IMPARIAMO A PROGRAMMARE IN ASSEMBLER

di Valter Di Dio

Assembler terza parte, già dovremmo essere in grado di muovere i primi passi in questo strano mondo fatto di numeri che non si possono contare sulla punta delle dita. Certo è che se avessimo avuto sedici dita, forse ci saremmo trovati molto avvantaggiati nell'uso dei microprocessori, ma non disperate: un giorno riuscirete anche voi a pensare a B5 come ad un numero e non ai colpi della battaglia navale che si giocava a scuola; chissà che facendo dei tabelloni per battaglia navale con sedici caselle per lato non si riesca a far diventare meno aliena questa aritmetica!

Digressioni a parte siamo convinti che già molti di voi si saranno divertiti a smaneggiare l'Apple con l'asterisco per prompt e a lanciare in esecuzione un po' di routine in linguaggio macchina anche senza conoscerne l'esito.

Speriamo ardentemente che ad alcuni si sia bloccato tutto e siano stati costretti a spegnere e riaccendere il computer, non per nostro speciale sadismo ma perché solo il vero spirito eletto, colui che non segue gli altri ma si spinge da solo oltre il limite della conoscenza finisce prima o poi per fare quella fine.

Soluzioni

Nella scorsa puntata avevamo chiesto di correggere il programma che azzerava 256 locazioni di memoria a partire dalla $400 e di scriverne uno che pulisse tutta la Ram dello schermo che va da $400 a $7FF.

L'errore presente nel primo programma era costituito dal fatto che non veniva azzerata la locazione $400, le possibili correzioni sono diverse: si può aggiungere una STA $400 prima della fine del programma o più semplicemente basta cambiare LDX #$FF in LDX #$00 affinché venga prima azzerata la locazione $400 + 0 poi le altre a partire da $FF indietro fino alla $1.

Questo metodo non è però utilizzabile se si devono azzerare meno di 256 locazioni; infatti ricordiamo che i registri e le locazioni a otto bit vanno visti come un quadrante rotondo munito di lancetta e con i numeri scritti attorno, quindi la lancetta può girare all'infinito perché dopo 255 viene zero e prima di zero viene 255!

Il microprocessore non ci avverte del passaggio per zero e tocca al programmatore controllare, mediante i flag del registro P, cosa stia accadendo dentro la CPU. Allora, se dobbiamo azzerare $E0 locazioni soltanto conviene scambiare di posto la

```
*300L

0300-   A9 A0      LDA   #$A0
0302-   A2 00      LDX   #$00
0304-   9D 00 04   STA   $0400,X
0307-   9D 00 05   STA   $0500,X
030A-   9D 00 06   STA   $0600,X
030D-   9D 00 07   STA   $0700,X
0310-   CA         DEX
0311-   D0 F1      BNE   $0304
0313-   60         RTS
```

Figura 1 - Routine di HOME due volte più veloce di quella originale.

DEX con la STA (in modo che venga prima decrementato il puntatore e poi caricata la locazione) e mettere $E1 in X. Da notare che il modus operandi è lo stesso del Basic nel caso che non fossero disponibili istruzioni di tipo FOR-NEXT.

Vediamo ora come sia possibile realizzare in linguaggio macchina la funzione HOME. Un primo metodo potrebbe essere quello di scrivere quattro programmi identici per ciascuno dei quattro blocchi da 256 Byte che compongono la Memory Map del video. Se lo facessimo ci accorgeremmo subito che molte istruzioni sarebbero ripetute in tutti e quattro i blocchi (in particolare tutte quelle che gestiscono il ciclo) e cambierebbero solo le STA diventando via via STA $400,X poi $500,X poi $600,X e infine $700,X. Dal momento che X varia sempre da $FFa $00 tanto vale mettere un solo ciclo e quattro STA! Il programma diventa così quello di figura 1. Esistono tuttavia altre soluzioni interessanti; se qualcuno avesse infatti usato

0300- 4C 58 FC JMP $FC58

che significa salta alla HOME del Monitor, sarebbe stato certamente molto furbo e bisogna dire che questa, a patto che esista nel Monitor o nel Basic la routine giusta, resta sempre senza dubbio la soluzione migliore.

Un altro metodo molto interessante, soprattutto per i risvolti psicologici che sottintende è quello chiamato dell'auto-modifica ovvero il programma modifica se stesso!

A parte strani presagi Frankensteiniani di macchine che si auto-programmano il metodo è molto comodo; occorre naturalmente che il programma sia scritto in RAM e inoltre, dal momento che dopo aver girato è diverso da quello iniziale, diventa necessaria una routine che riinizializzi il programma ad ogni chiamata (è anche possibile che esistano varie routine di inizializzazione ad esempio per azzerare la prima o la seconda pagina di testo).

In figura 2 trovate un programma siffatto; vediamolo passo passo:

la parte da $300 a $30A è il nostro vecchio esempio, è stato solo cambiato il valore della locazione $301 da 0 in $A0 in modo da stampare uno spazio invece della chiocciola in inverso, nelle locazioni $305 e 306 si trova l'indirizzo base che dovrà essere cambiato di volta in volta in $400, $500, $600 e $700.

Dal momento che gli indirizzi sono scritti in due byte divisi tra parte bassa e parte alta, e che la parte bassa resta sempre uguale a zero, dovremo solo cambiare la parte alta (locazione $306) in $4, $5, $6 e $7 e lanciare ogni volta il programma modificato.

Il nostro programma di pulizia di 256 byte può essere visto ora come sottoprogramma, tanto termina con un RTS, ci basta solo realizzare adesso il programma principale che inizializza la locazione $306 al valore desiderato e chiama la subroutine $300.

Il programma principale inizia a $30B. Carica in A $4 e lo deposita in $306, poi effettua un JSR $300 (Jump to Subroutine) poi incrementa $306 e così via per quattro volte.

Due cose da notare: avevamo detto che non è possibile compiere alcuna operazione nelle celle della memoria; come vedete questo non era del tutto corretto; è infatti

possibile compiere direttamente alcune operazioni elementari senza passare per i registri del 6502, le operazioni possibili sono ASL, BIT, DEC, INC, LSR, ROL e ROR il cui significato lo trovate nella tabella 1 della scorsa puntata (della BIT parleremo alla prima occasione).

```
*300L

0300-   A9 A0       LDA   #$A0
0302-   A2 00       LDX   #$00
0304-   9D 00 07    STA   $0700,X
0307-   CA          DEX
0308-   D0 FA       BNE   $0304
030A-   60          RTS
030B-   A9 04       LDA   #$04
030D-   8D 06 03    STA   $0306
0310-   20 00 03    JSR   $0300
0313-   EE 06 03    INC   $0306
0316-   20 00 03    JSR   $0300
0319-   EE 06 03    INC   $0306
031C-   20 00 03    JSR   $0300
031F-   EE 06 03    INC   $0306
0322-   4C 00 03    JMP   $0300

*300.325

0300- A9 A0 A2 00 9D 00 07 CA
0308- D0 FA 60 A9 04 8D 06 03
0310- 20 00 03 EE 06 03 20 00
0318- 03 EE 06 03 20 00 03 EE
0320- 06 03 4C 00 03 00
```

Figura 2 - Routine di pulizia della pagina di testo scritta in modo Automodificante.

La seconda nota riguarda il fatto che in $322 è stato usato un JMP (salto brutale) invece di un JSR, questo consente di utilizzare l'RTS della subroutine come chiusura anche del programma principale risparmiando un byte in memoria e senza caricare lo stack con un indirizzo di ritorno inutile.

In figura 3 trovate un altro esempio di programma automodificante che utilizza il registro Y per contare i quattro cicli. In pratica sono due cicli FOR-NEXT nidificati; quello esterno gestito dal registro Y e quello interno dal registro X.

La pagina zero

Le 65536 celle di memoria del computer possono essere viste come 256 pagine di 256 locazioni ciascuna. Una pagina è composta da tutte quelle locazioni che hanno il byte alto dell'indirizzo uguale: perciò le locazioni da 2000 a 20FF appartengono alla pagina 20 quelle da $400 a $4FF alla pagina 4 e via di seguito. Useremo spesso il concetto di pagina per indicare banchi di memoria da 256 byte.

Una pagina in particolare si distingue dalle altre per le sue proprietà: la Pagina Zero. È composta ovviamente dalle prime 256 locazioni di memoria e va da $0000 a $00FF.

La sua particolarità sta nel fatto che le sue celle possono essere indirizzate comunicando al microprocessore solo la parte bassa dell'indirizzo, risparmiando così un byte nell'istruzione ed accelerando inoltre il calcolo degli indirizzi da parte del microprocessore. Di quasi tutte le istruzioni esiste la versione Zero-Page. Di solito la pagina zero viene usata per immagazzinare dati temporanei che devono essere usati molto spesso in modo da facilitarne l'accesso al 6502.

Lo Stack

Una seconda pagina assume particolare importanza proprio per come è stato realizzato internamente il 6502: la pagina uno. Qui è collocato infatti lo Stack del microprocessore.

Lo Stack va visto come una pila di registri a otto bit in cui sia possibile l'accesso solo all'ultimo dato inserito.

Un classico esempio di Stack è quello spillone in cui i bottegai infilavano un tempo i conti in sospeso; per leggere il foglietto "infilzato" per primo occorreva sfilare prima tutti quelli che gli stavano sopra. Proprio per questo sono sufficienti solo due comandi per la gestione dello stack: uno per depositare qualcosa sopra la pila e un altro per prelevare ciò che si trova in cima alla pila.

Lo Stack serve al microprocessore per depositarvi gli indirizzi di ritorno dalla subroutine, ma può essere contemporaneamente usato dal programmatore a patto di togliere sempre tutto quello che ci si è messo prima di effettuare un RTS. Un registro interno del 6502 punta sempre al dato più recente dello Stack e viene chiamato registro S. Dato che abbiamo detto che lo Stack si trova in pagina uno il registro S basta che sia a otto bit; questo da un canto facilita la gestione da programma dello Stack, dall'altro limita lo Stack a 256 locazioni e lo vincola alla pagina uno. Esistono due istruzioni che permettono di trasferire S in X e viceversa. Notate come sia comodo leggere una cella dello stack avendo in X il puntatore (LDA $100,X).

```
*300L

0300-   A9 A1       LDA   #$A1
0302-   A0 04       LDY   #$04
0304-   8C 0B 03    STY   $030B
0307-   A2 00       LDX   #$00
0309-   9D 00 08    STA   $0800,X
030C-   CA          DEX
030D-   D0 FA       BNE   $0309
030F-   EE 0B 03    INC   $030B
0312-   88          DEY
0313-   D0 F2       BNE   $0307
0315-   60          RTS
```

Figura 3 - Esempio di due cicli FOR-NEXT nidificati.

```
*300L

0300-   A9 C1       LDA   #$C1
0302-   20 ED FD    JSR   $FDED
0305-   18          CLC
0306-   69 01       ADC   #$01
0308-   C9 DB       CMP   #$DB
030A-   90 F6       BCC   $0302
030C-   60          RTS

*300G
ABCDEFGHIJKLMNOPQRSTUVWXYZ
*
```

Figura 4 - Programma per la stampa delle lettere dell'alfabeto.

```
*300L

0300-   A2 C1       LDX   #$C1
0302-   8A          TXA
0303-   20 ED FD    JSR   $FDED
0306-   E8          INX
0307-   E0 DB       CPX   #$DB
0309-   90 F7       BCC   $0302
030B-   60          RTS
```

Figura 5 - Un altro modo di stampare le lettere dell'alfabeto.

I salti incondizionati

Esistono solo due tipi di salto brutale nel 6502: il JMP (GOTO) e il JSR (GOSUB). Il JMP può essere immediato, per esempio JMP $0300 che equivale a un GOTO 300, o indiretto, per esempio JMP ($300) che significa salta alla locazione che c'è scritta in $300, $301 (ricordate che gli indirizzi sono sempre scritti con la parte bassa che precede quella alta; per cui se in $300 leggiamo $10 e in $301, $AF, il salto sarà a $AF10 e non a $10AF).

Il JSR è sempre immediato ma è possibile alterare il flusso delle subroutine intervenendo sullo Stack (ma ci vuole una certa esperienza!).

Nel programma di figura 2 abbiamo già visto il normale uso dei salti, del resto non c'è niente di diverso dai GOTO e GOSUB del Basic.

I salti condizionati

Nel primo programma che abbiamo fatto ci siamo serviti di un tipo di salto condizionato (del resto non esiste quasi nessun programma in cui non si debba prendere alcuna decisione). Diamo ora una occhiata veloce agli altri tipi di salto tanto per sapere che esistono.

Come ricorderete i salti condizionati, o Branch, fanno riferimento al contenuto dei flag del registro P, in particolare ai flag 7,6,1,0.

Al flag 7 (risultato negativo cioè maggiore di 127) fanno riferimento i due BPL e BMI rispettivamente salta per positivo (Plus) e salta per negativo (Minus).

Al flag 6 (overflow ovvero riporto errato) corrispondono i BVC e BVS: salta per flag di overflow 0 (Clear) o 1 (Set).

Il flag uno lo abbiamo già conosciuto e controlla la "zericità" grazie ai BNE e BEQ.

Il flag 0 è uno dei più usati e permette di controllare il Carry cioè il riporto e il prestito nelle operazioni aritmetiche o il risultato dei confronti tra dati. Vediamo subito questo utilizzo che è uno dei più comuni.

Problema: vogliamo scrivere sullo scher-

TABELLA DEI CODICI ASCII-VIDEO

	Inverse				Flashing				Normal (Control)						Normal (Lowercase)	
Decimal	0	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240
Hex	$00	$10	$20	$30	$40	$50	$60	$70	$80	$90	$A0	$B0	$C0	$D0	$E0	$F0
0 $0	@	P		0	@	P		0	@	P		0	@	P		0
1 $1	A	Q	!	1	A	Q	!	1	A	Q	!	1	A	Q	!	1
2 $2	B	R	"	2	B	R	"	2	B	R	"	2	B	R	"	2
3 $3	C	S	#	3	C	S	#	3	C	S	#	3	C	S	#	3
4 $4	D	T	$	4	D	T	$	4	D	T	$	4	D	T	$	4
5 $5	E	U	%	5	E	U	%	5	E	U	%	5	E	U	%	5
6 $6	F	V	&	6	F	V	&	6	F	V	&	6	F	V	&	6
7 $7	G	W	'	7	G	W	'	7	G	W	'	7	G	W	'	7
8 $8	H	X	(	8	H	X	(	8	H	X	(	8	H	X	(	8
9 $9	I	Y	)	9	I	Y	)	9	I	Y	)	9	I	Y	)	9
10 $A	J	Z	*	:	J	Z	*	:	J	Z	*	:	J	Z	*	:
11 $B	K	[	+	;	K	[	+	;	K	[	+	;	K	[	+	;
12 $C	L	\	,	<	L	\	,	<	L	\	,	<	L	\	,	<
13 $D	M	]	-	=	M	]	-	=	M	]	-	=	M	]	-	=
14 $E	N	^	.	>	N	^	.	>	N	^	.	>	N	^	.	>
15 $F	O	_	/	?	O	_	/	?	O	_	/	?	O	_	/	?

Tabella dei codici decimali ed esadecimali che caricati nelle locazioni della memoria video, che va da $400 (1024) a $7FF (2047), consentono la generazione dei caratteri alfanumerici e dei simboli speciali sia in normale che in inverso o in lampeggiante. Da notare che i caratteri della colonna CONTROL sono invisibili e che quelli della colonna LOWERCASE vengono sostituiti dalle lettere minuscole e accentate quando si monti la eprom Apple Minus.

mo tutte le lettere maiuscole dell'alfabeto.

Dati: la routine che stampa un carattere sul video alla posizione corrente del cursore è la $FDED, il codice video relativo deve essere contenuto in A. Il codice della A è $C1 quello della Z è $DA.

Tutto il programma si riduce a caricare in A $C1, effettuare un JSR a FDED incrementare A di uno, controllare se è maggiore di $DA e continuare o finire a seconda del risultato del confronto.

Scriviamo il programma passo passo:

1) carichiamo in A il primo codice da stampare che è $C1 (lettera a maiuscola in Normal come risulta dalla tabella dei codici video dell'Apple che si trova a pagina 15 del Reference Manual). Utilizziamo la nota LDA # dato.

2) Effettuiamo un GOSUB alla COUT del Monitor. La COUT è la routine che si occupa di depositare il contenuto dell'accumulatore nella posizione attuale del cursore, incrementa poi il cursore e controlla i margini dello schermo effettuando eventualmente lo scrolling del video. Il suo indirizzo è FDED (vedi figura 1 della scorsa puntata).

3) A questo punto occorre incrementare di uno l'accumulatore; purtroppo non esiste nel 6502 una istruzione che permetta di farlo così come per la INX o la INC. Siamo costretti ad usare l'istruzione di somma! La ADC effettua però la somma di tre dati: il contenuto dell'accumulatore, il dato (immediato o in memoria) e il CARRY. Dal momento che non sappiamo cosa contiene il CARRY è sempre meglio azzerarlo prima di effettuare una somma (mentre dovremo sempre settarlo prima di una sottrazione!). Quindi CLC (clear Carry) poi ADC #$01 (somma 1 ad A).

4) Controlliamo adesso se abbiamo superato la lettera zeta (codice $DA). L'istruzione CMP confronta un dato, che come al solito può essere il contenuto di una locazione di memoria indirizzato in vari modi o proprio il numero che segue l'istruzione, con il contenuto dell'accumulatore. In pratica esegue una sottrazione tra A ed il dato, il risultato va perso ma vengono aggiornati in conseguenza i flag N,Z e C. Se A è < del Dato la sottrazione ha necessitato di prestito dal Carry che quindi va a zero, se A = Dato allora il flag Z passa ad uno per segnalarci il risultato di zero, se A ≥ Dato allora il Carry resta settato (uno). Il programma diventa quindi CMP #$DB e BCC.

5) Fine del programma in caso di Carry Settato: RTS.

In figura 4 trovate il disassemblato del programma che stampa le lettere dell'alfabeto.

La figura 5 mostra un altro modo di ottenere lo stesso risultato senza utilizzare la "pesante" ADC.

In questo caso è stato però necessario utilizzare il registro X che viene incrementato con la comoda INX e trasferito in A con TXA (transfer X -> A) prima del salto alla COUT. Come potete notare dall'indirizzo dell'RTS questo programma è più corto di un Byte ed è anche più veloce, ma impegna un registro in più e questo a volte potrebbe impedirne l'uso.

Conclusioni

In questa puntata abbiamo imparato altre cose interessanti, ma ricordiamo che in questa serie di articoli non dovete cercare soluzioni di particolari problemi quanto piuttosto metodologie di lavoro.

Si tratta di apprendere un linguaggio nuovo e di trovare altre soluzioni a vecchi problemi.

Come avrete visto, in quasi tutti gli esempi non esiste un solo modo di fare le cose e non ne esiste, o non sempre, nemmeno uno che sia più o meno buono di un altro, tutto dipende dal bisogno del momento.

Per esempio i programmi più veloci occupano in genere più memoria degli altri mentre un programma ottimizzato in termini di spazio richiede spesso molte più risorse esterne e impiega in esecuzione molto più tempo.

Un tipico esempio lo presenteremo la prossima volta con un programma che pulisce lo schermo in alta risoluzione in soli 42 millesimi di secondo.

Intanto continuiamo ad invitarvi a non limitarvi a copiare gli esempi riportati ma a provare a modificarli o adattarli ad altri usi magari banali ma che vi possano servire nel vostro futuro di programmatori in linguaggio macchina. MC

Ti occorre un personal computer o un sistema multiterminale?
Se vuoi l'uno senza rinunciare all'altro...

Grappolo

IL MULTI PERSONAL

Con Grappolo puoi iniziare con un personal, tutto tuo, per arrivare al Multipersonal con otto posti di lavoro indipendenti, ciascuno con 64K di memoria e unità centrale proprie, collegati via bus veloce ad una base dati comune. Con Grappolo è già disponibile una vasta biblioteca di programmi pronti all'uso, CP/M compatibili!
Grappolo, l'efficienza di un sistema distribuito con l'individualità del personal computer. Grappolo, il Multipersonal, costruito e garantito in Italia dalla lunga esperienza SAICO.

saico

SOCIETÀ AZIONARIA ITALIANA COMPUTERS

20121 MILANO - Via S. Giovanni sul Muro, 1 - Tel. (02) 3452116 • 00199 ROMA - Via Asmara, 58 - Tel. (06) 8310063 • 80146 NAPOLI - Via Ferrante Imparato, 35 - Tel. (081) 7523744 • 95123 CATANIA - Via A. De Cosmi, 5 - Tel. (095) 326356