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 **32 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

```
/* costanti */
#define  N  6
/* variabili globali */
int x = N;
int y;
int *p;
/* funzione funz */
int funz (int a, int * q) {
   int b;
   if (*q == a) {
      return a;
   } else {
      b = funz (a - 1, q);
      return b;
   } /* if */
} /* fine funz */
/* programma main */
void main ( ) {
   p = &x;
   y = funz (N, p);
} /* fine main */
```

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

	memoria variabili globali (domanda **1**)	
ind min	*X*	*- 4 byte*
	Y	*- 4 byte*
	P	*- 4 byte*
ind max		

	area di attivazione di ***funz*** (domanda **1**)	
	registri salvati	
– 4	*B*	*- 4 byte*
0	*FP precedente*	*- 4 byte*
+ 4	*indirizzo di rientro*	*- 4 byte*
+ 8	*A*	*- 4 byte*
+ 12	*Q*	*- 4 byte*

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

```
DATA:   ORG     2000        // indir. virt. segmento dati statici
N:      EQU     6           // costante N
X:      DC.L    N           // variabile intera X inizializzata a N
Y:      DS.L    1           // variabile intera Y
P:      DS.L    1           // variabile puntatore P
```

codice 68000 di ***main*** (domanda **2** – num. righe non signif.) – *NON OTTIMIZZATO*			
MAIN:	**LINK**	FP, #-0	// collega
	MOVEA.L	#X, A0	// inizializza A0
	MOVE.L	A0, P	// memorizza p
	MOVEA.L	P, A0	// (ri)carica p
	MOVE.L	A0, -(SP)	// impila 2o param di funz
	MOVEI.L	#N, D0	// inizializza D0
	MOVE.L	D0, -(SP)	// impila 1o param di funz
	BSR	FUNZ	// chiama funz
	ADDA.L	#4, SP	// abbandona 2o param di funz
	MOVE.L	(SP)+, D0	// spila valusc di funz
	MOVE.L	D0, Y	// memorizza y
	UNLK	FP	// scollega
	RTS		// rientra
	END	MAIN	// dichiara indirizzo iniziale

Sono possibili varie ottimizzazioni, evitando di ricaricare variabili (ossia tenendole nei registri) e sfruttando l'ortogonalità di MOVE.

codice 68000 di ***funz*** (domanda **3**) – *NON OTTIMIZZATO*			
FUNZ:	**LINK**	FP, #-4	// area di attivazione di funz
A:	**EQU**	+12	// spiazzamento param a
Q:	**EQU**	+8	// spiazzamento param q
B:	**EQU**	-4	// spiazzamento varloc b
	MOVEM.W	D1-D3/A0, -(SP)	// salva registri
	MOVEA.L	Q(FP), A0	// carica q
	MOVE.L	(A0), D1	// carica *q
	MOVE.L	A(FP), D2	// carica a
	CMP.L	D1, D2	// confronta
	BNE	ELSE	// se != 0 va' a ELSE
THEN:	**MOVE.L**	A(FP), D2	// (ri)carica a
	MOVE.L	D2, Q(FP)	// sovrascrivi valusc
	BRA	ENDIF	// va' a ENDIF
ELSE:	**MOVEA.L**	Q(FP), A0	// (ri)carica q
	MOVE.L	A0, (SP)-	// impila 2° param di funz
	MOVE.L	A(FP), D2	// (ri)carica a
	SUBI.L	#1, D2	// calcola a - 1
	MOVE.L	D2, (SP)-	// impila 1° param di funz
	BSR	FUNZ	// chiama (ricorsivamente) funz
	ADDA.L	#4, SP	// abbandona 1° param di funz
	MOVE.L	+(SP), D3	// spila valusc di funz
	MOVE.L	D3, B(FP)	// memorizza b
	MOVE.L	B(FP), D3	// (ri)carica b
	MOVE.L	D3, Q(FP)	// sovrascrivi valusc
ENDIF:	**MOVEM.W**	+(SP), D1-D3/A0	// ripristina registri
	UNLK	FP	// scollega
	RTS		// rientra

Sono possibili varie ottimizzazioni, evitando di ricaricare variabili (ossia tenendole nei registri) e sfruttando l'ortogonalità di MOVE.

seconda parte – assemblaggio e collegamento

Si consideri la codifica della funzione **F1** sotto riportata che viene assemblata come un modulo a sé stante e 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 noti che la riga della prima istruzione di F1 è già stata compilata e i valori inseriti devono essere tenuti in conto per calcolare l'indirizzo successivo. 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			
F1:	**LINK**	FP, #-2.W	**2**	**4**	**5000**
X:	**EQU**	+8			
Y:	**EQU**	+3			
RES:	**EQU**	-2			
	MOVEM	D0-D1,-(SP)	*1*	*2*	*5004*
	MOVE	#0.W, RES(FP)	*3*	*6*	*5006*
WHILE:	**MOVE**	X(FP), D0	*2*	*4*	*5012*
	CMPI	#0.W, D1	*2*	*4*	*5016*
	BLT	END_WHILE	*2*	*4*	*5020*
	MOVE	Y(FP), D1	*2*	*4*	*5024*
	SUB	D0, D1	*1*	*2*	*5028*
	MOVE	D1, -(SP)	*1*	*2*	*5030*
	BSR	F2	*2*	*4*	*5032*
	MOVE	(SP)+, RES(FP)	*2*	*4*	*5036*
	ADDI	#-1.W, D0	*2*	*4*	*5040*
	MOVE	D0, X(FP)	*2*	*4*	*5044*
	BRA	WHILE	*2*	*4*	*5048*
END_WHILE:	**MOVE**	RES(FP), Y(FP)	*3*	*6*	*5052*
	UNLINK	FP	*1*	*2*	*5058*
	RTS		*1*	*2*	*5060*

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

Tabella dei simboli	
nome simbolo	*valore numerico (in decimale)*
F1	*5000*
X	*8*
Y	*3*
RES	*-2*
WHILE	*5012*
END_WHILE	*5052*
F2	*simbolo esterno il cui valore non può essere calcolato durante l'assemblaggio del modulo in esame*

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

spiazzamento in BRA = *5012 – 5052 = –40*

d) **Si dica** quale valore (decimale) verrà caricato nel registro PC all'attivazione della funzione F1: 5000

e) **Si dica** qual è il valore (decimale) dell'ultimo indirizzo di memoria occupato dalla funzione F1: 5061