Assembler 68000

di Marco Pesce

Comincia da questo mese una serie di articoli dedicati al linguaggio macchina di Amiga, un breve corso per chi vuole sfruttare a fondo le risorse di questo potente computer. Il microprocessore lo avrete per lo meno sentito nominare, un bel 68000 con i suoi 16 bit. Ci occuperemo di tutti gli argomenti necessari al fine di possedere una più che sufficiente padronanza di questo linguaggio di programmazione, tedioso ma incredibilmente veloce. Cominceremo dalle basi, quindi il discorso è rivolto anche a chi non ha neppure un briciolo di conoscenza sull'argomento

La prima cosa che ci deve interessare è avere a portata di mano un programma di codifica del codice Assembler detto anche semplicemente Assembler (in italiano «assemblatore»), che ci permetterà di trasformare i nostri listati in codice binario direttamente eseguibile dal computer. Personalmente utilizzo da molto tempo il Macro Assembler dell'AmigaDOS e se non ne avete già uno per conto vostro, vi consiglio di procurarvi proprio questo, per evitare anche i più piccoli problemi di «incompatibilità» con i listati che verranno proposti in questa rubrica.

Cominciamo con una introduzione generale al linguaggio macchina. Un linguaggio, cosiddetto, «interpretato», come ad esempio il Basic, appartiene alla categoria dei linguaggi ad alto livello, vale a dire progettati appositamente per facilitare il compito di programmazione all'utente e quindi più vicini al modo di ragionare di quest'ultimo che a quello della macchina. Tali linguaggi necessitano di un processo di conversione, in quanto non sono direttamente «eseguibili» dal computer. Per il Basic abbiamo appunto «l'interprete Basic», che nell'Amiga, come noto, viene caricato in memoria da un dischetto allegato. Esso altro non è che un programma che trasforma linee di Basic in una sequenza di istruzioni direttamente eseguibile dal computer, che è per l'appunto il linguaggio macchina. Questo processo di conversione «ruba» una quantità molto elevata di tempo e il risultato di ciò è una relativamente lenta esecuzione del programma. Per aggirare questo problema si è costretti ad utilizzare direttamente il «linguaggio» del microprocessore, che si avvale di istruzioni molto semplici; tuttavia queste non permettono di eseguire operazioni complesse come, ad esempio, una «radice quadrata». Resta inteso che tutto è possibile (anche la radice quadrata), ma deve essere realizzato con le «semplici» istruzioni che abbiamo a disposizione, il che ci complica un po' la vita. Lo sforzo ovviamente è in larga misura ricompensato dalla incredibile velocità di esecuzione.

Piedinatura del microprocessore 68000

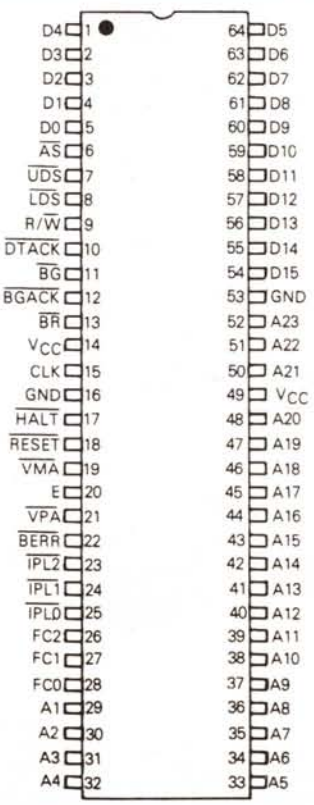

Figura 1

Per capire la logica del linguaggio macchina occorre una infarinatura del funzionamento del microprocessore ed è proprio questo il prossimo passo che voglio fare. Il microprocessore è il chip principale del computer, ovvero quello che comanda tutti i circuiti circostanti e che si occupa dell'esecuzione dei programmi. Esso possiede, tra l'altro, dei canali di comunicazione con la memoria (RAM e ROM) detti bus. Ne esistono due: il bus indirizzi e il bus dati. Il primo è quello che specifica quale delle numerose (numerosissime nel caso di Amiga) celle di memoria deve essere «letta» (o meglio «indirizzata»), mentre il secondo è quello che la «legge». Per poter scegliere tra i teorici 16 Mbyte «indirizzabili» occorre un bus indirizzi a 24 bit, che è proprio quello utilizzato dal 68000. I famosi «16 bit» si riferiscono all'ampiezza del bus dati, che ovviamente è di 16 bit (2 byte). Ciò significa che il 68000 preleva 2 byte alla volta per ogni indirizzamento fatto dal bus indirizzi. Da notare che il chip 68000 non possiede il piedino (o «pin») «A0», ovvero quello che dovrebbe indicare il bit «meno significativo» del bus indirizzi; conseguenza di ciò è il fatto che comunque tale microprocessore preleva dalla RAM sempre a salti di 16 bit, cosa che per chi è ancora a digiuno di nozioni in tal senso potrebbe sembrare inutile.

La differenza tra un microprocessore a 8 bit e uno a 16 bit sta nel fatto che il primo è meno potente del secondo (facile da intuirsi), per il fatto che il secondo manipola dati di dimensioni maggiori con più dimestichezza, grazie alla maggiore ampiezza del bus dati. La potenza si traduce soprattutto in maggiore velocità. In figura 1 potete vedere una rappresentazione dei «pin» del 68000; quelli con le sigle da A1 ad A23

sono i pin del bus indirizzi (come detto manca l'A0), mentre quelli con le sigle da D0 a D15 sono i pin del bus dati; gli altri sono pin di controllo vari (alimentazione, Clock, IRQ, ecc.).

Chiarito come avviene la comunicazione con la memoria (che ovviamente è quella che contiene dati e programmi) vediamo come opera con essa il microprocessore. Non appena viene avviato (all'accensione del computer), esso comincia ad operare prelevando da zone di ROM del sistema operativo (scritto ovviamente in linguaggio macchina) i primi byte. I primi 16 bit prelevati con il bus dati dalla cella (chiamiamola così) indicata dal bus indirizzi contengono la prima istruzione da eseguire.

Questo insieme di bit viene decodificato e stabilisce quale delle varie istruzioni i circuiti interni del 68000 dovranno apprestarsi ad eseguire. Normalmente una istruzione comprende più di 16 bit quindi magari, dopo aver incrementato il valore del bus indirizzi, vengono prelevate altre «celle» per completare le informazioni necessarie all'esecuzione di questa prima istruzione. Si prosegue quindi con le successive fino allo spegnimento del computer.

Per il suo funzionamento il microprocessore si avvale di numerosi «registri» interni, che sono celle di memoria RAM, appunto, interne ad esso. La maggior parte di questi è utilizzata soprattutto per operazioni di «tamponamento», ovvero per mantenere nel microprocessore dei dati che verrano utilizzati in seguito, senza doverli cercare sempre nella memoria esterna, con conseguente «perdita di tempo». Tra poco li vedremo in dettaglio.

La cosa che dobbiamo aver ben presente quando programmiamo in Assembler è che stiamo utilizzando direttamente la RAM, quindi non esistono variabili stringa, numeriche e via dicendo e i nostri dati li dobbiamo organizzare con cura e stabilire come e dove andranno depositati in RAM; grazie al Macro Assembler tuttavia ci sono delle semplificazioni che permettono ad esempio di trasformare un insieme di caratteri ASCII nei rispettivi byte da inserire nelle celle di RAM. Un programma in linguaggio macchina quindi risiede in una ben determinata zona RAM, tuttavia con Amiga tale affermazione è da chiarire meglio, in quanto, trattandosi di un sistema multitasking, un programma deve essere eseguibile in qualunque zona della memoria, di conseguenza i programmi in codice macchina devono essere completamente «rilocabili», (posizionabili ovunque), ma anche a questo ingrato compito pensa il Macro Assembler. Tra gli altri vantaggi detto «pacchetto» permette di utilizzare «label», ovvero di etichettare con dei nomi scelti dall'utente delle subroutine o delle locazioni RAM e grazie a ciò, come vedremo, la logica di programmazione si avvicina a quella di linguaggi ben più evoluti.

```
Esempio di funzionamento dello stack

STATO INIZIALE:
          locazioni ram
               ---
               998
               999
              1000
              1001<-- stack pointer (registro A7)
              1002
              1003
              1004
               ---
IMMISSIONE DATI:
          locazioni ram
               ---
               998
               999
              1000<-- stack pointer (registro A7)
              1001
              1002
              1003
              1004
               ---
PRELIEVO DATI:
          locazioni ram
               ---
               998
               999
              1000
              1001<-- stack pointer (registro A7)
              1002
              1003
              1004
               ---
```

Figura 2

Riprendiamo il discorso sui registri interni. Otto di essi sono dedicati ai dati, e sono chiamati appunto «registri dati»; hanno una ampiezza di 32 bit (4 byte) e verrano utilizzati in prevalenza per memorizzare temporaneamente risultati di operazioni in corso (quindi dati). Per identificarli si usano le sigle D0, D1, D2, D3, D4, D5, D6, D7. Altri otto registri sono chiamati «registri indirizzi» (sempre a 32 bit) e servono prevalentemente per il già citato indirizzamento della memoria. Per la loro identificazione qui si usano le sigle da A0 ad A7. Il registro A7 è dedicato ad una particolare funzione, lo «stack pointer» ed ora vedremo di cosa si tratta. Innanzi tutto, essendo un registro indirizzi, la sua funzione è quella di indirizzare, o meglio di fare da puntatore, ad una zona di RAM, che sarà utilizzata per memorizzare particolari dati utili al microprocessore, ad esempio, per «ricordarsi» la locazione di memoria alla quale deve ritornare una volta terminata l'esecuzione di una subroutine. Inoltre esso può essere utile anche all'utente, per avere a disposizione una ulteriore locazione tampone (magari perché sono finiti i registri); apposite istruzioni permettono di memorizzare dati (e prelevarli) in quella struttura che abbiamo appena descritto e che è appunto «stack». Il suo funzionamento è molto semplice; quando viene inserito un dato nello stack il puntatore (il registro A7) viene decrementato, mentre una operazione opposta comporta il suo incremento; ciò significa che l'ultimo dato inserito nello stack sarà anche il primo ad essere prelevato in seguito. Ciò è molto importante in quanto, ad esempio, se vengono «chiamate» più subroutine in successione, il microprocessore dovrà ritornare prima a quella che ha eseguito la chiamata per ultima e poi alle altre, ovvero esattamente come i dati relativi verranno prelevati dallo stack. In figura 2 possiamo vedere uno schema che ne esemplifica il funzionamento.

Cominciamo a dare un'occhiata più approfondita alle caratteristiche del 68000. Esistono 56 istruzioni diverse e 14 sono i modi di indirizzamento della memoria. Ci sono fondamentalmente tre tipi di dati: ad 8, a 16 e a 32 bit, rispettivamente byte, parola e parola lunga (indicati simbolicamente nelle istruzioni con B, W ed L). Ciò vuol dire che possiamo scegliere se usufruire di un'ampiezza ridotta od estesa a seconda del nostro scopo. Ad esempio è inutile utilizzare una parola lunga per contenere una «variabile» che oscilla tra 0 e 10, per la quale è sufficiente un dato in formato byte.

Vediamo una prima istruzione, la MOVE, ovvero quella che permette di trasferire valori da una zona (intesa sia

come RAM che come registri) ad un'altra. Ad esempio l'istruzione

MOVE.L D1, D2

trasferisce il valore a 32 bit (parola lunga)

Figura 3

contenuto nel registro dati D1 al registro dati D2. L'operando «sorgente», ovvero quello che «dà» (in questo caso D1), si trova sempre a sinistra dell'istruzione, mentre l'operando destinazione, quello che «riceve», si trova sempre a destra (in questo caso D2). Per trasferire solo 8 bit (un byte) basta sostituire la L dopo il MOVE con una B. Per 16 bit (parola) occorre la lettera W. Per default si assume proprio la W. Una operazione in modo byte altererà solo i primi 8 bit del registro, mentre in modo parola (word) modificherà solo i primi 16. Essi si possono usare sia come sorgente che come destinazione (vedi esempio precedente).

I registri degli indirizzi non possono contenere byte e quando vi si trasferisce un dato in formato parola, detto dato viene modificato con una procedura chiamata estensione del segno, ovvero il bit più significativo (della parola) viene copiato nei restanti bit non utilizzati al fine di estendere a 32 bit il «segno» del valore contenuto (tecnica del complemento a due; tra breve vedremo come funziona). Da notare che solo i primi 24 bit sono effettivamente utilizzati per l'indirizzamento (24 bit = 16 Mbyte). L'indirizzamento di una locazione di memoria può essere scelto tra i tre formati, byte parola e parola lunga, ovvero si può scegliere se dobbiamo prelevare appunto un solo byte, una parola o una parola lunga. Per i formati maggiori l'indirizzo deve essere rispettivamente pari o multiplo di 4 (ad esempio 0, 2, 4, 6, ecc. per le parole e 0, 4, 8, 12 ecc, per le parole lunghe).

La tecnica del complemento a due sarà senz'altro nota ad alcuni di voi; essa si avvale di un particolare accorgimento per poter rappresentare numeri sia di segno positivo che di segno negativo. Sostanzialmente il bit più significativo (quello più a sinistra) rappresenta il «segno» (positivo se a 0 e negativo se a 1). Nel caso di un byte il segno è rappresentato dal bit 7 (vedi figura 3), ovvero da quello che in tale formato è il più significativo. Stesso discorso per parola e parola lunga. Non basta comunque cambiare il bit più significativo per passare, esempio da +10 a −10; occorre fare il complemento a due del dato, che consiste nell'invertire tutti i bit (da 0 a 1 e viceversa) e aggiungere 1 al risultato. Ad esempio il numero +1 è rappresentato da tutti i bit a 0 tranne il meno significativo (bit 0); per passare a −1 occorre invertire tutti i bit e poi incrementare di uno (questo vale anche per altri valori), per ottenere così tutti i bit ad 1, che rappresentano il numero −1. Tale convenzione è stata adottata in quanto quando si decrementa il numero zero (tutti i bit a 0) si dovrebbe passare al valore −1 (come ci insegna la matematica) e dato che i bit della cella di memoria vengono tutti impostati a 1 si è stabilito che tale stato dei bit deve indicare proprio il numero −1. Si spiega così anche il mistero della «estensione del segno»; infatti se ad esempio una parola ha tutti i bit ad 1 (quindi rappresenta il numero −1) passando in formato parola lunga anche i restanti 16 bit dovrebbero essere ad uno, ed è proprio quello che si ottiene con l'estensione del segno. Concluso questo (noioso) discorso torniamo a parlare di registri. Esiste un registro interno detto program counter; tale registro contiene l'indirizzo della prossima istruzione da eseguire e viene di volta in volta alterato a seconda del «flusso» del programma; inutile sottolinearne l'importanza.

Come abbiamo detto in precedenza, lo stack pointer è realizzato con l'ausilio del registro indirizzi A7. Ci sono due stack che possono essere selezionati agendo sul bit 13 del registro di stato, altro importante registro che esamineremo tra breve.

Lo stack più usato (dall'utente) e il cosiddetto USP, mentre l'altro è chiamato SSP (stack supervisore) ed è generalmente utilizzato dal sistema operativo.

Vediamo quindi il registro di stato che è un insieme di 16 bit che in questo caso vengono chiamati flag in quanto sono degli indicatori di avvenimenti che si verificano con il succedersi delle istruzioni «condizionate» ovvero che agiscono in funzione del verificarsi di una determinata situazione.

Vediamoli in dettaglio;

bit 0: è il flag del carry, che viene posto a 1 quando una operazione di somma ha generato un «riporto» o nel caso una sottrazione abbia richiesto un «prestito». Viene posto a zero quando l'ultima operazione effettuata non verifica nessuna delle due condizioni;

bit 1: è il flag di overflow, che viene posto ad 1 quando una operazione va oltre i limiti impostati dal formato dell'istruzione;

bit 2: è il flag di zero e viene settato a 1 quando il risultato dell'ultima operazione è uguale a zero;

bit 3: è il flag di negativo, che viene settato se il bit più significativo del valore trattato è anche settato, secondo le regole del complemento a due;

bit 4: è il flag di extende, ovvero quello che agisce similmente al carry ma in operazioni in formato BCD (tecnica particolare di rappresentare valori);

bit da 8 a 10: sono la «maschera delle interruzioni» e indicano quale delle sette interruzioni è settata (in seguito ci occuperemo anche delle interruzioni);

bit 13: come abbiamo detto è il flag che indica quale dei due stack stiamo utilizzando;

bit 15: è il flag che indica se il modo trace è settato o meno (posto a 1 o 0); tale modo è un sistema per rallentare l'esecuzione del programma al fine di individuare eventuali «bug».

Prima di continuare voglio farvi notare la piccola differenza che viene a crearsi quando utilizziamo lo stack del modo supervisore: solo con questa modalità possono essere utilizzate certe istruzioni particolari, che tuttavia non tratteremo.

Vediamo come vengono gestiti i dati in riferimento alla memoria. Quando un'operazione in formato parola lunga viene effettuata, il byte 3 viene trasferito nell'indirizzo N, il byte 2 nell'indirizzo N+1 e così via, mentre se l'operazione è in formato parola è il byte 1 ad essere trasferito nell'indirizzo N ed il byte 0 nell'indirizzo N+1. I dati vengono trasferiti dalla memoria con il procedimento inverso, ovvero byte più alto dall'indirizzo più basso.

In figura 4 potete osservare la tabella dei modi di indirizzamento; esaminiamoli in dettaglio. Il primo modo è il «registro dati diretto», ed è quello usato, ad esempio, nell'istruzione:

MOVE.W D1, D2

I dati vengono prelevati dal registro D1 e

Tabella dei modi di indirizzamento e relativa sintassi

indirizzamento	D	M	C	A	sintassi
registro dati diretto	x			x	Dn
registro indirizzi diretto				x	An
registro ind. indiretto	x	x	x	x	(An)
registro ind. ind. + inc. successivo	x	x		x	(An)+
registro ind. ind. + dec. precedente	x	x		x	-(An)
registro ind. ind. + scostamento	x	x	x	x	d(An)
registro indirizzo ind. + indice	x	x	x	x	d(An,Ri)
registro assoluto corto	x	x	x	x	xxx
registro assoluto lungo	x	x	x	x	xxxxxx
PC con scostamento	x	x	x		d(PC)
PC con indice	x	x	x		d(PC,Ri)
immediato	x	x			#xxx

D=dati; M=memoria; C=controllo; A=alterabile

Figura 4

spostati in D2 (in questo caso solo i primi 16 bit). La seconda modalità è quella chiamata «registro indirizzo diretto» che preleva i dati da un registro degli indirizzi; ad esempio:

```
MOVE.L A1, D1
```

trasferisce l'intero contenuto a 32 bit del registro indirizzi A1 nel registro dati D1. Le due modalità di indirizzamento appena viste rientrano nella categoria dei «modi a registro diretto». Una seconda categoria è quella che si occupa degli indirizzamenti fatti sulla memoria e comprende i successivi 5 modi della tabella; vediamoli in dettaglio. Il «registro indirizzo indiretto» è quello usato in una istruzione del tipo:

```
MOVE.L (A1), D1
```

Tale istruzione preleva il contenuto della locazione di memoria indicata dal registro indirizzi A1 e lo trasferisce (32 bit) in D1. Da notare che il registro degli indirizzi viene sempre considerato con tutti i suoi 32 bit (anche se in realtà sono solo 24), mentre è la lunghezza del dato che viene interessata dalla specificazione («.B», «.W», o «.L»).

Il «registro indirizzo diretto + incremento successivo» è simile al precedente indirizzamento ma in più aggiunge al registro indirizzi interessato nell'operazione un valore in funzione della lunghezza del dato prelevato, in modo che l'indirizzo punti esattamente alla locazione di memoria successiva a quella appena «letta». L'incremento è «successivo», ovvero avviene dopo il trasferimento, un esempio è il seguente:

```
MOVE.W (A0)+, D0
```

prende il contenuto della locazione di memoria indicata da A0 e lo trasferisce in D0 e successivamente il registro A0 viene incrementato di 2, automaticamente.

Il prossimo indirizzamento «registro indirizzo indiretto + decremento precedente» è l'opposto del precedente. Ad esempio:

```
MOVE.W D0, -(A0)
```

trasferisce D0 nella locazione puntata da A0, ma prima decrementa di 2 A0. Da notare che anche l'istruzione:

```
MOVE.W -(A0), D0
```

utilizza lo stesso modo di indirizzamento, ma il trasferimento avviene in direzione opposta (da locazione puntata da A0 a D0).

Il prossimo modo è «registro indirizzo indiretto + scostamento», che viene utilizzato di solito per prelevare dati, con uno stesso registro degli indirizzi, da due «tabelle» differenti, infatti l'istruzione:

```
MOVE.L D4, -15(A0)
```

prende il contenuto di D4 e lo trasferisce nella locazione puntata dal registro indirizzi A0 decrementato di 15. Il valore del registro A0 non viene alterato e può essere utilizzato in seguito per depositare dati in un punto diverso della memoria semplicemente cambiando lo «scostamento» (ovvero il valore -15, che è a 16 bit).

Quest'ultimo viene in precedenza trattato con l'operazione di «estensione del segno» (in modo fittizio, visto che è a 16 bit) così da trasformarlo in un complemento a due a 32 bit (per poterlo sommare al registro indirizzi, che è appunto in formato 32 bit). Ultimo modo del gruppo è «registro indirizzo indiretto + indice». Spieghiamoci subito con un esempio. L'istruzione:

```
MOVE.W 40 (A0,D0.L), D1
```

ricerca il dato, da depositare in D1, prelevando dall'indirizzo di memoria ricavato sommando il contenuto di D0 (parola lunga) ad A0, il tutto sommato ancora al valore 40. La lunghezza del registro dati (nell'esempio D0) può essere specificata. Lo «scostamento» è ad 8 bit. Sia lo scostamento che il registro dati subiscono una «estensione del segno». I prossimi modi di indirizzamento li vedremo nella puntata seguente.

Una raccomandazione da farvi è la seguente: soprattutto se siete alle prime armi con questo linguaggio non scoraggiatevi della mole di nuove nozioni da apprendere perché le cose diverranno molto più chiare man mano che ci addentreremo nelle lezioni successive.

Vi lascio con un piccolo programma Assembler da utilizzare come esperimento con il vostro editor:

```
      MOVE.L # 0,D0
LOOP: MOVE D0,$DFF180
      SUB.L # 1,D0
      BNE LOOP
      RTS
```

Se state usando il Macro Assembler dell'AmigaDOS, la procedura è la seguente: dovete utilizzare un editor di testi, ad esempio quello fornito sul dischetto del workbench «ED» (vedi il manuale di istruzioni di Amiga per il suo utilizzo), quindi salvare il testo e poi, sempre da CLI impostare il comando:

```
ASSEM nometesto -0 nometesto.obj
```

quindi l'assemblatore partirà con le sue passate, restituendo come risultato il file «nometesto.obj». A questo punto occorre «linkare» il file «nometesto.obj» con il seguente comando:

```
ALINK nometesto.obj TO nometesto.exe
```

per ottenere il file eseguibile CLI. È chiaro che i nomi dei file possono essere diversi, a seconda dei vostri gusti. Sembra una procedura alquanto bizzarra, ma ha una sua logica ben precisa.

Il miniprogramma presentato trasferisce nella palette colore (quella dello sfondo, ovvero del colore numero 0), tutta una serie di valori in modo rapidissimo; completamente inutile ma efficace come test. Alla prossima puntata.