## esercizio n. 5 – linguaggio macchina

Si vuole **tradurre** in linguaggio macchina simbolico 68000 (linguaggio assemblatore) il frammento di programma riportato qui sotto: programma principale ***main*** e funzioni ***F*** e ***G***. Nel tradurre non si tenti di ottimizzare accorpando istruzioni C indipendenti. La memoria ha indirizzo da **32 bit** ed è indirizzabile per **byte**. Le variabili intere sono da **32 bit**. Si noti che:

- il programma principale ***main*** è una funzione e ha area di attivazione, ma non restituisce nulla; esso non ha parametri e ha una variabile locale intera; chiama la funzione ***F*** come procedura, passandole come parametro l'indirizzo di un elemento del vettore *vet*, che è una variabile globale
- la funzione ***F*** non restituisce nulla; essa ha un parametro puntatore a carattere (ossia a byte ovvero a intero da 8 bit) e non ha variabili locali; chiama la funzione ***G***; ha effetto collaterale sulla variabile globale *vet* tramite il parametro puntatore *p*
- la funzione ***G*** restituisce un valore carattere; essa ha un parametro carattere e non ha variabili locali

Le funzioni salvano i registri utilizzati, e le convenzioni per il passaggio di parametri e valore di uscita sono:

- il parametro, se esiste, viene passato sulla **pila**
- il valore di uscita, se esiste, viene sovrascritto al **parametro** sulla **pila**

**Si chiede** di scrivere quanto segue:

1. in tabella **1.a** le costanti e le variabili globali, in tabella **1.b** l'area di attivazione del programma principale ***main*** e in tabella **1.c** il codice in linguaggio macchina 68000 di ***main***,, secondo le convenzioni del corso
2. in tabella **2.a** l'area di attivazione della funzione ***G*** e in tabella **2.b** il codice in linguaggio macchina 68000 di ***G***, secondo le convenzioni del corso; inoltre si disegni l'albero sintattico dell'espressione in ***G***
3. in tabella **3.a** l'area di attivazione della funzione ***F*** , e in tabella **3.b** il contenuto **simbolico** (espresso usando i nomi delle variabili, parametri, registri, ecc), e dove possibile anche i valori **numerici**, e i valori dell'**indirizzo** delle aree di attivazione impilate di ***main***, di ***F*** e di ***G*** **,** durante la **seconda** iterazione del ciclo **for** (ossia quando si ha *i* = 1), così come risultano subito dopo l'esecuzione dell'istruzione LINK e l'eventuale salvataggio di registri

**programma in linguaggio C**

```
/* programma principale */
#define  N  3      /* costante simbolica */
char vet [N];      /* variabile globale  */
void main ( ) {
   int i;          /* variabile locale   */
   for (i = 0; i < N; i++) {
      F (&vet [i]);
   } /* for */
   return;
} /* main */
```

```
/* funzione F */
void F (char * p) {
   *p = G (*p);
   return;
} /* F */
```

```
/* funzione G */
char G (char c) {
   return (c * (c - 1) + 2);
} /* G */
```

**domanda 1**

| tabella 1.a – costanti e variabili globali | | | |
|---|---|---|---|
| | **ORG** | 200000 | *// indirizzo iniziale del segmento dati* |
| N: | **EQU** | 3 | *// costante simbolica* |
| VET: | **DS.B** | N | *// vettore di caratteri non inizializzato* |

**tabella 1.b – area di attivazione del programma principale *main*** (da completare)

| contenuto simbolico |
|---|
| |
| *eventuali reg. salvati – qui D0 e A0* |
| *varloc i* |
| FP precedente (= NULL poiché torna al SO) |
| indirizzo di rientro (= NULL poiché torna al SO) |
| cella già occupata |

si chiede solo il contenuto simbolico (non l'indirizzo)

| tabella 1.c – codice del programma principale *main* (non ottimizzato) | | | |
|---|---|---|---|
| MAIN: | **LINK** | FP, *#-4* | *// collega* |
| I: | **EQU** | *-4* | *// spiazzamento varloc i* |
| | ***MOVEM.L*** | *D0/A0, -(SP)* | *// salva registri D0 e A0* |
| | ***MOVE.L*** | *#0, D0* | *// inizializza registro D0* |
| | ***MOVE.L*** | *D0, I(FP)* | *// inizializza varloc i* |
| *LOOP:* | ***MOVE.L*** | *I(FP), D0* | *// carica varloc i* |
| | ***CMPI.L*** | *#N, D0* | *// confronta registro D0* |
| | ***BGE*** | *POOL* | *// se esito >= 0 va' a POOL* |
| | ***MOVE.L*** | *I(FP), D0* | *// carica varloc i* |
| | ***MOVEA.L*** | *#VET, A0* | *// inizializza registro A0* |
| | ***ADDA.L*** | *D0, A0* | *// calcola ind. di elem. vet [i]* |
| | ***MOVE.L*** | *A0, -(SP)* | *// impila param di F* |
| | ***BSR*** | *F* | *// chiama funzione F* |
| | ***ADDA.L*** | *#4, SP* | *// abbandona param di F* |
| | ***MOVE.L*** | *I(FP), D0* | *// carica varloc i* |
| | ***ADDI.L*** | *#1, D0* | *// incrementa registro D0* |
| | ***MOVE.L*** | *D0, I(FP)* | *// memorizza varloc i* |
| | ***BRA*** | *LOOP* | *// torna a LOOP* |
| *POOL:* | ***MOVEM.L*** | *(SP)+, D0/A0* | *// ripristina registri D0 e A0* |
| | ***UNLK*** | *FP* | *// scollega* |
| | ***RTS*** | | *// rientra* |

*Nota: codifica secondo le convenzioni del corso, senza alcuna ottimizzazione; si può ottimizzare (vedi sotto).*

| tabella 1.c – codice del programma principale ***main*** (ottimizzato) | | | |
|---|---|---|---|
| MAIN: | **LINK** | FP, #-4 | // collega |
| I: | **EQU** | -4 | // spiazzamento varloc i |
| | ***MOVEM.L*** | A0, -(SP) | // salva registro A0 |
| | ***MOVE.L*** | #0, I(FP) | // inizializza varloc i |
| LOOP: | ***CMPI.L*** | #N, I(FP) | // confronta varloc i |
| | ***BGE*** | POOL | // se esito >= 0 va' a POOL |
| | ***MOVEA.L*** | #VET, A0 | // inizializza registro A0 |
| | ***ADDA.L*** | I(FP), A0 | // calcola ind. di elem. vet [i] |
| | ***MOVE.L*** | A0, -(SP) | // impila param di F |
| | ***BSR*** | F | // chiama funzione F |
| | ***ADDA.L*** | #4, SP | // abbandona param di F |
| | ***ADDI.L*** | #1, I(FP) | // incrementa varloc i |
| | ***BRA*** | LOOP | // torna a LOOP |
| POOL: | ***MOVEM.L*** | (SP)+, A0 | // ripristina registro A0 |
| | ***UNLK*** | FP | // scollega |
| | ***RTS*** | | // rientra |

*Nota: qui il codice è ottimizzato sfruttando l'ortogonalità di MOVE e la semiortogonalità di CMP e ADD.*

**domanda 2**

**tabella 2.a – area di attivazione della funzione *G***

| contenuto simbolico |
|---|
| |
| |
| *eventuali reg. salvati – qui D0 e D1* |
| *FP precedente* |
| *indirizzo di rientro* |
| *param c e valusc* |
| cella già occupata |

si chiede solo il contenuto simbolico
(non l'indirizzo)

**albero sintattico dell'espressione nella funzione *G***

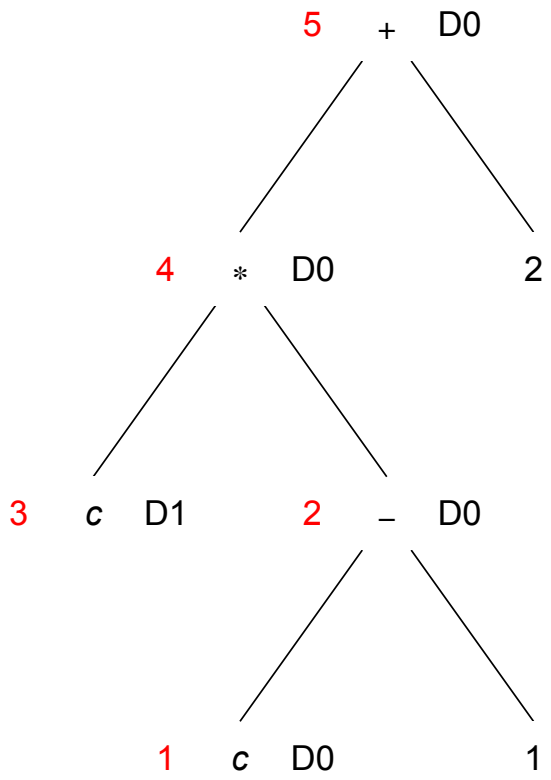


**tabella 2.b – codice della funzione *G***

| | | | |
|---|---|---|---|
| G: | **LINK** | *FP, #0* | *// collega (nessuna varloc)* |
| C: | **EQU** | *+8* | *// spiazzamento param c* |
| | ***MOVEM.B*** | *D0-D1, -(SP)* | *// salva registri D0 e D1* |
| | ***MOVE.B*** | *C(FP), D0* | *// 1) carica c* |
| | ***SUBI.B*** | *#1, D0* | *// 2) calcola sottrazione* |
| | ***MOVE.B*** | *C(FP), D1* | *// 3) (ri)carica c* |
| | ***MULS*** | *D1, D0* | *// 4) calcola moltiplicazione* |
| | ***ADDI.B*** | *#2, D0* | *// 5) calcola addizione* |
| | ***MOVE.B*** | *D0, C(FP)* | *// sovrascrivi valusc a param* |
| | ***MOVEM.B*** | *(SP)+, D0-D1* | *// ripristina registri D0 e D1* |
| | ***UNLK*** | *FP* | *// scollega* |
| | ***RTS*** | | *// rientra* |
| | | | |
| | | | |
| | | | |
| | | | |

*Nota: questa funzione lavora solo su dati di tipo char, ossia byte.*

**domanda 3**

**tabella 3.a – area di attivazione della funzione *F***

| **contenuto simbolico** |
|---|
| |
| *eventuali reg. salvati - NESSUNO* |
| *FP precedente* |
| *indirizzo di rientro* |
| *param p* |
| cella già occupata |

si supponga che la funzione *F* non salvi nessun registro

si chiede solo il contenuto simbolico (non l'indirizzo)

**tabella 3.b – contenuto effettivo** (simbolico e dove possibile anche valore numerico) **e indirizzo delle aree di attivazione impilate**

| **indirizzo** | **contenuto simbolico** (dove possibile anche valore numerico) |
|---|---|
| | |
| | |
| | |
| | |
| *100045-100037* | *registri D0 e D1* |
| *100049* | *FP precedente (= 100058)* |
| *100053* | *indirizzo di rientro a F* |
| *area G 100054* | *param c (c = vet[1]) – ingombra 1 byte !* |
| *100058* | *FP precedente (= 100082)* |
| *100062* | *indirizzo di rientro a main* |
| *area F 100066* | *param p (p = &vet[1] = 200001)* |
| *100074-100070* | *registri D0 e A0* |
| *100078* | *varloc i (i = 1)* |
| *100082* | *FP precedente (= NULL)* |
| *area main 100086* | *indirizzo di rientro (= NULL)* |
| 100090 | cella già occupata |

## esercizio n. 5 – linguaggio macchina

### prima parte – traduzione da C a codice macchina

Si vuole **tradurre** in linguaggio macchina simbolico (linguaggio assemblatore) 68000 il frammento di programma (funzioni ***call*** e ***f_ric***) riportato qui sotto. Nel tradurre non si tenti di accorpare od ottimizzare insieme istruzioni C indipendenti. La memoria ha indirizzo da **32 bit** ed è indirizzabile **per byte**. Le variabili intere sono da **32 bit**. Si noti che:

- la funzione ***call*** non restituisce nulla (di fatto è una procedura), non ha parametri e ha due variabili locali intere; essa chiama la funzione ***f_ric*** (che è una funzione a tutti gli effetti), trattandola come una procedura ossia disinteressandosi del valore in uscita di ***f_ric***
- la funzione ricorsiva ***f_ric*** restituisce un valore intero, ha due parametri e non ha variabili locali

Le funzioni salvano sempre i registri utilizzati e le convenzioni per il passaggio dei parametri e del valore di ritorno sono le seguenti:

- i parametri vengono passati sulla **pila** e caricati in ordine inverso di elencazione (cioè il primo parametro elencato nella testata della funzione è l'ultimo da caricare sulla pila)
- il valore di uscita, se esiste, viene sovrascritto al **primo parametro caricato sulla pila**

**Si chiede** di scrivere:

- in tabella 1.a e in tabella 2.a il codice in linguaggio macchina 68000 delle funzioni ***call*** e ***f_ric,*** rispettivamente, tradotte secondo le convenzioni del corso
- in tabella 1.b e in tabella 2.b il contenuto **simbolico** (espresso usando i nomi delle variabili, parametri, registri, ecc) e i **valori dell'indirizzo** dell'area di attivazione di ***call*** e di quella relativa alla prima invocazione di ***f_ric,*** così come risultano **subito dopo l'esecuzione dell'istruzione LINK e il salvataggio di eventuali registri**

**frammento di programma in linguaggio C**

```
/* funzione call */
void call ( ) {
  int i;
  int j = 3;
  i = 0;
  f_ric (&i, j);
  return;
} /* call */
-------------------------------------
/* funzione f_ric */
int f_ric (int *p, int q) {
   if (q == 0) {
      return 0;
   } else {
      *p = *p + 1;
      return f_ric (p, q - 1);
   } /* if */
} /* f_ric */
```

| tabella 1.a – codice 68000 della funzione *call* | | | |
|---|---|---|---|
| CALL: | **LINK** | FP, #-8 | // collega |
| I: | **EQU** | -4 | // spiazzamento varloc i |
| J: | **EQU** | -8 | // spiazzamento varloc j |
| | ***MOVEM.L*** | A0, -(SP) | // salva registro A0 |
| | ***MOVE.L*** | #3, J(FP) | // inizializza varloc j a 3 |
| | ***MOVE.L*** | #0, I(FP) | // inizializza varloc i a 0 |
| | ***MOVE.L*** | J(FP), -(SP) | // impila param j di f_ric |
| | ***MOVEA.L*** | FP, A0 | // calcola ind di varloc i (a) |
| | ***ADDA.L*** | #I, A0 | // calcola ind di varloc i (b) |
| | ***MOVEA.L*** | A0, -(SP) | // impila param &i di f_ric |
| | ***BSR*** | F_RIC | // chiama funzione f_ric |
| | ***ADDA.L*** | #8, SP | // abbandona param p e q di f_ric |
| | ***MOVEM.L*** | (SP)+, A0 | // ripristina registro A0 |
| | ***UNLK*** | FP | // scollega |
| | ***RTS*** | | // ritorna |
| | | | |
| | | | |
| | | | |
| *Nota: qui i e j sono inizializzate senza caricarle in un reg. D* | | | |
| *Nota: la coppia di istruzioni macchina (a) e (b) si ottimizza così:* | | | |
| | ***LEA*** | I(FP), A0 | // calcola ind di i e scrivilo in A0 |
| *Nota: qui MOVEM salva un solo registro ed equivale a MOVE* | | | |
| *Nota: call abbandona il valore in uscita di f_ric, senza usarlo* | | | |

**tabella 1.b – area di attivazione di *call***

| indirizzo | contenuto simbolico |
|---|---|
| | |
| | |
| | *(eventuali reg. salvati – qui A0)* |
| *100044* | *varloc j* |
| *100048* | *varloc i* |
| *100052* | *FP precedente* |
| *100056* | *indirizzo di ritorno* |
| 100060 | cella già occupata |

*Nota: poiché la funzione call non restituisce nulla (è di tipo void e in C corrisponde a una procedura), nell'area di attivazione di call non occorre lasciare alcuno spazio dove sovrascrivere il valore in uscita; non sarebbe peraltro errato lasciare spazio, che però non verrebbe usato e dunque andrebbe sprecato.*

| tabella 2.a – codice 68000 della funzione *f_ric* | | | |
|---|---|---|---|
| F_RIC: | **LINK** | *FP, #0* | *// collega* |
| P: | **EQU** | *+8* | *// spiazzamento param p* |
| Q: | **EQU** | *+12* | *// spiazzamento param q* |
| | ***MOVEM.L*** | *A0/D0, -(SP)* | *// salva registri D0 e A0* |
| | ***CMPI.L*** | *#0, Q(FP)* | *// valuta condizione IF* |
| | ***BNE*** | *ELSE* | *// se != 0 va' a ramo ELSE* |
| *THEN:* | ***MOVE.L*** | *#0, D0* | *// carica costante 0* |
| | ***MOVE.L*** | *D0, Q(FP)* | *// sovrascrivi param q di f_ric* |
| | ***BRA*** | *END_IF* | *// va' a fine IF* |
| *ELSE:* | ***MOVEA.L*** | *P(FP), A0* | *// dereferenzia param p* |
| | ***MOVE.L*** | *(A0), D0* | *// dereferenzia param p* |
| | ***ADDI.L*** | *#1, D0* | *// incrementa oggetto puntato *p* |
| | ***MOVE.L*** | *D0, (A0)* | *// incrementa oggetto puntato *p* |
| | ***MOVE.L*** | *Q(FP), D0* | *// carica param q* |
| | ***SUBI.L*** | *#1, D0* | *// calcola espressione q – 1* |
| | ***MOVE.L*** | *D0, -(SP)* | *// impila param q – 1 di f_ric* |
| | ***MOVE.***L | *P(FP), D0* | *// carica param p* |
| | ***MOVE.L*** | *D0, -(SP)* | *// impila param p di f_ric* |
| | ***BSR*** | *F_RIC* | *// chiama funz ricorsiva f_ric* |
| | ***ADDA.L*** | *#4, SP* | *// abbandona param p di f_ric* |
| | ***MOVE.L*** | *(SP)+, D0* | *// spila valusc di f_ric* |
| | ***MOVE.L*** | *D0, Q(FP)* | *// sovrascrivi param q di f_ric* |
| *END_IF:* | ***MOVEM.L*** | *(SP)+, A0/D0* | *// ripristina registri A0 e D0* |
| | ***UNLK*** | *FP* | *// scollega* |
| | ***RTS*** | | *// rientra* |
| *Nota: qui si usa un solo reg dati D0, e si può ancora ottimizzare* | | | |
| *Nota: qui q viene confrontato senza caricarlo in un reg. D* | | | |

**tabella 2.b – area di attivazione di *f_ric***

| indirizzo | contenuto simbolico |
|---|---|
| | |
| | *(eventuali reg. salvati – qui D0 e A0)* |
| *100020* | *FP precedente* |
| *100024* | *indirizzo di ritorno* |
| *100028* | *param p* |
| *100032* | *param q e valusc* |
| 100036 | cella già occupata |

| **tabella 2.a – codice 68000 della funzione *f_ric* - ottimizzata** | | | |
|---|---|---|---|
| F_RIC: | **LINK** | *FP, #0* | *// collega* |
| P: | **EQU** | *+8* | *// spiazzamento param p* |
| Q: | **EQU** | *+12* | *// spiazzamento param q* |
| | ***MOVEM.L*** | *A0/D0, -(SP)* | *// salva registri D0 e A0* |
| | ***CMPI.L*** | *#0, Q(FP)* | *// valuta condizione IF* |
| | ***BNE*** | *ELSE* | *// se != 0 va' a ramo ELSE* |
| *THEN:* | ***MOVE.L*** | *#0, Q(FP)* | *// sovrascrivi param q di f_ric* |
| | ***BRA*** | *END_IF* | *// va' a fine IF* |
| *ELSE:* | ***MOVEA.L*** | *P(FP), A0* | *// dereferenzia param p* |
| | ***ADDI.L*** | *#1, (A0)* | *// incrementa oggetto puntato *p* |
| | ***MOVE.L*** | *Q(FP), D0* | *// carica param q* |
| | ***SUBI.L*** | *#1, D0* | *// calcola espressione q – 1* |
| | ***MOVE.L*** | *D0, -(SP)* | *// impila param q – 1 di f_ric* |
| | ***MOVE.L*** | *P(FP), -(SP)* | *// impila param p di f_ric* |
| | ***BSR*** | *F_RIC* | *// chiama funz ricorsiva f_ric* |
| | ***ADDA.L*** | *#4, SP* | *// abbandona param p di f_ric* |
| | ***MOVE.L*** | *(SP)+, Q(FP)* | *// sovrascrivi param q di f_ric* |
| *END_IF:* | ***MOVEM.L*** | *(SP)+, A0/D0* | *// ripristina registri A0 e D0* |
| | ***UNLK*** | *FP* | *// scollega* |
| | ***RTS*** | | *// rientra* |
| *Nota: qui si ottimizza sfruttando l'ortogonalità di MOVE e ADDI* | | | |

**seconda parte – simulazione delle chiamate ai sottoprogrammi**

In modo coerente con quanto risposto nella domanda precedente, si chiede ora di riportare nella tabella 3 i valori effettivi di parametri e variabili (ove disponibili) presenti nelle aree di attivazione di ***call*** e di ***f_ric*** subito dopo il salvataggio dei registri relativo alla **terza invocazione di *f_ric***.

**tabella 3 – contenuto effettivo delle aree di attivazione**

| indirizzo | contenuto simbolico |
|---|---|
| | |
| | |
| | |
| | |
| | |
| *99968 - 99972* | *D0 e A0* |
| *99976* | *FP precedente* |
| *99980* | *indirizzo di ritorno* |
| *99984* | *P = 100048* |
| *area f_ric 99988* | *Q = 0* |
| *99992 - 99996* | *D0 e A0* |
| *100000* | *FP precedente* |
| *100004* | *indirizzo di ritorno* |
| *100008* | *P = 100048* |
| *area f_ric 100012* | *Q = 1* |
| *100016 - 100020* | *D0 e A0* |
| *100024* | *FP precedente* |
| *100028* | *indirizzo di ritorno* |
| *100032* | *P = 100048* |
| *area f_ric 100036* | *Q = 2* |
| *100040* | *A0* |
| *100044* | *J = 3* |
| *100048* | *I = ~~0, 1,~~ 2* |
| *100052* | *FP precedente* |
| *area call 100056* | *indirizzo di ritorno* |
| 100060 | cella già occupata |

*Nota: la variabile locale i di call, assume tre valori nel corso dell'esecuzione poiché viene aggiornata per puntatore da parte della funzione ricorsiva f_ric che è chiamata tre volte, appunto. La sequenza di valori di i è 0, 1 e 2;i primi due vengono progressivamente sovrascritti e il valore finale è 2.*

## esercizio n. 5 – linguaggio macchina

### prima parte – traduzione da C a codice

Si vuole **tradurre** in linguaggio macchina simbolico (linguaggio assemblatore) 68000 la funzione ricorsiva ***F*** riportata qui sotto. La funzione *F* invoca anche la funzione ***G***, di cui è riportata la testata. Nel tradurre non si tenti di accorpare od ottimizzare insieme istruzioni C indipendenti. La memoria ha indirizzo da **32 bit** ed è indirizzabile **per byte**. Le variabili intere sono da **32 bit**.

Le funzioni salvano sempre i registri utilizzati e le convenzioni per il passaggio dei parametri e del valore di ritorno sono le seguenti:

- i parametri sono passati sulla **pila** e caricati in ordine inverso di elencazione (cioè il primo parametro elencato nella testata della funzione è l'ultimo a essere caricato sulla pila)
- il valore di uscita è **sovrascritto** al primo parametro caricato sulla pila

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

1. **Si scriva** in tabella 1 il contenuto simbolico (cioè nomi di variabili, di parametri, ecc) della pila subito dopo l'esecuzione dell'istruzione LINK in seguito a un'invocazione di *F*.
2. **Si scriva** in tabella 2 il codice in linguaggio macchina 68000 di *F* tradotta con le convenzioni del corso.

**funzione *F* in linguaggio C**

```
/* testata funzione G  */
int G (int);

/* funzione F          */
int F (int x, int y) {
   /* variabile locale */
   int i;
   /* parte esecutiva  */
   i = x * y;
   if (x <= 0) return G ( y + i );
   else return y * F ( x - y, G ( y ) );
} /* fine F */
```

**Tabella 1**

| indirizzo | contenuto simbolico |
|---|---|
| | |
| | *(eventuali reg. salvati)* |
| *100040* | *i* |
| *100044* | *FP precedente* |
| *100048* | *indirizzo di ritorno* |
| *100052* | *x* |
| *100056* | *y* |
| 100060 | cella già occupata |

| Tabella 2 – codice 68000 della funzione *F* – non ottimizzato | | | |
|---|---|---|---|
| F: | **LINK** | FP, #-4 | // collega |
| | **MOVEM.L** | D0-D3, -(SP) | // salva registri D0, D1, D2 e D3 |
| X: | **EQU** | 8 | // spiazzamento di param x |
| Y: | **EQU** | 12 | // spiazzamento di param y |
| I: | **EQU** | -4 | // spiazzamento di varloc i |
| | **MOVE.L** | X(FP), D0 | // carica param x |
| | **MOVE.L** | Y(FP), D1 | // carica param y |
| | **MULS** | D0, D1 | // calcola x * y |
| | **MOVE.L** | D1, I(FP) | // aggiorna varloc i |
| IF: | **MOVE.L** | X(FP), D0 | // carica param x |
| | **CMPI.L** | #0, D0 | // confronta param x |
| | **BGT** | ELSE | // se > 0 va' a ELSE |
| THEN: | **MOVE.L** | Y(FP), D1 | // carica param y |
| | **MOVE.L** | I(FP), D2 | // carica varloc i |
| | **ADD.L** | D1, D2 | // calcola y + i |
| | **MOVE.L** | D2, -(SP) | // impila arg y + i di G |
| | **BSR** | G | // chiama funzione G |
| | **MOVE.L** | (SP)+, D3 | // spila valusc di G |
| | **MOVE.L** | D3, Y(FP) | // restituisci valusc di F |
| | **BRA** | ENDIF | // va' a ENDIF |
| ELSE: | **MOVE.L** | Y(FP), D1 | // carica param y |
| | **MOVE.L** | D1, -(SP) | // impila arg y di G |
| | **BSR** | G | // chiama G (valusc di G resta in pila) |
| | **MOVE.L** | X(FP), D0 | // carica param x |
| | **SUB.L** | D1, D0 | // calcola x - y |
| | **MOVE.L** | D0, -(SP) | // impila arg x - y di F |
| | **BSR** | F | // chiama ricorsivamente funzione F |
| | **ADDA.L** | #4, SP | // abbandona param x |
| | **MOVE.L** | (SP)+, D3 | // spila valusc di F |
| | **MULS** | D1, D3 | // calcola y * valusc di F |
| | **MOVE.L** | D3, Y(FP) | // restituisci valusc di F |
| ENDIF: | **MOVEM.L** | (SP)+, D0-D3 | // ripristina registri |
| | **UNLK** | FP | // scollega |
| | **RTS** | | // rientra |
| | | | |
| | | | |
| | | | |
| | | | |

*I registri D0, D1, D2 e D3 sono assegnati a param x, y, varloc i e valori di uscita di G e F, rispettivamente.*

**seconda parte – processo di assemblaggio**

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 di memoria | # byte | indirizzo (in decimale) |
|---|---|---|---|---|---|
| | **ORG** | 1000 | 0 | 0 | |
| N: | **EQU** | -10 | 0 | 0 | |
| B: | **DS.W** | 10 | 10 | 20 | 1000 |
| A: | **DC.L** | 10 | 2 | 4 | 1020 |
| | **MOVE** | A.L, D2 | 3 | 6 | 1024 |
| CICLO: | **CMP** | #N.L, D2 | 3 | 6 | 1030 |
| | **BLT** | FINE | 2 | 4 | 1036 |
| START: | **MOVE** | A.L, D2 | 3 | 6 | 1040 |
| | **MOVEA** | B.L, A0 | 3 | 6 | 1046 |
| | **MOVE** | (A0), D1 | 1 | 2 | 1052 |
| | **SUB** | D1, D2 | 1 | 2 | 1054 |
| | **BRA** | CICLO | 2 | 4 | 1056 |
| | **BSR** | FUNZ | 2 | 4 | 1060 |
| FINE: | **END** | START | 0 | 0 | 1064 |

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

| tabella dei simboli | |
|---|---|
| nome simbolo | valore numerico (in decimale) |
| *N* | *−10* |
| *B* | *1000* |
| *A* | *1020* |
| *CICLO* | *1030* |
| *FINE* | *1064* |
| *START* | *1040* |
| *FUNZ* | |

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

spiazzamento in BRA = $1030 - 1060 = -30_{dieci}$

d) **Si indichi** quale valore verrà caricato nel Program Counter quando questo programma verrà lanciato in esecuzione: **PC** = *1040*

## esercizio n. 2 – linguaggio macchina

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

La memoria ha indirizzi da **32 bit** ed è indirizzabile per byte. Le variabili intere sono da **32 bit**, le variabili carattere sono da **8 bit** e i puntatori 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 devono sempre salvare i registri che utilizzano
- il programma principale *main* è anch'esso una funzione e pertanto ha una sua area di attivazione (che qui non si chiede di progettare), ma non ha variabili locali e non salva registri in pila

Si chiede di svolgere i punti seguenti:

1. **Riportare**, nelle colonne "domanda 1" delle tabelle, il contenuto simbolico (espresso usando i nomi delle variabili, dei parametri, dei registri, ecc) e i valori dell'indirizzo dell'area dati statici e dell'area di attivazione della funzione *funz*, così come risultano subito dopo l'esecuzione dell'istruzione macchina LINK presente all'inizio di *funz*.
2. **Riportare**, nella colonne "domanda 2" delle tabelle, quanto segue:
   a) il valore delle celle (il valore numerico effettivo - si scriva X se il valore è sconosciuto) subito **dopo** l'esecuzione dello **statement 1** nella **seconda** chiamata a *funz*
   b) il valore numerico contenuto nei registri FP e SP subito **dopo** l'esecuzione dell'istruzione macchina LINK presente all'inizio di *funz*
3. **Scrivere** il codice in linguaggio macchina 68000 di *main*, coerente con le specifiche e con le risposte ai punti precedenti (il numero di righe non è significativo e le prime righe sono già date).
4. **Disegnare** l'albero sintattico dell'espressione che figura nello **statement 2** di *funz*, **numerare** i nodi e **assegnare** i registri sull'albero, e **scrivere** il codice in linguaggio macchina 68000 dello **statement 2** (trascurando il resto del codice di *funz*), coerente con le specifiche e le risposte ai punti precedenti (il numero di righe non è significativo e le prime righe sono già parzialmente date).

---

**programma in linguaggio C**

```
#define   N = 3

/* variabili globali                                          */
/* stringa pre-inizializzata                                  */
char string [N] = { 'a', 'b', 'c' };
int i;      /* intero    */

/* funzione randc – restituisce un carattere casuale    */
char randc ( ) /* testata funzione di libreria randc    */

/* funzione funz – restituisce un carattere             */
char funz (char * p) {   /* param. p è puntatore a char */
   /* variabili locali                                        */
   int a;   /* intero    */
   char c;  /* carattere */
   /* parte esecutiva                                         */
   a = 0;
   c = *p;                                  /* statement 1 */
   c = (randc ( ) +  c) * 8 - 2;            /* statement 2 */
   return c;
} /* funz */

/* programma principale main                                  */
void main ( ) {
   /* parte esecutiva                                         */
   i = N;
   do {
       i--;
       string [i] = funz (&string [i]);
   } while (i > 0); /* do_while */
} /* main */
```

## spazio per le variabili globali

| indirizzo | **contenuto** (simbolico) (domanda 1) | |
|---|---|---|
| ... | **celle già occupate** | |
| 10000 | *string [0]* | *- 1 byte* |
| *10001* | *string [1]* | *- 1 byte* |
| *10002* | *string [2]* | *- 1 byte* |
| *10003* * | *i* | *- 4 byte* |
| *10007* * | | |
| *...* | | |
| *...* | | |
| *...* | | |
| *...* | | |

## area di attivazione della funzione *funz*

| indirizzo | **contenuto** (simbolico) (domanda 1) | | **valore** (reale) subito **dopo** avere eseguito lo **statement 1** nella **seconda** chiamata a *funz* (domanda 2) |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| *200067* | *C* | *- 1 byte* | *'b' (valore di string [1])* |
| *200068* | *A* | *- 4 byte* | *0 (valore iniziale di varloc a)* |
| *200072* | *FP precedente* | *- 4 byte* | *200084 (ind campo FP di main)* |
| *200076* | *indirizzo di rientro* | *- 4 byte* | *X (indirizzo di rientro a main)* |
| *200080* | *P (e valusc 1 byte)* | *- 4 byte* | *10001 (indirizzo di string [1])* |
| 200084 | **cella già occupata** | | *NULL (non punta a niente)* |

*poiché main non ha variabili locali, la cella di indirizzo 200084 è quella dello FP precedente di main (e avrà valore NULL poiché non punta a niente)*

| | **valore** (reale) subito **dopo** avere eseguita l'istruzione LINK in *funz* (domanda 2) |
|---|---|
| registro FP | *200072* |
| registro SP | *200067* |

**Oppure 10004 e 10008, rispettivamente, per conservare l'allineamento delle variabili agli indirizzi.*

| **codice 68000 di** *main* (domanda 3) | | | |
|---|---|---|---|
| | **ORG** | 10000 | // indirizzo virtuale del segmento unico |
| N: | **EQU** | 3 | // costante N = 3 |
| STRING: | **DC.B** | 'a','b','c' | // vettore di N = 3 caratteri (byte) |
| I: | **DS.L** | 1 | // varglob i a 32 bit |
| MAIN: | ... | | // inizio del codice di MAIN (LINK) |
| | **MOVE.L** | #N, D0 | // inizializza D0 |
| | **MOVE.L** | D0, I | // inizializza varglob i |
| DO_WHILE: | **MOVE.L** | I, D0 | // carica varglob i in D0 |
| | **SUBI.L** | #1, D0 | // decrementa D0 |
| | **MOVE.L** | D0, I | // aggiorna varglob i |
| | **MOVEA.L** | #STRING, A0 | // carica costante STRING in A0 |
| | **MOVE.L** | I, D0 | // carica varglob i in D0 |
| | **ADDA.L** | D0, A0 | // calcola indirizzo di elem string [i] |
| | **MOVE.L** | A0, -(SP) | // impila param p di funz |
| | **BSR** | FUNZ | // chiama funzione funz |
| | **MOVE.B** | (SP)+, D1 | // spila valusc (1 byte) di funz |
| | **ADDA.L** | #3, SP | // abbandona 3 byte in cima alla pila |
| | **MOVEA.L** | #STRING, A0 | // carica costante STRING in A0 |
| | **MOVE.L** | I, D0 | // carica varglob i in D0 |
| | **ADDA.L** | D0, A0 | // calcola indirizzo di elem string [i] |
| | **MOVE.B** | D1, (A0) | // aggiorna elem string [i] |
| | **MOVE.L** | I, D0 | // carica varglob i in D0 |
| | **CMPI.L** | #0, D0 | // confronta varglob D0 |
| | **BGT** | DO_WHILE | // ripeti ciclo DO_WHILE |
| | ... | | // resto del codice di MAIN (UNLK e RTS) |
| | | | |
| | | | |
| | | | |
| | | | |

*Osservazioni: qui si usano tre registri, A0, D0 e D1; sono possibili varie ottimizzazioni, usando più registri, riusandone il valore tra due o più statement o parti di statement, e ricorrendo ai modi di indirizzamento più sofisticati e/o all'ortogonalità. Abbandonare 3 byte sulla pila segue dai diversi ingombri di p (puntatore ingombra 4 byte) e del valore di uscita di funz (carattere ingombra 1 byte).*

## **codice 68000 di** *main* (domanda 3) - OTTIMIZZATO

```
          ORG      10000          // indirizzo virtuale del segmento unico
N:        EQU      3              // costante N = 3
STRING:   DC.B     'a','b','c'    // vettore di N = 3 caratteri (byte)
I:        DS.L     1              // varglob i a 32 bit
MAIN:     ...                     // inizio del codice di MAIN (LINK)
          MOVE.L   #N, I          // inizializza varglob i
DO_WHILE: SUBI.L   #1, I          // decrementa varglob i
          MOVEA.L  #STRING, A0    // carica costante STRING in A0
          ADDA.L   I, A0          // calcola indirizzo di elem string [i]
          MOVE.L   A0, -(SP)      // impila param p di funz
          BSR      FUNZ           // chiama funzione funz
          MOVE.B   (SP)+, (A0)    // spila valusc funz e agg. string [i]
          ADDA.L   #3, SP         // abbandona 3 byte in cima alla pila
          CMPI.L   #0, I          // confronta varglob i
          BGT      DO_WHILE       // ripeti ciclo DO_WHILE
          ...                     // resto del codice di MAIN (UNLK e RTS)
```

*Osservazioni: qui si usa un solo registro, A0, e si sfruttano a fondo la (semi)ortogonalità e riusare i registri tra più statement o parti di statement; si noti come qui (e anche nella versione non ottimizzata) le istruzioni LINK, UNLK e RTS abbiano scarsa rilevanza poiché main, benché sia modellato come funzione, non ha né parametri né variabili locali, non salva registri e non restituisce niente; pertanto qui queste istruzioni non sono neppure indicate esplicitamente.*

| **codice 68000 dello statement 2 in** *funz* (domanda 4) | | | |
|---|---|---|---|
| FUNZ: | **LINK** | FP, #-5 | // AGGIUNGERE INGOMBRO VARLOC ! |
| P: | **EQU** | 8 | // parametro p AGGIUNGERE SPIAZZ. |
| A: | **EQU** | -4 | // varloc a AGGIUNGERE SPIAZZ. |
| C: | **EQU** | -5 | // varloc c AGGIUNGERE SPIAZZ. |
| | ... | | // parte iniziale di funz |
| | **SUBA.L** | #1, SP | // 1a spazio per valusc di randc |
| | **BSR** | RANDC | // 1b chiama funzione randc |
| | **MOVE.B** | (SP)+, D0 | // 1c spila valusc di randc |
| | **MOVE.B** | C(FP), D1 | // 2 carica varloc c |
| | **ADD.B** | D1, D0 | // 3 addiziona |
| | **MULS** | #8, D0 | // 4 moltiplica |
| | **SUBI.B** | #2, D0 | // 5 sottrai |
| | **MOVE.B** | D0, C(FP) | // aggiorna varloc c |
| | ... | | // parte finale di funz |
| | | | |
| | **ADD.B** | C(FP), D0 | // ottimizza 2 e 3 (e non usa D1) |
| | | | |
| | | | |

*Osservazioni: qui si usano due registri, D0 e D1; sono possibili ottimizzazioni con la (semi)ortogonalità.*

**albero sintattico dell'espressione nello statement 2 in** *funz* (domanda 4)