Università degli Studi di Udine

DMIF — Dipartimento di Scienze Matematiche, Informatiche e Fisiche

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

Soluzione

A.A. 2017-18 — V appello — 25 settembre 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 numero $333.\overline{3}_{10}$ alla base 3.

   R: (3 pt) $333.\overline{3}_{10} = 110100.1_3$. Infatti,

```
333 | 3      .333.. | 3
 -------     ----------
111 | 0           0 | 1
 37 | 0
 12 | 1
  4 | 0
  1 | 1
  0 | 1
```

2. Sono date le seguenti codifiche in complemento a 2 a 7 bit: $n_1 =$ `1001000`, $n_2 =$ `1111010`. Si calcoli la somma $n_1 + n_2$, si dica quanto vale e se possibile la si esprima nella stessa codifica.

   R: (3 pt) Si può direttamente riprodurre i calcoli realizzati da un'ipotetica ALU che opera in aritmetica in complemento a 2 a 7 bit:

```
 1001000 +
 1111010 =
---------
 1000010
```

   Complementando a 2 il risultato si ottiene `0111110`, equivalente al valore 62. La somma cercata vale dunque -62. Per la validità della codifica associata si veda l'esercizio 4.

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 $-1.1111\mathtt{E5}_2$. La codifica richiesta avrà dunque bit di segno asserito, esponente uguale a $127+5 = 132 = 10000100_2$ e infine mantissa uguale a `1111`. Sistemando sui 32 bit previsti dallo standard IEEE 754 e convertendo alla base esadecimale per semplicità di lettura:

```
1|1 0 0 0 0 1 0 0|1 1 1 1 0 0 0 0 ...
  C   |   2   |   7   |   8   | 0 ...
```

   da cui la codifica richiesta: `0xC27800000`.

4. Con riferimento all'esercizio 2, si spieghi come la ALU decide la validità del risultato di un'operazione di somma in complemento a 2.

   R: (3 pt) Se i termini da sommare differiscono nel bit più significativo allora hanno segno diverso e non possono dare luogo a *overflow*. Altrimenti, se il bit più significativo del risultato è diverso da quello dei due termini da sommare allora c'è certamente *overflow*, viceversa non può esserci *overflow*.

5. Adoperando le regole di equivalenza booleana, calcolare quanto vale E nell'espressione seguente:

   $E = ABC \cdot \overline{A\overline{C}}$

   R: (3 pt) Sfruttando le regole di De Morgan si ha $E = ABC \cdot (\overline{A} + \overline{\overline{C}}) = A\overline{A}BC + ABCC = 0 + ABC$.

6. [INF] Verificare il risultato ottenuto sopra con una mappa di Karnaugh.

   R: (3 pt)

BC / A	00	01	11	10
0	0	0	0	0
1	0	0	1	0

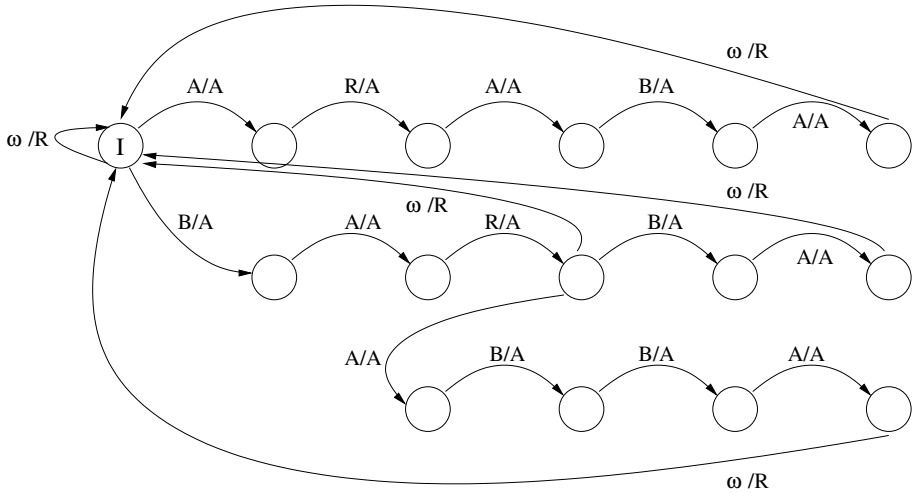
   da cui $E = ABC$.

7. Si può realizzare l'espressione all'esercizio 5 avendo a disposizione solamente delle porte NOR a 3 ingressi? Motivare la risposta.

   R: (3 pt) Sì. Con 4 porte: $ABC = \overline{\overline{A} + \overline{B} + \overline{C}} = \overline{\overline{A + A + A} + \overline{B + B + B} + \overline{C + C + C}}$.

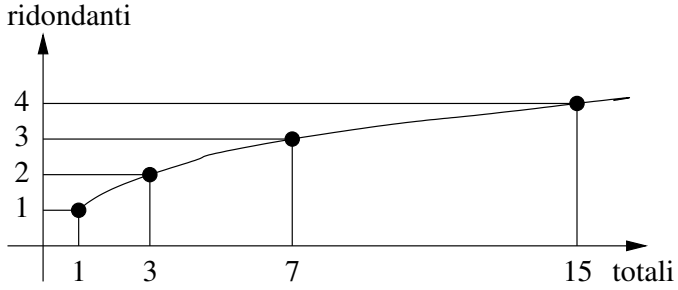
8. [INF] Disegnare il diagramma della macchina di Mealy che, leggendo simbolo dopo simbolo una sequenza indefinitamente lunga di caratteri appartenenti all'alfabeto $\mathcal{A} = \{\mathtt{A}, \mathtt{B}, \mathtt{R}, \omega\}$, riconosce le parole `ARABA`, `BAR`, `BARABBA`, `BARBA`. Il simbolo ω è adoperato esclusivamente per terminare ogni parola presente nella sequenza: per esempio, la sequenza `ARABBA`ω`BAR`$\omega\omega$`ARBA`ω contiene la sola parola riconoscibile `BAR`. Per semplicità si omettano dal disegno tutti gli archi associati a simboli appartenenti a parole non riconoscibili e le etichette degli stati non iniziali.

   R: (3 pt) Se tutti gli archi associati a simboli appartenenti a parole non riconoscibili, omessi in figura, producono in uscita per esempio il simbolo `B` ("bocciato"), allora la macchina disegnata qui sotto riconosce, partendo dallo stato etichettato, solo le parole richieste. Esse infatti vengono segnalate da una sequenza non vuota di simboli `A` ("accettato") aperta e terminata da `R` ("riconosciuto").



9. Si tracci una funzione che esprime il rapporto $\frac{\text{dati ridondanti}}{\text{dati totali}}$ dei codici di Hamming, fermandosi al valore in ascissa dati totali = 16.

   R: (3 pt) Per esempio:



10. Un protocollo I/O di tipo programmato è sempre meno efficiente di uno di tipo controllato? Si motivi la risposta.

R: (3 pt) Poichè la gestione dell'interruzione in un protocollo I/O di tipo programmato è più semplice, rapida ed economica dell'interruzione in un protocollo I/O di tipo controllato, allora il primo può essere più efficiente del secondo a patto che la richiesta di servizio arrivi in modo periodico e altamente frequente, in modo da soddisfare tutte (o quasi tutte) le interruzioni.

11. Che cos'è il *bus skew*?

    R: (3 pt) Il *bus skew* è il disallineamento temporale dei bit trasmessi lungo un bus parallelo, a causa dell'imperfetta sincronizzazione tra le linee che formano il bus.

12. Quali registri della CPU rendono possibile il costrutto di programmazione noto come *chiamata a procedura*?

    R: (3 pt) Il registro comunemente chiamato *link register* e, addizionalmente, il registro *stack pointer*.

13. In una memoria principale segmentata di 1 MB trovano posto segmenti tutti uguali di 8 kB. Nel tentativo di caricare un nuovo segmento, il sistema operativo trova la memoria piena. Quanti segmenti stanno in quel momento occupando la memoria nel caso di occupazione in assoluto più efficiente? Quanti nel caso di occupazione in assoluto meno efficiente? Qual è la percentuale di occupazione della memoria nei due casi?

    R: Se è piena, allora la regione di memoria in questione non contiene spazi liberi contigui di 8 kB. Nel caso di occupazione in assoluto più efficiente essa è occupata da $1024/8 = 128$ segmenti, corrispondenti a una percentuale di utilizzo del 100%, mentre nel caso di occupazione in assoluto meno efficiente esiste uno spazio di 8 kB meno 1 Byte tra ogni coppia di segmenti occupati di 8 kB adiacenti. In questo caso, dunque, i segmenti presenti sono $2^{20}/(8192 + 8191) \approx 64.004$. In altre parole, i 64 segmenti occupano in totale $64 \cdot (8192 + 8191) = 1048512$ Byte. La memoria libera ma inutilizzabile ammonta quindi al totale delle locazioni di 1 Byte lasciate vuote: $64 \cdot 8191 + (2^{20} - 1048512) = 64 \cdot 8191 + 64 = 64 \cdot 8192 = 2^{6+13} = 2^{20}/2$ Byte, dunque equivalenti esattamente al 50% della memoria principale.

14. [INF] Scrivere un programma in assembly per ARM il quale calcola la somma S dei primi $n + 1$ termini di una *serie geometrica* di ragione naturale positiva r: $S = \sum_{i=0}^{n} r^i$. Per esempio, se $r = 2$ e $n = 4$ avremo

$$S = \sum_{i=0}^{4} 2^i = 2^0 + 2^1 + 2^2 + 2^3 + 2^4 = 1 + 2 + 4 + 8 + 16 = 31.$$

    Per convenzione il programma legge r e n da due *word* consecutivi in memoria, e li carica rispettivamente nei registri `r1` e `r3`. Inoltre mantiene il registro `r4` per il risultato. Infine, salva il risultato nel word in cui inizialmente compariva r. Per semplicità si assuma in ogni caso $r \geq 0$ e $n \geq 0$, e non ci preoccupi del possibile *overflow* del risultato.

    R: (9 pt)

```
        .data
inout:
        .word 2,6               ; root, power
        .text
main:
        ldr r0, =inout          ; memory addr in r0
        ldr r1, [r0], #4        ; load root in r1, update r0
        mov r2, #1              ; set r2 to initial multiplier
        mov r4, #1              ; set result in r4 to power^0 = 1
        ldr r3, [r0]            ; load power in r3
        cmp r3, #0              ; if power null..
        beq fine                ; ..then go to end
loop:
        mul r2, r1, r2          ; r2 = r2*r1 = r1^n
        add r4, r4, r2          ; r4 = r4+r2
        subs r3, r3, #1         ; decrement power in r3
        bne loop                ; loop again if power is not zero
```

```
fine:
        ldr r0, =inout          ; memory addr in r0
        str r4, [r0]            ; save output

        swi 0x11                ; end
        .end
```