IMPARIAMO A PROGRAMMARE IN assembler

di Valter Di Dio

In questa puntata ci occuperemo di alcune semplici routine di calcolo intero che possono essere utili nei programmi per fare piccoli conti. Si possono però facilmente estendere a più di sedici bit e ottenere così dei programmini capaci di eseguire le quattro operazioni su interi particolarmente grandi. Usando per esempio 5 byte per ciascun operando si può lavorare con numeri interi compresi tra -549 e più 549 miliardi!

Tutto questo senza alcuna approssimazione e occupando lo stesso spazio in memoria di un numero in floating point.

L'operazione più semplice

Contrariamente a quanto si può pensare l'operazione più semplice non è la somma ma la moltiplicazione, naturalmente per un preciso moltiplicatore. Infatti la prima operazione che si impara è la moltiplicazione per dieci, che si esegue semplicemente aggiungendo uno zero. Si può più correttamente vedere l'aggiunta di uno zero come lo scorrimento a sinistra di un posto di tutte le cifre che compongono il numero.

Non è certo un caso però che, in base dieci, la moltiplicazione per dieci sia l'operazione più immediata, ma dipende proprio dal fatto che i valori posizionali delle cifre sono potenze di dieci. Se trasferiamo il discorso sull'aritmetica binaria, dove i valori posizionali delle cifre sono potenze di due, l'operazione più facile diventa la moltiplicazione per due! Basta infatti far scorrere a sinistra tutti i bit di un numero per averlo moltiplicato per due; un altro scorrimento e abbiamo moltiplicato per quattro poi per 8, 16, 32 ecc. Naturalmente ad un certo punto, lavorando sul singolo byte, cominceranno a cadere fuori da sinistra i bit più significativi; nessun problema: l'istruzione di scorrimento a sinistra (ASL) manda automaticamente nel Carry i bit che escono da sinistra (uno alla volta!) come risulta evidente dalla figura 1a. Cosa fare del bit caduto fuori? Beh, se il numero era a otto bit allora è un overflow, se era a sedici allora lo dobbiamo spingere nella locazione che contiene la parte alta. Il modo più comodo è di usare un'istruzione diversa per far scorrere la parte alta: la ROL (ROtate Left). Questa istruzione effettua una rotazione a sinistra a nove bit, dove il nono bit è il Carry e per rotazione si intende che quello che esce da sinistra finisce nel Carry mentre quello che era nel Carry rientra da destra. Vedi schema in figura 1b.

Usando la ROL per la parte alta automaticamente infiliamo da destra quello che cade fuori da sinistra della parte bassa. Vedi schemino in figura 1c.

Moltiplicazione per due a sedici bit:

```
ASL memL
ROL memH
BCS overflow
RTS
```

Se dopo la ROL il Carry è pieno vuol dire che è caduto fuori un bit da sinistra della parte alta. Se avessimo un numero a 24 bit basterebbe aggiungere un'altra ROL per recuperare il bit caduto fuori e così via. Altrimenti andremo ad una routine di overflow che ci avverte che il risultato dell'operazione è un numero di diciassette bit.

Operazione del tutto simile sia come logica che, in pratica, come programma è la divisione per due, che si ottiene facendo scorrere i bit verso destra anziché verso sinistra. Le due istruzioni di scorrimento a destra del 6502 sono la LSR (Logical Shift Right) e la ROR (ROtate Right) del tutto simili alle precedenti salvo per il verso di scorrimento. Il Carry settato dopo una divisione per due significa che ci stiamo perdendo il primo decimale ($2^{-1} = 0.5$); in altre parole che il numero appena diviso per due era dispari!

L'addizione

Lasciamo per ora la moltiplicazione e impariamo a fare le somme.

L'istruzione che dice al 6502 di sommare due numeri è la ADC (ADd with Carry) che effettua la somma tra il contenuto di una memoria, l'accumulatore e il Carry.

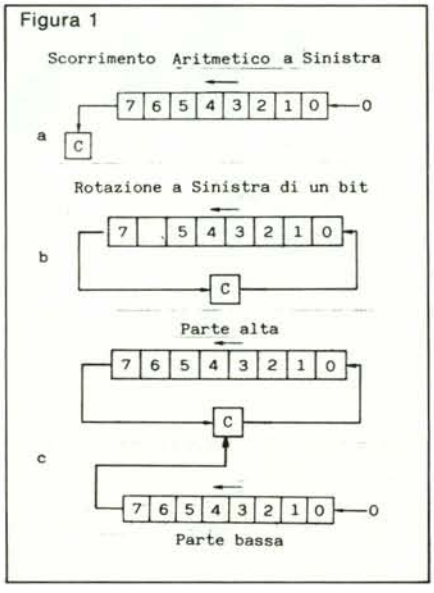
Vediamo subito la somma di due numeri a otto bit: uno sta nella locazione MEM, l'altro si trova già nell'Accumulatore e il risultato lo mettiamo nella memoria RIS. Dato che la somma di due numeri a otto bit può generare un risultato a nove bit, il nono bit si troverà alla fine nel Carry.

```
CLC
ADC MEM
STA RIS
RTS
```

Quindi: puliamo il Carry, sommiamo e immagazziniamo il risultato.

Attenzione, il 6502 dispone anche della somma in decimale, e il programma è identico al precedente; quindi per essere sicuri che il risultato sia corretto conviene aggiungere all'inizio delle nostre routine in binario l'istruzione CLD (CLear Decimal mode) che costringe il 6502 a lavorare in binario!

Figura 1



Proviamo ora una somma a 16 bit; i due operandi sono OP1L OP1H ed OP2L, OP2H, il risultato lo mettiamo in RISL e RISH.

Il programma risulta ancora molto semplice, grazie al fatto che il riporto è automaticamente sommato dalla istruzione ADC.

```
CLD
CLC
LDA OP1L legge la parte bassa di OP1
ADC OP2L (OP1+OP2) parti basse
STA RISL conserva il risultato L
LDA OP1H metà alta di OP1
ADC OP2H (OP1+OP2) parti alte
STA RISH conserva la parte alta di RIS
RTS
```

Si potrebbe procedere all'infinito sommando via via byte sempre più significativi senza alcun problema. Attenzione anche qui al carry dopo la somma, che se pieno indica un overflow.

La sottrazione

La sottrazione a otto bit è talmente semplice, a questo punto, che vediamo subito il programma a 16 bit. Eseguiamo quindi OP1-OP2

```
CLD
SEC
LDA OP1L
SBC OP2L
STA RISL
LDA OP1H
SBC OP2H
STA RISH
RTS
```

Vediamo cosa è cambiato dalla corrispondente operazione di somma a otto bit: per prima cosa abbiamo settato il Carry invece di vuotarlo. Questo perchè l'operazione di sottrazione usa il Carry come "prestito" e per poter prestare qualcosa bisogna prima averla!

Seconda differenza, del resto ovvia, è che abbiamo usato l'istruzione SBC che significa appunto Sottrai con Carry. La SBC esegue l'operazione Accumulatore — Memoria — Carry (negato). Alla fine dell'operazione il Carry vuoto non significa overflow ma che il risultato è errato in quanto il contenuto della memoria era superiore al contenuto dell'Accumulatore e l'operazione ha generato un numero negativo. Se proprio volete sapere quale era il risultato esatto dovete fare il complemento a 2 dell'Accumulatore e cambiarlo di segno.

Notate che se state già lavorando con numeri in complemento a due, il Carry lo potete trascurare ma dovete tenere d'occhio il flag di Overflow che vi segnalerà un riporto errato tra il Bit 6, che fa parte del dato, e il bit 7, che rappresenta il segno!

La moltiplicazione

Per capire come si esegue la moltiplicazione binaria vediamo come si opera la normale operazione in decimale con carta e penna.

Eseguiamo 12*23

```
 12 x   (moltiplicando)
 23 =   (moltiplicatore)
-----
 36 +   (prodotto parziale)
 24 =   (secondo parziale)
-----
276     (risultato finale)
```

La moltiplicazione è stata perciò eseguita sommando al risultato della moltiplicazione della prima cifra del moltiplicando per il moltiplicatore, il risultato del prodotto tra la seconda cifra del moltiplicando e il moltiplicatore, ma

nota bene, scalata a sinistra di un posto! La moltiplicazione binaria è eseguita esattamente allo stesso modo:

```
5: =        101 ×   (MPD)
3: =        011 =   (MPR)
            ---
            101 +   (PP1)
           101  +   (PP2)
           000  =   (PP3)
           ---
15: =      01111    (RIS)
```

Se osservate i prodotti parziali notate che in binario si deve solo riscrivere l'MPD se il bit dell'MPR è uno altrimenti si effettua solo lo scorrimento a sinistra. Quindi moltiplicare significa in pratica solo verificare ad uno ad uno i bit del moltiplicatore e sommare o no ad un registro temporaneo il moltiplicando spostato di un posto.

Dato che non esistono nel 6502 istruzioni che permettono di verificare i singoli bit di una memoria qualsiasi (e neppure dell'Accumulatore!) ci serviremo allo scopo dell'istruzione LSR che applicata all'MPR ci farà cadere ad uno ad uno i bit di destra nel Carry, dove possiamo controllarli. Altro piccolo problema è il fatto che la moltiplicazione di due numeri ad otto bit può generare un risultato a sedici; questo ci impedisce di usare uno dei registri interni del microprocessore per accumulare il prodotto parziale, cosa che renderebbe molto più veloce tutta l'operazione.

Vediamo ora passo passo il programma:

```
        LDA #$0  ; azzera A
        STA TMP; azzera la loc.
                 di appoggio
        STA RISL; azzera il RISL
        STA RISH; e il RISH
        LDX #$8  ; otto volte per otto bit
MULT    LSR MPR  ; casca fuori il primo bit
        BCC
        NOSUM    ; se è zero non sommare
        LDA RISL ; riprendi il risultato parziale
        CLC      ; sempre prima di sommare!
        ADC MPD  ; sommaci il moltiplicando
        STA RISL ; rimetti a posto RISL
        LDA RISH ; prendi la parte alta
        ADC TMP  ; sommaci TMP+il Carry
        STA RISH ; e rimettila a posto
NOSUM   ASL MPD  ; fai scorrere MPD a sinistra
        ROL TMP  ; conserva il bit cascato fuori
                 da MPD
        DEX      ; x=x-1
        BNE MULT ; ripeti per tutti e otto i bit
        RTS
```

Questo programma è la traduzione esatta dell'algoritmo di moltiplicazione appena visto, ma ci possono essere altri modi di eseguire la stessa operazione; ad esempio invece di far scorrere a sinistra il moltiplicando si può far scorrere a destra il risultato parziale e questo ci eviterebbe l'uso della locazione TMP.

Vediamo allora come solo con un uso più razionale dei registri e della memoria si possa risparmiare spazio e tempo.

Dei registri interni del 6502 l'accumulatore è l'unico che può esser fatto scorrere, è quindi qui che dovremo mettere o il risultato o il moltiplicatore. Ma, dato che è sempre solo il registro A a poter eseguire l'addizione, non è conveniente metterci il moltiplicatore (che non viene mai sommato) ci metteremo perciò il risultato. Il registro X continuiamo ad usarlo come contatore di bit e il registro Y non ci serve a niente dato che non può essere sommato ad A (altrimenti sarebbe stato comodo per metterci il moltiplicando).

Il moltiplicando, il moltiplicatore e parte del risultato dovranno quindi stare in memoria. Possibilmente in pagina zero.

Invece di far scorrere il moltiplicatore a sinistra otto volte faremo scorrere il risultato a destra, il che è la stessa cosa, ma ci permette di trasferire ad uno ad uno i bit del risultato dalla parte alta (che sta nell'Accumulatore) alla parte bassa (che si trova in pagina zero) con una semplice operazione

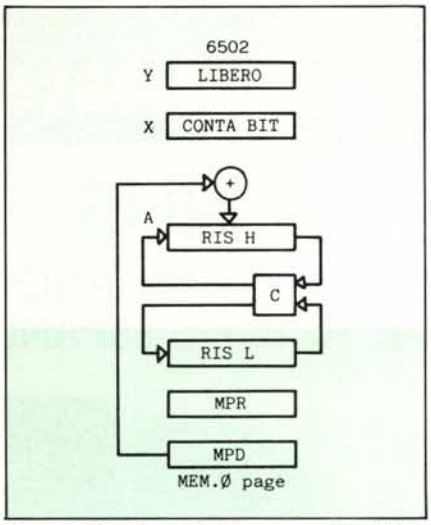


Figura 2 - Uso dei registri nel programma di moltiplicazione ottimizzato.

di scorrimento attraverso il Carry. Vedi schema di figura 2. Il programma diventa il seguente:

```
        LDA #$0
        STA RISL
        LDX #$8
loop    LSR MPR
        BCC nosum
        CLC          ; indispensabile il carry è 1
        ADC MPD      ; A = A + MPD
nosum   ROR          ; scorre il risultato H
        ROR RISL     ; cattura il bit uscito da RISH
        DEX
        BNE loop
        RTS
```

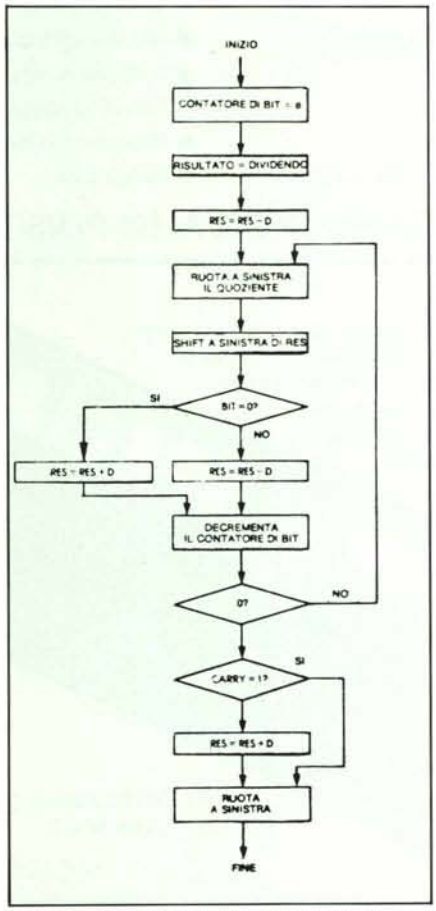


Figura 3 - Diagramma di flusso divisione 16 × 8.

Provate, seguendo lo schema di figura 2, ad eseguire l'operazione a mano e vedrete come i singoli bit del risultato scorrono ad uno ad uno dall'accumulatore alla memoria RISL e come nell'accumulatore resti proprio la parte alta del risultato.

Anche questo programma potrebbe essere ulteriormente ottimizzato notando come l'istruzione CLC sia indispensabile dato che se incontriamo un uno nel moltiplicatore allora dobbiamo eseguire la somma, ma così facendo ci ritroviamo proprio quell'uno nel Carry.

Se noi allora, prima di eseguire la moltiplicazione, invertiamo tutti i bit del moltiplicatore e sostituiamo la BCC con una BCS, l'istruzione CLC diventa inutile; con un risparmio medio di "ben" 8 microsecondi.

La divisione

L'algoritmo della divisione è del tutto simile a quello della moltiplicazione solo che invece di sommare si sottrae.

Ovvero: il divisore è via via sottratto dai bit di sinistra del dividendo.

Dopo ogni sottrazione, il risultato è sostituito al dividendo iniziale e il quoziente viene aumentato di uno.

Se la sottrazione genera un risultato negativo (prestito!) si decrementa il quoziente di uno, si riaggiunge il divisore al risultato parziale e si fanno scorrere di una posizione sia il dividendo che il quoziente.

Effettuando il controllo del risultato prima di eseguire la sottrazione si risparmia il reimmagazzinamento del divisore in caso di eccesso. Il flow chart relativo è quello di figura 3. Le locazioni usate sono: il dividendo in A (parte alta) e DIV (parte bassa); il divisore in DVS. Il listato del programma è il seguente:

```
        LDY #$8
        SEC
        SBC DVS
loop    PHP          ; salva lo status
        ROL QZT      ; ruota il quoziente
        ASL DIV      ; shift del divisore L
        ROL          ; e del divisore H
        PLP          ; riprendi lo status
        BCC add      ; somma se DVS > RES
        SBC DVS      ; altrimenti sottrai
                     ; nota: il carry è già ad 1
        JMP next
add     ADC DVS      ; il carry è già 0
next    DEY          ; decrementa il contatore
        BNE loop     ; ancora
        BCS last     ; nessun eccesso
        ADC DVS
        CLC          ; pulisci il carry prima della ROL
last    ROL QZT      ; aggiusta il risultato
        RTS          ; e fine
```

Il risultato si trova ora in QZT e il resto nell'Accumulatore.

Conclusioni

Con questa panoramica non si esaurisce certo il discorso sulle operazioni possibili, rimangono infatti ancora le operazioni con numeri negativi o in BCD e soprattutto quelle in virgola mobile. Per tutti questi casi conviene, piuttosto che riscriversi le routine, cercare di usare quelle già esistenti all'interno dell'interprete Basic e in alcuni casi anche nelle ROM del MONITOR.

Come consiglio generale conviene sempre usare il Basic per le routine algebriche, soprattutto se complesse, e piccole routine in Linguaggio Macchina per quei casi in cui la velocità sia indispensabile e i calcoli ridotti all'osso.

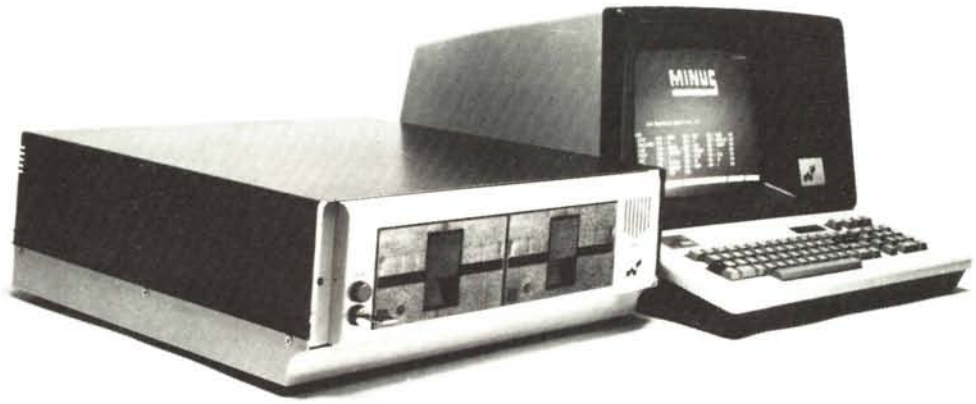
Nella prossima puntata, che concluderà questo ciclo di articoli, vedremo un po' in generale la gestione degli interrupt, il problema delle temporizzazioni e i tempi di esecuzione di routine in Assembler. MC

INCREDIBILE!

per sole 3.750.000 acquistate un KYBER

MINUS

IL PERSONAL COMPUTER ITALIANO PIÙ COMPLETO MA ANCHE IL PIÙ ESPANDIBILE

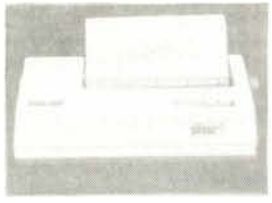


ECCO I SUOI PRIMATI:

- Memoria residente da 64K a 256K.
- 3 sistemi operativi CP/M2.2, CP/M3.0, TURBODOS.
- Sistema operativo grafico GSX.
- Memoria di massa da 800K a 10Mb.
- Solid State Disk™: 128K.
- Grafica professionale alta risoluzione: 512x512 pixel.
- Autodiagnostica.
- Alta velocità di accesso ai dischi.
- Software professionale per tutte le applicazioni. Tutti i linguaggi evoluti.
- Assistenza in tutta Italia.
- Basso costo.

(CERCATELI NEGLI ALTRI PERSONAL!)

STX 80



Stampante 80 colonne, 9 aghi, 60 cps bidirezionale, interfaccia parallela o seriale. Grafica ad alta risoluzione bit image.

OFFERTA SPECIALE

– Configurazione da 64K RAM.
2 Floppy disk 400+400K.
Display 2000 caratteri.

– Sistema operativo.
– Linguaggio Basic L. 3.750.000 + IVA
– Stampante STX 80 L. 360.000 + IVA

Offerta valida, per una sola unità, fino al 31.02.1984 per acquisto in fabbrica.

KYBER CALCOLATORI

S.R.L. 51100 Pistoia (Italy) Tel. 0573/368113 (2 linee)
Via Ariosto 16-22

SI CERCANO RIVENDITORI