1. Calcolare le seguenti conversioni di base:
    a. $(263)_8 \rightarrow (\ )_3$
    b. $(242)_5 \rightarrow (\ )_7$

Soluzione:

a) Per trasformare 263_8 in base 3, bisogna per prima cosa passare alla base 10 per ottenere il valore del numero, per poi attraverso divisioni successive giungere alla base 3.

$263_8=179_{10}$

$263_8=8^2*2+ 8*6+3=179_{10}$

179/3=59 con resto 2

59/3 =19 con resto 2

19/3 =6 con resto 1

6/3 =2 con resto 0

2/3 =0 con resto 2

Ora riordinando i resti della divisione (dall' ultimo al primo) si ottiene il risultato.

$263_8= 20122_3$

b) Per trasformare 242_5 in base 7, bisogna per prima cosa passare alla base 10; per poi attraverso divisioni successive giungere alla base 7.

$242_5=72_{10}$

$242_5=5^2*2+5*4+2=50+20+2= 72_{10}$

72/7=10 con resto 2

10/7=1 con resto 3

1/7 =0 con resto 1

Ora riordinando i resti della divisione (dall' ultimo al primo) si ottiene il risultato.

$242_5 = 132_7$

2. Si valuti per quale base x sono vere le seguenti relazioni:

a) $(45)_x \rightarrow (100101)_2$

Deve essere una base $\geq$ 6 in quanto presenta cifre fino a 4 e 5.
Opero la conversione da binario a decimale e ottengo il numero 37 in base 10.
Quindi $(45)_x = 37$

E per la regola che usiamo per convertire a base 10 si ha: $4x + 5 = 37 \quad x = 8$ che è la base cercata.

b) $(431)_x \rightarrow (71)_{13}$

Deve essere una base $\geq$ 5. Il numero 71 in base 13 diventa 92 in base 10 sempre con il medesimo procedimento.

Per la definizione di numero i base fissa, dobbiamo porre: $4X^2+3x+1 = 92$, da cui $4X^2+3x-91 = 0$.
$x = \frac{-3 \pm \sqrt{9 + 4*4*91}}{2} = \frac{-3 \pm \sqrt{1463}}{2} = \frac{-3 \pm 38.249}{2}$. Nessuna radice è intera, quindi non esiste una base che verifichi la corrispondenza. Metodo intuitivo:
per la regola di conversione in base 10 si ha: $(4x + 3)x + 1 = 92$ che non dà risultati accettabili. Si può concludere che questa uguaglianza non è valida per alcuna base, anche perché già con base 5 il numero in questione varrebbe 116 $>$ 92, aumentando ulteriormente la base il numero superebbe sempre di più 92.

3. Valutare le seguenti operazioni assumendo una rappresentazione in complemento a 2 su 4 bit. Per ciascuna di esse spiegare se il risultato è esprimibile su 4 bit.

a) -3+5

Il modulo di 3 in complemento a due è 11 perciò per trovare -3 e rappresentarlo su 4 bit faccio il complemento a due e ottengo 1101. Per +5 so che il modulo è 101 quindi su 4 bit il suo valore sarà 0101.

```
 1101 +
 0101 =
10010
```

Nota: il carry finale cade fuori dalla rappresentazione e si scarica $0010_2 = 2_{10}$ risultato corretto in 4 bit e questo si poteva già osservare prima in quanto i due operandi hanno segno discorde quindi non si può avere overflow nella somma CA2.

-9+3

non si può fare perché -9 non può essere rappresentato in 4 bit. Infatti il modulo di 9 è 1001 e -9 sarebbe 10111 su 5 bit.

b) +2+6= 0010+0110

Qui c'è possibilità di overflow poiché i due operandi sono concordi.

```
0010 +
0110 =
1000    overflow: cambia il segno!
```

Infatti +8 in CA2 si scrive 01000 e quindi si può rappresentare solo in 5 bit. Inoltre in 4 bit posso rappresentare solo numeri da -2^3 a $2^3 - 1$ quindi da -8 a +7 quindi +8 non avrei comunque potuto rappresentarlo.

-4+4

Il modulo di 4 è 100 quindi +4 in CA2 su 4 bit sarà 0100 e -4 sarà 1100.

```
 1100 +
 0100 =
10000
```

Nota: il carry finale cade fuori dalla rappresentazione e si scarica $0000_2 = 0_{10}$ risultato corretto su 4 bit, inoltre i due operandi erano discordi quindi non poteva esserci overflow.

c)-7+8

Il modulo di 7 in CA2 è 111 quindi -7= 1001 su 4 bit, +8 non si può rappresentare su 4 bit, quindi l'operazione non si può fare. Un matematico, ma non una macchina, potrebbe osservare che -8 si può rappresentare su 4 bit (è il numero più negativo, 1000), e che possiamo vedere -7+8 come -(+7-8). Inoltre avendo gli addendi segni opposti non ci saranno problemi di overflow.

```
 0111 +
 1000 =
 1111
```

Per ottenere il risultato corretto, occorre complementare a 2 il risultato ottenuto:
1111 → 0001= 1 risultato richiesto

4+4

Il modulo di 4 come detto precedentemente è 100 quindi +4 sarà 0100. Ci potrà essere overflow in quanto i due operandi sono concordi e ciò si avrà se il risultato sarà discorde.

```
 0100 +
 0100 =
 1000  = -8 overflow perché su 4 bit non si può rappresentare +8 ma solo su 5 (01000).
```

4. Si eseguano le seguenti operazioni su numeri in complemento a 2, espressi su 5 bit, indicando una eventuale situazione di overflow:

a) 10111 + 10011

```
 10111 +  → -16+4+2+1= -9
 10011 =  → -16+2+1= -13
01010     → overflow perché cambia il bit del segno. Infatti -22 può essere rappresentato solo su 6 bit. Ciò si poteva capirlo dal fatto che su 5 bit si possono rappresentare numeri solo da -16 a +15, inoltre i due operandi erano concordi quindi c'era possibilità di ovreflow.
```

b) 00011 + 10001

```
 00011 +  → +3
 10001 =  → -15
 10100  = -12 → risultato corretto, i due operandi sono discordi.
```

c) 01101 – 10011

Per effettuare una sottrazione in c.a 2 occorre sommare al minuendo il c.a 2 del sottraendo:

```
 01101 +  → +13
 01101 =  → +13
 11010 → overflow perché cambia il segno. Infatti il risultato sarebbe +26 ma non è rappresentabile su 5 bit.
```

5. Si considerino i numeri in modulo e segno espressi su 8 bit. Si valuti l'intervallo di rappresentazione (min e max) nel caso di numeri interi e di numeri con virgola binaria tra il secondo e terzo bit da destra (es 101010.00)

Su 8 bit i numeri espressi in modulo e segno possono andare da $-(2^7-1)$ a $+(2^7+-1)$, cioè da -127 (11111111) a +127 (01111111).

Questi saranno anche i valori massimo e minimo dell'intervallo dei numeri interi. Nel caso, invece, dei numeri con la virgola, essi saranno del tipo: xxxxxx.yy dove yy può essere 00, 11,10,01.

Inoltre dopo la virgola il numero massimo che possiamo avere è 11 cioè 0.75, il minimo è 00 cioè 0.00.

La parte intera è perciò costituita da 6 bit che potranno variare in un intervallo da $-(2^5-1)$ a $+(2^5-1)$, cioè da -31 a +31.

A questo punto si può concludere che il minimo numero con la virgola è -31.75, cioè 111111.11 mentre il massimo è +31.75, cioè 011111.11.

6. Calcolare il risultato delle seguenti operazioni su numeri assoluti, considerando operandi e risultato rappresentati in binario su 5 bit. Per ciascuna di esse verificare preventivamente se l'operazione è possibile, spiegando in caso contrario la ragione

a) 01011+00111

Operazioni su numeri assoluti, cioè senza segni, quindi è come se fosse un'operazione in binario puro.

01011+ → 11

00111= → 7

L'operazione può avvenire senza problemi in quanto nel posto dell'MSB c'è, in entrambi gli operandi, la cifra 0. Quindi è come se fossero due numeri da 4 bit che sommati tra di loro possono dare al massimo un numero da 5 bit, non di più.

Perciò

```
01011 +
00111 =
10010     = 18
```

b) 11011+10001

11011+ → 27

10001= → 17

La somma di questi due operandi darebbe 44 che non può esser rappresentato su soli 5 bit, ma ne 6 (44= 101100). Inoltre si può notare che le due MSB dei due operandi sono uguali a 1 e solamente la somma di queste due richiede già un sesto bit, quindi ci sarà sicuramente overflow.

```
 11011 +
 10001 =
1 01100
```

→ nella somma di numeri assoluti la presenza di carry dopo il bit più significativo indica overflow

c) 01011+10000

01011+ → 11

10000= → 16

Il risultato è 27 quindi non ci sarà overflow, inoltre tutte le cifre 1 vengono sommate a cifre uguali a 0 quindi non si avranno mai riporti e non si potrà "sforare" nel sesto bit.

```
01011 +
10000 =
11011  = 27
```

d) 01000+01101

01000+ → 8

01101= → 13

Anche in questo caso non ci sarà overflow poiché gli unici 1 che si sommano tra loro e danno un riporto uguale a 1 sono nel quarto bit e nel quinto bit ci sono solo zeri.

```
01000 +
01101 =
10101  = 21
```

Esercizio 7

Si descriva come si codificano in un elaboratore i numeri relativi e i numeri reali. Si indichi come può essere codificato in binario la frazione ¼ e il numero 0.75.

Soluzione:

a) I numeri relativi (numeri con segno) si possono codificare o in modulo e segno o in complemento a 2.
   a. Modulo e segno: il bit significativo rappresenta il segno (0 = +, 1 = -), i restanti n-1 bit rappresentano il modulo. I numeri rappresentati vanno da $-(2^{n-1}-1)$ a $-(2^{n-1}-1)$, con doppia rappresentazione dello 0 (+0 e -0).
   b. Rappresentazione in complemento a 2:
      i. I numeri positivi hanno 0 nel bit del segno (MSB), i restanti n-1 bit rappresentano il modulo (nota: non si fa nessuna complementazione!!!)
      ii. I numeri negativi sono rappresentati con il complemento a 2 del corrispondente positivo (nota: in questo caso c'è la complementazione!). Fa eccezione il numero più negativo (1000…00), che non ha il corrispondente positivo. I numeri rappresentati vanno da $-(2^{n-1})$ a $-(2^{n-1}-1)$ e lo 0 è rappresentato una sola volta.

b) I numeri reali (numeri con parte frazionaria) si possono codificare o in virgola fissa o in virgola mobile. *La virgola non viene rappresentata, ma supposta* in una data posizione.
   a. Virgola fissa: dati n bit per la rappresentazione, si riservano m (m < n) bit per la parte frazionaria, n-m bit per la parte intera (in cui è compreso un eventuale segno).

Occorre scegliere se privilegiare la *precisione* della rappresentazione, data dal numero di bit nella parte frazionaria, o il *range* (intervallo) dei valori rappresentati, dato dal numero di bit nella parte intera.

b. Virgola mobile: il numero, tradotto in binario, viene prima trasformato nella forma normalizzata $\pm 1.xxxxxxx \cdot 2^e$, successivamente le diverse parti (il segno, xxxxxxx che viene chiamata mantissa, l'esponente e) vengono opportunamente codificate in binario e impaccate su n bit. Ad esempio, nella codifica più diffusa, quella dell'IEEE P754 a 32 bit, viene usato 1 bit per il segno, 8 bit per l'esponente e (dopo un'opportuna codifica), 23 bit per la mantissa. Si noti che il primo 1 del numero, sempre presente, viene sottinteso (*hidden bit*, bit nascosto), così come non viene rappresentato il 2 del moltiplicatore. I numeri rappresentati, in modulo, vanno da $1.0 \cdot 2^{-126}$ ($\cong 10^{-38}$) a $1.11111\ldots111 \cdot 2^{+127}$ ($\cong 2 \cdot 2^{+127} = 2^{+128} \cong 10^{+38}$). Per lo 0, che non è riconducibile alla forma normalizzata 1.xxxx, si adopera un apposito codice, così come per altre informazioni (overflow, rappresentazione non significativa, detta anche NAN, not a number, ed altro). In questa rappresentazione si conserva l'errore relativo (pari a 2^{-23} nella codifica P754 a 32 bit). Ad esempio, tra due valori successivi rappresentati, nell'intorno dell'unità c'è una differenza di $2^{-23} = 2^{-3} \cdot 2^{-20} \cong 0.125 \cdot 10^{-6} \cong 10^{-7}$, nell'intorno del massimo rappresentabile c'è una differenza di $2^{-23} \cdot 2^{+127} = 2^{+104} = 2^4 \cdot 2^{100} = 2^4 \cdot (2^{10})^{10} \cong 16 \cdot (10^3)^{10} = 1.6 \cdot 10^{31}$.

c) Codifica di ¼ e di 0.75:

a. ¼ = $1/2^2 \rightarrow 0.01_2$
b. 0.75: si procede con la regola della moltiplicazione per 2, prendendo (ed eliminando) la parte intera:

0.75	0.5	0
1.5	**1**.0	

Quindi $0.75 \rightarrow 0.11_2$

Esercizio 8

Siano dati e seguenti numeri in floating point, con esponente in complemento a due e la mantissa con la virgola posta a sinistra:

0 11110 11001000 0 11100 10101000

Si traducano tali numeri in base 10.

a) 0 11110 11001000

0 → segno positivo

11110 (in CA2) = - 00010 (in binario puro) = -2 → esponente

$0.11001000 \cdot 2^{-2} = 0.\ 0011001 = 2^{-3} + 2^{-4} + 2^{-7} = 0.125 + 0.0625 + 0.0078125 = 0.1953125$ → mantissa

Il valore rappresentato è + 0,1953125

b) 0 11100 10101000

0 → segno positivo

11100 (in CA2) = - 00100 (in binario puro) = -4 → esponente

$10101000 = 2^{-1} + 2^{-3} + 2^{-5} = 0.5 + 0.125 + 0.03125 = 0.65625$ → mantissa

Numero = segno * mantissa * (base 2)$^{\text{esponente}}$ = (+1) * 0.65625 * 2^{-4} = + 0,041015625

Esercizio 9

Si valuti la dimensione di un file contenente 500 fotografie, ciascuna con occupazione non compressa pari a 14kB. Di quanto diminuisce l'occupazione del file se si applica una codifica JPEG con fattore di compressione 75:1?

Svolgimento:

Sapendo che vi è una compressione di fattore 75:1, ossia $75 = \frac{\text{dim}ensione\_dati\_estesi}{\text{dim}ensione\_dati\_compressi}$

Inizialmente si può calcolare la memoria occupata dal numero totale dei file non compressi, moltiplicando la memoria occupata da un file per il numero di fotografie, ossia:

14 kB * N = 14kB * 500 = 7000 kB = memoria estesa

In seguito si può determinare la dimensione dei file compressi, ossia:

dimensione file compressi $= \frac{dimensione\_dati\_estesi}{75} = \frac{7000kB}{75} \cong 93.3kB$

Perciò la memoria risparmiata si ottiene dalla differenza tra memoria occupata dai file estesi e memoria occupata dai file compressi, ossia:

risparmio = memoria estesa – memoria compressa = (7000 – 93.3) kB $\cong$ 6906.7 kB

Infine si può esprimere questa quantità in Megabyte dividendo per 2^{10}, perciò

$$\frac{6906.7}{2^{10}} \cong 6.74 \text{ MB}$$

Risposta

Grazie alla compressione si riesce a risparmiare uno spazio circa pari a 6906.7 kB, ossia 6.74 MB.

Esercizio 10

Si discuta la differenza tra le codifiche GIF, JPEG e TIFF

Soluzione:

GIF	Usa un sistema di compressione lossless a dizionario del tipo LZW, 1byte/pixel (originariamente i colori sono su 24 bit, ma ne vengono scelti solo 256 per la rappresentazione). Adatto per il WEB, inadatto per le foto (tra l'altro, il file GIF non contiene informazioni dpi per la stampa). Superato dal formato PNG.
JPEG	Usa sempre una sofisticata tecnica lossy per la compressione, ma il grado di compressione si può selezionare: si può avere qualità più alta (e dimensioni maggiori del file) o qualità più bassa (e dimensioni minori del file). Adatto per la fotografia, poco adatto per testo o grafica.
TIFF	È per certi versi il formato più versatile (è stato concepito piuttosto come un contenitore, in cui è l'utente a decidere il formato dei contenuti). Per i colori, supporta le seguenti modalità: RGB – 24 o 48 bit, Grayscale – 8 o 16 bit Indexed color – da 1 a 8 bit, Line Art – 1 bit Solitamente o non è utilizzata nessuna compressione o si usa la compressione LZW lossless (ma meno efficace per immagini con colore a 24 bit). Potrebbero essere usati altri metodi di compressione (JPG, ZIP, ecc). Questa eccessiva libertà di codifica costituisce anche il limite di questo formato (per leggere il file è richiesto un programma adeguato). Molto usato negli scanner, è anche presente in molte macchine fotografiche.

Esercizio 11

Si valuti l'occupazione di memoria richiesta per memorizzare un filmato digitale di 2 minuti, sapendo che ogni immagine occupa 23 kB e che sono trasmesse 25 immagini al secondo. Se lo stesso filmato fosse codificato in MPEG, con fattore di compressione medio pari a 120:1, quale risulterebbe l'occupazione?

Soluzione:

Un filmato è dato dallo scorrimento veloce di immagini, l'occupazione della memoria sarà quindi dato da:

(Dimensione_immagine)*(Immagini_al_secondo)*(Numero_secondi) =

23 * 25 * 120 = 69000 KB

Se non piacesse un risultato con un così alto numero di cifre, lo si può convertire in MB semplicemente dividendolo per 2^{10} , ottenendo così 67,38 MB .

Se fosse codificato in MPEG con compressione 120:1 , il che vorrebbe dire

(Dimensione_dati) : (Dimensione_dati_compressi)

avremmo un'occupazione pari a:

(Dati_non_compressi) : 120 = 69000 : 120 = 575 KB

Contributi.
I seguenti esercizi sono stati realizzati da:
es. 1: Carlo Stola matr. 172868
es. 2 – 6: Francesca Tosto matr. 171693
es. 8 e 9: Alessandro Tassone matr. 175597
es. 11: Tartaglia Giuseppe matr. n° 171482