esercizio n. 5 – linguaggio macchina

prima parte – codifica in linguaggio macchina

Si deve tradurre in linguaggio macchina simbolico (linguaggio assemblatore) 68000 il programma (main e funzione *funz*) riportato qui sotto. Nel tradurre non si tenti di accorpare od ottimizzare insieme istruzioni C indipendenti.

La memoria ha parole da **16 bit**, indirizzi da **32 bit** ed è indirizzabile per byte. Le variabili intere sono da **16 bit**. Ulteriori specifiche al problema e le convenzioni da adottare nella traduzione sono le seguenti:

- i parametri di tutte le funzioni sono passati sulla pila in ordine inverso di elencazione
- i valori restituiti dalle funzioni ai chiamanti rispettivi sono passati sulla pila, sovrascrivendo il primo dei parametri passati o nello spazio libero opportunamente lasciato
- le variabili locali vengono impilate in ordine di elencazione
- le funzioni (tranne *main*) devono sempre salvare i registri che utilizzano

Si chiede di svolgere i **tre** punti seguenti:

1. **Si descriva** la struttura della memoria delle variabili globali e l'area di attivazione della funzione *funz*, secondo il modello 68000, nelle tabelle predisposte, indicando gli spiazzamenti rispetto a FP.
2. **Si dichiarino** in linguaggio macchina 68000 le variabili globali e **si scriva** il codice macchina della funzione *main*, coerentemente con le specifiche e le risposte precedenti, usando le tabelle predisposte.
3. **Si scriva** in linguaggio macchina 68000 il codice macchina della funzione *funz*, coerentemente con le specifiche e le risposte ai punti precedenti, usando le tabelle predisposte.

programma in linguaggio C

```
#define N = 5

/* variabili globali */

int dati [N];
int i;
int r = 0;

/* programma main (alcune parti sono volutamente omesse) */

void main ( ) {
   i = N - 1;
   do {
      if (dati [i] < dati [i - 1]) {
         r = r + funz (dati [i], i);
      }  /* fine if */
      i--;
   while (i >= 1);
   }  /* fine do */
} /* fine main */

/* funzione funz */

int funz (int a, int b) {
   return (a + r) * b;
} /* fine funzione */
```

domanda **1** (numero di righe non significativo)

	memoria variabili globali (domanda **1**)	
ind min	*dati [0]*	*- 2 byte*
	dati [1]	*- 2 byte*
	dati [2]	*- 2 byte*
	dati [3]	*- 2 byte*
	dati [4]	*- 2 byte*
	I	*- 2 byte*
	R	*- 2 byte*
ind max		

	area di attivazione di ***funz*** (domanda **1**)	
	registri salvati	
0	*FP precedente*	*- 4 byte*
+ 4	*indirizzo di rientro*	*- 4 byte*
+ 8	*A*	*- 2 byte*
+ 10	*B*	*- 2 byte*

dichiarazione delle **variabili globali** (domanda **2** – num. righe non signif.)

```
DATA:   ORG     1000            // indir. virt. segmento dati statici
N:      EQU     5               // costante N
DATI:   DS.W    N               // vettore DATI
I:      DS.W    1               // variabile I
R:      DC.W    0               // variabile R inizializzata a 0
```

codice 68000 di ***main*** (domanda **2** – num. righe non signif.) – *UN PO' OTTIMIZZATO*			
MAIN:	**LINK**	FP, #-0	// collega
	MOVE.W	#N, D0	// carica N
	SUBI.W	#1, D0	// calcola N - 1
	MOVE.W	D0, I	// memorizza I
	MOVEA.L	#DATI, A0	// inizializza A0
	MULS	#2, D0	// allinea indice (sizeof(int) = 2)
	ADDA.L	D0, A0	// calcola indir. ultimo elem. DATI
LOOP:	**MOVE.W**	(A0), D1	// carica DATI [I]
	SUBA.L	#2, A0	// aggiorna A0 (con allineamento)
	MOVE.W	(A0), D2	// carica DATI [I - 1]
	CMP.W	D2, D1	// confronta
	BGE	ENDIF	// se >= 0 va' a ENDIF
	MOVE.W	I, D0	// (ri)carica I
	MOVE.W	D0, -(SP)	// impila 2° param. di funz
	MOVE.W	D1,-(SP)	// impila 1° param. di funz
	BSR	FUNZ	// chiama funz
	ADDA.L	#2, SP	// abbandona 1° param. di funz
	MOVE.W	(SP)+, D3	// spila valusc di funz
	MOVE.W	R, D4	// carica R
	ADD.W	D3, D4	// addiziona
	MOVE.W	D4, R	// memorizza R
ENDIF:	**SUBI.W**	#1, D0	// calcola I - 1
	MOVE.W	D0, I	// memorizza I
	CMPI.W	#1, D0	// confronta
	BGE	LOOP	// se >= 0 va' a LOOP
	UNLK	FP	// scollega
	RTS		// rientra
	END	MAIN	// fine main

Per accedere al vettore, si ne carica l'indirizzo base in A0, poi si allinea l'indice I (gli elementi hanno taglia di due byte) e lo si addiziona ad A0; l'accesso usa l'indirizzamento indiretto da registro. Esistono soluzioni diverse. Il codice è un po' ottimizzato: si evita di ricaricare la variabile I e l'elemento DATI[I], se sono già nei registri D0 e D1 . Si può ottimizzare di più, sfruttando l'ortogonalità di alcune istruzioni.

codice in linguaggio macchina 68000 della funzione *f*unz (domanda **3**)			
FUNZ:	**LINK**	FP, #0	// area di attivazione di funz
B:	**EQU**	+10	// spiazzamento param b
A:	**EQU**	+8	// spiazzamento param a
	MOVEM.W	D0-D2, -(SP)	// salva registri
	MOVE.W	A(FP), D0	// carica param a
	MOVE.W	R, D1	// carica varglob r
	ADD.W	D0, D1	// addiziona
	MOVE.W	B(FP), D2	// carica param b
	MULS	D1, D2	// moltiplica
	MOVE.W	D2, B(FP)	// impila valusc
	MOVEM.W	(SP)+, D0-D2	// ripristina registri
	UNLK	FP	// scollega
	RTS		// rientra

seconda parte – assemblaggio e collegamento

Si supponga che la memoria abbia parole da **16 bit** e sia indirizzata per **byte**. Si svolgano i punti seguenti:

a) Nella tabella sotto, **si riportino** gli indirizzi dove collocare dati e istruzioni. Si consideri che ogni istruzione ha sempre una parola di codice operativo e, se serve, una o più parole aggiuntive. Si tenga conto che lo spiazzamento sia in istruzioni che manipolano dati sia in istruzioni di salto è sempre da **16 bit** e che indirizzi e costanti sono **corti** (16 bit) o **lunghi** (32 bit) come specifica l'istruzione.

			# parole	# byte	indirizzo (in decimale)
	ORG	5000			
ROW:	**EQU**	2			
COL:	**EQU**	3			
MAT:	**DS.L**	6	*12*	*24*	5000
SUM:	**DC.W**	0	*1*	*2*	*5024*
INIZ:	**MOVEA.L**	#MAT.L, A0	*3*	*6*	*5026*
	CLR.L	D1	*1*	*2*	*5032*
OUTER:	**CMPI.L**	#ROW.L, D1	*3*	*6*	*5034*
	BGE	FINISH	*2*	*4*	*5040*
	CLR.W	D2	*1*	*2*	*5044*
INNER:	**ADD.W**	(A0, D2), SUM.L	*3*	*6*	*5046*
	ADDI.W	#1.W, D2	*2*	*4*	*5052*
	CMPI.W	#COL.W, D2	*2*	*4*	*5056*
	BLT	INNER	*2*	*4*	*5060*
	ADDI.L	#ROW.L, A0	*3*	*6*	*5064*
	ADDI.L	#1.L, D1	*3*	*6*	*5070*
	BRA	OUTER	*2*	*4*	*5076*
FINISH:	**END**	INIZ			*5080*

b) Nella tabella data sotto, **si riportino** i simboli e i rispettivi valori numerici.

Tabella dei simboli	
nome simbolo	*valore numerico (in decimale)*
ROW	*2*
COL	*3*
MAT	*5000*
SUM	*5024*
INIZ	*5026*
OUTER	*5034*
INNER	*5046*
FINISH	*5080*

c) **Si dica** quanto vale in decimale lo spiazzamento codificato nell'istruzione BGE:

spiazzamento in BGE = *5080 – 5044 = 36*

d) **Si dica** quanto vale in decimale lo spiazzamento codificato nell'istruzione BLT:

spiazzamento in BLT = *5046 – 5064 = –18*

e) **Si dica** quale valore (decimale) verrà caricato nel registro PC al lancio del programma: 5026