caracteres con terminación nula

Variables: ejemplos

- Variables de distinto tamaño

```
variable1:    .word 12
variable2:    .hword 12
variable3:    .byte 12
```

El número 12 es guardado ocupando 4, 2 y 1 byte/s respectivamente

- Cadenas

```
cadena1:    .ascii "hola mundo"
cadena2:    .asciz "hola mundo"
```

Igual a cadena1 pero se le agrega un 0 al final

- Arreglos

Los números se guardan sin signo si son positivos y C2 si son negativos

```
vector1:    .hword 2524,54,-588
cadena2:    .byte 'h','o','l','a',0
```

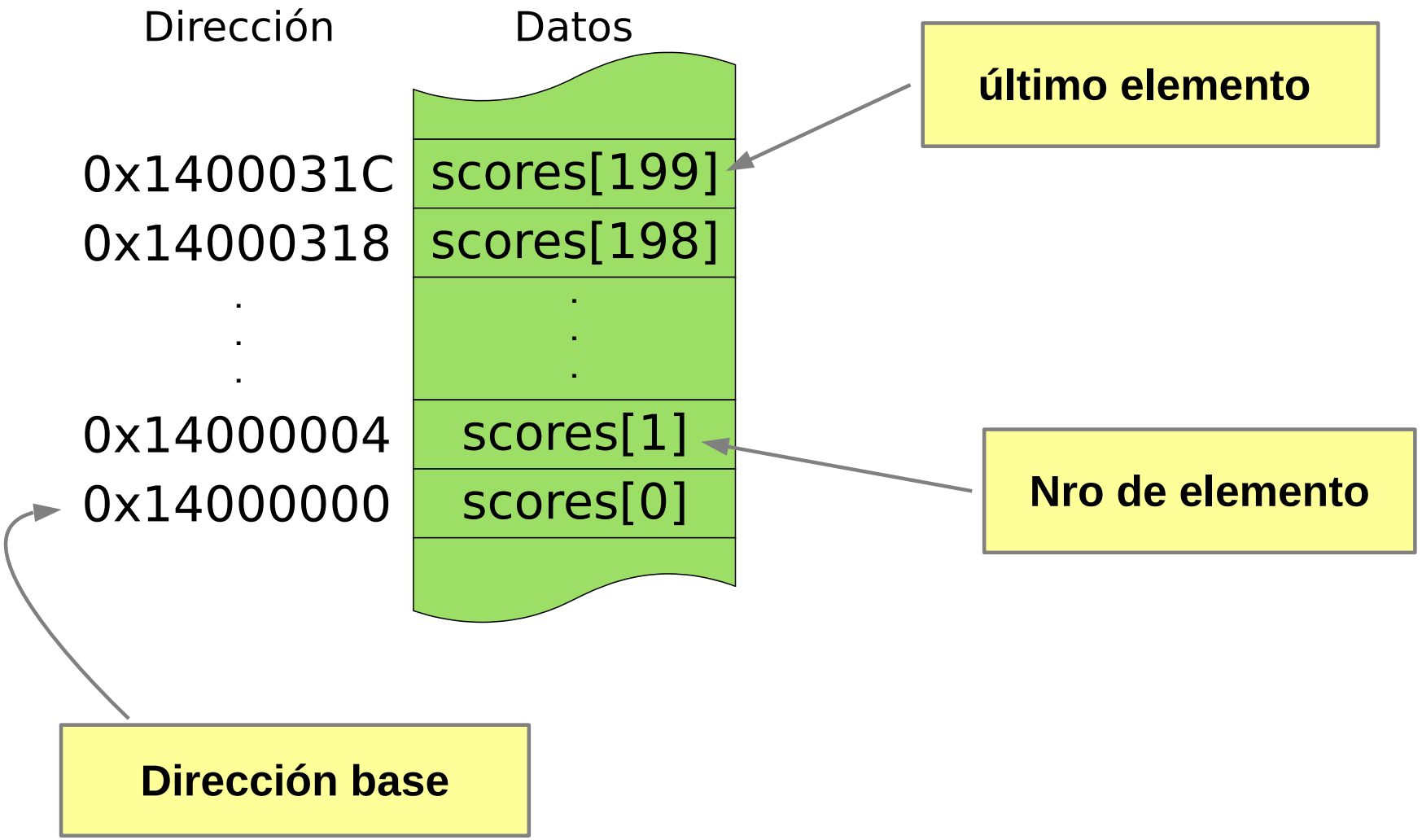
Se pueden guardar el ASCII de un carácter o un número

Variables: Arreglo (*Array*)

- Un arreglo es un conjunto de datos del mismo tipo guardados en una área de memoria de manera contigua o secuencial.
- Los elementos dentro del arreglo se identifica con un indice (*index*).
- La cantidad de elementos del arreglo es su longitud (*lenght*).
- Y comenzará en una dirección base (*base address*).

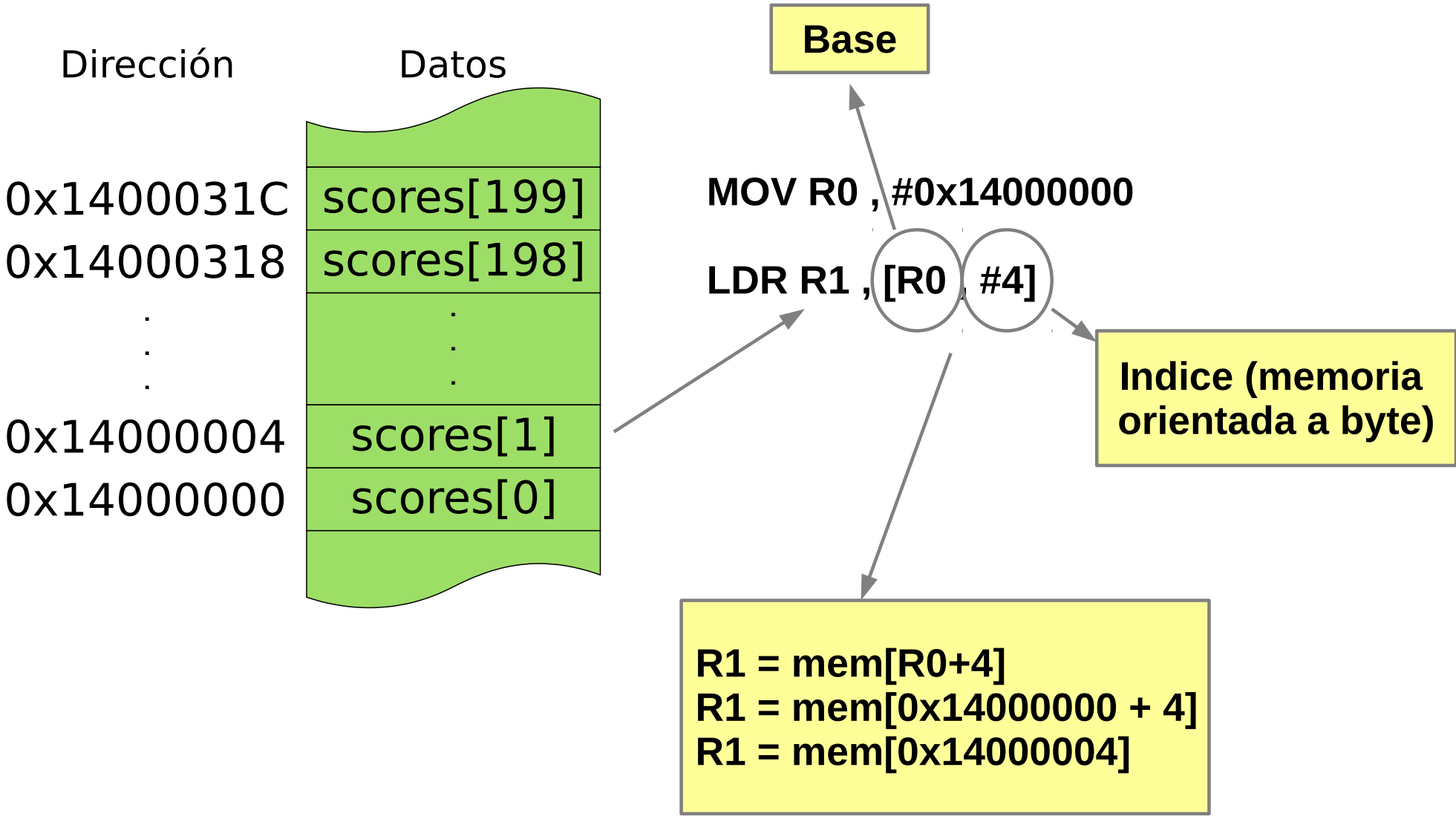
Arreglo (*Array*)

Vector de word de 200 elementos y dirección base 0x14000000



Arreglo (*Array*)

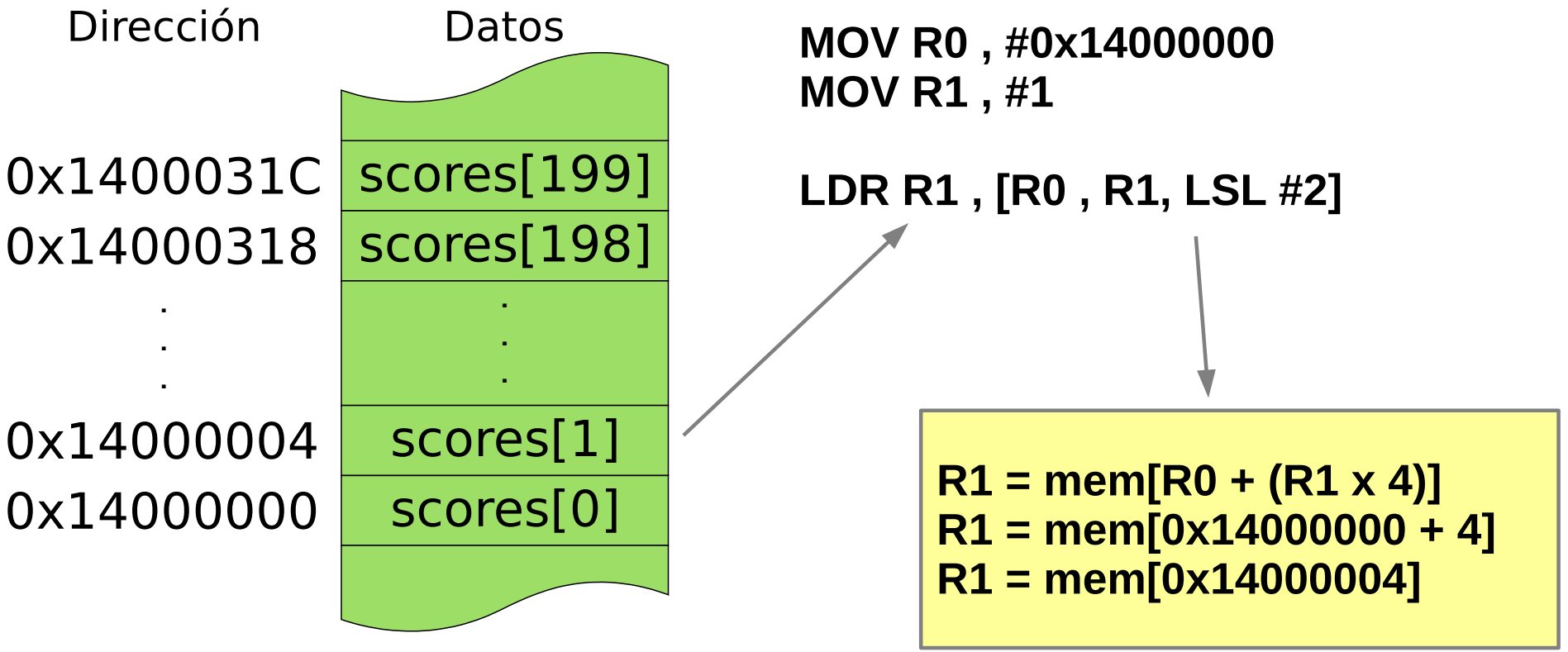
Vector de word de 200 elementos y dirección base 0x14000000



Arreglo (*Array*)

- ARM utilizando offset por registro escalado, permite multiplicar el indice por un una constante, sumando el resultado a la base.

Vector de word de 200 elementos y dirección base 0x14000000



Arreglo (*Array*)

- Ejemplo, sumar 10 a los elementos de un vector

```
Int i;
Int scores[200];
```

```
                                        MOV R0,#0x14000000
for(i=0;i<200;i=i+1) {                  MOV R1,#0
                                LOOP:   CMP R1,#200
                                        BGE L3
    scores[i]=scores[i]+10;             LDR R3, [R0, R1, LSL #2]
                                        ADD R3, R3, #10
                                        STR R3, [R0, R1, LSL #2]
                                        ADD R1,R1,#1
}                                       B LOOP
                                L3:
```

Uso del Indice

- Ejemplo, sumar 10 a los elementos de un vector con post-indexado.

```
Int i;
Int scores[200];
```

```
                                    MOV R0,#0x14000000
                                    ADD R1, R0, #800 ; R1 = R0 + (200x4)
for(i=0;i<200;i=i+1) {      LOOP:   CMP R0,R1
                                    BGE L3
    scores[i]=scores[i]+10;         LDR R3, [R0]
                                    ADD R3, R3, #10
                                    STR R3, [R0], #4
}                                   B LOOP
                            L3:
```

Variables: Bytes y Caracteres

- En lengua natural, los grafemas son unidades mínimas e indivisibles de la escritura.
- El carácter es la unidad de información que representa aproximadamente a un grafema y otros símbolos como caracteres de control, partes de dibujos, etc.
- En una computadora, estos caracteres están codificados, es decir, se asigna un valor generalmente entero a cada carácter.

Codificación

- Actualmente existen dos codificaciones en una computadora
  - ASCII (*American Standard Code for Information Interchange* Código Estándar Estadounidense para el Intercambio de Información).
    Creado en 1963
  - Unicode
    Define un estándar mucho mas amplio que ASCII, cada símbolo esta definido con un nombre, uso, sistema de escritura, direccionalidad, mayúscula, etc.
    Está dividido en planos de 16 bits cada uno

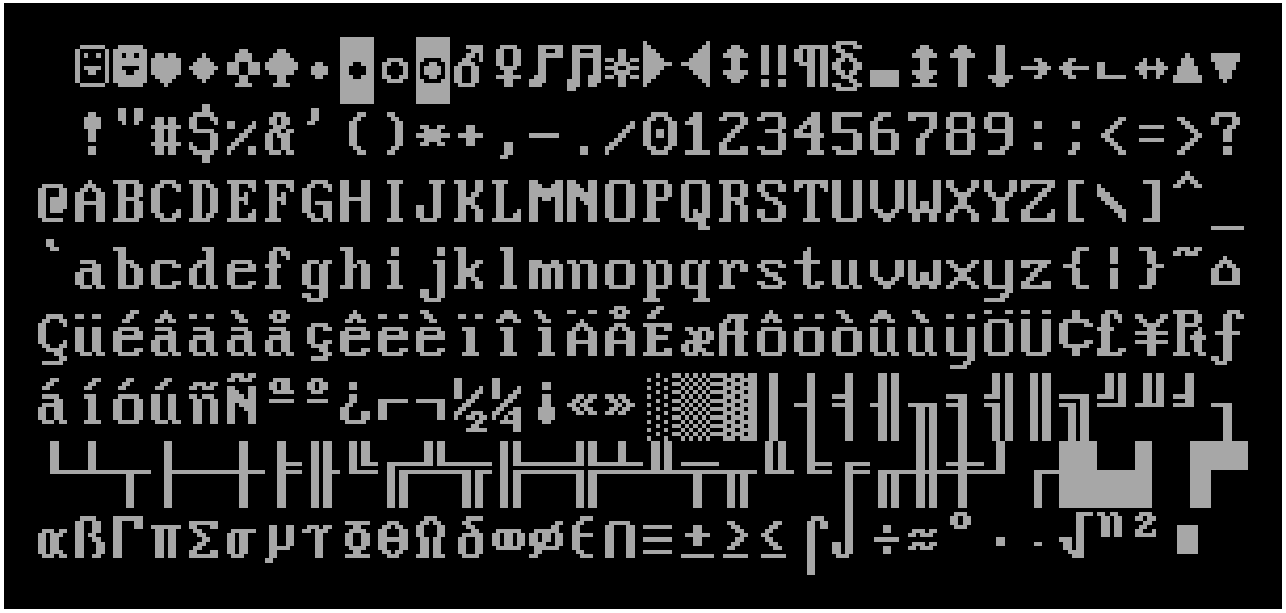
ASCII

- Creado en 1963 como una evolución de los códigos usados en telegrafía.
- En 1967 se incorpora las minúsculas y se redefinen algunos códigos de control (US-ASCII), es publicado por primera vez.
- Posee 32 códigos no imprimibles y 96 imprimibles.
- Es de 7 bits, el octavo bit era usado como bit de paridad o permanecía en 0.

```
 ! " # $ % & ' ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ?
@ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [ \ ] ^ _
 ` a b c d e f g h i j k l m n o p q r s t u v w x y z { | } ~
```

Variante de ASCII (página de códigos 437)

- Incorporada en la primera PC de IBM con el MS-DOS año 1981.
- Algunos códigos no imprimibles pasan a tener una representación gráfica.
- Actualmente es el código básico de cualquier tarjeta de video.
- Mantiene los demás caracteres del ASCII



Variante de ASCII (ISO/IEC 8859-1)

- Incorpora los caracteres necesarios para la mayoría de las lenguas con alfabeto latino.
- Pertenece al conjunto de estándar 8859 que poseen los primeros 128 códigos iguales al ASCII.
- Una versión posterior la 8859-15, agrega el Euro y algunos caracteres extras

¡ ¢ £ ¤ ¥ ¦ § ¨ © ª « ¬ - ® ¯ ° ± ² ³ ´ µ ¶ · ¸ ¹ º » ¼ ½ ¾ ¿ À
Á Â Ã Ä Å Æ Ç È É Ê Ë Ì Í Î Ï Ð Ñ Ò Ó Ô Õ Ö × Ø Ù Ú Û Ü Ý Þ ß
à á â ã ä å æ ç è é ê ë ì í î ï ð ñ ò ó ô õ ö ÷ ø ù ú û ü ý þ ÿ

Tabla ASCII (Caracteres Imprimibles)

32	20	**SP**	48	30	**0**	64	40	**@**	80	50	**P**	96	60	**`**	112	70	**p**
33	21	**!**	49	31	**1**	65	41	**A**	81	51	**Q**	97	61	**a**	113	71	**q**
34	22	**"**	50	32	**2**	66	42	**B**	82	52	**R**	98	62	**b**	114	72	**r**
35	23	**#**	51	33	**3**	67	43	**C**	83	53	**S**	99	63	**c**	115	73	**s**
36	24	**$**	52	34	**4**	68	44	**D**	84	54	**T**	100	64	**d**	116	74	**t**
37	25	**%**	53	35	**5**	69	45	**E**	85	55	**U**	101	65	**e**	117	75	**u**
38	26	**&**	54	36	**6**	70	46	**F**	86	56	**V**	102	66	**f**	118	76	**v**
39	27	**'**	55	37	**7**	71	47	**G**	87	57	**W**	103	67	**g**	119	77	**w**
40	28	**(**	56	38	**8**	72	48	**H**	88	58	**X**	104	68	**h**	120	78	**x**
41	29	**)**	57	39	**9**	73	49	**I**	89	59	**Y**	105	69	**i**	121	79	**y**
42	2A	*****	58	3A	**:**	74	4A	**J**	90	5A	**Z**	106	6A	**j**	122	7A	**z**
43	2B	**+**	59	3B	**;**	75	4B	**K**	91	5B	**[**	107	6B	**k**	123	7B	**{**
44	2C	**,**	60	3C	**<**	76	4C	**L**	92	5C	****	108	6C	**l**	124	7C	**|**
45	2D	**-**	61	3D	**=**	77	4D	**M**	93	5D	**]**	109	6D	**m**	125	7D	**}**
46	2E	**.**	62	3E	**>**	78	4E	**N**	94	5E	**^**	110	6E	**n**	126	7E	**~**
47	2F	**/**	63	3F	**?**	79	4F	**O**	95	5F	**_**	111	6F	**o**			

Tabla ASCII (Caracteres no imprimibles)

0	0	NUL	^@	Carácter Nulo	17	11	DC1	^Q	Ctrol de disp.1 (XON)
1	1	SOH	^A	Inicio de Encabez.	18	12	DC2	^R	Ctrol de disp. 2
2	2	STX	^B	Inicio de Texto	19	13	DC3	^S	Ctrol de disp.3 (XOFF)
3	3	ETX	^C	Fin de Texto	20	14	DC4	^T	Control de dispositivo 4
4	4	EOT	^D	Fin de Transmisión	21	15	NAK	^U	Acuse de recibo neg.
5	5	ENQ	^E	Consulta	22	16	SYN	^V	Síncronía en espera
6	6	ACK	^F	Acuse de recibo	23	17	ETB	^W	Fin del bloque de trans.
7	7	BEL	^G	Timbre	24	18	CAN	^X	Cancelar
8	8	BS	^H	Retroceso	25	19	EM	^Y	Fin del medio
9	9	HT	^I	Tabulación horiz.	26	1A	SUB	^Z	Substitución
10	0A	LF	^J	Salto de línea	27	1B	ESC	ESC	Escape
11	0B	VT	^K	Tabulación Vertical	28	1C	FS	^\	Separador de archivo
12	0C	FF	^L	Avance de página	29	1D	GS	^]	Separador de grupo
13	0D	CR	^M	Retorno de carro	30	1E	RS	^^	Separador de registro
14	0E	SO	^N	Desactivar may.	31	1F	US	^_	Separador de unidad
15	0F	SI	^O	Activar mayúsculas	127	7F	DEL	DEL	Suprimir
16	10	DLE	^P	Esc.vínculo de datos					

- Bibliografía

Harris & Harris. Digital design and computer architecture: ARM edition. Elsevier, 2015. Capítulo 6.

William Hohl & Christopher Hinds. ARM assembly language. Fundamentals and techniques. 2nd edition. CRC press, 2015. Capítulo 5.

¿ Preguntas ?