TEMA 3
REPRESENTACIÓN DE DATOS

REPRESENTACIÓN DE DATOS

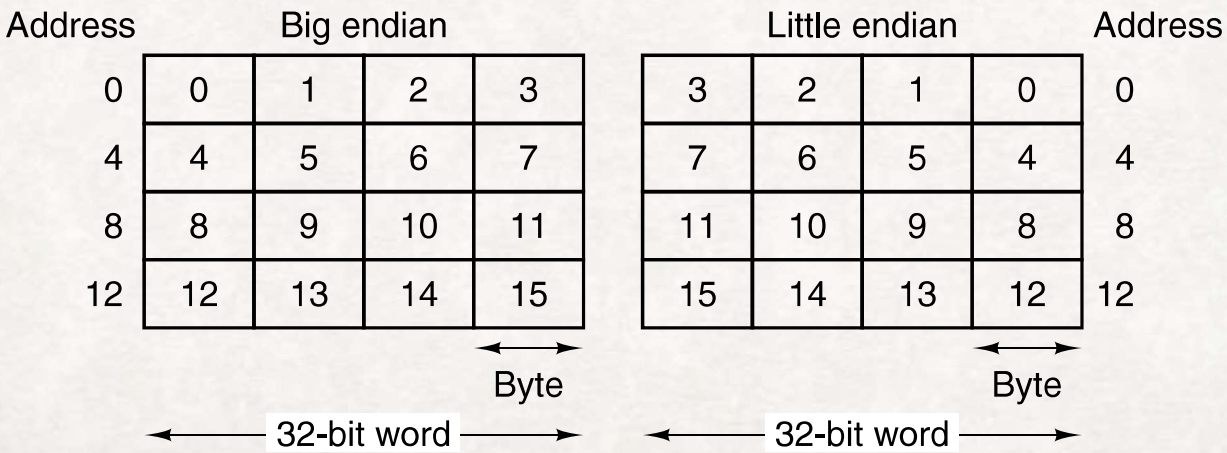
CONTENIDO

- *Big Endian* y *Little Endian*.
- Números enteros:
  - Enteros sin signo y con signo.
  - Detección del rebose (*overflow*) en operaciones aritméticas.
- Números de coma flotante: IEEE 754.
- Codificación de texto:
  - ASCII, ISO 8859.
  - Unicode. Transformaciones UTF-8, UTF-16.

BIG ENDIAN vs. LITTLE ENDIAN

LA "ENDIANNESS WAR"

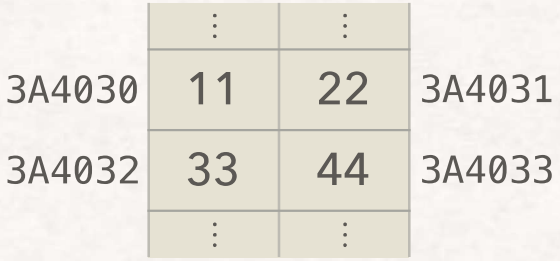
- Los datos de tamaño multi-byte se pueden disponer de dos formas:
  - Empezando (desde la dirección más baja) por el byte más significativo: **Big Endian**.
  - Empezando por el byte menos significativo: **Little Endian**.
- Referencia a las guerras en Liliput sobre por qué extremo se debe empezar a abrir un huevo: el "big end" o el "little end".



BIG ENDIAN vs. LITTLE ENDIAN

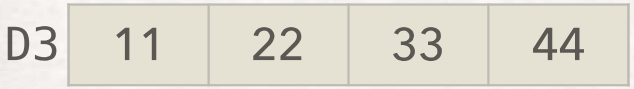
EJEMPLO

Supongamos:



El 68000 es una CPU Big Endian:

MOVE.L 3A4030,D3

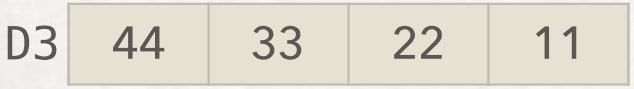


MOVE.W 3A4030,D3

MOVE.B 3A4030,D3

Pero si fuese Little Endian...

MOVE.L 3A4030,D3



MOVE.W 3A4030,D3

MOVE.B 3A4030,D3

BIG ENDIAN vs. LITTLE ENDIAN

¿QUIÉN USA QUÉ?

- Existen CPUs Big Endian, Little Endian y “bi-endian”:
  - Big Endian: Motorola (todo), Freescale ColdFire; IBM POWER, zArchitecture; Sun SPARC, Atmel AVR32…
  - Little Endian: Intel (todo), Zilog Z80, MOS 6502, VAX, Atmel AVR (8 bit)…
  - Bi-endian: IBM/Freescale PowerPC, Alpha, MIPS, PA-RISC, SPARCv9, ARM…
- Internet (específicamente la arquitectura de protocolos TCP/IP) usa Big Endian, por lo que también se conoce como “network order”.

BIG ENDIAN vs. LITTLE ENDIAN

CONSECUENCIAS PERNICIOSAS

- Incompatibilidad de archivos binarios (conteniendo números enteros o de coma flotante en su representación binaria nativa).
- Incompatibilidad de archivos de texto Unicode UTF-16 o UTF-32 salvo que se prefijen con un BOM (*Byte Order Mark*).
- Necesidad de realizar conversiones de Little a Big Endian ("network order") en aplicaciones que manipulan direcciones IP, números de puerto TCP/UDP, y otros protocolos de Internet.
- Y qué es mejor: ¿Big Endian o Little Endian? No hay argumentos de peso a favor o en contra, pero **es evidente que lo correcto es Big Endian** (culpable de herejía quien opine lo contrario).

NÚMEROS ENTEROS

REBOSE (*OVERFLOW*)

- Cuando el resultado de una operación aritmética no se puede representar en el nº de bits utilizado se produce un **rebose** (*overflow*).
- En sumas y restas, el rebose se puede detectar:
  - Aritmética **sin signo**: si el acarreo (*carry*) es 1.
  - Aritmética **con signo**:
    - Suma: si ambos operandos tienen el mismo signo y el resultado tiene signo distinto.
    - Resta: si ambos operandos tienen distinto signo y el signo del resultado coincide con el del sustraendo.
- La multiplicación y la división sí son dependientes del tipo de aritmética (sin o con signo), por lo que habrá operaciones distintas.

NÚMEROS ENTEROS

EJEMPLOS

SUMAS:

```
  0110   6  (+6)
+ 0001   1  (+1)
  0111   7  (+7)  C=0,V=0

  0110   6  (+6)
+ 0010   2  (+2)
  1000   8  (−8)  C=0,V=1

  0110   6  (+6)
+ 1010  10  (−6)
  0000   0  (+0)  C=1,V=0

  1010  10  (−6)
+ 1101  13  (−3)
  0111   7  (+7)  C=1,V=1
```

RESTAS:

```
  0110   6  (+6)
- 0001   1  (+1)
  0101   5  (+5)  C=0,V=0

  1000   8  (−8)
- 0001   1  (+1)
  0111   7  (+7)  C=0,V=1

  0011   3  (+3)
- 0100   4  (+4)
  1111  15  (−1)  C=1,V=0

  0110   6  (+6)
- 1110  14  (−2)
  1000   8  (−8)  C=1,V=1
```

NÚMEROS ENTEROS

INDICADORES DE LA ALU

- Toda CPU posee un registro con *indicadores* de la ALU: bits que expresan qué ha sucedido en la última operación. Típicamente esos indicadores son:
  - **Carry (C)**: Acarreo desde el bit más significativo (tras una suma o resta).
  - **Overflow (V)**: Se ha producido un rebose (V=1) o no (V=0). Tras una suma o resta sólo tiene sentido si usamos aritmética con signo.
  - **Zero (Z)**: El resultado ha sido cero (Z=1) o no (Z=0).
  - **Negative (N)**: Copia del bit más significativo del resultado. Sólo tiene sentido si usamos aritmética con signo.

NÚMEROS ENTEROS

COMPARACIONES ENTRE NÚMEROS

- Toda CPU cuenta con una instrucción de *comparación* entre dos números. Simplemente realiza una resta pero no se almacena el resultado. Su único objetivo es afectar los indicadores de la ALU.
- Tras una instrucción de comparación siempre se programa una instrucción *condicional*, típicamente de *salto* (modificación del PC). Esa instrucción evalúa cierta *condición* y sólo realiza su operación si resulta cierta.
- Este mecanismo es la base para poder tomar decisiones durante la ejecución de un programa, y así implementar estructuras iterativas y de decisión.

NÚMEROS ENTEROS

EJEMPLO: CONDICIONES EVALUABLES POR EL 68000

Condición	Significado	Evaluación
T	True	1
F	False	0
NE	Not Equal	$\overline{Z}$
EQ	Equal	Z
CC/HS	Carry Clear / Higher or Same	$\overline{C}$
CS/LO	Carry Set / Lower	C
HI	Higher	$\overline{C} \cdot \overline{Z}$
LS	Lower or Same	C + Z
VC	Overflow Clear	$\overline{V}$
VS	Overflow Set	V
PL	Plus	$\overline{N}$
MI	Minus	N
GE	Greater or Equal	$\overline{V}\cdot\overline{N} + V\cdot N$
LT	Less Than	$V\cdot\overline{N} + \overline{V}\cdot N$
GT	Greater Than	$(\overline{V}\cdot\overline{N} + V\cdot N) \cdot \overline{Z}$
LE	Less or Equal	$(V\cdot\overline{N} + \overline{V}\cdot N) + Z$

NÚMEROS ENTEROS

EJEMPLO: SELECCIONAR EL MAYOR DE DOS NÚMEROS

Queremos copiar al registro D0 el mayor de los dos números de 8 bits almacenados respectivamente en D1 y D2:

```
                    ⋮
            CMP.B   D1,D2       ;D2–D1
            BHI     COPIA_D2    ;¿D2>D1?
            MOVE.B  D1,D0       ;No: D1→D0
            BRA     CONT        ;y continúa
COPIA_D2    MOVE.B  D2,D0       ;Sí: D2→D0
CONT                ⋮
```

Con BHI $[\overline{C}\cdot\overline{Z}]$ asumimos que los números no tienen signo.
En caso contrario, habría que utilizar BGT $[(\overline{V}\cdot\overline{N}+V\cdot N)\cdot\overline{Z}]$.