Lezione LACD

Mercoledì 19 marzo 2025 ore 11-13

Binary System

Il sistema numerico binario
- base 2
- usa solo due simboli, di solito 0 1

Quello decimale → base 10 → 10 cifre

E' il sistema usato in informatica (nei computer) per la rappresentazione interna dell'informazione perché come visto nei circuiti digitali (tipo Flip-Flop) è molto conveniente la gestione dell'informazione con due soli valori (livelli di tensione/corrente elettrica). Vero - falso logica booleana.

Vediamo adesso con esempi come gestisco in binario
- un numero decimale
- il suo segno
- le operazioni somme, sottrazioni
- ottali / esadecimali

Vediamo esempi
- Conversione da base 2 a base 10:

Premessa in base 10 $237 = 2 \cdot 10^2 + 3 \cdot 10^1 + 7 \cdot 10^0$

Analogamente in base 2 $1\ 1\ 0\ 1\ 0\ 0\ 1\ 0 =$

$= 1 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 0 + 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 210$

128 64 16 2

- Viceversa: Conversione base 10 → base 2.

145		→ divido per due e
72	1 (LSB)	scrivo sotto valore e resto
36	0	↖ Bit meno significativo - Least Significant Bit
18	0	Il primo resto che ottengo mi dice
9	0	semplicemente se il numero è
4	1	pari 0 o dispari 1
2	0	
1	0	
0	1 (MSB)	bit più significativo

Ottengo quindi la sequenza inversa di bit in binario

Quindi il numero binario si riscrive dal MSB al LSB

MSB							LSB
1	0	0	1	0	0	0	1

$2^7 + 2^4 + 2^0 = 145$

- Se uso 8 bit

da	0	0	0	0	0	0	0	0	0
a	1	1	1	1	1	1	1	1	255

Ad esempio ogni casella di memoria è un flip-flop che ha come possibili valori 0 e 1

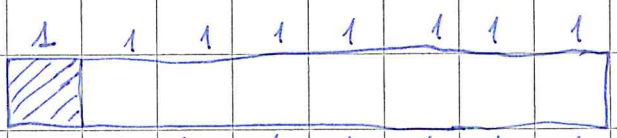
Posso rappresentare 256 numeri naturali (per ora solo positivi)

- Come faccio per i numeri negativi?

Prendo il MSB e lo riservo per descrivere il segno 0 = + e 1 = −

In questo caso su 8 bit: 7 per il numero e 1 per il segno

da 0 a 127



$-127 \leq x \leq +127$

$-(2^7)+1 \leq x \leq 2^7-1$

In totale ho 255 numeri x $-127 < x < +127$

ho due modi di scrivere lo zero +0 e -0

Se invece di avere 8 bit, ne avessi 16 $\pm 2^{15}$ (meno 1)

$\pm 65\,535$

A 32 bit $\pm 2^{31}$ $2 \cdot 10^9$

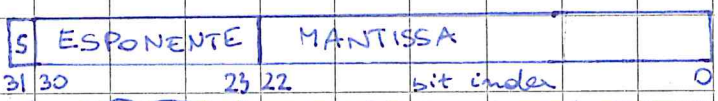
A 64 bit $\pm 2^{63}$ $9 \cdot 10^{18}$

I computer rappresentano trattano i numeri razionali suddividendoli in

MANTISSA × BASE$^{\text{ESPONENTE}}$ $a = M \times b^E$

Es.

$12345 = 12345 \times 10^{-4}$



- Nomi alternativi equivalenti
  - Mantissa o significando; in inglese { significand / mantissa / coefficient
  - Esponente o caratteristica; " " exponent

Oss: Passando da un intero a un razionale perdo informazioni

Posso "rigenerarla" usando double precision

64 bit invece di 32 bit $s = 1$ $E = 11$ $M = 52$

Rappresentazione in OTTALI

base 8 → uso le cifre da 0 a 7

raccolgo i bit in gruppi di 3

binario	1	0	0	1	0	0	0	1	=	decimale 145
	(0	1	0)	(0	1	0)	(0	0	1)	
ottale		2			2			1	=	ottale 221

I primi computer avevano sistemi di indirizzamento delle memorie di 6, 12, 24 bit tutti divisibili per 3 e usavano la rappresentazione ottale

Ad esempio CAMAC 24 bit

Rappresentazione esadecimale EXADECIMAL HEX

base 16 0 1 2 3 4 5 6 7 8 9 A B C D E F

→ raccolgo i bit in gruppi di 4

145 decimale	=	binario	1	0	0	1	0	0	0	1
		Hex		9				1		

(5)

Binary and Hexadecimal System

Decimal System
237 = 2 * 10^2 + 3 * 10^1 + 7 * 10^0
Conversion: Base 2 → Base 10
11010010 =
1*2^7 + 1*2^6 + 0*2^5 + 0*2^4 + 0*2^3 + 0*2^2 + 1*2^1 + 0*2^0
= 210
Conversion: Base 10 → Base 2
145 | → divide 145:2=72 with remainder of 1...
72 | 1 (least significative digit)
36 | 0
18 | 0
9 | 0
4 | 1
2 | 0
1 | 0
0 | 1 (most significative digit)

145 in base 10 corresponds to 10010001 in base 2
with N bits we can represent all integer numbers
between 0 and 2^N-1

DEC 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
HEX 0 1 2 3 4 5 6 7 8 9 A B C D E F
Hex Bin

0 0000
1 0001
2 0010
3 0011
4 0100
5 0101
6 0110
7 0111
8 1000
9 1001
A 1010
B 1011
C 1100
D 1101
E 1110
F 1111

Conversion: Base 16 → Base 2:
Hexadecimal number 3F5 corresponds
to 0011 1111 0101

Conversion Base 2 → Base 16:

10 0101 1101 0101 0010

0010 → 2
0101 → 5
1101 → D
10 → 2

0010 0101 1101 0101 0010 corresponds to
2 5 D 5 2

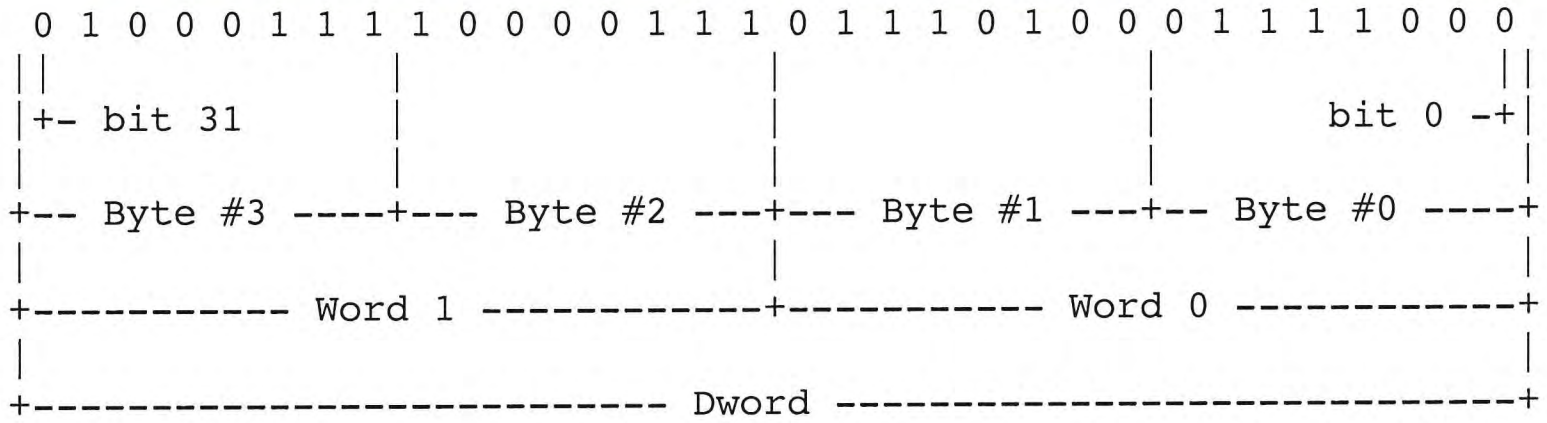
Operazioni fatte bit per bit

Bitwise Operators

Bitwise operators work on binary data

1 Byte = 8 bits; 1 Word = 2 Bytes = 16 bits; Dword = 2 Words = 32 bits

Double word



These are the 6 most important Bitwise operators:

1. **&** **AND operator**
2. **|** **OR operator**
3. **^** **XOR operator**
4. **~** **Ones Complement (Bitwise Not)**
5. **>>** **Right Shift operator**
6. **<<** **Left Shift operator**

Bitwise Operators

SHIFT : sposto tutti i bit rigidamente

1 AND

```
1   &   1   ==   1
1   &   0   ==   0
0   &   1   ==   0
0   &   0   ==   0
```

```
  00110010
& 00010000
 ----------
  00010000
```

2 OR

```
1   |   1   ==   1
1   |   0   ==   1
0   |   1   ==   1
0   |   0   ==   0
```

```
  00110010
| 00000100
 ----------
  00110110
```

3 XOR

```
1   ^   1   ==   0
1   ^   0   ==   1
0   ^   1   ==   1
0   ^   0   ==   0
```

```
  00110010
^ 00011000
 ----------
  00101010
```

4 Bitwise NOT

```
~ 1   ==   0
~ 0   ==   1
```

```
~ 00000011
 ----------
  11111100
```

5 LEFT SHIFT

altro esempio ↓

```
 00001100 << 2 (decimal)      11111111
----------
 00110000                     11 11 1100
```

6 RIGHT SHIFT

```
 00001100 >> 2 (decimal)      11 11 11 11
----------
 00000011                     00 11 11 11
```

RIGHT AND LEFT SHIFT
(be careful...)

```
 00001111 >> 2 (decimal)
----------
 00000011 << 2 (decimal)
----------
 00001100                     00 11 11 00
```

Inoltre **MASK**

se sono interessato solo ad alcuni bit

faccio l'AND tra la parola (word) binaria e la maschera (mask)
nella posizione dei o del bit che mi interessa:

A 1 0 1 1 0 1 0 1

MASK 0 0 0 0 0 1 0 0 mi interessa il 3° bit

A & MASK = 0 0 0 0 0 1 0 0

Con queste operazioni "bit wise" riesco a gestire
facilmente le operazioni fra numeri positivi.
Per quelli negativi serve usare qualche
"trucco": complemento (negazione) a 1
(aggiunta del bit perso).

ES1 3 0 0 0 0 0 0 1 1

-1 1 0 0 0 0 0 0 1

anziché -1 prendo il complemento a 1 di 1

finiamo di vedere le operazioni prox volta