Università degli Studi di Udine

DMIF — Dipartimento di Scienze Matematiche, Informatiche e Fisiche

Esame **B** 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 2035.4_6 alla base 12.

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

   $2035.4_6 = 2 \cdot 6^3 + 0 \cdot 6^2 + 3 \cdot 6^1 + 5 \cdot 6^0 + 4 \cdot 6^{-1} = 455 + 2/3 = 455.\overline{8}$

   ```
   455 | 12        .666666666 | 12        2/3 | 12
    -------        ---------------  -->   --------
    37 | 11=B               0 | 8          0 | 8
     3 | 1
     0 | 3
   ```

   da cui il risultato: $31B.8_{12}$.

2. É data la seguente codifica in complemento a 2 a 10 bit: $n_{C2} = \texttt{1110000111}$. 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 $-n_{C2} = \texttt{0001111001}$. Per valori positivi le due codifiche combaciano, sicché cambiando il segno secondo le regole del complemento a uno otteniamo subito la corrispondente codifica: $n_{C1} = \texttt{1110000110}$. 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 510/256.

   R: (3 pt) La stessa divisione espressa in notazione binaria risulta essere immediatamente uguale a $\texttt{111111110}_2/\texttt{100000000}_2 = \texttt{1.1111111}_2$. La codifica richiesta avrà dunque bit di segno non asserito, esponente uguale a $127 + 0 = 127 = \texttt{01111111}_2$ e infine mantissa uguale a $\texttt{1111111}_2$. Sistemando sui 32 bit previsti dallo standard IEEE 754 e convertendo alla base esadecimale:

   ```
   0|0 1 1 1 1 1 1 1|1 1 1 1 1 1 1 0 ...
      3   |   F   |   F   |   F   | 0 ...
   ```

   da cui la codifica richiesta: `0x3FFF0000`.

4. Un *chip* di memoria di forma quadrata e capacità uguale a 10^{12} byte occupa un'area di 1.6 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^{13} byte?

   R: (3 pt) Se la capacità della seconda è 10 volte quella della prima allora il nuovo quadrato sarà di $10 \cdot 1.6 = 16$ cm^2. Di conseguenza, il nuovo lato misura 4 cm.

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

   R: (3 pt) $B + A\overline{B} = (\overline{A} + A)B + A\overline{B} = \overline{A}B + AB + A\overline{B} = \overline{A}B + A(B + \overline{B}) = \overline{A}B + A = A + \overline{A}B$.

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

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) $A + B = \overline{\overline{A + B}}$. 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 binaria NOR adoperando un *demultiplexer* a due ingressi? In caso affermativo se ne dia la rete logica.

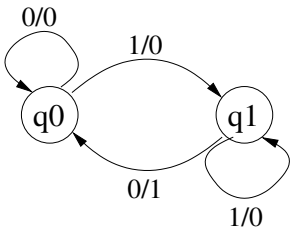
   R: (3 pt) É sufficiente collegare l'ingresso del *demultiplexer* al valore alto di tensione e l'uscita associata all'ingresso `00` all'uscita della rete logica richiesta. Era accettato anche l'utilizzo di un decoder, nel qual caso l'ingresso poteva essere omesso.

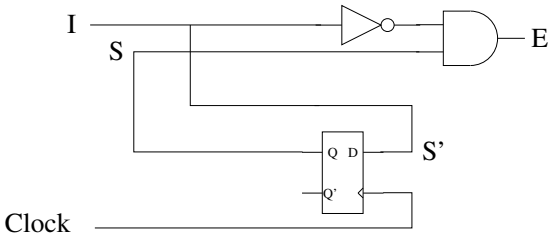
8. [INF] Progettare la rete sequenziale che realizza la macchina di Mealy che riconosce ogni sottostringa `10` contenuta in una sequenza definita sull'alfabeto $\mathcal{A} = \{0, 1\}$. La rete restituisce il simbolo `1` solo quando riconosce la sottostringa, altrimenti restituendo `0` a fronte di ogni nuovo ingresso appartenente alla sequenza.

   R: (3 pt)

   S I S' E

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





9. Si sceglie di definire la seguente funzione di efficienza per un codice: efficienza $= \frac{\text{distanza di Hamming}}{\text{costo}}$. Calcolare l'efficienza di un codice di *parità pari* a 14 bit.

   R: (3 pt) Ricordando che un codice di parità pari a 14 bit ha distanza di Hamming uguale a 2 e costo uguale a 1/13 si ha subito efficienza $= \frac{2}{1/13} = 26$.

10. Un bus parallelo a 4 linee trasmette bit sincronizzati, simultaneamente su ciascuna linea a una frequenza uguale a 133 MHz. A quale valore dovrebbe essere reimpostato il periodo di clock affinché il bus abbia globalmente una banda di 266 Mbyte/s ?

    R: (3 pt) Considerato che adoperando l'impostazione iniziale la banda del bus è uguale a $4 \cdot 133$ Mbit/s, corrispondente a $0.5 \cdot 133$ Mbyte/s, per ottenere la banda richiesta occorre moltiplicare la frequenza di clock per un fattore 4. Infatti: $4 \cdot 0.5 \cdot 133 = 266$ Mbyte/s. Dunque, il periodo di clock risulta uguale a 1/frequenza $= 1/(4 \cdot 133 \cdot 10^6) \approx 1.88$ ns.

11. a conoscenza del linguaggio Assembly è utile allo sviluppatore di applicazioni per l'*Internet of Things*? Se sì, si dia un possibile contesto applicativ otra 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 *branch* incondizionato richiede la presenza di microcodice progettato allo scopo, oppure può appoggiarsi direttamente su microcodice che realizza altre istruzioni?

    R: (3 pt) Un'istruzione di *branch* incondizionato corrisponde a spostare un determinato valore nel *program counter* (`pc`), che all'interno del processore ARM è realizzato su un registro accessibile con le stesse modalità di tutti gli altri. Quindi, l'istruzione equivale a una `mov pc,arg2` in cui il secondo argomento è specificato secondo le regole dell'istruzione *move*.

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 64 byte e il campo *TAG* che contiene l'indirizzo di memoria da cui proviene la linea è lungo 20 bit?

R: Lo spazio indirizzabile in questione è associato a indirizzi di 32 bit. Di questi, i 20 bit più significativi indirizzano la linea e quindi fanno parte del campo TAG. In parallelo, i 6 meno significativi indirizzano il word e l'eventuale byte nella linea. Conseguentemente restano 6 bit per indirizzare le righe della cache, ognuna delle quali per quanto detto è lunga 1 (validità) + 20 (TAG) + $8 \cdot 2^6 = 533$ bit. La dimensione della cache in definitiva è di $533 \cdot 2^6 = 34112$ bit ≈ 4.16 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
```