M. Moro

esercizi di PROGRAMMAZIONE IN LINGUAGGIO ASSEMBLY per i calcolatori PDP11

Edizioni LIBRERIA PROGETTO Padova

EDIZIONI LIBRERIA PROGETTO
Via Marzolo, 28 - PADOVA

M. Moro

Ricercatore del Dipartimento di Elettronica
e Informatica. Università di Padova

esercizi di
PROGRAMMAZIONE IN LINGUAGGIO ASSEMBLY
per i calcolatori PDP11

Edizioni Libreria **Progetto** Padova 1987

A MARCELLA

INDICE

PREFAZIONE

Questo libro costituisce una guida all'apprendimento delle tecniche di programmazione in linguaggio assembly, introdotto tramite la presentazione di un numero rilevante di "esercizi". Ciascun esercizio richiede la soluzione di problemi, spesso non banali, dei quali viene fornita un'analisi attenta su cui si basa la proposta di soluzione che viene formulata nel linguaggio assembly dei calcolatori della famiglia PDP11.

Particolare attenzione e` stata posta nell'indicare, tramite esempi, che la soluzione dei problemi complessi puo` essere affrontata vantaggiosamente predisponendo un insieme di subroutine elementari da usare come strumenti per la costruzione di funzioni di complessita` via via crescente.

Il libro e` nato allo scopo di fornire un supporto alle esercitazioni associate all'insegnamento di Calcolatori Elettronici per gli allievi ingegneri elettronici presso la Facolta` di Ingegneria dell'Universita` di Padova. Al di la` di questa motivazione contingente, il lettore trovera` certamente, tra i molti esercizi proposti, gli spunti che gli consentiranno di affrontare anche altri problemi di programmazione in linguaggio assembly.

S. Congiu

Padova, ottobre 1987

INTRODUZIONE

Gli esercizi oggetto di questa pubblicazione costituiscono il risultato di uno sforzo di arricchimento e completamento della raccolta di esempi preparata nell'ambito dell'insegnamento di Calcolatori Elettronici e presentata agli studenti nel corso dei passati anni accademici. Ho ritenuto opportuno farla precedere da un capitolo dedicato a cenni di algebra booleana e alla rappresentazione numerica, affrontata in modo generale, con lo scopo di abituare il lettore ad applicare metodi di rappresentazione e di calcolo diversi da quelli della usuale notazione decimale. Nel successivo capitolo, l'assembly della famiglia PDP11 viene presentato in modo graduale, con facili esempi di prova. Il terzo e quarto capitolo sono dedicati allo sviluppo di programmi e subroutine, molte delle quali di uso generale, al fine di evidenziare, anche nel caso della programmazione assembly, l'importanza di suddividere il problema complessivo in sottoproblemi piu` semplici.

Desidero ringraziare vivamente il Prof. Sergio Congiu per l'incoraggiamento e i validi suggerimenti a me rivolti durante la preparazione di questo testo.

M. Moro

Padova, ottobre 1987

CAPITOLO 1

ESERCIZI PRELIMINARI

In questo capitolo vengono proposti alcuni esercizi propedeutici sulle funzioni booleane e sulla rappresentazione dei numeri. Si ritengono utili per la comprensione dei metodi di rappresentazione e di elaborazione delle informazioni nei calcolatori.

Esercizio 1.1

Date 3 variabili booleane A, B, C, scrivere la tabella di verita` per la seguente funzione:

$$F = (A \# B) \mid (A \& B \& C) \mid (\sim B \& \sim C)$$

ove gli operatori hanno il seguente significato:

```
~    not (unario)
#    or esclusivo
&    and
|    or
--------
```

Puo` essere utile dapprima ricordare le tabelle di verita` per le funzioni elementari costituite dagli operatori booleani (1=TRUE, 0=FALSE).

x	y	#	&	|	~(x)
0	0	0	0	0	1
1	0	1	0	1	0
0	1	1	0	1	1
1	1	0	1	1	0

Tab. 1.1

Supponendo che l'operatore ~ abbia priorita` superiore sugli altri e gli operatori # e & sull'operatore |, si ottiene che la funzione F e` a 1 se e` a 1 almeno uno dei tre termini separati da | cioe` (A#B), (A&B&C) e (~B&~C), 0 in caso contrario. Il primo e` 1 se e solo se A<>B; il secondo se e solo se A=B=C=1 e il terzo se e solo se B=C=0. Da cio` si ricava la seguente tabella (a fianco di ogni valore per F si sono indicati quali dei tre termini sono a 1):

n	A	B	C	F	
0	0	0	0	1	III
1	0	0	1	0	
2	0	1	0	1	I
3	0	1	1	1	I
4	1	0	0	1	I, III
5	1	0	1	1	I
6	1	1	0	0	
7	1	1	1	1	II

Tab. 1.2

Un modo alternativo di fornire la tabella e` quello di dare una matrice in cui gli ingressi sono separati in due gruppi, uno associato alle righe e uno alle colonne: ogni riga (colonna) e` associata ad una configurazione per il gruppo di ingressi corrispondenti che differisce da quelle associate alle righe (colonne) adiacenti solo per uno degli ingressi. L'adiacenza va intesa in senso circolare (l'ultima riga (colonna) e` adiacente alla prima e viceversa). Questa tabella e` detta *Mappa di Karnaugh*.

	A	0	0	1	1
	B	0	1	1	0
C	0	1	1	0	1
	1	0	1	1	1

Fig. 1.1

Questa rappresentazione, grazie alla particolare disposizione delle caselle, permette una rapida semplificazione della espressione logica. Infatti, data una mappa di Karnaugh si possono mettere in evidenza tutte e sole le caselle che contengono un 1 ed e` possibile esprimere la funzione booleana rappresentata mediante un *OR* multiplo di termini detti *minterm*, ciascuno associato ad una di queste caselle. Il minterm si costruisce con un *AND* multiplo tra tutti gli ingressi, ciascuno negato o meno a seconda che il valore dell'ingresso associato alla casella che si sta rappresentando sia rispettivamente 0 o 1. Quindi, nel caso in esame, F e` esprimibile come (l'operatore & e` implicito per economia di notazione):

F = ~A~B~C	~AB~C	A~B~C	~ABC	ABC	A~BC
n 0	2	4	3	7	5

Si puo` facilmente verificare che, con un metodo simile, e` possibile rappresentare la funzione come *OR* multiplo di termini ciascuno associato non piu` a singole caselle pari a 1 della mappa ma a gruppi di 2^m caselle di contenuto 1 e in posizione tale che m ingressi assumono, all'interno di ciascun gruppo, tutte le possibili combinazioni. In questo modo i termini in *OR* differiscono dai minterm in quanto non compaiono gli m ingressi appena menzionati, per cui la funzione risulta semplificata. Ad esempio, con m=2 si possono evidenziare quaterne di caselle adiacenti in senso esteso, cioe` su un'unica riga o colonna contigue strettamente oppure in senso circolare, oppure su due righe o colonne contigue in senso stretto o circolare. Un raggruppamento valido e` ad esempio quello costituito dalle quattro caselle che stanno ai vertici della

tabella.

Nell'esempio, si possono fare raggruppamenti al massimo di 2 caselle che, per soddisfare la regola, devono essere adiacenti. Raggruppando le caselle nel modo seguente (le caselle sono identificate dall'indice n della tabella di verita` equivalente):

(0, 2), (3, 7), (4, 5)
I II III

la funzione e` esprimibile nel modo seguente :

```
F = (~A~B~C | ~AB~C) |        I
    (~ABC | ABC) |            II
    (A~B~C | A~BC)    =       III
  = ~A~C(~B | B) | BC(~A | A) | A~B(~C | C) =
  = ~A~C | BC | A~B
```

dove si e` applicata la proprieta` distributiva e la eguaglianza:

(X | ~X) = 1

qualsiasi sia X. La nuova espressione ha lo stesso numero di termini in | di quella originaria ma ciascun termine ha un numero di fattori non superiore a 2.
++++++++

Esercizio 1.2

Esprimere una funzione booleana per la seguente tabella di verita` a 3 ingressi:

n	C	B	A	F
0	0	0	0	1
1	0	0	1	0
2	0	1	0	1
3	0	1	1	0
4	1	0	0	0
5	1	0	1	0
6	1	1	0	1
7	1	1	1	0

Tab. 1.3

Analogamente al precedente esercizio, risulta conveniente convertire la tabella in mappa di Karnaugh.

	A	0	0	1	1
	B	0	1	1	0
C					
0		1	1	0	0
1		0	1	0	0

Fig. 1.2

Raggruppando i termini di indice n (0, 2) e (2, 6) si ottiene:

$$F = \tilde{A}\tilde{C} \mid \tilde{A}B$$

La stessa espressione si poteva ottenere mediante manipolazione di quella ottenuta direttamente dalla tabella di verita` con la funzione *OR* di tutte le configurazioni che danno uscita 1:

$$\begin{aligned} F &= \tilde{A}\tilde{B}\tilde{C} \mid \tilde{A}B\tilde{C} \mid \tilde{A}BC = \tilde{A}\tilde{B}\tilde{C} \mid \tilde{A}B(\tilde{C} \mid C) = \\ &= \tilde{A}\tilde{B}\tilde{C} \mid \tilde{A}B = \tilde{A}\tilde{B}\tilde{C} \mid \tilde{A}B(1 \mid \tilde{C}) = \\ &= \tilde{A}\tilde{B}\tilde{C} \mid \tilde{A}B \mid \tilde{A}B\tilde{C} = \tilde{A}\tilde{C}(\tilde{B} \mid B) \mid \tilde{A}B = \\ &= \tilde{A}\tilde{C} \mid \tilde{A}B \end{aligned}$$

Nei passaggi si sono sfruttate le proprieta` commutativa e distributiva e l'eguaglianze:

$$X = X \,\&\, 1 = X \,\&\, (1 \mid Y)$$

valide per ogni X, Y.
++++++++

Esercizio 1.3

Si definisca la funzione *sommatore binario completo* (full binary adder).

La funzione sommatore completo e` una funzione a 3 ingressi e due uscite. Degli ingressi, due rappresentano i valori binari da sommare e il terzo un eventuale riporto, anch'esso da sommare, proveniente da sommatori precedenti. Le uscite sono la somma e il riporto ai sommatori successivi. Se la somma supera l'unita` occorre fornire un riporto in uscita. Il valore dell'uscita somma e` pari alla somma complessiva modulo 2.

C	riporto d'ingresso
A, B	valori binari
FS	uscita somma
FC	uscita riporto

C	A	B	FS	FC
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Tab. 1.4

Con la mappa di Karnaugh si ottiene:

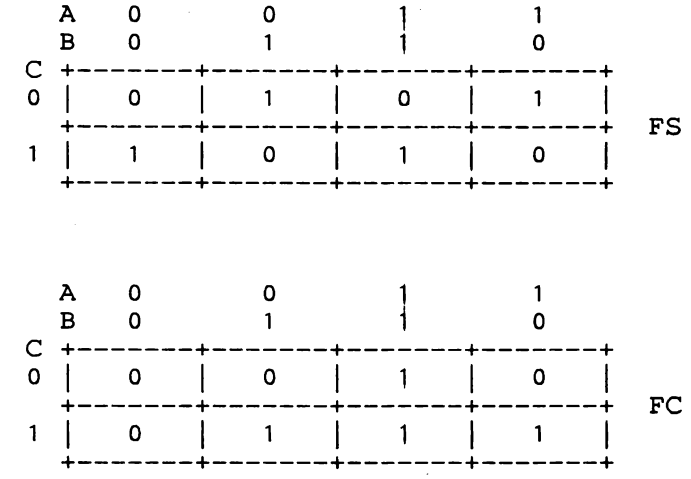


Fig. 1.3

Si ottiene quindi:

$$FS = \tilde{}AB\tilde{}C \mid A\tilde{}B\tilde{}C \mid \tilde{}A\tilde{}BC \mid ABC =$$
$$= C\,(AB \mid \tilde{}A\tilde{}B) \mid \tilde{}C\,(A\tilde{}B \mid \tilde{}AB)$$

Ma si verifica facilmente (vedi es. 1.1) che:

$$X\tilde{}Y \mid \tilde{}XY = X \# Y$$
$$XY \mid \tilde{}X\tilde{}Y = \tilde{}(X \# Y)$$

Detto Z=A#B si ha:

$$FS = C\tilde{}Z \mid \tilde{}CZ = C \# Z = C \# (A \# B)$$

$$FC = AC \mid BC \mid AB \text{ (dalla mappa di K.)} = AB \mid (A \mid B)C$$

$$FC = AB\tilde{}C \mid \tilde{}ABC \mid ABC \mid A\tilde{}BC \text{ (dalla tabella)} =$$
$$= AB(\tilde{}C \mid C) \mid (\tilde{}AB \mid A\tilde{}B)C = AB \mid (A \# B)C = AB \mid ZC$$

La seconda versione per FC e` quella generalmente utilizzata nelle realizzazioni cablate dei sommatori poiche` risultano complessivamente necessari: 2 EX-OR, 2 AND e 1 OR. Predisponendo una sequenza di n sommatori di questo tipo in cascata si ottiene un sommatore a n bit: il riporto d'uscita dal bit piu` significativo, detto generalmente *carry*, e` il riporto complessivo della somma.
++++++++

Esercizio 1.4

Utilizzando le funzioni &(AND) e ~(NOT) applicate ai singoli bit, si definisca:

a) una funzione Fa per ottenere il complemento dei bit da 3 a 6 di una parola A da 8 bit, mantenendo inalterati gli altri;

b) una funzione Fb che operi l'azzeramento dei bit da 0 a 3 e l'assegnazione del valore 1 ai bit 6 e 7 della parola A, mantenendo inalterati gli altri.

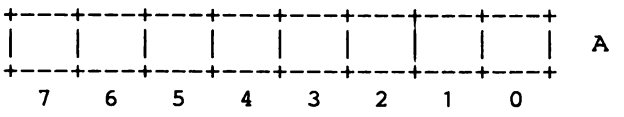


Fig. 1.4

a) Poiche` i bit interessati alla variazione sono quelli da 3 a 6, si fissi una maschera che ha questi bit a 1 e gli altri a 0 (d'ora in avanti si indichera` tra parentesi quadre la base, sempre espressa in decimale):

M = 01111000[2]

La funzione Fa e` data da:

Fa = (~A & M) | (A & ~M) = A # M

Infatti, per i bit 3<=k<=6 e 0<=h<=2 e h=7 si ha:

Fa(k) = (~A(k) & 1) | (A(k) & 0) = ~A(k)

Fa(h) = (~A(h) & 0) | (A(h) & 1) = A(h)

Poiche` pero` si dispone solo delle funzioni elementari & e occorre trasformare la funzione applicando le leggi di De Morgan:

X | Y = ~ (~X & ~Y)

X & Y = ~ (~X | ~Y)

Si ottiene quindi (l'operatore & viene reso implicito per economia di notazione):

Fa = ~AM | A~M = ~(~(~AM) ~(A~M)) = ~((A | ~M) (~A | M)) =
= ~(AA~ | ~M~A | AM | ~MM) = ~ (AM | ~A~M) =
= ~(AM) ~(~A~M)

b) Con considerazioni analoghe, si puo` porre:

M1 = 00001111[2] M2 = 11000000[2]

Fb = (A & ~M1) | M2

Infatti per 0<=k<=3, 4<=h<=5 e 6<=l<=7 si ha:

Fb(k) = (A(k) & 0) | 0 = 0

Fb(h) = (A(h) & 1) | 0 = A(h)

Fb(l) = (A(l) & 1) | 1 = 1

Fb = A~M1 | M2 = ~(~(A~M1) ~M2)

++++++++

Esercizio 1.5

Utilizzando la notazione posizionale, si convertano in ottale i seguenti numeri espressi in altra base (indicata fra parentesi):

123[10] 1101001[2] 1101001[3] 5AC3[16]

La notazione posizionale e` la rappresentazione di una quantita` a come successione di cifre $a_{n-1}a_{n-2}...a_1a_o$ che hanno un peso dipendente dalla posizione e pari ad una potenza della base b secondo la seguente espressione:

$$a = a_o*b^o + a_1*b^1 + \ldots + a_{n-2}*b^{n-2} + a_{n-1}*b^{n-1}$$

Le quantita` rappresentabili in questo modo sono del tipo $0<=a<=b^n-1$ e le singole cifre sono $0<=a_k<=b-1$.

La conversione da decimale ad ottale puo` essere ottenuta ricordando la fattorizzazione per un numero espresso con n cifre in notazione posizionale di base b:

$$a = a_o*b^o + a_1*b^1 + \ldots + a_{n-2}*b^{n-2} + a_{n-1}*b^{n-1} =$$
$$= ((..(a_{n-1}*b + a_{n-2})*b + a_{n-3})*b + \ldots)*b + a_1)*b + a_o$$

Le cifre della rappresentazione possono essere ottenute mediante n successive divisioni intere per la base (int e mod sono rispettivamente le funzioni che calcolano la parte intera e il resto della divisione):

$$q_1 = int(a/b) = ((..(a_{n-1}*b + a_{n-2})*b + \ldots + a_2)*b + a_1$$
$$r_1 = mod\ (a,\ b) = a_o$$

$$q_2 = int(q_1/b) = ((..(a_{n-1}*b + a_{n-2})*b + \ldots + a_3)*b + a_2$$
$$r_2 = mod\ (q_1,\ b) = a_1$$

...

$$q_{n-1} = int(q_{n-2}/b) = a_{n-1}$$
$$r_{n-1} = a_{n-2}$$

$$q_n = int(q_{n-1}/b) = 0$$
$$r_n = a_{n-1}$$

essendo $a_i < b$ per ogni i.

Pertanto con base b = 8. (per comodita` d'ora in avanti, salvo diversamente specificato, i numeri decimali saranno distinguibili da un punto terminale mentre i numeri privi di specificazione della base si intenderanno ottali):

$q_1 = int(123./8.) = 15.$ $r_1 = a_o = mod(123.,\ 8.) = 3$

$q_2 = int(15./8.) = 1$ $r_2 = a_1 = mod(15.,\ 8.) = 7$

$q_3 = int(1/8.) = 0$ $r_3 = a_2 = mod\ (1,\ 8.) = 1$

123. = 173

A verifica:

$$1 * 8.^2 + 7 * 8.^1 + 3 * 8.^o = 64. + 56. + 3 = 123.$$

Per i numeri binari, la conversione e` ottenibile in modo piu` semplice in quanto:

$$a = a_0*2^0 + a_1*2^1 + \ldots + a_{n-2}*2^{n-2} + a_{n-1}*2^{n-1} =$$
$$= (a_0 + a_1*2 + a_2*4) + (a_3 + a_4*2 + a_5*4)*8. + \ldots. +$$
$$(a_{k-2} + a_{k-1}*2 + a_k*4)*8.^{int((n-1)/3)} =$$
$$= c_0 * 8.^0 + c_1 * 8.^1 + \ldots + c_{m-1} * 8.^{m-1}$$

con:

$k = int((n-1)/3) * 3 + 2$
$a_{k-1} = 0$ se $k = n+1$
$a_k = 0$ se $k >= n$
m = numero cifre ottali = $int((n-1)/3)+1$

Le cifre della rappresentazione ottale si ottengono pertanto raggruppando a tre a tre a partire dal bit meno significativo le cifre della rappresentazione binaria. I casi piu` frequenti sono quelli con n=8. e n=16.. Con n=8.:

$k = int(7/3)*3 + 2 = 8. = n$
$a_{k-2} = a_6$
$a_{k-1} = a_7$
$a_k = 0$ $\qquad\qquad$ $m = 3$

quindi si ottiene un numero ottale di 3 cifre di cui la prima e` <= 3 (essendo il terzo bit nullo).

Con n = 16.:

$k = int(15./3)*3 + 2 = 17. = n + 1$
$a_{k-2} = a_{15}$
$a_{k-1} = 0$
$a_k = 0$
$m = 6$

quindi si ottiene un numero ottale di 6 cifre di cui la prima e` 0 o 1 (essendo il secondo e il terzo bit nulli).

Nel caso in esame:

1101001[2] = (001) (101) (001) = 151 con n=7, k=8., m=3 .

A verifica:

$1101001[2] = 2^6 + 2^5 + 2^3 + 1 = 105.$
$151 = 8.^2 + 5*8. + 1 = 105.$

Per la base 3, puo` essere conveniente dapprima passare a decimale applicando la definizione:

$1101001[3] = 1*3^6 + 1*3^5 + 1*3^3 + 1 = 1000.$

Convertendo ora da decimale a ottale con il metodo visto si ottiene:

$q_1 = int(1000./8.) = 125.$ $\qquad$ $a_0 = 0$
$q_2 = int(125./8.) = 15.$ $\qquad$ $a_1 = 5$
$q_3 = int(15./8.) = 1$ $\qquad$ $a_2 = 7$
$q_4 = 0$ $\qquad$ $a_3 = 1$
$1101001[3] = 1750$

Per la conversione da base 16, risulta piu` conveniente passare a quella binaria poiche`, per considerazioni analoghe al caso ottale, la conversione binario-esadecimale si ottiene per raggruppamento a quattro a quattro dei bit. Quindi:

5AC3[16] = (0101) (1010) (1100) (0011) = 0101101011000011[2] =
= (000) (101) (101) (011) (000) (011) = 055303
++++++++

Esercizio 1.6

Si convertano in decimale i seguenti numeri espressi in altra base:

234[8] 234[16] 1011010[2]

Per queste conversioni e` sufficiente applicare la definizione:

$234 = 2*8.^2 + 3*8. + 4 = 156.$

$234[16] = 2*16.^2 + 3*16. + 4 = 564.$

$1011010[2] = 2^6 + 2^4 + 2^3 + 2 = 90.$
++++++++

Esercizio 1.7

Si convertano in binario i seguenti numeri espressi in altra base:

147[8] D3[16] 281[10]

Per i primi due numeri e` sufficiente applicare il raggruppamento all'inverso:

147 = (001) (100) (111) = 1100111[2]

D3[16] = (1101) (0011) = 11010011[2]

Per il terzo si applica la regola della fattorizzazione, osservando che con b=2, il resto della divisione e` sempre 0 o 1 a seconda che il quoziente sia pari o dispari:

q_i	a_{i-1}	q_i	a_{i-1}
281	1	17	1
140	0	8	0
70	0	4	0
35	1	2	0
		1	1

281. = 100011001[2]
++++++++

Esercizio 1.8

Calcolare il campo dei numeri naturali rappresentabili con parole di 16. bit, espresso in base 5 e in base 7.

Il campo dei valori rappresentabili con 16 bit, espresso in base 2, e`:

$0 <= a < 2^{16}$

Mentre 0 (e anche 1) e` espresso allo stesso modo in tutte le basi, il limite superiore e`:

$2^{16} = 65536.$

Eseguendo la conversione a base 5 e 7 di tale numero si ottiene:

q_{k5}	65536	13107	2621	524	104	4	4
a_{k5}	1	2	1	4	4	0	4

q_{k7}	65536	9362	1337	191	27	3
a_{k7}	2	3	0	2	6	3

$0 <= a <= 4044121[5]$ $\quad$ $0 <= a <= 362032[7]$

++++++++

Esercizio 1.9

Eseguire le seguenti somme e sottrazioni di valori rappresentati con parole di 16. bit ma espressi in varie basi.

1011010111101011[2] +
0110110010001110[2]

1011010111101011[2] -
0100110001010110[2]

025417[8] +
043635[8]

131642[8] -
074156[8]

031642[8] -
174156[8]

4B26[16] +
30F5[16]

E915[16] -
4AB6[16]

0233412[5] +
0304424[5]

326145[7] -
254066[7]

Espressi i due numeri in notazione posizionale si ha:

$a = a_{n-1}*b^{n-1} + a_{n-2}*b^{n-2} + ... + a_1*b + a_o$
$c = c_{n-1}*b^{n-1} + c_{n-2}*b^{n-2} + ... + c_1*b + c_o$

$s = a + c = (a_{n-1} + c_{n-1})*b^{n-1} + (a_{n-2} + c_{n-2})*b^{n-2} + .. + (a_1 + c_1)*b + a_o + c_o$

Indicando con:

$s_o = a_o + c_o$
$s_k = int(s_{k-1}/b) + a_k + c_k$ $\quad$ per $k>0$

si ha che:

$$s = mod\ (a_o+c_o,\ b) + int\ ((a_o+c_o)/b)*b + (a_1+c_1)*b +\ldots =$$
$$= mod\ (s_o,\ b) + (int(s_o/b) + a_1 + c_1)*b + (a_2+c_2)*b^2 +\ldots =$$
$$= mod\ (s_o,\ b) + s_1*b + (a_2+c_2)*b^2 + \ldots =$$
$$= mod\ (s_o,\ b) + mod(s_1,\ b)*b + (int(s_1/b) + a_2 + c_2)*b^2 +\ldots =$$
$$= mod\ (s_o,\ b) + mod\ (s_1,\ b)*b + mod\ (s_2,\ b)*b^2 +..+$$
$$+ mod\ (s_{n-1},\ b)*b^{n-1} + int(s_{n-1}/b)*b^n$$

Essendo $a_k<=b-1$ e $c_k<=b-1$, risulta (per induzione):

$s_o = a_o+c_o <= 2*b-2$ $\qquad$ $int\ (s_o/b) <= 1$

$s_k = int(s_{k-1}/b) + a_k + c_k <= 2*b-1$
$int\ (s_k/b) <= 1$

$$mod\ (s_k,\ b) = s_k \qquad \text{per } s_k<b$$
$$= s_k - b \qquad \text{per } b<=s_k<=2*b-1$$

Quindi la somma va effettuata cifra per cifra, a partire da destra, sottraendo la base al risultato della somma e incrementando di uno la somma successiva se il risultato e` maggiore o uguale alla base. Riguardo all'ultimo termine della espressione di s, esso e` presente se vi e` un riporto dalla cifra piu` significativa e denota che il risultato non e` rappresentabile con n cifre. Nei processori aritmetici ove b=2 e n e` una potenza di 2, viene generalmente predisposto un flag (carry) per memorizzare il valore del riporto dal bit di indice n-1 e quindi segnalare, a somma effettuata, l'eventuale superamento di capacita`.

Per la differenza, va subito osservato che essa da` un risultato rappresentabile se e solo se a >= c. In questo caso si possono fare considerazioni analoghe al caso della somma:

$$d = a - c = (a_{n-1}-c_{n-1})*b^{n-1} + (a_{n-2}-c_{n-2})*b^{n-2} +..+$$
$$+ (a_1-c_1)*b + a_o - c_o$$

Indicando con:

$$d_o = a_o - c_o \qquad \text{per } a_o >= c_o$$
$$= a_o + b - c_o \qquad \text{per } a_o < c_o$$

$$d_k = a_k - c_k \qquad \text{per } d_{k-1}<=a_{k-1}-c_{k-1} \text{ e } a_k>=c_k$$
$$= a_k + b - c_k \qquad \text{per } d_{k-1}<=a_{k-1}-c_{k-1} \text{ e } a_k<c_k$$
$$= a_k - c_k - 1 \qquad \text{per } d_{k-1}>a_{k-1}-c_{k-1} \text{ e } a_k>=c_k$$
$$= a_k + b - c_k -1 \qquad \text{per } d_{k-1}>a_{k-1}-c_{k-1} \text{ e } a_k<c_k-1$$

si puo` verificare che la differenza e` esprimibile in:

$$d = d_{n-1}*b^{n-1} + d_{n-2}*b^{n-2} +..+ d_1*b + d_o$$

Infatti le varie possibilita` per $d_k$ tengono conto dell'eventuale richiesta di prestito dalla cifra precedente con un decremento unitario e qualora $a_k-c_k$ piu` l'eventuale decremento dessero un valore negativo, e` necessario richiedere un prestito unitario alla cifra successiva che e` di peso b volte superiore, garantendo un valore positivo per d_k. Quindi la differenza va eseguita cifra per cifra, a partire da destra e richiedendo un prestito in caso di risultato negativo, corrispondente ad un decremento della differenza successiva.

Per i valori proposti:

```
1011010111101011[2] +          1011000111101011[2] -
0110110010001110[2]            0100110001010110[2]
--------------------           --------------------
0010001001111001[2]            0110010110010101[2]
RRRRRR RR   RRR                 P  PP     P P
```

Si sono indicate con R e P rispettivamente le cifre in cui la somma da` riporto e le cifre per cui la differenza genera una richiesta di prestito. Si noti che per la somma vi e` un riporto dalla cifra piu` significativa: infatti, convertendo in decimale i due addendi, si verifica che la somma non e` rappresentabile con 16 bit.

```
025417[8] +          131642[8] -          031642[8] -
043635[8]            074156[8]           174156[8]
----------           ----------          ----------
071254[8]            035464[8]           635464[8]
  RR R                PP PP              PPP PP
```

Si noti che l'ultima differenza presenta un prestito sulla cifra piu` significativa, denotando la non correttezza del risultato poiche` in questo caso a<c.

```
4B26[16] +                     E915[16] -
30F5[16]                       4AB6[16]
----------                     ----------
7C1B[16]                       9E5F[16]
  R                             PPP

0233412[5] +                   326145[7] -
0304424[5]                     254066[7]
------------                   -----------
1043341[5]                     042046[7]
 R RR R                         P  PP
++++++++
```

Esercizio 1.10

Fornire il risultato di operazioni di scorrimento (shift) a destra e a sinistra per i seguenti valori a 16 bit, espressi pero` nella base indicata:

```
012345[8]   045716[8]   127764[8]   A5D5[16]    7B10[16]
2043241[5]  205346[7]
--------
```

Lo shift a sinistra di una cifra nella notazione posizionale in base b corrisponde alla moltiplicazione per b del numero. Infatti lo shift fa in modo che ciascuna cifra abbia un peso moltiplicato per b, supponendo di inserire una cifra 0 come nuova cifra meno significativa. Precisamente:

$$a = a_{n-1}*b^{n-1} + a_{n-2}*b^{n-2} + \ldots + a_1*b + a_0$$

shiftleft (a, b) =

$$= a_{n-1}*b^{n} + a_{n-2}*b^{n-1} + \ldots + a_1*b^2 + a_0*b + 0*b^0 =$$

$$= (a_{n-1}*b^{n-1} + a_{n-2}*b^{n-2} + \ldots + a_1*b + a_0) * b =$$

$$= a * b$$

Poiche` normalmente nello scorrimento a sinistra la cifra piu` significativa a_{n-1} viene persa, l'ultima eguaglianza va corretta nel modo seguente (trunc (x, m) e` il troncamento alle m cifre meno significative della rappresentazione di x):

$$\text{trunc (shiftleft (a, b), n)} = a_{n-2}*b^{n-1}+\ldots+a_o*b = \text{mod } (a*b, b^n)$$

Indicando con:

lshift (a, b, n) = trunc (shiftleft (a, b), n)

si ha che:

lshift (a, b, n) = a*b se e solo se $a_{n-1}=0$

cioe` se non vi e` perdita di cifre significative.

Per b=2 e n=16. lshift(a, 2, 16.) corrisponde ad una moltiplicazione per 2 se, come normalmente e` realizzato nei processori in modo aritmetico, il bit che va a sostituire quello meno significativo e` 0 e se $a_{15}=0$. Infatti in queste ipotesi:

$$0 <= a < 2^{15} \quad e \quad 0 <= a*2 < 2^{16}$$

e pertanto il prodotto e` ancora rappresentabile. In caso di superamento di capacita` ($a_{15}=1$) tale bit viene perso oppure piu` spesso salvato in un flag apposito (Carry).

Una moltiplicazione per 2 di un numero espresso in altra base puo` essere eseguita facilmente moltiplicando per 2 le singole cifre a partire da quella meno significativa. Infatti si ha che:

$$a = a_{n-1}*b^{n-1}+ a_{n-2}*b^{n-2}..+ a_1*b + a_o$$

$$\begin{aligned} a * 2 &= a_{n-1}*2*b^{n-1}+ a_{n-2}*2*b^{n-2}..+ a_1*2*b + a_o*2 = \\ &= \text{int}((a_{n-1}*2)/b)*b^n + \\ &\quad + (\text{mod}(a_{n-1}*2, b) + \text{int}((a_{n-2}*2)/b))*b^{n-1}+ \\ &\quad + (\text{mod}(a_{n-2}*2, b) + \text{int}((a_{n-3}*2)/b))*b^{n-2}+\ldots+ \\ &\quad + \text{mod}(a_1*2, b) + \text{int}((a_o*2)/b) \end{aligned}$$

Essendo:

$$\text{mod}(a_k*2, b) = \begin{cases} a_k*2 & \text{se } a_k < b/2 \\ a_k*2-b & \text{se } b > a_k >= b/2 \end{cases}$$

$$\text{int}(a_k*2/b) = \begin{cases} 0 & \text{se } a_k < b/2 \\ 1 & \text{se } b > a_k >= b/2 \end{cases}$$

Se ad esempio il numero e` espresso in ottale, per le cifre maggiori di 3, si sottrae 8. al doppio della cifra e si riporta 1 alla cifra successiva. Se espresso in esadecimale, per le cifre maggiori di 7, si sottrae 16. al doppio e si riporta 1.

Lo shift a destra, per ragioni analoghe al precedente, corrisponde ad una divisione per la base b del numero. Infatti il peso delle cifre viene diviso per b, supponendo che uno 0 sostituisca la cifra piu` significativa. Precisamente:

shiftright (a, b) =
$= a_{n-1}*b^{n-2} + a_{n-2}*b^{n-3} + .. + a_1 + a_o/b =$
$= (a_{n-1}*b^{n-1} + a_{n-2}*b^{n-2} + ... + a_1*b + a_o) / b =$
$= a / b$

Analogamente allo scorrimento a sinistra, nello scorrimento a destra viene persa la cifra meno significativa a_o. Pertanto vale la seguente:

int (shiftright (a, b)) $= a_{n-1}*b^{n-2} + .. + a_1 =$
$= int (a/b)$
$a_o = mod(a, b)$

Indicando con:

rshift (a, b, n) = int (shiftright (a, b))

si ha che:

rshift (a, b, n) = a/b se e solo se $a_o=0$

cioe` a e` un multiplo di b. In ogni caso:

$a = b * rshift(a, b, n) + a_o$

Per b=2 e n=16. rshift(a, 2, 16.) corrisponde ad una divisione per 2 se il bit che va a sostituire quello piu` significativo e` 0 e se $a_o=0$ cioe` il numero e` pari. Se a e` dispari, il bit $a_o=1$ viene perso oppure piu` spesso salvato nel flag carry come resto della divisione.

Una divisione per 2 di un numero espresso in altra base puo` essere eseguita in modo analogo alla moltiplicazione ma partendo dalla cifra piu` significativa. Infatti:

$a / 2 = (a_{n-1}/2)*b^{n-1} + (a_{n-2}/2)*b^{n-2} + .. + (a_1/2)*b + a_o/2 =$

Essendo:

$a_k/2 = int(a_k/2) + mod(a_k, 2)/2$

mod (a_k, 2) = 0 se a_k pari
= 1 se dispari

si ha:

$a / 2 = (a_{n-1}/2)*b^{n-1} + (a_{n-2}/2)*b^{n-2} + .. + (a_1/2)*b + a_o/2 =$
$= int (a_{n-1}/2)*b^{n-1} +$
$+ (mod (a_{n-1}, 2)/2)*b^{n-1} + (a_{n-2}/2)*b^{n-2} + .. + a_o/2 =$
$= int (a_{n-1}/2)*b^{n-1} +$
$+ ((mod(a_{n-1}, 2)/2)*b + (a_{n-2}/2))*b^{n-2} + .. + a_o/2 =$
$= int (a_{n-1}/2)*b^{n-1} +$
$+ ((mod(a_{n-1}, 2)*b + a_{n-2})/2)*b^{n-2} + .. + a_o/2$

Indicando con:

$a'_k = a_k$ per k=n-1
$= (mod(a'_{k+1}, 2)*b + a_k)$ per 0 <= k < n-1

si ottiene:

$$
\begin{aligned}
a / 2 &= \text{int}\,(a'_{n-1}/2)*b^{n-1} + \\
&\quad + (a'_{n-2}/2)*b^{n-2} + (a_{n-3}/2)*b^{n-3} + .. + a_0/2 = \\
&= \text{int}\,(a'_{n-1}/2)*b^{n-1} + \text{int}\,(a'_{n-2}/2)*b^{n-2} + \\
&\quad + ((\text{mod}(a'_{n-2},\ 2)*b + a_{n-3})/2)*b^{n-3} + .. + a_0/2 = \\
&= .. = \\
&= \text{int}\,(a'_{n-1}/2)*b^{n-1} + \text{int}\,(a'_{n-2}/2)*b^{n-2} + .. + \\
&\quad + \text{int}\,(a'_1/2)*b + \text{int}(a'_0/2)
\end{aligned}
$$

In altre parole si tratta di dividere per 2 le singole cifre da quella piu` significativa, tenendo pero` conto di sommare la base b alla cifra corrente se la divisione precedente e` stata effettuata su una cifra dispari.

Nel caso ad esempio della divisione per 2 su un numero ottale, si tratta di aggiungere 8. alla cifra corrente se la quantita` divisa per 2 al passo precedente era dispari (essendo la base pari, in questo caso basta osservare la cifra precedente). Per la base 16. occorre invece aggiungere 16.

numero	shift sinistra		shift destra	
012345[8]	024712	C=0	005162	C=1
045716[8]	113634	C=0	022747	C=0
127764[8]	057750	C=1	053772	C=0
A5D5[16]	4BAA	C=1	52EA	C=1
7B10[16]	F620	C=0	3D81	C=0
2043241[5]	4142032		1021343	C=0
205346[7]	414025		102523	C=0

In merito allo shift a sinistra degli ultimi due valori, va osservato che entrambi i risultati sono maggiori del piu` grande numero rappresentabile con 16. bit (vedi esercizio 1.8). Di conseguenza occorre applicare esplicitamente il modulo 2^{16} al risultato ottenendo:

lshift (2043241[5], 2, 16.) = mod (4142032[5], 4044121) =
= 4142032[5] - 4044121[5] = 0042411[5] e C=1

lshift (205346[5], 2, 16.) = mod (414025[7], 362032[7]) =
= 414025[7] - 362032[7]) = 021663[7] e C=1

Si suggerisce di effettuare la verifica delle operazioni convertendo operandi e risultati in decimale.
++++++++

Esercizio 1.11

Si esprima in notazione ottale su 16. bit la giustapposizione dei seguenti valori a 8. bit (espressi in ottale):

Byte piu` sign.	Byte meno sign.
301	205
000	300
177	377

Mentre i bit del byte meno significativo hanno lo stesso peso inseriti nel word giustapposizione, quelli del byte piu` significa-

tivo risultano moltiplicati per il peso $2^8 = (8.^3)/2$. Quindi, indicando con h, l e w rispettivamente byte piu` significativo, byte meno significativo e word, si ha che:

$$w = h*2^8 + l = int(h/2)*8.^3 + mod(h, 2)*4 + l =$$
$$= int(h_2/2)*8.^5 + (mod(h_2, 2) + int(h_1/2))*8.^4 +$$
$$+ (mod(h_1, 2) + int(h_0/2))*8.^3 + (mod(h_0, 2) + l_2)*8.^2 +$$
$$+ l_1*8. + l_0$$

Le tre cifre piu` significative di w si ottengono pertanto mediante uno shift a destra del byte piu` significativo mentre le tre cifre meno significative di w coincidono con quelle del byte meno significativo a meno che quello piu` significativo non sia dispari, nel qual caso si somma al byte meno significativo la quantita` 400[8].

301 || 205 = 140*8.^{3} + 400 + 205 = 140605

000 || 300 = 000300

177 || 377 = 077*8.^{3} + 400 + 377 = 077777

++++++++

Esercizio 1.12

Si separino i byte componenti dei seguenti word a 16. bit:

123456 000001 101010 152603

Con considerazioni analoghe all'esercizio precedente, si ha che:

$$l = trunc(w, 8.) = w_0 + w_1*8. + mod(w_2, 4)*8.^2$$

$$h = int(w/(2^8)) = int((w*2) / (8.^3)) =$$
$$= (w_5*2 + int(w_4/4))*8.^2 + (mod(w_4, 4)*2 + int(w_3/4))*8. +$$
$$+ mod(w_3, 4)*2 + int(w_2/4)$$

Il byte l si ottiene quindi dalle tre cifre meno significative di w, sottraendo 400 se >=400. Il byte h si ottiene dalle tre cifre piu` significative mediante moltiplicazione per due ed eventuale incremento se la quarta cifra ($w_2$) e` >=4.

w	h	l
123456	247	056
000001	000	001
101010	202	010
152603	325	203

++++++++

Esercizio 1.13

Si calcolino gli opposti dei seguenti numeri espressi in ottale a 16. bit secondo la convenzione del complemento a 2:

000146 144447 100010 0

Nella convenzione del complemento a b (base) (assunta pari) su n cifre, i numeri positivi sono rappresentati con la normale notazione posizionale ma il range rappresentabile e` compreso tra 0 e $(b^n/2-1)$. I numeri negativi -a sono rappresentati con il *complemento a b* della rappresentazione in base b del positivo corrispondente a), che e` definito come:

$$c'(b, n, a) = b^n - a$$

Per il campo dei valori rappresentabili di a positivo, risulta:

$$b^n/2+1 <= c'(b, n, a) <= b^n$$

In realta` si ritiene valido numero negativo anche quello rappresentato da $b^n/2$ che e` l'opposto del numero positivo $b^n/2$ (non rappresentabile). Quindi tutti i numeri negativi hanno una rappresentazione $>= b^n/2$.

Se si applica il complemento c' ad un numero negativo si ottiene:

$$c'(b, n, -a) = c'(b, n, c'(b, n, a)) = b^n - (b^n - a) = a$$

quindi applicando il complemento a b ad un numero positivo o ad un numero negativo si ottiene la rappresentazione dell'opposto. Un discorso a se` merita a=0 per cui:

$$c'(b, n, 0) = b^n$$

non e` rappresentabile con n cifre. A scopo di generalita`, si definisce il complemento in modo leggermente diverso come:

$$\begin{aligned} c(b, n, a) &= \text{trunc}\,(b^n-a, n) = \text{mod}(b^n-a, b^n) = \\ &= c'(b, n, a) \quad \text{per } a<>0 \\ &= 0 \quad \text{per } a=0 \end{aligned}$$

In questo modo si verifica che l'opposto di 0 coincida con 0: questa e` una proprieta` della convenzione che, in modo corretto, ammette una sola rappresentazione dello 0.

Questa rappresentazione e` coerente in quanto sommando su n cifre due numeri opposti si ottiene:

$$\begin{aligned} \text{add}(a, -a) &= \text{mod}(a + c(b, n, a), b^n) = \\ &= \text{mod}(a + \text{mod}(b^n-a, b^n), b^n) = 0 \end{aligned}$$

Il calcolo del complemento a b risulta piu` agevole se si passa dapprima al calcolo del *complemento a (b-1)* definito come:

$$\begin{aligned} c_1(b, n, a) &= (b^n-1) - a = \\ &= c(b, n, a) - 1 \quad \text{per } a<>0 \\ &= b^n-1 \quad \text{per } a=0 \end{aligned}$$

cioe`:

$$\begin{aligned} c(b, n, a) &= \text{mod}(c_1(b, n, a)+1, b^n) = \\ &= c_1(b, n, a)+1 \quad \text{per } a<>0 \\ &= 0 \quad \text{per } a=0 \end{aligned}$$

$c_1(b, n, a)$ e` facilmente calcolabile osservando che, rappresentando in modo analogo al precedente i negativi con il complemento a (b-1), si ha che:

$$add_1(a, -a) = a + ((b^n)-1) - a = b^n - 1$$

La quantita` $b^n-1$, rappresentata con n cifre in base b, e` costituita da n cifre eguali pari a (b-1). Quindi per ottenere l'opposto di un numero in questa rappresentazione, e` sufficiente per ogni cifra calcolarne la differenza con (b-1) cioe` il complemento a (b-1) (da cui il nome della rappresentazione; un discorso analogo si potrebbe fare per il complemento a b) non essendovi, nella somma poco sopra, alcun riporto da una cifra all'altra. Per inciso si noti che in questa rappresentazione l'opposto di 0 e` diverso da 0 (cioe` lo zero e` rappresentato in due modi distinti). Dal complemento a (b-1), mediante incremento di 1, si ottiene il complemento a b, trascurando l'eventuale riporto.

Ad esempio con b=10. e n=3, nella rappresentazione in complemento a 10., un numero a e` rappresentabile se 0<= a <=499. Il complemento a 10. di a e` dato da:

$$c(10., 3, a) = mod(1000. - a, 1000.)$$

mentre il complemento a 9 e` pari a:

$$c_1(10., 3, a) = 999. - a$$

Quindi, per esempio:

$$c(10., 3, 327.) = c_1(10., 3, 327.) + 1 = 672. + 1 = 673.$$

Infatti, sommando il numero e il suo opposto si ottiene:

$$mod(327. + 673., 1000.) = 0$$

Quando b=2, il calcolo e` ancora piu` semplice poiche` $c_1(2, n, a)$, per definizione, si ottiene negando (0->1, 1->0) i singoli bit della rappresentazione di a. Con n=16. i numeri positivi rappresentabili sono del tipo $0 <= a <= 2^{15}-1$ (32767.) e quelli negativi del tipo -2^{15} (-32768.) <= -a <=-1. Inoltre, con a positivo, 2^{15} (32768.) <= $c(2, 16., a) <= 2^{16}-1$ (65535.), cioe` tutti i numeri negativi sono rappresentati da configurazioni con il bit piu` significativo (a_{n-1}) pari a 1 e viceversa quelli positivi: tale bit assume pertanto il significato di bit di segno.

Concludendo, con b=2, dette neg(a)=c(2, n, a) e com(a)=$c_1(2, n, a)$ (assumendo n fissato), si ha che:

$$a + com(a) = a \mid com(a) = 2^n-1 \quad (\mid = OR \text{ bit a bit})$$

$$com(a) = 2^n - 1 - a$$

$$neg(a) = mod(2^n-a, 2^n) = mod(com(a)+1, 2^n)$$

Per i dati dell'esercizio, conviene pensare b=8. e n=6 e trascurare i due bit piu` significativi del risultato (essendo in realta` b=2 e n=16.) cioe` calcolare com e neg modulo 2^{16}.

a	com(a)	neg(a)
000146	177631	177632
144447	033330	033331
100010	077767	077770
0	177777	0

++++++++

Esercizio 1.14

Si esprima, in base ottale, la rappresentazione in complemento a 2 su 8. bit dei seguenti numeri decimali:

12. -12. -128. 255. 0

In base alle considerazioni dell'esercizio precedente, si ha che con b=2 e n=8. i positivi rappresentabili sono del tipo:

$0 <= a <= 2^7-1$ (127.)

e i negativi:

-2^7 (-128.) <= -a <= -1

Per il calcolo degli opposti, si puo` passare attraverso il complemento a (b-1) con b=8. e n=3 trascurando il bit piu` significativo dei risultati (cioe` calcolando mod (r, 400)).

12. = 14

-12. = mod (c_1(8., 3, 14)) + 1, 400) = mod (763 + 1, 400) = = 364

-128. = 200 (l'opposto non e` rappresentabile)

255. (non rappresentabile)

0 = 0

-1 = mod (c_1(8., 3, 1) + 1, 400) = mod (776 + 1, 400) = 377

++++++++

Esercizio 1.15

Si esprimano in decimale con segno i seguenti numeri rappresentati con la convenzione del complemento a 2 su 8. bit:

00011001(2) 11111000(2) E0[16] 13[8]

Si verifica facilmente che i numeri negativi sono riconoscibili in quanto, nella convenzione adottata, il bit 7 e` pari a 1, quando espressi in binario; sono maggiori di 7F[16], se espressi in esadecimale; sono maggiori di 177, se espressi in ottale. Da cio` si ha:

00011001(2) = + dec (00011001(2)) = +25.

```
    11111000(2) = - dec (com (11111000(2)) + 1) =
                = - dec (00000111(2) + 1) = - 8.
    E0[16] = - dec (c1(16, 2, E0[16]) + 1) = - dec (1F[16] + 1) =
           = -32.
    13[8] = + dec (13[8]) = +11.
++++++++
```

Esercizio 1.16

Verificare la condizione di overflow (non rappresentabilita` del risultato) sulla somma per i seguenti numeri a 5 bit, dapprima interpretati come numeri senza segno e successivamente come numeri con segno in complemento a 2.

```
    00011(2) + 11011(2)
    11111(2) + 11111(2)
    01100(2) + 01000(2)
    10100(2) + 11011(2)
--------
```

Come visto nell'esercizio 1.9, la condizione di superamento di capacita` sulla somma di n cifre in notazione posizionale di base b si verifica se vi e` un riporto dalla cifra piu` significativa (r_{n-1} = carry (C)).

Interpretando i numeri a e d con segno secondo la convenzione del complemento a b, la condizione di superamento, propriamente detta *overflow*, e` piu` complessa e va esaminata nei 4 casi rilevanti:

a) a>=0, d>=0

b) a<0, d<0

c) a>=0, d<0 , a >= -d

d) a>=0, d<0, a < -d

a) Risulta a+d positiva ed eseguendo la somma su n cifre, essendo a,d < $b^n/2$, risulta a+d < b^n. Pertanto:

$int(s_{n-1}/b) = 0$

L'overflow non si verifica se a+d<$b^n/2$ cioe`:

$mod\ (s_{n-1},\ b) < b/2 \iff$
$\iff mod\ (int(s_{n-2}/b)+a_{n-1}+d_{n-1},\ b) < b/2$

Per b=2, poiche` $a_{n-1}=d_{n-1}=0$ (entrambi positivi) e ponendo:

$r_{n-2}= int(s(_{n-2})/2)$

la condizione diventa:

$mod\ (r_{n-2}+a_{n-1}+d_{n-1},\ 2) < 1 \iff r_{n-2} = 0$

cioe` non vi deve essere riporto nel bit di segno dal bit precedente.

b) Si esegue la somma tra c(b, n, -a) e c(b, n, -d). La somma e` corretta cioe` e` pari a c(b, n, (-a)+(-d)) se la quantita` positiva (-a)+(-d) e` rappresentabile ovvero (-a)+(-d) < $(b^n)/2$. In questo caso si ha che:

$$c(b, n, (-a)+(-d)) = b^n - ((-a)+(-d)) >= b^n/2$$

Essendo:

$$c(b, n, -a) = b^n - (-a) >= b^n/2$$
$$c(b, n, -d) = b^n - (-d) >= b^n/2$$

si ha che:

$$c(b, n, -a) + c(b, n, -d) = b^n + (b^n - ((-a)+(-d))) > b^n$$

che coincide con c(b, n, (-a)+(-d)) solo trascurando il riporto dalla cifra n-1, riporto sempre presente in questo caso. La condizione di correttezza si traduce nella somma dei complementi in questo modo:

$$c(b, n, -a) + c(b, n, -d) = 2*(b^n) - ((-a)+(-d)) >= b^n + b^n/2$$

cioe`(con s_k, a_k, d_k riferiti ai complementi):

$$int(s_{n-1}/b) = 1$$

$$mod\ (s_{n-1}, b) >= b/2 \iff$$
$$\iff mod\ (int(s_{n-2}/b)+a_{n-1}+d_{n-1}, b) >= b/2$$

Per b=2, poiche` $a_{n-1}=d_{n-1}=1$ (entrambi negativi) la condizione diventa:

$$mod\ (r_{n-2}+a_{n-1}+d_{n-1}, 2) >= 1 \iff r_{n-2} = 1$$

cioe` vi e` riporto nel bit di segno dal bit precedente.

c) Se si sommano due numeri di segno diverso si ottiene sempre un numero rappresentabile se lo erano gli addendi poiche` il valore assoluto della somma e` inferiore a quello di entrambi, quindi l'overflow non si verifica mai. Se a >= -d <=> a-(-d)>=0 si ha:

$$a, -d < b^n/2$$

$$a + d = a + c(b, n, -d) = a + b^n - (-d) >= b^n \quad e$$

$$a + d = a + b^n - (-d) < b^n + b^n/2$$

Quindi, analogamente a prima:

$$int(s_{n-1}/b) = 1$$
$$mod\ (s_{n-1}, b) < b/2 \iff$$
$$\iff mod\ (int(s_{n-2}/b)+a_{n-1}+d_{n-1}, b) < b/2$$

Per b=2, poiche` $a_{n-1}=0$ e $d_{n-1}=1$ la condizione diventa:

$$r_{n-1} = 1$$
$$mod\ (r_{n-1}+a_{n-2}+d_{n-2}, 2) < 1 \iff r_{n-2} = 1$$

cioe` vi deve essere riporto nel bit di segno dal bit precedente.

d) L'unica differenza dal caso precedente e` che ora a < -d e quindi:

$$a + d = a + c(b, n, -d) = a + b^n - (-d) < b^n$$

Poiche` risulta una somma negativa rappresentabile deve essere:

$$a + d = a + b^n - (-d) >= b^n/2$$

Quindi, analogamente a prima:

$$int(s_{n-1}/b) = 0$$

$$mod\ (s_{n-1}, b) >= b/2 \quad <=>$$
$$<=> mod\ (int(s_{n-2}/b)+a_{n-1}+d_{n-1}, b) >= b/2$$

Per b=2, poiche` $a_{n-1}=0$ e $d_{n-1}=1$ la condizione diventa:

$$r_{n-1} = 0$$

$$mod\ (r_{n-1}+a_{n-2}+d_{n-2}, 2) >= 1 \quad <=> r_{n-2} = 0$$

cioe` non vi puo` essere riporto nel bit di segno dal bit precedente.

Riassumendo i 4 casi per b=2 e n qualsiasi, si puo` affermare che, utilizzando la convenzione del complemento a 2, la somma su n bit di due numeri da` la rappresentazione corretta del risultato se e solo se i riporti r_{n-2} e r_{n-1} rispettivamente nel bit di segno e dal bit di segno, coincidono cioe` se:

$$r_{n-2} \# r_{n-1} = 0$$

I processori che adottano questa convenzione posseggono generalmente un flag su cui viene caricato il valore $r_{n-2} \# r_{n-1}$ ed e` per questo chiamato flag di "overflow" (V).

Per gli esempi proposti, le condizioni sono:

b=2 n=5
senza segno V1 = C = r_{n-1}
con segno V2 = $r_{n-2} \# r_{n-1} = r_{n-2} \# C$

Nella tabella sono indicati anche i corrispondenti decimali con e senza segno.

a	d	somma	V1	V2
00011(2)	11011(2)	11110	0	0
3.	27. (-5.)	30. (-2)		
11111(2)	11111(2)	11110	1	0
31. (-1.)	31. (-1.)	30. (-2)		
01100(2)	01000(2)	10100	0	1
12.	8.	20. (-12)		
10100(2)	11011(2)	01111	1	1
20. (-12)	27. (-5)	15.		

++++++++

Esercizio 1.17

Verificare in quali condizioni lo shift a sinistra (destra) di una cifra nella notazione posizionale di base b corrisponde ad una moltiplicazione (divisione) per b se il numero a cui si applica e` con segno e rappresentato in complemento a b. Si verifichino i risultati nel caso b=2 e n=8. con i seguenti numeri (in ottale):

076 111 304 205 377

Con riferimento a quanto descritto nell'esercizio 1.10, occorre aggiungere i limiti di rappresentabilita` nella convenzione del complemento a b.

Per i numeri positivi a tali che:

$0 <= a < (b^{n-1})/2$

si ha che:

$0 <= a*b < b^n/2$

cioe` il prodotto (positivo) a*b e` ancora rappresentabile con n cifre e la sua rappresentazione equivale effettivamente a lshift(a, b, n).

Per i numeri negativi -a vale un discorso analogo, per cui se:

$-(b^{n-1})/2 <= -a <= -1$

si ha che:

$-b^n/2 <= -a*b <= -b$

e anche in questo caso il prodotto (negativo) -a*b e` rappresentabile. Si tratta di dimostrare che:

lshift(c(b, n, a), b, n) = c(b, n, a*b)

$a * b = a_{n-1}*b^n + a_{n-2}*b^{n-1} +..+ a_1*b^2 + a_o*b + 0*b^o$

Se a*b e` rappresentabile, deve essere $a<(b^{n-1})/2$ cioe` $a_{n-1}=0$ e $a_{n-2}<b/2$.

$$c(b, n, a*b) = c_1(b, n, a*b) + 1 =$$
$$= c_1(b, 1, a_{n-2})*b^{n-1} +..+$$
$$+ c_1(b, 1, a_1)*b^2 + c_1(b, 1, a_o)*b + (b-1) + 1 < b^n$$

$$c(b, n, a) = c_1(b, n, a) + 1 =$$
$$= (b-1)*b^{n-1} + c_1(b, 1, a_{n-2})*b^{n-2} +..+$$
$$+ c_1(b, 1, a_1)*b + c_1(b, 1, a_o) + 1$$

$$lshift(c(b, n, a), b, n) = trunc\ (shiftleft\ (c(b, n, a), b), n) =$$
$$= trunc\ (\ ((b-1)*b^n + c_1(b, 1, a_{n-2})*b^{n-1} +..+$$
$$+ c_1(b, 1, a_1)*b^2 + (c_1(b, 1, a_o) + 1)*b), n) =$$
$$= trunc\ (\ c(b, n, a*b), n) = c(b, n, a*b)$$

essendo $c(b, n, a*b) < b^n$.

Per lo shift a destra, a/b con a positivo e` rappresentabile e valgono le stesse considerazioni fatte per i numeri senza segno.

$$a/b = int\ (a_{n-1}*b^{n-2} + a_{n-2}*b^{n-3} + ..+ a_1 + a_o/b) =$$
$$= int\ (shiftright\ (a,\ b)) = rshift\ (a,\ b,\ n)$$

Se -a e` negativo e a>=b, anche -a/b e` negativo e rappresentabile con:

$$c(b,\ n,\ a) = c_1(b,\ n,\ a) + 1 =$$
$$= c_1(b,\ 1,\ a_{n-1})*b^{n-1} + c_1(b,\ 1,\ a_{n-2})*b^{n-2} + ..+$$
$$+ c_1(b,\ 1,\ a_1)*b + c_1(b,\ 1,\ a_o) + 1$$

$$rshift\ (c(b,\ n,\ a),\ b,\ n) = c_1(b,\ 1,\ a_{n-1})*b^{n-2} + ..+$$
$$+ c_1(b,\ 1,\ a_2)*b + c_1(b,\ 1,\ a_1) + int\ ((c_1(b,\ 1,\ a_o)+1)/b)$$

$$c(b,\ n,\ -a/b) = c_1(b,\ n,\ -a/b) + 1 =$$
$$= (b-1)*b^{n-1} + c_1(b,\ 1,\ a_{n-1})*b^{n-2} + ..+$$
$$+ c_1(b,\ 1,\ a_2)*b + c_1(b,\ 1,\ a_1) + 1$$

Definendo la funzione:

$$srshift(c(b,\ n,\ a),\ b,\ n) =$$
$$= (b-1)*b^{n-1} + rshift(c(b,\ n,\ a),\ b,\ n) \qquad \text{se } a_o=0$$
$$= mod((b-1)*b^{n-1} +$$
$$+ rshift(c(b,\ n,\ a),\ b,\ n)+1,\ b^n) \qquad \text{se } a_o<>0$$

si ha che:

$$srshift(c(b,\ n,\ a),\ b,\ n) = c(b,\ n,\ -a/b)$$

La funzione mod nel caso $a_o<>0$ e` necessaria per includere i valori -a<b che danno risultato nullo.

Quindi per i numeri negativi e` necessario adottare uno shift di tipo diverso che, dopo lo scorrimento, inserisce (b-1) come nuova cifra significativa e incrementa il risultato se $a_o<>0$.

Nel caso b=2 e n=8. la condizione di rappresentabilita` per lo shift a sinistra e`:

$$0 <= a <= 63. = 77$$

mentre la quantita` da sommare nel calcolo di srshift e`:

$$(b-1)*b^{n-1} = 128. = 200$$

e l'incremento va effettuato se il numero e` dispari.

Con i numeri proposti si ha:

076 = +62.
lshift (076, 2, 8.) = 174 = +124.
rshift (076, 2, 8.) = 037 = +31.

111 = +73.
lshift (111, 2, 8.) = 222 = -46.
(Infatti 111 e` superiore al limite di rappresentabilita` per lo shift)
rshift (111, 2, 8.) = 044 = +36.

```
    304 = c(2, 8., 074) = -60. (pari)
    lshift (304, 2, 8.) = 210 = c(2, 8., 170) = -120.
    srshift (304, 2, 8.) = 200 + rshift(304, 2, 8.) = 200 + 142 =
                         = 342 = c(2, 8., 036) = -30.

    205 = c(2, 8., 173) = -123. (dispari)
    lshift (205, 2, 8.) = 012 = +10. (shift non rappresentabile)
    srshift (205, 2, 8.) = mod(200 + rshift(205, 2, 8.) + 1, 2^8) =
           = mod(200 + 102 + 1, 2^8) = 303 = c(2, 8., 075) = -61.

    377 = c(2, 8., 1) = -1 (dispari)
    lshift (377, 2, 8.) = 376 = c(2, 8., 2) = -2
    srshift (377, 2, 8.) = mod(200 + rshift(377, 2, 8.) + 1, 2^8) =
                         = mod(200 + 177 + 1, 2^8) = 0
++++++++
```

Esercizio 1.18

Fornire un metodo di calcolo della moltiplicazione tra due numeri a, d senza segno di n cifre in base b utilizzando operazioni piu` elementari.

Si osservi dapprima che, essendo gli operandi:

$$a,d <= b^n-1,$$

il prodotto risulta:

$$a*d <= (b^n -1)^2 < b^{2*n}$$

quindi e` sicuramente rappresentabile mediante 2*n cifre nella stessa base. Utilizzando la notazione posizionale, espansa per uno dei fattori, si ha:

$$\begin{aligned} a * d &= a_{n-1}*d*b^{n-1} +..+ a_1*d*b + a_o*d = \\ &= ((..(0*b + a_{n-1}*d)*b + a_{n-2}*d)*b +..+a_1*d)*b +a_o*d \end{aligned}$$

La prima forma esprime di fatto il metodo adottato nel calcolo manuale della moltiplicazione mentre la seconda espressione (fattorizzata) si presta meglio ad una realizzazione automatica in quanto si configura come la iterazione eseguita n volte dell'operazione piu` elementare:

$$s_k = s_{k-1}*b + a_{n-k}*d$$

con:

$$s_o = 0$$

$$s_n = a*d = s_{n-1}*b + a_o*d$$

Il prodotto $s_{k-1}*b$ corrisponde ad uno shift di una cifra a sinistra su 2*n cifre mentre il prodotto $a_{n-k}*d$ puo` essere eseguito direttamente disponendo di un moltiplicatore n*1 cifre oppure con al massimo (b-1) somme ripetute di d, fornendo un risultato contenuto in (n+1) cifre.

$$a_{n-k}*d < (b-1)*b^n < b^{n+1}$$

Il calcolo si semplifica ulteriormente nel caso b=2 poiche` 0 <= a_{n-k} <= 1, pertanto ogni passo s_k consiste in uno shift e un somma (eventuale) per d su 2*n bit.

CAPITOLO 2

INTRODUZIONE ALL'ASSEMBLY PDP11

In questo capitolo vengono riassunte in breve le caratteristiche essenziali del processore PDP11 e del suo linguaggio assembly. Gli esercizi proposti hanno lo scopo di fornire una prima base di studio delle istruzioni del linguaggio. In questo capitolo e nei successivi si adottera` nella descrizione e nei commenti al testo, ai programmi e alle figure, la seguente convenzione:

nnnnnn oppure ALFA	quantita` numerica a 16 bit (ad esempio indirizzi) (i numeri sono generalmente in ottale)
Rn	contenuto del registro Rn; quando e` necessario distinguere un registro dal suo contenuto, quest'ultimo si indica con (Rn).
M[nnnnnn] oppure M[ALFA]	contenuto della locazione di indirizzo nnnnnn o ALFA
M[Rn]	contenuto della locazione puntata dal registro Rn
M[M[nnnn]] oppure M[M[ALFA]]	contenuto della locazione puntata dalla locazione di indirizzo nnnnnn o ALFA

2.1 - L'unita` centrale e la memoria

Con la sigla PDP11/xx si identifica una famiglia di elaboratori prodotti dalla DEC e aventi una architettura incentrata attorno al bus UNIBUS. I vari processori centrali appartenenti a questa famiglia differiscono per alcune caratteristiche particolari e per potenza elaborativa; condividono altresi` gran parte del set di istruzioni e a questo nucleo comune si riferira` la trattazione seguente. Per ogni maggior dettaglio si rimanda alla letteratura specifica.

Per il PDP11 l'unita` fondamentale di elaborazione e` la parola (word) da 16 bit: molte istruzioni occupano un word e della stessa dimensione e` pure un indirizzo di memoria. Cio` nonostante, un indirizzo identifica una locazione elementare di memoria di 8 bit (byte), percio` la memoria direttamente indirizzabile e` estesa 64 Kbyte (2^{16} byte).

Alcune istruzioni sono in grado di trattare byte, altre trattano word interi. Un word in memoria centrale e` contenuto in due successive locazioni: indicando simbolicamente con ALFA il suo riferimento, ALFA e` l'indirizzo, necessariamente pari, del byte meno significativo del word. Il byte piu` significativo e` contenuto all'indirizzo ALFA+1. Il tentativo di accedere ad un word di indirizzo dispari viene considerato illegale (fig. 2.1).

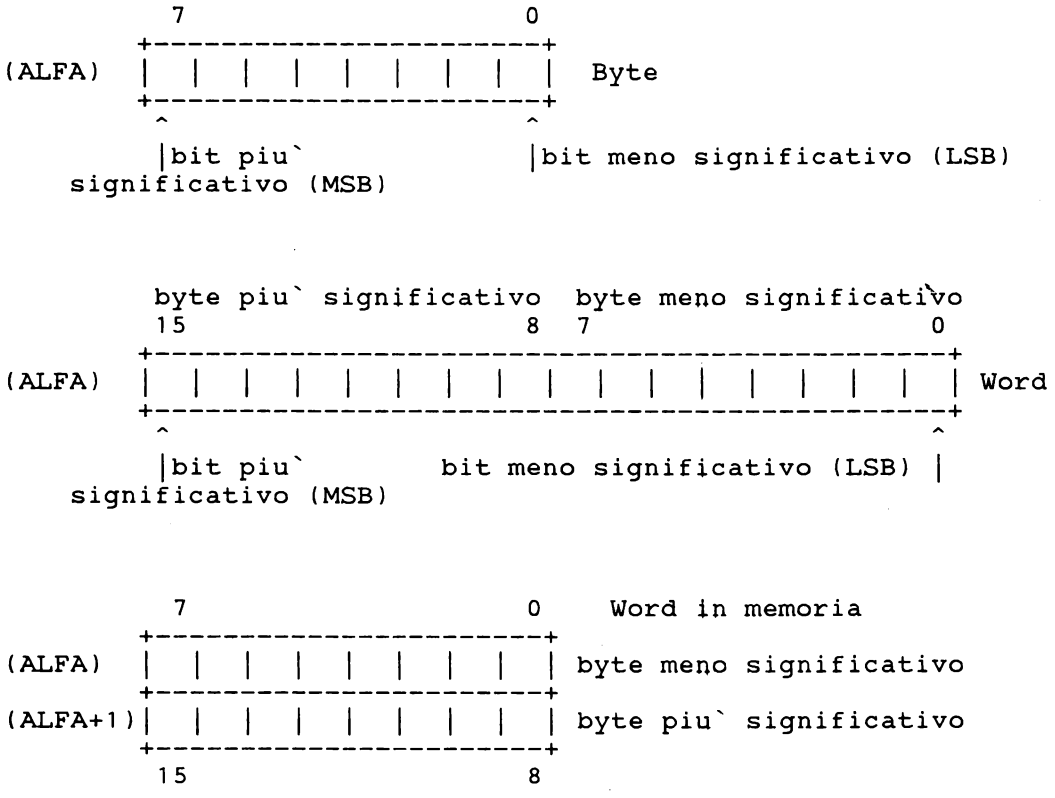


Fig. 2.1

L'unita` centrale dispone di 8 registri da 16 bit (simbolicamente indicati con %0..%7 oppure R0..R7) di cui i primi 6 sono di uso generale ed equivalenti (R0..R5) e vengono utilizzati per specificare gli operandi in istruzioni che ne prevedono almeno uno. I registri R6 e R7 hanno un significato particolare, anche se in linea di principio possono essere utilizzati come tutti gli altri. Il registro R6 funge da *Stack Pointer* (SP) cioe` da puntatore (contenitore di un indirizzo di memoria) che individua, in ogni istante, il word che sta in cima ad una coda LIFO (Last In First Out) che cresce per indirizzi successivamente inferiori. Lo stack puntato da SP e` utilizzato direttamente dal processore per memorizzare gli indirizzi di ritorno dalle subroutine e dalle routine di interruzione (vedi oltre) e per questo si parla di *Hardware Stack*. Il registro R7 funge da *Program Counter* (PC) ed e` utilizzato per contenere in ogni istante l'indirizzo del word in cui e` contenuta la prossima istruzione da eseguire.

L'unita` centrale contiene anche un registro di stato (Processor Status Word PSW) di 16 bit la cui parte d'interesse ha il seguente significato (fig. 2.2):

- bit 0: C Carry
posto a 1 se un'operazione ha provocato un riporto dal bit piu` significativo.

- bit 1: V Overflow
posto a 1 se un'operazione ha provocato superamento di capacita` aritmetica (in complemento a 2)

- bit 2: Z Zero
posto a 1 se il risultato di un'operazione e` pari a 0

- bit 3: N Negative
posto a 1 se il risultato di un'operazione e` negativo secondo la convenzione del complemento a 2 (copia del MSB)

- bit 4: T Trace Trap
se posto a 1, provoca un trace trap dopo l'esecuzione di ciascuna istruzione

- bit 5,6,7: Priority
fissa il livello corrente di priorita` del processore nella risposta alle interruzioni

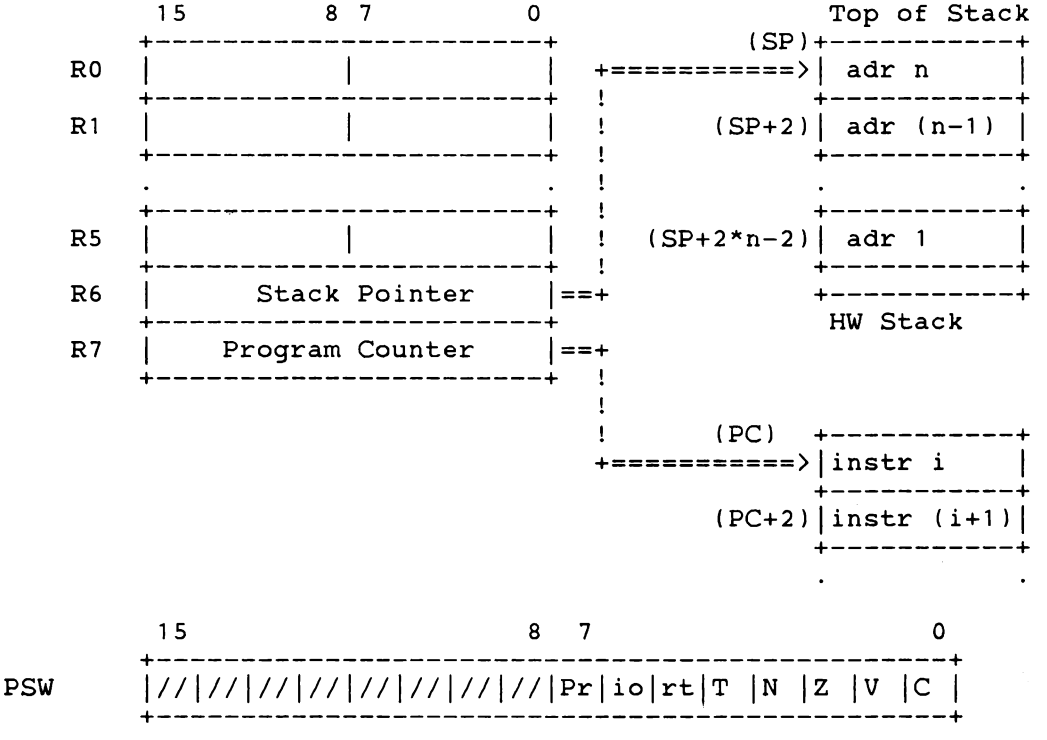
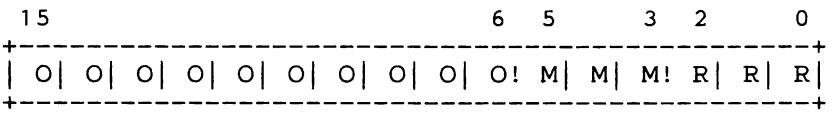


Fig. 2.2

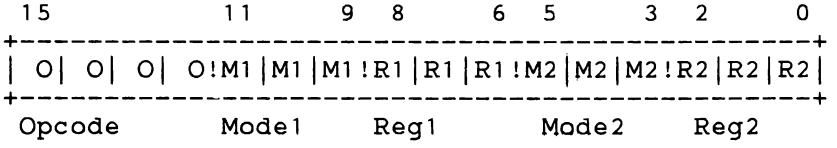
2.2 - Modalita` di indirizzamento

Una istruzione prevede l'indicazione di un Codice Operativo (OPCODE) e di zero, uno o due operandi (OPNDi). Gli operandi sono individuati dalle informazioni contenute nei campi Mode-Reg (nel complesso chiamati *Indirizzo Apparente*) da cui il processore ricava lo *Indirizzo Effettivo* in base ad un algoritmo che varia secondo la *Modalita` di Indirizzamento* specificata.

Una tipica istruzione ad un operando ha il formato di fig. 2.3a. Il campo Opcode, che contiene il codice operativo dell'istruzione, occupa i bit 6..15, mentre il campo Mode-Reg, che individua un operando, occupa i bit 0..5. Quest'ultimo e` suddiviso nel campo Registro (bit 0..2) che specifica un registro (0..7) della CPU e nel campo Modo (bit 3..5) che specifica la modalita` di indirizzamento.



Istruzione ad un operando
Fig. 2.3 a



Istruzione a due operandi
Fig. 2.3 b

Una istruzione e` simbolicamente indicata nel modo seguente:

OPCODE [OPND1 [, OPND2]] [; Commento]

([X] significa X opzionale) dove OPCODE e` un mnemonico per il codice operativo e OPND1, OPND2 sono gli operandi la cui specificazione dipende dalla modalita` di indirizzamento, come riassunto in tabella 2.1.

n	Modalita`	Simbolo	Commento
0	Registro o Diretto	Rn	l'operando nel registro specificato dal campo Reg
1	Registro Differito o Indiretto	(Rn) o @Rn	l'operando e` nella locazione di memoria puntata dal registro specificato ovvero Rn contiene l'indirizzo effettivo dell'operando
2	Autoincremento (Indiretto)	(Rn)+	Rn contiene l'ind. effettivo dell'operando; Rn viene incrementato della ampiezza in byte dell'operando
3	Autoincremento (Indiretto) Differito	@(Rn)+	M[Rn] contiene l'ind. effettivo dell'operando; Rn viene incrementato di 2
4	Autodecremento (Indiretto)	-(Rn)	Rn viene decrementato del numero di byte dell'operando; Rn contiene l'ind. effettivo dell'operando

Tab. 2.1

5	Autodecremento (Indiretto) Differito	@-(Rn)	Rn viene decrementato di 2; M[Rn] contiene l'ind. effettivo dell'operando
6	Indice (Indiretto)	X(Rn)	(Rn)+X e` l'ind. effettivo dell'operando
7	Indice (Indiretto) Differito	@X(Rn)	M[(Rn)+X] contiene l'ind. effettivo dell'operando

Tab. 2.1 (cont)

L'insieme delle istruzioni del PDP11 si puo` dividere in quattro classi:

a) Istruzioni a singolo operando. (fig. 2.3a)
Sono istruzioni che prevedono un solo operando che in generale ha il ruolo contemporaneo di sorgente e destinazione. Appartengono a questa classe le istruzioni (il suffisso opzionale B sta ad indicare istruzioni che operano su byte): CLR(B), COM(B), INC(B), DEC(B), NEG(B), ADC(B), SBC(B), TST(B), ROR(B), ROL(B), ASR(B), ASL(B), SWAB, SXT, MFPS, MTPS.

b) Istruzioni a doppio operando. (fig. 2.3b e fig. 2.6)
Sono istruzioni che prevedono un primo operando sorgente e un secondo operando destinazione oppure sorgente-destinazione. Appartengono a questa classe le istruzioni: MOV(B), CMP(B), BIT(B), BIC(B), BIS(B), ADD, SUB, MUL, DIV, XOR.

c) Istruzioni di controllo del flusso.
Sono istruzioni il cui effetto principale e` la modifica (eventualmente condizionata) del PC. Si possono dividere in:

c1 - istruzioni di salto limitato (o relativo o di diramazione) (fig. 2.7b): BR, BNE, BEQ, BPL, BMI, BVC, BVS, BCC, BCS, BGE, BLT, BGT, BLE, BHI, BLOS, BHIS, BLO, SOB

c2 - istruzione di salto incondizionato (fig. 2.3a): JMP

c3 - istruzioni di chiamata a subroutine e ritorno (fig. 2.7a e 2.7d): JSR, RTS

c4 - istruzioni di interruzione sincrona (trap) e di ritorno da interruzione: EMT, TRAP, BPT, IOT, RTI, RTT

d) Istruzioni su bit di condizione (fig. 2.11).
Sono istruzioni che modificano specificamente singoli bit o gruppi di bit di condizione. Appartengono a questa classe le istruzioni: CLC, CLV, CLZ, CLN, CCC, SEC, SEV, SEZ, SEN, SCC.

e) Altre istruzioni
A questa classe appartengono le istruzioni: HALT, WAIT, RESET, NOP.

Esercizio 2.1

Sapendo che il codice operativo dell'istruzione:

```
INC     OPND
```

che incrementa di una unita` l'operando, e` 0052, si codifichi l'istruzione:

```
INC     R3
```

e si dica quali effetti produce.

Il registro da specificare e` il numero 3 e l'indirizzamento e` quello diretto (Mode = 0). Pertanto:

```
005203  INC     R3
```

L'istruzione equivale a:

```
R3 + 1 -> R3
```

assegnando nuovi valori ai flag N, Z, V in conseguenza del risultato. Ad esempio, con i seguenti valori iniziali di R3 si ha:

```
R3=0          1 -> R3          0 -> NZV
R3=177777     0 -> R3          2 -> NZV
R3=077777     100000 -> R3     5 -> NZV
```

Inoltre viene eseguito:

```
PC + 2 -> PC
++++++++
```

La modifica dei flag e` conseguente all'operazione effettuata sullo o sugli operandi specificati e non gia` per effetto dell'incremento o decremento del puntatore per le modalita` 2..5.

Si noti che per le modalita` 6 e 7, X e` rappresentato dal contenuto della parola successiva a quella che contiene l'istruzione e quindi in questo caso l'istruzione occupa di fatto 2 word. La somma (Rn)+X va intesa su 16 bit, trascurando l'eventuale riporto. X e` detto "Spostamento" (Offset). (fig. 2.4)

Nella specificazione del registro, Rn puo` essere uno qualsiasi degli 8 registri generali. In particolare, se Rn = R7 = PC, alcune modalita` assumono un significato particolare, riassunto in tabella 2.2.

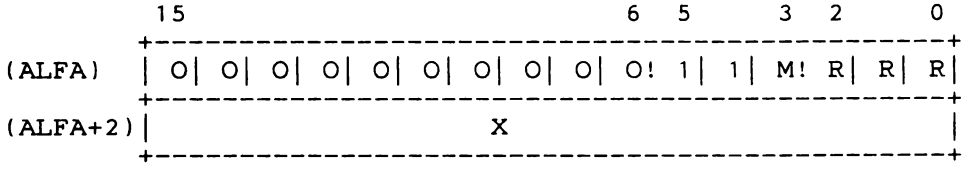


Fig. 2.4

n	Modalita`	Simbolo	Commento
2	Immediato	#nnnnnn o #ALFA	l'operando e` quello che segue l'istruzione
3	Assoluto	@#nnnnnn o @#ALFA	il word che segue l'istruzione contiene l'ind. effettivo dell'operando
6	Relativo	X(PC) o ALFA	L'ind. eff. dell'operando e` pari a ALFA = X+Ind.Istruz.+4
7	Relativo Differito	@X(PC) o @ALFA	L'ind. eff. dell'operando e` contenuto in M[ALFA] = M[X+Ind.Istr.+4]

Tab. 2.2

Esercizio 2.2

Nelle seguenti ipotesi:

```
R0 = 003002          R1 = 003020          NZVC = 00
M[003000] = 003040   M[003002] = 003060   M[003020] = 003100
M[003040] = 003010   M[003060] = 003020   M[003100] = 003050
```

e sapendo che l'istruzione:

```
CLRB    OPND
```

con Opcode = 1050 azzera il byte meno significativo dell'operando, si traducano e si descrivano gli effetti di ciascuna delle seguenti istruzioni, supponendo che venga caricata all'indirizzo 001000:

```
a)  INC     R0          b)  INC     (R0)+
c)  INC     -(R0)       d)  INC     16(R0)
e)  INC     -20(R1)     f)  INC     (R0)
g)  INC     @(R0)+      h)  INC     @-(R0)
i)  INC     @16(R0)     l)  INC     @#003000
m)  INC     003000      n)  INC     @003000
o)  CLRB    (R0)+       p)  CLRB    -(R0)
q)  CLRB    @(R0)+      r)  CLRB    @-(R0)
--------
```

Per ciascun caso viene qui di seguito fornita la traduzione, preceduta dall'indirizzo di caricamento e seguita dall'indirizzo dell'eventuale successiva istruzione, nonche` le modifiche significative apportate a regestri, bit di condizione e locazioni di memoria.

```
    Ind.    Word1   Word2   Simbolo
-------------------------------------------------------------
a)  001000  005200          INC     R0
    001002
    R0 <- 003003   NZVC <- 00

b)  001000  005220          INC     (R0)+
    001002
    M[003002] <- 003061   R0 <- 003004   NZVC <- 00
```

```
c)  001000  005240          INC     -(R0)
    001002
    R0 <- 003000   M[003000] <- 003041   NZVC <- 00

d)  001000  005260  000016  INC     16(R0)
    001004
    M[003020] <- 003101   NZVC <- 00

e)  001000  005261  177760  INC     -20(R1)
    001004
    M[003000] <- 003041   NZVC <- 00

f)  001000  005210          INC     (R0)
    001002
    M[003002] <- 003061   NZVC <- 00

g)  001000  005230          INC     @(R0)+
    001002
    M[M[003002]] = M[003060] <- 003021  R0 <- 003004   NZVC <- 00

h)  001000  005250          INC     @-(R0)
    001002
    R0 <- 003000   M[M[003000]] = M[003040] <- 003011  NZVC <- 00

i)  001000  005270  000016  INC     @16(R0)
    001004
    M[M[003020]] = M[003100] <- 003051  NZVC <- 00

l)  001000  005237  003000  INC     @#003000
    001004
    M[003000] <- 003041   NZVC <- 00

m)  001000  005267  001774  INC     003000
    001004
    M[003000] <- 003041   NZVC <- 00

n)  001000  005277  001774  INC     @003000
    001004
    M[M[003000]] = M[003040] <- 003011  NZVC <- 00

o)  001000  105020          CLRB    (R0)+
    001002
    M[003002]b <- 0  => M[003002] = 003000
    R0 <- 3003 NZVC <- 04

p)  001000  105040          CLRB    -(R0)
    001002
    R0 <- 003001   M[003001]b <- 0  => M[003000] = 000040
    NZVC <- 04

q)  001000  105030          CLRB    @(R0)+
    001002
    M[M[003002]]b = M[003060]b <- 0     =>  M[003060] = 003000
    R0 <- 003004   NZVC <- 04

r)  001000  105050          CLRB    @-(R0)
    001002
    R0 <- 003000
    M[M[003000]]b  = M[003040]b <- 0 => M[003040] = 003000
    NZVC <- 04
```

Osservazioni:

Nei casi d), e), i), l), m) e n) l'istruzione e` di fatto lunga due word ma, mentre nei casi d), e) e i) cio` e` dovuto alla modalita` di indirizzamento che prevede un registro indice, negli altri casi e` dovuto all'uso del particolare registro PC nelle diverse ammissibili modalita`.

La posizione del segno + (-) nel caso di autoincremento (decremento), differito o meno, ricorda l'ordine di esecuzione tra l'incremento (decremento) del registro e l'istruzione effettiva, cioe` l'incremento (decremento) precede (segue) l'istruzione. Inoltre, se l'indirizzamento di questo tipo e` differito, l'incremento (decremento) riguarda il registro e non gia` il puntatore in memoria.

Nel caso i) si noti che il riferimento indiretto avviene DOPO la somma dell'offset con il contenuto del registro (post-indexing).

Gli effetti dei casi l) ed m) sono gli stessi: infatti l'unica differenza tra le due istruzioni e` la quantita` che segue il codice operativo, in un caso e` il valore espresso con una costante o con un simbolo, nell'altro e` l'effetto di un calcolo che coinvolge tale valore e la posizione dell'istruzione. La seconda istruzione produce codice indipendente dalla posizione che esso occupa in memoria, la prima no.

Quando un'istruzione tratta un byte anziche` un word (M[nnnnnn]b significa locazione di un byte di indirizzo nnnnnn), con l'indirizzamento ad autoincremento (decremento) diretto, come nel caso o) (p)), il puntatore viene incrementato (decrementato) di 1 e non di due, in modo da puntare al byte successivo (precedente). Cio` non si verifica nei casi q) ed r) poiche` il differimento fa si` che il puntatore punti ad un puntatore e quest'ultimo ha lunghezza 2 byte.
++++++++

Esercizio 2.3

Sapendo che A=2000 e` (l'indirizzo di) una locazione di memoria e che l'istruzione:

```
MOV     OPND1, OPND2
```

di opcode = 01 trasferisce il contenuto di OPND1 (sorgente) in OPND2 (destinazione), scrivere un segmento di programma che incrementa la locazione A e carica l'indirizzo di A nella locazione ad essa successiva. Il codice deve essere caricato alla locazione 1000.

Si propongono varie soluzioni per evidenziare l'uso delle varie modalita` di indirizzamento.

```
    Ind.    Word1   Word2   Word3   Simbolo
-------------------------------------------------------------
a)  001000  005267  000774          INC     A ; <=> INC 2000
    001004  012767  002000  000770  MOV     #A, A+2; <=>
    001012                          ; MOV #2000, 2002
```

Indirizzamenti: relativo, immediato, relativo.

Osservazioni: poiche` nella seconda istruzione si utilizzano due indirizzamenti riferiti al PC, l'istruzione e` lunga 3 word e compare prima la costante riferita all'operando sorgente e successivamente quella riferita all'operando destinazione. Pertanto, il valore del PC utilizzato nel calcolo del secondo indirizzo effettivo non e` Ind.Istr.+4 ma Ind.Istr+6. Infatti X=(A+2)-Ind.Istr.-6=2002-1004-6=770 .

```
b)  001000  005237  002000          INC     @#A ; <=> INC @#2000
    001004  012737  002000  002002  MOV     #A, @#A+2; <=>
    001012                          ; MOV #2000, @#2002
```

Indirizzamenti: assoluto, immediato, assoluto.

```
c)  001000  012700  002000          MOV     #A, R0
    001004  005220                  INC     (R0)+
    001006  012710  002000          MOV     #A, (R0)
    001012
```

Indirizzamenti: immediato, diretto, autoincremento ind., immediato, indiretto.

```
d)  001000  012700  002000          MOV     #A, R0
    001004  005260  000000          INC     0(R0)
    001010  010060  000002          MOV     R0, 2(R0)
    001014
```

Indirizzamenti: immediato, diretto, indice, diretto, indice.

```
e)  001000  013700  002000          MOV     @#A, R0
    001004  012767  002002  000766  MOV     #A+2, A
    001012  012777  002000  000760  MOV     #A, @A
                                ; <=> MOV #2000, @2000
    001020  005200                  INC     R0
    001022  010067  000752          MOV     R0, A
```

Indirizzamenti: assoluto, diretto, immediato, relativo, immediato, relativo differito, diretto, diretto, relativo.

Osservazioni: in questa (piu` lunga) soluzione, si vede come, una volta salvato il contenuto di A, questa locazione possa essere utilizzata come puntatore per poi essere ricaricata con il valore richiesto.

2.3 - Istruzioni a singolo operando

A questa classe appartengono le istruzioni del tipo:

OPCODE DD

dove DD rappresenta l'unico operando che funge in generale da sorgente e destinazione. Il formato dell'istruzione e` quello di fig. 2.3a. In particolare il bit 15 specifica se l'operazione e` di trattamento di byte (1) o di word (0) per quelle che prevedono le

due forme. Gli Opcode delle istruzioni di questa classe sono elencati in tabella 2.3 con una breve descrizione. In fig. 2.5 sono rappresentate le operazioni si scorrimento a word e a byte. Esse coinvolgono anche il bit di condizione C.

Opcode	Simbolo	Commento
Generali		
B050DD	CLR(B)	Azzera l'operando
B051DD	COM(B)	Sostituisce il contenuto dell'operando con il suo complemento a 1
B052DD	INC(B)	Incrementa di una unita` l'operando
B053DD	DEC(B)	Decrementa di una unita` l'operando
B054DD	NEG(B)	Sostituisce il contenuto dell'operando con il suo complemento a 2
B057DD	TST(B)	Valuta il contenuto dell'operando e imposta conseguentemente i bit di condizione Z e N
Rotazioni e scorrimenti		
B060DD	ROR(B)	Rotazione destra dell'operando
B061DD	ROL(B)	Rotazione sinistra dell'operando
B062DD	ASR(B)	Scorrimento aritmetico a destra
B063DD	ASL(B)	Scorrimento aritmetico a sinistra
0003DD	SWAB	Scambio dei byte nell'operando
Precisione multipla		
B055DD	ADC(B)	Somma il valore del flag C all'operando
B056DD	SBC(B)	Sottrae il valore del flag C all'operando
0067DD	SXT	Estensione del segno (non presente in tutti i modelli)
Speciali		
1067DD	MFPS	Copia da PSW in operando
1064DD	MTPS	Copia da operando a PSW

Tab. 2.3

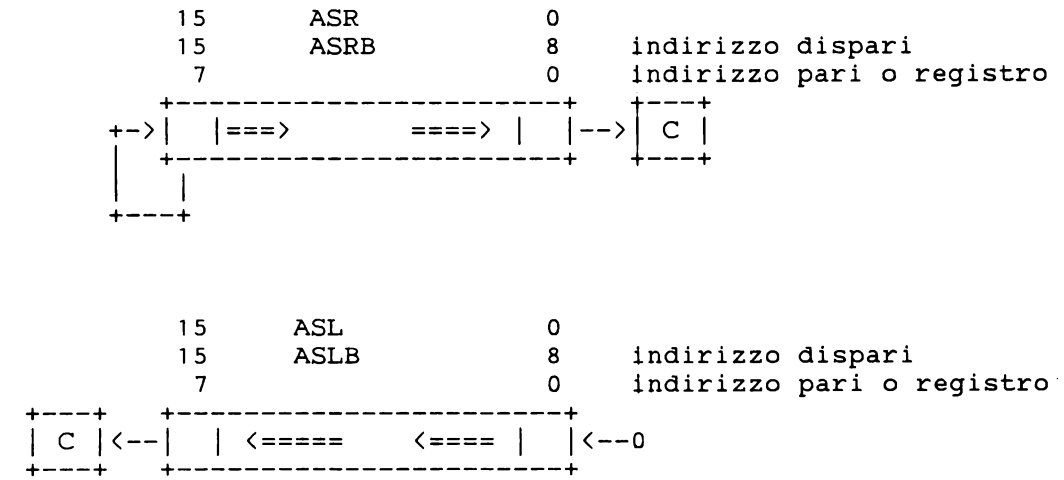


Fig. 2.5

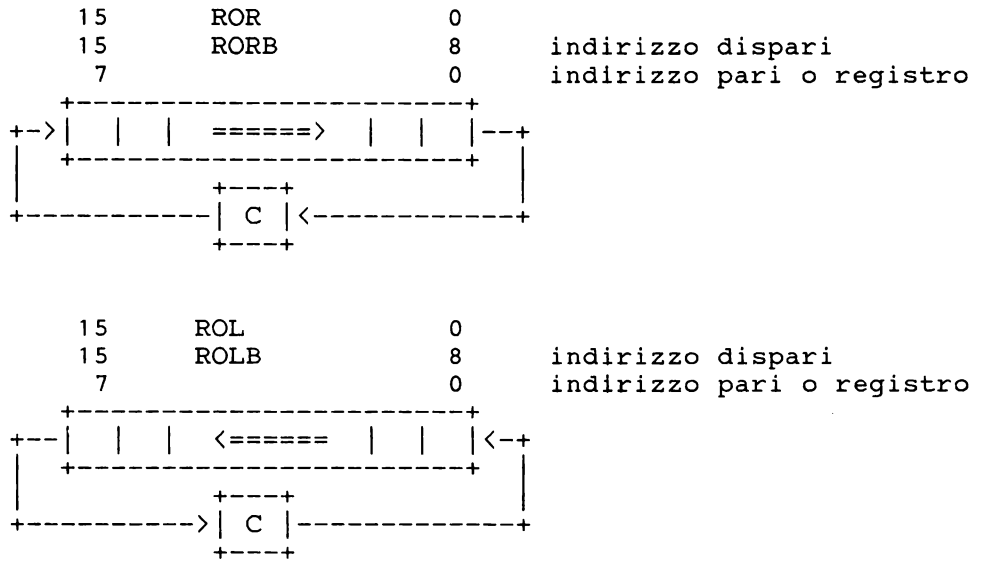


Fig. 2.5 (cont.)

Come si vede molte istruzioni prevedono la versione che si applica su byte. La modifica dei flag avviene concordemente al risultato dell'operazione (word o byte nei due casi).

Esercizio 2.4

Tradurre e verificare gli effetti delle seguenti istruzioni:

```
a)  CLR     R0          R0 = 012345
b)  COMB    R0          R0 = 177777
c)  INCB    R0          R0 = 123777   C = 0
d)  DEC     R0          R0 = 100000   C = 1
e)  NEG     R0          R0 = 177777
f)  TST     R0          R0 = 123456
g)  ROR     R0          R0 = 123456   C = 1
h)  ROLB    R0          R0 = 123456   C = 1
i)  ASR     R0          R0 = 177773
l)  ASLB    R0          R0 = 123567
m)  SWAB    R0          R0 = 000321
n)  ADCB    R0          R0 = 177777   C = 1
o)  SBC     R0          R0 = 0        C = 1
--------

a)  050000  R0 <- 0             NZVC <- 04

b)  105100  R0 <- 177400        NZVC <- 05
Il byte risultato e` nullo.

c)  105200  R0 <- 0123400       NZVC <- 04
Il  byte risultato e` nullo e non vi e` overflow poiche`:
inc(-1) = 0.

d)  005300  R0 <- 077777        NZVC <- 03
Vi e` overflow poiche` 1000000 = -32768.  e` il piu` piccolo  numero
rappresentabile con 16 bit in complemento a 2.
```

e) 005400 R0 <- 1 NZVC <- 01
Il risultato e` positivo poiche` neg(-1)=1. Questa istruzione pone C=0 se il risultato e` nullo, C=1 altrimenti.

f) 005700 R0 = 123456 NZVC <- 10
L'istruzione TST non modifica l'operando ma solo i flag N e Z concordemente al suo contenuto e pone V = C = 0.

g) 006000 R0 <- 151627 NZVC <- 12
L'istruzione ROR coinvolge nella rotazione il flag C e pertanto un totale di 17 bit. Il flag V viene posto pari a C#N.

h) 106100 R0 <- 123535 NZVC <- 00
Infatti: 123456 = 247 || 056 (giustapposizione)
rolb(056, 1) = (135, 0) 247 || 135 = 123535 V = C # N = 0

i) 006200 R0 <- 177775 NZVC <- 11 V = C # N = 0
Si noti che: asr(177773) = asr(-5) <> int ((-5)/2) = -2 = 177776
verificando effettivamente che la meta` di un numero negativo dispari e` ottenibile da asr mediante incremento unitario.

l) 106300 R0 <- 123756 NZVC <- 12
Infatti: 123567 = 247 || 167
aslb(167) = aslb (+119.) = (356, 0) = -18. (su 8 bit)
247 || 356 = 123756
verificando la presenza di overflow denotata da V=C#N=1. Infatti il numero +119.*2 non e` rappresentabile in complemento a 2 su 8 bit.

m) 000300 R0 <- 150400 NZVC <- 04
Infatti: 000321 = 0 || 321 321 || 0 = 150400
N = bit 7 del risultato = 0
Z = 1 se il byte meno significativo del risultato e` 0.

n) 105500 R0 <- 177400 NZVC <- 05
Infatti: 177777 = 377 || 377 adcb(377, 1) = (0, 1)
Tutto va come se si eseguisse una somma tra due quantita` a 8. bit di cui la seconda e` pari a 0 se C=0 e 1 se C=1. I flag sono modificati di conseguenza.

o) 005600 R0 <- 177777 NZVC <- 11
Infatti: sbc(0, 1) = sub(0, 1) = -1 = 177777
++++++++

Esercizio 2.5
Considerando R0 come puntatore ad una locazione a 32 bit (due word successivi), si scrivano segmenti di programma che realizzino una forma equivalente delle istruzioni COM, NEG, ROR, ROL, ASR, ASL su 32 bit.

```
COM     COM     (R0)
        COM     2(R0)
```

Infatti (ah=high word; al=low word):

$c_1(2, 32., a) = 2^{32}-1-a =$
$= 2^{16}*2^{16} + (2^{16} - 2^{16}) - 1 - ah*2^{16} - al =$
$= (2^{16}-1-ah)*2^{16} + (2^{16}-1-al) =$
$= c_1(2, 16., ah)*2^{16} + c_1(2, 16., al)$

Inoltre il segno di ah e` il segno di a, quindi il valore di N dopo le due istruzioni e` corretto, mentre non lo e` per ragioni analoghe e opposte il valore di Z. I valori di C e V sono corretti.

```
NEG     COM     2(R0)
        NEG     (R0)
        ROL     R1
        COM     R1
        ROR     R1   ; carry complementato
        ADC     2(R0)
```

Infatti:

$$c(2, 32., a) = \mathrm{mod}(c_1(2, 32., a) + 1, 2^{32}) =$$
$$= \mathrm{mod}((c_1(2, 16., ah) + \mathrm{int}((c_1(2, 16., al+1)/2^{16})*2^{16}, 2^{32}) +$$
$$+ \mathrm{mod} (c_1(2, 16., al)+1, 2^{16}) =$$
$$= c_1(2, 16., ah)*2^{16} + c(2, 16., al) \quad \text{se al<>0}$$
$$= \mathrm{mod}(c(2, 16., ah)*2^{16}, 2^{32}) \quad \text{se al=0}$$

L'istruzione NEG imposta i flag nel modo seguente:

```
N = 1   se risultato < 0
Z = 1   se risultato = 0
V = 1   se risultato = -2'15
C = 1   se risultato <> 0
```

Essendo neg(0)=0, l'incremento sul word piu` significativo va eseguito se C=0. Per poter sfruttare l'istruzione ADC e` quindi necessario complementare il carry: un modo e` dato dalla sequenza di tre istruzioni che utilizzano R1. Inoltre, poiche` ADC modifica i flag nel modo seguente:

```
N = 1   se risultato < 0
Z = 1   se risultato = 0
V = 1   se valore prec. = 2'15-1 e prec. C = 1
C = 1   se valore prec. = -1 e prec. C = 1
```

Quindi per confronto, la sequenza di istruzioni imposta correttamente N (ultimo valore trattato ah), V (risulta 1 se com(ah)=$2^{15}-1$ e se neg(al) produce C=0 cioe` ah=$-2^{15}$, al=0 => a = $-2^{31}$ che e` l'analoga per 32 bit della condizione di neg su 16 bit) ma non Z (solo ah e non al) e non C (risulta 1 se com(ah)=-1 e neg(al)=0 cioe` ah = al = a = 0: pertanto risulta complementato rispetto all'analogo del neg).

Si suggerisce al lettore di verificare le stesse condizioni per questo segmento:

```
NEG     COM     (R0)
        COM     2(R0)
        ADD     #1, (R0)
        ADC     2(R0)
----

ROR     ROR     2(R0)
        ROR     (R0)
```

In questo caso solo C risultante e` corretto su 32 bit (Z e N solo su al, V=N#C)

```
ROL     ROL     (R0)
        ROL     2(R0)
```

In questo caso sono corretti C, N e V risultanti su 32 bit (Z e` solo su ah).

```
ASR     ASR     2(R0)
        ROR     (R0)
```

Analogamente a ROR, solo C risultante e` corretto su 32 bit.

```
ASL     ASL     (R0)
        ROL     2(R0)
```

Analogamente a ROL, C, N e V risultanti sono corretti su 32 bit.
++++++++

2.4 - Istruzioni a doppio operando

A questa classe appartengono le istruzioni del tipo:

OPCODE SS, DD

dove SS e DD rappresentano gli operandi Sorgente e Destinazione. Il formato dell'istruzione e` quello di fig. 2.3 b.

Opcode	Simbolo	Commento
Generali		
B1SSDD	MOV(B)	Trasferisce il contenuto di SS in DD
B2SSDD	CMP(B)	Compara contenuto di SS con DD e imposta i flag
06SSDD	ADD	Somma SS a DD (risultato in DD)
16SSDD	SUB	Sottrae SS a DD (risultato in DD)
Logiche		
B3SSDD	BIT(B)	Bit Test (AND) impostando i flag
B4SSDD	BIC(B)	Bit azzeramento DD <- ~SS & DD
B5SSDD	BIS(B)	Bit Set (OR) DD <- SS | DD
Registro		
074RDD	XOR	Or esclusivo DD <- Rn # DD
070RDD	MUL	Moltiplicazione Rn <- Rn * SS (non presente in tutti i modelli)
071RDD	DIV	Divisione Rn <- Rn / SS (non presente in tutti i modelli)

Tab. 2.4

Anche in questo caso molte istruzioni prevedono la versione che si applica su byte. Se l'operando SS e` distinto dall'operando DD, SS non viene modificato.

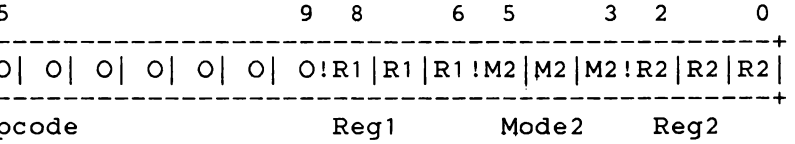
L'istruzione MOVB con destinazione un registro produce non solo il trasferimento del byte in quello meno significativo del registro ma anche l'estensione del segno, cioe` il byte piu` significativo del registro viene posto a 0 o a -1 a seconda che il byte trasferito sia positivo o negativo.

L'istruzione ADD esegue la somma in complemento a 2 su 16 bit, ponendo i flag in base al risultato, secondo il significato suesposto. Le istruzioni SUB e CMP sono rispettivamente equivalenti alle istruzioni add(neg(SS), DD) e add(neg(DD), SS) dal punto di vista del risultato e dei flag Z, N, ma non esattamente per i flag C e V.

```
SUB     C = 1 se DD+com(SS)+1 < 2^16
        V = 1 se overflow nella differenza, cioe` SS
            e DD discordi, risultato e SS concordi
CMP     C = 1 se SS+com(DD)+1 < 2^16
        V = 1 se overflow nella differenza, cioe` SS
            e DD discordi, risultato e DD concordi
```

L'istruzione CMP non modifica DD ma solo i flag.

L'istruzione XOR e le altre istruzioni della stessa classe prevedono che un operando sia un necessariamente registro indirizzato in modo diretto, per cui il formato e` quello di fig. 2.6.



Istruzione registro, operando
Fig. 2.6

Esercizio 2.6

Tradurre e verificare gli effetti delle seguenti istruzioni:

```
a)  MOV     #15, R0          R0 = 012345   C = 0
b)  MOVB    #200, R0         R0 = 012345   C = 1
c)  MOV     R0, R1           R0 = 0        R1 = 123456
                             C = 0
d)  CMP     #3,  #5
e)  CMP     #3,  #-5
f)  ADD     #-2, R0          R0 = 2
g)  SUB     #-2, R0          R0 = 2
h)  BIT     #40, #436        C = 0
i)  BIT     #3,  R0          R0 = 107      C = 1
l)  BIC     #11, R0          R0 = 345674   C = 1
m)  BIS     #11, R0          R0 = 0        C = 0
n)  XOR     R0,  @#4000      R0 = 012345   M[4000] = 123456  C = 1
--------
```

```
a)  012700  000015      R0 <- 15            NZVC <- 00
MOV modifica N e Z in base all'informazione trasferita e pone V=0.

b)  112700  000200      R0 <- 177600        NZVC <- 11
Si noti l'estensione del segno.

c)  010001              R1 <- 0             NZVC <- 04

d)  022727  000003  000005                  NZVC <- 11
Infatti:        cmp = SS+com(DD)+1 = 3+177772+1 = 177776
177776 < 2^16; SS e DD concordi.
```

e) 022727 000003 177773 NZVC <- 01
Infatti: cmp = SS+com(DD)+1 = 3+4+1 = 8.
8. < 2^{16}; SS e DD discordi; DD e risultato discordi.

f) 062700 177776 R0 <- 0 NZVC <- 05

g) 162700 177776 R0 <- 4 NZVC <- 01
Infatti: sub = DD+com(SS)+1 = 2+1+1 = 4
4 < 2^{16}; SS e DD discordi; SS e risultato discordi.

h) 032727 000040 000436 NZVC <- 04
Infatti le istruzioni logiche pongono V=0, lasciano inalterato C e pongono i flag Z e N in base al risultato. Nell'esempio la maschera 40 (SS) cattura il bit 5 che in 436 e` 0, ponendo Z=1 e N=0.

i) 032700 000003 NZVC <- 01
La maschera 3 cattura i bit 0 e 1 in 107 che sono entrambi a 1.

l) 042700 000011 R0 <- 145664 NZVC <- 11
La maschera 11 pone a zero i bit 0 e 3 nell'operando, lasciando inalterati gli altri.

m) 052700 000011 R0 <- 000011 NZVC <- 00
Analogo a l) ma ponendo i bit a 1.

n) 074037 004000 M[4000] <- 131713 NZVC <- 11
++++++

Esercizio 2.7

Esprimere i campi di variabilita` per i flag NZVC nelle operazioni SUB e CMP.

Nella seguente risoluzione, onde distinguere un valore con segno dalla sua rappresentazione in complemento a 2, si indicheranno SS, DD i valori e [SS], [DD] le rappresentazioni. Per esse vale:

$NN \geq 0 \Rightarrow 0 \leq [NN] = NN < 2^{15}$
$-2^{15} < NN < 0 \Rightarrow 2^{15} \leq [NN] < 2^{16}$ $[NN] = 2^{16} + NN$ $NN = [NN] - 2^{16}$

SUB SS, DD (DD - SS)
---

se DD>=0
N=1 se SS>DD>=0 oppure DD-SS>=2^{15} e SS<0 <=>
<=> 0<=[DD]<[SS]<2^{15} oppure 2^{15}<=[SS]<=[DD]+2^{15}

N=0 se DD>=SS>=0 oppure DD-SS<2^{15} e SS<0 <=>
<=> 0<=[SS]<=[DD]<2^{15} oppure [SS]>[DD]+2^{15}

se DD<0
N=1 se SS>=0 e DD-SS>=-2^{15} oppure DD<SS<0 <=>
<=> 0<=[SS]<=[DD]-2^{15} oppure 2^{15}<=[DD]<[SS]

N=0 se SS>=0 e DD-SS<-2^{15} oppure SS<=DD<0 <=>
<=> 0<=[DD]-2^{15}<[SS]<2^{15} oppure 2^{15}<=[SS]<=[DD]

Z=1 se SS=DD

Z=0 se SS<>DD

se DD>=0
V=1 se -2^{15}<=SS<=DD-2^{15}<0
<=> 2^{15}<=[SS]<=2^{15}+[DD]
V=0 se SS>=0 oppure DD-2^{15}<SS<0 <=>
<=> [SS]<2^{15} oppure 2^{15}<=2^{15}+[DD]<[SS]<2^{16}

se DD<0
V=1 se SS>2^{15}+DD>=0 <=>
<=> se 2^{15}>[SS]>[DD]-2^{15}>=0

V=0 se 0<=SS<=2^{15}+DD oppure SS<0 <=>
<=> se 0<=[SS]<=[DD]-2^{15} oppure [SS]>=2^{15}

C=1 se [SS]>[DD] <=>
<=> se DD>=0: SS>DD oppure SS<0
<=> se DD<0: 0>SS>DD

C=0 se [SS]<=[DD] <=>
<=> se DD>=0: 0<=SS<=DD
<=> se DD<0: SS>=0 oppure -2^{15}<=SS<=DD<0

NZVC	Condizione
DD>=0	
0000	0<=[SS]<[DD]<2^{15}
0100	0<=[SS]=[DD]<2^{15}
1001	0<=[DD]<[SS]<2^{15}
1011	[DD]<2^{15}<=[SS]<=[DD]+2^{15}
0001	[DD]<2^{15}<=[DD]+2^{15}<[SS]
DD<0	
1000	0<=[SS]<=[DD]-2^{15}<2^{15}<=[DD]
0010	0<=[DD]-2^{15}<[SS]<2^{15}<=[DD]
0000	2^{15}<=[SS]<[DD]
0100	2^{15}<=[SS]=[DD]
1001	2^{15}<=[DD]<[SS]

CMP SS, DD (SS - DD)

se DD>=0
N=1 se DD>SS>=0 oppure SS<0 e SS-DD>=-2^{15}
<=> 0<=[SS]<[DD]<2^{15} oppure [SS]>=[DD]+2^{15}>=2^{15}

N=0 se SS>=DD>=0 oppure SS<0 e SS-DD<-2^{15}
<=> 0<=[DD]<=[SS]<2^{15} oppure 2^{15}<=[SS]<[DD]+2^{15}

se DD<0
N=1 se SS>=0 e SS-DD>=2^{15} oppure SS<DD<0
<=> 2^{15}>[SS]>=[DD]-2^{15}>=0 oppure 2^{15}<=[SS]<[DD]

N=0 se SS>=0 e SS-DD<2^{15} oppure 0>SS>=DD
<=> 0<=[SS]<[DD]-2^{15}<2^{15} oppure [SS]>=[DD]>=2^{15}

Z=1 se SS=DD

Z=0 se SS<>DD

C=1 se [DD]>[SS] <=>
<=> se DD>=0: 0<=SS<DD<2^{15}
<=> se DD<0: SS<DD<0 oppure 2^{15}>SS>=0
C=0 se [DD]<=[SS] <=>
<=> se DD>=0: SS<0 oppure 0<=DD<=SS<2^{15}
<=> se DD<0: DD<=SS<0

se DD>=0
V=1 se -2^{15}<=SS<DD-2^{15}<0 <=>
<=> 2^{15}<=[SS]<[DD]+2^{15}

V=0 se SS>=0 oppure DD-2^{15}<=SS<0 <=>
<=> [SS]<2^{15} oppure 2^{15}<=2^{15}+[DD]<=[SS]<2^{16}

se DD<0
V=1 se 2^{15}>SS>=2^{15}+DD>=0 <=>
<=> 2^{15}>[SS]>=[DD]-2^{15}>=0

V=0 se 0<=SS<=DD+2^{15}<2^{15} oppure SS<0 <=>
<=> 0<=[SS]<[DD]-2^{15}<2^{15} oppure [SS]>=2^{15}

NZVC	Condizione
DD>=0	
a) 1001	0<=[SS]<[DD]<2^{15}
b) 0100	0<=[SS]=[DD]<2^{15}
c) 0000	0<=[DD]<[SS]<2^{15}
d) 0010	[DD]<2^{15}<=[SS]<[DD]+2^{15}
e) 1000	[DD]<2^{15}<=[DD]+2^{15}<=[SS]
DD<0	
f) 0001	0<=[SS]<[DD]-2^{15}<2^{15}<=[DD]
g) 1011	0<=[DD]-2^{15}<=[SS]<2^{15}<=[DD]
h) 1001	2^{15}<=[SS]<[DD]
i) 0100	2^{15}<=[SS]=[DD]
j) 0000	2^{15}<=[DD]<[SS]

++++++++

Esercizio 2.8

Determinare per quali valori di R0 si verifica overflow nell'istruzione:

SUB R1, R0

con i seguenti valori di R1:

a) 0 b) 100000
c) 077777 d) 1
e) 100400

Applicando le conclusioni dell'esercizio 2.7 si ha che:

a) [SS]=0 ==> 2^{15}<=[DD] [SS]>[DD]-2^{15} ==> mai

b) [SS]=100000 ==> [DD]<2^{15} [SS]<=[DD]+2^{15} ==>
0<=R0<2^{15} (tutti i positivi)

c) [SS]=077777 ==> 2^{15}<[DD] [SS]>[DD]-2^{15} ==>
2^{15}<=R0<2^{16}-1 (tutti i negativi escluso -1)

d) [SS]=1 ==> 2^{15}<[DD] [SS]>[DD]-2^{15} ==>
R0=2^{15}

e) [SS]=100400 ==> [DD]<2^{15} [SS]<=[DD]+2^{15} ==>
000400<=R0<2^{15}
++++++++

Esercizio 2.9

Considerando R0 come puntatore ad una locazione a 32 bit (due word successivi), si scrivano segmenti di programma che realizzino una forma equivalente delle istruzioni ADD e SUB su 32 bit.

Analogamente a quanto fatto nell'esercizio 2.5, si ha:

```
ADD     ADD     0(R0), 0(R1)    ; somma word meno sign.
        ADC     2(R1)           ; somma eventuale riporto
        ADD     2(R0), 2(R1)    ; somma word piu` sign.

SUB     NEG     2(R0)           ; compl. a 2 word piu` sign.
        NEG     0(R0)           ; compl. a 2 word meno sign.
        SBC     2(R0)           ; C=1 se neg<>0
        ADD     0(R0), 0(R1)
        ADC     2(R1)
        ADD     2(R0), 2(R1)
```

oppure

```
SUB     SUB     0(R0), 0(R1)    ; sottrae word meno sign.
        SBC     2(R1)           ; sottrae eventuale riporto
        SUB     2(R0), 2(R1)    ; sottrae word piu` sign.
```

La prima soluzione per SUB tiene conto che:

c(2, 32., x) = c(2, 16., xh)*2^{16} se xl=0
= c_1(2, 16., xh)*2^{16}+c(2, 16., xl) se xl<>0

Dopo il calcolo completo, l'unico flag valido su 32 bit e` N. Z e` riferito solo al word piu` significativo. Se dopo la prima somma (differenza) non vi e` carry, allora sono corretti anche C e V. Si suggerisce di verificare gli algoritmi di sottrazione con le seguenti coppie:

SS=2 DD=1; SS=2;DD=2^{16}+1; SS=1;DD=2; SS=2^{16};DD=2
++++++

2.5 - Istruzioni di salto

A questa classe appartengono le istruzioni che consentono di modificare il flusso sequenziale di esecuzione, caricando un valore nel registro PC senza modificare i flag di PSW. Le istruzioni di salto possono essere suddivise in istruzioni a salto esteso (Jump) e istruzioni a salto limitato o relativo (Branch). Queste ultime si dividono poi in condizionate e non condizionate.

Opcode	Simbolo	Commento
Salti estesi		
0001DD	JMP	Trasferisce il controllo all'indirizzo specificato dall'operando
004RDD	JSR	Salva il contenuto di un registro e nel registro il contenuto di PC (ind. di ritorno) e trasferisce il controllo all'indirizzo specificato dall'operando
00020R	RTS	Restituisce il controllo ricaricando il PC dallo stack.
Salto limitato (relativo) incondizionato		
0004XXX	BR	Trasferisce il controllo all'indirizzo specificato
Salti limitati (relativi) condizionati		Trasferiscono il controllo all'indirizzo specificato se e` verificata la condizione associata.
Su singola condizione		
0010XXX	BNE	Z = 0
0014XXX	BEQ	Z = 1
1000XXX	BPL	N = 0
1004XXX	BMI	N = 1
1020XXX	BVC	V = 0
1024XXX	BVS	V = 1
1030XXX	BCC	C = 0
1034XXX	BCS	C = 1
Su condizione per numeri con segno		
0020XXX	BGE	N # V = 0
0024XXX	BLT	N # V = 1
0030XXX	BGT	Z | (N # V) = 0
0034XXX	BLE	Z | (N # V) = 1
Su condizione per numeri senza segno		
1010XXX	BHI	C | Z = 0
1014XXX	BLOS	C | Z = 1
1030XXX	BHIS	C = 0
1034XXX	BLO	C = 1
Su conteggio		
077RNN	SOB	Decrementa il registro specificato e se risultato <> 0 effettua il salto.

Tab. 2.5

Il formato dell'istruzione JMP e` quello delle istruzioni ad un operando (fig. 2.3a) e pertanto il valore da caricare in PC puo` essere specificato mediante un qualsiasi indirizzamento esclusi quello di registro diretto e immediato. Ad esempio:

JMP	(R0)	salto indiretto	PC <- R0
JMP	X(R0)	salto indiretto con indice	PC <- (R0)+X
JMP	@#A	salto assoluto	PC <- A
JMP	A	salto relativo	PC <- A
JMP	(R0)+	salto indiretto con inc.	PC <- R0, R0 <- R0+2
JMP	@(R0)+	salto differito con inc.	PC <- M[R0], R0 <- R0+2

Si noti che un Jump relativo o assoluto puo` essere specificato con l'uso di un indirizzo simbolico (Label) che puo` essere dichiarato nel modo seguente:

```
    ....
A:  Istruzione
    .....
```

Al simbolo A viene associato l'indirizzo in cui verra` caricata l'istruzione immediatamente successiva al carattere ':'.

Il formato dell'istruzione JSR e` quello di fig. 2.7a e corrisponde all'istruzione simbolica:

```
    JSR     Reg, ADDR
```

dove Reg e` un registro e ADDR e` l'indirizzo a cui effettuare il salto (come per JMP). Quello che avviene a fronte dell'esecuzione di questa istruzione e` riassunto qui sotto:

```
Temp <- ADDR
Push (Reg)              { Push (x) :: SP <- SP-2, M[SP] <- x)
Reg <- PC
PC <- Temp
```

Temp e` un registro interno del processore in cui viene temporaneamente memorizzato l'indirizzo che verra` inserito nel PC dopo aver completato le altre operazioni: il fatto acquista significato se si pensa che vi e` la possibilita` che quelle operazioni modifichino gli elementi in funzione dei quali viene calcolato ADDR.

L'istruzione di JSR ha lo scopo di realizzare i concetti di *Subroutine* e *Coroutine*. Una subroutine e` un segmento di programma che puo` essere attivato da piu` punti di altri programmi, a cui restituisce il controllo mediante una istruzione apposita di Return. Una coroutine e` un segmento di programma che puo` chiamare (simmetricamente) un altra coroutine in piu` punti: in ciascuno di questi avviene il salvataggio del punto di ritorno, in cui la coroutine viene riattivata a fronte di un'operazione analoga svolta dalla coroutine gemella.

Per le subroutine, la chiamata avviene con una istruzione del tipo:

```
    JSR     Reg, SUBR
```

che salva il contenuto di Reg nello stack, l'indirizzo di ritorno in Reg e salta all'indirizzo specificato. Alla fine della subroutine deve trovarsi l'istruzione(fig. 2.7d):

```
    RTS     Reg
```

la cui esecuzione comporta il recupero dell'indirizzo di ritorno e successivamente del contenuto del registro specificato:

```
PC <- Reg
Reg <- Pop()        { Pop() :: SP <- SP+2, M[SP-2])
```

Ovviamente il registro specificato con RTS deve essere lo stesso della chiamata JSR. Un caso particolare e` Reg=PC in cui l'indirizzo

di ritorno viene direttamente salvato nello stack e da qui recuperato. Infatti:

```
    JSR     PC, ADDR

Temp <- ADDR
Push (PC)
PC <- PC
PC <- Temp


    RTS     PC

PC <- PC
PC <- Pop ()
```

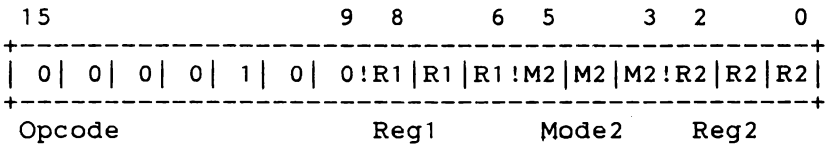
Per la coroutine, la attivazione simmetrica (Resume o Transfer) avviene mediante l'istruzione:

```
    JSR     PC, @(SP)+
```

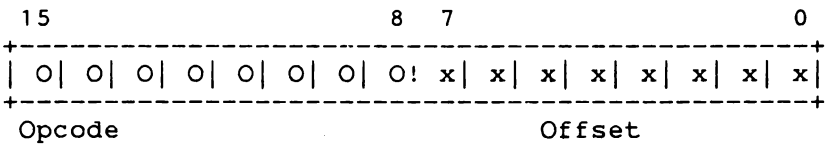
Infatti, si ha:

```
Temp <- Pop ()
Push (PC)
PC <- PC
PC <- Temp
```

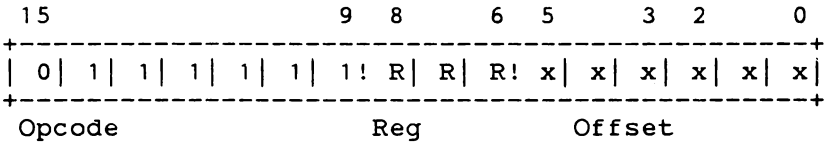
cioe` si ha uno scambio tra il contenuto di PC e quello del top dello stack.



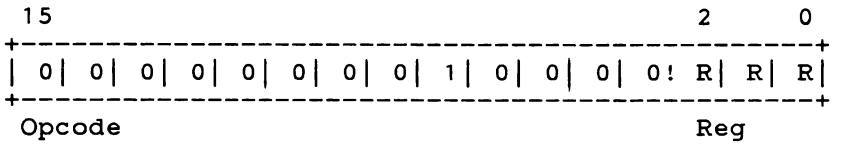
Istruzione JSR
Fig. 2.7 a



Istruzione di salto limitato (branch)
Fig. 2.7 b



Istruzione SOB
Fig. 2.7 c



Istruzione RTS
Fig. 2.7 d

Le istruzioni di salto limitato hanno la particolarita` (fig. 2.7b) di avere la specificazione dell'indirizzo di salto all'interno del codice operativo. Infatti, al cosiddetto *Base Code* che specifica la particolare istruzione, viene sommato un byte di *Offset* (spostamento) in base al quale si calcola il valore del nuovo PC:

PC <- PC corrente + 2 * Offset

La quantita` Offset e` interpretata come un numero con segno a 8 bit in complemento a 2. Ad esempio, sapendo che il base code dell'istruzione BR e` 000400, l'istruzione:

BR .

(il punto identifica l'indirizzo in cui viene caricata l'istruzione stessa) viene tradotta 000777. Infatti, in questo caso, detto PC0 l'indirizzo di caricamento dell'istruzione, si ha:

c(2, 8, -Offset) = [Offset] = 377 => Offset= -1
PC = (PC0+2) + 2 * Offset = PC0

Si noti che il valore corrente di PC al momento del calcolo e` quello che fa seguito al fetch dell'istruzione cioe` PC0+2. Si ricava pertanto che per una siffatta istruzione il salto e` possibile solo nel range:

PC0-376 <= PC0 <= PC0+400 $-2^7 <= Offset <= 2^7-1$
PC0-254.<= PC0 <= PC0+256.

I salti condizionati su singola condizione valutano uno dei flag di PSW. Quelli cosiddetti "su interi con segno" si applicano generalmente dopo operazioni aritmetiche-logiche, interpretando il risultato come un numero con segno. In questo caso il salto avviene se e` verificata la condizione ricordata dal mnemonico associato, rispetto al valore 0. Ad esempio la sequenza:

```
SUB     R0, R1
BGT     GREAT
```

corrisponde al salto al label GREAT se il risultato della differenza e` maggiore di zero. Un discorso analogo e` applicabile ai salti condizionati per numeri senza segno, pur di interpretare come tale il risultato.

L'istruzione SOB ha il formato di fig. 2.7c corrispondente simbolicamente a:

SOB Reg, ADDR

Scopo dell'istruzione e` la rapida realizzazione di iterazioni con-

dizionate da conteggio: il registro specificato funziona da contatore e il salto avviene se, dopo il decremento di questo, il risultato e` <> 0. A causa del particolare scopo, l'offset e` ridotto a 6 bit e interpretato come numero senza segno. Detto ancora PC0 l'indirizzo di caricamento, il nuovo valore di PC viene calcolato secondo:

PC <- PC corrente - 2 * Offset

Ad esempio, sapendo che l'Opcode dell'istruzione SOB e` 077, l'istruzione:

SOB R0, .

viene tradotta 077001. Infatti, in questo caso:

PC = (PC0+2) - 2 * 1 = PC0

Si ricava che per una siffatta istruzione il salto e` possibile solo nel range:

PC0-174 <= PC0 <= PC0+2 $0<=Offset<=2^6-1$
PC0-124.<= PC0 <= PC0+2.

Esercizio 2.10

Verificare che per le istruzioni di salto condizionato relative a numeri con e senza segno vale quanto ricordato dal mnemonico associato.

Si osservi dapprima che dalla tabella si verifica che questi salti sono accoppiati per condizioni complementari nel modo seguente:

BGE-BLT BGT-BLE BHI-BLOS BHIS-BLO

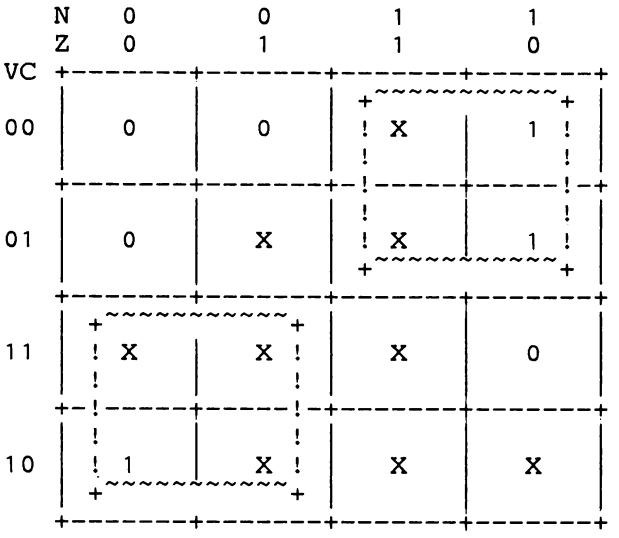
Pertanto, si dimostrera` la validita` richiesta solo per quelli in cui la condizione per la quale il salto si verifica e` quella in cui la funzione associata applicata ai bit di condizione e` pari a 1, cioe` BLT, BLE, BLOS, BLO. Allo scopo, si supponga che l'istruzione di salto segua l'istruzione:

CMP SS, DD

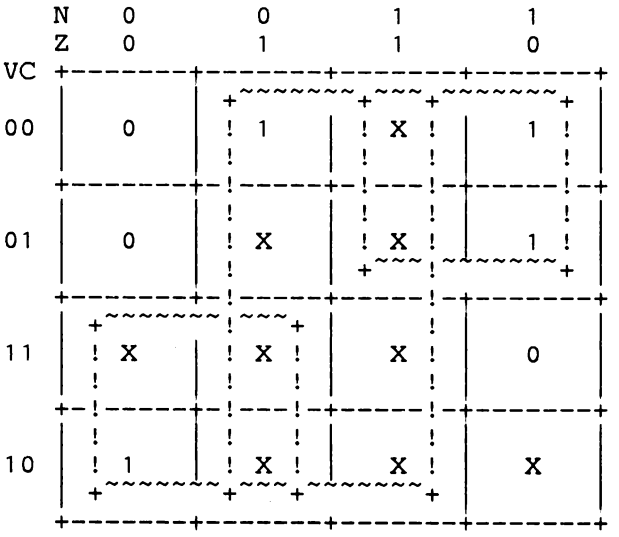
e si verifichi quindi la validita` nel salto del confronto SS,DD.

Con riferimento alle conclusioni dell'esercizio 2.7, nel confronto SS, DD sono rispettivamente vere le seguenti condizioni:

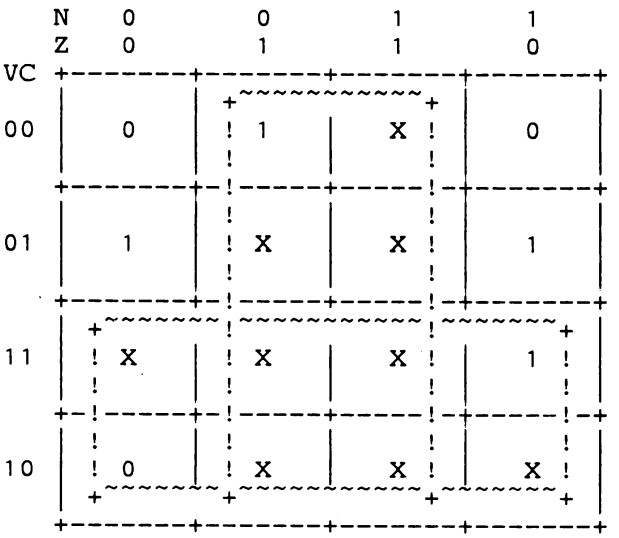
Condizione	Casi
BLT (SS < DD)	a, d, e, h
BLE (SS <= DD)	a, b, d, e, h, i
BLOS ([SS] <= [DD])	a, b, f, g, h, i
BLO ([SS] < [DD])	a, f, g, h



BLT =
= ~NV | N~V =
= N # V



BLE =
= Z | ~NV | N~V =
= Z | (N # V)



BLOS = C | Z

Fig. 2.8

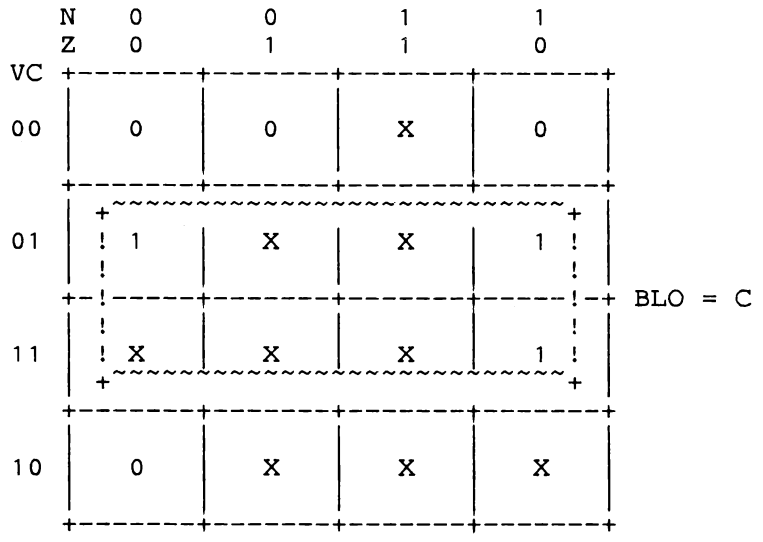


Fig. 2.8 (cont.)

Considerando NZVC come ingressi di una funzione booleana che e` vera se la relativa condizione e` verificata, si possono disegnare le 4 mappe di Karnaugh per le 4 condizioni (fig. 2.8), tenendo conto che alcune configurazioni per gli ingressi sono impossibili (si lascia al lettore la dimostrazione). Per questi ingressi, la funzione booleana puo` assumere qualsiasi valore e quindi possono essere coinvolti o meno a piacimento nella determinazione dei gruppi di ingressi (sono infatti anche detti condizioni di indifferenza e indicati sulla mappa con una X).
++++++++

Esercizio 2.11

Dire se avviene il salto per le seguenti istruzioni di branch, precedute dalle istruzioni e dalle condizioni singolarmente indicate, e giustificarne il motivo.

```
1)  R0 = 100001      XOR R0,R0          BNE A1
2)  R0 = 104400      TSTB R0            BEQ A2
3)  R0 = 100000      NEG R0             BPL A3
4)  R0 = 123456      BIC #400,R0        BMI A4
5)  CMP #134321, #042517                BVC A5
6)  V=1              MOV #1, R0         BVS A6
7)  CLR R0  COM R0   ADC R0             BCC A7
8)  R0 = 024715      SUB #032100,R0     BCS A8
9)  CMP #134321, #042517                BGE A9
10) R0 = 152340      BIS #1,R0          BLT A10
11) R0 = 140000      ASL R0             BGT A11
12) R0 = 100401      SWAB R0            BLE A12
13) R0 = 000001      COM R0
                     BIT #100000,R0     BHI A13
14) R0 = 077777      COM R0
                     ROL R0             BLOS A14
15) R0 = 012345      SUB #054321,R0     BHIS A15
16) R0 = 123456      NEG R0             BLO A16
--------
```

1) NO. XOR Rn, Rn ==> Rn=0 Z=1
2) SI. R01 = 0 ==> Z=1
3) NO. NEG(-2^{15}) = -2^{15} NZVC=13
4) SI. L'istruzione BIC pone a zero nell'operando i bit che corrispondono a bit 1 nella maschera (solo bit 8) ==> R0 = 123056 N=1
5) NO. [SS]>=2^{15}>[DD] ==> C=0
6) NO. L'istruzione MOV pone V=0.
7) NO. CLR R0 ==> R0=0
COM R0 ==> R0=177777 C=1
ADC R0 ==> R0=0 C=1
8) SI. 0<=[DD]<[SS]<2^{15} ==> R0 = 172615 NZVC=11
9) NO. [DD]<2^{15}<=[SS]<[DD]+2^{15} NZVC=02 N#V=1
10) SI. R0=152341 BIS pone a 1 i bit corrispondenti a 1 nella maschera e non modifica C NZVC=100? N#V=1
11) NO. R0=100000 NZVC=11 Z|(N#V)=1
12) SI. R0=000601 Il byte meno significativo del risultato, considerato come numero con segno, e` negativo ==> NZVC=10 Z|(N#V)=1
13) NO. R0=177776 L'istruzione BIT testa in questo caso il bit di segno NZVC=11 C|Z=1
14) SI. COM(077777)=100000 C=1; R0=000003 NZVC=03 C|Z=1
15) NO. 0<=[DD]<[SS]<2^{15} NZVC=11
16) SI. R0=054322 NZVC=01 C=1 poiche` ris<>0
++++++++

Esercizio 2.12

Si voglia inizializzare un vettore a[1..10] di 10 elementi da 16 bit in modo che a[i]=i.

Indirizzo	Istruzione		Simbolo	
001000	012700	000012	MOV	#10., R0
001004	012701	003024	MOV	#A+20., R1
001010	010041		MOV	R0, -(R1)
001012	077002		SOB	R0, .-2

La prima istruzione carica il contatore con l'estensione del vettore mentre la seconda posiziona il puntatore dopo l'ultimo elemento del vettore (lungo 20 byte). L'inizializzazione procede con un ciclo basato sul contatore R0 il cui valore, decrementandosi, viene trasferito all'indietro nelle locazioni del vettore, come richiesto. Il ciclo ha termine quando R0=0. "A" e` il simbolo che sta ad identificare l'indirizzo iniziale del vettore. Gli assemblatori solitamente ammettono una dichiarazione di vettore mediante una pseudo-istruzione del tipo:

```
A:      .BLKW       10.
```

che riserva 10. locazioni di memoria (word) da 16 bit. In modo analogo e` possibile utilizzare un label che sta ad identificare l'indirizzo in cui effettuare il salto per l'istruzione SOB:

```
        MOV     #10., R0
        MOV     #A+20., R1
LOOP:   MOV     R0, -(R1)
        SOB     R0, LOOP
```
++++++++

Esercizio 2.13

Si voglia realizzare un salto scegliendo l'indirizzo di destinazione tra N possibilita` predefinite.

Conviene costruire un vettore (*jump table*) inizializzato con N indirizzi di salto tra cui scegliere quello da effettuare. Supponendo che R0 contenga l'indice di scelta (0..N-1) e JTAB sia l'indirizzo iniziale della tabella, il semplice segmento:

```
        ASL     R0              ; R0*2
        JMP     @JTAB(R0)       ; M[JTAB+R0] e` l'indir. a cui saltare
    ...
JTAB:   .WORD   #ADR1, #ADR2, #ADR3...
```

realizza quanto richiesto. La pseudo-istruzione .WORD permette l'inizializzazione di un vettore in fase di caricamento. Il vettore, posizionato all'indirizzo corrente di caricamento (JTAB), contiene locazioni (word) da 16 bit i cui valori iniziali sono listati a fianco del pseudo-codice operativo .WORD.

Qualora l'indirizzo della tabella sia pure contenuto in un registro, ad es. R1, occorre modificare il programma nel modo seguente:

```
        ASL     R0
        ADD     R0, R1          ; R1 <- JTAB+R0*2
        JMP     @0(R1)          ; salto ad un indirizzo contenuto
                                ; nella tabella
```

In alternativa, e` possibile definire una sequenza di jump di cui uno solo viene attivato sulla base della scelta. Nelle stesse ipotesi di prima:

```
        ASL     R0
        ASL     R0              ; R0*lunghezza istr. jump = R0*4
        JMP     JSEQ(R0)        ; salto all'indirizzo JSEQ+R0
JSEQ:   JMP     ADR1
        JMP     ADR2
        JMP     ADR3
++++++++
```

Esercizio 2.14

Esemplificare come sia possibile, in modi diversi, passare ad una subroutine SUBR un parametro per valore, uno per riferimento e ricevere da essa un valore di ritorno.

Il comunicare informazioni in fase di chiamata di una subroutine e il ricevere valori di ritorno e`, come noto, di grande importanza per una subroutine. I parametri di chiamata possono essere passati in termini di valore (parametri di ingresso) oppure possono essere costituiti da locazioni di memoria, elementari o multiple, il cui riferimento (indirizzo iniziale) viene comunicato alla subroutine dal programma chiamante, nel qual caso la subroutine e` in grado di leggerne e/o modificarne il contenuto (parametri di ingresso e/o uscita).

Quando i parametri sono in numero limitato, il metodo piu` semplice e` di utilizzare i registri. Per l'esempio, la chiamata potrebbe essere cosi` realizzata:

```
        MOV     VAL1, R0
        MOV     #VAR2, R1
        JSR     PC, SUBR
        MOV     R2, RETVAL
```

VAL1 e VAR2 sono due locazioni di memoria; R0 contiene il parametro passato per valore e R1 quello per riferimento. R2 conterra` il valore di ritorno dalla subroutine. Quest'ultima puo` far uso diretto dei valori passati nei registri oppure memorizzarli in "proprie" variabili:

```
SUBR:   MOV     (R1), R4
        ...
        MOV     R0, PARAM1
        ...
        MOV     SUBRET, R2
        RTS     PC
```

Un altro modo e` quello di utilizzare apposite locazioni di memoria per contenere i parametri e/o i valori di ritorno:

```
        MOV     VAL1, PARAM1
        MOV     #VAR2, PARAM2
        JSR     PC, SUBR
        MOV     SUBRET, RETVAL
```

Entrambi i metodi necessitano una convenzione esplicita per ogni subroutine. Un modo piu` generale, indipendente dalla particolare subroutine, e` quello cosiddetto della *In-line Calling Sequence*:

```
        JSR     PC, SUBR
PARAM1: .WORD   0
PARAM2: .WORD   0
RETVAL: .WORD   0
CONTINUE:
        ...
```

Il programma chiamato provvede a caricare nelle locazioni successive alla chiamata i parametri. Se questi vanno passati per riferimento e l'indirizzo e` noto al tempo di caricamento del programma, il parametro e` specificabile a priori.

```
        MOV     VAL1, PARAM1
        JSR     PC, SUBR
PARAM1: .WORD   0
PARAM2: .WORD   #VAR2
RETVAL: .WORD   0
CONTINUE:
        ...
```

Si noti che in questo caso non risulta conveniente e spesso nemmeno possibile il caricamento al tempo di esecuzione del valore del parametro nella calling sequence (si pensi ad esempio a programmi che debbano essere caricati in ROM) per cui questa tecnica risulta applicabile se il parametro ha valore noto al tempo di caricamento, cioe` e` un valore predefinito oppure il parametro e` passato

per riferimento e quest'ultimo e` predefinito.

Con la modalita` in-line calling sequence, la subroutine puo` accedere ai parametri utilizzando il valore di PC salvato nello stack; essa deve anche provvedere affinche` il ritorno alla procedura chiamata avvenga all'istruzione successiva all'ultimo parametro nella C.S.

```
SUBR:   MOV     (SP), R0     ; carica PC salvato
        MOV     (R0)+, PAR1  ; carica parametri
        MOV     (R0)+, PAR2
        ...
        ...
        MOV     SUBRET, (R0)+    ; memorizza valore di ritorno
        MOV     R0, (SP)     ; aggiorna sullo stack l'indirizzo
                             ; di ritorno
        RTS     PC
```

Utilizzando la versione di JSR che specifica un registro diverso da PC, il caricamento dei parametri e` piu` immediato:

```
        MOV     VAL1, PARAM1
        JSR     R5, SUBR
PARAM1: .WORD   0
PARAM2: .WORD   #VAR2
RETVAL: .WORD   0
CONTINUE:
        ...
SUBR:   MOV     (R5)+, PAR1  ; carica parametri
        MOV     (R5), ADR2   ; carica riferimento
        MOV     @(R5)+, VAL2; carica valore (elementare)
        ...
        ...
        MOV     SUBRET, (R5)+    ; memorizza valore di ritorno
        RTS     R5
```

Un altro metodo, egualmente generale, e` quello cosiddetto *Stack Calling Sequence* in cui i parametri vengono passati sullo stack riferito da SP:

```
        MOV     VAL1, -(SP)
        MOV     #VAR2, -(SP)
        JSR     PC, SUBR
        MOV     (SP)+, RETVAL
        ...
        ...

SUBR:   MOV     (SP)+, R0    ; salva PC
        MOV     (SP)+, PAR2  ; carica parametri (ordine
        MOV     (SP)+, PAR1  ; inverso)
        ...
        ...
        MOV     SUBRET, -(SP)    ; memorizza valore di ritorno
        MOV     R0, -(SP)    ; ricarica PC su stack
        RTS     PC
```

Un altro metodo e` quello di utilizzare una struttura di dati che faccia da contenitore per i parametri, riducendosi a comunicare alla subroutine il riferimento alla struttura con uno dei metodi gia` visti:

```
        MOV     VAL1, STRUCT
        MOV     #VAR, STRUCT+2
        MOV     #STRUCT, R0
        JSR     PC, SUBR
        ...

SUBR:   MOV     0(R0), PAR1     ; carica parametri da
        MOV     2(R0), PAR2     ; struttura
        ...
        ...
        MOV     SUBRET, 4(R0)   ; memorizza valore di
                                ; ritorno in struttura
        RTS     PC
STRUCT: .WORD   0,0,0
```

oppure

```
        MOV     VAL1, STRUCT
        MOV     #VAR, STRUCT+2
        MOV     #STRUCT, R0
        JSR     R0, SUBR
        ...

SUBR:   MOV     (SP), R4        ; carica puntatore struttura
        MOV     0(R4), PAR1     ; carica parametri da
        MOV     2(R4), PAR2     ; struttura
        ...
        ...
        MOV     SUBRET, 4(R4)
        RTS     R0
++++++++
```

Esercizio 2.15

Si scriva una subroutine ricorsiva (diretta) che riceve in R0 un intero N >= 1 e restituisce nello stesso registro il valore S(N) dato dalla somma dei primi N numeri naturali >= 0.

Una subroutine e` detta ricorsiva se e` in grado di chiamare se stessa, direttamente o innescando una successione (temporale) di chiamate ad altre subroutine per la quale alla fine viene chiamata quella ricorsiva. Questo significa che, in un certo momento, possono esistere piu` "istanze" della subroutine: queste sono identificate da altrettanti indirizzi di ritorno sullo stack dovuti a chiamate alla subroutine ricorsiva. Perche` il funzionamento della subroutine sia corretto, e` necessario che ogni istanza operi su un insieme di variabili distinto. Il metodo piu` semplice e` di allocare tali variabili sullo stack: cio` garantisce che siano in ogni istante accessibili le variabili dell'istanza in quel momento attiva. Questo vale in particolare per i parametri eventualmente passati in sede di chiamata mediante registri.

Nel caso dell'esercizio, l'algoritmo puo` essere espresso in forma ricorsiva nel modo seguente:

a) S(N) = N - 1 + S(N-1) N>1
b) S(1) = 0

Quindi, S(1)=0, S(2)=1; S(3)=3 ....

a) rappresenta la parte ricorsiva e b) la condizione di terminazione. Il tutto e` realizzabile con la seguente subroutine, che riceve il valore di N in R0 e restituisce nel medesimo registro il valore S(N):

```
SUM:    DEC     R0          ; N-1
        BEQ     ENDED       ; termine in N-1=0 , S(N)=0
        MOV     R0, (-SP)   ; salvataggio N-1 su stack
        JSR     PC, SUM     ; R0 <- S(N-1)
RET:    ADD     (SP)+, R0   ; R0 <- N - 1 + S(N-1)
ENDED:  RTS     PC
```

Con N(3), al momento della chiamata della terza istanza di SUM, lo stack e` cosi` configurato:

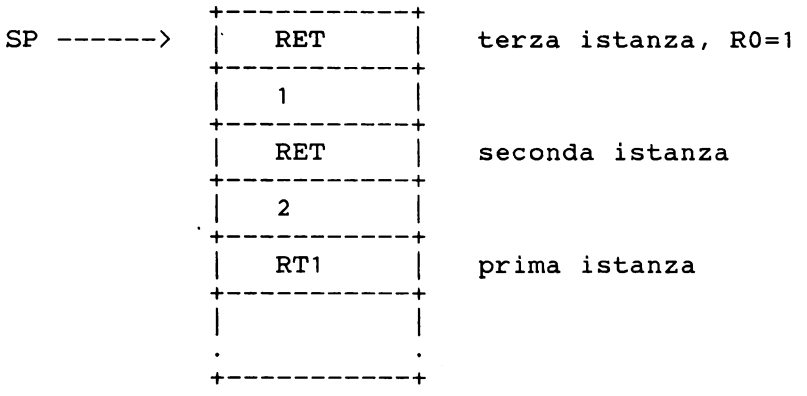


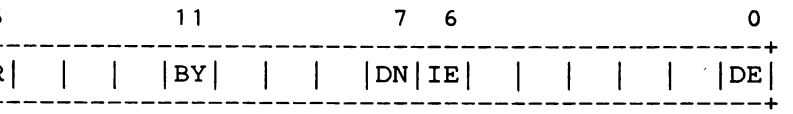
Fig. 2.9

Il metodo del salvataggio dei registri sullo stack puo` essere piu` in generale utilizzato per salvaguardare i loro contenuti quando risulta necessario utilizzarli temporaneamente per altri scopi, comprendendo eventuali chiamate ad altre subroutine. Un caso particolare e` quello delle routine di interruzione (vedi oltre).
++++++++

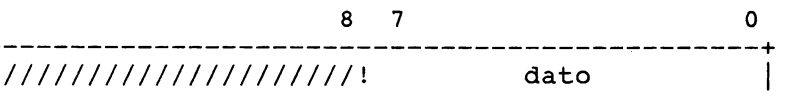
2.6 - Istruzioni di input/output

Il PDP11 non prevede specifiche istruzioni di ingresso uscita: l'unita` centrale comunica con i dispositivi esterni mediante registri, indirizzabili in modo del tutto analogo alle locazioni di memoria. Pertanto tutte le istruzioni che prevedono come operando una locazione di memoria, possono egualmente accedere a registri di I/O: a questo scopo il sistema riserva una parte dello spazio indirizzabile (generalmente quella piu` alta) per gli indirizzi di dispositivi.

Ad esempio, un tipico dispositivo di ingresso puo` comunicare con il processore attraverso due registri: un registro dati (o buffer) e un registro di stato (fig. 2.10). Il buffer contiene il dato (8 bit) letto mentre il registro di stato ha lo scopo di sincronizzare la CPU con il dispositivo.



Registro di Stato di un dispositivo di ingresso



Registro Dati di un dispositivo di ingresso

0	DE (Device Enable)	dispositivo abilitato (Read Only)
6	IE (Interrupt Enable)	interruzioni da dispositivo abilitate (R/W)
7	DN (Done)	operazione terminata e dato disponibile (Read Only)
11	BY (Busy)	operazione in corso (Read Only)
15	ER (Error)	errore in lettura (Read Only)

Fig. 2.10

Alcuni bit del registro di stato possono essere esaminati dalla CPU ma non modificati (Read Only), altri viceversa (Write Only), per altri entrambe le operazioni sono possibili (R/W). Una normale operazione di lettura prevede di porre a 1 il bit DE (inizialmente a 0) per abilitare il dispositivo, a cui quest'ultimo risponde ponendo a 1 il bit BY e 0 il bit DN finche` l'operazione non e` completata. Quando questo avviene, il dispositivo pone BY e DE =0 e DN=1, in questo modo segnalando al processore la effettiva disponibilita` del dato nel buffer. Generalmente a fronte della lettura del dato il bit DN viene posto a 0. Se durante la lettura si verifica un errore, in corrispondenza alla commutazione di BY 1->0, viene anche posto a 1 il bit ER. Effettuata la lettura del dato, il processore puo` riabilitare il dispositivo ponendo DE=1 per una successiva lettura. Il processore quindi puo` dedurre lo stato del dispositivo dai due bit BY e DN.

BY	DN	Stato
0	0	Non attivo
1	0	Lettura in corso
0	1	Lettura completata, dato disponibile
1	1	-----

La condizione di dato disponibile puo` essere rilevata dal processore eseguendo ripetutamente un'istruzione che legga lo stato del bit DN. Si dice allora che il processore e` in attesa passiva (busy waiting) che il dispositivo abbia completato l'operazione. Questa attesa passiva da parte della CPU puo` essere ovviata con la tecnica della interruzione, che verra` trattata successivamente, ma il busy waiting ha il vantaggio di garantire la massima prontezza nell'acquisizione di un dato non appena disponibile.

Esercizio 2.16

Supponendo che INS e INB siano gli indirizzi rispettivamente dei registri di stato e dati di un dispositivo di ingresso, si

scriva un segmento di programma che legge 10 dati (byte) successivi, memorizzandoli in locazioni contigue di memoria.

```
GOREAD: CLR     R1        ; azzera indice
LOOP:   INCB    @#INS     ; DE <- 1
WAIT:   TSTB    @#INS     ; valuta se DN=1
        BPL     WAIT      ; cicla se dato non disponibile (DN=0)
        TST     @#INS     ; valuta se errore
        BMI     ERROR     ; salta se si`
        MOVB    @#INB, AREA(R1) ; legge e memorizza in vettore
        INC     R1        ; incrementa indice
        CMP     #10.,R1   ; fine vettore
        BNE     LOOP      ; cicla non terminato
        ....
ERROR:  ......            ; trattamento della condizione d'errore
        ....
AREA:   .BLKB   10.
```

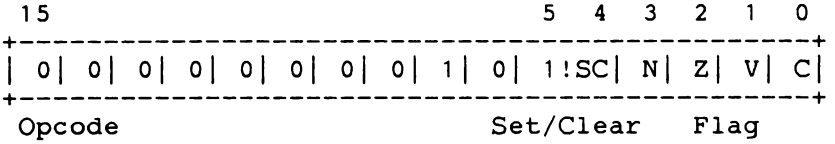
La pseudo-istruzione .BLKB riserva uno spazio di 10 byte, a partire dall'indirizzo di caricamento corrente, come area di memorizzazione per i dati letti.
++++++++

2.7 - Altre istruzioni

Si fornisce qui un breve elenco di altre istruzioni di vario tipo con una succinta spiegazione.

Opcode	Simbolo	Commento
Miscellanea		
000000	HALT	Arresta il processore
000001	WAIT	Sospende il processore in attesa di interr.
000005	RESET	Inizializza il sistema
000240	NOP	Operazione nulla
Operazioni sui bit di condizione		
000241	CLC	C = 0
000242	CLV	V = 0
000244	CLZ	Z = 0
000250	CLN	N = 0
000257	CCC	NZVC = 0000
000261	SEC	C = 1
000262	SEV	V = 1
000264	SEZ	Z = 1
000270	SEN	N = 1
000277	SCC	NZVC = 1111

Tab. 2.6



Operazioni sui bit di condizione
Fig. 2.11

Vi e` da notare che le operazioni sui bit di condizione sono "componibili" nel senso che, pur non esistendo un mnemonico associato, sono ammissibili istruzioni che sono l'OR bit a bit di due di quelle sopra descritte e omologhe (cioe` entrambe di Set o di Clear).

2.8 - Interruzioni e Trap

Il PDP11 dispone di un meccanismo di risposta alle interruzioni e ai trap vettorizzato. A fronte di un segnale di interruzione asincrona, il processore verifica se la priorita` dell'interruzione e` superiore alla priorita` del processore e, in caso affermativo, attiva la sequenza di servizio; altrimenti, l'interruzione rimane pendente in attesa che si verifichino le condizioni di priorita` richieste. Mentre la priorita` dell'interruzione e` prefissata, dipendendo esclusivamente dalla connessione fisica del dispositivo interrompente, quella del processore e` variabile ed e` rappresentata dai bit ad essa riservati nella parola di stato PSW. La priorita` del processore puo` essere letta e modificata da programma rispettivamente con le istruzioni MFPS e MTPS. Per un trap, la sequenza di servizio viene comunque attivata, indipendentemente dalla priorita` del processore.

La sequenza di servizio prevede le seguenti operazioni:

1) completamento della istruzione correntemente in esecuzione;
2) salvataggio di PC e PSW sullo stack;
3) caricamento di PC e PSW con il vettore d'interruzione associato;
4) normale ciclo di esecuzione, che parte con la prima istruzione della routine di servizio dell'interruzione.

Un vettore di interruzione e` una coppia di word contigui, sistemati ad un indirizzo prefissato e quindi associati univocamente ad una particolare interruzione o trap. A questo scopo tutti gli indirizzi da 0 a 376 compresi sono riservati come indirizzi di vettori d'interruzione, al punto che certi modelli sollevano una condizione eccezionale (stack violation trap) se lo SP assume valori inferiori a 420, onde proteggere quest'area da accessi indebiti. Il programmatore o il sistema operativo devono provvedere ad inizializzare tutti i vettori per le interruzioni e trap di interesse.

Perche` il meccanismo di servizio funzioni correttamente, e` necessario che la nuova priorita` del processore, specificata come seconda componente del vettore, sia maggiore o uguale a quella fisica nel caso di interruzione asincrona, onde evitare che, rimanendo questa pendente anche durante l'esecuzione della prima istruzione della routine di servizio, venga in ripetizione attivata la sequenza di servizio, fino allo sfondamento dello stack.

La routine di servizio deve tipicamente effettuare operazioni di ingresso e/o di uscita per soddisfare la richiesta proveniente dal dispositivo esterno ed eliminare quindi la causa della richiesta stessa. Ad esempio, nel caso del dispositivo tipo precedentemente esaminato (fig. 2.10), la richiesta di interruzione si verifica quando DN=IE=1 e puo` essere eliminata leggendo un dato dal dispositivo (DN<-0). Tenendo conto che la routine puo` venire eseguita in un qualsiasi punto del programma interrotto, e` necessario che vengano salvati all'ingresso e ripristinati all'uscita della routine tutti i registri da essa utilizzati: e` comune norma adottare il

metodo del salvataggio su stack che garantisce la correttezza anche per routine a loro volta interrompibili (se la priorita` lo consente).

La routine di servizio deve terminare con una speciale istruzione di ritorno:

```
    RTI

PC <- Pop ()
PSW <- Pop ()
```

per cui viene ridato il controllo al programma interrotto, ristabilendo la priorita` precedentemente salvata. Si noti che e` necessario disporre di un'unica istruzione che contempli le due operazioni poiche` l'ordine non puo` essere invertito (priorita` illegale ancora all'interno della routine di servizio) e, se il ripristino di PC fosse separato da quello di PSW, si ritornerebbe al programma interrotto con priorita` non ripristinabile.

La CPU e` generalmente in grado di abilitare e disabilitare la possibilita` di richiesta di interruzione da parte di singoli dispositivi, impostando un bit allo scopo riservato nel registro di stato del dispositivo (vedi IE fig. 2.10). Cio` consente un maggior controllo nel mascheramento delle interruzioni anche nell'ipotesi di piu` dispositivi collegati alla medesima linea interruzione fisica.

In alcuni sistemi, le interruzioni possono avere priorita` fisica nel campo 4..7 per cui un programma e` senz'altro interrompibile nelle sezioni in cui la priorita` del processore e` compresa tra 0 e 3, mentre per valori compresi tra 4 e 6, solo alcuni dispositivi sono in grado di interrompere la CPU. Se la priorita` e` massima (7), le interruzioni mascherabili sono di fatto tutte disabilitate.

Le interruzioni sincrone (trap) si distinguono in istruzioni vere e proprie e interruzioni dovute ad effetti collaterali dell'esecuzione del programma o del funzionamento della CPU.

Opcode	Simbolo	Vector Address	Commento
Trap			
000003	BPT	14	Usata come breakpoint
000004	IOT	20	Usata per attivare routine di I/O del sistema operativo
104NNN	EMT	30	Usata per attivare un emulatore
104NNN	TRAP	34	Interruzione sincrona generica

Tab. 2.7

Le due istruzioni EMT e TRAP si distinguono perche` rispettivamente e` 0<=NNN<=377 e 400<=NNN<=777. Il campo NNN rappresenta un parametro passato alla routine di servizio direttamente all'interno del codice operativo. Poiche` EMT e` normalmente utilizzata da componenti del sistema operativo, viene consigliato all'utente l'uso della sola istruzione di TRAP come interruzione sincrona generica. L'istruzione completa ha il seguente formato:

```
        TRAP    NNN
```

Gli eventi collaterali che producono interruzione sincrona sono:

- Modalita` registro utilizzata per JMP e JSR (Vector Address = 4)
- Uso di istruzioni illegali e riservate (Vector Address = 10)
- Power Fail (Vector Address = 24)

Ogni interruzione sincrona ha priorita` 8 e quindi non e` mascherabile. Produce un effetto del tutto analogo a quello delle interruzioni asincrone e la relativa routine di servizio deve terminare con l'istruzione di ritorno RTI.

Le istruzioni di Trap sono utilizzate generalemente per attivare funzioni del sistema operativo. Si noti che, se il bit T (bit 4) di PSW e` posto a 1, dopo l'esecuzione di ciascuna istruzione di programma viene sollevata una interruzione sincrona analoga a BPT (cioe` con lo stesso vector address) cosicche` e` molto facile ottenere la funzione di *Single Step* in emulazione. Inoltre l'istruzione RTT viene in questo caso usata in sostituzione a RTI quando si voglia anche inibire il trace trap. Se il nuovo valore di PSW ha il T bit pari a 1, la prima interruzione si verifica dopo l'esecuzione della prima istruzione dopo RTT.

Esercizio 2.17

Nelle stesse ipotesi dell'esercizio 2.16, sapendo che viene richiesta una interruzione se DN=IE=1, modificare la soluzione in modo che la lettura avvenga con il meccanismo dell'interruzione.

```
        PEI     = 0             ; processor interrupt enable
        INPIE = 100             ; maschera bit 6
        INPVA = indirizzo del vettore di interruzione
        INPSW = PSW per la routine di interruzione

IREAD:
        MOV     R1, -(SP)       ; salva R1
        MOV     INDEX, R1       ; carica indice
        TST     @#INS           ; valuta se errore
        BMI     ERROR           ; salta se si`
        MOVB    @#INB, AREA(R1) ; legge e memorizza in vettore
        INC     R1              ; incrementa indice
        CMP     #10., R1        ; fine vettore
        BEQ     ENDED           ; salta se terminato
        MOV     R1, INDEX       ; salva indice
        MOV     (SP)+, R1       ; ripristina R1
        INCB    @#INS           ; pone DE=1
        RTI
ENDED:  BICB    #INPIE, @#INS   ; disabilita interruzioni
                                ; dispositivo
        MOV     (SP)+, R1       ; ripristina R1
        RTI

        ; programma principale

GOREAD: CLR     INDEX           ; azzera indice
        MOV     #INPVA, R1      ; carica vector address
        MOV     #IREAD,  0(R1)  ; vettore interruzioni
        MOV     #INPSW, 2(R1)
        BISB    #INPIE, @#INS   ; abilita interruzioni
```

```
                                ; dispositivo
        MTPS    #PEI            ; abilita interruzioni
LOOP:   INC     R1              ; incremento di un contatore
        BR      LOOP            ; (in genere il programma
                                ; potra` svolgere attivita
                                ; piu` "utili"

        ....
ERROR:  ....
        ....

INDEX:  .WORD   0
AREA:   .BLKB   10.
```

Il programma principale inizializza il vettore di interruzione all'indirizzo INPVA con l'indirizzo della routine di servizio IREAD e con il relativo valore di PSW. Deve poi provvedere in generale ad abilitare le interruzioni nel complesso e quella specifica del dispositivo. Fatto questo puo` svolgere una qualsiasi attivita` che verra` interrotta dalla ricezione dei dati successivi, caricati da parte della routine di servizio nell'area predisposta, fino a completamento, raggiunto il quale il dispositivo viene disabilitato a interrompere. Si noti la necessita` di prevedere una variabile che conservi l'indice tra una istanza e la successiva della routine.
++++++++

2.9 - Linguaggio di assemblatore e strutture di dati

Si e` gia` visto in precedenza che un programma e` esprimibile mediante una sequenza di istruzioni e pseudo-istruzioni, scritte secondo un opportuno formalismo detto *Linguaggio Assembly*. In questo paragrafo si vuole fornire una definizione piu` precisa di tale formalismo per il PDP11, ampliandolo con ulteriori pseudo-istruzioni e con dettagli sulla struttura di un programma.

Poiche` un programma in linguaggio assembly non puo` essere direttamente eseguito dalla CPU, e` necessaria una traduzione in linguaggio macchina: e` questo il compito principale di un *Assemblatore*. Un assemblatore e` un programma in grado di ricevere in ingresso un testo ASCII, che costituisce un modulo sorgente, scritto secondo le convenzioni del linguaggio assembly e in grado di produrre in uscita un modulo assoluto oppure rilocabile. Se rilocabile, il modulo verra` successivamente collegato ad altri moduli rilocabili per formare un modulo assoluto e caricabile.

Viene qui di seguito definita mediante una grammatica la parte essenziale del linguaggio assembly per il PDP11. Eventuali estensioni o limitazioni dipendono dal particolare assemblatore utilizzato. Nella grammatica si sono utilizzate le seguenti convenzioni:

```
Label   simbolo non terminale
STRINGA simbolo terminale
::=     simbolo di definizione
#       simbolo di alternativa
[x]     termine opzionale
[x]*    termine opzionale ricorsivo (ripetuto n>=0 volte)
'a      carattere a
"str"   stringa str
```

```
ModuloSorgente ::=
    SezioneImplicita
      [SezioneAssoluta # SezioneRilocabile]* FineModulo

SezioneImplicita ::=
    [ElementoDiAssembly]*

SezioneAssoluta ::=
    ".ASECT" FINELINEA [ElementoDiAssembly]*

SezioneRilocabile ::=
    ".CSECT" [NomeSezione] FINELINEA [ElementoDiAssembly]*

FineModulo ::=
    ".END" [ESPRESSIONE] FINELINEA

ElementoDiAssembly ::=
    [Label ':]  [ParteOperativa] ['; Commento] FINELINEA #
    MacroDefinizione # SezioneRipetitiva # SezioneCondizionale

Label ::=
    NOME

ParteOperativa ::=
    Direttiva # Istruzione # AssegnazioneSimbolica

Commento ::=
    [STRINGA]*

MacroDefinizione ::=
    ".MACRO" NomeMacro [[',] ParametroMacro]* FINELINEA
      [ElementoDiAssembly]*
      ".ENDM" FINELINEA

SezioneRipetitiva ::=
    CondizioneDiRipetizione [ElementoDiAssembly]*
      ".ENDR" FINELINEA

SezioneCondizionale ::=
    ".IF" Condizione ', ArgomentiDiCondizione FINELINEA
      SezCondImplicita
      [SezCondVera # SezCondFalsa # SezNonCondizionale]*
      ".ENDC" FINELINEA

Direttiva ::=
    ".GLOBL" Simbolo [', Simbolo]* #
      ".MEXIT" #
      ".ASCII" '/ STRINGA '/ #
      ".BLKB" ESPRESSIONE #
      ".BLKW" ESPRESSIONE #
      ".BYTE" ESPRESSIONE [', ESPRESSIONE]* #
      ".EVEN" #
      ".WORD" ESPRESSIONE [', ESPRESSIONE]*

Istruzione ::=
    CodiceOperativo [Argomento [', Argomento]* ]


AssegnazioneSimbolica ::=
    Simbolo '= ESPRESSIONE
```

```
NomeMacro , ParametroMacro, CodiceOperativo, Simbolo ::=
    NOME

Argomento ::=
    ESPRESSIONE

CondizioneDiRipetizione ::=
    ".IRP" Simbolo ', '< STRINGA [', STRINGA]* '> #
    ".IRPC" Simbolo ', '< STRINGA '> #
    ".REPT" ESPRESSIONE

Condizione ::=
    "EQ" # "NE" # "GT" # "LE" # "LT" # "GE" #
    "DF" # "NDF" # "B" # "NB" # "IDN" # "DIF"

ArgomentiDiCondizione ::=
    ESPRESSIONE # NOME [[',] NOME] #

SezCondImplicita ::=
    [ElementoDiAssembly]*

SezCondVera ::=
    ".IFT" FINELINEA
      [ElementoDiAssembly]*

SezCondFalsa ::=
    ".IFF" FINELINEA
      [ElementoDiAssembly]*

SezNonCondizionale ::=
    ".IFTF" FINELINEA
      [ElementoDiAssembly]*
```

Riassumendo, un modulo sorgente e` costituito da una successione di sezioni, assolute o rilocabili, composte di linee corrispondenti a singole istruzioni del linguaggio. La sezione implicita e` di tipo rilocabile e non e` identificata da alcun prologo.

Il significato di ciascuna sezione e` legato al concetto di *contatore di locazione*. Esso rappresenta in ogni punto del modulo l'indirizzo (virtuale) di caricamento delle singole istruzioni ed e` simbolicamente indicato con '.'. Una sezione assoluta precisa completamente il valore del contatore di locazione, eventualmente mediante assegnazioni simboliche che ne modificano la normale progressione di valori dovuti alla lunghezza delle istruzioni. In una sezione rilocabile, con o senza nome, il valore del contatore di locazione e` specificato solo in termini relativi rispetto all'inizio della sezione stessa: la specificazione completa viene rimandata al momento del collegamento con altri moduli. Di fatto esiste un'unica sezione assoluta, poiche` una nuova sezione aperta con la pseudo-istruzione .ASECT dopo una sezione rilocabile e` considerata il prolungamento della sezione assoluta precedente; lo stesso vale per la sezione rilocabile senza nome e per una singola sezione con nome. Anche nelle sezioni rilocabili il contatore di locazione puo` essere modificato con una assegnazione simbolica ma non puo` mai essere riportato indietro rispetto al valore che aveva raggiunto.

Le linee di assembly possono essere istruzioni del linguaggio

del PDP11, eventualmente precedute da label e seguite da commento che termina alla fine della linea. Gli operandi possono essere specificati come simboli predefiniti (ad esempio registri), simboli definiti nel programma (ad esempio label, simboli definiti mediante assegnazione simbolica oppure mediante apposite direttive) oppure espressioni i cui operandi sono costanti o simboli del tipo detto. Le espressioni possono generalmente contenere operatori algebrici e logici e parentesi '< e '> accoppiate per la definizione delle priorita`, pero` devono soddisfare alcune limitazioni che dipendono dall'assemblatore usato e da altre caratteristiche. Una trattazione completa dell'argomento esula comunque dagli scopi di questa breve descrizione. Per la stessa ragione il simbolo ESPRESSIONE e` considerato terminale nella grammatica sopra esposta.

Le linee di assembly possono essere anche direttive (pseudo-istruzioni), comprendendo in esse anche l'assegnazione simbolica, che esprimono comandi, interpretati dal programma assemblatore, il cui scopo principale e` di modificare il normale funzionamento incrementale dell'assemblaggio e di fornire informazioni accessorie per tale processo. In Tab. 2.8 vengono riassunte le direttive principali.

Formalismo	Commento
Definizione di simboli	
Simbolo '= ESPRESSIONE	Definisce il simbolo assegnando ad esso come valore il risultato dell'espressione (calcolata in fase di assemblaggio)
.GLOBL Simbolo [', Simbolo]*	Definisce i simboli che seguono come globali
Sezioni e macro	
.ASECT	Apre una sezione assoluta
.CSECT [NomeSezione]	Apre una sez. rilocabile (di nome NomeSezione)
.IF Condizione, ArgomentiDiCondizione	Apre una sezione ad assemblaggio condizionale
.IFF	All'interno di una sezione condizionale, apre una sottosezione da assemblare se la condizione e` falsa
.IFT	All'interno di una sezione condizionale, apre una sottosezione da assemblare se la condizione e` vera
.IFTF	All'interno di una sezione condizionale, apre una sottosezione da assemblare incondizionatamente
.IRP Simbolo ', '< STRINGA [', STRINGA]* '>	Apre una sezione ripetitiva, eseguita assegnando per ogni ciclo al simbolo specificato ciascuno dei valori della lista
.IRPC Simbolo ', '< STRINGA '>	Apre una sezione ripetitiva, eseguita assegnando per ogni ciclo al simbolo specificato ciascuno dei caratteri presenti in STRINGA
.REPT ESPRESSIONE	Apre una sezione ripetitiva, ripetuta un numero di volte pari al valore dell'espressione

Tab. 2.8

.MACRO NomeMacro [', ParametroMacro]*	Apre una definizione di macro di nome NomeMacro
.MEXIT	Forza la terminazione nell'espansione della macro o sezione ripetitiva corrente
.END [ESPRESSIONE]	Termina il modulo sorgente e precisa eventualmente il punto di attivazione se il modulo e` principale
.ENDC	Termina la sezione ad assemblaggio condizionale piu` interna
.ENDM	Termina la definizione di macro piu` interna
.ENDR	Termina la sezione ripetitiva piu` interna

Allocazione di memoria

.ASCII '/STRINGA'/	Genera un vettore di byte e lo inizializza con i codici ASCII dei caratteri contenuti tra i due delimitatori (due caratteri eguali non spazio)
.BLKB ESPRESSIONE	Riserva un blocco di memoria di ampiezza in byte pari al valore della espressione (cioe` incrementa il contatore di locazione di tale valore)
.BLKW ESPRESSIONE	Riserva un blocco di memoria di ampiezza in word pari al valore della espressione (cioe` incrementa il contatore di locazione del doppio di tale valore)
.BYTE ESPRESSIONE [', ESPRESSIONE]*	Genera un vettore di byte, inizializzato con i valori delle espressioni che seguono
.EVEN	Incrementa di uno il contatore di locazione se dispari (per allineamento sulla frontiera pari)
.WORD ESPRESSIONE [', ESPRESSIONE]*	Genera un vettore di word, inizializzato con i valori delle espressioni che seguono

Tab. 2.8 (cont.)

La direttiva .GLOBL permette di definire simboli che devono essere resi noti agli altri moduli con cui quello corrente deve essere collegato.

La direttiva .IF permette l'assemblaggio condizionale, ovvero sulla base della condizione specificata, si decide se procedere o meno alla espansione di tutto il blocco condizionale di cui essa e` prologo. Il blocco termina in corrispondenza di una pseudo-istruzione .ENDC; i blocchi condizionali sono innestabili. Le condizioni possibili per .IF sono:

EQ	eguale a 0
NE	diverso da 0
GT	maggiore di 0
GE	maggiore o uguale a 0
LT	minore di 0
LE	minore o uguale a 0
DF	simbolo gia` definito
NDF	simbolo non ancora definito
B	argomento di macro nullo
NB	argomento di macro nonnullo
IDN	coppia di argomenti eguali (in macro)
DIF	coppia di argomenti differenti (in macro)

Il codice che segue .IF viene espanso solo se la condizione e` vera. All'interno del blocco condizionale vi possono essere sotto-

sezioni di istruzioni precedute da .IFF, .IFT e .IFTF che vengono espanse rispettivamente se la condizione e` vera, falsa e indifferentemente vera o falsa.

Ad esempio:

```
        .IF NE, ITA
        .IFT
            .ASCII /Numero Versione: /
        .IFF
            .ASCII /Version Number: /
        .IFTF
            .ASCII /3.12/
        .ENDC
```

La stringa italiana o quella inglese vengono espanse in alternativa in base al valore assegnato al simbolo ITA (italiana se diverso da 0).

Le direttive .IRP, .IRPC e .REPT definiscono blocchi di assemblaggio ripetitivo, terminanti con un'istruzione .ENDR e innestabili. Tutto il codice all'interno del blocco viene espanso piu` volte sulla base della condizione di ripetizione. Nel caso di .IRP il simbolo indicato assume ad ogni ciclo uno dei valori elencati nella lista tra parentesi angolari; nel caso di .IRPC il simbolo assume come valore ciascun carattere della stringa indicata; per .REPT la ripetizione ha luogo tante volte quanto e` il valore dell'espressione indicata.

La direttiva .MACRO costuituisce il prologo della definizione di una macro, che termina con una istruzione .ENDM. Una macro e` un meccanismo parametrico di generazione di codice. La definizione di una macro viene memorizzata dall'assemblatore e, ogni qualvolta compare nel testo successivo il nome di una macro precedentemente definita, la pseudo-istruzione che contiene tale nome viene sostituita dalla espansione della macro corrispondente. Quindi una chiamata a macro assume l'aspetto di una normale istruzione con parametri:

```
[Label:] NomeMacro [Argomento [[',] Argomento]* ] [';Commento]
```

Al momento della chiamata viene stabilita una corrispondenza biunivoca tra i parametri formali contenuti nella definizione della macro e gli argomenti di chiamata. Ciascuna linea del corpo della macro viene ora considerata dall'assemblatore come se fosse una normale linea sorgente, salvo la sostituzione del nome di un parametro formale all'interno dell'istruzione con il relativo argomento attuale (come stringa). La linea cosi` generata viene risottomessa allo stesso procedimento, permettendo pertanto di avere chiamate a macro anche all'interno di macro definizioni.

Ad esempio:

```
        .MACRO  PUSH    A   ; push nello stack
        MOV     A, -(SP)
        .ENDM

        .MACRO  POP     A   ; pop dallo stack
        MOV     (SP)+, A
        .ENDM
```

Istruzioni del tipo:

```
        PUSH    R0
        POP     @#PIPPO
```

vengono espanse in:

```
        MOV     R0, -(SP)
        MOV     (SP)+, @#PIPPO
```

Risulta molto conveniente utilizzare blocchi condizionali e ripetitivi all'interno di una macro, poiche` le condizioni associate possono essere rese parametriche. La direttiva .MEXIT forza la terminazione della espansione di una macro, qualora fosse in certe condizioni necessario, in un punto intermedio del suo corpo.

Ad esempio:

```
        .MACRO  TEST RX, ERR
        TST     RX
        .IF     NB, ERR          ; espande se ERR non vuoto
        .IF     LT, 254.-.+ERR   ; salto relativo se possibile
        BEQ     .+4
        JMP     ERR
        .IFF
        BNE     ERR
        .ENDC
        .ENDM
```

Questa macro si espande in un'istruzione di test seguita da un salto condizionale ad una routine d'errore. Il salto avviene solo se l'indirizzo della routine viene effettivamente passato come parametro, altrimenti l'istruzione di salto non viene espansa. Inoltre, in caso di salto sceglie uno relativo se nel campo possibile. Per l'esempio, se ERROR = 3000 e la seguente istruzione:

```
        TEST    R0, ERROR
```

e` caricata all'indirizzo 1000, l'espansione prodotta e` la seguente:

```
        TST     R0
        BEQ     +.4
        JMP     ERROR
```

Se invece ERROR = 1100, l'espansione e` la seguente:

```
        TST     R0
        BNE     ERROR
```

Esercizio 2.18

Si definisca una coppia di macro SAVE e RESTORE atte a salvare e ripristinare la lista di registri specificata come parametro.

In questo caso e` necessario risolvere il problema del passaggio di una lista come unico parametro. Infatti una lista contiene caratteri di separazione (spazi o virgole) che normalmente separano fra

loro argomenti distinti nella chiamata. Allo scopo i macro-assemblatori utilizzano una convenzione apposita, come per esempio inserire tutta la lista tra parentesi angolari. In questo caso cio` che e` compreso fra queste parentesi viene considerato come argomento del corrispondente parametro formale. Inoltre, perche` il salvataggio e successivo ripristino funzionino correttamente, e` necessario che la lista in RESTORE sia data in ordine inverso rispetto a SAVE.

```
        .MACRO  SAVE REGL    ; salvataggio registri
        .IF NB, <REGL>       ; lista non vuota
        .IRP    REG, <REGL> ; ciclo su lista
        MOV     REG, -(SP)
        .ENDR
        .ENDC
        .ENDM

        .MACRO  RESTORE REGL     ; ripristina registri
        .IF NB, <REGL>       ; lista non vuota
        .IRP    REG, <REGL> ; ciclo su lista
        MOV     (SP)+, REG
        .ENDR
        .ENDC
        .ENDM
```

Chiamate del tipo:

```
        SAVE    <R0, R1, R4>
        ....
        RESTORE <R4, R1, R0>
```

vengono espanse in:

```
        MOV     R0, -(SP)
        MOV     R1, -(SP)
        MOV     R4, -(SP)
        ....
        MOV     (SP)+, R4
        MOV     (SP)+, R1
        MOV     (SP)+, R0
```

++++++++

Esercizio 2.19

Definire una serie di macro che simulino il comportamento di una macchina ad un indirizzo (con registro accumulatore), che sia in grado di eseguire le seguenti istruzioni:

```
LOAD    OPND        carica nell'accumulatore il contenuto
                    dell'operando
STORE   OPND        carica nell'operando il contenuto
                    dell'accumulatore
SUM     OPND        somma all'accumulatore il contenuto
                    dell'operando
DIF     OPND        sottrae all'accumulatore il contenuto
                    dell'operando
JZC     DD          salta a DD se risultato
                    (nell'accumulatore) <> 0
HLT                 ferma il processore
```

Si supponga che l'operando possa essere una locazione di memoria, riferita in modo diretto oppure indirettamente attraverso un puntatore) o un valore immediato (ad eccezione di STORE). La destinazione di un salto e` un indirizzo di programma (Label).

Scrivere nel nuovo linguaggio un segmento di programma per inizializzare un vettore di N locazione a[i] = i-1 con i=1..N e rappresentarne l'espansione.

Viene scelto come accumulatore R0:

```
        .MACRO  LOAD    OP
        MOV     OP, R0          ; Z = *
        .ENDM

        .MACRO  STORE   OP
        MOV     R0, OP          ; Z = *
        .ENDM

        .MACRO  SUM     OP
        ADD     OP, R0          ; Z = *
        .ENDM

        .MACRO  DIF     OP
        SUB     OP, R0          ; Z = *
        .ENDM

        .MACRO  JZC     DD
        BEQ     .+4
        JMP     DD              ; salto esteso
        .ENDM

        .MACRO  HLT
        HALT
        .ENDM
```

Il programma e` il seguente:

```
INIT:   LOAD    #A              ; carica indirizzo vettore
        SUM     N
        SUM     N
        DIF     #2              ; A+2*M[N]-2 (ind. di a[n])
        STORE   PTR             ; carica puntatore
        LOAD    N               ; carica contatore
RIP:    DIF     #1
        STORE   COUNT           ; memoria conteggio
        STORE   @PTR            ; a[i] <- i-1
        LOAD    PTR
        DIF     #2
        STORE   PTR             ; decrementa puntatore
        LOAD    COUNT
        JZC     RIP             ; cicla fino a COUNT=0
        HLT
```

L'espansione e` la seguente:

```
INIT:   MOV     #A, R0
        ADD     N, R0
        ADD     N, R0
        SUB     #2, R0
        MOV     R0, PTR
        MOV     N, R0
RIP:    SUB     #1, R0
        MOV     R0, COUNT
        MOV     R0, @PTR
        MOV     PTR, R0
        SUB     #2, R0
        MOV     R0, PTR
        MOV     COUNT, R0
        BEQ     .+4
        JMP     RIP
        HALT
++++++++
```

Le direttive per l'allocazione della memoria sono gia` state in gran parte illustrate in precenza. Hanno tutte la caratteristica di incrementare il contatore di locazione di una quantita` che e` fornita in modo parametrico nella direttiva, riservando pertanto lo spazio richiesto. Generalmente questo e` fatto in connessione con la dichiarazione di un label che assume il significato di indirizzo iniziale di una struttura di dati. Lo spazio, nel caso delle direttive .WORD, .BYTE e .ASCII viene in piu` inizializzato in fase di caricamento con valori forniti numericamente e, per .ASCII, sottoforma di sequenza di codici ASCII corrispondenti ai caratteri di una stringa tra delimitatori eguali. Le direttive .ASCII, .BYTE e .BLKB possono produrre come effetto indesiderato che il contatore di locazione assuma un valore dispari: in questo caso e` possibile utilizzare la direttiva .EVEN per riportarlo ad una frontiera pari.

Anche in linguaggio assembly puo` risultare conveniente dare una strutturazione piu` precisa ai dati non elementari, in modo che l'accesso ad essi sia agevole. Ad esempio, si e` gia` visto che un vettore puo` essere realizzato riservando uno spazio lineare di memoria di cui si conosce l'indirizzo iniziale: l'accesso per indice e` possibile utilizzando l'omonima modalita` di indirizzamento.

Nel caso di vettori multidimensionali, il meccanismo di accesso e` piu` complesso. Detti d_1, d_2, ... dn gli estremi delle n dimensioni del vettore, esso occupa uno spazio in byte pari a:

$$d = d_1 * d_2 * \ldots * d_n * e$$

dove e e` l'occupazione in byte di un elemento del vettore. Per esempio, se gli elementi sono word, la direttiva:

VECT: .BLKW $d_1*d_2*\ldots*d_n$

riserva per VECT lo spazio richiesto. Detto i[1..n] con $0<=i[k]<=d_k-1$ un generico vettore i cui elementi forniscono gli indici di un elemento di VECT, supponendo che la memorizzazione degli elementi avvenga facendo variare piu` rapidamente gli indici a destra, si ha che l'indirizzo dell'elemento VECT[i] = VECT[i[1], i[2]...i[n]) e`:

$$ELEM = VECT + i[1]*d_2*d_3*...*d_n*e + i[2]*d_3*...*d_n*e + ... + i[n-1]*d_n*e + i[n]*e =$$
$$= VECT + (..(i[1]*d_2 + i[2])*d_3 + ...+i[n-1])*d_{n-1} + i[n])*e$$

Per le matrici bidimensionali di word, ad esempio, il calcolo si semplifica in:

$$ELEM = VECT + (i[1]*d_2 + i[2])*2$$

Volendo definire strutture di dati disomogenee, similmente al concetto di record dei linguaggi ad alto livello, occorre tener conto che non e` possibile operare direttamente con istruzioni assembly sui simboli associati ai vari campi del record. Risulta necessario trasformare tale informazione in un offset rispetto ad un indirizzo base rappresentato dall'indirizzo iniziale di una istanza del record.

Ad esempio, una dichiarazione del tipo:

```
VAR     Tempo:  RECORD
        Ora, Minuti, Secondi:integer;
        END
```

corrisponde alla dichiarazione di una struttura di dati composta da tre word, ordinatamente contenenti ora, minuti e secondi. Quindi per l'allocazione si puo` utilizzare al solito:

```
TEMPO:      .BLKW   3
```

mentre l'accesso puo` avvenire con istruzioni del tipo:

```
        ORA = 0
        MINUTI = 2
        SECONDI = 4

        MOV     #TEMPO, R0
        MOV     <campo>(R0), R1
```

ove <campo> puo` essere ORA o MINUTI o SECONDI e rappresenta l'offset relativo al campo richiesto. Questa tecnica e` utile anche nel caso vi siano piu` istanze dello stesso tipo di record, poiche` il registro R0 puo` essere caricato con l'istanza richiesta (si pensi ad esempio che R0 sia un parametro d'ingresso per una subroutine).

Esercizio 2.20

Definire un insieme di macro con cui sia possibile:

- dichiarare tipi di dati che sono vettori monodimensionali, di estensione generica, di dati di tipo gia` definito e record con 3 campi di tipo gia` definito;

- dichiarare istanze dei tipi definiti;

- consentire un accesso per indici con i vettori e simbolico con i record.

Inoltre, tradurre il seguente codice PASCAL in macro assembly che utilizzi le macro definite.

```
type    A = record
            F1, F2:integer; F3:char;
            end

        B = array [-1..3] of A;

        C = record
            F1:B; F2:A; F3:char;
            end

var     A1:A; B1:B; C1, C2:C;

C1.F1[2].F3 := C2.F3;
C2.F2 := C1.F1[0];
--------
```

In questo esempio si vedra` una applicazione di macro innestate cioe` di macro la cui espansione definisce nuove macro. Infatti la dichiarazione di un nuovo tipo deve corrispondere alla definizione di macro, specifiche per quel tipo, utili a dichiarare istanze e a generare il codice di accesso per le stesse. In altri termini, si supporra` di disporre di macro del tipo:

NomeTipo$DCL VAR	dichiara l'istanza di nome VAR e di tipo NomeTipo.
NomeTipo$OFS I, OFS	calcola l'offset per l'elemento di indice I in una generica istanza del tipo vettore NomeTipo, e lo somma a OFS. I e` un registro.
NomeTipo$OFS C, OFS	calcola l'offset per il campo C in una generica istanza del tipo record NomeTipo, e lo somma a OFS.
NomeTipo$VAL OFS, B, DP	carica in M[D] il valore dell'elemento di tipo NomeTipo ottenuto mediante indirizzo base B e offset OFS. OFS e` un registro; DP e` un registro che punta ad un'area di memoria di estensione almeno pari a quella del tipo.
NomeTipo$STO OFS, B, PS	carica da M[S] il valore dell'elemento di tipo NomeTipo ottenuto mediante indirizzo base B e offset OFS. OFS e` un registro; PS e` un registro che punta ad un'area di memoria di estensione almeno pari a quella del tipo.

Allo scopo, vengono definite le due macro di ordine superiore, una riservata ai vettori e una ai record, che generano le macro sopra elencate:

```
$MVDEF  T, N1, N2, TC
            definisce il nuovo tipo T come vettore i cui indici
            variano da N1 a N2 e il cui tipo componente e` TC
```

```
$MRDEF  T, C1, T1, C2, T2, C3, T3
            definisce il nuovo tipo T come record di campi  C1,  C2,
            C3 ciascuno rispettivamente di tipo T1, T2, T3.
```

Nelle definizioni che seguono viene utilizzato l'operatore '(apice) che ha il significato di concatenazione e serve a definire in fase di espansione simboli alfanumerici di cui una parte e` costituita da un parametro. Ci si e` anche serviti della possibilita`, disponibile in fase di espansione di macro, di generare simboli unici, che si ottiene aggiungendo un parametro formale preceduto da '?' nella lista della macro definizione e inserendo tale parametro nel codice di espansione nei punti ove deve comparire il simbolo generato.

```
 .MACRO  $MVDEF  T, N1, N2, TC   ; definiz. tipo vettore

    T'$E = <<N2-N1+1>*TC'$E+1>/2*2
    ; definisce l'estensione complessiva del tipo
    ; come il piu` piccolo multiplo di 2 >= al
    ; prodotto del numero degli elementi per l'estensione
    ; del tipo componente, che si presume gia` definito


    .MACRO  T'$DCL VAR ; dichiarazione di istanza di vettore
        .CSECT DAT  ; segmento dati
VAR:    .BLKB T'$E  ; allineato sulla frontiera pari
        .CSECT COD  ; segmento codice
    .ENDM


    .MACRO T'$OFS I, OFS    ; calcolo dell'offset per un elem.
        SUB     #N1, I      ; I <- I-N1
        MOV     I, -(SP)    ; salva I
        MOV     #TC'$E, I   ; I <- estensione TC = E
        MUL     (SP)+, I    ; I <- (I-N1)*E
        ADD     I, OFS      ; lo somma a OFS
    .ENDM


    .MACRO T'$VAL OFS, B, DP, ?L    ; legge valore da istanza
                                    ; usando R5
        MOV     #T'$E, R5       ; carica estensione
        ASR     R5              ; R5 <- E/2
L:      MOV     B'(OFS), (DP)+  ; legge word componenti e
        TST     (OFS)+          ; copia
        SOB     R5, L
    .ENDM


    .MACRO T'$STO OFS, B, PS, ?L    ; copia valore su istanza
                                    ; usando R5
        MOV     #T'$E, R5       ; carica estensione
        ASR     R5              ; R5 <- E/2
L:      MOV     (PS)+, B'(OFS)  ; legge word componenti e
        TST     (OFS)+          ; copia
        SOB     R5, L
    .ENDM
 .ENDM
```

```
 .MACRO  $MRDEF  T, C1, T1, C2, T2, C3, T3
            ; definiz. tipo record

    T'$E = <T1'$E+T2'$E+T3'$E+1>/2*2
    ; definisce l'estensione complessiva del tipo
    ; come il piu` piccolo multiplo di 2 >= alla
    ; somma delle estensioni dei tipi componenti
    ; che si presumono gia` definiti


    .MACRO  T'$DCL VAR ; dichiarazione di istanza di record
        .CSECT DAT  ; segmento dati
VAR:    .BLKB T'$E  ; allineato sulla frontiera pari
        .CSECT COD  ; segmento codice
    .ENDM


    .MACRO T'$OFS C, OFS     ; calcolo dell'offset per un campo
                             ; se primo campo, OFS inalterato
      .IF IDN, C, C2         ; secondo campo
        ADD     #T1'$E, OFS  ; OFS <- OFS + E1
      .IFF
        .IF IDN, C, C3       ; terzo campo
        ADD     #T1'$E+T2'$E, OFS
        .ENDC
      .ENDC
    .ENDM


    .MACRO T'$VAL OFS, B, DP, ?L     ; legge valore da istanza
                                     ; usando R5
        MOV     #T'$E, R5        ; carica estensione
        ASR     R5               ; R5 <- E/2
L:      MOV     B'(OFS), (DP)+   ; legge word componenti e
        TST     (OFS)+           ; copia
        SOB     R5, L
    .ENDM


    .MACRO T'$STO OFS, B, PS, ?L     ; copia valore su istanza
                                     ; usando R5
        MOV     #T'$E, R5        ; carica estensione
        ASR     R5               ; R5 <- E/2
L:      MOV     (PS)+, B'(OFS)   ; legge word componenti e
        TST     (OFS)+           ; copia
        SOB     R5, L
    .ENDM
 .ENDM
```

Si suppone inoltre che siano definite le macro WORD$VAL, WORD$STO, BYTE$VAL e BYTE$STO che realizzano nei due sensi gli accessi a word e byte e le relative estensioni (costanti).

```
        WORD$E = 2
        BYTE$E = 1

 .MACRO  WORD$VAL    OFS, B, DP
        MOV     B'(OFS), (DP)+
 .ENDM
```

```
.MACRO  WORD$STO    OFS, B, PS
        MOV.    (PS)+, B'(OFS)
.ENDM

.MACRO  BYTE$VAL    OFS, B, DP
        MOVB    B'(OFS), (DP)+
.ENDM

.MACRO  BYTE$STO    OFS, B, PS
        MOVB    (PS)+, B'(OFS)
.ENDM
```

Utilizziamo ora le definizioni introdotte per realizzare quanto richiesto nell'esercizio.

```
type    A = record
            F1, F2:integer; F3:char;
            end

        B = array [-1..3] of A;

        C = record
            F1:B; F2:A; F3:char;
            end
```

puo ` essere tradotto in:

```
        $MRDEF  A, F1, WORD, F2, WORD, F3, BYTE
        $MVDEF  B, <-1>, 3, A
        $MRDEF  C, F1, B, F2, A, F3, BYTE
```

che produce, tra l'altro, le seguenti definizioni:

```
        A$E = <WORD$E+WORD$E+BYTE$E+1>/2*2
        B$E = <<3--1+1>*A$E+1>/2*2
        C$E = <B$E+A$E+BYTE$E+1>/2*2
```

Si lascia al lettore la determinazione del rimanente codice espanso per queste chiamate. Le dichiarazioni:

```
var     A1:A; B1:B; C1, C2:C;
```

possono essere tradotte in:

```
        A$DCL   A1
        B$DCL   B1
        C$DCL   C1
        C$DCL   C2
```

ed espanse in:

```
A1:     .BLKB   A$E         ; = 6
B1:     .BLKB   B$E         ; = 36 = 30.
C1:     .BLKB   C$E         ; = 46 = 38.
C2:     .BLKB   C$E
```

La struttura di dati per il tipo C e` rapresentata in figura 2.12.

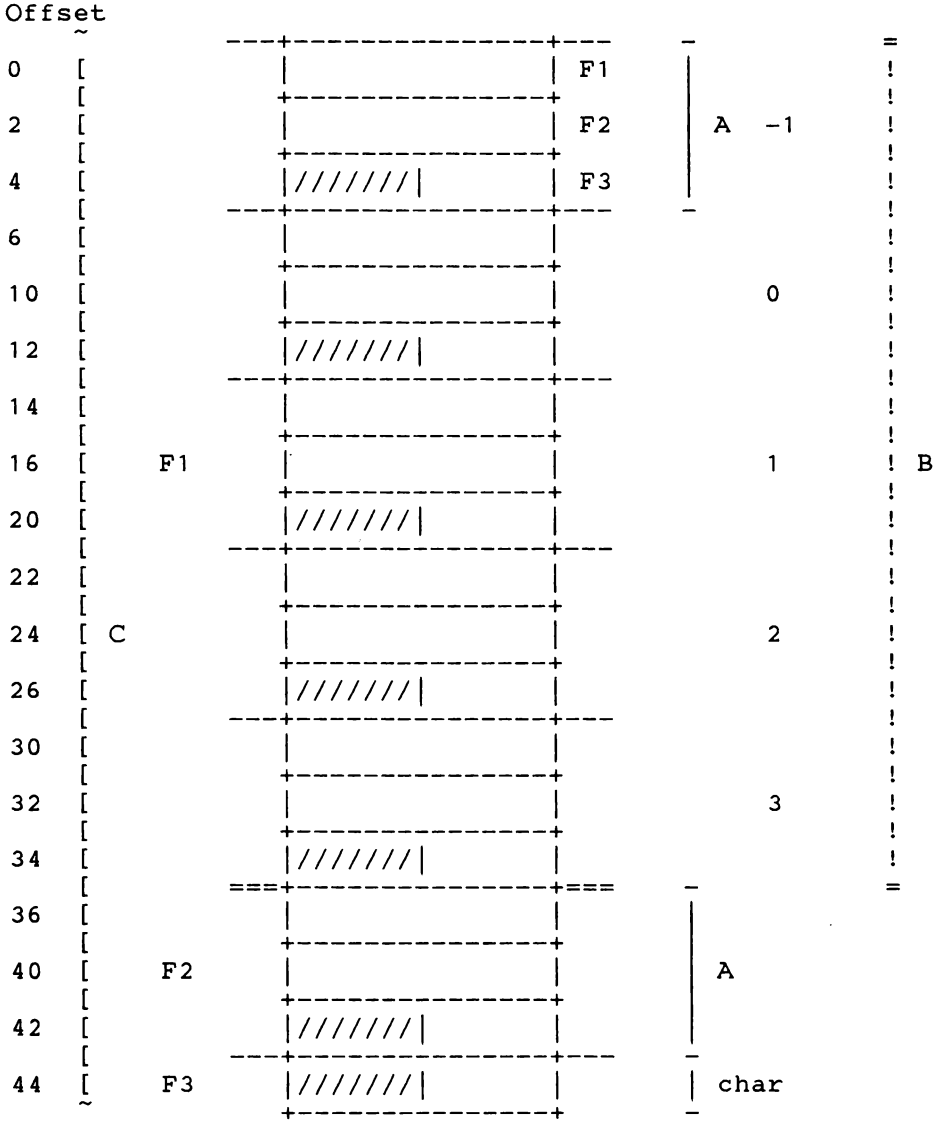


Fig. 2.12

Le due assegnazioni possono essere tradotte nel modo seguente:

```
C1.F1[2].F3 := C2.F3;

        CLR     R2        ; azzera offset
        A$OFS   F3, R2    ; somma offset campo F3
        ADD     #C2, R2   ; calcola puntatore per istanza C2
        MOV     R2, R3    ; salva in R2
        CLR     R2        ; azzera offset
        MOV     #2, R0    ; carica indice
        C$OFS   F1, R2    ; calcola primo offset campo F1
        B$OFS   R0, R2    ; somma secondo offset indice 2
        A$OFS   F3, R2    ; somma terzo offset campo F3
        BYTE$STO    R2, C1, R3   ; copia byte
```

che viene espansa in:

```
        CLR     R2
;        A$OFS   F3, R2
        ADD     #WORD$E+WORD$E, R2
        ;
        ADD     #C2, R2
        MOV     R2, R3
        CLR     R2
        MOV     #2, R0
;        C$OFS   F1, R2
        ; espansione nulla
;        B$OFS   R0, R2
        SUB     #-1, R0
        MOV     R0, -(SP)
        MOV     #A$E, R0
        MUL     (SP)+, R0
        ADD     R0, R2
        ;
;       A$OFS   F3, R2
        ADD     #WORD$E+WORD$E, R2
        ;
;       BYTE$STO    R2, C1, R3
        MOVB        (R3)+, C1(R2)
        ;
```

```
C2.F2 := C1.F1[0];

        CLR     R0      ; azzera indice
        CLR     R2      ; azzera offset
        C$OFS   F1, R2  ; somma offset campo F1
        B$OFS   R0, R2  ; somma secondo offset indice 0
        ADD     #C1, R2 ; calcola puntatore per istanza C1
        MOV     R2, R3  ; salva in R2
        CLR     R2      ; azzera offset
        C$OFS   F2, R2  ; calcola offset campo F2
        A$STO   R2, C2, R3  ; copia elemento tipo A
```

che viene espansa in:

```
        CLR     R0
        CLR     R2
;        C$OFS   F1, R2
        ; espansione nulla
;        B$OFS   R0, R2
        SUB     #-1, R0
        MOV     R0, -(SP)
        MOV     #A$E, R0
        MUL     (SP)+, R0
        ADD     R0, R2
        ;
        ADD     #C1, R2
        MOV     R2, R3
        CLR     R2
;        C$OFS   F2, R2
        ADD     #B$E, R2
```

```
;         A$STO   R2, C2, R3
          MOV     #A$E, R5
          ASR     R5
30000$:   MOV     (R3)+, C2(R2)
          TST     (R2)+
          SOB     R5, 30000$
```

Si noti che, nella definizione delle macro, si e` anticipato al tempo di assemblaggio il calcolo di tutto cio` che era noto e quindi calcolabile a quel tempo, rimandando al tempo di esecuzione solo i calcoli non anticipabili. Se il processore usato non dispone dell'istruzione MUL, essa deve essere sostituita dalla chiamata ad una subroutine che effettui tale operazione. Si ricordi che tale istruzione carica anche il byte piu` significativo del prodotto nel registro successivo a quello specificato solo se quest'ultimo e` di indice pari (R0, R2, R4, [R6]).

Si suggerisce di modificare le macro in modo da effettuare controlli sull'accesso corretto ai dati (limiti per gli indici, nomi corretti per i campi) e, inoltre, di generalizzare la macro $MRDEF in modo da poter definire tipi di record con numero qualsiasi di campi.
++++++++

2.10 - Esercizi riassuntivi

Esercizio 2.21

Tradurre in linguaggio assembly la seguente istruzione in linguaggio ad alto livello:

```
    if (X = 0) then
        X := Y
    else
        Y := X;
--------
                                    . = 1000
                                   X = 2000
                                   Y = 3000

001000  005767  000774           COMP:   TST     X
                                       ; test su M[X]
001004  001004                         BNE     ELSE
                                       ; se <> 0 salta a ELSE
001006  016767  001766  000764   THEN:   MOV     Y, X
                                       ; parte THEN, M[X] <- M[Y]
001014  000403                         BR      GOON
001016  016767  000756  001754   ELSE:   MOV     X, Y
                                       ; parte ELSE, M[Y] <- M[X]
001024                             GOON:   ...
```

Si noti che, avendo utilizzato salti e indirizzamenti relativi il codice prodotto risulta rilocabile senza modifiche, essendo del tutto indipendente dalla posizione di caricamento. Cio` non avviene, in generale, se si utilizzano salti ed indirizzamenti assoluti.
++++++++

Esercizio 2.22

Sommare due vettori (A1, A2) di dimensione N con elementi da 16 bit: il risultato viene memorizzato in un terzo vettore (A3) dello stesso tipo.

Vediamo in questo esempio l'utilizzazione delle varie modalita` di indirizzamento. Si supponga, come primo caso, che gli indirizzi iniziali dei vettori siano contenuti nelle variabili A1, A2, A3 che fungono da puntatori mentre nella variabile N sia contenuta la dimensione comune.

```
ALFA:   MOV     N, R0           ; ind. relativo e di registro per
                                ; il caricamento del valore di conteggio
        MOV     @#A1, R1        ; ind. assoluto e registro per
        MOV     @#A2, R2        ; il caricamento dei puntatori
        MOV     @#A3, R3
L1:     MOV     (R1)+, (R3)     ; ind. autoincr. e indiretto
        ADD     (R2)+, (R3)+    ; ind. autoincr. per somma
        DEC     R0              ; ind. registro
        BGT     L1              ; cicla se >= 0
```

La seguente variante utilizza direttamente A1, A2 e A3 come puntatori e N come contatore, modificandone il contenuto:

```
BETA:   MOV     #2, R3          ; ind. immediato e registro per la
                                ; costante di incremento
L2:     MOV     @A1, @A3        ; ind. relativo differito
        ADD     @A2, @A3
        ADD     R3, A1          ; ind. relativo e registro per
        ADD     R3, A2          ; l'aggiornamento dei puntatori
        ADD     R3, A3
        DEC     @#N             ; ind. assoluto per decr. contatore
        BGT     L2
```

Una terza variante utilizza un registro come puntatore relativo e un registro come contatore con una diversa valutazione di fine ciclo; inoltre V1, V2 e V3 rappresentano effettivamente gli indirizzi iniziali dei tre vettori e non piu` variabili puntatore:

```
GAMMA:  CLR     R0              ; ind. registro, azzera punt. relativo
        MOV     @#N, R1         ; ind. assoluto e registro per contatore
L3:     MOV     V1(R0), V3(R0)  ; ind. indice
        ADD     V2(R0), V3(R0)
        TST     (R0)+           ; ind. autoincremento per
                                ; aggiornare il puntatore relativo
        SOB     R1, L3          ; ind. registro e assoluto per il
                                ; ciclo
```

Si possono ad esempio fare le seguenti dichiarazioni:

```
        SIZ = dimensione vettori
N:      .WORD   SIZ
A1:     .WORD   #V1
A2:     .WORD   #V2
A3:     .WORD   #V3
V1:     .BLKW   SIZ
V2:     .BLKW   SIZ
V3:     .BLKW   SIZ
```

Si suggerisce al lettore di effettuare la traduzione in codice macchina delle tre varianti proposte e di fare la traccia dell'esecuzione.
++++++++

Esercizio 2.23

Si scriva un segmento di codice in grado di trasformare il registro R0 nella sua versione speculare bit a bit (15->0; 14->1; ; ... ; 1->14; 0->15).

Chiamando b il contenuto originario di R0:

$R0 = b = b_{15}b_{14}...b_1b_0$

e b' il contenuto di R0 ottenuto con successive operazioni di ASL, detto $0<=i<=15$ il passo generico di scorrimento, per b'(i) e C(i), valore di R0 e del flag carry rispettivamente, dopo il passo i-esimo, vale la seguente:

$C(i) = b_{15-i}$
$b'_{15-k}(i) = b_{14-i-k}$ per ogni $0<=k<=14-i$
$b'_p(i) = 0$ per ogni $0<=p<=i$

Se ad ogni passo si effettua anche un ROR su R1, detto d'(i) il valore di R1 dopo ogni passo e d il valore originale, si ha:

$d'_{15-j}(i) = C(i-j)$ per ogni $0<=j<=i$
$d'_{14-i-m}(i) = d_{15-m}$ per ogni $0<=m<=14-i$

Pertanto dopo il passo i=15 si ha:

$d'_{15-j}(15) = C(15-j)$ per ogni $0<=j<=15$ <==>

$d'_h(15) = C(h) = b_{15-h}$ per ogni $0<=h<=15$

che e` la versione speculare di b.

```
INVB:   MOV     #16., R0        ; carica contatore
ONEB:   ASL     R0
        ROR     R1
        SOB     R2, ONEB
        MOV     R1, R0
```
++++++++

Esercizio 2.24

Descrivere l'effetto sui bit di condizione nei seguenti gruppi di istruzioni:

```
    a)              b)
1)MOV #0, Rn      CLR Rn

    a)              b)          c)
2)TST (Rn)+       INC Rn      ADD #2, Rn
                  INC Rn
```

```
    a)              b)          c)
3)ASL Rn            ADD Rn, Rn  CLC
                                ROL Rn

    a)              b)
4)TST Rn            CMP Rn, #0

    a)              b)
5)NEG Rn            COM Rn
                    INC Rn

    a)              b)
6)RTS PC            MOV (SP)+, PC

    a)              b)
7)SUB Rn, Rn'       NEG Rn
                    ADD Rn, Rn'
                    NEG Rn

    a)              b)
8)JSR PC, ALFA      MOV #.+8, -(SP)
                    JMP ALFA

    a)              b)              c)          d)      e)
9)NOP     TST Rn    MOV Rn, Rn      ADD #0, Rn  COM Rn  SWAB Rn
                                                COM Rn  SWAB Rn
    f)      g)      h)              i)          l)
  NEG Rn  INC Rn    CMP Rn, Rn'     BIT Rn, Rn' BR .+2
  NEG Rn  DEC Rn

    a)              b)
10)   TST @#A       TST A
--------
```

1) azzerano Rn; a) non modifica C , b) modifica tutti i flag

2) incrementano di 2 Rn; a) modifica i flag sulla base del contenuto del word puntato; b) non modifica C; c) modifica i flag sulla base del nuovo valore di Rn

3) raddoppiano Rn; equivalenti

4) pone V=C=0 e N e Z in base a Rn; equivalenti

5) il contenuto viene sostituito dal suo complemento a 2; i flag N e Z vengono posti in base al risultato e V=1 se risultato = -2^{15}; a) pone C=1 se risultato <> 0; b) pone C=1 comunque

6) caricano PC da top dello stack; b) modifica N e Z in base a nuovo PC e pone V=0

7) effettuano Rn' <- Rn'- Rn; b) non imposta i flag in base al risultato ma in base all'ultima operazione NEG.

8) analogo a 6)

9) non modificano alcun registro o alcuna locazione ma **solo flag**; a) nemmeno i flag; b) V=C=0 e N e Z in base a Rn; c) come b); d) C=1 V=0 e N e Z in base a Rn; e) V=C=0 e N e Z in base a Rnl (interpretato come compl. a 2 a 8 bit); f) C=1 se Rn<>0; V=1 se Rn=-2^{15}; N e

Z in base a Rn; g) V=1 se Rn=2^{15}-1; N e Z in base a Rn; h) modifica tutti i flag; i) pone V=0 e N e Z in base al risultato; l) non modifica i flag

10) del tutto equivalenti: sono solo codificate in modo diverso.
++++++++

Esercizio 2.25

Descrivere l'effetto delle seguenti istruzioni:

```
1)  TST #0                2)  TST #-1
3)  TST #1                4)  CMP DD, DD
5)  ADD DD, DD            6)  SUB DD, DD
7)  BIT #0, DD            8)  BIC DD, DD
9)  BIC #0, DD            10) BIS DD, DD
11) BIS #0, DD            12) BIS #-1, DD
13) XOR R0, R0            14) MOV (Rn), Rn
15) TST (Rn)+             16) TST -(Rn)
17) TST (Rn) o TST  0(Rn)
18) MOV SS, -(Rn)         19) MOV (Rn)+, DD
20) MOV (Rn), -(Rn)       21) MOV 2*X(Rn), -(Rn)
22) MOV @(Rn)+, -(Rn) oppure     MOV @0(Rn), (Rn)
23) MOV (Rn)+, @(Rn)+     24) <unaryop> (Rn)  o <unaryop> 0(Rn)
25) <binaryop> (Rn)+, (Rn)
--------
```

```
1)  TST #0       -> NZVC = 04
2)  TST #-1      -> NZVC = 10
3)  TST #1       -> NZVC = 00
4)  CMP DD, DD   -> come TST #0
5)  ADD DD, DD   -> DD <- DD*2
6)  SUB DD, DD   -> DD <- 0; NZVC = 04
7)  BIT #0, DD   -> NZV = 2, C non modificato
8)  BIC DD, DD   -> DD <- 0; NZV = 2
9)  BIC #0, DD   -> come TST DD ma C non modificato
10) BIS DD, DD   -> come 9)
11) BIS #0, DD   -> come 10)
12) BIS #-1, DD -> DD <- -1  NZV = 4
13) XOR R0, R0   -> come CLR R0 ma C non modificato
```

Molte delle istruzioni proposte utilizzano la modalita` ad autoincremento o autodecremento; ricordando il significato di struttura LIFO (Last In First Out), indicheremo con USP un registro, diverso da PC, che viene utilizzato come *user stack pointer* cioe` come puntatore ausiliario ad una struttura siffatta. Incidentalmente, USP potra` coincidere con SP.

14) MOV (Rn), Rn -> dereferencing;
se Rn=PC, il word che segue diventa indirizzo assoluto di salto <=> JMP @#A;
se Rn=USP, lo USP viene aggiornato con il valore sul top dello stack. Volendo definire per questa e per successive istruzioni un pseudo-codice operativo che ricordi la particolare funzione svolta, possiamo chiamare questa operazione:

```
prest USP
```

atta a ripristinare un valore per USP precedentemente salvato sullo stack.

15) TST (Rn)+ -> incrementa di 2 Rn (trascurando flag);
se Rn=PC si ha come noto la modalita` di indirizzamento immediato;
se Rn=USP si ha la valutazione del top e successivo pop, per cui potremmo chiamare questa operazione:

```
tdrop USP
```

16) TST -(Rn) -> decrementa di 2 Rn (trascurando i flag)
se Rn=PC corrisponde ad un loop su se stessa
se Rn=USP, effettua un push di una locazione non inizializzata:

```
alloc USP
```

17) TST (Rn) o TST 0(Rn) -> valutazione differita
se Rn=USP corrisponde ad una valutazione del top:

```
tstop USP
```

18) MOV SS, -(Rn) -> se Rn=USP esegue push di SS:

```
push SS, USP
```

19) MOV (Rn)+, DD -> se Rn=USP esegue pop su DD:

```
pop USP, DD
```

20) MOV (Rn), -(Rn) -> se Rn=USP copia il top dello stack:

```
dup USP
```

21) MOV 2*X(Rn), -(Rn) -> se Rn=USP copia sul top un word all'interno dello stack di indice X (0<=X<=size(stack)-1):

```
pick X, USP
```

22) MOV @(Rn)+, -(Rn) oppure MOV @0(Rn), (Rn) -> se Rn=USP effettua il "dereferencing" sul top:

```
fetch USP
```

23) MOV (Rn)+, @(Rn)+ -> se Rn=USP esegue la memorizzazione del valore contenuto nel top all'indirizzo contenuto nella locazione successiva al top; entrambi vengono eliminati dallo stack:

```
store USP
```

24) <unaryop> (Rn) -> se Rn=USP l'operazione unaria indicata viene effettuata sul top:

```
CLR (Rn)    ->  clrtop USP
COM (Rn)    ->  comtop USP
INC (Rn)    ->  inctop USP
DEC (Rn)    ->  dectop USP
NEG (Rn)    ->  negtop USP
ROR (Rn)    ->  rortop USP
ROL (Rn)    ->  roltop USP
ASR (Rn)    ->  asrtop USP
```

```
ASL (Rn)     ->  asltop USP
SWAB (Rn)    ->  swatop USP
ADC (Rn)     ->  adctop USP
SBC (Rn)     ->  sbctop USP
```

25) <binaryop> (Rn)+, (Rn) -> se Rn=USP i due valori in cima allo stack vengono sotituiti da un unico valore dato dalla operazione binaria effettuata sui valori anzidetti. Alcuni esempi:

```
MOV      (Rn)+, (Rn) ->  down USP
ADD      (Rn)+, (Rn) ->  sum USP
SUB      (Rn)+, (Rn) ->  dif USP ((top)-(top+1))
BIT      (Rn)+, (Rn) ->  bitop USP
BIC      (Rn)+, (Rn) ->  bictop USP
BIS      (Rn)+, (Rn) ->  bistop USP
```

Per l'istruzione di confronto CMP, forse e` opportuno eliminare entrambi i valori:

```
CMP      (Rn)+, (Rn)+    ->  cmptop USP
```

Si lascia al lettore di definire tutte le pseudo-operazioni sottoforma di macro in modo da simulare una generica macchina a stack, per la quale poter scrivere programmi nel nuovo linguaggio.
++++++++

Esercizio 2.26

Scrivere 2 subroutine che consentono di copiare un vettore di byte in un vettore di word mediante estensione del segno, e viceversa trascurando il byte piu` significativo; nel secondo caso si segnali la eventuale perdita di precisione in almeno un elemento del vettore.

Si supponga di ricevere in R1 l'indirizzo del vettore di byte, in R3 quello del vettore di word e in R2 il numero di elementi da trasferire.

```
BY2WO:  SAVE     <R0, R1, R2, R3>
LP1:    MOVB     (R1)+, R0     ; estensione del segno
        MOV      R0, (R3)+     ; copia su vettore word
        SOB      R2, LP1       ; ciclo su numero di byte
        RESTORE  <R3, R2, R1, R0>
        RTS      PC

WO2BY:  SAVE     <R0, R1, R2, R3, R4>
        CLR      -(SP)         ; variabile temporanea sul top
LP2:    MOV      (R3)+, R0     ; carica word da vettore
        MOVB     R0, R4        ; copia con estensione del segno
        CMP      R0, R4        ; se eguali, non perdita
        BEQ      OK
        BIS      #1, (SP)      ; perdita di precisione, bit 0 del
                               ; della variabile temporanea = 1
OK:     MOVB     R0, (R1)+     ; copia su vettore byte
        SOB      R2, LP2       ; ciclo su nuemro byte
        ROR      -(SP)         ; C=1 se perdita precisione
        RESTORE  <R4, R3, R2, R1, R0>
        RTS      PC
```

Nelle due subroutine si sono per brevita` utilizzate, come sara` fatto anche in seguito, le macro SAVE e RESTORE che salvano sullo stack una lista di registri.
++++++++

Esercizio 2.27

Si scriva una subroutine che riceve in R0 e R1 due cifre decimali (0..9) e restituisce in R2 (cifra di peso 1) e R3 (cifra di peso 10) la rappresentazione in base 10 del prodotto ed un'altra subroutine che, ricevendo in R2 ed R3 un analogo numero decimale di 2 cifre, restituisce negli stessi registri il complemento a 10 del numero rappresentato.

Per il prodotto vengono proposte due soluzioni diverse. Nella prima il prodotto viene eseguito come somma multipla per se stesso di R0, R1 volte. Conviene quindi definire una subroutine di servizio (SUMDEC) che esegue la somma di una cifra (R0) ad un numero a 2 cifre (R3-R2) ricordando che, in queste ipotesi:

$a+b = a_0 + (b_0+b_1*10) = mod(a_0+b_0, 10)+10*(int((a_0+b_0)/10)+b_1)$

$mod(a_0+b_0, 10) = a_0+b_0$ se $0<=a_0+b_0<=9$
$= a_0+b_0-10$ se $a_0+b_0>9$

$int((a_0+b_0)/10 = 0$ se $0<=a_0+b_0<=9$
$= 1$ se $a_0+b_0>9$

```
SUMDEC: ADD     R0, R2    ; a0+b0
ADJUST: CMP     #9., R2
        BGE     NOOV      ; salta se non riporto
        SUB     #10., R2; a0+b0-10
        INC     R3        ; b1+1
NOOV:   RTS     PC

PRDDEC: CLR     R2        ; b=0
        CLR     R3
        TST     R1        ; moltiplicatore = 0 ?
        BEQ     ISZ       ; salta se si`
        SAVE    <R1>
LP1:    JSR     PC, SUMDEC
        SOB     R1, LP1 ; ciclo su moltiplicatore
        RESTORE <R1>
ISZ:    RTS     PC
```

Il secondo metodo si basa su 10 segmenti di codice differenti, corrispondenti agli altrettanti valori del moltiplicatore, attivati mediante jump table. Vengono sfruttate le seguenti relazioni:

```
a*0=0   a*1=a   a*2=asl(a)  a*3=asl(a)+a    a*4=asl(asl(a))
a*5=(a*10)/2=asr(a*10)  a*6=a*5+a   a*7=a*5+a+a
a*8=asl(asl(asl(a)))    a*9=a*10-a

PRDDE2: CLR     R2
        CLR     R3
        ASL     R1
        ASL     R1        ; R1*4
        JMP     JTAB(R1)
```

```
JTAB:   JMP     M0          ; jump table
        JMP     M1
        JMP     M2
        JMP     M3
        JMP     M4
        JMP     M5
        JMP     M6
        JMP     M7
        JMP     M8
        JMP     M9

M1:     MOV     R0, R2      ; * 1
        JMP     M0

M2:     MOV     R0, R2      ; * 2
        ASL     R2
        JSR     PC, ADJUST
        JMP     M0

M3:     MOV     R0, R2      ; * 3
        ASL     R2
        JSR     PC, ADJUST
        JSR     PC, SUMDEC
        JMP     M0

M4:     MOV     R0, R2      ; * 4
        ASL     R2
        JSR     PC, ADJUST
        ASL     R2
        ASL     R3
        JSR     PC, ADJUST
        JMP     M0

M5:     MOV     R0, R3      ; * 5
        ASR     R3
        BCC     CY01
        MOV     #5, R2
CY01:   JMP     M0

M6:     MOV     R0, R3      ; * 6
        ASR     R3
        BCC     CY02
        MOV     #5, R2
CY02:   JSR     PC, SUMDEC
        JMP     M0

M7:     MOV     R0, R3      ; * 7
        ASR     R3
        BCC     CY03
        MOV     #5, R2
CY03:   JSR     PC, SUMDEC
        JSR     PC, SUMDEC
        JMP     M0

M8:     MOV     R0, R2      ; * 8
        ASL     R2
        JSR     PC, ADJUST
        ASL     R2
        ASL     R3
        JSR     PC, ADJUST
```

```
        ASL     R2
        ASL     R3
        JSR     PC, ADJUST
        JMP     M0

M9:     MOV     R0, R3          ; * 9
        JSR     PC, DIFDEC

M0:     ASR     R1              ; * 0
        ASR     R1
        RTS     PC

DIFDEC: SUB     R0, R2
        BGE     OKDIF   ; salta se R2>=R0
        ADD     #10., R2; alla differenza si somma la base
        DEC     R3      ; b1-1
OKDIF:  RTS     PC
```

La routine SUMDEC ha un ingresso alternativo (ADJUST) che permette la correzione su R2 se >=10.

Per il complemento a 10, basta ricordare che:

$c(10, 2, a) = c_1(10, 2, a) + 1$

e il complemento a 9 si ottiene facendo la differenza con 9 di tutte le cifre.

```
CDEC:   TST     R2
        BNE     NOT0
        TST     R3
        BEQ     IS0     ; nop se R2-R3 = 0
NOT0:   SUB     #9., R2 ; c1(10, 2, a)
        NEG     R2
        SUB     #9., R3
        NEG     R3
        INC     R2      ; + 1
        JSR     PC, ADJUST
IS0:    RTS     PC
++++++++
```

Esercizio 2.28

Si scriva un segmento di programma in grado di ricercare il valore massimo tra N (>1) costanti in complemento a 2 in memoria.

La soluzione e` semplice e si commenta da sola.

```
CMAX:
        MOV     #CNUM, R0       ; numero di costanti da
                                ; valutare
        DEC     R0              ; N-1 confronti
        MOV     #DATA, R2       ; indirizzo prima costante
        MOV     (R2), R5        ; inizializza TEMPMAX

LOOP:
        TST     (R2)+           ; <=> R2 <- R2+2
        CMP     R5, (R2)        ; confronto con TEMPMAX
```

```
        BGE     NEXT            ; e` maggiore di TEMPMAX?
        MOV     (R2), R5        ; si, sostituzione di TEMPMAX
NEXT:   DEC     R0              ; decrementa contatore
        BNE     LOOP            ; loop
        HALT                    ; in R5 il massimo

        CNUM = 5
DATA:   .WORD 403, 102304, 23704, 7702, 176337
++++++++
```

Esercizio 2.29

Si definisca un protocollo di attivazione di procedure che preveda:

- il caricamento sullo stack di una successione di parametri di ingresso (passati per valore e/o per indirizzo);
- l'allocazione sullo stack di spazio per variabili locali alla procedura;
- l'accesso agli oggetti allocati sullo stack mediante Frame Pointer (FP).

Si definiscano poi alcune macro per la realizzazione del protocollo e si esemplifichi l'uso di queste, definendo una procedura che, ricevendo come parametri d'ingresso due operandi, uno passato per valore e uno per indirizzo, restituisca la somma come nuovo valore dell'operando passato per indirizzo.

Si supponga di utilizzare R5 come FP. Una configurazione dello stack all'ingresso della procedura che realizzi quanto richiesto e` presente in fig. 2.13.

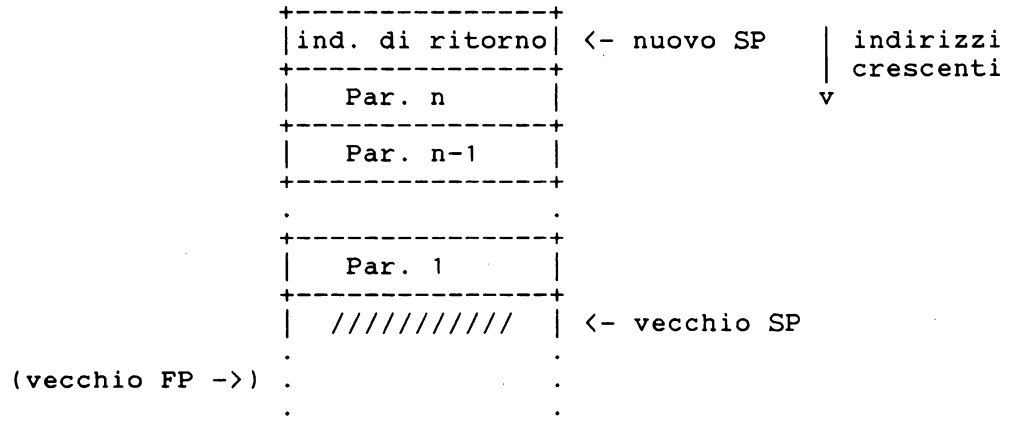


Fig. 2.13

All'ingresso della procedura vi e` il codice di allocazione del frame ottenuto salvando sullo stack il vecchio valore di FP ed assegnando come nuovo valore il valore corrente di SP; vi e` poi il codice di allocazione delle eventuali variabili locali alla procedura, ottenendo la configurazione di fig. 2.14. A questo punto, l'accesso alle variabili locali e ai parametri puo` avvenire utilizzando FP come registro indice. Piu` precisamente la posizione della k-esima variabile locale e` data da FP-2*k, mentre quella dell'i-

esimo parametro e` data da FP+4+(n-i)*2 essendo n il numero complessivo di parametri.

Alla terminazione, la procedura deve deallocare il proprio frame: questo e` ottenuto ripristinando il vecchio valore di FP, salvato in M[FP], ed anche il valore che SP aveva all'ingresso (cioe` FP+2). Al ritorno la procedura chiamante dovra` deallocare dallo stack i parametri, il cui numero e` ad essa noto.

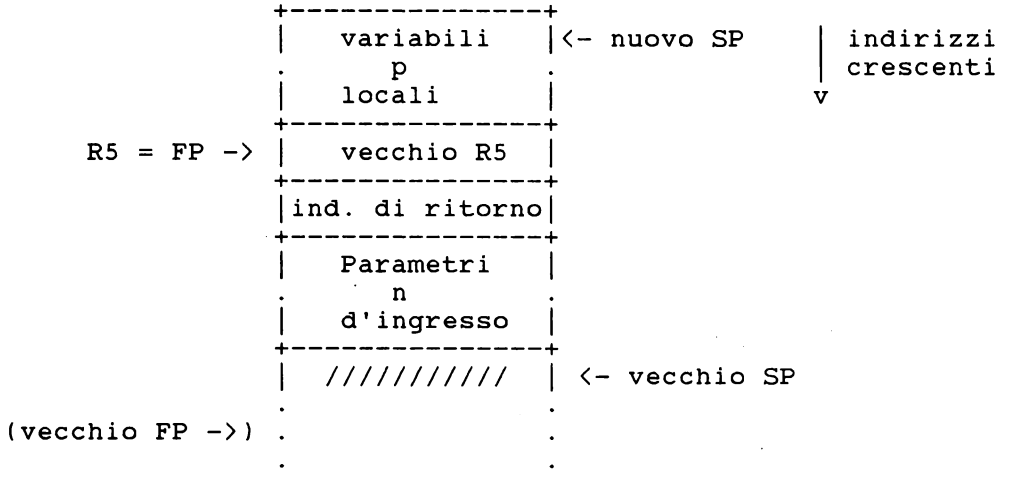


Fig. 2.14

Le macro che allocano e rilasciano il frame sono le 3 seguenti:

```
.MACRO  CALL PROC PARL          ; lista parametri
 .IF NB, <PARL>
   $NPAR = 0
   .IRP P, <PARL>
       $NPAR = $NPAR+2          ; contatore parametri
       MOV     P, -(SP)         ; carica parametri su stack
   .ENDR
 .ENDC
       JSR     PC, PROC         ; chiama procedura
       ADD     #$NPAR, SP       ; dealloca parametri
.ENDM

.MACRO  PROC LOCN               ; num. var. locali
       MOV     R5, -(SP)        ; salva R5
       MOV     SP, R5           ; definisce R5 come FP
 .IF NE, LOCN
       ADD     #-2*LOCN, SP     ; alloca var. locali
 .ENDC
.ENDM

.MACRO  RETURN
       MOV     R5, SP           ; ripristina SP
       MOV     (SP)+, R5        ; ripristina R5
       RTS     PC
.ENDM
```

L'accesso ai parametri e alle variabili locali puo` essere agevolato dalla definizione di macro del tipo:

```
 .MACRO  PVAL P, N, D, OPC       ; operaz. con sorgente param.
        OPC 4+<<N-P>*2>(R5), D
 .ENDM

 .MACRO  PREF P, N, D, OPC       ; operaz. con sorgente param. diff.
        OPC @4+<<N-P>*2>(R5), D
 .ENDM

 .MACRO  PSTO S, N, P, OPC       ; operaz. con dest. param. (diff.)
        OPC S, @4+<<N-P>*2>(R5)
 .ENDM

 .MACRO  LVAL L, D, OPC          ; operaz. con sorgente var. locale
        OPC -2*L(R5), D
 .ENDM

 .MACRO  LSTO S, L, OPC          ; operaz. con dest. var. locale
        OPC S, -2*L(R5)
 .ENDM
```

dove P, L e N sono rispettivamente l'indice di un parametro (1..n), di una variabile locale (1..p) e il numero di parametri.

Nell'esempio richiesto:

```
C:  CALL    F  <VAR1, #VAR2>
    MOV     VAR2, SUMV       ; preleva risultato

F:  PROC    1                ; una variabile locale
    LSTO    R0, 1, MOV       ; salva registro
    PVAL    1, 2, R0, MOV    ; carica valore 1
    PREF    2, 2, R0, ADD    ; somma valore 2 via riferimento
    PSTO    R0, 2, 2, MOV    ; memorizza somma
    LVAL    1, R0, MOV       ; ripristina R0
    RETURN

    .CSECT  DAT
VAR1:   .WORD   5
VAR2:   .WORD   7
SUMV:   .WORD   1
```

che viene espansa in:

```
C:      MOV     VAR1, -(SP)
        MOV     #VAR2, -(SP)
        JSR     PC, F
        ADD     #4, SP
        MOV     VAR2, SUMV

F:      MOV     R5, -(SP)
        MOV     SP, R5
        ADD     #-2*1, SP
        MOV     R0, -2*1(R5)
        MOV     4+<<2-1>*2>(R5), R0
        ADD     @4+<<2-2>*2>(R5), R0
        MOV     R0, @4+<<2-2>*2>(R5)
        MOV     -2*1(R5), R0
        MOV     R5, SP
        MOV     (SP)+, R5
        RTS     PC
```

Si cerchi di modificare le macro in modo da poter accedere simbolicamente a parametri di ingresso, d'uscita e alle variabili locali (sugg.: si possono definire costanti simboliche omonime ai parametri ecc., il cui valore e` pari agli indici degli oggetti associati).
++++++++

Esercizio 2.30

Si supponga che, connesso alla linea di interruzione PF (Power Fail VA=24, non mascherabile) sia collegato un tasto la cui pressione genera una interruzione. Si scriva un programma che, dopo aver inizializzato il vettore d'interruzione, esegua come attivita` di base l'aggiornamento di un contatore corrispondente a sei cifre ottali complete e che, a fronte di interruzione, visualizzi, utilizzando la subroutine PUT che riceve un carattere in R0, la cifra piu` significativa del contatore.

Il contatore sara` costituito, per le 5 cifre meno significative, dal registro R1 e per la sesta dai tre bit meno significativi di R0. Il programma si commenta da se`.

```
        KEYVEC = 24      ; vector address di PF
        KEYPS = 340      ; non mascherabile = prior. Max

MAIN:   MOV     #KEYVEC, R0      ; carica vector address
        MOV     #KEYIR, 0(R0)    ; carica vettore
        MOV     #KEYPS, 2(R0)
L1:     CLR     R0               ; inizializza parte alta
L2:     CLR     R1               ; inizializza parte bassa
L3:     INC     R1
        CMP     R1, #100000      ; raggiunto limite 5 cifre
                                 ; ottali?
        BLO     L3               ; no, cicla
        INC     R0               ; incrementa parte alta
        CMP     R0, #8.          ; raggiunto limite ultima
                                 ; cifra ?
        BLT     L2               ; no, init parte bassa
        BR      L1               ; si, da capo

KEYIR:                           ; routine di servizio
        MOV     R0, -(SP)        ; salva registro
        ADD     #'0, R0          ; ottale->ASCII
        JSR     PC, PUT          ; output
        MOV     (SP)+, R0        ; recupera registro
        RTI                      ; ritorno da interruzione
++++++++
```

CAPITOLO 3

ESEMPI DI SUBROUTINE

In questo capitolo vengono realizzate, sottoforma di esercizi, un certo numero di subroutine di uso generale con il duplice scopo di fornire esempi di programmazione in linguaggio assembly e un insieme incrementale di subroutine da utilizzare successivamente alla stregua di istruzioni ad alto livello. Il capitolo viene diviso per argomenti. Il codice di ciascuna subroutine viene preceduto da una intestazione di commento che evidenzia il protocollo di chiamata e fornisce una breve descrizione. In questa le notazioni Rn.l e Rn.h rappresentano rispettivamente il byte meno e piu` significativo del registro Rn mentre i bit di condizione sono citati con la simbologia convenzionale.

3.1 - Gestione delle stringhe di caratteri

Nella maggior parte dei casi, un compilatore per linguaggio di alto livello possiede una libreria di routine per la gestione delle stringhe di caratteri che spesso risulta conveniente, per ragioni di efficienza, scrivere in assembly. Una stringa e` una successione (logica) di caratteri codificati di lunghezza qualsiasi ma finita. D'ora in avanti si adotteranno le seguenti convenzioni:

- il codice di rappresentazione dei caratteri e` quello ASCII (7 bit per carattere);

- i caratteri di una stringa sono memorizzati in locazioni contigue (byte);

- la lunghezza non e` memorizzata all'interno della stringa ma e` deducibile dalla presenza di un carattere speciale (EOS) che termina la stringa. Pertanto, se len(s) e` la lunghezza della stringa s, il numero totale di caratteri che la formano e` len(s)+1. La stringa nulla e` costituita dal solo carattere EOS. Poiche` EOS non puo` essere presente all'interno della stringa, si preferisce utilizzare per esso un carattere di controllo: verra` adottato il carattere NUL (codice ASCII = 0) che costituisce una convenzione diffusa.

E` in generale compito del programmatore verificare che, durante operazioni di scrittura su stringhe non si ecceda lo spazio massimo riservato come contenitore della stessa.

Esercizio 3.1

Si realizzino alcune subroutine che siano in grado di stabilire se un carattere appartiene ad una delle seguenti classi:

- cifra binaria (codice ASCII 60..61)

- cifra ottale (c.A. 60..67)

- cifra decimale (c.A. 60..71)

- cifra esadecimale (c.A. 60..71, 101..106, 141..146)

- lettera maiuscola (c.A. 101..132)

- lettera minuscola (c.A. 141..172)

- simbolo operatore (c.A. 41..57, 72..100, 133..140, 173..176)

- carattere separatore (spazio, tab, vert. tab, carriage return, line feed, form feed) (c.A. 11..15, 40)

- carattere di controllo non separatore (c.A. 0..10, 16..37, 177)

Le subroutine che seguono tengono conto che le classi richieste sono composte di unioni di insiemi di caratteri che hanno codifiche contigue. Sono considerati simboli operatoriali tutti i simboli stampabili che non appartengano ad altre classi. Il carattere DEL e` considerato carattere di controllo. Con l'occasione vengono anche definite le subroutine di conversione di un carattere da maiuscolo a minuscolo e viceversa.

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   ISLOLT  verifica carattere lettera minuscola              ;
; DESCRIZIONE                                                 ;
;   Stabilisce se il carattere fornito e` nel range 'a..'z.   ;
; INTERFACCIA                                                 ;
;   JSR PC, ISLOLT                                            ;
;                                                             ;
;   R0.l    input   carattere da verificare                   ;
;   C       output  = 1 se lettera minuscola                  ;
; USA                                                         ;
;   --                                                        ;
;-------------------------------------------------------------;
ISLOLT:
        CMPB    #'a-1, R0
        BHIS    ISL1            ; salta se < 'a, C=0
        CMPB    R0, #'z+1       ; C=0 se > 'z
ISL1:   RTS     PC

;-------------------------------------------------------------;
; NOME                                                        ;
;   ISUPLT  verifica carattere lettera maiuscola              ;
; DESCRIZIONE                                                 ;
;   Stabilisce se il carattere fornito e` nel range 'A..'Z.   ;
; INTERFACCIA                                                 ;
```

```
;   JSR PC, ISUPLT                                                  ;
;                                                                   ;
;   R0.1     input   carattere da verificare                        ;
;   C        output  = 1 se lettera maiuscola                       ;
; USA                                                               ;
;   --                                                              ;
;-------------------------------------------------------------------;
ISUPLT:
        CMPB    #'A-1, R0
        BHIS    ISU1            ; salta se < 'A, C=0
        CMPB    R0, #'Z+1       ; C=0 se > 'Z
ISU1:   RTS     PC

;-------------------------------------------------------------------;
; NOME                                                              ;
;   ISESA, ISDEC, ISOCT, ISBIN                                      ;
;       routine di verifica per cifre esadec., dec. ecc.            ;
; DESCRIZIONE                                                       ;
;   Entry point differenziati nella stessa subroutine per           ;
;   la verifica che un carattere ASCII sia una cifra                ;
;   esadecimale, decimale, ottale o binaria.                        ;
; INTERFACCIA                                                       ;
;   JSR PC, ISESA o altre                                           ;
;                                                                   ;
;   R0.1     input   cifra da verificare                            ;
;   C        output  = 1 se verifica corretta                       ;
; USA                                                               ;
;   Subr: LO2UPC                                                    ;
;-------------------------------------------------------------------;
ISESA:
        MOV     R0, -(SP)       ; salva carattere
        JSR     PC, LO2UPC      ; minuscolo -> maiuscolo
        CMPB    R0, #'F+1
        BHIS    ISHL1           ; salta se >'F, C = 0
        CMPB    #'A-1, R0
        MOV     (SP)+, R0
        BLO     ISLL1           ; salta se >= 'A, C = 1
ISDEC:
        CMPB    R0, #'9+1
        BLO     ISLN1           ; salta se <= '9, altrimenti
                                ; C = 0
ISHN1:  RTS     PC
ISOCT:
        CMPB    R0, #'8
        BLO     ISLN1           ; salta se <= '7, altrimenti
        RTS     PC              ; C = 0
ISBIN:
        CMPB    R0, #'2
        BHIS    ISHN1           ; salta se > '1, C = 0
ISLN1:  CMPB    #'0-1, R0       ; C = *
        RTS     PC

ISHL1:  MOV     (SP)+, R0       ; ripristina registro,
ISLL1:  RTS     PC


;-------------------------------------------------------------------;
; NOME                                                              ;
;   ISOPER  verifica se il carattere e` simbolo operatore           ;
; DESCRIZIONE                                                       ;
```

```
;   Routine di verifica di appartenenza di un carattere ad   ;
;   uno dei seguenti insiemi:                                 ;
;   '! .. '/,                                                 ;
;   ': .. '@,                                                 ;
;   '[ .. '`,                                                 ;
;   '{ .. '~.                                                 ;
; INTERFACCIA                                                 ;
;   JSR PC, ISOPER                                            ;
;                                                             ;
;   R0.l    input   cifra da verificare                       ;
;   C       output  = 1 se verifica corretta                  ;
; USA                                                         ;
;   --                                                        ;
;-------------------------------------------------------------;
ISOPER:
        CMPB    #'!-1, R0
        BHIS    ISOP1           ; salta se < '!, C=0
        CMPB    R0, #'/+1       ; salta se <= 'Z, C=1
        BLO     ISOP1
        CMPB    #':-1, R0
        BHIS    ISOP1           ; salta se < ':, C=0
        CMPB    R0, #'@+1       ; salta se <= '@, C=1
        BLO     ISOP1
        CMPB    #'[-1, R0
        BHIS    ISOP1           ; salta se < '[, C=0
        CMPB    R0, #'`+1       ; salta se <= '`, C=1
        BLO     ISOP1
        CMPB    #'{-1, R0
        BHIS    ISOP1           ; salta se < '{, C=0
        CMPB    R0, #'~+1       ; salta se <= '~, C=1
ISOP1:
        RTS     PC


;-------------------------------------------------------------;
; NOME                                                        ;
;   ISSEP   verifica se il carattere e` un separatore         ;
; DESCRIZIONE                                                 ;
;   Routine di verifica di appartenenza di un carattere alla  ;
;   classe dei separatori (spazio, sep. oriz. e verticali).   ;
; INTERFACCIA                                                 ;
;   JSR PC, ISSEP                                             ;
;                                                             ;
;   R0.l    input   cifra da verificare                       ;
;   C       output  = 1 se verifica corretta                  ;
; USA                                                         ;
;   --                                                        ;
;-------------------------------------------------------------;
        TAB     = 11            ; caratteri di controllo
        CR      = 15
ISSEP:
        CMPB    #TAB-1, R0
        BHIS    ISSP1           ; salta se < TAB, C=0
        CMPB    R0, #CR+1       ; salta se <= CR, C=1
        BLO     ISSP1
        CMPB    #' , R0
        SEC
        BEQ     ISSP1           ; salta se = ' , C=1
        CLC
ISSP1:  RTS     PC
```

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   ISCTRL  verifica se il carattere e` di controllo          ;
; DESCRIZIONE                                                 ;
;   Routine di verifica di appartenenza di un carattere alla  ;
;   classe dei caratteri di controllo non separatori oppure   ;
;   DEL.                                                      ;
; INTERFACCIA                                                 ;
;   JSR PC, ISCTRL                                            ;
;                                                             ;
;   R0.l     input   cifra da verificare                      ;
;   C        output  = 1 se verifica corretta                 ;
; USA                                                         ;
;   --                                                        ;
;-------------------------------------------------------------;
        BS      = 10            ; caratteri di controllo
        SO      = 16
        US      = 37
        DEL     = 177
ISCTRL:
        CMPB    R0, #BS+1       ; salta se <= BS, C=1
        BLO     ISCT1
        CMPB    #SO-1, R0
        BHIS    ISCT1           ; salta se < SO, C=0
        CMPB    R0, #US+1       ; salta se <= US, C=1
        BLO     ISOP1
        CMPB    #DEL, R0
        SEC
        BEQ     ISCT1           ; salta se = DEL, C=1
        CLC
ISCT1:  RTS     PC

;-------------------------------------------------------------;
; NOME                                                        ;
;   LO2UPC  conversione minuscolo maiuscolo di un carattere   ;
; DESCRIZIONE                                                 ;
;   Verifica se il carattere e` lettera ed eventualmente      ;
;   lo trasforma in maiuscolo.                                ;
; INTERFACCIA                                                 ;
;   JSR PC, LO2UPC                                            ;
;                                                             ;
;   R0.l     input   carattere da trasformare                 ;
;   R0.l     output  carattere trasformato                    ;
;   C        output  = 1 se lettera minuscola trasformata     ;
; USA                                                         ;
;   Subr: ISLOLT                                              ;
;-------------------------------------------------------------;
LO2UPC:
        JSR     PC, ISLOLT
        BCC     LUC1            ; salta se non minuscola, C=0
        SUB     #'a-'A, R0      ; conversione
        SEC
LUC1:   RTS     PC

;-------------------------------------------------------------;
; NOME                                                        ;
;   UP2LOC  conversione maiuscolo minuscolo di un carattere   ;
; DESCRIZIONE                                                 ;
;   Verifica se il carattere e` lettera ed eventualmente      ;
;   lo trasforma in minuscolo.                                ;
```

```
; INTERFACCIA                                                    ;
;   JSR PC, UP2LOC                                               ;
;                                                                ;
;   R0.1     input   carattere da trasformare                    ;
;   R0.1     output  carattere trasformato                       ;
;   C        output  = 1 se lettera minuscola trasformata        ;
; USA                                                            ;
;   Subr: ISUPLT                                                 ;
;----------------------------------------------------------------;
UP2LOC:
        JSR     PC, ISUPLT
        BCC     UPC1            ; salta se non maiuscola, C=0
        SUB     #'A-'a, R0      ; conversione
        SEC
UPC1:   RTS     PC
++++++++
```

Esercizio 3.2

Si realizzino le seguenti subroutine:

STLEN	calcola la lunghezza di una stringa
STREV	inverte una stringa su se stessa
STCMP	confronta l'ordine lessicografico di due stringhe
STNCMP	confronta l'ordine lessicografico di due stringhe esaminando al massimo i primi n caratteri
STCPY	copia una stringa in un'altra
STNCPY	copia fino ad n caratteri di una stringa in un'altra
STCAT	concatena una stringa ad un'altra
STNCAT	concatena fino a raggiungere un massimo di n caratteri complessivi una stringa ad un'altra
STSUB	estrae una stringa da un'altra (copia parziale)
STINX	cerca una sottostringa in una stringa

Si adotti la convenzione di utilizzare come parametri R1 puntatore alla stringa sorgente e, se necessario, R2 puntatore alla stringa destinazione.

```
;----------------------------------------------------------------;
; NOME                                                           ;
;   STLEN   lunghezza di una stringa                             ;
; DESCRIZIONE                                                    ;
;   Restituisce la lunghezza della stringa fornita (carattere;
;   EOS escluso).                                                ;
; INTERFACCIA                                                    ;
;   JSR PC, STLEN                                                ;
;                                                                ;
;   R1       input   puntatore stringa fornita                   ;
;   R3       output  lunghezza stringa (len)                     ;
```

```
; USA                                                          ;
;   --                                                         ;
;-------------------------------------------------------------;
        EOS     = 0             ; end of string
                                ; (terminatore di stringa)
STLEN:
        MOV     R1, -(SP)       ; salva registro
        MOV     #-1, R3         ; init len
STL1:   CMPB    #EOS, (R1)+     ; stringa terminata
        BNE     STL1            ; salta se no
        SUB     (SP), R1        ; R1-s = len+1
        ADD     R1, R3          ; len ok
        MOV     (SP)+, R1       ; ripristina registro
        RTS     PC


;-------------------------------------------------------------;
; NOME                                                        ;
;   STREV   rovesciamento di una stringa                      ;
; DESCRIZIONE                                                 ;
;   Produce sopra la stringa fornita una versione speculare   ;
;   della stessa.                                             ;
; INTERFACCIA                                                 ;
;   JSR PC, STREV                                             ;
;                                                             ;
;   R1      input   puntatore stringa fornita                 ;
; USA                                                         ;
;   Subr: STLEN                                               ;
;-------------------------------------------------------------;
STREV:
        SAVE    <R0,R1,R2,R3>   ; salva registri
        JSR     PC, STLEN       ; R3 = len
        MOV     R1, R2          ; puntatore start
        ADD     R3, R1          ; puntatore EOS
        ASR     R3              ; len/2
        BEQ     STRV1           ; salta se len < 2
STRV2:  MOVB    (R2), R0        ; temp <- start
        MOVB    -(R1), (R2)+    ; start <- end
        MOVB    R0, (R1)        ; end <- temp
        SOB     R3, STRV2       ; ciclo su len/2
STRV1:  RESTORE <R3,R2,R1,R0>   ; ripristina registri
        RTS     PC


;-------------------------------------------------------------;
; NOME                                                        ;
;   STCMP   confronto tra stringhe in senso lessicografico    ;
; DESCRIZIONE                                                 ;
;   Restituisce l'informazione di confronto (dell'ordine      ;
;   lessicografico) tra due stringhe.                         ;
; INTERFACCIA                                                 ;
;   JSR PC, STCMP                                             ;
;                                                             ;
;   R1      input   puntatore prima stringa                   ;
;   R2      input   puntatore seconda stringa                 ;
;   Z       output  = 1 se STR1 = STR2                        ;
;   C       output  = 1 se STR1 < STR2                        ;
; USA                                                         ;
;   --                                                        ;
;-------------------------------------------------------------;
```

```
STCMP:
        MOV     R1, -(SP)       ; salva registri
        MOV     R2, -(SP)
STCM1:  CMPB    #EOS, (R1)      ; fine prima stringa ?
        BEQ     STCM2           ; salta se si`
        CMPB    #EOS, (R2)      ; fine seconda stringa ?
        BEQ     STCM3           ; salta se si`
        CMPB    (R1)+, (R2)+    ; non EOS
        BEQ     STCM1           ; cicla se eguali
        MOV     (SP)+, R2       ; ripristina registri
        MOV     (SP)+, R1
        CLZ                     ; Z = 0; C = *
        RTS     PC              ; prologhi diversi
STCM2:  CMPB    #EOS, (R2)      ; STR1 = STR2 ?
        BEQ     STCM4           ; salta se si`, Z=1; C=0
        MOV     (SP)+, R2       ; ripristina registri
        MOV     (SP)+, R1
        CLZ                     ; Z = 0; C = 1
        SEC
        RTS     PC              ; STR1 contenuta in STR2
STCM3:                          ; STR2 contenuta in STR1
        MOV     (SP)+, R2       ; ripristina registri
        MOV     (SP)+, R1
        CLZ                     ; Z = 0; C = 0
        CLC
        RTS     PC
STCM4:                          ; STR1 = STR2
        MOV     (SP)+, R2       ; ripristina registri
        MOV     (SP)+, R1
        SEZ                     ; Z = 1; C = 0
        RTS     PC

;-------------------------------------------------------------;
; NOME                                                        ;
;   STNCMP  confronto tra stringhe fino a n caratteri         ;
; DESCRIZIONE                                                 ;
;   Restituisce l'informazione di confronto (dell'ordine      ;
;   lessicografico) tra due stringhe, limitando               ;
;   l'operazione ad un massimo di n caratteri.                ;
; INTERFACCIA                                                 ;
;   JSR PC, STNCMP                                            ;
;                                                             ;
;   R1      input   puntatore prima stringa                   ;
;   R2      input   puntatore seconda stringa                 ;
;   R3      input   n                                         ;
;   Z       output  = 1 se subst(STR1, n) = subst(STR2, n)    ;
;   C       output  = 1 se subst(STR1, n) < subst(STR2, n)    ;
; USA                                                         ;
;   --                                                        ;
;-------------------------------------------------------------;
STNCMP:
        SAVE    <R1,R2,R3>      ; salva registri
        TST     R3
        BEQ     STCN4           ; per definizione se n=0 le
                                ; stringhe sono eguali
STCN1:  CMPB    #EOS, (R1)      ; fine prima stringa ?
        BEQ     STCN2           ; salta se si`
        CMPB    #EOS, (R2)      ; fine seconda stringa ?
        BEQ     STCN3           ; salta se si`
        CMPB    (R1)+, (R2)+    ; non EOS
```

```
        BNE     STCN5           ; salta se diversi
        SOB     R3, STCN1       ; ciclo su n
        BR      STCN4           ; SUBSTR1 = SUBSTR2
STCN5:  RESTORE <R3,R2,R1>      ; ripristina registri
        CLZ                     ; Z = 0; C = *
        RTS     PC              ; prologhi diversi
STCN2:  CMPB    #EOS, (R2)      ; STR1 = STR2 ?
        BEQ     STCN4           ; salta se si`, Z=1; C=0
        RESTORE <R3,R2,R1>      ; ripristina registri
        CLZ                     ; Z = 0; C = 1
        SEC
        RTS     PC              ; STR1 contenuta in STR2
STCN3:                          ; STR2 contenuta in STR1
        RESTORE <R3,R2,R1>      ; ripristina registri
        CLZ                     ; Z = 0; C = 0
        CLC
        RTS     PC
STCN4:                          ; STR1 = STR2
        RESTORE <R3,R2,R1>      ; ripristina registri
        SEZ                     ; Z = 1; C = 0
        RTS     PC


;-------------------------------------------------------------;
; NOME                                                        ;
;   STCPY   copia di stringa su altra                         ;
; DESCRIZIONE                                                 ;
;   Copia il contenuto della stringa sorgente, fino a EOS,    ;
;   nella stringa destinazione. Si richiede un'area di        ;
;   destinazione di lunghezza sufficiente. La stringa         ;
;   sorgente non e` modificata.                               ;
; INTERFACCIA                                                 ;
;   JSR PC, STCPY                                             ;
;                                                             ;
;   R1      input   puntatore stringa sorgente                ;
;   R2      input   puntatore stringa destinazione            ;
; USA                                                         ;
;   --                                                        ;
;-------------------------------------------------------------;
STCPY:
        MOV     R1, -(SP)       ; salva registri
        MOV     R2, -(SP)
STCP1:
        MOVB    (R1), (R2)+     ; copia carattere
        CMPB    #EOS, (R1)+     ; fine prima stringa ?
        BNE     STCP1           ; ciclo se no
        MOV     (SP)+, R2       ; ripristina registri
        MOV     (SP)+, R1
        RTS     PC


;-------------------------------------------------------------;
; NOME                                                        ;
;   STNCPY  copia di stringa su altra fino a n caratteri      ;
; DESCRIZIONE                                                 ;
;   Copia il contenuto della stringa sorgente, fino a EOS     ;
;   oppure fino a n caratteri, nella stringa destinazione.    ;
;   Si richiede un'area di destinazione almeno di lunghezza   ;
;   n. La stringa sorgente non e` modificata.                 ;
; INTERFACCIA                                                 ;
```

```
;   JSR PC, STNCPY                                                  ;
;                                                                   ;
;   R1      input   puntatore stringa sorgente                      ;
;   R2      input   puntatore stringa destinazione                  ;
;   R3      input   n                                               ;
; USA                                                               ;
;   --                                                              ;
;-------------------------------------------------------------------;
STNCPY:
        SAVE    <R1,R2,R3>      ; salva registri
        TST     R3
        BEQ     STCY2           ; nessun carattere copiato
STCY1:
        MOVB    (R1), (R2)+     ; copia carattere
        CMPB    #EOS, (R1)+     ; fine prima stringa ?
        BEQ     STCY2           ; salta se si`
        SOB     R3, STCY1
STCY2:
        RESTORE <R3,R2,R1>      ; ripristina registri
        RTS     PC

;-------------------------------------------------------------------;
; NOME                                                              ;
;   STCAT   concatenazione di stringa ad altra                      ;
; DESCRIZIONE                                                       ;
;   Concatena la stringa sorgente (non modificata) alla             ;
;   stringa destinazione. Si richiede un'area di                    ;
;   destinazione di lunghezza sufficiente.                          ;
; INTERFACCIA                                                       ;
;   JSR PC, STCAT                                                   ;
;                                                                   ;
;   R1      input   puntatore stringa sorgente                      ;
;   R2      input   puntatore stringa destinazione                  ;
; USA                                                               ;
;   --                                                              ;
;-------------------------------------------------------------------;
STCAT:
        MOV     R1, -(SP)       ; salva registri
        MOV     R2, -(SP)
STCA1:
        CMPB    (R2)+, #EOS     ; fine stringa dest.
        BNE     STCA1
        DEC     R2
STCA2:
        MOVB    (R1), (R2)+     ; copia carattere
        CMPB    #EOS, (R1)+     ; fine prima stringa ?
        BNE     STCA2           ; ciclo se no
        MOV     (SP)+, R2       ; ripristina registri
        MOV     (SP)+, R1
        RTS     PC

;-------------------------------------------------------------------;
; NOME                                                              ;
;   STNCAT  concatenazione di stringa ad altra fino n carat.        ;
; DESCRIZIONE                                                       ;
;   Concatena la stringa sorgente (non modificata) alla             ;
;   stringa destinazione fino a raggiungere la fine della           ;
;   stringa sorgente oppure n caratteri complessivi. Si             ;
;   richiede un'area di destinazione almeno di lunghezza n.         ;
; INTERFACCIA                                                       ;
```

```
;   JSR PC, STNCAT                                                  ;
;                                                                   ;
;   R1      input   puntatore stringa sorgente                      ;
;   R2      input   puntatore stringa destinazione                  ;
;   R3      input   n                                               ;
; USA                                                               ;
;   --                                                              ;
;-------------------------------------------------------------------;
STNCAT:
        SAVE    <R1,R2,R3>      ; salva registri
        TST     R3
        BEQ     STCT4           ; nessun carattere copiato
STCT1:  CMPB    (R2)+, #EOS     ; fine stringa dest.
        BEQ     STCT2
        SOB     R3, STCT1       ; decrementa n
        BR      STCT4           ; nessun carattere copiato
STCT2:  DEC     R2
STCT3:  MOVB    (R1), (R2)+     ; copia carattere
        CMPB    #EOS, (R1)+     ; fine prima stringa ?
        BEQ     STCT4           ; salta se si`
        SOB     R3, STCT3       ; ciclo su n
STCT4:  RESTORE <R3,R2,R1>      ; ripristina registri
        RTS     PC

;-------------------------------------------------------------------;
; NOME                                                              ;
;   STSUB   estrazione di stringa da altra                          ;
; DESCRIZIONE                                                       ;
;   Estrae dalla stringa sorgente (non modificata) una              ;
;   sottostringa nella posizione (0..len-1) e della lunghezza;
;   specificate, copiandola nella destinazione. E` richiesta ;
;   un'area di destinazione di lunghezza sufficiente.               ;
; INTERFACCIA                                                       ;
;   JSR PC, STSUB                                                   ;
;                                                                   ;
;   R1      input   puntatore stringa sorgente                      ;
;   R2      input   puntatore stringa destinazione                  ;
;   R3      input   n caratteri trasferiti, compreso EOS            ;
;   R4      input   posizione inizio sottostringa (0..len-1) ;
; USA                                                               ;
;   Subr: STNCPY                                                    ;
;-------------------------------------------------------------------;
STSUB:
        MOV     R1, -(SP)       ; salva registri
        MOV     R2, -(SP)
        ADD     R4, R1          ; primo carattere da estrarre
        DEC     R3              ; n-1
        JSR     PC, STNCPY
        ADD     R3, R2          ; dopo n-1 caratteri
        MOVB    #EOS, (R2)      ; EOS inserito in ogni caso
        MOV     (SP)+, R2       ; ripristina registri
        MOV     (SP)+, R1
        RTS     PC

;-------------------------------------------------------------------;
; NOME                                                              ;
;   STINX   ricerca di sottostringa in una stringa                  ;
; DESCRIZIONE                                                       ;
;   Ricerca nella stringa sorgente (non modificata) una             ;
;   sottostringa (non modificata) a partire da una certa            ;
```

```
;   posizione. Ritorna la posizione della sottostringa.      ;
; INTERFACCIA                                                ;
;   JSR PC, STINX                                            ;
;                                                            ;
;   R1      input   puntatore stringa sorgente               ;
;   R2      input   puntatore sottostringa                   ;
;   R4      input   posizione iniziale                       ;
;           output  posizione sottostringa oppure -1 se non  ;
;                   trovata.                                 ;
; USA                                                        ;
;   --                                                       ;
;-----------------------------------------------------------;
STINX:
        SAVE    <R1,R2,R3>      ; salva registri
        ADD     R4, R1          ; puntatore a posizione iniz.
        MOV     R2, R3          ; salva punt. sottostr.
STIX1:
        MOV     R1, R4          ; salva punt. str.
STIX2:
        CMPB    #EOS, (R1)      ; fine stringa ?
        BEQ     STIX4           ; si, non trovata
        CMPB    #EOS, (R2)      ; fine sottostringa ?
        BEQ     STIX3           ; si, trovata
        CMPB    (R1)+, (R2)+    ; caratteri eguali ?
        BEQ     STIX2           ; si, ciclo
        MOV     R4, R1
        INC     R1              ; avanti un carattere
        MOV     R3, R2          ; recupera punt. sottost.
        BR      STIX1
STIX3:
        SUB     4(SP), R4       ; calcola posiz. raggiunta
        BR      STIX5
STIX4:
        CMPB    #EOS, (R2)      ; fine sottostringa ?
        BEQ     STIX3           ; si, trovata
        MOV     #-1, R4         ; non trovata
STIX5:
        RESTORE <R3,R2,R1>      ; ripristina registri
        RTS     PC
```

Quest'ultima routine merita una breve digressione. Essa realizza un algoritmo, il piu` semplice per il problema posto, riconducibile agli algoritmi di ricerca. Questa classe di algoritmi riveste una importanza fondamentale poiche` la loro efficienza incide in modo notevole in alcune applicazioni (editor, compilatori, basi di dati ecc.). La soluzione proposta e` in molti casi soddisfacente poiche` il ciclo interno viene eseguito poche volte per ogni ciclo esterno, spesso una sola, almeno per testi in linguaggio di programmazione o naturale. In altri casi (vedi ad esempio riconoscimento d'immagini), l'efficienza dell'algoritmo puo` decadere notevolmente. Ad esempio, detta s la stringa sorgente di lunghezza n e p la stringa nota da confrontare, di lunghezza m<=n, nel caso che sia s che p siano composte da una sequenza di lettere A terminata da una singola lettera B, il numero di confronti complessivo si avvicina a m*n. Sono stati sviluppati altri algoritmi piu` sofisticati in grado di ottenere una complessita` inferiore, quale ad esempio quello di Knuth-Morris-Pratt, qui brevemente trattato, di complessita` m+n.

L'algoritmo KMP si basa sulla constatazione che, se il ciclo interno viene eseguito k volte con altrettanti confronti positivi e

fallisce alla k+1-esima volta, i confronti validi consentono di conoscere la sottostringa di s di lunghezza k appena esaminata. Con l'algoritmo precedente, quando il ciclo interno fallisce, il ciclo esterno riparte un carattere dopo del primo carattere esaminato nel ciclo precedente, mentre con KMP e` possibile ripartire da un numero superiore di caratteri in avanti, sfruttando la conoscenza acquisita dai k confronti non falliti. Alcuni infatti dei k-1 caratteri che seguono il primo del ciclo precedente potrebbero costituire un prologo valido della stringa da ricercare e comunque i confronti possono ripartire dal k+1-esimo carattere. Per far questo, occorre conoscere quale parte dei k-1 caratteri puo` essere utilizzata come prologo per un nuovo passo di confronto, trascurando la corrispondente parte della stringa p. Viene pertanto costruita in fase di inizializzazione una tabella di dimensione m+1 che rappresenta, in caso di fallimento sul carattere j di p, qual'e` il carattere j' da cui ripartire per un nuovo ciclo esterno. Detta d questa tabella di dimensione m con d[j] generico elemento e 0<=j<=m, essa viene calcolata secondo il seguente algoritmo:

```
i:=0; j:=-1; d[0]:=-1;
repeat
    if (j<0) or (p[i] = p[j]) then begin
        i:=i+1; j:=j+1;
        d[i]:=j;
        end
    else
        j:=d[j]
    until i>m-1;
```

Ad esempio, si verifichi che per la stringa p=ABABC, si ha:

```
    d[0]=-1  d[1]=d[2]=d[5]=0  d[3]=1  d[4]=2
```

L'algoritmo di ricerca e` il seguente:

```
i:=0; j:=0;
repeat
    if (j<0) or (s[i] = p[j]) then begin
        i:=i+1; j:=j+1;
        end
    else
        j:=d[j]
    until (j>m-1) or (i>n-1);
if j>m-1 then found := i-m else found := -1;
```

Si verifichi ad esempio con s=ABABAABABC e p come sopra. Si noti che l'algoritmo di inizializzazione e` in pratica quello di ricerca applicato a due stringhe eguali a p.

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   STKMP   ricerca di sottostringa con KMP                   ;
; DESCRIZIONE                                                 ;
;   Ricerca nella stringa sorgente (non modificata) una       ;
;   sottostringa (non modificata) a partire da una certa      ;
;   posizione. Ritorna la posizione della sottostringa.       ;
;   L'algoritmo utilizzato e` quello di Knuth-Morris-Pratt.   ;
; INTERFACCIA                                                 ;
```

```
;   JSR PC, STKMP                                                    ;
;                                                                    ;
;   R1      input   puntatore stringa sorgente                       ;
;   R2      input   puntatore sottostringa                           ;
;   R4      input   posizione iniziale                               ;
;           output  pos. sottostr. oppure -1 se non trovata          ;
;   C       output  = 1 se M > DSIZ                                  ;
; USA                                                                ;
;   Subr: KMPINI KMPSUB STLEN                                        ;
;--------------------------------------------------------------------;
        DSIZ = 20.
STKMP:
        SAVE    <R1,R2,R3,R4>   ; salva registri
        MOV     R2, R1          ; calcola M
        JSR     PC, STLEN
        MOV     R3, M
        JSR     PC, KMPINI      ; calcola vettore d
        BCS     STK1            ; salta se M > DSIZ
        MOV     6(SP), R1       ; ripristina punt. s
        MOV     4(SP), R2       ; ripristina punt. p
        JSR     PC, STLEN       ; calcola N
        MOV     R3, N
        MOV     (SP)+, R4       ; indice iniziale per i
        ADD     R4, R1          ; s[i]
        JSR     PC, KMPSUB
        CLC
        RESTORE <R3,R2,R1>      ; ripristina registri
        RTS     PC
STK1:   RESTORE <R4,R3,R2,R1>   ; ripristina registri
        RTS     PC              ; C = 1

;--------------------------------------------------------------------;
; NOME                                                               ;
;   KMPINI  inizializzazione tabella per KMP                         ;
; DESCRIZIONE                                                        ;
;   Elabora la stringa di confronto per inizializzare                ;
;   la tabella d. La lunghezza massima di confronto e` DSIZ.         ;
; INTERFACCIA                                                        ;
;   JSR PC, KMPINI                                                   ;
;                                                                    ;
;   R1      input   puntatore stringa confronto p                    ;
;   R2      input       "       "                                    ;
;   C       output  = 1 se M > DSIZ                                  ;
; USA                                                                ;
;   Regs: R1 R2 R3 R4                                                ;
;   Memo: M DV                                                       ;
;--------------------------------------------------------------------;
KMPINI:
        CMP     #DSIZ, M
        BLO     KMIN1           ; stringa p troppo grande
        MOV     #-1, DV         ; d[0] = -1
        DEC     R2              ; p[-1]
        MOV     #-1, R3         ; j = -1
        CLR     R4              ; i = 0
KMIN2:  TST     R3
        BMI     KMIN3           ; salta se j<0
        CMPB    (R1), (R2)
        BNE     KMIN4           ; salta se p[i]<>p[j]
KMIN3:  INC     R1              ; p[i+1]
        INC     R2              ; p[j+1]
```

```
        INC     R3              ; j <- j+1
        INC     R4              ; i <- i+1
        ASL     R4              ; i*2
        MOV     R3, DV(R4)      ; d[i] <- j
        ASR     R4              ; ripristina i
        BR      KMIN5
KMIN4:  SUB     R3, R2          ; p[i]<>p[j]
        ASL     R3              ; j*2
        MOV     DV(R3), R3      ; j <- d[j]
        ADD     R3, R2          ; p[j]
KMIN5:  CMP     R4, M
        BLT     KMIN2           ; i<M: ciclo
        CLC
KMIN1:  RTS     PC
```

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   KMPSUB  ricerca sottostringa con KMP                      ;
; DESCRIZIONE                                                 ;
;   Ricerca nella stringa sorgente (non modificata) una       ;
;   sottostringa (non modificata) a partire da una certa      ;
;   posizione. Ritorna la posizione della sottostringa.       ;
;   Utilizza l'algoritmo di KMP e la tabella DV (d[]).        ;
; INTERFACCIA                                                 ;
;   JSR PC, KMPSUB                                            ;
;                                                             ;
;   R1      input   puntatore stringa sorgente                ;
;   R2      input   puntatore sottostringa                    ;
;   R4      input   posizione iniziale                        ;
;           output  posizione sottostringa oppure -1 se non   ;
;                   trovata.                                  ;
; USA                                                         ;
;   Regs: R1 R2 R3                                            ;
;   Memo: M N DV                                              ;
;-------------------------------------------------------------;
KMPSUB:
        CLR     R3              ; j=0
KMSU2:  TST     R3
        BMI     KMSU3           ; salta se j<0
        CMPB    (R1), (R2)
        BNE     KMSU4           ; salta se s[i]<>p[j]
KMSU3:  INC     R1              ; s[i+1]
        INC     R2              ; p[j+1]
        INC     R3              ; j <- j+1
        INC     R4              ; i <- i+1
        BR      KMSU5
KMSU4:  SUB     R3, R2          ; s[i]<>p[j]
        ASL     R3              ; j*2
        MOV     DV(R3), R3      ; j <- d[j]
        ADD     R3, R2          ; p[j]
KMSU5:  CMP     R3, M
        BGE     KMSU6           ; salta se j>=M, trovata
        CMP     R4, N
        BGE     KMSU7           ; i>=N: non trovata
        BR      KMSU2           ; ciclo
KMSU6:  SUB     M, R4           ; posizione = i-M
        RTS     PC
KMSU7:  MOV     #-1, R4         ; non trovata
        RTS     PC
```

```
        .CSECT  DAT
DV:     .BLKB   DSIZ+1
M:      .WORD   0
N:      .WORD   0
++++++++
```

Esercizio 3.3

Definire le subroutine LO2UP e UP2LO che convertono una stringa rispettivamente trasformando le lettere minuscole in maiuscole e viceversa. I caratteri non letterali sono lasciati inalterati.

Nella soluzione sottostante si e` utilizzato il registro R2 come puntatore alla routine di basso livello (LO2UPC, UP2LOC) in modo da realizzare di fatto un'unica routine per entrambe le operazioni.

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   LO2UP, UP2LO    conv. minuscolo maiuscolo di stringa      ;
; DESCRIZIONE                                                 ;
;   Trasforma tutte le lettere minuscole di una stringa in    ;
;   maiuscole e viceversa.                                    ;
; INTERFACCIA                                                 ;
;   JSR PC, LO2UP (UP2LO)                                     ;
;                                                             ;
;   R1      input   puntatore stringa da trasformare          ;
; USA                                                         ;
;   Subr: LO2UPC, UP2LOC                                      ;
;-------------------------------------------------------------;
LO2UP:
        SAVE    <R0, R1, R2>    ; salva registri
        MOV     #LO2UPC, R2     ; puntatore routine
        BR      LUS4
UP2LO:
        SAVE    <R0, R1, R2>    ; salva registri
        MOV     #UP2LOC, R2     ; puntatore routine
LUS4:   CLR     R0
LUS1:   MOVB    (R1), R0
        CMPB    #EOS, R0
        BEQ     LUS2            ; salta se fine stringa
        JSR     PC, (R2)        ; converte carattere
        BCC     LUS3            ; salta se non convertita
        MOVB    R0, (R1)
LUS3:   INC     R1
        BR      LUS1
LUS2:   RESTORE <R2, R1, R0>
        RTS     PC
++++++++
```

Esercizio 3.4

Si realizzi un riconoscitore a stati finiti in grado di stabilire se una stringa fornita come parametro puo` essere generata dalla seguente espressione (regolare):

```
    a[a#b]*c # acc
--------
```

Le stringhe da riconoscere sono "acc" e tutte quelle del tipo:

ac aac abc aaac aabc abac abbc aaaac ...

cioe` quelle che iniziano con 'a, terminano con 'c e tra queste due lettere possono avere un qualsiasi numero (anche nullo) di lettere 'a o 'b. Si puo` facilmente verificare che un automa a stati finiti che riconosca stringhe siffatte e` quello di fig. 3.1.

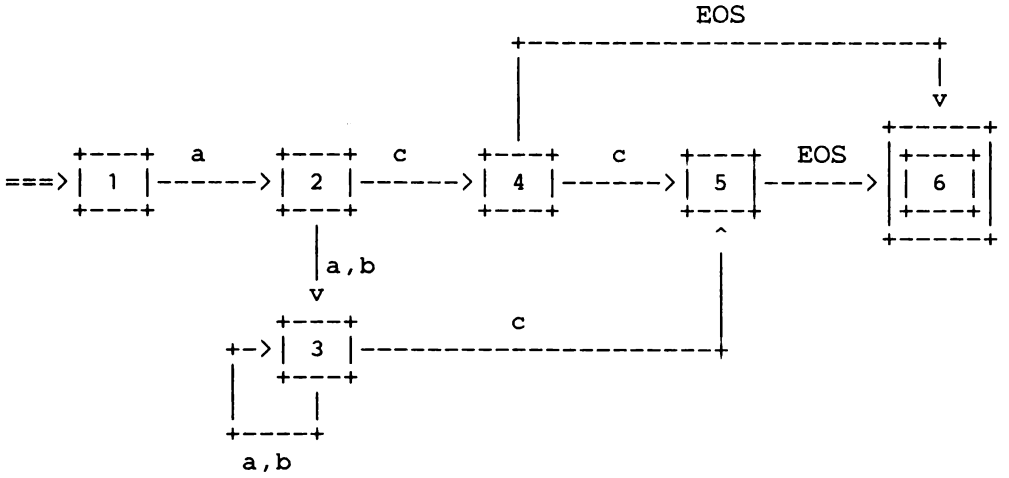


Fig. 3.1

Lo stato 1 e` lo stato iniziale e 6 quello finale. I caratteri indicati accanto agli archi sono quelli che determinano le corrispondenti transizioni di stato. Se nella scansione della stringa si incontra un carattere diverso da quelli associati alle transizioni possibili a partire dallo stato presente, allora la stringa non appartiene all'insieme da riconoscere.

Vi sono due metodi per realizzare il riconoscitore: il primo e` di tipo procedurale e consiste nel rappresentare gli stati dell'automa come particolari posizioni del programma e di simulare le transizioni come passaggi da un punto all'altro, condizionati da opportuni confronti. I confronti da effettuare sono quelli basati sui caratteri che etichettano gli archi in uscita agli stati dell'automa. Una stringa e` riconosciuta se, sulla base dei confronti effettuati, si raggiunge uno stato finale dopo aver scandito tutti i suoi caratteri, compreso EOS.

La seconda soluzione e` guidata da una tabella e pertanto piu` generale. Si tratta di costruire una tabella che in base allo stato corrente, memorizzato in opportuna variabile, e al carattere letto determini lo stato futuro che sara` quello esplicitato dall'automa, se e` prevista una transizione per quelle particolari condizioni. La tabella puo` essere organizzata in forma di matrice bidimensionale: gli indici di ingresso sono rappresentati dallo stato corrente e da una codifica interna di tutti i caratteri d'interesse, in modo da limitare la estensione della tabella. Vi e` inoltre un vettore che stabilisce quali siano gli stati finali. Una stringa e` riconosciuta se, partendo dallo stato iniziale ed effettuando tutte le transizioni che si rendono fattibili sulla base dei caratteri successivamente

letti dalla stringa, si perviene ad uno stato finale: la definizione dell'automa e` impostata in modo che allo stato finale si puo` pervenire solo a stringa completamente letta.

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   PRIC    riconoscitore procedurale di stringhe             ;
; DESCRIZIONE                                                 ;
;   Verifica che una stringa appartenga ad un linguaggio      ;
;   definito dalla espressione regolare:                      ;
;   a[a#b]*c # acc.                                           ;
; INTERFACCIA                                                 ;
;   JSR PC, PRIC                                              ;
;                                                             ;
;   R1      input   puntatore stringa da verificare           ;
;   C       output  = 1 se verifica ok                        ;
; USA                                                         ;
;   --                                                        ;
;-------------------------------------------------------------;
PRIC:
        MOV     R1, -(SP)       ; salva registro
PRC1:   CMPB    (R1)+, #'a      ; transizione 1 -> 2
        BNE     PRC7            ; non ok
PRC2:   CMPB    (R1), #'a       ; 2 -> 3
        BEQ     PRC3            ; ok
        CMPB    (R1), #'b       ; 2 -> 3
        BEQ     PRC3            ; ok
        CMPB    (R1)+, #'c      ; 2 -> 4
        BNE     PRC7            ; non ok
PRC4:   CMPB    (R1)+, #'c      ; 4 -> 5
        BEQ     PRC5            ; ok
        DEC     R1
        CMPB    (R1), #EOS      ; fine stringa ?
        BNE     PRC7            ; no, verifica non corretta
        BR      PRC6
PRC3:   INC     R1              ; incrementa puntatore
        CMPB    (R1), #'a       ; 3 -> 3
        BEQ     PRC3            ; ok
        CMPB    (R1), #'b       ; 3 -> 3
        BEQ     PRC3            ; ok
        CMPB    (R1)+, #'c      ; 3 -> 5
        BNE     PRC7            ; non ok
PRC5:   CMPB    (R1), #EOS      ; fine stringa ?
        BNE     PRC7            ; no, non ok
PRC6:   MOV     (SP)+, R1       ; si, verifica corretta, C=1
        SEC
        RTS     PC
PRC7:   CLC
        MOV     (SP)+, R1       ; verifica non corretta, C=0
        RTS     PC

;-------------------------------------------------------------;
; NOME                                                        ;
;   TRIC    riconoscitore tabellare di stringhe               ;
; DESCRIZIONE                                                 ;
;   Verifica che una stringa appartenga ad un linguaggio      ;
;   definito dalla espressione regolare:                      ;
;   a[a#b]*c # acc.                                           ;
;   Allo scopo viene utilizzata una tabella che fornisce      ;
;  . stato futuro e condizione di terminazione sulla base     ;
```

```
;   dello stato corrente e della seguente codifica per i      ;
;   caratteri di interesse:                                   ;
;   a = 0   b = 1   c = 2   EOS = 3                           ;
;   Per comodita` gli stati sono codificati 0..5. Il valore   ;
;   -1 nella tabella di transizione corrisponde alla non      ;
;   appartenenza della stringa al linguaggio.                 ;
; INTERFACCIA                                                 ;
;   JSR PC, TRIC                                              ;
;                                                             ;
;   R1      input   puntatore stringa da verificare           ;
;   C       output  = 1 se verifica ok                        ;
; USA                                                         ;
;   Memo: CTAB ATAB TTAB                                      ;
;-------------------------------------------------------------;
TRIC:
        SAVE    <R0,R1,R2,R4>   ; salva registri
        CLR     R0              ; stato corrente iniziale
TRC1:   MOVB    (R1)+, R2       ; carica carattere
        CLR     R4
TRC2:   CMPB    R2, CTAB(R4)    ; cerca in tabella conversione
        BEQ     TRC3
        INC     R4
        CMP     #4, R4
        BNE     TRC2
        BR      TRC4            ; carattere illegale, C=0
TRC3:   ASL     R0              ; R0*4 (numero caratteri)
        ASL     R0
        ADD     R0, R4          ; indice stato futuro, C=0
        MOVB    ATAB(R4), R0    ; carica stato futuro
        BMI     TRC4            ; salta se input illegale
        TSTB    TTAB(R0)        ; stato finale ?
        BEQ     TRC1            ; no, ciclo
        SEC                     ; si, verifica corretta
TRC4:   RESTORE <R4,R2,R1,R0>   ; ripristina registri
        RTS     PC

        .CSECT  DAT             ; tabella di transizione
                                ;   | stato corrente
                                ;   |   | carattere d'ingresso
ATAB:   .BYTE   1, -1, -1, -1   ; s=0; c=a,b,c,EOS
        .BYTE   2, 2, 3, -1     ; s=1; c=a,b,c,EOS
        .BYTE   2, 2, 4, -1     ; s=2; c=a,b,c,EOS
        .BYTE   -1, -1, 4, 5    ; s=3; c=a,b,c,EOS
        .BYTE   -1, -1, -1, 5   ; s=4; c=a,b,c,EOS
        .BYTE   -1, -1, -1, -1  ; s=5; c=a,b,c,EOS

TTAB:   .BYTE   0, 0, 0, 0, 0, 1; stati terminali = 1

CTAB:   .BYTE   'a, 'b, 'c, EOS ; codifica caratteri
                                ; 0, 1, 2, 3
++++++++
```

3.2 - Routine numeriche

L'assembly del PDP11 fornisce le operazioni di elaborazione numerica fondamentali. Altri tipi di elaborazioni possono essere ottenuti mediante l'uso di routine appositamente predisposte. Gli

esercizi che seguono illustrano alcune routine di uso generale che estendono le capacita` di elaborazione numerica, permettendo di eseguire moltiplicazioni e divisioni (per i processori che non dispongono di queste istruzioni) e calcolo di funzioni.

Esercizio 3.5

Definire le routine SMUL e DMUL in grado di eseguire la moltiplicazione tra quantita` senza segno a 16 bit, fornendo la prima un word come risultato e la seconda, piu` estesa, due word. SMUL deve segnalare l'eventuale overflow nel calcolo. Successivamente definire le versioni rispettive SSMUL e SDMUL per la moltiplicazione di numeri con segno.

L'algoritmo da utilizzare per il calcolo della moltiplicazione e` quello illustrato nell'esercizio 1.18. La versione estesa DMUL coinvolge due registri come accumulatori.

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   SMUL        moltiplicazione 16 bit senza segno            ;
; DESCRIZIONE                                                 ;
;   Subroutine di moltiplicazione di due numeri di 16 bit     ;
;   senza segno. Il prodotto (16 bit) puo` presentare         ;
;   overflow.                                                 ;
; INTERFACCIA                                                 ;
;   JSR PC, SMUL                                              ;
;                                                             ;
;   R0      input   moltiplicando                             ;
;   R1      input   moltiplicatore                            ;
;   R1      output  prodotto                                  ;
;   C       output  1 se c'e` overflow                        ;
; USA                                                         ;
;   --                                                        ;
;-------------------------------------------------------------;
SMUL:
        MOV     R4, -(SP)       ; salva i registri usati
        MOV     R3, -(SP)
        MOV     R1, R3          ; MPER -> R3
        CLR     R1              ; risultato (PROD)
        MOV     #16., R4        ; contatore
MU1:    ASL     R1              ; PROD * 2 -> PROD
        BCS     MU3             ; overflow
        ASL     R3              ; b15(MPER) -> C
        BCC     MU2
        ADD     R0, R1          ; somma MPND se C=1
        BCS     MU3             ; overflow
MU2:    SOB     R4, MU1         ; itera 16 volte
        CLC                     ; non c'e` overflow
MU3:    MOV     (SP)+, R3       ; ripristina i registri
        MOV     (SP)+, R4
        RTS     PC

;-------------------------------------------------------------;
; NOME                                                        ;
;   DMUL        moltiplicazione 16 bit senza segno            ;
; DESCRIZIONE                                                 ;
;   Subroutine di moltiplicazione di due numeri di 16 bit     ;
```

```
;   senza segno. Il prodotto (32 bit) non presenta mai       ;
;   overflow.                                                ;
; INTERFACCIA                                                ;
;   JSR PC, DMUL                                             ;
;                                                            ;
;   R0      input   moltiplicando                            ;
;   R1      input   moltiplicatore                           ;
;   R2      output  prodotto (parte meno significativa)      ;
;   R3      output  prodotto (parte piu` significativa)      ;
; USA                                                        ;
;   --                                                       ;
;-----------------------------------------------------------;
DMUL:
        MOV     R4, -(SP)       ; salva i registri usati
        MOV     R1, -(SP)
        CLR     R2              ; azzera i registri destinati
        CLR     R3              ; a contenere il risultato
        MOV     #16., R4        ; contatore
DMU1:   ASL     R2              ; PROD * 2 -> PROD
        ROL     R3
        ASL     R1              ; b15(MPER) -> C
        BCC     DMU2
        ADD     R0, R2          ; somma MPND se C=1
        ADC     R3              ; somma su 32 bit
DMU2:   SOB     R4, DMU1        ; itera 16 volte
        MOV     (SP)+, R1       ; ripristina i registri
        MOV     (SP)+, R4
        RTS     PC
```

Le versioni con segno si ottengono facilmente dalle precedenti tenendo conto che:

```
p = a * b   = sign(a) * sign(b) * abs(a) * abs(b) =
            = abs(a)*abs(b)     se sign(a)*sign(b) = 1
            = -(abs(a)*abs(b))  se sign(a)*sign(b) = -1
sign(x)     = 1     se x>=0
            = -1    se x<0
```

```
;-----------------------------------------------------------;
; NOME                                                       ;
;   SSMUL       moltiplicazione 16 bit con segno             ;
; DESCRIZIONE                                                ;
;   Subroutine di moltiplicazione di due numeri di 16 bit    ;
;   con segno. Il prodotto (16 bit con segno) puo` presentare;
;   overflow.                                                ;
; INTERFACCIA                                                ;
;   JSR PC, SSMUL                                            ;
;                                                            ;
;   R0      input   moltiplicando                            ;
;   R1      input   moltiplicatore                           ;
;   R1      output  prodotto                                 ;
;   C       output  1 se c'e` overflow                       ;
; USA                                                        ;
;   Subr: SMUL                                               ;
;-----------------------------------------------------------;
        NOSIGN = 077777
SSMUL:
        MOV     R0, -(SP)       ; salva registro
```

```
        MOV     R0, -(SP)
        BPL     SSM1            ; salta se positivo
        NEG     R0              ; abs(R0)
SSM1:   BIC     #NOSIGN, (SP)   ; solo bit segno
        ADD     R1, (SP)        ; sign(R0)*sign(R1) come segno
        TST     R1
        BPL     SSM2            ; salta se positivo
        NEG     R1
SSM2:   JSR     PC, SMUL        ; moltiplicazione
        BCS     SSM4
        TST     (SP)+
        BPL     SSM3            ; salta se ris. positivo
        NEG     R1              ; risultato negativo
        CLC
SSM3:   MOV     (SP)+, R0       ; ripristina registro
        RTS     PC
SSM4:   MOV     (SP)+, R0       ; C=1, overflow
        MOV     (SP)+, R0       ; ripristina registro
        RTS     PC

;---------------------------------------------------------------;
; NOME                                                          ;
;   SDMUL       moltiplicazione 16 bit con segno                ;
; DESCRIZIONE                                                   ;
;   Subroutine di moltiplicazione di due numeri di 16 bit       ;
;   con segno. Il prodotto (32 bit con segno) non presenta      ;
;   mai overflow.                                               ;
; INTERFACCIA                                                   ;
;   JSR PC, SDMUL                                               ;
;                                                               ;
;   R0      input   moltiplicando                               ;
;   R1      input   moltiplicatore                              ;
;   R2      output  prodotto (parte meno significativa)         ;
;   R3      output  prodotto (parte piu` significativa con      ;
;                   segno).                                     ;
; USA                                                           ;
;   Subr: DMUL                                                  ;
;---------------------------------------------------------------;
SDMUL:
        MOV     R0, -(SP)       ; salva registri
        MOV     R1, -(SP)
        MOV     R0, -(SP)
        BPL     SDM1            ; salta se positivo
        NEG     R0              ; abs(R0)
SDM1:   BIC     #NOSIGN, (SP)   ; solo bit segno
        ADD     R1, (SP)        ; sign(R0)*sign(R1) come segno
        TST     R1
        BPL     SDM2            ; salta se positivo
        NEG     R1
SDM2:   JSR     PC, DMUL        ; moltiplicazione
        TST     (SP)+
        BPL     SDM3            ; salta se ris. positivo
        COM     R2              ; negazione su 32 bit
        COM     R3
        ADD     #1, R2
        ADC     R3
SDM3:   MOV     (SP)+, R1       ; ripristina registri
        MOV     (SP)+, R0
        RTS     PC
++++++++
```

Esercizio 3.6

Definire la routine DDIV in grado di calcolare il quoziente e il resto della divisione di un dividendo senza segno di 32 bit e divisore di 16 bit senza segno. Successivamente definire la versione SDDIV per la divisione di numeri con segno.

Analogamente al caso della moltiplicazione, detto a il dividendo, d il divisore, q il quoziente e r il resto, si ha:

$$a = q * d + r \qquad r < d$$

Espandendo q nella sua rappresentazione in base 2 con n bit, si ha:

$$q = q_{n-1}*2^{n-1} + q_{n-2}*2^{n-2} + \ldots + q_1*2 + q_o$$

$$a = q_{n-1}*2^{n-1}*d + q_{n-2}*2^{n-2}*d + \ldots + q_1*2*d + q_o*d + r$$

si ricava:

$$q_{n-1} = \text{int}(a / (2^{n-1}*d)) = \text{int}((a*2) / (2^n*d))$$

Definiti $r_n=a$ e r_{n-1} come:

$$r_{n-1} = (a*2) - q_{n-1}*2^n*d = q_{n-2}*2^{n-1}*d + \ldots + q_o*2*d+r*2$$

si ha:

$$q_{n-2} = \text{int}(r_{n-1} / (2^{n-1}*d)) = \text{int}((r_{n-1}*2) / (2^n*d))$$
$$r_{n-2} = r_{n-1}*2 - q_{n-2}*2^n*d = q_{n-3}*2^{n-1}*d + \ldots + q_o*2^2*d+r*2^2$$

Generalizzando:

$$q_{k-1} = \text{int}((r_k*2) / (2^n*d)) \qquad \text{con } 1<=k<=n$$
$$r_{k-1} = r_k*2 - q_{k-1}*2^n*d = q_{k-2}*2^{n-1}*d + \ldots + q_1*2^{n-k+2}*d + q_o*2^{n-k+1}*d+r*2^{n-k+1}$$

$$r_1 = r_2*2-q_1*2^n d = q_o*2^{n-1}*d+r*2^{n-1}$$
$$r_o = r_1*2-q_o*2^n*d = r*2^n$$

La realizzazione dell'algoritmo e` naturalmente facilitata dalla rappresentazione binaria, in quanto la moltiplicazione per 2 di r_k si ottiene con uno shift a sinistra su 32 bit mentre essendo q_{k-1} 0 o 1, si ha che:

$$q_{k-1} = \begin{cases} 0 & \text{se } (r_k*2) < (2^n*d) \\ 1 & \text{se } (r_k*2) >= (2^n*d) \end{cases}$$

Quindi e` sufficiente fare un confronto tra d e il word piu` significativo di r_k a shift eseguito. Se $q_{k-1}=1$ allora si effettua la differenza con d sempre limitatamente al word piu` significativo, altrimenti $r_k=r_{k-1}$. Dopo 16 cicli, $r_o$ corrisponde al resto della divisione, moltiplicato per $2^{16}$. Utilizzando il word meno significativo come accumulatore per $q_{k-1}$, alla fine il resto si trova nel word piu` significativo e il quoziente nell'altro.

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   DDIV    subroutine divisione interi senza segno           ;
; DESCRIZIONE                                                 ;
;   Effettua la divisione tra interi senza segno, dividendo   ;
;   di 32 bit e divisore di 16 bit, fornendo quoziente e      ;
;   resto.                                                    ;
; INTERFACCIA                                                 ;
;   JSR PC, DDIV                                              ;
;                                                             ;
;   R0      input   dividendo (parte meno significativa)      ;
;   R1      input   dividendo (parte piu` significativa)      ;
;   R2      input   divisore                                  ;
;   R0      output  resto                                     ;
;   R1      output  quoziente                                 ;
;   C       output  = 1 se overflow                           ;
; USA                                                         ;
;   --                                                        ;
;-------------------------------------------------------------;
DDIV:   MOV     R3, -(SP)    ; salva registro
        CMP     R1, R2       ; (R1)>(R2) ?
        BGE     DDV1         ; si`, overflow
        MOV     #16., R3     ; contatore posto a 16 (k)
        ASL     R0           ; shift r_n parte meno sign.
DDV4:
        ROL     R1           ; shift r_k parte piu` sign.
        CMP     R1, R2       ; r_k*2 >= 2^n*d ?
        BLT     DDV2         ; salta se q_k-1=0, r_k-1=r_k
        SUB     R2, R1       ; calcola r_k-1
        SEC                  ; q_k-1 = 1
        BR      DDV3
DDV2:   CLC                  ; q_k-1 = 0
DDV3:
        ROL     R0           ; shift r_k-1 parte meno sign.
        DEC     R3           ; k <- k-1
        BNE     DDV4         ; k=0 ?
        CLC                  ; ris. in R1 C=0
        BR      DDV5
DDV1:   SEC                  ; overflow C=1
DDV5:   MOV     R0, R3       ; R0 <-> R1
        MOV     R1, R0
        MOV     R3, R1
        MOV     (SP)+, R3    ; ripristina registro
        RTS     PC
```

Per la divisione con segno, analogamente alla moltiplicazione si ha:

```
q = int(a/d)    = sign(a) * sign(d) * int(abs(a)/abs(d)) =
                = int(abs(a)/abs(d))    se sign(a)*sign(d) = 1
                = -int(abs(a)/abs(d))   se sign(a)*sign(d) =-1
```

```
r = a - q*d = sign(a)*abs(a) - sign(q)*sign(d)*abs(q)*abs(d) =
= sign(a)*abs(a) - sign(a)*sign(d)*sign(d)*abs(q)*abs(d) =
= sign(a) * (abs(a) - abs(q)*abs(d))
```

cioe` il resto ha lo stesso segno del dividendo.

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   SDDIV   divisione interi con segno                        ;
; DESCRIZIONE                                                 ;
;   Effettua la divisione tra interi con   segno, dividendo   ;
;   di 32 bit e divisore di 16 bit, fornendo quoziente e      ;
;   resto.                                                    ;
; INTERFACCIA                                                 ;
;   JSR PC, SDDIV                                             ;
;                                                             ;
;   R0      input   dividendo (parte meno significativa)      ;
;   R1      input   dividendo (parte piu` significativa)      ;
;   R2      input   divisore                                  ;
;   R0      output  resto                                     ;
;   R1      output  quoziente                                 ;
;   C       output  = 1 se overflow                           ;
; USA                                                         ;
;   Subr: DDIV                                                ;
;-------------------------------------------------------------;
        SIGN = 100000
        CSIGN = 040000
SDDIV:
        MOV     R2, -(SP)       ; salva registro
        MOV     R1, -(SP)       ; segno dividendo
        ASR     (SP)            ; copia segno su b14
        BPL     SDV1            ; salta se dividendo positivo
        COM     R0              ; negazione su 32 bit
        COM     R1
        ADD     #1, R0
        ADC     R1
SDV1:   TST     R2
        BPL     SDV2            ; salta se divisore positivo
        ADD     #SIGN, (SP)     ; sign(a)*sign(d) come segno
        NEG     R2              ; divisore negativo
SDV2:   JSR     PC, DDIV        ; divisione
        BCS     SDV5            ; salta se overflow
        BIT     #CSIGN, (SP)    ; resto stesso segno dividendo
        BEQ     SDV3            ; salta se positivo
        NEG     R0
SDV3:   TST     (SP)+
        BPL     SDV4            ; salta se quoziente positivo
        NEG     R1              ; quoziente < 0
SDV4:   CLC
        MOV     (SP)+, R2       ; rispristina registro
        RTS     PC
SDV5:   MOV     (SP)+, R1       ; C=1, overflow
        MOV     (SP)+, R2       ; rispristina registro
        RTS     PC
++++++++
```

Esercizio 3.7

Definire la routine SMAX che stabilisce il valore massimo di una lista di interi con segno passata come parametro.

Il numero degli elementi della lista e un puntatore alla stessa vengono passati come parametri con la tecnica della in-line calling sequence.

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   SMAX    ricerca massimo interi con segno                  ;
; DESCRIZIONE                                                 ;
;   Subroutine per la ricerca del massimo di una lista di     ;
;   interi con segno.  Il numero di elementi della lista e    ;
;   un puntatore alla lista sono passati alla subroutine      ;
;   nella calling sequence.                                   ;
; INTERFACCIA                                                 ;
;   JSR R1, SMAX                                              ;
;   ...             ; numero elementi                         ;
;   ...             ; indirizzo del primo elemento            ;
;   ...             ; punto di ritorno                        ;
;                                                             ;
;   R0      output  valore massimo trovato                    ;
; USA                                                         ;
;   --                                                        ;
;-------------------------------------------------------------;
SMAX:
        MOV     R3, -(SP)       ; salva i registri usati
        MOV     R2, -(SP)
        MOV     (R1)+, R3       ; numero di elementi
        MOV     (R1)+, R2       ; indirizzo del primo numero
        DEC     R3              ; decrementa il contatore
        MOV     (R2)+, R0       ; massimo temporaneo
SMX1:
        CMP     R0, (R2)+       ; confronto con il massimo
        BGE     SMX2            ; e` maggiore ?
        MOV     -2(R2), R0      ; si, nuovo massimo temporaneo
SMX2:
        SOB     R3, SMX1        ; ci sono altri elementi ?
        MOV     (SP)+, R2       ; no, ripristina i registri
        MOV     (SP)+, R3
        RTS     R1

;*************************************************************;
;       programma di prova
;
PROVA:
        JSR     R1, SMAX
        .WORD   5               ; calling sequence
        .WORD   MATRIX+24       ; terza riga della matrice 5x5

        .CSECT  DAT

MATRIX: .WORD   -1, 2, -3, 4, 5     ; prima riga
        .WORD   0, 4, -6, -10, 12   ; seconda riga
        .WORD   -7, 4, 5, 4, 0      ; terza riga
        .WORD   2, 4, -5, 6, -3     ; quarta riga
        .WORD   0, 0, -2, 1, -1     ; quinta riga
++++++++
```

Esercizio 3.8

Definire una routine di calcolo del fattoriale di un numero intero.

Vengono qui proposte varie soluzioni del problema. La prima, la

piu` semplice, si basa sull'algoritmo iterativo:

```
    fact(n) = n * (n-1) * (n-2) *...* 2 * 1.
```

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   FACT    subroutine iterativa per fattoriale               ;
; DESCRIZIONE                                                 ;
;   Realizzazione iterativa del calcolo del fattoriale di un  ;
;   numero naturale n.                                        ;
; INTERFACCIA                                                 ;
;   JSR PC, FACT                                              ;
;                                                             ;
;   R0      input   valore di n                               ;
;   R1      output  fattoriale di n                           ;
; USA                                                         ;
;   Subr: SMUL                                                ;
;   Regs:  R0                                                 ;
;-------------------------------------------------------------;
FACT:
        MOV R0, R1              ; R1 <- n
FA1:    DEC R0                  ; n-1
        BEQ FA2                 ; fine ciclo
        JSR PC, SMUL            ; R1 <- fact(i)
        BR FA1
FA2:    RTS PC
```

La seconda versione si basa invece sulla definizione ricorsiva:

```
    fact(n) = n * fact(n-1)     n > 1
    fact(1) = 1
```

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   RFACT   subroutine ricorsiva per fattoriale               ;
; DESCRIZIONE                                                 ;
;   Realizzazione ricorsiva del calcolo del fattoriale di un  ;
;   numero naturale n.                                        ;
; INTERFACCIA                                                 ;
;   JSR PC, RFACT                                             ;
;                                                             ;
;   R0      input   valore di n                               ;
;   R1      output  fattoriale di n                           ;
; USA                                                         ;
;   Subr: SMUL RFACT                                          ;
;   Regs: R0                                                  ;
;-------------------------------------------------------------;
RFACT:
        CMP     R0, #2          ;FACT(1)=1, FACT(2)=2
        BGT     FA1
        MOV     R0, R1          ;risultato (1 o 2) in R1
        RTS     PC
FA1:
        MOV     R0, -(SP)       ;i
        DEC     R0              ;i-1
        JSR     PC, RFACT       ;FACT(i-1) -> R1
        MOV     (SP)+, R0       ;i -> R0
        JSR     PC, SMUL        ;i*FACT(i-1) -> R1
        RTS     PC
```

La terza versione, egualmente ricorsiva, utilizza una modalita` di passaggio dei parametri tramite lo stack.

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   RFACT1  subroutine ricorsiva per fattoriale               ;
; DESCRIZIONE                                                 ;
;   Realizzazione ricorsiva del calcolo del fattoriale di un  ;
;   numero naturale n. Non usa registri. L'operando e il      ;
;   risultato sono passati tramite lo stack.                  ;
; INTERFACCIA                                                 ;
;   JSR PC, RFACT1                                            ;
;                                                             ;
;   stack top   input   valore di n                           ;
;   stack top   output  fattoriale di n                       ;
; USA                                                         ;
;   Subr: SMUL RFACT1                                         ;
;-------------------------------------------------------------;
RFACT1:
        CMP     2(SP), #2       ;FACT(1)=1, FACT(2)=2
        BGT     FA1
        RTS     PC
FA1:
        MOV     R0, -(SP)       ; salva registri
        MOV     R1, -(SP)
        MOV     6(SP), R1       ;i -> R1
        MOV     R1, -(SP)
        DEC     (SP)            ;i-1
        JSR     PC, RFACT1      ;FACT(i-1)
        MOV     (SP)+, R0       ;FACT(i-1) -> R0
        JSR     PC, SMUL        ;i*FACT(i-1) -> R1
        MOV     R1, 6(SP)
        MOV     (SP)+, R1       ; ripristina registri
        MOV     (SP)+, R0
        RTS     PC
```

L'ultima versione vuole essere un esempio di possibile traduzione, effettuata da un compilatore, da un linguaggio ad alto livello. Si mettono in evidenza le singole istruzioni con la loro traduzione e la configurazione dello stack.

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   RFACT2  subroutine ricorsiva per fattoriale               ;
; DESCRIZIONE                                                 ;
;   Realizzazione ricorsiva del calcolo del fattoriale di un  ;
;   numero naturale n. E` organizzata come fosse la           ;
;   espansione della procedura PASCAL:                        ;
;            procedure RFACT2(N:integer; var FN:integer);     ;
;               var TN, TFN:integer                           ;
;               if N=1 then FN:=1                             ;
;                      else     begin                         ;
;                               TN:=N-1;                      ;
;                               RFACT(TN, TFN);               ;
;                               FN:=N*TFN                     ;
;                               end                           ;
;            end                                              ;
;   con allocazione dinamica dei parametri e delle            ;
;   variabili locali in uno stack. Usa R5 come Frame Pointer  ;
```

```
; INTERFACCIA                                                    ;
;   JSR PC, RFACT2                                               ;
;                                                                ;
;   st.top-1     input   valore di n                             ;
;   st. top      input   indirizzo del risultato                 ;
; USA                                                            ;
;   Subr: SMUL RFACT2                                            ;
;   (Regs: R5 come Frame Pointer)                                ;
;----------------------------------------------------------------;

;procedure RFACT2(N:integer; var FN:integer)
;
RFACT2:
        MOV     R5, -(SP)       ;salva il FP precedente
        MOV     SP, R5          ;  FN=@+4(R5)   (by name)
                                ;   N= +6(R5)   (by value)
        ADD     #-4, SP         ;variabili locali:
                                ;   TN=-2(R5)
                                ;  TFN=-4(R5)
                                ;
                                ;contenuto dello stack frame:
                                ;
                                ;     |________|
                                ;  -4 |__TFN___| <----(SP)
                                ;  -2 |___TN___|
                                ;     |_old_FP_| <----(FP)
                                ;  +2 |___PC___|
                                ;  +4 |__#FN___|
                                ;  +6 |____N___|
                                ;     |________|
                                ;

;if N=1 then FN:=1
;
        CMP     6(R5), #1       ;if N=1
        BNE     $1              ;then
        MOV     #1, @4(R5)      ;FN:=1
;else begin
;
        BR      $2
$1:
;TN:=N-1
;
        MOV     6(R5), -2(R5)
        DEC     -2(R5)
;RFACT2(TN,TFN)
;
        MOV     -2(R5), -(SP)   ;push TN
        MOV     #-4, -(SP)      ;push #TFN
        ADD     R5, (SP)
        JSR     PC, RFACT2      ;(N-1)! -> TFN
        ADD     #4, SP          ;dealloca i parametri
;FN:=N*TFN
;
        MOV     R0, -(SP)
        MOV     R1, -(SP)
        MOV     6(R5), R0       ;N
        MOV     -4(R5), R1      ;(N-1)!
        JSR     PC, SMUL        ;N*(N-1)! -> R1
        MOV     R1, @4(R5)      ;          -> FN
```

```
        MOV     (SP)+, R1
        MOV     (SP)+, R0
;end {begin}
;
$2:

;end {RFACT2}
;
        MOV     R5, SP          ;dealloca il frame
        MOV     (SP)+, R5       ;ripristina il vecchio FP
        RTS     PC
++++++++
```

Esercizio 3.9

Definire una routine ricorsiva di generazione della serie dei numeri di Fibonacci.

La serie dei numeri di Fibonacci e` la serie di numeri naturali ottenuta dalla seguente definizione ricorsiva:

```
F(n) = F(n-1)+F(n-2)    n >= 2
F(0) = 0        F(1) = 1
```

Senza entrare nel merito delle proprieta` di questa serie, la sua definizione si presta ad una realizzazione di tipo ricorsivo.

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   FIBO    generazione serie numeri di Fibonacci             ;
; DESCRIZIONE                                                 ;
;   Subroutine ricorsiva per il calcolo della serie di        ;
;   Fibonacci:                                                ;
;   F(n) = F(n-1)+F(n-2)                                      ;
; INTERFACCIA                                                 ;
;   JSR PC, FIBO                                              ;
;                                                             ;
;   R0      input   n                                         ;
;   R1      ouput   F(n)                                      ;
; USA                                                         ;
;   Subr: FIBO                                                ;
;   Regs: R0                                                  ;
;-------------------------------------------------------------;
FIBO:
        CMP     R0, #2
        BGE     FB1
        MOV     R0, R1          ; F(0)=0; F(1)=1
        RTS     PC
FB1:
        DEC     R0              ; i-1
        MOV     R0, -(SP)       ; salva R0
        JSR     PC, FIBO        ; F(i-1)
        MOV     (SP)+, R0       ; recupera R0
        DEC     R0              ; i-2
        MOV     R1, -(SP)       ; salva F(i-1)
        JSR     PC, FIBO        ; F(i-2)
        ADD     (SP)+, R1       ; R1<-F(i-2)+F(i-1)
        RTS     PC
```

Si suggerisce al lettore di provare per qualche n la routine, evidenziando l'evoluzione dello stack.
++++++++

Esercizio 3.10

Definire una routine di generazione di numeri casuali contenuti in word a 16 bit.

Il piu` noto generatore di sequenze di numeri casuali e` basato sulla seguente formula:

$r = (r*b+1) \bmod m$

dove r rappresenta la successione di numeri generati, b ed m sono costanti opportunamente scelte. Per b viene consigliato di scegliere un numero che non presenti particolari configuazioni di cifre, che abbia una cifra decimale in meno di m e che termini con le cifre decimali 821. Per m e` conveniente scegliere una potenza di due per semplificare l'operazione di modulo. Nel caso presente:

$m = 2^{16} = 65536.$ $\quad b = 5821.$

Partendo da diversi valori iniziali di r, si ottengono sequenze diverse. Ad esempio, con r_o = 12345.(030071), si ottiene la sequenza:

r_1 = 32790. (100026)
r_2 = 29759. (072077)
r_3 = 15492. (036204)
r_4 = 1397. (002565)
r_5 = 5474. (012542)

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   RNDINI  inizializzazione generatore numeri casuali        ;
; DESCRIZIONE                                                 ;
;   Inizializza il valore di stato del generatore.            ;
; INTERFACCIA                                                 ;
;   JSR PC, RNDINI                                            ;
;                                                             ;
;   R0      input   valore iniziale                           ;
; USA                                                         ;
;   Memo: RNDR                                                ;
;-------------------------------------------------------------;
        RNDB    = 5821.
RNDINI:
        MOV     R0, RNDR        ; valore di stato del
                                ; generatore
        RTS     PC

;-------------------------------------------------------------;
; NOME                                                        ;
;   RNDGEN  generatore di numeri casuali                      ;
; DESCRIZIONE                                                 ;
;   Genera il successivo numero casuale della serie basata    ;
;   sul valore iniziale fornito per RNDINI.                   ;
; INTERFACCIA                                                 ;
```

```
;   JSR PC, RNDGEN                                                  ;
;                                                                   ;
;   R0      output  numero generato                                 ;
; USA                                                               ;
;   Subr: DMUL                                                      ;
;   Memo: RNDR                                                      ;
;-------------------------------------------------------------------;
RNDGEN:
        SAVE    <R1,R2,R3>      ; salva registri
        MOV     RNDR, R0        ; carica valore di stato
        MOV     #RNDB, R1       ; carica b
        JSR     PC, DMUL        ; prodotto
        INC     R2              ; (r*b+1) mod 2^16
        MOV     R2, R0
        MOV     R2, RNDR        ; nuovo valore di stato
        RESTORE <R3,R2,R1>      ; ripristina registri
        RTS     PC

        .CSECT  DAT
RNDR:   .WORD   0               ; valore di stato
++++++++
```

Esercizio 3.11

Definire una routine di calcolo della radice quadrata basata sul fatto che i quadrati dei numeri naturali si possono ottenere come somme di numeri dispari crescenti (ad esempio 16 = 1 + 3 + 5 + 7).

```
;-------------------------------------------------------------------;
; NOME                                                              ;
;   SQRT        radice quadrata                                     ;
; DESCRIZIONE                                                       ;
;   Calcola la radice quadrata di un numero naturale n (il          ;
;   quadrato di un naturale n e` la somma dei primi n               ;
;   numeri dispari.  Es.  16 = 4^2 = 1+3+5+7).                      ;
; INTERFACCIA                                                       ;
;   JSR PC, SQRT                                                    ;
;                                                                   ;
;   R0  input   il valore di cui calcolare la radice                ;
;   R0  output  resto rispetto al quadrato effettivo                ;
;   R1  output  valore intero della radice quadrata di N            ;
; USA                                                               ;
;   --                                                              ;
;-------------------------------------------------------------------;
SQRT:
        MOV     R2, -(SP)       ; salva registro
        CLR     R1
SQ1:    MOV     R1, R2          ; R2 <- contatore
        SEC
        ROL     R2              ; dispari successivo
        SUB     R2, R0
        BCS     SQ2
        INC     R1              ; contatore differenze
        BR      SQ1
SQ2:    ADD     R2, R0
        MOV     (SP)+, R2       ; ripristina registro
        RTS     PC
++++++++
```

Esercizio 3.12

Definire una routine di calcolo (approssimato) del seno di un angolo.

Viene qui proposta una soluzione semplificata di uso di tabelle di calcolo funzionale. Occorre prima di tutto osservare che e` possibile limitare la determinazione diretta della funzione seno nell'intervallo 0..(pi-greco/4) poiche` la funzione e` calcolabile negli altri intervalli mediante le relazioni:

$$\sin(x+2*pi) = \sin(x) = \sin(pi-x) = -\sin(x+pi)$$
$$\sin(pi/2-x) = \cos(x) = \mathrm{sqrt}(1-\sin^2(x))$$

Considerando inoltre che la funzione ha derivata abbastanza costante per valori di x vicini a 0, utilizzando una approssimazione lineare tra due valori, conviene suddividere l'intervallo 0..pi/4 in due parti 0..a e a..pi/4, con una maggiore densita` di campioni nel secondo intervallo.

Poiche` $pi/4 \tilde{=} 0.8$ e $\sin(pi/4) \tilde{=} 0.707$, conviene codificare l'argomento x della funzione come frazione di pi/4, in modo che all'intervallo 0..pi/4 corrispondano i valori $0..2^{16}$, secondo la formula:

$$xf = x * 65536 * 4 / pi \tilde{=} x * 83443.027$$

e codificare i valori di funzione allo stesso modo nell'intervallo 0..0.707 con la formula:

$$yf = y * 65536 * \mathrm{sqrt}(2) \tilde{=} y * 92681.9$$

Supponendo che i quanti di discretizzazione q_1 e q_2 nei due intervalli siano potenze di due e scegliendo $q_2 = 1024 = q_1/4$ $q_1 = 4096$, scegliendo inoltre a = pi/8, si hanno 8 campioni nell'intervallo $0<=x<pi/8$ e 32 campioni nell'intervallo $pi/8<=x<pi/4$.

L'approssimazione lineare si ottiene nei due intervalli con le formule:

$$yf = yc_1(i) + (yc_1(i+1)-yc_1(i)) * (xf-i*q_1) / q_1$$
$$0<=xf<7*q_1$$

$$yf = yc_1(7) + (yc_2(0)-yc_1(7)) * (xf-7*q_1) / q_1$$
$$7*q_1<=xf<8*q_1$$

$$yf = yc_2(j) + (yc_2(j+1)-yc_2(j)) * (xf-8*q_1-j*q_2) / q_2$$
$$8*q_1<=xf<8*q_1+31*q_2$$

$$yf = yc_2(31) + (65536-yc_2(31)) * (xf-8*q_1-31*q_2) / q_2$$
$$8*q_1+31*q_2<=xf<8*q_1+32*q_2$$

con $0<=i<8$ e $8<=j<32$. yc_1 e yc_2 sono i campioni memorizzati nei due intervalli prestabiliti.

Nella routine TSIN che segue si presume di ricevere gia` il valore corretto xf e di restituire il valore yf. Il primo confonto da effettuare serve a stabilire a quale dei due intervalli appar-

tiene x, cosa nelle ipotesi fatta molto semplice poiche` discriminata dal bit di segno, e quale campione i o j nei due casi scegliere. La scelta e` basata sulle diseguaglianze:

$i*q_1 <= xf < (i+1)q_1$ $\qquad$ $8*q_1+j*q_2 <= xf < 8*q_1+(j+1)*q_2$

cioe`:

$i <= xf/q_1 < i+1$ $\qquad$ $j <= (xf-8*q_1)/q_2 < j+1$

Con q_1=4096 e q_2=1024 si ha:

xf/q_1 = rshift(xf, 12)
$(xf-8*q_1)/q_2$ = rshift((xf-32768), 10)

Al lettore si suggerisce di stimare l'errore massimo che si compie nelle ipotesi fatte.

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   TSIN    funzione seno in forma tabellare                  ;
; DESCRIZIONE                                                 ;
;   Calcola la funzione seno in forma approssimata nello      ;
;   intervallo 0<=x< pi/4, diviso in due parti a densita` di  ;
;   campioni differente. Il valore di x viene fornito come    ;
;   frazione dell'intervallo stabilito mediante la formula:   ;
;   xf = x * 65536 * 4 / pi ~= x * 83443.027                  ;
;   Il valore d'uscita e` egualmente dato come frazione del   ;
;   valore corrispondente a x=pi/4 cioe` 1/sqrt(2) mediante   ;
;   la formula:                                               ;
;   yf = y * 65536 * sqrt(2) ~= y * 92681.9                   ;
; INTERFACCIA                                                 ;
;   JSR PC, TSIN                                              ;
;                                                             ;
;   R0      input   valore xf                                 ;
;   R0      output  valore yf derivato da sin(x)              ;
; USA                                                         ;
;   Subr: DMUL                                                ;
;   Memo: SINTAB SINTA2                                       ;
;-------------------------------------------------------------;
        SIGN = 100000
TSIN:
        SAVE    <R1,R2,R3,R4>   ; salva registri
        TST     R0
        BMI     TSI1            ; x >= pi/8.
        MOV     R0, R3
        BIC     #170000, R3     ; xf-i*q1
        MOV     #12., R1        ; carica contatore
TSI2:   ASR     R0              ; indice = (xf/q1)*2
        SOB     R1, TSI2
        ASL     R0
        MOV     #12., R1        ; esponente q1
        BR      TSI4

TSI1:   MOV     #10., R1        ; carica contatore
        MOV     R0, R3
        BIC     #176000, R3     ; xf-8*q1-j*q2
        BIC     #SIGN, R0       ; xf-32768.
TSI3:   ASR     R0              ; indice=(xf-32768./q2)*2+16.
```

```
        SOB     R1, TSI3
        ASL     R0
        ADD     #16., R0
        MOV     #10., R1        ; esponente q2

TSI4:   MOV     SINTAB(R0), R2  ; yc(i) o yc(j)
        TST     (R0)+           ; R0 <- R0+2
        MOV     SINTAB(R0), R4  ; yc(i+1) o yc(j+1)
        SUB     R2, R4          ; delta yc
        MOV     R2, -(SP)       ; salva yc(i) o yc(j)
        MOV     R1, -(SP)       ; salva esp. q1 o q2
        MOV     R4, R1          ; delta yc come moltiplicatore
        MOV     R3, R0          ; delta xf come moltiplicando
        JSR     PC, DMUL
        MOV     (SP)+, R1       ; ripristina esp. q1 o q2
TSI5:   ASR     R3              ; prod/q1 o prod/q2
        ROR     R2
        SOB     R1, TSI5
        ADD     (SP)+, R2       ; yf
        MOV     R2, R0
        RESTORE <R4,R3,R2,R1>   ; ripristina registri
        RTS     PC


        .CSECT  DAT
                                ; tabella seno
SINTAB: 0, 10704, 21574, 32437,
                        ; 0, 4548., 9084., 13599.
        43241, 53770, 64430, 74770
                        ; 18081., 22520., 26904., 31224.


SINTA2: 105214, 107244, 111267, 113303
                        ;35468., 36516., 37559., 38595.
        115313, 117314, 121307, 123274
                        ;39627., 40652., 41671., 42684.
        125252, 127222, 131163, 133115
                        ;43690., 44690., 45683., 46669.
        135040, 136754, 140661, 142556
                        ;47648., 48620., 49585., 50542.
        144443, 146321, 150167, 152025
                        ;51491., 52433., 53367., 54293.
        153653, 155470, 157275, 161071
                        ;55211., 56120., 57021., 57913.
        162655, 164430, 166171, 167722
                        ;58797., 59672., 60537., 61394.
        171441, 173147, 174644, 176327
                        ;62241., 63079., 63908., 64727.
++++++++
```

Esercizio 3.13

Si definiscano le subroutine di somma e sottrazione per numeri con segno rappresentati mediante le convenzioni:

a) complemento a 1
b) segno e modulo
c) eccesso 2^{15}.

---

a) Nel confronto tra rappresentazione in complemento a 2 e in complemento a 1 e` noto che la prima si ottiene dalla seconda mediante aggiunta di 1. Sommando due numeri x, y rappresentati in complemento a 1 si hanno i seguenti casi:

1) x>0, y>0 => x+y
2) x>0, y<0, x+y>0 => $x+2^{16}-1+y = 2^{16}+(x+y)-1$
3) x>0, y<0, x+y<0 => $x+2^{16}-1+y = 2^{16}-1+(x+y)$
4) x<0, y<0 => $2^{16}-1+x+2^{16}-1+y =$
$= 2^{16}+(2^{16}-1+(x+y))-1$

Quindi, per ottenere in complemento a 1 la corretta rappresentazione della somma di due numeri, e` necessario alcune volte aggiungere 1 alla somma delle rappresentazioni in complemento a 1 dei numeri stessi e precisamente nei casi 2 e 4 (trascurando gli eventuali riporti). L'algoritmo completo e` il seguente:

a) sommare i due operandi
b) se vi e` un riporto dal bit di segno, sommare 1
c) ispezionare al solito i riporti nel e dal bit di segno per la determinazione della condizione di overflow.

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   CM1ADD  somma in complemento a 1                          ;
; DESCRIZIONE                                                 ;
;   Effettua la somma tra due numeri in complemento a 1 e     ;
;   verifica la eventuale condizione di overflow. Se corretto;
;   converte ad unica rappresentazione lo zero.               ;
; INTERFACCIA                                                 ;
;   JSR PC, CM1ADD                                            ;
;                                                             ;
;   R0      input   operando 1                                ;
;   R1      input   operando 2                                ;
;   R0      output  risultato                                 ;
;   V       output  = 1 se overflow                           ;
;   Z       output  = 1 se risultato 0                        ;
; USA                                                         ;
;   --                                                        ;
;-------------------------------------------------------------;
CM1ADD:
        CLR     -(SP)           ; azzera memoria riporti
        ADD     R1, R0
        BVC     CM11            ; salta se 0 o 2 riporti
        INC     (SP)            ; un riporto
        CLV                     ; V=0
CM11:   BCC     CM12            ; salta se non riporto da b15
        INC     R0              ; somma 1
        BVC     CM12            ; salta se non overflow
        INC     (SP)            ; un riporto
CM12:   CMP     (SP)+, #1       ; un solo riporto
        BEQ     CM13            ; salta se overflow
        COM     R0              ; unica rappr. dello zero
        BEQ     CM14            ; salta se zero, V=0, Z=1
        COM     R0              ; non zero, V=0, Z=0
CM14:   RTS     PC
CM13:   TST     R0              ; imposta Z e N
        SEV                     ; V=1
        RTS     PC
```

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   CM1SUB  sottrazione in complemento a 1                    ;
; DESCRIZIONE                                                 ;
;   Effettua la diff. tra due numeri in complemento a 1 e     ;
;   verifica la eventuale condizione di overflow. Se corretto;
;   converte ad unica rappresentazione lo zero.               ;
; INTERFACCIA                                                 ;
;   JSR PC, CM1SUB                                            ;
;                                                             ;
;   R0      input   operando 1                                ;
;   R1      input   operando 2                                ;
;   R0      output  risultato                                 ;
;   V       output  = 1 se overflow                           ;
;   Z       output  = 1 se risultato 0                        ;
; USA                                                         ;
;   Subr: CM1ADD                                              ;
;-------------------------------------------------------------;
CM1SUB:
        COM     R1              ; com(y)
        JMP     CM1ADD
```

b) Nella rappresentazione segno e modulo, il bit piu` significativo rappresenta ancora il bit di segno ma i restanti bit rappresentano il valore assoluto del numero. In questa notazione e` quindi necessario stabilire per confronto di segni se eseguire la somma o la sottrazione tra i moduli. Se il risultato e` negativo, deve essere complementato e si deve successivamente complementare il bit di segno. In questa notazione, come in quella in complemento a 1, lo zero ha una doppia rappresentazione.

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   SMADD   somma in segno e modulo                           ;
; DESCRIZIONE                                                 ;
;   Effettua la somma tra due numeri in segno e modulo e      ;
;   verifica la eventuale condizione di overflow. Se corretto;
;   converte ad unica rappresentazione lo zero.               ;
; INTERFACCIA                                                 ;
;   JSR PC, SMADD                                             ;
;                                                             ;
;   R0      input   operando 1                                ;
;   R1      input   operando 2                                ;
;   R0      output  risultato                                 ;
;   V       output  = 1 se overflow                           ;
;   Z       output  = 1 se risultato 0                        ;
; USA                                                         ;
;   --                                                        ;
;-------------------------------------------------------------;
        LOWBY   = 177400        ; maschera byte meno significativo
        SIGN    = 100000        ; maschera bit di segno
SMADD:
        TST     R0
        BPL     SMA1            ; salta se op1 positivo
        BIC     #SIGN, R0       ; op1 negativo
        NEG     R0
SMA1:
        TST     R1
        BPL     SMA2            ; salta se op2 positivo
        BIC     #SIGN, R1       ; op2 negativo
```

```
        NEG     R1
SMA2:
        ADD     R1, R0
        BVS     SMA3            ; salta se overflow
        CMP     R0, #SIGN       ; -2**15 non rappresentabile
        BEQ     SMA3
        TST     R1
        BPL     SMA6            ; salta se R1 positivo
        NEG     R1
        BIS     #SIGN, R1       ; ripristina R1
SMA6:   TST     R0
        BPL     SMA4            ; salta se ris. >=0, Z=*
        NEG     R0              ; risultato negativo
        BIS     #SIGN, R0       ; V=0, Z=0
SMA4:   RTS     PC
SMA3:   TST     R1
        BPL     SMA5            ; salta se R1 positivo
        NEG     R1
        BIS     #SIGN, R1       ; ripristina R1
SMA5:   SEV
        RTS     PC

;-------------------------------------------------------------;
; NOME                                                        ;
;   SMSUB   sottrazione in segno e modulo                     ;
; DESCRIZIONE                                                 ;
;   Effettua la diff. tra due numeri in segno e modulo e      ;
;   verifica la eventuale condizione di overflow. Se corretto;
;   converte ad unica rappresentazione lo zero.               ;
; INTERFACCIA                                                 ;
;   JSR PC, SMSUB                                             ;
;                                                             ;
;   R0      input   operando 1                                ;
;   R1      input   operando 2                                ;
;   R0      output  risultato                                 ;
;   V       output  = 1 se overflow                           ;
;   Z       output  = 1 se risultato 0                        ;
; USA                                                         ;
;   Subr: SMADD                                               ;
;-------------------------------------------------------------;
SMSUB:
        TST     R1
        BMI     SMS1            ; salta se op2 negativo
        BIS     #SIGN, R1       ; -op2
        JSR     PC, SMADD
        MFPS    -(SP)           ; salva flag
        BIC     #SIGN, R1       ; op2
        MTPS    (SP)+           ; ripristina flag
        RTS     PC
SMS1:   BIC     #SIGN, R1       ; -op2
        JSR     PC, SMADD
        MFPS    -(SP)           ; salva flag
        BIS     #SIGN, R1       ; op2
        MTPS    (SP)+           ; ripristina flag
        RTS     PC
```

c) La notazione eccesso 2^{m-1} rappresenta con m bit tutti i numeri $-2^{m-1} <= n <= 2^{m-1}-1$ con e(n, m):

$$e(n, m) = n+2^{m-1} \qquad 0 <= n+2^{m-1} <= 2^m-1$$

cioe`:

$n>=0 \quad e(n, m) = n+2^{m-1} = n \mid 2^{m-1}$
$n<0 \quad e(n, m) = (2^m+n)-2^{m-1} = neg(n) \& (2^{m-1}-1)$
$neg(e(n, m)) = 2^m - 2^{m-1} - n = 2^{m-1}-n = e(-n, m)$

Quindi questa notazione differisce dal complemento a due solo perche` il significato del bit di segno e` invertito. Di questo fatto occorre tener conto per le correzioni da effettuare sul risultato della somma e per la verifica di overflow. Infatti, nel campo di rappresentabilita` dei numeri in eccesso 2^{15}, che coincide con quello della notazione complemento a due, l'overflow nella somma si verifica se:

$2^{16}-2 >= x+y >= 2^{15}$ $(x>0, y>0)$ oppure
$-2^{16} <= x+y < -2^{15}$ $(x<0, y<0)$

che per la rappresentazione in eccesso si traducono in:

$2^{17}-2 >= (x+2^{15})+(y+2^{15}) >= 2^{15}+2^{16}$
$0 <= (x+2^{15})+(y+2^{15}) < 2^{15}$

Pertanto l'equivalente della condizione di overflow nella somma eccesso 2^{15} e`:

C=1 N=1 oppure C=0 N=0 ovvero C # N = 0.

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   EXADD   somma in eccesso 2'15                             ;
; DESCRIZIONE                                                 ;
;   Effettua la somma tra due numeri in eccesso 2'15 e        ;
;   verifica la eventuale condizione di overflow.             ;
; INTERFACCIA                                                 ;
;   JSR PC, EXADD                                             ;
;                                                             ;
;   R0      input   operando 1                                ;
;   R1      input   operando 2                                ;
;   R0      output  risultato                                 ;
;   V       output  = 1 se overflow                           ;
;   Z       output  = 1 se risultato 0                        ;
; USA                                                         ;
;   --                                                        ;
;-------------------------------------------------------------;
EXADD:
        ADD     R1, R0
        BCS     EXA1            ; salta se C=1
        BPL     EXA2            ; salta se N=0, overflow
EXA3:   ADD     #SIGN, R0       ; no overflow
        CMP     #SIGN, R0       ; confronta con zero
        CLV                     ; V=0, Z=*
        RTS     PC
EXA1:   BPL     EXA3            ; salta se N=0, no overflow
EXA2:   SEV                     ; V=1
        RTS     PC


;-------------------------------------------------------------;
; NOME                                                        ;
;   EXSUB   sottrazione in eccesso 2'15                       ;
```

```
; DESCRIZIONE                                                    ;
;   Effettua la diff. tra due numeri in eccesso 2^15 e           ;
;   verifica la eventuale condizione di overflow.                ;
; INTERFACCIA                                                    ;
;   JSR PC, EXSUB                                                ;
;                                                                ;
;   R0      input   operando 1                                   ;
;   R1      input   operando 2                                   ;
;   R0      output  risultato                                    ;
;   V       output  = 1 se overflow                              ;
;   Z       output  = 1 se risultato 0                           ;
; USA                                                            ;
;   Subr: EXADD                                                  ;
;--------------------------------------------------------------;
EXSUB:
        NEG     R1              ; neg(e(n,m)) = e(-n,m)
        JSR     PC, EXADD
        MFPS    -(SP)           ; salva flag
        NEG     R1
        MTPS    (SP)+           ; ripristina flag
        RTS     PC
++++++++
```

Esercizio 3.14

Si definiscano una o piu` tabelle che consentano di accedere ai singoli elementi di un vettore multidimensionale in modo indiretto allo scopo di evitare le moltiplicazioni presenti nel calcolo della funzione di mappa. Si scrivano subroutine di accesso che utilizzino le tabelle.

Si ricordi che per un vettore a n dimensioni d_1, d_2,...,d_n, la funzione di mappa per l'elemento di estensione e di indici $i_1 \ldots i_n$ con $0<=i_k<=d_k-1$ e` la seguente:

$$Adr = VECT + i_1*d_2*d_3*\ldots*d_n*e + i_2*d_3*\ldots*d_n*e + \ldots + i_{n-1}*d_n*e + i_n*e$$

Si tratta di costruire inizialmente n tabelle di dimensioni d_1, d_2, ... d_n in modo che, accedendovi rispettivamente con gli indici i_1, i_2, ..., i_n, si possano ottenere direttamente gli offset da sommare nella espressione della funzione. La prima tabella, quella cioe` a cui si accede con l'indice $i_1$, puo` contenere in alternativa gli offset sommati all'indirizzo base VECT. Quando gli elementi hanno estensione atomica (1 o 2) l'ultima tabella puo` essere sostituita da un semplice calcolo. In questo caso e in particolare con n=2, la tabella di indirezione e` unica e puo` essere calcolata con facilita`.

```
;--------------------------------------------------------------;
; NOME                                                         ;
;   IVINI   costruzione tabella di indirezione per vettori     ;
; DESCRIZIONE                                                  ;
;   Provvede a inizializzare una tabella di indirezione per    ;
;   l'accesso a vettori multidimensionali. Ogni elemento       ;
;   della tabella fornisce l'offset del piano di dimensioni    ;
;   dk*dk+1*...*dn. Per la prima tabella e` possibile dare     ;
;   un offset iniziale rappresentato dall'indirizzo iniziale   ;
;   del vettore. Non viene fatto alcun controllo su overflow.  ;
```

```
; INTERFACCIA                                                    ;
;   JSR PC, IVINI                                                ;
;                                                                ;
;   R0      input   numero elementi tabella                      ;
;   R1      input   estensione piani riferiti (p)                ;
;   R2      input   puntatore vettore (VECT) (prima tabella)     ;
;   R3      input   puntatore tabella (ITAB)                     ;
;   R4      input   molteplicita` elementi (e) in byte           ;
; USA                                                            ;
;   Subr: SMUL                                                   ;
;--------------------------------------------------------------;
IVINI:
        SAVE    <R0,R1,R2,R3,R4>; salva registri
        MOV     R4, R0          ; e come moltiplicando
                                ; est. piano come
                                ; moltiplicatore
        CMP     #1, R0
        BEQ     IVI1            ; salta se prodotto inutile
        JSR     PC, SMUL
IVI1:   MOV     R1, R4          ; R4 <- p*e
        MOV     10(SP), R0      ; ripristina numero elementi
IVI2:   MOV     R2, (R3)+       ; carica offset
        ADD     R4, R2          ; somma ampiezza piano
        SOB     R0, IVI2        ; ciclo su numero elementi
        RESTORE <R4,R3,R2,R1,R0>; ripristina registri
        RTS     PC

;--------------------------------------------------------------;
; NOME                                                         ;
;   IVRD    accesso mediante tabella di indirezione            ;
; DESCRIZIONE                                                  ;
;   Provvede a sommare ad un indirizzo fornito l'offset letto;
;   da tabella di indirezione.                                 ;
; INTERFACCIA                                                  ;
;   JSR PC, IVRD                                               ;
;                                                              ;
;   R0      input   indice                                     ;
;   R1      input   puntatore tabella indirezione (ITAB)       ;
;   R2      input   indirizzo da sommare                       ;
;   R2      output  nuovo indirizzo                            ;
; USA                                                          ;
;   --                                                         ;
;--------------------------------------------------------------;
IVRD:
        MOV     R1, -(SP)       ; salva registro
        ADD     R0, R1
        ADD     R0, R1          ; ITAB+2*i
        ADD     (R1), R2
        MOV     (SP)+, R1       ; ripristina registro
        RTS     PC
```

Ad esempio, per un vettore VE[5, 7] di word, si inizializza il vettore VEI[5] e si accede all'elemento VE[2, 4] nel modo seguente:

```
;*****************;*****************************************;
;       esempio di prova
;
;       VE[5, 7] di word        VEI[5]
;
```

```
PRO1:   MOV     #5, R0          ; VEI[5]
        MOV     #7, R1
        MOV     #VE, R2
        MOV     #VEI, R3
        MOV     #2, R4
        JSR     PC, IVINI
        MOV     #2, R0
        MOV     #VEI, R1
        CLR     R2
        JSR     PC, IVRD
        MOV     #4, R0
        ASL     R0              ; elemento word
        ADD     R0, R2
        MOV     #6, (R2)        ; VE[2, 4] <- 6


        .CSECT  DAT
VE:     .BLKW   5*7
VEI:    .BLKW   5
```

Analogamente per un vettore VC[5, 4, 3] di record di 5 byte, occorre produrre le tabelle VCI1[5], VCI2[4] e VCI3[3] e l'uso delle subroutine definite puo` essere esemplificato dall'accesso all'elemento VC[1, 3, 2].

```
;*************************************************************;
;       esempio di prova
;
;   VC[5, 4, 3] di record da 5 byte VCI1[5], VCI2[4], VCI3[3]
;
PRO2:   MOV     #5, R0          ; VCI1[5]
        MOV     #4*3, R1
        MOV     #VC, R2
        MOV     #VCI1, R3
        MOV     #5, R4
        JSR     PC, IVINI
        MOV     #4, R0          ; VCI2[4]
        MOV     #3, R1
        CLR     R2
        MOV     #VCI2, R3
        JSR     PC, IVINI
        MOV     #3, R0          ; VCI3[3]
        MOV     #5, R1
        CLR     R2
        MOV     #1, R4
        MOV     #VCI3, R3
        JSR     PC, IVINI
        MOV     #1, R0          ; R2 = Adr(VC[1, 3, 2])
        MOV     #VCI1, R1
        CLR     R2
        JSR     PC, IVRD
        MOV     #3, R0
        MOV     #VCI2, R1
        JSR     PC, IVRD
        MOV     #2, R0
        MOV     #VCI3, R1
        JSR     PC, IVRD
```

```
        .CSECT  DAT
VC:     .BLKB   5*4*3*5
        .EVEN
VCI1:   .BLKW   5
VCI2:   .BLKW   4
VCI3:   .BLKW   3
```

Questa tecnica puo` sostituire quella che fa uso di macro di accesso, vista in precedenza, nell'ipotesi che le dimensioni del vettore non siano costanti ma debbano venire calcolate al tempo di esecuzione. In generale, la tecnica mediante macro, non comportando chiamate a subroutine, produce un codice piu` efficiente.
++++++++

Esercizio 3.15

Si definisca una subroutine in grado di valutare il numero di bit pari a 1 in un word.

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   NBIT1       subroutine calcolo numero di bit pari a 1     ;
; DESCRIZIONE                                                 ;
;   Fornisce il numero di bit pari a 1 della                  ;
;   rappresentazione binaria del numero fornito.              ;
; INTERFACCIA                                                 ;
;   JSR PC, NBIT1                                             ;
;                                                             ;
;   R0      input   word fornito                              ;
;   R1      output  numero di bit pari a 1                    ;
; USA                                                         ;
;   --                                                        ;
;-------------------------------------------------------------;
NBIT1:
        MOV     R2, -(SP)       ; salva registro
        MOV     #16., R2        ; carica contatore
        CLR     R1              ; inizializza conteggio
L1:     ROR     R0              ; shift LSB in carry
        BCC     NOINC           ; bit = 1 ?
        INC     R1              ; si`
NOINC:  SOB     R2, L1
        ROR     R0              ; R0 non modificato
        MOV     (SP)+, R2       ; ripristina registro
        RTS     PC
++++++++
```

3.3 - Applicazioni non numeriche

In questo paragrafo verranno illustrate alcune routine di elaborazione non numerica per lo piu` concernenti la gestione di strutture di dati. Naturalmente, per gli scopi di questo testo, ci si limitera` ad alcuni esempi semplificati, limitatamente alle applicazioni che meglio si prestano ad una realizzazione in linguaggio assembly.

Esercizio 3.16

Si scriva una subroutine di *hashing* che, a partire da una stringa non vuota, calcola il valore hash associato come somma dei codici ASCII dei caratteri componenti la stringa, modulo 6 bit. Si definisca inoltre una subroutine che tenga aggiornata una tabella di stringhe, nella quale ciascuna stringa compare una sola volta. La tabella ha dimensione prefissata (2^6 stringhe di 8 caratteri), pertanto vengono inseriti al piu` i primi 8 caratteri di ciascuna stringa. La routine restituisce l'indice di inserimento nella tabella di stringhe della stringa fornita come parametro.

La prima subroutine (HASH) si realizza facilmente trascurando i riporti nella somma dei codici dei caratteri (8 al massimo) e infine azzerando i 10 bit piu` significativi della somma. Viene anche realizzata una routine di inizializzazione (HINIT) che pone a 0 tutti i puntatori nella tabella hash e inizializza il puntatore corrente di inserzione nella tabella di stringhe.

La routine HINS e` organizzata nel modo seguente: sulla base del valore hash associato alla stringa fornita, effettua un accesso alla tabella hash: se la locazione e` vuota, significa che nessuna stringa con il valore hash appena calcolato e` stata ancora inserita e pertanto la locazione assume il valore corrente del puntatore di inserimento in tabella stringhe e la stringa viene in quest'ultima inserita, con l'aggiornamento del puntatore. Se la locazione in tabella hash non e` vuota, si fa un confronto con la stringa associata e se eguale alla stringa fornita, ci si limita a restituirne l'indice di inserimento. Viceversa si passa alla successiva locazione della tabella hash (in senso circolare) e il procedimento si ripete e ha termine o perche` la stringa viene trovata, o perche` viene inserita una prima volta o perche` e` stata esaurita tutta la tabella. Nei primi due casi viene restituito l'indice di inserimento (1..64.).

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   HASH    funzione di hashing                               ;
; DESCRIZIONE                                                 ;
;   Calcola il valore di hash associato ad una stringa come   ;
;   somma modulo 6 bit dei suoi primi 8 caratteri.            ;
; INTERFACCIA                                                 ;
;   JSR PC, HASH                                              ;
;                                                             ;
;   R1  input   puntatore stringa                             ;
;   R0  output  valore di hash                                ;
; USA                                                         ;
;   --                                                        ;
;-------------------------------------------------------------;
        EOS =   0
HASH:
        SAVE    <R1,R2,R3>          ; salva registri
        MOV     #8., R3             ; iniz. contatore
        CLR     R0                  ; iniz. hash
HA1:    MOVB    (R1)+, R2
        CMPB    #EOS,R2             ; fine stringa ?
        BEQ     HA2                 ; salta se si`
        ADD     R2, R0              ; somma carattere
```

```
        SOB     R3, HA1
HA2:    BIC     #177700, R0     ; modulo 6 bit
        RESTORE <R3,R2,R1>      ; ripristina registri
        RTS     PC

;-------------------------------------------------------------;
; NOME                                                        ;
;   HINIT   inizializzazione tabella hash                     ;
; DESCRIZIONE                                                 ;
;   Provvede a inizializzare la tabella hash HTAB e           ;
;   la variabile di stato INX che rappresenta il puntatore    ;
;   corrente di inserzione nella tabella di stringhe.         ;
; INTERFACCIA                                                 ;
;   JSR PC, HINIT                                             ;
; USA                                                         ;
;   Memo: HTAB AHTAB INX                                      ;
;-------------------------------------------------------------;
HINIT:
        MOV     R1, -(SP)       ; salva registro
        MOV     #AHTAB, INX     ; iniz. puntatore corrente
        MOV     #HTAB, R1       ; carica ind. tabella hash
HI1:    CLR     (R1)+           ; pone 0
        CMP     R1, #HTAB+128.  ; fine tabella ?
        BNE     HI1             ; ciclo se no
        MOV     (SP)+, R1       ; ripristina registro
        RTS     PC

;-------------------------------------------------------------;
; NOME                                                        ;
;   HINS    inserzione di stringa mediante hash               ;
; DESCRIZIONE                                                 ;
;   Provvede a inserire una stringa, se non gia` presente,    ;
;   nella tabella predisposta, restituendo comunque l'indice  ;
;   di inserzione.                                            ;
; INTERFACCIA                                                 ;
;   JSR PC, HINS                                              ;
;                                                             ;
;   R0      output  indice di inserzione o 0 se tabella piena;
;   R1      input   puntatore stringa                         ;
; USA                                                         ;
;   Subr: HASH STNCMP STNCPY                                  ;
;   Memo: HTAB AHTAB INX                                      ;
;-------------------------------------------------------------;
HINS:
        SAVE    <R2,R3,R4>      ; salva registri
        JSR     PC, HASH        ; calcola valore hash
        MOV     #64., R4        ; dimensione tabella hash
        ASL     R0              ; offset tabella
        MOV     R0, R3
        MOV     #8., R0         ; n per confronto stringhe
HS1:    MOV     HTAB(R3), R2    ; carica contenuto tabella h.
        BEQ     HS3             ; entry vuoto, stringa non
                                ; trovata
        JSR     PC, STNCMP      ; confronta stringhe
        BEQ     HS2             ; salta se trovata
        ADD     #2, R3
        CMP     R3, #128.       ; fine tabella ?
        BLT     HS4
        CLR     R3              ; tabella circolare
HS4:    SOB     R4, HS1         ; ciclo su totale elementi
```

```
        CLR     R0              ; tabella piena e non trovata
        BR      HS5
HS3:    MOV     INX, R2         ; non trovata, inserisce
        JSR     PC, STNCPY      ; copia fino 8 caratteri
        MOV     R2, HTAB(R3)    ; inser. punt. in tabella h.
        ADD     #8., INX        ; aggiorna puntatore
HS2:    SUB     #AHTAB, R2      ; indice di inserimento
        MOV     R2, R0
        ASR     R0              ; offset/8
        ASR     R0
        ASR     R0
        INC     R0              ; 1..64
HS5:    RESTORE <R4,R3,R2>      ; ripristina registri
        RTS     PC

        .CSECT  DAT
HTAB:   .BLKW   64.
INX:    .WORD   0
AHTAB:  .BLKW   64.*8.
```

Si suggerisce al lettore di modificare le routine in modo che effettuino le medesime operazioni su tabelle generiche.
++++++++

Esercizio 3.17

Si scriva una subroutine che effettua una ricerca con il metodo della bisezione su un vettore di byte di dimensioni qualsiasi. Successivamente, utilizzando tale subroutine, si definisca una subroutine che tiene aggiornato un vettore di byte in modo che i suoi elementi siano ordinati in senso crescente e non ripetuti.

Il metodo della bisezione costituisce un fondamentale metodo di ricerca su archivio ordinato ad accesso casuale. Esso e` basato sul dimezzamento, ad ogni ciclo, dell'intervallo di incertezza che inizialmente e` pari alla dimensione della tabella. Il dimezzamento e` effettivo se la tabella ha ampiezza $2^n-1$: in ogni caso l'algoritmo garantisce che in circa log2(dim) cicli il valore cercato, se presente, venga trovato. In linguaggio ad alto livello, l'algoritmo si puo` esprimere nel modo seguente, supponendo di chiamare a[] il vettore di ricerca, v il valore da ricercare, l e r due indici compresi nell'intervallo 0..dim-1:

```
l:=0; r:=dim-1;
repeat
    x := (l+r) div 2;
    if v < a[x] then
        r := x-1
    else
        l := x+1
until (v = a[x]) or (l > r);
```

Se il valore viene trovato, x ne rappresenta la posizione, altrimenti il valore di x rappresenta la posizione dell'elemento presente immediatamente superiore o inferiore all'elemento cercato: quale dei due casi si verifichi dipende dalle condizioni iniziali.

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   BINSEA  ricerca mediante bisezione                        ;
; DESCRIZIONE                                                 ;
;   Provvede a ricercare il valore fornito in un vettore di   ;
;   di byte ordinato in senso crescente. Se trova il valore,  ;
;   restituisce la posizione relativa, altrimenti restituisce;
;   l'indice del minimo elemento presente maggiore del valore;
;   cercato oppure l'elemento massimo minore del valore.      ;
; INTERFACCIA                                                 ;
;   JSR PC, BINSEA                                            ;
;                                                             ;
;   R0.l     input   valore da ricercare                      ;
;   R1       input   puntatore vettore                        ;
;   R2       input   dimensione vettore                       ;
;   R3       output  posizione restituita                     ;
; USA                                                         ;
;   --                                                        ;
;-------------------------------------------------------------;
BINSEA:
        SAVE    <R0,R2,R4,R1>   ; salva registri
        CLR     R4              ; R4=l, R2=r
        DEC     R2              ; l=0, r=dim-1
BIS1:   MOV     R4, R1
        ADD     R2, R1
        ASR     R1              ; x = (l+r) div 2
        MOV     R1, R3
        ADD     (SP), R1        ; somma offset a puntatore
        CMPB    R0, (R1)        ; confronto
        BEQ     BIS2            ; trovato
        BGT     BIS3            ; salta se v > a[x]
        DEC     R3              ; v < a[x]
        MOV     R3, R2          ; r <- x-1
        BR      BIS4
BIS3:   INC     R3              ; v > a[x]
        MOV     R3, R4          ; l <- x+1
BIS4:   CMP     R4, R2
        BLE     BIS1            ; salta se l<=r
BIS2:   RESTORE <R1,R4,R2,R0>   ; ripristina registri
        RTS     PC
```

La seconda routine richiesta costituisce un esempio di ordinamento di un vettore mediante inserzione. Infatti ad ogni passo, il valore viene inserito nella posizione corretta, spostando di una posizione in avanti tutti gli elementi maggiori. La posizione di inserimento viene calcolata mediante la routine precedente: l'elemento viene inserito solo se non gia` presente. Come sopra detto, e` in questo caso necessario effettuare un confronto aggiuntivo per stabilire qual'e` il primo elemento della successione da spostare. Questo test e` inutile se la posizione restituita e` -1 (sono da spostare tutti gli elementi correntemente presenti) oppure e` pari alla dimensione corrente (il valore deve essere inserito come ultimo).

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   BIINS   inserzione ordinata mediante bisezione            ;
; DESCRIZIONE                                                 ;
;   Provvede a inserire un valore, se non gia` presente,      ;
;   nel vettore fornito in modo da mantenerlo ordinato in     ;
```

```
;   senso crescente.                                                ;
; INTERFACCIA                                                       ;
;   JSR PC, BIINS                                                   ;
;                                                                   ;
;   R0.1     input   valore da inserire                             ;
;   R1       input   puntatore vettore                              ;
;   R2       input   dimensione corrente                            ;
;   R3       input   dimensione massima                             ;
;   R2       output  dimensione corrente                            ;
;   R3       output  posizione di inserimento o -1 se pieno         ;
; USA                                                               ;
;   Subr: BINSEA                                                    ;
;-------------------------------------------------------------------;
BIINS:
        CMP     R2, R3
        BHIS    BII1            ; overflow
        TST     R2              ; dimensione corrente
        BNE     BII2            ; salta se >0
        MOVB    R0, (R1)        ; inserisci come primo
        CLR     R3              ; punto iniziale
        INC     R2              ; incrementa dimensione
        RTS     PC
BII2:
        JSR     PC, BINSEA      ; ricerca
        MOV     R1, -(SP)       ; salva puntatore
        TST     R3
        BPL     BII3            ; salta se >=0
        INC     R3
        BR      BII5
BII3:   ADD     R3, R1          ; punta a posizione trovata
        CMP     R3, R2          ; confronta con dim
        BGE     BII7            ; salta se >= dim
        CMPB    R0, (R1)        ; confronto
        BEQ     BII4            ; gia` presente
        BLT     BII5            ; salta se ok
        INC     R3              ; posizione successiva
BII5:   MOV     (SP), R1        ; ricarica puntatore
        MOV     R3, -(SP)       ; salva posizione
        SUB     R2, R3
        NEG     R3              ; dim-x
        ADD     R2, R1          ; oltre fine tabella
BII6:   MOVB    -1(R1), (R1)    ; shift una posizione in basso
        DEC     R1
        SOB     R3, BII6
        MOV     (SP)+, R3       ; ripristina registro
BII7:   MOVB    R0, (R1)        ; inserisce elemento
        MOV     (SP)+, R1       ; ripristina registro
        INC     R2              ; incrementa dimensione
        RTS     PC
BII4:   MOV     (SP)+, R1       ; ripristina registro
        RTS     PC              ; gia` presente
BII1:   MOV     #-1, R3         ; overflow tabella
        RTS     PC
```

Si suggerisce al lettore di definire un segmento di prova che, ad esempio, chiami la routine BIINS con tabella di dimensione massima 5 e tentando di inserire i valori 7, 5, 7, 0, 4, 11, 6. Si segua l'esecuzione della routine.

++++++++

Esercizio 3.18

Si scriva una subroutine in grado di impostare a 0 o a 1 un singolo bit di un word.

Le istruzioni BIC e BIS consentono di azzerare e impostare a 1 singoli bit di un word, anche per gruppi, ma non consentono direttamente di indirizzare per indice un singolo bit. A questo provvede la subroutine SRBIT che si commenta da sola.

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   SRBIT   Set o reset singolo bit                           ;
; DESCRIZIONE                                                 ;
;   Permette di porre a 1 o 0 un singolo bit di una parola    ;
;   selezionato mediante indice.                              ;
; INTERFACCIA                                                 ;
;   JSR PC, SRBIT                                             ;
;                                                             ;
;   R0       input   valore da modificare                     ;
;   R1.1     input   indice bit da modificare (0..15.)        ;
;   b15(R1)  input   1 se set, 0 se reset                     ;
;   R0       output  valore modificato                        ;
; USA                                                         ;
;   --                                                        ;
;-------------------------------------------------------------;
        SIGN = 100000           ; maschera per bit di segno
        BIT0 = 1                ; maschera bit 0
SRBIT:
        MOV     R2, -(SP)       ; salva registri
        MOV     R3, -(SP)
        MOV     R1, R2          ; carica contatore
        MOV     #BIT0, R3       ; iniz. maschera
        BIC     #SIGN, R2
        BEQ     SRB1            ; salta se bit 0, no shift
SRB2:   ASL     R3              ; shift a sinistra maschera
        SOB     R2, SRB2        ; ciclo su indice bit
SRB1:   TST     R1              ; set o reset
        BMI     SRB3            ; salta se set
        BIC     R3, R0          ; reset
        BR      SRB4
SRB3:   BIS     R3, R0          ; set
SRB4:   MOV     (SP)+, R3       ; ripristina registri
        MOV     (SP)+, R2
        RTS     PC
++++++++
```

Esercizio 3.19

Scrivere una subroutine che gestisce un contatore in codice Gray a n bit (due configurazioni successive variano per un solo bit).

Il codice Gray possiede, come accennato, la proprieta` che due valori successivi differiscono solo per un bit. Un modo "grafico" di ottenere la successione con n bit, e` quello di applicare la regola della specularita`: a partire da una sottosuccessione di 2^m valori, la successiva sottosuccessione della stessa ampiezza si ottiene

variando il bit di indice m con 0<=m<=n-1 da 0 a 1 e replicando in forma speculare i rimanenti bit rispetto alla successione precedente. Nel caso n=4 la successione e` qui sotto presentata:

```
3 2 1 0
-------
0 0 0 0
        <- sottosucc. 2^0
0 0 0 1
        <-- sottosucc. 2^1
0 0 1 1
0 0 1 0
        <--- sottosucc. 2^2
0 1 1 0
0 1 1 1
0 1 0 1
0 1 0 0
        <---- sottosucc. 2^3
1 1 0 0
1 1 0 1
1 1 1 1
1 1 1 0
1 0 1 0
1 0 1 1
1 0 0 1
1 0 0 0
```

Definendo con n(a, b) il numero di bit pari a b nella rappresentazione binaria di a e con f(a, m) il valore del bit di indice m nel successore di a in codice Gray, si possono verificare facilmente le seguenti eguaglianze:

$$f(a,0) = \begin{cases} \tilde{a}_0 & \text{se } n(a,1) \text{ e` pari} \\ a_0 & \text{se } n(a,1) \text{ e` dispari} \end{cases}$$

$$f(a,1) = \begin{cases} \tilde{a}_1 & \text{se } n(a,1) \text{ e` dispari e } a_0 = 1 \\ a_1 & \text{altrimenti} \end{cases}$$

...

$$f(a,k) = \begin{cases} \tilde{a}_k & \text{se } n(a,1) \text{ e` dispari, } a_{k-1} = 1 \text{ e } a_{k-2},\ldots,a_0 = 0 \text{ con } k>=2 \\ a_k & \text{altrimenti} \end{cases}$$

Sulla base di esse e` possibile calcolare il successore nella sequenza di un qualsiasi valore fornito.

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   GRAY    generatore in codice Gray                         ;
; DESCRIZIONE                                                 ;
;   Provvede a fornire il successore nella codifica Gray a    ;
;   n bit di un valore fornito.                               ;
; INTERFACCIA                                                 ;
;   JSR PC, GRAY                                              ;
;                                                             ;
;   R0      input   valore fornito                            ;
;   R4      input   16.-n+1 (n numero di bit della codifica)  ;
; USA                                                         ;
;   Subr: NBIT1                                               ;
;-------------------------------------------------------------;
        BIT0 = 1
```

```
GRAY:
        SAVE    <R1,R2,R3>      ; salva registri
        MOV     #BIT0, R3
        JSR     PC, NBIT1       ; n(R0, 1) in R1
        MOV     R1, R2
        ROR     R2              ; e` pari ?
        BCS     GRY1            ; salta se no
        XOR     R3, R0          ; pari, varia a0
        BR      GRY2
GRY1:   MOV     #16., R2        ; dispari, varia ak
GRY3:   ROR     R0              ; cerca primo 1 cioe` ak-1
        BCS     GRY4
        SOB     R2, GRY3
GRYER:  HALT                    ; errore
GRY4:   CMP     R2, R4          ; ak-1=an-1 ?
        CLC                     ; se si, ritorna a 0
        BEQ     GRY5            ; salta se si
        XOR     R3,R0           ; complementa ak, C=1
        SEC
GRY5:   ROR     R0              ; ripristina R0
        SOB     R2, GRY5        ; ciclo bit rimanenti
GRY2:   RESTORE <R3,R2,R1>      ; ripristina registri
        RTS     PC
```

E` possibile che il programma raggiunga l'istruzione all'indirizzo rappresentato da GRYER?
++++++++

Esercizio 3.20

Si voglia disporre della gestione di un pool dimamico, mediante il quale e` possibile allocare e rilasciare, in fase di esecuzione dei programmi, aree di memoria. Allo scopo, debbano essere rese disponibili le seguenti routine:

```
PLINIT:     inizializza il pool;
POLMAX:     restituisce l'ampiezza della piu` grande area libera;
ALLOC:      alloca un'area di dimensione almeno pari a quella
            richiesta;
FREE:       rilascia un'area precedentemente allocata.
```

La gestione del pool deve permettere il recupero delle aree rilasciate mediante compattamento di aree contigue libere.

Anche in questo caso, di un problema per il quale si sono offerte soluzioni varie ed anche sofisticate, ne viene fornita una abbastanza semplice a titolo esemplificativo.

Il pool viene suddiviso in aree, collegate sottoforma di lista concatenata mediante una intestazione inserita prima di ciascuna area. L'intestazione contiene un puntatore all'area successiva (subito dopo la relativa intestazione) e un valore numerico di cui i 15 bit meno significativi rappresentano l'estensione dell'area (esclusa l'intestazione) e il bit piu` significativo viene posto a 1 se l'area risulta in uso (allocata). L'ultima intestazione contiene un puntatore nullo. Una intestazione puo` pertanto essere descritta nel modo seguente:

```
type    POOLHEAD = record
        NEXT: ^ Byte;
        SIZE: integer
        end
```

Inizialmante il pool e` costituito da una sola area di dimensioni prefissate. A fronte di una richiesta di allocazione, si effettua una ricerca dell'area libera di minima dimensione maggiore o uguale a quella richiesta. Se tale area e` presente e se la sua dimensione permette una separazione in due, in modo da definire una nuova area libera, con propria intestazione, di dimensioni pari alla differenza rispetto all'area allocata, tale operazione di suddivisione viene compiuta.

Quando un'area viene restituita, in primo luogo si verifica se si tratta di area allocata in precedenza: infatti le intestazioni conservano anche l'informazione delle aree gia` allocate. L'area viene dichiarata libera e se l'area immediatamente precedente e/o seguente sono pure libere, viene effettuato il compattamento in un'unica area con eliminazione di intestazioni inutili.

Una tipica situazione per il pool e` quella di fig. 3.2a; dopo una allocazione di m byte, si passa alla configurazione di fig. 3.2b. (A=allocata; L=libera)

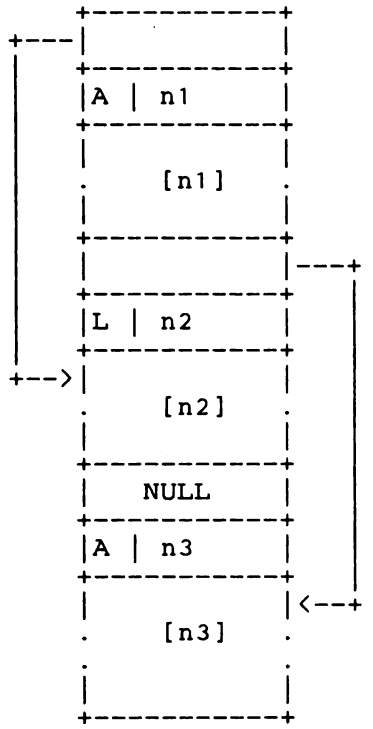


Fig. 3.2a

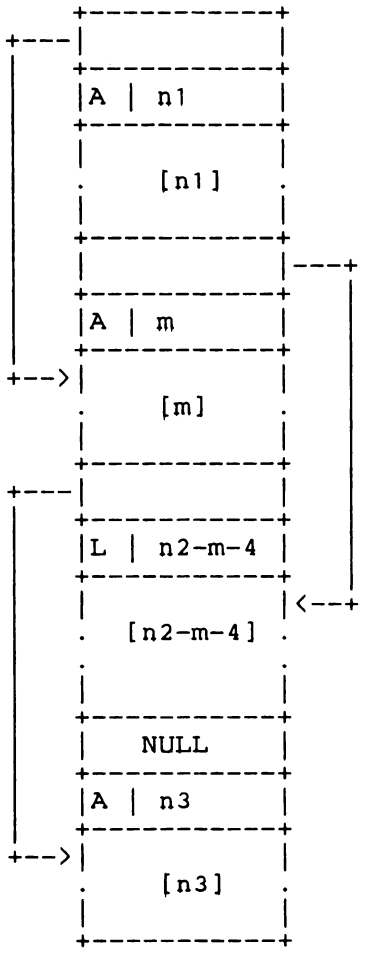


Fig. 3.2b

```
;       Dichiarazioni costanti
;
        NULL = 0                ; puntatore nullo
        POLSIZ = 256.           ; dimensione pool
        NEXT = -4               ; campo succ. in testata
        SIZ = -2                ; campo dim. area in testata
        SIGN = 100000           ; maschera per bit di segno

;-------------------------------------------------------------;
; NOME                                                        ;
;   PLINIT  inizializzazione pool                             ;
; DESCRIZIONE                                                 ;
;   Definisce il pool come un'unica area di dimensione        ;
;   iniziale nota.                                            ;
; INTERFACCIA                                                 ;
;   JSR PC, PLINIT                                            ;
; USA                                                         ;
;   Memo: POOLH                                               ;
;-------------------------------------------------------------;
PLINIT:
        MOV     #NULL, POOLH    ; puntatore next
        MOV     #POLSIZ, POOLH+2; dimensione area libera
        RTS     PC

;-------------------------------------------------------------;
; NOME                                                        ;
;   POLMAX  massima area disponibile                          ;
; DESCRIZIONE                                                 ;
;   Restituisce l'informazione della piu` grande area         ;
;   libera del pool.                                          ;
; INTERFACCIA                                                 ;
;   JSR PC, POLMAX                                            ;
;                                                             ;
;   R0      output  dimensione massima di area libera         ;
; USA                                                         ;
;   Memo: POOLM                                               ;
;-------------------------------------------------------------;
POLMAX:
        MOV     R1, -(SP)       ; salva registro
        MOV     #POOLM, R1      ; carica rif. primo header
        CLR     R0              ; massimo temporaneo
POX1:   TST     SIZ(R1)         ; dimensione area corrente
        BMI     POX2            ; salta se in uso
        CMP     SIZ(R1), R0     ; confronta con massimo temp.
        BLOS    POX2            ; salta se <= max
        MOV     SIZ(R1), R0
POX2:   MOV     NEXT(R1), R1    ; prossima area
        CMP     #NULL, R1       ; fine lista ?
        BNE     POX1            ; salta se no
        MOV     (SP)+, R1
        RTS     PC

;-------------------------------------------------------------;
; NOME                                                        ;
;   ALLOC   allocazione area da pool                          ;
; DESCRIZIONE                                                 ;
;   Provvede ad allocare un'area della dimensione richiesta.  ;
;   Allo scopo ricerca area di dimensione minima maggiore o   ;
;   uguale a quella richiesta. Se maggiore, effettua una      ;
;   suddivisione aggiungendo come nuova area libera quella    ;
```

```
;   rimasta rispetto all'area allocata.                          ;
; INTERFACCIA                                                    ;
;   JSR PC, ALLOC                                                ;
;                                                                ;
;   R0      input   numero byte da allocare                      ;
;   R1      output  puntatore area allocata o NULL se non        ;
;                   disponibile                                  ;
; USA                                                            ;
;   Memo: POOLM                                                  ;
;--------------------------------------------------------------;
ALLOC:
        SAVE    <R0,R2,R3,R5>   ; salva registri
        ADD     #4, R0          ; n+4
        MOV     #POOLM, R2      ; carica rif. primo header
        MOV     #NULL, R1       ; iniz. puntatore
        MOV     #-1, R5         ; iniz. minimo m = 2^16-1
ALC1:   MOV     SIZ(R2), R3     ; dimensione area
        BMI     ALC2            ; salta se in uso
        CMP     R0, R3
        BHI     ALC2            ; salta se siz<n+4
        CMP     R3, R5
        BHIS    ALC2            ; salta se siz>=m
        MOV     R2, R1          ; salva indirizzo area
        MOV     R3, R5          ; nuovo m
ALC2:   MOV     NEXT(R2), R2    ; salta prossimo blocco
        CMP     #NULL, R2
        BNE     ALC1            ; ciclo su prossimo blocco
        CMP     R2, R1          ; R1 = NULL ?
        SEC
        BEQ     ALC3            ; salta se si`, area non trovata
        CMP     R0, R5          ; dimensione corretta ?
        BEQ     ALC4            ; salta se si`, no suddiv.
        MOV     R1, R2          ; suddivisione di area
        ADD     R0, R2          ; R2 <- R1+n+4
        SUB     R0, R5          ; m-n-4
        MOV     R5, SIZ(R2)
        MOV     6(SP), SIZ(R1)  ; memorizza dimensione area
        MOV     NEXT(R1), NEXT(R2)
        MOV     R2, NEXT(R1)
ALC4:   BIS     #SIGN, SIZ(R1)  ; in uso
ALC3:   RESTORE <R5,R3,R2,R0>   ; ripristina registri
        RTS     PC

;--------------------------------------------------------------;
; NOME                                                         ;
;   FREE    rilascio area in pool                              ;
; DESCRIZIONE                                                  ;
;   Provvede a restituire al pool un'area precedentemente      ;
;   allocata: e` necessario restituire un indirizzo di quelli  ;
;   forniti dalla routine ALLOC. Se possibile, effettua il     ;
;   compattamento di aree contigue libere.                     ;
; INTERFACCIA                                                  ;
;   JSR PC, FREE                                               ;
;                                                              ;
;   R1      input   puntatore area restituita                  ;
;                   disponibile                                ;
;   C       output  1 se indirizzo illegale                    ;
; USA                                                          ;
;   Memo: POOLM                                                ;
;--------------------------------------------------------------;
```

```
FREE:
        SAVE    <R1,R2,R3>      ; salva registri
        MOV     #POOLM, R2      ; iniz. prec.
FRE1:   MOV     NEXT(R2), R3    ; carica succ. corrente
        CMP     R3, #NULL       ; fine lista ?
        SEC
        BEQ     FRE3            ; si, area illegale
        CMP     R3, R1          ; succ. corr. = succ. area ?
        BEQ     FRE2            ; salta se si`
        MOV     R3, R2          ; prossima area
        BR      FRE1
FRE2:   BIC     #SIGN, SIZ(R1)  ; area resa libera
        TST     SIZ(R2)         ; prec. libera ?
        BMI     FRE4            ; salta se no
                                ; compattamento con prec.
        ADD     SIZ(R1), SIZ(R2); dim. somma
        ADD     #4, SIZ(R2)     ; rilascio testata
        MOV     NEXT(R1), NEXT(R2)
        MOV     R2, R1
FRE4:   MOV     NEXT(R1), R2    ; esame del succ.
        TST     SIZ(R2)         ; area succ. libera ?
        BMI     FRE5            ; salta se no
        ADD     SIZ(R2), SIZ(R1); compattamento con succ.
        ADD     #4, SIZ(R1)
        MOV     NEXT(R2), NEXT(R1)
FRE5:   CLC
FRE3:   RESTORE <R3,R2,R1>      ; ripristina registri
        RTS     PC

        .CSECT  DAT
POOLH:  .BLKW   2
POOLM:  .BLKW   POLSIZ
++++++++
```

Esercizio 3.21

Si scriva una subroutine FGEN in grado di generare una subroutine che realizzi una funzione booleana generica a 4 ingressi e una uscita. Ad FGEN viene fornito un valore i cui bit sono ordinatamente i valori d'uscita relativi alle successive configurazioni d'ingresso (una configurazione e` l'indice del bit nel valore d'uscita) ed inoltre viene fornito l'indirizzo ove caricare la subroutine generata. Quest'ultima riceve nei 4 bit meno significativi di un registro i valori d'ingresso e restituisce il valore d'uscita nel flag di Carry.

Viene stabilito un modello di funzione booleana che utilizza un word in memoria di cui ciascun bit rappresenta un particolare valore d'uscita, come richiesto dall'esercizio. Affinche` la parte di codice non debba subire correzioni dopo il caricamento di una particolare istanza della funzione, si utilizzano indirizzamenti relativi.

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   BFUN        modello funzione generata                     ;
; DESCRIZIONE                                                 ;
;   Modello della funzione booleana che viene copiato         ;
;   come una nuova istanza. La funzione e` specificata dal    ;
```

```
;   word caricato in BTAB che da` il valore booleano          ;
;   d'uscita per ogni configurazione d'ingresso.              ;
; INTERFACCIA                                                 ;
;   JSR PC, BFUN                                              ;
;                                                             ;
;   R0      input   valore degli ingressi (b0..b3)            ;
;   C       output  valore booleano d'uscita                  ;
; USA                                                         ;
;   Memo: BTAB                                                ;
;-------------------------------------------------------------;
BFUN:
        MOV     R0, -(SP)       ; salva registri
        MOV     R1, -(SP)
        INC     R0
        MOV     BTAB, R1        ; carica valori d'uscita
                                ; (indirizzamento relativo)
BFU1:   ROR     R1              ; cattura in C bit indirizzato
        SOB     R0, BFU1        ; (salto relativo)
        MOV     (SP)+, R1       ; ripristina registri
        MOV     (SP)+, R0
        RTS     PC
BTAB:   .WORD   4               ; esempio con uscita 1 solo
                                ; per ingresso = 2
```

La funzione generatrice, dopo il caricamento del codice, provvede a caricare il word che rappresenta le uscite nella locazione per esso predisposta. Naturalmente, si suppone che il caricamento avvenga su RAM.

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   FGEN        subroutine generatrice                        ;
; DESCRIZIONE                                                 ;
;   Provvede a generare una istanza di BFUN copiando la       ;
;   funzione all'indirizzo richiesto e caricando BTAB.        ;
; INTERFACCIA                                                 ;
;   JSR PC, FGEN                                              ;
;                                                             ;
;   R0      input   valore dell'uscita nelle varie config.    ;
;   R1      input   indirizzo di caricamento dell'istanza     ;
; USA                                                         ;
;   Memo: BFUN                                                ;
;-------------------------------------------------------------;
        BF1SIZ = BFU1-BFUN      ; dimensione prima parte
        BFSIZ = BTAB-BFUN       ; dimensione totale BFUN

FGEN:
        MOV     R2, -(SP)       ; salva registro
        CLR     R2              ; iniz. offset
FGE1:   MOV     BFUN(R2), (R1)+ ; copia BFUN
        TST     (R2)+
        CMP     R2, #BFSIZ      ; fine funzione ?
        BNE     FGE1            ; salta se no
        MOV     R0, (R1)
        MOV     (SP)+, R2       ; ripristina registro
        RTS     PC
```

Come esempio si definiscano due funzioni, la prima realizzazione della funzione XOR per 4 ingressi (EXOR) che vale 0 solo se gli ingressi sono tutti eguali, 1 altrimenti; la seconda realizzazione

della funzione AND con 3 ingressi (AND3) ottenuta considerando il valore del quarto ingresso indifferente. Si definisca poi una subroutine che calcola il valore di uscita della complessiva funzione rappresentata in fig. 3.3.

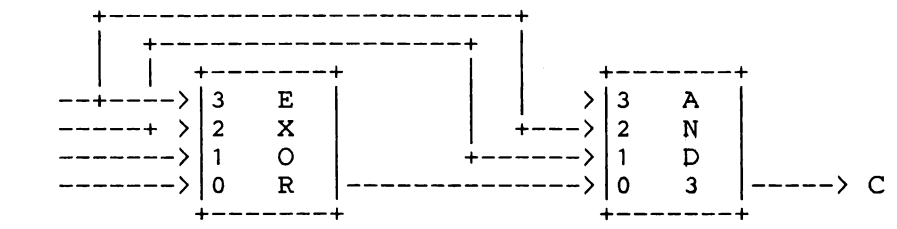


Fig. 3.3

```
;*************************************************************;
; PROGRAMMA PRINCIPALE                                        ;
;   FGENPR      prova funzione generatrice                    ;
; DESCRIZIONE                                                 ;
;   Genera due funzioni booleane EXOR e AND3 e le collega     ;
;   come in figura, provando due conf. d'ingresso.            ;
; USA                                                         ;
;   Memo: EXOR AND3                                           ;
;   Subr: FGEN                                                ;
;*************************************************************;
FGENPR:
        MOV     #077776, R0     ; funzione XOR su 4 ingressi
        MOV     #EXOR, R1       ; indirizzo caricamento
        JSR     PC, FGEN
        MOV     #100200, R0     ; AND 3 ingressi
        MOV     #AND3, R1       ; indirizzo di caricamento
        JSR     PC, FGEN
        MOV     #14, R0         ; ingressi
        JSR     PC, PR1         ; in C valori d'uscita comples.
        MOV     #17, R0         ; ingressi
        JSR     PC, PR1         ; in C valori d'uscita comples.
        HALT

;
;       routine di collegamento
;

PR1:    JSR     PC, EXOR        ; in C EXOR dei 4 ingressi
        ROR     R1              ; salva C
        ROR     R0
        ROR     R0              ; scorrimento ingressi per
                                ; AND3
        ROL     R1
        ROL     R0
        JSR     PC, AND3
        RTS     PC

        .CSECT  RAM

EXOR:   .BLKB   BF1SIZ
EXORL:  .BLKB   BFSIZ-BF1SIZ    ; loop label
EXORT:  .BLKW   1
```

```
AND3:   .BLKB   BF1SIZ
AND3L:  .BLKB   BFSIZ-BF1SIZ    ; loop label
AND3T:  .BLKW   1
```

Si suggerisce al lettore di apportare le necessarie modifiche affinche` il codice della funzione booleana non sia replicato in piu` versioni e ciascuna istanza sia semplicemente rappresentata dalla locazione che contiene i valori di uscita.
++++++++

3.4 - Conversioni di rappresentazioni numeriche

La rappresentazione interna dei numeri e` essenzialmente quella binaria e la convenzione adottata per i numeri con segno e` quella del complemento a 2. I numeri possono essere anche rappresentati mediante una sequenza di caratteri ASCII che possono essere visualizzati e che corrispondono ciascuno alla rappresentazione delle singole cifre nel numero, pensato in una certa base.

Un singolo bit, terne e quaterne di bit successivi (quest'ultime chiamate anche "nibble") rappresentano rispettivamente singole cifre binarie, ottali ed esadecimali. Un word di 16 bit (senza segno) e` in grado di contenere l'equivalente di 16 cifre binarie, 6 cifre ottali di cui la prima limitata a 0..1 e 4 cifre esadecimali. E` quindi pensabile di poter tradurre una simile quantita` in una stringa ASCII riferentesi ad una delle basi elencate. Si adottera` la convenzione che la stringa contenga solo le cifre se in base ottale; sia terminata dal carattere '.' se in base decimale; sia seguita dalla notazione [b] dove b e` la rappresentazione decimale della base se quest'ultima e` 2 o 16.

Si svilupperanno in questo capitolo alcune routine di conversione dalla rappresentazione interna a quella esterna (ASCII). In alcuni casi la stringa ASCII sara` contenuta in registri e in questo caso e` priva di terminatori, essendo la base specificata dal contesto.

Esercizio 3.22

Si definiscano le routine ESA2AS e AS2ESA che convertono da nibble a carattere ASCII e viceversa.

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   ESA2AS  conversione binario <= 15. -> carattere ASCII     ;
; DESCRIZIONE                                                 ;
;   Trasforma un nibble in cifra ASCII 0..9, A..F.            ;
; INTERFACCIA                                                 ;
;   JSR PC, ESA2AS                                            ;
;                                                             ;
;   R0      input   nibble                                    ;
;   R0      output  carattere ASCII                           ;
; USA                                                         ;
;   --                                                        ;
;-------------------------------------------------------------;
```

```
        NIBL    = 177760        ; maschera nibble meno sign.
ESA2AS:
        BIC     #NIBL, R0
        CMP     R0, #9.
        BGT     ESAA2           ; salta se > 9.
        ADD     #'0, R0         ; '0..'9, C=0
        RTS     PC
ESAA2:  ADD     #'A-10., R0     ; 'A..'F, C=0
        RTS     PC

;-------------------------------------------------------------;
; NOME                                                        ;
;   AS2ESA  conversione carattere ASCII -> binario            ;
; DESCRIZIONE                                                 ;
;   Trasforma un carattere ASCII 0..9, A..F in un nibble.     ;
; INTERFACCIA                                                 ;
;   JSR PC, AS2ESA                                            ;
;                                                             ;
;   R0      input   carattere ASCII                           ;
;   R0      output  nibble                                    ;
;   C       output  = 1 se carattere illegale                 ;
; USA                                                         ;
;   Subr: ISESA LO2UPC                                        ;
;-------------------------------------------------------------;
        LOWBY   = 177400        ; maschera byte meno sign.
AS2ESA:
        BIC     #LOWBY, R0      ; evidenzia byte low
        JSR     PC, ISESA
        BCC     ASB1            ; salta se non corretto
        JSR     PC, LO2UPC      ; converte maiuscolo
        CMPB    #'A-1, R0
        BHIS    ASB2            ; salta se 0..9
        SUB     #'A-10., R0     ; A..F, C=0
        RTS     PC
ASB2:   SUB     #'0, R0         ; C = 0
        RTS     PC
ASB1:   SEC
ASB3:   RTS     PC
++++++++
```

Esercizio 3.23

Definire le routine BY2HEX, HEX2BY, WO2HEX e HEX2WO che permettono la conversione, in un senso e nell'altro, tra valori a 8 e 16 bit senza segno e coppie e quaterne rispettivamente di cifre esadecimali ASCII.

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   BY2HEX  conversione byte a esadecimale ASCII              ;
; DESCRIZIONE                                                 ;
;   Converte un numero a 8 bit senza segno in una coppia di   ;
;   caratteri esadecimali ASCII.                              ;
; INTERFACCIA                                                 ;
;   JSR PC, BY2HEX                                            ;
;                                                             ;
;   R0.1    input   valore binario 8 bit                      ;
;   R1      output  2 cifre esadecimali (R1.1 meno sign.)     ;
```

```
; USA                                                          ;
;   Subr: ESA2AS                                               ;
;-------------------------------------------------------------;
BY2HEX:
        MOV     R0, -(SP)       ; salva registro
        ASR     R0
        ASR     R0
        ASR     R0
        ASR     R0
        JSR     PC, ESA2AS      ; cifra piu` significativa
        MOVB    R0, R1          ; estensione segno 0
        SWAB    R1
        MOV     2(SP), R0       ; ripristina valore
        JSR     PC, ESA2AS      ; cifra meno significativa
        BISB    R0, R1
        MOV     (SP)+, R0       ; ripristina registro
        RTS     PC

;-------------------------------------------------------------;
; NOME                                                         ;
;   HEX2BY  conversione esadecimale ASCII a byte               ;
; DESCRIZIONE                                                  ;
;   Converte una coppia di caratteri esadecimali ASCII in      ;
;   un numero binario senza segno a 8 bit.                     ;
; INTERFACCIA                                                  ;
;   JSR PC, HEX2BY                                             ;
;                                                              ;
;   R0.1    output  valore binario 8 bit                       ;
;   R1      input   2 cifre esadecimali (R1.1 meno sign.)      ;
; USA                                                          ;
;   Subr: AS2ESA                                               ;
;-------------------------------------------------------------;
HEX2BY:
        SWAB    R1
        MOVB    R1, R0          ; cifra piu` significativa
        JSR     PC, AS2ESA
        ASL     R0
        ASL     R0
        ASL     R0
        ASL     R0
        MOV     R0, -(SP)       ; salva prima parte
        SWAB    R1
        MOVB    R1, R0          ; cifra meno significativa
        JSR     PC, AS2ESA
        BISB    (SP)+, R0
        RTS     PC

;-------------------------------------------------------------;
; NOME                                                         ;
;   WO2HEX  conversione word a esadecimale ASCII               ;
; DESCRIZIONE                                                  ;
;   Converte un numero a 16 bit senza segno in una quaterna    ;
;   di caratteri esadecimali ASCII.                            ;
; INTERFACCIA                                                  ;
;   JSR PC, WO2HEX                                             ;
;                                                              ;
;   R0      input   valore binario 16 bit                      ;
;   R1      output  2 cifre esadecimali (parte meno sign.)     ;
;   R2      output  2 cifre esadecimali (parte piu` sign.)     ;
; USA                                                          ;
```

```
;   Subr: BY2HEX                                                    ;
;-----------------------------------------------------------------;
WO2HEX:
        SWAB    R0              ; byte piu` significativo
        JSR     PC, BY2HEX
        MOV     R1, R2
        SWAB    R0              ; byte meno significativo
        JSR     PC, BY2HEX
        RTS     PC

;-----------------------------------------------------------------;
; NOME                                                            ;
;   HEX2WO  conversione esadecimale ASCII a word                   ;
; DESCRIZIONE                                                     ;
;   Converte una quaterna di caratteri esadecimali ASCII in       ;
;   un numero binario senza segno a 16 bit.                       ;
; INTERFACCIA                                                     ;
;   JSR PC, HEX2WO                                                ;
;                                                                 ;
;   R0      ouput   valore binario 16 bit                         ;
;   R1      input   2 cifre esadecimali (parte meno sign.)        ;
;   R2      input   2 cifre esadecimali (parte piu` sign.)        ;
; USA                                                             ;
;   Subr: HEX2BY                                                  ;
;-----------------------------------------------------------------;
HEX2WO:
        MOV     R1, -(SP)       ; salva registro
        MOV     R2, R1          ; coppia piu` significativa
        JSR     PC, HEX2BY
        MOV     (SP)+, R1       ; ripristina registro
        SWAB    R0
        MOV     R0, -(SP)       ; salva parte piu` sign.
        JSR     PC, HEX2BY
        BIS     (SP)+, R0
        RTS     PC
```

Si suggerisce al lettore di realizzare le routine DW2HEX e HEX2DW che effettuano lo stesso tipo di conversione ma da una coppia di word ad una stringa ASCII di 8 caratteri in memoria e viceversa.
++++++++

Esercizio 3.24

Definire le routine BY2DEC, WO2DEC e DW2DEC che convertono rispettivamente un numero a 8, 16 e 32 bit con segno alla equivalente stringa ASCII in notazione decimale.

Per la conversione da 8 e 16 bit viene adottata una tecnica particolare: in pratica si divide il numero, mediante successive sottrazioni, per potenze di 10. decrescenti, ottenendo quindi le cifre di vario peso a partire da quelle di peso maggiore. Alla fine il numero viene compattato mediante eliminazione degli eventuali zeri non significativi.

```
;-----------------------------------------------------------------;
; NOME                                                            ;
;   BY2DEC      conversione binario -> ASCII decimale             ;
```

```
; DESCRIZIONE                                                    ;
;   Converte un intero a  8 bit con segno nella equivalente     ;
;   rappresentazione decimale ASCII che occupa un massimo di    ;
;   8 byte.                                                     ;
; INTERFACCIA                                                   ;
;   JSR PC, BY2DEC                                              ;
;                                                               ;
;   R0.1     input   valore binario da convertire               ;
;   R1       input   puntatore all'area (di almeno 8 byte)      ;
;                    destinata a contenere la stringa di        ;
;                    cifre decimali                             ;
; USA                                                           ;
;   Subr: WO2DEC                                                ;
;---------------------------------------------------------------;
BY2DEC:
        MOV     R0, -(SP)       ; salva registro
        MOVB    R0, R0          ; estensione del segno
        JSR     PC, WO2DEC
        MOV     (SP)+, R0       ; ripristina registro
        RTS     PC

;---------------------------------------------------------------;
; NOME                                                          ;
;   WO2DEC      conversione binario -> ASCII decimale           ;
; DESCRIZIONE                                                   ;
;   Converte un intero a 16 bit con segno nella equivalente     ;
;   rappresentazione decimale ASCII che occupa un massimo di    ;
;   8 byte.                                                     ;
; INTERFACCIA                                                   ;
;   JSR PC, WO2DEC                                              ;
;                                                               ;
;   R0       input   valore binario da convertire               ;
;   R1       input   puntatore all'area (di almeno 8 byte)      ;
;                    destinata a contenere la stringa di        ;
;                    cifre decimali                             ;
; USA                                                           ;
;   Memo: PW10                                                  ;
;---------------------------------------------------------------;
WO2DEC:
        SAVE    <R0,R3,R4,R1,R1>; salva i registri
        TST     R0
        BPL     BD2
        NEG     R0
        MOVB    #'-, (R1)+
BD2:
        MOV     #PW10, R3       ; tabella potenze 10
BD5:
        CLR     R4
        TST     (R3)+
        BEQ     BD3             ; fine conversione
BD4:
        INC     R4
        SUB     -2(R3), R0      ; sottrazione potenza
        BPL     BD4
        ADD     -2(R3), R0
        ADD     #'0-1, R4       ; cifra convertita
        MOVB    R4, (R1)+       ; inserisce cifra convertita
        BR      BD5
BD3:
        MOV     #5, R3          ; contatore
```

```
        MOV     (SP), R1        ; ripristina puntatore 1^
        CLR     R4
        CMPB    #'-, (R1)
        BNE     BD7
        INC     R1              ; negativo
        INC     R4
BD7:
        CMPB    #'0, (R1)       ; elimina zeri non utili
        BNE     BD6
        INC     R1
        SOB     R3, BD7
                                ; tutti zeri
        MOV     (SP)+, R1       ; ripristina puntatore 2^
        MOVB    #'0, (R1)+      ; senza segno
        BR      BD8
BD6:
        TST     R4
        BEQ     BD9
        INC     (SP)            ; salta segno
BD9:
        MOV     (SP)+, R4       ; ripristina vecchio puntatore
BD11:
        TST     R3
        BEQ     BD10
        MOVB    (R1)+, (R4)+    ; compatta
        DEC     R3
        BR      BD11
BD10:
        MOV     R4, R1
BD8:
        MOVB    #'., (R1)+
        MOVB    #EOS, (R1)
        RESTORE <R1,R4,R3,R0>   ; rispristina i registri
        RTS     PC

PW10:   .WORD 10000.
        .WORD 1000.
        .WORD 100.
        .WORD 10.
        .WORD 1.
        .WORD 0
```

Per la conversione da 32 bit viene viceversa adottato l'algoritmo descritto nell'esercizio 1.5, applicato a numeri binari da convertire in decimali. Allo scopo viene definita la divisione tra un numero a 32 bit e uno a 16 con quoziente a 32 bit. Poiche` il calcolo delle cifre di vario peso viene effettuato in ordine inverso, la stringa ASCII viene dapprima preparata in ordine inverso e poi rovesciata.

```
;-----------------------------------------------------------;
; NOME                                                      ;
;   DWDIV   subroutine divisione interi senza segno doppi   ;
; DESCRIZIONE                                               ;
;   Effettua la divisione tra interi senza segno, dividendo ;
;   da 2 word e divisore da 16 bit, fornendo quoziente e    ;
;   resto.                                                  ;
; INTERFACCIA                                               ;
;   JSR PC, DWDIV                                           ;
;                                                           ;
```

```
;   R0      input   dividendo parte meno sign.                 ;
;   R1      input   dividendo parte piu` sign.                 ;
;   R2      input   divisore                                   ;
;   R0      output  quoziente parte meno sign.                 ;
;   R1      output  quoziente parte piu` sign.                 ;
;   R2      output  resto                                      ;
; USA                                                          ;
;   Subr: DDIV                                                 ;
;------------------------------------------------------------;
DWDIV:
        MOV     R3,-(SP)    ; salva registri
        MOV     R0, -(SP)   ; salva dividendo parte meno sign.
        MOV     R1, R0
        CLR     R1
        JSR     PC, DDIV    ; divisione
        MOV     R1, R3      ; salva quoziente
        MOV     R0, R1      ; resto come dividendo piu` sign.
        MOV     (SP)+, R0   ; rispristina dividendo meno sign.
        JSR     PC, DDIV    ; divisione
        MOV     R0, R2      ; resto
        MOV     R1, R0      ; quoziente come meno sign.
        MOV     R3, R1      ; quoziente come piu` sign.
        MOV     (SP)+, R3   ; ripristina registri
        RTS     PC

;------------------------------------------------------------;
; NOME                                                       ;
;   DW2DEC  conversione da 32 bit a ASCII decimale           ;
; DESCRIZIONE                                                ;
;   Converte un intero a 32 bit con segno nella equivalente  ;
;   rappresentazione decimale ASCII che occupa un massimo di ;
;   12 byte.                                                 ;
; INTERFACCIA                                                ;
;   JSR PC, DW2DEC                                           ;
;                                                            ;
;   R0      input   dividendo parte meno sign.               ;
;   R1      input   dividendo parte piu` sign.               ;
;   R2      input   divisore                                 ;
;   R0      output  quoziente parte meno sign.               ;
;   R1      output  quoziente parte piu` sign.               ;
;   R2      output  resto                                    ;
; USA                                                        ;
;   Subr: DWDIV STREV                                        ;
;------------------------------------------------------------;
DW2DEC:
        SAVE    <R0,R1,R2,R3>   ;salva registri
        TST     R1
        BPL     DWD2            ; salta se positivo
        COM     R1              ; negato
        NEG     R0
        BNE     DWD2            ; salta se <> 0
        INC     R1
DWD2:   MOVB    #'., (R3)+      ; stringa inversa
DWD1:   MOV     #10., R2
        JSR     PC, DWDIV       ; divisione
        ADD     #'0, R2         ; carattere ASCII
        MOVB    R2, (R3)+
        TST     R0
        BNE     DWD1
        TST     R1
```

```
        BNE     DWD1            ; quoziente zero, fine ciclo
        TST     4(SP)           ; negativo ?
        BPL     DWD3            ; salta se no
        MOVB    #'-, (R3)+
DWD3:   MOVB    #EOS, (R3)
        MOV     0(SP), R1
        JSR     PC, STREV       ; rovesciamento
        RESTORE <R3,R2,R1,R0>   ; ripristina registri
        RTS     PC
++++++++
```

Esercizio 3.25

Definire le routine BY2OCT, WO2OCT e DW2OCT che convertono rispettivamente un numero a 8, 16 e 32 bit con segno alla equivalente stringa ASCII in notazione ottale. Di ciascuna si realizzi inoltre la versione senza segno.

Questa conversione e` relativamente semplice poiche` basta considerare gruppi di tre bit. Essendo la cifra ottale piu` pesante in tutti i casi limitata ad un numero < 7, risulta conveniente procedere alla conversione partendo dalla cifra meno significativa.

```
;-----------------------------------------------------------;
; NOME                                                      ;
;   BY2OCT, ABY2OC  conversione binario -> ASCII ottale byte ;
; DESCRIZIONE                                               ;
;   Converte un intero a  8 bit con segno nella equivalente ;
;   rappresentazione ottale ASCII che occupa al massimo 5   ;
;   byte. La versione ABY2OC non considera il segno         ;
; INTERFACCIA                                               ;
;   JSR PC, BY2OCT (ABY2OC)                                 ;
;                                                           ;
;   R0.l    input   valore binario da convertire            ;
;   R1      input   puntatore all'area (di almeno 5 byte)   ;
;                   destinata a contenere la stringa di     ;
;                   cifre ottali                            ;
; USA                                                       ;
;   --                                                      ;
;-----------------------------------------------------------;
        LOWBY   = 177400        ; maschera byte meno sign.
ABY2OC:
        SAVE    <R0,R1,R4>      ; salva registri
        BIC     #LOWBY, R0      ; solo byte meno sign.
        BR      BYC1
BY2OCT:
        SAVE    <R0,R1,R4>      ; salva registri
        BIC     #LOWBY, R0      ; solo byte meno sign.
        TSTB    R0
        BPL     BYC1            ; salta se positivo
        NEGB    R0
        MOVB    #'-, (R1)+
BYC1:   ADD     #3, R1          ; a fine stringa
        MOV     #EOS, (R1)
        MOV     #3, R4          ; contatore
BYC2:   MOV     R0, -(SP)
        BIC     #177770, R0     ; ultimi 3 bit
        ADD     #'0, R0         ; cifra convertita, C=0
```

```
        MOVB    R0, -(R1)
        MOV     (SP)+, R0       ; ripristino valore
        ROR     R0              ; / 8, con 0 a sinistra
        ASR     R0
        ASR     R0
        SOB     R4, BYC2
        RESTORE <R4,R1,R0>      ; ripristina registri
        RTS     PC

;-------------------------------------------------------------;
; NOME                                                        ;
;   WO2OCT,AWO2OC   conversione binario -> ASCII ottale word  ;
; DESCRIZIONE                                                 ;
;   Converte un intero a 16 bit con segno nella equivalente   ;
;   rappresentazione ottale ASCII che occupa al massimo 8     ;
;   byte. La versione AWO2OC non considera il segno.          ;
; INTERFACCIA                                                 ;
;   JSR PC, WO2OCT (AWO2OC)                                   ;
;                                                             ;
;   R0      input   valore binario da convertire              ;
;   R1      input   puntatore all'area (di almeno 8 byte)     ;
;                   destinata a contenere la stringa di       ;
;                   cifre ottali                              ;
; USA                                                         ;
;   --                                                        ;
;-------------------------------------------------------------;
AWO2OC:
        SAVE    <R0,R1,R4>      ; salva registri
        BR      WOC1
WO2OCT:
        SAVE    <R0,R1,R4>      ; salva registri
        TST     R0
        BPL     WOC1            ; salta se positivo
        NEG     R0
        MOVB    #'-, (R1)+
WOC1:   ADD     #6, R1          ; a fine stringa
        MOV     #EOS, (R1)
        MOV     #6, R4          ; contatore
WOC2:   MOV     R0, -(SP)
        BIC     #177770, R0     ; ultimi 3 bit
        ADD     #'0, R0         ; cifra convertita, C=0
        MOVB    R0, -(R1)
        MOV     (SP)+, R0       ; ripristino valore
        ROR     R0              ; / 8, con 0 a sinistra
        ASR     R0
        ASR     R0
        SOB     R4, WOC2
        RESTORE <R4,R1,R0>      ; ripristina registri
        RTS     PC

;-------------------------------------------------------------;
; NOME                                                        ;
;   DW2OCT, ADW2OC conv. binario doppio word -> ASCII ottale  ;
; DESCRIZIONE                                                 ;
;   Converte un intero a 32 bit con segno nella equivalente   ;
;   rappresentazione ottale ASCII che occupa al massimo 13    ;
;   byte. La versione ADW2OC non considera il segno.          ;
; INTERFACCIA                                                 ;
;   JSR PC, DW2OCT (ADW2OC)                                   ;
;                                                             ;
```

```
;   R0      input   valore binario parte meno significativa  ;
;   R1      input   valore binario parte piu` significativa  ;
;   R3      input   puntatore all'area (di almeno 13 byte)   ;
;                   destinata a contenere la stringa di      ;
;                   cifre ottali                             ;
; USA                                                        ;
;   --                                                       ;
;-----------------------------------------------------------;
ADW2OC:
        SAVE    <R0,R1,R3,R4>   ; salva registri
        BR      DWO1
DW2OCT:
        SAVE    <R0,R1,R3,R4>   ; salva registri
        TST     R1
        BPL     DWO1            ; salta se positivo
        COM     R1              ; negato
        NEG     R0
        BNE     DWO1            ; salta se <> 0
        INC     R1
        MOVB    #'-, (R3)+
DWO1:
        ADD     #11., R3        ; a fine stringa
        MOV     #EOS, (R3)
        MOV     #11., R4        ; contatore
DWO2:
        MOV     R0, -(SP)
        BIC     #177770, R0     ; ultimi 3 bit
        ADD     #'0, R0         ; cifra convertita, C=0
        MOVB    R0, -(R3)
        MOV     (SP)+, R0       ; ripristino valore
        ROR     R1              ; / 8, con 0 a sinistra
        ROR     R0
        ASR     R1
        ROR     R0
        ASR     R1
        ROR     R0
        SOB     R4, DWO2
        RESTORE <R4,R3,R1,R0>   ; ripristina registri
        RTS     PC
++++++++
```

Esercizio 3.26

Si scriva la subroutine WO2BIN che converte un numero senza segno di 16 bit nella equivalente rappresentazione ASCII in base 2. Si provveda ad eliminare gli zeri non significativi.

```
;-----------------------------------------------------------;
; NOME                                                      ;
;   WO2BIN      conversione word binario -> ASCII binario   ;
; DESCRIZIONE                                               ;
;   Converte un intero a 16 bit senza segno nella equivalente;
;   rappresentazione binaria ASCII che occupa al massimo 20 ;
;   byte.                                                   ;
; INTERFACCIA                                               ;
;   JSR PC, WO2BIN                                          ;
;                                                           ;
```

```
;   R0      input   valore binario da convertire            ;
;   R1      input   puntatore all'area (di almeno 20 byte)  ;
;                   destinata a contenere la stringa di     ;
;                   cifre ottali                            ;
; USA                                                       ;
;   --                                                      ;
;-----------------------------------------------------------;
WO2BIN:
        SAVE    <R0,R1,R3,R4,R1>; salva registri
        ADD     #21., R1        ; a fine stringa
        MOV     #EOS, -(R1)
        MOVB    #'], -(R1)
        MOVB    #'2, -(R1)
        MOVB    #'[, -(R1)
        MOV     #16., R4        ; contatore
BYB1:
        MOV     R0, -(SP)
        BIC     #177776, R0     ; ultimo bit
        ADD     #'0, R0         ; cifra convertita, C=0
        MOVB    R0, -(R1)
        MOV     (SP)+, R0       ; ripristino valore
        ASR     R0
        SOB     R4, BYB1
        MOV     #20., R3        ; contatore
BYB3:   CMPB    #'0, (R1)       ; elimina zeri non utili
        BNE     BYB2
        INC     R1
        SOB     R3, BYB3
                                ; tutti zeri
        MOV     (SP)+, R1       ; ripristina puntatore 2^
        MOVB    #'0, (R1)+      ; senza segno
        BR      BYB4
BYB2:
        MOV     (SP)+, R4       ; ripristina vecchio puntatore
BYB5:
        TST     R3
        BEQ     BYB6
        MOVB    (R1)+, (R4)+    ; compatta
        DEC     R3
        BR      BYB5
BYB6:
        MOV     R4, R1
BYB4:
        RESTORE <R4,R3,R1,R0>   ; ripristina registri
        RTS     PC
```

Si suggerisce al lettore di realizzare le routine equivalenti a questa per numeri a 8 e 32 bit.
++++++++

Esercizio 3.27

Si definisca la routine ASC2WO che consente di convertire in valore numerico (contenuto in un word) una stringa che rappresenta un numero in decimale, ottale o binario, con le convenzioni stabilite in precedenza. Il numero puo` essere opzionalmente preceduto da segno ('+ o '-).

La routine dovra` preliminarmente stabilire il tipo di numero (binario, decimale o ottale) osservando la parte terminale della stringa. Una volta stabilito a quale classe appartiene, dovra` rilevare la presenza del segno e memorizzare l'eventuale richiesta di negazione, indi procedera` alla conversione specifica per ogni classe. La piu` complicata e` evidentemente quella per i numeri decimali, per i quali si sfrutta la forma fattorizzata della notazione posizionale:

$$a = a_0 + a_1*10. + a_2*100. + a_3*1000. + a_4*10000. =$$
$$= a_0 + 10.*(a_1 + 10.*(a_2 + 10.*(a_3 + 10.*a_4)))$$

Il prodotto $a_i*10.$ viene eseguito nel modo seguente:

$$a_i*10. = a_i * 8. + a_i * 2 = asl(asl(asl(a_i)))+asl(a_i)$$

```
;-------------------------------------------------------------;
; NOME                                                        ;
;  ASC2WO   conversione ASCII (generale) -> binario word      ;
; DESCRIZIONE                                                 ;
;   Coverte una stringa di caratteri ASCII corrispondente ad  ;
;   un numero rappresentato in decimale, ottale o binario     ;
;   con segno opzionale nella equivalente rappresentazione    ;
;   a 16 bit con segno. Rappresentazioni convertibili:        ;
;   (+/-) bbb...b[2]    binario 1<=nb<=16                     ;
;   (+/-) dddddd.       decimale 1<=nd<=5 -32768.<=d<=32767.  ;
;   (+/-) oooooo        ottale  1<=nd<=6 -10000<=o<=077777    ;
; INTERFACCIA                                                 ;
;   JSR PC, ASC2WO                                            ;
;                                                             ;
;   R0      output  valore convertito                         ;
;   R1      input   puntatore al primo carattere della        ;
;                   stringa da convertire                     ;
;   R1      output  puntatore al primo carattere illegale     ;
;                   se presente, altrimenti non modificato    ;
;   C       output  1 se ci sono caratteri illegali           ;
;   V       output  1 se overflow                             ;
; USA                                                         ;
;   Subr: ISDEC ISBIN ISOCT STLEN TSTSGN                      ;
;-------------------------------------------------------------;
ASC2WO:
        SAVE    <R2,R3,R4,R1>   ; salva registri
        JSR     PC, STLEN       ; lunghezza stringa ASCII
        ADD     R3, R1          ; punta a EOS
        CMPB    -(R1), #']      ; binario ?
        BEQ     AWO1            ; salta se forse si`
        CMPB    (R1), #'.       ; decimale ?
        BEQ     AWO2            ; salta se si`
        MOV     0(SP), R1       ; ottale, ripristina punt.
        JSR     PC, TSTSGN      ; test per segno
        CLR     R2              ; iniz. valore
AWO3:   MOVB    (R1)+, R0       ; carica carattere
        JSR     PC, ISOCT
        BCC     AWO4            ; salta se illegale
        SUB     #'0, R0         ; cifra binaria
        ASL     R2              ; *2
        BMI     AWO5            ; overflow
        ASL     R2              ; *2
        BMI     AWO5            ; overflow
        ASL     R2              ; *2
```

```
        BMI     AWO5            ; overflow
        ADD     R0, R2          ; no overflow
        BR      AWO3
AWO4:   CMPB    #EOS, R0        ; e` EOS ?
        BNE     AWO6            ; illegale
        MOV     R2, R0
        JMP     AWOEX           ; ok

AWO2:
        MOV     0(SP), R1       ; decimale, ripristina punt.
        JSR     PC, TSTSGN      ; test per segno
        CLR     R2
AWO7:   MOVB    (R1)+, R0       ; carica carattere
        JSR     PC, ISDEC
        BCC     AWO8            ; carattere non decimale
        SUB     #'0, R0         ; cifra binaria
        ASL     R2              ; *2
        BMI     AWO5            ; overflow
        MOV     R2, R4          ; R2*10. = R2 * 2 * (4+1)
        ASL     R2
        BMI     AWO5            ; overflow
        ASL     R2
        BMI     AWO5            ; overflow
        ADD     R4, R2
        BVS     AWO5            ; overflow
        ADD     R0, R2
        BVS     AWO11           ; overflow, salvo 2'15
        BR      AWO7

AWO11:  CMP     #100000, R2     ; accetta solo -2'15
        BNE     AWO5            ; salta se overflow
        TST     R3
        BEQ     AWO5            ; overflow se positivo
        BR      AWO7

AWO8:   CMPB    #'., R0         ; e` il . ?
        BNE     AWO6            ; no, illegale
        MOV     R2, R0          ; carica valore
        JMP     AWOEX           ; ok

AWO1:   CMPB    -(R1), #'2      ; binario ?
        BNE     AWO6            ; salta se no, illegale
        CMPB    -(R1), #'[      ; binario
        BNE     AWO6            ; salta se no, illegale
        MOV     0(SP), R1       ; binario, ripristina punt.
        JSR     PC, TSTSGN      ; test per segno
        CLR     R2
AWO9:   MOVB    (R1)+, R0       ; carica carattere
        JSR     PC, ISBIN       ; binario ?
        BCC     AWO10           ; illegale
        SUB     #'0, R0         ; cifra binaria
        ROR     R0
        ROL     R2              ; shift cifra
        BMI     AWO5            ; overflow
        BR      AWO9
AWO10:  ADD     #3, R1          ; salta qualificatore
        CMPB    (R1)+, #EOS     ; e` EOS ?
        BNE     AWO6            ; illegale
        MOV     R2, R0          ; carica valore
AWOEX:  TST     R3
```

```
        BEQ     NONEG           ; salta se non negato
        NEG     R0
NONEG:  CLC                     ; C=0, V=0
        RESTORE <R1,R4,R3,R2>   ; ripristina registri
        RTS     PC

AWO5:   CLC                     ; overflow, C=0, V=1
        RESTORE <R1,R4,R3,R2>   ; ripristina registri
        SEV
        RTS     PC

AWO6:   DEC     R1              ; R1 punta a caratt. illegale
        TST     (SP)+
        SEC                     ; C=1
        RESTORE <R4,R3,R2>      ; ripristina registri
        RTS     PC
```

```
;-------------------------------------------------------------;
; NOME                                                        ;
;  TSTSGN   verifica presenza segno                           ;
; DESCRIZIONE                                                 ;
;   Provvede a verificare se la stringa e` introdotta da un   ;
;   segno + o -; in questo caso incrementa il puntatore e     ;
;   imposta un apposito registro.                             ;
; INTERFACCIA                                                 ;
;   JSR PC, TSTSGN                                            ;
;                                                             ;
;   R1      input   puntatore al primo carattere della        ;
;                   stringa da convertire                     ;
;   R1      output  puntatore al primo carattere dopo         ;
;                   eventuale segno                           ;
;   R3      output  1 se negato, 0 altrimenti                 ;
; USA                                                         ;
;   --                                                        ;
;-------------------------------------------------------------;
TSTSGN:
        CLR     R3
        CMPB    (R1), #'+
        BEQ     TSG1            ; salta se +
        CMPB    (R1), #'-
        BNE     TSG2            ; salta se segno non presente
        INC     R1              ; segno -
        INC     R3
TSG2:   RTS     PC
TSG1:   INC     R1              ; segno +
        RTS     PC
```

Il lettore puo` cimentarsi a definire una routine similare per numeri senza segno e per numeri a 32 bit.
++++++++

3.5 - L'ingresso-uscita

Come gia` detto nel capitolo 2, l'ingresso uscita viene gestito in modo memory-mapped: pertanto il processore comunica con i dispositivi esterni mediante registri accessibili direttamente conoscendo

l'indirizzo. La varieta` dei dispositivi disponibili induce a progettare specifiche routine di interfacciamento (driver) sulla base delle informazioni fornite dai costruttori. Una trattazione completa dell'interfacciamento con i vari tipi di dispositivi andrebbe oltre gli scopi di questo testo.

Qui ci si limitera` a definire alcune semplici routine per interfacciare due tipi di dispositivi: un dispositivo standard quale quello di controllo di una linea seriale di collegamento con un videoterminale (STDIN, STDOUT), e un generatore di impulsi a tempo (Real Time Clock o RTC), collegato ad una particolare linea di interruzione. Il lettore e` vivamente consigliato di ipotizzare altri tipi di dispositivi o di apportare le modifiche e le estensioni che si rendono necessarie per i dispositivi effettivamente disponibili. Le routine qui sviluppate dovrebbero costituire una guida di massima per la realizzazione di simili interfacce.

Nelle ipotesi presenti, il dispositivo standard e` sia di ingresso che d'uscita: allo scopo sono riservati 4 registri, 2 per l'ingresso e 2 per l'uscita. Di ciascuna coppia, un registro ha il significato di registro di stato e l'altro di registro dati (buffer).

Per l'ingresso, si supporra` che il registro di stato (RCSR = Receive Control Status Register) abbia indirizzo ARCSR = 177560 e che i suoi bit abbiano il seguente significato (R/W read/write; RO: read only):

bit	Simbolo	Accesso	Commento
6	RCV IE	R/W	Quando questo bit e` posto a 1 dal programma, la linea RCV IRQ (linea richiesta interruzione) segue l'andamento del bit RCV DONE
7	RCV DONE	RO	Il dispositivo pone a 1 questo bit quando il dato e` disponibile nel relativo registro. E` posto a 0 quando un dato viene letto dalla CPU.
11	RCV ACT	RO	Questo bit e` posto a 1 dal dispositivo quando e` in corso la ricezione dall'esterno di un dato. A ricezione avvenuta viene posto a 0.

Gli altri bit sono RO e sempre pari a 0.

Il registro dati (RBUF = Receive Buffer) ha indirizzo ARBUF = 177562 e contiene, oltre al dato ricevuto, altri bit che segnalano il verificarsi di particolari condizioni; tra questi sono di interesse, per gli esempi di programmi di questo capitolo, i seguenti:

bit	Simbolo	Accesso	Commento
0..7	RCV DATA	RO	Dato ricevuto (8 bit)
13	FR ERR	RO	Dato ricevuto non corretto (errore tipo frame)
14	OR ERR	RO	Dato non corretto perche` pervenuto prima della lettura del dato precedente (tipo overrun)
15	ERR	RO	Or dei due precedenti bit. Posto a 0 quando viene rimossa la causa d'errore

La linea RCV IRQ richiesta di interruzione si supporra` collegata ad una interruzione di priorita` 4 e con VA = 60. Quindi, quando il bit RCV IE e` posto a 1, la presenza di un 1 sul bit DONE corrisponde anche ad una richiesta di interruzione.

Esercizio 3.28

Si scriva la routine GET che acquisisce in R0 un carattere dal dispositivo standard di input. Si definisca successivamente la versione GETI che effettua la stessa operazione ma come routine di interruzione, che comunica il carattere letto attraverso un buffer di memoria.

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   GET     lettura di un carattere da standard input         ;
; DESCRIZIONE                                                 ;
;   Legge da standard input un carattere in polling.          ;
; INTERFACCIA                                                 ;
;   JSR PC, GET                                               ;
;                                                             ;
;   R0.l    output  carattere letto                           ;
; USA                                                         ;
;   Memo: ARCSR ARBUF                                         ;
;-------------------------------------------------------------;
        ARCSR = 177560
        ARBUF = 177562
GET:
        MOVB    @#ARCSR, R0
        BPL     GET             ; attende caratt. disponibile
        MOVB    @#ARBUF, R0     ; legge carattere
        RTS     PC
```

La versione come routine di interruzione deve provvedere a comunicare ad un programma utente non solo il carattere ma anche il fatto che esso e` effettivamente disponibile nel buffer: questo puo` essere realizzato mediante un flag (DONE) che viene posto a 1 dalla routine. La routine non garantisce che non vengano perduti caratteri se il programma utente non provvede a prelevarli dal buffer con sufficiente tempestivita`. Si vedra` piu` avanti una soluzione alternativa mediante buffer multicarattere.

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   GETI    routine di servizio per interrupt da STDIN        ;
; DESCRIZIONE                                                 ;
;   Risponde ad un interrupt da standard input leggendo il    ;
;   carattere battuto ed effettuando l'inserimento nel buffer;
; INTERFACCIA                                                 ;
;   --                                                        ;
; USA                                                         ;
;   Memo: ARBUF BUF DONE                                      ;
;-------------------------------------------------------------;
GETI:
        MOVB    @#ARBUF, BUF    ; legge carattere
        BISB    #1, DONE
        RTI
```

```
        .CSECT  DAT
BUF:    .BLKB   1
DONE:   .BYTE   0
```

Una possibile inizializzazione per la routine di interruzione da parte del programma principale e` la seguente:

```
        INVEC   = 60            ; STDIN interrupt vector
        INPSW   = 200           ; priorita` della routine (4)
        EI      = 100           ; interrupt enable

        MOV     #INVEC, R1
        MOV     #GETI, 0(R1)    ; vettore interruzioni
        MOV     #INPSW, 2(R1)
        BISB    #EI, @#ARCSR    ; abilita le interruzioni del
                                ; particolare dispositivo
        MTPS    #INPSW-40       ; priorita` che non maschera
                                ; l'interruzione da STDIN
++++++++
```

Per l'uscita, si supporra` che il registro di stato (XCSR) abbia indirizzo AXCSR = 177564 e dei suoi bit sono d'interesse i seguenti:

bit	Simbolo	Accesso	Commento
2	MAINT	R/W	Posto a 1 dal programma, forza il collegamento dell'uscita seriale all'ingresso seriale a scopo diagnostico
6	XMIT IE	R/W	Quando il programma imposta a 1 questo bit, la linea XMIT IRQ segue l'andamento di XMIT RDY
7	XMIT RDY	RO	Posto a 1 dal dispositivo quando il buffer che contiene il carattere da inviare in uscita e` vuoto e disponibile a riceverne uno dalla CPU

Il registro dati (XBUF) ha indirizzo AXBUF = 177566 e i seguenti bit significativi:

bit	Simbolo	Accesso	Commento
0..7	XMIT DATA	R/W	Buffer per il dato in uscita (8 bit)

Gli altri bit sono RO e sempre 0. La linea XMIT IRQ richiesta di interruzione si supporra` collegata ad una interruzione di priorita` 4 e con VA = 64.

Esercizio 3.29

Si scriva la routine PUT che riceve in R0 un carattere e lo invia al dispositivo standard di output. Si definisca successivamente la versione PUTI che effettua la stessa operazione ma come routine di interruzione, ricevendo il carattere letto attraverso un buffer di memoria. Si scriva un programma principale che fa l'eco da STDIN a STDOUT di tutti i caratteri utilizzando GETI e PUTI.

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   PUT     invio di un carattere allo standard output        ;
; DESCRIZIONE                                                 ;
;   Invia allo standard output un carattere ASCII in polling. ;
; INTERFACCIA                                                 ;
;   JSR PC, PUT                                               ;
;                                                             ;
;   R0.1   input   carattere da visualizzare                  ;
; USA                                                         ;
;   --                                                        ;
;-------------------------------------------------------------;
        AXCSR = 177564
        AXBUF = 177566
PUT:
        TSTB    @#AXCSR         ; buffer del disp. libero?
        BPL     PUT             ; salta se no
        MOVB    R0, @#AXBUF     ; invia carattere
        RTS     PC
```

Per la versione come routine di interruzione occorre far in modo che il programma principale effettui la prima operazione di output e successivamente abiliti il dispositivo ad interrompere. Ogni qualvolta il dispositivo libera il proprio buffer, invia una richiesta interruzione la cui routine di servizio provvedera` ad inviare al dispositivo un ulteriore carattere, se gia` prodotto, altrimenti sara` costretta a disabilitare le interruzioni dal dispositivo (per rimuovere la causa dell'interruzione) e a informare il programma utente che una interruzione non e` stata servita. Quest'ultimo dovra` in questo caso ripetere l'operazione autonoma di output. Chiaramente, nel caso di buffer mono-carattere e quando la generazione dei caratteri e` legata all'attivita` dell'operatore al terminale, essendo la rapidita` di produzione inferiore a quella del consumo dei caratteri ed inoltre nel caso presente avendo l'interruzione di output la stessa priorita` dell'interruzione di input, la routine di servizio di output in pratica non riesce mai a completare l'operazione. La soluzione proposta assume significato disponendo di un buffer multicarattere.

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   PUTI    routine di interruzione STDOUT                    ;
; DESCRIZIONE                                                 ;
;   Servizio per l'interruzione da STDOUT. Se carattere gia`  ;
;   prodotto, lo trasmette al dispositivo, altrimenti         ;
;   disabilita l'interruzione.                                ;
;   BUSY    0   routine disabilitata                          ;
;   BUSY    1   output in corso                               ;
;   BUSY    2   output in corso e nuovo caratt. disponibile   ;
; INTERFACCIA                                                 ;
;   --                                                        ;
; USA                                                         ;
;   Memo: BUSY AXCSR AXBUF OBU                                ;
;-------------------------------------------------------------;
        INVEC   = 60            ; STDIN interrupt vector
        INPSW   = 200           ; priorita` della routine (4)
        OUVEC   = 64            ; STDOUT interrupt vector
        OUPSW   = 200           ; priorita` della routine (4)
        EI      = 100           ; interrupt enable
```

```
        PDI     = 340           ; processor interrupt disable
        AXCSR   = 177564
        AXBUF   = 177566
PUTI:
        CMPB    BUSY, #2        ; carattere disponibile
        BEQ     PTI1            ; salta se si`
        CLRB    BUSY            ; no, disabilita interr.
        BICB    #EI, @#AXCSR    ; output
        RTI
PTI1:   DECB    BUSY            ; dichiara buffer vuoto
        MOVB    OBU, @#AXBUF    ; invia carattere
        RTI

;*************************************************************;
; PROGRAMMA PRINCIPALE                                        ;
;   PUTIPR      prova output ad interruzione                  ;
; DESCRIZIONE                                                 ;
;   Inizializza interruzioni per STDIN e STDOUT e si          ;
;   sincronizza con PUTI per l'eco dei caratteri.             ;
;   DONE    1   carattere prodotto                            ;
; USA                                                         ;
;   Memo: INVEC ARCSR OUVEC DONE BUSY BUF OBU                 ;
;   Subr: PUTI PUT                                            ;
;*************************************************************;
PUTIPR:
        MOV     #INVEC, R1
        MOV     #GETI, 0(R1)    ; vettore interruzioni
        MOV     #INPSW, 2(R1)
        BISB    #EI, @#ARCSR    ; abilita le interruzioni del
                                ; particolare dispositivo

        MOV     #OUVEC, R1
        MOV     #PUTI, 0(R1)    ; vettore interruzioni
        MOV     #OUPSW, 2(R1)
        MTPS    #140            ; priorita` che non maschera
                                ; l'interruzione da STDIN

P1:     TSTB    DONE
        BEQ     P1
        CLRB    DONE            ; azzera flag input
        MOVB    BUF, R0         ; legge carattere
        MTPS    #PDI            ; disabilita interruzioni
        TST     BUSY            ; test flag routine output
        BEQ     P2
        MOVB    R0, OBU         ; routine di servizio attiva
        MOVB    #2, BUSY        ; segnala carattere prodotto
        MTPS    #140            ; riabilita interruzioni
        BR      P1
P2:     MTPS    #140            ; riabilita interruzioni
        JSR     PC, PUT         ; output disabilitato
        INCB    BUSY            ; dichiara carattere
                                ; disponibile
        BISB    #EI, @#AXCSR    ; abilita interruzioni output
        BR      P1

        .CSECT  DAT
BUF:    .BLKB   1
OBU:    .BLKB   1
DONE:   .BYTE   0
BUSY:   .BYTE   0
```

Si noti che la valutazione del flag BUSY da parte del programma principale e` inserita in una sezione a interruzioni disabilitate. Infatti, in generale e` necessario proteggere l'accesso per modifica a variabili comuni a piu` processi (in questo caso i processi sono costituiti dall'attivita` del programma principale e da quella della routine di interruzione) quando questo accesso viene eseguito con operazioni non indivisibili (elementari). Ad esempio in questo caso potrebbe accadere che la richiesta di interruzione del dispositivo di output avvenga tra l'esecuzione dell'istruzione TST BUSY e BEQ P2. Supponendo che BUSY valga 1, la routine troverebbe il buffer vuoto e si disattiverebbe portando BUSY a 0. Al ritorno al programma principale, il salto non verrebbe eseguito, poiche` la valutazione era gia` stata effettuata con BUSY=1 prima dell'interruzione, e il carattere prodotto sarebbe inserito nel buffer e la routine di servizio per l'output non sarebbe piu` riattivata. Poiche` in questo caso particolare il dispositivo fisico e` unico, probabilmente questa condizione non si puo` verificare ma egualmente il problema si puo` presentare in condizioni piu` generali.

Si suggerisce al lettore di modificare la routine PUTI in modo che prima di inviare un carattere valido, generi un ritardo inviando caratteri NUL (0) a STDOUT. Se il ritardo e` sufficiente, e` possibile ottenere che i caratteri prodotti vengano effettivamente inseriti nel buffer OBU.
++++++++

Esercizio 3.30

Si definiscano le routine GETS e PUTS che rispettivamente permettono l'acquisizione e la visualizzazione di stringhe di caratteri. La routine GETS deve consentire la correzione dei caratteri battuti mediante carattere DEL e considera come terminatore il carattere CR che viene sostituito da EOS. Si definisca inoltre la routine PNEWL che invia la coppia CR-LF (NewLine) allo STDOUT.

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   GETS    lettura di una stringa da STDIN                   ;
; DESCRIZIONE                                                 ;
;   Legge da STDIN una stringa mediante la subroutine         ;
;   GET, consentendo la cancellazione dei caratteri battuti   ;
;   per errore. La stringa va terminata con un CR, che        ;
;   viene sostituito con un EOS. La lunghezza massima         ;
;   consentita comprende il terminatore EOS.                  ;
; INTERFACCIA                                                 ;
;   JSR PC, GETS                                              ;
;                                                             ;
;   R1      input   puntatore all'area di memoria riservata   ;
;                   a contenere la stringa.                   ;
;   R2      input   lunghezza massima                         ;
; USA                                                         ;
;   Subr: GET PUT PNEWL                                       ;
;-------------------------------------------------------------;
GETS:
        SAVE    <R0,R1,R2,R3>    ; salva registri
```

```
        CLR     R3              ; contatore caratteri
GL1:
        JSR     PC, GET
        BIC     #177600, R0     ; 7 bit
        CMPB    #21, R0         ; ignora ^Q
        BEQ     GL1
        CMPB    #23, R0         ; ignora ^S
        BEQ     GL1             ; (questi caratteri sono talvolta
                                ;  generati dal terminale per
                                ;  controllare lo scroll su video)
        CMPB    #177, R0        ; DEL ?
        BNE     GL2
        TST     R3              ; qualche carattere ?
        BEQ     GL3
        MOVB    #10, R0         ; si, BS (cancella dal video)
        JSR     PC, PUT
        MOVB    #' , R0         ; SPACE
        JSR     PC, PUT
        MOVB    #10, R0         ; BS
        JSR     PC, PUT
        DEC     R3              ; elimina il carattere
        DEC     R1              ; dalla stringa
        BR      GL1
GL2:
        CMPB    R3, R2          ; troppo lunga
        BHIS    GL3
        CMPB    #15, R0         ; CR fine linea
        BEQ     GL4
        MOVB    R0, (R1)+
        INC     R3
        JSR     PC, PUT         ; eco
        BR      GL1
GL4:
        JSR     PC, PNEWL       ; eco CR + LF
        MOVB    #EOS, (R1)      ; sostituisce con EOS
        RESTORE <R3,R2,R1,R0>   ; ripristina registri
        RTS     PC
GL3:
        MOVB    #7, R0          ; errore BEL
        JSR     PC, PUT
        BR      GL1

;-------------------------------------------------------------;
; NOME                                                        ;
;   PUTS    invio di una stringa di caratteri                 ;
; DESCRIZIONE                                                 ;
;   Invia a STDOUT una stringa di caratteri ASCII             ;
;   utilizzando la subroutine PUT. La stringa e` terminata    ;
;   dal carattere EOS.                                        ;
; INTERFACCIA                                                 ;
;   JSR PC, PUTS                                              ;
;                                                             ;
;   R1      input   puntatore al primo carattere della        ;
;                   stringa                                   ;
; USA                                                         ;
;   Subr: PUT                                                 ;
;-------------------------------------------------------------;
PUTS:
        MOV     R0, -(SP)       ; salva registri
        MOV     R1, -(SP)
```

```
PS2:
        MOVB    (R1)+, R0
        CMPB    #EOS, R0        ; terminatore ?
        BEQ     PS1             ; si
        JSR     PC, PUT
        BR      PS2
PS1:
        MOV     (SP)+, R1       ; ripristina registri
        MOV     (SP)+, R0
        RTS     PC

;-------------------------------------------------------------;
; NOME                                                        ;
;   PNEWL   invio a STDOUT di NewLine                         ;
; DESCRIZIONE                                                 ;
;   Invia a STDOUT la sequenza CR-LF.                         ;
; INTERFACCIA                                                 ;
;   JSR PC, PNEWL                                             ;
; USA                                                         ;
;   Subr: PUT                                                 ;
;-------------------------------------------------------------;
        CR  = 15
        LF  = 12
PNEWL:
        MOV     R0, -(SP)
        MOVB    #CR, R0
        JSR     PC, PUT
        MOVB    #LF, R0
        JSR     PC, PUT
        MOV     (SP)+, R0
        RTS     PC
++++++++
```

Esercizio 3.31

Supponendo che il dispositivo di standard output sia collegato ad un terminale che accetti sequenze di Escape ANSI (ad esempio VT100) definire le routine PUTRC, PUTSRC che visualizzano rispettivamente un carattere e una stringa dopo aver posizionato il cursore alla riga e colonna specificate; le routine SAVCUR e RESCUR che permettono il salvataggio da parte del terminale della posizione del cursore e il suo ripristino; la routine CLS che cancella lo schermo.

Le sequenze di escape costituiscono un particolare protocollo per l'interpretazione, da parte dei dispositivi riceventi, di comandi ausiliari rispetto all'acquisizione dei caratteri stampabili e di controllo. Per il problema posto sono di interesse:

ESC '7 sequenza di controllo che fa memorizzare al VT100 la posizione del cursore

ESC '8 sequenza di controllo che riposiziona il cursore alla posizione memorizzata

ESC '[r '; c 'f
sequenza di controllo che posiziona il cursore alla riga e colonna specificate. r e c sono numeri decimali >= 1.

ESC '['2 'J
sequenza di controllo che cancella tutto lo schermo del terminale e posiziona il cursore a (1,1).

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   PUTRC   posizionamento cursore e visualizzazione          ;
; DESCRIZIONE                                                 ;
;   Posiziona il cursore del terminale nella riga e colonna   ;
;   fornite e quindi visualizza il carattere fornito          ;
; INTERFACCIA                                                 ;
;   JSR PC, PUTRC                                             ;
;                                                             ;
;   R0.l    input   carattere da visualizzare                 ;
;   R1.l    input   riga di posizionamento                    ;
;   R1.h    input   colonna di posizionamento                 ;
; USA                                                         ;
;   Subr: PUT SB2DEC                                          ;
;-------------------------------------------------------------;
PUTRC:
        MOV     R1, -(SP)       ; salva registri
        MOV     R0, -(SP)
        MOVB    #ESC, R0        ; invia prima parte seq. ANSI
        JSR     PC, PUT
        MOVB    #'[, R0
        JSR     PC, PUT
        MOV     R1, -(SP)       ; salva registro
        CLR     R0
        MOVB    R1, R0          ; invia riga
        JSR     PC, SB2DEC      ; conversione a coppia di
                                ; cifre decimali
        SWAB    R1
        ADD     #'0, R1         ; ASCII seconda cifra
        MOVB    R1, R0
        JSR     PC, PUT

        SWAB    R1
        ADD     #'0, R1         ; ASCII prima cifra
        MOVB    R1, R0
        JSR     PC, PUT
        MOVB    #';, R0
        JSR     PC, PUT
        MOV     (SP)+, R1       ; ripristina registro
        SWAB    R1              ; invia colonna
        MOVB    R1, R0
        JSR     PC, SB2DEC      ; conversione a coppia di
                                ; cifre decimali
        SWAB    R1
        ADD     #'0, R1         ; ASCII seconda cifra
        MOVB    R1, R0
        JSR     PC, PUT

        SWAB    R1
        ADD     #'0, R1         ; ASCII prima cifra
        MOVB    R1, R0
        JSR     PC, PUT
        MOVB    #'f, R0         ; invia ultima parte
        JSR     PC, PUT
        MOV     (SP)+, R0       ; ripristina registro
```

```
        JSR     PC, PUT         ; invia carattere
        MOV     (SP)+, R1       ; ripristina registro
        RTS     PC
```

In PUTRC viene utilizzata la routine SB2DEC che e` una versione alternativa alla BY2DEC. La conversione avviene per confronto con multipli di 10. inseriti in una tabella a cui si accede con il metodo della bisezione.

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   SB2DEC  conversione binario a decimale 2 cifre            ;
; DESCRIZIONE                                                 ;
;   Converte un numero x con -99<=x<=99 in un numero decimale;
;   codificato con due cifre piu` bit di segno.               ;
; INTERFACCIA                                                 ;
;   JSR PC, SB2DEC                                            ;
;                                                             ;
;   R0      input   numero da covertire                       ;
;   R1.l    output  cifra decimale meno significativa         ;
;   R1.h    output  cifra decimale piu` significativa e con   ;
;                   b15 come bit di segno                     ;
;   C       output  = 1 se numero illegale                    ;
; USA                                                         ;
;   Memo: DECT                                                ;
;-------------------------------------------------------------;
SB2DEC:
        CMP     R0, #99.        ; errore se maggiore
        BGT     BY2ERR
        CMP     R0, #-99.       ; errore se minore
        BLT     BY2ERR
        SAVE    <R0,R2,R3>      ; salva registri
        BPL     BYISP1          ; salta se positivo
        NEG     R0
BYISP1: CLR     R1
        MOV     #7, R2          ; posiz. intermedia
        MOV     #8., R3         ; incertezza iniz.
AN1:    ASR     R3              ; inc./2
        BEQ     FO2             ; inc. = 0
        CMPB    R0, DECT(R2)    ; confronta in tabella
        BEQ     FO1             ; = a locaz.
        BGT     ISGT1           ; > di locaz.
        SUB     R3, R2          ; sottrae inc.
        BR      AN1             ; cicla
ISGT1:  ADD     R3, R2          ; < di locaz., aggiunge inc.
        BR      AN1             ; cicla
FO2:    CMPB    R0, DECT(R2)    ; compar. aggiuntiva
        BGE     FO1             ; salta se >=
        DEC     R2              ; aggiusta posiz.
FO1:    MOVB    R0, R1          ; carica numero
        MOVB    DECT(R2), R3    ; carica multiplo 10
        SUB     R3, R1          ; cifra meno sign.
        SWAB    R2              ; C = 0
        BIS     R2, R1          ; cifra piu` sign.
        RESTORE <R3,R2,R0>      ; ripristina registri
        BPL     BYISP2
        BIS     #100000, R1     ; bit segno = 1
BYISP2: RTS     PC              ; C = 0
```

```
BY2ERR: SEC                         ; numero errato
        RTS     PC

DECT:   .BYTE   0, 10., 20., 30., 40., 50., 60., 70., 80., 90.
        .BYTE   100., 100.


;-------------------------------------------------------------;
; NOME                                                        ;
;   PUTSRC  posizionamento cursore e visualizzazione stringa  ;
; DESCRIZIONE                                                 ;
;   Posiziona il cursore del terminale nella riga e colonna   ;
;   fornite e quindi visualizza la stringa fornita.           ;
; INTERFACCIA                                                 ;
;   JSR PC, PUTSRC                                            ;
;                                                             ;
;   R1      input   puntatore stringa da visualizzare         ;
;   R2.l    input   riga di posizionamento                    ;
;   R2.h    input   colonna di posizionamento                 ;
; USA                                                         ;
;   Subr: PUTRC PUTS                                          ;
;-------------------------------------------------------------;
PUTSRC:
        MOV     R0, -(SP)       ; salva registro
        MOVB    (R1)+, R0       ; invia primo carattere
        MOV     R1, -(SP)       ; salva registro
        MOV     R2, R1
        JSR     PC, PUTRC       ; posizionamento
        MOV     (SP)+, R1       ; ripristina registro
        JSR     PC, PUTS        ; il resto della stringa
        DEC     R1
        MOV     (SP)+, R0       ; ripristina registro
        RTS     PC


;-------------------------------------------------------------;
; NOME                                                        ;
;   SAVCUR  memorizzazione su terminale di posizione cursore  ;
; DESCRIZIONE                                                 ;
;   Invia al terminale la sequenza ANSI corrispondente        ;
;   alla memorizzazione della posizione del cursore.          ;
; INTERFACCIA                                                 ;
;   JSR PC, SAVCUR                                            ;
; USA                                                         ;
;   Subr: PUTS                                                ;
;   Memo: SVCS                                                ;
;-------------------------------------------------------------;
SAVCUR:
        MOV     R1, -(SP)       ; salva registro
        MOV     #SVCS, R1
        JSR     PC, PUTS
        MOV     (SP)+, R1       ; ripristina registro
        RTS     PC


SVCS:
        .BYTE   ESC, '7, EOS    ; sequenza di caratteri di
        .EVEN                   ; controllo  riconosciuta dal
                                ; terminale VT100 come comando
                                ; di salvataggio cursore
```

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   RESCUR  ripristino posizione cursore video                ;
; DESCRIZIONE                                                 ;
;   Invia al terminale la sequenza ANSI corrispondente        ;
;   al ripristino della posizione del cursore precedentemente;
;   memorizzata.                                              ;
; INTERFACCIA                                                 ;
;   JSR PC, RESCUR                                            ;
; USA                                                         ;
;   Subr: PUTS                                                ;
;   Memo: RECS                                                ;
;-------------------------------------------------------------;
RESCUR:
        MOV     R1, -(SP)       ; salva registro
        MOV     #RECS, R1
        JSR     PC, PUTS
        MOV     (SP)+, R1       ; ripristina registro
        RTS     PC
RECS:
        .BYTE   ESC, '8, EOS    ; sequenza di caratteri di
        .EVEN                   ; controllo  riconosciuta dal
                                ; terminale VT100 come comando
                                ; di ripristino cursore


;-------------------------------------------------------------;
; NOME                                                        ;
;   CLS     cancellazione dello schermo video                 ;
; DESCRIZIONE                                                 ;
;   Invia al terminale la sequenza ANSI corrispondente        ;
;   a CLS.                                                    ;
; INTERFACCIA                                                 ;
;   JSR PC, CLS                                               ;
; USA                                                         ;
;   Subr: PUTS                                                ;
;   Memo: CLSS                                                ;
;-------------------------------------------------------------;
CLS:
        MOV     R1, -(SP)       ; salva registro
        MOV     #CLSS, R1
        JSR     PC, PUTS
        MOV     (SP)+, R1       ; ripristina registro
        RTS     PC

CLSS:
        .BYTE   ESC             ; sequenza di caratteri di
        .ASCII  /[2J/           ; controllo  riconosciuta dal
        .BYTE   EOS             ; terminale VT100 come comando
        .EVEN                   ; di cancellare lo schermo
++++++++
```

Si supponga di disporre di un generatore di impulsi (RTC) a frequenza 800 Hz collegato ad una interruzione di priorita` 6 e VA=104. Si supponga inoltre che la generazione delle interruzioni ad ogni impulso sia abilitata inviando il byte CLKIE=7 al registro CLKSR di indirizzo 177444 e venga disabilitata inviando il byte CLKID=10 allo stesso registro. La disabilitazione corrisponde anche ad eliminare la causa che ha prodotto la richiesta di interruzione.

Esercizio 3.32

Si definisca una routine di interruzione per il dispositivo RTC che tenga aggiornato un orologio completo. In prossimita` dello scadere di ogni minuto, essa inoltre invia una sequenza di riconoscimento di caratteri BEL a STDOUT. Il programma principale, oltre alla inizializzazione, dovra` provvedere a leggere l'orologio e allo scadere di ogni minuto inviare a STDOUT una stringa corrispondente all'ora corrente.

L'orologio e` rappresentato dalle locazioni HOURS, MINUTS e TIME. In particolare, l'ultima di queste conserva l'informazione sia dei secondi che degli impulsi elementari pervenuti (tick). Allo scopo i 16 bit di TIME sono divisi in due parti a cui si adeguano le valutazioni per lo scadere del secondo (800 tick) e del minuto (60 sec).

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   CLKIR   routine di servizio del RTC a 800 Hz              ;
; DESCRIZIONE                                                 ;
;   Tenendo conto della frequenza del clock, la routine       ;
;   aggiorna un contatore di secondi, minuti, ore e se in     ;
;   prossimita` dello scadere del minuto, fa suonare          ;
;   l'avvisatore del terminale.                               ;
; INTERFACCIA                                                 ;
;   --                                                        ;
; USA                                                         ;
;   Subr: PUT                                                 ;
;   Memo: TIME MINUTS HOURS CLKSR                             ;
;-------------------------------------------------------------;
        BEL     = 7
        CLKSR   = 177444        ; registro di stato RTC
CLKIR:
        MOVB    #10, @#CLKSR    ; disabilita inter. clock
        INC     @#TIME          ; contatore tick
                                ; i bit 0..9 sono utilizzati
                                ; per contare fino a 800;
                                ; i bit 10..15 sono
                                ; utilizzati per contare i
                                ; secondi in un minuto
        MOV     R0, -(SP)       ; salva registro
        MOV     @#TIME, R0
        COM     R0
        BIT     #001440, R0     ; = 800 ?
        BNE     NOV             ; no overflow
        BIC     #001777, @#TIME ; clear 800
        ADD     #002000, @#TIME ; aggiunge 1 sec.

        CMP     @#TIME, #154000 ; >= 54 sec
        BLO     NOBEL
        CMP     @#TIME, #166000 ; no 59 sec
        BEQ     NOBEL
        MOVB    #BEL, R0        ; suona
        JSR     PC, PUT

NOBEL:
        CMP     @#TIME, #170000 ; overflow 60 sec
        BNE     NOV
        CLR     @#TIME
```

```
        INC     @#MINUTS
        CMP     @#MINUTS, #60.   ; overflow 60 min
        BLO     NOV
        CLR     @#MINUTS
        INC     @#HOURS
        CMP     @#HOURS, #24.    ; overflow 24 h
        BLO     NOV
        CLR     @#HOURS
NOV:
        MOV     (SP)+, R0        ; ripristina registro
        MOVB    #7, @#CLKSR      ; abilita inter. clock
        RTI

;*************************************************************;
; PROGRAMMA PRINCIPALE                                        ;
;   TIMER       orologio con segnale orario                   ;
; DESCRIZIONE                                                 ;
;   Inizializza il vettore collegato a RTC e visualizza l'ora;
;   ad ogni cambio di minuto.                                 ;
; USA                                                         ;
;   Memo: CLKVEC CLKSR HOURS MINUTS MSG1 STR1                 ;
;   Subr: PNEWL PUTS WO2DEC                                   ;
;*************************************************************;
TIMER:
        MOV     #CLKVEC, R1
        MOV     #CLKIR, 0(R1)
        MOV     #CLKPSW, 2(R1)   ; modifica interrupt vector

        JSR     PC, PNEWL
        MOVB    #7, @#CLKSR      ; abilita inter. clock
LOOP:
        MOV     @#MINUTS, R0
LO2:
        CMP     @#MINUTS, R0     ; minuto scaduto ?
        BEQ     LO2
        MOV     #MSG1, R1        ; si, visualizza orario
        JSR     PC, PUTS
        MOV     @#HOURS, R0
        MOV     #STR1, R1
        JSR     PC, WO2DEC
        MOV     #STR1, R1
        JSR     PC, PUTS
        MOV     @#MINUTS, R0
        MOV     #STR1, R1
        JSR     PC, WO2DEC
        MOV     #STR1, R1
        JSR     PC, PUTS
        JSR     PC, PNEWL
        BR      LOOP
;
;       Variabili e messaggi
        .CSECT  DAT
TIME:   .WORD 0
MINUTS: .WORD 0
HOURS:  .WORD 0
MSG1:   .ASCII /Ora -> /
        .BYTE   EOS
        .EVEN
STR1:
        .BLKB   STRSIZ
```

Si osservi che in questo caso, poiche` il programma principale si limita a leggere le variabili comuni con il processo a tempo, non e` necessaria alcuna forma di protezione. Piuttosto la durata della routine di servizio di RTC potrebbe essere superiore al periodo del clock stesso, in particolare quando invia caratteri BEL a STDOUT (800 Hz --> T = 1/800 s = 1.25 ms). Si cerchi di riorganizzare il programma in modo da ovviare al problema.
++++++++

CAPITOLO 4

TEMI VARI

In questo capitolo viene proposta una serie di temi che utilizzano le routine sviluppate in precedenza e danno ulteriori spunti per la realizzazione di applicazioni complete.

Esercizio 4.1

Si scriva una subroutine che conta il numero di bit pari a 0 presenti nelle N successive locazioni a partire da ADDR. Il valore di ADDR e` contenuto in R1 e quello di N in R2. Il risultato e` restituito in R0.

Disponendo della subroutine NBIT1 che calcola il numero di bit 1 in una parola, la subroutine richiesta e` relativamente semplice.

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   NNBIT0      subroutine calcolo numero di bit pari a 0     ;
; DESCRIZIONE                                                 ;
;   Fornisce il numero di bit pari a 0 della                  ;
;   rappresentazione binaria di una sequenza di numeri        ;
; INTERFACCIA                                                 ;
;   JSR PC, NNBIT0                                            ;
;                                                             ;
;   R0      output  totale bit pari a 0                       ;
;   R1      input   puntatore primo word della sequenza       ;
;   R2      input   numero word della sequenza                ;
; USA                                                         ;
;   Subr: NBIT1                                               ;
;-------------------------------------------------------------;
NNBIT0:
        SAVE    <R1,R2,R3,R4>   ; salva registri
        CLR     R4              ; iniz. totale
        MOV     R1, R3          ; R3 come puntatore
NNB1:   MOV     (R3)+, R0
        JSR     PC, NBIT1
        ADD     #16., R4        ; Tot = Tot + 16 - Nbit1
        SUB     R1, R4
        SOB     R2, NNB1
        MOV     R4, R0          ; Tot in R0
        RESTORE <R4,R3,R2,R1>   ; ripristina registri
        RTS     PC
++++++++
```

Esercizio 4.2

Si scriva una subroutine filtro che riceve testo in linguaggio assembly e provvede ad eliminarvi gli eventuale commenti. Si scriva un programma di prova che riceve un testo da STDIN, terminato dal carattere '? ad inizio riga, e lo invia, filtrato, in STDOUT.

La subroutine CFILT deve copiare una stringa in un'altra ma eliminando tutto cio` che e` inserito tra '; (compreso) e CR (escluso). Allo scopo si supponga che il testo all'interno della stringa sia suddiviso in linee terminate dalla coppia CR-LF cosicche` siano direttamente visualizzabili. Il programma principale provvede a concatenare le varie linee inserendo CR-LF e riconoscendo il fine testo con il carattere '? ricevuto a inizio linea.

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   CFILT   filtraggio commenti                               ;
; DESCRIZIONE                                                 ;
;   Copia una stringa in un'altra eliminando tutte le parti   ;
;   inserite tra '; (compreso) e CR (escluso).                ;
; INTERFACCIA                                                 ;
;   JSR PC, CFILT                                             ;
;                                                             ;
;   R1      input   puntatore alla stringa sorgente           ;
;   R2      input   puntatore alla stringa destinazione       ;
; USA                                                         ;
;   --                                                        ;
;-------------------------------------------------------------;
        EOS     = 0
        CR      = 15
CFILT:
        SAVE    <R0,R1,R2>      ; salva registri
CFI1:   MOVB    (R1)+, R0
        CMPB    #';, R0         ; inizio commento ?
        BNE     CFI2            ; salta se no
CFI3:   MOVB    (R1)+, R0
        CMPB    #CR, R0         ; cerca fine linea
        BNE     CFI3
        MOVB    R0, (R2)+       ; copia CR
        MOVB    (R1)+, (R2)+    ; LF
        BR      CFI1
CFI2:   MOVB    R0, (R2)+       ; copia carattere
        CMPB    #EOS, R0
        BNE     CFI1
        RESTORE <R2,R1,R0>      ; ripristina registri
        RTS     PC

;*************************************************************;
; PROGRAMMA PRINCIPALE                                        ;
;   CFLTPR      prova filtro testo assembly                   ;
; DESCRIZIONE                                                 ;
;   Riceve un testo assembly da STDIN diviso in righe,        ;
;   lo trasforma in una unica stringa, lo filtra e lo         ;
;   invia a STDOUT.                                           ;
; USA                                                         ;
;   Memo: PRG FPRG                                            ;
;   Subr: PNEWL GETS STLEN CFILT PUTS                         ;
;*************************************************************;
```

```
        LF      = 12
CFLTPR:
        JSR     PC, PNEWL
        MOV     #PRG, R1
PR2:    MOV     #80., R2
        JSR     PC, GETS        ; riceve una linea
        CMPB    (R1), #'?       ; fine testo
        BEQ     PR1             ; salta se si`
        JSR     PC, STLEN
        ADD     R3, R1          ; R1 punta a EOS
        MOVB    #CR, (R1)+      ; CR-LF al posto di EOS
        MOVB    #LF, (R1)+
        BR      PR2
PR1:    MOVB    #EOS, (R1)      ; fine testo
        MOV     #PRG, R1        ; testo originale
        MOV     #FPRG, R2       ; testo filtrato
        JSR     PC, CFILT
        MOV     R2, R1
        JSR     PC, PUTS
        BR      PROVA

        .CSECT  DAT
PRG:    .BLKB   2000
FPRG    .BLKB   2000
++++++++
```

Esercizio 4.3

Si scriva una subroutine filtro che riceve come ingresso una stringa di caratteri ASCII a 7 bit, terminata da EOS, eseguendo su di essa le seguenti operazioni:

- conversione minuscolo-maiuscolo per tutti i caratteri alfabetici minuscoli;

- sostituzione di tutti i caratteri TAB, sequenza di spazi e sequenze di TAB e spazi con un unico spazio;

- sostituzione dei caratteri di controllo diversi da BEL, BS, TAB, LF, VTAB, FF, CR con un byte con $b_7 = 1$ e i rimanenti 7 bit corrispondenti al carattere ottenuto sommando 100 (ottale) al codice del carattere di controllo;

- tutti i rimanenti caratteri vengono lasciati inalterati.

L'operazione della routine si traduce in un compattamento della stessa. Si scriva inoltre un programma di prova che riceve una stringa da STDIN e la visualizza su STDOUT, a filtraggio eseguito, sostituendo i caratteri con $b_7 = 1$ con il carattere '~ seguito dal carattere identificativo memorizzato nella stringa. Il carattere stampabile '~ e` rappresentato da "~~".

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   ASCFLT  filtro spazi e caratteri controllo                ;
; DESCRIZIONE                                                 ;
;   Compatta una stringa eliminando gli spazi multipli;       ;
```

```
;   trasforma minuscolo in maiuscolo e rende stampabili i    ;
;   caratteri di controllo diversi da quelli che controllano ;
;   la stampa.                                               ;
; INTERFACCIA                                                ;
;   JSR PC, ASCFLT                                           ;
;                                                            ;
;   R1      input   puntatore stringa da trattare            ;
; USA                                                        ;
;   Subr: SKIPBL LO2UPC                                      ;
;------------------------------------------------------------;
        BSIGN   = 200           ; maschera segno byte
        EOS     = 0
        BEL     = 7
        CR      = 15
ASCFLT:
        SAVE    <R0,R1,R2>      ; salva registri
        MOV     R1, R2          ; copia puntatore
ASF1:   JSR     PC, SKIPBL
        MOVB    (R1)+, R0       ; copia non spazio
        CMPB    #EOS, R0
        BEQ     ASF2            ; salta se fine stringa
        CMPB    #' , R0
        BHI     ASF3            ; salta se di controllo
        JSR     PC, LO2UPC      ; maiuscolo
ASF4:   MOVB    R0, (R2)+
        BR      ASF1
ASF3:   CMPB    #BEL, R0        ; carattere di controllo
        BHI     ASF5            ; salta se da convertire
        CMPB    #CR, R0
        BHI     ASF4
ASF5:   ADD     #BSIGN+100, R0  ; b7 = 1, add 100
        BR      ASF4
ASF2:   MOVB    #EOS, (R2)+     ; fine stringa
        RESTORE <R2,R1,R0>      ; ripristina registri
        RTS     PC
```

La routine SKIPBL ha lo scopo di compattare a spazio unico gli spazi multipli, incrementando di conseguenza i due puntatori sorgente e destinazione.

```
;------------------------------------------------------------;
; NOME                                                       ;
;   ISBLK   verifica carattere spazio o tab                  ;
; DESCRIZIONE                                                ;
;   Verifica se il carattere e` uno spazio o un tab.         ;
; INTERFACCIA                                                ;
;   JSR PC, ISBLK                                            ;
;                                                            ;
;   R0.l    input   carattere da verificare                  ;
;   Z       output  = 1 se carattere spazio                  ;
; USA                                                        ;
;   --                                                       ;
;------------------------------------------------------------;
        TAB     = 11
ISBLK:
        CMPB    #TAB, R0
        BEQ     ISB1            ; TAB
        CMPB    #' , R0         ; SPACE
ISB1:   RTS     PC
```

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   SKIPBL  riduzione degli spazi a uno                       ;
; DESCRIZIONE                                                 ;
;   Copia un solo carattere spazio se trova una sequenza.     ;
; INTERFACCIA                                                 ;
;   JSR PC, SKIPBL                                            ;
;                                                             ;
;   R1      input   puntatore sottostringa da esaminare       ;
;   R2      input   puntatore sottostringa destinazione       ;
;   R1      output  puntatore sottostringa non spazio         ;
;   R2      output  incrementato se trovato spazio            ;
; USA                                                         ;
;   Subr: ISBLK                                               ;
;   Regs: R0                                                  ;
;-------------------------------------------------------------;
SKIPBL:
        MOVB    (R1), R0
        JSR     PC, ISBLK
        BEQ     SKB1            ; primo spazio
        RTS     PC              ; nessuno spazio
SKB1:   MOVB    #' , (R2)+      ; inserisci spazio unico
SKB2:   INC     R1
        MOVB    (R1), R0        ; cerca primo non spazio
        JSR     PC, ISBLK
        BEQ     SKB2
        RTS     PC
```

Per la visualizzazione della stringa trattata, e` utilizzata la routine FPUTS che riconosce la presenza dei caratteri speciali (bit 7 = 1) convertiti nella coppia di caratteri stampabili richiesta.

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   FPUTS   output stringa con caratteri contr. trasformati   ;
; DESCRIZIONE                                                 ;
;   Effettua l'invio di una stringa su STDOUT convertendo i   ;
;   caratteri speciali nella sequenza '~ carattere            ;
;   identificativo, Il carattere '~ viene visualizzato con    ;
;   due caratteri '~.                                         ;
; INTERFACCIA                                                 ;
;   JSR PC, FPUTS                                             ;
;                                                             ;
;   R1      input   puntatore stringa da visualizzare         ;
; USA                                                         ;
;   Subr: PUT                                                 ;
;-------------------------------------------------------------;
FPUTS:
        MOV     R0, -(SP)       ; salva registri
        MOV     R1, -(SP)
FPU4:   MOVB    (R1)+, R0
        BPL     FPU1            ; salta se normale
        BICB    #BSIGN, R0
        MOV     R0, -(SP)       ; salva registro
        MOVB    #'~, R0         ; carattere speciale
        JSR     PC, PUT
        MOV     (SP)+, R0       ; ripristina registro
        BR      FPU2
FPU1:   CMPB    R0, #EOS
        BEQ     FPU3            ; salta se fine stringa
```

```
        CMPB    R0, #'~
        BNE     FPU2            ; salta se non speciale
        JSR     PC, PUT         ; out 2 volte
FPU2:   JSR     PC, PUT
        BR      FPU4
FPU3:   MOV     (SP)+, R1       ; ripristina registri
        MOV     (SP)+, R0
        RTS     PC

;*************************************************************;
;       esempio di prova
;
        STRSIZ = 400
PROVA:
        MOV     #STR1, R1       ; puntatore stringa
        MOV     #STRSIZ, R2     ; dimensione stringa
        JSR     PC, GETS        ; acquisizione
        JSR     PC, ASCFLT      ; filtraggio
        JSR     PC, FPUTS       ; visualizzazione
        JSR     PC, PNEWL       ; fine linea
        BR      PROVA

        .CSECT  DAT
STR1:   .BLKB   STRSIZ
++++++++
```

Esercizio 4.4

Si scriva un programma che visualizzi su STDOUT in binario il contenuto delle locazioni di memoria comprese fra i due indirizzi N1 e N2. La visualizzazione di ciascun carattere va affidata ad una routine che viene attivata dall'interrupt associato a STDOUT. In questo modo, mentre il processo di visualizzazione procede autonomamente, un secondo processo puo` far uso del tempo di CPU che rimane inutilizzato, incrementando iterativamente un contatore binario a 48 bit, realizzato mediante la concatenazione di 3 parole di memoria.

La routine di servizio puo` eseguire autonomamente la conversione a carattere ASCII, essendo un'operazione semplice, da effettuare sui singoli bit di ciascuna parola. Allo scopo vengono aggiornate una locazione che fa da puntatore in memoria ed una con funzione di buffer della parola correntemente indirizzata. Raggiunto il limite superiore N2, la routine di servizio disabilita le interruzioni dal dispositivo.

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   OUTIR   routine di servizio visualizz. memoria            ;
; DESCRIZIONE                                                 ;
;   Risponde ad un interrupt da STDOUT inviando il bit        ;
;   successivo della corrente parola di memoria oppure        ;
;   considerando una nuova parola. Se perviene alla fine      ;
;   dell'area interessata, si autodisattiva.                  ;
; INTERFACCIA                                                 ;
;   --                                                        ;
; USA                                                         ;
;   Memo: COU VAL AXBUF ARCSR                                 ;
;-------------------------------------------------------------;
```

```
        LF   = 12
        CR   = 15
        EI   = 100
        OUVEC = 64
        OUPSW = 200
        AXCSR = 177564
        AXBUF = 177566
OUTIR:
        TST     COU
        BEQ     OUI1            ; salta se fine numero
        BMI     OUI2            ; salta se nuovo numero
OUI3:   MOV     R0, -(SP)       ; salva registro
        CLR     R0              ; COU > 0, R0<-0, C<-0
        ROL     VAL             ; C<-bit; b0<-0
        ROL     R0              ; b0 come bit corrente
        BIS     R0, VAL         ; rotazione su VAL
        ADD     #'0, R0
        MOVB    R0, @#AXBUF     ; out bit
        DEC     COU
OUI4:   MOV     (SP)+, R0
        RTI
OUI1:
        MOVB    #CR, @#AXBUF    ; out CR
        DEC     COU
        RTI
OUI2:
        MOVB    #LF, @#AXBUF    ; out LF
        CMP     PTR, N2
        BHI     OUI5            ; salta se fine area
        MOV     @PTR, VAL       ; carica nuovo valore
        ADD     #2, PTR
        MOV     #16., COU       ; iniz. contatore
        RTI
OUI5:
        BICB    #EI, @#AXCSR    ; disabilita interruzioni
        RTI

;*************************************************************;
; PROGRAMMA PRINCIPALE                                        ;
;   OBINPR      prova conversione binario con interruzioni    ;
; DESCRIZIONE                                                 ;
;   Dopo aver inizializzato l'interruzione su STDOUT per      ;
;   la conversione e visualizzazione, compie attivita` di     ;
;   fondo, incrementando un contatore a 48 bit.               ;
; USA                                                         ;
;   Memo: PTR COU OUVEC AXCSR AXBUF W0 W1 W2                  ;
;*************************************************************;
OBINPR:
        MOV     N1, PTR
        MOV     #-1, COU
        MOV     #OUVEC, R1      ; carica vettore int.
        MOV     #OUTIR, 0(R1)
        MOV     #OUPSW, 2(R1)
        BISB    #EI, @#AXCSR    ; abilita interruzioni
        MOVB    #CR, @#AXBUF    ; primo carattere
        CLR     W0              ; iniz. contatore 48 bit
        CLR     W1
        CLR     W2
MA1:    INC     W0
        BNE     MA1             ; salta se <> 0
```

```
        INC     W1
        BNE     MA1             ; salta se <> 0
        INC     W2
        BR      MA1

        .CSECT  DAT
COU:    .WORD   0
VAL:    .WORD   0
PTR:    .WORD   0
W0:     .WORD   0
W1:     .WORD   0
W2:     .WORD   0
N1:     .WORD   140000
N2:     .WORD   150000
++++++++
```

Esercizio 4.5

Si consideri una tabella di corrispondenza tra simboli e valori organizzata in modo che ciascun elemento della tabella sia costituito da quattro parole di 16 bit. Le prime tre contengono (uno per ciascun byte) i codici ASCII della stringa di 6 caratteri costituente il simbolo; la quarta contiene il valore associato al simbolo. Gli elementi della tabella sono memorizzati in posizioni contigue e l'ultimo elemento e` seguito da una parola nulla. Si scriva la subroutine LOOKUP in grado di ricercare un simbolo nella tabella e, se trovato, di ritornare il valore associato al simbolo. La subroutine deve prevedere un ritorno d'errore diverso da quello normale se il simbolo non viene trovato.

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   COMP    Confronto tra simboli                             ;
; DESCRIZIONE                                                 ;
;   Determina se due simboli sono eguali o meno.              ;
; INTERFACCIA                                                 ;
;   JSR PC, COMP                                              ;
;                                                             ;
;   R0      input   indirizzo primo simbolo                   ;
;   R1      input   indirizzo secondo simbolo                 ;
;   Z       output  1 se equali, 0 altrimenti                 ;
; USA                                                         ;
;   --                                                        ;
;-------------------------------------------------------------;
COMP:
        CMP     0(R0), 0(R1)    ; compara primo word
        BNE     EX1
        CMP     2(R0), 2(R1)    ; compara secondo word
        BNE     EX1
        CMP     4(R0), 4(R1)    ; compara terzo word
EX1:    RTS     PC

;-------------------------------------------------------------;
; NOME                                                        ;
;   LOOKUP  subroutine di ricerca su tabella di simboli       ;
; DESCRIZIONE                                                 ;
;   Confronta una tabella di simboli con un simbolo fornito   ;
;   secondo lo schema seguente e se trovato restituisce il    ;
```

```
;   valore associato, altrimenti effettua un ritorno di      ;
;   errore.                                                  ;
;   +-------+-------+                                        ;
;   |   B   |   A   |   elemento della tabella               ;
;   +-------+-------+                                        ;
;   |   D   |   C   |                                        ;
;   +-------+-------+                +-------+-------+       ;
;   |   F   |   E   |        simbolo |   Y   |   S   |       ;
;   +-------+-------+                +-------+-------+       ;
;   |     valore    |                |   B   |   M   |       ;
;   +-------+-------+                +-------+-------+       ;
;                                    |   L   |   O   |       ;
;                                    +-------+-------+       ;
;                                                            ;
;           +-------+                                        ;
;           | El 1  |                                        ;
;           +-------+                                        ;
;           | El 2  |                                        ;
;           +-------+      configurazione tabella            ;
;           .       .                                        ;
;           +-------+                                        ;
;           | El n  |                                        ;
;           +-------+                                        ;
;           |   0   |                                        ;
;           +-------+                                        ;
; INTERFACCIA                                                ;
;   JSR R2, LOOKUP                                           ;
;   BR NOK          ; ritorno di errore                      ;
;   ...             ; ritorno normale                        ;
;                                                            ;
;   R0      input   indirizzo tabella                        ;
;   R1      input   puntatore simbolo                        ;
;   R1      output  valore associato                         ;
; USA                                                        ;
;   Subr: COMP                                               ;
;   Regs: R0 R2(JSR)                                         ;
;-----------------------------------------------------------;
LOOKUP:
        TST     (R0)            ; fine tabella ?
        BEQ     ENDED           ; salta se si
        JSR     PC, COMP
        BEQ     ENDED           ; salta se trovato
        ADD     #8., R0
        BR      LOOKUP
ENDED:  TST     (R0)
        BEQ     NOTF
FOUND:  MOV     6(R0), R1
        TST     (R2)+
NOTF:   RTS     R2

;************************************************************;
;       esempio di prova
;
PROVA:
        MOV     #TABLE, R0
        MOV     #SYMB, R1
        JSR     R2, LOOKUP
        BR      NOK
OK:     ...
NOK:    ...
```

```
        ...
TABLE:  .ASCII /TIZIO1/
        .WORD   12
        .ASCII /CARLO1/
        .WORD   23
        .ASCII /PIPPO2/
        .WORD   34
        .ASCII /TIZIO2/
        .WORD   45
        .ASCII /TIZIO1/
        .WORD   56
        .ASCII /PLUTO3/
        .WORD   67
        .WORD   0

SYMB:   .BLKW   3       ; sistemare in questo spazio
                        ; il simbolo da ricercare
++++++++
```

Esercizio 4.6

Definire una subroutine che simuli il comportamento di un Flip-Flop di tipo D con ingressi asincroni. La subroutine riceve nei 4 bit meno significativi di R0 il valore degli ingressi (S in b_0, R in b_1, C in b_2, D in b_3) e fornisce nei 2 bit meno significativi di R2 le uscite (Q in b_0, ~Q in b_1). Il registro R1 e` usato come puntatore alla locazione che rappresenta lo stato di un FF. Si realizzi un programma che attiva ciclicamente 2 istanze della subroutine che rappresentano i FF collegati come in figura. Lo stesso programma fornisce ingressi 0 per F-S-C se nessun tasto sul terminale collegato a STDIN e` premuto oppure la configurazione corrispondente a 1..7 se uno di questi tasti numerici viene premuto. Il programma visualizza ad ogni ciclo il valore delle uscite B e D.

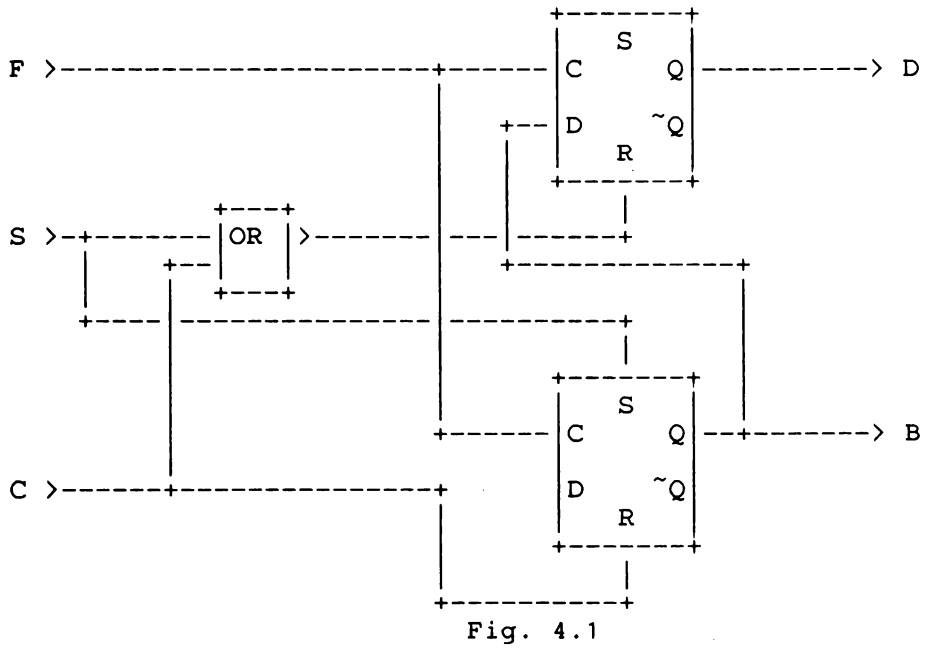


Fig. 4.1

Il FF considerato e` un componente dotato di stato (binario). Le uscite, una opposta rispetto all'altra, comunicano all'esterno tale stato che puo` modificarsi sul fronte di salita della linea C (clock) oppure per imposizione di valore 1 su una delle sue linee S (set) o R(reset): in quest'ultimo caso il valore di Q viene comunque forzato a 1 (set) o a 0 (reset). Sul fronte di salita di C il FF assume come valore di uscita quello presente all'ingresso D in quell'istante.

Nella presente realizzazione, dapprima si esaminano i valori di S e di R e se uno dei due e` a 1 si agisce sullo stato di conseguenza. In linea di principio, la presenza di un 1 in entrambi questi ingressi e` una configurazione vietata: qui si dara` priorita` alla funzione di reset. Il fronte di salita del segnale C viene qui interpretato come derivata tra il valore corrente e quello presente sul medesimo ingresso all'istante precedente, opportunamente memorizzato. Gli istanti vanno considerati in senso discreto e corrispondono alle successive chiamate della routine di simulazione.

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   DSRFF   Simulazione Flip-Flop tipo D con ingr. asincroni  ;
; DESCRIZIONE                                                 ;
;   Subroutine di simulazione per Flip-Flop di tipo D con     ;
;   ingressi asincroni S-R                                    ;
;                     +-------+                               ;
;                     |   S   |                               ;
;                     |C     Q|                               ;
;                     |       |                               ;
;                     |D    ~Q|                               ;
;                     |   R   |                               ;
;                     +-------+                               ;
;                                                             ;
;   Lo stato del FF e` contenuto in una locazione in cui      ;
;   vengono salvati, ad ogni ciclo, Q in b0, ~Q in b1 e C in  ;
;   b2. La commutazione avviene sul fronte del clock che,     ;
;   ad un istante k, e` data da ~C(k-1) & Ck.                 ;
; INTERFACCIA                                                 ;
;   JSR PC, DSRFF                                             ;
;                                                             ;
;   R0      input   4 bit ingressi (SRCD)                     ;
;   R1      input   puntatore stato FF                        ;
; USA                                                         ;
;   --                                                        ;
;-------------------------------------------------------------;
        BIT0 = 1
        BIT2 = 4
        BC01 = 177774
        EOS = 0
DSRFF:
        MOV     R2, -(SP)       ; salva registri
        MOV     R0, -(SP)
        ROR     R0              ; test S
        BCS     SONE
        ROR     R0              ; test R
        BCS     SZERO
        MOV     (SP), R0        ; ripristina R0
        MOV     (R1), R2        ; carica stato
        BIC     R2, R0          ; b2 <- ~C(k-1)Ck
        BIT     #BIT2, R0       ; test bit
```

```
        BEQ     NOVAR           ; no commutazione se 0
        MOV     (SP), R0        ; ripristina R0
        ROR     R0
        ROR     R0
        ROR     R0
        ROR     R0              ; Cy = D
        ROL     (R1)            ; set Q(stato) = D
NOVAR:  MOV     (SP)+ , R0      ; set C(stato)= clock
        BIT     #BIT2, R0
        BEQ     CLK0            ; salta se C = 0
        BIS     #BIT2, (R1)
        BR      A1
CLK0:   BIC     #BIT2, (R1)
A1:     ASR     (R1)            ; set ~Q(stato)
        ROR     (R1)
        CLC
        BMI     QIS1
        SEC
QIS1:   ROL     (R1)
        ROL     (R1)
        MOV     (SP)+, R2       ; ripristina registro
        RTS     PC
SONE:   BIS     #BIT0, (R1)
        BR      NOVAR
SZERO:  BIC     #BIT0, (R1)
        BR      NOVAR
```

La simulazione della rete proposta puo` avvenire dapprima definendo due istanze di FFD, rappresentate da altrettante locazioni di stato, e chiamando ciclicamente una routine che, a sua volta, chiama la routine di simulazione dei singoli FF, realizzando da programma le connessioni della rete.

```
;-----------------------------------------------------------;
; NOME                                                      ;
;   CICLE   Simulazione rete sequenziale                    ;
; DESCRIZIONE                                               ;
;   Subroutine per l'esecuzione di un ciclo di simulazione  ;
;   della rete richiesta.                                   ;
;               b3  b2  b1      b0                        ;
;   input FF0 = Q1  F   (S|C)   0                         ;
;   input FF1 = 0   F   C       S                         ;
; INTERFACCIA                                               ;
;   JSR PC, CICLE                                         ;
;                                                         ;
;   R0      input   3 bit ingressi (b0=F b1=S b2=C)       ;
;   R2      input   uscite (b0=D b1=B)                    ;
; USA                                                       ;
;   Regs: R0 R1 R2                                        ;
;   Memo: FF0 FF1                                         ;
;   Subr: DSRFF                                           ;
;-----------------------------------------------------------;
CICLE:
        MOV     R0, -(SP)       ; salva registro
        MOV     R0, R2          ; carica ingressi in R2
        MOV     FF1, R0         ; stato seconda istanza
        ROR     R2              ; S,R,C,D = 0,S|C,F,Q1
        ROL     R0              ; b1=Q1, b0=F
        MOV     R2, R3          ; salva R2
```

```
        ROR     R2
        BIS     R3, R2          ; b0 = S | C
        ROR     R2
        ROL     R0              ; b2=Q1, b1=F, b0 = S|C
        ASL     R0              ; b3=Q1, b2=F, b1=S|C b0=0
        MOV     #FF0, R1        ; prima istanza
        JSR     PC, DSRFF
        MOV     (SP)+, R0       ; ripristina ingressi
        MOV     R0, R2          ; S,R,C,D = S,C,F,0
        ASL     R0
        ROR     R2              ; b0=S, b1=C
        ROL     R0              ; b1=0, b0=F
        ASL     R0
        ASL     R0              ; b3=0, b2=F
        BIC     #BC01, R2
        BIS     R2, R0          ; b3=0, b2=F, b1=C, b0=S
        MOV     #FF1, R1        ; seconda istanza
        JSR     PC, DSRFF
        MOV     (R1), R0        ; carica stato seconda istanza
        ROR     R0
        ROL     R2              ; b0=Q1
        MOV     FF0, R0         ; carica stato prima istanza
        ROR     R0
        ROL     R2              ; b1=Q1, b0=Q0
        RTS     PC
```

I programmi di prova sono due: uno per il singolo FF e uno per l'intera rete. Per la rete, il programma di prova e` un ciclo che chiama la routine di simulazione di rete alternativamente alla acquisizione non sospensiva dei caratteri da STDIN. A quest'ultima funzione viene dedicata la routine NGET.

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   NGET    legge un carattere da tastiera                    ;
; DESCRIZIONE                                                 ;
;   Legge dalla tastiera un carattere senza attesa            ;
; INTERFACCIA                                                 ;
;   JSR PC, NGET                                              ;
;                                                             ;
;   R0.l    output  carattere letto                           ;
;   C       output  1 se carattere valido                     ;
; USA                                                         ;
;   Memo: ARCSR ARBUF                                         ;
;-------------------------------------------------------------;
        ARCSR   = 177560
        ARBUF   = 177562
NGET:
        MOVB    @#ARCSR, R0     ; estensione segno
        ROL     R0              ; salva segno
        BCC     NOWA            ; salta se non disponibile
        MOVB    @#ARBUF, R0
NOWA:   RTS     PC


;*************************************************************;
; PROGRAMMA PRINCIPALE                                        ;
;   PROFF       prova simulazione un FFD                      ;
; DESCRIZIONE                                                 ;
;   Invia, sulla base di un carattere numerico disponibile    ;
```

```
;   su STDIN, una configurazione d'ingresso per un FFD e      ;
;   visualizza l'uscita.                                      ;
; USA                                                         ;
;   Memo: MSG1 FF0 DELAY                                      ;
;   Subr: NGET PUT PUTS DSRFF PNEWL                           ;
;***********************************************************;
PROFF:
        CLR     R3
RIP1:
        JSR     PC, NGET        ; test su tasto premuto
        BCS     SETK1
        MOVB    #'0, R0         ; nessun tasto premuto
SETK1:  JSR     PC, PUT
        CMP     #'9, R0         ; conversione
        BCC     SETN1
        SUB     #'A-#'9-1, R0
SETN1:  MOV     #MSG1, R1
        JSR     PC, PUTS
        SUB     #'0, R0
        MOV     #FF0, R1
        JSR     PC, DSRFF       ; simulazione FF0
        MOV     FF0, R2
        BIC     #BC01, R2
        ADD     #'0, R2
        MOV     R2, R0          ; valore d'uscita
        JSR     PC, PUT
        JSR     PC, PNEWL
        MOV     DELAY, R4       ; ritardo
L11:    SOB     R3, L11
        SOB     R4, L11
        BR      RIP1
```

In merito al ritardo, una stima approssimativa fa affermare che esso e` pari a:

```
        DT = 65537 * t(SOB) * DELAY
```

per cui, supponendo t(SOB)=2.4 micros. (corrispondente al tempo min con REFRESH e 16 bit bus mode nel processore DCT11 a 7.5 Mhz) si ottiene circa DT=157.3*DELAY ms.

```
;***********************************************************;
; PROGRAMMA PRINCIPALE                                      ;
;   PRONET      prova rete sequenziale                      ;
; DESCRIZIONE                                               ;
;   Invia, sulla base di un carattere numerico disponibile  ;
;   su STDIN, una configurazione d'ingresso per la rete di  ;
;   figura e ne visualizza l'uscita.                        ;
; USA                                                       ;
;   Memo: MSG1 DELAY                                        ;
;   Subr: NGET PUT PUTS CICLE PNEWL                         ;
;***********************************************************;
PRONET:
        CLR     R3
RIP2:
        JSR     PC, NGET        ; test su tasto premuto
        BCS     SETK2
        MOVB    #'0, R0         ; nessun tasto premuto
SETK2:  JSR     PC, PUT         ; eco
        MOV     #MSG1, R1
```

```
        JSR     PC, PUTS
        SUB     #'0, R0         ; simulazione
        JSR     PC, CICLE
        BIC     #BC01, R2
        ADD     #'0, R2
        MOV     R2, R0          ; valori uscita
        JSR     PC, PUT
        JSR     PC, PNEWL
        MOV     DELAY, R4       ; ritardo
L12:    SOB     R3, L12
        SOB     R4, L12
        BR      RIP2

FF0:    .WORD   0
FF1:    .WORD   0

MSG1:   .ASCII /  <<--ingresso  uscita -->> /
        .BYTE   EOS
        .EVEN

DELAY:  15.
++++++++
```

Esercizio 4.7

Si scriva una subroutine che costruisce una tabella T[1..N] inserendo nell'elemento T[i] il numero dei bit uguali ad uno della rappresentazione binaria di i. L'indirizzo iniziale della tabella T e la sua estensione N sono passati alla subroutine come parametri. Si scriva un programma che, dopo aver costruito la tabella TAB[1.100], visualizzi in ottale il numero complessivo di bit uguali ad uno contenuti nella tabella stessa.

```
;-------------------------------------------------------------;
; NOME                                                        ;
;  TBIT     inizializzazione bit table                        ;
; DESCRIZIONE                                                 ;
;   Provvede alla costruzione di una tabella che contiene     ;
;   in ogni locazione il numero di bit pari a 1 della         ;
;   rappresentazione binaria dell'indice associato            ;
; INTERFACCIA                                                 ;
;   JSR PC, TBIT                                              ;
;                                                             ;
;   R2      input   N estensione tabella                      ;
;   R3      input   puntatore alla tabella                    ;
; USA                                                         ;
;   Subr: NBIT1                                               ;
;-------------------------------------------------------------;
TBIT:
        MOV     R0, -(SP)       ; salva registro
        MOV     #1, R0          ; inizializzazione indice
L2:     JSR     PC, NBIT1       ; bit pari a 1 in indice
        MOV     R1, (R3)+       ; carica valore e
                                ; incrementa puntatore
        INC     R0
        SOB     R2, L2
        MOV     (SP)+, R0       ; ripristina registro
        RTS     PC
```

```
;************************************************************;
;       esempio di prova
;
PROVA:
        CLR     R4              ; inizializza contatore
        MOV     #100., R2       ; inizializza dimensione
        MOV     #TAB, R3        ; inizializza puntatore
        JSR     PC, TBIT        ; costruisce tabella
        MOV     #100., R2       ; inizializza dimensione
        MOV     #TAB, R3        ; inizializza puntatore
L3:     MOV     (R3)+, R0       ; carica locazione
        JSR     PC, NBIT1       ; calcola numero bit pari a 1
        ADD     R1, R4          ; somma al contatore
        SOB     R2, L3
        MOV     #VAL, R1        ; conversione ottale
        MOV     R4, R0
        JSR     PC, AWO2OC
        MOV     #VAL, R1        ; output
        JSR     PC, PUTS
        HALT


        .CSECT  DAT
        STRSIZ  = 12
VAL:    .BLKB   #STRSIZ
        .EVEN
TAB:    .BLKW   100.
++++++++
```

Esercizio 4.8

Si definiscano coroutine che collaborano nella realizzazione di un programma che riceve caratteri da STDIN e li visualizza in STDOUT previo trattamento.

La prima coroutine riceve i singoli caratteri da STDIN e comunica via registro con la seconda, fornendo egualmente singoli caratteri. Il carattere comunicato coincide con quello letto se stampabile. Se il carattere letto e` di controllo, la prima coroutine invia alla seconda una sequenza di tre caratteri: BEL, '~ e il carattere il cui codice e` quello ricevuto da STDIN sommato a 100 (ottale). Il particolare carattere stampabile '~ viene tradotto nella coppia "~~".

La seconda coroutine riceve i singoli caratteri dalla prima e li memorizza in un buffer. Essa inoltre riconosce l'invio di un carattere BEL, di cui fa eco immediata su STOUT, e memorizza la coppia di caratteri successivi nel buffer, conteggiandoli come carattere singolo. Dopo ogni gruppo di 16 caratteri, la coroutine invia a STDOUT l'intero contenuto del buffer, svuotandolo, e il totale dei caratteri ricevuti.

```
;------------------------------------------------------------;
; NOME                                                       ;
;   INPUT   coroutine di input                               ;
; DESCRIZIONE                                                ;
;   Provvede a leggere i caratteri da STDIN. Se il carattere ;
;   e` di controllo, invia all'altra coroutine il carattere  ;
```

```
;   BEL seguito da ~ e dal codice del carattere ricevuto      ;
;   aumentato del codice di @. Il carattere ~ viene tradotto  ;
;   in ~~. Tutti gli altri caratteri vengono comunicati via   ;
;   registro all'altra coroutine.                             ;
; INTERFACCIA                                                 ;
;   JMP INPUT                                                 ;
; USA                                                         ;
;   Coru: OUTPUT                                              ;
;   Subr: GET                                                 ;
;   Regs: R0 (carattere trasmesso) R1(salvataggio)            ;
;-------------------------------------------------------------;
        BEL     = 7
INPUT:
        MOV     #OUTPUT, -(SP)  ; prepara indirizzo di resume
INP1:   JSR     PC, GET
        CMPB    R0, #'          ; carattere di controllo ?
        BGE     INP2            ; salta se no
        MOV     R0, R1          ; si, salva carattere
        MOV     #BEL, R0
        JSR     PC, @(SP)+      ; resume
        MOV     #'~, R0
        JSR     PC, @(SP)+      ; resume
        MOV     R1, R0          ; ripristina carattere
        ADD     #'@, R0         ; trasformazione, somma 100
        JSR     PC, @(SP)+      ; resume
        BR      INP1
INP2:   CMPB    R0, #'~         ; carattere speciale ?
        BNE     INP3            ; salta se no
        JSR     PC, @(SP)+      ; resume, out due volte
INP3:   JSR     PC, @(SP)+      ; resume
        BR      INP1

;-------------------------------------------------------------;
; NOME                                                        ;
;   OUTPUT  coroutine di output                               ;
; DESCRIZIONE                                                 ;
;   Provvede a ricevere i singoli caratteri da INPUT e li     ;
;   memorizza nel buffer, salvo il carattere BEL che viene    ;
;   subito inviato a STDOUT. Le coppie di caratteri in cui    ;
;   il primo e` ~ sono conteggiati come un solo carattere.    ;
;   Dopo 16. caratteri provvede alla visualizzazione del      ;
;   buffer e del numero totale di caratteri ricevuti e gia`   ;
;   visualizzati.                                             ;
; INTERFACCIA                                                 ;
;   JSR PC, @(SP)+                                            ;
; USA                                                         ;
;   Coru: INPUT                                               ;
;   Subr: PNEWL PUT PUTS WO2DEC                               ;
;   Regs: R0 (non modificato) R2 R3 R4                        ;
;   Memo: BUF MSG1 STR1                                       ;
;-------------------------------------------------------------;
OUTPUT:
        CLR     R3              ; contatore
        CLR     R2              ; offset
        JSR     PC, PNEWL
OUT1:   CMPB    R0, #BEL        ; unico carattere speciale
        BNE     OUT2            ; salta se no
        JSR     PC, PUT         ; eco
        JSR     PC, @(SP)+      ; resume, riceve ~
        MOVB    R0, BUF(R2)     ; carica in buffer
```

```
        INC     R2              ; incrementa offset
        JSR     PC, @(SP)+      ; resume
OUT2:   MOVB    R0, BUF(R2)     ; carica in buffer
        INC     R2              ; incrementa offset
        INC     R3              ; incrementa contatore
        MOV     R3, R4          ; output buffer ?
        BIC     #177760, R4
        BNE     OUT3            ; salta se no
        CLR     R5
        MOV     R0, -(SP)       ; salva carattere
OUT4:   MOVB    BUF(R5), R0
        JSR     PC, PUT
        INC     R5              ; incrementa offset
        SOB     R2, OUT4        ; ciclo su caratteri buffer
        MOV     #MSG1, R1       ; R2 = 0
        JSR     PC, PUTS
        MOV     R3, R0          ; contatore
        MOV     #STR1, R1
        JSR     PC, WO2DEC
        JSR     PC, PUTS
        JSR     PC, PNEWL
        MOV     (SP)+, R0       ; ripristina carattere
OUT3:   JSR     PC, @(SP)+      ; resume
        BR      OUT1


        .CSECT  DAT
        EOS     = 0
        STRSIZ = 20.
MSG1:   .ASCII /  Numero totale = /
        .BYTE   EOS
        .EVEN
STR1:   .BLKB   STRSIZ
BUF:    .BLKB   2 * STRSIZ      ; per tener conto del
                                ; raddoppio di '~
++++++++
```

Esercizio 4.9

Si scriva un programma che consenta di esaminare e di modificare il contenuto di word in memoria centrale, secondo le seguenti modalita`:

a) Dopo che l'operatore ha fornito, tramite tastiera, l'indirizzo del word da esaminare (numero ottale rappresentabile con 16 bit), lo termina con il carattere '/': il programma ne visualizza a questo punto il contenuto, sempre in ottale.

b) Successivamente all'operatore deve essere consentito di:

- modificare tale contenuto, fornendo quello nuovo da tastiera, sempre in ottale, terminato da '/', passando all'indirizzo successivo; oppure:
- lasciare inalterato il word indirizzato mediante l'invio del carattere CR (indipendentemente da quanto gia` battuto) e passare al punto a); oppure:
- lasciare inalterato il word indirizzato mediante l'invio del carattere LF (indipendentemente da quanto gia` battuto), passando

subito all'esame del word successivo.

Il programma deve cautelarsi contro la possibilita` che l'operatore commetta errori.

Viene dapprima definita la routine GETADR che permette di ricevere da STDIN un valore ottale, terminato da '/, CR o LF. Il punto di ritorno e` diversificato nei vari casi. Questa routine viene utilizzata sia per fornire l'indirizzo che l'eventuale nuovo valore da inserire nella locazione riferita. Il programma, nel suo complesso, salvaguarda se stesso dall'essere modificato.

```
INIZIO:          ; per stabilire occupazione programma
;
;... inserire le altre routine
;
;-------------------------------------------------------------;
; NOME                                                        ;
;   GETADR       acquisizione valore 16 bit in ottale         ;
; DESCRIZIONE                                                 ;
;   Permette l'acquisizione di un valore a 16 bit in ottale   ;
;   da tastiera, effettuando ritorni diversi a seconda del    ;
;   terminatore (/ CR LF).                                    ;
; INTERFACCIA                                                 ;
;   JSR R5, GETADR                                            ;
;   BR  IFSLA                                                 ;
;   BR  IFCR                                                  ;
;   BR  IFLF                                                  ;
;                                                             ;
;   R1       output  valore acquisito                         ;
; USA                                                         ;
;   Subr: GET PUT PNEWL                                       ;
;-------------------------------------------------------------;
         BEL      = 7
GETADR:
         MOV      R0, -(SP)        ; salva registro
A1:      MOV      #'?, R0          ; R0h = 0
         JSR      PC, PUT
         CLR      R1               ; init valore
A2:      JSR      PC, GET
         CMPB     #'0, R0
         BGT      A3               ; salta se < '0
         CMPB     #'7, R0
         BLT      A3               ; salta se > '7
         JSR      PC, PUT          ; '0..'7, echo
         SUB      #'0, R0          ; 0..7
         ASL      R1               ; Val*8
         BCS      A4               ; overflow
         ASL      R1
         BCS      A4
         ASL      R1
         BCS      A4
         ADD      R0, R1           ; somma cifra
         BR       A2
A3:      CMPB     #'/, R0          ; altri caratteri
         BEQ      A5               ; salta se conferma valore
         CMPB     #CR, R0
         BEQ      A6               ; salta se quit valore
         CMPB     #LF, R0
```

```
        BEQ     A7              ; salta se prossimo
        MOVB    #BEL, R0        ; carattere illegale, BIP
        JSR     PC, PUT
        BR      A2
A4:     MOVB    #'?, R0         ; richiesto nuovo valore
        JSR     PC, PUT
        MOVB    #BEL, R0
        JSR     PC, PUT
        JSR     PC, PNEWL
        BR      A1
A7:     TST     (R5)+           ; ritorno LF
A6:     TST     (R5)+           ; ritorno CR
A5:     MOV     (SP)+, R0       ; ritorno /
        RTS     R5

;-------------------------------------------------------------;
; NOME                                                        ;
;   PUTVAL      visualizza   valore 16 bit in ottale          ;
; DESCRIZIONE                                                 ;
;   Permette la visualizzazione di un valore a 16 bit in      ;
;   ottale in forma semplice.                                 ;
; INTERFACCIA                                                 ;
;   JSR PC, PUTVAL                                            ;
;                                                             ;
;   R1      input   valore da visualizzare                    ;
; USA                                                         ;
;   Subr: PUT                                                 ;
;-------------------------------------------------------------;
PUTVAL:
        MOV     R1, -(SP)       ; salva registri
        MOV     R2, -(SP)
        MOV     #'0/2, R0       ; prima cifra = 0,1
        ASL     R1
        ROL     R0
        JSR     PC, PUT
        MOV     #5, R2          ; contatore cifre successive
P1:     MOV     #'0/8., R0      ; altre cifre 0..7
        ASL     R1              ; shift 3 bit
        ROL     R0
        ASL     R1
        ROL     R0
        ASL     R1
        ROL     R0
        JSR     PC, PUT
        SOB     R2, P1          ; ciclo su numero cifre
        MOV     (SP)+, R2       ; ripristina registri
        MOV     (SP)+, R1
        RTS     PC

;-------------------------------------------------------------;
; NOME                                                        ;
;   PUTCON      visualizza contenuto word da 16 bit in ottale;
; DESCRIZIONE                                                 ;
;   Permette la visualizzazione in ottale di un indirizzo di  ;
;   16 bit e del word da esso riferito.                       ;
; INTERFACCIA                                                 ;
;   JSR PC, PUTCON                                            ;
;                                                             ;
;   R1      input   indirizzo word da visualizzare            ;
; USA                                                         ;
```

```
;   Subr: PUTVAL PNEWL PUT                                          ;
;-----------------------------------------------------------------;
PUTCON:
        MOV     R0, -(SP)       ; salva registri
        MOV     R1, -(SP)
        JSR     PC, PNEWL
        JSR     PC, PUTVAL      ; visualizza indirizzo
        MOVB    #'=, R0
        JSR     PC, PUT
        MOV     (R1), R1
        JSR     PC, PUTVAL      ; visualizza contenuto
        MOV     (SP)+, R1       ; ripristina registri
        MOV     (SP)+, R0
        RTS     PC
```

Confrontando il riferimento corrente con il campo di occupazione del programma, e` possibile rendere immodificabile il programma.

```
;*****************************************************************;
; PROGRAMMA PRINCIPALE                                            ;
;   MEMOPR      esame e modifica di locazioni di memoria          ;
; DESCRIZIONE                                                     ;
;   Consente la visualizzazione e l'eventuale modifica di         ;
;   locazioni di memoria, proteggendo il programma.               ;
; USA                                                             ;
;   Memo: INIZIO FINE                                             ;
;   Subr: PNEWL GETADR PUTCON PUT                                 ;
;*****************************************************************;
MEMOPR:
        MOV     #INIZIO, R3     ; campo occupazione programma
        MOV     #FINE, R4
C1:     JSR     PC, PNEWL
        JSR     R5, GETADR      ; acquisisce indirizzo
        BR      C2              ; /
        BR      C1              ; CR
C6:     TST     (R1)+           ; LF
C2:     JSR     PC, PUTCON      ; visualizza contenuto
        CMP     R1, R3          ; non ammessa modifica per
                                ; area programma
        BLO     C3              ; salta se < INIZIO
        CMP     R1, R4
        BLO     C4              ; salta se in area programma
C3:     MOV     R1, -(SP)       ; salva indirizzo
        JSR     PC, GETADR
        BR      C5              ; /
        BR      C1              ; CR
        MOV     (SP)+, R1       ; LF
        BR      C6
C5:     MOV     R1, @0(SP)
        MOV     (SP)+, R1
        BR      C6
C4:     MOVB    #BEL, R0        ; in area programma
        JSR     PC, PUT
        BR      C1
FINE:
```

Si suggerisce al lettore di modificare il programma in modo che si possa fornire in anticipo una lista di dimensione variabile di campi di indirizzi entro i quali imporre una protezione analoga a quella sopra realizzata.

++++++++

Esercizio 4.10

Si scriva un programma costituito da una routine di interruzione collegata a STDIN che provvede a fare l'eco su STDOUT dei caratteri ricevuti e un segmento principale che provvede ad incrementare un contatore a 16. bit e a visualizzarne il valore in decimale con una certa cadenza in posizione predefinita dello schermo. L'eco dei caratteri viene invece effettuato nella posizione corrente del cursore.

Occorre porre attenzione al fatto che, durante la visualizzazione del valore del contatore, non deve essere effettuato l'eco dell'input in quanto cio` avverrebbe con il cursore in posizione scorretta. Allo scopo il salvataggio del cursore, il posizionamento predefinito, la visualizzazione del contatore e il ripristino del cursore devono avvenire in zona protetta ovvero a interruzioni disabilitate.

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   EGETI   routine di servizio per interrupt da tastiera     ;
; DESCRIZIONE                                                 ;
;   Risponde ad un interrupt da tastiera leggendo il          ;
;   carattere battuto ed effettuando l'eco su terminale.      ;
; INTERFACCIA                                                 ;
;   --                                                        ;
; USA                                                         ;
;   Memo: ARBUF                                               ;
;   Subr: PUT                                                 ;
;-------------------------------------------------------------;
        ARBUF = 177562
EGETI:
        MOV     R0, -(SP)       ; salva R0
        MOVB    @#ARBUF, R0
        JSR     PC, PUT         ; eco
        MOV     (SP)+, R0       ; ripristina R0
        RTI



;*************************************************************;
; PROGRAMMA PRINCIPALE                                        ;
;   EGETPR      eco con interruzioni                          ;
; DESCRIZIONE                                                 ;
;   Incrementa un contatore a 16 bit, visualizzandone in      ;
;   cadenza il valore, permettendo l'eco di caratteri da      ;
;   STDIN a STDOUT mediante interruzioni.                     ;
; USA                                                         ;
;   Memo: CONT DELAY INVEC ARCSR STR                          ;
;   Subr: CLS WO2DEC SAVCUR PUTSRC RESCUR                     ;
;*************************************************************;
        INVEC   = 60
        GETPSW  = 200
        PEI     = 0             ; processor interrupt enable
        PDI     = 340
        EI      = 100
        STRSIZ  = 21
        SETCUR  = 60*256.+1     ; riga 1, colonna 48.
```

```
EGETPR:
        JSR     PC, CLS          ; clear screen
        CLR     @#CONT           ; azzera contatore
        CLR     @#DELAY
        MOV     #INVEC, R1
        MOV     #EGETI, 0(R1)     ; vettore interruzioni
        MOV     #GETPSW, 2(R1)
        BISB    #EI, @#ARCSR
LOOP:
        INC     @#CONT           ; contatore
LOOP1:  INC     @#DELAY
        BNE     LOOP1
        MOV     @#CONT, R0
        MOV     #STR, R1
        JSR     PC, WO2DEC       ; converte in decimale il
                                 ; valore del contatore
        MTPS    #PDI             ; disable interrupt
        JSR     PC, SAVCUR       ; salva la posizione attuale
                                 ; del cursore sullo schermo
        MOV     #SETCUR, R2      ; posiziona il cursore per la
        JSR     PC, PUTSRC       ; visualizzazione del contato-
                                 ; re
        JSR     PC, RESCUR       ; ripristina la posizione del
                                 ; cursore
        MTPS    #PEI
        BR      LOOP

        .CSECT  DAT
STR:                             ; area destinata a contenere
        .BLKB   #STRSIZ          ; le cifre ASCII che rappre-
        .EVEN                    ; sentano il valore decimale
CONT:   .BLKW 1
DELAY:  .BLKW 1
++++++++
```

Esercizio 4.11

Si scrivano le routine di servizio alle interruzioni CLKMR collegata al RTC a 800Hz e IGETI collegata a STDIN, e un programma principale in modo che, di ogni carattere inviato da STDIN, venga fatto l'eco su STDOUT, inserendo un CR-LF ogni N caratteri. Allo scadere di ogni M secondi, viene inviato a STDOUT, su una nuova riga, il numero di caratteri ricevuti nel periodo immediatamente precedente.

Quanto richiesto viene qui realizzato con l'utilizzo di contatori e di flag di comunicazione tra i tre processi: quello di acquisizione ed eco dei caratteri, quello di aggiornamento dell'orologio e quello rappresentato dal programma principale che provvede a verificare alternativamente le due condizioni di raggiungimento di fine linea e dello scadere del tempo. A causa di questo occorre porre attenzione affinche` l'accesso alle variabili comuni sia regolato in modo che la successione di eventi sia corretta. In particolare non deve verificarsi che un carattere battuto non venga conteggiato o che l'intervallo di tempo previsto non sia calcolato correttamente. Certe operazioni devono pertanto essere inserite in una zona protetta da interferenze e la protezione puo` essere ottenuta disabilitando le interruzioni che attiverebbero il processo interferente.

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   CLKMR   routine di servizio del clock a 800 Hz            ;
; DESCRIZIONE                                                 ;
;   Tenendo conto della frequenza del clock, la routine       ;
;   aggiorna un contatore di secondi e se il conteggio supera;
;   il valore di M, segnala tale condizione al programma      ;
;   principale mediante un flag.                              ;
; INTERFACCIA                                                 ;
;   --                                                        ;
; USA                                                         ;
;   Memo: TIME CLKSR M GOOUT                                  ;
;-------------------------------------------------------------;
        CLKSR   = 177444        ; indirizzo registro stato
CLKMR:
        MOVB    #10, @#CLKSR    ; disabilita inter. clock
        INC     TIME            ; contatore tick
        CMP     TIME, M         ; >= M sec
        BLO     NONEW
        CLR     TIME            ; azzera orologio
        INC     GOOUT           ; pone a 1 flag output NCHAR
NONEW:  MOVB    #7, @#CLKSR     ; abilita inter. clock
        RTI

;-------------------------------------------------------------;
; NOME                                                        ;
;   IGETI   routine di servizio STDIN                         ;
; DESCRIZIONE                                                 ;
;   Esegue l'eco su STDOUT del carattere letto e incrementa   ;
;   due contatori.                                            ;
; INTERFACCIA                                                 ;
;   --                                                        ;
; USA                                                         ;
;   Memo: ARBUF LINS NCHAR                                    ;
;   Subr: PUT                                                 ;
;-------------------------------------------------------------;
        ARBUF   = 177562
IGETI:
        MOVB    @#ARBUF, R0
        JSR     PC, PUT
        INC     LINS
        INC     NCHAR
        RTI

;*************************************************************;
; PROGRAMMA PRINCIPALE                                        ;
;   IGETPR      conteggio caratteri con interruzioni          ;
; DESCRIZIONE                                                 ;
;   Provvede a inviare un CR-LF ogni N caratteri ricevuti,    ;
;   sincronizzandosi con il processo di input.                ;
; USA                                                         ;
;   Memo: CLKVEC GETVEC LINS NCHAR ARCSR CLKSR N GOOUT STR    ;
;   Subr: PNEWL PUTS WO2DEC                                   ;
;*************************************************************;
        CLKVEC  = 104           ; clock vettore interruzioni
        CLKPSW  = 300
        GETVEC  = 60            ; stdin interrupt vector
        GETPSW  = 200           ; priorita` della routine (4)
        ARCSR   = 177560
        EI      = 100           ; interrupt enable
```

```
        STRSIZ  = 20.
IGETPR:
        MOV     #CLKVEC, R1
        MOV     #CLKMR, 0(R1)
        MOV     #CLKPSW, 2(R1)  ; imposta interrupt vector
        MOV     #GETVEC, R1
        MOV     #IGETI, 0(R1)   ; vettore input
        MOV     #GETPSW, 2(R1)
        JSR     PC, PNEWL
        CLR     LINS            ; iniz. contatori
        CLR     NCHAR
        BISB    #EI, @#ARCSR    ; abilita interruzioni input
        MOVB    #7, @#CLKSR     ; abilita inter. clock
MA1:
        BICB    #EI, @#ARCSR    ; disabilita inter. input
                                ; a protezione dell'accesso a
                                ; LINS
        CMP     LINS, N         ; fine linea ?
        BLO     MA2             ; salta se no
        CLR     LINS
        JSR     PC, PNEWL
MA2:
        BISB    #EI, @#ARCSR    ; abilita interruzioni input
        TST     GOOUT           ; output NCHAR
        BEQ     MA1             ; salta se no
        CLR     GOOUT
        JSR     PC, PNEWL
        MOV     #MSG, R1
        JSR     PC, PUTS
        BICB    #EI, @#ARCSR    ; disabilita inter. input
                                ; a protezione dell'accesso a
                                ; NCHAR
        MOV     NCHAR, R0       ; output NCHAR
        CLR     NCHAR
        BISB    #EI, @#ARCSR    ; abilita interruzioni input
        MOV     #STR, R1
        JSR     PC, WO2DEC
        JSR     PC, PUTS
        JSR     PC, PNEWL
        BR      MA1


        .CSECT  DAT
;       Variabili e messaggi

TIME:   .WORD   0               ; orologio

GOOUT:  .WORD   0               ; flag output NCHAR
LINS:   .WORD   0               ; lunghezza linea eco

NCHAR:  .WORD   0               ; numero totale caratteri

M:      .WORD   15.*800.        ; 15. secondi

N:      .WORD   10.             ; 16. caratteri per linea

MSG:    .ASCII /Totale caratteri = /
        .BYTE   EOS
        .EVEN
STR:    .BLKB   STRSIZ
```

Si noti che la maggiore priorita` dell'interruzione di clock rispetto a STDIN e il fatto che l'operazione di output del numero di caratteri deve essere assunta di durata inferiore all'intervallo M, non occorre prevedere protezioni sull'accesso alla variabili interessate dalla routine CLKMR. Pero`, e` possibile che l'eco di qualche carattere si inframmezzi alla visualizzazione del messaggio e del numero di caratteri? In caso di risposta affermativa, si cerchi di ovviare.
++++++++

Esercizio 4.12

Si supponga di aver collegato un dispositivo a tasto (KEY) all'interruzione non mascherabile PF (VA=24): ad ogni pressione del tasto corrisponde la generazione di una richiesta di interruzione. Si scriva un programma che consenta di misurare la durata degli intervalli di tempo tra due pressioni del tasto successive (una con funzione di START e una di STOP). A tale scopo si utilizzi il clock a 800 Hz per mantenere l'informazione di tempo. Il programma principale dovra` provvedere alla visualizzazione dell'intervallo trascorso in millisecondi.

Poiche` l'interruzione KEY e` non mascherabile, non si ha alcuna garanzia, in linea di principio, che la cadenza di impulsi START-STOP sia sufficientemente lunga da consentire la visualizzazione del tempo intercorso durante un intervallo prima che si completi l'intervallo successivo: nel caso presente, la condizione e` invece di fatto garantita poiche` la generazione delle interruzioni e` legata ad organi meccanici, quindi lenti. E` conveniente imporre che un intervallo non possa partire prima che il tempo accumulato in quello precedente non sia stato visualizzato. La cosa puo` essere ottenuta facendo in modo che la routine di servizio di KEY conservi uno stato particolare, che e` anche quello iniziale, in cui permane fintanto che non si e` completata una visualizzazione precedente. Si indichera` con -1 tale stato mentre lo stato 0 e` conseguente ad una interruzione di tipo START ammissibile e lo stato 1 ad una interruzione di tipo STOP. La transizione dallo stato 0 allo stato 1, l'unica possibile verso lo stato 1, comporta la comunicazione al programma principale del tempo di visualizzazione. Lo stato e` conservato nella variabile KEYST mentre il flag GOOUT viene posto a 1 dalla routine di servizio come comando di visualizzazione e a 0 dal programma principale come conseguenza del completamento di quest'ultima.

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   CLKTR   routine di servizio del clock a 800 Hz            ;
; DESCRIZIONE                                                 ;
;   Tenendo conto della frequenza del clock, la routine       ;
;   aggiorna un contatore di tick (1.25 millisecondi).        ;
; INTERFACCIA                                                 ;
;   --                                                        ;
; USA                                                         ;
;   Memo: TIME CLKSR                                          ;
;-------------------------------------------------------------;
        CLKSR   = 177444        ; indirizzo registro stato
CLKTR:
        MOVB    #10, @#CLKSR    ; disabilita inter. clock
```

```
        INC     TIME            ; contatore tick
        MOVB    #7, @#CLKSR     ; abilita inter. clock
        RTI

;-------------------------------------------------------------;
; NOME                                                        ;
;   KEYIR   routine di servizio KEY                           ;
; DESCRIZIONE                                                 ;
;   Esegue l'azzeramento del contatore di tick e comunica     ;
;   al programma principale il numero da visualizzare.        ;
; INTERFACCIA                                                 ;
;   --                                                        ;
; USA                                                         ;
;   Memo: TIME OUTV GOOUT                                     ;
;-------------------------------------------------------------;
KEYIR:
        TST     KEYST           ; stato corrente
        BPL     GO1             ; salta se non -1
        TST     GOOUT           ; visualizzazione completa ?
        BNE     NOGO            ; salta se no, permane in -1
        INC     KEYST           ; START, stato <- 0
ISTA:   CLR     TIME            ; azzera contatore
NOGO:   RTI
GO1:    BEQ     GO2             ; salta se stato = 0
        DEC     KEYST           ; stato <- 0
        TST     GOOUT           ; visualizzazione completa ?
        BEQ     ISTA            ; salta se si`, START
        DEC     KEYST           ; stato <- -1
        RTI
GO2:    INC     KEYST           ; stato <- 1
        INC     GOOUT           ; comando visualizzazione
        MOV     TIME, OUTV      ; comunica conteggio
        RTI
```

Per la visualizzazione del tempo in millisecondi, occorre tener conto che il contatore deve essere moltiplicato per 1.25 cioe`:

```
        TMS = TIME * 1.25 = TIME + TIME/4
```

```
;*************************************************************;
; PROGRAMMA PRINCIPALE                                        ;
;   KEYPR       prova interruzioni da KEY                     ;
; DESCRIZIONE                                                 ;
;   Provvede alla visualizzazione in modo sincronizzato del   ;
;   tempo in millisecondi tra una coppia START-STOP da KEY.   ;
; USA                                                         ;
;   Memo: CLKVEC KEYVEC KEYST GOOUT OUTV STR                  ;
;   Subr: PNEWL PUTS WO2DEC                                   ;
;*************************************************************;
        LOWBY   = 177400        ; maschera byte meno sign.
        CLKVEC  = 104           ; clock interrupt vector
        CLKPSW  = 300           ; priorita` 6
        KEYVEC  = 24            ; KEY interrupt vector
        KEYPSW  = 340           ; interrupt non mascherabile
                                ; -> priorita` massima
        STRSIZ  = 20.
KEYPR:
        MOV     #CLKVEC, R1
        MOV     #CLKTR, 0(R1)
        MOV     #CLKPSW, 2(R1)  ; imposta interrupt vector
```

```
        MOV     #KEYVEC, R1
        MOV     #KEYIR, 0(R1)    ; vettore KEY
        MOV     #KEYPSW, 2(R1)
        JSR     PC, PNEWL
        CLR     GOOUT            ; iniz. variabili
        MOV     #-1, KEYST
        MOVB    #7, @#CLKSR      ; abilita inter. clock
MA1:
        TST     GOOUT            ; output OUTV ?
        BEQ     MA1              ; salta se no
        CLR     GOOUT
        MOV     #MSG, R1
        JSR     PC, PUTS
        MOV     OUTV, R0         ; output OUTV
        MOV     R0, R1
        ASR     R0
        ASR     R0
        ADD     R1, R0
        MOV     #STR, R1
        JSR     PC, WO2DEC
        JSR     PC, PUTS
        JSR     PC, PNEWL
        BR      MA1
;
;       Variabili e messaggi
;
        .CSECT  DAT
TIME:   .WORD   0                ; orologio
KEYST:  .WORD   -1               ; stato KEYIR
GOOUT:  .WORD   0                ; flag output OUTV
OUTV:   .WORD   0                ; valore da visualizzare
MSG:    .ASCII /Millisecondi trascorsi = /
        .BYTE   EOS
        .EVEN
STR:    .BLKB   STRSIZ
```

Si noti che nessuna protezione e` necessaria tra il processo di conteggio del tempo e quello principale in quanto non interagiscono su variabili comuni.

Si suggerisce al lettore modificare il programma in modo che l'invio su STDOUT del messaggio e dell'intervallo trascorso avvenga con minimo impegno della CPU mediante l'uso di una routine di servizio per l'output.
++++++++

Esercizio 4.13

Si gestisca un buffer di caratteri circolare, di dimensione pari a BUFSIZ potenza di due, sottoforma di coda FIFO (First In First Out). Allo scopo si prevedano le routine BUFIN e BUFOUT di inserzione ed estrazione di un carattere nel e dal buffer. Si collaudino le due routine, utilizzando il buffer come area di comunicazione tra un processo ad interruzione che acquisisce caratteri da STDIN e un processo di consumo che visualizza su STDOUT a cadenza fissa i caratteri nello stesso ordine in cui sono stati ricevuti da STDIN. Se un carattere viene ricevuto a buffer pieno, si invia BEL a STDOUT e il carattere viene perso.

Il buffer e` organizzato come un'area di posizione e dimensioni note e con una coppia di puntatori relativi, quello di testa (BH) e quello di coda (BT). Il primo denota la posizione del primo carattere da estrarre mentre il secondo la posizione della prima locazione libera su cui inserire. La condizione di buffer vuoto e` data da BH=BT mentre quella di buffer pieno da BH=BT+1 oppure BH=0 e BT=BUFSIZ-1. L'aggiornamento dei puntatori relativi e` facilitato dal fatto che la dimensione del buffer e` una potenza di 2.

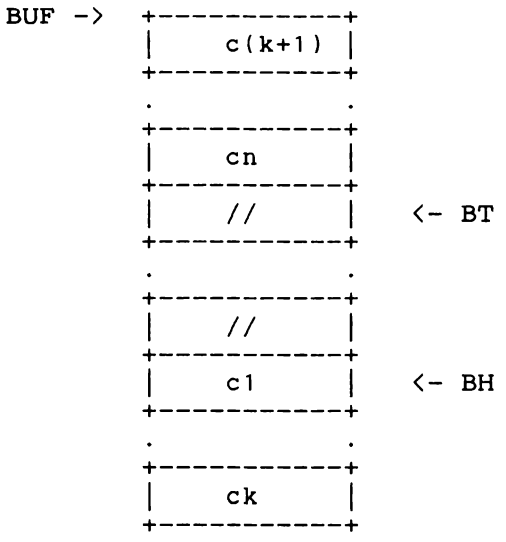


Fig. 4.2

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   BUFIN   inserzione nel buffer                             ;
; DESCRIZIONE                                                 ;
;   Il carattere fornito viene inserito in coda al buffer     ;
;   circolare se non gia` pieno. La condizione di pieno e`    ;
;   data dalla presenza di una sola locazione libera del      ;
;   buffer (per distinguerla dalla condizione di vuoto).      ;
; INTERFACCIA                                                 ;
;   JSR PC, BUFIN                                             ;
;                                                             ;
;   R0.l    input   carattere da inserire                     ;
;   C       output  1 se buffer pieno                         ;
; USA                                                         ;
;   Regs: R1 R2                                               ;
;   Memo: BUF BT BH                                           ;
;-------------------------------------------------------------;
        MASK = 177770       ; maschera relativa
BUFIN:
        MOV     BT, R1      ; carica puntatore coda
        INC     R1          ; incrementa punt. modulo
        BIC     #MASK, R1   ; dimensione buffer
        CMP     R1, BH      ; buffer pieno ?
        BEQ     FULL
        MOV     BT, R2      ; carica puntatore coda
        MOVB    R0, BUF(R2) ; inserisce carattere
        MOV     R1, BT      ; aggiorna puntatore
```

```
        CLC                     ; operazione OK
        RTS     PC
FULL:   SEC                     ; buffer pieno
        RTS     PC


;-------------------------------------------------------------;
; NOME                                                        ;
;   BUFOUT  estrazione dal buffer                             ;
; DESCRIZIONE                                                 ;
;   Il carattere inserito in testa al buffer viene da esso    ;
;   estratto se il buffer non e` vuoto. La condizione di      ;
;   vuoto e` data dalla eguaglianza dei due puntatori.        ;
; INTERFACCIA                                                 ;
;   JSR PC, BUFOUT                                            ;
;                                                             ;
;   R0.1    output  carattere estratto                        ;
;   C       output  1 se buffer vuoto                         ;
; USA                                                         ;
;   Regs: R1                                                  ;
;   Memo: BUF BT BH                                           ;
;-------------------------------------------------------------;
BUFOUT:
        CMP     BT, BH          ; confronta puntatori
        BEQ     EMPTY
        MOV     BH, R1          ; carica puntatore di testa
        MOVB    BUF(R1), R0     ; estrae carattere
        INC     R1              ; aggiorna puntatore
        BIC     #MASK, R1
        MOV     R1, BH
        CLC                     ; operazione OK
        RTS     PC
EMPTY:  SEC                     ; buffer vuoto
        RTS     PC


;-------------------------------------------------------------;
; NOME                                                        ;
;   BGETI   routine di servizio per interrupt da STDIN        ;
; DESCRIZIONE                                                 ;
;   Risponde ad un interrupt da STDIN ricevendo il carattere  ;
;   prodotto ed effettuando l'inserimento nel buffer.         ;
;   Poiche` il buffer e` condiviso con il processo di output  ;
;   a tempo, l'accesso al buffer e` controllato in mutua      ;
;   esclusione mediante il flag BUSY. Poiche` l'interrupt da  ;
;   clock e` piu` prioritario, questa routine non valuta il   ;
;   flag ma si limita a modificarlo.                          ;
; INTERFACCIA                                                 ;
;   --                                                        ;
; USA                                                         ;
;   Subr: BUFIN PUT                                           ;
;   Memo: BUSY ARBUF                                          ;
;-------------------------------------------------------------;
        BEL = 7
        ARBUF   = 177562
BGETI:
        MOV     R0, -(SP)       ; salva R0
        MOV     R1, -(SP)       ; salva R1
        MOV     R2, -(SP)       ; salva R2
        MOV     @#ARBUF, R0
```

```
        INC     BUSY            ; la sezione che segue e`
                                ; eseguita in mutua esclusione
                                ; con quella analoga del
                                ; processo di consumo.
        JSR     PC, BUFIN
        MOV     #0, BUSY        ; C non modificato
        BCC     GEX1
        MOV     #BEL, R0
        JSR     PC, PUT
GEX1:
        MOV     (SP)+, R2       ; ripristina R2
        MOV     (SP)+, R1       ; ripristina R1
        MOV     (SP)+, R0       ; ripristina R0
        RTI


;-------------------------------------------------------------;
; NOME                                                        ;
;   CLKBR   routine di servizio del clock a 800 Hz            ;
; DESCRIZIONE                                                 ;
;   Tenendo conto della frequenza del clock, la routine       ;
;   aggiorna un contatore e dopo un certo quanto estrae un    ;
;   carattere dal buffer e lo visualizza. Si sincronizza con  ;
;   il processo di produzione dei caratteri per l'accesso     ;
;   al buffer.                                                ;
; INTERFACCIA                                                 ;
;   --                                                        ;
; USA                                                         ;
;   Subr: BUFOUT PUT                                          ;
;   Memo: TIME BUSY FLOUT CLKSR                               ;
;-------------------------------------------------------------;
        CLKSR   = 177444        ; indirizzo registro di stato
        MAXT = 1600.            ; 2 sec. delay
CLKBR:
        MOVB    #10, @#CLKSR    ; disabilita INTA
        INC     TIME            ; contatore tick
        CMP     TIME, #MAXT     ; quanto raggiunto ?
        BLO     NOUT1
        INC     FLOUT           ; si, flag output <- 1
        CLR     TIME            ; contatore <- 0
NOUT1:  TST     FLOUT           ; output richiesto ?
        BEQ     NOUT2
        TST     BUSY            ; si, buffer accessibile ?
        BNE     NOUT2
                                ; la sezione che segue viene
                                ; eseguita in mutua esclusione
                                ; con quella analoga del
                                ; processo di produzione
        MOV     R0, -(SP)       ; salva R0
        MOV     R1, -(SP)       ; salva R1
        JSR     PC, BUFOUT      ; estrae carattere
        BCS     NOCHAR
        JSR     PC, PUT
NOCHAR:
        CLR     FLOUT
        MOV     (SP)+, R1       ; ripristina R1
        MOV     (SP)+, R0       ; ripristina R0
NOUT2:
        MOVB    #7, @#CLKSR     ; abilita INTA
        RTI
```

```
;*************************************************************;
; PROGRAMMA PRINCIPALE                                        ;
;   BUFPR       prova comunicazione via buffer                ;
; DESCRIZIONE                                                 ;
;   Inizializza il buffer di comunicazione e le routine di    ;
;   servizio per la produzione e il consumo.                  ;
; USA                                                         ;
;   Memo: TIME BUSY FLOUT BT BH CLKVEC GETVEC CLKSR ARCSR     ;
;*************************************************************;
        CLKVEC  = 104           ; clock interrupt vector
        CLKPSW  = 300
        GETVEC   = 60           ; tast. interrupt vector
        GETPSW  = 200
        ARCSR   = 177560
        CLKSR   = 177444
        EI = 100                ; interrupt enable
        BSIZ = 8.               ; dimensione buffer
BUFPR:
        CLR     TIME            ; inizializzazione variabili
        CLR     BUSY
        CLR     FLOUT
        CLR     BT
        CLR     BH

        MOV     #CLKVEC, R1
        MOV     #CLKBR, 0(R1)
        MOV     #CLKPSW, 2(R1)  ; imposta interrupt vector
        MOVB    #7, @#CLKSR     ; abilita inter. clock

        MOV     #GETVEC, R1
        MOV     #BGETI, 0(R1)   ; vettore interruzioni
        MOV     #GETPSW, 2(R1)
        BISB    #EI, @#ARCSR
LOOP:   INC     R1              ; attivita` di fondo
        BR      LOOP

        .CSECT  DAT

TIME:   .WORD 0
BUSY:   .WORD 0
FLOUT:  .WORD 0
BT:     .WORD 0
BH:     .WORD 0
BUF:    .BLKW   BSIZ
```

La protezione sull'accesso alle variabili comuni e` qui realizzata mediante un flag di mutua esclusione (BUSY). Per rendere in qualche modo simmetrico il sistema formato dai due processi di produzione e consumo, tenendo conto che in ogni caso la routine per il clock e` piu` prioritaria della routine per STDIN, la prima memorizza nel flag privato FLOUT la condizione di quanto temporale scaduto. Il flag BUSY viene valutato in successive interruzioni se FLOUT e` 1, fino a che l'accesso risulta possibile e solo a questo punto si ha il consumo del dato, mentre FLOUT viene posto a 0. Per lo stesso motivo, l'estrazione del dato diventa automaticamente indivisibile, non potendo la routine essere interrotta, e pertanto la seconda routine puo` limitarsi a modificare il flag BUSY prima dell'inserimento di un dato mediante un'unica operazione indivisibile senza una precedente valutazione dello stesso.

Si suggerisce al lettore di modificare il programma in modo che sia il programma principale a sincronizzarsi e a provvedere per l'output, riducendo la durata della routine di servizio del clock.
++++++++

Esercizio 4.14

Si definisca la routine PUTSI che effettua la visualizzazione su STDOUT di una stringa con minimo impegno del tempo di CPU mediante routine di interruzione.

La routine PUTSI offre il vantaggio di permettere attivita` concorrenti con quella di visualizzazione di una stringa. Naturalmente occorre vietare che due successive chiamate di PUTSI possano produrre effetti indesiderati: non avendo a disposizione alcun sistema di archiaviazione delle stringhe da visualizzare, e` necessario bloccare l'attivazione di una nuova visualizzazione se quella precedente non e` terminata, che e` lo scopo del flag BUSY.

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   PUTSI   visualizza una stringa mediante interrupt         ;
; DESCRIZIONE                                                 ;
;   Invia a STDOUT il primo carattere di una stringa          ;
;   e attiva la routine di servizio del dispositivo di output;
;   per l'invio dei successivi caratteri fino a EOS. Il flag  ;
;   BUSY segnala se un'altra visualizzazione e` ancora in     ;
;   corso.                                                    ;
; INTERFACCIA                                                 ;
;   JSR PC, PUTSI                                             ;
;                                                             ;
;   R1      input   puntatore alla stringa                    ;
; USA                                                         ;
;   Memo: BP BUSY AXCSR AXBUF OUVEC                           ;
;-------------------------------------------------------------;
        AXCSR = 177560
        AXBUF = 177562
        OUVEC = 64
        OUPSW = 200
        EI = 100                ; interrupt enable
PUTSI:
        TST     BUSY            ; stampa in corso ?
        BNE     PUTSI           ; si, il programma attende qui
                                ; che la stampa in corso termini
START:  MOV     R0, -(SP)       ; salva registri
        MOV     R1, -(SP)
        MOVB    (R1)+, R0
        BEQ     EX1             ; lunghezza nulla ?
        INC     BUSY
        MOV     R1, BP          ; memorizza il pointer al car. succ.
        MOV     #OUVEC, R1
        MOV     #OUTL, 0(R1)
        MOV     #OUPSW, 2(R1)   ; modifica interrupt vector
        MOVB    R0, AXBUF       ; out primo carattere
        BISB    #EI, AXCSR      ; abilita interruzioni
EX1:    MOV     (SP)+, R1       ; ripristina registri
        MOV     (SP)+, R0
        RTS     PC
```

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   OUTL    routine di servizio output su STDOUT              ;
; DESCRIZIONE                                                 ;
;   Invia i successivi caratteri di una stringa a fronte      ;
;   del segnale di interrupt corrispondente all'avvenuta      ;
;   trasmissione del carattere precedente.                    ;
; INTERFACCIA                                                 ;
;   --                                                        ;
; USA                                                         ;
;   Memo: BP AXCSR AXBUF                                      ;
;-------------------------------------------------------------;
OUTL:
        MOV     R0, -(SP)       ; salva i registri usati
        MOV     R1, -(SP)
        MOV     BP, R1          ; punta al carattere da stampare
        MOVB    (R1)+, R0
        BEQ     EOM             ; fine messaggio
        MOV     R1, BP          ; punta al carattere seguente
        MOVB    R0, AXBUF       ; out carattere
REX:
        MOV     (SP)+, R1
        MOV     (SP)+, R0       ; ripristina i registri
        RTI
EOM:
        BICB    #EI, AXCSR      ; disabilita  le interruzioni
        CLR     BUSY
        BR      REX


        .CSECT  DAT
BP:     .BLKW   1
BUSY:   .WORD   0
++++++++
```

Esercizio 4.15

Modificare la soluzione dell'esercizio precedente in modo da ovviare all'attesa passiva dovuta alla presenza di una richiesta di visualizzazione prima che termini quella precedente.

Quanto richiesto viene risolto con l'uso di una coda FIFO i cui elementi sono messaggi costituiti da un puntatore di link seguito dal corpo del messaggio che, nel caso presente, coincide con la stringa da visualizzare. L'accodamento e` eseguito dalla routine QPUTSI qualora una visualizzazione sia ancora in corso. La routine di servizio provvede invece ad una estrazione al completamento della visualizzazione precedente e, solo se la coda e` vuota, si autodisattiva: in questo caso la routine QPUTSI provvede ad una ulteriore attivazione in sede di successiva richiesta di visualizzazione.

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   ENQUE   subroutine di inserzione in coda                  ;
; DESCRIZIONE                                                 ;
;   Provvede a inserire nella coda il messaggio fornito.      ;
;   Allo scopo mantiene due puntatori (HEAD e TAIL)           ;
```

```
;   inizializzati entrambi a NIL. Si conviene che i messaggi ;
;   abbiano un word iniziale libero gestito dalla subroutine ;
;   come link.                                                ;
;            ---------       ---------       TAIL -> ---------;
;   HEAD -> | link  |------>| link  |--- .... ----->|  NIL  |;
;           ---------       ---------               ---------;
;           | car1  |       | car1  |               | car1  |;
;           ---------       ---------               ---------;
;           |  ...  |       |  ...  |               |  ...  |;
;           ---------       ---------               ---------;
;           |  EOS  |       |  EOS  |               |  EOS  |;
;           ---------       ---------               ---------;
; INTERFACCIA                                                ;
;   JSR PC, ENQUE                                            ;
;                                                            ;
;   R1      input   puntatore al messaggio da inserire       ;
;                   (campo link)                             ;
; USA                                                        ;
;   Memo: TAIL HEAD                                          ;
;------------------------------------------------------------;
        NIL = -1                ; puntatore di fine coda
ENQUE:
        CMP     R1, #NIL
        BEQ     EQ1             ; messaggio nullo
        CMP     @#TAIL, #NIL    ; la coda era vuota
        BEQ     EQ2
EQ3:    MOV     R1, @TAIL       ; link
        MOV     R1, @#TAIL
        MOV     #NIL, @R1
EQ1:    RTS     PC

EQ2:    MOV     R1, @#HEAD
        BR      EQ3



;------------------------------------------------------------;
; NOME                                                       ;
;   DEQUE   subroutine di estrazione dalla coda              ;
; DESCRIZIONE                                                ;
;   Provvede a estrarre dalla coda FIFO un messaggio.        ;
;   Valgono le medesime considerazioni fatte per ENQUE.      ;
; INTERFACCIA                                                ;
;   JSR PC, DEQUE                                            ;
;                                                            ;
;   R1      output  puntatore al messaggio estratto oppure   ;
;                   NIL se coda vuota.                       ;
; USA                                                        ;
;   Memo: TAIL HEAD                                          ;
;------------------------------------------------------------;
DEQUE:
        MOV     @#HEAD, R1
        CMP     R1, #NIL
        BEQ     DQ1             ; la coda era vuota
        MOV     @R1, @#HEAD
        CMP     @#HEAD, #NIL    ; la coda e` ora vuota?
        BNE     DQ1
        MOV     #NIL, @#TAIL    ; si
DQ1:
        RTS     PC
```

```
        .CSECT  DAT
HEAD:   .WORD   NIL                     ; puntatori della coda
TAIL:   .WORD   NIL
        .CSECT  COD


;-------------------------------------------------------------;
; NOME                                                        ;
;   QPUTSI  visualizza una stringa usando una coda di mess.   ;
; DESCRIZIONE                                                 ;
;   Invia al terminale il primo carattere di un messaggio     ;
;   e attiva la routine di servizio del dispositivo di output;
;   per l'invio dei successivi caratteri fino a EOS, se non   ;
;   e` gia` in corso un precedente output. In caso contrario  ;
;   provvede ad accodare il messaggio fornito.                ;
;              +-----+----+----+----+----+-----+             ;
;        R1 ->|link | c1 | c2 | .. | cn | EOS |             ;
;              +-----+----+----+----+----+-----+             ;
; INTERFACCIA                                                 ;
;   JSR PC, QPUTSI                                            ;
;                                                             ;
;   R1      input   puntatore al primo carattere del          ;
;                   messaggio                                 ;
; USA                                                         ;
;   Subr: ENQUE                                               ;
;   Memo: AXCSR BUSY                                          ;
;-------------------------------------------------------------;
        AXCSR = 177560
        AXBUF = 177562
        EI    = 100             ; enable interrupt
        OUVEC = 64
        OUPSW = 200
QPUTSI:
        TST     BUSY            ; stampa in corso ?
        BEQ     START
        BICB    #EI, @#AXCSR    ; LOCK, protegge accesso coda
        JSR     PC, ENQUE       ; si, accoda
        BISB    #EI, @#AXCSR    ; UNLOCK
        RTS     PC

START:  MOV     R0, -(SP)       ; salva registri
        MOV     R1, -(SP)
        CMP     R1, #NIL
        BEQ     PS1             ; messaggio nullo
        INC     R1              ; salta il link
        INC     R1
        MOVB    (R1)+, R0
        BEQ     PS1             ; lunghezza nulla ?
        INC     BUSY
        MOV     R1, BP          ; punta al carattere seguente
        MOV     #OUVEC, R1
        MOV     #QOUTL, 0(R1)
        MOV     #OUPSW, 2(R1)   ; modifica interrupt vector
        MOVB    R0, AXBUF
        BISB    #EI, AXCSR      ; abilita interruzioni
PS1:
        MOV     (SP)+, R1       ; ripristina registri
        MOV     (SP)+, R0
        RTS     PC
```

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   QOUTL   routine di servizio output su STDOUT              ;
; DESCRIZIONE                                                 ;
;   Invia i successivi caratteri di un messaggio a fronte     ;
;   del segnale di interrupt corrispondente all'avvenuta      ;
;   trasmissione del carattere precedente.                    ;
; INTERFACCIA                                                 ;
;   --                                                        ;
; USA                                                         ;
;   Subr: DEQUE                                               ;
;   Memo: BP AXBUF BUSY AXCSR                                 ;
;-------------------------------------------------------------;
QOUTL:
        MOV     R0, -(SP)       ; salva i registri usati
        MOV     R1, -(SP)
        MOV     BP, R1          ; punta al carattere da stampare
OU1:    MOVB    (R1)+, R0
        BEQ     EOM             ; fine messaggio
        MOV     R1, BP          ; punta al carattere seguente
        MOVB    R0, @#AXBUF
REX:    MOV     (SP)+, R1
        MOV     (SP)+, R0       ; ripristina i registri
        RTI
EOM:    JSR     PC, DEQUE
        CMP     R1, #NIL        ; c'era qualcosa in coda?
        BEQ     NOM             ; no
        INC     R1
        INC     R1
        BR      OU1
NOM:    BICB    #EI, @#AXCSR    ; disabilita interruzioni
        CLR     BUSY
        BR      REX
```

Anche in questo caso, essendo l'accesso alla coda possibile in modo concorrente da parte del processo di produzione, costituito ad esempio dal programma principale che chiama la subroutine QPUTSI, e del processo di consumo, legato alla routine di interruzione di output, e` necessario inserire una protezione nella subroutine in modo da rendere tale accesso mutuamente esclusivo. La protezione e` realizzata mediante le istruzioni di disabilitazione e abilitazione della interruzione interferente, qui indicate con LOCK e UNLOCK.
++++++++

Esercizio 4.16

Supponendo che l'insieme dei caratteri ASCII venga diviso nelle tre classi cifre (Digit = '0..'9), lettere (Letter = 'A..'Z, 'a..'z) e separatori (Space = tutti gli altri), definire un analizzatore lessicale che identifichi i seguenti elementi:

```
IDEN :  Letter [Letter # Digit]*
NUM  :  Digit  [Digit]*
```

cioe` gli identificatori alfanumerici e i numeri.

Il riconoscimento puo` avvenire sulla base del seguente automa a stati finiti:

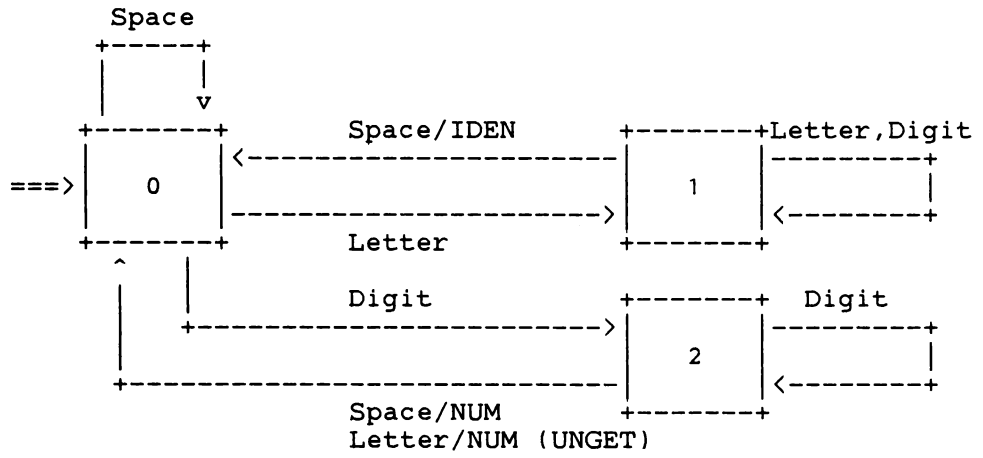


Fig. 4.3

Il programma acquisisce in forma bufferizzata i caratteri da STDIN, utilizzando le routine BUFIN e BUFOUT dell'esercizio 4.13. Come valore di uscita dell'automa viene fornita la codifica del tipo di elemento identificato. Poiche` la presenza di una lettera immediatamente dopo una serie di cifre identifica la terminazione di un numero ma anche l'inizio di un nuovo elemento di tipo identificatore, si rende necessario, in questo caso, esaminare due volte lo stesso carattere. Allo scopo viene predisposta una funzione (BUFUN) che reinserisce il carattere nel buffer. La realizzazione dell'automa e` mediante tabella di transizione di stato e di uscita.

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   CLASS       classe del carattere                          ;
; DESCRIZIONE                                                 ;
;   Determina la classe di appartenenza di un carattere       ;
;   ASCII secondo la seguente tabella:                        ;
;   A..Z a..z   0   Letter                                    ;
;   0..9        1   Digit                                     ;
;   tutti gli   2   Space                                     ;
;   altri                                                     ;
; INTERFACCIA                                                 ;
;   JSR PC, CLASS                                             ;
;                                                             ;
;   R0      input   carattere fornito                         ;
;   R3      output  classe di appartenenza                    ;
; USA                                                         ;
;   --                                                        ;
;-------------------------------------------------------------;
CLASS:
        CMPB    R0, #'0
        BLO     SPA
        CMPB    R0, #'9
        BLOS    DIG
        CMPB    R0, #'A
        BLO     SPA
        CMPB    R0, #'Z
        BLOS    LET
        CMPB    R0, #'a
        BLO     SPA
        CMPB    R0, #'z
```

```
        BLOS    LET
SPA:    MOV     #2, R3          ; Space
        RTS     PC
LET:    MOV     #0, R3          ; Letter
        RTS     PC
DIG:    MOV     #1, R3          ; Digit
        RTS     PC

;-------------------------------------------------------------;
; NOME                                                        ;
;   BUFUN   reinserimento in buffer                           ;
; DESCRIZIONE                                                 ;
;   Il carattere precedentemente estratto viene reinserito    ;
;   nel buffer. Il reinserimento ha luogo solo se il buffer   ;
;   non e` pieno.                                             ;
; INTERFACCIA                                                 ;
;   JSR PC, BUFUN                                             ;
;                                                             ;
;   C       output  1 se buffer pieno                         ;
; USA                                                         ;
;   Regs: R1                                                  ;
;   Memo: BH BT                                               ;
;-------------------------------------------------------------;
        MASK = 177770           ; maschera relativa
BUFUN:
        MOV     BH, R1          ; carica puntatore testa
        DEC     R1              ; undo
        BIC     #MASK, R1
        CMP     R1, BT          ; buffer pieno ?
        BEQ     FULL
        MOV     R1, BH          ; aggiorna puntatore
        CLC                     ; operazione OK
        RTS     PC

;-------------------------------------------------------------;
; NOME                                                        ;
;   BEGETI  routine di servizio per interrupt da STDIN        ;
; DESCRIZIONE                                                 ;
;   Risponde ad un interrupt da STDIN ricevendo il carattere  ;
;   prodotto ed effettuando l'inserimento nel buffer.         ;
; INTERFACCIA                                                 ;
;   --                                                        ;
; USA                                                         ;
;   Subr: GET BUFIN PUT                                       ;
;-------------------------------------------------------------;
        BEL     = 7
BEGETI:
        MOV     R0, -(SP)       ; salva registri
        MOV     R1, -(SP)
        MOV     R2, -(SP)
        JSR     PC, GET
        JSR     PC, PUT         ; eco
        JSR     PC, BUFIN       ; inserimento nel buffer
        BCC     GEX1
        MOV     #BEL, R0        ; buffer pieno
        JSR     PC, PUT
GEX1:   MOV     (SP)+, R2       ; ripristina registri
        MOV     (SP)+, R1
        MOV     (SP)+, R0
        RTI
```

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   LEX     subroutine di scansione lessicale                 ;
; DESCRIZIONE                                                 ;
;   Restituisce una stringa che e` un token letto da          ;
;   tastiera assieme alla sua qualifica. L'area riservata     ;
;   al token deve avere ampiezza minima TSIZ. Il prelievo     ;
;   dei caratteri dal buffer avviene in mutua esclusione con  ;
;   il processo di inserzione.                                ;
; INTERFACCIA                                                 ;
;   JSR PC, LEX                                               ;
;                                                             ;
;   R0      output  qualifica del token (1=IDEN, 2=NUM 0=ERR);
;   R1      input   puntatore all'area in cui viene copiato   ;
;                   il token                                  ;
; USA                                                         ;
;   Regs: R1 R2 R3 R4                                         ;
;   Subr: BUFOUT CLASS PUT BUFUN                              ;
;   Memo: STAT OTAB STAB                                      ;
;-------------------------------------------------------------;
        EOS     = 0
        ERR     = 0             ; token incompleto
        PEI     = 0             ; processor interrupt enable
        PDI     = 340
LEX:
        MOV     R1, -(SP)       ; salva puntatore token
        MTPS    #PDI            ; disabilita interruzioni
                                ; per la mutua esclusione
        JSR     PC, BUFOUT      ; acquisisce carattere
        MTPS    #PEI            ; esce dalla mutua escl.
        MOV     (SP)+, R1
        BCS     LEX             ; buffer vuoto ?
        CMP     R1, #TOK+TSIZ-1 ; no, area token piena ?
        BLO     TOKOK
        MOV     #ERR, R0        ; token incompleto
        RTS     PC
TOKOK:
        JSR     PC, CLASS       ; classe del carattere
        CMP     #2, R3          ; se Space, non copia
        BEQ     NOCOPY
        MOVB    R0, (R1)+       ; copia in token
NOCOPY: MOV     STAT, R2        ; aggiorna stato
                                ; qual = OTAB (stato, classe)
                                ; stato = STAB(stato, classe)
        ASL     R2              ; R2 = (stato*3+classe)*2
        ADD     STAT, R2        ; STAB e OTAB sono matrici 3x3
        ADD     R3, R2          ; di parole a 16 bit
        ASL     R2
        MOV     OTAB(R2), R4    ; R4 = qual
        MOV     STAB(R2), STAT
        TST     R4              ; se qual=0, token non
                                ; completo
        BEQ     LEX
        MOVB    #EOS, -(R1)     ; token completo
        MTPS    #PDI            ; disabilita interruzioni
        JSR     PC, BUFUN       ; reinserisce nel buffer
                                ; ultimo carattere non facente
                                ; parte del token corrente
        MTPS    #PEI
        BCC     BOK             ; buffer pieno ?
```

```
        MOV     #ERR, R0        ; condizione di errore
        RTS     PC
BOK:    MOV     R4, R0          ; token completo, ritorna
                                ; qualificatore
        RTS     PC

;*************************************************************;
;       esempio di prova
;
        GETVEC  = 60            ; STDIN interrupt vector
        GETPSW  = 200
        EI      = 100           ; enable interrupt
        ARCSR   = 177560
        TAB     = 11
        BSIZ = 8.               ; buffer size
        TSIZ = 64.              ; token size
PROVA:
        CLR     BT              ; inizializzazione variabili
        CLR     BH
        CLR     STAT
        MOV     #GETVEC, R1
        MOV     #BEGETI, 0(R1)  ; vettore interruzioni
        MOV     #GETPSW, 2(R1)
        BISB    #EI, @#ARCSR
AGAIN:  JSR     PC, PNEWL
        MOV     #TOK, R1
        JSR     PC, LEX         ; acquisisce un token
        JSR     PC, PNEWL
        MOV     #TOK, R1
        JSR     PC, PUTS        ; visualizza il token
        CMP     #1, R0
        BNE     ISNUM
ISID:   MOV     #IDSTR, R1
        JSR     PC, PUTS        ; visualizza il qualificatore
        BR      AGAIN
ISNUM:  MOV     #NUMSTR, R1
        JSR     PC, PUTS        ; visualizza il qualificatore
        BR      AGAIN

;
;       Variabili
        .CSECT  DAT
BT:     .WORD   0               ; buffer tail
BH:     .WORD   0               ; buffer head
STAT:   .WORD   0               ; stato automa
BUF:    .BLKW   BSIZ            ; buffer
TOK:    .BLKW   TSIZ            ; token
STAB:   .WORD   1, 2, 0, 1, 1, 0, 0, 2, 0
                        ; tabella di transizione di stato
OTAB:   .WORD   0, 0, 0, 0, 0, 1, 2, 0, 2
                        ; tabella di uscita

IDSTR:  .BYTE #TAB
        .ASCII /IDEN/
        .BYTE #EOS
        .EVEN
NUMSTR: .BYTE #TAB
        .ASCII /NUM/
        .BYTE #EOS
        .EVEN
```

Si noti che la condizione di errore e` data dal fatto che l'area riservata al token viene completamente riempita prima della terminazione del token oppure perche` in occasione del reinserimento di un carattere nel buffer si trova il buffer pieno.
++++++++

Esercizio 4.17

Si realizzi un programma che sia in grado di risolvere il problema della *Torre di Hanoi* e visualizzarne la soluzione. Il problema consiste nello spostare N dischi di larghezza diversa, impilati in un piolo in ordine di larghezza decrescente (il piu` piccolo in cima), in un altro piolo, utilizzando come deposito temporaneo un terzo piolo. Lo spostamento degli N dischi deve avvenire mediante spostamenti elementari di un singolo disco da un piolo ad un altro, garantendo che non accada mai che in un qualsiasi piolo vi sia un disco sistemato piu` in alto di un disco di larghezza inferiore.

E` questo un noto problema risolubile con una certa semplicita` in forma ricorsiva. Infatti la soluzione si puo` esprimere verbalmente in questi termini:

a) si spostano i primi (i-1) dischi dal piolo attualmente di partenza al piolo ausiliario (macro-spostamento);

b) si sposta l'i-esimo disco dal piolo di partenza al piolo destinazione (spostamento elementare);

c) si spostano gli (i-1) dischi dal piolo ausialiario a quello destinazione (macro-spostamento).

Tutti i macro-spostamenti vengono eseguiti dalla procedura ricorsiva, mentre quelli elementari vengono eseguiti da una procedura ad hoc. L'algoritmo viene chiamato una prima volta con i=N e non viene eseguito con i=0 (condizione terminale). In ogni fase di esecuzione, le funzioni di piolo di partenza, ausiliario e di destinazione vengono assunte da pioli diversi secondo una precisa permutazione che garantisce il vincolo posto. In linguaggio di programmazione:

```
RECURSIVE PROCEDURE Hanmov (Ord,  Pi,  Pa,  Pd);
/*  Ord = larghezza disco 1..N
    Pi  = piolo iniziale
    Pa  = piolo ausiliario
    Pd  = piolo destinazione
*/
begin
    if Ord<>0 then begin
        Hanmov (Ord-1, Pi, Pd, Pa);
        /*  macro-spostamento di (i-1) dischi da Pi a Pa
            utilizzando Pd */
        Inform (Ord, Pi, Pd);
        /* spostamento elementare disco i */
        Hanmov (Ord-1, Pa, Pi, Pd);
        /*  macro-spostamento di (i-1) dischi da Pa a Pd
            utilizzando Pi */
        end
    end
```

Nella presente realizzazione i parametri vengono passati nello stack, in modo da garantire la ricorsivita`.

I dischi vengono rappresentati da una sequenza di 2*Ord caratteri '= adiacenti. I pioli non sono effettivamente visualizzati ma rappresentano vincoli sul posizionamento dei dischi e sugli spostamenti. I pioli possono essere pensati come segmenti verticali di altezza sufficiente a contenere gli N dischi iniziali; sono rappresentati dalla posizione riga-colonna della loro base. I parametri tipo riga-colonna seguono le convenzione della routine PUTRC (es. 3.31). Lo spostamento di un disco deve essere visualizzato in modo che appaia che il disco viene sfilato dal piolo, spostato lateralmente e infilato nel piolo destinazione.

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   HANMOV  macro-movimento di sottotorri                     ;
; DESCRIZIONE                                                 ;
;   Effettua la movimentazione completa di una parte della    ;
;   torre da una posizione all'altra. Il posizionamento di    ;
;   colonna e` quello dei pioli. Il posizionamento di riga    ;
;   e` quello del disco alla base della sottotorre. Viene     ;
;   utilizzata la parte precisata del piolo intermedio come   ;
;   zona di transito.                                         ;
; INTERFACCIA                                                 ;
;   JSR PC, HANMOV                                            ;
;                                                             ;
;   2(SP)   input   Col.Riga finale       (3)                 ;
;   4(SP)   input   Col.Riga intermedia   (2)                 ;
;   6(SP)   input   Col.Riga iniziale     (1)                 ;
;   10(SP)  input   Tipo disco base (dimensione)              ;
; USA                                                         ;
;   Subr: INFORM HANMOV                                       ;
;-------------------------------------------------------------;
HANMOV:
        TST     10(SP)          ; Ord = 0 ?
        BEQ     HAN1            ; salta se si`
        MOV     10(SP), -(SP)
        DEC     (SP)            ; Ord-1
        MOV     10(SP), -(SP)
        DEC     (SP)            ; Col1.Riga1-1
        MOV     6(SP), -(SP)    ; Col3.Riga3
        MOV     12(SP), -(SP)   ; Col2.Riga2
        JSR     PC, HANMOV      ; muovi Ord-1 dischi
        MOV     10(SP), -(SP)   ; Ord
        MOV     10(SP), -(SP)   ; Col1.Riga1
        MOV     6(SP), -(SP)    ; Col3.Riga3
        JSR     PC, INFORM      ; muovi disco Ord
        MOV     10(SP), -(SP)
        DEC     (SP)            ; Ord-1
        MOV     6(SP), -(SP)    ; Col2.Riga2
        MOV     12(SP), -(SP)   ; Col1.Riga1
        MOV     10(SP), -(SP)
        DEC     (SP)            ; Col3.Riga3-1
        JSR     PC, HANMOV      ; muovi Ord-1 dischi
HAN1:   MOV     0(SP), 10(SP)   ; rilascia parametri
        ADD     #10, SP
        RTS     PC
```

Lo spostamento elementare e` effettuato dalla routine INFORM che

utilizza a sua volta le routine MOVHOR e MOVVER che provvedono a spostare in orizzontale e in verticale singoli dischi. Gli spostamenti vengono simulati mediante scritture e riscritture in posizioni contigue, utilizzando la routine MOVES. Le routine di movimentazione possono non essere ricorsive. Viene anche utilizzata l'informazione globale TOP che rappresenta la comune riga in cui sono posizionate le cime dei pioli.

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   MOVES   movimento elementare per un disco                 ;
; DESCRIZIONE                                                 ;
;   Effettua la visualizzazione ripetuta del carattere        ;
;   fornito a partire dalla posizione precisata. Il           ;
;   posizionamento e` riferito al primo elemento a sinistra   ;
;   del disco.                                                ;
; INTERFACCIA                                                 ;
;   JSR PC, MOVES                                             ;
;                                                             ;
;   R0.l    input   carattere da visualizzare                 ;
;   R1      input   dimensione disco                          ;
;   R2.l    input   riga di posizionamento                    ;
;   R2.h    input   colonna di posizionamento                 ;
; USA                                                         ;
;   Subr: PUTRC PUT                                           ;
;-------------------------------------------------------------;
MOVES:
        MOV     R1, -(SP)       ; salva registro
        MOV     R2, R1          ; Col.Riga
        JSR     PC, PUTRC       ; primo carattere
        MOV     0(SP), R1
        DEC     R1
        BEQ     MVS1            ; salta se solo 1
MVS2:   JSR     PC, PUT         ; caratteri successivi
        SOB     R1, MVS2        ; ciclo su numero caratteri
MVS1:   MOV     (SP)+, R1       ; ripristina registro
        RTS     PC

;-------------------------------------------------------------;
; NOME                                                        ;
;   MOVVER  movimento verticale di un disco                   ;
; DESCRIZIONE                                                 ;
;   Effettua la movimentazione verticale di un disco. Il      ;
;   posizionamento e` riferito al primo elemento a sinistra   ;
;   del disco.                                                ;
; INTERFACCIA                                                 ;
;   JSR PC, MOVVER                                            ;
;                                                             ;
;   R1      input   dimensione disco                          ;
;   R2.l    input   riga di posizionamento iniziale           ;
;   R2.h    input   colonna di posizionamento iniziale        ;
;   R3.l    input   riga di posizionamento finale             ;
;   R3.h    input   colonna di posizionamento finale (=R2.h)  ;
; USA                                                         ;
;   Subr: MOVES                                               ;
;-------------------------------------------------------------;
        DCH = '=                ; carattere disco
MOVVER:
        MOV     R0, -(SP)       ; salva registri
        MOV     R2, -(SP)
```

```
        CMP     R2, R3          ; Inizio = Fine ?
        BEQ     MVE1            ; salta se si`
        BGT     MVE2            ; salta se muove in su
MVE3:   CMP     R2, R3          ; in giu`, test fine ciclo
        BGE     MVE1            ; salta se corrente=Fine
        INC     R2              ; Riga+1
        MOVB    #DCH, R0        ; carattere disco
        JSR     PC, MOVES
        DEC     R2              ; Riga
        MOVB    #' , R0         ; cancellazione
        JSR     PC, MOVES
        INC     R2              ; Riga+1
        BR      MVE3
MVE2:   CMP     R2, R3          ; muove in su, test fine
        BLE     MVE1            ; salta se corrente=Fine
        DEC     R2              ; Riga-1
        MOVB    #DCH, R0        ; carattere disco
        JSR     PC, MOVES
        INC     R2              ; Riga
        MOVB    #' , R0         ; cancellazione
        JSR     PC, MOVES
        DEC     R2              ; Riga+1
        BR      MVE2
MVE1:   MOV     (SP)+, R2       ; ripristina registri
        MOV     (SP)+, R0
        RTS     PC

;-------------------------------------------------------------;
; NOME                                                        ;
;   MOVHOR  movimento orizzontale di un disco                 ;
; DESCRIZIONE                                                 ;
;   Effettua la movimentazione orizzontale di un disco. Il    ;
;   posizionamento e` riferito al primo elemento a sinistra   ;
;   del disco.                                                ;
; INTERFACCIA                                                 ;
;   JSR PC, MOVHOR                                            ;
;                                                             ;
;   R1      input   dimensione disco                          ;
;   R2.l    input   riga di posizionamento iniziale           ;
;   R2.h    input   colonna di posizionamento iniziale        ;
;   R3.l    input   riga di posizionamento finale (=R2.l)     ;
;   R2.h    input   colonna di posizionamento finale          ;
; USA                                                         ;
;   Subr: MOVES                                               ;
;-------------------------------------------------------------;
        ONEC = 400              ; una colonna
MOVHOR:
        MOV     R0, -(SP)       ; salva registri
        MOV     R2, -(SP)
        MOV     R4, -(SP)
        MOV     R1, R4          ; salva Dim
        SWAB    R4
        MOV     #1, R1
        CMP     R2, R3          ; Inizio = Fine ?
        BEQ     MVH1            ; salta se si`
        BGT     MVH2            ; salta se muove a sinistra
MVH3:   CMP     R2, R3          ; muove a destra, valuta fine
        BGE     MVH1            ; salta se corrente = Fine
        ADD     R4, R2          ; Col+Dim
        MOVB    #DCH, R0        ; carattere disco
```

```
        JSR     PC, MOVES
        SUB     R4, R2          ; Col
        MOVB    #' , R0         ; cancellazione
        JSR     PC, MOVES
        ADD     #ONEC, R2       ; Col+1
        BR      MVH3
MVH2:   CMP     R2, R3          ; muove a sinistra, valuta fine
        BLE     MVH1            ; salta se corrente = Fine
        SUB     #ONEC, R2       ; Col-1
        MOVB    #DCH, R0        ; carattere disco
        JSR     PC, MOVES
        ADD     R4, R2          ; Col-1+Dim
        MOVB    #' , R0         ; cancellazione
        JSR     PC, MOVES
        SUB     R4, R2          ; Col-1
        BR      MVH2
MVH1:   SWAB    R4              ; ripristina Dim
        MOV     R4, R1
        MOV     (SP)+, R4       ; ripristina registri
        MOV     (SP)+, R2
        MOV     (SP)+, R0
        RTS     PC

;-------------------------------------------------------------;
; NOME                                                        ;
;   INFORM  movimento elementare di un disco                  ;
; DESCRIZIONE                                                 ;
;   Effettua la movimentazione completa di un disco da una    ;
;   posizione ad un'altra. Il posizionamento di colonna e`    ;
;   quello del piolo, cioe` e` la posizione del primo         ;
;   elemento della meta` a destra del disco.                  ;
; INTERFACCIA                                                 ;
;   JSR PC, INFORM                                            ;
;                                                             ;
;   2(SP)   input   Col.Riga finale                           ;
;   4(SP)   input   Col.Riga iniziale                         ;
;   6(SP)   input   Tipo disco (dimensione)                   ;
; USA                                                         ;
;   Subr: MOVVER MOVHOR                                       ;
;   Regs: R1 R2 R3                                            ;
;   Memo: TOP                                                 ;
;-------------------------------------------------------------;
INFORM:
        MOV     6(SP), R1       ; Ord
        MOV     4(SP), R2       ; Col1.Riga1
        SWAB    R1
        SUB     R1, R2          ; Col1-Ord.Riga1
        SWAB    R1
        MOV     R2, R3          ; Col2.Riga2 = Col1-Ord.Riga1
        CLRB    R3
        BISB    TOP, R3         ; Col1-Ord.Top
        ASL     R1              ; Ord*2
        JSR     PC, MOVVER      ; movimento in su
        MOV     R3, R2          ; Col1-Ord.Top
        MOV     2(SP), R3       ; Col2.Riga2
        ASR     R1
        SWAB    R1
        SUB     R1, R3          ; Col2-Ord.Riga2
        SWAB    R1
        ASL     R1              ; 2*Ord
```

```
        CLRB    R3
        BISB    R2, R3          ; Col2-Ord.Top
        JSR     PC, MOVHOR      ; movimento laterale
        MOV     R3, R2          ; Col2-Ord.Top
        CLRB    R3
        BISB    2(SP), R3       ; Col2-Ord.Riga2
        JSR     PC, MOVVER      ; movimento in giu`
        MOV     0(SP), 6(SP)    ; rilascia parametri
        ADD     #6, SP
        RTS     PC
```

Un programma di prova utilizza la routine POLE che "carica" il primo piolo con il numero di dischi previsto inizialmente.

```
;-------------------------------------------------------------;
; NOME                                                        ;
;   POLE    inizializzazione di una torre di hanoi            ;
; DESCRIZIONE                                                 ;
;   Carica il primo piolo a sinistra del numero di dischi     ;
;   precisato.                                                ;
; INTERFACCIA                                                 ;
;   JSR PC, POLE                                              ;
;                                                             ;
;   R1      input   numero totale dischi                      ;
;   R2      input   Col.Riga primo piolo                      ;
; USA                                                         ;
;   Subr: INFORM                                              ;
;-------------------------------------------------------------;
POLE:
        MOV     R2, -(SP)       ; salva registri
        MOV     R4, -(SP)
        MOV     R1, -(SP)
        MOV     R1, R4          ; Corrente = Tot
POL1:   TST     R4              ; Corrente = 0 ?
        BEQ     POL2
        MOV     R2, -(SP)       ; salva Col.Riga
        MOV     R4, -(SP)       ; Ord = Corrente
        MOV     R2, -(SP)       ; Col1.Riga1
        MOV     R2, -(SP)       ; Col2.Riga2
        JSR     PC, INFORM
        MOV     (SP)+, R2       ; ripristina Col.Riga
        DEC     R2              ; Riga - 1
        DEC     R4              ; Corrente - 1
        BR      POL1
POL2:   MOV     (SP)+, R1       ; ripristina registri
        MOV     (SP)+, R4
        MOV     (SP)+, R2
        RTS     PC


;*************************************************************;
;
;       Programma di prova
PROVA:
        JSR     PC, CLS
        MOV     #2, TOP         ; cima dei pioli
        MOV     #6, R1          ; Tot
        MOV     #10.*256.+8.,R2 ; Col=10., Riga=8.
        JSR     PC, POLE
        MOV     R1, -(SP)       ; Ord
```

```
        MOV     R2, -(SP)       ; Col1.Riga1
        ADD     #20.*256., R2
        MOV     R2, -(SP)       ; Col2.Riga2
        ADD     #20.*256., R2
        MOV     R2, -(SP)       ; Col3.Riga3
        JSR     PC, HANMOV
        HALT

        .CSECT  DAT
TOP:    .WORD 0
```

Questa soluzione evidenzia come, pur essendo possibile l'accesso ai parametri di procedura mediante SP, esso risulti poco agevole, dovendosi tener conto dei valori che si possono via via accumulare sullo stack (variabili temporanee oppure parametri per una successiva chiamata). Si suggerisce di modificare le routine HANMOV e INFORM in modo che utilizzino un Frame Pointer.
++++++++
++++++++

I successivi temi vengono solo proposti, con un suggerimento per la soluzione.

Esercizio 4.18

Si progetti un metodo per la rappresentazione di matrici bidimensionali, quadrate e simmetriche (v[i,j] = v[j,i] 0<=i,j<=n-1) in modo da evitare la memorizzazione della meta` superiore della matrice, cioe` degli elementi v[i,j] con j>i. Si utilizzi allo scopo un metodo di accesso mediante indirezione, supponendo i valori memorizzati per righe. Si definisca una routine di inizializzazione delle tabelle necessarie e due routine che consentano di effettuare la somma e il prodotto tra coppie di matrici di elementi tipo word.

La matrice viene rappresentata da righe di lunghezza variabile da 1 a n. L'unica tabella di indirezione necessaria puo` contenere gli indirizzi iniziali delle righe. Occorre modificare la mappa di accesso nel modo seguente:

```
adr(v[i,j]) =   VEI[i]+j*2   j<=i
            =   VEI[j]+i*2   j>i
```

La somma risulta estremamente semplice.
++++++++

Esercizio 4.19

Si scriva un programma "debugger" che consenta di controllare la correttezza dell'esecuzione di altri programmi. Indicato con P il programma da controllare, il debugger riceve in apposite locazioni predefinite un puntatore alla prima istruzione di P e un puntatore ad una tabella contenente una lista (variabile) di indirizzi di istruzioni di P in corrispondenza delle quali l'esecuzione di P va arrestata (breakpoint). A fronte di un tale arresto, il debugger deve visualizzare il contenuto di tutti i registri del processore e

attendere da STDIN l'invio di un carattere, per riattivare il programma con l'esecuzione dell'istruzione in corrispondenza della quale P si era fermato.

Si possono adottare due tecniche:

a) sostituire l'istruzione nel punto di arresto con una istruzione di salto a un punto determinato del debugger (come si risolve il problema di salvare il valore di PC?);

b) analogo ad a) ma con una istruzione di TRAP.

In tutti e due i casi occorre salvare i registri utilizzati dal debugger per far successivamente ripartire P nelle stesse condizioni in cui si trovava all'arresto. Occorre inoltre osservare che l'istruzione di P che era stata sostituita dal breakpoint e salvata da qualche parte, non puo` a rigore essere ricaricata al suo posto se si vuole mantenere ancora attivo il breakpoint (si pensi ad esempio a istruzioni all'interno di cicli). Si deve quindi adottare una tecnica diversa, come ad esempio eseguire a parte l'istruzione (va bene in tutti casi?).
++++++++

Esercizio 4.20

Si scriva un programma che verifichi la funzionalita` della memoria centrale. Il programma, avente una struttura ciclica, deve ad ogni ciclo:

a) azzerare il contenuto di tutte le locazioni della memoria, escluse quelle comprese in un insieme finito di sezioni i cui indirizzi iniziali e finali vengono passati al programma in una tabella;

b) rileggere il contenuto delle locazioni modificate al punto a), inviando su STDOUT un messaggio d'errore quando venga rilevata una locazione diversa da 0;

c) ripetere i due passi precedenti con tutti i bit posti al valore 1;

d) depositare in ciascuna locazione, con gli stessi vincoli del punto a), l'indirizzo della locazione stessa;

e) rileggere le locazioni modificate in c) e verificarne la corrispondenza con il proprio indirizzo, segnalando su STDOUT eventuali errori.

Il programma deve in particolare preservare se stesso dall'essere modificato. Inoltre, ogni volta che perviene un carattere da STDIN, si deve visualizzare su STDOUT il numero di cicli eseguiti e il numero di errori rilevati.

Conviene disporre della routine TSTPUN che incrementa un puntatore alle locazioni da verificare, saltando le sezioni da proteggere. La visualizzazione del numero di cicli e di errori puo` essere effettuata da una routine di servizio che va a leggere due variabili di stato all'uopo predisposte e modificate dal programma principale.

E` opportuno che la visualizzazione dei messaggi di errore avvenga a interruzioni disabilitate, onde evitare interferenza.
++++++++

Esercizio 4.21

Si supponga di conoscere l'indirizzo iniziale di una subroutine generica che realizza una funzione booleana a 4 ingressi e 1 uscita. La subroutine riceve nei 4 bit meno significativi di R0 gli ingressi e restituisce nel flag C l'uscita. Si scriva una subroutine in grado di visualizzare su STDOUT la tabella di verita` della funzione sottoforma di mappa di Karnaugh, ricevendo in R1 l'indirizzo della funzione.

Poiche` la mappa di Karnaugh prevede che sulle righe e sulle colonne vi siano configurazioni "adiacenti", e` possibile utilizzare il generatore GRAY per ottenere la corretta sequenza. Il resto e` solo un problema di gestione di stringhe e conversioni.
++++++++

Esercizio 4.22

Si definiscano le subroutine COMPAC e DECOMP che provvedono a compattare e riespandere un'area di stringhe di caratteri a 7 bit. Il compattamento e` basato sulla regola che per ogni byte compattato, i bit (piu` significativi) rimasti liberi, sono disponibili a contenere lo stesso numero di bit (meno significativi) del byte non compattato successivo. Si scriva un programma principale che riceve da tastiera una o piu` stringhe, le compatta visualizzando la versione compattata in esadecimale e le riespande visualizzandole nella forma originaria.

Occorre tener conto che il minimo comun multiplo tra 7 e 8 e` 56, quindi il compattamento puo` procedere ciclicamente per gruppi di 8 caratteri compattati in 7 byte. Occorre sfruttare la funzione di shift, effettuata un numero variabile di volte in dipendenza della posizione nel gruppo di 8 del carattere da compattare.
++++++++

Esercizio 4.23

Si scriva un programma che esamini ripetutamente un'area di memoria centrale, conteggiando il numero di locazioni il cui contenuto ha almeno 5 bit adiacenti pari a 1. Il programma deve consentire di modificare in qualsiasi momento il contenuto di una locazione fornendo da STDIN una stringa del tipo:

xxxxxx = nnnnnn

ove xxxxxx e` un indirizzo in ottale e nnnnnn il valore in ottale da inserire nella locazione di indirizzo xxxxxx. Se invece si batte il solo carattere '?, si ottiene su STDOUT in decimale il numero di volte che l'area e` stata interamente esaminata e il totale, relativo all'ultimo esame effettuato, di locazioni verificanti la condizione sui 5 bit.

Definito un segmento di programma o, meglio, una routine che stabilisca se una locazione soddisfa alla condizione posta, si tratta di definire una routine di servizio per STDIN che possa sia gestire l'acquisizione della stringa (con eco e possibilita` di correzione) sia quella del carattere '? e attivare le relative funzioni associate. L'esame della memoria e` lasciato al programma principale, che non deve arrestarsi durante l'acquisizione della stringa.
++++++++

Esercizio 4.24

Si convenga di definire come notazione BCN (Binary Coded N-base number) quella in cui un numero senza segno, espresso in base N qualsiasi compresa tra 2 e 127, sia rappresentato da una sequenza di "cifre" ciascuna compresa tra 0 e N-1. Ogni cifra e` a sua volta rappresentata in memoria mediante un byte e, limitatamente al caso N <= 36, puo` essere visualizzata mediante i caratteri ASCII 0..9 A..Z ordinatamente. Si convenga inoltre che un numero BCN sia sinteticamente rappresentato da due word: il primo (BCNA) e` l'indirizzo iniziale dell'area di memoria riservata a memorizzare le cifre, partendo da quella meno significativa; il secondo word ha come byte meno significativo (BCNB) la rappresentazione binaria della base e come byte piu` significativo (BCNC) la lunghezza (1..127) in numero di cifre del numero BCN. Le cifre virtualmente oltre la lunghezza sono considerate comunque nulle (fig. 4.4). Si definisca un insieme di subroutine per la gestione dei numeri BCN.

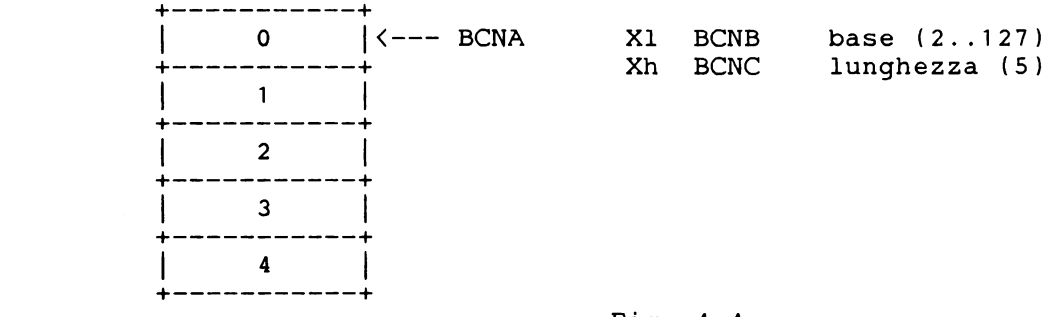


Fig. 4.4

La convenzione suggerita permette in pratica di memorizzare numeri interi senza segno con un numero variabile e anche elevato di cifre e con base qualsiasi ancorche` limitata. Poiche` le singole cifre sono rappresentate in forma binaria, risultano abbastanza agevoli conversioni ad altra rappresentazione ed elaborazioni aritmetiche e logiche.

Si possono realizzare le seguenti routine:

BCC2AS Converte una cifra BCN a carattere ASCII 0..9, A..Z
AS2BCC Converte un carattere ASCII 0..9, A..Z in cifra BCN
BCNTSB Verifica se base nel campo 1..127.
BCN2BC Separazione tra base e lunghezza
BCNTSV Verifica correttezza della rappresentazione rispetto alla base

BCNTS0 Valuta se il numero e` = 0
BCNRED Aggiorna la lunghezza trascurando zeri non significativi
BCNCPY Effettua una copia di un numero in un'altra area predisposta
BCNEQV Verifica se due numeri sono confrontabili (stessa base)
BCNCMP Confronto tra due numeri
BCNADD Somma due numeri
BCNSUB Sottrazione due numeri
BCNRSH Shift a destra di una cifra
BCNLSH Shift a sinistra di una cifra
BINBCN Conversione binario 16 bit in BCN
BCNBIN Conversione BCN in binario 16 bit
BIEBCN Conversione binario esteso in BCN
BCNBIE Conversione BCN in binario esteso

La notazione binario esteso e` quella binaria su piu` word.
++++++++

Esercizio 4.25

Supponendo di rappresentare con un word un numero frazionario senza segno x con $0<=x<=1-2^{-16}$ nel modo seguente:

$$x = b_{15}*2^{-1} + b_{14}*2^{-2} +...+b_0*2^{-16}$$

si definiscano le routine FRDEC e DECFR in grado di convertire un numero nella equivalente stringa ASCII decimale .dd...d. con un massimo di 16 cifre decimali frazionarie e viceversa.

Per la routine FRDEC occorre esprimere x in base 10:

$$x = d_{-1}*10^{-1} + d_{-2}*10^{-2} +...+d_{-16}*10^{-16}$$

Si puo` facilmente verificare che un numero binario frazionario di 16 bit ha un equivalente decimale di al piu` 16 cifre significative. Si hanno le seguenti relazioni:

$$int(x*10)=d_{-1}$$
$$r_{-1}=x*10-d_{-1}=d_{-2}*10^{-1}+...+ d_{-16}*10^{-15}$$
$$int(r_{-1}*10)=d_{-2}$$
$$r_{-2}=r_{-1}*10-d_{-2}=d_{-3}*10^{-1}+...+ d_{-16}*10^{-14}$$
$$...$$
$$int(r_{-k}*10)=d_{-k-1}$$
$$r_{-k-1}=r_{-k}*10-d_{-k-1}$$
$$...$$
$$int(r_{-15}*10)=d_{-16}$$
$$r_{-16}=0$$

Per DECFR si possono fare considerazioni del tutto analoghe, solo che le moltiplicazioni vanno eseguite sul numero rappresentato in decimale per la nuova base 2. Le cifre corrispondenti alle d_{-k} delle precedenti espressioni, sono in questo caso semplici riporti unitari dopo una operazione di scorrimento esteso verso sinistra (in senso equivalente). Di conseguenza, risulta conveniente dapprima convertire il numero ASCII decimale in BCN (vedi esercizio precedente) e definire la routine BCNX2 che effettua la moltiplicazione per 2 di un numero BCN, restituendo l'eventuale riporto dalla cifra piu` significativa.
++++++++

APPENDICE

Lista istruzioni in ordine di codice operativo

Opcode	Simbolo	Opcode	Simbolo
000000	HALT	01SSDD	MOV
000001	WAIT	02SSDD	CMP
000002	RTI	03SSDD	BIT
000003	BPT	04SSDD	BIC
000004	IOT	05SSDD	BIS
000005	RESET	06SSDD	ADD
000006	RTT	070RDD	MUL
0001DD	JMP	071RDD	DIV
00020R	RTS	074RDD	XOR
000240	NOP	077RNN	SOB
000241	CLC	1000XXX	BPL
000242	CLV	1004XXX	BMI
000244	CLZ	1010XXX	BHI
000250	CLN	1014XXX	BLOS
000257	CCC	1020XXX	BVC
000261	SEC	1024XXX	BVS
000262	SEV	1030XXX	BCC
000264	SEZ	1030XXX	BHIS
000270	SEN	1034XXX	BCS
000277	SCC	1034XXX	BLO
0003DD	SWAB	104NNN	EMT
0004XXX	BR	104NNN	TRAP
0010XXX	BNE	1050DD	CLRB
0014XXX	BEQ	1051DD	COMB
0020XXX	BGE	1052DD	INCB
0024XXX	BLT	1053DD	DECB
0030XXX	BGT	1054DD	NEGB
0034XXX	BLE	1055DD	ADCB
004RDD	JSR	1056DD	SBCB
0050DD	CLR	1057DD	TSTB
0051DD	COM	1060DD	RORB
0052DD	INC	1061DD	ROLB
0053DD	DEC	1062DD	ASRB
0054DD	NEG	1063DD	ASLB
0055DD	ADC	1064DD	MTPS
0056DD	SBC	1067DD	MFPS
0057DD	TST	11SSDD	MOVB
0060DD	ROR	12SSDD	CMPB
0061DD	ROL	13SSDD	BITB
0062DD	ASR	14SSDD	BICB
0063DD	ASL	15SSDD	BISB
0067DD	SXT	16SSDD	SUB

Modalita` di indirizzamento

n	Modalita`	Simbolo	Commento	|M |o |d !R |e |g |
0	Registro	Rn	(Rn) e` l'operando	
1	Registro Diff.	(Rn) o @Rn	M[Rn] e` l'operando	
2	Autoincremento	(Rn)+	M[Rn] e` l'operando; Rn <- Rn+(1:2)	
3	Autoinc. Diff.	@(Rn)+	M[M[Rn]] e` l'operando; Rn <- Rn+2	
4	Autodecremento	-(Rn)	Rn <- Rn-(1:2); M[Rn] e` l'operando	
5	Autodecremento	@-(Rn)	Rn <- Rn-2; M[M[Rn]] e` l'operando	
6	Indice	X(Rn)	M[(Rn)+X] e` l'operando	
7	Indice Diff.	@X(Rn)	M[M[(Rn)+X]] e` l'operando	

Modalita` di indirizzamento mediante PC

n	Modalita`	Simbolo	Commento	|M |o |d !1 |1 |1 |
2	Immediato	#ALFA	l'operando ALFA segue l'istruzione	
3	Assoluto	@#ALFA	M[ALFA] e` l'operando; ALFA segue	
6	Relativo	ALFA	M[ALFA] e` l'operando; ALFA-ind.istr.-4 segue	
7	Relativo Diff.	@ALFA	M[M[ALFA]] e` l'operando; ALFA-ind.istr.-4 segue	

Legenda

Opcode

B	0 per word, 1 per byte
SS	campo sorgente (6 bit)
DD	campo destinazione (6 bit)
R	registro (3 bit)
XXX	offset (8 bit con segno)
N	numero (3 bit)
NN	numero (6 bit)

Operazioni

()	contenuto di
s	contenuto sorgente
d	contenuto destinazione
r	contenuto registro
<-	assegnazione
X	indirizzo relativo

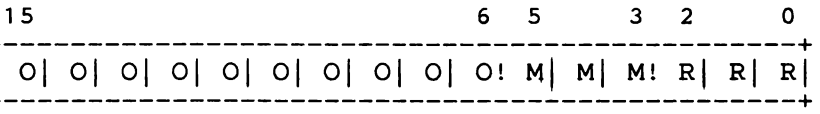
Operazioni booleane

&	AND
|	OR
#	OR esclusivo
~	NOT

Codici di condizione

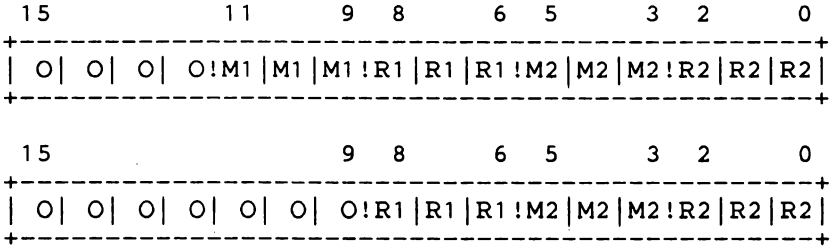
*	impostato su condizione
-	non modificato
0	impostato a 0
1	impostato a 1

Istruzioni ad un operando: OPR dst



Opcode	Simbolo	Commento	Dst	N	Z	V	C
Generali							
B050DD	CLR(B)	azzeramento	0	0	1	0	0
B051DD	COM(B)	complemento a 1	~d	*	*	0	1
B052DD	INC(B)	incremento	d+1	*	*	*	–
B053DD	DEC(B)	decremento	d-1	*	*	*	–
B054DD	NEG(B)	complemento a 2	-d	*	*	*	*
B057DD	TST(B)	Valuta contenuto	d	*	*	0	0
Rotazioni e scorrimenti							
B060DD	ROR(B)	rotazione destra	->C, d	*	*	*	*
B061DD	ROL(B)	rotazione sinistra	C, d <-	*	*	*	*
B062DD	ASR(B)	aritm. a destra	d/2	*	*	*	*
B063DD	ASL(B)	aritm. a sinistra	2d	*	*	*	*
0003DD	SWAB	scambio byte		*	*	0	0
Precisione multipla							
B055DD	ADC(B)	somma flag C	C+d	*	*	*	*
B056DD	SBC(B)	sottrazione flag C	C-d	*	*	*	*
0067DD	SXT	estensione segno	0 o -1	–	*	0	–
Speciali							
1067DD	MFPS	Copia da PSW	d <- PSW	*	*	0	–
1064DD	MTPS	Copia a PSW	PS <- s	*	*	*	*

Istruzioni a due operandi:
OPR src, dst OPR src, R OPR R, dst



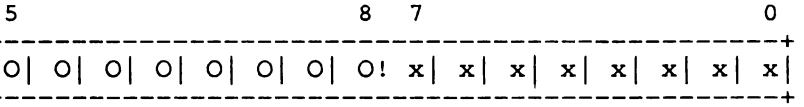
Opcode	Simbolo	Commento	Operazione	N	Z	V	C
Generali							
B1SSDD	MOV(B)	copia	d <- s	*	*	0	–
B2SSDD	CMP(B)	comparazione	s-d	*	*	*	*
06SSDD	ADD	somma	d <- s+d	*	*	*	*
16SSDD	SUB	sottrazione	d <- d-s	*	*	*	*
Logiche							
B3SSDD	BIT(B)	bit test (AND)	s&d	*	*	0	–
B4SSDD	BIC(B)	bit azzeramento	d <- (~s)&d	*	*	0	–
B5SSDD	BIS(B)	bit set (OR)	d <- s|d	*	*	0	–

Registro

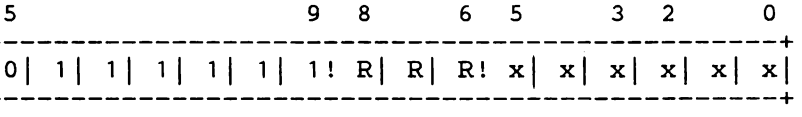
074RDD	XOR	or esclusivo	d <- r#d	* * 0 -
070RDD	MUL	moltiplicazione	r <- r*s	* * 0 *
071RDD	DIV	divisione	r <- r/s	* * * *

Istruzioni di salto

Salto relativo



Istruz. SOB



Opcode	Simbolo	Commento	Oper. (Condizione)
Salto limitato (relativo) incondizionato			
0004XXX	BR	Salto incondiz.	
Salti limitati (relativi) condizionati			
		Se condiz. verificata	PC <- PC+2*Offset
Su singola condizione			
0010XXX	BNE	non eguale (a 0)	Z = 0
0014XXX	BEQ	eguale (a 0)	Z = 1
1000XXX	BPL	positivo	N = 0
1004XXX	BMI	negativo	N = 1
1020XXX	BVC	non overflow	V = 0
1024XXX	BVS	overflow	V = 1
1030XXX	BCC	non carry	C = 0
1034XXX	BCS	carry	C = 1
Su condizione per numeri con segno			
0020XXX	BGE	maggiore o eguale (a 0)	N # V = 0
0024XXX	BLT	minore (di 0)	N # V = 1
0030XXX	BGT	maggiore (di 0)	Z | (N # V) = 0
0034XXX	BLE	minore o eguale (a 0)	Z | (N # V) = 1
Su condizione per numeri senza segno			
1010XXX	BHI	maggiore	C | Z = 0
1014XXX	BLOS	minore o eguale	C | Z = 1
1030XXX	BHIS	maggiore o eguale	C = 0
1034XXX	BLO	minore	C = 1
Su conteggio			
077RNN	SOB	salta se val. decr.<>0	R <- R-1; se R<>0 PC <- PC - 2*NN
Salti estesi			
0001DD	JMP	salto incondiz.	PC <- dst
004RDD	JSR	salto a subroutine	| usano stesso R
00020R	RTS	ritorno da subroutine	|

Opcode	Simbolo	Commento
Miscellanea		
000000	HALT	arresta il processore
000001	WAIT	sospende il processore in attesa di interr.
000005	RESET	inizializza il sistema
000240	NOP	operazione nulla

Opcode	Simbolo	Vector Address	Commento
Trap			
000002	RTI		ritorno da interruzione
000006	RTT		ritorno da interruz. con inibiz. T
000003	BPT	14	Usata come breakpoint
000004	IOT	20	Usata per routine I/O
104NNN	EMT	30	Usata per attivare un emulatore
104NNN	TRAP	34	Interruzione sincrona generica

Operazioni sui bit di condizione

```
          15                      8  7                       0
         +-----------------------------------------------------+
PSW      |//|//|//|//|//|//|//|//|Pr|io|rt|T |N |Z |V |C |
         +-----------------------------------------------------+

          15                               5  4  3  2  1  0
         +-----------------------------------------------------+
OPR      | 0| 0| 0| 0| 0| 0| 0| 0| 1| 0| 1|SC| N| Z| V| C|
         +-----------------------------------------------------+
```

Opcode	Simbolo	Commento
000241	CLC	C <- 0
000242	CLV	V <- 0
000244	CLZ	Z <- 0
000250	CLN	N <- 0
000257	CCC	NZVC <- 0000
000261	SEC	C <- 1
000262	SEV	V <- 1
000264	SEZ	Z <- 1
000270	SEN	N <- 1
000277	SCC	NZVC <- 1111

Vettori di interruzione

Ind.	Commento
000	riservato
004	time out e altri errori
010	operaz. illegale e istr. riservate
014	istruz. BPT
020	istruz. IOT
024	power fail
030	istruz. EMT
034	istruz. TRAP
060	STDIN (priorita` 4)
064	STDOUT (priorita` 4)
104	RTC (priorita` 6)

Registri di I/O

Ind.	Commento	
177444	CLKSR	RTC stato
177560	ARCSR	STDIN stato
177562	ARBUF	STDIN buffer
177564	AXCSR	STDOUT stato
177566	AXBUF	STDOUT buffer

STDIN registro di stato

bit	Simbolo	Accesso	Commento
6	RCV IE	R/W	abilitazione interruz. dispositivo
7	RCV DONE	RO	dato disponibile
11	RCV ACT	RO	ricezione in corso

STDIN registro dati

bit	Simbolo	Accesso	Commento
0..7	RCV DATA	RO	dato ricevuto (8 bit)
13	FR ERR	RO	errore frame
14	OR ERR	RO	errore overrun
15	ERR	RO	errore

STDOUT registro di stato

bit	Simbolo	Accesso	Commento
2	MAINT	R/W	by-pass ingresso-uscita
6	XMIT IE	R/W	abilitazione interruz. dispositivo
7	XMIT RDY	RO	buffer vuoto

STDOUT registro dati

bit	Simbolo	Accesso	Commento
0..7	XMIT DATA	R/W	dato in uscita (8 bit)

RTC registro stato

CLKIE = 7 abilitazione interruz. dispositivo
CLKID = 10 disabilitazione interruz. dispos. e azzeramento

Potenze di 2

n	2^n	n	2^n
0	1	17	131072
1	2	18	262144
2	4	19	524288
3	8	20	1048576
4	16	21	2097152
5	32	22	4194304
6	64	23	8388608
7	128	24	16777216
8	256	25	33554432
9	512	26	67108864
10	1024	27	134217728
11	2048	28	268435456
12	4096	29	536870912
13	8192	30	1073741824
14	16384	31	2147483648
15	32768	32	4294967296
16	65536		

Tabella codici ASCII

Ott	Dec	Simbolo	Ott	Dec	Simbolo	Ott	Dec	Simbolo	Ott	Dec	Simbolo
0	0	NUL	40	32	SP	100	64	@	140	96	`
1	1	SOH	41	33	!	101	65	A	141	97	a
2	2	STX	42	34	"	102	66	B	142	98	b
3	3	ETX	43	35	#	103	67	C	143	99	c
4	4	EOT	44	36	$	104	68	D	144	100	d
5	5	ENQ	45	37	%	105	69	E	145	101	e
6	6	ACK	46	38	&	106	70	F	146	102	f
7	7	BEL	47	39	'	107	71	G	147	103	g
10	8	BS	50	40	(	110	72	H	150	104	h
11	9	HT	51	41	)	111	73	I	151	105	i
12	10	LF	52	42	*	112	74	J	152	106	j
13	11	VT	53	43	+	113	75	K	153	107	k
14	12	FF	54	44	,	114	76	L	154	108	l
15	13	CR	55	45	-	115	77	M	155	109	m
16	14	SO	56	46	.	116	78	N	156	110	n
17	15	SI	57	47	/	117	79	O	157	111	o
20	16	DLE	60	48	0	120	80	P	160	112	p
21	17	DC1	61	49	1	121	81	Q	161	113	q
22	18	DC2	62	50	2	122	82	R	162	114	r
23	19	DC3	63	51	3	123	83	S	163	115	s
24	20	DC4	64	52	4	124	84	T	164	116	t
25	21	NAK	65	53	5	125	85	U	165	117	u
26	22	SYN	66	54	6	126	86	V	166	118	v
27	23	ETB	67	55	7	127	87	W	167	119	w
30	24	CAN	70	56	8	130	88	X	170	120	x
31	25	EM	71	57	9	131	89	Y	171	121	y
32	26	SUB	72	58	:	132	90	Z	172	122	z
33	27	ESC	73	59	;	133	91	[	173	123	{
34	28	FS	74	60	<	134	92	\	174	124	|
35	29	GS	75	61	=	135	93	]	175	125	}
36	30	RS	76	62	>	136	94	^	176	126	~
37	31	US	77	63	?	137	95	_	177	127	DEL

INDICE ALFABETICO DELLE SUBROUTINE

Litografia LA PHOTOGRAPH
Tel. 28.292 PADOVA

esercizi di PROGRAMMAZIONE IN LINGUAGGIO ASSEMBLY

M. Moro