



**L'Assembler
per il QL**

C. Opie

L'Assembler per il QL

L'Assembler per il QL

C. Opie

McGRAW-HILL Book Company GmbH

Amburgo · New York · St Louis · San Francisco · Auckland · Bogotá
Città del Guatemala · Città del Messico · Johannesburg · Lisbona · Londra
Madrid · Montreal · Nuova Delhi · Panama · Parigi · San Juan · San Paolo
Singapore · Sydney · Tokyo · Toronto

Ogni cura è stata posta nella creazione, realizzazione, verifica e documentazione dei programmi contenuti in questo libro. Tuttavia né l'Autore né la McGraw-Hill Book Co. possono assumersi alcuna responsabilità derivante dall'implementazione dei programmi stessi, né possono fornire alcuna garanzia sulle prestazioni o sui risultati ottenibili dal loro uso, né possono essere ritenuti responsabili di danni o benefici risultanti dall'utilizzo dei programmi. Lo stesso dicasi per ogni persona o società coinvolta nella creazione, nella produzione e nella distribuzione di questo libro.

Titolo originale: *QL Assembly Language Programming*
Copyright © 1984 McGraw-Hill Book Company (UK) Ltd.

Copyright © 1985 McGraw-Hill Book Co. GmbH
Lademannbogen 136
D 2000 Hamburg 63, RFT

I diritti di traduzione, di riproduzione, di memorizzazione elettronica e di adattamento totale e parziale con qualsiasi mezzo (compresi i microfilm e le copie fotostatiche) sono riservati per tutti i paesi.

Realizzazione editoriale: EDIGEO srl, via del Lauro 3, 20121 Milano
Traduzione: Paolo Lamponi
Grafica di copertina: Valentina Boffa
Composizione e stampa: Litovelo, Trento

ISBN 88-7700-029-5

1^a edizione novembre 1985

QL, QDOS, ZX Microdrive, SuperBASIC, ZX Spectrum, ZX Net, QLUB sono marchi registrati della *Sinclair Research Ltd.*; QL Quill, QL Abacus, QL Easel, QL Archive sono marchi registrati della *Psion Ltd.*

Indice

Prefazione 11

Introduzione 13

- Vista di insieme 13
- Come usare questo libro 14
- Per stuzzicare l'appetito 15

Parte Prima - LA MPU 68000

Capitolo 1 Il microprocessore 68000 21

- 1.1 Modi operativi 21
- 1.2 I registri del 68000 22
 - Registri dati 22
 - Registri indirizzi e puntatori allo stack 23
 - Registro di stato 23
- 1.3 Uso della memoria 24
- 1.4 Passaggio tra i modi supervisore e utente 25

Capitolo 2 Istruzioni e modi di indirizzamento del 68000 27

- 2.1 Modi di indirizzamento 28
 - Indirizzamento relativo al contatore di programma 29
 - Operandi sorgente e destinazione 30
- 2.2 Codici di condizione 31
- 2.3 Trattamento dei flag dei codici di condizione 32
- 2.4 Mnemonici alternativi 32
- 2.5 Elenco delle istruzioni 68000 33

Parte Seconda - PROCEDURE DI SISTEMA DEL QL

Capitolo 3 Il QDOS 65

- 3.1 Mappa della memoria di sistema 65
 - Mappa dello spazio degli indirizzi 66
 - Mappa della RAM imposta dal QDOS 67
 - Area delle procedure residenti 67
 - Area dei programmi transienti 68
 - Area del SuperBASIC 69
 - Tabelle e variabili di sistema 69
 - Area dei canali e di heap 69
 - Blocchi di servizio dei sottosistemi di gestione dei file 70
- 3.2 Il bootstrap 70
- 3.3 Chiamate al sistema operativo e utility 71
 - Routine del QDOS 71
 - Routine di utility 72

Capitolo 4 Gestione delle risorse 75

- 4.1 Il multitasking nel QDOS 75
 - Stato dei job 76
 - Scheduling 76
 - Timeout 76
- 4.2 Uso dei registri del 68000 77

Capitolo 5 Allocazioni di input/output 101

- 5.1 Input e output ridirezionabile 101
- 5.2 Nomi di unità standard 102
- 5.3 Uso dei registri del 68000 103

Capitolo 6 Le operazioni di input/output 109

- 6.1 Timeout 109
- 6.2 Principi fondamentali 110
- 6.3 Blocchi di definizione dei canali associati allo schermo 111
 - ID di canale 111
 - Canali associati allo schermo 112
- 6.4 Colore 114
- 6.5 Uso dei registri del 68000 115

Capitolo 7 Routine di utility 179

- 7.1 Routine con accesso per mezzo di trap semplificato 179
- 7.2 Routine di utility del SuperBASIC 179
- 7.3 Uso dei registri del 68000 180

Capitolo 8 L'integrazione con il SuperBASIC 213

- 8.1 Comandi SuperBASIC per la programmazione in codice macchina 213
 - CALL 214
 - EXEC 214
 - LBYTES 215
 - PEEK 215
 - POKE 215
 - RESPR 216
 - SBYTES 216
 - SEXEC 217
- 8.2 Interfacciamento al SuperBASIC 217
- 8.3 L'ambiente SuperBASIC 218
 - Area buffer 218
 - Area dei programmi SuperBASIC 218
 - Tabella dei nomi 220
 - Lista dei nomi 221
 - Valori delle variabili 221
 - Tabella dei canali 222
 - Stack aritmetico 222
 - Stack di sistema 223
- 8.4 Implementazione di procedure in codice macchina 224
- 8.5 Creazione di nuove posizioni nella tabella dei nomi 224
 - Istruzioni di inizializzazione 225
- 8.6 Inizializzazione dei parametri 226
- 8.7 Acquisizione degli argomenti 227
- 8.8 Valori di funzione da restituire 228
- 8.9 Valori di parametri da restituire 228
- 8.10 Restituzione di stringhe 228
- 8.11 Particolari sull'uso dello stack aritmetico 229
- 8.12 TRAP # 4 229

Parte Terza - ESEMPI DI PROGRAMMAZIONE

Capitolo 9 Semplici programmi eseguibili 233

- 9.1 Esempio 1 — MESSG 234
- 9.2 Esempio 2 — SCELTA 235
- 9.3 Esempio 3 — OROL 235

Capitolo 10 La grafica 243

- 10.1 Organizzazione della memoria video 244
- 10.2 Esempio 1 — ORLANLG 245
- 10.3 Esempio 2 — PALLA 245

Capitolo 11 Estensione del SuperBASIC 255

- 11.1 Uso dei programmi 255
- 11.2 Esempio 1 — CURSORE 256
- 11.3 Esempio 2 — FUNZBAS 257
- 11.4 Esempio 3 — ARRAY 258
- 11.5 Esempio 4 — JOBS 258

Capitolo 12 Accesso casuale ai file 273

- 12.1 Uso del programma 273
- 12.2 Esempio — FILER 274

Parte Quarta - L'EDITOR/ASSEMBLER**Capitolo 13 Uso dell'editor 285**

- 13.1 Le window dell'editor 285
- 13.2 Modi dell'editor 287
- 13.3 Messaggio di aiuto 287
- 13.4 Introduzione del testo 288
- 13.5 Movimento del cursore 288
- 13.6 Cancellazione del testo 290
- 13.7 Il tasto ENTER 291
- 13.8 Opzioni del comando Esegue 291
 - Trova stringa 291
 - Cancella blocco 292
 - Va a linea 292
 - Inserisce file 292
- 13.9 Lettura di un file di testo 293
- 13.10 Salvataggio del testo corrente 294

Capitolo 14 L'Assembler del 68000 295

- 14.1 Funzionamento dell'Assembler 296
- 14.2 Sintassi delle linee Assembler 297
 - Label 297
 - Operatori e argomenti 297
 - Commenti 297
 - Lo pseudo operatore END 298
- 14.3 Simboli 298
 - Definizione nel programma sorgente (EQU) 298
 - Definizione al momento dell'assemblaggio (QRY) 299
- 14.4 Label 299
 - Label standard 299
 - Label temporanee 300
- 14.5 Espressioni 300

| | |
|---|---|
| Numeri | 301 |
| Semplici espressioni | 301 |
| 14.6 Definizione dei dati | 302 |
| DEFB | 303 |
| DEFW e DEFL | 303 |
| DEFS | 304 |
| 14.7 Determinazione dell'origine | 304 |
| 14.8 Assemblaggio su condizione | 305 |
| 14.9 Direttive | 306 |
| *Pagina e *Titolo | 306 |
| *Listato | 307 |
| *Numerazione | 307 |
| *Inclusione | 307 |
| 14.10 Mnemonici alternativi | 308 |
| 14.11 Messaggi di errore | 309 |
| Messaggi di errore di tipo generale | 309 |
| 14.12 Allineamento al margine di word (ALIGN) | 310 |
| Appendice A | Sommario del set di istruzioni del 68000 |
| A.1 | Modi di indirizzamento 311 |
| A.2 | Codici di condizione 311 |
| A.3 | Set di istruzioni del 68000 311 |
| Appendice B | Chiamate di sistema |
| | 317 |
| Appendice C | Codici di errore per il sistema QDOS |
| | 323 |
| Appendice D | Guida rapida ai comandi editor/Assembler |
| | 325 |
| D.1 | Guida all'uso dell'editor 325 |
| D.2 | Guida all'uso dell'Assembler 326 |
| Indice analitico | 327 |

Prefazione

La recente evoluzione della microelettronica ha generato un microcomputer di grande potenza dal costo contenuto: il Sinclair QL. Le caratteristiche di questo microcomputer sono tali da renderlo il capostipite di una nuova generazione. Il leggero ed elegante involucro nero del QL contiene un microprocessore della famiglia 68000, uno dei più avanzati microprocessori attualmente disponibili. Il QL permette il trattamento di dati a 32 bit, dispone di un set di chip appositamente progettati per ottenere il massimo dall'attuale tecnologia, è fornito di una memoria RAM da 128 K, di un sistema operativo multitasking, di due Microdrive per la memorizzazione permanente delle informazioni e di un flessibile sistema di I/O che comprende delle connessioni per rete locale. Ciò che rende questo microcomputer unico, oltre alla potenza dei suoi componenti elettronici, è il fatto che il suo prezzo non è superiore a quello dei microcomputer a 8 bit concorrenti.

Il QL ha, come dotazione standard, un potente ed estensibile SuperBASIC. Una caratteristica importante di questa estensibilità è che è possibile scrivere delle routine nel codice macchina del 68000 per poi fonderle nel SuperBASIC estendendone, così, il numero di comandi disponibili. Ovviamente, è anche possibile scrivere interi programmi in Assembler, ottenendo la massima velocità di esecuzione dei programmi applicativi. Questo libro è dedicato alla programmazione del QL in Assembler del 68000. Esistono molti buoni libri che introducono la programmazione in Assembler e non è il caso di riproporre qui tali argomenti. Il concetto alla base di questo libro è, quindi: "Supponendo di avere un libro che spiega in dettaglio il set di istruzioni del 68000, come posso usare il QL per creare dei programmi applicativi in Assembler che possano essere esegui-

ti su di esso?" È nostra speranza che un tale criterio ci abbia portato a realizzare un testo utile per programmatori, sistemisti, progettisti OEM, con probabili applicazioni anche nel campo della scuola. In questo libro non vengono descritte in dettaglio le istruzioni del 68000, ma i capitoli 1 e 2 e l'appendice A contengono la maggior parte delle informazioni utili sul set di istruzioni di questo microprocessore.

Un libro come questo non sarebbe completo se non permettesse di acquisire una esperienza pratica della teoria qui illustrata: a questo scopo, sono stati realizzati un editor di programmi e un Assembler/loader per 68000 che fanno da ideale complemento a questo testo. Come sarà spiegato nella Parte Quarta (che descrive in dettaglio l'uso del software), l'insieme dei programmi per lo sviluppo di software in Assembler 68000 supporta un completo ambiente di programmazione; l'Assembler vero e proprio, per esempio, ha delle caratteristiche che normalmente si possono trovare solo nelle più costose versioni per minicomputer. L'Assembler è disponibile separatamente su una cartuccia per Microdrive.

I miei più sinceri ringraziamenti per l'aiuto prestato nella realizzazione di questo libro vanno a John Watson, Tom Blackall, Liz Nemecek e Jenny Wright. Uno speciale ringraziamento va a Tony Tebby per il prezioso aiuto e i consigli di carattere tecnico, senza i quali questo libro non avrebbe potuto essere scritto. Come sempre, esprimo la mia gratitudine a mia moglie, la cui pazienza e capacità di sostenermi non sembrano avere limiti.

Colin N. Opie

Introduzione

“Una delle più belle cose del mondo è fare un viaggio...”

William Hazlitt

In questo libro esploreremo l'ambiente operativo del microcomputer QL; durante il viaggio sarà possibile fare molte nuove esperienze e per chiunque ci saranno opportunità di dare libero sfogo alla fantasia e all'inventiva. L'ambiente operativo del QL è basato su un kernel di procedure noto come QDOS. Oltre al QDOS sono presenti un certo numero di utility a cui si può accedere tramite dei vettori ben definiti in ROM. Le procedure QDOS e le routine di uso generale forniscono al programmatore in Assembler una quantità di strumenti di supporto che vanno dal semplice output di caratteri fino all'aritmetica in virgola mobile. Il processore principale del QL è un Motorola 68008; questo processore a 16 bit della nuova generazione è dotato di un'ottima architettura, caratterizzata da un ampio spazio di memoria indirizzata linearmente e da un set di istruzioni di estrema coerenza. Questo libro spiega come usare questo set di istruzioni all'interno dell'architettura del QL.

VISTA DI INSIEME

Questo libro è diretto in modo particolare al programmatore in Assembler ed è quindi logico introdurre l'architettura generale del microprocessore 68000, i suoi modi di indirizzamento e l'uso delle sue istruzioni.

Questi argomenti saranno trattati nella Parte Prima. Le istruzioni del 68000 non verranno descritte in dettaglio per due motivi: per prima cosa, includere tale materiale avrebbe reso il libro troppo esteso e costoso; in secondo luogo, sono già disponibili diversi testi adatti allo scopo (per esempio: G. Kane, *Il Manuale MC68000*, della McGraw-Hill). Lo scopo dei capitoli di questo libro dedicati al processore 68000 è di fornire un conciso sommario delle informazioni utili per la programmazione di questo processore.

La Parte Seconda, che comprende i capitoli 3-8, descrive in dettaglio il QDOS e le procedure di uso generale; queste procedure costituiscono i mattoni per la costruzione dei programmi applicativi Assembler. Il capitolo 8 descrive le opzioni disponibili nel caso si vogliano caricare ed eseguire programmi in codice macchina; viene anche spiegato come aggiungere al SuperBASIC delle procedure in Assembler che costituiscono delle estensioni del linguaggio vero e proprio.

La Parte Terza contiene quattro capitoli dedicati ad esempi di programma. I capitoli 9 e 10 contengono esempi di programmi eseguibili in modo indipendente: il capitolo 9 è costituito da numerosi programmi di utilità mentre il capitolo 10 è dedicato alla grafica. I capitoli 11 e 12 contengono esempi di programmi che costituiscono estensioni del SuperBASIC: il capitolo 11 contiene alcune procedure di uso generale mentre il capitolo 12 è dedicato totalmente al trattamento dei file con i Microdrive. I programmi illustrati, oltre a fornire delle procedure di utilità pratica completamente sviluppate, costituiscono anche un esempio di come caricare ed eseguire materialmente i programmi.

La Parte Quarta descrive l'editor screen oriented e l'Assembler/loader usati per creare i programmi Assembler descritti nella Parte Terza. I programmi di questo package sono facili da usare e permettono un approccio professionale alla programmazione in Assembler sul QL. L'ultima parte del libro contiene diverse appendici che servono come rapida guida alle informazioni di uso più comune.

COME USARE QUESTO LIBRO

Il modo di usare questo libro dipende fondamentalmente dal vostro livello di conoscenza delle tecniche di programmazione in Assembler: daremo per scontato che termini come registri, modi di indirizzamento, puntatori, vi siano già familiari. Se siete in grado di scrivere programmi in Assembler per altri processori (Z80, 6502 o altri) sarete in breve in grado di programmare anche il QL. Quelli che conoscono già l'Assembler del 68000 possono cominciare a leggere questo libro partendo direttamente dalla Parte Seconda.

Se conoscete l'Assembler di altri microprocessori ma non quello del

68000, troverete nei capitoli 1 e 2 e nell'appendice A tutte le informazioni su ciò che il 68000 è in grado di fare. Come abbiamo già detto, per conoscere in dettaglio ciascuna delle istruzioni del 68000 sarà necessario fare riferimento ad un altro testo dedicato all'argomento. Dopo aver assimilato le nozioni riguardanti la struttura generale e le possibilità del 68000, sarete in grado di utilizzare a fondo la Parte Seconda per scrivere, caricare ed eseguire programmi in Assembler.

PER STUZZICARE L'APPETITO

La programmazione in Assembler del QL richiede l'uso di un adatto pacchetto software; i programmi sviluppati in questo modo saranno solitamente collegati al SuperBASIC, estendendo così il linguaggio, oppure potranno essere eseguiti come programmi in codice macchina, in modo indipendente per mezzo del comando EXEC (vedi capitolo 8). I più semplici programmi in codice macchina possono anche essere caricati in memoria ed eseguiti per mezzo del comando CALL: è questo il metodo che useremo nell'esempio che segue al solo scopo di stuzzicarvi un po' l'appetito! Il programma seguente è una versione Assembler (prodotta con l'apposito programma McGraw-Hill descritto nella Parte Quarta) del comando di SuperBASIC RECOL. Questo comando accetta un numero di canale associato allo schermo seguito da 8 parametri di definizione del colore:

```
RECOL #n, c1, c2, c3, c4, c5, c6, c7, c8
```

Ciascun parametro del colore definisce il nuovo colore dei pixel che hanno come colore corrente, rispettivamente: nero, blu, rosso, magenta, verde, cyan, giallo e bianco. Per riscrivere questa procedura in Assembler in modo da poterla usare con il comando CALL, è necessario conoscere i meccanismi di passaggio dei parametri. Nel capitolo 8 troviamo che possono essere passati fino a 13 parametri sotto forma di 13 parole doppie contenute nei registri da D1 a D7 e da A0 ad A5 (nell'ordine) del 68000. Nel nostro esempio vengono passati nove parametri che, per quanto appena detto, troveranno posto nei registri da D1 a D7 e in A0 e A1. Oltre a ciò, è anche necessario conoscere le modalità di accesso alla appropriata routine di QDOS: in questo caso la routine che ci interessa è SD.RECOL (TRAP #3, D0=\$26). Una descrizione approfondita di questa routine verrà data nel capitolo 6.

Vediamo come viene eseguito il programma.

```

00030000          0003      org $30000
0004 ;
0005 ; Breve programma dimostrativo in Assembler.
0006 ; Usato insieme al SuperBASIC.
0007 ; Copyright (c) 1985 McGraw-Hill Book Co. GmbH
0008 ;
0009 ;
00030000 45FA0036 0010 demo: lea    dati(pc),a2    ; trova buffer
00030004 15420000 0011      move.b d2,0(a2)      ; memorizza tabella
00030008 15430001 0012      move.b d3,1(a2)
0003000C 15440002 0013      move.b d4,2(a2)
00030010 15450003 0014      move.b d5,3(a2)
00030014 15460004 0015      move.b d6,4(a2)
00030018 15470005 0016      move.b d7,5(a2)
0003001C 3408     0017      move.w a0,d2
0003001E 15420006 0018      move.b d2,6(a2)
00030022 3409     0019      move.w a1,d2
00030024 15420007 0020      move.b d2,7(a2)
00030028 224A     0021      move.l a2,a1          ; inizializza punt. ai dati
0003002A 363C0000 0022      move.w #0,d3          ; timeout = 0
0003002E 2041     0023      move.l d1,a0          ; canale 1
00030030 103C0026 0024      move.b #$26,d0        ; RECOLOUR
00030034 4E43     0025      trap    #3
00030036 4E75     0026      rts
0027 ;
0028 ; Area di lavoro
0029 ;
00030038 00       0030 dati:  defb    0
0031      defb    20
0032 ;
0033 end

```

Simboli:

```
00030038 DATI      00030000 DEMO
```

```

Errori:      0000
Byte liberi: 4F08
Codice (byte): 30040

```

Assembler terminato

La procedura QDOS richiede che gli otto parametri del colore siano predisposti in una tabella di byte: ciò significa che il contenuto dei registri che stiamo usando dovrà essere trasferito in una piccola area dati. A questo scopo, individuiamo la posizione effettiva dell'area dati di questo particolare programma e usiamo l'indirizzamento indicizzato per eseguire il trasferimento dei dati sotto forma di byte. Penserete: "Ma noi sappiamo dov'è l'area dati: parte dalla locazione \$00030038". In un certo senso ciò è vero, in quanto l'istruzione ORG stabilisce proprio questo, ma la maggior parte dei programmi in codice macchina del QL devono essere rilocabili. A pagina 18 viene mostrato un programma SuperBASIC che utilizza la routine in codice macchina di cui ci stiamo occupando: quando vie-

ne richiesto un po' di spazio in memoria per contenere la routine in codice macchina (per mezzo del comando RESPR), noi non possiamo sapere in anticipo dove il SuperBASIC lo ricaverà. Nell'esempio illustrato noi chiediamo semplicemente che ci vengano riservati 70 byte e il SuperBASIC ci restituisce l'indirizzo di partenza di tale area (nella variabile 'mc'). Il nostro programma, perciò, dovrà funzionare indipendentemente da dove sia stato caricato. L'istruzione LEA posta all'inizio del programma in Assembler viene usata con il modo di indirizzamento relativo al contatore di programma con spiazzamento e caricherà nel registro indirizzi indicato la effettiva posizione assoluta dell'inizio dell'area dati. Nel capitolo 2 viene descritto dettagliatamente questo modo di indirizzamento con la relativa sintassi per l'Assembler.

Quando tutti i parametri sono stati memorizzati nella tabella dati, tutto ciò che resta da fare è predisporre gli appropriati registri per la chiamata al QDOS ed eseguire l'istruzione TRAP #3. Un'istruzione RTS posta al termine del programma effettuerà il ritorno al SuperBASIC.

Esaminiamo, adesso, il programma SuperBASIC. All'inizio, il programma disegna tre cerchi di colore diverso; due cerchi sono colorati anche all'interno, mentre il terzo è solo una circonferenza. L'operazione successiva consiste nella memorizzazione della routine in codice macchina in un'area di memoria riservata. Il programma è lungo 56 byte, quindi un'area di 70 byte è più che sufficiente (visto che il buffer dati deve essere lungo solo otto byte). Le istruzioni DATA contengono i valori decimali corrispondenti ai codici operativi in esadecimale forniti dall'Assembler. Dopo che è stata eseguita questa inizializzazione, il programma entra in un piccolo loop infinito: le istruzioni che lo compongono provocano, ogni 10 secondi, l'inversione dei colori delle figure che compaiono sullo schermo. In SuperBASIC l'output dei programmi normalmente viene inviato al canale #1. La nostra subroutine in codice macchina prevede che la ID di canale sia passata come primo argomento nella lista di parametri dell'istruzione CALL. Come potete vedere esaminando le linee 170 e 190, questo parametro non è 1. I numeri di canale del Super BASIC non hanno alcuna corrispondenza con i valori di ID dei canali QDOS. Se non vengono effettuate riaperture dei canali associati allo schermo, l'effettiva corrispondenza tra canali di schermo SuperBASIC e ID di canale QDOS è:

| SuperBASIC # | ID QDOS | |
|--------------|-------------|----------|
| | Esadecimale | Decimale |
| 0 | \$00000 | 0 |
| 1 | \$10001 | 65537 |
| 2 | \$20002 | 131074 |

È importante ricordare che le ID del QDOS vengono alterate se viene effettuata una riapertura di un canale di schermo (per esempio, eseguendo

OPEN#1,...). Solitamente, quando si programma il QL in Assembler, la ID di un particolare canale viene determinata per mezzo di un apposito algoritmo; il capitolo 11 contiene una routine adatta allo scopo.

```
100 REMark Semplice programma dimostrativo
120 REMark Copyright (c) 1985 McGraw-Hill Book Co. GmbH
130 REMark PROGRAMMA PRINCIPALE
140 colori
150 locazione = RESPR(70) : memo_codice
160 PAUSE 200
170 CALL locazione,65537,7,6,5,4,3,2,1,0
180 PAUSE 200
190 CALL locazione,65537,0,1,2,3,4,5,6,7
200 GO TO 160
210 :
220 REMark ROUTINE
230 DEFine PROCedure colori
240 INK 0 : FILL 1 : CIRCLE 30,60,15
250 INK 4 : FILL 0 : CIRCLE 60,60,15
260 INK 6 : FILL 1 : CIRCLE 45,30,15
270 FILL 0
280 END DEFine
290 :
300 DEFine PROCedure memo_codice
310 RESTORE 350
320 FOR c = 0 TO 55 : READ n : POKE locazione + c, n
330 END DEFine
340 :
350 DATA 69,250,0,54,21,66,0,0,21,67,0,1
360 DATA 21,68,0,2,21,69,0,3,21,70,0,4
370 DATA 21,71,0,5,52,8,21,66,0,6,52,9
380 DATA 21,66,0,7,34,74,54,60,0,0,32,65
390 DATA 16,60,0,38,78,67,78,117
```

Val la pena di sottolineare che per programmare il QL in Assembler in modo ottimale è necessario utilizzare un adatto supporto software per l'Assembler. I programmi vengono normalmente uniti al SuperBASIC, costituendone delle estensioni, oppure vengono eseguiti come programmi indipendenti in codice macchina (*job*) sotto il controllo del QDOS; nel secondo caso, il programma può essere lanciato per mezzo del comando EXEC (SuperBASIC) oppure sfruttando le procedure QDOS di creazione e attivazione dei job. Il comando SuperBASIC CALL è semplice da usare, ma non è molto potente; il suo uso dovrebbe essere limitato all'esecuzione di piccole routine di prova o al collegamento iniziale di un certo numero di procedure SuperBASIC.

Parte Prima

La MPU 68000

Il microprocessore 68000

1

Il cuore del QL è un microprocessore della famiglia Motorola 68000: il 68008. Per quanto riguarda il software, il 68008 è dotato di tutte le proprietà caratteristiche della famiglia 68000; la differenza principale di questo processore rispetto ai membri più potenti della famiglia è che il package in cui esso è contenuto è più piccolo e permette di realizzare un bus dati di soli 8 bit. Come conseguenza, il *throughput* di questo processore viene ridotto a causa del sovraccarico introdotto nell'indirizzamento della memoria. Questo dettaglio non deve preoccupare chi intende programmare il QL in Assembler, in quanto l'architettura avanzata a 16/32 bit di questo processore fa del 68008 uno dei microprocessori più potenti attualmente disponibili. Un'altra limitazione del 68008 è che solo 20 dei 32 bit usati per gli indirizzamenti vengono presentati ai piedini del chip: ciò significa che lo spazio di indirizzi è limitato a 1 megabyte (se 1 megabyte può essere definito un limite!). Prima di occuparci di come usare questo processore all'interno del QL vediamo le caratteristiche generali.

1.1 Modi operativi

Il 68000 può funzionare in due modi operativi diversi detti *modo utente* e *modo supervisore*. Un flag nel registro di stato stabilisce in ogni momento quale sia il modo operativo corrente. Alcune istruzioni (ad esempio STOP) non possono essere eseguite con il 68000 in modo utente; se viene

tentata l'esecuzione di tali istruzioni in modo utente, il processore genererà un'eccezione di violazione di privilegio.
Quando il processore è in modo utente, le operazioni che riguardano lo stack fanno riferimento allo stack pointer di utente (USP); simmetricamente, quando il processore è in modo supervisore viene utilizzato lo stack pointer di supervisore.

1.2 I registri del 68000

Il 68000 è dotato di 18 registri a 32 bit e di un registro di stato a 16 bit (vedi Figura 1.1). Il set di registri a 32 bit è suddiviso in otto registri dati, sette registri indirizzi, due puntatori allo stack e un contatore di programma.

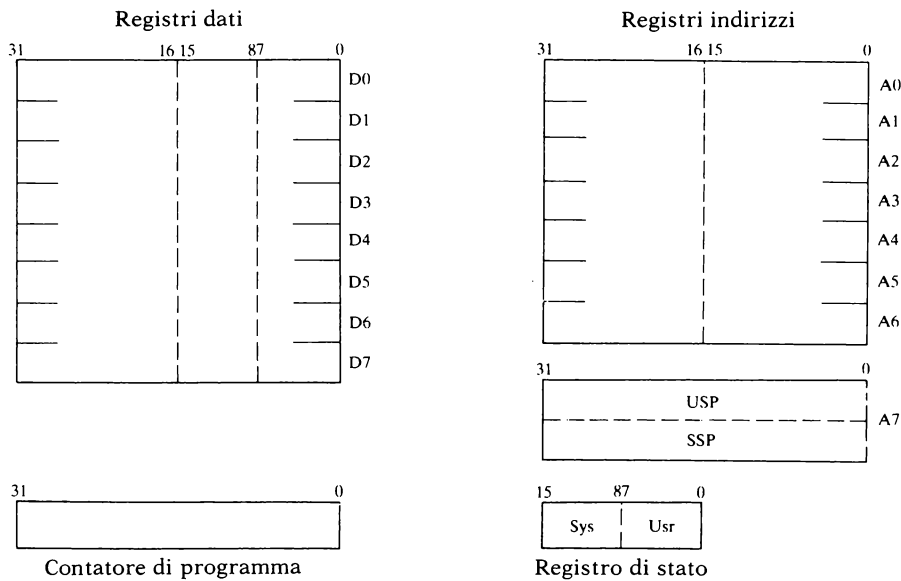


Figura 1.1 Registri del 68000

REGISTRI DATI

Gli otto registri dati vengono indicati con i simboli D0 ... D7. I tipi di dati trattati dalle operazioni che utilizzano questi registri sono: bit, BCD (nibble), byte, word (16 bit) o long word (32 bit). Quelle istruzioni che preve-

dono l'uso di uno o più registri dati come operandi consentono di specificare qualsiasi registro dati; in pratica, ciò significa che ogni registro dati può essere usato sia come accumulatore che come registro indice, registro dati generico o contatore di loop.

Questo approccio alla allocazione dei registri del processore si rivela molto flessibile e, insieme ad altri fattori, permette di programmare facilmente il 68000 in modo efficiente.

REGISTRI INDIRIZZI E PUNTATORI ALLO STACK

I sette registri indirizzi vengono indicati con i simboli A0 ... A6. Anche i due puntatori allo stack vengono considerati come registri indirizzi e il simbolo che identifica entrambi è A7. Gli mnemonici alternativi per i due puntatori allo stack sono USP (*User Stack Pointer*, puntatore allo stack di utente) e SSP (*Supervisor Stack Pointer*, puntatore allo stack di supervisore).

Lo stato in cui si trova il processore (cioè utente o supervisore) è indicato da un flag nel registro di stato e in ognuno dei due stati viene usato il puntatore allo stack corrispondente. Le operazioni che fanno riferimento ai registri indirizzi trattano solo i tipi di dati word (16 bit) e long word (32 bit). In altre parole, un registro indirizzi non può essere usato come sorgente o destinazione in istruzioni logiche o che operano su bit e byte. Se un registro indirizzi viene utilizzato come operando destinazione, l'informazione memorizzata sarà sempre una long word e, se necessario, l'operando sorgente subirà un'estensione di segno prima di essere utilizzato. I registri indirizzi vengono generalmente usati per contenere e manipolare indirizzi piuttosto che dati. Essi possono anche essere utilizzati come registri indice.

È importante notare che, poiché i puntatori allo stack sono in effetti il registro indirizzi A7, qualsiasi modo di indirizzamento valido per i registri indirizzi è anche valido per i puntatori allo stack. Ciò significa che i modi di indirizzamento del 68000 che fanno riferimento ai puntatori allo stack sono molto più versatili di quelli di molti altri processori.

REGISTRO DI STATO

Il registro di stato del 68000 è un registro a 16 bit diviso in due byte distinti. I due byte corrispondono al byte di stato di sistema (bit da 8 a 15) e al byte di stato di utente (bit da 0 a 7). Il byte di stato di sistema può essere modificato solo quando il processore è in modo supervisore. Esistono due mnemonici che definiscono il registro di stato: il primo è CCR e indica il solo byte di utente che contiene i codici di condizione; le istru-

zioni che fanno riferimento a questo mnemonico accedono a soli 8 bit di dati. L'altro mnemonico è SR e indica l'intero registro di stato; le istruzioni che lo utilizzano, possono essere eseguite solo quando il 68000 è in modo supervisore.

La Figura 1.2 mostra la disposizione dei flag all'interno del registro di stato. I bit di Carry (C), Zero (Z), Negativo (N) e Overflow (O) sono i flag di condizione presenti in tutti i processori. C'è anche un flag di Estensione (E) che viene sempre settato allo stesso valore del flag di Carry, se questo viene modificato durante l'esecuzione di particolari istruzioni. Il flag di Estensione viene usato dalle operazioni aritmetiche in precisione multipla. Il capitolo 2 descrive la funzione di questi flag all'interno delle istruzioni.

I tre bit meno significativi del byte di stato di sistema costituiscono la maschera di disabilitazione dell'interrupt (IDM, *Interrupt Disable Mask*) del 68000: essi mettono a disposizione sette livelli di priorità dell'interrupt e, per ogni interrupt di priorità diversa, la sequenza di esecuzione delle istruzioni viene modificata per mezzo di un vettore di interrupt. La maschera contenuta nel registro di stato specifica la priorità al di sotto della quale gli interrupt vanno ignorati: se, per esempio, la maschera ha la configurazione 011 (3 in decimale), il processore ignora gli interrupt con priorità da 1 a 3.

Osserviamo che il 68008 ha solo tre livelli di interrupt (i livelli 2, 5 e 7). Nel QL un interrupt di livello 5 ha carattere transitorio e genera sempre un interrupt di livello 7 (non mascherabile) dopo 20 millisecondi.

Il flag di Supervisore (S) stabilisce se il 68000 sta funzionando in modo supervisore oppure no: se il bit è posto a 1, il processore è in modo supervisore. L'ultimo flag, ma non per questo il meno importante, è il flag di Trace (T): esso abilita il processore al funzionamento in modo single step e permette ai debugger di controllare l'esecuzione delle istruzioni.

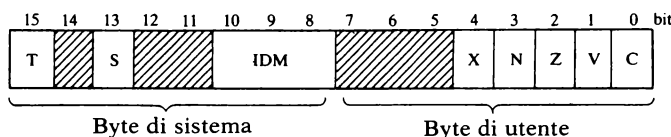


Figura 1.2 Registro di stato del 68000

1.3 Uso della memoria

Il 68008 può indirizzare direttamente fino a 1 megabyte di memoria e ciò richiede un bus indirizzi da 20 bit. Gli indirizzi, perciò, possono essere rappresentati da numeri composti da cinque cifre esadecimali, da

#00000 a #FFFFFF. Per accedere a un certo byte di dati in memoria, può essere specificato un indirizzo qualunque. L'accesso a una word (cioè 16 bit) o a una long word (32 bit) ha alcune limitazioni: questi tipi di dati richiedono che gli indirizzi specificati siano solo pari, cioè #00000, #00002 e così via fino a #FFFFFFE. Quando l'accesso a memoria riguarda una word, il byte più significativo della word si trova all'indirizzo pari, mentre il byte meno significativo si trova all'indirizzo dispari immediatamente seguente (vedi Figura 1.3). L'indirizzamento di long word è simile a quello delle word in quanto può essere visto come due distinti accessi a word. La word più significativa si trova all'indirizzo (pari) più basso, seguita immediatamente dalla seconda word (anch'essa ad un indirizzo pari) della long word.

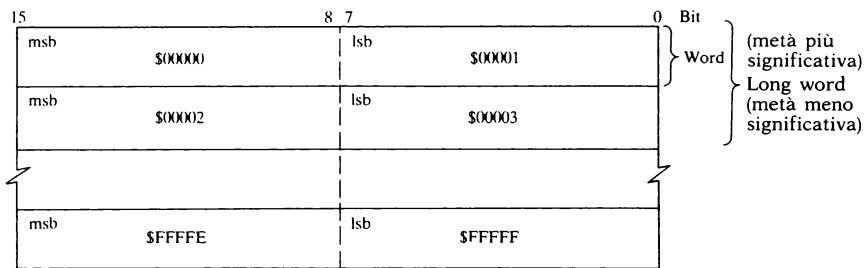


Figura 1.3 Uso della memoria

1.4 Passaggio tra i modi supervisore e utente

Sapere che il 68000 può funzionare in due modi diversi non è particolarmente utile se non si sa come passare da un modo all'altro. Quando il 68000 viene resettato (per esempio all'accensione del sistema) gli otto byte che si trovano all'estremità inferiore della memoria vengono caricati nel puntatore allo stack di supervisore e nel contatore di programma, e l'esecuzione comincia in modo supervisore.

Dato che alla partenza il processore si trova nello stato supervisore, la prima cosa che ci interessa sapere è come passare al modo utente. Niente di più facile! Qualsiasi istruzione in grado di modificare lo stato del flag S nel registro di stato è anche in grado di far passare il processore dal modo supervisore al modo utente. Un esempio di tali istruzioni è RTE (*ReTurn from Exception*, ritorno da eccezione): questa istruzione carica nel registro di stato la word che si trova sullo stack e carica nel contatore di programma le successive due parole lette dallo stack. Se la parola caricata nel registro di stato pone il flag S a zero, ciò provoca auto-

maticamente il passaggio al modo utente. Se il flag S rimane a 1, l'esecuzione procede in modo supervisore.

Quando il processore è in modo utente, il passaggio a modo supervisore avviene al verificarsi di una forma qualsiasi di eccezione. Elenchiamo, di seguito, un certo numero di condizioni che prevedono il trattamento delle eccezioni:

1. Violazione di indirizzamento. Si è tentato l'accesso a una word o long word utilizzando un indirizzo dispari.
2. Violazione di istruzione privilegiata. È stata eseguita una istruzione privilegiata.
3. Codice operativo illegale o non implementato. L'istruzione eseguita non era compresa nel set di istruzioni del 68000.
4. Esecuzione di istruzione TRAP. Tutte le istruzioni TRAP vengono trattate come eccezioni interne.
5. Si è verificata una eccezione dovuta a una condizione di errore per le istruzioni TRAP, CHK, DIVS, DIVU (per esempio, divisione per zero).
6. Trace attivo. Se il flag T nel registro di stato è posto a uno, si verifica una eccezione al termine dell'esecuzione di ogni istruzione.
7. Interrupt esterno. È stato ricevuto un interrupt avente uno dei sette (tre sul QL) livelli di priorità.
8. Reset. Il processore 68000 è stato fisicamente resettato.
9. Errore di bus. Si è verificato un errore sul bus fisico degli indirizzi o dei dati del processore 68000.

Le ultime due condizioni di eccezioni (reset e errore di bus) non sono granché utili per chi scrive programmi applicativi. Il sistema più semplice per passare al modo supervisore prevede l'uso dell'istruzione TRAP.

Istruzioni e modi di indirizzamento del 68000

2

Nel 1971 la *Intel* introdusse il 4004, che disponeva di 46 istruzioni. In seguito la stessa società mise sul mercato l'8008, dotato di un set di 48 istruzioni. Nel 1973 apparve l'8080 con 78 tipi base di istruzione, per un totale di più di 200 istruzioni diverse. Nel 1974 venne messo sul mercato il Motorola 6800 che, dopo un paio d'anni, fu seguito dallo Z80 che, con i suoi 158 tipi base di istruzioni, poteva disporre di un set di circa 700 istruzioni diverse. I primi anni '80 videro la comparsa del 68000: questo processore possiede 56 mnemonici base che permettono di creare più di mille istruzioni differenti.

Un decennio di ricerca nel campo della microelettronica ha prodotto risultati decisamente sorprendenti. Uno dei punti più interessanti che riguardano il 68000, è che il numero delle istruzioni disponibili è aumentato moltissimo, mentre il numero degli mnemonici è aumentato solo di poco. Ciò ha delle conseguenze molto importanti: per prima cosa, non è difficile imparare il set di istruzioni; secondo, il set di istruzioni deve disporre di molti modi di indirizzamento differenti e, difatti, il 68000 ne ha 14. Oltre a ciò, il 68000 è in grado di trattare cinque tipi base di dato diversi. L'unione omogenea di questi mnemonici base delle istruzioni, dei modi di indirizzamento e dei tipi di dato, mette a disposizione un insieme di istruzioni molto vasto, eppure facile da imparare.

In questo capitolo ci occuperemo delle istruzioni base disponibili e dei modi di indirizzamento che possono essere usati dalle istruzioni stesse. Per ragioni già esposte, questo capitolo rappresenta solo una introduzione agli argomenti citati: quello che ci proponiamo, è di fornire una descrizione concisa delle possibilità offerte dal processore, cosicché chi co-

nosce già abbastanza l'Assembler può rapidamente impadronirsi delle nozioni base per la programmazione del 68000.

Quando si descrive un set di istruzioni con i suoi modi di indirizzamento, è importante far riferimento alla forma effettiva con la quale le istruzioni stesse vengono presentate a un programma assembler. Ogni singolo programma per lo sviluppo di software in Assembler prevede l'uso di una sintassi diversa; per questo motivo, nel seguito faremo riferimento all'Assembler McGraw-Hill documentato nel capitolo 14. Questo particolare Assembler è stato utilizzato per creare tutti gli esempi di programma descritti in questo libro. Chiaramente, volendo utilizzare un programma assembler differente, bisogna fare attenzione agli inevitabili cambiamenti di sintassi necessari. In particolare, chi ha già letto un testo dettagliato sul 68000 che utilizza la nomenclatura Motorola, avrà notato che l'Assembler McGraw-Hill sembra mancare di alcune istruzioni (per esempio, ADDI non è prevista). Questo Assembler implementa interamente il set di istruzioni del 68000, perciò tutte le istruzioni sono disponibili: il fatto è che alcune istruzioni, considerate dalla Motorola come variazione, sono state nel nostro caso incluse come parte della istruzione base. Per esempio, l'istruzione ADDI della Motorola è semplicemente una istruzione ADD che usa un dato immediato come operando sorgente.

2.1 Modi di indirizzamento

I sei modi base di indirizzamento del 68000 danno origine a 14 modi effettivi. I modi di indirizzamento sono elencati in Figura 2.1, assieme alla appropriata sintassi Assembler. Non è possibile usare tutti i modi di indirizzamento con ogni istruzione; nonostante ciò l'omogeneità è notevole. La singolarità più importante è forse quella che riguarda l'uso dei registri indirizzi come operandi: quando questi registri vengono usati con il modo di indirizzamento diretto, non è mai possibile specificare dati di tipo bit o byte. Ci sono anche alcune istruzioni per le quali la scelta degli operandi è fortemente limitata: per esempio, le istruzioni Bcc (*Branch*) e DBcc (*Decrement and Branch*) possono avere come operando solo un indirizzo assoluto. Questo indirizzo assoluto viene in genere specificato riferendosi a una label. Il valore effettivamente introdotto nel codice sarà in effetti un offset, cosicché le suddette istruzioni permettono l'indirizzamento relativo. Esiste anche ciò che viene detto indirizzamento intrinseco. Le istruzioni che usano questa forma di indirizzamento non richiedono alcun operando. Ci sono sei istruzioni che rientrano in questa categoria: NOP, RESET, RTE, RTR, RTS e TRAPV.

| Modo | Sintassi |
|---|------------------|
| Implicito a registro | SR, CCR, USP, PC |
| Immediato immediato | #n |
| immediato veloce | #b |
| Assoluto corto | a16 |
| lungo | a32 |
| Diretto a registro registro dati | Dn |
| registro indirizzi | An |
| Indiretto a registro registro indirizzi | (An) |
| postincremento | (An) + |
| predecremento | -(An) |
| registro indirizzi con offset | d16(An) |
| registro con indice e offset | d8(An,i) |
| Relativo al contatore di programma registro indirizzi con offset | d16(PC) |
| registro con indice e offset | d8(PC,i) |

Note:

b = 3, 4 o 8 bit

n = 8, 16 o 32 bit

d8 = offset 8 bit

d16 = offset 16 bit

a16 = indirizzo 16 bit

a32 = indirizzo 32 bit

i = An o Dn

An = registro indirizzi

Dn = registro dati

PC = locazione corrente

SR = registro di stato

CCR = codici di condizione

USP = puntatore stack utente

Figura 2.1 Modi di indirizzamento del 68000

INDIRIZZAMENTO RELATIVO AL CONTATORE DI PROGRAMMA

Il modo con cui il programma Assembler tratta questa forma di indirizzamento merita una particolare attenzione. In linea di principio, i modi di indirizzamento relativi al contatore di programma sono equivalenti a quelli relativi ai registri indirizzo con offset (o offset e indice), per cui an-

che la sintassi è la stessa. La differenza, ovviamente, è che per il calcolo indiretto dell'indirizzo viene utilizzato il contatore di programma anziché un registro indirizzi.

In ogni caso, lo scopo principale di questa forma di indirizzamento è di permettere la creazione di codice indipendente dalla posizione. Gli offset indicati nelle istruzioni, perciò, saranno normalmente sotto forma di label all'interno dei programmi. Chiaramente, l'Assembler non deve considerare come offset il valore della label; l'offset, piuttosto, sarà la differenza tra il valore del contatore di programma nel momento in cui viene eseguita l'istruzione che richiede il modo di indirizzamento relativo al PC e il valore della locazione a cui si riferisce la label. Di conseguenza, l'Assembler manipolerà l'istruzione in uno di due modi differenti. Primo modo: se l'espressione dell'offset inizia con un intero, il valore assoluto dell'espressione sarà considerato come offset:

| Codice | Istruzione |
|---------------|-------------------|
| 4BFA1040 | lea \$1040(PC),a5 |

Secondo modo: se l'espressione dell'offset comincia con un simbolo, il simbolo verrà trattato come label (che lo sia oppure no; l'Assembler non può saperlo) e, in questo caso, l'offset è quello effettivo (differenza tra PC e locazione indicata dalla label):

| Indirizzo | Codice | Istruzione |
|------------------|---------------|-------------------|
| 282AE | 44FA0012 | move lab(PC),CCR |

Il valore effettivo del simbolo (label) lab è \$282C2, e \$0012 è l'offset necessario per raggiungerlo dalla data istruzione.

Osserviamo che i modi di indirizzamento relativi al contatore di programma non possono mai essere usati per indirizzare gli operandi destinazione.

OPERANDI SORGENTE E DESTINAZIONE

Le istruzioni del 68000 possono non richiedere alcun operando, oppure possono richiedere un solo operando, sorgente o destinazione, o entrambi gli operandi (sorgente e destinazione) o, infine, (in un caso solo — BTST) due operandi sorgente. Quando vengono richiesti due operandi, l'operando sorgente deve essere sempre specificato per primo, separato dal secondo da una virgola. Per esempio:

- | | |
|--------------------------------------|---------------|
| 1. Nessun operando: | rts |
| 2. Operando sorgente: | clr (a0) |
| 3. Operandi sorgente e destinazione: | divs (a1)+,d1 |

Il risultato di qualunque operazione che riguarda due operandi viene memorizzato all'indirizzo effettivo dell'operando destinazione, se ciò è previsto. Ciò che è contenuto all'indirizzo effettivo dell'operando sorgente non viene alterato.

Questo ordine di dichiarazione degli operandi è solitamente molto pratico e leggibile. Un'eccezione all'ordine appena descritto si verifica con l'istruzione CMP; per esempio, CMP.L D1,D2 confronta D2 con D1. Se D1 è minore di D2, questo confronto risulta nella condizione GT (*Greater Than*: maggiore, con segno) o HI (*Higher*: più alto, senza segno).

2.2 Codici di condizione

Nelle descrizioni delle istruzioni del 68000 che seguono, ci sono tre istruzioni (Bcc, DBcc e Scc) che utilizzano un insieme di test di condizione. A questi test viene abbinato uno mnemonico di uno o due caratteri e lo mnemonico completo dell'istruzione consiste dei nomi elencati in precedenza nei quali cc è sostituito dallo mnemonico del test (per esempio: BHI, BF, DBEQ, SNE e così via). Ciascun test produce come risultato "vero" o "falso" a seconda dello stato di determinati flag di condizione del registro CCR del 68000. I test, i loro mnemonici e la loro interpretazione sono come segue:

| Mnemonico | Test | Interpretazione |
|-----------|------------------|-------------------------------------|
| T | 1 | vero (sempre) |
| F | 0 | falso (sempre) |
| HI | not(C).not(Z) | più alto (senza segno) |
| LS | C+Z | più basso o lo stesso (senza segno) |
| CC (HS) | not(C) | carry falso (senza segno) |
| CS (LO) | C | carry vero (senza segno) |
| NE | not(Z) | diverso |
| EQ | Z | uguale |
| VC | not(V) | overflow falso |
| VS | V | overflow vero |
| PL | not(N) | più |
| MI | N | meno |
| GE | not(N xor V) | maggiore o uguale (con segno) |
| LT | N xor V | minore (con segno) |
| GT | not(Z+(N xor V)) | maggiore |
| LE | Z+(N xor V) | minore o uguale |

Alcuni mnemonici della tabella hanno anche una forma alternativa per migliorarne la leggibilità in determinate circostanze (vedi paragrafo 2.4).

2.3 Trattamento dei flag dei codici di condizione

I flag dei codici di condizione (X, N, V, Z e C) vengono manipolati, in tempi diversi e in vari modi, dal set di istruzioni del 68000. Il trattamento di questi flag può sembrare un po' irregolare, e in effetti lo è, ma questa irregolarità non è frutto di qualche strano comportamento del processore; è, piuttosto, un suo aspetto positivo. In generale, i codici di condizione vengono stabiliti in modo congruente con il valore che viene assegnato all'operando destinazione. Ciò è vero anche per l'operando dell'istruzione TST e per il secondo operando dell'istruzione CMP, sebbene questi operandi destinazione non vengano alterati. Un'eccezione praticamente sempre verificata riguarda l'uso dei registri indirizzi come operando destinazione: in questo caso i codici di condizione non vengono alterati. Ciò permette di eseguire il calcolo degli indirizzi e di operare sui puntatori allo stack senza alterare lo stato dei flag di condizione generato da una operazione precedente. Tuttavia, va ricordato che i codici di condizione vengono settati quando un registro indirizzi viene indicato come registro destinazione di una istruzione CMP.

Il modo in cui viene trattato il flag X è ancora più variabile. Le operazioni con estensione di segno (ABCD, ADDX, NEGX, SBCD e SUBX) provocano l'azzeramento del flag Z se il risultato è diverso da zero; negli altri casi questo flag viene lasciato immutato. Ciò significa che alla fine di una serie di operazioni con estensione di segno, il flag sarà settato solo se tutti i risultati erano zero. Nelle operazioni che trattano i bit (BCHG, BCLR, BSET, BTST e TAS) il flag Z viene settato coerentemente allo stato del bit specificato prima dell'operazione.

2.4 Mnemonici alternativi

L'Assembler mette a disposizione del programmatore un insieme di mnemonici alternativi che si adattano a stili diversi e migliorano la leggibilità. Vediamo, per primo, lo mnemonico dell'istruzione exclusive-or. Gli mnemonici più diffusi di questa istruzione sono due e l'Assembler li prevede entrambi:

| Standard | Alternativo |
|-----------------|--------------------|
| EOR | XOR |

Nella successiva descrizione delle istruzioni del 68000, compare solo lo mnemonico EOR.

Passiamo alle altre istruzioni che prevedono mnemonici alternativi. C'è

spesso confusione circa l'esatta interpretazione da dare ai codici di condizione carry-clear e carry-set, specialmente con quei processori che permettono operazioni aritmetiche con segno oppure senza. Per questo, l'Assembler prevede quanto segue:

| Standard | Alternativo |
|-----------------|--------------------|
| BCC, BCS | BHS, BLO |
| DBCC, DBCS | DBHS, DBLO |
| SCC, SCS | SHS, SLO |

La parte di mnemonico HS sta per *Higher or Same* (più alto o uguale), mentre LO sta per *LOwer* (più basso). Il significato di questi simboli differisce da *Greater or Equal* (GE, maggiore o uguale) e *Less Than* (LT, minore di) per il fatto che i primi si riferiscono a condizioni che si sono verificate in seguito a operazioni in aritmetica senza segno.

2.5 Elenco delle istruzioni 68000

Il processore 68000 dispone di 56 istruzioni base. L'Assembler ne prevede altre sette (come variazioni) per un totale di 63 mnemonici di istruzione. Ciascuno mnemonico viene illustrato concisamente e vengono forniti, tra l'altro, dettagli sui modi di indirizzamento consentiti. Le istruzioni sono elencate in ordine alfabetico per un rapido reperimento delle informazioni. L'appendice A contiene un sommario delle istruzioni che mostra il loro effetto sui flag del CCR.

Nel seguito di questo capitolo si fa riferimento a dei simboli detti descrittori di dato. Questi descrittori possono essere abbinati a certe istruzioni per indicare quale dimensione usare per i dati. Per esempio, vediamo l'istruzione MOVE. Con questa istruzione è possibile manipolare byte (8 bit), word (16 bit) o long word (32 bit). In tutti e tre i casi viene usato lo mnemonico MOVE: è il descrittore di dato che determina l'effettiva operazione. Ciò dà origine alle seguenti tre forme di istruzione:

MOVE.B MOVE.W MOVE.L

Il descrittore .L e l'altro descrittore .S possono essere trovati anche in quelle istruzioni che specificano una label come operando (per esempio, BSR, *Branch to SubRoutine*, salto a subroutine). In questo contesto .L sta per lungo e .S sta per corto. Una label il cui indirizzo viene definito corto deve trovarsi entro 128 byte prima o 127 byte dopo la posizione corrente specificata dal contatore di programma. Una label lunga può trovarsi 32768 byte prima della posizione corrente o 32767 byte dopo. Le istruzio-

ni di salto corto utilizzano meno byte quando sono tradotte in forma binaria, perciò vale la pena di usarle quando il programmatore sa che la destinazione è a distanza inferiore al limite di circa 128 byte ma questa informazione risulta sconosciuta all'Assembler (in quanto è una istruzione di salto in avanti che viene elaborata durante la prima passata). Non è necessario specificare il descrittore nei casi di salto a locazioni precedenti in quanto l'Assembler userà sempre il modo di indirizzamento corto ogni volta che ciò è possibile.

ABCD SOMMA DECIMALE CON ESTENSIONE DI SEGNO

Modi di indirizzamento: $-(An), -(An)$
 Dn, Dn

Flag utilizzati: $X\ N\ Z\ V\ C$

Istruzione privilegiata: no

Solo dati di dimensione byte. Somma due cifre BCD contenute nel byte sorgente, con estensione di segno, alle due cifre BCD del byte di destinazione.

ADD SOMMA

| | | | |
|--------------------------------|----------|----------|---------|
| <i>Modi di indirizzamento:</i> | # n | } | |
| | a16 | | |
| | a32 | | |
| | Dn | | |
| | (An) | | |
| | (An)+ | | ,Dn |
| | -(An) | | ,An |
| | d16(An) | | |
| | d8(An,i) | | |
| | d16(PC) | | |
| | d8(PC,i) | | |
| | | } | |
| | # n, | | a16 |
| | Dn, | | a32 |
| | | | (An) |
| | | | (An)+ |
| | | | -(An) |
| | | | d16(An) |
| | | d8(An,i) | |

An,An
An,Dn

Flag utilizzati: X N Z V C

Istruzione privilegiata: no

Il registro An non può essere specificato come destinazione nelle operazioni su dati di dimensione byte. Somma la sorgente alla destinazione.

ADDQ SOMMA VELOCE

Modi di indirizzamento:

| | | |
|-----|---|----------|
| #b, | } | Dn |
| | | An |
| | | a16 |
| | | a32 |
| | | (An) |
| | | (An)+ |
| | | -(An) |
| | | d16(An) |
| | | d8(An,i) |

Flag utilizzati: X N Z V C

Istruzione privilegiata: no

Il registro An non può essere specificato come destinazione nelle operazioni su dati di dimensione byte. Somma una costante (da 1 a 8) alla destinazione.

ADDX SOMMA CON ESTENSIONE DI SEGNO

Modi di indirizzamento: -(An), -(An)
Dn,Dn

Flag utilizzati: X N Z V C

Istruzione privilegiata: no

Somma la sorgente, con estensione di segno, alla destinazione.

AND AND LOGICO

Modi di indirizzamento:

| | | |
|----------|---|----------|
| #n | } | |
| a16 | | |
| a32 | | |
| Dn | | |
| (An) | | |
| (An)+ | | ,Dn |
| -(An) | | |
| d16(An) | | |
| d8(An,i) | | |
| d16(PC) | | |
| d8(PC,i) | | |
| | } | a16 |
| | | a32 |
| #n, | | (An) |
| Dn, | | (An)+ |
| | | -(An) |
| | | d16(An) |
| | | d8(An,i) |
| | | #n,SR |

Flag utilizzati: N Z V C

Istruzione privilegiata: no

Il registro SR non può essere specificato nelle operazioni su dati di dimensione long word. CCR è il byte meno significativo di SR e vi si può accedere per mezzo dell'istruzione AND.B #n,SR. Esegue l'AND logico di tutti i bit della sorgente con i corrispondenti bit della destinazione.

ASL SHIFT ARITMETICO A SINISTRA

Modi di indirizzamento:

| | |
|----------|--------------------------|
| a16 | (senza descrittore dato) |
| a32 | |
| (An) | |
| (An)+ | |
| -(An) | |
| d16(An) | |
| d8(An,i) | |

#b,Dn (con descrittore dato)
Dn,Dn

Flag utilizzati: X N Z V C

Istruzione privilegiata: no

I dati sono sempre di dimensione word. La destinazione viene sempre shiftata di un bit se non viene specificato alcun parametro; aggiungendo il parametro è possibile shiftare la destinazione fino a 63 posizioni. Nel modo di indirizzamento immediato, è possibile specificare solo uno shift limitato, da 0 a 7. Uno shift di 0 posizioni viene interpretato come uno shift di 8. Il flag V viene settato se il bit di segno viene invertito in qualsiasi momento durante l'operazione di shift.

ASR SHIFT ARITMETICO A DESTRA

Modi di indirizzamento: a16 (senza descrittore dato)
a32
(An)
(An)+
-(An)
d16(An)
d8(An,i)

#b,Dn (con descrittore dato)
Dn,Dn

Flag utilizzati: X N Z V C

Istruzione privilegiata: no

I dati sono sempre di dimensione word. La destinazione viene sempre shiftata di un bit se non viene specificato alcun parametro; aggiungendo il parametro è possibile shiftare la destinazione fino a 63 posizioni. Nel modo di indirizzamento immediato, è possibile specificare solo uno shift limitato, da 0 a 7. Uno shift di 0 posizioni viene interpretato come uno shift di 8. Il bit di segno viene replicato.

Bcc BRANCH SU CONDIZIONE

Modi di indirizzamento: label

Flag utilizzati: nessuno

Istruzione privilegiata: no

È possibile dichiarare label sia corte (.S) che lunghe (.L). Vengono usati offset di dimensione byte o word, rispettivamente.

BCHG TEST E MODIFICA DI BIT

| | | |
|--------------------------------|----------|-------|
| <i>Modi di indirizzamento:</i> | { | a16 |
| | | a32 |
| | | Dn |
| #n, | | (An) |
| Dn, | | (An)+ |
| | | -(An) |
| | d16(An) | |
| | d8(An,i) | |

Flag utilizzati: Z

Istruzione privilegiata: no

Solo dati di dimensione byte, eccetto il caso in cui la destinazione è un registro dati. In questo caso, la dimensione del dato è long word. Controlla lo stato del bit specificato (influenzando il flag Z) e lo inverte.

BCLR TEST E AZZERAMENTO DI BIT

| | | |
|--------------------------------|---|----------|
| <i>Modi di indirizzamento:</i> | { | a16 |
| | | a32 |
| | | Dn |
| #n, | | (An) |
| Dn, | | (An)+ |
| | | -(An) |
| | | d16(An) |
| | | d8(An,i) |

Flag utilizzati: Z

Istruzione privilegiata: no

Solo dati di dimensione byte, eccetto il caso in cui la destinazione è un registro dati. In questo caso, la dimensione del dato è long word. Controlla lo stato del bit specificato (influenzando il flag Z) dopodiché lo azzerà.

BRA BRANCH IN OGNI CASO

Modi di indirizzamento: label

Flag utilizzati: nessuno

Istruzione privilegiata: no

È possibile dichiarare label sia corte (.S) che lunghe (.L). Vengono usati offset di dimensione byte o word, rispettivamente.

BSET TEST E SET DI BIT

Modi di indirizzamento:

n,
Dn,

a16
a32
Dn
(An)
(An) +
-(An)
d16(An)
d8(An,i)

Flag utilizzati: Z

Istruzione privilegiata: no

Solo dati di dimensione byte, eccetto il caso in cui la destinazione è un registro dati. In questo caso, la dimensione del dato è long word. Controlla lo stato del bit specificato (influenzando il flag Z) dopodiché lo setta.

BSR BRANCH A SUBROUTINE

Modi di indirizzamento: label

Flag utilizzati: nessuno

Istruzione privilegiata: no

Possono essere specificate label corte (.S) o lunghe (.L). Copia l'indirizzo dell'istruzione successiva sulla cima dello stack e salta secondo un offset che può essere di dimensione byte o word.

BTST TEST DI BIT

Modi di indirizzamento:

| | | |
|------|---|----------|
| # n, | } | a16 |
| Dn, | | a32 |
| | | Dn |
| | | (An) |
| | | (An)+ |
| | | -(An) |
| | | d16(An) |
| | | d8(An,i) |
| | | d16(PC) |
| | | d8(PC,i) |

Flag utilizzati: Z

Istruzione privilegiata: no

Solo dati di dimensione byte, eccetto il caso in cui la destinazione è un registro dati. In questo caso, la dimensione del dato è long word. Controlla lo stato del bit specificato (influenzando il flag Z).

CHK CONFRONTA UN REGISTRO CON DEI LIMITI

Modi di indirizzamento:

| | | |
|----------|---|-----|
| # n | } | |
| a16 | | |
| a32 | | |
| Dn | | |
| (An) | | |
| (An)+ | | |
| -(An) | | |
| d16(An) | | |
| d8(An,i) | | |
| d16(PC) | | |
| d8(PC,i) | | ,Dn |

Flag utilizzati: N Z V C

Istruzione privilegiata: no

I dati sono solo di dimensione word. Genera un'eccezione se Dn è minore di zero o maggiore del contenuto dell'operando.

CLR AZZERA OPERANDO

Modi di indirizzamento: a16
a32
Dn
(An)
(An)+
-(An)
d16(An)
d8(An,i)

Flag utilizzati: N Z V C

Istruzione privilegiata: no

Viene azzerata quella parte dell'operando identificata dal descrittore della dimensione dei dati.

CMP CONFRONTA

| | | | |
|--------------------------------|----------|----------|-------|
| <i>Modi di indirizzamento:</i> | #n | } | |
| | a16 | | |
| | a32 | | |
| | Dn | | |
| | (An) | | |
| | (An)+ | | ,Dn |
| | -(An) | | ,An |
| | d16(An) | | |
| | d8(An,i) | | |
| | d16(PC) | | |
| | d8(PC,i) | | |
| | | } | |
| | | | a16 |
| | | | a32 |
| | | | (An) |
| | | | (An)+ |
| | | | -(An) |
| | | d16(An) | |
| | | d8(An,i) | |

An,An
An,Dn

Flag utilizzati: N Z V C

Istruzione privilegiata: no

Non è possibile specificare il registro An come destinazione per le operazioni su dati di dimensione byte. Sottrae la sorgente dalla destinazione ma non memorizza il risultato.

CMPM CONFRONTA OPERANDI DI MEMORIA

Modi di indirizzamento: (An)+,(An)+

Flag utilizzati: N Z V C

Istruzione privilegiata: no

Sottrae la sorgente dalla destinazione ma non memorizza il risultato. Non è un'istruzione di tipo esteso.

DBcc DECREMENTA E SALTA SU CONDIZIONE

Modi di indirizzamento: Dn,label

Flag utilizzati: nessuno

Istruzione privilegiata: no

Se la condizione non è verificata, la word contenuta nel registro dati viene decrementata. Se il risultato non è -1 , salta secondo un offset di dimensione word (DBT è una nop di 4 byte).

DBRA DECREMENTA E SALTA IN OGNI CASO

Modi di indirizzamento: Dn,label

Flag utilizzati: nessuno

Istruzione privilegiata: no

Decrementa la word contenuta nel registro dati e salta secondo un offset di dimensione word.

DIVS DIVISIONE CON SEGNO

Modi di indirizzamento:

| | | |
|----------|---|-----|
| #n | } | ,Dn |
| a16 | | |
| a32 | | |
| Dn | | |
| (An) | | |
| (An)+ | | |
| -(An) | | |
| d16(An) | | |
| d8(An,i) | | |
| d16(PC) | | |
| d8(PC,i) | | |

Flag utilizzati: N Z V C

Istruzione privilegiata: no

Divide la long word della destinazione per la word sorgente. Il quoziente viene memorizzato nella word meno significativa, il resto (con lo stesso segno del dividendo!) va nella word più significativa.

DIVU DIVISIONE SENZA SEGNO

Modi di indirizzamento:

| | | |
|----------|---|-----|
| #n | } | ,Dn |
| a16 | | |
| a32 | | |
| Dn | | |
| (An) | | |
| (An)+ | | |
| -(An) | | |
| d16(An) | | |
| d8(An,i) | | |
| d16(PC) | | |
| d8(PC,i) | | |

Flag utilizzati: N Z V C

Istruzione privilegiata: no

Divide la long word della destinazione per la word sorgente. Il quoziente viene memorizzato nella word meno significativa, il resto va nella word più significativa.

EOR OR ESCLUSIVO

Modi di indirizzamento:

| | | |
|-----|---|----------|
| #n, | { | Dn |
| Dn, | | a16 |
| | | a32 |
| | | (An) |
| | | (An)+ |
| | | -(An) |
| | | d16(An) |
| | | d8(An,i) |

#n,SR

Flag utilizzati: N Z V C

Istruzione privilegiata: no, a parte il caso EOR.W #n,SR

Il registro SR non può essere specificato nelle operazioni su dati di dimensione long word. CCR è il byte meno significativo di SR e vi si può accedere per mezzo della istruzione EOR.B #,SR. Esegue l'OR esclusivo tra tutti i bit della sorgente e i corrispondenti bit della destinazione.

EXG SCAMBIA I REGISTRI

Modi di indirizzamento: An,Dn
Dn,Dn
An,An
Dn,An

Flag utilizzati: nessuno

Istruzione privilegiata: no

Solo dati di dimensione long word. Scambia l'intero contenuto dei due registri.

EXT ESTENSIONE DI SEGNO

Modi di indirizzamento: Dn

Flag utilizzati: N Z V C

Istruzione privilegiata: no

Non sono permesse le operazioni su dati di dimensione byte. Estende il bit di segno della metà meno significativa della destinazione all'interno della intera metà più significativa, sempre della destinazione: i bit della metà più significativa vengono posti uguali al bit di segno della metà meno significativa.

JMP SALTO ASSOLUTO

Modi di indirizzamento: a16
a32
(An)
d16(An)
d8(An,i)
d16(PC)
d8(PC,i)

Flag utilizzati: nessuno

Istruzione privilegiata: no

Carica nel contatore di programma l'indirizzo contenuto nell'operando.

JSR SALTA A SUBROUTINE

Modi di indirizzamento: a16
a32
(An)
d16(An)
d8(An,i)
d16(PC)
d8(PC,i)

Flag utilizzati: nessuno

Istruzione privilegiata: no

Copia l'indirizzo dell'istruzione successiva sulla cima dello stack e carica nel contatore di programma l'indirizzo contenuto nell'operando.

LEA CARICA INDIRIZZO EFFETTIVO

Modi di indirizzamento: a16
 a32
 (An)
 d16(An)
 d8(An,i)
 d16(PC)
 d8(PC,i) } ,An

Flag utilizzati: nessuno

Istruzione privilegiata: no

Carica l'indirizzo effettivo della sorgente nel registro destinazione.

LINK COLLEGAMENTO DI STACK

Modi di indirizzamento: An, #n

Flag utilizzati: nessuno

Istruzione privilegiata: no

Il contenuto di An viene copiato sulla cima dello stack. Successivamente, nel registro An viene caricato il puntatore allo stack aggiornato. Infine, lo spiazzamento, a cui è stato esteso il segno, viene sommato al puntatore allo stack.

LSL SHIFT LOGICO A SINISTRA

Modi di indirizzamento: a16 (senza descrittore dato)
 a32
 (An)
 (An)+
 -(An)
 d16(An)
 d8(An,i)

#b,Dn (con descrittore dato)
Dn,Dn

Flag utilizzati: X N Z V C

Istruzione privilegiata: no

I dati sono sempre di dimensione word. La destinazione viene sempre shiftata di un bit se non viene specificato alcun parametro; aggiungendo il parametro è possibile shiftare la destinazione fino a 63 posizioni. Nel modo di indirizzamento immediato, è possibile specificare solo uno shift limitato, da 0 a 7. Uno shift di 0 posizioni viene interpretato come uno shift di 8.

LSR SHIFT LOGICO A DESTRA

Modi di indirizzamento: a16 (senza descrittore dato)
a32
(An)
(An)+
-(An)
d16(An)
d8(An,i)

#b,Dn (con descrittore dato)
Dn,Dn

Flag utilizzati: X N Z V C

Istruzione privilegiata: no

I dati sono sempre di dimensione word. La destinazione viene sempre shiftata di un bit se non viene specificato alcun parametro; aggiungendo il parametro è possibile shiftare la destinazione fino a 63 posizioni. Nel modo di indirizzamento immediato, è possibile specificare solo uno shift limitato, da 0 a 7. Uno shift di 0 posizioni viene interpretato come uno shift di 8.

MOVE COPIA DI DATI

Esistono due tipi di istruzione MOVE, a seconda che venga indicato un descrittore della dimensione dei dati oppure no.

1. MOVE quando è richiesto il descrittore

Modi di indirizzamento:

| | | |
|----------|--|----------|
| #n | | |
| a16 | | a16 |
| a32 | | a32 |
| Dn | | Dn |
| ** An | | An ** |
| (An) | | (An) |
| (An)+ | | (An)+ |
| -(An) | | -(An) |
| d16(An) | | d16(An) |
| d8(An,i) | | d8(An,i) |
| d16(PC) | | |
| d8(PC,i) | | |

(**Non è possibile utilizzare i registri indirizzi An come operando sorgente o destinazione se il dato è di dimensione byte)

Flag utilizzati:

N Z V C

(Non viene alterato alcun flag se la destinazione è An)

Istruzione privilegiata: no

2. MOVE quando il descrittore non è richiesto

Modi di indirizzamento:

| | | |
|----------|--|------|
| #n | | |
| a16 | | |
| a32 | | |
| Dn | | |
| (An) | | ,CCR |
| (An)+ | | ,SR |
| -(An) | | |
| d16(An) | | |
| d8(An,i) | | |
| d16(PC) | | |
| d8(PC,i) | | |

An, USP
USP, An

| | | |
|-----|---|----------|
| SR, | { | a16 |
| | | a32 |
| | | Dn |
| | | (An) |
| | | (An)+ |
| | | -(An) |
| | | d16(An) |
| | | d8(An,i) |

Flag utilizzati: X N Z V C
(Non viene alterato alcun flag se la sorgente è SR o USP)

Istruzione privilegiata: sì, tranne quando il dato viene copiato da SR o CCR

I dati copiati nei registri CCR o SR possono essere solo di dimensione word. Quando la destinazione specificata è il registro CCR, solo il byte meno significativo della sorgente viene utilizzato per aggiornare i codici di condizione. Quando la sorgente è SR il dato è sempre di dimensione word. Le operazioni su USP trattano sempre dati di dimensione long word.

MOVEM COPIA PIÙ REGISTRI

| | | | |
|--------------------------------|---------------------|---|--------------------|
| <i>Modi di indirizzamento:</i> | a16 | { | ,< lista-registri> |
| | a32 | | |
| | (An) | | |
| | (An)+ | | |
| | d16(An) | | |
| | d8(An,i) | | |
| | d16(PC) | | |
| | d8(PC,i) | | |
| | < lista-registri> , | { | a16 |
| | | | a32 |
| | | | (An) |
| | | | -(An) |
| | | | d16(An) |
| | | | d8(An,i) |

Flag utilizzati: nessuno

Istruzione privilegiata: no

La dimensione dei dati può essere word o long word. La lista di registri può contenere qualsiasi registro dati o indirizzi, separato dagli altri da una virgola (l'Assembler non prevede l'uso di sequenze di registri, es.: A2-A5 → da A2 ad A5).

Esempio: MOVEM.L locmem, A1,A2,A3,D4,D6

In memoria, il registro D0 viene posto all'indirizzo più basso e A7 a quello più alto.

MOVEP COPIA I DATI DA/VERSO PERIFERICA

Modi di indirizzamento: Dn,d16(An)
d16(An),Dn

Flag utilizzati: nessuno

Istruzione privilegiata: no

La dimensione dei dati può solo essere word o long word. I byte vengono trasferiti in/da locazioni di memoria non contigue. Se l'indirizzo è pari, il trasferimento avviene attraverso la metà più significativa del bus dati (vale solo per il 68000; il 68008 ha un bus dati di soli 8 bit).

MOVEQ COPIA PICCOLO DATO IMMEDIATO

Modi di indirizzamento: #b,Dn

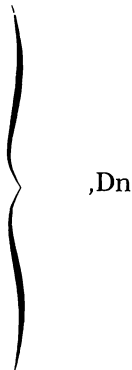
Flag utilizzati: N Z V C

Istruzione privilegiata: no

Copia un dato immediato (da -128 a +127) in un intero registro dati.

MULS MOLTIPLICAZIONE CON SEGNO

Modi di indirizzamento: #n
 a16
 a32
 Dn
 (An)
 (An)+
 -(An)
 d16(An)
 d8(An,i)
 d16(PC)
 d8(PC,i)



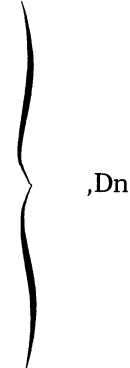
Flag utilizzati: N Z V C

Istruzione privilegiata: no

La metà meno significativa della long word di destinazione viene moltiplicata per la word sorgente.

MULU MOLTIPLICAZIONE SENZA SEGNO

Modi di indirizzamento: #n
 a16
 a32
 Dn
 (An)
 (An)+
 -(An)
 d16(An)
 d8(An,i)
 d16(PC)
 d8(PC,i)



Flag utilizzati: N Z V C

Istruzione privilegiata: no

La metà meno significativa della long word di destinazione viene moltiplicata per la word sorgente.

NBCD NEGAZIONE DECIMALE CON ESTENSIONE DI SEGNO

Modi di indirizzamento: a16
a32
Dn
(An)
(An)+
-(An)
d16(An)
d8(An,i)

Flag utilizzati: X N Z V C

Istruzione privilegiata: no

Solo dati di dimensione byte. Sottrae da zero le due cifre BCD della destinazione, con estensione di segno.

NEG NEGAZIONE

Modi di indirizzamento: a16
a32
Dn
(An)
(An)+
-(An)
d16(An)
d8(An,i)

Flag utilizzati: X N Z V C

Istruzione privilegiata: no

Sottrae la destinazione da zero.

NEGX NEGAZIONE CON ESTENSIONE DI SEGNO

Modi di indirizzamento: a16
a32
Dn
(An)
(An)+
-(An)
d16(An)
d8(An,i)

Flag utilizzati: X N Z V C

Istruzione privilegiata: no

Sottrae la destinazione da zero, con estensione di segno.

NOP NESSUNA OPERAZIONE

Modi di indirizzamento: implicito

Flag utilizzati: nessuno

Istruzione privilegiata: no

NOT COMPLEMENTO A UNO

Modi di indirizzamento: a16
a32
Dn
(An)
(An)+
-(An)
d16(An)
d8(An,i)

Flag utilizzati: N Z V C

Istruzione privilegiata: no

Inverte tutti i bit della destinazione.

OR OR LOGICO

Modi di indirizzamento: #n
a16
a32
Dn
(An)
(An)+
-(An)
d16(An)
d8(An,i)
d16(PC)
d8(PC,i)

} ,Dn

| | | |
|------|---|----------|
| # n, | } | a16 |
| Dn, | | a32 |
| | | (An) |
| | | (An)+ |
| | | -(An) |
| | | d16(An) |
| | | d8(An,i) |

n,SR

Flag utilizzati: N Z V C

Istruzione privilegiata: no, a parte il caso OR.B #n,SR'.

Il registro SR non può essere usato nelle operazioni su dati di dimensione long word. CCR è il byte meno significativo di SR e vi si può accedere per mezzo dell'istruzione OR.B #n,SR. Esegue l'OR logico di tutti i bit della sorgente con i rispettivi bit della destinazione.

PEA COPIA SU STACK INDIRIZZO EFFETTIVO

Modi di indirizzamento: a16
 a32
 (An)
 d16(An)
 d8(An,i)
 d16(PC)
 d8(PC,i)

Flag utilizzati: nessuno

Istruzione privilegiata: no

Copia sulla cima dello stack l'indirizzo effettivo della sorgente.

RESET RESET DI DISPOSITIVI ESTERNI

Modi di indirizzamento: implicito

Flag utilizzati: nessuno

Istruzione privilegiata: sì

Attiva il pin di reset.

ROL ROTAZIONE A SINISTRA

Modi di indirizzamento: a16 (senza descrittore dato)
a32
(An)
(An)+
-(An)
d16(An)
d8(An,i)

#b,Dn (con descrittore dato)
Dn,Dn

Flag utilizzati: N Z V C

Istruzione privilegiata: no

I dati sono sempre di dimensione word. La destinazione viene sempre ruotata di un bit se non viene specificato alcun parametro; aggiungendo il parametro è possibile ruotare la destinazione fino a 63 posizioni. Nel modo di indirizzamento immediato, è possibile specificare solo una rotazione limitata, da 0 a 7. Una rotazione di 0 posizioni viene interpretata come una rotazione di 8. Non setta il flag di estensione.

ROR ROTAZIONE A DESTRA

Modi di indirizzamento: a16 (senza descrittore dato)
a32
(An)
(An)+
-(An)
d16(An)
d8(An,i)

#b,Dn (con descrittore dato)
Dn,Dn

Flag utilizzati: N Z V C

Istruzione privilegiata: no

I dati sono sempre di dimensione word. La destinazione viene sempre ruotata di un bit se non viene specificato alcun parametro; aggiungendo

b,Dn (con descrittore dato)
Dn,Dn

Flag utilizzati: X N Z V C

Istruzione privilegiata: no

I dati sono sempre di dimensione word. La destinazione viene sempre ruotata di un bit se non viene specificato alcun parametro; aggiungendo il parametro è possibile ruotare la destinazione fino a 63 posizioni. Nel modo di indirizzamento immediato, è possibile specificare solo una rotazione limitata, da 0 a 7. Una rotazione di 0 posizioni viene interpretata come una rotazione di 8. La rotazione avviene attraverso il flag di estensione.

RTE RITORNO DA ECCEZIONE

Modi di indirizzamento: implicito

Flag utilizzati: X N Z V C

Istruzione privilegiata: sì

Copia dalla cima dello stack il registro di stato e il contatore di programma.

RTR RITORNO E RIPRISTINO DEL CCR

Modi di indirizzamento: implicito

Flag utilizzati: X N Z V C

Istruzione privilegiata: no

Copia dalla cima dello stack il registro dei codici di condizione e il contatore di programma.

RTS RITORNO DA SUBROUTINE

Modi di indirizzamento: implicito

Flag utilizzati: nessuno

Istruzione privilegiata: no

Copia dalla cima dello stack il contatore di programma.

SBCD SOTTRAZIONE DECIMALE CON ESTENSIONE DI SEGNO

Modi di indirizzamento: $-(An), -(An)$
Dn, Dn

Flag utilizzati: X N Z V C

Istruzione privilegiata: no

Solo dati di dimensione byte. Sottrae, con estensione di segno, le due cifre BCD della sorgente dalle due cifre BCD della destinazione.

Scc SET SU CONDIZIONE

Modi di indirizzamento: a16
a32
Dn
(An)
(An)+
 $-(An)$
d16(An)
d8(An,i)

Flag utilizzati: nessuno

Istruzione privilegiata: no

Solo dati di dimensione byte. Se la condizione è verificata, scrive nel byte destinazione \$FF, altrimenti azzerà il byte destinazione.

STOP ARRESTO DELL'ESECUZIONE

Modi di indirizzamento: #n

Flag utilizzati: X N Z V C

Istruzione privilegiata: sì

Carica il registro di stato e si ferma fino al verificarsi di una interruzione o di un reset.

SUB SOTTRAZIONE

| | | | |
|--------------------------------|----------|---|----------|
| <i>Modi di indirizzamento:</i> | # n | } | |
| | a16 | | |
| | a32 | | |
| | Dn | | |
| | (An) | | |
| | (An)+ | | ,Dn |
| | − (An) | | ,An |
| | d16(An) | | |
| | d8(An,i) | | |
| | d16(PC) | | |
| | d8(PC,i) | | |
| | | } | a16 |
| | | | a32 |
| | # n, | | (An) |
| | Dn, | | (An)+ |
| | | | − (An) |
| | | | d16(An) |
| | | | d8(An,i) |
| | An,An | | |
| | An,Dn | | |

Flag utilizzati: X N Z V C

Istruzione privilegiata: no

Non è possibile utilizzare il registro An come destinazione per le operazioni su dati di dimensione byte. Sottrae la sorgente dalla destinazione.

SUBQ SOTTRAZIONE VELOCE

Modi di indirizzamento:

| | | |
|-----|---|----------|
| #b, | { | Dn |
| | | An |
| | | a16 |
| | | a32 |
| | | (An) |
| | | (An) + |
| | | -(An) |
| | | d16(An) |
| | | d8(An,i) |

Flag utilizzati: X N Z V C

Istruzione privilegiata: no

Non è possibile utilizzare il registro An come destinazione per le operazioni su dati di dimensione byte. Le operazioni su word o long word sono identiche. Sottrae una costante (da 1 a 8) dalla destinazione.

SUBX SOTTRAZIONE CON ESTENSIONE DI SEGNO

Modi di indirizzamento: -(An), -(An)
Dn, Dn

Flag utilizzati: X N Z V C

Istruzione privilegiata: no

Sottrae la sorgente dalla destinazione, con estensione di segno.

SWAP SCAMBIA LE DUE METÀ DI UN REGISTRO DATI

Modi di indirizzamento: Dn

Flag utilizzati: N Z V C

Istruzione privilegiata: no

Scambia la word meno significativa con la word più significativa e viceversa in un registro dati.

TAS TEST E SET DEL BIT 7

Modi di indirizzamento: a16
a32
Dn
(An)
(An)+
-(An)
d16(An)
d8(An,i)

Flag utilizzati: N Z V C

Istruzione privilegiata: no

Solo dati di dimensione byte. Controlla il bit 7 del byte indicato (influenzando i flag N e Z), quindi pone a 1 il bit 7.

TRAP

Modi di indirizzamento: #b

Flag utilizzati: nessuno

Istruzione privilegiata: no

Il valore immediato è un vettore compreso tra 0 e 15. Genera l'eccezione di tipo TRAP specificata.

TRAPV TRAP AL VERIFICARSI DI OVERFLOW

Modi di indirizzamento: implicito

Flag utilizzati: nessuno

Istruzione privilegiata: no

Se il flag di overflow (V) è posto a 1, genera un'eccezione TRAPV.

TST TEST

Modi di indirizzamento: a16
a32
Dn
(An)
(An)+
-(An)
d16(An)
d8(An,i)

Flag utilizzati: N Z V C

Istruzione privilegiata: no

Esegue un test sulla destinazione, che non viene alterata.

UNLK SCOLLEGAMENTO DI STACK

Modi di indirizzamento: An

Flag utilizzati: nessuno

Istruzione privilegiata: no

Il puntatore allo stack viene preso dal registro An. Nel registro An viene poi caricata la long word acquisita dallo stack.

Parte Seconda

Procedure di sistema del QL

Capitolo

Il QDOS

3

Concettualmente il QDOS è il telaio dell'automobile e voi siete il carrozziere. È importante considerare il QDOS in questa ottica: probabilmente sono molti i programmatori che pensano ai sistemi operativi come ad un insieme di programmi di allocazione delle risorse, sotto il cui controllo devono girare i loro programmi applicativi. Per quanto riguarda il QDOS, ciò non è esatto.

Il QDOS è un telaio di procedure. I programmi applicativi che voi scrivete possono usare liberamente ciascuna di queste procedure. Inoltre, il QDOS è dotato di opportune connessioni che permettono di espandere o modificare l'insieme di procedure di cui è costituito a seconda delle necessità. La flessibilità d'uso di cui dispone il programmatore Assembler deriva proprio da questo tipo di struttura.

3.1 Mappa della memoria di sistema

Il modo in cui è suddivisa la memoria fisica del QL e l'uso che il QDOS fa della RAM hanno una tale importanza per il programmatore in Assembler che ci occuperemo per prima cosa di questi argomenti. La Figura 3.1 mostra le due mappe della memoria: la Mappa 1 descrive l'organizzazione dell'intero spazio degli indirizzi del microcomputer; la Mappa 2 è la mappa della RAM.

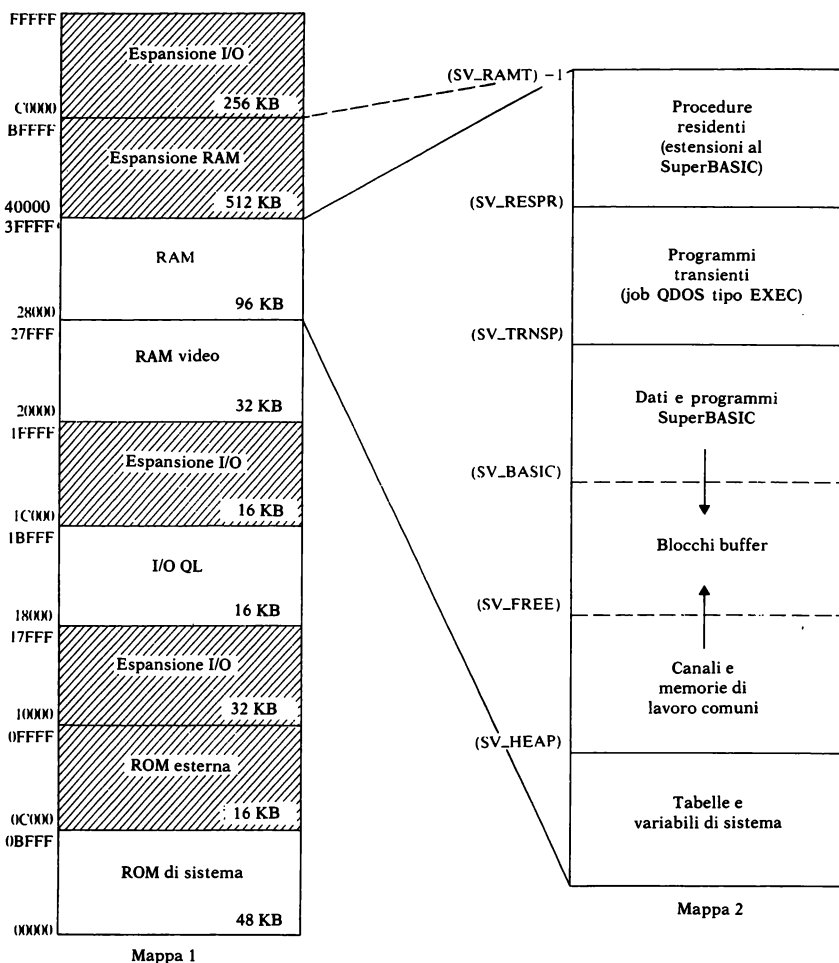


Figura 3.1 Mappa della memoria per il QL/QDOS

MAPPA DELLO SPAZIO DEGLI INDIRIZZI

L'ampiezza totale della memoria è 1 megabyte. La ROM di sistema, unita alla ROM esterna, occupa i 64 K inferiori (da \$00000 a \$0FFFF). I successivi 64 K (da \$10000 a \$1FFFF) sono dedicati ai dispositivi di I/O. Attualmente, vengono utilizzati solo 16 K di questa area. Al di sopra del primo blocco di I/O si trova la RAM, il cui indirizzo di partenza è \$20000. L'ultimo indirizzo di RAM disponibile si trova, in un QL con la memoria standard di 128 K, alla locazione \$3FFFF; quando viene inserito il modulo ag-

giuntivo da 0.5 megabyte, l'ultima locazione di RAM si trova all'indirizzo \$BFFFF. Gli ultimi 256 K dello spazio di indirizzamento sono riservati per espansioni di I/O. Quest'ultima area di memoria, unita all'altra area dedicata all'I/O, fornisce all'utente 304 K da dedicare all'espansione dell'I/O. Ciò può sembrare esagerato, ma bisogna considerare che in quest'area, oltre ai dispositivi fisici di I/O, sono allocati anche i programmi di gestione delle periferiche; ciò comporta un evidente vantaggio: non è necessario destinare spazio di memoria RAM alle periferiche quando si procede ad una espansione di I/O.

MAPPA DELLA RAM IMPOSTA DAL QDOS

Ora che sappiamo come è suddiviso lo spazio fisico degli indirizzi, occupiamoci di come è strutturata la RAM. I 32 K inferiori (da \$20000 a \$27FFF) sono dedicati alla memoria video. I rimanenti 96 K (da \$28000 a \$3FFFF) o 608 K (da \$28000 a \$BFFFF) vengono suddivisi dal QDOS in cinque aree diverse. Ci sono diverse variabili di sistema che vengono usate per determinare in qualsiasi momento le dimensioni di queste aree di QDOS e il puntatore alla zona di RAM ancora disponibile. Le variabili, il loro uso e la loro posizione assoluta all'interno della memoria sono indicate nella Figura 3.2. Ciascuna variabile è memorizzata in una long word ed è quindi lunga 32 bit. Gli mnemonici assegnati alle variabili hanno solo uno scopo illustrativo: essi non vengono riconosciuti né dal SuperBASICS né dal QDOS.

AREA DELLE PROCEDURE RESIDENTI

Nella parte più alta della RAM trovano posto le procedure residenti; queste procedure, insieme ad eventuali tabelle associate, vengono caricate in memoria quando viene fatto il boot del sistema (cioè quando il sistema viene resettato). L'unico modo di modificare il contenuto di quest'area è di effettuare un nuovo boot del sistema. Di questo, comunque, parleremo più diffusamente nel capitolo 8. Il nome "area delle procedure residenti" significa proprio che le procedure caricate in questa area al momento del boot, resteranno permanentemente allocate fino al successivo boot o reset. Normalmente, questa area viene utilizzata per contenere le procedure SuperBASIC definite dall'utente. Aggiungendo i nomi di entry point delle procedure alla lista dei nomi di procedure dell'interprete SuperBASICS, le procedure stesse diventano estensioni del SuperBASIC. Ogni procedura aggiunta al sistema in questo modo deve essere rientrante (cioè non deve contenere variabili locali o codice automodificante) e indipendente dalla posizione. È chiaro che l'area delle procedure residenti

non deve necessariamente esistere e, in effetti, non esiste se non vengono caricate procedure esterne all'atto del boot.

| Base delle variabili di sistema (SV_BASE) = \$28000 | | |
|---|--|--------------|
| SV_HEAP | Base dell'area comune di heap | SV_BASE+\$04 |
| SV_CHPFRR | Primo spazio dell'area di heap | SV_BASE+\$08 |
| SV_FREE | Base dell'area di RAM libera | SV_BASE+\$0C |
| SV_BASIC | Base dell'area del SuperBASIC | SV_BASE+\$10 |
| SV_TRNSP | Base dell'area transiente | SV_BASE+\$14 |
| SV_TRNFR | Primo spazio dell'area transiente | SV_BASE+\$18 |
| SV_RESPR | Base dell'area delle procedure residenti | SV_BASE+\$1C |
| SV_RAMT | Limite superiore della RAM (più 1) | SV_BASE+\$20 |

Figura 3.2 Variabili di sistema associate alla RAM

AREA DEI PROGRAMMI TRANSIENTI

L'area successiva è l'area dei programmi transienti (TPA, *Transient Program Area*). Come nel caso dell'area delle procedure residenti, la TPA non deve necessariamente esistere. Volendo scrivere ed eseguire un programma transiente in codice macchina, è necessario allocare una opportuna quantità di RAM utilizzando una chiamata al QDOS per mezzo dell'istruzione TRAP #1. Quando è stata generata una TPA, un programma al suo interno (trattato dal QDOS come job) può essere attivato o inattivato. Il job può anche essere sospeso, oppure la TPA può venire eliminata. Il processo di allocazione di spazio di TPA e di scheduling dei job fa parte delle possibilità di multitask di cui è dotato il QDOS. Il capitolo 4 contiene una discussione completa di tali processi. Oltre ad essere eseguito come un job di QDOS, un programma transiente può anche essere eseguito dal SuperBASIC utilizzando il comando EXEC.

Ciascun programma di TPA deve essere scritto in codice indipendente dalla posizione, ma non è necessario che sia rientrante; inoltre ogni programma deve essere dotato delle sue aree dati e stack. Un'area TPA può anche essere usata semplicemente come area dati: in tal caso è ovviamente importante che il job non diventi mai attivo e quindi non venga mai eseguito. A causa del fatto che i job possono essere creati e distrutti praticamente a piacimento, l'area totale di TPA può aumentare o restringersi in modo dinamico. Bisogna tenere presente, comunque, che qualsiasi programma in TPA utilizza sempre una quantità di memoria che deve essere dichiarata in precedenza.

AREA DEL SUPERBASIC

Questa area contiene tutti i programmi SuperBASIC correntemente in memoria insieme ai relativi dati (cioè sia i dati dei programmi che i dati dell'interprete del SuperBASIC). Ovviamente, non è possibile predire quanto spazio dovrà essere allocato per un dato programma in SuperBASIC. Tenendo presente quanto appena detto, il QDOS tratta in modo particolare questa area e le permette di aumentare e diminuire dinamicamente. Anche l'area di TPA totale che sta al di sopra dell'area del SuperBASIC può aumentare e diminuire dinamicamente, perciò l'intera area SuperBASIC può muoversi dinamicamente.

TABELLE E VARIABILI DI SISTEMA

Tutti i computer utilizzano una piccola quantità di RAM per memorizzare importanti variabili, puntatori e così via. Il QDOS, inoltre, utilizza delle tabelle contenute in RAM per operazioni come la gestione dei job e dei canali. Nel QDOS, questa piccola quantità di memoria viene prelevata dalla base della RAM.

Il processore 68000 è in grado di funzionare sia in modo utente che in modo supervisore. Ad ognuno dei due modi sono associate differenti aree di stack con differenti puntatori; lo stack del modo supervisore è collocato tra le variabili di sistema e le tabelle.

AREA DEI CANALI E DI HEAP

Quest'area si trova in RAM, direttamente al di sopra dell'area delle variabili e delle tabelle di sistema e viene usata sia per stabilire le connessioni tra l'I/O e i canali specificati, sia come area comune di heap. Un altro uso di quest'area è quello di memoria di lavoro per i programmi di gestione delle periferiche (per esempio, la routine di gestione della tastiera). In casi come questo, è il gestore della periferica stesso che si alloca dello spazio all'interno di questa area. Anche i job di QDOS possono richiedere come spazio parte di questa area. Quando un dato job viene rimosso, viene anche rimossa qualsiasi area di heap che gli fosse stata assegnata.

Quest'area di RAM è concettualmente simile all'area dei programmi transienti (entrambe sono di tipo heap) e, come l'area del SuperBASIC, non è possibile prevederne la dimensione effettiva. Anche questa, quindi, può espandersi e restringersi dinamicamente. A questo punto, ci sono due aree che variano di dimensione dinamicamente: l'area TPA+SuperBASIC e questa area. Questa è esattamente la situazione che si presenta in piccoli sistemi monoutente che utilizzano un solo stack. Il modo più sempli-

ce di implementare un tale sistema consiste nel fare crescere una regione di memoria da una estremità verso il centro e la seconda regione dall'altra estremità verso il centro. Quando le due regioni si incontrano, la memoria è esaurita! La Figura 3.1 mostra che il QDOS è implementato proprio in questo modo: l'area TPA+SuperBASIC cresce verso l'alto. La regione di memoria che avanza tra le due aree viene ceduta ai sottosistemi di gestione dei file sotto forma di blocchi di servizio.

BLOCCHI DI SERVIZIO DEI SOTTOSISTEMI DI GESTIONE DEI FILE

Quest'area è collocata tra le aree del SuperBASIC e di canale e heap che possono variare dinamicamente. Tutto lo spazio di RAM che avanza, in qualsiasi momento, viene ceduto ai sottosistemi di gestione dei file che lo utilizzano sotto forma di blocchi di servizio. I blocchi di servizio non sono visibili all'utente e il loro ~~loro~~ unico scopo è di duplicare i dati contenuti nei Microdrive. L'uso di questi blocchi consente al QDOS di accedere ai Microdrive nel modo più efficiente possibile: più RAM libera c'è e maggiore è l'efficienza degli accessi ai Microdrive. Questo meccanismo implica che il QDOS sfrutti costantemente tutta la memoria disponibile per ricavarne il maggior vantaggio.

3.2 Il bootstrap

Quando il computer viene acceso o resettato, l'esecuzione comincia dall'indirizzo di partenza (base) della ROM di sistema. Non appena sono state determinate le variabili di sistema, e dopo che è stato eseguito un test della RAM, il sistema operativo esegue un esame del sistema per determinarne la configurazione. Per prima cosa, viene verificato il contenuto della prima locazione della ROM esterna (indirizzo \$00000) alla ricerca della long word caratteristica \$4AFB0001. Se questo dato è effettivamente presente, il sistema operativo assume che la ROM esterna sia effettivamente inserita e che contenga del codice eseguibile. Successivamente, vengono ricercati i programmi di gestione delle periferiche (*device driver*) nelle aree di memoria dedicate all'espansione. Supponiamo che, infine, il controllo venga restituito alla routine di bootstrap: a questo punto essa tenterà di aprire un dispositivo chiamato BOOT o il file MDV1__BOOT. Se questa operazione ha successo, il rispettivo file viene caricato in memoria (come programma SuperBASIC) ed eseguito.

3.3 Chiamate al sistema operativo e utility

Esistono principalmente due tipi di routine che i programmatori Assembler possono richiamare dall'interno dei loro programmi applicativi: le routine del primo tipo sono quelle accessibili per mezzo delle istruzioni TRAP #n indirizzate al QDOS; il secondo tipo comprende quelle routine di uso generale che vengono richiamate per mezzo di vettori.

ROUTINE DEL QDOS

Le chiamate al QDOS possono essere considerate sia indivisibili che parzialmente indivisibili. La maggior parte delle routine del QDOS sono intrinsecamente indivisibili e vengono eseguite con il processore 68000 in modo supervisore: in questo modo nessun altro job può accedere al processore, per cui le routine QDOS vengono eseguite dall'inizio alla fine prima che altri programmi vengano autorizzati ad utilizzare il processore. Tenete presente che questo è solo il caso generale: le routine possono essere interrotte da una routine di servizio dell'interrupt. Le routine parzialmente indivisibili, in questo caso, completano un certo insieme di operazione fondamentali, poi cedono il processore a un altro job fino a quando, in un momento successivo, il processo originale torna in esecuzione. Tutte le chiamate di I/O sono solo parzialmente indivisibili a meno che non si specifichi esplicitamente che devono essere eseguite in modo non interrompibile. Anche le chiamate allo scheduler sono parzialmente indivisibili.

Osserviamo che l'esecuzione dell'istruzione TRAP #0, sul QL, provoca il passaggio al modo supervisore. Non viene alterato nessun registro (eccetto, ovviamente, il puntatore allo stack, che diventa l'SSP). Per tornare al modo utente è sufficiente alterare il registro di stato.

L'accesso alle procedure di QDOS avviene tramite le chiamate di tipo TRAP #n con l'indicazione della particolare procedura richiesta nel registro D0. I capitoli dal 4 al 6 descrivono queste procedure in dettaglio, ma vale la pena di esaminarne qualche aspetto in questo capitolo. Il registro D0, oltre a contenere il codice che identifica la procedura (in formato byte) all'atto della chiamata, viene anche usato per restituire un codice di errore (long word) al processo chiamante. Se il codice di errore è diverso da zero, ciò indica che si è verificato un errore. Gli errori standard vengono identificati da piccoli (in valore assoluto) numeri negativi; questi errori sono elencati nell'appendice C. Se per mezzo della TRAP è stata chiamata una qualche forma di device driver supplementare, il codice di errore restituito può, in effetti, essere un puntatore a uno specifico messaggio di errore. Per evitare che i due tipi di codice di errore possano es-

sere confusi, il codice di tipo puntatore punta, in realtà, a un indirizzo che si trova \$8000 byte sotto l'indirizzo reale del messaggio di errore. Tutte le routine del QDOS poi, possono, almeno in teoria, restituire l'errore ERR.BP (-15), che significa *bad parameter* (parametro illegale). Oltre al registro D0, anche i registri dati da D1 a D3 e i registri indirizzi da A0 ad A3 possono essere usati in diversi modi per passare valori da e verso le procedure di QDOS. Quando gli opportuni registri sono stati predisposti per una data chiamata, l'accesso alla routine desiderata avviene semplicemente eseguendo l'appropriata istruzione di TRAP del 68000. Per esempio, per disabilitare il cursore nella window associata alla ID di canale \$10001, possono essere usate le seguenti istruzioni:

```
;
move.b      #15,d0      ; routine che disabilita il cursore
move.w      #0,d3       ; ritorno immediato
move.l      #$10001,a0   ; ID di canale
trap        #3
;
```

La descrizione completa di questa routine di QDOS, presentata nel capitolo 6, mostra che è in grado di segnalare tre errori (i due elencati più l'errore generico di parametro illegale). È ovviamente saggio controllare se si sono verificati errori dopo che è stata eseguita la chiamata tramite TRAP.

ROUTINE DI UTILITY

Queste routine, per quanto riguarda questo libro, sono un insieme di routine con accesso tramite TRAP semplificato e routine di utility del Super-BASIC. Ciascuna routine viene descritta in dettaglio nel capitolo 7. Utilizzando queste routine, il programmatore in Assembler può semplificare moltissimo il codice per le operazioni fondamentali di I/O e può addirittura incorporare la possibilità di calcoli in virgola mobile all'interno dei programmi applicativi.

Il metodo di accesso per ciascun tipo di utility è lo stesso e richiede semplicemente che vengano predisposti gli appropriati parametri di chiamata e quindi venga effettuata una chiamata a subroutine indirizzata per mezzo dell'opportuno vettore. Per esempio, per inviare al canale dei comandi (#0) la rappresentazione ASCII di una word contenente un intero che si trova alla locazione di memoria che ha come label RISULT, si può usare il seguente codice:

```
;
move.w    result(pc),d1    ; legge risultato
sub.l     a0,a0            ; seleziona il canale 0
move.w    #$CE,a4         ; routine di conversione e
                                ; visualizzazione
jrs       (a4)
;
;
result:    defs 2          ; locazione risultato intero
;
```

Anche in questo caso è bene controllare che non si siano verificati errori al ritorno dalla subroutine.

Ci sono quattro routine di gestione dei Microdrive che devono essere trattate in modo leggermente diverso. Per prima cosa, i loro vettori di accesso puntano, in realtà, a un indirizzo che si trova \$4000 byte prima dell'inizio effettivo delle routine. È perciò importante sommare questo offset quando si eseguono le relative chiamate vettorizzate:

```
move.w    md.verin,a4
jsr       $4000(a4)
```

In secondo luogo, queste routine non restituiscono codici di errore ma possono restituire il controllo al programma chiamante da più punti differenti.

Le procedure del QDOS richiamabili tramite l'istruzione TRAP #1 hanno il compito di gestire le risorse del computer. Il QDOS permette il multitasking (cioè l'esecuzione in pseudo parallelismo di più job), perciò l'allocazione delle risorse di CPU e di RAM riveste un'importanza fondamentale.

4.1 Il multitasking nel QDOS

Le chiamate al sistema operativo dirette al QDOS possono essere trattate come indivisibili o parzialmente indivisibili. La maggior parte delle routine del QDOS sono, per natura, indivisibili e vengono eseguite con il processore 68000 in modo supervisore: in questo modo nessun altro job può avere una priorità tale da poter pretendere di utilizzare il processore e, quindi, le routine di sistema operativo vengono eseguite dall'inizio alla fine senza essere private dell'uso del processore. Va notato che quanto detto finora riguarda solo il caso più generale: le routine possono essere interrotte da una routine di risposta ad un interrupt. Le routine parzialmente indivisibili si limitano a completare le operazioni più importanti tra quelle di cui sono incaricate, prima di permettere ad un altro job di scavalcare il processo chiamante, che verrà ripreso nel seguito. Tutte le routine che eseguono operazioni di I/O sono parzialmente indivisibili, a meno che non vengano chiamate specificando che devono essere eseguite senza essere interrotte. Tutte le chiamate allo scheduler sono parzialmente indivisibili.

STATO DEI JOB

Un job, quando viene creato dalle procedure del QDOS, può esistere in diversi stati. In primo luogo, il job può essere attivo; ciò significa che al job è stata assegnata una priorità, nell'ambiente di esecuzione multitasking, che gli permette di ottenere una parte di risorse di CPU proporzionale a quella priorità. Se il job ha una bassa priorità, potrà ottenere una percentuale relativamente bassa del tempo di CPU.

In alternativa, il job può essere sospeso, sia per un periodo di tempo determinato che indefinitamente. Di solito i job vengono sospesi quando devono attendere il completamento di operazioni di I/O o di operazioni di altri job.

Come ultimo caso, il job può essere inattivo: ciò significa che il job è contenuto in memoria centrale ma ad esso non verrà mai assegnato del tempo di CPU. Dire che un job ha priorità 0 equivale a dire che il job è inattivo. La differenza principale tra un job inattivo e un job che è stato sospeso indefinitamente è che quest'ultimo non può essere rimosso per mezzo di una semplice chiamata alla routine di rimozione job, cioè MT.RJOB (TRAP #1, D0=4). La differenza tra job sospesi e rilasciati e job attivi e inattivi è la seguente: la sospensione e il successivo rilascio di un job non alterano il flusso di esecuzione delle istruzioni, semplicemente lo interrompono. Al contrario, un job inattivo è un job che ha completato la sua esecuzione: se viene riattivato, l'esecuzione riparte dal principio.

SCHEDULING

Con *scheduling* si intende il processo di allocazione ai job delle risorse di CPU, tenuto conto delle priorità dei job stessi. Nel QDOS, il re-scheduling avviene in sincronismo con la frequenza di refresh del video (50/60 Hz). Alcune routine di QDOS provocano autonomamente l'esecuzione del re-scheduling, per esempio, MT.SUSJB (TRAP #1, D0=8).

TIMEOUT

Un certo numero di chiamate al QDOS permette di specificare un limite di tempo massimo (*timeout*); ciò è possibile, ad esempio, con la procedura di sospensione dei job (MT.SUSJB). Il periodo di timeout è sempre un multiplo del periodo di refresh del video (50/60 Hz). Un timeout uguale a 1, quindi, equivale a 20 ms per una frequenza video di 50 Hz, e a 16.666 ms per una frequenza di 60 Hz.

Per convenzione, le procedure di QDOS considerano un timeout uguale a -1 come un periodo di tempo infinito (il solo valore negativo consenti-

to è proprio -1). Il più lungo timeout che può essere specificato è 32767 volte il periodo base; ciò equivale a 10 minuti e 55.3 secondi alla frequenza di 50 Hz e a 9 minuti e 6.1 secondi a 60 Hz.

4.2 Uso dei registri del 68000

Alle procedure relative alla TRAP #1 si accede indicando nel registro D0 (byte) quale particolare chiamata si intende effettuare; lo stesso registro viene anche utilizzato per restituire al processo chiamante una long word che descrive lo stato in cui è terminata l'esecuzione della routine: se il dato restituito è diverso da zero, vuol dire che si è verificato un errore. Per indicare gli errori più comuni, vengono usati dei piccoli valori negativi; questi codici di errore sono elencati nell'appendice C. Se la chiamata per mezzo della istruzione di trap prevedeva l'esecuzione di una routine di tipo device driver, il codice di errore restituito può in effetti essere un puntatore ad uno specifico messaggio di errore. Per evitare che i due tipi di codice di errore possano venire confusi, il codice di errore di tipo puntatore è in realtà un puntatore a un indirizzo inferiore di \$8000 rispetto al vero indirizzo del messaggio. Tutte le routine del QDOS prevedono la possibilità di restituire il codice di errore ERR.BP (-15), che significa parametro non valido. La descrizione completa delle procedure richiamabili con TRAP #1 stabilisce quali siano i possibili errori che possono essere segnalati da ciascuna procedura; è ovviamente utile e saggio esaminare il registro D0 dopo ogni chiamata per mezzo di trap per controllare se si sono verificati degli errori.

Oltre al registro D0, anche i registri dati D1, D2 e D3 e i registri indirizzi A0, A1, A2 e A3 vengono utilizzati in vari modi per passare parametri alle routine di QDOS e in senso inverso. Dopo che gli appropriati registri sono stati predisposti per la chiamata ad una particolare routine, la chiamata stessa viene effettuata semplicemente eseguendo l'istruzione TRAP #1. Quando non viene specificato il descrittore di dimensione dei parametri scambiati tra QDOS e job utente (cioè .B, .W o .L), si sottointende che il dato deve essere una long word (cioè .L).

MT.INF \$00 (0)

Acquisisce informazioni di sistema

Parametri in ingresso: nessuno

Parametri in uscita: D1.L ID del job corrente
 D2.L Versione QDOS (in ASCII)
 A0 Puntatore alle variabili di sistema

Registri interessati: D1, D2, A0

Errori possibili: nessuno

Descrizione:

MT.INF restituisce delle informazioni sul sistema. Il numero che indica la versione del QDOS viene passato come stringa ASCII di 4 byte nel formato:

v.xx

dove *v* è la versione principale e *xx* è il codice dell'aggiornamento. La word più significativa di D2 contiene il codice della versione e il punto, mentre il codice dell'aggiornamento viene posto nella word meno significativa di D2.

Il puntatore alle variabili di sistema che viene restituito in A0 è il valore del puntatore alla base SV.BASE (di solito, \$28000).

MT.CJOB \$01 (1)

Crea un job nell'area dei programmi transienti

Parametri in ingresso:

| | |
|------|-----------------------------|
| D1.L | ID del job padre |
| D2.L | Lunghezza del codice (byte) |
| D3.L | Lunghezza area dati (byte) |
| A1 | Indirizzo di partenza |

Parametri in uscita:

| | |
|------|-------------------------|
| D1.L | ID del job creato |
| A0 | Base dell'area allocata |

Registri interessati: D1, A0

Errori possibili:

| | |
|--------|-----------------------|
| ERR.OM | (-3) memoria esaurita |
| ERR.NJ | (-2) job non valido |

Descrizione:

Sotto QDOS, i job vengono creati nell'area dei programmi transienti. Ciascun job risiede in un'area di memoria di dimensioni prefissate, che deve comprendere anche tutte le aree di dati e di lavoro (incluso lo stack). È sempre meglio prevedere almeno 64 byte in più per lo stack, rispetto al valore previsto, che servono al QDOS per eseguire le sue routine. Quando viene chiamata MT.CJOB, i registri D2 e D3 devono specificare la quantità totale di memoria che sarà utilizzata dal job. La lunghezza dello stack va inclusa nell'indicazione dell'ampiezza dell'area dati.

L'indirizzo di partenza specificato sarà zero se l'entry point del job coincide con la sua base. Qualsiasi altro indirizzo indicato dovrà essere assoluto. Se il job creato dovrà essere indipendente, la ID del job padre sarà zero. In caso contrario (quando, cioè, il job corrente è il padre del job da creare) la ID del job padre che viene passata può essere -1 (cioè un valore negativo).

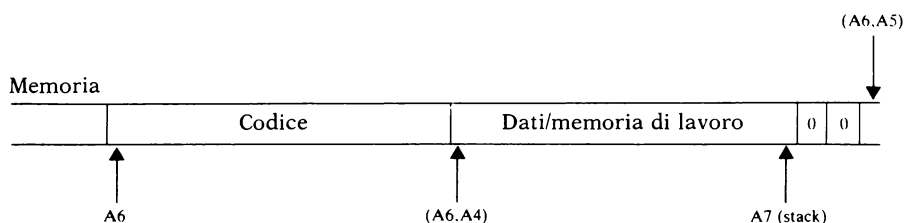


Figura 4.1 Puntatori all'area job

Chiariamo che questa procedura non carica né esegue il job in questione: tenta semplicemente di allocare spazio in TPA e di predisporre un descrittore di job nella tabella dello scheduler. Solitamente, il job vero e proprio viene caricato da un altro job dopo che la chiamata a questa routine è stata completata con successo. Il programma del job deve essere scritto in codice indipendente dalla posizione. Quando un job viene attivato, il registro A6 punta alla base dell'area del job, A6 con A4 come indice punta alla base dell'area dati e A6 con A5 come indice punta all'estremo superiore dell'intera area. Il registro puntatore allo stack, A7, punterà, nel caso più semplice, a due word contenenti zero poste sullo stack dalla procedura MT.CJOB (vedi Figura 4.1).

Queste due word a zero costituiscono un pacchetto di informazioni in un formato standard. Nei casi meno semplici, gli zeri vengono sostituiti da pacchetti di informazioni più dettagliate come, per esempio, una word di dati che rappresenta il numero di canali aperti per quel job, alcune long word per le ID dei canali e una stringa di comandi per il job. Anche la stringa di comandi ha un formato particolare: una word di dati che rappresenta la lunghezza della stringa di comandi, seguita dalla stringa stessa. Osserviamo che il comando SuperBASIC EXEC predispone l'area in TPA, carica ed esegue un job, tutto in un colpo solo. Perciò, di solito si usa EXEC (vedi capitolo 8) per invocare un programma transiente, a meno che non sia necessario che un dato job controlli direttamente i suoi job figli.

MT.JINF \$02 (2)

Acquisisce informazioni sui job

Parametri in ingresso: D1.L ID del job
D2.L ID del job in cima all'albero

Parametri in uscita: D1.L ID del job successivo nell'albero
D2.L ID del job padre
D3.L Stato/priorità
A0 Indirizzo della base del job

Registri interessati: D1 – D3, A0, A1

Errori possibili: ERR.NJ (–2) job non valido

Descrizione:

Questa procedura restituisce lo stato di un dato job. I job possono essere indipendenti oppure possono essere figli di altri job (a parte il job zero che non può essere figlio di nessun altro job). La struttura dell'appartenenza dei job ad altri job può essere rappresentata da uno schema ad albero (vedi Figura 4.2). Con MT.JINF è possibile esaminare lo stato di un intero albero di job. A questo scopo, è sufficiente far puntare D1 e D2 alla cima dell'albero e richiamare continuamente MT.JINF finché in D1 non viene restituito zero che indica che non c'è nessun altro job). Al ritorno dalla routine, D2 conterrà zero se il job esaminato è indipendente. Il byte più significativo di D3 sarà negativo se il job è sospeso e il byte meno significativo di D3 conterrà la priorità del job.

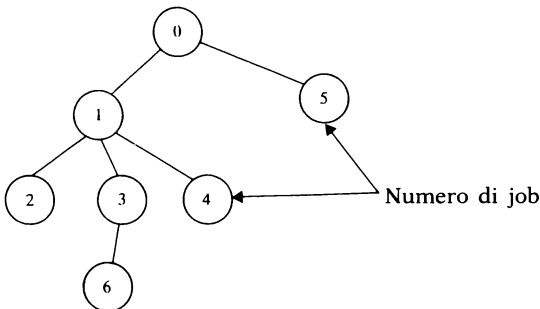


Figura 4.2 Albero di appartenenza dei job

MT.RJOB \$04 (4)

Rimuove un job inattivo dalla TPA

Parametri in ingresso: D1.L ID del job
 D3.L Codice di errore

Parametri in uscita: nessuno

Registri interessati: D1 – D3, A0 – A3

Errori possibili: ERR.NJ (– 2) job non valido
 ERR.NC (– 1) job non attivo

Descrizione:

Questa procedura rimuove un dato dall'area dei programmi transienti. Viene rimosso anche qualsiasi job figlio del job indicato. Perché la procedura possa funzionare, i job coinvolti devono essere inattivi. All'atto della chiamata, D3 deve contenere il codice di errore che deve essere restituito dalla routine di attivazione che ha creato il job (vedi MT.ACTIV; D0 = \$0A). Se non esiste alcun errore di questo tipo, D3 conterrà zero. Il job zero non può essere rimosso. Non è garantito che la procedura sia eseguita in modo indivisibile.

MT.FRJOB \$05 (5)

Rimuove d'autorità i job dalla TPA

Parametri in ingresso: D1.L ID del job
 D3.L Codice di errore

Parametri in uscita: nessuno

Registri interessati: D1 – D3, A0 – A3

Errori possibili: ERR.NJ (–2) job non valido

Descrizione:

Questa procedura inattiva e rimuove un intero albero di job. Tra i parametri passati, D1 può essere negativo se il job da rimuovere è quello corrente; inoltre, D3 deve contenere il codice di errore che deve essere restituito dalla routine di attivazione che ha creato il job (vedi MT.ACTIV; D0=\$0A). Se non esiste un tale errore, D3 conterrà zero.

Se un altro job sta attendendo il termine dell'esecuzione del job che deve essere rimosso, esso verrà tolto dallo stato di attesa con D0 contenente il codice di errore inizialmente restituito dalla routine di attivazione che aveva creato il job da rimuovere.

Il job zero non può essere rimosso. Non è garantito che l'esecuzione della procedura avvenga in modo indivisibile.

MT.FREE \$06 (6)

Dimensione del più grande spazio libero in TPA

Parametri in ingresso: nessuno

Parametri in uscita: D1.L Lunghezza dello spazio trovato

Registri interessati: D1 – D3, A0 – A3

Errori possibili: nessuno

Descrizione:

Questa routine restituisce la dimensione della più grande area di memoria in un blocco unico che possa poi essere allocata a un programma transiente. Il valore restituito può solo servire come indicazione se i job attivi sono numerosi: il sistema di scheduling può aver concesso a un altro job una parte della memoria libera (o tutta) nel tempo che intercorre tra la chiamata a questa routine e il tentativo del job che l'ha fatta di allocarsi un'area di memoria.

MT.TRAPV \$07 (7)

Assegna al job il puntatore al vettore di trap

Parametri in ingresso: D1.L ID del job
A1 Puntatore alla tabella dei vettori

Parametri in uscita: D1.L ID del job
A0 Indirizzo della base del job

Registri interessati: D1, A0, A1

Errori possibili: nessuno

Descrizione:

Le trap e i vettori di eccezione della CPU 68000 che non vengono utilizzati dal QDOS possono essere ridefiniti per mezzo di una tabella creata da un dato job. Se un job crea una tale tabella, essa verrà utilizzata durante l'esecuzione del job. Inoltre, ogni job generato da un job che possiede una sua tabella dei vettori adotta automaticamente la stessa tabella fino a che essa non viene ridefinita localmente.

Le tabelle di vettori create dai job sono del tutto locali al job che le crea (con l'eccezione che riguarda l'adozione vista in precedenza) e non hanno effetto sulle tabelle degli altri job. Se il parametro passato alla procedura tramite D1 è negativo, la tabella verrà associata al job che ha chiamato la procedura. La tabella dei vettori a cui punta A1 all'atto della chiamata deve contenere indirizzi in formato long word per ciascuna trap o eccezione. L'ordine della tabella, insieme all'indirizzo di offset per ciascun vettore (rispetto alla base della tabella), è il seguente:

| Offset | Vettore/eccezione | Offset | Vettore/eccezione |
|--------|--------------------------|--------|-------------------|
| 00 | Errore di indirizzamento | 28 | TRAP # 7 |
| 04 | Istruzione illegale | 2C | TRAP # 8 |
| 08 | Divisione per zero | 30 | TRAP # 9 |
| 0C | CHK | 34 | TRAP # 10 |
| 10 | TRAPV | 38 | TRAP # 11 |
| 14 | Violazione di privilegio | 3C | TRAP # 12 |
| 18 | Trace | 40 | TRAP # 13 |
| 1C | Interrupt di livello 7 | 44 | TRAP # 14 |
| 20 | TRAP # 5 | 48 | TRAP # 15 |
| 24 | TRAP # 6 | | |

MT.SUSJB \$08(8)

Sospende un job

Parametri in ingresso: D1.L ID del job
 D3.W Limite di tempo (timeout)
 A1 Indirizzo del byte di flag

Parametri in uscita: D1.L ID del job
 A0 Base dell'area di controllo del job

Registri interessati: D1, A0

Errori possibili: ERR.NJ (−2) job non valido

Descrizione:

Questa procedura sospende un job per un dato tempo o indefinitamente. Se, all'atto della chiamata, D1 è negativo, viene sospeso il job corrente. La sospensione a tempo indefinito ha luogo se il valore della word di D3 viene passato come −1. Non deve essere usato nessun altro valore negativo. La sospensione di un job già sospeso provoca il rinnovo del periodo di sospensione.

Il byte di flag che si trova nell'area di controllo del job viene posto a zero quando, in seguito, il job viene risvegliato. Il flag indica a un job che sospende un altro job due cose: o che il timeout della sospensione è terminato o che un altro job ha già risvegliato il job sospeso. La manipolazione dell'area di controllo dei job è un'operazione piuttosto sofisticata e, di solito, non è alla portata del programmatore non professionista. Pertanto, all'atto della chiamata, il registro A1 deve contenere zero.

MT.SUSJB provoca l'attivazione dello scheduler che manipola la coda dei job in attesa di essere eseguiti: pertanto, la stessa MT.SUSJB non può essere totalmente indivisibile.

MT.RELJB \$09 (9)

Risveglia un job e provoca l'intervento dello scheduler

Parametri in ingresso: D1.L ID del job

Parametri in uscita: D1.L ID del job
A0 Base dell'area di controllo del job

Registri interessati: D1, A0

Errori possibili: ERR.NJ (− 2) job non valido

Descrizione:

Questa chiamata risveglia (cioè toglie dallo stato di sospensione) un dato job e provoca l'intervento dello scheduler per tutti i job. Risvegliare un job non implica che quel job diventi attivo: lo stato di attività dei job è anche legato alla loro priorità e un job con priorità zero risulta inattivo. A causa dell'intervento dello scheduler, MT.RELJB non può essere totalmente indivisibile.

MT.ACTIV \$0A (10)

Attiva un job

Parametri in ingresso: D1.L ID del job
 D2.B Priorità del job (da 0 a 127)
 D3.W Timeout (0 o -1)

Parametri in uscita: D1.L ID del job
 A0 Base dell'area di controllo del job

Registri interessati: D1, A0, A3

Errori possibili: ERR.NJ (-2) job non valido
 ERR.NC (-1) job già attivo

Descrizione:

Il job indicato viene attivato nell'area dei programmi transienti. Quando la CPU viene assegnata al job, l'esecuzione comincia all'indirizzo specificato quando il job è stato creato (vedi MT.CJOB (TRAP #1, D0=1)). La priorità 127 è la più alta prevista.

Sono disponibili due possibilità di timeout. La prima: se viene dato un timeout =0, l'esecuzione del job corrente (chiamante) prosegue. Il job appena creato comincerà ad essere eseguito in un momento successivo, quando e se lo scheduler lo stabilirà. In tale caso, gli unici errori addizionali che possono essere restituiti sono ERR.NJ (-2) e ERR.NC (-1).

Seconda possibilità: se il timeout indicato è -1, il job corrente viene sospeso fino a quando il job appena attivato ha terminato la sua esecuzione. In questo caso può essere restituito qualsiasi errore, quando il job termina (cioè quando il job si autorimuove o è rimosso da un altro job. Vedi MT.RJOB; D0=4, e MT.FRJOB; D0=5).

MT.PRIOR \$0B (11)

Modifica la priorità di un job

Parametri in ingresso: D1.L ID del job
D2.B Priorità del job (da 0 a 127)

Parametri in uscita: D1.L ID del job
A0 Base dell'area di controllo del job

Registri interessati: D1, A0

Errori possibili: ERR.NJ (−2) job non valido

Descrizione:

Questa chiamata può essere utilizzata per modificare la priorità di un job. Se, all'atto della chiamata, D1 è negativo, viene modificata la priorità del job corrente. Una priorità =0 provoca l'inattivazione. Questa procedura invoca lo scheduler, perciò un job si inattiva immediatamente se pone la propria priorità =0.

MT.ALRES \$0E (14)

Alloca l'area delle procedure residenti

Parametri in ingresso: D1.L Numero di byte richiesti

Parametri in uscita: A0 Indirizzo della base dell'area

Registri interessati: D1 – D3, A0 – A3

Errori possibili: ERR.OM (– 3) memoria esaurita
ERR.NC (– 1) TPA non vuota

Descrizione:

Questa procedura è usata per allocare memoria nell'area delle procedure residenti. Può essere usata solo quando la TPA è completamente vuota (cioè quando non esiste alcun programma transiente).

Va notato che, di solito, per questo scopo viene usata la funzione Super-BASIC RESPR, chiamata per mezzo di un dispositivo o file di BOOT (vedi capitolo 8).

MT.RERES \$0F (15)

Rilascia l'area delle procedure residenti

Parametri in ingresso: nessuno

Parametri in uscita: nessuno

Registri interessati: D1 – D3, A0 – A3

Errori possibili: ERR.NC (– 1) TPA non vuota

Descrizione:

Questa procedura libera l'area delle procedure residenti. È evidente un paradosso: la chiamata non può essere fatta se la TPA non è vuota, ma deve esistere un programma per effettuare la chiamata; d'altra parte, non è il caso di inserire la chiamata in un programma dell'area delle procedure residenti in quanto si autodistruggerebbe nel processo.

C'è un modo per superare questi ostacoli ma il suo uso non va incoraggiato. In pratica, perciò, questa chiamata non verrà mai effettuata e l'area delle procedure residenti verrà rilasciata solo resettando l'intero sistema o rieseguendo il boot.

MT.DMODE \$10 (16)

Modifica/verifica del modo display

Parametri in ingresso: D1.B Flag di modifica/verifica modo display
 D2.B Flag di modifica/verifica tipo display

Parametri in uscita: D1.B Modo display
 D2.B Tipo display

Registri interessati: D1, D2, A4

Errori possibili: nessuno

Descrizione:

Questa procedura può essere utilizzata per due scopi differenti. Nel primo caso, se D1 e D2 vengono passati alla procedura posti a -1, vengono restituiti, nei rispettivi registri, il modo display (cioè 4 o 8 colori) e il tipo display (cioè TV o monitor). I valori restituiti sono i seguenti:

 modo display: 0-4 colori
 8-8 colori

 tipo display: 0-monitor
 1-TV (625 linee)

Seconda funzione: se alla procedura vengono passati, nei rispettivi registri, i codici appena visti (uno per tipo), il display verrà modificato in modo da essere coerente con i codici passati. La procedura va utilizzata in questo secondo modo solo se si è certi che nessun altro job sta accedendo al display. Come ulteriore effetto della chiamata, tutte le window vengono ripulite e la dimensione dei caratteri può cambiare. In alcuni casi (per esempio, come avviene nel Programma 9.3) è possibile scrivere i programmi in modo che lo stato del display possa essere modificato senza gravi effetti secondari.

MT.IPCOM \$11 (17)

Comandi per IPC (controllore intelligente di periferiche):
scansione delle righe della tastiera, suono

Parametri in ingresso: A3 Puntatore al comando

Parametri in uscita: D1.B Parametro di ritorno

Registri interessati: D1, D5, D7

Errori possibili: nessuno

Descrizione:

Bisogna fare la massima attenzione quando si vuole usare questa procedura: la comunicazione con l'IPC è del tutto priva di protezione e può verificarsi una perdita completa di operazioni macchina se la chiamata è affetta da errore. Inoltre, la maggior parte dei comandi di IPC è di uso riservato al QDOS, per cui è probabile che un tentativo di utilizzare questi comandi provochi una perdita di dati o altri effetti altrettanto spiacevoli. Sono tre i comandi di IPC utilizzabili; ciascun comando consiste di una stringa di byte contenente un byte di comando, un blocco di parametri e un byte che indica la lunghezza della risposta prevista. Il blocco di parametri è costituito da un byte che indica il numero di parametri (un parametro è lungo un byte), da una long word contenente fino a 16 codici costituiti da una coppia di bit ciascuno, e dai byte che rappresentano i parametri effettivi. I codici lunghi 2 bit servono per determinare quanti bit di un dato parametro (lungo un byte) devono essere effettivamente inviati all'IPC; la loro codifica è:

- 00 — invia i quattro bit meno significativi
- 01 — non invia alcun bit
- 10 — invia tutti i bit

I bit 1,0 della long word sono associati al byte di parametro 0; i bit 3,2 sono associati al parametro 1 e così via. Il byte finale che indica la lunghezza della risposta è codificato in modo simile utilizzando i bit 1,0 (per esempio, un byte = \$02 significa che il valore restituito nel registro D1 deve essere lungo otto bit).

I tre comandi disponibili sono:

1. Scansione delle righe della tastiera

Il byte di comando è \$09. Un parametro di quattro bit indica la riga di cui va effettuata la scansione. La risposta è lunga otto bit, ciascuno dei quali è posto a uno se il tasto della colonna corrispondente è premuto. La relazione tra righe, colonne e tasti della tastiera si può trovare nel manuale d'uso del QL, nella parte che descrive il comando SuperBASIC KEYROW.

2. Creazione di suoni

Il byte di comando è \$0A. Ci sono otto parametri:

| | |
|-------------------|--------|
| tono 1 | 8 bit |
| tono 2 | 8 bit |
| periodo del passo | 16 bit |
| durata | 16 bit |
| passo del tono | 4 bit |
| involuppo | 4 bit |
| rumore | 4 bit |
| purezza | 4 bit |

Non è prevista alcuna risposta.

3. Soppressione del suono

Il byte di comando è \$0B. Non ci sono né parametri né risposta.

MT.BAUD \$12 (18)

Stabilisce la frequenza di trasmissione (baud rate)

Parametri in ingresso: D1.W Baud rate

Parametri in uscita: nessuno

Registri interessati: D1

Errori possibili: nessuno

Descrizione:

Questa procedura stabilisce la frequenza di trasmissione delle due interfacce seriali, SER1 e SER2. La frequenza di trasmissione delle due interfacce deve essere la stessa (vale sia per SER1 che per SER2) e viene passata alla procedura come un numero binario puro. Per esempio, per ottenere una frequenza di trasmissione di 9600 baud può essere utilizzata la seguente sequenza di istruzioni:

```
;
move.w    #9600,d1    ; 9600 baud
move.b    #$12,d0     ; chiama la procedura per baud rate
trap      #1
;
```

MT.RCLCK \$13 (19)

Legge il valore del tempo

Parametri in ingresso: nessuno

Parametri in uscita: D1.L Ora in secondi

Registri interessati: D1, D2, D3

Errori possibili: nessuno

Descrizione:

Questa procedura restituisce l'ora in secondi e può essere usata in due modi. In primo luogo, può essere utilizzata insieme alle trap di inizializzazione dell'orologio e regolazione dell'ora e alle utility per la data, il giorno della settimana e l'ora per ottenere un effettivo orologio-calendario durante il periodo di accensione del computer. In questo caso si assume che il tempo inizi dalle ore 00:00 del 1° gennaio 1961.

Nel secondo modo, il valore restituito può essere utilizzato semplicemente per misurare il tempo trascorso tra due chiamate successive.

MT.SCLCK \$14 (20)

Inizializza l'orologio

Parametri in ingresso: D1.L Ora in secondi

Parametri in uscita: D1.L Ora in secondi

Registri interessati: D1 – D3, A0

Errori possibili: nessuno

Descrizione:

Questa procedura serve a inizializzare l'orologio in modo che possano essere utilizzate delle routine per la visualizzazione di un orologio-calendario. Si assume che il tempo inizi alle ore 00:00 del 1° gennaio 1961.

MT.ACLCK \$15 (21)

Regola l'orologio

Parametri in ingresso: D1.L Correzione in secondi

Parametri in uscita: D1.L Ora in secondi

Registri interessati: D1 – D3, A0

Errori possibili: nessuno

Descrizione:

Chiamando questa routine è possibile regolare l'ora dell'orologio sia in avanti che indietro. Poiché occorre un certo tempo per regolare l'orologio, non viene effettuata alcuna correzione se in D1 viene passato zero.

MT.ALCHP \$18 (24)

Alloca l'area comune di heap

Parametri in ingresso: D1.L Numero di byte richiesti
D2.L ID del job a cui va assegnata l'area.

Parametri in uscita: D1.L Numero di byte allocati
A0 Indirizzo della base dell'area

Registri interessati: D1 – D3, A0 – A3

Errori possibili: ERR.OM (–3) memoria esaurita
ERR.NJ (–2) job non valido

Descrizione:

I job possono appropriarsi di aree di memoria prendendole dall'area comune di heap. L'area può venire concessa a un job per conto di un altro job e tutto lo spazio allocato per mezzo di questa procedura viene rilasciato quando il job destinatario (e non "l'allocatore") viene rimosso. Lo spazio, una volta concesso, viene ripulito ed è totalmente disponibile per il job.

MT.RECHP \$19 (25)

Rilascia spazio all'area comune di heap

Parametri in ingresso: A0 Base dell'area da liberare

Parametri in uscita: nessuno

Registri interessati: D1 – D3, A0 – A3

Errori possibili: nessuno

Descrizione:

Questa procedura rilascia lo spazio specificato all'area comune di heap. Il programmatore, prima di chiamare questa procedura, ha la responsabilità di accertarsi che l'area da rilasciare non debba effettivamente più essere utilizzata .

Le procedure di input e output del QDOS rientrano in due categorie principali. La prima riunisce quelle procedure che si occupano della allocazione di canali per unità e file. L'altra comprende le procedure che eseguono effettivamente le operazioni di input e output tramite i canali allocati. La prima categoria viene associata alle chiamate TRAP #2 descritte in questo capitolo; la seconda viene associata alle chiamate TRAP #3 descritte nel capitolo 6.

5.1 Input e output ridirezionabile

Un concetto importante riguardo al QDOS è quello dell'I/O ridirezionabile: non è necessario che il programmatore applicativo sappia quale dispositivo fisico è effettivamente collegato a un canale di I/O. Aprendo un canale che deve essere usato con un Microdrive si accede esattamente alla stessa routine utilizzata per aprire una window supplementare sullo schermo. È il device driver che si incarica di muovere le informazioni dentro e fuori dai canali.

Talvolta è necessario specificare alcuni aspetti dei dispositivi fisici insieme al nome del dispositivo logico. Queste informazioni supplementari vengono usate dall'opportuno device driver per configurare il canale a seconda delle necessità.

5.2 Nomi di unità standard

Il QDOS prevede l'uso di un certo numero di device driver standard. Ciascun driver accetta un nome prestabilito seguito da opportuni parametri di configurazione. I dispositivi attualmente previsti e i loro parametri sono i seguenti:

CON_*wXhAxXy_k* I/O da console. Viene definita una window di larghezza *w*, altezza *h*, posizionata in *x*, *y*. La lunghezza del buffer di tastiera viene definita uguale a *k*. Osserviamo che la dimensione e la posizione della window viene specificata in termini di pixel in una matrice 512 per 256. La larghezza *w* e la coordinata *x* devono essere specificate in multipli di due pixel.

La definizione di default è:

CON_448x180a32x16_128

SCR_*wXhAxXy* Screen di output. Ha la stessa definizione di window che vale per il dispositivo di I/O CON_.

La definizione di default è:

SCR_448x180a32x16

SER_{*npz*} I/O su linea seriale. Il numero della porta è indicato da *n*. La parità (Even, Odd, Mark o Space) è definita da *p* che può essere E, O, M o S. Il parametro *z* specifica il protocollo e può assumere i valori R (*raw*, dati generici), Z (CTRL Z è eof), o C (<CR> viene convertito in <LF> in input, <LF> viene convertito in <CR> in output e CTRL Z è eof).

La definizione di default è:

SER1R

NETI_*nn* Collegamento su rete seriale dal nodo *nn*.

NETO_*nn* Collegamento su rete seriale verso il nodo *nn*.

MDV_{*n_nome*} Microdrive. *n* è il numero che identifica la particolare unità; *nome* è il nome del file. Non c'è default.

5.3 Uso dei registri del 68000

Le procedure che utilizzano la TRAP #2 vengono chiamate ponendo nel registro D0 (byte) il numero che identifica la particolare chiamata richiesta. Questo stesso registro viene anche usato per restituire al processo chiamante un codice di errore (long word). Se il codice di errore è diverso da zero ciò indica che si è verificato un errore. Gli errori standard sono indicati da piccoli (in valore assoluto) numeri negativi e sono elencati nell'appendice C. Se la chiamata tramite TRAP era riferita a una qualche forma di device driver supplementare, il codice di errore restituito può essere un puntatore a uno specifico messaggio di errore. Per evitare che i due tipi di codice di errore possano essere confusi, il codice di tipo puntatore punta in realtà a un indirizzo \$8000 byte al di sotto dell'indirizzo effettivo del messaggio di errore. Tutte le routine di QDOS possono, almeno in teoria, restituire il codice di errore ERR.BP (-15), che significa parametro illegale. La descrizione completa delle procedure di TRAP #2 stabilisce quali altri errori possano essere notificati. È ovviamente saggio controllare l'esistenza di una condizione di errore subito dopo che è stata effettuata la chiamata per mezzo di TRAP.

Oltre al registro D0, vengono usati anche i registri dati da D1 a D3 e i registri indirizzi da A0 ad A3 per passare valori da e verso le procedure QDOS. Quando gli opportuni registri sono stati predisposti per una data chiamata, l'accesso alla routine appropriata avviene semplicemente eseguendo l'istruzione TRAP #2. Nei casi in cui il descrittore del dato (cioè .B, .W, o .L) non è indicato nella descrizione, il descrittore di default è .L (cioè long word).

IO.OPEN \$01 (1)

Apri un canale

Parametri in ingresso: D1.L ID del job
 D3.L Codice di accesso
 A0 Indirizzo del nome del canale

Parametri in uscita: D1 Job ID
 A0 ID del canale

Registri interessati: D1, A0

Errori possibili: ERR.NJ (-2) job non valido
 ERR.OM (-3) memoria esaurita
 ERR.NO (-6) troppi canali aperti
 ERR.NF (-7) file o unità non trovato
 ERA.EX (-8) file già esistente
 ERR.IV (-9) file o unità già in uso
 ERR.BN (-12) nome unità o file non corretto

Descrizione:

Un canale viene aperto quando si specifica un appropriato nome di unità e/o file. Quando si effettua la chiamata, A0 punta al nome, che è una stringa di caratteri ASCII preceduta dal numero dei caratteri (dato di tipo word); questo è infatti, il formato standard dei parametri di tipo stringa. Il registro A0 punta alla word di dati che rappresenta il numero dei caratteri.

A ciascun job è associata una lista privata dei canali di I/O, pertanto quando si chiama questa procedura bisogna fornire anche la ID del job. Se la ID del job passata è uguale a -1, il canale viene associato al job corrente.

Si possono eseguire più tipi di open, quindi D3 può contenere un codice che indica l'opzione di open richiesta. Le opzioni disponibili insieme ai corrispondenti codici, sono riportate di seguito:

| Codice di accesso | Tipo |
|--------------------------|--|
| 0 | Apri file/unità esistente. Uso esclusivo. Lettura e scrittura |
| 1 | Apri file/unità esistente. Uso condiviso. Sola lettura |

- | | |
|---|--|
| 2 | Apri nuovo file. Uso esclusivo. Lettura e scrittura |
| 3 | Apri nuovo file (sovrascrittura) |
| 4 | Apri file di catalogo. Sola lettura |

La maggior parte dei dispositivi ignora il codice di accesso. Il QDOS supporta il tipo di accesso sovrascrittura ma il programma di gestione dei Microdrive no.

L'errore ERR.BN (-12) viene solitamente generato quando il nome dell'unità è stato riconosciuto, ma le informazioni passate con i parametri sono scorrette. In caso di errore, non viene aperto alcun canale.

IO.CLOSE \$02 (2)

Chiude un canale

Parametri in ingresso: A0 ID del canale

Parametri in uscita: nessuno

Registri interessati: A0

Errori possibili: ERR.NO (–6) canale non aperto

Descrizione:

Questa procedura non fa altro che chiudere il canale specificato.

IO.FORMT \$03 (3)

Formatta un supporto di memoria di massa organizzato a settori

Parametri in ingresso: A0 Puntatore al nome del supporto

Parametri in uscita: D1.W Numero di settori utilizzati
D2.W Numero totale di settori

Registri interessati: D1, D2, A0

Errori possibili: ERR.OM (-3) memoria esaurita
ERR.NF (-7) unità non trovata
ERR.IU (-9) unità già in uso
ERR.FF (-14) formattazione non riuscita

Descrizione:

Il nome del supporto di memoria di massa a cui punta A0 quando viene chiamata la procedura deve essere nel formato standard dei parametri stringa. Ciò significa che A0 deve puntare a una word di dati (che specifica la lunghezza della stringa) seguita dalla stringa ASCII stessa.

La stringa deve essere costituita da: il nome del drive, seguito dal numero del drive, a sua volta seguito dal simbolo di sottolineatura e, infine, dal nome del supporto (fino a 10 caratteri). Il nome del supporto è quel nome che va indicato quando viene richiesto il catalogo dei file di quel supporto. Per esempio, la stringa del comando:

MDV1_MIOTESTO

formatterà il supporto nel Microdrive 1 e gli metterà come nome "MIOTESTO".

IO.DELET \$04 (4)

Cancella un file

Parametri in ingresso: D1.L ID del job
 A0 Indirizzo del nome di unità/file

Parametri in uscita: nessuno

Registri interessati: D1, D3, A0 – A2

Errori possibili: ERR.OM (–3) memoria esaurita
 ERR.NO (–6) troppi canali aperti
 ERR.NF (–7) file o unità non trovato
 ERR.BN (–12) nome unità o file non corretto

Descrizione:

La cancellazione di un file è una forma di operazione di open e pertanto è necessario fornire una ID di job. Si può usare qualunque ID, ma –1 è la più opportuna (indica il job corrente). All'atto della chiamata, A0 punta al nome, che è una stringa di caratteri ASCII preceduta dal numero di caratteri che fanno parte della stringa (dato di lunghezza .W), dato che questo è il formato standard per i parametri di tipo stringa. Il registro A0 punta alla word di dati che rappresenta il numero di caratteri della stringa. Non viene eseguita nessuna operazione di cancellazione se si verifica un errore.

Le operazioni di input/output

6

Le procedure di input e output del QDOS rientrano in due categorie principali. La prima riunisce quelle procedure che si occupano della allocazione di canali per unità e file. L'altra comprende le procedure che eseguono effettivamente le operazioni di input e output tramite i canali allocati. La prima categoria viene associata alle chiamate TRAP #2 descritte nel capitolo 5; la seconda viene associata alle chiamate TRAP #3 descritte in questo capitolo.

6.1 Timeout

È necessario specificare un timeout per tutte le procedure associate alla TRAP #3. Il timeout è un multiplo del periodo del video (il cui contenuto viene ridisegnato alla frequenza di 50/60 Hz). Un timeout di lunghezza unitaria è perciò equivalente a 20 ms alla frequenza di 50 Hz, e a 16.667 ms alla frequenza di 60 Hz.

Se una routine viene chiamata specificando un timeout uguale a zero, essa restituirà il controllo al programma chiamante immediatamente dopo aver tentato di eseguire il suo compito, sia che il tentativo abbia avuto successo oppure no. Un timeout positivo indica che la procedura deve ritornare non appena eseguito il suo compito o allo scadere del timeout, a seconda di quale evento si verifichi per primo. Il massimo timeout consentito è di 32767 periodi video, che corrisponde a 10 minuti e 55.3 secondi alla frequenza di 50 Hz, o a 9 minuti e 6.1 secondi alla frequenza di 60 Hz. Se il valore di timeout specificato è -1, ciò indica che il periodo

di attesa richiesto è infinito; -1 è l'unico valore negativo che può essere usato per questo scopo.

Un esempio adatto è la procedura `IO.FBYTE`. Supponiamo che questa procedura venga usata per leggere un byte (carattere) da tastiera: se viene specificato un `timeout=0`, la procedura restituirà un carattere se lo avrà trovato già in attesa di essere letto, altrimenti restituirà l'errore `ERR.NC` (*not complete*) se non era pronto alcun carattere. In ogni caso, il controllo viene restituito immediatamente. Un `timeout` positivo costringe la procedura ad attendere l'arrivo di un carattere per il periodo di tempo specificato; se il carattere arriva prima dello scadere del limite di tempo, la procedura restituisce immediatamente il controllo e il carattere acquisito. Se il carattere non arriva entro il limite di tempo previsto, la routine, dopo avere atteso lo scadere del `timeout`, restituirà l'errore `ERR.NC`. Un `timeout = -1` costringerà la procedura ad attendere fino al completamento dell'operazione, cioè fino al momento in cui viene inserito un carattere.

6.2 Principi fondamentali

Bisogna sempre tener presente che il QDOS supporta il multitasking e, pertanto, più di un job può tentare di accedere, nello stesso momento, a un dato canale. Se il vostro job richiede l'accesso a un canale che ha già una richiesta in coda, possono succedere due cose: se il `timeout` indicato era zero, la chiamata a procedura restituirà immediatamente il controllo al job di partenza trasmettendogli l'errore `ERR.NC`; se il `timeout` era diverso da zero, il job viene rimesso nella coda dello scheduler fino al momento in cui la richiesta di I/O può essere servita. Tenete presente in modo particolare che il periodo di attesa (`timeout`) parte dal momento in cui il vostro job ottiene di accedere al canale e non dal momento in cui era stata effettuata la richiesta per la prima volta. Ciò significa, ovviamente, che il periodo di `timeout` reale può essere maggiore del `timeout` programmato.

La situazione di non completamento al ritorno da una chiamata a procedure di I/O ha degli importanti effetti collaterali per i job nel caso di operazioni di output. In tali casi, l'insuccesso nel completamento delle operazioni significa che la procedura di QDOS non è riuscita a inviare in output tutti i dati che le erano stati passati. È chiaro che tutti i dati che non sono stati inviati per mezzo di una data chiamata devono essere inviati in un momento successivo. Nel caso di output di singoli byte (caratteri) non ci sono problemi: basta inviare una seconda volta il byte. Nel caso che sia stata inviata una stringa di caratteri, la procedura di QDOS restituisce il numero di caratteri effettivamente emessi sul canale.

L'I/O di grandi quantità di dati richiede sempre una particolare attenzione. È inefficiente eseguire tali operazioni di I/O un byte alla volta; il QL è dotato di una grande quantità di RAM che permette di adottare delle opportune tecniche di I/O che fanno uso di buffer in memoria. In tali casi, i caratteri verrebbero importati ed esportati in blocchi aventi le maggiori dimensioni possibili.

6.3 Blocchi di definizione dei canali associati allo schermo

Circa il 70% delle procedure di QDOS richiamabili per mezzo della TRAP #3 è dedicato a operazioni di gestione dello schermo. La procedura di definizione delle window (SD.WDEF) è un esempio. Tutte le procedure richiedono che venga specificata una ID di canale. L'argomento che riguarda le ID di canale e i canali associati allo schermo è, perciò, così importante che ce ne occuperemo adesso approfonditamente.

ID DI CANALE

Una ID di canale è una long word che contiene due dati diversi. La word più significativa della ID contiene una label che viene incrementata ogni volta che un canale viene aperto. La word meno significativa della ID contiene un codice, l'indice di canale, che viene usato internamente dal QDOS. A titolo di informazione (nella pratica non si usa mai), il valore del codice, moltiplicato per quattro, fornisce un opportuno indice per accedere alla tabella dei canali (vedi Figura 6.1).

All'accensione del computer, vengono aperti tre canali associati allo schermo. In SuperBASIC questi canali vengono chiamati #0, #1 e #2. La effettiva corrispondenza tra questi canali di schermo del SuperBASIC e le ID di canale del QDOS è la seguente:

| SuperBASIC # | ID QDOS | | |
|--------------|-------------|--------|----------|
| | Esadecimale | | Decimale |
| | Contatore | Canale | |
| 0 | 0000 | 0000 | 0 |
| 1 | 0001 | 0001 | 65537 |
| 2 | 0002 | 0002 | 131074 |

Se, per esempio, il canale #2 del SuperBASIC viene riaperto, esso rimane sempre il canale #2 per quanto riguarda il SuperBASIC, ma la sua ID in-

ternamente al QDOS sarà #00030002 (196610 in decimale).

È importante rendersi conto che, in pratica, non è mai necessario conoscere questa corrispondenza. Quando scrivete programmi applicativi interamente in Assembler, avete sempre a disposizione una copia della ID di canale del QDOS, e ciò è tutto quello che vi serve. Quando, invece, passate un numero di canale dal SuperBASIC a una utility in Assembler, la utility riceverà l'intero che rappresenta il canale standard (cioè #0, #1, ecc.) e lo userà per calcolare la ID di canale usata internamente dal QDOS (vedi capitoli 8 e 11).

CANALI ASSOCIATI ALLO SCHERMO

Il QDOS gestisce una tabella delle risorse di canale che contiene informazioni quali il numero di canale più alto in uso e altre cose (vedi Figura 6.1). Due puntatori contenuti nella tabella, SV_CHBAS e SV_CHTOP, puntano, rispettivamente, alla base e all'estremo superiore della tabella dei canali. I puntatori (long word) all'interno della tabella dei canali puntano, a loro volta, ai corrispondenti blocchi di definizione dei canali.

Se un programma applicativo ha effettivamente bisogno di accedere alle informazioni relative a un dato canale di schermo, può ottenerlo chiamando la propria subroutine "trova le informazioni" attraverso la procedura per operazioni estese associata alla TRAP #3 (SD.EXTOP; D0=9). La chiamata a SD.EXTOP fa sì che la vostra subroutine venga considerata un device driver standard e che vengano effettuate alcune operazioni importanti. Per prima cosa, alla subroutine viene passato, nel registro A6, l'indirizzo della base delle variabili di sistema (SV_BASE). Secondo, il registro A7 viene associato allo stack di supervisore in modo tale che la subroutine possa usare fino a 64 byte di stack. Tenete presente che, in effetti, la vostra subroutine verrà eseguita con il processore in modo supervisore. Terzo, la ID di canale passata a SD.EXTOP nel registro A0 viene convertita in un puntatore (in A0) alla base del blocco di definizione del canale corrispondente. Tutte queste operazioni forniscono alla vostra subroutine delle informazioni di vitale importanza che riguardano i puntatori, in un modo semplice e lineare. Quando al termine della subroutine viene eseguita l'istruzione di ritorno da subroutine RTS (notate: RTS, non RTE), nel registro A0 viene nuovamente caricata la ID di canale che era stata passata a SD.EXTOP.

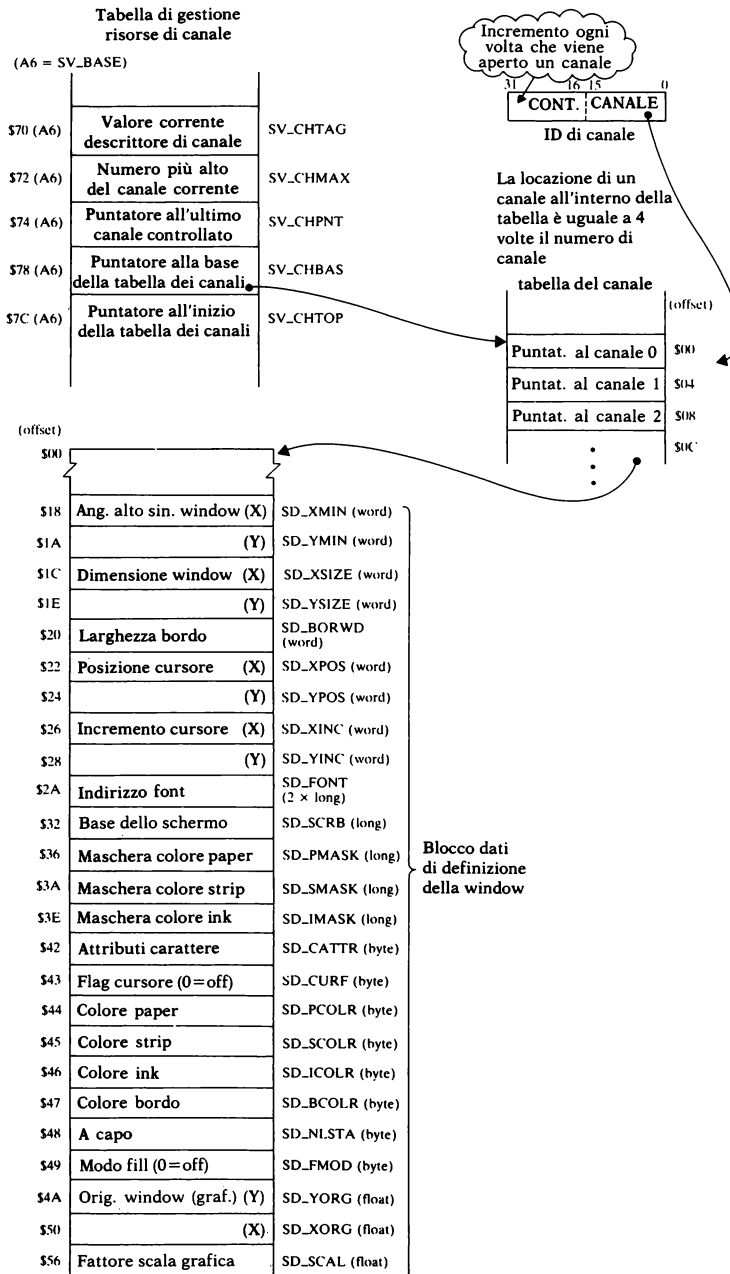


Figura 6.1 Canali associati allo schermo e blocchi di definizione

6.4 Colore

Un certo numero di procedure richiamabili tramite TRAP #3 consente di specificare un colore oltre ad altri parametri; una di queste procedure, per esempio, è quella che determina quale colore vada usato come ink (SD.SETIN). I driver di schermo utilizzano i colori in tre modi diversi: c'è il colore *paper* che corrisponde al colore dello sfondo; *ink* è il colore usato per le scritte e la grafica; *strip*, infine, definisce il colore usato per evidenziare le scritte.

I colori fondamentali che possono essere specificati con le due diverse opzioni di risoluzione sono:

| Modo 8 (256) | | Modo 4 (512) | |
|--------------|---------|--------------|--------|
| 0 | nero | 0 | nero |
| 1 | blu | 1 | nero |
| 2 | rosso | 2 | rosso |
| 3 | magenta | 3 | rosso |
| 4 | verde | 4 | verde |
| 5 | cyan | 5 | verde |
| 6 | giallo | 6 | bianco |
| 7 | bianco | 7 | bianco |

Oltre ai colori fondamentali può essere specificato uno *stipple*. Per stipple si intende una particolare distribuzione di colori in una matrice di 2×2 pixel che viene riprodotta in tutta l'area da colorare. Il tipo dello stipple può essere selezionato settando in modo appropriato dei particolari bit nel byte di colore (vedi Figura 6.2).

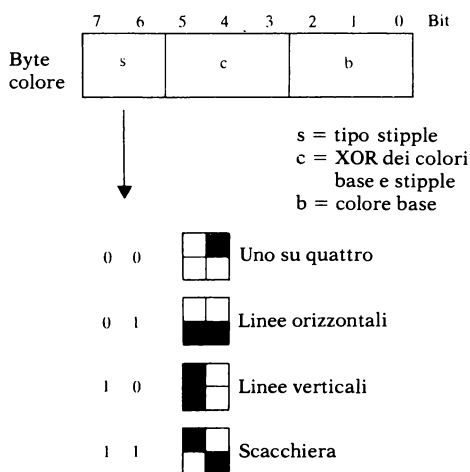


Figura 6.2 Descrizione del byte colore

I bit 7 e 6 specificano il tipo di stipple come indicato. I bit da 0 a 2 specificano il colore base (0..7) e i bit da 3 a 5 contengono un valore che corrisponde allo XOR (OR esclusivo) del colore base e del colore usato nello stipple. Supponiamo di volere il blu come colore base e il giallo come colore dello stipple. Il colore base ha come valore binario 001 e il colore dello stipple ha il valore binario 110. Lo XOR di questi due valori è il numero binario 111 e questo è il valore che deve essere caricato nei bit 3, 4 e 5 del byte di colore. Se volessimo anche uno stipple a scacchiera, l'intero byte di colore verrebbe codificato in binario come 11 111 001 che, in decimale, è il colore 249.

Questo metodo di codifica dell'organizzazione dei colori consente di ottenere i colori fondamentali in modo del tutto naturale. Per esempio, prendiamo il colore verde (codice 4). Il suo byte di colore è 00 000 100. Il tipo di stipple è 0 (un punto su quattro). Lo XOR dei valori deve dare zero, quindi il colore dello stipple non può essere che 4 (100 in binario), che è verde. Questa codifica ci dà lo stesso colore sia per la base che per lo stipple, perciò otteniamo il colore fondamentale che volevamo.

6.5 Uso dei registri del 68000

Le procedure che utilizzano la TRAP #3 vengono chiamate ponendo nel registro D0 (byte) il numero che identifica la particolare chiamata richiesta. Questo stesso registro viene anche usato per restituire al processo chiamante un codice di errore (long word). Se il codice di errore è diverso da zero ciò indica che si è verificato un errore. Gli errori standard sono indicati da piccoli (in valore assoluto) numeri negativi e sono elencati nell'appendice C. Se la chiamata tramite TRAP era riferita a una qualche forma di device driver supplementare, il codice di errore restituito può essere un puntatore a uno specifico messaggio di errore. Per evitare che i due tipi di codice di errore possano essere confusi, il codice di tipo puntatore punta in realtà a un indirizzo \$8000 byte al di sotto dell'indirizzo effettivo del messaggio di errore. Tutte le routine di QDOS possono, almeno in teoria, restituire il codice di errore ERR.BP (-15), che significa *bad parameter* (parametro illegale). La descrizione completa delle procedure di TRAP #3 stabilisce quali altri errori possano essere notificati. È ovviamente saggio controllare l'esistenza di una condizione di errore subito dopo che è stata effettuata la chiamata per mezzo di TRAP.

Oltre al registro D0, vengono usati anche i registri dati da D1 a D3 e i registri indirizzi da A0 ad A3 per passare valori da e verso le procedure QDOS. Quando gli opportuni registri sono stati predisposti per una data chiamata, l'accesso alla routine appropriata avviene semplicemente eseguendo l'istruzione TRAP #3. Nei casi in cui il descrittore del dato (cioè

.B, .W, o .L) non è indicato nella descrizione, il descrittore di default è .L (cioè long word).

La ID di canale, che indica quale canale debba essere usato per il trasferimento di I/O, viene sempre passata come long word in A0 e non viene mai alterata dalle procedure della TRAP #3. Il timeout viene sempre passato come word in D3 e anche questo dato non viene mai modificato dalla routine chiamata. Se il registro A1 punta a un array di byte quando viene generata la TRAP #3, al ritorno dalla procedura A1 punterà al byte successivo nell'array. I registri da D2 a D7, A0 e da A2 ad A7 non vengono mai alterati dalle procedure di I/O.

IO.PEND \$00 (0)

Controlla se sono pronti dati in input

Parametri in ingresso: D3.W Timeout
A0 ID di canale

Parametri in uscita: nessuno

Registri interessati: D1, A1

Errori possibili: ERR.NC (-1) nessun input in attesa
ERR.NO (-6) canale non aperto
ERR.EF (-10) fine del file

Descrizione:

Questa chiamata per mezzo di trap può essere usata per verificare se un canale ha dei dati in input pronti. Va tenuto presente che viene effettuata solo una verifica: i dati in input non vengono in alcun modo alterati né viene eseguita alcuna operazione di input.

IO.FBYTE \$01 (1)

Acquisisce un byte

Parametri in ingresso: D3.W Timeout
 A0 ID di canale

Parametri in uscita: D1.B Byte acquisito

Registri interessati: D1, A1

Errori possibili: ERR.NC (–1) operazione non completata
 ERR.NO (–6) canale non aperto
 ERR.EF (–10) fine del file

Descrizione:

Questa procedura acquisisce un byte dal canale di input specificato.

IO.FLINE \$02 (2)

Acquisisce una linea (terminata da <LF> ASCII)

Parametri in ingresso:

| | |
|------|----------------------|
| D2.W | Lunghezza del buffer |
| D3.W | Timeout |
| A0 | ID di canale |
| A1 | Base del buffer |

Parametri in uscita:

| | |
|------|--------------------------------|
| D1.W | Numero di byte acquisiti |
| A1 | Puntatore aggiornato al buffer |

Registri interessati: D1, A1

Errori possibili:

| | |
|--------------|---------------------------|
| ERR.NC (-1) | operazione non completata |
| ERR.BO (-5) | overflow del buffer |
| ERR.NO (-6) | canale non aperto |
| ERR.EF (-10) | fine del file |

Descrizione:

Questa procedura ha caratteristiche speciali quando viene utilizzata per l'input da console. In primo luogo, i caratteri letti dalla tastiera vengono riprodotti nella window appropriata. Secondo, si possono usare i tasti di controllo del cursore (← e →) per semplici operazioni di editing, così:

| | |
|--------|--|
| ← | muove il cursore a sinistra |
| → | muove il cursore a destra |
| CTRL ← | cancella un carattere a sinistra |
| CTRL → | cancella il carattere sotto il cursore |

Va osservato che il cursore, all'interno della window specificata, viene abilitato solo per la durata della chiamata alla procedura. Il numero di byte acquisiti, restituito in D1, include anche il terminatore di linea (<LF> ASCII) se viene incontrato.

Il line feed può esserci se il timeout è scaduto; non c'è mai se si verifica overflow del buffer: in tali casi il cursore viene lasciato abilitato.

Al termine della procedura, il puntatore contenuto nel registro A1 punterà al byte successivo all'ultimo byte acquisito.

IO.FSTRG \$03 (3)

Acquisisce una stringa di byte

Parametri in ingresso: D2.W Lunghezza del buffer
 D3.W Timeout
 A0 ID del canale
 A1 Base del buffer

Parametri in uscita: D1.W Numero di byte acquisiti
 A1 Puntatore al buffer aggiornato

Registri interessati: D1, A1

Errori possibili: ERR.NC (−1) operazione non completata
 ERR.NO (−6) canale non aperto
 ERR.EF (−10) fine del file

Descrizione:

Questa procedura acquisisce una stringa o blocco di byte dal canale di input specificato. I byte acquisiti non vengono riprodotti sullo schermo, anche se l'unità rappresentata dal canale è una window. Il ritorno dalla procedura avviene quando il timeout è scaduto o quando il buffer è pieno.

IO.EDLIN \$04 (4)

Edita una linea (solo da console)

Parametri in ingresso: D1.L Parametri di linea/cursore
D2.W Lunghezza del buffer
D3.W Timeout
A0 ID di canale
A1 Puntatore alla fine della linea

Parametri in uscita: D1 Parametri di linea/cursore
A1 Puntatore alla fine della linea

Registri interessati: D1, A1

Errori possibili: ERR.NC (-1) operazione non completata
ERR.BO (-5) overflow del buffer
ERR.NO (-6) canale non aperto

Descrizione:

Questa procedura è simile alla IO.FLINE (\$02) eccetto che all'utente viene fornita una linea di partenza sulla quale cominciare le operazioni di editing. All'atto della chiamata, il registro D1 deve contenere due word di informazioni che riguardano la linea: la word più significativa deve contenere la posizione corrente del cursore (0..n) e la word meno significativa deve specificare la lunghezza della linea. La linea su cui operare non deve contenere il carattere di terminazione (<LF> ASCII). All'utente viene resa disponibile solo la parte di linea compresa tra la posizione corrente del cursore e la fine della linea stessa.

All'uscita dalla procedura, i parametri nei registri D1 e A1 saranno stati modificati in conformità al nuovo stato della linea editata. I caratteri di terminazione validi sono <LF> e i due tasti di movimento cursore freccia in su e in giù. Il carattere di terminazione viene incluso nella linea (e quindi viene conteggiato nella lunghezza della linea) al termine della procedura. Il puntatore nel registro A1 punta sempre al byte successivo a quello corrispondente all'ultimo carattere inserito.

IO.SBYTE \$05 (5)

Invia un byte

Parametri in ingresso: D1.B Byte da inviare
 D3.W Timeout
 A0 ID di canale

Parametri in uscita: nessuno

Registri interessati: D1, A1

Errori possibili: ERR.NC (−1) operazione non completata
 ERR.OR (−4) superamento dei limiti
 ERR.NO (−6) canale non aperto
 ERR.DF (−11) unità completa

Descrizione:

Questa procedura invia un byte al canale di output specificato. L'output di un line feed ad una unità di tipo schermo o console viene trattato in un modo particolare: per prima cosa, viene inserito un carattere di a capo non appena viene ricevuto un line feed o quando il cursore raggiunge il lato destro della window. Se il cursore è disabilitato il carattere di a capo viene trattenuto; per rilasciarlo si può eseguire una delle seguenti operazioni:

1. Inviare un altro byte
2. Modificare la dimensione dei caratteri
3. Abilitare il cursore
4. Richiedere la posizione del cursore

Se il cursore viene posizionato esplicitamente, il carattere di a capo in sospenso viene cancellato. Va osservato che un a capo esplicito ne sostituisce sempre uno implicito (se è presente), fornendo così un output coerente, senza linee vuote indesiderate.

Il codice di errore ERR.OR (−4) viene restituito se è stata eseguita una operazione di salto a nuova linea.

IO.SSTRG \$07 (7)

Invia una stringa di byte

Parametri in ingresso: D2.W Numero di byte da inviare
D3.W Timeout
A0 ID di canale
A1 Base del buffer

Parametri in uscita: D1.W Numero di byte inviati
A1 Puntatore al buffer aggiornato

Registri interessati: D1, A1

Errori possibili: ERR.NC (– 1) operazione non completata
ERR.NO (– 6) canale non aperto
ERR.DF (– 11) unità completa

Descrizione:

Questa procedura invia una stringa di byte al canale specificato. L'output di un line feed ad una unità di tipo schermo o console viene trattato in un modo particolare: per prima cosa, viene inserito un carattere di a capo quando viene ricevuto un line feed o quando il cursore raggiunge il margine destro della window. Se il cursore è disabilitato, il salto a nuova linea viene sospeso; per rilasciarlo, è possibile eseguire una delle seguenti operazioni:

1. Inviare un'altra stringa di byte
2. Modificare la dimensione dei caratteri
3. Abilitare il cursore
4. Richiedere la posizione del cursore

Se il cursore viene posizionato esplicitamente, il salto a nuova linea in sospeso viene cancellato. Va osservato in particolare che un carattere di a capo esplicito cancella un eventuale salto a nuova linea implicito; viene così fornito un output coerente, privo di linee vuote indesiderate.

SD.EXTOP \$09 (9)

Chiama una operazione estesa

Parametri in ingresso:

| | |
|------|--------------------------|
| D1 | Parametro (se richiesto) |
| D2 | Parametro (se richiesto) |
| D3.W | Timeout |
| A0 | ID di canale |
| A1 | Parametro (se richiesto) |
| A2 | Indirizzo della routine |

Parametri in uscita:

| | |
|----|----------------------|
| D1 | Parametro (se usato) |
| A1 | Parametro (se usato) |

Registri interessati: D1, A1

Errori possibili:

| | |
|--------|--------------------------------|
| ERR.NC | (-1) operazione non completata |
| ERR.NO | (-6) canale non aperto |

Descrizione:

Questa chiamata tramite TRAP fornisce un meccanismo per accedere in modo supervisore a una routine scritta dall'utente. La routine indicata nella chiamata deve essere compatibile con le regole previste per i device driver. La descrizione dettagliata delle routine di gestione delle periferiche esula dagli scopi di questo libro, ma il paragrafo 6.3 presenta i punti ritenuti interessanti in questo contesto.

I registri disponibili per il passaggio dei parametri (cioè i tre per il passaggio di parametri verso la subroutine e i due per i parametri che la subroutine restituisce) non devono essere necessariamente usati. Ciò che intendiamo dire è che, semplicemente, i dati contenuti in quei registri non vengono alterati nell'interfaccia tra la routine SD.EXTOP e la routine fornita dall'utente.

SD.PXENQ \$0A (10)

Restituisce la dimensione di una data window e la posizione del cursore in coordinate riferite ai pixel

Parametri in ingresso: D3.W Timeout
 A0 ID di canale
 A1 Base del blocco delle richieste

Parametri in uscita: (Blocco delle richieste aggiornato)

Registri interessati: A1

Errori possibili: ERR.NC (-1) operazione non completata
 ERR.NO (-6) canale non aperto

Descrizione:

A questa procedura deve essere passato un blocco delle richieste che consiste di quattro word:

| Offset nel blocco | Significato |
|--------------------------|-----------------------|
| \$00 | Dimensione window (X) |
| \$02 | Dimensione window (Y) |
| \$04 | Posizione cursore (X) |
| \$06 | Posizione cursore (Y) |

L'angolo in alto a sinistra della window corrisponde alla coordinata (0,0). Se nella coda di output di una window esiste un salto a nuova linea in attesa, esso viene attivato (cioè, rilasciato).

SD.CHENQ \$0B (11)

Restituisce la dimensione di una window e la posizione del cursore in coordinate riferite ai caratteri

Parametri in ingresso: D3.W Timeout
 A0 ID di canale
 A1 Base del blocco delle richieste

Parametri in uscita: (Blocco delle richieste aggiornato)

Registri interessati: A1

Errori possibili: ERR.NC (– 1) operazione non completata
 ERR.NO (– 6) canale non aperto

Descrizione:

A questa procedura deve essere passato un blocco delle richieste che consiste di quattro word:

| Offset nel blocco | Significato |
|--------------------------|-----------------------|
| \$00 | Dimensione window (X) |
| \$02 | Dimensione window (Y) |
| \$04 | Posizione cursore (X) |
| \$06 | Posizione cursore (Y) |

L'angolo in alto a sinistra della window corrisponde alla coordinata (0,0). Se nella coda di output di una window esiste un salto a nuova linea in attesa, esso viene attivato (cioè, rilasciato).

SD.BORDR \$0C (12)

Assegna la larghezza e il colore del bordo

Parametri in ingresso: D1.B Colore
 D2.W Larghezza
 D3.W Timeout
 A0 ID di canale

Parametri in uscita: nessuno

Registri interessati: D1

Errori possibili: ERR.NC (– 1) operazione non completata
 ERR.NO (– 6) canale non aperto

Descrizione:

Questa procedura ridefinisce il bordo della window specificata. La larghezza di default del bordo è zero; quando il bordo viene creato, esso viene posto all'interno dei limiti della window e gli spigoli verticali hanno larghezza doppia. Tutti i successivi riferimenti allo schermo avranno a disposizione delle dimensioni di window ridotte (eccetto una successiva chiamata a SD.BORDR). Se la larghezza del bordo viene modificata, il cursore viene inviato nella posizione di home (angolo superiore sinistro). I codici standard dei colori sono descritti nel paragrafo 6.4. Esiste anche il colore \$80 (128) che viene trattato in un modo speciale e consente di creare un bordo trasparente che lascia intatto il contenuto originale del bordo.

SD.WDEF \$0D (13)

Definisce una window e il corrispondente bordo

Parametri in ingresso: D1.B Colore del bordo
 D2.W Larghezza del bordo
 D3.W Timeout
 A0 ID di canale
 A1 Base del blocco della window

Parametri in uscita: nessuno

Registri interessati: D1, A1

Errori possibili: ERR.NC (–1) operazione non completata
 ERR.OR (–4) window troppo grande
 ERR.NO (–6) canale non aperto

Descrizione:

Questa chiamata viene usata per ridefinire la forma e la posizione di una window. I contenuti originali della window non vengono modificati né spostati, ma il cursore viene posizionato nell'angolo in alto a sinistra della nuova window.

Il blocco di definizione delle window consiste di quattro word che devono essere predisposte prima che venga eseguita la chiamata tramite TRAP. Il contenuto delle word è:

| Offset nel blocco | Significato |
|--------------------------|-----------------------|
| \$00 | Dimensione window (X) |
| \$02 | Dimensione window (Y) |
| \$04 | Origine window (X) |
| \$06 | Origine window (Y) |

L'origine della window corrisponde all'angolo in alto a sinistra della window stessa.

SD.CURE \$0E (14)

Abilita il cursore

Parametri in ingresso: D3.W Timeout
 A0 ID di canale

Parametri in uscita: nessuno

Registri interessati: D1, A1

Errori possibili: ERR.NC (– 1) operazione non completata
 ERR.NO (– 6) canale non aperto

Descrizione:

Questa procedura abilita il cursore nel canale della window specificata. Osserviamo che il cursore viene automaticamente abilitato ogni volta che viene chiamata una procedura di lettura linea (IO.FLINE) o di editing di linea (IO.EDLIN).

SD.CURS \$0F (15)

Disabilita il cursore

Parametri in ingresso: D3.W Timeout
 A0 ID di canale

Parametri in uscita: nessuno

Registri interessati: D1, A1

Errori possibili: ERR.NC (–1) operazione non completata
 ERR.NO (–6) canale non aperto

Descrizione:

Questa procedura disabilita il cursore associato al canale della window specificata. Osserviamo che il cursore viene automaticamente disabilitato ogni volta che una procedura di lettura linea (IO.FLINE) o editing di linea (IO.EDLIN) termina normalmente.

SD.POS \$10 (16)

Sposta il cursore a posizione assoluta utilizzando coordinate riferite ai caratteri

Parametri in ingresso: D1.W Posizione colonna
D2.W Posizione riga
D3.W Timeout
A0 ID di canale

Parametri in uscita: nessuno

Registri interessati: D1, A1

Errori possibili: ERR.NC (−1) operazione non completata
ERR.OR (−4) non è nella window
ERR.NO (−6) canale non aperto

Descrizione:

Questa procedura posiziona il cursore nella posizione assoluta specificata. L'angolo in alto a sinistra della window corrisponde alla posizione (0,0).

Se esiste un salto a nuova linea in sospeso, esso viene cancellato da questa chiamata. La posizione originale del cursore non viene alterata se si verifica un errore.

SD.TAB \$11 (17)

Tabulazione

Parametri in ingresso: D1.W Posizione colonna
 D3.W Timeout
 A0 ID di canale

Parametri in uscita: nessuno

Registri interessati: D1, A1

Errori possibili: ERR.NC (– 1) operazione non completata
 ERR.OR (– 4) non è nella window
 ERR.NO (– 6) canale non aperto

Descrizione:

Questa procedura posiziona il cursore alla colonna di tabulazione specificata. La posizione indicata può essere in qualunque punto della linea corrente del cursore.

Se esiste un salto a nuova linea in sospeso, esso viene cancellato da questa chiamata. La posizione originale del cursore non viene modificata se si verifica un errore.

SD.NL \$12 (18)

Salto a nuova linea

Parametri in ingresso: D3.W Timeout
A0 ID di canale

Parametri in uscita: nessuno

Registri interessati: D1, A1

Errori possibili: ERR.NC (-1) operazione non completata
ERR.OR (-4) non è nella window
ERR.NO (-6) canale non aperto

Descrizione:

Questa procedura provoca un salto a nuova linea nel canale associato alla window specificata.

Se esiste un salto a nuova linea in sospeso, esso viene cancellato da questa chiamata. La posizione originale del cursore non viene modificata se si verifica un errore.

SD.PCOL \$13 (19)

Fa arretrare il cursore di una posizione

Parametri in ingresso: D3.W Timeout
 A0 ID di canale

Parametri in uscita: nessuno

Registri interessati: D1, A1

Errori possibili: ERR.NC (–1) operazione non completata
 ERR.OR (–4) non è nella window
 ERR.NO (–6) canale non aperto

Descrizione:

Questa procedura arretra il cursore di una posizione senza effetti distruttivi (cioè il cursore non cancella il carattere che va a ricoprire).

Se esiste un salto a nuova linea in sospeso, esso viene cancellato da questa chiamata. La posizione originale del cursore non viene alterata se si verifica un errore.

SD.NCOL \$14 (20)

Fa avanzare il cursore di una posizione

Parametri in ingresso: D3.W Timeout
A0 ID di canale

Parametri in uscita: nessuno

Registri interessati: D1, A1

Errori possibili: ERR.NC (−1) operazione non completata
ERR.OR (−4) non è nella window
ERR.NO (−6) canale non aperto

Descrizione:

Questa procedura avanza il cursore di una posizione, senza effetti distruttivi.

Se esiste un salto a nuova linea in sospenso, esso viene cancellato da questa chiamata. La posizione originale del cursore non viene modificata se si verifica un errore.

SD.PROW \$15 (21)

Sposta il cursore verso l'alto di una posizione

Parametri in ingresso: D3.W Timeout
 A0 ID di canale

Parametri in uscita: nessuno

Registri interessati: D1, A1

Errori possibili: ERR.NC (–1) operazione non completata
 ERR.OR (–4) non è nella window
 ERR.NO (–6) canale non aperto

Descrizione:

Questa procedura sposta il cursore verso l'alto di una posizione senza effetti distruttivi. Il cursore viene posizionato sulla stessa colonna che occupava prima della chiamata.

Se esiste un salto a nuova linea in sospeso, esso viene cancellato da questa chiamata. La posizione originale del cursore non viene modificata se si verifica un errore.

SD.NROW \$16 (22)

Sposta il cursore verso il basso di una posizione

Parametri in ingresso: D3.W Timeout
A0 ID di canale

Parametri in uscita: nessuno

Registri interessati: D1, A1

Errori possibili: ERR.NC (-1) operazione non completata
ERR.OR (-4) non è nella window
ERR.NO (-6) canale non aperto

Descrizione:

Questa procedura sposta il cursore verso il basso di una posizione senza effetti distruttivi. Il cursore viene posizionato sulla stessa colonna che occupava prima della chiamata. Se esiste un salto a nuova linea in sospeso, esso viene cancellato da questa chiamata. La posizione del cursore non viene modificata se si verifica un errore.

SD.PIXP \$17 (23)

Sposta il cursore in posizione assoluta utilizzando coordinate riferite ai pixel

Parametri in ingresso: D1.W Coordinata X
 D2.W Coordinata Y
 D3.W Timeout
 A0 ID di canale

Parametri in uscita: nessuno

Registri interessati: D1, A1

Errori possibili: ERR.NC (–1) operazione non completata
 ERR.OR (–4) non è nella window
 ERR.NO (–6) canale non aperto

Descrizione:

Questa procedura posiziona il cursore secondo coordinate assolute. L'angolo in alto a sinistra della window corrisponde alle coordinate (0,0). Le coordinate in termini di pixel devono corrispondere all'angolo in alto a sinistra del rettangolo contenente il carattere richiesto (di destinazione). Se esiste un salto a nuova linea in sospeso, esso viene cancellato da questa chiamata. La posizione originale del cursore non viene modificata se si verifica un errore.

SD.SCROL \$18 (24)

Fa scorrere il contenuto di una window verso l'alto o il basso

Parametri in ingresso: D1.W Ampiezza dello scorrimento
D3.W Timeout
A0 ID di canale

Parametri in uscita: nessuno

Registri interessati: D1, A1

Errori possibili: ERR.NC (-1) operazione non completata
ERR.NO (-6) canale non aperto

Descrizione:

Questa procedura fa scorrere il contenuto della window specificata. Per ottenere uno scorrimento verso l'alto è sufficiente specificare un'ampiezza negativa. L'ampiezza dello scorrimento viene sempre specificata in termini di pixel. Le righe di pixel lasciate vuote vengono riempite con il colore paper.

La posizione del cursore non viene alterata.

SD.SCRTP \$19 (25)

Fa scorrere in senso verticale la parte alta di una window

Parametri in ingresso: D1.W Ampiezza dello scorrimento
 D3.W Timeout
 A0 ID di canale

Parametri in uscita: nessuno

Registri interessati: D1, A1

Errori possibili: ERR.NC (–1) operazione non completata
 ERR.NO (–6) canale non aperto

Descrizione:

Questa procedura provoca lo scorrimento in senso verticale della parte superiore della window specificata. Per ottenere uno scorrimento verso il basso è sufficiente specificare un'ampiezza negativa. L'ampiezza dello scorrimento viene sempre specificata in termini di pixel. Le righe di pixel lasciate vuote vengono riempite con il colore paper.

Come "parte superiore di una window" si definisce quella parte di window che sta al di sopra della linea che contiene il cursore (la linea del cursore è esclusa). La posizione del cursore non viene alterata.

SD.SCRBT \$1A (26)

Fa scorrere in senso verticale la parte bassa di una window

Parametri in ingresso: D1.W Ampiezza dello scorrimento
 D3.W Timeout
 A0 ID di canale

Parametri in uscita: nessuno

Registri interessati: D1, A1

Errori possibili: ERR.NC (-1) operazione non completata
 ERR.NO (-6) canale non aperto

Descrizione:

Questa procedura provoca lo scorrimento in senso verticale della parte inferiore della window specificata. Per ottenere uno scorrimento verso il basso è sufficiente specificare un'ampiezza negativa. L'ampiezza dello scorrimento viene sempre specificata in termini di pixel. Le righe di pixel lasciate vuote vengono riempite con il colore paper.

Come "parte inferiore di una window" si definisce quella parte di window che sta al di sotto della linea che contiene il cursore (la linea del cursore è esclusa). La posizione del cursore non viene alterata.

SD.PAN \$1B (27)

Fa scorrere in senso orizzontale una window

Parametri in ingresso: D1W Ampiezza dello scorrimento
 D3.W Timeout
 A0 ID di canale

Parametri in uscita: nessuno

Registri interessati: D1, A1

Errori possibili: ERR.NC (−1) operazione non completata
 ERR.NO (−6) canale non aperto

Descrizione:

Questa procedura provoca lo scorrimento in senso orizzontale della window specificata. Per ottenere uno scorrimento verso destra è sufficiente specificare un'ampiezza negativa.

L'ampiezza dello scorrimento viene sempre specificata in termini di pixel. Le colonne di pixel lasciate vuote vengono riempite con il colore paper.

La posizione del cursore non viene alterata.

SD.PANLN \$1E (30)

Fa scorrere la linea che contiene il cursore

Parametri in ingresso: D1.W Ampiezza dello scorrimento
 D3.W Timeout
 A0 ID di canale

Parametri in uscita: nessuno

Registri interessati: D1, A1

Errori possibili: ERR.NC (–1) operazione non completata
 ERR.NO (–6) canale non aperto

Descrizione:

Questa procedura provoca lo scorrimento in senso orizzontale della linea che contiene il cursore nella window specificata. Per ottenere uno scorrimento verso destra è sufficiente specificare un'ampiezza negativa. L'ampiezza dello scorrimento viene sempre specificata in termini di pixel. Le colonne di pixel lasciate vuote vengono riempite con il colore paper. L'altezza della linea del cursore dipende dalla dimensione del font dei caratteri (cioè 10 o 20 righe di pixel). La posizione del cursore non viene alterata.

SD.PANRT \$1F (31)

Fa scorrere la parte destra della linea che contiene il cursore

Parametri in ingresso: D1.W Ampiezza dello scorrimento
 D3.W Timeout
 A0 ID di canale

Parametri in uscita: nessuno

Registri interessati: D1, A1

Errori possibili: ERR.NC (–1) operazione non completata
 ERR.NO (–6) canale non aperto

Descrizione:

Questa procedura provoca lo scorrimento in senso orizzontale della parte destra della linea che contiene il cursore nella window specificata. Per ottenere uno scorrimento verso sinistra è sufficiente specificare un'ampiezza negativa. L'ampiezza dello scorrimento viene sempre specificata in termini di pixel. Le colonne di pixel lasciate vuote vengono riempite con il colore paper.

L'altezza della linea del cursore dipende dalla dimensione del font dei caratteri (cioè 10 o 20 righe di pixel). La parte destra della linea comprende il carattere corrispondente alla posizione del cursore. La posizione del cursore non viene alterata.

SD.CLEAR \$20 (32)

Cancella il contenuto di una window

Parametri in ingresso: D3.W Timeout
A0 ID di canale

Parametri in uscita: nessuno

Registri interessati: D1, A1

Errori possibili: ERR.NC (-1) operazione non completata
ERR.NO (-6) canale non aperto

Descrizione:

Questa procedura cancella il contenuto della window specificata. La window viene riempita con il colore paper.

SD.CL RTP \$21 (33)

Cancella la parte alta di una window

Parametri in ingresso: D3.W Timeout
 A0 ID di canale

Parametri in uscita: nessuno

Registri interessati: D1, A1

Errori possibili: ERR.NC (– 1) operazione non completata
 ERR.NO (– 6) canale non aperto

Descrizione:

Questa procedura cancella la parte alta della window specificata. Le posizioni dei pixel cancellati vengono riempite con il colore paper. Come “parte alta di una window” si intende l’area di una window al di sopra della linea che contiene il cursore (la linea del cursore è esclusa). La posizione del cursore non viene alterata.

SD.CLRBT \$22 (34)

Cancella la parte bassa di una window

Parametri in ingresso: D3.W Timeout
 A0 ID di canale

Parametri in uscita: nessuno

Registri interessati: D1, A1

Errori possibili: ERR.NC (–1) operazione non completata
 ERR.NO (–6) canale non aperto

Descrizione:

Questa procedura cancella la parte bassa della window specificata. Le posizioni dei pixel cancellati vengono riempite con il colore paper. Come “parte bassa di una window” si intende l’area di una window al di sotto della linea che contiene il cursore (la linea del cursore è esclusa). La posizione del cursore non viene alterata.

SD.CLRLN \$23 (35)

Cancella la linea che contiene il cursore

Parametri in ingresso: D3.W Timeout
 A0 ID di canale

Parametri in uscita: nessuno

Registri interessati: D1, A1

Errori possibili: ERR.NC (–1) operazione non completata
 ERR.NO (–6) canale non aperto

Descrizione:

Questa procedura cancella la linea che contiene il cursore nella window specificata. Le posizioni dei pixel cancellati vengono riempite con il colore paper.

L'altezza della linea che contiene il cursore dipende dalla dimensione del font dei caratteri (cioè 10 o 20 righe di pixel). La posizione del cursore non viene alterata.

SD.CLRRT \$24 (36)

Cancella la parte destra della linea che contiene il cursore

Parametri in ingresso: D3.W Timeout
 A0 ID di canale

Parametri in uscita: nessuno

Registri interessati: D1, A1

Errori possibili: ERR.NC (–1) operazione non completata
 ERR.NO (–6) canale non aperto

Descrizione:

Questa procedura cancella la parte destra della linea che contiene il cursore nella window specificata. Le posizioni dei pixel cancellati vengono riempite con il colore paper.

L'altezza della linea che contiene il cursore dipende dalla dimensione del font dei caratteri (cioè 10 o 20 righe di pixel). La parte destra comprende il carattere corrispondente alla posizione del cursore. La posizione del cursore non viene alterata.

SD.FONT \$25 (37)

Passa da un font di caratteri all'altro

Parametri in ingresso: D3.W Timeout
 A0 ID di canale
 A1 Base del primo font
 A2 Base del secondo font

Parametri in uscita: nessuno

Registri interessati: D1, A1

Errori possibili: ERR.NC (-1) operazione non completata
 ERR.NO (-6) canale non aperto

Descrizione:

Lo spazio in cui può essere definito un carattere consiste in un array di 5×9 pixel all'interno di un rettangolo di spazio di 6×10 pixel. La riga superiore e la colonna destra del carattere sono vuote, per definizione. Il QL dispone normalmente di due diversi font, ma altri possono essere aggiunti, se necessario. I font standard forniscono i caratteri corrispondenti ai codici da \$20 a \$7F.

Se, all'atto della chiamata, gli indirizzi delle basi sono zero, vengono utilizzati i font standard. Il cambiamento di font non altera il contenuto corrente dello schermo.

Le tabelle di definizione dei font devono avere la seguente struttura:

| Offset | Significato |
|-----------|--|
| \$00 | Il più basso carattere valido nel font (ad esempio: \$20 <spazio>) |
| \$01 | Numero di caratteri validi — byte |
| \$02-\$0A | Nove byte che specificano il primo carattere valido |
| \$0B-\$13 | Nove byte che specificano il secondo carattere valido |
| \$14-\$1C | ecc. |

La Figura 6.3 illustra come vengono usati i nove byte della definizione di ciascun carattere. I bit 7, 1 e 0 devono sempre essere uguali a zero. Il byte 0 va collocato per primo nella tabella di definizione del font.

Se il carattere che deve essere visualizzato è illegale per quanto riguarda il primo font (cioè è al di fuori del range ammesso), esso viene prelevato

dal secondo font. Se è illegale per il secondo font, viene usato il carattere valido più basso del secondo font.

Esempio:

Supponendo che il carattere della Figura 6.3 sia il secondo carattere valido del font, i byte della sua definizione sono:

| Offset | Byte |
|--------|------|
| \$0B | \$40 |
| \$0C | \$20 |
| \$0D | \$10 |
| \$0E | \$08 |
| \$0F | \$04 |
| \$10 | \$08 |
| \$11 | \$10 |
| \$12 | \$20 |
| \$13 | \$40 |

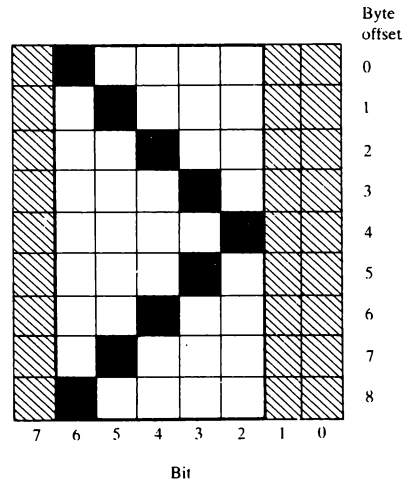


Figura 6.3 Definizione di un nuovo carattere

Tenete presente che con questo sistema non viene ridefinito il font di default, semplicemente viene creato un font interamente nuovo.

Non ci sono limiti, a parte la dimensione fisica della memoria, al numero di set di caratteri che possono essere definiti.

SD.RECOL \$26 (38)

Ricolora una window

Parametri in ingresso: D3.W Timeout
 A0 ID di canale
 A1 Puntatore alla lista dei colori

Parametri in uscita: nessuno

Registri interessati: D1, A1

Errori possibili: ERR.NC (–1) operazione non completata
 ERR.NO (–6) canale non aperto

Descrizione:

È possibile ricolorare il contenuto di una window per mezzo di questa procedura. Le informazioni visualizzate nella window non subiscono alterazioni. È possibile definire un massimo di otto colori per ciascun pixel di ciascun canale tipo screen e, all'atto della chiamata alla procedura, il registro A1 deve puntare a una lista di colori di otto byte che definisce il colore nuovo (valori 0..7) per ogni possibile colore originale:

| Offset | Significato |
|---------------|----------------------------------|
| 0 | Nuovo colore per i pixel neri |
| 1 | Nuovo colore per i pixel blu |
| 2 | Nuovo colore per i pixel rossi |
| 3 | Nuovo colore per i pixel magenta |
| 4 | Nuovo colore per i pixel verdi |
| 5 | Nuovo colore per i pixel cyan |
| 6 | Nuovo colore per i pixel gialli |
| 7 | Nuovo colore per i pixel bianchi |

A questo punto val la pena di notare due cose. Primo, la tabella qui sopra si riferisce solo allo schermo nel modo a otto colori. Nel modo a quattro colori possono essere definiti solo i byte 0, 2, 4 e 6 per ricolorare il nero, il rosso, il verde e il bianco. Secondo, in questo contesto ci occupiamo solo di ricolorare pixel per pixel. Ovviamente, a un singolo pixel non può essere associato uno stipple, perciò il range dei colori da specificare va da 0 a 7. Qualsiasi stipple presente sullo schermo viene ricolorato secondo le alterazioni subite dal colore dei pixel che lo compongono.

SD.SETPA \$27 (39)

Assegna il colore paper

Parametri in ingresso: D1.B Colore
 D3.W Timeout
 A0 ID di canale

Parametri in uscita: nessuno

Registri interessati: A1

Errori possibili: ERR.NC (–1) operazione non completata
 ERR.NO (–6) canale non aperto

Descrizione:

Questa procedura assegna il colore paper (cioè il colore dello sfondo) per la window associata al canale specificato. Può essere specificato qualunque colore, stipple compreso (vedi paragrafo 6.4).

SD.SETST \$28 (40)

Assegna il colore strip

Parametri in ingresso: D1.B Colore
 D3.W Timeout
 A0 ID di canale

Parametri in uscita: nessuno

Registri interessati: A1

Errori possibili: ERR.NC (–1) operazione non completata
 ERR.NO (–6) canale non aperto

Descrizione:

Questa procedura assegna il colore di evidenziazione (cioè lo sfondo locale per la visualizzazione dei caratteri) alla window associata al canale specificato. Può essere specificato qualunque colore, stipple compreso (vedi paragrafo 6.4).

SD.SETIN \$29 (41)

Assegna il colore ink

Parametri in ingresso: D1.B Colore
 D3.W Timeout
 A0 ID di canale

Parametri in uscita: nessuno

Registri interessati: A1

Errori possibili: ERR.NC (−1) operazione non completata
 ERR.NO (−6) canale non aperto

Descrizione:

Questa procedura assegna il colore ink (cioè il colore usato nella visualizzazione dei caratteri e nei tracciamenti di linee e punti nella grafica) per la window associata al canale specificato. Può essere specificato qualunque colore, stipple compreso (vedi paragrafo 6.4).

SD.SETFL \$2A (42)

Attiva / disattiva il modo flash

Parametri in ingresso: D1.B Flag di attributo
 D3.W Timeout
 A0 ID di canale

Parametri in uscita: nessuno

Registri interessati: D1, A1

Errori possibili: ERR.NC (–1) operazione non completata
 ERR.NO (–6) canale non aperto

Descrizione:

Questa procedura può essere usata per attivare o disattivare il modo flash per la window specificata. Se il flag di attributo è zero, all'atto della chiamata, il modo flash viene disattivato, e viceversa. Osserviamo che l'attivazione del modo flash influenza solo i caratteri visualizzati successivamente (e non la grafica). Le informazioni contenute nella window non vengono alterate. Allo stesso modo, quando il modo flash viene disattivato i caratteri lampeggianti continuano a lampeggiare.

SD.SETUL \$2B (43)

Attiva / disattiva la sottolineatura

Parametri in ingresso: D1.B Flag di attributo
D3.W Timeout
A0 ID di canale

Parametri in uscita: nessuno

Registri interessati: D1, A1

Errori possibili: ERR.NC (-1) operazione non completata
ERR.NO (-6) canale non aperto

Descrizione:

Questa procedura permette di attivare o disattivare la sottolineatura per la window specificata. Se il flag di attributo è zero all'atto della chiamata, la sottolineatura viene disattivata, e viceversa.

Osserviamo che l'attivazione della sottolineatura influenza solo i caratteri visualizzati successivamente (e non la grafica). Il contenuto della window non viene alterato. Allo stesso modo, quando la sottolineatura viene disattivata, i caratteri già sottolineati rimangono sottolineati.

SD.SETMD \$2C (44)

Modifica il modo di scrittura / grafica

Parametri in ingresso: D1.W Byte di modo
 D3.W Timeout
 A0 ID di canale

Parametri in uscita: nessuno

Registri interessati: D1, A1

Errori possibili: ERR.NC (–1) operazione non completata
 ERR.NO (–6) canale non aperto

Descrizione:

Questa procedura influenza le successive operazioni di scrittura e grafiche. Il byte di modo può assumere uno dei seguenti tre valori:

| Modo | Effetto |
|-------------|---|
| –1 | Viene eseguito lo XOR tra il colore ink e il colore dello sfondo |
| 0 | Ai caratteri viene assegnato come sfondo il colore di evidenziazione e la grafica viene eseguita con il colore ink |
| 1 | Lo sfondo dei caratteri diventa trasparente (solo ciò che rappresenta effettivamente i caratteri va a sostituire il contenuto precedente dello schermo); la grafica viene eseguita con il colore ink. |

SD.SETSZ \$2D (45)

Stabilisce la dimensione dei caratteri

Parametri in ingresso: D1.W Larghezza / spaziatura dei caratteri
 D2.W Altezza / spaziatura dei caratteri
 D3.W Timeout
 A0 ID di canale

Parametri in uscita: nessuno

Registri interessati: D1, A1

Errori possibili: ERR.NC (-1) operazione non completata
 ERR.NO (-6) canale non aperto

Descrizione:

Il generatore di caratteri del QL dispone di due altezze e di due larghezze dei caratteri. Nel modo a otto colori, possono essere usati solo i caratteri di larghezza doppia, altezza singola o doppia. In alternativa, sono già disponibili direttamente altre due spaziature in larghezza. In teoria la spaziatura dei caratteri è completamente flessibile, ma per sfruttare questa flessibilità bisogna accedere alle variabili SD.XINC e SD.YINC nel blocco di definizione della window.

I parametri di ingresso che specificano la larghezza, l'altezza e la spaziatura possono assumere i seguenti valori:

| D1.W | Carattere | |
|------|-----------|------------|
| | Larghezza | Spaziatura |
| 0 | singola | 6 pixel |
| 1 | singola | 8 pixel |
| 2 | doppia | 12 pixel |
| 3 | doppia | 16 pixel |

| D2.W | Carattere | |
|------|-----------|------------|
| | Altezza | Spaziatura |
| 0 | singola | 10 pixel |
| 1 | doppia | 20 pixel |

Dato che nel modo a otto colori sono previsti solo i caratteri di larghezza doppia, una chiamata a questa procedura (nel modo a otto colori) con

$D1=0$ o 1 ha lo stesso effetto che avrebbe se $D1$ fosse stato uguale a 2 o 3 , rispettivamente.

Osserviamo che se si passa dal modo a otto colori a quello a quattro colori, o viceversa, la larghezza passa da doppia a singola, coerentemente al cambiamento di modo. L'altezza dei caratteri, invece, rimane invariata.

SD.FILL \$2E (46)

Riempie un rettangolo

Parametri in ingresso: D1.B Colore
 D3.W Timeout
 A0 ID di canale
 A1 Base del blocco di definizione

Parametri in uscita: nessuno

Registri interessati: D1, A1

Errori possibili: ERR.NC (−1) operazione non completata
 ERR.OR (−4) blocco al di fuori della window
 ERR.NO (−6) canale non aperto

Descrizione:

Questa procedura riempie un blocco rettangolare, nella window associata al canale specificato, con il colore indicato. Può essere indicato qualsiasi colore, stripple compreso (vedi paragrafo 6.4).

Quando la procedura viene chiamata, il registro A1 deve contenere un puntatore a un blocco di definizione composto da quattro word che definisce la dimensione e la posizione del rettangolo:

| Offset | Significato |
|--------|---|
| 0 | Larghezza del rettangolo |
| 1 | Altezza del rettangolo |
| 2 | Coordinata X dell'angolo superiore sinistro |
| 3 | Coordinata Y dell'angolo superiore sinistro |

Le coordinate X e Y devono essere specificate rispetto all'origine della window.

Questa procedura può essere utilizzata per eseguire il tracciamento veloce di linee orizzontali e verticali. Inoltre, può essere usata come procedura veloce di tipo "ricolora blocco in window", assegnando −1 (modo XOR) al modo carattere/disegno prima di eseguire la chiamata (vedi SD.SETMD; D0=\$2C).

SD.POINT \$30 (48)

Disegna un punto

Parametri in ingresso: D3.W Timeout
 A0 ID di canale
 A1 Puntatore allo stack aritmetico

Parametri in uscita: nessuno

Registri interessati: D1, A1

Errori possibili: ERR.NC (-1) operazione non completata
 ERR.NO (-6) canale non aperto

Descrizione:

Quando questa procedura viene chiamata, A1 deve puntare ad uno stack aritmetico locale per i parametri di almeno 240 byte. Sulla cima dello stack ci devono essere due parametri in virgola mobile (sei byte ciascuno) con il seguente significato:

| Posizione sullo stack | Significato |
|------------------------------|--------------------|
| 0(A1) | Y del punto |
| 6(A1) | X del punto |

Ricordate che gli stack si accrescono verso il basso, quindi la cima dello stack si trova a un indirizzo fisico più basso rispetto alla base dello stack.

Le coordinate passate sono relative ad un'origine arbitraria (default (0,0)) e a una scala arbitraria (default 0..100). Se viene specificato un punto che è al di fuori della window specificata, quel punto non viene disegnato; in tale caso, non viene riportato alcun errore.

SD.LINE \$31 (49)

Traccia una linea

Parametri in ingresso: D3.W Timeout
 A0 ID di canale
 A1 Puntatore allo stack aritmetico

Parametri in uscita: nessuno

Registri interessati: D1, A1

Errori possibili: ERR.NC (-1) operazione non completata
 ERR.NO (-6) canale non aperto

Descrizione:

Quando questa procedura viene chiamata, A1 deve puntare ad uno stack aritmetico locale per i parametri di almeno 240 byte. Sulla cima dello stack ci devono essere quattro parametri in virgola mobile (sei byte ciascuno) con il seguente significato:

| Posizione sullo stack | Significato |
|------------------------------|---------------------------|
| 0(A1) | Y della fine della linea |
| 6(A1) | X della fine della linea |
| C(A1) | Y dell'inizio della linea |
| 12(A1) | X dell'inizio della linea |

Ricordate che gli stack si accrescono verso il basso, quindi la cima dello stack si trova a un indirizzo fisico più basso rispetto alla base dello stack.

Le coordinate passate sono relative ad un'origine arbitraria (default (0,0)) e ad una scala arbitraria (default 0..100). Se parte della linea è al di fuori della window specificata, quella parte non viene disegnata. In tale caso non viene riportato alcun errore.

SD.ARC \$32 (50)

Traccia un arco

Parametri in ingresso: D3.W Timeout
 A0 ID di canale
 A1 Puntatore allo stack aritmetico

Parametri in uscita: nessuno

Registri interessati: D1, A1

Errori possibili: ERR.NC (-1) operazione non completata
 ERR.NO (-6) canale non aperto

Descrizione:

Quando questa procedura viene chiamata, A1 deve puntare ad uno stack aritmetico locale per i parametri di almeno 240 byte. Sulla cima dello stack ci devono essere cinque parametri in virgola mobile (sei byte ciascuno) con il seguente significato:

| Posizione sullo stack | Significato |
|------------------------------|--------------------------|
| 0(A1) | Angolo sotteso dall'arco |
| 6(A1) | Y della fine dell'arco |
| C(A1) | X della fine dell'arco |
| 12(A1) | Y dell'inizio dell'arco |
| 18(A1) | X dell'inizio dell'arco |

Ricordate che gli stack si accrescono verso il basso, quindi la cima dello stack si trova a un indirizzo fisico più basso rispetto alla base dello stack.

Le coordinate passate sono relative ad una origine arbitraria (default (0,0)) e a una scala arbitraria (default 0..100). Se parte dell'arco è al di fuori della window specificata, quella parte non viene disegnata. In tale caso non viene riportato alcun errore.

SD.ELIPS \$33 (51)

Traccia una ellisse

Parametri in ingresso: D3.W Timeout
A0 ID di canale
A1 Puntatore allo stack aritmetico

Parametri in uscita: nessuno

Registri interessati: D1, A1

Errori possibili: ERR.NC (-1) operazione non completata
ERR.NO (-6) canale non aperto

Descrizione:

Quando questa procedura viene chiamata, A1 deve puntare ad uno stack aritmetico locale per i parametri di almeno 240 byte. Sulla cima dello stack ci devono essere cinque parametri in virgola mobile (sei byte ciascuno) con il seguente significato:

| Posizione sullo stack | Significato |
|------------------------------|---------------------------|
| 0(A1) | Angolo di rotazione |
| 6(A1) | Asse dell'ellisse |
| C(A1) | Eccentricità dell'ellisse |
| 12(A1) | Y del centro |
| 18(A1) | X del centro |

Ricordate che gli stack si accrescono verso il basso, quindi la cima dello stack si trova a un indirizzo fisico più basso rispetto alla base dello stack.

Le coordinate passate sono relative ad una origine arbitraria (default (0,0)) e ad una scala arbitraria (default 0..100). Se parte dell'ellisse è al di fuori della window specificata, quella parte non viene disegnata. In tale caso non viene riportato alcun errore.

SD.SCALE \$34 (52)

Stabilisce una scala

Parametri in ingresso: D3.W Timeout
 A0 ID di canale
 A1 Puntatore allo stack aritmetico

Parametri in uscita: nessuno

Registri interessati: D1, A1

Errori possibili: ERR.NC (−1) operazione non completata
 ERR.NO (−6) canale non aperto

Descrizione:

Quando questa procedura viene chiamata, A1 deve puntare ad uno stack aritmetico locale per i parametri di almeno 240 byte. Sulla cima dello stack ci devono essere tre parametri in virgola mobile (sei byte ciascuno) con il seguente significato:

| Posizione sullo stack | Significato |
|------------------------------|---|
| 00(A1) | Y della linea inferiore della window |
| 6(A1) | X dei pixel più a sinistra della window |
| C(A1) | Lunghezza asse Y (altezza window) |

Ricordate che gli stack si accrescono verso il basso, quindi la cima dello stack si trova a un indirizzo fisico più basso rispetto alla base dello stack.

Questa procedura assegna la scala arbitraria per tutte le successive operazioni grafiche di tracciamento.

SD.FLOOD \$35 (53)

Attiva / disattiva il riempimento delle aree

Parametri in ingresso: D1.L Flag per riempimento
D3.W Timeout
A0 ID di canale

Parametri in uscita: nessuno

Registri interessati: D1, A1

Errori possibili: ERR.NC (–1) operazione non completata
ERR.NO (–6) canale non aperto

Descrizione:

Questa procedura viene usata per abilitare o disabilitare il riempimento delle aree chiuse per la window specificata. Se il flag passato contiene zero, il riempimento viene disabilitato, e viceversa.

L'abilitazione del riempimento influenza soltanto le operazioni grafiche successive. Il contenuto corrente della window non viene alterato. Allo stesso modo, quando il riempimento viene disattivato, le eventuali aree già riempite rimangono tali.

SD.GCUR \$36 (54)

Assegna al cursore la posizione per le operazioni grafiche

Parametri in ingresso: D3.W Timeout
 A0 ID di canale
 A1 Puntatore allo stack aritmetico

Parametri in uscita: nessuno

Registri interessati: D1, A1

Errori possibili: ERR.NC (– 1) operazione non completata
 ERR.NO (– 6) canale non aperto

Descrizione:

Quando questa procedura viene chiamata, A1 deve puntare ad uno stack aritmetico locale per i parametri di almeno 240 byte. Sulla cima dello stack ci devono essere quattro parametri in virgola mobile (sei byte ciascuno) con il seguente significato:

| Posizione sullo stack | Significato |
|------------------------------|--------------------------------|
| 0(A1) | X grafica |
| 6(A1) | Y grafica |
| C(A1) | Offset verso destra in pixel |
| 12(A1) | Offset verso il basso in pixel |

Ricordate che gli stack si accrescono verso il basso, quindi la cima dello stack si trova a un indirizzo fisico più basso rispetto alla base dello stack.

Le coordinate specificate sono relative a una origine arbitraria (default (0,0)) e ad una scala arbitraria (default 0..100).

FS.CHECK \$40 (64)

Controlla se esiste operazione su file in sospeso

Parametri in ingresso: D3.W Timeout
A0 ID di canale

Parametri in uscita: nessuno

Registri interessati: D1, A1

Errori possibili: ERR.NC (-1) operazione non completata
ERR.NO (-6) canale non aperto

Descrizione:

Quando vengono eseguite operazioni di lettura, scrittura o modifica di puntatore su un file, il device driver standard si occupa, in modo invisibile al mondo esterno, di compiere queste operazioni in modo efficiente utilizzando dei buffer in memoria. Queste operazioni tramite buffer, che vengono eseguite in background e continuano anche se la chiamata che le ha richieste ha restituito l'errore "not complete", provocano il caricamento di blocchi fisici di un file nelle aree di lavoro in memoria (vedi capitolo 3). Questa procedura richiamabile tramite TRAP può anche essere utilizzata per verificare che siano state completate tutte le operazioni in sospeso.

FS.FLUSH \$41 (65)

Svuota i buffer di un file

Parametri in ingresso: D3.W Timeout
 A0 ID di canale

Parametri in uscita: nessuno

Registri interessati: D1, A1

Errori possibili: ERR.NC (–1) operazione non completata
 ERR.NO (–6) canale non aperto

Descrizione:

Questa procedura svuota i buffer del file specificato, ma non nel senso che distrugge il contenuto dei buffer. Quando le operazioni di scrittura su file di un programma applicativo sono state completate, può succedere che i dati siano ancora memorizzati nei blocchi di servizio in memoria, anziché essere già scritti fisicamente in memoria di massa.

Questa procedura, allora, può essere usata per avere la certezza che tutti i dati sono stati scritti sul file contenuto nell'unità di memoria di massa fisica. La procedura può essere chiamata durante le operazioni su file, per sicurezza; l'operazione di chiusura di un canale, comunque, prevede in ogni caso anche lo svuotamento dei buffer.

FS.POSAB \$42 (66)

Assegna posizione assoluta al puntatore di un file

Parametri in ingresso: D1.L Posizione puntatore di file
D3.W Timeout
A0 ID di canale

Parametri in uscita: D1.L Nuova posizione del puntatore di file

Registri interessati: D1, A1

Errori possibili: ERR.NC (−1) operazione non completata
ERR.NO (−6) canale non aperto
ERR.EF (−10) fine del file

Descrizione:

Questa procedura consente di assegnare una posizione al puntatore di un file. La posizione specificata è assoluta.

Viene restituito l'errore ERR.EF (−10) se richiede il posizionamento del puntatore a una posizione prima dell'inizio del file o dopo la fine. In entrambi i casi, al puntatore viene assegnata la corrispondente posizione estrema (cioè 0 o fine file).

FS.POSRE \$43 (67)

Assegna posizione relativa al puntatore di un file

Parametri in ingresso: D1.L Posizione relativa del puntatore di file
 D3.W Timeout
 A0 ID di canale

Parametri in uscita: D1.L Nuova posizione del puntatore di file

Registri interessati: D1, A1

Errori possibili: ERR.NC (–1) operazione non completata
 ERR.NO (–6) canale non aperto
 ERR.EF (–10) fine del file

Descrizione:

Questa procedura consente di assegnare una posizione (in termini di numero di byte) a un puntatore di file. La posizione specificata è relativa alla posizione corrente. È facile rendersi conto che la procedura può essere utilizzata per ottenere la posizione assoluta corrente del puntatore di file: basta passare zero (in D1) come posizione relativa.

Viene restituito l'errore ERR.EF (–10) se la posizione assegnata al puntatore si trova prima dell'inizio del file o dopo la fine. In entrambi i casi al puntatore viene assegnata la corrispondente posizione estrema (cioè 0 o fine file).

FS.MDINF \$45 (69)

Acquisisce informazioni riguardanti il supporto di memoria di massa

| | |
|-------------------------------|---------------------------------------|
| <i>Parametri in ingresso:</i> | D3.W Timeout |
| | A0 ID di canale |
| | A1 Puntatore al buffer |
| <i>Parametri in uscita:</i> | D1.L Numero di settori |
| | A1 Puntatore alla fine del buffer |
| <i>Registri interessati:</i> | D1, A1 – A3 |
| <i>Errori possibili:</i> | ERR.NC (–1) operazione non completata |
| | ERR.NO (–6) canale non aperto |

Descrizione:

Supponendo che siano stati aperti un file o un'unità, questa procedura consente di ottenere informazioni che riguardano il supporto di memoria di massa.

Quando la procedura viene chiamata, A1 deve puntare all'inizio di un buffer lungo 10 byte che viene poi usato per memorizzare il nome del supporto.

All'uscita della procedura, A1 punta al byte che si trova dopo la fine del buffer specificato. Inoltre, la word più significativa di D1 contiene il numero di settori liberi, mentre la word meno significativa di D1 contiene il numero totale di settori giudicati idonei.

Un settore è costituito da 512 byte.

FS.HEADS \$46 (70)

Assegna il contenuto dell'intestazione di un file

| | |
|-------------------------------|---|
| <i>Parametri in ingresso:</i> | D3.W Timeout |
| | A0 ID di canale |
| | A1 Base della tabella dell'intestazione |
| <i>Parametri in uscita:</i> | D1.W Lunghezza dell'intestazione assegnata |
| | A1 Puntatore alla fine della tabella della intestazione |
| <i>Registri interessati:</i> | D1, A1 |
| <i>Errori possibili:</i> | ERR.NC (–1) operazione non completata |
| | ERR.NO (–6) canale non aperto |

Descrizione:

Ciascun file è dotato di una intestazione contenente informazioni relative al file stesso. Nel caso di file standard che possono essere elencati con un comando di catalogo dei file, questa intestazione è lunga 64 byte e ha la seguente struttura:

| Offset | Significato |
|---------------|------------------------------------|
| 00 | Lunghezza del file (long word) |
| 04 | Byte di tipo accesso a file |
| 05 | Byte di tipo file |
| 06 | Otto byte di informazioni di tipo |
| 0E | Lunghezza del nome del file (word) |
| 10 | Nome file (massimo 36 byte) |
| 34 | Riservato |

Il byte di accesso è normalmente zero. Per file dati e per programmi in SuperBASIC anche il byte di tipo è zero. I programmi eseguibili hanno un tipo file uguale a 1 e i primi quattro byte delle informazioni di tipo contengono la dimensione di default dell'area dati del programma.

Questa procedura consente di assegnare i primi 14 byte dell'intestazione. In ogni caso il file system, normalmente, è in grado di modificare il parametro "lunghezza del file" durante le operazioni di scrittura. Questa procedura non può essere utilizzata per modificare il nome del file.

La lunghezza della intestazione assegnata (questo dato viene restituito dalla procedura) conterrà dati privi di senso se la ID di canale si riferisce ad una unità puramente seriale. Inoltre, quando l'intestazione viene inviata a tale unità, i 14 byte vengono preceduti dal byte \$FF (255).

FS.HEADR \$47 (71)

Legge l'intestazione di file

Parametri in ingresso: D2.W Lunghezza buffer
 D3.W Timeout
 A0 ID di canale
 A1 Base del buffer

Parametri in uscita: D1.W Lunghezza letta dell'intestazione
 A1 Cima del buffer di lettura

Registri interessati: D1, A1

Errori possibili: ERR.NC (– 1) operazione non completata
 ERR.BO (– 5) overflow del buffer
 ERR.NO (– 6) canale non aperto

Descrizione:

Ciascun file è dotato di una intestazione contenente informazioni relative al file stesso. Nel caso di file standard che possono essere elencati con un comando di catalogo dei file, questa intestazione è lunga 64 byte e ha la seguente struttura:

| Offset | Significato |
|--------|------------------------------------|
| 00 | Lunghezza del file (long word) |
| 04 | Byte di tipo accesso a file |
| 05 | Byte di tipo file |
| 06 | Otto byte di informazioni di tipo |
| 0E | Lunghezza del nome del file (word) |
| 10 | Nome file (massimo 36 byte) |
| 34 | Riservato |

Il byte di accesso è normalmente zero. Per file dati e per programmi in SuperBASIC anche il byte di tipo è zero. I programmi eseguibili hanno un tipo file uguale a 1 e i primi quattro byte delle informazioni di tipo contengono la dimensione di default dell'area dati del programma.

Questa procedura consente di leggere un numero di byte della intestazione di un file corrispondenti alla lunghezza del buffer specificato (o i primi 14 byte (\$0E) nel caso di unità seriale). Le informazioni acquisite consentono di allocare spazio per una operazione di caricamento di file e

anche per determinare certe caratteristiche del file stesso. Il buffer deve essere lungo almeno 14 byte.

La posizione zero del puntatore di un file corrisponde al primo byte dopo la fine del blocco di intestazione. L'intestazione fa parte di un normale settore, perciò i limiti dei settori si trovano alle posizioni del file corrispondenti a $512 * n - 64$ ($n=0,1,2,\dots$).

La lunghezza dell'intestazione restituita dalla procedura non sarà corretta se la ID di canale si riferisce ad un'unità puramente seriale.

FS.LOAD \$48 (72)

Legge un file

Parametri in ingresso: D2.L Lunghezza del file
 D3.W Timeout
 A0 ID di canale
 A1 Indirizzo base per l'operazione di lettura

Parametri in uscita: A1 Indirizzo finale dopo la lettura

Registri interessati: D1, A1

Errori possibili: ERR.NO (-6) canale non aperto

Descrizione:

Questa procedura trasferisce un intero file in memoria. Se per l'operazione di trasferimento viene usata l'area dei programmi transienti, prima della chiamata dovrà essere allocato spazio sufficiente (tramite TRAP # 1, MT.CJOB; D0=1).

Quando la procedura viene chiamata, il registro D3 deve contenere -1 (timeout infinito) e l'indirizzo base specificato in A1 deve essere pari.

Quando la procedura restituisce il controllo, il registro A1 punta al byte che segue l'ultimo byte trasferito in memoria.

FS.SAVE \$49 (73)

Salva un file

Parametri in ingresso: D2.L Lunghezza del file
 D3.W Timeout
 A0 ID di canale
 A1 Indirizzo della base del file

Parametri in uscita: A1 Indirizzo finale del file

Registri interessati: D1, A1

Errori possibili: ERR.NO (– 6) canale non aperto

Descrizione:

Questa procedura trasferisce un intero file dalla memoria centrale a memoria di massa. Quando la procedura viene chiamata, il registro D3 deve contenere – 1 (timeout infinito) e l'indirizzo della base specificato da A1 deve essere pari. Quando la procedura restituisce il controllo, il registro A1 punta al byte che segue l'ultimo byte salvato.

La ROM del QL contiene molte utili routine che possono essere di grande aiuto per chi scrive programmi applicativi. Per esempio, cosa ne dite di una routine che confronta due stringhe o, ancora meglio, di un completo package per l'aritmetica in virgola mobile? Tutte queste routine sono a vostra disposizione!

7.1 Routine con accesso per mezzo di trap semplificato

È possibile classificare un certo numero (sette, per la precisione) di routine di utility come routine di trap semplificato. Queste routine comportano un sovraccarico ridotto nell'accesso a procedure di I/O comuni, ma devono essere chiamate in modo utente.

7.2 Routine di utility del SuperBASIC

Il secondo gruppo di routine di utility descritto in questo capitolo comprende le utility del SuperBASIC. Esse possono essere chiamate da qualsiasi tipo di codice (cioè codice che gira sia in modo supervisore che utente). Una restrizione nell'accesso alle routine consiste nel fatto che tutti gli indirizzi passati a una routine devono essere relativi ad A6. Se

questa regola non viene rispettata per ogni indirizzo che viene passato, non c'è modo di prevedere che cosa farà la routine!

Come regola generale che vi può semplificare la vita, potete azzerare il registro A6 all'inizio del programma (per esempio, con SUB.L A6,A6), se il programma è caricato nell'area delle procedure residenti. In questo modo potete dimenticarvi di A6 e concentrarvi nel produrre codice relativo al contatore di programma (cioè indipendente dalla posizione). In alternativa, potete assemblare il programma facendolo partire dall'indirizzo zero. In questo modo, il codice finirà sempre per essere relativo a un indirizzo base uguale a zero. I job della TPA vengono eseguiti con l'indirizzo della base del codice in A6, perciò gli indirizzi del codice saranno automaticamente relativi ad A6. È opportuno allocare almeno 64 byte di stack prima di usare una di queste utility. Nel caso di routine del package aritmetico, è meglio passare a 96 byte. Tenete presente che questi numeri rappresentano i valori minimi.

7.3 Uso dei registri del 68000

L'accesso a ciascuna routine di utility avviene per mezzo del suo vettore (vedi capitolo 3), perciò non c'è alcun bisogno di passare un codice che indichi il tipo di operazione da eseguire. Se una routine restituisce un codice di errore, questo sarà in formato long word e sarà contenuto nel registro D0. In tali casi, se il codice di errore è diverso da zero, ciò significa che si è verificato un errore. Gli errori standard vengono indicati con piccoli (in valore assoluto) numeri negativi. Questi errori sono elencati nell'appendice C. Se tramite la TRAP viene chiamata una qualche forma di device driver supplementare, il codice di errore restituito può essere un puntatore a un particolare messaggio di errore. Per evitare che i due tipi di codice di errore possano venir confusi, il codice di errore di tipo puntatore punta in effetti a un indirizzo che è \$8000 byte al disotto del messaggio di errore effettivo. La descrizione completa delle routine che possono restituire un tale tipo di codice di errore elenca anche gli altri errori previsti dalle routine stesse. È ovviamente saggio controllare se si è verificato un errore dopo che una routine è stata chiamata.

Bisogna tener presente che non tutte le routine restituiscono un codice di errore. Addirittura, alcune routine utilizzano D0 per restituire un risultato al programma chiamante (non un errore!). Sta al programmatore sapere se viene restituito un codice di errore oppure no e, nel secondo caso, se il registro D0 viene usato per altri scopi.

Oltre ai registri D0 e A6, anche i registri da D1 a D3 e da A0 ad A3 vengono utilizzati in diversi modi per passare valori da e verso le routine. Alcune routine utilizzano anche D7 o A4. Per accedere a una data routine,

dopo aver predisposto gli opportuni registri, è sufficiente specificare un ben preciso vettore associato alla routine (vedi capitolo 3) ed eseguire un salto a subroutine. Nei casi in cui non viene specificato il descrittore dei dati (cioè .B, .W o .L) all'interno della descrizione, si intende sempre long word (cioè .L).

UT.WINDW Vettore \$C4 (196)

Predisporre una window identificandola con un nome

Parametri in ingresso: A0 Puntatore al nome
 A1 Puntatore al blocco di parametri

Parametri in uscita: A0 ID di canale

Registri interessati: D1—D3, A0—A3

Errori possibili: ERR.OM (–3) memoria esaurita
 ERR.OR (–4) superamento dei limiti
 ERR.NO (–6) canale non disponibile
 ERR.BN (–12) nome unità non corretto

Descrizione:

Questa routine predispone una window definita dal nome indicato e avente gli attributi specificati nel blocco di parametri. Il nome deve soddisfare le regole stabilite per i nomi dei device di tipo screen (vedi paragrafo 5.2). Il blocco di parametri è costituito da quattro byte con il seguente significato:

| Byte | Significato |
|-------------|---------------------|
| 0 | Colore del bordo |
| 1 | Larghezza del bordo |
| 2 | Colore paper |
| 3 | Colore ink |

Il colore per la evidenziazione delle stringhe viene scelto implicitamente uguale al colore paper.

Il nome specificato deve essere fornito in conformità al formato standard delle stringhe. La definizione di una stringa consiste di un dato di formato word che indica la lunghezza della stringa stessa, seguito dalla stringa.

UT.CON Vettore \$C6 (198)

Predisporre una window di tipo console

Parametri in ingresso: A1 Puntatore al blocco di parametri

Parametri in uscita: A0 ID di canale

Registri interessati: D1—D3, A0—A3

Errori possibili:
 ERR.OM (–3) memoria esaurita
 ERR.OR (–4) superamento dei limiti
 ERR.NO (–6) canale non disponibile

Descrizione:

Questa routine predisporre una window di tipo console definita dagli attributi specificati nel blocco di parametri. Il blocco è costituito da quattro byte e quattro word, con il seguente significato:

| Offset | Significato | |
|--------|------------------------|--------|
| 00 | Colore del bordo | (byte) |
| 01 | Larghezza del bordo | |
| 02 | Colore paper | |
| 03 | Colore ink | |
| 04 | Larghezza della window | (word) |
| 06 | Altezza della window | |
| 08 | X dell'origine | |
| 0A | Y dell'origine | |

Il colore per la evidenziazione dei testi viene implicitamente assunto uguale al colore paper. La lunghezza del buffer della tastiera è quella standard: 128 byte.

UT.SCR Vettore \$C8 (200)

Predisporre una window di tipo screen

Parametri in ingresso: A1 Puntatore al blocco di parametri

Parametri in uscita: A0 ID di canale

Registri interessati: D1—D3, A0—A3

Errori possibili: ERR.OM (–3) memoria esaurita
ERR.OR (–4) superamento dei limiti
ERR.NO (–6) canale non disponibile

Descrizione:

Questa routine predispone una window di tipo screen definita dagli attributi specificati nel blocco di parametri. Il blocco consiste di quattro byte e quattro word, con il seguente significato:

| Offset | Significato | |
|---------------|------------------------|--------|
| 00 | Colore del bordo | (byte) |
| 01 | Larghezza del bordo | |
| 02 | Colore paper | |
| 03 | Colore ink | |
| 04 | Larghezza della window | (word) |
| 06 | Altezza della window | |
| 08 | X dell'origine | |
| 0A | Y dell'origine | |

Il colore per la evidenziazione delle stringhe viene implicitamente assunto uguale al colore paper.

UT.ERRO Vettore \$CA (202)

Invia un messaggio di errore al canale 0

Parametri in ingresso: D0.L Codice di errore

Parametri in uscita: nessuno

Registri interessati: nessuno

Errori possibili: nessuno

Descrizione:

Questa routine visualizza il messaggio di errore, identificato dal parametro contenuto in D0, nella window associata al canale 0. Se il codice di errore è in realtà un puntatore ad uno specifico messaggio (vedi paragrafo 7.3), il messaggio stesso deve essere predisposto nel formato standard delle stringhe e deve essere terminato da un <LF> ASCII (\$0A). La definizione della stringa consiste di un dato in formato word che indica la lunghezza della stringa, seguito dalla stringa stessa. La lunghezza della stringa comprende <LF>.

UT.ERR Vettore \$CC (204)

Invia un messaggio di errore al canale n

Parametri in ingresso: D0.L Codice di errore
 A0 ID di canale

Parametri in uscita: nessuno

Registri interessati: nessuno

Errori possibili: ERR.NC (–1) operazione non completata
 ERR.NO (–6) canale non aperto

Descrizione:

Questa routine visualizza il messaggio di errore, identificato dal parametro contenuto in D0, nella window associata al canale indicato. Se il codice di errore è in realtà un puntatore ad uno specifico messaggio (vedi paragrafo 7.3), il messaggio stesso deve essere predisposto nel formato standard delle stringhe e deve essere terminato da un <LF> ASCII (\$0A). La definizione della stringa consiste di un dato in formato word che indica la lunghezza della stringa, seguito dalla stringa stessa. La lunghezza della stringa comprende <LF>.

UT.MINT Vettore \$CE (206)

Converte un numero intero in caratteri ASCII e lo invia al canale n

Parametri in ingresso: D1.W Valore intero
 A0 ID di canale

Parametri in uscita: nessuno

Registri interessati: D1—D3, A0—A3

Errori possibili: ERR.NC (–1) operazione non completata
 ERR.NO (–6) canale non aperto

Descrizione:

Questa routine converte un intero, contenuto nel registro D1, in una stringa ASCII; in seguito, alla fine della stringa viene aggiunto uno spazio e la stringa risultante viene inviata al canale specificato.

UT.MTEXT Vettore \$D0 (208)

Invia un messaggio al canale n

Parametri in ingresso: A0 ID di canale
 A1 Base del messaggio

Parametri in uscita: nessuno

Registri interessati: D1—D3, A1—A3

Errori possibili: ERR.NC (– 1) operazione non completata
 ERR.NO (– 6) canale non aperto

Descrizione:

Questa routine invia il messaggio indicato al canale specificato. Il messaggio deve rispettare il formato previsto per le stringhe. La definizione della stringa comprende un dato di lunghezza word che indica il numero di caratteri che costituiscono la stringa, seguito dalla stringa stessa. La lunghezza della stringa deve tenere conto anche del terminatore (<LF> ASCII), nel caso che si desideri inserirlo.

UT.CSTR Vettore \$E6 (230)

Confronta due stringhe

Parametri in ingresso: D0.B Tipo di confronto
 A0 Base della stringa 0
 A1 Base della stringa 1

Parametri in uscita: D0.L risultato

Registri interessati: nessuno

Errori possibili: nessuno

Descrizione:

Questa è un'utility SuperBASIC. Tutti gli indirizzi devono essere relativi ad A6.

Il parametro passato per mezzo del registro D0, indica le modalità secondo le quali deve avvenire il confronto. Le modalità previste sono:

| Tipo | Confronto |
|-------------|--|
| 0 | Standard, carattere per carattere |
| 1 | Come il tipo 0, ma le lettere maiuscole sono equivalenti alle corrispondenti minuscole |
| 2 | Come il tipo 0, ma il valore dei numeri che fanno parte della stringa è significativo |
| 3 | Come il tipo 1, ma il valore dei numeri che fanno parte della stringa è significativo |

Il risultato restituito nel registro D0 può assumere uno dei seguenti tre valori:

| Risultato | Significato |
|------------------|---|
| -1 | La stringa 0 è minore della stringa 1 |
| 0 | La stringa 0 e la stringa 1 sono uguali |
| +1 | La stringa 1 è minore della stringa 0 |

La definizione delle stringhe consiste di un dato di lunghezza word che contiene il numero di caratteri che costituiscono la stringa, seguito dalla stringa stessa.

CN.DATE Vettore \$EC (236)

Acquisisce la data e l'ora

Parametri in ingresso: D1.L Data (valore interno)
 A1 Puntatore a un'area stack

Parametri in uscita: A1 Puntatore allo stack aggiornato

Registri interessati: A1

Errori possibili: nessuno

Descrizione:

Questa è un'utility SuperBASIC. Tutti gli indirizzi devono essere relativi ad A6.

Questa routine restituisce la data e l'ora correnti sotto forma di stringa di 20 byte in formato standard. La stringa contenente l'informazione richiesta dovrà essere interpretata secondo il seguente schema:

aaaa mmmm gg hh:mm:ss

Quando la routine viene chiamata, il registro A1 deve puntare all'estremo superiore dell'area stack. Lo stack deve disporre di almeno 22 byte di spazio libero, in modo che la routine vi possa caricare la stringa e la sua lunghezza. Quando il controllo viene nuovamente ceduto al programma chiamante, A1 punta alla nuova cima dello stack, che contiene la lunghezza della stringa.

CN.DAY Vettore \$EE (238)

Acquisisce il giorno della settimana

Parametri in ingresso: D1.L Data (valore interno)
 A1 Puntatore a un'area stack

Parametri in uscita: A1 Puntatore allo stack aggiornato

Registri interessati: A1

Errori possibili: nessuno

Descrizione:

Questa è un'utility SuperBASIC. Tutti gli indirizzi devono essere relativi ad A6.

Questa routine restituisce una stringa di tre byte in formato standard contenente il giorno della settimana. Quando la routine viene chiamata, il registro A1 deve puntare all'estremo superiore dell'area stack. Lo stack deve disporre di almeno sei byte liberi, in modo che la routine vi possa caricare la stringa e la sua lunghezza. Quando il controllo viene restituito al programma chiamante, A1 punta alla nuova cima dello stack, che contiene la lunghezza della stringa.

CN.FTOD Vettore \$F0 (240)

Converte da virgola mobile ad ASCII

Parametri in ingresso: A0 Puntatore al buffer
 A1 Puntatore allo stack

Parametri in uscita: A0 Puntatore al buffer aggiornato
 A1 Puntatore allo stack aggiornato

Registri interessati: D1—D3, A0—A3

Errori possibili: nessuno

Descrizione:

Questa è un'utility SuperBASIC. Tutti gli indirizzi devono essere relativi ad A6.

Questa routine converte un numero in virgola mobile in una stringa di caratteri ASCII. La stringa non viene generata secondo il formato standard e, pertanto, non è preceduta da un dato di tipo word contenente il numero di byte che fanno parte della stringa stessa.

Quando la routine viene chiamata, il numero in virgola mobile deve essere sulla cima dello stack a cui punta A1 e il registro A0 deve puntare a un'area buffer. Al ritorno dalla routine, la stringa viene memorizzata nel buffer, che deve essere lungo almeno 14 byte. Vengono aggiornati sia il puntatore allo stack che il puntatore al buffer. Il puntatore allo stack viene incrementato di 6 e il puntatore al buffer punta al byte che segue l'ultimo carattere memorizzato.

CN.ITOD Vettore \$F2 (242)

Converte un intero in ASCII

Parametri in ingresso: A0 Puntatore al buffer
 A1 Puntatore allo stack

Parametri in uscita: A0 Puntatore al buffer aggiornato
 A1 Puntatore allo stack aggiornato

Registri interessati: D1—D3, A0—A3

Errori possibili: nessuno

Descrizione:

Questa è un'utility SuperBASIC. Tutti gli indirizzi devono essere relativi ad A6.

Questa routine converte un numero intero in una stringa di caratteri ASCII. La stringa non viene generata secondo il formato standard e, pertanto, non è preceduta da un dato di tipo word contenente il numero di byte che fanno parte della stringa stessa.

Quando la routine viene chiamata, il numero intero deve essere sulla cima dello stack a cui punta A1 e il registro A0 deve puntare a un'area buffer. Al ritorno dalla routine, la stringa viene memorizzata nel buffer, che deve essere lungo almeno 6 byte. Vengono aggiornati sia il puntatore allo stack che il puntatore al buffer. Il puntatore allo stack viene incrementato di due e il puntatore al buffer punta al byte che segue l'ultimo carattere memorizzato.

CN.ITOBB Vettore \$F4 (244)

Converte un byte (binario) in ASCII

Parametri in ingresso: A0 Puntatore al buffer
 A1 Puntatore allo stack

Parametri in uscita: A0 Puntatore al buffer aggiornato
 A1 Puntatore allo stack aggiornato

Registri interessati: D1—D3, A0—A3

Errori possibili: nessuno

Descrizione:

Questa è un'utility SuperBASIC. Tutti gli indirizzi devono essere relativi ad A6.

Questa routine converte un numero binario di dimensione byte in una stringa di caratteri ASCII. La stringa non viene generata secondo il formato standard e, pertanto, non è preceduta da un dato di tipo word contenente il numero di byte che fanno parte della stringa stessa.

Quando la routine viene chiamata, il byte deve essere sulla cima dello stack a cui punta A1 e il registro A0 deve puntare a un'area buffer. Al ritorno dalla routine, la stringa viene memorizzata nel buffer, che deve essere lungo almeno 8 byte. Vengono aggiornati sia il puntatore allo stack che il puntatore al buffer. Il puntatore allo stack viene incrementato di uno e il puntatore al buffer punta al byte che segue l'ultimo carattere memorizzato.

CN.ITOBW Vettore \$F6 (246)

Converte una word (in binario) in ASCII

Parametri in ingresso: A0 Puntatore al buffer
 A1 Puntatore allo stack

Parametri in uscita: A0 Puntatore al buffer aggiornato
 A1 Puntatore allo stack aggiornato

Registri interessati: D1—D3, A0—A3

Errori possibili: nessuno

Descrizione:

Questa è un'utility SuperBASIC. Tutti gli indirizzi devono essere relativi ad A6.

Questa routine converte un numero binario di dimensione word in una stringa di caratteri ASCII. La stringa non viene generata secondo il formato standard e, pertanto, non è preceduta da un dato di tipo word contenente il numero di byte che fanno parte della stringa stessa.

Quando la routine viene chiamata, il dato di dimensione word deve essere sulla cima dello stack a cui punta A1 e il registro A0 deve puntare a un'area buffer. Al ritorno dalla routine, la stringa viene memorizzata nel buffer, che deve essere lungo almeno 16 byte. Vengono aggiornati sia il puntatore allo stack che il puntatore al buffer. Il puntatore allo stack viene incrementato di due e il puntatore al buffer punta al byte che segue l'ultimo carattere memorizzato.

CN.ITOBL Vettore \$F8 (248)

Converte una long word (in binario) in ASCII

Parametri in ingresso: A0 Puntatore al buffer
 A1 Puntatore allo stack

Parametri in uscita: A0 Puntatore al buffer aggiornato
 A1 Puntatore allo stack aggiornato

Registri interessati: D1—D3, A0—A3

Errori possibili: nessuno

Descrizione:

Questa è un'utility SuperBASIC. Tutti gli indirizzi devono essere relativi ad A6.

Questa routine converte un numero binario di dimensione long word in una stringa di caratteri ASCII. La stringa non viene generata secondo il formato standard e, pertanto, non è preceduta da un dato di tipo word contenente il numero di byte che fanno parte della stringa stessa.

Quando la routine viene chiamata, il dato di dimensione long word deve essere sulla cima dello stack a cui punta A1 e il registro A0 deve puntare a un'area buffer. Al ritorno dalla routine, la stringa viene memorizzata nel buffer, che deve essere lungo almeno 32 byte. Vengono aggiornati sia il puntatore allo stack che il puntatore al buffer. Il puntatore allo stack viene incrementato di quattro e il puntatore al buffer punta al byte che segue l'ultimo carattere memorizzato.

CN.ITOHB Vettore \$FA (250)

Converte un byte in ASCII esadecimale

Parametri in ingresso: A0 Puntatore al buffer
 A1 Puntatore allo stack

Parametri in uscita: A0 Puntatore al buffer aggiornato
 A1 Puntatore allo stack aggiornato

Registri interessati: D1—D3, A0—A3

Errori possibili: nessuno

Descrizione:

Questa è un'utility SuperBASIC. Tutti gli indirizzi devono essere relativi ad A6.

Questa routine converte un unico byte in una stringa di caratteri ASCII. La stringa non viene generata secondo il formato standard e, pertanto, non è preceduta da un dato di tipo word contenente il numero di byte che fanno parte della stringa stessa.

Quando la routine viene chiamata, il byte deve essere sulla cima dello stack a cui punta A1 e il registro A0 deve puntare a un'area buffer. Al ritorno dalla routine, la stringa viene memorizzata nel buffer, che deve essere lungo almeno 2 byte. Vengono aggiornati sia il puntatore allo stack che il puntatore al buffer. Il puntatore allo stack viene incrementato di uno e il puntatore al buffer punta al byte che segue l'ultimo carattere memorizzato.

CN.ITOHW Vettore \$FC (252)

Converte una word in ASCII esadecimale

Parametri in ingresso: A0 Puntatore al buffer
 A1 Puntatore allo stack

Parametri in uscita: A0 Puntatore al buffer aggiornato
 A1 Puntatore allo stack aggiornato

Registri interessati: D1—D3, A0—A3

Errori possibili: nessuno

Descrizione:

Questa è un'utility SuperBASIC. Tutti gli indirizzi devono essere relativi ad A6.

Questa routine converte un dato di dimensione word in una stringa di caratteri ASCII. La stringa non viene generata secondo il formato standard e, pertanto, non è preceduta da un dato di tipo word contenente il numero di byte che fanno parte della stringa stessa.

Quando la routine viene chiamata, il dato di dimensione word deve essere sulla cima dello stack a cui punta A1 e il registro A0 deve puntare a un'area buffer. Al ritorno dalla routine, la stringa viene memorizzata nel buffer, che deve essere lungo almeno 4 byte. Vengono aggiornati sia il puntatore allo stack che il puntatore al buffer. Il puntatore allo stack viene incrementato di due e il puntatore al buffer punta al byte che segue l'ultimo carattere memorizzato.

CN.ITOHL Vettore \$FE (254)

Converte una long word in ASCII esadecimale

Parametri in ingresso: A0 Puntatore al buffer
 A1 Puntatore allo stack

Parametri in uscita: A0 Puntatore al buffer aggiornato
 A1 Puntatore allo stack aggiornato

Registri interessati: D1—D3, A0—A3

Errori possibili: nessuno

Descrizione:

Questa è un'utility SuperBASIC. Tutti gli indirizzi devono essere relativi ad A6.

Questa routine converte un dato di dimensione long word in una stringa di caratteri ASCII. La stringa non viene generata secondo il formato standard e, pertanto, non è preceduta da un dato di tipo word contenente il numero di byte che fanno parte della stringa stessa.

Quando la routine viene chiamata, il dato di dimensione long word deve essere sulla cima dello stack a cui punta A1 e il registro A0 deve puntare a un'area buffer. Al ritorno dalla routine, la stringa viene memorizzata nel buffer, che deve essere lungo almeno 8 byte. Vengono aggiornati sia il puntatore allo stack che il puntatore al buffer. Il puntatore allo stack viene incrementato di quattro e il puntatore al buffer punta al byte che segue l'ultimo carattere memorizzato.

CN.DTOF Vettore \$100 (256)

Converte una stringa ASCII in virgola mobile

Parametri in ingresso: D7 Puntatore alla fine del buffer
 A0 Puntatore al buffer
 A1 Puntatore allo stack

Parametri in uscita: A0 Puntatore al buffer aggiornato
 A1 Puntatore allo stack aggiornato

Registri interessati: D1—D3, A0—A3

Errori possibili: ERR.XP (–17) errore in espressione

Descrizione:

Questa è un'utility SuperBASIC. Tutti gli indirizzi devono essere relativi ad A6.

Questa routine converte una stringa di caratteri ASCII in un numero in virgola mobile. La stringa non deve essere nel formato standard e, pertanto, non deve essere preceduta da un dato di lunghezza word che specifica la lunghezza della stringa stessa. La conversione si arresta quando A0 diventa uguale a D7, oppure quando nella stringa ASCII viene incontrato un carattere non ammesso.

Quando la routine viene chiamata, il registro A1 deve puntare alla cima di un'area stack e il registro A0 deve puntare al buffer che contiene la stringa. Lo stack deve disporre di almeno sei byte liberi. Il registro D7 può puntare alla fine del buffer, oppure può essere uguale a zero. Al ritorno dalla subroutine, il numero in virgola mobile viene caricato sulla cima dello stack. Se non si sono verificati errori, vengono aggiornati sia il puntatore allo stack che il puntatore al buffer. Il puntatore allo stack viene decrementato di sei e il puntatore al buffer viene fatto puntare al byte successivo all'ultimo carattere valido incontrato.

CN.DTOI Vettore \$102 (258)

Converte una stringa ASCII in un numero intero

Parametri in ingresso: D7 Puntatore alla fine del buffer
 A0 Puntatore al buffer
 A1 Puntatore allo stack

Parametri in uscita: A0 Puntatore al buffer aggiornato
 A1 Puntatore allo stack aggiornato

Registri interessati: D1—D3, A0—A3

Errori possibili: ERR.XP (– 17) errore in espressione

Descrizione:

Questa è un'utility SuperBASIC. Tutti gli indirizzi devono essere relativi ad A6.

Questa routine converte una stringa di caratteri ASCII in un numero intero. La stringa non deve essere nel formato standard e, pertanto, non deve essere preceduta da un dato di lunghezza word che specifica la lunghezza della stringa stessa. La conversione si arresta quando A0 diventa uguale a D7, oppure quando nella stringa ASCII viene incontrato un carattere non ammesso.

Quando la routine viene chiamata, il registro A1 deve puntare alla cima di un'area stack e il registro A0 deve puntare al buffer che contiene la stringa. Lo stack deve disporre di almeno quattro byte liberi. Il registro D7 può puntare alla fine del buffer, oppure può essere uguale a zero. Al ritorno dalla subroutine, il numero intero viene caricato sulla cima dello stack.

Se non si sono verificati errori, vengono aggiornati sia il puntatore allo stack che il puntatore al buffer. Il puntatore allo stack viene decrementato di due e il puntatore al buffer viene fatto puntare al byte successivo all'ultimo carattere valido incontrato.

CN.BTOIB Vettore \$104 (260)

Converte una stringa ASCII in un byte

Parametri in ingresso: D7 Puntatore alla fine del buffer
 A0 Puntatore al buffer
 A1 Puntatore allo stack

Parametri in uscita: A0 Puntatore al buffer aggiornato
 A1 Puntatore allo stack aggiornato

Registri interessati: D1—D3, A0—A3

Errori possibili: ERR.XP (– 17) errore in espressione

Descrizione:

Questa è un'utility SuperBASIC. Tutti gli indirizzi devono essere relativi ad A6.

Questa routine converte una stringa di uno e zero ASCII in un byte di dati. La stringa non deve essere nel formato standard e, pertanto, non deve essere preceduta da un dato di lunghezza word che specifica la lunghezza della stringa stessa. La conversione si arresta quando A0 diventa uguale a D7, oppure quando nella stringa ASCII viene incontrato un carattere non ammesso.

Quando la routine viene chiamata, il registro A1 deve puntare alla cima di un'area stack (ad un indirizzo pari) e il registro A0 deve puntare al buffer che contiene la stringa. Lo stack deve disporre di almeno quattro byte liberi. Il registro D7 può puntare alla fine del buffer, oppure può essere uguale a zero. Al ritorno dalla subroutine, il byte viene caricato sulla cima dello stack (come byte meno significativo di una word).

Se non si sono verificati errori, vengono aggiornati sia il puntatore allo stack che il puntatore al buffer. Il puntatore allo stack viene decrementato di uno e il puntatore al buffer viene fatto puntare al byte successivo all'ultimo carattere valido incontrato.

Questa routine non funziona nella versione 1.03 QDOS e nelle precedenti.

CN.BTOIW Vettore \$106 (262)

Converte una stringa ASCII in una word

Parametri in ingresso: D7 Puntatore alla fine del buffer
A0 Puntatore al buffer
A1 Puntatore allo stack

Parametri in uscita: A0 Puntatore al buffer aggiornato
A1 Puntatore allo stack aggiornato

Registri interessati: D1—D3, A0—A3

Errori possibili: ERR.XP (–17) errore in espressione

Descrizione:

Questa è un'utility SuperBASIC. Tutti gli indirizzi devono essere relativi ad A6.

Questa routine converte una stringa di uno e zero ASCII in una word di dati. La stringa non deve essere nel formato standard e, pertanto, non deve essere preceduta da un dato di lunghezza word che specifica la lunghezza della stringa stessa. La conversione si arresta quando A0 diventa uguale a D7, oppure quando nella stringa ASCII viene incontrato un carattere non ammesso.

Quando la routine viene chiamata, il registro A1 deve puntare alla cima di un'area stack e il registro A0 deve puntare al buffer che contiene la stringa. Lo stack deve disporre di almeno quattro byte liberi. Il registro D7 può puntare alla fine del buffer, oppure può essere uguale a zero. Al ritorno dalla subroutine, la word viene caricata sulla cima dello stack. Se non si sono verificati errori, vengono aggiornati sia il puntatore allo stack che il puntatore al buffer. Il puntatore allo stack viene decrementato di due e il puntatore al buffer viene fatto puntare al byte successivo all'ultimo carattere valido incontrato.

Questa routine non funziona nella versione 1.03 del QDOS e nelle precedenti.

CN.BTOIL Vettore \$108 (264)

Converte una stringa ASCII in una long word

Parametri in ingresso: D7 Puntatore alla fine del buffer
 A0 Puntatore al buffer
 A1 Puntatore allo stack

Parametri in uscita: A0 Puntatore al buffer aggiornato
 A1 Puntatore allo stack aggiornato

Registri interessati: D1—D3, A0—A3

Errori possibili: ERR.XP (– 17) errore in espressione

Descrizione:

Questa è un'utility SuperBASIC. Tutti gli indirizzi devono essere relativi ad A6.

Questa routine converte una stringa di uno e zero ASCII in una long word di dati. La stringa non deve essere nel formato standard e, pertanto, non deve essere preceduta da un dato di lunghezza word che specifica la lunghezza della stringa stessa. La conversione si arresta quando A0 diventa uguale a D7, oppure quando nella stringa ASCII viene incontrato un carattere non ammesso.

Quando la routine viene chiamata, il registro A1 deve puntare alla cima di un'area stack e il registro A0 deve puntare al buffer che contiene la stringa. Lo stack deve disporre di almeno quattro byte liberi. Il registro D7 può puntare alla fine del buffer, oppure può essere uguale a zero. Al ritorno dalla subroutine, la long word viene caricata sulla cima dello stack. Se non si sono verificati errori, vengono aggiornati sia il puntatore allo stack che il puntatore al buffer. Il puntatore allo stack viene decrementato di quattro e il puntatore al buffer viene fatto puntare al byte successivo all'ultimo carattere valido incontrato.

Questa routine non funziona nella versione 1.03 del QDOS e nelle precedenti.

CN.HTOIB Vettore \$10A (266)

Converte una stringa ASCII esadecimale in un byte

Parametri in ingresso:

| | |
|----|--------------------------------|
| D7 | Puntatore alla fine del buffer |
| A0 | Puntatore al buffer |
| A1 | Puntatore allo stack |

Parametri in uscita:

| | |
|----|---------------------------------|
| A0 | Puntatore al buffer aggiornato |
| A1 | Puntatore allo stack aggiornato |

Registri interessati: D1—D3, A0—A3

Errori possibili: ERR.XP (– 17) errore in espressione

Descrizione:

Questa è un'utility SuperBASIC. Tutti gli indirizzi devono essere relativi ad A6.

Questa routine converte una stringa ASCII in un byte. La stringa non deve essere nel formato standard e, pertanto, non deve essere preceduta da un dato di lunghezza word che specifica la lunghezza della stringa stessa. La conversione si arresta quando A0 diventa uguale a D7, oppure quando nella stringa ASCII esadecimale viene incontrato un carattere non ammesso.

Quando la routine viene chiamata, il registro A1 deve puntare alla cima di un'area stack (partendo da un indirizzo pari) e il registro A0 deve puntare al buffer che contiene la stringa. Lo stack deve disporre di almeno quattro byte liberi. Il registro D7 può puntare alla fine del buffer, oppure può essere uguale a zero. Al ritorno dalla subroutine, il byte viene caricato sulla cima dello stack (come byte meno significativo di una word). Se non si sono verificati errori, vengono aggiornati sia il puntatore allo stack che il puntatore al buffer. Il puntatore allo stack viene decrementato di uno e il puntatore al buffer viene fatto puntare al byte successivo all'ultimo carattere valido incontrato.

Questa routine non funziona nella versione 1.03 del QDOS e nelle precedenti.

CN.HTOIW Vettore \$10C (268)

Converte una stringa ASCII esadecimale in una word

Parametri in ingresso: D7 Puntatore alla fine del buffer
 A0 Puntatore al buffer
 A1 Puntatore allo stack

Parametri in uscita: A0 Puntatore al buffer aggiornato
 A1 Puntatore allo stack aggiornato

Registri interessati: D1—D3, A0—A3

Errori possibili: ERR.XP (–17) errore in espressione

Descrizione:

Questa è un'utility SuperBASIC. Tutti gli indirizzi devono essere relativi ad A6.

Questa routine converte una stringa ASCII in una word. La stringa non deve essere nel formato standard e, pertanto, non deve essere preceduta da un dato di lunghezza word che specifica la lunghezza della stringa stessa. La conversione si arresta quando A0 diventa uguale a D7, oppure quando nella stringa ASCII esadecimale viene incontrato un carattere non ammesso.

Quando la routine viene chiamata, il registro A1 deve puntare alla cima di un'area stack e il registro A0 deve puntare al buffer che contiene la stringa. Lo stack deve disporre di almeno quattro byte liberi. Il registro D7 può puntare alla fine del buffer, oppure può essere uguale a zero. Al ritorno dalla subroutine, la word viene caricata sulla cima dello stack. Se non si sono verificati errori, vengono aggiornati sia il puntatore allo stack che il puntatore al buffer. Il puntatore allo stack viene decrementato di due e il puntatore al buffer viene fatto puntare al byte successivo all'ultimo carattere valido incontrato.

Questa routine non funziona nella versione 1.03 del QDOS e nelle precedenti.

CN.HTOIL Vettore \$10E (270)

Converte una stringa ASCII esadecimale in una long word

Parametri in ingresso: D7 Puntatore alla fine del buffer
A0 Puntatore al buffer
A1 Puntatore allo stack

Parametri in uscita: A0 Puntatore al buffer aggiornato
A1 Puntatore allo stack aggiornato

Registri interessati: D1—D3, A0—A3

Errori possibili: ERR.XP (–17) errore in espressione

Descrizione:

Questa è un'utility SuperBASIC. Tutti gli indirizzi devono essere relativi ad A6.

Questa routine converte una stringa ASCII in una long word. La stringa non deve essere nel formato standard e, pertanto, non deve essere preceduta da un dato di lunghezza word che specifica la lunghezza della stringa stessa. La conversione si arresta quando A0 diventa uguale a D7, oppure quando nella stringa ASCII viene incontrato un carattere non ammesso.

Quando la routine viene chiamata, il registro A1 deve puntare alla cima di un'area stack e il registro A0 deve puntare al buffer che contiene la stringa. Lo stack deve disporre di almeno quattro byte liberi. Il registro D7 può puntare alla fine del buffer, oppure può essere uguale a zero. Al ritorno dalla subroutine, la long word viene caricata sulla cima dello stack. Se non si sono verificati errori, vengono aggiornati sia il puntatore allo stack che il puntatore al buffer. Il puntatore allo stack viene decrementato di quattro e il puntatore al buffer viene fatto puntare al byte successivo all'ultimo carattere valido incontrato.

Questa routine non funziona nella versione 1.03 del QDOS e nelle precedenti.

RI.EXEC Vettore \$11C (284)

Esegue una singola operazione aritmetica

Parametri in ingresso: D0.B Codice dell'operazione

D7 0

A1 Puntatore allo stack aritmetico

A4 Puntatore alla base dell'area di lettura/scrittura

Parametri in uscita: A1 Puntatore allo stack aritmetico aggiornato

Registri interessati: A1

Errori possibili: ERR.OV (−18) overflow aritmetico

Descrizione:

Questa è un'utility SuperBASIC. Tutti gli indirizzi devono essere relativi ad A6 e, inoltre, è bene che il registro D7 venga azzerato.

Il package aritmetico opera su numeri in virgola mobile che siano stati caricati su uno stack specificato. Il package opera sul numero in virgola mobile che si trova sulla cima dello stack (CDS), a cui punta 0(A6,A1.L); per quelle operazioni per le quali ciò sia necessario, il package opererà anche sul numero che occupa la posizione successiva sullo stack (SSS), a cui punta 6(A6,A1.L). Il formato dei numeri in virgola mobile viene discusso nel capitolo 8.

Il package accetta codici di operazioni di due tipi. In primo luogo, ci sono le operazioni aritmetiche vere e proprie, alle quali corrispondono codici che vanno da \$02 a \$30, estremi inclusi:

| Codice | Nome | Operazione |
|--------|----------|---|
| 02 | RI.NINT | CDS all'intero più vicino, $A1 = A1 + 4$ |
| 04 | RI.INT | CDS troncato ad intero, $A1 = A1 + 4$ |
| 06 | RI.NLINT | CDS all'intero in doppia precisione più vicino, $A1 = A1 + 2$ |
| 08 | RI.FLOAT | Intero CDS in virgola mobile, $A1 = A1 - 4$ |
| 0A | RI.ADD | Somma CDS a SSS, $A1 = A1 + 6$ |
| 0C | RI.SUB | Sottrae CDS da SSS, $A1 = A1 + 6$ |
| 0E | RI.MULT | Moltiplica CDS per SSS, $A1 = A1 + 6$ |
| 10 | RI.DIV | Divide CDS per SSS, $A1 = A1 + 6$ |

| Codice | Nome | Operazione |
|--------|----------|----------------------------------|
| 12 | RI.ABS | Valore assoluto di CDS |
| 14 | RI.NEG | CDS negato |
| 16 | RI.DUP | Duplica CDS, $A1 = A1 - 6$ |
| 18 | RI.COS | Coseno di CDS |
| 1A | RI.SIN | Seno di CDS |
| 1C | RI.TAN | Tangente di CDS |
| 1E | RI.COT | Cotangente di CDS |
| 20 | RI.ASIN | Arcoseno di CDS |
| 22 | RI.ACOS | Arcocoseno di CDS |
| 24 | RI.ATAN | Arcotangente di CDS |
| 26 | RI.ACOT | Arcocotangente di CDS |
| 28 | RI.SQRT | Radice quadrata di CDS |
| 2A | RI.LN | Logaritmo in base e di CDS |
| 2C | RI.LOG | Logaritmo in base 10 di CDS |
| 2E | RI.EXP | e elevato a CDS |
| 30 | RI.POWFP | SSS elevato a CDS, $A1 = A1 + 6$ |

Come potete notare, alcune di queste operazioni alterano le dimensioni dello stack; tenete presente che gli stack si accrescono verso il basso. Chiariremo la situazione con alcuni esempi, cominciando con RI.COS (codice \$18). Questa operazione non altera il puntatore allo stack (A1). Il numero in virgola mobile che si trova sulla cima dello stack viene assunto come argomento per la funzione coseno e il risultato che ne deriva viene posto sulla cima dello stack. Siccome viene prima rimosso e poi caricato un numero in virgola mobile, il puntatore allo stack rimane invariato. Un altro esempio: RI.NINT (codice \$02). Questa operazione converte in intero il numero in virgola mobile che si trova sulla cima dello stack. L'intero risultante viene poi caricato sullo stack. I numeri in virgola mobile sono lunghi sei byte, ma gli interi sono lunghi solo due byte: perciò il puntatore allo stack viene incrementato di quattro (cioè ora lo stack occupa quattro byte di meno). Esaminando i cambiamenti subiti dal puntatore allo stack (A1), dovreste ora essere in grado di dedurre cosa succede sullo stack per ogni operazione dell'elenco.

Il secondo tipo di codice di operazione consiste di numeri negativi che vanno da \$FF a \$31 (interpretati come numeri compresi tra \$FFFF e \$FF31, estremi inclusi). Questi codici vengono associati a operazioni di lettura (da memoria a stack) se il bit meno significativo del codice è zero, o a operazioni di scrittura (da stack a memoria) se il bit meno significativo è 1. L'indirizzo di memoria utilizzato per l'operazione di lettura o

scrittura è:

$$A6.L + A4.L + ((\text{codice AND } \$FE) \text{ OR } \$FF00)$$

Possono essere trasferiti solo valori in virgola mobile. Una operazione di lettura fa decrementare di sei il puntatore allo stack A1 (creando così un nuovo CDS). Una operazione di scrittura, viceversa, fa incrementare di sei il suddetto puntatore (SSS diventa il nuovo CDS).

RI.EXECB Vettore \$11E (286)

Esegue una lista di operazioni aritmetiche

Parametri in ingresso:

| | |
|----|---|
| D7 | 0 |
| A1 | Puntatore allo stack aritmetico |
| A3 | Puntatore alla lista di operazioni |
| A4 | Puntatore alla base dell'area delle variabili |

Parametri in uscita:

| | |
|----|---------------------------------|
| A1 | Puntatore allo stack aggiornato |
|----|---------------------------------|

Registri interessati:

| |
|----|
| A1 |
|----|

Errori possibili:

| |
|-----------------------------------|
| ERR.OV (– 18) overflow aritmetico |
|-----------------------------------|

Descrizione:

Questa è un'utility SuperBASIC. Tutti gli indirizzi devono essere relativi ad A6 e, inoltre, è bene che il registro D7 venga azzerato.

Questa routine permette di eseguire una lista di operazioni per mezzo del package aritmetico. Può essere specificata qualsiasi operazione tra quelle elencate sotto RI.EXEC (vettore \$11C). La lista di byte contenente i codici delle operazioni è accessibile attraverso il puntatore che viene passato alla routine nel registro A3; la lista deve essere terminata da un byte uguale a zero.

L'integrazione con il SuperBASIC

8

Questo capitolo tratta principalmente di due argomenti: il primo riguarda i comandi SuperBASIC per la programmazione in codice macchina (per esempio, CALL, SBYTES, ecc.); il secondo argomento riguarda l'interfacciamento al SuperBASIC di procedure e funzioni in codice macchina, allo scopo di estendere il linguaggio. Per il momento, ci occuperemo esclusivamente della teoria; i quattro capitoli che costituiscono la terza parte di questo libro contengono numerosi esempi di come applicare in pratica quanto qui spiegato.

8.1 Comandi SuperBASIC per la programmazione in codice macchina

Il SuperBASIC dispone in totale di otto procedure e funzioni che permettono al programmatore Assembler di caricare, salvare ed eseguire routine in codice macchina. Sei di questi comandi vengono usati correntemente e sono: CALL, EXEC, SEXEC, LBYTES, SBYTES e RESPR; gli altri due (PEEK e POKE) sono stati inclusi per completezza. In ogni caso, nessuno dei programmi di esempio contenuti nella Parte Terza del libro usa questi due comandi; per quanto riguarda la programmazione Assembler sul QL, l'uso di PEEK e POKE viene sconsigliato (perlomeno da parte dell'autore!!). Bisognerebbe sempre utilizzare un completo package Assembler per creare, modificare e convertire in codice macchina i programmi in Assembler.

Sebbene sia possibile, in linea di principio, raccogliere i comandi in gruppi, per poi descrivere separatamente ciascun gruppo, preferiamo occuparci dei comandi prendendoli in considerazione in ordine alfabetico. Ciò è coerente con quanto fatto nei capitoli precedenti e consente, nel seguito, di fare riferimento agli argomenti di questo capitolo in modo semplice.

CALL

Questa procedura accetta come argomenti un indirizzo seguito da un massimo di 13 parametri. La sintassi generale del comando è la seguente:

CALL ind, p1, p2, ..., pn

ind è l'indirizzo di partenza del codice macchina che deve essere eseguito e che deve risiedere nell'area delle procedure residenti (vedi RESPR). Gli eventuali parametri forniti vengono caricati nei registri dati e indirizzi del 68000, da D1 a D7 e da A0 ad A5, rispettivamente. Per esempio, se venisse passato un solo parametro, esso verrebbe caricato nel registro dati D1; se vengono passati otto parametri, i primi sette vanno nei registri da D1 a D7, rispettivamente, e l'ottavo va nel registro indirizzi A0.

Non è possibile ricevere parametri al ritorno da una istruzione CALL. Il SuperBASIC segnala un eventuale errore che si sia verificato, ponendone il codice nel registro D0 al ritorno dalla routine chiamata. Se non si sono verificati errori durante l'esecuzione della routine in codice macchina, è bene azzerare D0.L prima di eseguire il ritorno. La procedura CALL è particolarmente utile quando si vuole eseguire la routine di inizializzazione di un programma per l'estensione del SuperBASIC.

EXEC

Questa procedura viene utilizzata per richiedere l'esecuzione di un file di codice eseguibile in modo indipendente, o una sequenza di tali file. Ciascun programma eseguibile viene visto dal QDOS come un job indipendente e viene eseguito all'interno dell'area dei programmi transienti. Esistono due forme di questo comando:

EXEC job1, job2, ..., jobn

EXEC_W job1, job2, ..., jobn

La prima forma (cioè EXEC) richiama il job e ritorna immediatamente al SuperBASIC. La seconda forma (cioè EXEC_W) richiama il job ma non torna al SuperBASIC finché il job non ha terminato la sua esecuzione.

Tutti i programmi dei capitoli 9 e 10 sono stati concepiti per essere eseguiti in modo indipendente, cioè per mezzo del comando EXEC.

LBYTES

Utilizzando questa procedura è possibile caricare in memoria, partendo da un dato indirizzo, qualunque file. La sintassi generale del comando è:

LBYTES unità__file, ind

L'uso più ovvio di questa procedura consiste nel caricare le estensioni al SuperBASIC in codice macchina nell'area delle procedure residenti, prima dell'inizializzazione e del conseguente uso. Il parametro *ind* è l'indirizzo base dell'area di memoria dove deve essere caricato il codice. Va notato che questa procedura si occupa solo di caricare il file, ma non tenta di eseguire i byte caricati.

PEEK

Questa funzione restituisce il contenuto della locazione di memoria specificata. Esiste in tre forme che consentono di accedere a dati di dimensioni diverse:

valore = PEEK ind

valore = PEEK__W ind

valore = PEEK__L ind

La prima forma della funzione restituisce un valore di lunghezza byte (8 bit). L'indirizzo specificato può essere un indirizzo qualunque. Le ultime due restituiscono una word (16 bit) e una long word (32 bit), rispettivamente. Entrambe queste due ultime forme richiedono che *ind* sia un indirizzo pari.

POKE

Questa procedura carica nella locazione di memoria specificata il dato indicato nel comando. Sono disponibili tre forme differenti di POKE, che consentono di trattare dati di lunghezze diverse:

POKE ind, dato

POKE__W ind, dato

POKE__L ind, dato

Nel primo caso viene caricato un byte (8 bit). L'indirizzo indicato può essere un indirizzo qualunque. Negli altri due casi vengono caricate una word (16 bit) e una long word (32 bit) rispettivamente. Entrambe queste due ultime forme richiedono che *ind* sia un indirizzo pari. Nelle operazioni che riguardano dati di tipo word vengono utilizzate le locazioni agli indirizzi *ind* e *ind+1* e il byte più significativo va nella locazione *ind*. Le operazioni su dati long word utilizzano le locazioni da *ind* a *ind+3*, con il byte più significativo caricato all'indirizzo *ind*.

RESPR

Questa funzione viene usata per riservare spazio nell'area di memoria delle procedure residenti ed ha la sintassi generale:

base = RESPR (*spazio*)

Alla funzione va passato un parametro che specifica la quantità di memoria richiesta. Se la memoria libera rimasta è insufficiente a soddisfare la richiesta, viene emesso il messaggio di errore "out of memory" e l'esecuzione della funzione viene abortita. L'esecuzione viene anche interrotta, con l'emissione del messaggio "not complete", nel caso in cui un programma eseguibile è in corso di esecuzione. Se la funzione termina correttamente, viene restituito l'indirizzo base dell'area di memoria allocata. Lo spazio allocato da RESPR nell'area delle procedure residenti di solito non può essere recuperato se non rieseguendo il bootstrap del computer. Da ciò segue che se RESPR viene richiamata più volte senza operazioni di bootstrap intermedie, l'area delle procedure residenti si estende fino ad esaurire l'intera memoria. Ciò consente di eseguire un'allocazione della RAM a blocchi, associando ad ogni blocco un proprio indirizzo base. I vari blocchi non si sovrappongono.

SBYTES

Questa procedura esegue la funzione inversa di LBYTES, descritta in precedenza. La sintassi generale del comando è:

SBYTES *unità_file, ind, lung*

L'area di memoria da *ind* a *ind+lung* viene salvata nel file indicato sull'unità specificata.

SEXEC

Questa procedura salva un'area di memoria sull'unità specificata in una forma tale per cui i dati salvati possono poi essere eseguiti, come programma indipendente, per mezzo del comando SuperBASIC EXEC (o EXEC_W). La sintassi generale del comando è:

SEXEC *unità__file, ind, lung, areadati*

La procedura considera i byte contenuti nell'area di memoria compresa tra *ind* e *ind+lung* come un programma in codice macchina. Il parametro *areadati* indica lo spazio di memoria che il programma utilizzerà per memorizzare i dati in fase di esecuzione (l'area comprende anche lo stack). Osserviamo che l'area dati utilizzata in fase di esecuzione non viene salvata: viene salvato solo il codice; l'area dati verrà allocata solo al momento di caricare il programma, seguendo quanto indicato dal parametro *areadati*.

8.2 Interfacciamento al SuperBASIC

Per interfacciamento al SuperBASIC si intende un'effettiva estensione del linguaggio ottenuta aggiungendo procedure e funzioni scritte in modo opportuno in codice macchina. In realtà, far capire al SuperBASIC che esistono delle routine supplementari (comandi o istruzioni) è un'operazione estremamente semplice; ciò che può non essere altrettanto semplice (sebbene non sia poi così difficile) è organizzare il passaggio dei parametri, la manipolazione delle variabili e dei canali del SuperBASIC e l'acquisizione da parte dei programmi SuperBASIC dei risultati forniti dalle routine di estensione del linguaggio.

Per eseguire il collegamento al SuperBASIC, dobbiamo conoscere dei dettagli quali il contesto della nostra routine (cioè quali sono i puntatori disponibili e in quali registri indirizzi sono stati messi prima di accedere alla nostra routine da SuperBASIC) e la struttura della memoria di lavoro del SuperBASIC. Il resto di questo capitolo è dedicato all'interfacciamento con il SuperBASIC; inoltre, i capitoli 11 e 12 contengono alcuni esempi applicativi di quanto qui esposto.

8.3 L'ambiente SuperBASIC

Come è stato spiegato nel capitolo 3, l'intera area riservata al SuperBASIC può cambiare posizione all'interno della memoria in modo dinamico. Per questo motivo, è necessario che le routine di SuperBASIC dispongano di un puntatore che indichi loro il punto di partenza di tale area. Tutti i riferimenti al programma in memoria, alle tabelle delle variabili e a tutte le altre entità coinvolte nell'esecuzione dei programmi avviene attraverso tale puntatore (variabile nel tempo). In pratica, questo puntatore è contenuto nel registro A6.

All'atto della chiamata alla vostra estensione in codice macchina, il registro A6 punterà alla base della tabella di puntatori dell'area di lavoro del SuperBASIC (vedi Figura 8.1). Questa tabella contiene informazioni di importanza fondamentale che riguardano la posizione delle diverse aree di lavoro distinte utilizzate dal SuperBASIC. Per esempio, una di queste aree contiene il programma da eseguire. Ciascun puntatore ha il formato long word e il suo valore è relativo ad A6. In altre parole, i puntatori non puntano direttamente (con indirizzi assoluti) alle aree di lavoro ma, al contrario, contengono degli offset. Occupiamoci più approfonditamente di ciascuna di queste aree.

AREA BUFFER

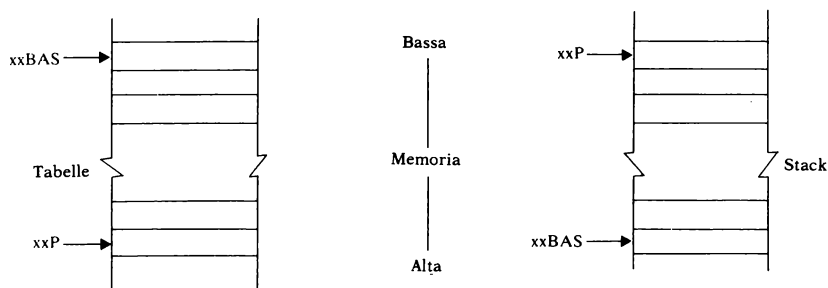
È costituita da un minimo di 128 byte e le routine possono utilizzarla liberamente. La lunghezza effettiva di questa area può essere calcolata sottraendo il puntatore alla base dell'area buffer dal puntatore contenuto in \$08(a6), per esempio:

```
;
move.l    bv__bfbas(a6),d1    ; acquis. punt. base buffer
move.l    $08(a6),d2          ; acquis. punt. locazione
                                ; succ. alla cima del buffer
subq.l    #1,d2               ; (questa è la cima)
sub.l     d1,d2               ; ora la lunghezza è in d2
;
```

Esiste un puntatore corrente (BV__BFP) che può essere utilizzato quando si opera in questa area e che, di solito, punta alla prima locazione libera.

AREA DEI PROGRAMMI SUPERBASIC

Questa è semplicemente l'area che contiene il programma corrente in SuperBASIC. I puntatori BV__PFBAS e BV__PFP puntano, rispettivamente,



**Tabella dei puntatori
alle variabili SuperBASIC**

| | | | |
|------------------------|-----------|--|----------|
| Puntatori a tabelle | \$00 (A6) | Base del buffer | BV_BFBAS |
| | \$04 (A6) | Puntatore locazione corrente del buffer | BV_BFP |
| | \$08 (A6) | | |
| | \$10 (A6) | Base programma SuperBASIC | BV_PFBAS |
| | \$14 (A6) | Cima programma SuperBASIC | BV_PFP |
| | \$18 (A6) | Base tabella dei nomi | BV_NTBAS |
| | \$1C (A6) | Cima tabella dei nomi | BV_NTP |
| | \$20 (A6) | Base lista dei nomi | BV_NLBAS |
| | \$24 (A6) | Cima lista dei nomi | BV_NLP |
| | \$28 (A6) | Base valori delle variabili | BV_VVBAS |
| | \$2C (A6) | Cima valori delle variabili | BV_VVP |
| | \$30 (A6) | Base tabella dei canali | BV_CHBAS |
| | \$34 (A6) | Cima tabella dei canali | BV_CHP |
| | | | |
| | | | |
| Puntatori a stack | \$58 (A6) | Cima stack aritmetico | BV_RIP |
| | \$5C (A6) | Base stack aritmetico | BV_RIBAS |
| | \$60 (A6) | Cima stack di sistema | BV_SSP |
| | \$64 (A6) | Base stack di sistema | BV_SSBAS |

****Ciascun puntatore è
a sua volta relativo
ad A6**

Figura 8.1 Tabella dei puntatori e aree di lavoro SuperBASIC

alla base di questa area e alla locazione successiva all'estremo superiore dell'area stessa.

TABELLA DEI NOMI

Questa tabella è di vitale importanza. Ciascun riferimento alle variabili dei programmi SuperBASIC viene effettuato tramite un puntatore a un elemento della tabella dei nomi. Ciascun elemento è lungo 8 byte, come mostrato in Figura 8.2.

| Byte | Uso |
|-------------|---|
| 0,1 | Descrizione del tipo associato ai nomi: 0001 Variabile stringa, valore non ancora assegnato 0002 Variabile virgola mobile, valore non ancora assegnato 0003 Variabile intera, valore non ancora assegnato 0201 Variabile stringa 0202 Variabile virgola mobile 0203 Variabile intera 0300 Array sottostringa (solo per uso interno) 0301 Array di tipo stringa 0302 Array di tipo virgola mobile 0303 Array di tipo intero 0400 Procedura SuperBASIC 0501 Funzione SuperBASIC di tipo stringa 0502 Funzione SuperBASIC di tipo virgola mobile 0503 Funzione SuperBASIC di tipo intero 0602 Variabile indice per loop REPEAT 0702 Variabile indice per loop FOR 0800 Procedura in codice macchina 0900 Funzione in codice macchina |
| 2,3 | Puntatore al nome. È un offset che permette di accedere al nome nella lista dei nomi. Se l'offset vale -1 ciò significa che l'elemento nella tabella dei nomi si riferisce a un valore che deve essere utilizzato dall'interprete delle espressioni (in tal caso i byte 0,1 della tabella dei nomi varranno 01xx). |
| 4-7 | Puntatore al valore (long word). È un offset che permette di accedere al valore dell'elemento (nel caso di variabili) o al descrittore (nel caso di array), partendo dalla base dell'area dei valori delle variabili. Se il puntatore è negativo, ciò indica che a quell'elemento non è ancora stato assegnato un valore. Viene anche usato come puntatore assoluto a una procedura o funzione o come numero di linea di una procedura o funzione in SuperBASIC. |

Figura 8.2 Elemento della tabella dei nomi

Osserviamo che le procedure (sia in SuperBASIC che in codice macchina) e le funzioni in codice macchina non sono dotate di tipo. Per le funzioni SuperBASIC, al contrario, è possibile definire un tipo, che è indicato dall'ultimo carattere del nome della funzione stessa (cioè: \$ -stringa-, % -intero-, altri caratteri -virgola mobile-).

LISTA DEI NOMI

È una lista dei nomi effettivi. Ogni nome è composto da un byte che contiene il numero dei caratteri che compongono il nome, seguito dai caratteri stessi.

VALORI DELLE VARIABILI

Questa è un'area di lavoro che viene allocata a blocchi di otto byte ciascuno. A seconda del tipo di variabile vengono allocati uno o più blocchi.

1. Un intero è lungo due byte e viene rappresentato nell'usuale formato in complemento a due.
2. Un numero in virgola mobile viene memorizzato come un esponente, con offset, di due byte seguito da una mantissa di quattro byte. Esempi di codifica di numeri in virgola mobile sono:

| Esponente | Mantissa | | Valore |
|-----------|----------|------|--------|
| 0000 | 0000 | 0000 | 0.0 |
| 0801 | 4000 | 0000 | 1.0 |
| 0800 | 8000 | 0000 | -1.0 |
| 0804 | 5000 | 0000 | 10.0 |

3. Una stringa viene memorizzata come una word di dati, che contiene il numero di byte che compongono la stringa, seguita dalla stringa stessa. Lo spazio effettivo occupato dalla stringa viene arrotondato al numero pari più vicino alla lunghezza della stringa.
4. I descrittori degli array hanno una long word di intestazione che contiene l'offset dei valori dell'array dalla base dell'area dei valori delle variabili. Successivamente, viene il numero di dimensioni dell'array (contenuto in una word di dati) e quindi delle coppie di word di dati, una per ogni dimensione. La prima word di dimensione indica il valore massimo dell'indice per quella dimensione e la seconda word costituisce il moltiplicatore dell'indice per quella dimensione. La Figura 8.3 mostra come viene memorizzato un array in virgola mobile. Facciamo un esempio: se un'istruzione di dimensionamento in SuperBASIC è:

DIM A(3,2)

il descrittore avrà il formato:

base, 2, 3, 3, 2, 1

La memorizzazione di array di numeri in virgola mobile e di interi non presenta particolarità: gli elementi di array in virgola mobile sono lunghi 6 byte, mentre gli elementi di array di interi sono lunghi 2 byte. Gli array di stringhe sono regolari (cioè sono array di stringhe standard) a parte quanto riguarda l'elemento zero dell'ultima dimensione: infatti, l'ultima dimensione di un array di stringhe definisce la lunghezza massima delle stringhe e viene sempre arrotondata al numero pari più vicino.

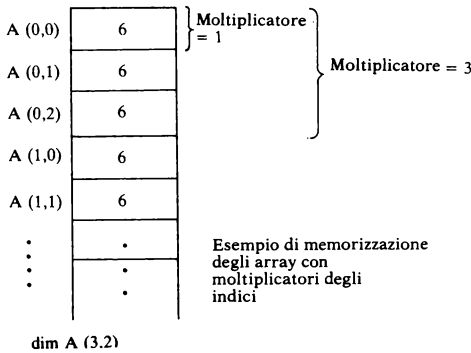


Figura 8.3 Indirizzamento degli array

TABELLA DEI CANALI

I numeri di canale del SuperBASIC ($\#n$) sono dei puntatori agli elementi della tabella dei canali. Questa tabella è costituita da un insieme di sottotabelle, una per ogni canale aperto (vedi Figura 8.4). Ciascuna sottotabella è lunga 40 byte; pertanto, l'inizio della sottotabella per $\#n$ sarà alla locazione:

$$\text{BV_CHBAS}(\text{A6}) + (n \times \text{CH.LENCH})$$

STACK ARITMETICO

Lo stack aritmetico costituisce l'area di lavoro dell'interprete delle espressioni e viene anche usato per valutare i parametri che vengono

| | | | | |
|-------------------|------|--|----------|-------------------|
| BV_CHBAS (A6) | \$00 | ID di canale (long) | CH_ID | SuperBASIC # 0 |
| | \$04 | Posizione del cursore (Y) (fp) | CH_CCPY | |
| | \$0A | Posizione del cursore (X) (fp) | CH_CCPX | |
| | \$10 | Angolo turtle (fp) | CH_ANGLE | |
| | \$16 | Stato penna (su/giù) (byte) | CH_PEN | |
| | \$20 | Posizione carattere nella linea (word) | CH_CHPOS | |
| | \$22 | Lunghezza della linea in caratteri (word) | CH_WIDTH | |
| | \$24 | | CH_SPARE | SuperBASIC # 1 |
| = (CH_LENCH) \$28 | | . | | |
| | | . | | |
| | | . | | |

Figura 8.4 Tabella di definizione dei canali SuperBASIC

passati alle/dalle routine. Inoltre, può anche essere usato come area di lavoro di uso generale. Ricordate che gli stack si accrescono verso il basso (cioè dalle locazioni di memoria alte verso quelle basse).

Il sistema di interfacciamento del SuperBASIC si occupa automaticamente di riordinare lo stack aritmetico dopo le chiamate a procedure e nel caso che si verifichino errori nelle funzioni. D'altra parte, per effettuare un buon ritorno da una funzione bisogna che lo stack aritmetico sia in ordine: l'argomento da restituire deve essere collocato sulla cima dello stack (cioè all'estremità dello stack che si trova nella parte più bassa della memoria) e non devono esserci altri dati al disotto dell'argomento (cioè in locazioni di memoria più alte). Vedi anche il paragrafo 8.11.

STACK DI SISTEMA

Questa area viene utilizzata quando si fa riferimento (implicitamente o esplicitamente) al registro indirizzi A7. Per esempio, quando si esegue una istruzione 68000 BSR l'indirizzo di ritorno viene salvato in questa area.

8.4 Implementazione di procedure in codice macchina

Quando si scrivono estensioni al SuperBASIC in codice macchina è necessario tenere presenti alcune semplici regole.

1. Ricordate che l'intera area del SuperBASIC può muoversi all'interno della memoria, pertanto tutti i riferimenti a questa area devono essere fatti relativamente al registro indirizzi A6 (o A7 nel caso di stack). Questi due registri non devono mai essere salvati per essere utilizzati in un momento successivo (ovviamente!), non devono essere usati in calcoli aritmetici o di indirizzi e non devono essere alterati (le uniche operazioni possibili sono push e pop sullo stack di A7).
2. Non possono essere utilizzati più di 128 byte sullo stack di utente.
3. Il registro dati D0, al ritorno da una procedura, deve contenere un codice di errore in formato long word.

Prima di passare il controllo a una routine, il SuperBASIC carica dei valori nei registri indirizzi A3 e A5 (oltre a predisporre opportunamente i registri di cui parlavamo prima). Per ogni parametro passato alla routine viene predisposta una posizione all'interno della tabella dei nomi (vedi i paragrafi 8.3 e 8.6). Il registro A3 punta alla locazione della tabella prevista per il primo parametro e il registro A5 punta alla fine dell'ultima posizione che contiene l'ultimo parametro (ricordiamo che A3 e A5 sono relativi ad A6).

Il numero di parametri passati sarà quindi uguale ad $(A5 - A3)/8$. Nel caso in cui A5 è uguale ad A3 è chiaro che non avviene alcun passaggio di parametri.

All'interno della routine, tutti i registri da D1 a D7 e da A0 ad A5 possono essere alterati (anche se è poco saggio alterare il contenuto di A3 o A5 con troppa fretta!).

8.5 Creazione di nuove posizioni nella tabella dei nomi

Esiste un modo semplice per inizializzare estensioni al SuperBASIC residenti in RAM. Le estensioni devono essere caricate nell'area delle procedure residenti per mezzo dei comandi SuperBASIC RESPR e LBYTES. Per semplicità e per migliorare la leggibilità del programma sorgente, è bene che le istruzioni di inizializzazione si trovino all'inizio del codice macchina (sebbene ciò non sia essenziale).

ISTRUZIONI DI INIZIALIZZAZIONE

Le istruzioni per l'inizializzazione delle routine di estensione, con la tabella associata, sono semplicissime. Il registro indirizzi A1 deve puntare all'inizio della tabella di definizione della procedura, dopodiché bisogna chiamare la routine di utility BP.INIT (vettore \$110):

```

;
lea      proc_def(pc),a1      ;carica indirizzo della tabella
move.w   $110,a2              ;predispone chiamata BP.INIT
jsr      (a2)                 ;effettua la chiamata
moveq    #0,d0                ;nessun errore
rts                      ;fine della inizializzazione
;

```

Può esistere più di una estensione, perciò il formato della tabella è il seguente:

| Dimensione dato | Uso |
|-----------------|--------------------------------------|
| Word | Numero delle procedure |
| | Per ciascuna procedura: |
| Word | — puntatore alla routine |
| Byte | — lunghezza del nome della procedura |
| Caratteri | — nome della procedura |
| Word | 0 |
| Word | Numero delle funzioni |
| | Per ciascuna funzione: |
| Word | — puntatore alla routine |
| Byte | — lunghezza del nome della funzione |
| Caratteri | — nome della funzione |
| Word | 0 |

Il numero di procedure e/o funzioni serve unicamente per riservare dello spazio all'interno della tabella. Se la lunghezza media dei nomi è maggiore di sette, questo numero deve essere:

$$(\text{num. tot. di caratteri} + \text{num. di routine} + 7)/8$$

I puntatori alle routine sono relativi all'indirizzo dell'inizio della tabella delle definizioni. Nessun registro (eccetto A1) viene alterato dalla procedura BP.INIT. Sullo stack utente non vengono utilizzati più di 48 byte.

8.6 Inizializzazione dei parametri

Quando viene chiamata una procedura o funzione in codice macchina, nella tabella dei nomi viene predisposta una posizione per ogni parametro che viene passato. Al termine della esecuzione, le posizioni della tabella occupate dai parametri vengono rimosse e lo stesso avviene per altre posizioni temporanee create in altre tabelle (per esempio, nella tabella dei valori delle variabili).

I vari separatori dei parametri di chiamata (virgola, punto e virgola, la parola riservata TO e così via) vengono codificati, nella tabella dei nomi, nel byte meno significativo del codice di descrizione (cioè nel byte 1, vedi paragrafo 8.3). La configurazione completa di questo byte è:

| | | | | | | | | |
|-----|---|---|---|---|---|---|---|---|
| bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | c | s | s | s | t | t | t | t |

Il bit 7 (c) viene settato se il parametro era preceduto da un cancelletto (#). I bit da 4 a 6 specificano il separatore che segue il parametro:

| Bit 654 | Separatore |
|------------|-----------------------|
| 000 | Nessun separatore |
| 001 | Virgola (,) |
| 010 | Punto e virgola (;) |
| 011 | Barra inversa (\) |
| 100 | Punto esclamativo (!) |
| 101 | Parola riservata TO |

I bit da 0 a 3 specificano il tipo del parametro:

| Bit 3210 | Tipo |
|-------------|----------------|
| 0000 | Niente |
| 0001 | Stringa |
| 0010 | Virgola mobile |
| 0011 | Intero |

Osserviamo che se come parametro attuale è stata passata una espressione, al puntatore al nome, nella tabella dei nomi (byte 2,3 - vedi paragrafo 8.3), viene assegnato un valore negativo.

8.7 Acquisizione degli argomenti

Esistono quattro utility del QDOS che permettono di leggere un numero indeterminato di parametri dello stesso tipo; si può accedere a queste routine nello stesso modo applicabile anche alle altre utility (cioè per mezzo di vettori) e i quattro vettori corrispondenti sono:

| | | |
|-------|----------|---|
| #0112 | CA.GTINT | Legge interi (16 bit) |
| #0114 | CA.GTFP | Legge numeri in virgola mobile (6 byte) |
| #0116 | CA.GTSTR | Legge stringhe ($2 + n$ byte per lunghezze pari, $3 + n$ byte se dispari) |
| #0118 | CA.GTLIN | Legge interi in doppia precisione (32 bit) |

Se una lista di parametri contiene parametri di tipi differenti, è necessario eseguire più chiamate alle appropriate routine per poter acquisire tutti i parametri.

Quando vengono chiamate queste utility, è necessario che i registri A3 e A5 puntino, rispettivamente, alla base e alla cima della lista dei parametri di chiamata all'interno della tabella dei nomi. I risultati vengono depositati sulla cima dello stack aritmetico, con il primo argomento all'indirizzo fisico più basso, puntato da A6,A1.L. Il numero di argomenti acquisiti viene restituito nel registro D3 in formato word. Il registro D0 conterrà il codice di errore (inoltre, anche i flag di stato saranno settati coerentemente al codice di errore) e i registri D1, D2, D4, D6, A0 e A2 vengono alterati; pertanto, sarà necessario aggiornare A3 se è prevista una nuova chiamata per mezzo di uno di questi vettori.

Gli argomenti passati come parametri possono anche essere utilizzati uno alla volta sotto il diretto controllo del programmatore, ovviamente. In questo caso, bisogna estrarre il cancelletto (#) e i codici dei separatori, far puntare A5 otto byte al disopra della locazione puntata da A3 e, quindi, chiamare l'utility appropriata. Quando si adotta questo metodo, è chiaramente importante fare attenzione nel manipolare i registri A3 e A5, in modo da non perdere qualche parametro o leggerne uno più di una volta.

8.8 Valori di funzione da restituire

Il valore di una funzione viene restituito ponendo semplicemente il valore stesso sullo stack aritmetico (usando come puntatore A6,A1.L). Inoltre, il valore del registro A1 deve essere in BV__RIP(A6) (vedi Figura 8.1). Il tipo del parametro restituito deve essere caricato nel registro D4 (1=stringa, 2=virgola mobile, 3=intero). Gli interi in doppia precisione (32 bit) devono essere convertiti in virgola mobile prima di essere restituiti.

8.9 Valori di parametri da restituire

È possibile restituire dei valori al programma chiamante da parte della procedura (o della funzione) utilizzando la lista dei parametri di chiamata. I valori dei parametri devono trovarsi sullo stack aritmetico, come nel caso delle funzioni, e BV__RIP(A6) deve contenere il valore appropriato. I valori restituiti devono essere interi, stringhe o in virgola mobile in modo tale da corrispondere al tipo dei parametri che erano stati passati. Il registro A3 deve puntare, di volta in volta, alla posizione corrispondente al parametro che deve essere restituito, all'interno della tabella dei nomi; infine, è necessario chiamare l'utility BP.LET (vettore #0120). Al ritorno da BP.LET, il registro D0 conterrà il codice di errore e i registri da D1 a D3 e da A0 ad A2 risulteranno alterati. Se il parametro attuale che era stato passato alla procedura era sotto forma di espressione, al ritorno dalla procedura l'assegnamento viene eseguito, ma il valore va perso.

8.10 Restituzione di stringhe

Le limitazioni del processore 68000 nell'indirizzamento di word possono provocare alcuni problemi quando si tratta di restituire delle stringhe: è necessario, in particolare, assicurarsi che il dato che rappresenta il numero di caratteri nella stringa, in formato word, si trovi allineato rispetto alle word di memoria (cioè deve partire da un indirizzo pari). In pratica, ciò si ottiene aggiungendo un byte vuoto alla fine delle stringhe di lunghezza dispari. Pertanto, per esempio, una stringa lunga 3 byte e una stringa lunga 4 byte occupano lo stesso spazio di stack: 6 byte.

8.11 Particolari sull'uso dello stack aritmetico

Le utility fornite dal QDOS per l'acquisizione dei parametri si riservano autonomamente sullo stack aritmetico lo spazio sufficiente ai loro scopi. Se una estensione in codice macchina ha bisogno di utilizzare lo stack aritmetico in modo autonomo, deve provvedere a riservarsi lo spazio necessario e, per fare ciò, deve chiamare la utility BV_CHRIX (vettore \$011A). Il numero di byte richiesti deve essere contenuto in D1 (in formato long word) quando viene chiamata la utility. Al ritorno, i registri D0 e D3 risulteranno alterati.

È possibile che lo stack aritmetico venga spostato durante questa operazione. Se la procedura ha memorizzato dei dati sullo stack aritmetico prima di chiamare BV.CHRIX, è necessario salvare lo stack pointer (di solito il registro A1) in BV_RIP(A6) prima della chiamata e recuperarlo da BV.RIP(A6) in seguito.

8.12 TRAP #4

Esiste una speciale TRAP per l'interprete di comandi SuperBASIC che può anche essere utilizzata dalle procedure in codice macchina. Questa particolare trap è la TRAP #4 e il suo scopo è di rendere relativi ad A6 gli indirizzi passati alle trap di I/O (vedi capitoli 5 e 6). Questa chiamata deve essere effettuata tutte le volte che si esegue una istruzione TRAP #2 o TRAP #3, in quanto queste due ultime chiamate ne cancellano l'effetto. Per quanto riguarda la TRAP #2, all'atto della chiamata il registro A6 viene sommato ad A0. Per la TRAP #3, invece, il registro A6 viene sommato ad A1, alla chiamata, ma viene sottratto al ritorno. Osserviamo che la chiamata TRAP #4 non viene cancellata da una TRAP #3 che venga abortita a causa di un errore "not open".

Parte Terza

Esempi di
programmazione

Semplici programmi eseguibili

9

In questo capitolo ci occuperemo di tre semplici programmi eseguibili in modo indipendente (cioè i programmi vengono eseguiti come job per mezzo del comando SuperBASIC EXEC). I tre programmi sono:

- 1. MESSG Scrive un messaggio sullo schermo
- 2. SCELTA Permette di selezionare da tastiera un messaggio da visualizzare sullo schermo
- 3. OROL Visualizza un orologio digitale

Di ciascun programma è disponibile il listato completo prodotto dall'Assembler, preceduto da una breve descrizione. Le descrizioni partono dal presupposto che il lettore abbia letto e capito gli esempi precedenti, quando gli argomenti sono simili: ciò permette di evitare le ripetizioni e di arrivare in fretta al nocciolo dei nuovi argomenti. Il package editor/Assembler utilizzato per sviluppare i programmi presentati in questo libro (e descritto nella Parte Quarta) è disponibile su cartuccia Microdrive.

I listati Assembler estesi risulteranno utili in molti modi: per prima cosa, sono un esempio di creazione di semplici programmi eseguibili; secondo, forniscono numerosi esempi di istruzioni Assembler 68000 per quelli che sono abbastanza digiuni in materia; terzo, il listato del codice in esadecimale può essere utilizzato per caricare manualmente il programma in memoria. Sebbene ciò sia lungo, noioso e soggetto a errori, vi dà almeno la possibilità di provare i programmi senza dover per forza comprare il package Assembler.

9.1 Esempio 1 — MESSG

Il primo esempio di programma eseguibile (Programma 9.1) illustra diversi concetti importanti. In primo luogo dà un esempio della quantità di codice necessaria per creare anche solo un semplice programma. Quando scrivete programmi in linguaggi ad alto livello, come il SuperBASIC, la maggior parte delle grane vi viene risparmiata; in Assembler, al contrario, non vi viene risparmiato proprio niente.

All'inizio del programma vengono dichiarate alcune costanti che vengono poi utilizzate all'interno del programma. Queste dichiarazioni non sono indispensabili, ma l'uso di costanti all'interno del codice, al posto dei valori effettivi, fa parte di quelle pratiche nebulose note come buone abitudini. È chiaro che queste dichiarazioni non occupano alcuno spazio all'interno del programma.

Dopo le dichiarazioni viene l'intestazione del job. Quando il programma viene eseguito (per mezzo del comando SuperBASIC EXEC), al contatore di programma viene assegnato l'indirizzo dell'inizio del programma, cosicché la prima istruzione eseguita salterà al di là dell'intestazione. Non è un errore. L'intestazione serve unicamente a identificare il programma, non è essenziale e può essere omessa. L'esempio JOBS del capitolo 11, però, produrrà un output più ricco di informazioni se i job contengono l'intestazione.

La prima operazione eseguita dal programma, che comincia alla label MESSG, è di predisporre uno schermo (cioè una window senza la possibilità di accesso da tastiera). L'indirizzo del blocco di definizione dello schermo viene caricato nel registro A1 (per mezzo dell'istruzione LEA, *Load Effective Address*), dopodiché viene chiamata la routine UT_SCR per aprire lo schermo, assegnarne i colori, il bordo e per ripulirlo. UT_SCR controlla il codice di errore in D0 prima di ritornare; pertanto, se il codice è diverso da zero, tutto ciò che resta da fare è saltare alla fine del programma.

In seguito, il programma predispone la dimensione dei caratteri (solo per la propria window). UT_SCR carica la ID del canale nel registro A0, e lì è già pronta per la successiva istruzione TRAP #3 con la chiamata a UT_MTEXT. La chiamata a UT_MTEXT è uguale a quella a UT_SCR (cioè in A1 viene caricato l'indirizzo del messaggio e poi viene chiamata la routine).

Alla fine del programma il codice di errore (che proviene da UT_SCR o UT_MTEXT) viene spostato in D3 e il job viene rimosso d'autorità dal QL.

9.2 Esempio 2 — SCELTA

Il secondo esempio di job illustra, in più, la lettura di singoli caratteri da tastiera e la notifica degli errori. Il Programma 9.2 deve essere eseguito per mezzo del comando SuperBASIC EXEC_W, se si vuole evitare di usare CTRL C per passare da una coda di input all'altra. (Quando ci sono più window in attesa di input, la tastiera può venire associata da una window all'altra. Se vi capita di non vedere più il cursore che lampeggia, provate a dare CTRL C.)

In questo programma, che comincia alla label SCELTA, viene utilizzata l'utility UT_CON per aprire una console (una window dotata di coda di input da tastiera). In seguito, viene visualizzato un prompt nella nuova console per mezzo di UT_MTEXT. Va osservato che la UT_MTEXT contenuta nelle ROM delle prime versioni di QDOS non si preoccupava di controllare il proprio codice di errore al momento del ritorno.

Ora il cursore è abilitato. Se non ci sono altre console con il cursore abilitato, la tastiera viene automaticamente associata a questa console e nella window appare un cursore della dimensione di un carattere.

A questo punto viene eseguita una nuova TRAP per caricare un byte di input nel registro D1. Il timeout indicato è di 500 periodi video (10 secondi a 50 Hz). Se non viene battuto niente da tastiera entro 10 secondi, la TRAP esegue un ritorno notificando l'errore "not complete".

Il programma, a questo punto, assume che l'utente non abbia premuto né 1 né 2 e provvede a caricare il codice di errore appropriato (ERR_NF, "not found"). Il programma poi controlla che D1 non sia minore di 1 e che non sia maggiore di 2. Dopo questo piccolo blocco di controllo c'è un trucco di programmazione. Il programma prima carica l'indirizzo del primo messaggio (la destinazione è un registro indirizzi, per cui i codici di condizione non vengono alterati), poi controlla se D1 era minore di 2; in caso negativo, carica l'indirizzo del secondo messaggio.

9.3 Esempio 3 — OROL

Il terzo esempio (Programma 9.3) di semplice job eseguibile è un orologio digitale. Questo orologio viene posizionato in alto a destra nella window riservata ai comandi, in qualunque posizione essa si trovi.

La prima azione del programma, che parte dalla label OROL, è di azzerare il registro A6. Ciò è necessario in quanto il programma non utilizza mai A6, ma la routine di conversione della data lo usa come indirizzo base. Se A6 è zero, la conversione di data utilizzerà l'indirizzamento assoluto.

L'azione successiva del job è di porre la propria priorità a 1: in questo caso il job viene considerato in background, caricando pochissimo la macchina. In seguito, il programma apre una window e ne assegna i colori. Queste operazioni vengono effettuate esplicitamente in quanto, in questo momento, il programma non sa ancora dove sarà posizionata la window. Osserviamo che UT_SCR avrebbe ripulito la window, se fosse stata usata per aprirla. Il loop (che comincia alla label locale 10%) comincia con una TRAP che sospende il job. Visto che la priorità è già la più bassa possibile, ciò potrebbe sembrare irrilevante. In effetti il job, dato che non deve mai attendere l'esecuzione di operazioni di I/O, riceverebbe una priorità superiore a quella di un altro job che dovesse attendere in continuazione il completamento dell'I/O. Sospendendo il job si riduce ulteriormente il carico della macchina.

La successiva operazione del loop è, forse, un po' strana: è un driver di schermo EXTOP. Lo scopo delle EXTOP è di permettere l'estensione dei driver di schermo da parte dei programmi applicativi. In questo caso, il codice EXTOP è una routine chiamata GET_WIND che è scritta come se facesse parte di un device driver. All'interno della routine il registro A0 punta al blocco di definizione dello schermo per la window del job (non è più la ID) e A6 punta alla base delle variabili di sistema. La prima azione della routine è di trovare l'indirizzo della base della window del canale 0. Trasferisce l'origine delle X e delle Y nel proprio blocco di definizione, definisce la scala delle X e delle Y (che dipende dalla dimensione dei caratteri) e ricalcola l'origine delle X (come: $window_0_origine_X + larghezza_window_0 - larghezza_propria_window$). Vedi paragrafo 6.3 per i blocchi di definizione dei canali tipo schermo.

L'ora viene caricata nel registro D1 e convertita in una stringa standard per mezzo della routine di sistema CN_DATE. Il formato standard delle stringhe prevede che il numero di byte della stringa sia nella prima word della stringa stessa e questo dato viene caricato in D2 (il registro per la lunghezza delle stringhe nelle chiamate di I/O) prima di visualizzare la stringa nella window. Sia MT_SUSJB che MT_RCLCK alterano A0, perciò è necessario ripristinare questo registro prima delle chiamate TRAP # 3. L'ultima routine del programma è una semplice routine che sopprime il job se il codice di errore è diverso da zero.

Programma 9.1 MESSG

Scrittura di un messaggio

McGraw-Hill Book Co. GmbH Assembler 68000 v3.0A Pagina: 0001

```
0001 #H Scrittura di un messaggio
0002 ;
0003 ; Copyright (c) 1985 McGraw-Hill Book Co. GmbH
0004 ;
```

```

00000000          0005      ORG 0
0006 ;
FFFFFFF =        0007 ID      EQU    -1
00000005 =        0008 MT_FRJOB EQU    $05
0000002D =        0009 SD_SETSZ EQU    $2D
000000C8 =        0010 UT_SCR  EQU    $C8
000000D0 =        0011 UT_MTEXT EQU    $D0
0012 ;
0013 ; Intestazione per debugger, etc.
0014 ;
00000000 6012      0015      BRA.S  MESSG          ; salta al codice
00000002 00000000  0016      DEFL   0
00000006 4AFB      0017      DEFW   $4AFB          ; intestazione standard
00000008 0007      0018      DEFW   7
0000000A 4D6573736167676966 0019      DEFB  'Messaggio'
0020      ALIGN
0021 ;
00000014          0022 MESSG:
00000014 43FA0026  0023      LEA    SCHERMO(PC),A1 ; prepara schermo
00000018 3B7B00CB  0024      MOVE.W UT_SCR,A4
0000001C 4E94      0025      JSR    (A4)
0000001E 6614      0026      BNE.S  SUICIDIO          ; errore?
0027 ;
00000020 702D      0028      MOVEQ  #SD_SETSZ,D0 ; dimens. caratteri
00000022 7203      0029      MOVEQ  #3,D1 ; larghezza
00000024 7401      0030      MOVEQ  #1,D2 ; altezza
00000026 76FF      0031      MOVEQ  #-1,D3 ; non timeout
00000028 4E43      0032      TRAP   #3
0033 ;
0000002A 43FA001C  0034      LEA    CIAO(PC),A1 ; scrive messaggio
0000002E 3B7B00D0  0035      MOVE.W UT_MTEXT,A4
00000032 4E94      0036      JSR    (A4)
0037 ;
00000034          0038 SUICIDIO:
00000034 2600      0039      MOVE.L  D0,D3 ; riporta errore
00000036 7005      0040      MOVEQ  #MT_FRJOB,D0 ; rimozione
00000038 72FF      0041      MOVEQ  #10,D1 ; di se stesso
0000003A 4E41      0042      TRAP   #1
0000003C          0043 SCHERMO:
0000003C FF      0044      DEFB   $FF ; bordo a scacchi
0000003D 04      0045      DEFB   $04 ; largo 4 pixel
0000003E 04      0046      DEFB   $04 ; sfondo verde
0000003F 00      0047      DEFB   $00 ; lettere nere
00000040 00C8      0048      DEFW   200 ; larghe 200 pixel
00000042 0023      0049      DEFW   35 ; alte 35
00000044 009C      0050      DEFW   156 ; al centro
00000046 0064      0051      DEFW   100
00000048          0052 CIAO:
00000048 0004      0053      DEFW   4
0000004A 43696166  0054      DEFB  'Ciao'
0055 ;
0056 END
    
```

Simboli:

```

00000048 CIAO      FFFFFFFF ID      00000014 MESSG      00000005 MT_FRJOB      0000003C SCHERMO
0000002D SD_SETSZ  00000034 SUICIDIO  000000D0 UT_MTEXT  000000C8 UT_SCR
    
```

Errori: 0000
 Byte liberi: 4F4C
 Codice (byte): 0004E

Assembler terminato

Programma 9.2 SCELTA

Scrivi uno di due messaggi McGraw-Hill Book Co. 6mbH Assembler 68000 v3.0A Pagina: 0002

```

0001 #H Scrivi uno di due messaggi
0002 ;
0003 ; Copyright (c) 1985 McGraw-Hill Book Co. 6mbH
0004 ;
00000000 0005      ORG 0
0006 ;
FFFFFFF = 0007 ID      EQU    -1
FFFFFFF9 = 0008 ERR_NF EQU    -7
00000005 = 0009 MT_FRJOB EQU    $05
00000001 = 0010 ID_FBYTE EQU    $01
0000000E = 0011 SD_CURE EQU    $0E
000000C6 = 0012 UT_CON  EQU    $C6
000000D0 = 0013 UT_MTEXT EQU    $D0
0014 ;
0015 ; Intestazione per debugger, etc.
0016 ;
00000000 600E 0017      BRA.S   SCELTA      ; salta al codice
00000002 00000000 0018      DEFL    0
00000006 4AFB 0019      DEFW    $4AFB      ; intestazione standard
00000008 0006 0020      DEFW    6
0000000A 5363656C7461 0021      DEFB    'Scelta'
0022      ALIGN 1
0023 ;
00000010 43FA004A 0024 SCELTA: LEA    CON(PC),A1      ; prepara schermo
00000014 3B7800C6 0025      MOVE.W  UT_CON,A4
00000018 4E94 0026      JSR      (A4)
0000001A 663B 0027      BNE.S   SUICIDIO      ; errore?
0028 ;
0000001C 43FA004A 0029      LEA    MESSG(PC),A1      ; scrive invito
00000020 3B7800D0 0030      MOVE.W  UT_MTEXT,A4
00000024 4E94 0031      JSR      (A4)
00000026 4A80 0032      TST.L    D0
00000028 662A 0033      BNE.S   SUICIDIO      ; errore?
0034 ;
0000002A 700E 0035      MOVEQ    #SD_CURE,D0      ; abilita cursore
0000002C 76FF 0036      MOVEQ    #-1,D3
0000002E 4E43 0037      TRAP     #3
0038 ;
00000030 7001 0039      MOVEQ    #ID_FBYTE,D0      ; legge un byte
00000032 363C01F4 0040      MOVE.W  #500,D3      ; aspetta risp. 10 sec.
00000036 4E43 0041      TRAP     #3
00000038 4A80 0042      TST.L    D0      ; errore?
0000003A 661B 0043      BNE.S   SUICIDIO
0044 ;
0000003C 70F9 0045      MOVEQ    #ERR_NF,D0      ; assume risp. in errore

```

```

0000003E 04010031      0046      SUB.B    #'1',D1      ; confronta con 1
00000042 6D10         0047      BLT.S    SUICIDIO      ; ... troppo piccola
00000044 5301         0048      SUBQ.B   #'1,D1      ; confronta con 2
00000046 6E0C         0049      BGT.S    SUICIDIO      ; ... troppo grande
00000048 43FA002E      0050      LEA      MESSG1(PC),A1    ; assume messg = 1
0000004C 6D04         0051      BLT.S    SCRIVE      ; era giusto?
0000004E 43FA0034      0052      LEA      MESSG2(PC),A1    ; no, era il 2
00000052              0053 SCRIVE:
00000052 4E94         0054      JSR      (A4)      ; scrive messaggio
00000054              0055      ;
00000054              0056 SUICIDIO:
00000054 2600         0057      MOVE.L   D0,D3      ; riporta tempo
00000056 7005         0058      MOVEQ   #MT_FRJOB,D0    ; rimozione
00000058 72FF         0059      MOVEQ   #10,D1      ; di se stesso
0000005A 4E41         0060      TRAP     #1
0000005A              0061      ;
0000005C 7F          0062 CON:  DEFB     $7F      ; strisce orizzontali
0000005D 04          0063      DEFB     $04      ; larghe 4 pixel
0000005E 02          0064      DEFB     $02      ; sfondo rosso
0000005F 07          0065      DEFB     $07      ; lettere bianche
00000060 00C8        0066      DEFW     200      ; larghe 200 pixel
00000062 0023        0067      DEFW     35       ; alte 35
00000064 009C        0068      DEFW     156      ; al centro
00000066 0064        0069      DEFW     100
00000066              0070      ;
00000068 000C        0071 MESSG: DEFW     12
0000006A 536365676C692031206F 0072      DEFB     'Scegli 1 o 2'
00000076 0A          0073      DEFB     $A       ; a capo (new line)
00000076              0074      ALIGN
00000078 000A        0075 MESSG1: DEFW     10
0000007A 42656E20666174746F21 0076      DEFB     'Ben fatto!'
0000007A              0077      ALIGN
00000084 000A        0078 MESSG2: DEFW     10
00000086 42656E697373696D6F21 0079      DEFB     'Benissimo!'
00000086              0080      ;
00000086              0081 END
    
```

Simboli:

```

0000005C CON      FFFFFFFF ERR_NF      FFFFFFFF IO      00000001 IO_FBYTE  00000068 MESSG
00000078 MESSG1   00000084 MESSG2      00000005 MT_FRJOB  00000010 SCELTA   00000052 SCRIVE
0000000E SD_CURE  00000054 SUICIDIO      000000C6 UT_CON   000000D0 UT_MTEXT
    
```

Errori: 0000
 Byte liberi: 4EE8
 Codice (byte): 00090

Assembler terminato

Programma 9.3 OROL

Orologio nella window 0

McGraw-Hill Book Co. GmbH Assembler 68000 v3.0A Pagina: 0001

```

0001 MH Orologio nella window 0
0002 ;
    
```

```

0003 ; Copyright (c) 1985 McGraw-Hill Book Co., GmbH
0004 ;
0005      ORG 0
0006 ;
0007 ID      EQU    -1
0008 MT_FRJOB EQU    $05
0009 MT_SUSJB EQU    $08
0010 MT_PRIOR EQU    $08
0011 MT_RCLK EQU    $13
0012 ID_OPEN EQU    $01
0013 ID_SSTRG EQU    $07
0014 SD_EXTOP EQU    $09
0015 SD_TAB EQU    $11
0016 SD_SETST EQU    $28
0017 SD_SETIN EQU    $29
0018 ;
0019 SV_CHBAS EQU    $78
0020 ;
0021 SD_XMIN EQU    $18
0022 SD_XSIZE EQU    $1C
0023 SD_YSIZE EQU    $1E
0024 SD_XINC EQU    $26
0025 ;
0026 CN_DATE EQU    $EC
0027 ;
0028 ; Inserimento intestazione per debugger, etc.
0029 ;
0030      BRA.S  OROL      ; salta a progr. orol.
0031      DEFL  0           ; 4 byte di riempimento
0032      DEFW  $4AFB      ; flag standard per job
0033      DEFW  8          ; nome lungo 8 byte
0034      DEFB  'Orologio'
0035      ALIGN 1
0036 ;
0037 OROL:  SUB.L  A6,A6    ; A6 non serve, azzerare
0038      MOVEQ  #MT_PRIOR,D0 ; priorit 
0039      MOVEQ  #ID,D1      ; ... di questo job
0040      MOVEQ  #1,D2      ; ... = 1 (la + bassa)
0041      TRAP   #1
0042 ;
0043      MOVEQ  #ID_OPEN,D0 ; window per orologio
0044      MOVEQ  #ID,D1      ; ... di questo job
0045      MOVEQ  #0,D3      ; ... (e' un "device")
0046      LEA    SCR(PC),A0 ; indirizzo del nome
0047      TRAP   #2
0048      BSR.S  ERRORE      ; ci sono errori?
0049      MOVE.L A0,A4      ; salvataggio ID canale
0050 ;
0051      MOVEQ  #SD_SETST,D0 ; decide per la striscia
0052      MOVEQ  #$10,D1      ; ... il colore
0053      MOVEQ  #-1,D3
0054      TRAP   #3
0055 ;
0056      MOVEQ  #SD_SETIN,D0 ; inchiostro
0057      MOVEQ  #$4,D1      ; ... verde
0058      TRAP   #3
0059 ;

```

```

0000003A 700B      0060 10Z:  MOVEQ  #MT_SUSJB,D0 ; sospensione
0000003C 72FF      0061      MOVEQ  #10,D1 ; di questo job
0000003E 760A      0062      MOVEQ  #10,D3 ; per 1/5 di secondo
00000040 93C9      0063      SUB.L  A1,A1 ; senza indirizzo di flag
00000042 4E41      0064      TRAP  #1
00000044 7009      0065 ;
00000046 76FF      0066      MOVEQ  #SD_EXTOP,D0 ; dove va la window?
00000048 204C      0067      MOVEQ  #-1,D3 ; attesa del termine
0000004A 45FA0034  0068      MOVE.L  A4,A0 ; predispone canale
0000004E 4E43      0069      LEA  GET_WIND(PC),A2
00000050 7011      0070      TRAP  #3
00000052 7200      0071 ;
00000054 4E43      0072      MOVEQ  #SD_TAB,D0 ; va a capo
00000056 7013      0073      MOVEQ  #0,D1
00000058 4E41      0074      TRAP  #3
0000005A 43FA006B  0075 ;
0000005E 347B00EC  0076      MOVEQ  #MT_RCLCK,D0 ; lettura tempo in D1
00000062 4E92      0077      TRAP  #1
00000064 7007      0078 ;
00000066 3419      0079      LEA  BUF_TOP(PC),A1 ; buffer da alto v. basso
00000068 76FF      0080      MOVE.W  CN_DATE,A2 ; per conversione data
0000006A 204C      0081      JSR  (A2)
0000006C 4E43      0082 ;
0000006E 6102      0083      MOVEQ  #10,SSTRB,D0 ; invia il risultato
00000070 60C8      0084      MOVE.W  (A1)+,D2 ; di 20 caratteri
00000072 4A80      0085      MOVEQ  #-1,D3 ; ... senza timeout
00000074 6708      0086      MOVE.L  A4,A0 ; alla window prestab.
00000076 2600      0087      TRAP  #3
00000078 7005      0088      BSR.S  ERRORE ; ci sono errori?
0000007A 72FF      0089      BRA.S  10Z
0000007C 4E41      0090 ;
0000007E 4E75      0091 ; Verifica errori di I/O
00000080 246E007B  0092 ;
00000082 2452      0093 ERRORE: TST.L  D0 ; c'e' stato un errore?
00000084 216A001B001E  0094      BEQ.S  OK ; ... no
00000086 C0E80026  0095      MOVE.L  D0,D3 ; ... si' - va riferito
00000088 317C000A001E  0096      MOVEQ  #MT_FRJOB,D0 ; rimozione del job
0000008A 906A001C  0097      MOVEQ  #10,D1 ; ... proprio questo
0000008C 916B0018  0098      TRAP  #1 ; (da qui non si torna
0000008E 916B0018  0099 ; piu' indietro)
00000090 916B0018  0100 ;
00000092 916B0018  0101 OK:  RTS
00000094 916B0018  0102 ;
00000096 916B0018  0103 ; Questa routine prepara la window necessaria a sovrapp-
00000098 916B0018  0104 ; porre la window 0 in cima all'RHS. Questa routine fa
0000009A 916B0018  0105 ; parte di un device driver ed e' in modo supervisor.
0000009C 916B0018  0106 ;
0000009E 916B0018  0107 GET_WIND:
000000A0 916B0018  0108      MOVE.L  SV_CHBAS(A6),A2
000000A2 916B0018  0109      MOVE.L  (A2),A2 ; acquis. origine (X,Y)
000000A4 916B0018  0110      MOVE.L  SD_XMIN(A2),SD_XMIN(A0)
000000A6 916B0018  0111      MOVEQ  #20,D0 ; 20 caratteri
000000A8 916B0018  0112      MULLU  SD_XINC(A0),D0 ; ... dimensione corrente
000000AA 916B0018  0113      MOVE.W  D0,SD_XSIZE(A0) ; predisp. dimensione X
000000AC 916B0018  0114      MOVE.W  #10,SD_YSIZE(A0) ; ... e Y
000000AE 916B0018  0115      SUB.W  SD_XSIZE(A2),D0 ; calc. origine delle X
000000B0 916B0018  0116      SUB.W  D0,SD_XMIN(A0) ; ... dall'RHS
    
```

```
000000A4 7000          0117      MOVEQ    #0,D0          ; non ci sono errori
000000A6 4E75          0118      RTS
                                0119 ;
000000AB 0003          0120 SCR:   DEFW     3              ; nome perif. di output
000000AA 534352        0121      DEFB     'SCR'
                                0122      ALIGN
000000AE          0123 BUFFER:
                                0124      DEFS     22              ; questo e' per CN_DATE
000000C4          0125 BUF_TOP:
                                0126 ;
                                0127 END
```

Simboli:

| | | | | |
|-------------------|-------------------|-------------------|-------------------|-------------------|
| 000000AE BUFFER | 000000C4 BUF_TOP | 000000EC CN_DATE | 00000072 ERRORE | 00000080 GET_WIND |
| FFFFFFF IQ | 00000001 IQ_OPEN | 00000007 IQ_SSTRG | 00000005 MT_FRJOB | 0000000B MT_PRIOR |
| 00000013 MT_RCLK | 00000008 MT_SUSJB | 0000007E OK | 00000012 DRDL | 000000AB SCR |
| 00000009 SD_EXTOP | 00000029 SD_SETIN | 0000002B SD_SETST | 00000011 SD_TAB | 00000026 SD_XINC |
| 0000001B SD_XMIN | 0000001C SD_XSIZE | 0000001E SD_YSIZE | 0000007B SV_CHBAS | |

Errori: 0000
Byte liberi: 4E16
Codice (byte): 000C4

Assembler terminato

In questo capitolo ci occuperemo di due programmi eseguibili orientati alla grafica (cioè programmi di natura grafica creati come job ed eseguiti per mezzo del comando SuperBASIC EXEC). I due programmi sono:

1. ORLANLG Orologio con quadrante analogico
2. PALLA Palla multicolore che rotola

Il primo esempio, ORLANLG, è un'applicazione grafica standard. In questo esempio vedremo come disegnare linee rette e archi utilizzando le routine di calcolo in virgola mobile. Il secondo esempio, PALLA, utilizza principalmente l'indirizzamento diretto dello schermo: questo tipo di approccio alla programmazione di routine di grafica è quello comunemente usato nella creazione di figure particolari e oggetti in movimento nei videogiochi.

Di ciascun programma viene dato il listato Assembler completo, preceduto da una breve descrizione. Nelle descrizioni si assume che il lettore abbia letto e compreso le descrizioni di quegli esempi (sia di questo capitolo che dei precedenti) che sono collegati a quello in esame. Ciò consente di evitare la maggior parte delle ripetizioni e di arrivare subito al nocciolo dell'argomento. Il package editor/Assembler utilizzato per la preparazione di questi programmi (e che viene descritto negli ultimi due capitoli di questo libro) è disponibile su cartuccia Microdrive.

Il listato delle istruzioni in esadecimale potrebbe essere usato per caricare manualmente i programmi in memoria. Sebbene ciò sia lungo, noioso e soggetto ad errori, dà almeno la possibilità di provare i programmi senza dover necessariamente comprare il package Assembler.

10.1 Organizzazione della memoria video

Prima di cominciare ad esaminare gli esempi è meglio assicurarsi di avere ben compreso l'organizzazione della memoria video. La decodifica delle informazioni necessarie alla visualizzazione dei pixel viene eseguita su dati di lunghezza word (16 bit), come mostrato in Figura 10.1. Nel modo a quattro colori ciascuna word rappresenta otto pixel; nel modo a otto colori ogni word rappresenta quattro pixel.

La memoria video parte dall'indirizzo \$20000 e occupa 32 K di RAM. Il limitato numero di colori disponibili nel modo a quattro colori significa che non è disponibile l'attributo flash: settando contemporaneamente i bit verde e rosso si ottiene, da parte dell'hardware che effettua l'interpretazione, un pixel bianco.

Ci vogliono 64 word per generare i pixel di una riga di schermo; nel modo ad alta risoluzione (quattro colori), quindi, la larghezza dello schermo

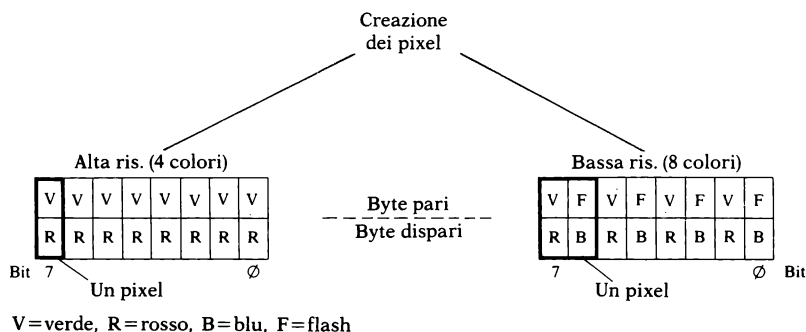
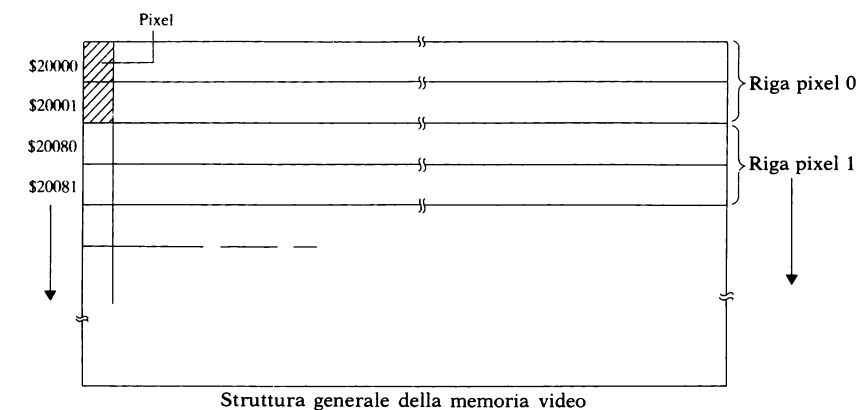


Figura 10.1 Mappa della memoria video

è 512 pixel. Nel modo a bassa risoluzione (otto colori) lo schermo è largo 256 pixel.

10.2 Esempio 1 — ORLANLG

Questo primo esempio di uso della grafica all'interno di un programma eseguibile (Programma 10.1), disegna un semplice quadrante di orologio con le lancette. In questo programma vengono utilizzate le TRAP grafiche del driver di schermo, che richiedono che i dati passati siano in virgola mobile; per questo motivo, viene usato estesamente il package di aritmetica in virgola mobile del SuperBASIC.

Il programma comincia più o meno nello stesso modo dell'esempio del clock digitale (vedi capitolo 9). Viene assegnata la priorità più bassa e viene aperta una window di tipo screen.

Successivamente, il programma adotta una scala per la grafica; per fare ciò viene chiamata una piccola procedura, `ORL_EXEC`, che carica tre costanti sullo stack aritmetico e genera una TRAP #3. I codici `RIS_0`, `RIS_2PI`, `RIS_1`, ecc. indicano all'interprete aritmetico di caricare le costanti (o i valori dello stack aritmetico) sulla cima dello stack aritmetico. I valori dei codici sono gli indirizzi delle costanti rispetto al registro A4. Siccome l'orologio ricopre parte della normale area di lavoro dello schermo, la window dell'orologio viene ripulita ogni volta che esso viene ridisegnato. Per evitare un continuo lampeggiamento dell'immagine, l'orologio viene ridisegnato solo ogni due secondi.

Per disegnare le lancette, l'ora viene divisa più volte fino a dare un resto il quale viene a sua volta diviso per dare la posizione (da 0 a 1) della lancetta appropriata. Questa posizione viene moltiplicata per $2 * \text{PI}$ per ottenere la posizione in radianti; a questo punto viene tracciata una linea da ($\text{dimens} * \text{SIN}$, $\text{dimens} * \text{COS}$) fino al centro della window (0,0). Questo programma funziona decisamente bene sia in alta che in bassa risoluzione.

10.3 Esempio 2 — PALLA

Questo secondo esempio di grafica (Programma 10.2) all'interno di un programma eseguibile, accede direttamente allo schermo, illustrando così una delle più complesse forme di animazione. Ciascuna rappresentazione dell'oggetto non deve necessariamente essere contenuta in un numero limitato di pixel e può essere fatta scorrere in qualsiasi posizione si desideri.

La curiosa organizzazione dello schermo nel modo a otto colori può far pensare che lo spostamento di un oggetto sia una operazione piuttosto difficile: infatti, ciascun pixel è rappresentato da due bit in un byte ad un indirizzo pari (o un solo bit se trascuriamo l'attributo flash) e da altri due bit nel byte successivo. Fortunatamente esiste l'istruzione MOVEP, che permette di leggere e scrivere in byte alternati! All'interno della routine che esegue il disegno, tutti i bit per il verde (con flash=0) vengono caricati in D6 e tutti i bit per il rosso e il blu in D7. Pertanto, ciascun registro contiene le informazioni sui pixel nello stesso ordine in cui compaiono sullo schermo. Dopo operazioni come spostamenti, mascherature o altre, i pixel possono essere scritti sullo schermo utilizzando più istruzioni MOVEP.

Per disegnare la palla che rotola, essa viene scritta nelle stesse word dello schermo per quattro volte di seguito ma, ogni volta, viene usata una rotazione differente e la palla viene spostata all'interno delle word di due bit. Quando la palla si muove verso destra, viene sempre fatta scorrere di almeno due bit per essere sicuri che la parte a sinistra sia vuota e che non vengano lasciate indietro parti della palla.

Programma 10.1 ORLANLG

Grafica

McGraw-Hill Book Co. GmbH Assembler 68000 v3.0A Pagina: 0001

```

0001 $H Grafica
0002 ;
0003 ; ORQL2 - orologio con quadrante analogico
0004 ;
0005 ; Copyright (c) 1985 McGraw-Hill Book Co. GmbH
0006 ;
0007 ORG 0
0008 ;
0009 IO EQU -1
0010 MT_FRJOB EQU $05
0011 MT_SUSJB EQU $08
0012 MT_PRIOR EQU $0B
0013 MT_RCLK EQU $13
0014 SD_CLEAR EQU $20
0015 SD_SETIN EQU $29
0016 SD_LINE EQU $31
0017 SD_ELIPS EQU $33
0018 SD_SCALE EQU $34
0019 ;
0020 UT_SCR EQU $C8
0021 CN_DATE EQU $EC
0022 RI_EXEC EQU $11C
0023 RI_EXECB EQU $11E
0024 RI_FLOAT EQU $08
0025 RI_ADD EQU $0A
0026 RI_MULT EQU $0E
0027 RI_DIV EQU $10

```

```

00000014 = 0028 RI_NEG EQU $14
00000016 = 0029 RI_DUP EQU $16
00000018 = 0030 RI_COS EQU $18
0000001A = 0031 RI_SIN EQU $1A
          0032 ;
          0033 RIS_0 EQU -6
          0034 RIS_2PI EQU -12
          0035 RIS_1 EQU -18
          0036 RIS_DIM EQU -24
          0037 RIS_ANG EQU -30
          0038 ;
          0039 ; Intestazione per debugger, etc.
          0040 ;
00000000 6012 0041 BRA.S OROL ; salta al codice
00000002 00000000 0042 DEFL 0 ; 4 byte di riempimento
00000006 4AFB 0043 DEFW $4AFB ; flag standard per job
00000008 0009 0044 DEFW 9
0000000A 5175616472616E7465 0045 DEFB 'Quadrante'
          0046 ALIGN
          0047 ;
00000014 9DCE 0048 OROL: SUB.L A6,A6 ; A6=0 per sempre
          0049 ;
00000016 700B 0050 MOVEQ #MT_PRIOR,D0 ; mette priorit 
00000018 72FF 0051 MOVEQ #I0,D1 ; ... di questo job
0000001A 7401 0052 MOVEQ #1,D2 ; ... =1 (la minima)
0000001C 4E41 0053 TRAP #1
          0054 ;
0000001E 347800CB 0055 MOVE.W UT_SCR,A2 ; apre window per orol.
00000022 43FA008C 0056 LEA SCR(PC),A1 ; indirizzo definizione
00000026 4E92 0057 JSR (A2)
00000028 2F08 0058 MOVE.L A0,-(A7) ; salva ID di canale
          0059 ;
0000002A 43FA01C0 0060 LEA RIS_COST(PC),A1 ; RI stack ptr top stack
0000002E 47FA00A4 0061 LEA SET_SCALE(PC),A3 ; punt. al blocco scala
00000032 49FA01CA 0062 LEA RIS_TOP(PC),A4 ; punt. cima costanti
00000036 7E34 0063 MOVEQ #SD_SCALE,D7 ; operazioni per scala
00000038 6164 0064 BSR.S ORL_EXEC ; predisp. RI ed esegue
                                grafica
          0065 ;
0000003A 0066 CICLO_OROL:
0000003A 700B 0067 MOVEQ #MT_SUSJB,D0 ; sospensione
0000003C 72FF 0068 MOVEQ #I0,D1 ; di questo job
0000003E 760A 0069 MOVEQ #I0,D3 ; per 1/5 di secondo
00000040 93C9 0070 SUB.L A1,A1 ; nessun flag
00000042 4E41 0071 TRAP #1
          0072 ;
00000044 7013 0073 MOVEQ #MT_RCLCK,D0 ; carica tempo in D1
00000046 4E41 0074 TRAP #1
00000048 E2B9 0075 LSR.L #1,D1 ; in unit  di 2 secondi
0000004A B8B1 0076 CMP.L D1,D4 ; e' cambiata l'ora?
0000004C 67EC 0077 BEQ.S CICLO_OROL ; ... no
0000004E 2B01 0078 MOVE.L D1,D4
          0079 ;
00000050 82FC5460 0080 DIVU #21600,D1 ; passa a orol. di 12 ore
00000054 4B41 0081 SWAP D1
00000056 7C00 0082 MOVEQ #0,D6
00000058 3C01 0083 MOVE.W D1,D6 ; e ricopia in D6
          0084 ;

```

```

0000005A 7020      0085      MOVEQ  #SD_CLEAR,D0      ; canc. quadrante prec.
0000005C 6148      0086      BSR.S   ORL_TRAP3
                                0087 ;
0000005E 7E33      0088      MOVEQ  #SD_ELIPS,D7      ; traccia ellisse
00000060 43FA018A   0089      LEA     RIS_COST(PC),A1 ; carica punt. a stack
00000064 47FA0075   0090      LEA     SET_CIRC(PC),A3 ; acquis. blocco cerchio
00000068 6134      0091      BSR.S   ORL_EXEC
                                0092 ;
0000006A 7A06      0093      MOVEQ  #6,D5          ; ink giallo
0000006C 7E31      0094      MOVEQ  #SD_LINE,D7      ; tracciamento linee
0000006E 47FA0071   0095      LEA     SET_LINE(PC),A3
00000072 4BFA0048   0096      LEA     ORL_DATI(PC),A5
00000076          0097 LANCETTA:
00000078 7029      0098      MOVEQ  #SD_SETIN,D0      ; colore lancette
0000007B 3205      0099      MOVE.W  D5,D1
0000007A 612A      0100      BSR.S   ORL_TRAP3
                                0101 ;
0000007C 43FA016E   0102      LEA     RIS_COST(PC),A1 ; reset punt. a stack RI
00000080 4261      0103      CLR.W   -(A1)
00000082 231D      0104      MOVE.L  (A5)+,-(A1)      ; dimensione lancette
                                0105 ;
00000084 3306      0106      MOVE.W  D6,-(A1)          ; nuova pos. su stack
00000086 7008      0107      MOVEQ  #RI_FLOAT,D0      ; converte a virg. mob.
00000088 347B011C   0108      MOVE.W  RI_EXEC,A2
0000008C 4E92      0109      JSR     (A2)
                                0110 ;
0000008E BCDD      0111      DIVU   (A5)+,D6          ; divide per divisore succ.
00000090 4246      0112      CLR.W   D6
00000092 4B46      0113      SWAP   D6              ; il resto torna in D6
                                0114 ;
00000094 331D      0115      MOVE.W  (A5)+,-(A1)      ; numero secondi del qua-
                                0116 ; drante su stack
00000096 6106      0117      BSR.S   ORL_EXEC      ; crea insieme completo
                                0118 ; parametri per chiamata
                                0119 ;
00000098 5545      0120      SUBQ   #2,D5          ; altro colore ink
0000009A 6EDA      0121      BGT.S   LANCETTA
                                0122 ;
0000009C 609C      0123      BRA.S   CICLO_ORL
                                0124 ;
0000009E          0125 ORL_EXEC:
0000009E 347B011E   0126      MOVE.W  RI_EXECB,A2      ; esegue istruzioni
000000A2 4E92      0127      JSR     (A2)
000000A4 2007      0128      MOVE.L  D7,D0          ; predis. chiave I/O
000000A6          0129 ORL_TRAP3:
000000A6 76FF      0130      MOVEQ  #-1,D3          ; senza timeout
000000AB 206F0004   0131      MOVE.L  4(A7),A0      ; predis. ID di canale
000000AC 4E43      0132      TRAP   #3
000000AE 4E75      0133      RTS
                                0134 ;
000000B0          0135 SCR:
000000B0 00000000   0136      DEFL   0              ; no bordo, nero su nero
000000B4 003C      0137      DEFW   60              ; rapporto dim. pixel 3:2
000000B6 0028      0138      DEFW   40
000000B8 01C4      0139      DEFW   512-60        ; top RHS (512-32-60 TV)
000000BA 0000      0140      DEFW   0              ; (16 per TV)
                                0141 ;

```

```

0142 ; Dati per quadrante con passo = 2 secondi
0143 ;
000000BC 0144 ORL_DATI:
000000BC 08006000 0145 DEFL $08006000 ; dim. lancetta ore=3/4
000000C0 0708 0146 DEFW 1800 ; 1800 unita' / ora
000000C2 5460 0147 DEFW 21600 ; num. unita' / quadr.
000000C4 08007000 0148 DEFL $08007000 ; dim. lancetta min.=7/8
000000C8 001E 0149 DEFW 30 ; 30 unita' per minuto
000000CA 0708 0150 DEFW 1800 ; num. unita' / quadr.
000000CC 08014000 0151 DEFL $08014000 ; dim. lancetta sec.=1
000000D0 0001 0152 DEFW 1 ; 1 unita' / unita'
000000D2 001E 0153 DEFW 30 ; 30 unita' / quadr.
0154 ;
000000D4 0155 SET_SCALA:
000000D4 EE 0156 DEFB RIS_1 ; stack 1
000000D5 EE 0157 DEFB RIS_1 ; 1,1
000000D6 0A 0158 DEFB RI_ADD ; 2
000000D7 EE 0159 DEFB RIS_1 ; 2,1
000000D8 14 0160 DEFB RI_NEG ; 2,-1
000000D9 16 0161 DEFB RI_DUP ; 2,-1,-1
000000DA 00 0162 DEFB 0
000000DB 0163 SET_CIRC:
000000DB FA 0164 DEFB RIS_0 ; stack 0
000000DC FA 0165 DEFB RIS_0 ; 0,0
000000DD EE 0166 DEFB RIS_1 ; 0,0,1
000000DE EE 0167 DEFB RIS_1 ; 0,0,1,1
000000DF FA 0168 DEFB RIS_0 ; 0,0,1,1,0
000000E0 00 0169 DEFB 0
000000E1 0170 SET_LINE:
000000E1 08 0171 DEFB RI_FLOAT ; stack dim., num.,
0172 ; divisore & num. div.
000000E2 10 0173 DEFB RI_DIV ; dim., posiz. (0->1)
000000E3 F4 0174 DEFB RIS_2PI ; dim., posiz., 2PI
000000E4 0E 0175 DEFB RI_MULT ; dim., angolo
000000E5 E2 0176 DEFB RIS_ANG ; ..... angolo
000000E6 1A 0177 DEFB RI_SIN ; x0
000000E7 E8 0178 DEFB RIS_DIM ; x0,dim.
000000E8 0E 0179 DEFB RI_MULT ; x
000000E9 E2 0180 DEFB RIS_ANG ; x,angolo
000000EA 18 0181 DEFB RI_COS ; x,y0
000000EB E8 0182 DEFB RIS_DIM ; x,y0,dim.
000000EC 0E 0183 DEFB RI_MULT ; x,y
000000ED FA 0184 DEFB RIS_0 ; x,y,0
000000EE FA 0185 DEFB RIS_0 ; x,y,0,0
000000EF 00 0186 DEFB 0
0187 ALIGN
0188 ;
000000F0 0189 RI_STACK:
000000F0 00000000000000000000 0190 DEFL 0,0,0,0,0,0,0 ; 240 byte piu' spazio
0000010C 00000000000000000000 0191 DEFL 0,0,0,0,0,0,0
0000012C 00000000000000000000 0192 DEFL 0,0,0,0,0,0,0
0000014C 00000000000000000000 0193 DEFL 0,0,0,0,0,0,0
0000016C 00000000000000000000 0194 DEFL 0,0,0,0,0,0,0
0000018C 00000000000000000000 0195 DEFL 0,0,0,0,0,0,0
000001AC 00000000000000000000 0196 DEFL 0,0,0,0,0,0,0
000001CC 00000000000000000000 0197 DEFL 0,0,0,0,0,0,0
0198 ;

```

```

000001EC          0199 RIS_COST:
000001EC 0801      0200      DEFW    $0801          ; uno
000001EE 40000000  0201      DEFL    $40000000
000001F2 0803      0202      DEFW    $0803          ; 2 * PI
000001F4 6487ED51  0203      DEFL    $6487ED51
000001F8 0000      0204      DEFW    $0000          ; zero
000001FA 00000000  0205      DEFL    $00000000
000001FE          0206 RIS_TDP:
0207 ;
0208 END

```

Simboli:

```

0000003A CICLO_OR 000000EC CN_DATE  FFFFFFFF IO      00000076 LANCETTA  00000005 MT_FRJOB
0000000B MT_PRIOR 00000013 MT_RCLK  00000008 MT_SUSJB  0000008C ORL_DATI  0000009E ORL_EXEC
000000A6 ORL_TRAP 00000014 ORL      FFFFFFFF RIS_0  FFFFFFFE RIS_1  FFFFFFFF RIS_2PI
FFFFFFE2 RIS_ANG  000001EC RIS_COST  FFFFFFFB RIS_DIM 000001FE RIS_TDP 0000000A RI_ADD
00000018 RI_COS   00000010 RI_DIV   00000016 RI_DUP   0000011C RI_EXEC  0000011E RI_EXECB
00000008 RI_FLDAT 0000000E RI_MULT  00000014 RI_NEG   0000001A RI_SIM  000000F0 RI_STACK
000000B0 SCR      00000020 SD_CLEAR 00000033 SD_ELIPS 00000031 SD_LINE  00000034 SD_SCALE
00000029 SD_SETIN 000000DB SET_CIRC 000000E1 SET_LINE 000000D4 SET_SCAL  000000CB UT_SCR

```

```

Errori:      0000
Byte liberi: 4CE0
Codice (byte): 001FE

```

Assembler terminato

Programma 10.2 PALLA

Palla che rotola

McGraw-Hill Book Co. GmbH Assembler 68000 v3.0A Pagina: 0001

```

0001 *H Palla che rotola
0002 ;
0003 ; Copyright (c) 1985 McGraw-Hill Book Co. GmbH
0004 ;
00000000 0005      ORG 0
0006 ;
0000000B = 0007 MT_SUSJB EQU    $0B
FFFFFFF = 0008 IO      EQU    -1
0009 ;
0010 ; Intestazione per debugger, etc.
0011 ;
00000000 600C 0012      BRA.S  PALLA          ; salta al codice
00000002 00000000 0013      DEFL    0
00000006 4AFB 0014      DEFW    '4AFB          ; intestazione standard
00000008 0004 0015      DEFW    4
0000000A 43697263 0016      DEFB    'Circ'
0017 ;
0000000E 4BF900020B10 0018 PALLA: LEA    $20B10,A5          ; ind. standard di linea
0019 ;

```



```

00000014 722D      0020 ; Prima, palla che rotola verso destra
00000016 49FA008E   0021 ;
0000001A 7003      0022     MOVEQ    #$2D,D1      ; disegna 46 cicli interi
0000001C 7802      0023 1Z:    LEA     PALLA_DAT(PC),A4; inizio dati per ciclo
0000001E 6140      0024     MOVEQ    #3,D0      ; lung. 1 ciclo = 4 byte
00000020 5444      0025     MOVEQ    #2,D4      ; shift iniziale = 2
00000022 51C8FFFA   0026 2Z:    BSR.S   DRAW_WAIT    ; pausa, disegna palla
00000026 544D      0027     ADDQ     #2,D4      ; shift a destra di 1 bit
00000028 51C9FFEC   0028     DBRA     D0,2Z
0000002C 544D      0029 ;
0000002E 51C9FFEC   0030     ADDQ     #2,A5      ; salto a word successiva
00000030 51C9FFEC   0031     DBRA     D1,1Z      ; dello schermo
00000032 51C9FFEC   0032 ;
00000034 51C9FFEC   0033 ; Ora la palla ruota su se stessa (senza muoversi)
00000036 51C9FFEC   0034 ;
00000038 51C9FFEC   0035     MOVEQ    #8,D4      ; con shift di 8 posiz.
0000003A 51C9FFEC   0036     SUBQ     #2,A5      ; all'ultima posizione
0000003C 51C9FFEC   0037     MOVEQ    #13,D1     ; 20 cicli completi
0000003E 51C9FFEC   0038 3Z:    LEA     PALLA_DAT(PC),A4; inizio dati per ciclo
00000040 51C9FFEC   0039     MOVEQ    #3,D0      ; lung. 1 ciclo = 4 byte
00000042 51C9FFEC   0040 4Z:    BSR.S   DRAW_WAIT    ; pausa, disegna palla
00000044 51C9FFEC   0041     DBRA     D0,4Z
00000046 51C9FFEC   0042     DBRA     D1,3Z
00000048 51C9FFEC   0043 ;
0000004A 51C9FFEC   0044 ; Ora la palla rotola verso sinistra
0000004C 51C9FFEC   0045 ;
0000004E 51C9FFEC   0046     MOVEQ    #$2D,D1
00000050 51C9FFEC   0047 5Z:    LEA     PALLA_ULT(PC),A4; inizio dati per ciclo
00000052 51C9FFEC   0048     MOVEQ    #3,D0      ; lung. 1 ciclo = 4 byte
00000054 51C9FFEC   0049     MOVEQ    #6,D4      ; shift iniziale 6 posiz.
00000056 51C9FFEC   0050 6Z:    BSR.S   DRAW_WAIT    ; pausa, disegna palla
00000058 51C9FFEC   0051     SUBQ     #2,D4      ; spostata a sinistra 1 bit
0000005A 51C9FFEC   0052     SUB.W    #88,A4      ; indietro di 2 oggetti
0000005C 51C9FFEC   0053     DBRA     D0,6Z
0000005E 51C9FFEC   0054 ;
00000060 51C9FFEC   0055     SUBQ     #2,A5      ; alla word precedente
00000062 51C9FFEC   0056     DBRA     D1,5Z      ; su schermo
00000064 51C9FFEC   0057 ;
00000066 51C9FFEC   0058     BRA.S    PALLA      ; ripetizione senza fine
00000068 51C9FFEC   0059 ;
0000006A 51C9FFEC   0060 ; Pausa per ottenere un movimento regolare
0000006C 51C9FFEC   0061 ;
0000006E 51C9FFEC   0062 DRAW_WAIT:
00000070 51C9FFEC   0063     MOVE.W    D0,-(A7)      ; salva i contatori
00000072 51C9FFEC   0064     MOVE.W    D1,-(A7)
00000074 51C9FFEC   0065     MOVEQ    #MT_SUSJB,D0    ; sospensione
00000076 51C9FFEC   0066     MOVEQ    #10,D1      ; di questo job
00000078 51C9FFEC   0067     MOVEQ    #2,D3      ; per due periodi video
0000007A 51C9FFEC   0068     SUB.L     A1,A1
0000007C 51C9FFEC   0069     TRAP     #1
0000007E 51C9FFEC   0070     MOVE.W    (A7)+,D1
00000080 51C9FFEC   0071     MOVE.W    (A7)+,D0
00000082 51C9FFEC   0072 ;
00000084 51C9FFEC   0073 ; Routine che disegna un oggetto di 8*11 pixel
00000086 51C9FFEC   0074 ;
00000088 51C9FFEC   0075 ; (A4) : blocco di 11 long word che contengono
0000008A 51C9FFEC   0076 ; i pixel dell'oggetto

```

```

0077 ; (A5) : estremo in alto a sin. dell'oggetto nello schermo
0078 ;      che si sposta a destra di D4 pixel (modo S12)
0079 ;
0080 ; A4 aumenta di 11 ad ogni chiamata
0081 ; D5, D6 e D7 sono registri di lavoro
0082 ;
00000072 0083 DRAW_BX11:
00000072 2F0D 0084     MOVE.L A5,-(A7)      ; salva indir. schermo
00000074 7A0A 0085     MOVEQ #10,D5        ; traccia 11 linee
00000076 0086 DRAW_CICLO:
00000076 7C00 0087     MOVEQ #0,D6        ; cancella (cima dei)
00000078 7E00 0088     MOVEQ #0,D7        ; registri di lavoro
0000007A 0D0C0000 0089     MOVEP.W 0(A4),D6      ; "verde" e "flash" in D6
0000007E 0F0C0001 0090     MOVEP.W 1(A4),D7      ; "rosso" e "blu" in D7
00000082 E8BE 0091     ROR.L D4,D6        ; ruota l'oggetto
00000084 E8BF 0092     ROR.L D4,D7
00000086 0D8D0000 0093     MOVEP.W D6,0(A5)      ; e lo mette su schermo
0000008A 0F8D0001 0094     MOVEP.W D7,1(A5)
0000008E E19E 0095     ROL.L #8,D6        ; recupera bit mancante
00000090 E19F 0096     ROL.L #8,D7        ; dell'oggetto
00000092 584D 0097     ADDQ #4,A5
00000094 1AC6 0098     MOVE.B D6,(A5)+      ; e mette anche quello
00000096 1A87 0099     MOVE.B D7,(A5)      ; sullo schermo
0100 ;
00000098 584C 0101     ADDQ #4,A4        ; va a linea succ. oggetto
0000009A DAFCD007B 0102     ADD.W #80-S,A5      ; e a linea succ. schermo
0000009E 51CDFFD6 0103     DBRA D5,DRAW_CICLO
0104 ;
000000A2 2A5F 0105     MOVE.L (A7)+,A5      ; riprist. ind. schermo
000000A4 4E75 0106     RTS
0107 ;
000000A6 0108 PALLA_DAT:
000000A6 000AA0F0 0109     DEFL $000AA0F0
000000AA 203AA0FB 0110     DEFL $203AA0FB
000000AE 283E80EB 0111     DEFL $283E80EB
000000B2 28BE82EB 0112     DEFL $28BE82EB
000000B6 02AB2ABF 0113     DEFL $02AB2ABF
000000BA 22BB22BB 0114     DEFL $22BB22BB
000000BE A8FE80EA 0115     DEFL $A8FE80EA
000000C2 82EB28BE 0116     DEFL $82EB28BE
000000C6 022B28BC 0117     DEFL $022B28BC
000000CA 0A2F08AC 0118     DEFL $0A2F08AC
000000CE 0A0F00A0 0119     DEFL $0A0F00A0
0120 ;
000000D2 080E20B0 0121     DEFL $080E20B0
000000D6 283E28BC 0122     DEFL $283E28BC
000000DA 082E20BB 0123     DEFL $082E20BB
000000DE 02AB80EA 0124     DEFL $02AB80EA
000000E2 A2FB8AEF 0125     DEFL $A2FB8AEF
000000E6 A8FE2ABF 0126     DEFL $A8FE2ABF
000000EA A2FB8AEF 0127     DEFL $A2FB8AEF
000000EE 02AB80EA 0128     DEFL $02AB80EA
000000F2 082E20BB 0129     DEFL $082E20BB
000000F6 283E28BC 0130     DEFL $283E28BC
000000FA 080E20B0 0131     DEFL $080E20B0
0132 ;
000000FE 0A0F00A0 0133     DEFL $0A0F00A0

```

```

00000102 0A2F08AC      0134      DEFL      $0A2F08AC
00000106 022B28BC      0135      DEFL      $022B28BC
0000010A 82EB28BE      0136      DEFL      $82EB28BE
0000010E ABFE80EA      0137      DEFL      $ABFE80EA
00000112 88EE88EE      0138      DEFL      $88EE88EE
00000116 02AB2ABF      0139      DEFL      $02AB2ABF
0000011A 28BE82EB      0140      DEFL      $28BE82EB
0000011E 283E80E8      0141      DEFL      $283E80E8
00000122 203AA0FB      0142      DEFL      $203AA0FB
00000126 000AA0F0      0143      DEFL      $000AA0F0
0000012A      0144  PALLA_ULT:
0000012A 020B80E0      0145      DEFL      $020B80E0
0000012E 022B80E8      0146      DEFL      $022B80E8
00000132 223B88EC      0147      DEFL      $223B88EC
00000136 ABFE2ABF      0148      DEFL      $ABFE2ABF
0000013A 08AE20BA      0149      DEFL      $08AE20BA
0000013E 02AB80EA      0150      DEFL      $02AB80EA
00000142 08AE20BA      0151      DEFL      $08AE20BA
00000146 ABFE2ABF      0152      DEFL      $ABFE2ABF
0000014A 223B88EC      0153      DEFL      $223B88EC
0000014E 022B80E8      0154      DEFL      $022B80E8
00000152 020B80E0      0155      DEFL      $020B80E0
      0156 ;
      0157 END

```

Simboli:

```

00000072 DRAW_BX1  00000076 DRAW_CIC  00000060 DRAW_WAI  FFFFFFFF ID      00000008 MT_SUSJB
0000000E PALLA    000000A6 PALLA_DA 0000012A PALLA_UL

```

```

Errori:      0000
Byte liberi:  4F24
Codice (byte): 00156

```

Assembler terminato

Capitolo

Estensione del SuperBASIC

11

In questo capitolo ci occuperemo di quattro programmi, ciascuno dei quali costituisce, in qualche modo, una estensione del SuperBASIC.

Di ciascun programma viene dato il listato Assembler completo, preceduto da una breve descrizione. Nelle descrizioni si assume che il lettore abbia letto e compreso le descrizioni di quegli esempi (sia di questo capitolo che dei precedenti) che sono collegati a quello in esame. Ciò consente di evitare la maggior parte delle ripetizioni e di arrivare subito al nocciolo dell'argomento. Il package editor/Assembler utilizzato per la preparazione di questi programmi (e che viene descritto negli ultimi due capitoli di questo libro) è disponibile su cartuccia Microdrive.

Il listato delle istruzioni in esadecimale potrebbe essere usato per caricare manualmente i programmi in memoria. Sebbene ciò sia lungo, noioso e soggetto ad errori, dà almeno la possibilità di provare i programmi senza dover necessariamente comprare il package Assembler.

11.1 Uso dei programmi

Le procedure e funzioni all'interno dei quattro programmi devono essere inizializzate in modo da informare il SuperBASIC della loro esistenza. La routine che esegue questa operazione è contenuta in tutti e quattro i programmi, ed è discussa nei paragrafi 8.4 e 8.5. Per ottenere fisicamente il collegamento delle procedure da SuperBASIC, si può creare un file BOOT contenente dei comandi come i seguenti:

```
100 base=RESPR(dimens)
110 LBYTES nomefile,base
120 CALL base
130 NEW
```

Il primo comando predispone una fetta di RAM di dimensioni adeguate nell'area delle procedure residenti. Il file di tipo *__code* viene quindi caricato in questa area e chiamato con CALL. Ciò provoca un salto all'inizio del file di procedure il quale, a sua volta, esegue semplicemente la breve routine di inizializzazione.

11.2 Esempio 1 — CURSORE

Questo file contiene due procedure di estensione, CURSEN e CURDIS. La breve routine di inizializzazione si trova proprio all'inizio e comincia dalla label EXTEN. Dopo di essa viene la tabella di definizione delle procedure, che comincia dalla label PROC_DEF.

La funzione INKEY\$, contenuta nella ROM del QL, non provvede ad abilitare il cursore. È, d'altra parte, possibile che un programma applicativo scritto in SuperBASIC richieda l'abilitazione del cursore per effettuare l'input da tastiera con INKEY\$. CURSEN soddisfa questa esigenza in modo molto semplice. Le operazioni da fare sono: caricare SD_CURE in D0, caricare -1 in D3 e la appropriata ID di canale in A0; infine, si esegue una TRAP #3. La procedura CURDIS disabilita il cursore.

Sfortunatamente, sebbene la teoria sia semplice, la vita è sempre più dura del necessario. A causa di una di quelle inspiegabili distrazioni che capitano sempre nella produzione di nuovo software, la routine nella ROM del QL che trova la ID di un canale SuperBASIC non dispone di un vettore, così vi tocca scrivervela da soli! La routine completa comincia a CANALE ed è indispensabile per tutte quelle funzioni e procedure SuperBASIC che utilizzano dei canali. Il paragrafo 6.3 descrive la struttura dei canali.

Come potete osservare, la parte di codice che parte da CHAN_LOOK (appena prima di CHAN_EXIT) controlla se il canale è aperto. Per prima cosa, la ID di canale viene caricata nel registro A0 (dato che è lì che serve!). Secondo, la word meno significativa di A0 viene copiata in un registro dati qualunque, al solo scopo di settare i flag di condizione. Un canale già chiuso o non ancora aperto viene codificato con -1 (long word). Per sapere se un dato canale è aperto, bisogna controllare che la word meno significativa sia maggiore o uguale a zero. La word più significativa è quella che identifica effettivamente il canale, pertanto, come è ovvio, può contenere qualsiasi valore.

11.3 Esempio 2 — FUNZBAS

Non tutte le procedure vengono utilizzate per compiere attività varie: molto spesso il loro compito è solo quello di cercare o calcolare un valore per poi restituirlo. Di solito ciò si ottiene chiamando una funzione che restituisce un valore ma, talvolta, è necessario restituire più di un valore. In questo caso può essere conveniente restituire i valori tramite la lista di parametri di una procedura. Il file di questo esempio contiene due funzioni (MEAN e NHEX\$) e una procedura (TIME).

La funzione MEAN riceve uno o più valori, li converte in virgola mobile, poi li somma per mezzo di un loop utilizzando la routine aritmetica RI_ADD. A questo punto sullo stack aritmetico esiste un solo valore; insieme ad esso viene posto sullo stack il numero dei parametri che erano stati passati. La somma dei parametri viene poi divisa per quest'ultimo numero, dando così la media. Infine, viene stabilito il tipo e l'indirizzo dell'argomento da restituire (D0 viene predisposto dalle routine aritmetiche) e la funzione restituisce il controllo al programma chiamante.

La funzione NHEX\$ è leggermente più complessa, in quanto, quando viene restituita una stringa di lunghezza dispari, l'inizio della stringa deve coincidere con un indirizzo pari (allineamento con le word di memoria). Per prima cosa, vengono acquisiti i due argomenti (in formato intero in doppia precisione, così ci possono stare otto cifre esadecimali). La routine non si preoccupa granché della allocazione dello stack aritmetico, in quanto CA_GTLIN ha già provveduto a predisporre almeno 10 byte per caricare due interi in doppia precisione sullo stack. In seguito, il secondo intero in doppia precisione viene convertito (direttamente sul posto) in otto cifre esadecimali. Se le cifre generate sono in numero dispari, i caratteri vengono spostati verso il basso nella memoria di una locazione (in questo caso non può essere utilizzato l'autoincremento in quanto tutti i riferimenti all'interno dell'area SuperBASIC devono essere relativi ad A6). Infine, la word che contiene la lunghezza della stringa viene posta davanti alla stringa stessa e vengono determinati il tipo e l'indirizzo dell'argomento da restituire.

La procedura TIME restituisce due valori che danno l'ora in ore e minuti. La parte critica della procedura è la routine TIME_SET, che deve restituire un valore intero in una variabile intera o in virgola mobile, ma non in una variabile di controllo dei loop REPEAT e FOR. La prima parte di TIME_SET mette un intero sullo stack aritmetico, quindi verifica se il parametro è una variabile o se è un'entità alla quale non è ancora stato assegnato un valore. In seguito, la procedura decide se l'intero sullo stack deve essere convertito in virgola mobile oppure no. Infine, il valore viene assegnato utilizzando BP_LET.

11.4 Esempio 3 — ARRAY

Gli array vengono allocati in un modo prestabilito. È possibile assegnare valori a singoli elementi di array per mezzo di BP_LET, ma è molto facile scrivere procedure che modificano interi array. Gli esempi illustrati in questo programma, spostano parti di array o sub-array azzerando lo spazio che viene lasciato libero. La procedura MAKE_ROOM crea spazio per elementi aggiuntivi di un array e TAKE_ROOM fa l'opposto: rimuove elementi da un array.

La routine ROOM_SET prima determina la quantità di spazio che deve essere aggiunto (o tolto). In seguito, si destreggia tra i puntatori agli array per trovare l'indirizzo della base dell'array, la lunghezza di ciascuno degli elementi più significativi (per esempio, per l'array A(20,3) la lunghezza è $(3+1)*6$ byte) e il numero di elementi che devono essere spostati. Il resto delle istruzioni di MAKE_ROOM e TAKE_ROOM esegue semplicemente le operazioni di spostamento e cancellazione dei blocchi, riferendo gli indirizzamenti al registro A6.

Per vedere l'effetto di MAKE_ROOM, provate ad eseguire il seguente programma SuperBASIC (esteso):

```
DIM array(9,4)
FOR i=0 TO 9
  FOR j=0 TO 4
    array(i,j)=10*i+j
  END FOR j
END FOR i

PRINT array,
MAKE_ROOM array(3 TO 8),2
PRINT array,
```

Ora provate un programma simile, ma stavolta con un array di stringhe.

11.5 Esempio 4 — JOBS

L'ultimo esempio di espansione del SuperBASIC consiste di un insieme di procedure per il controllo dei job. C'è una procedura che visualizza l'elenco di tutti i job che esistono nel QL (JOBS) e quattro procedure per il controllo dei job (SJOB, KJOB, RJOB e PJOB). Queste ultime quattro procedure sono molto simili e richiedono tutte, come argomento, il numero e l'identificatore del job. Il numero è un indice per la tabella dei

job e l'identificatore è un codice unico per ogni job. Queste due entità, unite formano il job ID completo richiesto dalle chiamate al QDOS.

La procedura JOBS, che elenca i job presenti nel computer, non richiede parametri (a parte un possibile numero di canale) e il loop che scandisce l'albero dei job è abbastanza semplice. Dopo il loop, ritroviamo il nostro caro amico CHANNEL e, infine, la routine per formattare ed emettere le informazioni sui job (JOB__INF).

L'output viene formattato riempiendo un buffer con i caratteri che devono essere emessi e inviando la linea così formata al canale richiesto. Il buffer utilizzato è quello associato all'interprete SuperBASIC ed è lungo almeno 128 byte. I primi due byte vengono usati per memorizzare gli interi che devono essere aggiunti al buffer. Per aggiungere un numero, l'intero successivo viene caricato nel registro D1 e il puntatore al termine del campo viene messo in A5. JOB__NUM inserisce il numero all'inizio del buffer e CN__ITOD lo converte in caratteri nel buffer. Quindi JOB__NUM inserisce spazi (almeno uno) fino ad A5.

Infine, viene esaminato l'inizio del job per vedere se contiene un'intestazione standard e, in caso positivo, i caratteri che identificano il programma vengono copiati nel buffer. Osservate come i dati che formano l'intestazione dei job vengano trattati per mezzo di indirizzi assoluti, mentre il buffer dell'interprete viene indirizzato relativamente ad A6.

Programma 11.1 CURSORE

Estensioni al SuperBASIC

McGraw-Hill Book Co. GmbH Assembler 68000 v3.0A Pagina: 0001

```

0001 #H Estensioni al SuperBASIC
0002 ;
0003 ; Copyright (c) 1985 McGraw-Hill Book Co. GmbH
0004 ;
0005 ; Questo programma contiene le seguenti estensioni:
0006 ;
0007 ; CURSEN  abilita il cursore
0008 ; CURDIS  disabilita il cursore
0009 ;
0010          ORG 0
0011 ;
0012 ERR_OR  EQU  -4
0013 ERR_NO  EQU  -6
0014 ERR_BP  EQU  -15
0015 SD_CURE EQU  $E
0016 SD_CURS EQU  $F
0017 BP_INIT EQU  $110
0018 CA_GTINT EQU  $112
0019 BV_VVBAS EQU  $28
0020 BV_CHBAS EQU  $30
0021 BV_CHP  EQU  $34
0022 ;
0023 ; Entry point per inizializzazione

```

```

00000000 43FA000C      0024 ;
00000004 34780110      0025 EXTEN: LEA    PROC_DEF(PC),A1 ; lettura definizioni
00000008 4E92          0026      MOVE.W  BP_INIT,A2
0000000A 7000          0027      JSR      (A2)
0000000C 4E75          0028      MOVEQ   #0,D0          ; non ci sono errori
                                0029      RTS              ; ritorna al SuperBASIC
                                0030 ;
0000000E          0031 PROC_DEF:
00000010 0002          0032      DEFW    2              ; 2 procedure
00000012 001A          0033 1%:  DEFW    CURSEN-1%      ; offset risp. inizio
                                0034      DEFB    6,'CURSEN'      ; nome di 6 caratteri
                                0035      ALIGN
00000014 0014          0036 2%:  DEFW    CURDIS-2%      ;
                                0037      DEFB    6,'CURDIS'
                                0038      ALIGN
0000001C 06435552444953 0039      DEFW    0              ; fine delle procedure
                                0040      DEFW    0              ; 0 funzioni
                                0041      DEFW    0              ; fine delle funzioni
                                0042 ;
                                0043 ; Abilitazione del cursore
                                0044 ;
0000002A 780E          0045 CURSEN: MOVEQ   #SD_CURE,D4      ; carica chiave abil. curs.
0000002C 6002          0046      BRA.S    CUR_COM
                                0047 ;
                                0048 ; Disabilitazione del cursore
                                0049 ;
0000002E 780F          0050 CURDIS: MOVEQ   #SD_CURS,D4      ; carica chiave disab. curs.
                                0051 ;
                                0052 CUR_COM:
00000030          0053      BSR.S    CANALE          ; usa routine : ID -> A0
00000032 6114          0054      BNE.S    CUR_EXIT      ; ... OK?
00000034 BBCB          0055      CMP.L    A3,A5          ; ci sono parametri?
00000036 660A          0056      BNE.S    ERRR_BP
00000038 1004          0057      MOVE.B    D4,D0          ; mette chiave in D0
0000003A 363CFFFF      0058      MOVE.W  #1,D3          ; stabilisce timeout
0000003E 4E43          0059      TRAP     #3          ; (codice errore in D0)
00000040          0060 CUR_EXIT:
00000042 4E75          0061      RTS
00000044 70F1          0062 ERRR_BP:
                                0063      MOVEQ   #ERR_BP,D0      ; parametro non valido
                                0064      RTS
                                0065 ;
                                0066 ; Predisporre canale scelto o di default
                                0067 ; Parametri chiamata: A3 e A5 puntatori standard alla
                                0068 ; tabella dei nomi per i parametri
                                0069 ; Parametri di ritorno: D6 punta alla tabella dei canali
                                0070 ; AO = ID di canale
                                0071 ;
00000046          0072 CANALE:
00000048 7C01          0073      MOVEQ   #1,D6          ; canale default = #1
0000004A BBCB          0074      CMP.L    A3,A5          ; ci sono parametri?
0000004C 6720          0075      BEQ.S    CHAN_LOOK      ; ... no
                                0076 ;
                                0077      .        #7,1(A6,A3.L)      ; primo param. casuale?
00000052 6718          0078      BEQ.S    CHAN_LOOK      ; ... no
                                0079 ;
00000054 2F0D          0080      MOVE.L    A5,-(A7)      ; salva punt. 1' param.

```

```

00000056 2A4B      0081      MOVE.L A3,A5      ; il nuovo l' param.
00000058 504D      0082      ADDQ    #8,A5      ; e' a 8 byte dal fondo
0000005A 2F0D      0083      MOVE.L A5,-(A7)    ; (diventa nuovo fondo)
0000005C 34780112    0084      MOVE.W CA_6TINT,A2 ; lettura di un intero
00000060 4E92      0085      JSR     (A2)
00000062 265F      0086      MOVE.L (A7)+,A3      ; ripristina pontatori
00000064 2A5F      0087      MOVE.L (A7)+,A5      ; (non infl. codici cond.)
00000066 661C      0088      BNE.S  CHAN_EXIT    ; tutto bene?
00000068 3C369800    0089      MOVE.W 0(A6,A1.L),D6 ; valore va in D6
0090 ;
0000006C      0091 CHAN_LOOK:
0000006C CCFC0028    0092      MULU   ##28,D6      ; D6 (lungo) puntera' a
00000070 DCAE0030    0093      ADD.L  BV_CHBAS(A6),D6 ; tabella dei canali
00000074 BCAE0034    0094      CMP.L  BV_CHP(A6),D6   ; e' in tabella?
00000078 6C0C      0095      BGE.S  ERR__NO      ; ... no
0000007A 20766800    0096      MOVE.L 0(A6,D6.L),A0 ; predis. ID di canale
0000007E 3008      0097      MOVE.W A0,D0      ; e' aperto?
00000080 6B04      0098      BMI.S  ERR__NO      ; ... no
00000082 7000      0099      MOVEQ  #0,D0      ; nessun errore
00000084      0100 CHAN_EXIT:
00000084 4E75      0101      RTS
0102 ;
00000086      0103 ERR__NO:
00000086 70FA      0104      MOVEQ  #ERR_NO,D0   ; canale non aperto
00000088 4E75      0105      RTS
0106 ;
0107 END

```

Simboli:

```

00000110 BP_INIT    00000030 BV_CHBAS    00000034 BV_CHP      00000028 BV_VVBAS    00000046 CANALE
00000112 CA_6TINT   00000084 CHAN_EXIT 0000006C CHAN_LOOK 0000002E CURDIS    0000002A CURSEN
00000030 CUR_CDM    00000040 CUR_EXIT   00000042 ERRR_BP    FFFFFFFF1 ERR_BP    FFFFFFFFA ERR_NO
FFFFFFFFC ERR_DR     00000086 ERR__NO    00000000 EXTEN     0000000E PROC_DEF    0000000E SD_CURE
0000000F SD_CURS

```

```

Errori:      0000
Byte liberi: 4E48
Codice (byte): 0008A

```

Assembler terminato

Programma 11.2 FUNZBAS

Funzioni BASIC aggiuntive McGraw-Hill Book Co. 6mbH Assembler 68000 v3.0A Pagina: 0001

```

0001 #H Funzioni BASIC aggiuntive
0002 ;
0003 ; Copyright (c) 1985 McGraw-Hill Book Co. 6mbH
0004 ;
0005 ; x=MEAN (valore{,valore}) ritorna la media aritmetica
0006 ;                               di tutti i parametri
0007 ; x=NHEX$ (numero cifre esadecimali, numero)

```

```

0008 ;                               converte num. in stringa esad.
0009 ;
0010 ; TIME ore, minuti             ritorna ora (orologio di 12 h)
0011 ;
00000000 0012 ORG 0
0013 ;
00000013 = 0014 MT_RCLK EQU $13
000000FE = 0015 CN_ITOHL EQU $FE
00000110 = 0016 BP_INIT EQU $110
00000114 = 0017 CA_GTFP EQU $114
00000118 = 0018 CA_GTLIN EQU $118
0000011A = 0019 BV_CHRIX EQU $11A
0000011C = 0020 RI_EXEC EQU $11C
00000120 = 0021 BP_LET EQU $120
00000008 = 0022 RI_FLOAT EQU $08
0000000A = 0023 RI_ADD EQU $0A
00000010 = 0024 RI_DIV EQU $10
0025 ;
00000058 = 0026 BV_RIP EQU $58
0027 ;
FFFFFFF1 = 0028 ERR_BP EQU -15
0029 ;
0030 ; Inizializzazione
0031 ;
00000000 43FA000C 0032 LEA PROC_TAB(PC),A1 ; tabella definizione procedura
00000004 34780110 0033 MOVE.W BP_INIT,A2 ; aggiunta alla tabella BASIC
00000008 4E92 0034 JSR (A2)
0000000A 7000 0035 MOVEQ #0,D0 ; nessun errore
0000000C 4E75 0036 RTS
0037 ;
0000000E 0038 PROC_TAB:
0000000E 0001 0039 DEFW 1 ; una procedura
00000010 00A2 0040 1$: DEFW TIME-1$ ; offset
00000012 0454494D45 0041 DEFB 4,'TIME'
0042 ALIGN
00000018 0000 0043 DEFW 0 ; fine delle procedure
0000001A 0002 0044 DEFW 2 ; due funzioni
0000001C 0012 0045 2$: DEFW MEAN-2$
0000001E 044D45414E 0046 DEFB 4,'MEAN'
0047 ALIGN
00000024 0042 0048 3$: DEFW NHEX-3$
00000026 054E48455824 0049 DEFB 5,'NHEX$'
0050 ALIGN
0000002C 0000 0051 DEFW 0 ; fine delle funzioni
0052 ;
0053 ; Funzione MEAN
0054 ;
0000002E 34780114 0055 MEAN: MOVE.W CA_GTFP,A2 ; acquisisce numeri virg. mob.
00000032 4E92 0056 JSR (A2)
00000034 662A 0057 BNE.S MEAN_RTS ; ... qualcosa non va!
00000036 3803 0058 MOVE.W D3,D4 ; salva numero di parametri
00000038 6728 0059 BEQ.S ERRR_BP ; ... non ce n'erano
0000003A 5543 0060 SUBQ.W #2,D3 ; n-1 somme (prepara per DBRA)
0000003C 6D1C 0061 BLT.S MEAN_GET ; un solo numero - va ritornato
0000003E 3478011C 0062 MOVE.W RI_EXEC,A2 ; accede alle proc. aritmetiche
00000042 0063 ADD_LOOP:
00000042 700A 0064 MOVEQ #RI_ADD,D0 ; somma

```

```

00000044 4E92          0065      JSR      (A2)
00000046 6618          0066      BNE.S    MEAN_RTS      ; ... qualcosa non va!
00000048 51CBFFFB      0067      DBRA     D3,ADD_LOOP    ; ... ripete ciclo
                                0068 ;
                                0069      SUBQ     #2,A1          ; numero parametri sullo stack
0000004C 5549          0070      MOVE.W   D4,0(A6,A1.L)
0000004E 3D849800      0071      MOVEQ    #RI_FLOAT,D0    ; converte in reale
00000052 7008          0072      JSR      (A2)
00000054 4E92          0073      MOVEQ    #RI_DIV,D0     ; e lo usa come divisore
00000056 7010          0074      JSR      (A2)
00000058 4E92          0075 MEAN_SET:
0000005A          0076      MOVE.L   A1,BV_RIP(A6) ; indirizzo argomento di ritorno
0000005C 2D490058      0077      MOVEQ    #2,D4          ; e suo tipo
0000005E 7802          0078 MEAN_RTS:
00000060          0079      RTS
00000062          0080 ERRR_BP:
00000064 70F1          0081      MOVEQ    #ERR_BP,D0
00000066 4E75          0082      RTS
                                0083 ;
                                0084 ; Conversione in esadecimale
                                0085 ;
00000068 34780118      0086 NHX:  MOVE.W   CA_GTLIN,A2    ; acquisisce due interi doppia word
0000006A 4E92          0087      JSR      (A2)
0000006C 6642          0088      BNE.S    NHX_RTS      ; qualcosa non va
0000006E 5543          0089      SUBQ.W   #2,D3          ; ci volevano due argomenti
00000070 66F0          0090      BNE.S    ERRR_BP      ; in effetti, non erano due
00000072 28369800      0091      MOVE.L   0(A6,A1.L),D4    ; numero di cifre richieste
00000074 67EA          0092      BEQ.S    ERRR_BP      ; ... come, nessuna!!
00000076 0C440008      0093      CMP.W    #8,D4          ; non piu' di 8
00000078 62E4          0094      BHI.S    ERRR_BP      ; ("maggiore di" in compl. a 1)
                                0095 ;
0000007E 3049          0096      MOVE.W   A1,A0          ; due long word = 8 caratteri
00000080 5849          0097      ADDQ     #4,A1          ; ora solo una long word
00000082 347800FE      0098      MOVE.W   CN_ITOHL,A2    ; converte in 8 cifre esadec.
00000084 4E92          0099      JSR      (A2)
                                0100      ; qui sorgono i problemi!!
00000088 08040000      0101      BTST     #0,D4          ; stringa di lung. dispari?
0000008C 6712          0102      BEQ.S    NHX_SET_LEN    ; ... no
0000008E 3204          0103      MOVE.W   D4,D1          ; si', quindi va spostata
00000090 92C1          0104      SUB.W    D1,A1          ; sposta il puntatore
00000092          0105 10%:          ; sposta una cifra
00000094 1DB6980098FF  0106      MOVE.B   0(A6,A1.L),0-1(A6,A1.L)
00000096 5249          0107      ADDQ     #1,A1
00000098 51C9FFF6      0108      DBRA     D1,10%          ; sposta D1+1 caratteri
0000009A 5549          0109      SUBQ     #2,A1          ; A1=A1+1, + 1 per gli spostamenti
0000009C          0110 NHX_SET_LEN:
0000009E 92C4          0111      SUB.W    D4,A1          ; manda A1 all'inizio della stringa
000000A2 5549          0112      SUBQ     #2,A1          ; ... e piu' in la' di una word
000000A4 3D849800      0113      MOVE.W   D4,0(A6,A1.L)    ; carica lunghezza stringa
000000A8 2D490058      0114      MOVE.L   A1,BV_RIP(A6)    ; prepara punt. a stack aritmetico
000000AC 7801          0115      MOVEQ    #1,D4          ; ... ed emette la stringa
000000AE 7000          0116      MOVEQ    #0,D0
000000B0          0117 NHX_RTS:
000000B2 4E75          0118      RTS
                                0119 ;
                                0120 ; Procedura che restituisce l'ora del giorno
                                0121 ;

```

```

000000B2 7010      0122 TIME: MOVEQ #16,D0      ; due parametri?
000000B4 D0B8      0123      ADD.L A3,D0
000000B6 70B8      0124      SUB.L A5,D0
000000B8 66AB      0125      BNE.S ERRR_BP      ; ... no
                                0126 ;
000000BA 720C      0127      MOVEQ #12,D1      ; spazio per 2 num. virg. mob.
000000BC 347B011A  0128      MOVE.W BV_CHRIX,A2      ; (ci sara' disordine)
000000C0 4E92      0129      JSR (A2)
                                0130 ;
000000C2 7013      0131      MOVEQ #MT_RCLCK,D0      ; legge l'orologio
000000C4 4E41      0132      TRAP #1
000000C6 82FCA8C0  0133      DIVU #43200,D1      ; meta' giorno in secondi
000000CA 4241      0134      CLR.W D1      ; ... senza i giorni
000000CC 4841      0135      SWAP D1
000000CE 82FC003C  0136      DIVU #60,D1      ; meta' giorno in minuti
000000D2 48C1      0137      EXT.L D1      ; ... senza i secondi
000000D4 82FC003C  0138      DIVU #60,D1      ; divide in ore e minuti
000000D8 2801      0139      MOVE.L D1,D4      ; salva minuti (parte sup. di D4)
                                0140 ;
000000DA 6108      0141      BSR.S TIME_SET      ; predis. un parametro di ritorno
000000DC 6640      0142      BNE.S TIME_RTS      ; ... qualcosa non va
000000DE 5048      0143      ADDQ #8,A3      ; punt. param. punta al successivo
000000E0 4844      0144      SWAP D4
000000E2 3204      0145      MOVE.W D4,D1      ; predis. l'altro param. di ritorno
000000E4           0146 TIME_SET:
000000E4 226E0058  0147      MOVE.L BV_RIP(A6),A1
000000EB 92FC0002  0148      SUB.W #2,A1      ; lo mette sullo stack di RI
000000EC 3DB19800  0149      MOVE.W D1,0(A6,A1.L)
                                0150 ;
000000F0 4A36B800  0151      TST.B 0(A6,A3.L)      ; il param. ha gia' un valore?
000000F4 6708      0152      BEQ.S TIME_TYPE      ; ... no, tutto a posto
000000F6 0C360002B800 0153      CMP.B #2,0(A6,A3.L)      ; e' una variabile?
000000FC 661E      0154      BNE.S ERRR_BP1      ; ... no, non si puo' assegn. valore
000000FE           0155 TIME_TYPE:
000000FE 720F      0156      MOVEQ #F,D1      ; maschera i separatori
00000100 C236B801  0157      AND.B 1(A6,A3.L),D1
00000104 5501      0158      SUBQ.B #2,D1      ; controlla di che tipo e'
00000106 6D14      0159      BLT.S ERRR_BP1      ; nessuno o stringa
00000108 6E08      0160      BGT.S TIME_LET      ; intero - nessuna conversione
0000010A 7008      0161      MOVEQ #RI_FLOAT,D0      ; virg. mob. - converte in virg. mob.
0000010C 347B011C  0162      MOVE.W RI_EXEC,A2
00000110 4E92      0163      JSR (A2)
00000112           0164 TIME_LET:
00000112 2D490058  0165      MOVE.L A1,BV_RIP(A6)      ; ptr stack RI punta al valore
00000116 347B0120  0166      MOVE.W BP_LET,A2      ; pone valore in struttura dati
0000011A 4ED2      0167      JMP (A2)
0000011C           0168 ERRR_BP1:
0000011C 70F1      0169      MOVEQ #ERRR_BP,D0
0000011E           0170 TIME_RTS:
0000011E 4E75      0171      RTS
                                0172 ;
                                0173 END

```

Simboli:

```

00000042 ADD_LOOP      00000110 BP_INIT      00000120 BP_LET      0000011A BV_CHRIX      00000058 BV_RIP
00000114 CA_GTFP      00000118 CA_GTLIN      000000FE CR_ITOHL      00000062 ERRR_BP      0000011C ERRR_BP1

```

```

FFFFFFFF1 ERR_BP      0000002E MEAN      00000060 MEAN_RTS  0000005A MEAN_SET  00000013 MT_RCLK
00000066 NHEX         00000080 NHEX_RTS 000000A0 NHEX_SET  0000000E PROC_TAB  0000000A RI_ADD
00000010 RI_DIV       0000011C RI_EXEC   00000008 RI_FLOAT  000000B2 TIME      00000112 TIME_LET
0000011E TIME_RTS    000000E4 TIME_SET  000000FE TIME_TYP

```

Errori: 0000

Byte liberi: 4DAB

Codice (byte): 00120

Assembler terminato

Programma 11.3 ARRAY

Estensioni del SuperBASIC

McGraw-Hill Book Co. GmbH Assembler 68000 v3.0A Pagina: 0001

```

0001 #H Estensioni del SuperBASIC
0002 ;
0003 ; Copyright (c) 1985 McGraw-Hill Book Co. GmbH
0004 ;
0005 ; Questo file contiene le seguenti estensioni
0006 ;
0007 ; MAKE_ROOM array,n      alloca spazio per n elementi
0008 ;                        aggiuntivi in un array
0009 ; TAKE_ROOM array,n      riduce di n elementi un array
0010 ;
0011 ORG 0
0012 ;
0013 ERR_OR EQU -4
0014 ERR_NO EQU -6
0015 ERR_BP EQU -15
0016 SD_CURE EQU $E
0017 SD_CURS EQU $F
0018 BP_INIT EQU $110
0019 CA_GTINT EQU $112
0020 BV_VVBAS EQU $2B
0021 BV_CHBAS EQU $30
0022 BV_CHP EQU $34
0023 ;
0024 ; Entry point per l'inizializzazione
0025 ;
0026 EXTEN: LEA PROC_DEF(PC),A1 ; acquisisce tabella procedure
0027 MOVE.W BP_INIT,A2
0028 JSR (A2)
0029 MOVEQ #0,D0 ; nessun errore
0030 RTS ; ritorna al SuperBASIC
0031 ;
0032 PROC_DEF:
0033- DEFW 4 ; solo 2 nomi, ma lunghi
0034 1Z: DEFW MAKE_ROOM-1Z ; offset risp. al codice
0035 DEFB 9,'MAKE_ROOM' ; nome
0036 ALIGN
0037 2Z: DEFW TAKE_ROOM-2Z
0038 DEFB 9,'TAKE_ROOM'
0039 DEFW 0 ; fine procedure

```

```

0000002A 0000      0040      DEFW 0          ; 0 funzioni
0000002C 0000      0041      DEFW 0          ; fine delle funzioni
                                0042 ;
                                0043 ;
                                0044 ; Procedure di manipolazione di array
                                0045 ;
0000002E      0046 MAKE_ROOM:
0000002E 6146      0047      BSR.S ROOM_SET      ; inizializza puntatori array
00000030 6620      0048      BNE.S MAKE_RTS
00000032 4BF5800   0049      LEA 0(A5,D5.L),A5      ; alloca spazio - alto -> basso
00000036 49F54800  0050      LEA 0(A5,D4.L),A4      ; inizial. punt. destinazione
0000003A      0051 MAKE_MOVE:
0000003A 554C      0052      SUBQ #2,A4          ; predecrementa
0000003C 554D      0053      SUBQ #2,A5          ; e prosegue
0000003E 3DB6DB00C800 0054      MOVE.W 0(A6,A5.L),0(A6,A4.L)
00000044 5545      0055      SUBQ #2,D5
00000046 62F2      0056      BHI.S MAKE_MOVE
00000048      0057 MAKE_CLEAR:
00000048 554C      0058      SUBQ #2,A4          ; predecrementa
0000004A 4276C800  0059      CLR.W 0(A6,A4.L)      ; e azzerà il bit avanzato
0000004E 5544      0060      SUBQ #2,D4
00000050 62F6      0061      BHI.S MAKE_CLEAR
00000052      0062 MAKE_RTS:
00000052 4E75      0063      RTS
                                0064 ;
00000054      0065 TAKE_ROOM:
00000054 6120      0066      BSR.S ROOM_SET      ; iniz. puntatori all'array
00000056 661C      0067      BNE.S TAKE_RTS
00000058 49F54800  0068      LEA 0(A5,D4.L),A4      ; iniz. puntatore sorgente
0000005C      0069 TAKE_MOVE:
0000005C 3DB6C800DB00  0070      MOVE.W 0(A6,A4.L),0(A6,A5.L)
00000062 544C      0071      ADDQ #2,A4          ; e postincrementa
00000064 544D      0072      ADDQ #2,A5
00000066 5545      0073      SUBQ #2,D5
00000068 62F2      0074      BHI.S TAKE_MOVE
0000006A      0075 TAKE_CLEAR:
0000006A 4276DB00  0076      CLR.W 0(A6,A5.L)      ; azzerà il bit piu' in alto
0000006E 544D      0077      ADDQ #2,A5          ; e postincrementa
00000070 5544      0078      SUBQ #2,D4
00000072 62F6      0079      BHI.S TAKE_CLEAR
00000074      0080 TAKE_RTS:
00000074 4E75      0081      RTS
                                0082 ;
                                0083 ; Codice comune per le routine di gestione spazio array
                                0084 ; Ritorna D4 distanza di spostamento
                                0085 ; D5 quantita' da spostare
                                0086 ; A5 indirizzo base dell'array
                                0087 ;
00000076      0088 ROOM_SET:
00000076 5048      0089      ADDQ #8,A3          ; ignora l'array per un po'
00000078 BBCE      0090      CMP.L A3,A5          ; e' rimasto qualche argomento?
0000007A 6F56      0091      BLE.S ERR_BP1
0000007C 34780112  0092      MOVE.W CA_GTINT,A2      ; ci vuole un intero
00000080 4E92      0093      JSR (A2)
00000082 6648      0094      BNE.S ROOM_RTS      ; qualcosa non va
00000084 5343      0095      SUBQ.W #1,D3          ; solo uno
00000086 664A      0096      BNE.S ERR_BP1      ; qualcosa va storto

```



```

0000008B 3B369800      0097      MOVE.W 0(A6,A1.L),D4 ; stabilisce distanza spostamento
0000008C 6F40        0098      BLE.S  ERR_OR      ; qualcosa non va
                                0099 ;
0000008E 514B        0100      SUBQ  #8,A3
00000090 0C360003B800  0101      CMP.B  #3,0(A6,A3.L) ; deve essere un array
00000096 663A        0102      BNE.S  ERR_BP1
0000009B 720F        0103      MOVER  #F,D1      ; maschera i separatori
0000009A C2368B01  0104      AND.B  1(A6,A3.L),D1 ; acquisisce tipo array
                                0105 ;
0000009E 2A6E002B  0106      MOVE.L  BV_VVBAS(A6),A5 ; acquisisce base area VV
000000A2 2B76B804  0107      MOVE.L  4(A6,A3.L),A4
000000A6 D9CD        0108      ADD.L  A5,A4      ; quindi la base del descrittore
000000AB DBF6C800  0109      ADD.L  0(A6,A4.L),A5 ; ... e la base dell'array
                                0110 ;
000000AC 3C36C80B  0111      MOVE.W  8(A6,A4.L),D6 ; lunghezza elemento
000000B0 5501        0112      SUBQ.B  #2,D1      ; adegua a seconda del tipo array
000000B2 6D0B        0113      BLT.S  ROOM_SIZE ; niente in caso di stringhe
000000B4 6E04        0114      BGT.S  ROOM_BY_2 ; #2 per gli interi
000000B6 CCFC0003  0115      MULU  #3,D6      ; #6 per num. in virg. mob.
000000BA      0116      ROOM_BY_2:
000000BA DC46        0117      ADD.W  D6,D6      ; assume lung. elem. < 64K
000000BC      0118      ROOM_SIZE:
000000BC 3A36C806  0119      MOVE.W  6(A6,A4.L),D5 ; acquisisce n. tot. di elementi
000000C0 5245        0120      ADDQ.W  #1,D5      ; dimensione massima + 1
000000C2 9A44        0121      SUB.W  D4,D5      ; e quindi n. elem. da spostare
000000C4 6F0B        0122      BLE.S  ERR_OR
000000C6 C8C6        0123      MULU  D6,D4      ; converte distanza di spost. in byte
000000C9 CAC6        0124      MULU  D6,D5      ; num. di byte da spostare
                                0125 ;
000000CA 7000        0126      MOVER  #0,D0
000000CC      0127      ROOM_RTS:
000000CC 4E75        0128      RTS
000000CE      0129      ERR_OR:
000000CE 70FC        0130      MOVER  #ERR_OR,D0
000000D0 4E75        0131      RTS
000000D2      0132      ERR_BP1:
000000D2 70F1        0133      MOVER  #ERR_BP,D0
000000D4 4E75        0134      RTS
                                0135 ;
                                0136      END

```

Simboli:

```

00000110 BP_INIT      00000030 BV_CHBAS      00000034 BV_CHP      0000002B BV_VVBAS      00000112 CA_STINT
FFFFFFFF ERR_BP      FFFFFFFF ERR_NO      FFFFFFFF ERR_OR      000000D2 ERR_BP1      000000CE ERR_OR
00000000 EXTEN      0000004B MAKE_CLE      0000003A MAKE_MOV      0000002E MAKE_R00      00000052 MAKE_RTS
0000000E PROC_DEF      000000BA ROOM_BY_      000000CC ROOM_RTS      00000076 ROOM_SET      000000BC ROOM_SIZ
0000000E SD_CURE      0000000F SD_CURS      0000006A TAKE_CLE      0000005C TAKE_MOV      00000054 TAKE_R00
00000074 TAKE_RTS

```

```

Errori:      0000
Byte liberi:  4DE4
Codice (byte): 000D6

```

Assembler terminato

Programma 11.4 JOBS

Controllo job da SuperBASIC

McGraw-Hill Book Co. GmbH Assembler 68000 v3.0A Pagina: 0001

```

0001 *H Controllo job da SuperBASIC
0002 ;
0003 ; Copyright (c) 1985 McGraw-Hill Book Co. GmbH
0004 ;
0005 ; Questo file contiene le seguenti estensioni:
0006 ;
0007 ; JOBS [#n]          elenco job presenti
0008 ; SJOB nr,tag,tempo  sospende un job
0009 ; KJOB nr,tag        uccide un job
0010 ; RJOB nr,tag        riattiva un job
0011 ; PJOB nr,tag,priorita' assegna priorit  a un job
0012 ;
0013             ORG 0
0014 ;
0015 ERR_NO EQU -6
0016 ERR_BP EQU -15
0017 MT_JINF EQU $02
0018 MT_FRJOB EQU $05
0019 MT_SUSJB EQU $08
0020 MT_RELJB EQU $09
0021 MT_PRIOR EQU $0B
0022 CN_ITOD EQU $F2
0023 BP_INIT EQU $110
0024 CA_STINT EQU $112
0025 BV_BFBAS EQU $00
0026 BV_CHBAS EQU $30
0027 BV_CHP EQU $34
0028 IO_SSTRG EQU $07
0029 ;
0030 ; Entry point per inizializzazione
0031 ;
0032 JOB: LEA PROC_DEF(PC),A1 ; legge puntatore a tabella
0033 MOVE.W BP_INIT,A2
0034 JSR (A2)
0035 MOVEQ #0,D0 ; non ci sono errori
0036 RTS ; ritorna a SuperBASIC
0037 ;
0038 PROC_DEF:
0039 DEFW 5 ; 5 procedure
0040 1X: DEFW JOBS-1X ; offset dall'inizio
0041 DEFB 4,'JOBS' ; nome di 4 caratteri
0042 ALIGN
0043 2X: DEFW SJOB-2X
0044 DEFB 4,'SJOB'
0045 ALIGN
0046 3X: DEFW KJOB-3X
0047 DEFB 4,'KJOB'
0048 ALIGN
0049 4X: DEFW RJOB-4X
0050 DEFB 4,'RJOB'
0051 ALIGN
0052 5X: DEFW PJOB-5X
0053 DEFB 4,'PJOB'

```

```

0000003B 0000      0054      ALIGN
0000003A 0000      0055      .DEFW 0          ; fine delle procedure
0000003C 0000      0056      .DEFW 0          ; 0 funzioni
0000003C 0000      0057      .DEFW 0          ; fine delle funzioni
0000003E 7808      0058      ;
00000040 7A03      0059      SJOB: MOVEQ #MT_SUSJB,D4 ; sospende job
00000042 6010      0060      MOVEQ #3,D5          ; acquisisce 3 parametri
00000044 7805      0061      BRA.S JOB_COMMON
00000046 7A02      0062      KJOB: MOVEQ #MT_FRJOB,D4 ; rimuove job d'autorita'
00000048 600A      0063      MOVEQ #2,D5          ; acquisisce 2 parametri
0000004A 7809      0064      BRA.S JOB_COMMON
0000004C 7A02      0065      RJOB: MOVEQ #MT_RELJB,D4 ; riattiva job
0000004E 6004      0066      MOVEQ #2,D5          ; acquisisce 2 parametri
00000050 7808      0067      BRA.S JOB_COMMON
00000052 7A03      0068      PJOB: MOVEQ #MT_PRIOR,D4 ; stabilisce priorit  job
00000052 7A03      0069      MOVEQ #3,D5          ; acquisisce 3 parametri
00000054          0070      ;
00000054          0071      JOB_COMMON:
00000054 34780112    0072      MOVE.W CA_GTINT,A2      ; acquisisce un po' di interi
00000058 4E92      0073      JSR (A2)
0000005A 6644      0074      BNE.S JOB_EXIT
0000005C 6645      0075      CMP.W D5,D3          ; sono abbastanza?
0000005E 6642      0076      BNE.S ERRR_BP
00000060 22369800    0077      MOVE.L 0(A6,A1.L),D1      ; acquis. tag e ID del job
00000064 4841      0078      SWAP D1          ; (nell'ordine corretto)
00000066 36369804    0079      MOVE.W 4(A6,A1.L),D3      ; acquis. timeout (SJOB)
0000006A 3403      0080      MOVE.W D3,D2          ; o priorit  (PJOB)
0000006C 2004      0081      MOVE.L D4,D0          ; predis. chiave operazione
0000006E 93C9      0082      SUB.L A1,A1          ; indirizzo flag (SJOB) = 0
00000070 4E41      0083      TRAP #1
00000072 4E75      0084      RTS
00000072 4E75      0085      ;
00000072 4E75      0086      ; Scrive un elenco di job al canale specif. o di default
00000072 4E75      0087      ;
00000074 6130      0088      JOBS: BSR.S CHANNEL
00000076 6628      0089      BNE.S JOB_EXIT          ; ... tutto bene?
00000078 B5CB      0090      CMP.L A3,A5          ; non ci dovr. essere param.
0000007A 6626      0091      BNE.S ERRR_BP
0000007C 2848      0092      MOVE.L A0,A4          ; salva ID di canale
0000007E 43FA00F0    0093      LEA JOB_HEAD(PC),A1      ; scrive intestazione
00000082 3419      0094      MOVE.W (A1)+,D2          ; ... predis. lunghezza
00000084 610000C4    0095      BSR JOB_WRITE
00000088 7200      0096      MOVEQ #0,D1          ; parte dal job 0.
00000088 7200      0097      ;
0000008A 2801      0098      IZ: MOVE.L D1,D4          ; salva ID di questo job
0000008C 7002      0099      MOVEQ #MT_JINF,D0      ; acquis. info. su job
0000008E 7400      0100      MOVEQ #0,D2          ; scansione di tutto l'albero
00000090 4E41      0101      TRAP #1
00000092 4AB0      0102      TST.L D0          ; ci rinuncia se c'  errore
00000094 660A      0103      BNE.S JOB_EXIT
00000096 2A01      0104      MOVE.L D1,D5          ; salva ID job successivo
00000098 6150      0105      BSR.S JOB_INF          ; emette info. su job
0000009A 6604      0106      BNE.S JOB_EXIT
0000009C 2205      0107      MOVE.L D5,D1          ; se job successivo
0000009E 66EA      0108      BNE.S IZ          ; <> 0 => continua
0000009E 66EA      0109      ;
000000A0          0110      JOB_EXIT:

```

```

000000A0 4E75      0111      RTS
                  0112 ;
000000A2          0113 ERRR_BP:
000000A2 70F1      0114      MOVEQ #ERRR_BP,D0 ; parametro non appropriato
000000A4 4E75      0115      RTS
                  0116 ;
                  0117 ; Stabilisce canale specificato o di default
                  0118 ; Parametri passati: A3 e A5, puntatori standard alla
                  0119 ; tabella dei nomi per i parametri
                  0120 ; Parametri ritornati: D6 puntatore alla tabella di canale
                  0121 ; A0 ID di canale
                  0122 ;
000000A6          0123 CHANNEL:
000000A6 7C01      0124      MOVEQ #1,D6 ; canale di default = #1
000000A8 BBC8      0125      CMP.L A3,A5 ; ci sono parametri?
000000AA 6720      0126      BEQ.S CHAN_LOOK ; ... no
                  0127 ;
000000AC 0B360007B801 0128      BTST #7,1(A6,A3.L) ; dati senza senso nel 1° param.?
000000B2 6718      0129      BEQ.S CHAN_LOOK ; ... no
                  0130 ;
000000B4 2F0D      0131      MOVE.L A5,-(A7) ; salva punt. al param. di testa
000000B6 2A4B      0132      MOVE.L A3,A5 ; stabilisce nuovo estremo sup.
000000B8 504D      0133      ADDQ #8,A5 ; a 8 byte dal fondo
000000BA 2F0D      0134      MOVE.L A5,-(A7) ; (diventa nuovo fondo)
000000BC 34780112 0135      MOVE.W CA_GTINT,A2 ; acquisisce un intero
000000C0 4E92      0136      JSR (A2)
000000C2 265F      0137      MOVE.L (A7)+,A3 ; ripristina i puntatori
000000C4 2A5F      0138      MOVE.L (A7)+,A5 ; (non altera codici condiz.)
000000C6 661C      0139      BNE.S CHAN_EXIT ; e' andato tutto bene?
000000C8 3C369800 0140      MOVE.W 0(A6,A1.L),D6 ; acquis. valore in D6
                  0141 ;
000000CC          0142 CHAN_LOOK:
000000CC CCFC0028 0143      MULU #28,D6 ; converte D6 (lungo) a punt. a
000000D0 0CAE0030 0144      ADD.L BV_CHBAS(A6),D6 ; tabella di canale
000000D4 8CAE0034 0145      CMP.L BV_CHP(A6),D6 ; e' contenuto in tabella
000000D8 6C0C      0146      BGE.S ERRR_NO ; ... no
000000DA 20766800 0147      MOVE.L 0(A6,D6.L),A0 ; assegna ID di canale
000000DE 3008      0148      MOVE.W A0,D0 ; e' aperto il canale?
000000E0 6B04      0149      BMI.S ERRR_NO ; ... no
000000E2 7000      0150      MOVEQ #0,D0 ; non ci sono errori
000000E4          0151 CHAN_EXIT:
000000E4 4E75      0152      RTS
                  0153 ;
000000E6          0154 ERRR_NO:
000000E6 70FA      0155      MOVEQ #ERRR_NO,D0 ; canale non aperto
000000E8 4E75      0156      RTS
                  0157 ;
                  0158 ; Routine di scrittura informazioni su job
                  0159 ;
000000EA          0160 JOB_INF:
000000EA 2C02      0161      MOVE.L D2,D6 ; distrugge contenuto
000000EC 2E03      0162      MOVE.L D3,D7 ; dei registri
000000EE 2F08      0163      MOVE.L A0,-(A7) ; e l'indirizzo del job
                  0164 ;
000000F0 206E0000 0165      MOVE.L BV_BFBAS(A6),A0 ; usa il buffer del SuperBASIC
000000F4 544B      0166      ADDQ #2,A0 ; lascia spazio in fondo per
000000F6 2248      0167      MOVE.L A0,A1 ; uno stack RI per 1 intero

```

```

000000F8 2A48      0168      MOVE.L A0,A5      ; e inizial. il punt. al campo
                                0169 ;
000000FA 3204      0170      MOVE.W D4,D1      ; numero del job
000000FC 584D      0171      ADDQ  #4,A5      ; in campo di 4 caratteri
000000FE 6156      0172      BSR.S JOB_NUM
00000100 2204      0173      MOVE.L D4,D1      ; tag del job
00000102 4841      0174      SWAP D1      ; che e' nella word + signif.
00000104 5E4D      0175      ADDQ  #7,A5      ; in un campo di 7 caratteri
00000106 614E      0176      BSR.S JOB_NUM
00000108 3206      0177      MOVE.W D6,D1      ; il numero del proprietario
0000010A 5C4D      0178      ADDQ  #6,A5      ; in un campo di 5+1 caratt.
0000010C 6148      0179      BSR.S JOB_NUM
0000010E 4A87      0180      TST.L D7      ; controlla se job e' sospeso
00000110 6A06      0181      BPL.S JOB_W_PR
00000112 1DBC005388FF 0182      MOVE.B #'S',0(A6,A0.L); si', aggiunge flag S
00000118      0183 JOB_W_PR:
0000011B 7200      0184      MOVEQ  #0,D1
0000011A 1207      0185      MOVE.B D7,D1      ; priorit' (byte)
0000011C 584D      0186      ADDQ  #4,A5      ; in un campo di 4 caratteri
0000011E 6136      0187      BSR.S JOB_NUM
                                0188 ;
                                0189 ; Sono stati letti tutti i numeri - ora ricerca un nome
                                0190 ; (lungo al massimo 60 caratteri)
                                0191 ;
00000120 7416      0192      MOVEQ  #22,D2      ; 21 caratteri nel buffer (+LF)
00000122 245F      0193      MOVE.L (A7)+,A2      ; riprist. indirizzo base job
00000124 5C4A      0194      ADDQ  #6,A2      ; cerca nei byte 6 e 7
00000126 0C5A4AFB 0195      CMP.W  ##4AFB,(A2)+ ; ... il flag
0000012A 6616      0196      BNE.S JOB_INF_DONE ; non c'e'
0000012C 321A      0197      MOVE.W (A2)+,D1      ; acquisisce lunghezza
0000012E 0C41003C 0198      CMP.W  #60,D1      ; e' troppo grande?
00000132 620E      0199      BHI.S JOB_INF_DONE ; ... si', lascia stare
00000134 D441      0200      ADD.W D1,D2      ; ci sono altri caratteri
00000136 6006      0201      BRA.S 2%      ; da mettere dentro
0000013B 1D9A8B00 0202 1%: MOVE.B (A2)+,0(A6,A0.L); copia i caratteri del nome
0000013C 5248      0203      ADDQ  #1,A0
0000013E 51C9FFFF 0204 2%: DBRA D1,1%
                                0205 ;
00000142      0206 JOB_INF_DONE:
00000142 1DBC000A8B00 0207      MOVE.B ##A,0(A6,A0.L) ; mette <LF> alla fine
00000148 4E44      0208      TRAP  #4      ; Al relativo ad A6
0000014A      0209 JOB_WRITE:
0000014A 7007      0210      MOVEQ  #ID_SSTR6,D0 ; invia la stringa
0000014C 76FF      0211      MOVEQ  #-1,D3      ; senza timeout
0000014E 204C      0212      MOVE.L A4,A0      ; riprist. ID di canale
00000150 4E43      0213      TRAP  #3
00000152 4A80      0214      TST.L D0      ; verifica di errore
00000154 4E75      0215      RTS
                                0216 ;
                                0217 ; Scrive un intero su una linea seguito da spazi fino a fine linea
0218 ;      (A6,A1.L) punta alla base del buffer, A1 non va perso
0219 ;      (A6,A0.L) punta al buffer
0220 ;      (A6,A5.L) punta alla fine del campo
0221 ;
00000156      0222 JOB_NUM:
00000156 5549      0223      SUBQ  #2,A1      ; fa posto per un intero
00000158 3D819800 0224      MOVE.W D1,0(A6,A1.L) ; e lo mette dentro

```

```

0000015C 347800F2      0225      MOVE.W  CN_ITOD,A2      ; converte l'intero in decimale
00000160 4E92          0226      JSR      (A2)
00000162          0227 JOB_N_LOOP:
00000162 1DBC00208800      0228      MOVE.B  #' ',0(A6,A0.L) ; mette dentro uno spazio
00000168 5248          0229      ADDQ    #1,A0      ; e avanza il puntatore
0000016A B8C8          0230      CMP.L   A0,A5      ; campo non ancora pieno?
0000016C 62F4          0231      BHI.L   JOB_N_LOOP ; ci riprova
0000016E 4E75          0232      RTS
                                0233 ;
                                0234 ; Intestazione per JOBS
                                0235 ;
00000170          0236 JOB_HEAD:
00000170 001A          0237      DEFW    26
00000172 4A6F6220746167202020 0238      DEFB    'Job tag  appart.  prior.'
                                0239 ;
                                0240 END

```

Simboli:

| | | | | |
|-------------------|-------------------|-------------------|-------------------|-------------------|
| 00000110 BP_INIT | 00000000 BV_BFBAS | 00000030 BV_CHBAS | 00000034 BV_CHP | 00000112 CA_GTINT |
| 000000A6 CHANNEL | 000000E4 CHAN_EXI | 000000CC CHAN_LOD | 000000F2 CN_ITOD | 000000A2 ERRR_BP |
| 000000E6 ERRR_NO | FFFFFFF1 ERR_BP | FFFFFFFA ERR_NO | 00000007 ID_SSTRG | 00000000 JOB |
| 00000074 JOBS | 00000054 JOB_COMM | 000000A0 JOB_EXIT | 00000170 JOB_HEAD | 000000EA JOB_INF |
| 00000142 JOB_INF_ | 00000156 JOB_NUM | 00000162 JOB_N_LO | 0000014A JOB_WRIT | 00000118 JOB_W_PR |
| 00000044 KJOB | 00000005 MT_FRJOB | 00000002 MT_JINF | 0000000B MT_PRIOR | 00000009 MT_RELJB |
| 00000008 MT_SUSJB | 00000050 PJOB | 0000000E PROC_DEF | 0000004A RJOB | 0000003E SJOB |

```

Errori:      0000
Byte liberi:  4CF4
Codice (byte): 0018C

```

Assembler terminato

In questo capitolo ci occuperemo di un unico programma, piuttosto grande, che estende il linguaggio SuperBASIC dotandolo di istruzioni per l'accesso casuale ai file.

Del programma viene dato il listato Assembler completo, preceduto da una breve descrizione. Nella descrizione si assume che il lettore abbia letto e compreso le descrizioni di quegli esempi (specialmente quelli del capitolo 11) che sono collegati a quello in esame. Ciò consente di evitare la maggior parte delle ripetizioni e di arrivare subito al nocciolo dell'argomento. Il package editor/Assembler utilizzato per la preparazione di questi programmi (e che viene descritto negli ultimi due capitoli di questo libro) è disponibile su cartuccia Microdrive.

12.1 Uso del programma

Le procedure e funzioni all'interno del programma devono essere inizializzate per informare il SuperBASIC della loro esistenza. La routine che esegue questa operazione è contenuta nel programma ed è spiegata nei paragrafi 8.4 e 8.5. Per collegare fisicamente le procedure da SuperBASIC, si può creare un file BOOT contenente i seguenti comandi:

```
100 base=RESPR(dimens)
110 LBYTES mdvn__filer__code, base
120 CALL base
130 NEW
```

Questo segmento di programma per prima cosa predispone una opportuna fetta di RAM nell'area delle procedure residenti. In seguito il file di tipo `_code` viene caricato in questa area e viene chiamato per mezzo di una CALL. Ciò provoca un salto all'inizio del file di procedure il quale, a sua volta, esegue semplicemente la breve routine di inizializzazione.

12.2 Esempio — FILER

In molti sistemi la gestione dell'accesso diretto ai file viene complicata dal fatto che i dati devono essere inseriti in record di dimensioni prefissate. Nel QL tale problema non esiste: è possibile leggere o scrivere dati di qualsiasi lunghezza in qualsiasi posizione del file. Da ciò segue che è facile scrivere procedure per il trattamento di record di lunghezza fissa, ma, poiché ciò avrebbe un interesse più che altro storico, qui non ce ne occupiamo.

In questo programma vengono definite delle procedure per leggere (GET) e scrivere (PUT) su file una lista di valori di tipi diversi, per leggere (BGET) e scrivere (BPUT) singoli byte su file e per posizionare il puntatore al file.

Viene anche definita una funzione che restituisce il puntatore corrente al file. Con queste routine è possibile creare file ad accesso sequenziale, con indice o con collegamento a lista. Gli algoritmi di movimento di file tra le unità di memoria di massa e la memoria del QL fanno sì che parti di file separate da una grande distanza possano essere acquisite nello stesso momento; inoltre, il comando di posizionamento del file può essere usato per sovrapporre le operazioni di acquisizione da file, aumentando ulteriormente l'efficienza.

GET, PUT, BGET e BPUT sono totalmente ridirezionabili e funzionano con qualunque dispositivo di I/O (come network o porte seriali). BGET e BPUT possono essere emulate con le procedure contenute nella ROM del QL:

| | | | |
|------|------|------------|------------------------|
| BGET | #n,x | equivale a | x=CODE(INKEY\$(#n,-1)) |
| BPUT | #n,x | equivale a | PRINT #n,CHR\$(x); |

Le routine ausiliarie di questo gruppo di procedure sono le più importanti. C'è, ovviamente, la fedele CHANNEL. FSTRG e SSTRG riducono leggermente la dimensione del codice accumulando in se stesse la parte comune delle istruzioni per la lettura e la scrittura di stringhe su file. PUT_ON_A1 è una routine di uso generale che acquisisce il valore di un parametro di un dato tipo e lo mette sullo stack aritmetico. TYPE_SET ha il compito di esaminare i parametri uno alla volta per controllare che siano effettivamente delle variabili; inoltre, gestisce un

flag di tipo in modo tale che, per esempio, PUT può scrivere due byte su file in caso di valore intero, sei byte per numeri in virgola mobile e $2+n$ byte per le stringhe.

Il programma potrebbe essere considerevolmente più conciso se si fossero usati alcuni trucchi per convertire i salti lunghi in salti brevi. Gli unici trucchi usati nel codice, però, sono solo quelli che riguardano il trattamento dello stack (A7): in alcuni casi le uscite dalle routine avvengono semplicemente rimuovendo l'indirizzo di ritorno dallo stack; ciò permette di risparmiare sia istruzioni che tempo.

Nel caso di SET_POS e POS, il ritorno dalle TRAP è immediato e gli errori "not complete" ed "end of file" vengono soppressi. Il posizionamento dei file avviene sempre immediatamente, quindi "not complete" indica semplicemente che il blocco di file richiesto non è ancora in RAM.

POS presenta un problema: il valore che essa restituisce è un intero a 32 bit. Ma non ci sono interi in doppia precisione in SuperBASIC! L'intero in doppia precisione può essere memorizzato come numero in virgola mobile senza perdita di precisione, ma in questo formato deve essere normalizzato. La routine di normalizzazione usata in questo esempio è abbastanza lenta ma è semplice: il numero viene semplicemente shiftato verso l'alto finché non avviene l'overflow, quindi viene shiftato indietro di una posizione. Un metodo di normalizzazione più veloce consiste nel provare a shiftare prima di 16 bit, poi di 7, 4, 2 e 1.

Programma 12.1 FILER

```

GESTIONE FILE                                McGraw-Hill Book Co. 6mbH Assembler 68000 v3.0A   Pagina: 0001

0001 #H GESTIONE FILE
0002 ;
0003 ; Copyright (c) 1985 McGraw-Hill Book Co. 6mbH
0004 ;
0005 ; Gestione dell'accesso diretto ai file
0006 ;
0007 ; Questo file contiene le seguenti estensioni
0008 ;
0009 ; GET #n {,variabile} legge valore/i da file
0010 ; BGET #n {,variabile} legge 1 o piu' byte da file
0011 ; PUT #n {,valore} scrive valore/i su file
0012 ; BPUT #n {,valore} scrive 1 o piu' byte su file
0013 ; SET_POS #n,valore assegna valore a puntatore al file
0014 ; POS {#n} funzione che ritorna il valore
0015 ; corrente del puntatore al file
0016 ;
0017 ;
0018 ORG 0
0019 ;
0020 ERR_NC EQU -1
0021 ERR_NO EQU -6
0022 ERR_EF EQU -10

```

```

FFFFF1 = 0023 ERR_BP EQU -15
FFFFFE = 0024 ERR_OV EQU -18
00000001 = 0025 IO_FBYTE EQU $01
00000003 = 0026 IO_FSTRG EQU $03
00000005 = 0027 IO_SBYTE EQU $05
00000007 = 0028 IO_SSTRG EQU $07
00000042 = 0029 FS_POSAB EQU $42
00000043 = 0030 FS_POSRE EQU $43
00000110 = 0031 BP_INIT EQU $110
00000112 = 0032 CA_GTINT EQU $112
00000114 = 0033 CA_GTFP EQU $114
00000116 = 0034 CA_GTSTR EQU $116
00000118 = 0035 CA_GTLIN EQU $118
0000011A = 0036 BV_CHRIX EQU $11A
0000011C = 0037 RI_EXEC EQU $11C
00000120 = 0038 BP_LET EQU $120
00000008 = 0039 RI_FLOAT EQU $08
00000030 = 0040 BV_CHBAS EQU $30
00000034 = 0041 BV_CHP EQU $34
00000058 = 0042 BV_RIP EQU $58
0043 ;
0044 ; Entry point per inizializzazione
0045 ;
00000000 43FA000C 0046 FILES: LEA PROC_DEF(PC),A1 ; punt. a definiz. procedure
00000004 34780110 0047 MOVE.W BP_INIT,A2
00000008 4E92 0048 JSR (A2)
0000000A 7000 0049 MOVEQ #0,D0 ; nessun errore
0000000C 4E75 0050 RTS ; ritorna al SuperBASIC
0051 ;
0000000E 0052 PROC_DEF:
0000000E 0005 0053 DEFW 5 ; 5 procedure
00000010 0032 0054 1Z: DEFW GET-1Z ; offset risp. inizio
00000012 03474554 0055 DEFB 3,'GET'
0056 ALIGN
00000016 0088 0057 2Z: DEFW BGET-2Z
00000018 0442474554 0058 DEFB 4,'BGET'
0059 ALIGN
0000001E 00BC 0060 3Z: DEFW PUT-3Z
00000020 03505554 0061 DEFB 3,'PUT'
0062 ALIGN
00000024 00EE 0063 4Z: DEFW BPUT-4Z
00000026 0442505554 0064 DEFB 4,'BPUT'
0065 ALIGN
0000002C 010E 0066 5Z: DEFW SET_POS-5Z
0000002E 075345545F504F53 0067 DEFB 7,'SET_POS'
0068 ALIGN
00000036 0000 0069 DEFW 0
00000038 0001 0070 DEFW 1 ; fine delle procedure
0000003A 011C 0071 6Z: DEFW FPOS-6Z ; 1 funzione
0000003C 03504F53 0072 DEFB 3,'POS'
00000040 0000 0073 DEFW 0 ; fine delle funzioni
0074 ;
0075 ; Legge oggetto da un file
0076 ;
00000042 6100018C 0077 GET: BSR CHAN_SET
00000046 0078 GET_LOOP:
00000046 61000198 0079 BSR CHK_R16 ; c'e' posto sullo stack RI?

```

```

0000004A 610001A2      0080      BSR      TYPE_SET      ; acquisisce tipo del successivo
0000004E 6E32         0081      BGT.S     GET_INT      ; ... intero
00000050 6734         0082      BEQ.S     GET_FP      ; ... virgola mobile
0083 ;
00000052 7402         0084      MOVEQ    #2,D2      ; acquisisce lunghezza stringa
00000054 61000160      0085      BSR      FSTRG_PUSH
00000058 6600014C      0086      BNE      EXIT_B      ; qualcosa non va
0000005C 38369800      0087      MOVE.W   0(A6,A1.L),D4 ; salvataggio
00000060 7200         0088      MOVEQ    #0,D1
00000062 3204         0089      MOVE.W   D4,D1      ; arrotonda a byte di ind. pari
00000064 5241         0090      ADDQ.W   #1,D1
00000066 08810000      0091      BCLR     #0,D1
0000006A 2A01         0092      MOVE.L   D1,D5      ; salva valore arrotondato
0000006C 61000174      0093      BSR      CHK_RI      ; c'e' spazio per stringa?
00000070 92C5         0094      SUB.W    D5,A1      ; sposta punt. allo stack in giu'
00000072 3D849800      0095      MOVE.W   D4,0(A6,A1.L) ; e inserisce lunghezza
00000076 5449         0096      ADDQ     #2,A1
00000078 3404         0097      MOVE.W   D4,D2      ; acquisisce caratteri della stringa
0000007A 6100013C      0098      BSR      FSTRG
0000007E 5549         0099      SUBQ     #2,A1      ; e aggiunge lunghezza stringa
00000080 600A         0100      BRA.S     GET_LET      ; nel valore ritornato
0101 ;
00000082 7402         0102 GET_INT: MOVEQ    #2,D2      ; acquisisce 2 byte
00000084 6002         0103      BRA.S     GET_BYTES
00000086 7406         0104 GET_FP: MOVEQ    #6,D2      ; acquisisce 6 byte
0105 ;
00000088         0106 GET_BYTES:
0000008B 6100012C      0107      BSR      FSTRG_PUSH      ; 'push' byte su stack (A1)
0000008C         0108 GET_LET:
0000008C 66000118      0109      BNE      EXIT_B      ; c'e' stato un errore in lettura
00000090 2D490058      0110      MOVE.L   A1,BV_RIP(A6) ; predispone puntatore allo stack
00000094 34780120      0111      MOVE.W   BP_LET,A2      ; e assegna il valore
00000098 4E92         0112      JSR      (A2)
0000009A 504B         0113      ADDQ     #8,A3      ; si porta al parametro successivo
0000009C 60AB         0114      BRA.S     GET_LOOP      ; e prosegue
0115 ;
0116 ; Legge un byte (e lo converte a virgola mobile, se necessario)
0117 ;
0000009E 61000130      0118 BGET:  BSR      CHAM_SET      ; predispos. ID di canale, etc.
000000A2         0119 BGET_LOOP:
000000A2 6100013C      0120      BSR      CHK_RI6      ; c'e' posto per 1 virgola mobile?
000000A6 61000146      0121      BSR      TYPE_SET      ; scopre il tipo
000000AA 6D0000FB      0122      BLT      ERRR_BP      ; ... stringa non va bene
000000AE 1C01         0123      MOVE.B   D1,D6      ; salva il flag di tipo
000000B0 7401         0124      MOVEQ    #1,D2      ; acquisisce un byte
000000B2 61000102      0125      BSR      FSTRG_PUSH
000000B6 660000EE      0126      BNE      EXIT_B      ; qualcosa non va
000000BA 5349         0127      SUBQ     #1,A1      ; carica byte=0 sullo stack
000000BC 42369800      0128      CLR.B    0(A6,A1.L)
000000C0 4A06         0129      TST.B    D6      ; ci vuole num. in virgola mobile?
000000C2 6E08         0130      BGT.S     BGET_LET      ; ... no
000000C4 7008         0131      MOVEQ    #RI_FLOAT,D0    ; ... si', converte a virgola mobile
000000C6 3478011C      0132      MOVE.W   RI_EXEC,A2
000000CA 4E92         0133      JSR      (A2)
000000CC         0134 BGET_LET:
000000CC 2D490058      0135      MOVE.L   A1,BV_RIP(A6) ; predispos. punt. a stack aritmetico
000000D0 34780120      0136      MOVE.W   BP_LET,A2      ; assegna valore

```

```

000000D4 4E92          0137      JSR      (A2)
000000D6 504B          0138      ADDQ     #8,A3          ; passa a parametro successivo
000000DB 60CB          0139      BRA.S    BGET_LOOP
                                0140 ;
                                0141 ; Scrive i dati sul file
                                0142 ;
000000DA 610000F4      0143 PUT:   BSR      CHAN_SET      ; predis. ID di canale, etc.
000000DE          0144 PUT_LOOP:
000000DE 6100010E      0145      BSR      TYPE_SET      ; scopre il tipo
000000E2 6712          0146      BEQ.S    PUT_FP        ; virgola mobile
000000E4 6E1C          0147      BGT.S    PUT_INT      ; intero
                                0148 ;
000000E6 34780116      0149      MOVE.W   CA_GTSTR,A2    ; acquisisce stringa
000000EA 610000BE      0150      BSR      PUT_ON_A1     ; una sola
000000EE 34369800      0151      MOVE.W   0(A6,A1.L),D2  ; trova lunghezza
000000F2 5442          0152      ADDQ     #2,D2          ; scrive lung. e stringa su file
000000F4 6016          0153      BRA.S    PUT_FILE
000000F6          0154 PUT_FP:
000000F6 34780114      0155      MOVE.W   CA_GTFP,A2    ; acquisisce num. in virgola mobile
000000FA 610000AE      0156      BSR      PUT_ON_A1     ; uno solo
000000FE 7406          0157      MOVEQ    #6,D2          ; e scrive 6 byte su file
00000100 600A          0158      BRA.S    PUT_FILE
00000102          0159 PUT_INT:
00000102 34780112      0160      MOVE.W   CA_GTINT,A2    ; acquisisce un intero
00000106 610000A2      0161      BSR      PUT_ON_A1     ; uno solo
0000010A 7402          0162      MOVEQ    #2,D2          ; e scrive 2 byte su file
0000010C          0163 PUT_FILE:
0000010C 610000AE      0164      BSR      SSTRG        ; scrive i byte su file
00000110 60CC          0165      BRA.S    PUT_LOOP      ; prosegue
                                0166 ;
                                0167 ; Scrive un byte sul file
                                0168 ;
00000112 610000BC      0169 BPUT:   BSR      CHAN_SET      ; predis. ID di canale, etc.
00000116          0170 BPUT_LOOP:
00000116 B7EF0004      0171      CMP.L    4(A7),A3        ; fine della lista?
0000011A 670000BA      0172      BEQ      EXIT_8          ; si' (D0 ha gia' un valore)
0000011E 34780112      0173      MOVE.W   CA_GTINT,A2    ; acquisisce un intero
00000122 610000B6      0174      BSR      PUT_ON_A1     ; uno solo
00000126 4A369800      0175      TST.B    0(A6,A1.L)     ; byte + signif. deve essere = 0
0000012A 6704          0176      BEQ.S    BPUT_FILE      ; tutto bene
0000012C 70EE          0177      MOVEQ    #ERR_OV,D0     ; no, lo marca come overflow
0000012E 6076          0178      BRA.S    EXIT_8
00000130          0179 BPUT_FILE:
00000130 5249          0180      ADDQ     #1,A1          ; solo il byte - signif
00000132 7401          0181      MOVEQ    #1,D2          ;
00000134 610000B6      0182      BSR      SSTRG        ; va sul file
00000138 60DC          0183      BRA.S    BPUT_LOOP
                                0184 ;
                                0185 ; Assegna valore al file pointer
                                0186 ;
0000013A          0187 SET_POS:
0000013A 61000094      0188      BSR      CHAN_SET      ; predis. ID di canale, etc.
0000013E 34780118      0189      MOVE.W   CA_GTLIN,A2    ; e acquisisce un intero a 32 bit
00000142 4E92          0190      JSR      (A2)
00000144 5343          0191      SUBQ.W   #1,D3          ; uno solo
00000146 665C          0192      BNE.S    ERRR_BP
00000148 22369800      0193      MOVE.L    0(A6,A1.L),D1  ; predis. puntatore al file
0000014C 7042          0194      MOVEQ    #FS_POSAB,D0   ; posiziona file a posizione assoluta

```

```

0000014E 7600      0195      MOVEQ    #0,D3          ; ritorna immediatamente
00000150 2057      0196      MOVE.L    (A7),A0        ; predis. ID di canale
00000152 4E43      0197      TRAP      #3
00000154 603A      0198      BRA.S     POS_D0        ; controlla D0 per ritorno valido
                                0199 ;
                                0200 ; Legge puntatore al file
                                0201 ;
00000156 6178      0202 FPOS:   BSR.S     CHAN_SET      ; predis. ID di canale, etc.
00000158 BBCB      0203      CMP.L     A3,A5          ; non ci devono essere altri param.
0000015A 6648      0204      BNE.S     ERRR_BP
0000015C 6100082     0205      BSR      CHK_R16        ; prep. spazio per virg. mob. ritorno
00000160 2849      0206      MOVE.L    A1,A4          ; salva puntatore allo stack RI
00000162 7043      0207      MOVEQ    #FS_POSRE,D0      ; posiziona file a posizione relativa
00000164 7200      0208      MOVEQ    #0,D1          ; distante nessun byte
00000166 7600      0209      MOVEQ    #0,D3          ; e ritorna immediatamente
00000168 2057      0210      MOVE.L    (A7),A0        ; predis. ID di canale
0000016A 4E43      0211      TRAP      #3
0000016C 224C      0212      MOVE.L    A4,A1          ; riprist. punt. allo stack RI
0000016E 5D49      0213      SUBQ     #6,A1          ; ci scrive
00000170 42769800     0214      CLR.W     0(A6,A1.L)      ; esponente = 0
00000174 383C0820     0215      MOVE.W    #0820,D4      ; predis. esponente non normaliz. (+1)
00000178                                0216 POS_NORM:
00000178 5344      0217      SUBQ.W    #1,D4          ; decrementa esponente
0000017A E3B1      0218      ASL.L     #1,D1          ; e moltiplica mantissa per 2
0000017C 6708      0219      BEQ.S     POS_MANT      ; ... se zero non prosegue
0000017E 6BF8      0220      BVC.S     POS_NORM      ; se non c'e' overflow, ci riprova
00000180 E291      0221      ROXR.L    #1,D1          ; riprist. mantissa a non overflow
00000182 3DB49800     0222      MOVE.W    D4,0(A6,A1.L) ; mette esponente effettivo su stack RI
00000186                                0223 POS_MANT:
00000186 2DB19802     0224      MOVE.L    D1,2(A6,A1.L) ; e la mantissa
0000018A 2D490058     0225      MOVE.L    A1,BV_RIP(A6) ; predis. punt. a stack RI
0000018E 7802      0226      MOVEQ     #2,D4          ; e ritorna il tipo
                                0227 ;
                                0228 ; Controlla D0 al ritorno dalle chiamate per posizione file
                                0229 ;
00000190 4AB0      0230 POS_D0: TST.L     D0          ; tutto bene?
00000192 6712      0231      BEQ.S     EXIT_0         ; si'
00000194 72FF      0232      MOVEQ    #ERR_NC,D1      ; 'not complete' e' abbast. normale
00000196 B0B1      0233      CMP.L     D1,D0          ;
00000198 6706      0234      BEQ.S     EXIT_OK8      ;
0000019A 72F6      0235      MOVEQ    #ERR_EF,D1      ; 'end of file' va ancora bene
0000019C B0B1      0236      CMP.L     D1,D0          ;
0000019E 6606      0237      BNE.S     EXIT_0         ; no, non e' 'end of file'
000001A0                                0238 EXIT_OK8:
000001A0 7000      0239      MOVEQ    #0,D0          ; si', val. 'nessun errore' in D0
000001A2 6002      0240      BRA.S     EXIT_0
                                0241 ;
000001A4                                0242 ERRR_BP:
000001A4 70F1      0243      MOVEQ    #ERR_BP,D0      ; parametro errato ('bad parameter')
000001A6                                0244 EXIT_0:
000001A6 504F      0245      ADDQ     #8,A7          ; rimuove ID di canale e la cima
000001A8 4E75      0246      RTS              ; della lista dei parametri
                                0247 ;
                                0248 ; Mette oggetto seguente su stack punt. da A1 (A2 = CA_GT...)
                                0249 ;
000001AA                                0250 PUT_ON_A1:
000001AA 4BE8000B     0251      LEA      B(A3),A5      ; acquisisce un solo oggetto

```

```

000001AE 4E92      0252      JSR      (A2)          ; chiama il tipo appropriato
000001B0 264D      0253      MOVE.L   A5,A3          ; passa all'oggetto successivo
000001B2 6642      0254      BNE.S    EXIT_12       ; rimuove ind. di ritorno, ID di
000001B4 4E75      0255      RTS              ; canale e A5 che era salvato
                                0256 ;
                                0257 ; Acquisisce i byte
                                0258 ;
000001B6          0259 FSTRG_PUSH:
000001B6 92C2      0260      SUB.W    D2,A1          ; trova posto su stack di A1
000001B8 7003      0261 FSTRG: MOVEQ   #10_FSTRG,D0 ; acquis. numero noto di byte
000001BA 6002      0262      BRA.S    TRAP4_3
                                0263 ;
                                0264 ; Invia i byte
                                0265 ;
000001BC 7007      0266 SSTRG: MOVEQ   #10_SSTRG,D0 ; invia numero noto di byte
                                0267 ;
                                0268 TRAP4_3:
000001BE          0269      TRAP      #4          ; A1 relativo ad A6
000001BE 4E44      0270      MOVEQ   #-1,D3        ; nessun timeout
000001C0 76FF      0271      MOVE.L   4(A7),A0       ; predis. ID di canale
000001C2 206F0004  0272      TRAP      #3
000001C6 4E43      0273      SUB.W    D1,A1          ; riassegna ad A1 valore originale
000001C8 92C1      0274      TST.L    D0            ; verifica se ritorno in errore
000001CA 4A80      0275      BNE.S    EXIT_12
000001CC 6628      0276      RTS
000001CE 4E75      0277 ;
                                0278 ; Predis. e salva ID di canale e cima della lista dei param.
                                0279 ;
000001D0          0280 CHAN_SET:
000001D0 6144      0281      BSR.S    CHANNEL       ; acquisisce ID di canale
000001D2 6608      0282      BNE.S    GET_OUT       ; ritorno diretto
000001D4 2257      0283      MOVE.L   (A7),A1        ; acquis. indirizzo di ritorno
000001D6 2EBD      0284      MOVE.L   A5,(A7)        ; salva cima della lista di param.
000001D8 2F08      0285      MOVE.L   A0,-(A7)      ; e l'ID di canale
000001DA 4ED1      0286      JMP      (A1)          ; e ritorna
                                0287 ;
                                0288 GET_OUT:
000001DC          0289      ADDQ    #4,A7          ; rimuove un ind. di ritorno
000001DC 584F      0290      RTS              ; e ritorna al SuperBASIC
000001DE 4E75      0291 ;
                                0292 ; Verifica se c'e' posto per 6 o D1 byte sullo stack RI
                                0293 ;
000001E0          0294 CHK_RI6:
000001E0 7206      0295      MOVEQ   #6,D1          ; c'e' posto per 6 byte?
000001E2          0296 CHK_RI:
000001E2 3478011A  0297      MOVE.W    BV_CHRIX,A2       ; c'e' posto su stack RI?
000001E6 4E92      0298      JSR      (A2)
000001E8 226E0058  0299      MOVE.L   BV_RIP(A6),A1      ; predis. punt. a stack RI
000001EC 4E75      0300      RTS
                                0301 ;
                                0302 ; Predisporre tipo parametro successivo
                                0303 ;
000001EE          0304 TYPE_SET:
000001EE B7EF0008  0305      CMP.L    8(A7),A3          ; fine della lista di parametri?
000001F2 6D06      0306      BLT.S    TYPE_VAR       ; ... no
000001F4 7000      0307      MOVEQ   #0,D0          ; ... si', va bene
000001F6          0308 EXIT_12:

```

```

000001F6 584F      0309      ADDQ      #4,A7          ; rimuove ind. di ritorno
000001FB 60AC      0310      BRA.S     EXIT_8          ; ed esce
000001FA          0311      TYPE_VAR:
000001FA 1236B800      0312      MOVE.B   0(A6,A3.L),D1    ; acquisisce tipo del nome
000001FE 343C00C5      0313      MOVE.W   #C5,D2          ; maschera dei tipi ammessi
00000202 0302      0314      BTST      D1,D2
00000204 670C      0315      BEQ.S     TYPE_BP          ; non ammesso
00000206 720F      0316      MOVEQ     #F,D1          ; maschera i separatori
0000020B C236B801      0317      AND.B     1(A6,A3.L),D1
0000020C 6704      0318      BEQ.S     TYPE_BP          ; e' uguale a 0
0000020E 5501      0319      SUBQ.B    #2,D1          ; assegna -ve per stringhe, 0 per virg.
00000210 4E75      0320      RTS              ; mobile e +ve per interi
00000212          0321      TYPE_BP:
00000212 584F      0322      ADDQ      #4,A7          ; rimuove ind. di ritorno
00000214 60BE      0323      BRA.S     ERRR_BP          ; e segnala errore 'bad parameter'
                                0324 ;
                                0325 ; Predisporre canale assegnato o di default
                                0326 ; Parametri di chiamata: A3 e A5, puntatori standard alla
                                0327 ; tabella dei nomi per i parametri
                                0328 ; Parametri di ritorno: D6, puntatore alla tabella di canale
                                0329 ; A0, ID di canale
                                0330 ;
00000216          0331      CHANNEL:
00000216 7C01      0332      MOVEQ     #1,D6          ; canale di default e' #1
00000218 BBCB      0333      CMP.L     A3,A5          ; ci sono parametri?
0000021A 6720      0334      BEQ.S     CHAN_LOOK      ; ... no
                                0335 ;
0000021C 08360007B801      0336      BTST      #7,1(A6,A3.L) ; dati senza senso nel 1' param.?
00000222 6718      0337      BEQ.S     CHAN_LOOK      ; ... no
                                0338 ;
00000224 2F0D      0339      MOVE.L     A5,-(A7)        ; salva punt. al 1' param.
00000226 2A4B      0340      MOVE.L     A3,A5          ; assegna posiz. nuovo 1' param.
00000228 504D      0341      ADDQ      #8,A5          ; a 8 byte dal fondo
0000022A 2F0D      0342      MOVE.L     A5,-(A7)        ; (quando fatto, diventa nuovo fondo)
0000022C 347B0112      0343      MOVE.W     CA_GTINT,A2    ; acquisisce un intero
00000230 4E92      0344      JSR        (A2)
00000232 265F      0345      MOVE.L     (A7)+,A3        ; ripristina puntatori ai parametri
00000234 2A5F      0346      MOVE.L     (A7)+,A5        ; (non influenza codici di condizione)
00000236 661C      0347      BNE.S     CHAN_EXIT      ; e' andato tutto bene?
00000238 3C369B00      0348      MOVE.W     0(A6,A1.L),D6 ; D6 sostituisce il default
                                0349 ;
                                0350      CHAN_LOOK:
0000023C          0351      MULLU     #D6,D6          ; D6 (32 bit) punta a tabella di canale
0000023C CCFC002B      0352      ADD.L     BV_CHBAS(A6),D6
00000240 DCAE0030      0353      CMP.L     BV_CHP(A6),D6 ; e' in tabella?
00000244 BCAE0034      0354      BGE.S     ERRR_NO          ; ... no
00000248 6C0C      0355      MOVE.L     0(A6,D6.L),A0 ; assegna ID di canale
0000024A 20766800      0356      MOVE.W     A0,D0          ; e' aperto?
0000024E 3008      0357      BMI.S     ERRR_NO          ; ... no
00000250 6B04      0358      MOVEQ     #0,D0          ; nessun errore
00000252 7000      0359      CHAN_EXIT:
00000254          0360      RTS
00000254 4E75      0361      ERRR_NO:
00000256          0362      MOVEQ     #ERR_NO,D0      ; canale non aperto
00000256 70FA      0363      RTS
00000258 4E75      0364 ;
                                0365      END

```

Simboli:

| | | | | |
|-------------------|-------------------|-------------------|-------------------|-------------------|
| 0000009E BGET | 000000CC BGET_LET | 000000A2 BGET_LOD | 00000112 BPUT | 00000130 BPUT_FIL |
| 00000116 BPUT_LOD | 00000110 BP_INIT | 00000120 BP_LET | 00000030 BV_CHBAS | 00000034 BV_CHP |
| 0000011A BV_CHRIX | 0000005B BV_RIP | 00000114 CA_GTFP | 00000112 CA_GTINT | 00000118 CA_GTLIN |
| 00000116 CA_GSTR | 00000216 CHANNEL | 00000254 CHAN_EXI | 0000023C CHAN_LOD | 000001D0 CHAN_SET |
| 000001E2 CHK_RI | 000001E0 CHK_RI6 | 000001A4 ERRR_BP | 00000256 ERRR_NO | FFFFFFF1 ERR_BP |
| FFFFFFF6 ERR_EF | FFFFFFF7 ERR_NC | FFFFFFFA ERR_NO | FFFFFFFEE ERR_OV | 000001F6 EXIT_12 |
| 000001A6 EXIT_8 | 000001A0 EXIT_DKB | 00000000 FILES | 00000156 FPOS | 000001B8 FSTRG |
| 000001B6 FSTRG_PU | 00000042 FS_POSAB | 00000043 FS_POSRE | 00000042 GET | 00000088 GET_BYTE |
| 00000086 GET_FP | 000000B2 GET_INT | 000000BC GET_LET | 00000046 GET_LOOP | 000001DC GET_OUT |
| 00000001 IQ_FBYTE | 00000003 IQ_FSTRG | 00000005 IQ_SBYTE | 00000007 IQ_SSTRG | 00000190 POS_D0 |
| 00000186 POS_MANT | 00000178 POS_NORM | 0000000E PROC_DEF | 000000DA PUT | 0000010C PUT_FILE |
| 000000F6 PUT_FP | 00000102 PUT_INT | 000000DE PUT_LOOP | 000001AA PUT_ON_A | 0000011C RI_EXEC |
| 00000008 RI_FLOAT | 0000013A SET_POS | 000001BC SSTRG | 000001BE TRAP4_3 | 00000212 TYPE_BP |
| 000001EE TYPE_SET | 000001FA TYPE_VAR | | | |

Errori: 0000
 Byte liberi: 4A88
 Codice (byte): 0025A

Assembler terminato

Parte Quarta

L'editor/Assembler

L'editor è semplice da usare, eppure la sua potenza consente di editare velocemente e in modo efficiente i programmi sorgente in Assembler. È importante usare questo editor e non, piuttosto, il word processor Quill perché quest'ultimo non produce (in modo semplice) file di testo ASCII (senza caratteri di controllo). I puri file di testo sono gli unici che possono essere elaborati dall'Assembler.

L'editor è stato progettato appositamente per consentire la creazione di programmi sorgente (sotto forma di testo). Con esso è possibile editare fino a 400 linee di 72 caratteri contemporaneamente. Queste dimensioni del programma sorgente sono più che adeguate per almeno due motivi. In primo luogo, è difficile che la dimensione del programma, perlomeno agli inizi, superi quella indicata; in secondo luogo l'Assembler permette di includere file esterni, di libreria. Nel caso si debba sviluppare un grosso programma, è sufficiente creare un programma centrale che ordini all'Assembler di includere tutti quei file sorgente che contengono quei moduli esterni che andranno a formare l'intero programma.

13.1 Le window dell'editor

All'inizio, lo schermo dell'editor appare come mostrato in Figura 13.1. Ci sono quattro window che vengono descritte di seguito, partendo dall'alto verso il basso.

La prima è la window che rappresenta lo "stato"; in essa compaiono, permanentemente, le indicazioni della linea sulla quale è posizionato il cur-

sore, il totale delle linee utilizzate e la linea sulla quale è posizionato uno speciale simbolo che può venire utilizzato durante le operazioni di editing.

Al di sotto della window che mostra lo stato c'è l'indicatore di tabulazione. Il contenuto di questa particolare window non viene mai modificato. L'indicatore di tabulazione visualizza la posizione dei nove intervalli di tabulazione che sono disponibili; inoltre, mostra anche che la distanza tra un intervallo di tabulazione e l'altro è di 8 colonne. Ogni volta che viene premuto il tasto TAB, il cursore va a posizionarsi al primo intervallo di tabulazione che viene incontrato muovendosi verso destra dalla posizione corrente. L'importanza di questo sistema di tabulazione è data dal fatto che lo utilizza anche l'Assembler, che viene descritto nel prossimo capitolo, quando crea il file che contiene il listato.

La terza window che appare sullo schermo è l'effettiva "window di testo": questa è la vera window che vi permette di osservare una parte del testo che state editando. In qualsiasi momento, lo schermo mostra fino a 17 linee del programma in fase di editing. Il cursore lampeggiante all'interno della window vi permette di apportare modifiche al testo senza difficoltà. Quando si comincia a lavorare con l'editor e in memoria non è ancora stato inserito del testo, il cursore si trova posizionato nell'angolo in alto a sinistra della window di testo; questa posizione corrisponde alla linea 0 e colonna 1.

L'ultima window, che si trova in fondo allo schermo, viene usata per la visualizzazione di messaggi e richieste di input. Il contenuto di questa window può cambiare in certe occasioni: per esempio, quando richiedete le informazioni di "Aiuto", esse vengono visualizzate in questa window.

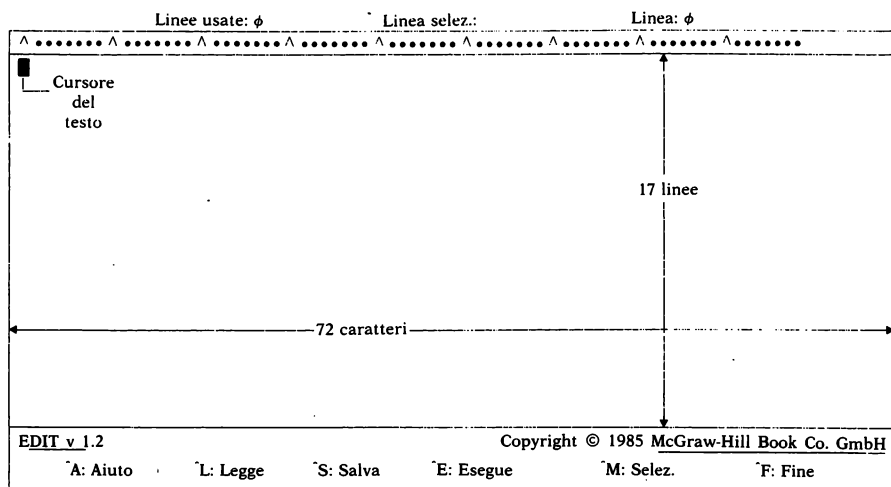


Figura 13.1 Schermo dell'editor

13.2 Modi dell'editor

Subito dopo l'inizializzazione, l'editor mette a disposizione il suo modo comandi/editing; questo modo permette di eseguire due tipi di operazioni principali. Una delle opzioni permette di eseguire uno dei sei comandi fondamentali che sono elencati nella window in fondo allo schermo: ognuno di questi comandi è rappresentato da un carattere di controllo; per lanciarlo, quindi, è necessario premere contemporaneamente il tasto CTRL e la lettera che indica il comando desiderato. L'altra opzione riguarda l'inserimento e la modifica del testo: a questo scopo è sufficiente introdurre l'opportuna sequenza di caratteri, comandi di controllo del cursore e comandi di cancellazione del testo.

A prima vista può sembrare che le opzioni da imparare ad usare siano tante; in pratica, invece, l'editor è molto semplice da usare. Nel caso ce ne fosse bisogno, poi, è possibile ottenere la visualizzazione di un messaggio (nell'apposita window in fondo allo schermo) che elenca tutti i comandi di controllo del cursore e di cancellazione del testo.

13.3 Messaggio di aiuto

L'editor dispone di tre gruppi di comandi; è possibile ottenere la visualizzazione dei comandi appartenenti a ciascun gruppo, assieme alla loro funzione. Questi sono i tre gruppi:

1. Comandi fondamentali (es., legge-file)
2. Comandi di controllo del cursore (es., sposta cursore di una linea verso il basso)
3. Comandi di cancellazione del testo (es., cancella un carattere a sinistra)

La window per i messaggi che si trova in fondo allo schermo, normalmente visualizza i sei comandi fondamentali. Uno di questi comandi è il comando di "Aiuto", che viene lanciato premendo ^A (abbreviazione di CTRL A). L'introduzione di questo carattere di controllo provoca la visualizzazione, nella window in fondo allo schermo, dei 12 comandi di controllo del cursore e dei 4 comandi di cancellazione del testo disponibili. Un secondo ^A richiama sullo schermo il messaggio originale; in questo modo, un unico comando viene utilizzato per passare da un elenco di comandi all'altro. È il caso di osservare che ciò significa che l'editor è totalmente autodocumentante per quanto riguarda la disponibilità dei comandi.

13.4 Introduzione del testo

Il testo viene scritto semplicemente battendo da tastiera i caratteri necessari. Ogni linea può contenere al massimo 72 caratteri. Per passare alla linea successiva basta premere ENTER, dopodiché si può procedere normalmente. In qualsiasi momento, tranne che all'ultima posizione della linea, l'uso del tasto TAB fa avanzare il cursore al primo intervallo di tabulazione disponibile sulla linea corrente.

La vera utilità dell'editor, ovviamente, sta nella possibilità di modificare il testo, sia nel caso in cui siano stati commessi degli errori, sia quando si vogliono togliere delle linee o aggiungerne delle altre. Per compiere queste operazioni è necessario poter spostare il cursore nell'opportuno punto del testo per poi inserire o cancellare caratteri secondo le necessità.

13.5 Movimento del cursore

Esistono, in totale, dodici comandi di controllo del cursore di tipo immediato; essi vengono inviati all'editor utilizzando uno alla volta i quattro tasti di controllo del cursore (su, giù, sinistra, destra) in uno di tre modi differenti:

- | | |
|------------|--|
| 1. Normale | I tasti vengono usati da soli |
| 2. SHIFT | I tasti vengono premuti contemporaneamente al tasto SHIFT |
| 3. ALT | I tasti vengono usati nel modo "alternativo" (cioè vengono premuti insieme al tasto ALT) |

Il cursore può essere spostato verso sinistra o destra lungo una data linea in diversi modi. Il cursore può anche essere mandato in su o in giù lungo il testo. Dopo aver inserito più di otto linee, noterete che il cursore rimane posizionato sulla linea centrale della window del testo mentre il testo gli scorre dietro, verso l'alto o verso il basso. Se, in un secondo tempo, posizionate il cursore entro le prime otto linee, vedrete, in questo caso, che il cursore si muove in su e in giù mentre il testo rimane immobile. Questa gestione del cursore è stata ideata appositamente in modo che voi possiate sempre vedere la linea indicata dal cursore nel suo contesto effettivo; in questo modo, le operazioni di editing del testo vengono rese molto più comode. Riportiamo di seguito la funzione di ciascun comando di controllo del cursore:

- | | |
|-------|---|
| 1. SU | — Il cursore si muove verso l'alto di una linea. Se il cursore si trova all'inizio del testo non avviene alcun cambiamento. Se la linea di destinazione è |
|-------|---|

- più breve della linea di partenza, il cursore viene posizionato alla fine della nuova linea.
2. GIÙ — Il cursore si muove verso il basso di una linea. Se il cursore si trova alla fine del testo non avviene alcun cambiamento. Se la linea di destinazione è più breve della linea di partenza, il cursore viene posizionato alla fine della nuova linea.
 3. SINISTRA — Il cursore si muove di un carattere verso sinistra. Se si trova all'inizio della linea non subisce alcuno spostamento.
 4. DESTRA — Il cursore si muove di un carattere verso destra. Se si trova alla fine della linea non subisce alcuno spostamento.
 5. SHIFT SU — Il cursore si muove verso l'alto di una pagina, che equivale a 16 linee. Osserviamo che, essendo la window del testo composta da 17 linee, c'è sempre una sovrapposizione di una linea. Questa particolarità consente di esaminare il testo più facilmente. Il cursore viene sempre posizionato all'inizio della nuova linea.
 6. SHIFT GIÙ — Il cursore si muove verso il basso di una pagina, che equivale a 16 linee. Osserviamo che, essendo la window del testo composta da 17 linee, c'è sempre una sovrapposizione di una linea. Questa particolarità consente di esaminare il testo più facilmente. Il cursore viene sempre posizionato all'inizio della nuova linea.
 7. SHIFT SINISTRA — Il cursore si muove di una parola verso sinistra. Se si trova all'inizio della linea non subisce alcuno spostamento.
 8. SHIFT DESTRA — Il cursore si muove di una parola verso destra, se si trova alla fine della linea non subisce alcuno spostamento.
 9. ALT SU — Il cursore viene posizionato all'inizio del testo.
 10. ALT GIÙ — Il cursore viene posizionato alla fine del testo.
 11. ALT SINISTRA — Il cursore viene posizionato all'inizio della linea corrente.
 12. ALT DESTRA — Il cursore viene posizionato alla fine della linea corrente.

Quando si inserisce del testo mentre il cursore è posizionato all'interno di una linea, i caratteri inseriti vengono collocati immediatamente prima del carattere evidenziato dal cursore. La parte di linea alla destra del cursore viene fatta scivolare verso destra.

13.6 Cancellazione del testo

Sono disponibili solo quattro comandi per la cancellazione immediata del testo i quali, inoltre, forniscono solo tre funzioni (in quanto due dei comandi eseguono la stessa operazione). I quattro comandi vengono inviati all'editor utilizzando i soliti tasti di controllo del cursore insieme al tasto CTRL. Le funzioni di questi comandi sono le seguenti:

1. CTRL SU — La linea dove è posizionato il cursore viene cancellata. Il comando non viene eseguito se esiste una linea contrassegnata in modo speciale.
2. CTRL GIÙ — La linea dove è posizionato il cursore viene cancellata. Il comando non viene eseguito se esiste una linea contrassegnata in modo speciale.
3. CTRL SINISTRA — Viene cancellato il carattere immediatamente a sinistra del cursore. La parte di linea dopo il cursore viene fatta scorrere verso sinistra di una posizione. Non viene eseguita alcuna operazione se il cursore si trova all'inizio della linea. Se il carattere da cancellare è uno spazio (unico o facente parte di una tabulazione) vengono cancellati tutti gli spazi che ci sono tra il cursore e l'ultimo carattere diverso da spazio alla sinistra del cursore.
4. CTRL DESTRA — Viene cancellato il carattere al quale è sovrapposto il cursore. La parte di linea alla destra del cursore viene fatta scorrere verso sinistra. Se il cursore si trova alla fine della linea non viene eseguita alcuna operazione. Se il carattere da cancellare è uno spazio (unico e facente parte di una tabulazione) vengono cancellati tutti gli spazi che si trovano tra il cursore e il primo carattere diverso da spazio alla destra del cursore.

I comandi appena descritti permettono di cancellare piccole parti di testo in prossimità del cursore. È spesso utile poter cancellare interi blocchi di linee di programma e ciò può essere fatto per mezzo di una delle opzioni del comando Esegue dell'editor.

13.7 Il tasto ENTER

Dopo avere inserito un buon numero di linee di programma nell'editor, muovendo anche il cursore qua e là, avrete senz'altro notato che il tasto ENTER esegue funzioni differenti in circostanze diverse. Queste funzioni possono essere definite come segue.

Se il cursore si trova proprio in fondo al testo, il tasto ENTER manda il cursore all'inizio di una linea appena creata, subito dopo la linea precedente. Ciò consente di inserire del testo partendo da zero, oppure di aggiungere linee a un testo già presente, in modo molto semplice. Se il cursore si trova all'inizio di una linea ma quella linea non è l'ultima del testo, premendo il tasto ENTER si ottiene la creazione di una nuova linea vuota, esattamente in quel punto, mentre il resto del testo viene spostato verso il basso. Ciò consente di inserire nuove linee all'interno di un testo preesistente in modo molto semplice. Se il cursore è in una posizione intermedia di una linea, il tasto ENTER lo manda semplicemente all'inizio della linea successiva.

13.8 Opzioni del comando Esegue

Per poter selezionare una opzione bisogna prima inviare all'editor il comando fondamentale ^E (CTRL E); le quattro opzioni disponibili vengono allora visualizzate nella window che occupa la parte bassa dello schermo. Per selezionare una delle quattro opzioni disponibili è sufficiente premere il tasto corrispondente alla prima lettera dell'opzione. Per esempio, per eseguire l'operazione Trova stringa bisogna premere il tasto con la lettera T. Le funzioni associate a ciascuna opzione sono le seguenti.

TROVA STRINGA

Un messaggio nella window in fondo allo schermo invita a specificare la stringa di testo da ricercare. Dopo aver introdotto la stringa che volete ricercare premete ENTER: l'editor comincerà a cercare la stringa, partendo dall'inizio della linea su cui è posizionato il cursore in quel momento. Se la stringa viene trovata, il cursore viene posizionato all'inizio della stringa stessa e la window del testo viene aggiornata per adeguarsi alla nuova posizione del cursore.

Se la stringa non è contenuta nel testo in cui viene effettuata la ricerca, il cursore non viene rimosso dalla posizione in cui si trova. Va tenuto presente che non fa differenza se le lettere che compongono la stringa sono minuscole o maiuscole: per esempio, "questo" è perfettamente equiva-

lente a "Questo" per quanto riguarda l'opzione di ricerca delle stringhe dell'editor.

Può succedere che, durante la ricerca, l'editor posizioni il cursore su una stringa che corrisponde a quella ricercata ma che non è quella particolare stringa che vi interessa; in questo caso, ricordatevi di spostare il cursore in basso di una linea prima di ripetere la ricerca, altrimenti verrebbe ritrovata ancora la stessa stringa.

CANCELLA BLOCCO

Questo è il comando che utilizza lo speciale contrassegno a cui avevamo accennato in precedenza; vediamo di che cosa si tratta: per mezzo del comando ^M (CTRL M) è possibile selezionare una qualunque linea del testo con un carattere speciale (un doppio segno di maggiore). Questo simbolo viene sempre introdotto e visualizzato all'inizio della linea corrente, qualsiasi sia la posizione del cursore all'interno della linea e il cursore viene spostato all'inizio della linea. Non è possibile marcare una linea completamente piena né è possibile marcare più di una linea.

Supponendo di aver marcato una linea, il comando Cancella blocco eliminerà tutte le linee comprese tra la linea marcata e la linea che contiene il cursore in quel momento (anche le linee di demarcazione del blocco così definite vengono cancellate). Se non esiste alcuna linea marcata, il tentativo di eseguire il comando genera un messaggio di errore. Dopo la comparsa del messaggio battete un tasto qualsiasi per continuare.

VA A LINEA

Questo comando vi permette di mandare il cursore ad una particolare linea del testo. Quando richiedete l'esecuzione di questa opzione vi viene richiesto il numero della linea alla quale volete andare: battete il numero seguito da ENTER e il cursore verrà spostato all'inizio della linea di testo corrispondente al numero da voi indicato; inoltre la window di testo verrà modificata di conseguenza.

Se indicate un numero di linea minore di zero, il cursore viene mandato all'inizio del testo. Al contrario, se viene indicato un numero maggiore del numero di linee presenti, il cursore viene mandato alla fine del testo.

INSERISCE FILE

Quando si crea il testo di un programma, è spesso utile poter fondere in esso alcune istruzioni contenute in un altro file. Con questo comando è

possibile includere un file di testo (prodotto con questo editor) nel testo sorgente col quale si sta lavorando nella posizione contrassegnata dal cursore al momento dell'esecuzione del comando. Viene visualizzato un messaggio che invita a indicare l'unità e il nome del file esterno, che deve essere memorizzato su una cartuccia inserita in un Microdrive. Dopo che le informazioni richieste sono state inserite, il file viene incluso nel testo corrente. Durante l'operazione di inclusione, nella window inferiore viene visualizzata una serie di simboli + che segnalano il tipo di operazione in corso. Senza questa indicazione si potrebbe pensare che qualcosa abbia bloccato l'editor mentre, in effetti, esso sta solo riordinando internamente il testo.

Per eseguire questa operazione senza danni è necessario fare attenzione: se il file indicato non esiste, l'editor visualizzerà un messaggio di errore fatale e abortirà l'esecuzione! Per questo motivo, è più prudente fare una copia del testo che si sta editando su Microdrive, prima di eseguire il comando di inclusione. Il comando provocherà la visualizzazione di un messaggio di errore se è presente una linea marcata (premete un tasto qualunque per continuare). Viene emesso un messaggio di errore anche quando l'editor esaurisce la memoria disponibile nel tentativo di includere il file esterno.

Quando si indica il nome del file da includere, l'estensione può essere specificata oppure no; nel secondo caso, viene utilizzata l'estensione standard `_ASM`.

13.9 Lettura di un file di testo

Un file sorgente creato in precedenza può essere caricato nell'editor prelevandolo da un Microdrive. Il comando fondamentale da usare a questo scopo è `-L` (CTRL L) e il file da caricare dovrà essere stato creato con questo editor. L'eventuale testo presente in memoria al momento della lettura del file esterno viene cancellato totalmente e il cursore viene posizionato all'inizio del nuovo testo. Prima che ciò avvenga, un messaggio avverte del pericolo di perdita del testo corrente e chiede conferma del comando di lettura file.

Quando si indica l'unità e il nome del file da includere, l'estensione può essere specificata oppure no. Nel secondo caso viene utilizzata l'estensione standard `_ASM`.

13.10 Salvataggio del testo corrente

Il contenuto del buffer dell'editor può essere salvato su Microdrive per mezzo del comando ~s (CTRL S). Il testo in memoria non viene cancellato, per cui è possibile utilizzare il comando Salva molte volte durante una stessa sessione di editing allo scopo di creare copie di sicurezza.

Quando si indica l'unità e il nome del file su cui salvare il testo, l'estensione può essere specificata oppure no. Nel secondo caso, viene utilizzata l'estensione standard __ASM. Non è possibile salvare un testo che contiene una linea marcata; se questa operazione viene tentata, viene emesso un messaggio di errore (premere un tasto qualunque per continuare dopo che è stato visualizzato il messaggio).

L'Assembler del 68000 descritto in questo libro è completo ed è scritto in codice macchina per un veloce assemblaggio dei programmi per il 68000. È stato progettato espressamente per funzionare con i programmi di gestione delle periferiche del QL e, pertanto, funzionerà con qualsiasi periferica collegata al QL (per esempio, floppy disk, dischi rigidi, interfacce per stampanti seriali e parallele e così via). Tra le sue caratteristiche vi sono:

1. Assemblaggio completo in 2 passate
2. Output indirizzato direttamente allo schermo, alla stampante o alla memoria di massa
3. Pseudo operazioni (es., ORG, COND)
4. Direttive Assembler (es., *TITOLO)
5. Interpretazione di semplici espressioni
6. Nomi di label globali e label locali
7. Mnemonici alternativi
8. Inclusione di file di libreria esterni

Questo capitolo descrive unicamente le possibilità offerte dall'Assembler e non illustra il set di istruzioni del 68000.

14.1 Funzionamento dell'Assembler

L'assemblatore è il cuore del sistema per la programmazione in Assembler. Esso riceve come input il contenuto di un file su Microdrive (o altro supporto di memoria di massa adatto) ed invia l'output allo schermo o alla stampante o alla memoria di massa. Nel seguito supporremo che le unità di memoria di massa utilizzate siano i Microdrive. La Figura 14.1 illustra il ciclo di sviluppo dei programmi. All'inizio, l'editor viene utilizzato per creare il programma sorgente; in seguito, il sorgente viene passato all'Assembler che produce vari file di output. Questi file, e in particolare il file oggetto (binario), possono essere manipolati in diversi modi. Per esempio, il file binario può essere lasciato così come è per venire utilizzato con il comando SuperBASIC LBYTES. In alternativa, il suo contenuto può essere caricato in memoria centrale per essere risalvato sotto forma di file eseguibile; a questo punto il programma può essere eseguito in modo indipendente per mezzo del comando SuperBASIC EXEC. Il manuale d'uso, incluso nel pacchetto Assembler, descrive dettagliatamente le opzioni dei comandi disponibili per l'Assembler e le interazioni dell'Assembler con l'Editor descritto nel capitolo precedente.

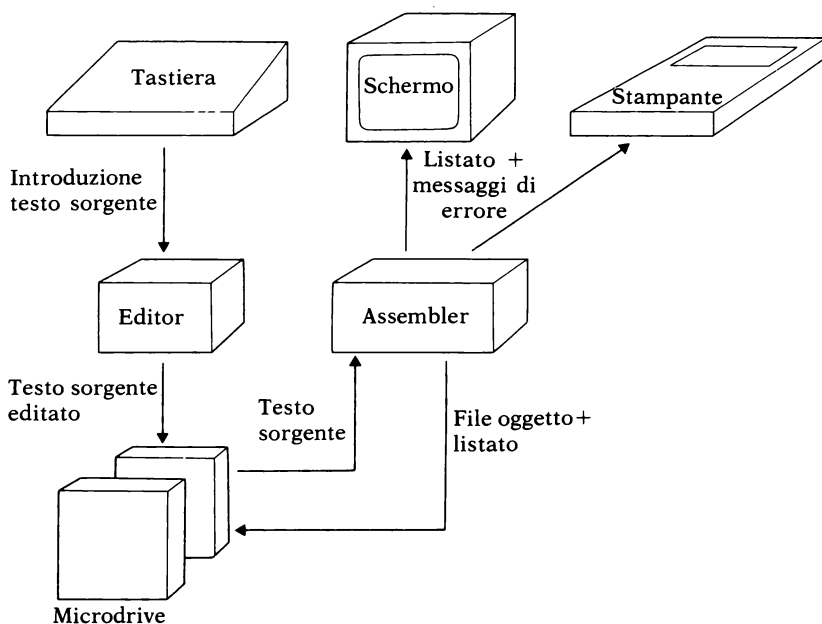


Figura 14.1 Sviluppo dei programmi Assembler

14.2 Sintassi delle linee Assembler

Le linee di programma sorgente trattate dall'Assembler sono linee contenenti una singola istruzione. In questo paragrafo spiegheremo la sintassi generale di queste linee, mentre una descrizione più dettagliata verrà data nel seguito nei paragrafi riguardanti gli argomenti specifici.

I programmi sorgente in Assembler consistono di una serie di linee di testo della lunghezza massima di 80 caratteri, create dall'editor descritto nel capitolo precedente. La sintassi di ciascuna linea è:

label: operatore argomento ;commento

Ciascuna di queste quattro parti — *label*, *operatore*, *argomento* o *commento* — può essere omessa, se necessario. È chiaro, ad esempio, che una linea vuota non ne conterrebbe alcuna e una linea di commento conterrebbe solo la quarta componente della linea. Ogni elemento è separato dagli altri da uno o più spazi o tab, dai due punti che seguono una label e dal punto e virgola che precede i commenti.

LABEL

Ogni nome di label deve iniziare con una lettera ma, per il resto, può contenere qualunque combinazione di caratteri di sottolineatura, lettere e cifre. Le lettere maiuscole e le corrispondenti minuscole sono del tutto equivalenti e tutto ciò che ha importanza per l'Assembler viene convertito in maiuscolo internamente. È anche possibile definire delle label temporanee (vedi paragrafo 14.4).

OPERATORI E ARGOMENTI

Gli operatori sono costituiti da mnemonici del 68000 (es., ADDX, ROR), pseudo operatori Assembler (es., DEFB, COND) o direttive Assembler (es., *INCLUSIONE). Il formato dei parametri dell'argomento dipende dall'operatore che lo precede.

COMMENTI

A qualunque linea può essere aggiunto un commento per facilitare la documentazione del programma sorgente. Ogni commento deve essere preceduto da un punto e virgola. Qualsiasi cosa posta dopo questo indicatore di commento viene ignorata dall'Assembler.

LO PSEUDO OPERATORE END

Il testo sorgente dei programmi in Assembler può essere terminato con lo pseudo operatore END. Se esso non viene usato, la normale fine del file viene considerata come fine del sorgente.

14.3 Simboli

I simboli, che equivalgono a costanti per la durata delle operazioni di assemblaggio, possono essere definiti sia all'interno del programma sorgente che dinamicamente, come costanti booleane (vero/falso), durante l'assemblaggio.

DEFINIZIONE NEL PROGRAMMA SORGENTE (EQU)

È possibile definire dei simboli alfanumerici per mezzo dello pseudo operatore Assembler EQU o anche, più semplicemente, per mezzo del segno di uguale; per esempio:

```
LETA EQU $41      ;'A'
LETB =   LETA+1    ;'B'
```

L'argomento che segue EQU può essere una qualsiasi espressione valida (come sarà definito in seguito). Se si tenta di definire due volte uno stesso simbolo, l'Assembler notificherà un errore "M" (definizione (M)ultipla), ma solo durante la prima passata. Quando si verifica un tale errore è meglio interrompere l'assemblaggio premendo il tasto esc dato che è facile che nel seguito vengano riscontrati molti errori correlati alla definizione multipla; ciò vale in particolare quando nel programma ci sono anche label temporanee (come normalmente avviene). Le lettere maiuscole e minuscole sono considerate equivalenti all'interno delle definizioni dei simboli; per esempio:

```
LETC EQU letb+1    ;'C'
letd EQU letc+1     ;'D'
LETE EQU LETD+1     ;'E'
```

Nei nomi dei simboli, sono considerati significativi solo i primi otto caratteri alfanumerici; i nomi stessi, inoltre, devono iniziare con una lettera (A..Z, a..z). Se quest'ultima regola viene violata, viene generato un messaggio di errore L ((L)abel illegale). Per esempio:


```
DELAYforTimer1 = 64  
Timer2Delay      = DELAYfor shl 2
```

DEFINIZIONE AL MOMENTO DELL'ASSEMBLAGGIO (QRY)

Se un simbolo viene definito per mezzo dello pseudo operatore QRY, ad esso può essere assegnato il valore 0 (falso), battendo N alla tastiera, oppure -1 (vero), battendo S. La stringa di richiesta di input viene visualizzata al momento dell'assemblaggio (solo durante la prima passata) e corrisponde all'argomento di QRY. Per esempio:

```
FLIST QRY Vuoi il listato completo
```

provocherà la visualizzazione di "Vuoi il listato completo?" al quale si dovrà rispondere con S o N. L'input da tastiera, in questo caso, è immediato (non è necessario usare ENTER) e l'Assembler produrrà, come eco, una S o una N a seconda di ciò che è stato inserito. Tenete presente che qualsiasi altra lettera inserita diversa da S equivarrà a N. La definizione per mezzo di QRY è particolarmente utile quando si usa la possibilità di assemblare su condizione, in quanto consente al programmatore di specificare valori di flag al momento di assemblare il programma, evitando di dover editare il testo sorgente.

14.4 Label

Due sono i tipi di label che possono essere usati: le label alfanumeriche hanno valore globale (lungo tutto il programma); le label temporanee, o locali, al contrario, hanno valore soltanto nel segmento di programma compreso tra due variabili standard, nel quale vengono definite.

LABEL STANDARD

Le normali label alfanumeriche sono dei simboli di tipo particolare. Vengono dichiarate facendole terminare con un due punti (:) e ad esse viene assegnato il valore del contatore delle locazioni corrispondente alla linea di programma in cui avviene la definizione. Le label devono obbedire alle stesse regole che valgono per gli altri simboli: (devono essere alfanumeriche, devono iniziare con una lettera e solo i primi otto caratteri sono significativi).

LABEL TEMPORANEE

Le label temporanee o locali hanno diversi attributi importanti. Ciascuna label occupa solo un terzo dello spazio della tabella dei simboli che viene utilizzato dagli altri simboli. Esse non compaiono nella tabella dei simboli, pertanto la tabella riporterà solo le locazioni più significative; inoltre, possono essere riutilizzate all'interno di blocchi di definizione differenti, riducendo così di molto la possibilità di definizioni multiple di label.

Le variabili locali hanno la forma 1% ... 255% e possono essere seguite opzionalmente dai due punti. Una label locale può esistere solo dopo che è stata dichiarata una label normale e la sua validità è limitata fino alla successiva label normale:

```
nlab1:  moveq    #0,d0
        moveq    #delay,d1
1%:     cmp.b     d0,d1
        beq.s     2%
        addq.b    #1,d0
        bra       1%
2%:     rts
;
nlab2:  bra       1%          ;1% non è più definita qui
2%:     nop
nlab3:
```

Durante la seconda passata, viene riportato un errore U se una label locale viene utilizzata all'esterno del campo di definizione nel quale la sua esistenza è ammessa.

14.5 Espressioni

L'Assembler consente l'uso di semplici espressioni nelle quali i vari operatori non hanno una priorità definita; le espressioni sono costituite da:

1. Simboli
2. Label normali o locali
3. Numeri decimali o esadecimali
4. Stringhe di un solo carattere
(L'uso del carattere ^, vedi paragrafo 14.6, non è necessario né permesso)
5. Gli operatori:
 - + più unario - somma
 - meno unario - sottrazione

| | |
|-----|--------------------------------------|
| * | moltiplicazione a 16 bit senza segno |
| / | divisione a 16 bit senza segno |
| SHR | shift a destra (di n posizioni) |
| SHL | shift a sinistra (di n posizioni) |
| OR | OR logico |
| AND | AND logico |
| NOT | complemento a uno |

NUMERI

I valori numerici possono essere definiti sia in decimale che in esadecimale. I numeri esadecimali devono essere preceduti dal simbolo & o dal simbolo \$; per esempio:

```
defb 12,45,&3A
defw $E2,$3AB0
```

Se la prima cifra che segue & o \$ non è compresa tra i simboli esadecimali ammessi viene generato un errore N (formato di numero) o un errore S (sintattico).

SEMPLICI ESPRESSIONI

Per semplice espressione priva di gerarchia degli operatori si intende, in questo caso, una espressione avente la seguente sintassi generale:

$+/-$ *operando* [*operatore operando*]

Il primo *operando* può essere preceduto da un più o un meno unari; è poi possibile far seguire nell'espressione altre coppie *operatore operando*. La valutazione dell'espressione avviene strettamente da sinistra verso destra. L'operatore NOT è di tipo particolare in quanto può essere applicato ad un solo operando (operatore di tipo strettamente unario) e questo operando deve essere un simbolo o una label normale. Se l'Assembler non riesce a interpretare l'espressione nel suo contesto, viene generato un errore E (espressione illegale); nella maggioranza dei casi esso viene immediatamente seguito anche da un errore S (sintattico). Alcuni esempi corretti sono:

```
vero      = -1
falso     = not vero
giorni    = 5
```

```
;
prog:   moveq   #vero and &FF,d0
        moveq   #nome and 255,d2
        moveq   #nome shr 8,d3
        moveq   #'A',d0
        moveq   #'z'+1,d0
;
        moveq   #'^',d0           ;up-arrow (vedi 14.6)
        moveq   #'^',d0           ;equivalente:
        moveq   #'A'+$80,d0       ;usa l'offset breve
;
        moveq   #nome/256+1,d2
        moveq   #giorni*24,d3
;
1%:     defb     0                  ;memoria dati
;
        move.w   menu,a0
;
menu:    defb     0,0
;
mask     =   vero shl 8+1
mask2    =   mask or $2020
```

I risultati delle espressioni possono essere valori a 8, 16 o 32 bit, a seconda del contesto dell'espressione. Viene generato un errore O (overflow) o R (range) se si verifica una incongruenza di contesto nella operazione di assegnamento (per esempio, se un valore a 16 bit viene usato dove è richiesto un valore a 8 bit). In questi casi, alcuni Assembler assegnano semplicemente il byte meno significativo, complicando enormemente le operazioni di debugging quando ci si rende conto che il programma non funziona nel modo desiderato. In caso di assemblaggio su condizione l'espressione che esprime la condizione viene valutata come vera se il bit più significativo del risultato è uguale a 1 (es., risultato = -1) o falsa se questo bit è uguale a 0 (es., risultato = 0).

14.6 Definizione dei dati

È possibile definire i dati utilizzando i seguenti pseudo operatori Assembler:

| | |
|------|----------------------------------|
| DEFB | Definisce byte/carattere (8 bit) |
| DEFW | Definisce word (16 bit) |
| DEFL | Definisce long word (32 bit) |

In alternativa, lo spazio per i dati in memoria può essere anche allocato (e inizializzato a zero) per mezzo dello pseudo operatore:

DEFS Definisce memoria (*n* byte)

I quattro pseudo operatori elencati permettono di definire qualsiasi combinazione di dati statici in memoria e possono essere utilizzati nei modi descritti di seguito.

DEFB

Questo pseudo operatore viene utilizzato per definire valori di tipo byte e stringhe di caratteri. Ogni linea di definizione può contenere qualunque combinazione di questi due tipi di dato:

```
defb 13,'Messaggio',13,0
defb 'ABCDEF'
defb 0,1,2,3,4,5,6,7,8,9
```

Ciascun elemento della linea di definizione è separato dal successivo da una virgola. Se il primo carattere di un elemento è un apice, i caratteri compresi tra di esso e l'apice successivo (escluso) vengono interpretati come stringa. Per quanto riguarda la definizione di stringhe, vale anche quanto segue:

1. Una freccia verso l'alto (^) seguita da un apice può essere utilizzata per inserire apici all'interno di stringhe: `defb '^'''`
2. Una freccia verso l'alto ^ seguita da un'altra freccia verso l'alto consente di inserire ^ all'interno di stringhe: `defb '^'^'`
3. Una freccia verso l'alto ^ seguita da un altro carattere consente di porre a 1 il bit più significativo di quel carattere: `defb '^A'`

Questi casi particolari possono essere inseriti liberamente all'interno della definizione di una stringa:

```
defb 'A^BC'
defb '^'su''      ;'su' (racchiuso tra apici)
defb 'A^2'
```

DEFW E DEFL

Questi pseudo operatori definiscono valori numerici che occupano 16 bit (nel caso di DEFW) o 32 bit (nel caso di DEFL) anche se potrebbero essere rappresentati in un byte.

```
defw 34,$56  
defl 900,$4B330,2
```

Non è possibile definire stringhe (secondo la definizione data sotto DEFB) utilizzando questi pseudo operatori. Ciascun elemento della linea di definizione deve essere separato dal successivo da una virgola.

DEFS

Se è necessario allocare una certa quantità di memoria ma il valore iniziale di quest'area non è noto in partenza (per esempio, spazio di heap), si può usare questo pseudo operatore. Esso deve essere seguito da un singolo argomento che specifichi il numero di byte da riservare; l'Assembler inizializzerà a zero l'intera area.

14.7 Determinazione dell'origine

L'indirizzo di memoria dal quale deve partire il codice macchina viene definito dalla pseudo operazione ORG:

```
ORG $2A000
```

Ci possono essere più comandi ORG all'interno di un programma ma è illegale definire una origine a un indirizzo più basso di quello a cui è giunto l'Assembler in quel punto. L'espressione passata come argomento a ORG può contenere label o simboli dichiarati in precedenza; per esempio, è possibile far partire un certo segmento di codice, all'interno di un programma, dall'indirizzo della prima locazione di una nuova pagina:

```
attuale:  
;  
    ORG attuale+256 and $FFFFFF00  
;  
nuovo:
```

Quando si scrivono programmi eseguibili in modo indipendente o estensioni al SuperBASIC, di solito si omette del tutto il comando ORG: la base del codice sarà posta, in questo caso, all'indirizzo zero.

Attenzione: le label e i simboli utilizzati nelle espressioni di ORG devono essere definiti in precedenza. In caso contrario, durante le passate 1 e 2

esisteranno origini diverse e l'assemblaggio del codice non sarà eseguito correttamente. Lo spazio compreso tra la fine di un tratto di codice e una nuova origine viene posto a zero dall'Assembler.

14.8 Assemblaggio su condizione

È possibile assemblare blocchi di codice su condizione utilizzando gli pseudo operatori COND, ELSE e ENDC. L'operatore COND richiede come argomento una espressione; se il bit più significativo del risultato è uguale a 1, il risultato stesso viene considerato vero e il segmento di codice che segue viene assemblato.

L'assemblaggio (o il non assemblaggio) del codice prosegue finché non viene incontrato un operatore ELSE o ENDC. Se viene trovato ELSE, la condizione per l'assemblaggio viene invertita e si procede nel nuovo modo fino al successivo ENDC. Ogni particolare livello di assemblaggio condizionale viene terminato al raggiungimento del corrispondente operatore ENDC.

L'assemblaggio su condizione può essere strutturato in più livelli annidati. Se la prima passata è terminata ma non sono stati ancora chiusi tutti i vari livelli di assemblaggio condizionale, viene generato un errore Assembler fatale e le operazioni di assemblaggio vengono interrotte (cioè la passata 2 non viene neanche cominciata). Un errore C viene provocato quando un operatore ELSE o ENDC viene incontrato prima del corrispondente operatore COND. Ecco, di seguito, alcuni esempi di annidamento:

| | |
|----|----------------------------|
| sì | = -1 |
| no | = not sì |
| | |
| 1. | cond sì |
| | subx d2,d0 ;assemblato |
| | else |
| | subx d0,d2 ;non assemblato |
| | endc |
| | |
| 2. | addx d1,d2 ;livello 0 |
| | cond no |
| | addx s2,d3 ;livello 1a |
| | cond vero |
| | addx d3,d4 ;livello 2a |
| | else |
| | subx d4,d3 ;livello 2b |

```
        endc
    else                                ;livello 1b
        subx    d3,d2
    endc
    nop                                ;di nuovo al livello 0
```

Ricordiamo che è possibile definire valori booleani (vero o falso) dei simboli per mezzo dell'operatore QRY; questo meccanismo si rivela molto utile nell'assemblaggio condizionale in quei casi nei quali deve essere generato dal codice che può essere leggermente differente (ad esempio, a seconda che il codice prodotto debba risiedere in ROM oppure no). Il programma sorgente, allora, non deve mai essere modificato: è sufficiente introdurre le risposte appropriate al momento dell'assemblaggio.

14.9 Direttive

L'Assembler è in grado di recepire diverse direttive, inviate ponendo un asterisco come primo carattere diverso da spazio di una linea di codice. Ecco le direttive previste:

1. ***Pagina**
2. ***Titolo** <stringa>
3. ***Listato** <si/no>
4. ***Numerazione** <si/no>
5. ***Inclusione** <indicazione_file>

Tutte le direttive possono essere abbreviate con il solo primo carattere (ad esempio, ***P** equivale a ***Pagina**).

***PAGINA E *TITOLO**

***Pagina** provoca un salto a nuova pagina nel file di listato e l'incremento di uno nel numero di pagina; se in precedenza era stata definita una intestazione, essa rimane invariata.

***Titolo** permette di definire un messaggio che viene poi utilizzato per documentare l'inizio delle pagine del file di listato. Questa direttiva provoca anche automaticamente un salto a nuova pagina (come con ***P**). La lunghezza massima dell'intestazione è di 35 caratteri e gli eventuali caratteri in sovrappiù vengono troncati.

Se nessuna di queste due direttive viene data prima che ci sia bisogno di un salto di pagina nel file di listato (per superare la piega tra le pagine dei tabulati), l'Assembler provvederà automaticamente a generare un sal-

to di pagina al momento opportuno (in genere dopo 56 linee di listato Assembler).

***LISTATO**

Con *Listato è possibile attivare e disattivare la generazione del listato. Se la direttiva è seguita dalla parola SÌ l'emissione del listato viene attivata. Se la parola che segue *Listato è NO, il listato cessa di essere emesso. In un caso, la direttiva *L SÌ non ha alcun effetto quando l'Assembler, avendo richiesto l'indicazione di un file di listato, ha ricevuto risposta nulla. Questa direttiva è particolarmente utile quando si vogliono listare condizionalmente parti di un grosso file sorgente. La tabella dei simboli viene comunque prodotta se è stato definito un file di listato; pertanto, un modo per ottenere proprio solo la tabella dei simboli come listato è di mettere a NO (su condizione) all'inizio del sorgente la direttiva *Listato:

```
FLST QRY È richiesto il listato
;
    cond not FLST
*L NO
    endc
;
<La tabella dei simboli viene prodotta in ogni caso!>
```

***NUMERAZIONE**

Questa direttiva ha la stessa sintassi di *Listato. Essa permette di abilitare e disabilitare la generazione e la stampa dei numeri di linea all'interno del file di listato. Se la direttiva non specifica il contrario, l'Assembler assume che i numeri di linea debbano essere generati.

***INCLUSIONE**

*Inclusione richiede come argomento i nomi di una unità e di un file. Il file indicato verrà incluso, in fase di assemblaggio, nel programma sorgente a partire dal punto dove è stata incontrata *I. Questa direttiva offre la possibilità di mantenere una libreria di routine in sorgente su una cartuccia per Microdrive e di includere nei programmi, di volta in volta, ciò di cui si ha bisogno e quando se ne ha bisogno. È permesso un solo livello di inclusione: non è possibile che un file che

viene incluso contenga, a sua volta, una direttiva di inclusione. Quando questo avviene, viene generato un errore I (inclusione di file) e l'Assembler riprende dalla linea successiva del file sorgente in corso di elaborazione in quel momento.

Se il file indicato non può essere aperto perché, ad esempio, l'indicazione è incompleta o sbagliata, viene emesso un messaggio di errore e le operazioni dell'Assembler vengono interrotte. Osserviamo che i file vanno dichiarati come se dovessero essere letti da Microdrive in SuperBASIC; inoltre, non ci sono restrizioni sull'uso delle estensioni, come invece avviene durante la dichiarazione dei file all'inizio dell'esecuzione dell'Assembler.

Quando si ha a che fare con grossi programmi sorgente, è pratica comune suddividerli in moduli più maneggevoli che vengono richiamati da un breve modulo principale per mezzo di direttive *Inclusione.

14.10 Mnemonici alternativi

L'Assembler dispone di un set di mnemonici alternativi che aiutano il programmatore a migliorare sia lo stile che la leggibilità. Primo mnemonico: quello dell'operazione di OR esclusivo. Sono due gli mnemonici più diffusi per questa istruzione e sono entrambi supportati dall'Assembler:

| Standard | Alternativo |
|----------|-------------|
| EOR | XOR |

Le altre alternative riguardano i codici di condizione: si fa spesso confusione, specialmente con quei processori che prevedono l'uso sia di aritmetica con segno che senza segno, su come vadano correttamente interpretati i codici di condizione *carry clear* e *carry set*; pertanto l'Assembler fornisce le seguenti possibilità:

| Standard | Alternativo |
|------------|-------------|
| BCC, BCS | BHS, BLO |
| DBCC, DBCS | DBHS, DBLO |
| SCC, SCS | SHS, SLO |

La parte di mnemonico HS sta per *Higher or Same* (più alto o lo stesso) e LO sta per *LOwer* (più basso). La differenza rispetto a GE (*Greater or Equal*: maggiore o uguale) e LT (*Less Than*: minore di) sta nel fatto che HS e LO si riferiscono a condizioni che si sono verificate in seguito a una operazione in aritmetica senza segno.

14.11 Messaggi di errore

L'Assembler esegue numerose verifiche durante la sua esecuzione ed emette tutta una serie di indicazioni di errore se il programma sorgente non corrisponde pienamente a quanto previsto dalle regole sintattiche dell'Assembler stesso. I codici di errore e i messaggi previsti sono i seguenti:

- N> Numero illegale: la rappresentazione di un numero esadecimale non è corretta.
- L> Label illegale: il formato di una label normale o locale non corrisponde alla sintassi prevista.
- S> Errore sintattico: comprende tutti quegli errori provocati da linee che contengono forme di sintassi illegale.
- M> Definizione multipla: si è tentato di ridefinire una label o un simbolo durante la prima passata.
- E> Espressione illegale: una espressione aritmetica o logica è illegale nel contesto in cui si trova.
- U> Identificatore non dichiarato: durante la seconda passata si fa riferimento a un simbolo o a una label la cui definizione non compare durante la prima passata.
- O> Overflow/salto fuori limite: un valore a 16 bit viene assegnato a una locazione a 8 bit; oppure: un salto relativo punta a una locazione fuori limite.
- C> Errore di assemblaggio su condizione: un operatore ELSE o ENDC precede il corrispondente COND.
- I> Errore nella inclusione di file: tentativo di eseguire una inclusione di file a più livelli.
- R> Superamento di range: all'interno di una particolare istruzione è stato specificato un valore al di fuori del range consentito.

MESSAGGI DI ERRORE DI TIPO GENERALE

Si possono verificare alcuni altri errori, generalmente di effetto fatale. Se l'Assembler non riesce ad aprire un file, oppure si verifica un errore nelle operazioni di I/O da Microdrive, viene emesso un messaggio appropriato e l'assemblaggio viene interrotto. Se durante la prima passata viene richiesto un assemblaggio su condizione scorretto, viene emesso un messaggio di errore e la seconda passata non viene nemmeno iniziata. In tutti questi casi il messaggio di errore indica la causa all'origine del problema.

14.12 Allineamento al margine di word (ALIGN)

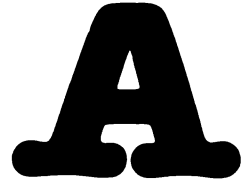
Il processore 68000 richiede sempre che una word o long word di dati comincino all'inizio di una word di memoria (cioè ad un indirizzo pari). Ciò implica che anche i codici operativi delle istruzioni devono essere allineati ai margini delle word di memoria. Quando si usano gli pseudo operatori Assembler DEFB e DEFS, può succedere che, al termine della linea di definizione, il contatore delle locazioni punti a un indirizzo dispari. Se la definizione dei dati è immediatamente seguita da una istruzione 68000, da una linea DEFW o da una linea DEFL, il codice oggetto che viene prodotto non corrisponderà al programma previsto; inoltre, il 68000 entrerà in uno stato di eccezione di tipo errore se si tenta di acquisire da memoria una istruzione o una word di dati ad un indirizzo dispari.

Per evitarvi di contare i byte delle definizioni per essere sicuri di aver definito un numero pari di byte (ottenendo sempre un risultato sbagliato, nonostante tutto), l'Assembler vi mette a disposizione lo pseudo operatore ALIGN. Questo operatore va posto dopo ogni definizione di byte che, per quello che segue, deve lasciare il contatore delle locazioni a un indirizzo pari. Per esempio:

```
;
dat1:  defb 6, 'NUMERI'
        align
dat2:  defw primo, ult, max, min
;
```

Se il contatore delle locazioni viene incrementato dall'Assembler per portarsi a un indirizzo pari, il byte lasciato inutilizzato viene posto a zero.

Sommario del set di istruzioni del 68000



A.1 Modi di indirizzamento

Il 68000, partendo da sei modi di indirizzamento base, mette a disposizione 14 modi effettivi differenti. Questi modi sono elencati in Figura A.1, insieme alla appropriata sintassi Assembler.

A.2 Codici di condizione

Ci sono tre istruzioni (Bcc, DBcc e Sc) che utilizzano un insieme di test di condizione. Ad ogni test è associato uno mnemonico di uno o due caratteri e lo mnemonico dell'intera istruzione è costituito dai nomi appena elencati dove a cc viene sostituito lo mnemonico del test (per esempio, BHI, BF, DBEQ, SNE e così via). Ciascun test dà come risultato vero o falso a seconda dello stato di certi flag nel registro CCR del 68000. Nella seguente tabella, gli mnemonici alternativi vengono elencati tra parentesi dopo lo mnemonico standard.

A.3 Set di istruzioni del 68000

La Figura A.2 mostra il set di istruzioni della MPU 68000 in ordine alfabetico. Viene indicato l'effetto che ogni istruzione ha sui flag del CCR; inoltre, viene indicato se l'istruzione è privilegiata (se, cioè, può essere

| Modo | Sintassi |
|---|------------------|
| Implicito a registro | SR, CCR, USP, PC |
| Immediato immediato | # n |
| immediato veloce | # b |
| Assoluto corto | a16 |
| lungo | a32 |
| Diretto a registro registro dati | Dn |
| registro indirizzi | An |
| Indiretto a registro registro indirizzi | (An) |
| postincremento | (An) + |
| predecremento | -(An) |
| registro indirizzi con offset | d16(An) |
| registro con indice e offset | d8(An,i) |
| Relativo al contatore di programma registro indirizzi con offset | d16(PC) |
| registro con indice e offset | d8(PC,i) |

Note:

| | |
|------------------------|------------------------------|
| b = 3, 4 o 8 bit | i = An o Dn |
| n = 8,16 o 32 bit | An = registro indirizzi |
| d8 = offset 8 bit | Dn = registro dati |
| d16 = offset 16 bit | PC = locazione corrente |
| a16 = indirizzo 16 bit | SR = registro di stato |
| a32 = indirizzo 32 bit | CCR = codici di condizione |
| | USP = puntatore stack utente |

Figura A.1 Modi di indirizzamento del 68000

eseguita solo quando il 68000 è in stato supervisore) e se di solito è richiesto un descrittore di dato (cioè .B, .W, .L o .S). Nell'elenco dei codici di condizione vengono utilizzate le seguenti convenzioni:

- x il flag viene alterato
- u il flag è indefinito

| Mnemonico | Test | Interpretazione |
|-----------|--------------------------------------|-------------------------------------|
| T | 1 | vero (sempre) |
| F | 0 | falso (sempre) |
| HI | $\text{not}(C).\text{not}(Z)$ | più alto (senza segno) |
| LS | $C+Z$ | più basso o lo stesso (senza segno) |
| CC (HS) | $\text{not}(C)$ | carry falso (senza segno) |
| CS (LO) | C | carry vero (senza segno) |
| NE | $\text{not}(Z)$ | diverso |
| EQ | Z | uguale |
| VC | $\text{not}(V)$ | overflow falso |
| VS | V | overflow vero |
| PL | $\text{not}(N)$ | più |
| MI | N | meno |
| GE | $\text{not}(N \text{ xor } V)$ | maggiore o uguale (con segno) |
| LT | $N \text{ xor } V$ | minore (con segno) |
| GT | $\text{not}(Z + (N \text{ xor } V))$ | maggiore |
| LE | $Z + (N \text{ xor } V)$ | minore o uguale |

Figura A.2 Sommario del set di istruzioni del 68000

- il flag non viene alterato
- 0 il flag viene posto a 0
- 1 il flag viene posto a 1

La colonna che indica le istruzioni privilegiate (P) utilizza queste convenzioni:

- n istruzione non privilegiata
- s istruzione privilegiata
- ? privilegiata sotto certe condizioni

Nei casi in cui appare ? nella colonna P, è necessario fare riferimento a un testo appropriato per determinare i casi particolari che si possono verificare.

La colonna del descrittore di dato (D) usa queste convenzioni:

- n nessun descrittore usato
- s descrittore usato (altrimenti viene sottinteso .W o salto lungo)
- ? parametri variabili a seconda che i descrittori di dato vengano usati oppure no

Nei casi in cui compare ? nella colonna D è necessario consultare un testo appropriato per determinare i casi particolari che si possono verificare.

| | | X | N | Z | V | C | P | D |
|-------|--|---|---|---|---|---|---|---|
| ABCD | Somma decimale con estensione di segno | x | u | x | u | x | n | n |
| ADD | Somma | x | x | x | x | x | n | s |
| | (Quando la destinazione è An) | — | — | — | — | — | n | s |
| ADDQ | Somma veloce | x | x | x | x | x | n | s |
| ADDX | Somma con estensione di segno | x | x | x | x | x | n | s |
| AND | AND logico | — | x | x | 0 | 0 | ? | s |
| ASL | Shift a sinistra aritmetico | x | x | x | x | x | n | ? |
| ASR | Shift a destra aritmetico | x | x | x | x | x | n | ? |
| Bcc | Salto condizionale | — | — | — | — | — | n | s |
| BCHG | Test e modifica di bit | — | — | x | — | — | n | n |
| BCLR | Test e azzeramento di bit | — | — | x | — | — | n | n |
| BRA | Salto incondizionale | — | — | — | — | — | n | s |
| BSET | Test e set di bit | — | — | x | — | — | n | n |
| BSR | Salto a subroutine | — | — | — | — | — | n | s |
| BTST | Test di bit | — | — | x | — | — | n | n |
| CHK | Controllo registro entro limiti | — | x | u | u | u | n | n |
| CLR | Azzeramento operando | — | 0 | 1 | 0 | 0 | n | s |
| CMP | Confronta | — | x | x | x | x | n | s |
| CMPM | Confronta con memoria | — | x | x | x | x | n | s |
| DBcc | Decrementa e salta su condizione | — | — | — | — | — | n | n |
| DBRA | Decrementa e salta sempre | — | — | — | — | — | n | n |
| DIVS | Divisione con segno | — | x | x | x | 0 | n | n |
| DIVU | Divisione senza segno | — | x | x | x | 0 | n | n |
| EOR | OR esclusivo | — | x | x | 0 | 0 | ? | s |
| EXG | Scambio di registri | — | — | — | — | — | n | n |
| EXT | Estensione di segno | — | x | x | 0 | 0 | n | s |
| JMP | Salto assoluto | — | — | — | — | — | n | n |
| JSR | Salto assoluto a subroutine | — | — | — | — | — | n | n |
| LEA | Carica indirizzo effettivo | — | — | — | — | — | n | n |
| LINK | Collegamento di stack | — | — | — | — | — | n | n |
| LSL | Shift a sinistra logico | x | x | x | 0 | x | n | ? |
| LSR | Shift a destra logico | x | x | x | 0 | x | n | ? |
| MOVE | Movimento dati | — | x | x | 0 | 0 | n | s |
| | (Quando dest. è An) | — | — | — | — | — | n | ? |
| | (Quando dest. è CCR) | x | x | x | x | x | n | n |
| | (Quando sorg. è SR) | — | — | — | — | — | n | n |
| | (Quando dest. è SR) | x | x | x | x | x | s | n |
| | (Quando viene usato USP) | — | — | — | — | — | y | n |
| MOVEM | Movimento di più registri | — | — | — | — | — | n | s |
| MOVEP | Movimento dati a periferica | — | — | — | — | — | n | s |
| MOVEQ | Movimento dati veloce | — | x | x | 0 | 0 | n | n |
| MULS | Moltiplicazione con segno | — | x | x | 0 | 0 | n | n |

| | | X | N | Z | V | C | P | D |
|-------|--|----------|----------|----------|----------|----------|----------|----------|
| MULU | Moltiplicazione senza segno | — | x | x | 0 | 0 | n | n |
| NBCD | Negazione decimale con estensione di segno | x | u | x | u | x | n | n |
| NEG | Negazione | x | x | x | x | x | n | s |
| NEGX | Negazione con estensione di segno | x | x | x | x | x | n | s |
| NOP | Nessuna operazione | — | — | — | — | — | n | n |
| NOT | Complemento a uno | — | x | x | 0 | 0 | n | s |
| OR | OR logico | — | x | x | 0 | 0 | ? | s |
| PEA | Indirizzo effettivo su stack | — | — | — | — | — | n | n |
| RESET | Reset dispositivi esterni | — | — | — | — | — | s | n |
| ROL | Rotazione a sinistra | — | x | x | 0 | x | n | ? |
| ROR | Rotazione a destra | — | x | x | 0 | x | n | ? |
| ROXL | Rotazione a sinistra attraverso extend | x | x | x | 0 | x | n | ? |
| ROXR | Rotazione a destra attraverso extend | x | x | x | 0 | x | n | ? |
| RTE | Ritorno da eccezione | x | x | x | x | x | s | n |
| RTR | Ritorno e ripristino di CCR | x | x | x | x | x | n | n |
| RTS | Ritorno da subroutine | — | — | — | — | — | n | n |
| SBCD | Sottrazione decimale con estensione di segno | x | u | x | u | x | n | n |
| Scc | Set su condizione | — | — | — | — | — | n | n |
| STOP | Stop | x | x | x | x | x | s | n |
| SUB | Sottrazione | x | x | x | x | x | n | s |
| | (Quando la destinazione è An) | — | — | — | — | — | n | s |
| SUBQ | Sottrazione veloce | x | x | x | x | x | n | s |
| SUBX | Sottrazione con estensione di segno | x | x | x | x | x | n | s |
| SWAP | Scambia le due metà di un registro dati | — | x | x | 0 | 0 | n | n |
| TAS | Test e set del bit 7 | — | x | x | 0 | 0 | n | n |
| TRAP | Trap | — | — | — | — | — | n | n |
| TRAPV | Trap per overflow | — | — | — | — | — | n | n |
| TST | Test | — | x | x | 0 | 0 | n | s |
| UNLK | Scollegamento | — | — | — | — | — | n | n |

Figura A.3 Sommario del set di istruzioni del 68000

Chiamate di sistema

B

Qui di seguito vengono elencate tutte le chiamate al QDOS tramite TRAP #*n* e le chiamate di utility vettorizzate, descritte in dettaglio nei capitoli da 4 a 7. I codici per le chiamate tramite TRAP da #1 a #3 vengono passati nel registro D0. Esistono numerose altre procedure e utility che possono servire quando, per esempio, si scrivono programmi di gestione delle periferiche (device driver) per I/O esterno su schede di espansione. Queste routine non sono state descritte in precedenza e non sono elencate nel seguito in quanto vanno al di là degli scopi di questo libro.

TRAP #1

| Mnemonico | Codice (esadec.) | Codice (decimale) | Descrizione |
|-----------|---------------------|----------------------|---|
| MT.INF | 0 | 0 | Informazioni di sistema |
| MT.CJOB | 1 | 1 | Crea un job in TPA |
| MT.JINF | 2 | 2 | Informazioni sui job |
| MT.RJOB | 4 | 4 | Rimuove da TPA job inattivo |
| MT.FRJOB | 5 | 5 | Rimuove d'autorità uno o più job da TPA |
| MT.FREE | 6 | 6 | Lunghezza spazio libero maggiore in TPA |
| MT.TRAPV | 7 | 7 | Stabilisce puntatore ai vettori per le trap di un job |
| MT.SUSJB | 8 | 8 | Sospende un job |
| MT.RELJB | 9 | 9 | Rilascia un job e lo reinserisce nella coda dello scheduler |

TRAP # 1

| Mnemonico | Codice (esadec.) | Codice (decimale) | Descrizione |
|------------------|-----------------------------|------------------------------|--|
| MT.ACTIV | A | 10 | Attiva un job |
| MT.PRIOR | B | 11 | Modifica priorità di un job |
| MT.ALRES | E | 14 | Alloca spazio in area delle procedure residenti |
| MT.RERES | F | 15 | Rilascia spazio all'area delle procedure residenti |
| MT.DMODE | 10 | 16 | Stabilisce/controlla modo display |
| MT.IPCOM | 11 | 17 | Comando IPC (scansione tastiera, suono) |
| MT.BAUD | 12 | 18 | Stabilisce baud rate |
| MT.RCLCK | 13 | 19 | Legge orologio |
| MT.SCLCK | 14 | 20 | Assegna ora all'orologio |
| MT.ACLCK | 15 | 21 | Rimette orologio |
| MT.ALCHP | 18 | 24 | Alloca area comune di heap |
| MT.RECHP | 19 | 25 | Rilascia area comune di heap |

TRAP # 2
Allocazione input/output

| Mnemonico | Codice (esadec.) | Codice (decimale) | Descrizione |
|------------------|-----------------------------|------------------------------|---|
| IO.OPEN | 1 | 1 | Apre un canale |
| IO.CLOSE | 2 | 2 | Chiude un canale |
| IO.FORMT | 3 | 3 | Formatta una memoria di massa a settori |
| IO.DELET | 4 | 4 | Cancella un file |

TRAP # 3
Operazioni di input/output

| Mnemonico | Codice (esadec.) | Codice (decimale) | Descrizione |
|------------------|-----------------------------|------------------------------|---------------------------------------|
| IO.PEND | 0 | 0 | Controlla presenza di input in attesa |
| IO.FBYTE | 1 | 1 | Acquisisce un byte |
| IO.FLINE | 2 | 2 | Acquisisce una linea (terminatore LF) |
| IO.FSTRG | 3 | 3 | Acquisisce una stringa di byte |

TRAP #3
Operazioni di input/output

| Mnemonico | Codice (esadec.) | Codice (decimale) | Descrizione |
|-----------|---------------------|----------------------|---|
| IO.EDLIN | 4 | 4 | Edita una linea (solo per console) |
| IO.SBYTE | 5 | 5 | Invia un byte |
| IO.SSTRG | 7 | 7 | Invia una stringa di byte |
| IO.EXTOP | 9 | 9 | Chiama operazione estesa |
| SD.PXENQ | A | 10 | Restituisce dimens. window e po- siz. cursore in coord. riferite ai pixel |
| SD.CHENQ | B | 11 | Restituisce dimens. window e po- siz. cursore in coord. riferite ai ca- ratteri |
| SD.BORDR | C | 12 | Stabilisce larghezza e colore bordo |
| SD.WDEF | D | 13 | Definisce window |
| SD.CURE | E | 14 | Abilita cursore |
| SD.CURS | F | 15 | Disabilita cursore |
| SD.POS | 10 | 16 | Sposta cursore in posizione assolu- ta (caratteri) |
| SD.TAB | 11 | 17 | Tabulazione |
| SD.NL | 12 | 18 | Salto a nuova linea |
| SD.PCOL | 13 | 19 | Cursore indietro |
| SD.NCOL | 14 | 20 | Cursore avanti |
| SD.PROW | 15 | 21 | Cursore su |
| SD.NROW | 16 | 22 | Cursore giù |
| SD.PIXP | 17 | 23 | Sposta cursore in posizione asso- luta (pixel) |
| SD.SCROL | 18 | 24 | Fa scorrere intera window (alto- basso) |
| SD.SCRTP | 19 | 25 | Fa scorrere parte alta di window (alto-basso) |
| SD.SCRBT | 1A | 26 | Fa scorrere parte bassa di window (alto-basso) |
| SD.PAN | 1B | 27 | Scorrimento intera window (late- rale) |
| SD.PANLN | 1E | 30 | Scorrimento linea del cursore (la- terale) |
| SD.PANRT | 1F | 31 | Scorrimento parte linea a destra del cursore |
| SD.CLEAR | 20 | 32 | Ripulisce intera window |
| SD.CL RTP | 21 | 33 | Ripulisce parte alta di window |
| SD.CLRBT | 22 | 34 | Ripulisce parte bassa di window |
| SD.CLRLN | 23 | 35 | Ripulisce linea del cursore |
| SD.CLRRT | 24 | 36 | Ripulisce linea a destra del cursore |
| SD.FONT | 25 | 37 | Setta/resetta font dei caratteri |

TRAP #3
Operazioni di input/output

| Mnemonico | Codice (esadec.) | Codice (decimale) | Descrizione |
|------------------|-----------------------------|------------------------------|--|
| SD.RECOL | 26 | 38 | Ricolora una window |
| SD.SETPA | 27 | 39 | Assegna colore paper |
| SD.SETST | 28 | 40 | Assegna colore strip |
| SD.SETIN | 29 | 41 | Assegna colore ink |
| SD.SETFL | 2A | 42 | Setta/resetta flash |
| SD.SETUL | 2B | 43 | Setta/resetta sottolineatura |
| SD.SETMD | 2C | 44 | Setta modo scrittura/disegno |
| SD.SETSZ | 2D | 45 | Assegna dimensione caratteri |
| SD.FILL | 2E | 46 | Colora rettangolo |
| SD.POINT | 30 | 48 | Disegna punto |
| SD.LINE | 31 | 49 | Disegna retta |
| SD.ARC | 32 | 50 | Disegna arco |
| SD.ELIPS | 33 | 51 | Disegna ellisse |
| SD.SCALE | 34 | 52 | Assegna valore scala |
| SD.FLOOD | 35 | 53 | Setta/resetta colorazione aree chiuse |
| SD.GCUR | 36 | 54 | Assegna posizione grafica del cursore |
| FS.CHECK | 40 | 64 | Controlla se c'è operazione su file in corso |
| FS.FLUSH | 41 | 65 | Svuota buffer di file |
| FS.POSAB | 42 | 66 | Assegna valore assoluto a punta- tore di file |
| FS.POSRE | 43 | 67 | Assegna valore relativo a punta- tore di file |
| FS.MDINF | 45 | 69 | Informazioni supporto memoria di massa |
| FS.HEADS | 46 | 70 | Assegna intestazione a file |
| FS.HEADR | 47 | 71 | Legge intestazione di file |
| FS.LOAD | 48 | 72 | Carica file |
| FS.SAVE | 49 | 73 | Salva file |

Utility vettorizzate

| Mnemonico | Vettore (esadec.) | Vettore (decimale) | Descrizione |
|-----------|----------------------|-----------------------|---|
| UT.WINDW | C4 | 196 | Predisporre window assegnandole un nome |
| UT.CON | C6 | 198 | Predisporre window di tipo console |
| UT.SCR | C8 | 200 | Predisporre window di tipo screen |
| UT.ERRO | CA | 202 | Invia messaggio di errore al canale 0 |
| UT.ERR | CC | 204 | Invia messaggio di errore al canale n |
| UT.MINT | CE | 206 | Converte intero in ASCII e lo invia al canale n |
| UT.MTEXT | D0 | 208 | Invia messaggio al canale n |
| UT.CSTR | E6 | 230 | Confronta due stringhe |
| UT.DATE | EC | 236 | Acquisisce data e ora |
| CN.DAY | EE | 238 | Acquisisce giorno della settimana |
| CN.FTOD | F0 | 240 | Converte da virgola mobile ad ASCII |
| CN.ITOD | F2 | 242 | Converte intero in ASCII |
| CN.ITOBB | F4 | 244 | Converte byte in ASCII |
| CN.ITOBW | F6 | 246 | Converte word in ASCII |
| CN.ITOBL | F8 | 248 | Converte long word in ASCII |
| CN.ITOHB | FA | 250 | Converte byte in ASCII esadec. |
| CN.ITOHW | FC | 252 | Converte word in ASCII esadec. |
| CN.ITOHL | FE | 254 | Converte long word in ASCII esadec. |
| CN.DTOF | 100 | 256 | Converte ASCII in virgola mobile |
| CN.DTOI | 102 | 258 | Converte ASCII in intero |
| CN.BTOIB | 104 | 260 | Converte ASCII in byte |
| CN.BTOIW | 106 | 262 | Converte ASCII in word |
| CN.BTOIL | 108 | 264 | Converte ASCII in long word |
| CN.HTOIB | 10A | 266 | Converte ASCII esadec. in byte |
| CN.HTOIW | 10C | 268 | Converte ASCII esadec. in word |
| CN.HTOIL | 10E | 270 | Converte ASCII esadec. in long word |
| RI.EXEC | 11C | 284 | Esegue una singola operazione aritmetica |
| RI.EXECB | 11E | 286 | Esegue una lista di operazioni aritmetiche |

Le quattro utility per la gestione dei Microdrive di cui si parla alla fine del capitolo 3 non sono comprese in questo elenco, in quanto esulano dallo scopo di questo libro. Il loro uso riguarda la lettura, scrittura e verifica di singoli settori con accesso diretto.

Codici di errore per il sistema QDOS

C

Il QDOS riconosce 21 condizioni di errore standard. Questi errori possono verificarsi (ed essere segnalati) sia nei programmi SuperBASIC che nei programmi in linguaggio Assembler. In quest'ultimo caso, il codice di errore viene restituito nel registro D0, come mostrato nei capitoli dal 4 al 7 (procedure di sistema del QL). Tutti i codici di errore sono long word (cioè 32 bit). L'elenco degli errori è il seguente:

| Mnemonico | Codice | Descrizione |
|-----------|--------|--|
| ERR.NC | -1 | Operazione non completata |
| ERR.NJ | -2 | Job non valido |
| ERR.OM | -3 | Memoria esaurita |
| ERR.OR | -4 | Superamento dei limiti |
| ERR.BO | -5 | Overflow del buffer |
| ERR.NO | -6 | Canale non aperto |
| ERR.NF | -7 | File, unità, variabile o procedura non trovata |
| ERR.EX | -8 | File già esistente |
| ERR.IU | -9 | File (o unità) già in uso |
| ERR.EF | -10 | Fine del file |
| ERR.DF | -11 | Unità completa |
| ERR.BN | -12 | Nome di unità o procedura non corretto |
| ERR.TE | -13 | Errore di trasmissione |
| ERR.FF | -14 | Formattazione non riuscita |
| ERR.BP | -15 | Parametro non corretto |
| ERR.FE | -16 | Errore su file |
| ERR.XP | -17 | Errore in espressione |
| ERR.OV | -18 | Overflow aritmetico |

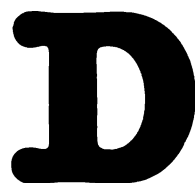
324 CODICI DI ERRORE PER IL SISTEMA QDOS

| Mnemonico | Codice | Descrizione |
|-----------|--------|---|
| ERR.NI | - 19 | Non (ancora) implementato |
| ERR.RO | - 20 | Sola lettura |
| ERR.BL | - 21 | Sintassi della linea scorretta (SuperBASIC) |

Come potete vedere, tutti i codici di errore sono costituiti da piccoli (in valore assoluto) numeri negativi. Ciò permette di distinguerli facilmente dai puntatori a particolari messaggi di errore per routine di gestione delle periferiche: questi ultimi, infatti, vengono passati come (puntatore__al__messaggio - \$8000).

Appendice

Guida rapida ai comandi editor/Assembler



L'editor e l'Assembler sono stati descritti nei capitoli 13 e 14. Riportiamo qui una breve guida al loro uso.

D.1 Guida all'uso dell'editor

a) Comandi fondamentali:

- ^E** — Esegue comandi estesi:
 - C Cancella blocco (dal contrassegno al cursore compresi)
 - T Trova stringa di testo
 - I Include file esterno (alla posizione del cursore)
 - V Va a una data linea del testo
- ^F** — Fine: ritorna al SuperBASIC
- ^A** — Aiuto: riassume i comandi di controllo del cursore e di cancellazione del testo
- ^L** — Legge un file da Microdrive
- ^M** — Seleziona la linea dove è posizionato il cursore
- ^S** — Salva il buffer dell'editor su Microdrive

b) Comandi di controllo del cursore:

| Tasto | Normale | SHIFT | ALT |
|--------------|----------------|--------------|--------------------|
| Su | Linea | Pagina | Inizio del testo |
| Giù | Linea | Pagina | Fine del testo |
| Sinistra | Carattere | Parola | Inizio della linea |
| Destra | Carattere | Parola | Fine della linea |

c) Comandi di cancellazione del testo:

| Tasto | CTRL |
|--------------|-------------------------------------|
| Su/giù | Cancella linea corrente |
| Sinistra | Cancella carattere/spazi a sinistra |
| Destra | Cancella carattere/spazi a destra |

D.2 Guida all'uso dell'Assembler

a) Commenti:

Devono essere preceduti da un punto e virgola

b) Label:

Devono terminare con i due punti

c) Direttive:

- | | |
|----------------------------|---|
| i) *PAGINA | Forza un salto a nuova pagina |
| ii) *TITOLO | Predisporre intestazione e salto a nuova pagina |
| iii) *LISTATO < sì/no > | Abilita o disabilita la produzione del listato |
| iv) *NUMERAZIONE < sì/no > | Abilita o disabilita la numerazione delle linee del listato |
| v) *INCLUSIONE < file > | Include file sorgente esterno |

d) Pseudo operatori:

- | | |
|-------------------------------------|--|
| i) EQU (=) | Uguaglianza definita staticamente |
| ii) ORY | Uguaglianza definita dinamicamente |
| iii) ORG | Stabilisce valore del contatore di programma |
| iv) ALIGN | Allineamento: dati tipo word e long word partono da indirizzo pari |
| v) COND < espress > ELSE ENDC | Assemblaggio su condizione |

e) Operatori nelle espressioni:

- | | | | |
|---|--------------------------|-----|-------------------|
| + | Somma | SHR | Shift a destra |
| - | Sottrazione | SHL | Shift a sinistra |
| * | Moltiplicazione (16 bit) | OR | OR logico |
| / | Divisione (16 bit) | AND | AND logico |
| | | NOT | Complemento a uno |

Indice analitico

Area di memoria di lavoro comune
(heap) 69

Area procedure residenti 67

Area programmi transienti 68

Assembler

argomenti 297

assemblaggio su condizione 305

commenti 297

definizione dati 302

definizione origine 304

direttive 306

Pagina 306

Titolo 306

Inclusione 307

Listato 307

Numerazione 307

espressioni 300

label 297, 299

mnemonici alternativi 308

notifica di errore 309

numeri 301

operatori 297

pseudo ALIGN 310

pseudo COND 305

pseudo DEFB 303

pseudo DEFL 303

pseudo DEFS 303, 304

pseudo DEFW 303

pseudo ELSE 305

pseudo END 298

pseudo ENDC 305

pseudo EQU 298

pseudo ORG 304

pseudo QRY 299

simboli 298

sintassi linea 297

uso 295

Attivazione dei job 75

Bootstrap 70

CALL 214

Canale

SuperBASIC 222

area di memoria 69

blocco di definizione 111

ID 17, 111

schermo 112

Codici di condizione (vedi Processore)

Colore 114

Editor

aiuto, come ottenerlo 287

comandi Esegue 291

cancella blocco 292

inserisce file 292

- trova stringa 291
- va a linea 292
- movimenti cursore 288
- testo
 - cancellazione 290
 - caricamento 293
 - introduzione 288
 - salvataggio 294
- window 285
- EXEC 214
- File BOOT 255, 273
- I/O semplificato, procedure 179, 207
- Inattivazione dei job 75
- Interrupt 24
- LBYTES 215
- Mappa della memoria
 - QL 65
 - video 244
- Mnemonici alternativi 32
- Modi di indirizzamento (vedi Processore)
- Modo
 - supervisore 21, 25
 - utente 21, 25
- Multitasking 75
- Nomi
 - tabella/lista dei (vedi SuperBASIC)
- Parametri, passaggio dei (vedi SuperBASIC)
- PEEK 215
- POKE 215
- Processore 68000
 - codici di condizione 31
 - trattamento dei flag 32
 - descrizione 21-26
 - istruzioni 33-62
 - operandi delle 30
 - memoria 24
 - mnemonici alternativi 32
 - modi di indirizzamento 28
 - modo supervisore 21, 25
 - modo utente 21, 25
 - registri 22
- Programmi eseguibili 233-242, 243-253
- Puntatore allo stack
 - supervisore (SSP) 21, 23
 - utente (USP) 21, 23
- QDOS 65
 - area buffer 70
 - bootstrap 70
 - I/O trap di allocazione 101-108
 - I/O trap per operazioni 109-178
 - multitasking 75
 - ridirezione dell'I/O 101
 - routine 71
 - scheduling 76
 - tabelle/variabili 67-69
 - trap per la gestione delle risorse 75-100
- Registro di stato
 - bit di modo 21
 - descrizione 23
- RESPR 216
- SBYTES 216
- SEXEC 217
- Sospensione dei job 75
- Stack aritmetico 222, 229
- SuperBASIC
 - area buffer 218
 - area dei valori delle variabili 221
 - area di memoria 69
 - area di programma 218
 - collegamento al 213
 - lista dei nomi 221
 - procedure in codice macchina 224
 - esempi 256-267, 273-282
 - passaggio parametri 227
 - restituzione di stringhe 228
 - restituzione parametri 228
 - restituzione valori funz. 228
 - stack aritmetico 222, 229
 - tabella area di lavoro 219
 - tabella dei canali 222
 - tabella dei nomi 220
 - nuovi elementi 224
- TRAP #4 229
- utility 72, 179

Timeout 76, 109

TRAP #0 71

TRAP #1 75

TRAP #2 101

TRAP #3 109

TRAP #4 229

Trattamento delle eccezioni

cause 26

violazione di privilegio 21

Virgola mobile, routine 208-211

La McGraw-Hill pubblica in tutto il mondo centinaia di libri di informatica per lo studio, la professione e il tempo libero. La produzione in lingua italiana comprende:

- 88 7700 001 5 J. Heilborn e R. Talbott, *Guida al Commodore 64*
- 88 7700 002 3 C.A. Street, *La gestione delle informazioni con lo ZX Spectrum*
- 88 7700 003 1 T. Woods, *L'Assembler per lo ZX Spectrum*
- 88 7700 004 X R. Jeffries, G. Fisher e B. Sawyer, *Divertirsi giocando con il Commodore 64*
- 88 7700 005 8 G. Bishop, *Progetti hardware con lo ZX Spectrum*
- 88 7700 006 6 H. Mullish e D. Kruger, *Il BASIC Applesoft*
- 88 7700 007 4 N. Williams, *Progettazione di giochi d'avventura con lo ZX Spectrum*
- 88 7700 008 2 H. Peckham, *Il BASIC e il PC-IBM in pratica*
- 88 7700 009 0 H. Peckham, *Il BASIC e il Commodore 64 in pratica*
- 88 7700 010 4 S. Nicholls, *Tecniche avanzate in Assembler con lo ZX Spectrum*
- 88 7700 011 2 K. Skier, *L'Assembler per il Commodore 64 e il VIC-20*
- 88 7700 012 0 S. Kamins e M. Waite, *Programmazione umanizzata in Applesoft*
- 88 7700 013 9 A. Pennell, *Guida allo ZX Microdrive e all'Interface 1*
- 88 7700 015 5 P. Cohen, *Grafica e animazione con gli Apple II*
- 88 7700 016 3 C. Duff, *Guida al Macintosh*
- 88 7700 017 1 G. Kane, *Il manuale MC68000*
- 88 7700 018 X P. Hoffman e T. Nicoloff, *Il manuale MS-DOS*
- 88 7700 020 1 S. Nicholls, *Grafica avanzata con lo ZX Spectrum*
- 88 7700 021 X L.J. Graham e T. Field, *Guida al PC-IBM*
- 88 7700 022 8 T. Field, *Come usare MacWrite e MacPaint*
- 88 7700 024 4 H. Peckham, *Il BASIC e gli Apple II in pratica*
- 88 7700 025 2 C. Morgan e M. Waite, *Il manuale 8086/8088*
- 88 7700 027 9 G. Mainis, *Il manuale ProDOS*
- 88 7700 028 7 J. Jones, *Il SuperBASIC del QL*
- 88 7700 029 5 C. Opie, *L'Assembler per il QL*
- 88 7700 030 9 W. Ettlin e G. Solberg, *Il BASIC Microsoft*
- 88 7700 034 1 P. Hoffman, *Il manuale MSX*
- 88 7700 035 X R. Person, *Le meraviglie dell'animazione con il PC-IBM*

- 88 7700 036 8 W. Ettlin, *Il Multiplan per il Macintosh*
- 88 7700 601 3 S. Harrington, *Computer Graphics - Corso di programmazione*
- 88 7700 602 1 O. Lecarme e J.L. Nebut, *Pascal - Guida per programmatori*

In preparazione

- 88 7700 019 8 E.M. Baras, *Come usare il Symphony*
- 88 7700 026 0 W. Ettlin, *Come usare il Multiplan*
- 88 7700 031 7 D.L. Toppen, *Il Forth in pratica*
- 88 7700 032 5 R. Person, *Le meraviglie dell'animazione con gli Apple II*
- 88 7700 037 6 D. Kruglinski, *Introduzione al Framework*
- 88 7700 038 4 L. Barnes, *Introduzione al dBase II*
- 88 7700 040 6 L. Barnes, *Introduzione al dBase III*
- 88 7700 603 X M. McGilton e R. Morgan, *Il sistema operativo UNIX*

L'Assembler per il QL è una guida rigorosa ed esauriente per scrivere programmi in Assembler che sfruttano a fondo la potenza del Sinclair QL e per espandere le prestazioni del SuperBASIC. Dopo una rapida introduzione al set di istruzioni e ai modi di indirizzamento del microprocessore Motorola 68008, viene descritto il QDOS. Questo flessibile sistema operativo multitasking mette a disposizione del programmatore una raccolta completa di routine, risparmiando la codifica di complesse quanto indispensabili operazioni, come quelle per l'aritmetica in virgola mobile, la gestione dello schermo per la grafica e l'input/output dei dati. Nel libro vengono presentate numerose routine in Assembler, alcune delle quali possono essere eseguite in modo indipendente dal SuperBASIC, mentre altre consentono di espandere il linguaggio e possono formare la base di una personale libreria di programmi. La creazione di programmi in Assembler per un computer della classe del QL richiede l'uso di strumenti potenti: per questo scopo la McGraw-Hill ha creato il **QL Machine-Code Editor/Assembler**, un completo package che viene diffusamente descritto in questo libro e che è disponibile su cartuccia Microdrive.

