



Esame A di Architetture degli Elaboratori

Soluzione

A.A. 2018-19 — II appello — 12 febbraio 2019

N.B.: il punteggio associato ad ogni domanda è solo una misura della difficoltà, e peso, di ogni domanda. Per calcolare il voto complessivo bisogna normalizzare a 32.

1. Convertire il valore 155.12_6 alla base 12.

R: (3 pt) Conviene dapprima convertire alla base 10 e poi alla base 12:

$$155.12_6 = 1 \cdot 6^2 + 5 \cdot 6^1 + 5 \cdot 6^0 + 1 \cdot 6^{-1} + 2 \cdot 6^{-2} = 71 + 1/6 + 2/36 = 71 + 2/9 = 71.\bar{2}$$

71 12 -----	.22222222 12 -----	-->	2/9 12 -----
5 11=B 0 5	.66666666 2 0 8		2/3 2 0 8

da cui il risultato: $5B.28_{12}$.

2. È data la seguente codifica in complemento a 2 a 10 bit: $n_{C2} = 1111110000$. Esprimere lo stesso numero in codifica in complemento a 1 a 10 bit.

R: (3 pt) Poiché n_{C2} codifica un numero negativo, cambiandolo di segno otteniamo $-n_{C2} = 0000010000$. Per valori positivi le due codifiche combaciano, sicché cambiando il segno secondo le regole del complemento a uno otteniamo subito la corrispondente codifica: $n_{C1} = 1111101111$. Alternativamente era sufficiente decrementare di uno n_{C2} inteso come valore assoluto, applicando direttamente la definizione di codifica in complemento a due.

3. [INF] Fornire in codifica *floating point* IEEE 754 a 32 bit il risultato (non intero) della divisione $255/128$.

R: (3 pt) La stessa divisione espressa in notazione binaria risulta essere immediatamente uguale a $11111111_2/10000000_2 = 1.1111111_2$. La codifica richiesta avrà dunque bit di segno non asserito, esponente uguale a $127 + 0 = 127 = 01111111_2$ e infine mantissa uguale a 1111111_2 . Sistemando sui 32 bit previsti dallo standard IEEE 754 e convertendo alla base esadecimale:

$$\begin{array}{cccccccccccccccc} 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & \dots \\ 3 & & & & & & & & & & & & & & & 0 \dots \end{array}$$

da cui la codifica richiesta: $0x3FFF0000$.

4. Un *chip* di memoria di forma quadrata e capacità uguale a 10^{10} byte occupa un'area di 0.25 cm^2 nella scheda madre. Quanto misura il lato di una memoria di forma e densità identica se la sua capacità è uguale a 10^{12} byte?

R: (3 pt) Se la capacità della seconda è 100 volte quella della prima allora il nuovo quadrato sarà di $100 \cdot 0.25 = 25 \text{ cm}^2$. Di conseguenza, il nuovo lato misura 5 cm.

5. Adoperando le regole dell'algebra booleana, dimostrare che $C + \overline{C}D = D + C\overline{D}$. Successivamente si minimizzi la stessa espressione attraverso l'uso di una tabella di verità.

R: (3 pt) $C + \overline{C}D = C(\overline{D} + D) + \overline{C}D = C\overline{D} + CD + \overline{C}D = C\overline{D} + (C + \overline{C})D = C\overline{D} + D = D + C\overline{D}$.

Dalla tabella di verità si verifica immediatamente l'uguaglianza $C + \overline{C}D = C + D$.

6. [INF] Si dica qual è il numero minimo di transistor necessari per realizzare la rete logica che calcola l'espressione dell'esercizio precedente.

R: (3 pt) $C + D = \overline{\overline{C} + \overline{D}}$. Ricordando che la porta NOR si compone di due transistor e la NOT di un transistor, si deduce che il numero minimo di transistor necessari per realizzare l'espressione in questione è 3.

7. É possibile realizzare una porta ternaria AND adoperando un *demultiplexer* a tre ingressi? In caso affermativo se ne dia la rete logica.

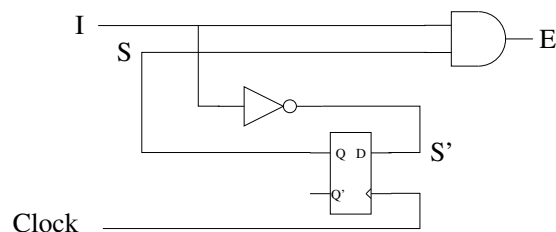
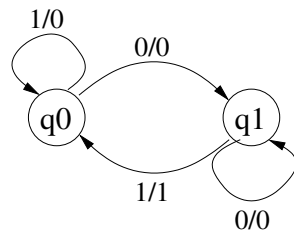
R: (3 pt) É sufficiente collegare l'ingresso del *demultiplexer* al valore alto di tensione e l'uscita associata all'ingresso 111 all'uscita della rete logica richiesta. Era accettato anche l'utilizzo di un decoder, nel qual caso l'ingresso poteva essere omissso.

8. [INF] Progettare la rete sequenziale che realizza la macchina di Mealy che riconosce ogni sottostringa 01 contenuta in una sequenza definita sull'alfabeto $\mathcal{A} = \{0, 1\}$. La rete restituisce il simbolo 1 solo quando riconosce la sottostringa, altrimenti restituendo 0 a fronte di ogni nuovo ingresso appartenente alla sequenza.

R: (3 pt)

S I S' E

0	0	1	0
0	1	0	0
1	0	1	0
1	1	0	1



9. Si sceglie di definire la seguente funzione di efficienza per un codice: $\text{efficienza} = \frac{\text{distanza di Hamming}}{\text{costo}}$. Calcolare l'efficienza di un codice di *parità dispari* a 16 bit.

R: (3 pt) Ricordando che un codice di parità dispari a 16 bit ha distanza di Hamming uguale a 2 e costo uguale a $1/15$ si ha subito efficienza $= \frac{2}{1/15} = 30$.

10. Un bus parallelo a 16 linee trasmette bit sincronizzati, simultaneamente su ciascuna linea a una frequenza uguale a 532 MHz. A quale valore dovrebbe essere reimpostato il periodo di clock affinché il bus abbia globalmente una banda di 133 Mbyte/s ?

R: (3 pt) Considerato che adoperando l'impostazione iniziale la banda del bus è uguale a $16 \cdot 532$ Mbit/s, corrispondente a $2 \cdot 532$ Mbyte/s, per ottenere la banda richiesta occorre dividere la frequenza di clock per un fattore 8. Infatti: $2 \cdot 532/8 = 133$ Mbyte/s. Dunque, il periodo di clock risulta uguale a $1/\text{frequenza} = 1/(532 \cdot 10^6/8) \approx 15$ ns.

11. La conoscenza del linguaggio Assembly è utile allo sviluppatore di applicazioni per l'*Internet of Things*? Se sì, si dia un possibile contesto applicativo tra quelli visti durante il corso.

R: (3 pt) Sì. Per esempio, al fine di ottimizzare le prestazioni del microcontrollore della scheda Arduino, molto adoperata per realizzare applicazioni nell'*Internet of Things* le quali spesso richiedono tempi di esecuzione molto ridotti pur nel contesto di *hardware* estremamente semplice ed economico.

12. La realizzazione nel processore ARM di un'istruzione di rientro da procedura, che possiamo chiamare **return**, richiederebbe la presenza di microcodice progettato allo scopo oppure può appoggiarsi direttamente su microcodice che realizza istruzioni già esistenti?

R: (3 pt) Un'istruzione di rientro da procedura corrisponde a spostare il contenuto del *link register* (lr) nel *program counter* (pc), entrambi realizzati all'interno del processore ARM su registri accessibili con le stesse modalità di tutti gli altri. Quindi, l'istruzione equivale a una *move*: `mov pc, lr`.

13. Assumendo uno spazio indirizzabile di memoria uguale a 4 GB, qual è la dimensione di una *cache* a corrispondenza diretta (cioè ad accesso diretto) nel caso in cui ogni linea di *cache* occupa 128 byte e il campo *TAG* che contiene l'indirizzo di memoria da cui proviene la linea è lungo 14 bit?

R: Lo spazio indirizzabile in questione è associato a indirizzi di 32 bit. Di questi, i 14 bit più significativi indirizzano la linea e quindi fanno parte del campo *TAG*. In parallelo, i 7 meno significativi indirizzano il word e l'eventuale byte nella linea. Conseguentemente restano 11 bit per indirizzare ogni riga della cache, che sarà lunga 1 (validità) + 14 (*TAG*) + $8 \cdot 2^7 = 1039$ bit. La dimensione della cache in definitiva è di $1039 \cdot 2^{11} = 2127872$ bit ≈ 260 kB.

14. [INF] Scrivere un programma in assembly per ARM il quale riordina in senso contrario gli elementi di un array presente in memoria. Inoltre, restituisce nel registro *r0* la somma degli elementi contenuti nell'array. La dimensione dell'array è anch'essa reperibile in memoria.

R: (9 pt)

```
.data
array_dim:
    .word 5
array_el:
    .word 18, 12, 1, 14, 3
    .text
main:
    ldr r0, =array_dim
    ldr r0, [r0]                ; array dim in r0
    ldr r1, =array_el           ; base in r1
    ldr r2, =array_el           ; base in r2
    add r2, r2, r0, lsl #2       ; base + dim in r2
    sub r2, r2, #4              ; last element position in r2
    mov r0, #0                  ; reset r0 (accumulator)
    cmp r2, r1                  ; set initial status
    blt end                     ; exit if array dim <=0
swap: ldr r3, [r1]              ; load lower element in r3
    ldr r4, [r2]                ; load higher element in r4
    add r0, r0, r3               ; accumulate lower element
    addgt r0, r0, r4             ; if r2-r1>0 then accumulate also higher element
    str r3, [r2], #-4           ; store r3 in higher position, update pos
    str r4, [r1], #4            ; store r4 in lower position, update pos
    cmp r2, r1                  ; if r2-r1..
    bge swap                    ; ..>=0 then repeat
end: swi 0x11                   ; exit
    .end
```