ESEMPI DI ASSEMBLY M68000

```
* Esercizio 1
*
* Scrivere un programma non segmentato in linguaggio macchina
* (simbolico), con sintassi nativa M6800, che rispetti la seguente
* specifica.
*
* Dati 7 numeri di tipo Word, memorizzati a partire dall'indirizzo
* $8800, scrivere un programma che ponga nel registro
* D3 la somma dei soli elementi di valore dispari e in D4 la somma
* dei soli elementi di valore pari.
*
* Es. 9, 11, 1, 4, 5, 7, 2

          ORG       $8800
N         DC        9,11,1,4,5,7,2

          ORG       $8000
START     CLR       D3        * accumulatore per num. dispari
          CLR       D4        * accumulatore per num. pari
          MOVE      #6, D0    * contatore per 7 iterazioni (da 6 a 0)
          MOVEA.L   #N, A0    * indirizzo primo valore della sequenza
LOOP      MOVE      (A0)+, D1
          BTST      #0, D1    * test sul bit meno significativo
          BNE       DISPARI
          ADD       D1, D4
          BRA       SUCC
DISPARI   ADD       D1, D3
SUCC      SUBQ      #1, D0    * il suo assemblaggio occupa una sola
                                * parola per entrambi codice operativo e
                                * primo operando (in [-4, +3])
          BPL       LOOP
          STOP      #$2000    * load SR:=$2000 (set bit S on),
                                * stop the CPU and wait for an interrupt

          END START
```

```
* Esercizio 2
*
* Scrivere un programma non segmentato in linguaggio macchina
* (simbolico), con sintassi nativa M6800, che rispetti la seguente
* specifica.
*
* Contare il numero di volte in cui compare il carattere 'A'
* all'interno di una stringa e porre tale numero nel registro D3.


          ORG  $8800
S         DC.B 'PROGRAMMA DI PROVA',0
X         DC.B 'A'

          ORG  $8000
START     CLR.L     D3         * ind. diretto (da reg.) dell' operando

          MOVE.B    X, D0      * ind. assoluto (o diretto da mem.)
                               * del 1mo op.: D0 <- [X]

          MOVEA.L   #S, A0     * ind. immediato del 1mo op.

LOOP      MOVE.B    (A0)+, D1  * ind. indiretto da reg. con auto inc.
                               * D1 <- [[A0]]e poi A0 <- [A0] + 1

          TST.B     D1
          BEQ       ENDLOOP
          CMP.B     D0, D1
          BNE       NOTCOUNT
          ADDQ      #1, D3
NOTCOUNT  BRA       LOOP
ENDLOOP   STOP      #$2000

          END START
```

```
* Esercizio 3
*
* Scrivere un programma non segmentato in linguaggio macchina
* (simbolico), con sintassi nativa M6800, che rispetti la seguente
* specifica.
*
* Convertire una stringa in maiuscolo, sottraendo 32 al codice ASCII
* delle lettere minuscole.
* La stringa sia terminata da uno zero.

*
* Dichiarazione delle variabili
*
          ORG       $8800
STRINGA   DC.B      'Oggi, 29 Maggio',0


          ORG       $8000
INIZIO    MOVE.L    #STRINGA,A0    * Mette l'indirizzo della stringa in A0
CICLO     TST.B     (A0)           * Legge il carattere corrente
          BEQ       FINECICLO      * Se e' il terminatore esce dal ciclo
          CMP.B     #'a',(A0)      * Se e' minore 'a', non convertire
          BLT       NONCONV
          CMP.B     #'z',(A0)      * Se e' maggiore di 'z', non convertire
          BGT       NONCONV
          SUB.B     #32,(A0)       * Converte il valore
NONCONV   ADDA      #1,A0          * Incrementa il puntatore
          BRA       CICLO          * e ricomincia il ciclo
FINECICLO STOP      #$2000         * Termina l'esecuzione

          END       INIZIO
```

```
* Esercizio 4
*
```

```
* Scrivere un programma non segmentato in linguaggio macchina
* (simbolico), con sintassi nativa M6800, che rispetti la seguente
* specifica.
*
* Siano dati due byte agli indirizzi di memoria A e B. Si valuti il
* numero di bit omologhi uguali di A e B e si ponga tale numero alla
* locazione di memoria RES. Usare l'istruzione assembly: EOR.X s, d
* ( xor logico: d <- s xor d
*   s: un registro dati
*   d: non ammette indirizzamento diretto con reg. indirizzi )
*

* Area Dati
          ORG       $8800
A         DC.B      %10100110  * primo byte
B         DC.B      %01101110  * secondo byte
RES       DS.B      1          * area di storage del risultato

* Area Istruzioni
          ORG       $8000
START     MOVE.B    A, D0      * Sposta il primo valore in D0
          MOVE.B    B, D1      * Sposta il secondo valore in D1
          EOR.B     D0, D1     * Realizza un OR esclusivo bit a bit
                               * tra D0 e D1 e pone il risultato in D1

          MOVE.B    #7, D2     * Inizializza il registro contatore D2 a 7
                               * (conteggio da 7 a 0 compreso)

          CLR.L     D3         * Azzera il registro (accumulatore) D3
                               * destinato a contenere il risultato

LOOP      BTST      D2, D1     * Controlla il D2-esimo bit di D1
          BNE       DEC        * Se è 1 salta a DEC
          ADDQ      #1, D3     * altrimenti incrementa D3:
                               *            i bit omologhi erano uguali
DEC       DBRA      D2, LOOP   * Decrementa D2 e salta se D2 >= 0

* Le istruzioni DBcc eseguono la valutazione solo quando la condizione cc
* e' FALSA.
* DBRA e' equivalente a DBF, quindi il significato dell'istruzione e'
* quello di decrementare il reg. al 1mo operando e se il risultato e' >=0
* il PC viene incrementato con il valore numerico della distanza tra
* [PC] e l'etichetta al 2ndo operando.

          MOVE.B    D3, RES    * Sposta il risultato in memoria
          STOP      #$2000

          END       START
```

RIEPILOGO VELOCE DELLE ISTRUZIONI
PER IL CONTROLLO DI FLUSSO IN ASSEMBLY M68000

Posizionamento dei flag - Istruzioni di comparazione e test

Nel 68000, così come nella maggioranza dei processori, i bit-flag sono posizionati in conseguenza dell'esecuzione di quasi tutte le istruzioni di trasferimento e di calcolo (vedi tutte le tabelle dei codici). Le regole per il posizionamento dei flag sono spesso ovvie; in particolare, per le operazioni aritmetiche si ha:

· Z=1 se il risultato è nullo,
· N=1 se il risultato è negativo,
· C=1 se c'è un riporto (o prestito) uscente,
· V=1 se c'è un overflow in aritmetica dei complementi,
· X=C

Così come in tutti i processori che fanno ricorso sistematicamente ai flag, il 68000 possiede altresì apposite istruzioni di *comparazione*, che comparano due operandi aritmetici, e di *test* che posizionano i flag in funzione di un unico operando.

Comparazione aritmetica

L'istruzione di comparazione fondamentale:

- CMP G,D

equivale a SUB G,D, con l'unica differenza che la quantità D-G non è memorizzata in D. I flag sono posizionati coerentemente con il valore della differenza D -G, e quindi:

· Z=1 se D = G;
· N=1 se il bit più significativo (bit segno) del risultato della sottrazione D-G è 1;
· C=1 per numeri unsigned indica che D < G;
· V=1 se D e G hanno segno opposto (cioè D e –G hanno lo stesso segno) ed il risultato della sottrazione D-G ha segno opposto a D;
· X viene lasciato inalterato.

Analoghe sono le istruzioni:

- CMPA G, A (vedi SUBA)
- CMPI im,G (vedi SUBI)
- CMPM Ma,Mb (vedi SBCD)

La Tabella seguente riporta, a seconda della relazione esistente tra i valori di G e D, sia per il caso signed che per il caso unsigned, le combinazioni di valori dei flag che si ottengono in seguito all'esecuzione dell'istruzione CMP G,D.

$$x \oplus y = (x \wedge \overline{y}) \vee (\overline{x} \wedge y)$$

Il simbolo ≡ indica invece la relazione binaria "equivale

$$x \equiv y = (x \wedge y) \vee (\overline{x} \wedge \overline{y})$$

	caso UNSIGNED	caso SIGNED
D=G	Z=1	Z=1
D<G	C=1	N⊕V=1
D>G	C=0 **and** Z=0	Z=0 **and** N≡V=1
D<=G	C=1 **or** Z=1	Z=1 **or** N⊕V=1
D>=G	C=0 **or** Z=1	N≡V=1

Test

Esistono nel 68000 due classi di istruzioni di test, l'una che opera su un bit, e l'altra su un dato di 8, 16 o 32 bit.

Il test su un bit avviene con l'istruzione:

- BTST im,G

che posiziona il flag Z in funzione del valore del bit *im*-esimo di G. Le istruzioni BCLR, BSET, BCHG operano come BTST ma inoltre azzerano, pongono ad 1 o complementano rispettivamente il bit sul quale è stato fatto il test

c'e' anche:

- BTST Dj, Di

posiziona il flag Z in funzione del valore del bit [*Dj*]-esimo di Di.

Il test su un byte, su una word o una longword avviene con l'istruzione:

- TST G

che posiziona i flag N e Z in funzione del valore di G. L'istruzione:

- TAS G

opera come TST ma solo su un dato di tipo byte, ed inoltre pone ad 1 il bit più significativo (msb, *most significative bit*) di G.

Istruzioni di salto

Si ricorda che:
· Per salto si intende un riposizionamento del contatore di programma ad un determinato valore a: *PC:=a*
· Il salto è *assoluto* se l'operando O dell'istruzione esprime in modo direttamente il valore dell'indirizzo cui saltare: *PC:= O*
· Il salto è *relativo* se l'operando O lo esprime come incremento da dare a PC: *PC:= PC+O*
· Il salto è *condizionato*, se avviene solo se una assegnata condizione logica è vera: *if …then PC:=…*
· Il salto è *incondizionato* se avviene comunque.

Salti incondizionati

Le istruzioni di salto incondizionato sono sia assolute (*jump*) che relative (*branch*).

Il salto assoluto:
- JMP M
provoca il salto all'indirizzo di memoria M specificato nell'operando G con le classiche tecniche del 68000 (sono ovviamente esclusi i modi di indirizzamento d iretti a registro, né hanno senso il post- o il pre-decremento). Ad esempio:

- JMP $7800: salto diretto all'indirizzo 780016
- JMP ciclo: salto diretto all'indirizzo associato all'etichetta "ciclo"
- JMP (A3): salto indiretto all'indirizzo contenuto in A3

Il salto relativo:
- BRA *dest*
provoca il salto all'indirizzo di memoria PC+*disp*, ove PC è il valore del Program Counter dopo che è stato incrementato di 2 nella fase di fetch , e *disp* è un displacement che viene calcolato dall'assemblatore come differenza tra l'indirizzo *dest* (operando dell'istruzione assembler) e (PC+2). Il displacement *disp* può essere un dato ad 8 oppure a 16 bit, espresso in aritmetica dei complementi, che, prima di essere addizionato a (PC+2), viene automaticamente esteso nel segno. Nel primo caso, l'istruzione occupa 2 byte (il displacement è inserito nella stessa word che contiene il codice operativo), mentre nel secondo caso l'istruzione occupa 4 byte (il displacement è codificato in una word aggiuntiva). Dato un indirizzo di destinazione *dest*, l'assemblatore automaticamente sceglie una forma di codifica tra le due possibili, con il seguente criterio:

- · se *dest*-(PC+2) è compreso in [-128,126] viene scelta la forma di salto "corto", cioè con displacement ad 8 bit;
- · altrimenti, viene scelta la forma di salto "lungo", cioè con displacement a 16 bit.

Salti condizionati

Diverse sono in generale le tecniche per i salti condizionati. Nel 68000, essi si basano esclusivamente sull'analisi dei flag di condizione contenuti nel registro CCR. Il salto è inoltre solo di tipo relativo (branch) ed
assume la forma:

- **Bcc** ***dest***

o ve cc specifica in qualche modo il valore di CCR:

if cc then PC:= PC+*disp*

In particolare cc viene espresso in assembler attraverso una sigla che esprime mnemonicamente il significato della condizione da testare (esempio, GE= maggiore o eguale, EQ= Eguale e così via) mentre a livello di linguaggio macchina cc viene espresso in un codice a 4 bit nel campo cc dell'istruzione. Nella tabella, riportata nella prossima pagina, sono listate le condizioni cc nella simbologia di ASM68K, in linguaggio macchina e nella semantica.
Si ricorda che cc è posizionato dalle istruzioni precedenti quella di salto, tipicamente da una istruzione CMP O1,O2 che calcola R=O2-O1 e posiziona i flag di conseguenza. Pertanto le sigle GT, LT, etc.vanno lette come O2>O1, O2<O1, etc. Il significato delle espressioni logiche in ultima colonna è immediato per le condizioni CC, CS, NE, EQ, VC, VS, VC, PL, MI. Per quanto attiene GE, si ricorda che nell'aritmetica dei complementi, in caso di overflow positivo il bit-segno del risultato è 1 e quindi N=1 e viceversa, in caso di overflow negativo; il
fatto dunque di $O2 \geq O1$ va individuato attraverso i flag con due casi: quello in cui non vi sia stato overflow (V) e sia $R \geq 0$ (N) oppure quello in cui, essendovi stato l'overflow (V) risulti R<0 (N). All'opposto LT: O2<O1 se il risultato è negativo (N) senza overlow (V) oppure positivo ma con overflow. GE è come GT, ma deve escludere il caso che sia R=0 (Z), LE ne è il negato. Infine, HI e LS sono utilizzate per il confronto di numeri unsigned: la prima è vera se e solo se non è 1 né C né Z (cioè O2>O1); la seconda è vera se e solo se o C o Z sono 1, (cioè $O2 \leq O1$).

ASM	mnemonico	semantica	macchina	condizione
HI	Higher	maggiore	0010	$\overline{C} \wedge \overline{Z}$
LS	Lower or the Same	minore o uguale	0011	$C \vee Z$
CC	Carry Clear	C=0	0100	$\overline{C}$
CS	Carry Set	C=1	0101	C
NE	Not Equal (to zero)	Z=0	0110	$\overline{Z}$
EQ	Equal (to zero)	Z=1	0111	Z
VC	oVerflow Clear	V=0	1000	$\overline{V}$
VS	oVerflow Set	V=1	1001	V
PL	Plus	positivo	1010	$\overline{N}$
MI	Minus	negativo	1011	N
GE	Greater than or Equal	$\geq$	1100	$(N \wedge V) \vee (\overline{N} \wedge \overline{V})$ ovvero $N \equiv V$
LT	Less than	$<$	1101	$(N \wedge \overline{V}) \vee (\overline{N} \wedge V)$ ovvero $N \oplus V$
GT	Greater than	$>$	1110	$(N \wedge V \wedge \overline{Z}) \vee (\overline{N} \wedge \overline{V} \wedge \overline{Z})$ ovvero $\overline{Z} \wedge (N \equiv V)$
LE	Less than or Equal	$\leq$	1111	$Z \vee (N \wedge \overline{V}) \vee (\overline{N} \wedge V)$ ovvero $Z \vee (N \oplus V)$

Istruzione DBcc

Allo scopo di rendere più efficiente la programmazione di cicli, il processore 68000 è stato dotato dell'istruzione:

- DBcc D,*dest*

la quale corrisponde alla seguente sequenza di micro-operazioni:

if (not cc) then
begin
D := D - 1;
if (D = -1) then
PC := PC + 2; { esegui l'istruzione seguente }
else
PC := PC + *disp*; { esegui l'istruzione all'indirizzo *dest* }
end
else
PC := PC + 2; { esegui l'istruzione seguente }

L'istruzione DBcc è impiegata per creare un ciclo che termina quando sia verificata la condizione cc oppure il contatore di iterazioni D abbia raggiunto il valore –1. In aggiunta ai codici di condizione cc della tabella alla pagina precedente, per l'istruzione DBcc è consentita la variante con cc=false, che in assembler è associata
ai due codici operativi equivalenti DBF (*Decrement And BRanch on False*) e DBRA (*Decrement And BRanch Always*). Il segmento di programma che segue illustra l'uso dell'istruzione DBEQ per realizzare la ricerca della prima occorrenza di un carattere (memorizzato in D0) all'interno di un'area di memoria.

```
SEARCH    LEA.L    BUFFER,A0  A0 punta all'inizio dell'area
          MOVE.W   #BUFSIZE,D1  D1 := dimensione di BUFFER (>0)
LOOP      CMP.B    (A0)+,D0  confronta un byte
          DBEQ     D1,LOOP  esci se (M[(A0)]==D0 or D1==-1)
```

Trasferimenti a blocchi

Una particolare istruzione, MOVEM, consente di trasferire un blocco di locazioni di memoria in un insieme specificato di registri o viceversa: per i dettagli si rinvia ai manuali. Utilizzando il modo di indirizzamento indiretto con pre-decremento su SP, l'istruzione MOVEM salva sulla cima dello stack il valore di un insieme di registri D ed A specificato come operando sorgente:

- MOVEM.L D0-D4/A0/A2-A4,-(SP)

L'operazione duale si realizza impiegando il modo di indirizzamento indiretto con post-incremento su SP:

- MOVEM.L (SP)+,D0-D4/A0/A2-A4

L'istruzione MOVEM è tipicamente impiegata all'entrata in una subroutine, per "salvare" il contenuto dei registri del processore, in modo che lo si possa ripristinare al momento di ritornare al programma chiamante.

Stop e Nop

L'istruzione:
- STOP *im*

carica il valore *im* nel registro di stato SR, incrementa PC in modo che esso punti alla istruzione successiva, e pone il processore in uno stato di attesa di un interrupt. Quando un device esterno produce una richiesta di interruzione, il processore serve l'interruzione e, successivamente, prosegue eseguendo l'istruzione
seguente la STOP.

L'istruzione No-Operation:
- NOP

non compie alcuna operazione in fase execute. L'esecuzione dell'istruzione NOP richiede comunque che il processore carichi l'istruzione dalla memoria in fase fetch. NOP è talvolta usata quando si effettua il debugging di un codice oggetto e si vuole eliminare un'istruzione senza dover ricompilare l'intero codice: un'istruzione lunga *n* word è sostituita con *n* istruzioni NOP.