esercizio n. 2 – linguaggio macchina

Si deve tradurre in linguaggio macchina simbolico (linguaggio assemblatore) 68000 il programma C riportato sotto, costituito da programma principale *main* e dalle funzioni *funz* e *rec*. Non si tenti di accorpare od ottimizzare insieme istruzioni C indipendenti. La memoria ha indirizzi da **32 bit**, è indirizzabile per byte e gli interi sono da **32 bit**. Si facciano le ipotesi seguenti:

- i parametri delle 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 parametro passato o nello spazio libero lasciato appositamente
- le variabili locali sono impilate in ordine di elencazione
- le funzioni salvano sempre i registri che utilizzano (tranne *main*)

Si chiede di svolgere i tre punti seguenti:

1. **Si descriva** l'area di attivazione della funzione *rec* e quella della funzione *funz*, secondo il modello 68000, nelle due tabelle predisposte.
2. **Si traduca** in linguaggio macchina 68000 il codice di *rec* e quello di *main*, coerentemente con le specifiche e le risposte ai punti precedenti, usando le due tabelle predisposte.
3. **Si immagini** di simulare il funzionamento del programma, con l'ipotesi **MAX = 1**, fino a quando la funzione *funz* ha appena messo il valore di uscita nella sua area di attivazione. **Si indichi** lo stato della pila, e precisamente il contenuto delle aree di attivazione delle funzioni *rec* e *funz* riferite a tale punto di esecuzione, riportando sia il contenuto simbolico delle celle della pila sia quello numerico (se precisabile), nella tabella predisposta.

```
#define MAX  = 3

/* variabili globali */
int a = 5;
int b = 10;
int h;

/* funzioni */
int funz (int p, int q) {
   int k;  /* variabile locale */
   k = p * q;
   return k - q;
} /* funz */

int rec (int n, int x, int y) {
   if (n == 0) {
      return x + y;
   } else {
      /* chiamata ricorsiva */
      return funz (x - y, rec (n - 1, x, y));
   } /* if */
} /* rec */

/* programma principale */
void main ( ) {
   h = rec (MAX, a, b);
} /* main */
```

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

area di attivazione di ***rec*** (domanda **1**)	
registri salvati	
FP precedente	*- 4 byte*
indirizzo di rientro	*- 4 byte*
N	*- 4 byte*
X	*- 4 byte*
Y	*- 4 byte*

area di attivazione di ***funz*** (domanda **1**)	
registri salvati	
K	*- 4 byte*
FP precedente	*- 4 byte*
indirizzo di rientro	*- 4 byte*
P	*- 4 byte*
Q	*- 4 byte*

domanda **3** (numero di righe non significativo)

	indirizzo	**contenuto** (numerico) della **pila al punto di esecuzione** specificato dalla domanda **3**	
		eventuali registri salvati di funz … (non noti)	*SP*
chiamata a funz (–5, 15)	*3004*	*K = P * Q = (A – B) (A + B) = –75*	
	3008	*3040 (FP in 1ª chiamata a rec)*	*FP*
	2012	*indirizzo di rientro a rec (1, 5, 10)*	
	3016	*P = X – Y = A – B = –5*	
	3020	~~*Q=X+Y=A+B=15*~~ *valusc=K–Q=(A–B)(A+B)–(A+B)=–90*	
	3024	*D0*	
	3028	*D1*	
	3032	*D2*	
chiamata a rec (1, 5, 10)	*3036*	*D3*	
	3040	*3060 (FP precedente in main)*	
	3044	*indirizzo di rientro a main*	
	3048	*N = MAX = 1 (per ipotesi)*	
	3052	*X = A = 5*	
	3056	*Y = B = 10*	
cima area *main*	3060	NULL (FP di *main* - non punta a niente)	

codice 68000 di ***main*** (domanda **2** – num. righe non signif.) – *NON OTTIMIZZATO*

```
MAX:    EQU     3               // costante numerica
A:      DC.L    5               // varglob a
B:      DC.L    10              // varglob b
H:      DS.L    1               // varglob h
MAIN:   LINK    FP, #0          // collega
        MOVE.L  B, D0           // carica varglob b
        MOVE.L  D0, -(SP)       // impila 1° param (y) di rec
        MOVE.L  A, D1           // carica varglob a
        MOVE.L  D1, -(SP)       // impila 2° param (x) di rec
        MOVE.L  #MAX, D2        // carica costante MAX
        MOVE.L  D2, -(SP)       // impila 3° param (n) di rec
        BSR     REC             // chiama funzione rec
        ADDA.L  #8, SP          // abbandona param n e x
        MOVE.L  (SP)+, D0       // spila valusc di rec
        MOVE.L  D0, H           // memorizza varglob h
        UNLK    FP              // scollega
        END     MAIN            // direttiva di fine – ind. iniz. main
```

Si possono facilmente ottimizzare le due istruzioni consecutive MOVE.L (SP)+, D0 e MOVE.L D0, H sfruttando l'ortogonalità di MOVE, compattandole nella sola istruzione MOVE (SP)+, H, nonché compattare gli impilamenti, sempre sfruttando l'ortogonalità di MOVE.

codice 68000 di ***main*** (domanda **2** – num. righe non signif.) – *OTTIMIZZATO*

```
MAX:    EQU     3               // costante numerica
A:      DC.L    5               // varglob a
B:      DC.L    10              // varglob b
H:      DS.L    1               // varglob h
MAIN:   LINK    FP, #0          // collega
        MOVE.L  B, -(SP)        // impila 1° param (y) di rec
        MOVE.L  A, -(SP)        // impila 2° param (x) di rec
        MOVE.L  #MAX, -(SP)     // impila 3° param (n) di rec
        BSR     REC             // chiama funzione rec
        ADDA.L  #8, SP          // abbandona param n e x
        MOVE.L  (SP)+, H        // spila valusc e memorizza varglob h
        UNLK    FP              // scollega
        END     MAIN            // direttiva di fine – ind. iniz. main
```

codice 68000 di ***rec*** (domanda **2** – num. righe non signif.) – *NON OTTIMIZZATO*			
REC:	**LINK**	FP, #0	// collega
N:	**EQU**	8	// spiazzamento di param n
X:	**EQU**	12	// spiazzamento di param x
Y:	**EQU**	16	// spiazzamento di param y
	MOVEM.L	D0-D3, -(SP)	// salva i registri D0-D3 in pila
	MOVE.L	N(FP), D0	// carica param *n*
	CMPI.L	#0, D0	// confronta *n*
	BNE	ELSE	// se != 0 va' a ELSE
THEN:	**MOVE.L**	X(FP), D1	// carica param *x*
	MOVE.L	Y(FP), D2	// carica param *y*
	ADD.L	D1, D2	// calcola *x + y*
	MOVE.L	D2, Y(FP)	// sovrascrivi *valusc*
	BRA	ENDIF	// va' a ENDIF
ELSE:	**MOVE.L**	Y(FP), D2	// carica param *y*
	MOVE.L	D2, -(SP)	// impila 1° param (*y*) di *rec*
	MOVE.L	X(FP), D1	// carica param *x*
	MOVE.L	D1, -(SP)	// impila 2° param (*x*) di *rec*
	MOVE.L	N(FP), D0	// carica param *n*
	SUBI.L	#1, D0	// calcola *n - 1*
	MOVE.L	D0, -(SP)	// impila 3° param (*n*) di *rec*
	BSR	REC	// chiama funzione *rec*
	ADDA.L	#8, SP	// abbandona param *n e x*
	MOVE.L	X(FP), D1	// carica param *x*
	MOVE.L	Y(FP), D2	// carica param *y*
	SUB.L	D2, D1	// calcola *x - y*
	MOVE.L	D1, -(SP)	// impila 2° param (*p*) di *funz*
	BSR	FUNZ	// chiama funzione *funz*
	ADDA.L	#4, SP	// abbandona param *p*
	MOVE.L	(SP)+, D3	// spila *valusc* di *funz*
	MOVE.L	D3, Y(FP)	// sovrascrivi *valusc di rec*
ENDIF:	**MOVEM.L**	(SP)+, D0-D3	// ripristina registri
	UNLK	FP	// scollega
	RFS		// rientra

Ci sono varie facili ottimizzazioni possibili, con ortgonalità di MOVE, e altri piccoli aggiustamenti.

codice 68000 di ***rec*** (domanda **2** – num. righe non signif.) – *OTTIMIZZATO*			
REC:	**LINK**	FP, #0	// collega
N:	**EQU**	8	// spiazzamento di param *n*
X:	**EQU**	12	// spiazzamento di param *x*
Y:	**EQU**	16	// spiazzamento di param *y*
VU:	**EQU**	Y	// spiazzamento di *valusc* (= Y)
	MOVEM.L	D0-D2, -(SP)	// salva i registri D0-D2 in pila
	CMPI.L	#0, N(FP)	// confronta *n*
	BNE	ELSE	// se != 0 va' a ELSE
THEN:	**MOVE.L**	X(FP), D1	// carica param *x*
	MOVE.L	Y(FP), D2	// carica param *y*
	ADD.L	D1, D2	// calcola *x* + *y*
	MOVE.L	D2, VU(FP)	// sovrascrivi *valusc*
	BRA	ENDIF	// va' a ENDIF
ELSE:	**MOVE.L**	Y(FP), -(SP)	// impila 1° param (*y*) di *rec*
	MOVE.L	X(FP), -(SP)	// impila 2° param (*x*) di *rec*
	MOVE.L	N(FP), -(SP)	// impila 3° param (*n*) di *rec*
	SUBI.L	#1, (SP)	// aggiorna 3° param (*n*) su pila
	BSR	REC	// chiama funzione *rec*
	ADDA.L	#8, SP	// abbandona param *n* e *x*
	MOVE.L	X(FP), D1	// carica param *x*
	MOVE.L	Y(FP), D2	// carica param *y*
	SUB.L	D2, D1	// calcola *x* - *y*
	MOVE.L	D1, -(SP)	// impila 2° param (*p*) di *funz*
	BSR	FUNZ	// chiama funzione *funz*
	ADDA.L	#4, SP	// abbandona param *p*
	MOVE.L	(SP)+, VU(FP)	// spila e sovrascrivi *valusc*
ENDIF:	**MOVEM.L**	(SP)+, D0-D2	// ripristina registri
	UNLK	FP	// scollega
	RFS		// rientra

Si potrebbe ottimizzare perfino di più, sfruttando a fondo la semiortogonalità di ADD e SUB, ma il codice diventerebbe più difficile da leggere. In generale il calcolo di espressioni come x + y o x – y, con due o più variabili, è più chiaro e leggibile se svolto nei registri (come del resto fanno numerosi tipi di processore con linguaggio macchina meno ortogonale di 68000), prima caricando le variabili, poi elaborandole nei registri e infine memorizzando il risultato.

Nota bene: il valore restituito dalla chiamata ricorsiva a rec, funziona come parametro attuale per il parametro formale q della chiamata a funz; quando rec rientra, esso si trova già posizionato correttamente sulla cima pila dove andrebbe impilato q, pertanto viene lasciato in pila.

Nota bene: è stato introdotto il simbolo costante VU, di fatto uguale a Y, per rendere esplicita e localizzabile la sovrascrittura del valore di uscita; è un accorgimento solo per maggiore chiarezza.

Ecco una versione SUPEROTTIMIZZATA della funzione rec, con aggiustamenti algoritmici.

codice 68000 di ***rec*** (domanda **2** – num. righe non signif.) – *SUPEROTTIMIZZATO*			
REC:	**LINK**	FP, #0	// collega
N:	**EQU**	8	// spiazzamento di param *n*
X:	**EQU**	12	// spiazzamento di param *x*
Y:	**EQU**	16	// spiazzamento di param *y*
VU:	**EQU**	Y	// spiazzamento di *valusc* (= Y)
	MOVEM.L	D0-D2, -(SP)	// salva i registri D0-D2 in pila
	MOVE.L	Y(FP), D2	// carica param *y*
	MOVE.L	X(FP), D1	// carica param *x*
	MOVE.L	N(FP), D0	// carica param *n*
	BNE	ELSE	// se != 0 va' a ELSE
THEN:	**ADD.L**	D1, D2	// calcola *x* + *y*
	MOVE.L	D2, VU(FP)	// sovrascrivi *valusc*
	BRA	ENDIF	// va' a ENDIF
ELSE:	**MOVE.L**	D2, -(SP)	// impila 1° param (*y*) di *rec*
	MOVE.L	D1, -(SP)	// impila 2° param (*x*) di *rec*
	SUBI.L	#1, D0	// calcola *n - 1*
	MOVE.L	D0, -(SP)	// impila 3° param (*n*) di *rec*
	BSR	REC	// chiama funzione *rec*
	ADDA.L	#8, SP	// abbandona param *n* e *x*
	SUB.L	D2, D1	// calcola *x - y*
	MOVE.L	D1, -(SP)	// impila 2° param (*p*) di *funz*
	BSR	FUNZ	// chiama funzione *funz*
	ADDA.L	#4, SP	// abbandona param *p*
	MOVE.L	(SP)+, VU(FP)	// spila e sovrascrivi *valusc*
ENDIF:	**MOVEM.L**	(SP)+, D0-D2	// ripristina registri
	UNLK	FP	// scollega
	RFS		// rientra

ATTENZIONE: qui questa versione è mostrata per completezza e come utile esercizio per lo studio personale, ma è SCONSIGLIABILE tentare di produrne una così nell'affrontare il tema in sede d'esame, poiché occorre molto tempo e parecchia riflessione per ottenerla con la ragionevole certezza che sia corretta; inoltre essa è molto sensibile agli errori e qualunque piccola svista avrebbe conseguenze pesanti sulla sua correttezza; invece le soluzioni precedenti sono di carattere più sistematico e più immuni a piccole sviste, poiché le istruzioni corrispondenti ai vari statement C che esse realizzano, sono meno interdipendenti e pertanto limitano la propagazione degli errori.