IMPARIAMO A PROGRAMMARE IN assembler

di Valter Di Dio

Ottava e ultima parte

Con la scorsa puntata sulle routine aritmetiche abbiamo praticamente concluso il discorso sulla programmazione in linguaggio macchina del 6502. Tutto ciò che abbiamo visto ci permette infatti di scrivere i nostri programmi senza problemi. Restano però aperte alcune questioni di programmazione "superiore"; ovvero alcuni problemi di gestione a livello Hardware delle periferiche della nostra macchina. Per fare un esempio se decidiamo che il nostro computer debba emettere una nota con una certa frequenza, dobbiamo sapere quanto impiega ad eseguire le istruzioni per poter calcolare esattamente i cicli di ritardo. Altra questione rimasta aperta è la funzione (e l'uso) dell'interrupt, che, con la diffusione dei VIC 20 e Commodore 64 che ne fanno un uso indiscriminato, non può più essere ignorata.

Cominciamo dal RESET

Appena si accende un qualsiasi computer, una apposita circuiteria (in genere un condensatorino) effettua il Reset del 6502; questo si ottiene semplicemente mettendo a massa per un istante il pin 40 del 6502 (vedi figura 1). Quando ciò si verifica il microprocessore effettua un JMP automatico alla routine il cui indirizzo si trova in $FFFC e $FFFD, naturalmente a queste due locazioni deve corrispondere una ROM altrimenti all'accensione non si potrà far sapere al 6502 dove andare a cercare la routine di reset. In quasi tutte le macchine recenti la routine di reset è vettorizzata, nel senso che è in genere possibile cambiarla con una nostra sostituendo alcuni puntatori in RAM. Operazione comunque abbastanza delicata per cui occorre un'ottima conoscenza del sistema operativo della macchina su cui si interviene. Un errore nella routine di reset blocca completamente la macchina e non si riesuma se non spegnendo e poi riaccendendo il tutto.

Gli Interrupt

Ai pin 4 e 6 del 6502 fanno capo gli interrupt. Il loro modo di comportarsi è molto simile al Reset solo che consentono un controllo maggiore da parte del programmatore. Gli Interrupt del 6502 sono due: il NMI (non mascherabile interrupt) e l'IRQ (richiesta di Interrupt) che, come dice il nome, può anche venire ignorata.

L'Interrupt non mascherabile si differenzia dal Reset solo per il fatto che l'istruzione in corso (o il gruppo di istruzioni che potrebbero ingenerare confusione) vengono completate, viene poi effettuato il salvataggio automatico sullo STACK del Program Counter e dello Status e viene effettuato un JMP alla routine puntata da $FFFA e $FFFB. Questa routine eseguirà una serie di controlli sulle periferiche per sapere chi ha attivato l'Interrupt e, una volta soddisfatta la richiesta, tornerà al punto in cui era stata interrotta con un RTI (Return from Interrupt). Due considerazioni:la prima è che il Microprocessore non salva tutti i registri interni, sarà quindi cura del programmatore l'eventuale salvataggio e il recupero dei registri che la routine di gestione dell'Interrupt utilizzerà in modo da rientrare nel programma con i valori corretti. Una tipica routine per il salvataggio dei registri è la seguente:

```
SAVEREG PHA   ; salva l'accumulatore
                ; sullo STACK
        TXA   ; trasferisce X in A
        PHA   ; e lo salva
        TYA   ; trasferisce y in A
        PHA   ; e lo salva
        CLI   ; riabilita l'interrupt
        ....  ; e prosegue
```

GND	1	40	$\overline{RES}$
RDY	2	39	Ø2
Ø1	3	38	SO
$\overline{IRQ}$	4	37	Øo
nc	5	36	nc
$\overline{NMI}$	6	35	nc
SYNC	7	34	R/$\overline{W}$
+5v	8	33	DØ
AØ	9	32	D1
A1	10	31	D2
A2	11	30	D3
A3	12	29	D4
A4	13	28	D5
A5	14	27	D6
A6	15	26	D7
A7	16	25	A15
A8	17	24	A14
A9	18	23	A13
A10	19	22	A12
A11	20	21	+5v

Figura 1 - Piedinatura del 6502. Da notare la linea SYNC che indica il prelievo di un codice operativo e il pin SO che serve a settare il flag di Overflow.

In figura 2 trovate il contenuto della parte alta dello STACK dopo un Interrupt e la SAVEREG. Il programma per riprendere i valori dei registri dallo STACK sarà il seguente

```
RESTORE PLA ;preleva X dallo stack
        TAX ;e lo mette a posto
        PLA ;lo stesso
        TAY ;per Y
        PLA ;questo è proprio A
        RTI ;RETURN from Interrupt
```

La seconda è sull'uso dell'NMI che essendo particolarmente brutale viene usato solo per i guasti Hardware e per le interruzioni dell'alimentazione quando indipendentemente dall'operazione in corso, occorre effettuare subito il salvataggio del contenuto della memoria prima che l'alimentazione di emergenza venga meno. Bisogna infatti fare attenzione al fatto che l'interrupt non mascherabile può interrompere anche una operazione di scrittura sul disco e ciò rischia di rendere illeggibile il contenuto dell'intero dischetto.

L'Interrupt Request (IRQ) permette invece il controllo del programmatore prima di eseguire l'interruzione vera e propria. Abbiamo infatti visto che nel registro dello Status uno dei flag prende il nome di Interrupt Disable e può essere settato o azzerato con le istruzioni SEI (Set Interrupt Disable) e CLI (Clear Interrupt Disable). Nel momento in cui il pin 4 del 6502 viene messo a massa il microprocessore, prima di eseguire il JMP alla locazione indicata in $FFFE e $FFFF, effettua il test del flag I, se viene trovato a uno (ad esempio dopo un SEI) la richiesta di interruzione viene ignorata e il microprocessore prosegue indisturbato il suo lavoro; in caso di Interrupt abilitato viene prima posto ad uno il Flag I (per impedire un nuovo Interrupt prima che sia stata completata la sequenza di SAVEREG) e poi viene eseguito il salto alla routine di gestione dell'IRQ. Questa salverà i registri interni del 6502 e poi passerà a controllare le periferiche per scoprire chi ha richiesto l'interruzione. Appena salvati i registri conviene riabilitare subito gli Interrupt per evitare che una nuova richiesta vada perduta. Con l'IRQ può succedere un'altra cosa:nessuna delle periferiche ha chiesto l'IRQ! Questo perché l'interruzione non era hardware ma software; ovvero il microprocessore ha trovato nel programma che stava eseguendo un Break (BRK). L'istruzione di Break si usa di solito nel debug di programmi in linguaggio macchina per scoprire cosa sta succedendo dentro la CPU. Quando il microprocessore incontra un BRK effettua un IRQ a tutti gli effetti, unica differenza consiste nel flag B che viene forzato a 1. Se si è verificato allora un Interrupt e il flag B è alto allora si tratta di un Break e la routine di gestione dell'IRQ invece di cercare il colpevole tra le periferiche andrà ad eseguire un programmino che visualizza il contenuto dei registri, il program counter e tutto quello che può interessare ai fini del debug di un programma. A questo proposito si deve

fare attenzione al fatto che il Break salva sullo Stack il program counter più uno; questo perché di solito si usa sostituire una istruzione con un Break e la maggior parte delle istruzioni del 6502 è lunga due byte.

Da rilevare che il flag B viene posto a uno prima che il registro P sia spinto sullo Stack; non è quindi possibile testare direttamente il flag B ma bisogna prima recuperare il contenuto del registro dalla cima dello Stack. Il programma per controllare il Flag B è il seguente:

```
PLA ; riprendi lo Status
PHA ; rimettilo a posto
AND#$10 ; maschera del flag B
BNE Break ; salta al programma Break
... ; Interrupt vero!
```

attenzione anche al fatto che l'istruzione BRK mette ad uno il flag I prima di simulare l'Interrupt, non è quindi possibile mascherare un Break.

Un uso particolarmente intensivo dell'IRQ lo fanno i personal della Commodore: sessanta volte al secondo per aggiornare l'orologio interno e per far eseguire al 6502 la scansione della tastiera e l'eventuale prelievo dei caratteri battuti. Questo naturalmente rallenta l'esecuzione di tutti i programmi e complica abbastanza la vita dei programmatori costretti a fare i conti con una CPU che ogni sessantesimo di secondo ti pianta in asso per andare a farsi i fatti suoi; tanto è che nei giochi "commerciali" si usa disabilitare l'Interrupt e gestire la tastiera solo nei momenti in cui si aspetta un dato.

I cicli e i tempi macchina

A volte, per scopi particolari o per mera curiosità, è utile sapere quanto tempo impiegherà il microprocessore ad eseguire una certa routine. Il tempo di esecuzione dipenderà naturalmente dalla frequenza di Clock del computer e dal numero e tipo delle istruzioni che la CPU deve eseguire. Alcune istruzioni possono poi impiegare più o meno tempo a seconda di certe condizioni o addirittura della posizione in memoria in cui si trovano.

Il Clock di sistema delle macchine basate sul 6502 è di solito di 1023 kHz questo valore abbastanza originale deriva dal fatto che è facile ricavare da questa frequenza i segnali di sincronismo del generatore video. A questa frequenza, approssimabile ad un megahertz, un ciclo corrisponde ad un microsecondo circa.

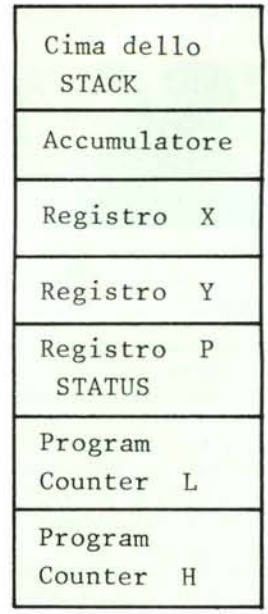


Figura 2 Situazione dello Stack dopo un interrupt e il successivo salvataggio dei registri.

Nella tabella 1 trovate per ciascuna istruzione i cicli macchina necessari all'esecuzione e i flag che l'istruzione stessa modifica.

Per i cicli si deve considerare che un Branch effettuato (ipotesi vera) impiega un ciclo in più di uno senza diramazione e che occorre aumentare di un ciclo tutti i tempi con un asterisco a fianco se si attraversa il confine di pagina. Quindi un BCC che si trova in $3F0 e che, se verificato, salta a $430, impiega due cicli in più di quanto indicato in tabella; uno perché si verifica la diramazione e uno perché dalla pagina $3 si passa alla pagina $4.

Conclusioni

Con questo abbiamo concluso questa serie di articoli sulla programmazione in Assembler. Per molti di voi si sarà trattato di un ripasso per altri sarà stata una vera faticaccia capirci qualcosa; probabilmente per loro non era ancora venuto il tempo dell'Assembler. Magari un giorno, tornandoci sopra, esclamerete "accidenti era già scritto qui e io ci ho perso una mattinata!". Contiamo comunque che vi sia servito a qualcosa e che proseguirete sulla via della programmazione in Linguaggio Macchina, nella speranza di poter pubblicare un giorno il lavoro di un lettore che qualche mese fa era convinto che SED fosse la Società Editrice Dalmata. MC

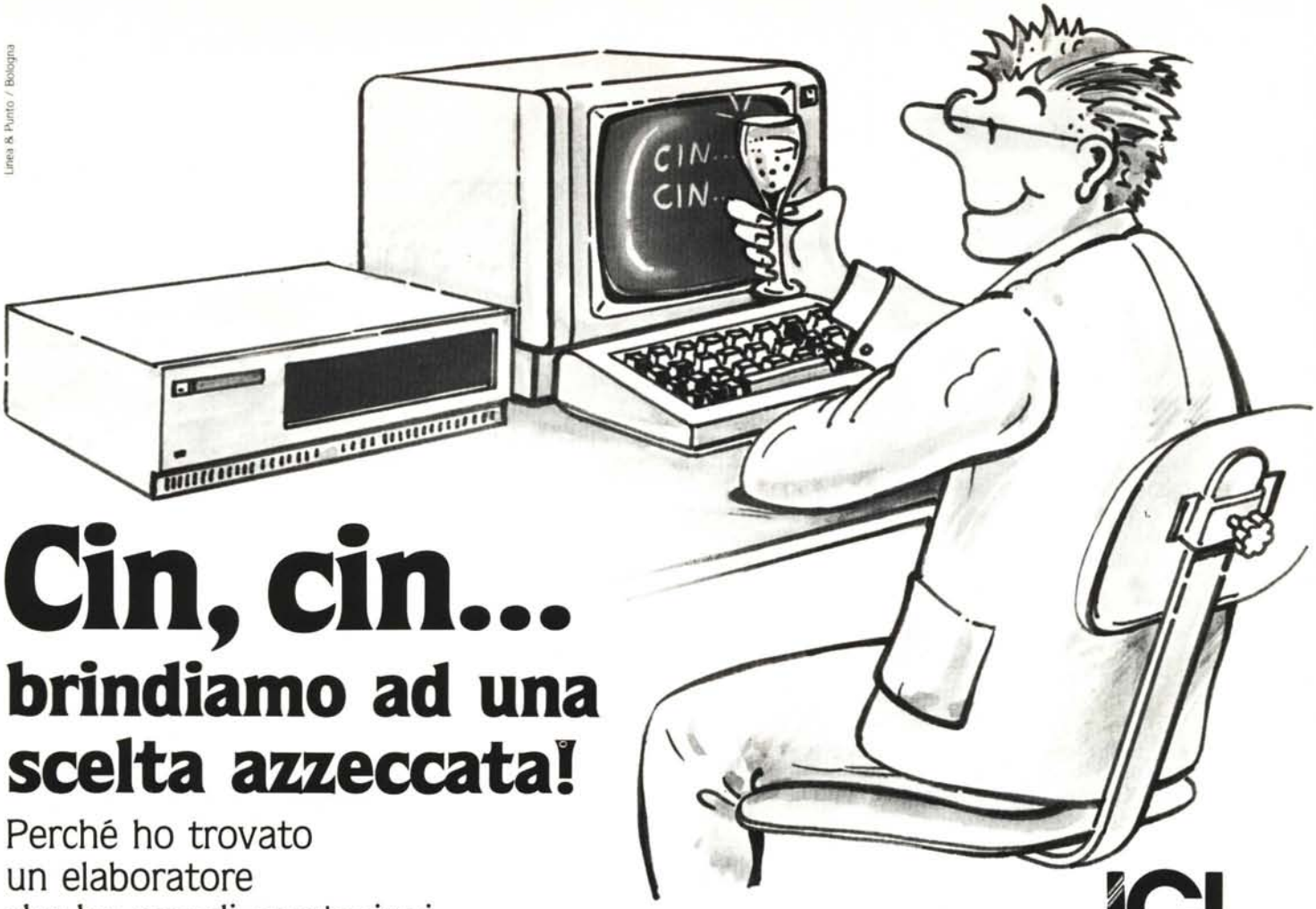
Tabella 1

	implicito	assoluto	pagina 0	immediato	abs. X	abs. Y	(ind.,X)	(ind.),Y	pag.0,X	pag.0,Y	relativo	indiretto
ADC		4	3	2	4*	4*	6	5*	4			
AND		4	3	2	4*	4*	6	5*	4			
ASL	2	6	5		7				6			
BCC											2+	
BCS											2+	
BEQ											2+	
BIT		4	3									
BMI											2+	
BNE											2+	
BPL											2+	
BRK	7											
BVC											2+	
BVS											2+	
CLC	2											
CLD	2											
CLI	2											
CLV	2											
CMP		4	3	2	4*	4*	6	5*	4			
CPX		4	3	2								
CPY		4	3	2								
DEC		6	5		7				6			
DEX	2											
DEY	2											
EOR		4	3	2	4*	4*	6	5*	4			
INC		6	5		7				6			
INX	2											
INY	2											
JMP		3										5
JSR		6										
LDA		4	3	2	4*	4*	6	5*	4			
LDX		4	3	2		4*				4		
LDY		4	3	2	4*				4			
LSR	2	6	5		7				6			
NOP	2											
ORA		4	3	2	4*	4*	6	5*	4			
PHA	3											
PHP	3											
PLA	4											
PLP	4											
ROL	2	6	5		7				6			
ROR	2	6	5		7				6			
RTI	6											
RTS	6											
SBC		4	3	2	4*	4*	6	5*	4			
SEC	2											
SED	2											
SEI	2											
STA		4	3		5	5	6	6	4			
STX		4	3							4		
STY		4	3						4			
TAX	2											
TAY	2											
TSX	2											
TXA	2											
TXS	2											
TYA	2											

	FLAG						
	N	V		B	D	I	Z	C
ADC	*	*					*	*
AND	*						*	
ASL	*						*	*
BCC								
BCS								
BEQ								
BIT	M_7	M_6					*	
BMI								
BNE								
BPL								
BRK				+		1		
BVC								
BVS								
CLC								0
CLD					0			
CLI						0		
CLV		0						
CMP	*						*	*
CPX	*						*	*
CPY	*						*	*
DEC	*						*	
DEX	*						*	
DEY	*						*	
EOR	*						*	
INC	*						*	
INX	*						*	
INY	*						*	
JMP								
JSR								
LDA	*						*	
LDX	*						*	
LDY	*						*	
LSR	0						*	*
NOP								
ORA	*						*	
PHA								
PHP								
PLA	*						*	
PLP	*	*		*	*	*	*	*
ROL	*						*	*
ROR	*						*	*
RTI	*	*		*	*	*	*	*
RTS								
SBC	*	*					*	*
SEC								1
SED					1			
SEI						1		
STA								
STX								
STY								
TAX	*						*	
TAY	*						*	
TSX	*						*	
TXA	*			-			*	
TXS								
TYA	*						*	

Le altre puntate di "Impariamo a programmare in Assembler"

MC n°	ARGOMENTO
20	Introduzione
21	Il monitor
22	La pagina zero
23	Miniassembler
24	Ancora indirizzamenti
25	Spostamenti di dati
26	Le Operazioni



Cin, cin...
brindiamo ad una scelta azzeccata!

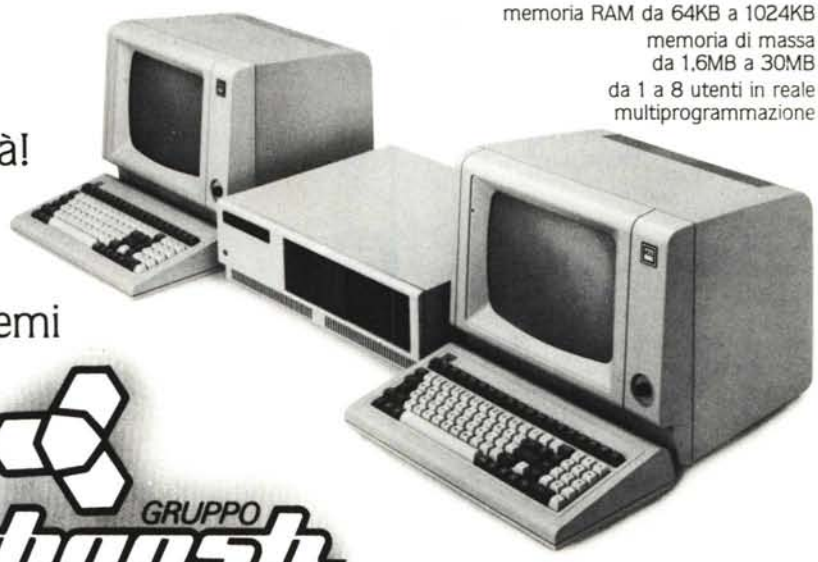
Perché ho trovato
un elaboratore
che ha grandi prestazioni
ed un piccolo prezzo!

Perché il Gruppo BAGSH
mi garantisce programmi
personalizzati di elevata qualità!

Perché le diverse esperienze
di un gruppo di qualificate
aziende ha risolto i miei problemi
ed aumentato i miei profitti!

ICL
trader point

memoria RAM da 64KB a 1024KB
memoria di massa
da 1,6MB a 30MB
da 1 a 8 utenti in reale
multiprogrammazione



GRUPPO **bagsh**

il punto d'incontro delle esperienze più qualificate.

Via Nicolò dell'Arca, 1 - 40129 BOLOGNA - Tel. (051) 35.32.31/37.10.99 (3 linee)

BOLOGNA, BRESCIA, CARPI, CATTOLICA, CESENA, FERRARA, FIRENZE
FOLIGNO, MILANO, MODENA, PADOVA, PARMA, REGGIO EMILIA, TRIESTE