Università degli Studi di Udine

DMIF — Dipartimento di Scienze Matematiche, Informatiche e Fisiche

Esame di **Architetture degli Elaboratori**

Soluzione

A.A. 2017-18 — IV appello — 3 luglio 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 $AA.\overline{F}_{16}$ alla base 4.

   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 4:

   $$AA.\overline{F}_{16} = \underbrace{1010_2}_{A_{16}}\underbrace{1010_2}_{A_{16}}.\underbrace{\overline{1111}_2}_{\overline{F}_{16}} = \underbrace{2_4}_{10_2}\underbrace{2_4}_{10_2}\underbrace{2_4}_{10_2}\underbrace{2_4}_{10_2}.\underbrace{\overline{33}_4}_{\overline{1111}_2} = 2222.\overline{3}_4.$$

2. Sono date le seguenti codifiche già in complemento a due a 8 bit: $n_1 =$ `00001010`, $n_2 =$ `11110101`. Se esiste, si calcoli il prodotto $n_1 \cdot n_2$ e se possibile lo si esprima in codifica *complemento a uno*.

   R: (3 pt) Conviene calcolare $n_1 \cdot |n_2|$ e poi ricordare che n_2 codifica un valore negativo. Dunque, cambiando di segno n_2 si ha $-n_2 =$ `00001011`. Il calcolo del prodotto $n_1 \cdot |n_2|$ porge

   ```
         00001010 x
         00001011
   --------------
         00001010
        00001010
       00000000
      00001010
   --------------
      00001101110
   ```

   che, restituendo il segno negativo in codifica complemento a uno, diventa `10010001`.

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 $-$`1.10111E6`. La codifica richiesta avrà dunque bit di segno asserito, esponente uguale a $127+6 = 133 =$ `10000101`$_2$ e infine mantissa uguale a `10111`. Sistemando sui 32 bit previsti dallo standard IEEE 754 e convertendo alla base esadecimale:

   ```
   1|1 0 0 0 0 1 0 1|1 0 1 1 1 0 0 0 ...
      C   |   2   |   D   |   C   | 0 ...
   ```

   da cui la codifica richiesta: `0xC2DC0000`.

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 quadrata di 256 MB oggi misura mediamente 1×1 cm, quale sarebbe la capacità di un chip delle stesse dimensioni tra 9 mesi se non esistessero vincoli di standardizzazione necessari per la compatibilità col resto dell'architettura?

   R: (3 pt) Se il numero di transistor nell'unità di memoria quadruplica ogni 36 mesi allora il raddoppio della densità avviene ogni 18 mesi, e quindi dopo 9 mesi la memoria cresce di un fattore $\sqrt{2}$ a parità di dimensioni. In definitiva dunque il chip in questione avrebbe capacità uguale a $\sqrt{2} \cdot 256 = 362$ MB.

5. È data la tabella a destra, in cui C, D, E, F e A, B sono rispettivamente i 4 ingressi e le 2 uscite di una rete booleana. Realizzare ciascuna uscita con un numero minimo di porte, esclusivamente binarie, scelte tra AND, NAND, OR, NOR e XOR. Quante porte occorrono per realizzare le rete in questione?

C	D	E	F	A	B
1	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	1
0	0	0	1	1	0

R: (3 pt) Si ha: $A = \overline{C}D\overline{E}\,\overline{F} + \overline{C}\,\overline{D}\,\overline{E}F = \overline{C}\,\overline{E}\,(D\overline{F} + \overline{D}F) = \overline{C + E}(D\overline{F} + \overline{D}F)$.

Analogamente: $B = \overline{C}\,\overline{D}E\overline{F} + \overline{C}D\overline{E}\,\overline{F} = \overline{C}\,\overline{F}\,(\overline{D}E + D\overline{E}) = \overline{C + F}\,(\overline{D}E + D\overline{E})$.

Sia A che B sono realizzate unendo le uscite da una porta NOR e una XOR attraverso una porta AND. Quindi in totale occorrono 6 porte, 3 per ciascuna uscita.

Questo se si assumeva che le uscite per gli ingressi non specificati fossero nulle. Se le stesse si fossero considerate indeterminate, allora in totale erano sufficienti 2 porte XOR: $A = D\overline{F} + \overline{D}F$ e $B = \overline{D}E + D\overline{E}$.

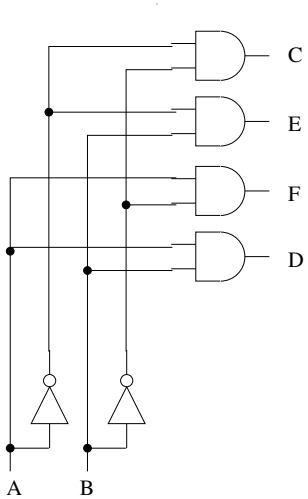
6. [INF] Si realizzi la rete all'esercizio sopra rinunciando alla porta AND. Quante porte occorrono stavolta?

R: (3 pt) Ricordando l'uguaglianza $E = \overline{\overline{E}\overline{E}}$, è sufficiente sostituire la porta AND con due NAND per ottenere lo stesso risultato. Quindi, in totale occorrono 8 porte per realizzare le due uscite A e B.

Anche qui, se si assumeva che le uscite per gli ingressi fossero indeterminate allora in totale erano sufficienti le stesse 2 porte XOR: $A = D\overline{F} + \overline{D}F$ e $B = \overline{D}E + D\overline{E}$.

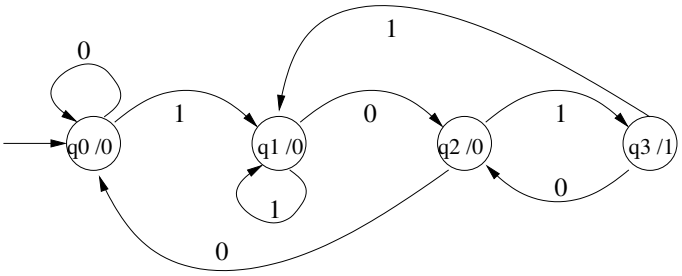
7. Invertire la tabella (cioè trasformare gli ingressi in uscite e viceversa) dell'esercizio 5. Realizzare la nuova tabella così ottenuta adoperando un *decodificatore*, il cui circuito combinatorio dev'essere chiaramente esplicitato all'interno della rete booleana proposta.

R: (3 pt)



8. [INF] Disegnare il diagramma della macchina di Mealy oppure di Moore che, leggendo simbolo dopo simbolo una sequenza indefinitamente lunga di caratteri appartenenti all'alfabeto binario $\mathcal{B} = \{0, 1\}$, riconosce *ogni* istanza delle due sottosequenze `1010` e `1011` all'interno della sequenza letta. Per esempio, la sequenza `10101011010` contiene *una* istanza della sottosequenza `1011` e *tre* istanze della sottosequenza `1010`. A ogni carattere letto la macchina in questione genera le seguenti uscite: `0` se non ha riconosciuto alcuna sottosequenza; `1` nell'istante in cui riconosce la sottosequenza `1010` o la sottosequenza `1011`.

R: (3 pt) Poichè la sequenza è indefinitamente lunga siamo certi che, dopo ogni sottosequenza `101`, la macchina *sicuramente* genererà un'uscita uguale a `1`. Dunque, per esempio scegliamo la macchina di Moore schematizzata qui sotto:



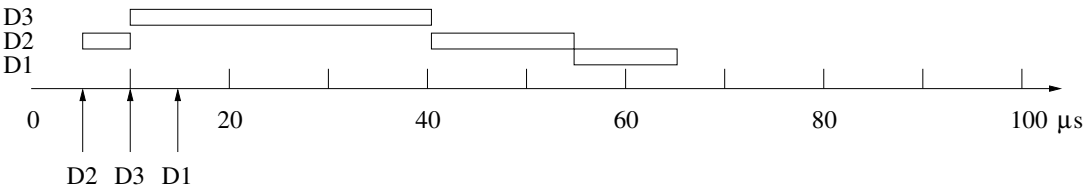
Alternativamente è corretto anche realizzare la macchina di Mealy che restituisce `1` in corrispondenza di ogni simbolo letto dopo avere riconosciuto la sottosequenza `101`.

9. Si vogliono codificare le 21 lettere dell'alfabeto italico A, B, ..., Z adoperando un codice binario a lunghezza variabile, il quale ottimizzi l'efficienza della codifica delle vocali A, E, I, O, U minimizzando la lunghezza dei rispettivi codici. Si dia un codice che rispetta questo requisito.

   R: (3 pt) Il codice binario dev'essere lungo almeno 5 bit. Essendo le vocali 5, le 16 consonanti possono essere codificate dai 4 bit meno significativi premessi per esempio dallo `0`: `00000`, `00001`, ..., `01111`. A questo punto ottimizziamo i codici delle vocali, che devono iniziare con `1`: poiché occorrono almeno 3 bit per codificare 5 simboli, i cinque codici possono essere i seguenti: `1000`, `1001`, `1010`, `1011`, `1100`. Infine, poiché non esistono due codifiche che iniziano con `11` l'ultimo codice può essere accorciato in `11`.

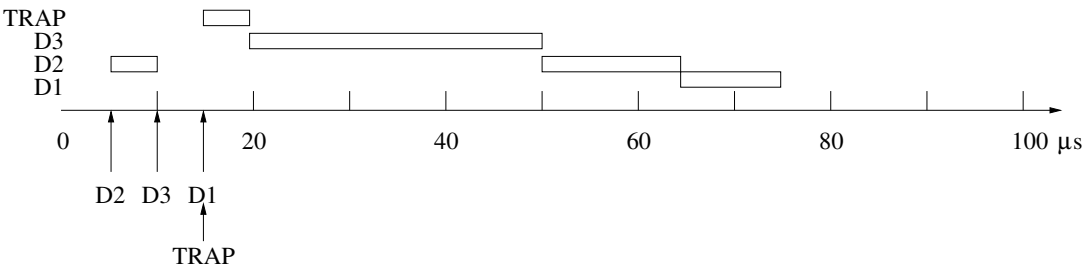
10. In un protocollo I/O di tipo controllato, tre dispositivi D1, D2 e D3 con priorità di servizio crescente competono per essere serviti dalla CPU. Il primo è servito in 10 μs, il secondo in 20 μs e il terzo in 30 μs. Posto a $t = 0$ μs l'attimo in cui il sistema entra in funzione, accade che: a $t = 5$ μs D2 inoltra una richiesta di servizio; a $t = 10$ μs D3 inoltra una richiesta di servizio; a $t = 15$ μs D1 inoltra una richiesta di servizio. Si tracci un diagramma temporale che illustra come e quando le richieste vengono servite dalla CPU.

    R: (3 pt)



11. Si modifichi il diagramma tracciato in precedenza, tenendo presente che 1) a $t = 15$ μs si verifica una *trap* a massima priorità di durata 5 μs, e che 2) la CPU solo nel caso di servizi verso dispositivi esterni impiega 5 μs per sospendere un servizio a priorità più bassa a fronte della necessità di servire un dispositivo a priorità maggiore.

    R: (3 pt)



12. Una memoria virtuale di 4 GB è organizzata in pagine di 32 kB, che possono trovare posto all'interno di una memoria principale di 4 MB. Non è prevista la presenza di pagine di dimensione nulla. Qual è la dimensione della *page table*?

    R: La memoria fisica, di 4 GB = 2^{32} Byte, necessita di 32 bit per essere indirizzata. Di essi, 15 individuano l'*offset* all'interno di ogni pagina di 2^{15} Byte = 32 kB. La *page table* dunque deve comprendere $2^{32-15} = 2^{17}$ righe indirizzate dalla parte più significativa dell'indirizzo. Poichè la memoria principale è

di 4 MB = 2^{22} Byte, ogni riga sarà lunga 22 bit più il bit di presenza/assenza della pagina in memoria principale. La dimensione della *page table* dunque è di $2^{17} \times 23$ bit.

13. Si spieghi la differenza esistente tra gli indirizzamenti *immediato*, *diretto* e *indiretto*, possibili argomenti del set di istruzioni a bordo del `Core i7`.

    R: (3 pt) L'indirizzo immediato è interpretato dalla CPU come un valore numerico; l'indirizzo diretto è interpretato come l'indirizzo di una locazione di memoria contenente un valore numerico; l'indirizzo indiretto infine è interpretato come l'indirizzo di una locazione di memoria, tipicamente il nome di un registro, in cui leggere l'indirizzo di una locazione di memoria contenente un valore numerico.

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 $\mathcal{B} = \{\texttt{0,1}\}$ e quindi realizza la macchina che riconosce le sottosequenze colà descritte. L'output è scritto a ogni ciclo di lettura nel registro `r4`. In più, il programma mantenga nel registro `r7` la somma delle sottosequenze `1010` trovate e nel registro `r8` la somma delle sottosequenze `1011` trovate. Il file di testo per comodità riporta nella prima riga il numero di caratteri presenti nel file a partire dalla seconda riga. 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
        mov r7, #0              ; reset r7
        mov r8, #0              ; reset r8
machine_q0:
        subs r3, r3, r6         ; decrement number of integers..
        beq end                 ; ..and exit if zero
        mov r4, #0              ; output zero
        mov r0, r2              ; load file handler
        swi 0x6c                ; read integer from file
        cmp r0, #1              ; if input==1..
        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 r4, #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 r4, #0              ; output zero
        mov r0, r2              ; load file handler
        swi 0x6c                ; read integer from file
        cmp r0, #1              ; if input==1
        beq machine_q3          ; ..jump to state q3
        b machine_q0            ; else jump to state q0
```

```
machine_q3:
        subs r3, r3, r6         ; decrement number of integers..
        beq end                 ; ..and exit if zero
        mov r4, #1              ; output one
        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
        add r7, r7, #1          ; else count up 0010 in r7
        b machine_q2            ; and jump to state q2
non_zero:
        add r8, r8, #1          ; count up 0011 in r8
        b machine_q1            ; and jump to state q1
end:
        ;; end
        swi 0x11                ; exit
        .end
```