Computers Fundamentals

Chapter 5. Data Representation

Objetivos

- Repasar los sistemas binario, octal y hexadecimal y sus relaciones.
- Conocer las operaciones aritméticas básicas en el sistema binario.
- Conocer las distintas formas de representación de la información en el computador.

Índice

- Introducción
- Operaciones binarias básicas
- Representación de números enteros
    - Convenio de representación: Signo y Magnitud
    - Convenio de representación: Complemento a 2
    - Convenio de representación: Exceso Z
- Representación de números reales
    - Coma fija
    - Coma flotante. Formato estándar IEEE 754
- Representación de caracteres

Bibliografía

- Principal
  - Introducción a los Computadores. J. Sahuquillo y otros. Ed. SP-UPV, 1997 (ref. 97.491).
    - Bloques I, II, III y IV
- Recomendable
  - Organización y Diseño de Computadores: La Interficie Circuitería/Programación. D.A. Patterson y J.L. Hennessy. Ed. Reverté.
    - Bloques III y IV
  - Digital Design: Principles and Practices. J.F. Wakerly. Ed. Prentice Hall.
    - Bloque II
- Otros
  - Computer Organization. V.C. Hamacher y otros. Ed. McGraw-Hill.
  - Organización de Computadoras: Un Enfoque Estructurado. A.S. Tanenbaum. Ed. Prentice Hall.
  - Sistemas Digitales. A. Lloris y otros. Ed. McGraw-Hill.

Índice

- Introducción
- Operaciones binarias básicas
- Representación de números enteros
    - Convenio de representación: Signo y Magnitud
    - Convenio de representación: Complemento a 2
    - Convenio de representación: Exceso Z
- Representación de números reales
    - Coma fija
    - Coma flotante. Formato estándar IEEE 754
- Representación de caracteres

Introducción

- Representación externa
  - Empleada por las personas: sistema decimal, caracteres alfanuméricos, gráficos, etc.

- Representación interna
  - Utilizada por el ordenador: sistema binario, código ASCII para los caracteres, etc.

Introducción

- En el tema 1 se abordaron las equivalencias entre los distintos sistemas de numeración:
  - Binario
  - Octal
  - Hexadecimal
  - Decimal
- Las equivalencias entre ellos y los métodos para cambiar de uno a otro.

Sistemas de numeración: recordatorio

- Conceptos a recordar de los sistemas de numeración
  - Sistemas de numeración posicionales: base, factor de escala
  - Sistema binario: sistema posicional de base 2
- Cambios de base: de una base B cualquiera a decimal
  - Desarrollo del polinomio de potencias de la base
    - $a_{-f}\ a_{-f+1}\ \ldots\ a_{-1}$ es la parte fraccionaria (f dígitos)
    - $a_0\ a_1\ \ldots\ a_{e-1}$ es la parte entera (e dígitos)

$$N = \sum_{i=-f}^{e-1} a_i B^i$$

- Cambios de base: de decimal a una base B cualquiera
  - Divisiones sucesivas entre la base B para la parte entera
  - Multiplicaciones sucesivas por la base B para la parte fraccionaria
  - La coma separa en las dos bases la parte entera de la fraccionaria

Ejemplos

Índice

- Introducción
- Operaciones binarias básicas
- Representación de números enteros
    - Convenio de representación: Signo y Magnitud
    - Convenio de representación: Complemento a 2
    - Convenio de representación: Exceso Z
- Representación de números reales
    - Coma fija
    - Coma flotante. Formato estándar IEEE 754
- Representación de caracteres

Natural numbers representation

- Positive natural numbers without sign and without fractional part are denominated **natural numbers**
- Natural numbers are represented by its corresponding binary value
  - They will be named “**natural binary**”

Values

Binary	Natural
0 0 0	0
0 0 1	1
0 1 0	2
0 1 1	3
1 0 0	4
1 0 1	5
1 1 0	6
1 1 1	7

Natural numbers representation

- The representation's range is given by the numbers of bits used to represent natural binary
  - With ***n*** bits the range of representation is: $[0, 2^n - 1]$
  - The number of different values represented is 2^n
- It is possible to overflow the representation range when we make arithmetic operations with natural numbers
  - Overflow means that the result cannot be represented using ***n*** bits

Natural numbers basic arithmetic operations

- The rules applied to natural binary numbers to obtain basic arithmetical operations (add, sub, mult, and div) are very close to the rules used when operating with decimal values. The difference is the base.
- When adding two natural binary numbers using n bits:
  - Carry is generated when the add of the numbers is greater or equal to the base (>= 2 in natural binary, and not >= 10)
- When substraction two natural binary numbers using n bits:
  - Borrow is calculated as: base + minuend – subtrahend (2 + minuend – subtrahend in binary,
    and not 10 + minuend - subtrahend)

Natural numbers basic arithmetic operations

- The add operation can be formalized using a truth table
- add of two natural numbers using 2 bits

0 + 0 =	0
0 + 1 =	1
1 + 0 =	1
1 + 1 =	10 (*carry*=1)

A	B	*Carry*	Add
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

The corresponding circuit to that truth table is known as "Half Adder" (HA):

$S = A \oplus B$, $Carry = A \cdot B$

Natural numbers basic arithmetic operations

- The sub operation can be formalized using a truth table
- Sub of two natural numbers using 2 bits

0 - 0 = 0
0 - 1 = 1 (and 1 borrow bit)
1 - 0 = 1
1 - 1 = 0

A	B	sub	borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Natural numbers basic arithmetic operations

- Procedure to add binary represented using ***n*** bits:
  - The *carry* bit is added to the MSB neighbor

Carry's bits	**0 1 1 1 1 1 0 1**	
+	**0 1 1 0 0 1 0 1**	**addends**
	0 0 1 1 1 1 0 1	
	1 0 1 0 0 0 1 0	**Result**

Natural numbers basic arithmetic operations

- Procedure to subtract binary values represented using ***n*** bits:
  – The borrow bit generated must be substract from the minuend

$$
\begin{array}{rl}
 & 0\,1\,0\,0\,0\,1\,0\,1 \quad \text{Minuend} \\
- & 0\,0\,0\,1\,1\,1\,0\,1 \quad \text{Subtrahend} \\
\text{Borrows} & 0\,0\,0\,1\,1\,0\,0 \\
\hline
 & 0\,0\,1\,0\,1\,0\,0\,0 \quad \text{Resultado}
\end{array}
$$

Overflow on arithmetic operations

- *Overflow*
  - The result is not in the range of representation
    - Can happen in any representation's agreement
    - The number of bits limitations is the responsible of the overflow
    - The overflow indicates that the result is not valid
- Overflow's flag (V o OV)
  - Arithmetic circuit's output used to indicate if an operation's results is valid or not

Overflow on arithmetic operations

- Overflow on natural binary numbers
  - It is detected if the MSB carry bit is 1,
    - Indicates that it is necessary at least one more bit to represent the result
  - Algebraic expression of the Overflow's flag, V = Cn-1
- Example. Binary naturals numbers using 6 bits

$$\begin{array}{r} +\ 44 \\ 10 \\ \hline 54 \end{array} \quad \begin{array}{c} \longrightarrow \\ \longrightarrow \end{array} \quad \begin{array}{r} 101100 \\ +\ 001010 \\ \hline 0110110 \end{array}$$

V=0

$$\begin{array}{r} +\ 44 \\ 24 \\ \hline 68 \end{array} \quad \begin{array}{c} \longrightarrow \\ \longrightarrow \end{array} \quad \begin{array}{r} 101100 \\ +\ 011000 \\ \hline 1000100 \end{array}$$

V = 1

The results can not be represented using 6 bits

Overflow on natural binary numbers

- Example

```
  44  →   101100
− 10  →  −001010
  34      0100010
          V=0

  24  →   011000
− 44  →  −101100
  ??      1101100
          V = 1
```

The results can not be represented using 6 bits

- Exercises
  - Add 101110001_2 y 001110110_2
  - Sub 101110001_2 y 001110110_2
  - Add 101110001_2, 001110110_2 y 011001100_2

Basic binary operations

- Multiplication
  - The result of multiply two binary numbers using ***n*** and ***m*** bits respectively should be represented using ***n+m*** bits to avoid overflow

```
  23        1 0 1 1 1
   7 ×      0 0 1 1 1 ×
 ----     -----------
 161        1 0 1 1 1
          1 0 1 1 1
        1 0 1 1 1
      0 0 0 0 0
    0 0 0 0 0
  -------------------
    0 1 0 1 0 0 0 0 1
```

$= 128+32+1 = 161_{10}$

Basic binary operations

- Multiplication
  - Exercises

```
        1 0 0 1 1
        1 0 1 0 1 ×
        ---------
        1 0 0 1 1
    1 0 0 1 1
1 0 0 1 1
-----------------
1 1 0 0 0 1 1 1 1
```

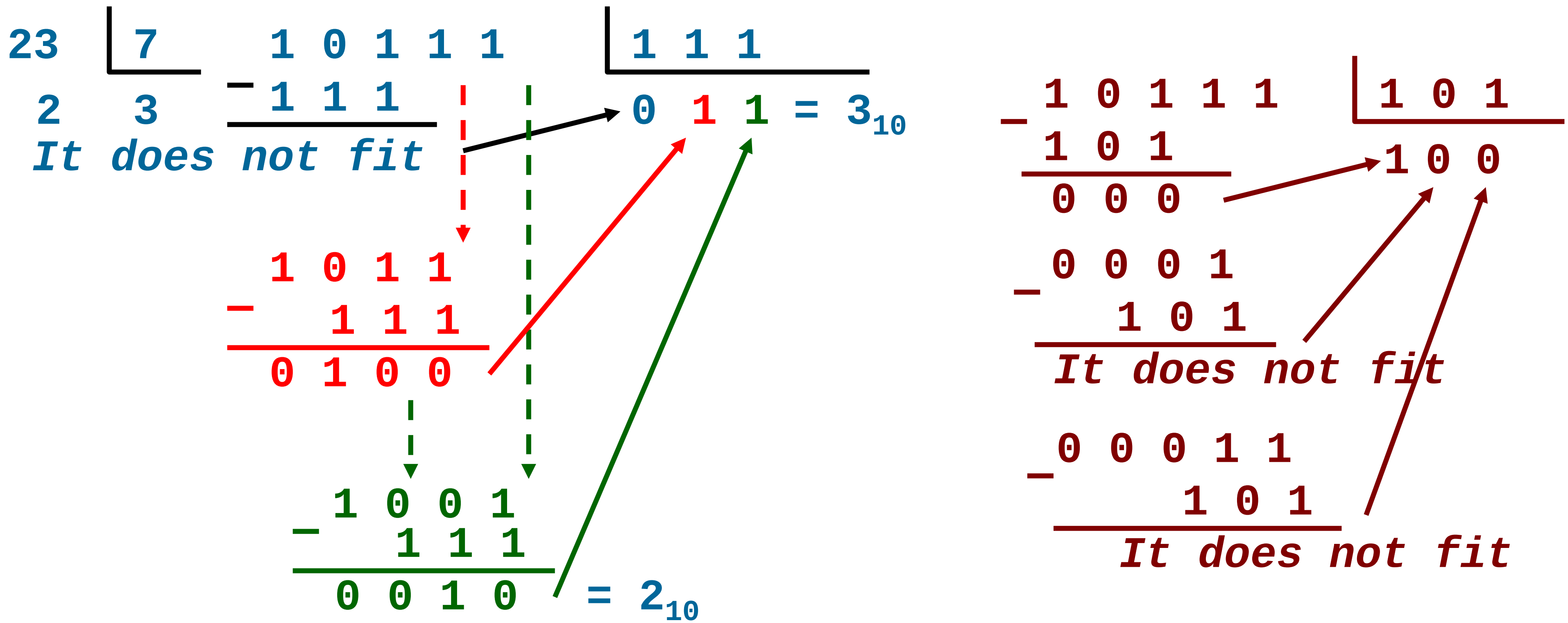
```
        1 0 0 0 1
        1 1 1 0 1 ×
        ---------
        1 0 0 0 1
    1 0 0 0 1
  1 0 0 0 1
1 0 0 0 1
-----------------
1 1 1 1 0 1 1 0 1
```

Basic binary operations

- Division
  - The procedure to divide binary numbers is similar to the procedure to divide decimal numbers.
  - The difference is when the quotient’s bit it is not zero it must be one. In decimal division, the quotient’s digit can be a number between 1 and 9

Basic binary operations

- Division
  - Examples



Basic binary operations

- One's complement of a X natural number using n bits
  - Definition: $C2One(X) = 2^n - X - 1$
  - It can be evaluated inverting each bit of the natural number X
- Two's complement of a X natural number using n bits
  - Definition: $C2Two(X) = 2^n - X$
  - It can be evaluated inverting each bit of the natural number X and adding an extra 1.
  - It can be evaluated also as $C2Two(X) = C2One(X) + 1$

Basic binary operations

- Complement operations are comp`lementaruy operations
  - C2One(C2One(X)) = X
    - C2One(C2One(X)) = 2^n - (2^n - X - 1) - 1 = 2^n - 2^n + X + 1 - 1 = X
  - C2Two(C2Two(X)) = X
    - C2Two(C2Two(X)) = 2^n - (2^n - X) = 2^n - 2^n + X = X
- Exercises:
  - C2One(111000101) = **000111010**
  - C2One(000111010) = **111000101**
  - C2Two(111000101) = **000111011**
  - C2Two(000111011) = **111000101**

Integers representation

- Integer numbers
  – Numbers with sign but without fractional part
- Representation's problem
  – Computers were designed to store bits on its memory
  – When storing a integer number the sign is not stored as "+" or "-"

Integers representation

- Solution
  - Define agreement’s representation which define the arbitrary rules to store positive and negative values using strings of ones and zeros
- Sign extension
  - Capacity to extend in a simple way the number of bits used to represent a binary value mantaining the criterion used on the agreement’s representation.
  - The phrase “in a simple way” means without aplying complicated arithmetical operations

Integers representation

- Criterion 1: Sign and Magnitude
  - The MSB is reserved to represent the sign of the binary number
    - MSB = 0 for positive numbers
    - MSB = 1 for negative numbers
  - The rest of bits represents the absolute value of the binary number (expressed as a natural binary but using n-1 bits)
  - Symmetrical representation's range [- (2^{n-1}-1), + 2^{n-1}-1]
  - Problem: there are two different string for representing the value zero (One positive and one negative)
  - Before adding or subtracting it is necessary to analyze the sign of the operands

Integers representation

- Criterion 1: Sign and Magnitude (SM)
  - Sign extension is made just adding zeros between the bit sign and the magnitude bits

```
-3 =>   1 11          +3 =>   0 11
        1 011                 0 011
        1 0011                0 0011
        1 00011               0 00011
```

SM	Decimal
0 000…000 0 000…001 0 000…010 0 000…011 … 0 111…110 0 111…111	+0 +1 +2 +3 … $+(2^{n-1} - 2)$ $+(2^{n-1} - 1)$

SM	Decimal
1 000…000 1 000…001 1 000…010 1 000…011 … 1 111…110 1 111…111	-0 -1 -2 -3 … $-(2^{n-1} - 2)$ $-(2^{n-1} - 1)$

Integers representation

- Criterion 2: Two's Complement (C2Two)
  - We do not have to confuse this representation 's agreement with the mathematical operation ***C2Two***
  - Depending on the sign of the number: there are used two different procedures to represent a number with ***n*** bits:
    - Positive numbers are represented in sign and magnitude
    - Negative numbers are represented by the result of the mathematical operation C2Two (positive value)
  - Example: Using 5 bits. Represent +4 y -4

+4 with 4 bits = 0100 (magnitude) adding sign: 00100_{2c} $=+4_{2c}$

$-4 = \text{C2Two}(+4) = \text{C2Two}(00100) = 11100_{2c} = -4_{2c}$

Integers representation

- Criterion 2: Two's complement
  - The MSB indicates the sign
    - MSB=0 for positive numbers
    - MSB=1 for negative numbers
  - Asymmetrical representation's range $[-2^{n-1}, +2^{n-1}-1]$
  - There is only one representation for zero

C2Two	Decimal
0000…000	+0
0000…001	+1
0000…010	+2
0000…011	+3
…	…
0111…110	$+(2^{n-1} - 2)$
0111…111	$+(2^{n-1} - 1)$

C2Two	Decimal
1000…000	-2^{n-1}
1000…001	$-(2^{n-1} - 1)$
…	…
1111…100	-4
1111…101	-3
1111…110	-2
1111…111	-1

Integers representation

- Criterion 2: Two's complement
  - The result of Add and Sub operations do not need changes.
    - Therefore it is the criterion more used when operating with integer numbers

```
-2 => 110          +2 => 010
      1110               0010
      11110              00010
      111110             000010
```

Integers representation

- Adding and substracting using Two's complement
  - Given two integer numbers A and B (positive or negatives) represented following the representation's agreement of Two's complement
  - To evaluate R=A+B and obtain R following the representation's agreement of Two's complement, it is necessary to add the operands and ignore the final carry
  - To evaluate R=A-B and obtain R following the representation's agreement of Two's complement, it is necessry to evaluate the binary sum of A + C2Two(B), The final carry is ignored
  - A - B = A + (-B) = A + C2Two(B)

Integers representation

- Exercise: Given A=-20 and B=+10. Using 6 bits and following the representation's agreement of two's complement: Obtain the result of the following operations:

- A+B

```
+  -20  →  101100  +
   +10  →  001010
   ----     ------
   -10    0 110110
```

- A-B

```
 −  -20  ────────────────────→  101100  −
    +10  →  C2Two(001010)  →   110110
    ----                        ------
    -30                       1 100010
```

- B-A

```
 −  +10  ────────────────────→  001010  −
    -20  →  C2Two(101100)  →   010100
    ----                        ------
    +30                       0 011110
```

Overflow on C2Two operations

- When operating with numbers represented with the representation's agreement of *C2Two* the overflow can:
  - Happen when the result can not be represented using n bits. The result is out of the representation's range.
  - Humans beings can detect the overflow just looking the MSB of the result.
    - If adding two positive numbers the result is One there is overflow and the result is not valid.
    - If adding two negative numbers the result is Zero there is overflow and the result is not valid.
  - Computers need a circuit designed to detect the overflow.

Overflow on C2Two operations

$$A + B = \begin{array}{ccccc} & C_{n-1} & C_{n-2} & \ldots & \\ & & A_{n-1} & A_{n-2} & \ldots & A_0 \\ & & B_{n-1} & B_{n-2} & \ldots & B_0 + \\ \hline & \cancel{C_{n-1}} & R_{n-1} & R_{n-2} & \ldots & R_0 \end{array}$$

A_{n-1}	B_{n-1}	C_{n-2}	C_{n-1}	R_{n-1}	
0	0	0	0	0	A and B >= 0
0	0	1	0	1	Overflow. if R < 0
0	1	0	0	1	A>=0 and B<0 or Viceversa
0	1	1	1	0	Overflow is impossible
1	0	0	0	1	
1	0	1	1	0	
1	1	0	1	0	A and B < 0
1	1	1	1	1	Overflow. if R>=0

$$V = C_{n-1} \oplus C_{n-2}$$

Overflow on C2Two's operations

Examples: Using 6 bits obtain the result of the following operations

```
        V=1
        010000
+ +17 →  010001 +
  +16 →  010000
  ---    -------
  +33   0100001
          R=-31???
```

```
                                   V=0
                                   110000
— +17 ──────────────────────────→   010001 +
  +16 → C2Two(010000) ─────────→   110000
  ---                              -------
  +01                              1000001
                                     R=+1
```

```
        V=1
        100000
+ -17 →  101111 +
  -16 →  110000
  ---    -------
  -33   1011111
          R=+31???
```

```
                                   V=0
                                   000000
— -17 ──────────────────────────→   101111 +
  -16 → C2Two(110000) ─────────→   010000
  ---                              -------
  -01                              0111111
                                     R=-1
```

Integers representation

- Criterion 3: Excess to Z
  - An arbitrary value denominated excess (z) is chosen to be the zero value.
  - Any integer A is represented by the natural binary A+Z.

  - Example: Represent +4 y -4 using 6 bits using criterion excess to 31

$$+4_{10} = 31 + 4 = 35 = 100011_{z31}$$
$$-4_{10} = 31 - 4 = 27 = 011011_{z31}$$

Integers representation

- Criterion 3: Excess to Z
  - Asymmetrical range of representation [- Z, + 2^n - 1 - Z]
  - Each Z's value defines a different representation's range
  - It is recommended to assign to Z the value $2^{n-1} - 1$
  - There is only one zero's representation

Excess to 2^{n-1}-1	Decimal
0000…000	$-(2^{n-1} - 1)$
0000…001	$-(2^{n-1} - 2)$
0000…010	$-(2^{n-1} - 3)$
0000…011	$-(2^{n-1} - 4)$
…	…
0111…110	-1
0111…111	0

Excess to 2^{n-1}-1	Decimal
1000…000	+1
1000…001	+2
…	…
1111…100	$+(2^{n-1} - 3)$
1111…101	$+(2^{n-1} - 2)$
1111…110	$+(2^{n-1} - 1)$
1111…111	$+2^{n-1}$

Integers representation

- Criterion 3: Excess to Z
  - The MSB does not indicates the sign.
  - The advantage of this criterion is that the represented values are ordered of minor to greater.
    - This allows to compare two numbers with a simple circuit and without conducting operations arithmetical
  - It is not possible to make sign's extension without evaluating mathematical operations
    - Normally the value of the excess depends on the amount of available bits

Integers representation

- Criterion 3: Excess to Z
  - When adding or substracting the result must be adjusted

$$
\begin{array}{rl}
+ & A \\
 & B \\
\hline
 & A+B
\end{array}
\quad
\begin{array}{c}
\longrightarrow \\
\longrightarrow
\end{array}
\quad
\begin{array}{rl}
 & A+Z \\
+ & B+Z \\
\hline
 & A+B+2Z \\
- & Z \\
\hline
 & A+B+Z
\end{array}
\qquad\qquad
\begin{array}{rl}
- & A \\
 & B \\
\hline
 & A-B
\end{array}
\quad
\begin{array}{c}
\longrightarrow \\
\longrightarrow
\end{array}
\quad
\begin{array}{rl}
 & A+Z \\
- & B+Z \\
\hline
 & A-B \\
+ & Z \\
\hline
 & (A-B)+Z
\end{array}
$$

Integers representation

- Example (using n=6 bits)

Decimal	*S/M*	*C2Two*	*Excess to 31*
0	0 00000	000000	011111
+1	0 00001	000001	100000
-1	1 00001	111111	011110
+5	0 00101	000101	100100
-5	1 00101	111011	011010
+31	0 11111	011111	111110
-31	1 11111	100001	000000
+32	----	----	111111
-32	----	100000	----
+33	----	----	----
-33	----	----	----
	[-31,+31]	*[-32,+31]*	*[-31,+32]*

Integers representation

- Summary

	S/M	C2Two	Excess to $2^{n-1}-1$
Range	$[-(2^{n-1}-1), +2^{n-1}-1]$	$[-2^{n-1}, +2^{n-1}-1]$	$[-(2^{n-1}-1), +2^{n-1}]$
Sign ext.	Yes (carefully)	Yes	No
Disadvantages	Complicated add and subb +0 and -0		Complcated add and subb
Advantages	Easy	Easy to operate	Easy to compare

Representation of real numbers

- Two ways of representing real numbers :
  - Fixed point
  - Floating point
- Fixed point: ***n*** bits are used for the integer part and ***p*** bits are used for the fractional part. The number of bits used fro the integer and the fractional part are fixed.
  - $R = i_{n-1}\ i_{n-2} \dots i_1\ i_0\ ,\ f_{-1}\ f_{-2} \dots f_{-p+1}\ f_{-p}$
  - Problems due to the fixed length fields:
    - *Using 3 digits for the fractional part and 3 digits for the fractional part obtain the product* $000{,}125_{10} \times 000{,}001_{10}$
    - $000{,}125_{10} \times 000{,}001_{10} = 000{,}000125_{10}$ = *truncated* = $000{,}000_{10}$

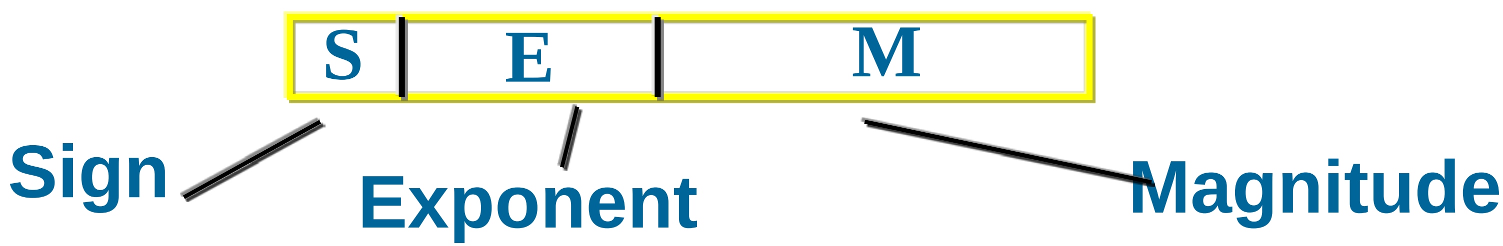
Representation of real numbers

- A real number R is represented by the expression:

$$R = M \times B^{E}$$

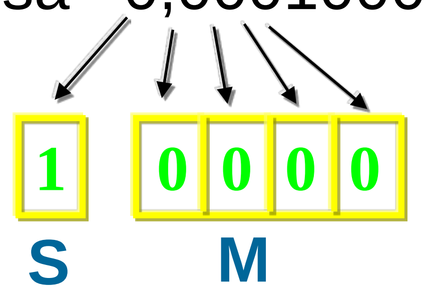
  - M is the mantissa, represented using q bits , fractional, fixed point and sign (SIM): $m_{q-1}\ m_{q-2}\ \ldots\ m_1\ m_0$
  - B is the base of the system
  - E is the exponent, represented using p bits, and generally represented on excess to $2^{p-1} – 1$: $e_{p-1}\ e_{p-2}\ \ldots\ e_1\ e_0$

– In computer’s memory the representation is made as:



Representation of real numbers

- Floating point
  - The arithmetic with floating point values is more complex than arithmetic with integers
    - There are specific circuits(floating point unit, FPU: Floating Point Unit)
- Normalization of the mantissa
  - The aim of normalizing the mantissa is not to lose significant bits (1) wasting space to store non-significant bits (0).
  - Example: store the mantissa –0,00010001 using 5 bits

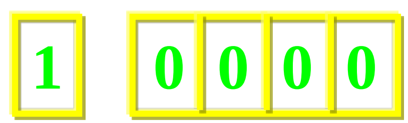


Representation of real numbers

- Normalization of the mantissa
  - By normalizing the mantissa representation ensures that it is possible to store the largest possible number of ones.
  - The first significant bit (1) is placed around the comma, as a result you have to modify the value of the exponent
  - Example: normalized mantissa: $-1{,}0001 \times 2^{-4}$
  - If the first significant bit is placed to the right of the decimal, the names will have the form 0,1 xx ... x. If placed on the left, will have the form 1, xx ... x

Representation of real numbers

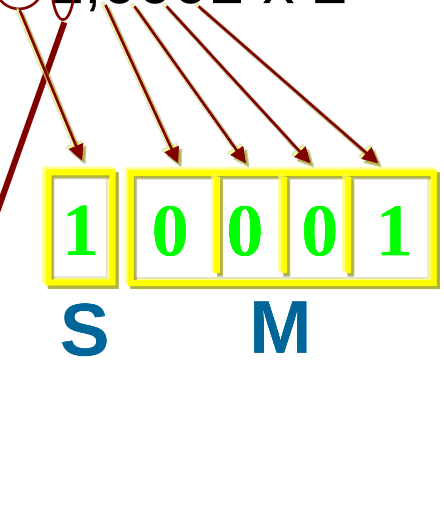
- Normalization of the mantissa
  - By normalizing the mantissa representation ensures that store the largest possible number of ones.
  - The first significant bit (1) is placed around the comma, to move you have to modify the value of the exponent
  - Example: normalized mantissa : $-1{,}0001 \times 2^{-4}$
  - If the first significant bit is placed to the right of the decimal, the names will have the form 0,1 xx ... x. If placed on the left, will have the form 1, xx ... x



Representation of real numbers

- Implicit leading bit
  - The position of the first significant bit does not have to be stored (although it must be taken into account on operations)
  - Example: normalized mantissa = –1,0001 x 2^{-4}

Representation in memory :



It is not stored, it is always 1!

Rep. of real numbers: standard IEEE 754

- IEEE 754 standard format
  - Created to facilitate data transfer between different machines
  - Two versions:

Precision	Total bits	Sign	Exponent	Magnitude
Simple	32	1	8	23*
Double	64	1	11	52*

*There is one additional bit (implicit leading bit)

Rep. of real numbers: standard IEEE 754

- **Main characteristics**
  - The mantissa is normalized to 1, xx ... x.
    It is represented in sign magnitude and the technique of leading implicit bit is used
  - The exponent is written in excess to 2^{n-1}–1 (127 or 1023).
  - The fields are stored in the following order: S, E and M. The base is 2.
  - Numbers represented as : $\pm 1,M \times 2^{-127}$ are called “Normal numbers”
  - Numbers represented as $\pm 0,M \times 2^{-126}$ are called denormalized numbers.

Rep. of real numbers: standard IEEE 754

Example:

- $-81{,}375_{10}$ represented using the IEEE format of single precision (standard format)
  - Absolute value of the integer part: $81_{10} = 1010001_2$
  - Absolute value of the fractional part: $0{,}375_{10} = 0{,}011_2$
  - Real Number = $-1010001{,}011_2$
  - Normalized real number= $-1{,}010001011_2 \times 2^{+6}$
  - Exponent = 6_{10}
  - Exponent in excess to 127 = $127_{10}+6_{10}=133_{10} = 10000101_2$

1 bit	8 bits	23 bits
1	10000101	01000101100000000000000
31	30 … 23	22 … 0

Rep. of real numbers: standard IEEE 754

- Exercise
  - The representation of 5,6875 in IEEE 754 single precision is
    - $+5{,}6875_{10} = 101{,}1011_2 = 101{,}1011 \times 2^0 = 1{,}011011 \times 2^2$

S	E	Mg
0	10000001	01101100000000000000000

Rep. of real numbers: standard IEEE 754

- Exercicises
  - If the sequence of bits (in hexadecimal) is 0xC0B60000, it is representing the value

S	E	Mg
1	10000001	01101100000000000000000

-5,6875

Rep. of real numbers: standard IEEE 754

- Special cases of standard format
  - If E = 00000000 and M = 0, it is representing the value 0, which can not be represented using the standard format
  - If E = 00000000 and M != 0, it is representing a very small value (denormalized numbers) : $\pm 0,M \times 2^{-126}$
  - If E = 11111111 and M = 0, it is being represented the values $+\infty$ o $-\infty$ according to the sign
  - If E = 11111111 and M != 0, It is being represented special results as for example: 0÷0, ½ , pi, etcetera. Such results are known as NaN ("Not a Number")
- Sequences 00000000 and 11111111 in the exponent are not used to represent "normal" numbers. They are used for the special cases of the standard

Rep. of real numbers: standard IEEE 754

- Rang of representation
  - Determined by the number of bits assigned to E
- Precision: distance between two consecutive values that can be represented
  - Determined by the number of bits allocated to M
- Possible overflows when representing values
  - *Overflow*. When the number is so far away from 0 (very large absolute value) that can not be represented
  - *Underflow*. When the name is so close to the value 0 (very small absolute value) that can not be represented

Rep. of real numbers: standard IEEE 754

- Range of standard number representation
  - The original range of the exponent should be [-127, +128] (or [-1023, +1024]) , but because of special cases:
    - The range is [-126, +127] (or [-1022, +1023])
  - In absolute value, the smallest number that can be represented using single precision is $1{,}00\ldots00_2 \times 2^{-126}$
  - In absolute value, the largest number that can be represented using single precision is $1{,}11\ldots111_2 \times 2^{+127} \approx$ $1{,}0 \times 2^{+128}$

Rep. of real numbers: standard IEEE 754

- Representation's Range of denormalized numbers
  - The denormalized numbers provide another set of numbers with the exponent fixed to:
    -126 (or -1022)
  - In absolute value, the smallest number that can be represented in single precision is:

    $0{,}00\ldots001_2 \times 2^{-126} = 1{,}0 \times 2^{-126-23} = 1{,}0 \times 2^{-149}$
  - In absolute value, the largest number that can be represented in single precision is:

    $0{,}11\ldots111_2 \times 2^{-126} \approx 1{,}0 \times 2^{-126}$

    which coincides with the start of the representation's range of normalized numbers

Rep. of real numbers: standard IEEE 754

- Representation's range of single precision

$$0 \cup \left[\pm 2^{-149}, \pm 2^{-126}\right[\cup \left[\pm 2^{-126}, \pm 2^{+128}\right[\approx \underline{\underline{\left[\pm 2^{-149}, \pm 2^{+128}\right]}}$$

Character representation

- Characters
  - Letters (“a”, …, ”z”, “A”, …, “Z”)
  - Digits (“0”, …, “9”)
  - Punctuation’s signs (“.”, “,”, “;”, …)
  - Special symbols (“*”, “&”, “$”, …)
- Characters are represented assigning a numerical code to each characters. A table is used to show the numerical codes used to represent characters
- The computer works with the codes, never with graphical symbols

Character representation

- The features of a representation are
  - Bit length of codes
  - Number of characters that can be represented
  - The assignment of codes to each character (character table)
- A.S.C.I.I. (American Standard Code for Information Interchange)
  - Fixed length, equal for all codes
  - Length of 7-bit
  - ASCII extended. Expansion for international characters with a length of 8-bit

Character representation

- E.B.C.D.I.C. (Extended Binary Coded Decimal Interchange Code)
  - Created in 1964 for the IBM 360
  - Fixed length of 8 bits
  - Is used only in some mainframe systems
- Unicode: Universality, Uniformity, Uniqueness
  - Word processing in different languages
  - Texts in dead languages and other disciplines
  - Lenght of 8, 16 or 32 bits

Character representation

- ASCII table (7 bits)

	0	16	32	48	64	80	96	112
+0	*NUL*	*DLE*	*SP*	0	@	P	`	p
+1	*SOH*	*DC1*	!	1	A	Q	a	q
+2	*STX*	*DC2*	"	2	B	R	b	r
+3	*ETX*	*DC3*	#	3	C	S	c	s
+4	*EOT*	*DC4*	$	4	D	T	d	t
+5	*ENQ*	*NAK*	%	5	E	U	e	u
+6	*ACK*	*SYN*	&	6	F	V	f	v
+7	*BEL*	*ETB*	'	7	G	W	g	w
+8	*BS*	*CAN*	(	8	H	X	h	x
+9	*HT*	*EM*	)	9	I	Y	i	y
+10	*LF*	*SUB*	*	:	J	Z	j	z
+11	*VT*	*ESC*	+	;	K	[	k	{
+12	*FF*	*FS*	,	<	L	\	l	|
+13	*CR*	*GS*	-	=	M	]	m	}
+14	*S0*	*RS*	.	>	N	^	n	~
+15	*S1*	*US*	/	?	O	_	o	*DEL*

The ASCII code of “z” is

112 + 10 = 122

Character representation

- ASCII table (7 bits)

	0x00	0x10	0x20	0x30	0x40	0x50	0x60	0x70
+0	*NUL*	*DLE*	*SP*	0	@	P	`	p
+1	*SOH*	*DC1*	!	1	A	Q	a	q
+2	*STX*	*DC2*	"	2	B	R	b	r
+3	*ETX*	*DC3*	#	3	C	S	c	s
+4	*EOT*	*DC4*	$	4	D	T	d	t
+5	*ENQ*	*NAK*	%	5	E	U	e	u
+6	*ACK*	*SYN*	&	6	F	V	f	v
+7	*BEL*	*ETB*	'	7	G	W	g	w
+8	*BS*	*CAN*	(	8	H	X	h	x
+9	*HT*	*EM*	)	9	I	Y	i	y
+A	*LF*	*SUB*	*	:	J	Z	j	z
+B	*VT*	*ESC*	+	;	K	[	k	{
+C	*FF*	*FS*	,	<	L	\	l	|
+D	*CR*	*GS*	-	=	M	]	m	}
+E	*S0*	*RS*	.	>	N	^	n	~
+F	*S1*	*US*	/	?	O	_	o	*DEL*

The ASCII code of “z” is 0x70+ A = 0x7A

Computers Fundamentals

Subject 6. Data Representation