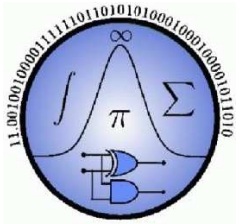
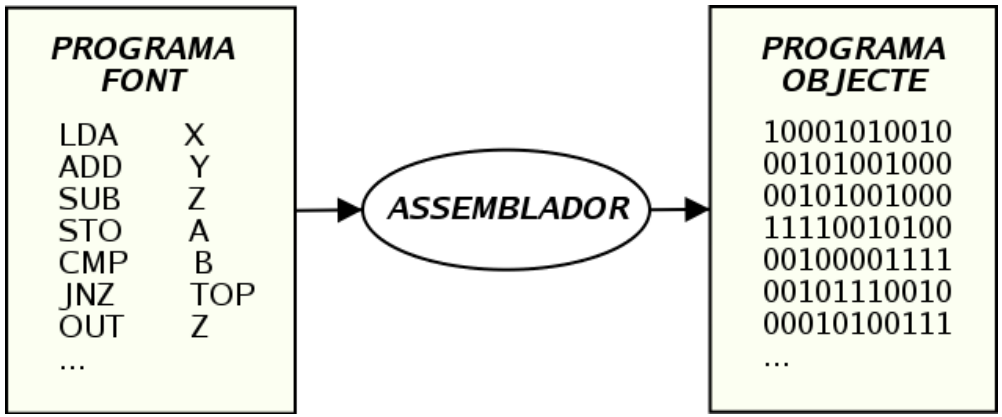


REPRESENTACIÓ DE DADES A MEMÒRIA

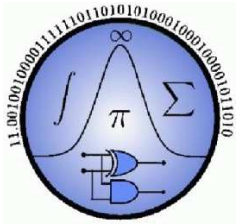
- LA MEMÒRIA D'UN COMPUTADOR CONTÉ SEQÜÈNCIES DE BITS
  - AQUESTES SEQÜÈNCIES TENEN SIGNIFICAT:
    - **PROGRAMES**
      - INSTRUCCIONS ASSEMBLADOR CODIFICADES
    - **DADES**
      - CADENES DE CÀRACTERS CODIFICADES (p.e. ASCII)
      - NÚMEROS ENTERS
        - Amb signe, sense signe, BCD, (de 8,16, 32 bits, etc...)
      - NÚMEROS REALS
        - Coma fixe, coma flotant (de simple o doble precisió)

PROGRAMES

- ELS MNEMÒNICS ASSEMBLADOR ES TRADUEIXEN A LLENGUATGE MÀQUINA

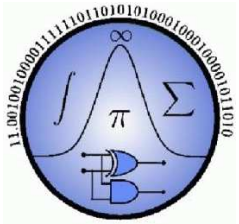


- ELS BITS QUE HI HA A MEMÒRIA SERAN CONSIDERATS INSTRUCCIONS QUAN EL ***PC*** HI APUNTA PER FER UN ***FETCH***



CADENES DE CARÀCTERS

- CODIFICACIÓ DELS CARÀCTERS MITJANÇANT NÚMEROS
  - **CODI ASCII** (7/8 bits per caràcter)
    - 128 Caràcters fixes [0..127]
      - '1' → 49
      - 'A' → 65
    - 128 Caràcters variables [128..255] (ASCII Extès)
  - **UNICODE** (16 bits per caràcter)
    - Cada caràcter és fixe independentment de tot
    - Bo perquè hi ha idiomes amb més de 255 caràcters (Japonès). **UNICODE** els representa **TOTS** de cop**.**

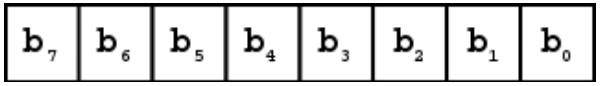


REPRESENTACIÓ NUMÈRICA

- DOS TIPUS BÀSICS DE NÚMEROS
  - **ENTERS**
    - SENSE SIGNE *(Utilitzat)*
    - AMB SIGNE
      - MÒDUL + SIGNE
      - COMPLEMENT A 1
      - COMPLEMENT A 2 *(Utilitzat)*
  - **REALS**
    - COMA FIXE *(Poc utilitzat)*
    - COMA FLOTANT *(Utilitzat)*

ENTERS SENSE SIGNE

- **REPRESENTACIÓ (amb 8 bits)**

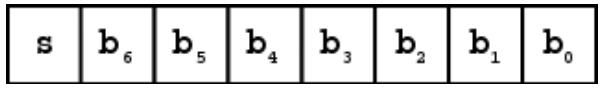


  - *Valor* = $b_0*2^0+b_1*2^1+b_2*2^2+...+b_n*2^n$
  - *Rang*: [0 .. 2^n-1]
  - *Exemples*: $10_d = 0A_h = 00001010_b$ $138_d = 9A_h = 10001010_b$

- **AVANTATGES**
  - Tots els bits s'utilitzen pel mòdul
- **INCONVENIENTS**
  - No es poden representar valors negatius

ENTERS AMB SIGNE (I): MÒDUL + SIGNE

- **REPRESENTACIÓ (amb 8 bits)**



  - *Valor* = $(-1)^{s}*(b_0*2^0+b_1*2^1+b_2*2^2+...+b_{n-1}*2^{n-1})$
  - *Rang*: $[-2^{n-1}+1 .. 2^{n-1}-1]$
  - *Exemples*: $+10_d = 0A_h = \mathbf{0}0001010_b$ $\quad -10_d = 8A_h = \mathbf{1}0001010_b$

- **AVANTATGES**
  - Es poden representar valors negatius
  - Representació simètrica
- **INCONVENIENTS**
  - Hi ha dues representacions pel 0

ENTERS AMB SIGNE (II): COMPLEMENT A 1

- **REPRESENTACIÓ (amb 8 bits)**

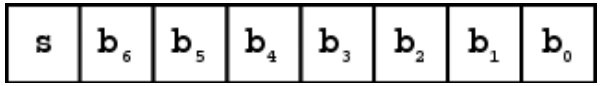
s	b_6	b_5	b_4	b_3	b_2	b_1	b_0

  - *Valor* = $(-1)^s*((1-b_0)*2^0+(1-b_1)*2^1+(1-b_2)*2^2+...+(1-b_{n-1})*2^{n-1})$
  - *Rang*: $[-2^{n-1}+1 .. 2^{n-1}-1]$
  - *Exemples*: $+10_d = 0A_h = \mathbf{0}0001010_b$ $\quad -10_d = F5_h = \mathbf{1}1110101_b$

- **AVANTATGES**
  - Es poden representar valors negatius
  - Representació simètrica
- **INCONVENIENTS**
  - Hi ha dues representacions pel 0

ENTERS AMB SIGNE (III): COMPLEMENT A 2

- **REPRESENTACIÓ (amb 8 bits)**



  - *Valor* = (Valor en complement a 1) + 1
  - *Rang*: $[-2^{n-1} .. 2^{n-1}-1]$
  - *Exemples*: $+10_d = 0A_h = \mathbf{0}0001010_b$ $\quad -10_d = F6_h = \mathbf{1}1110110_b$

- **AVANTATGES**
  - Es poden representar valors negatius
  - Els mateixos algorismes per sumar i restar sense signe i amb C_2
  - Representació única pel 0

- **INCONVENIENTS**
  - Representació assimètrica

SUMA EN COMPLEMENT A 2

- LA SUMA EN COMPLEMENT A 2 ÉS LA MATEIXA LA SUMA BINARIA (despreciant el *carry*)
- CÀLCUL DE L'OVERFLOW
  - Amb el bit de més pes dels operands i el resultat es pot calcular si hi ha hagut *Overflow*
  - Exemples:

    0 000 1010_b $0A_h$ 10_d

    1 111 1101_b FD_h -3_d

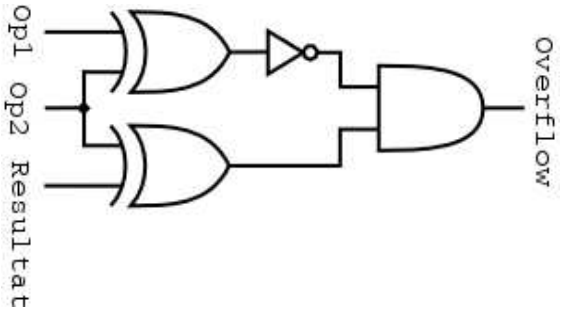
    1 **0** 000 0111_b 07_h 7_d *Carry!*

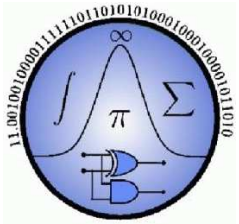
    0 110 1110_b $6E_h$ 130_d

    0 001 1110_b $1E_h$ 40_d

    1 000 1100_b $9B_h$ 140_d *Overflow!*

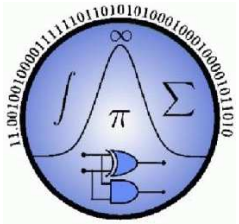
- Circuit per detectar l'*Overflow*: (*Bit 7 d'Op1, Op2 i Resultat*)





REALS (I): COMA FIXE

- **S'UTILITZA QUAN:**
  - No es disposa d'unitat de coma flotant ni per Hardware ni *emulada* per Software
  - Els càlculs en coma flotant són costosos
  - No es necessita gaire precisió o s'utilitza d'un número fixe i petit de decimals
- **TÈCNICA**
  - Considerar els valors “enters” com a valors multiplicats per un valor fixe
  - Es pot utilitzar 10^n on ***n*** és el numero de decimals



REALS (II): EXEMPLES EN COMA FIXE

- Els valors enters estan multiplicats per $10^2 = 100$ (2 decimals)
- La següent suma ***-1.5 + 3.14*** es pot expressar:

  -1.5 * 100 + 3.14 * 100 = **-150 + 314 = 164**

  164 ***DIV*** 100 = 1 (part entera)

  164 ***MOD*** 100 = 64 (part decimal)

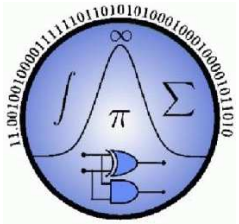
- **ULL!** La multiplicació genera més decimals i la divisió n'elimina. Utilitzant els valors anteriors però multiplicant (***-1.5 * 3.14***):

  -1.5 * 100 + 3.14 * 100 = -150 * 314 = -47100

  (Eliminar dos decimals generats) -471.00 = **-471**

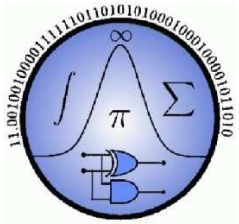
  471 ***DIV*** 100 = 4 (part entera)

  471 ***MOD*** 100 = 71 (part decimal)



REALS(III): COMA FLOTANT

- AVANTATGES
  - NECESSITAT DE PODER REPRESENTAR VALORS MOLT GRANS I VALORS MOLT PETITS AL MATEIX TEMPS
  - NO UTILITZAR UN NOMBRE DE BITS EXAGERAT
- INCONVENIENTS
  - PÈRDUA DE PRECISIÓ
  - L'ERROR AUGMENTA COM MÉS LLUNY DE L'INTÈRVAL [-1..1] ES TROBEN ELS VALORS



REALS(IV): COMA FLOTANT REPRESENTACIÓ

- Definició de signe, exponent i mantissa en els bits que representen els valors

$$\textbf{\textit{Valor}} = (-1)^{\text{Signe}} * \text{Mantissa} * \text{Base}^{\text{Exponent}}$$

- Es pot definir la *base* (2,4,8,10), els bits dedicats a l'exponent i a la mantissa i el format d'aquests
- L'IEEE 754 defineix un estàndard per reals de simple precisió (32 bits) anomenat ***float***s i de doble precisió (64 bits) anomenats ***double***s

REALS (V):FLOAT

- FORMAT D'UN **FLOAT** (32 BITS)

b_{31}	b_{30}	b_{29}	b_{28}	b_{27}	b_{26}	b_{25}	b_{24}	b_{23}	b_{22}	b_{21}	b_{20}	b_{19}	b_{18}	b_{17}	b_{16}	b_{15}	b_{14}	b_{13}	b_{12}	b_{11}	b_{10}	b_9	b_8	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
S	EXPONENT								s_1	s_2	s_3	MANTISSA																		s_{22}	b_{23}

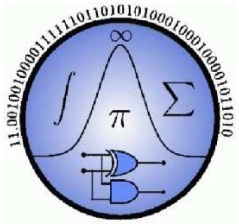
$$\text{Valor} = (-1)^S * (1 + \text{MANTISSA}) * 2^{\text{EXPONENT}}$$

$$\text{Valor} = (-1)^s * (1 + (s_1 * 2^{-1}) + (s_2 * 2^{-2}) + .. + (s_{23} * s^{-23})) * 2^E$$

- Normalitzat: La mantissa té sempre un 1 implícit (24 bits)
- El camp de l'exponent (8 bits) i conté el el signe de l'exponent
- Utilitza notació polaritzada per l'exponent (desplaçada en un *offset*) (aquest *offset* és 127 per ***float***s)

$$\textbf{\textit{Valor} = (-1)}^{\textbf{\textit{S}}} \textbf{\textit{ * (1 + MANTISSA) * 2}}^{\textbf{\textit{(EXPONENT - 127)}}}$$

- Utilitzant aquesta notació es poden ordenar els valors pel pes dels bits sense haver d'interpretar el valor que hi ha representat



REALS(VI): EXEMPLE FLOAT

- Com es representa el ***-0.75*** en un ***FLOAT***?

$$-0.75_d = -3/4_d = -3/2^2{}_d$$

$$-11_b/2^2{}_d = -0.11_b$$

Notació científica: $-0.11_b * 2^0$

Normalitzar (primer bit enter ha de ser 1): $-1.1_b * 2^{-1}$

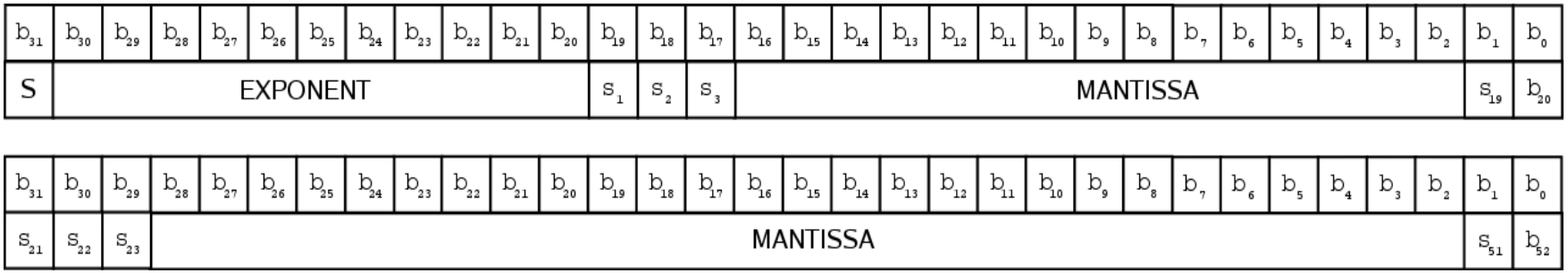
- Com que el valor d'un ***float*** és: $(-1)^s * (1 + \text{Mantissa}) * 2^{E-127}$

$(-1)^1 * (1 + .1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 000_b) * 2^{126}$

- Per tant -0.75 = **1** *0111 1110* 1000 0000 0000 0000 0000 000
- O bé: $-0.75 = BF40\ 0000_h$

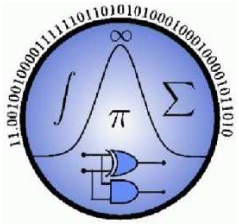
REALS (VII): DOUBLE

- FORMAT D'UN **DOUBLE** (64 BITS)



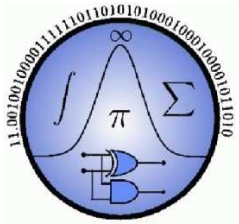
- L'exponent té 11 bits i s'utilitza un *offset* a 1023
- El camp *Mantissa* és de 52 bits (amb l'1 implícit 53)

$$Valor = (-1)^{S} * (1 + MANTISSA) * 2^{(\ EXPONENT - 1023\ )}$$



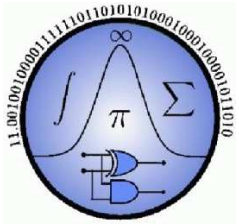
REALS (VII): PROBLEMES

- NO ES PODEN REPRESENTAR TOTS ELS VALORS
- EXISTEIX UN VALOR QUE EXPRESSA EL **MÍNIM** QUE HI POT HAVER ENTRE DOS REALS REPRESENTABLES (*Epsilon*)
- ES PRODUEIX ERROR EN ELS CÀLCULS
  - ELS VALORS SÓN APROXIMACIONS
- SI ES TREBALLA NORMALITZAT (ENTRE [-1..1]) ES PRODUEIX MENYS ERROR



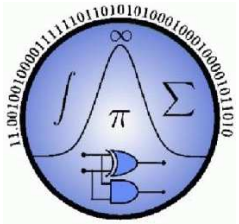
CONTINGUT D'UNA MEMÒRIA

```
0000:7F454C46 01010100 00000000 00000000 02000300 01000000 C8800408 34000000
0020:00000000 00000000 34002000 02000000 00000000 01000000 00000000 00800408
0040:00800408 F0020000 F0020000 05000000 00100000 01000000 00030000 00930408
0060:F0920408 00000000 20080000 06000000 00100000 00000000 00000000 00000000
0080:B8010000 00CD80C3 56538B54 24108B4C 240CBB00 000000B8 03000000 CD8089C6
00A0:89F05B5E C38D7600 56538B54 24108B4C 240CBB01 000000B8 04000000 CD8089C6
00C0:89F05B5E C38D7600 83EC1C55 57565383 C4F86A05 68009304 08E8AAFF FFFFE9C6
00E0:010000A0 2D930408 24070414 A2059304 0825FF00 0000BD20 9304088A 0428C0E8
0100:0424033C 010F858C 010000A0 01930408 32057593 04080FB6 D8C1E311 A0009304
0120:08320574 93040825 FF000000 05000100 00C1E008 31C3A002 93040832 05769304
0140:080FB6C8 A0049304 08320578 9304080F B6F0C1E6 11A00393 04083205 77930408
0160:25FF0000 00C1E009 31C68D04 0989CA81 E2F80100 0029D189 C729CF31 F783F708
0180:31F60FB6 05059304 08800428 F0C74424 1C190000 0001F689 D883E001 31C689D8
01A0:C1E81F01 C3C1FB01 89F82500 00000131 C301FFFF 4C241C75 DCBF8000 000089F6
01C0:89D885DB 7D038D43 07C1F803 88DA30C2 89D8C1F8 0430C289 D8C1F80C 30C20FB6
01E0:C2014424 1CC1FB08 C1E01131 C389F0C1 F80E31C6 89F2C1FA 088D04F5 00000000
0200:89F130C1 89F0C0E0 0630C188 4C241B0F B6E989EE C1E60931 D60FB68F 20930408
0220:89C883E0 070FBE80 D6820408 8D500289 C8C1F804 0FBE80DF 82040801 C031C289
0240:C8C1E008 01C889C1 C1F90331 D18D1409 21CA83E2 228D0412 01D001C0 88CA30C2
0260:8A44241B F6D00244 241C30C2 88972093 0408396C 241C0F9F C00FB6D0 8954241C
0280:4781FFFF 0700000F 8E33FFFF FF8A4C24 1B880D05 93040883 C4F86800 08000068
02A0:20930408 E8FFFDFF FF83C410 83C4F868 00080000 68209304 08E8CAFD FFFF83C4
02C0:1085C00F 851AFEFF FFE8B2FD FFFF5B5E 5F5D83C4 1CC33757 6F7E2747 5F8E0A63
02E0:72337366 7736763B 2A6B2B3E 2F6E2E31
```



QUÈ HI HA?

- CONSIDERANT UNA PLATAFORMA *LITTLE ENDIAN* DE *32 BITS*:
  1. QUINA ÉS LA PARAULA (SENSE SIGNE) QUE HI HA A L'ADREÇA 109h?
  2. QUIN ÉS LA MITJA PARAULA (AMB SIGNE) QUE HI HA A L'ADREÇA 005h?
  3. QUIN ÉS EL FLOAT QUE HI HA A L'ADREÇA 000h?



SOLUCIÓ (I)

1. **Paraula sense signe de l'adreça 109_h**

$$@109_h = 00\ 00\ A0\ 01_h$$

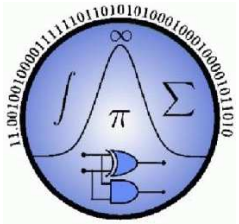
Lectura en *Little Endian*: $01\ A0\ 00\ 00_h$

Valor = $256_d^3 * 01_h + 256_d^2 * A0_h = \mathbf{27262976_d}$

o bé:

Valor = $0000\ 0001\ 1010\ 0000\ 0000\ 0000\ 0000\ 0000_b =$

$2^{21}+2^{23}+2^{24} = 2097152_d + 8388608_d + 16777216_d = \mathbf{27262976_d}$



SOLUCIÓ (II)

2. **Mitja paraula amb signe de l'adreça 005_h**

$@105_h = 01\ 01_h$

Lectura en *Little Endian*: $01\ 01_h$

Valor = $0000\ 0001\ 0000\ 0001_b$

En Complement a 2:

Valor = 1 + 256 = **+257**

SOLUCIÓ (III)

3. **Float** de l'adreça $\mathbf{000_h}$

$$@000_h = 7F454C46_h$$

Lectura en *Little Endian*: $464C457F_h$

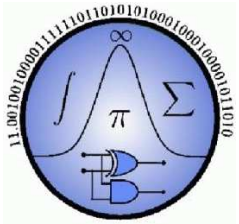
$$\mathbf{0}\boldsymbol{100\ 0110\ 0}100\ 1100\ 0100\ 0101\ 0111\ 1111_b$$

$$\boldsymbol{Valor = (-1)^S * (1 + MANTISSA) * 2^{(\ EXPONENT - 127\ )}}$$

$$\mathbf{Valor =} (1+ .100\ 1100\ 0100\ 0101\ 0111\ 1111) * 2^{140-127} =$$

$$= (1 + 2^{-1} + 2^{-4} + 2^{-5} + 2^{-9} + 2^{-13} + 2^{-15} + 2^{-17} + 2^{-18} + 2^{-19} +$$

$$+ 2^{-20} + 2^{-21} + 2^{-22} + 2^{-23}) * 2^{13} = \mathbf{13073.374023}$$



SOLUCIÓ (IV)

- QUÈ ERA REALMENT EL VALOR *7F454C46 010 ... 3E 2F6E2E31* ?
  - SI ES LLEGEIX EN DECIMAL
    - 49310835 ... 52543537 és un número primer de 1811 dígits.
  - SI ES LLEGEIX EN UNA MÀQUINA LINUX SOTA UN *ix86*
    - És el programa en codi màquina que desencripta els fitxers d'un DVD

```
$dd /dev/dvd | Programa(49310835...52543537) > Película.vob
```