CORSO DI CALCOLATORI ELETTRONICI
CdL in Ingegneria Elettronica e Informatica
Prof. Mauro Iacono

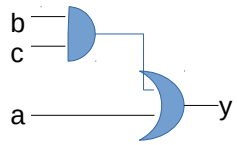
SOLUZIONE PROVA SCRITTA 9/1/2017

(gli studenti sono pregati di segnalare eventuali errori a mezzo email)

ESERCIZIO 1 (4 punti)

Semplificare la seguente funzione booleana applicando le regole dell'algebra di Boole; indicare esplicitamente tutti i passaggi svolti e disegnare la rete logica corrispondente:
$y = abc + a\overline{bc} + bc + a\overline{b} + a$

$y = a\,b\,c + b\,c + a\,\overline{b}\,\overline{c} + a\,\overline{b} + a = (a + 1)\;b\,c + a\,\overline{b}\,(\overline{c} + 1) + a = b\,c + a\,\overline{b} + a = b\,c + a\,(\overline{b} + 1) = b\,c + a$



Un errore comune è stato: $a\,b\,c + a\,\overline{b}\,\overline{c} = a\,(b\,c + \overline{b}\,\overline{c}) = a$. Questo è errato in quanto l'espressione tra parentesi non è sempre vera per qualsiasi valore di b e c, come si può facilmente verificare scrivendo la sua tabella di verità.

ESERCIZIO 2 (4 punti)

Calcolare il tempo di accesso medio per un sistema costituito da un processore, avente a sua disposizione due memorie cache L1 e L2 e una memoria RAM con le seguenti caratteristiche:

	L1	L2	RAM
Hit time (T)	2 cicli	15 cicli	100 cicli
Hit rate (P)	95% → P = 0.95	95% → P = 0.95	100% → P = 1

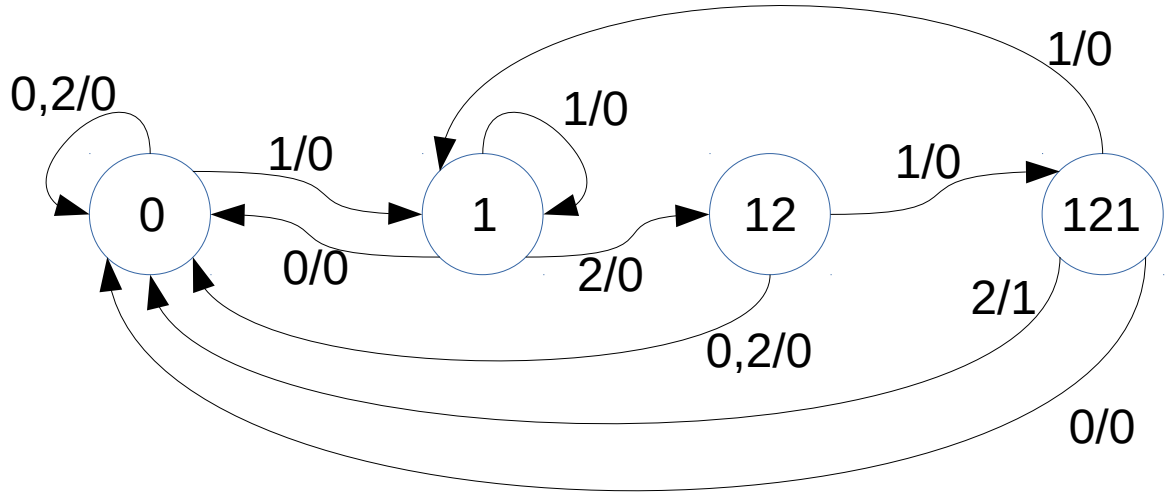
l'hit rate P di una cache è la probabilità di cercare un dato in cache e trovarlo: se esso viene trovato, il tempo di accesso è quello T della cache; nei restanti casi (ovvero con probabilità 1-P) il dato non viene trovato e quindi, dopo aver cercato invano con un tempo di accesso T, si deve procedere a cercare nel livello successivo, dove avviene lo stesso fenomeno, e così di nuovo fino alla RAM, che certamente contiene il dato cercato. Pertanto, essendo come da convenzioni correnti L1 la cache più vicina al processore e scrivendo le probabilità in forma non percentuale, si ottiene

$T_{\text{accesso medio totale}} = P_{L1} * T_{L1} + (1 - P_{L1}) * (T_{L1} + T_{\text{livello successivo}}) = P_{L1} * T_{L1} + (1 - P_{L1}) * (T_{L1} + (P_{L2} * T_{L2} + (1 - P_{L2}) * (T_{L2} + T_{RAM}))) = 0.95 * 2 + 0.05 * (2 + (0.95 * 15 + 0.05 * (100 + 15))) = 3$ cicli

ESERCIZIO 3 (8 punti)

Realizzare un automa di Mealy che riconosca la sequenza 1212 in un flusso continuo di simboli di ingresso appartenenti all'insieme di simboli di ingresso {0, 1, 2}. L'automa abbia un insieme di uscita composto da due simboli, uno dei quali rappresenti l'avvenuto riconoscimento della sequenza. L'automa ricominci a cercare la sequenza dall'inizio ogni volta che la sequenza sia stata riconosciuta, scartando i simboli della sequenza riconosciuta.
Il candidato riporti formalmente insieme di ingresso, insieme di uscita, grafo e tabelle dell'automa.

L'insieme degli ingressi è I = {0,1,2}, l'insieme delle uscite sia O = {0,1} con 0 che codifica "sequenza non trovata" e 1 che codifica "sequenza trovata", l'insieme degli stati sia S = {0, 1, 12, 121} che codificano rispettivamente "aspetto il primo 1", "aspetto il primo 2 perché è entrato il primo 1", "aspetto il secondo 1 perché è entrala la sequenza 12", "aspetto il secondo 2 perché è entrata la sequenza 121".

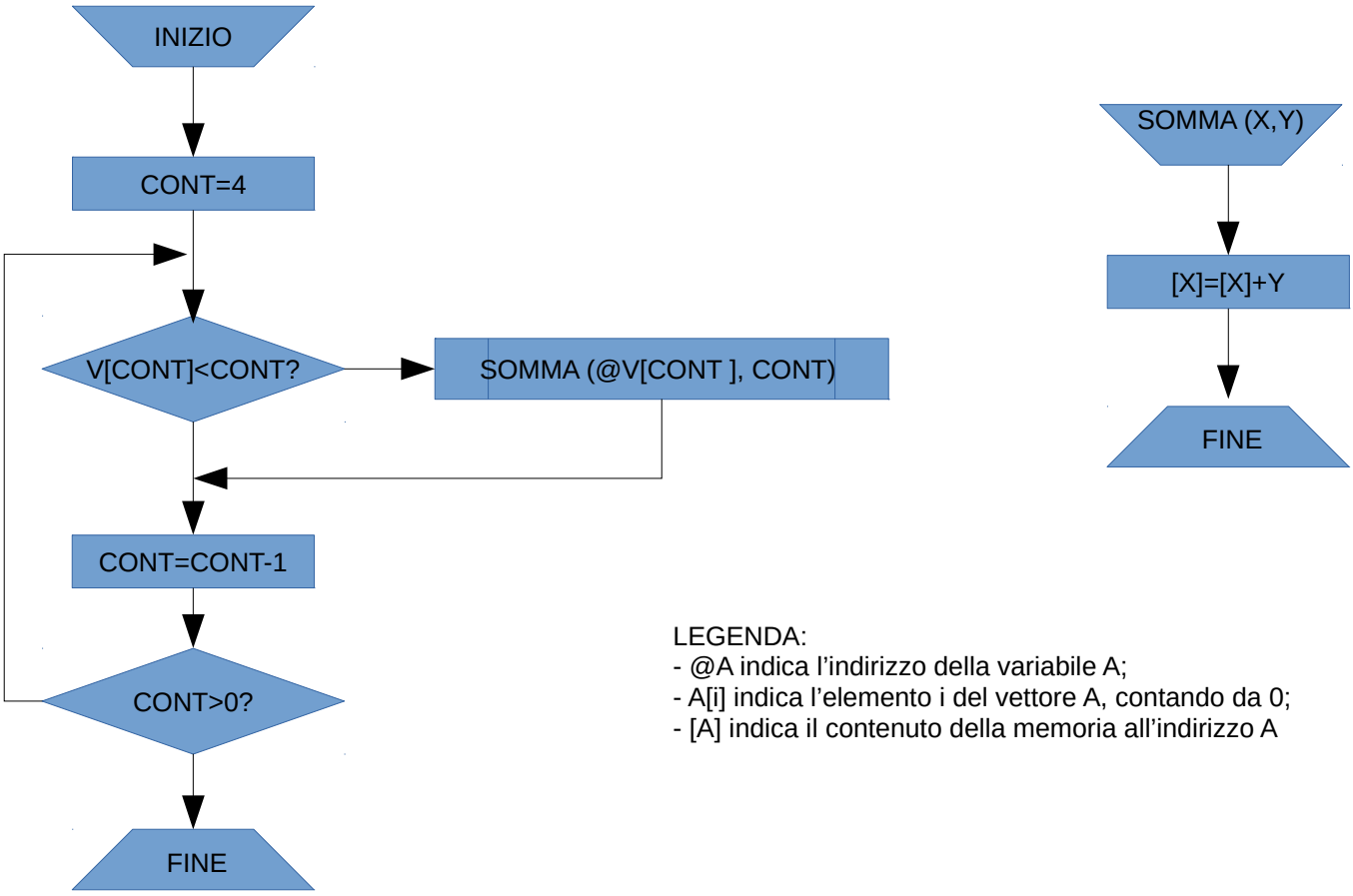


	I=0	I=1	I=2
S=0	S=0 / U=0	S=1 / U=0	S=0 / U=0
S=1	S=0 / U=0	S=1 / U=0	S=12 / U=0
S=12	S=0 / U=0	S=121 / U=0	S=0 / U=0
S=121	S=0 / U=0	S=1 / U=0	S=0 / U=1

ESERCIZIO 4 (14 punti)

Realizzare un programma in linguaggio assembly 8086 che visiti un vettore di interi positivi codificati a 16 bit di lunghezza fissata e, mediante un sottoprogramma che usi il meccanismo di passaggio di parametri mediante stack, sommi ad ogni numero minore della posizione occupata nel vettore la posizione da esso occupata nel vettore. Si riporti anche il diagramma di flusso utilizzato per progettare il programma.

Il programma principale effettuerà la visita per controllare se sussiste, per ogni elemento, la necessità di invocare il sottoprogramma; il sottoprogramma provvederà ad incrementare la locazione su cui viene chiamato, e pertanto necessita di ricevere come parametri l'indirizzo della locazione da modificare e l'incremento da applicare, o in alternativa l'indirizzo a cui inizia il vettore e l'incremento da applicare che fornisce anche la posizione da aggiornare nel vettore. Pertanto, qualora il programma principale debba invocare il sottoprogramma, prima dell'invocazione inserirà sullo stack una tra le due coppie di parametri di ingresso su identificati; in uscita dal sottoprogramma il vettore sarà già aggiornato e quindi non ci saranno parametri di uscita. Si noti che per convenzione la prima posizione di un vettore è identificata con l'indice 0, che è certamente non maggiore di tutti gli interi positivi: per cui, non potendosi mai verificare che il primo elemento del vettore contenga un valore minore di 0, si può omettere il controllo del primo elemento. Realizzando il controllo a partire dall'ultimo elemento si può beneficiare di LOOP, che ha il vantaggio di non effettuare il ciclo numero 0, ovvero, nel nostro caso, di non effettuare il controllo del vettore al posto 0, che abbiamo determinato essere inutile.



Il diagramma di flusso è stato disegnato pensando all'implementazione: ovviamente non vi compaiono i fatti puramente implementativi, come ad esempio quanto necessario per realizzare il passaggio di parametri e il salvataggio dei registri utilizzati dal sottoprogramma. Nella traduzione in assembly, inoltre, è necessario considerare che, poiché il vettore è un vettore di word, l'indice di posizione va convertito in numero di byte di spiazzamento dall'inizio del vettore, e quindi servirà una variabile ausiliaria in più che contenga il doppio dell'indice. Equivalentemente, quindi, si potrebbe riformulare il diagramma di flusso in modo che comprenda le operazioni legate a questo aspetto implementativo, o si potrebbe riportare tale secondo diagramma come trasformazione del primo in una forma più vicina all'implementazione.
Di seguito si riporta una soluzione possibile, commentata.

```
org 100h

.data
VET DW  1,3,1,6,8
L   EQU 4
.code
.start
    MOV CX, L         ; imposto contatore per LOOP
    LEA BX, VET       ; carico indirizzo del vettore in BX
LO: MOV DI, CX        ; carico DI con CX e poi, essendo il vettore
    SHL DI, 1         ; un vettore di word, moltiplico per 2
    CMP [BX][DI], CX ; VET[CX] >= CX?
    JGE PR            ; se non minore, salta la chiamata a SOMMA

                      ; ---------------------------------------------
```

```
                    ; |       SALVATAGGIO BASE STACK                |
    MOV BP, SP      ; altrimenti, salva in BP lo SP corrente e

                    ; ------------------------------------------------
                    ; |       PREPARAZIONE PARAMETRI                 |
    PUSH BX         ; prepara i parametri per SOMMA: in BX l'indirizzo
    PUSH DI         ; del vettore, in DI lo spiazzamento, e poi


    CALL SOMMA      ; chiama SOMMA

                    ; ------------------------------------------------
                    ; |       RIPRISTINO BASE STACK                  |
    MOV SP, BP      ; ripristina SP
PR: LOOP LO         ; passa al prossimo ciclo (saltando VET[0])
    RET             ; FINE

                    ; ------------------------------------------------
SOMMA PROC          ; |       PROCEDURA SOMMA (preserva BP)          |
    PUSH BP         ; salva BP del chiamante sullo stack
    MOV BP, SP      ; carica BP per accedere allo stack con spiazzamento

                    ; ------------------------------------------------
                    ; |       SALVATAGGIO DEI REGISTRI               |
                    ; |(si potrebbe omettere IN QUESTO CASO, in quan-|
                    ; |to sappiamo che non sono usati nel programma  |
                    ; |principale, e quindi si dovrebbe omettere di  |
                    ; |conseguenza il ripristino nel seguito)        |

    PUSH DI         ; salva DI, BX e AX del chiamante sullo stack
    PUSH BX
    PUSH AX

                    ; ------------------------------------------------
                    ; |       RECUPERO DEI PARAMETRI                 |
    MOV DI, 4[BP]   ; preleva il secondo parametro saltando BP e
    MOV BX, 6[BP]   ; indirizzo di ritorno e preleva il primo

    MOV AX, DI      ; carica in AX lo spiazzamento
    SHR AX, 1       ; dividi per 2 per avere l'incremento necessario
    ADD [BX][DI], AX ; incrementa VET[CX] di CX

                    ; ------------------------------------------------
                    ; |       RIPRISTINO DEI REGISTRI                |
    POP AX          ; ripristina AX, BX, DI e BP del chiamante
    POP BX
    POP DI
    POP BP
    RET             ; FINE SOMMA
SOMMA ENDP
```