Università degli Studi di Udine

DMIF — Dipartimento di Scienze Matematiche, Informatiche e Fisiche

Esame **A** di **Architetture degli Elaboratori**

Soluzione

A.A. 2017-18 — III appello — 19 giugno 2018

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 (circa).

1. Convertire il valore $13.\overline{2}_4$ alla base 8.

   R: (3 pt) Trattandosi di basi che sono una potenza di due, conviene dapprima convertire alla base 2 e poi raggruppare le cifre risultanti per riscrivere il valore nella base 8:

   $$13.\overline{2}_4 = \underbrace{01_2}_{1_4}\underbrace{11_2}_{3_4}.\underbrace{\overline{10}_2}_{\overline{2}_4} = \underbrace{01}_{1_4}\underbrace{11}_{3_4}.\underbrace{\overline{101010}}_{\overline{222}_4} = 0\underbrace{111}_{7_8}.\underbrace{\overline{101}}_{\overline{5}_8}.\underbrace{\overline{010}}_{\overline{2}_8} = 7.\overline{52}_8.$$

2. Sono date le seguenti codifiche in complemento a 2 a 8 bit: $n_1 = \texttt{10110100}$, $n_2 = \texttt{00000100}$. Se esiste, si calcoli il rapporto n_1/n_2 e se possibile lo si esprima nella stessa codifica.

   R: (3 pt) Per sicurezza conviene calcolare $|n_1|/|n_2|$ e poi ricordare che n_1 codifica un valore negativo. Dunque, cambiando di segno n_1 si ha $-n_1 = \texttt{01001100}$. Poiché n_2 vale 4, il calcolo del rapporto è immediato: basta traslare n_1 di due bit, ottenendo $|n_1|/|n_2| = |n_1|/n_2 = \texttt{00010011}$. Complementando nuovamente si ha subito $n_1/n_2 = -|n_1|/n_2 = \texttt{11101101}$.

   Si noti che lo stesso risultato sarebbe uscito anche senza passare per il valore assoluto di n_1.

3. [INF] Fornire il risultato dell'esercizio precedente in codifica *floating point* IEEE 754 a 32 bit.

   R: (3 pt) Il risultato trovato sopra può essere subito messo nella forma $-\texttt{1.0011E4}$. La codifica richiesta avrà dunque bit di segno asserito, esponente uguale a $127+4 = 131 = \texttt{10000011}_2$ e infine mantissa uguale a `0011`. Sistemando sui 32 bit previsti dallo standard IEEE 754 e convertendo alla base esadecimale:

   ```
   1|1 0 0 0 0 0 1 1|0 0 1 1 0 0 0 0 ...
     C   |   1   |   9   |   8   | 0 ...
   ```

   da cui la codifica richiesta: `0xC1980000`.

4. La *legge di Moore* afferma che il numero di transistor nell'unità di memoria quadruplica ogni tre anni. Se un chip di memoria di forma rettangolare oggi misura mediamente 1×0.5 cm, quanto misuravano mediamente i lati di un chip analogo 18 mesi fa, a parità di capacità di memoria?

   R: (3 pt) Se il numero di transistor nell'unità di memoria quadruplica ogni tre anni allora il raddoppio della densità avviene ogni anno e mezzo. Quindi, a parità di capacità di memoria un anno e mezzo prima i lati erano mediamente più lunghi di un fattore $\sqrt{2}$: $1 \cdot \sqrt{2} \times 0.5 \cdot \sqrt{2} \approx 1.4 \times 0.7$ cm.

5. È dato il seguente listino prezzi in microeuro ($\mu\epsilon$): porta NAND 1 $\mu\epsilon$; porta NOR 5 $\mu\epsilon$; porta NOT 10 $\mu\epsilon$; porta AND 20 $\mu\epsilon$; porta OR 20 $\mu\epsilon$. Adoperando le regole di equivalenza booleana, dare il circuito che realizza l'espressione booleana $\overline{A} + \overline{B}$ al costo minimo, quantificando inoltre il costo stesso.

   R: (3 pt) Adoperando le regole di De Morgan si ha $\overline{\overline{A} + \overline{B}} = AB = \overline{\overline{AB}}$, che adopera una porta NAND e una porta NOT costando dunque $1 + 10 = 11$ $\mu\epsilon$. Questa soluzione ha ricevuto punteggio pieno.

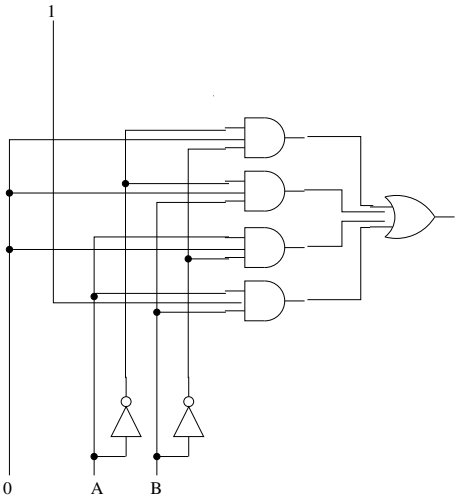
   Il costo minimo in assoluto si trovava ricordando l'uguaglianza $E = E \cdot E \cdot \ldots \cdot E$. Di qui l'ulteriore espressione $\overline{\overline{AB}} = \overline{AB \cdot AB}$ che, richiedendo solo due porte NAND, costa $1 + 1 = 2$ $\mu\epsilon$.

6. [INF] Si provi ad abbassare ulteriormente il costo del circuito all'esercizio precedente, ricordando che esso può accedere anche alle tensioni nulla e di alimentazione le quali rappresentano, rispettivamente, gli stati logici 0 e 1.

   R: (3 pt) Continuando ad adoperare le regole di De Morgan si ha $\overline{\overline{A} + \overline{B}} = \overline{\overline{AB}} = \overline{\overline{AB}} + 0 = \overline{\overline{AB} \cdot 1}$, che adoperando due porte NAND costa $1 + 1 = 2$ $\mu\epsilon$. Chi all'esercizio precedente aveva già trovato la soluzione in assoluto migliore non poteva comunque scendere sotto il costo di $1 + 1 = 2$ $\mu\epsilon$.

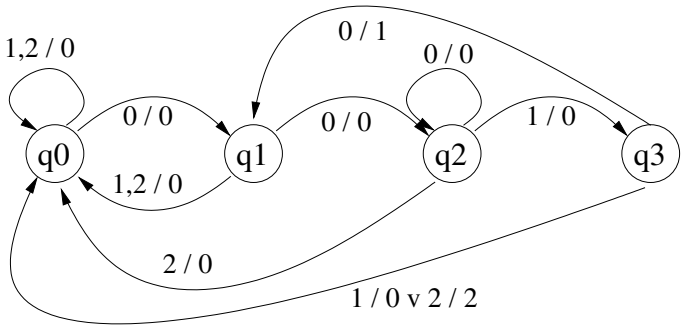
7. Realizzare l'espressione all'esercizio 5 adoperando un *multiplexer*, il cui circuito combinatorio dev'essere chiaramente esplicitato all'interno della rete booleana proposta.

   R: (3 pt)



8. [INF] Disegnare il diagramma della macchina di Mealy che, leggendo simbolo dopo simbolo una sequenza indefinitamente lunga di caratteri appartenenti all'alfabeto $\mathcal{A} = \{0, 1, 2\}$, riconosce *ogni* istanza delle due sottosequenze `0012` e `0010` all'interno della sequenza letta. Per esempio, la sequenza `22001001022` contiene *due* istanze della sottosequenza `0010`. A ogni ciclo di lettura la macchina in questione genera le seguenti uscite: `0` se non ha riconosciuto alcuna sottosequenza; `1` nell'istante in cui riconosce la sottosequenza `0010`; `2` nell'istante in cui riconosce la sottosequenza `0012`.

   R: (3 pt)



9. Quanti bit contenenti dati informativi possiede un *codice di Hamming a N bit*?

   R: (3 pt) I codici di Hamming a N bit contengono $\log_2(N + 1)$ bit di controllo. Di conseguenza i bit informativi sono $N - \log_2(N + 1)$. Per esempio, il codice di Hamming a 7 bit contiene $7 - 3 = 4$ bit informativi.

10. In un elementare sistema che deve servire un'unica periferica si deve scegliere tra un protocollo I/O, di tipo programmato oppure controllato. Di essi si sa che il primo serve la periferica in 3 ms, mentre il secondo necessita di 4 ms per espletare il medesimo servizio. In più si sa che la periferica può chiedere di essere servita esattamente ogni 100 ms, tuttavia questo avviene appunto solo se la periferica ha fatto richiesta; altrimenti essa non necessita di essere servita, e forse farà una richiesta trascorsi esattamente

altri 100 ms. Sulla base di questi dati si calcoli qual è il minimo numero di richieste che rende mediamente conveniente adoperare il protocollo I/O di tipo programmato.

R: (3 pt) Se la periferica necessita di essere servita ogni 100 ms allora il protocollo I/O di tipo programmato fa guadagnare al sistema 1 ms di tempo di calcolo ogni 100 ms. Il punto di parità tra i due protocolli avviene quando la periferica dev'essere mediamente servita tre volte su quattro, cioé il 75% delle volte. In tal caso anche il protocollo I/O di tipo controllato richiede 12 ms di tempo al sistema per le interruzioni. In definitiva il numero medio minimo di interruzioni che rende conveniente adoperare il protocollo I/O di tipo programmato è uguale a 3 ogni 400 ms.

11. Con riferimento all'esercizio precedente, si dica quante sono le richieste che saturano il sistema nel caso in cui si scelga di usare a) il protocollo I/O di tipo programmato e b) il protocollo I/O di tipo controllato.

    R: (3 pt) Il protocollo I/O di tipo programmato è saturato quando la periferica richiede il servizio ogni 100 ms. Il protocollo I/O di tipo controllato non può essere mai saturato da una periferica che richiede il servizio ogni 100 ms, in quanto la serve in 4 ms.

12. Si dia un esempio in cui un'operazione di *logical shift right* (`lsr`) eseguita da un processore ARM non corrisponde all'operazione di *arithmetic shift right* (`asr`) eseguita sullo stesso dato.

    R: (3 pt) Le due operazioni hanno esito diverso per esempio quando il dato rappresenta la codifica in complemento a due di un valore intero negativo.

13. Una memoria virtuale di 1 GB è organizzata in pagine di 64 kB, che possono trovare posto in memoria di massa oppure in posizioni casuali di una memoria principale di 4 MB. Non è prevista la presenza di pagine virtuali inesistenti. Come dev'essere dimensionata la corrispondente *page table*?

    R: La memoria principale necessita di 30 bit per essere indirizzata. Di essi, 16 individuano l'*offset* all'interno di ogni pagina di $2^{10+6} = 64$ kB. La *page table* dunque dev'essere formata da $2^{30-16} = 2^{14}$ righe indirizzate dalla parte più significativa dell'indirizzo. Ciascuna di esse dovrà contenere 22 bit per indirizzare ciascuna pagina casualmente in $2^{22} = 4$ Mbyte più il bit di presenza/assenza della pagina in memoria fisica, per un totale di 2^{14} righe di 23 bit ciascuna.

14. [INF] Con riferimento all'esercizio 8, scrivere un programma in assembly per ARM il quale legge dal file di testo `file.txt` una sequenza di caratteri appartenenti all'alfabeto $\mathcal{A} = \{$`0,1,2`$\}$ e quindi realizza la macchina sequenziale colà descritta. Il file di testo per comodità riporta nella prima riga il numero di caratteri presenti nel file a partire dalla seconda riga. L'output è scritto a ogni ciclo di lettura nel registro `r5`. Il programma termina allorquando il file è stato letto interamente.

    R: (9 pt)

```
        .data
stringa:
        .asciiz "input.txt"
        .text
main:
        ldr r0, =stringa        ; file name address in r0
        mov r1, #0              ; read mode
        swi 0x66                ; open file in read mode
        mov r2, r0              ; save file handler
        swi 0x6c                ; read number of integers
        mov r3, r0              ; save number of integers in r3
        mov r6, #1              ; r6 loaded with one
machine_q0:
        subs r3, r3, r6         ; decrement number of integers..
        beq end                 ; ..and exit if zero
        mov r5, #0              ; output zero
        mov r0, r2              ; load file handler
        swi 0x6c                ; read integer from file
        cmp r0, #0              ; if input==0..
        beq machine_q1          ; ..jump to state q1
        b machine_q0            ; else stay in current state
machine_q1:
```

```
        subs r3, r3, r6         ; decrement number of integers..
        beq end                 ; ..and exit if zero
        mov r5, #0              ; output zero
        mov r0, r2              ; load file handler
        swi 0x6c                ; read integer from file
        cmp r0, #0              ; if input==0..
        beq machine_q2          ; ..jump to state q2
        b machine_q0            ; else jump to state q0
machine_q2:
        subs r3, r3, r6         ; decrement number of integers..
        beq end                 ; ..and exit if zero
        mov r5, #0              ; output zero
        mov r0, r2              ; load file handler
        swi 0x6c                ; read integer from file
        cmp r0, #1              ; if..
        beq machine_q3          ; ..input==1 jump to state q3
        bgt machine_q0          ; ..input==2 jump to state q0
        blt machine_q2          ; ..input==0 stay in current state
machine_q3:
        subs r3, r3, r6         ; decrement number of integers..
        beq end                 ; ..and exit if zero
        mov r0, r2              ; load file handler
        swi 0x6c                ; read integer from file
        cmp r0, #0              ; if..
        bne non_zero            ; ..input!=0 jump to non_zero
        mov r5, #1              ; else output one
        b machine_q1            ; and jump to state q1
non_zero:
        cmp r0, #1              ; if..
        bne non_one             ; ..input!=1 jump to non_one
        mov r5, #0              ; else output zero
        b machine_q0            ; and jump to state q0
non_one:
        mov r5, #2              ; output two
        b machine_q0            ; jump to state q0
end:
        ;; end
        swi 0x11                ; exit
        .end
```