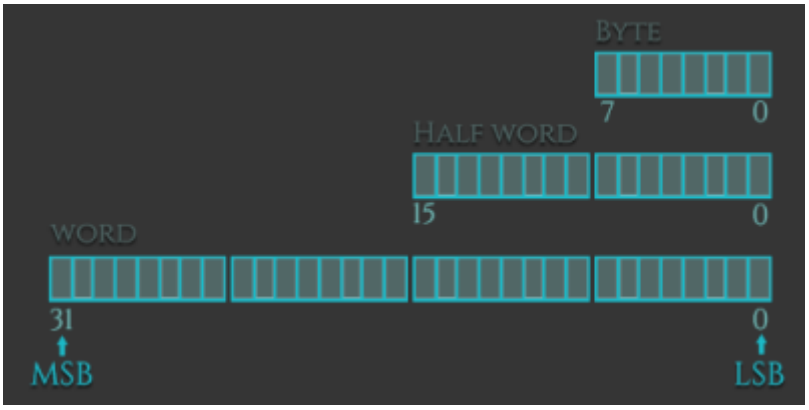
Tipos de Datos

Similar a los lenguajes de alto nivel, ARM admite operaciones de diferentes tipos de datos. Los tipos de datos que podemos cargar (o almacenar) pueden ser palabras (words) con signo y sin signo, medias palabras o bytes. Las extensiones para estos tipos de datos son: **-h** o **-sh** para medias palabras, **-b** o **-sb** para bytes, y ninguna extensión para palabras. La diferencia entre los tipos de datos con y sin signo es:



Los tipos de datos con signo deben contener valores positivos y negativos, esto hace que disminuyan el rango de valores que pueden respresentar. Los tipos de datos sin signo pueden contener valores positivos grandes (incluido “Cero”) pero no pueden contener valores negativos, tienen un rango más amplio.

Estos son algunos ejemplos de cómo se pueden usar estos tipos de datos con las instrucciones Load y Store:

```
ldr   =    Load Word
ldrh  =    Load unsigned Half Word
ldrsh =    Load signed Half Word
ldrb  =    Load unsigned Byte
ldrsb =    Load signed Bytes
```

```
str   =    Store Word
strh  =    Store unsigned Half Word
strsh =    Store signed Half Word
strb  =    Store unsigned Byte
strsb =    Store signed Byte
```

Tipos de datos básicos

En la siguiente tabla se recogen los diferentes tipos de datos básicos que podrán aparecer en los ejemplos, así como su tamaño y rango de representación.

ARM	Tipo en C	bits	Rango
.byte	unsigned char	8	0 a 255
	(signed) char	8	-128 a 127
.hword	unsigned short int	16	0 a 65.535
.short	(signed) short int	16	-32.768 a 32767
.word	unsigned int	32	0 a 4294967296
.int	(signed) int	32	-2147483648 a 2147483647
	unsigned long int	32	0 a 4294967296
	(signed) long int	32	-2147483648 a 2147483647
.quad	unsigned long long	64	0 a 2^{64}
	(signed) long long	64	-2^{63} a 2^{63}-1

Nótese como en ensamblador los tipos son neutrales al signo, lo importante es la longitud en bits del tipo. La mayoría de las instrucciones (salvo multiplicación) hacen la misma operación tanto si se trata de un número natural como si es entero en complemento a dos. Nosotros decidiremos el tipo mediante las constantes que pongamos o según los flags que interpretemos del resultado de la operación.

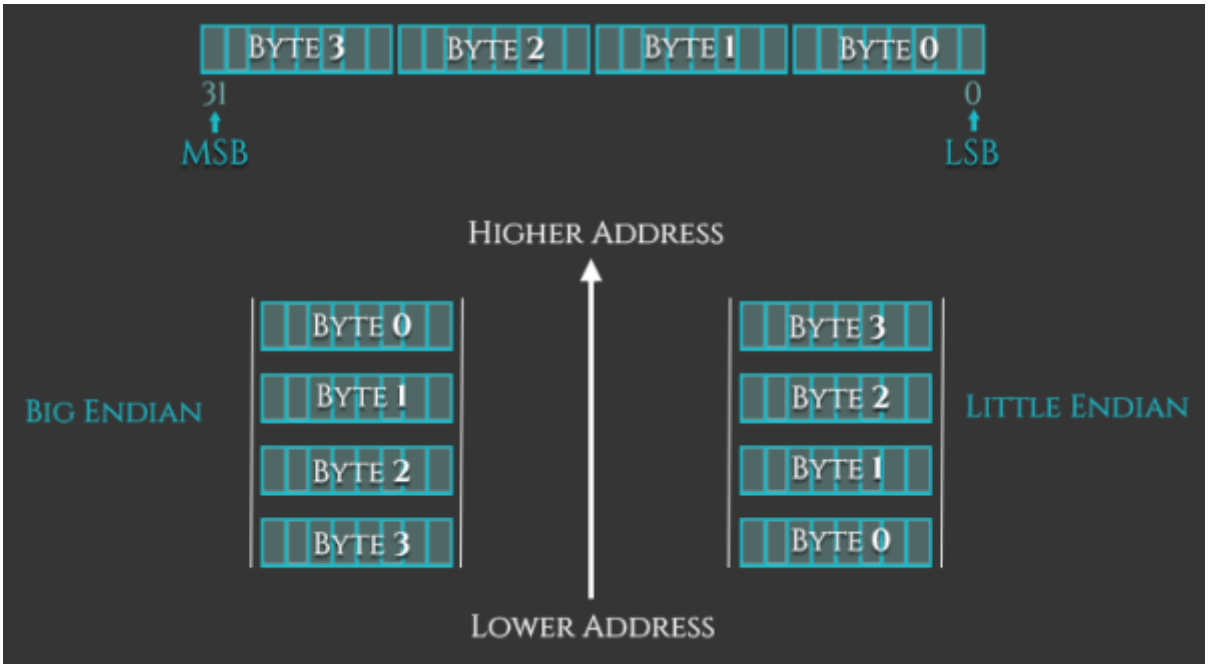
Punteros

Un puntero siempre ocupa 32 bits y contiene una dirección de memoria. En el siguiente ejemplo se puede observar que acceder al dato de var1 nos cuesta 2 ldrs a través del puntero:

```
.data
     var1:   .word 99
.text            @ Defincion de codigo del programa
.global main     @ global, visible en todo el programa
main:
  ldr r2, =var1      /* r2 apunta a var1*/
  ldr r3, [r2]
```

Atención con Bi-Endian

Hay dos formas básicas de ver los bytes en la memoria: Little-Endian (LE) o Big-Endian (BE). La diferencia es el orden de bytes en el que cada byte de un objeto se almacena en la memoria. En máquinas little endian como Intel x86, el byte menos significativo se almacena en la dirección más baja (la dirección más cercana a cero). En las máquinas big-endian, el byte más significativo se almacena en la dirección más baja. La arquitectura ARM era little-endian antes de la versión 3, desde entonces es bi-endian, lo que significa que presenta una configuración que permite la endianness conmutable. En ARMv6, por ejemplo, las instrucciones son little-endian fijas y los accesos a los datos pueden ser little-endian o big-endian según lo controlado por el bit 9, el bit E, del Registro de estado del programa (CPSR).



Volver