

Small Computer Monitor Tutorial

Monitor versione 1.0 per CPU Z80

Software and Documentation by Stephen C Cousins
Traduzione Michele Dalla Silvestra
Edition 1.0.0

CONTENUTI

PANORAMICA.....	3
PREREQUISITI.....	4
Numeri decimali.....	4
Numeri binari.....	4
Numeri esadecimali.....	5
Perché non usare numeri decimali?.....	5
Bit, Byte, Nibble e Word.....	6
Evitare confusione.....	7
Caratteri ASCII.....	8
CONNETTERSI.....	9
Seriali LiNC80.....	9
Seriali RC2014.....	9
PROGRAMMI TERMINALE.....	11
Tera Term.....	11
Putty.....	12
HyperTerminal.....	12
INSTALLARE SMALL COMPUTER MONITOR.....	13
Scheda LiNC80 SBC1.....	13
Scheda originale RC2014 8k.....	14
Scheda RC2014 con ROM configurabile.....	14
Scheda RC2014 con ROM paginata.....	14
Scheda RC2014 Mini.....	14
ACCENSIONE.....	15
PRIMI COMANDI.....	16
? oppure Help.....	16
Ripristino.....	16
Visualizzazione memoria.....	16
Visualizzazione e modifica dei registri.....	17
Ricezione da periferica.....	17
Invio a periferica.....	18
SCRITTURA DI PROGRAMMI.....	19
Linguaggi di Alto Livello.....	19
Linguaggio Assembly.....	20
IL PRIMO PROGRAMMA.....	21
Uso dell'assemblatore.....	22
Debugging.....	23
Passo singolo.....	26
HELLO WORLD!.....	29
Un programma migliore.....	33
REGISTRI E FLAG.....	35

ALTRI PROGRAMMI DI ESEMPIO.....	37
<i>Mostrare una nuova riga</i>	37
<i>Ingresso da tastiera</i>	38
<i>Lampeggio di un LED</i>	38
<i>Miglioriamo il lampeggio!</i>	40
<i>Altri modi di divertirsi con I LED</i>	44
ASSEMBLATORI.....	47
TROVARE ERRORI.....	48
<i>LiNC80</i>	48
<i>RC2014</i>	48
PARTI E FORNITORI.....	49
<i>Moduli ufficiali LiNC80</i>	49
<i>Moduli ufficiali RC2014</i>	49
<i>Programmatori di chip</i>	49
<i>EEPROM 8k x 8 bit</i>	49
<i>EEPROM 32k x 8 bit</i>	49
<i>FTDI cable</i>	49
<i>FTDI 'cable'</i>	50
<i>USB-RS232 cable</i>	50
INFORMAZIONI DI CONTATTO.....	51
<i>LiNC80</i>	51
<i>RC2014</i>	51

Panoramica

Small Computer Monitor è un software di monitoraggio macchina che consente il debug di programmi e l'interazione generare con hardware e software. Può anche essere usato come ROM di avvio, evitando la necessità di altri software nel sistema.

Questo tutorial è pensato per persone con poca o nessuna esperienza di monitoraggio codice e programmazione Z80. Se hai già esperienza di queste cose, la Small Computer Monitor User Guide dovrebbe essere tutto quello che ti serve.

Questo tutorial presume che tu stia usando un sistema Z80 basato su LiNC80 oppure RC2014. Si presume anche che il sistema sia operativo con la ROM fornita dal produttore e la configurazione di lavoro sia ben conosciuta.

A questo punto il sistema dovrebbe essere collegato a un terminale o a un PC (o simile) che sta eseguendo un software di emulazione terminale e il terminale dovrebbe funzionare correttamente.

Se il sistema non viene fornito con Small Computer Monitor nella ROM, la prima cosa da fare è installare una ROM contenente il programma Monitor e configurare l'hardware per l'avvio da questa ROM. Consultare il documento Small Computer Monitor Installation per i dettagli completi.

Successivamente è possibile inserire ed eseguire programmi in codice macchina, testare ed eseguire il debug dei programmi e in generale sperimentare l'hardware.

Alcune configurazioni includono anche il BASIC e un caricatore CP/M nella ROM, in modo che sia possibile avviarli con un semplice comando del monitor.

Questo tutorial si concentra sull'utilizzo di Small Computer Monitor, anziché essere una guida completa alla programmazione dello Z80. Tuttavia, per utilizzare il monitor è necessaria una certa conoscenza della programmazione Z80. Pertanto in questo documento viene spiegata la programmazione di base. Questo dovrebbe essere sufficiente per iniziare.

Una volta compresi Small Computer Monitor e le basi della programmazione Z80, su Internet sono disponibili molte risorse adatte che possono aiutare a progredire ulteriormente.

Prerequisiti

Prima di immergersi nell'uso del programma monitor, ci sono alcune cose con cui si deve avere familiarità. Queste sono:

- numeri Binari
- numeri Esadecimali (Hex)
- caratteri ASCII

Le prossime pagine forniscono una breve introduzione a questi argomenti, ma in caso di dubbi si prega di cercare ulteriori informazioni prima di continuare.

Numeri decimali

I numeri decimali usati durante la vita quotidiana sono espressi in base dieci. Questo significa che si usano dieci simboli distinti (0, 1, 2, 3, 4, 5, 6, 7, 8 e 9) chiamati cifre.

Per indicare numeri superiori a 9 si usano più cifre. Usando due cifre si possono rappresentare 100 numeri differenti (0, 1, 2, ... , 97, 98, 99).

La cifra meno significativa è a destra, con la cifra immediatamente a sinistra che rappresenta dieci volte (si ricorda la base dieci) la cifra alla sua destra. E così via.

In questo modo, il numero decimale 12 significa $(1 \times 10) + (2 \times 1)$, e il numero 123 significa $(1 \times 100) + (2 \times 10) + (3 \times 1)$.

Numeri binari

I numeri binari seguono lo stesso schema dei numeri decimali, ma in base due. Ciò significa che si utilizzano due simboli distinti (0 e 1).

Per esprimere numeri maggiori di 1 decimale, si utilizza più di una di queste cifre. L'uso di due cifre ci permette di rappresentare 4 numeri diversi (0, 1, 2 e 3).

La cifra meno significativa è a destra, con la cifra alla sua sinistra che rappresenta due volte (si ricorda la base due) il valore della cifra alla sua destra. E così via.

Pertanto il numero binario 10 significa decimale $(1 \times 2) + (0 \times 1) = 2$, e il numero binario 101 significa decimale $(1 \times 4) + (0 \times 2) + (1 \times 1) = 5$.

Numeri esadecimali

I numeri esadecimali (in breve Hex) seguono lo stesso schema dei numeri decimali, ma sono in base sedici. Ciò significa che si utilizzano sedici simboli distinti (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E e F).

Le lettere dalla A alla F rappresentano rispettivamente i numeri decimali da 10 a 15. Quindi A=10, B=11, C=12, D=13, E=14 e F=15.

Per esprimere numeri maggiori di 15 decimali si utilizzano più di una di queste cifre. Utilizzando due cifre si possono rappresentare 256 numeri diversi (0, 1, 2, ... , 253, 254, 255).

La cifra meno significativa è a destra, con la cifra alla sua sinistra che rappresenta sedici volte (si ricorda la base sedici) il valore della cifra alla sua destra. E così via.

Pertanto il numero esadecimale 1F significa decimale $(1 \times 16) + (15 \times 1) = 31$, e il numero decimale 1F2 significa $(1 \times 256) + (15 \times 16) + (2 \times 1) = 498$.

Perché non usare numeri decimali?

La ragione per cui tutto ciò è importante è che il nostro normale sistema di numerazione decimale in base dieci non si adatta bene ai computer digitali.

I computer sono costituiti da elettronica digitale. Digitali significa che ciascuno dei componenti più piccoli che compongono il computer può trovarsi solo in uno dei due stati, acceso o spento (o uno o zero).

La notazione binaria è quindi il modo più semplice per descrivere lo stato di qualsiasi parte, o gruppo di parti, in un computer.

I microprocessori a otto bit, come lo Z80, elaborano otto bit contemporaneamente. Per descrivere qualsiasi combinazione di questi otto bit sono necessarie otto cifre binarie. Ciò fornisce l'intervallo di numeri binari da 00000000 a 11111111 (o decimale da 0 a 255).

Ora noi semplici esseri umani abbiamo un po' di difficoltà a lavorare con numeri come 11010111, quindi abbiamo bisogno di una notazione più comoda. È qui che entra in gioco l'esadecimale.

Una cifra esadecimale ha sedici possibili valori, che risultano essere gli stessi di quattro bit binari. Sono quindi equivalenti:

- Binario 0000 = Esadecimale 0
- Binario 0001 = Esadecimale 1

- Binario 0010 = Esadecimale 2
- Binario 0011 = Esadecimale 3
- Binario 0100 = Esadecimale 4
- Binario 0101 = Esadecimale 5
- Binario 0110 = Esadecimale 6
- Binario 0111 = Esadecimale 7
- Binario 1000 = Esadecimale 8
- Binario 1001 = Esadecimale 9
- Binario 1010 = Esadecimale A
- Binario 1011 = Esadecimale B
- Binario 1100 = Esadecimale C
- Binario 1101 = Esadecimale D
- Binario 1110 = Esadecimale E
- Binario 1111 = Esadecimale F

Utilizzando due cifre esadecimali si possono descrivere tutte le 8 cifre binarie. Ciò consente di esprimere l'esempio precedente (binario 11010111) come esadecimale D7. È molto più semplice per noi gestire "D7" invece di "11010111", e quindi quando programiamo a livello "binario" tendiamo a usare la notazione esadecimale.

Allora perché non decimale? Poiché una singola cifra esadecimale può rappresentare esattamente quattro cifre binarie e due cifre esadecimali possono rappresentare esattamente otto cifre binarie, esiste una relazione semplice tra il binario e l'esadecimale. Ciò rende i numeri esadecimali una scelta naturale quando si lavora al livello più basso in cui siamo preoccupati di ciò che stanno facendo i singoli bit digitali. La stessa relazione non si applica alla notazione decimale, dove quattro cifre binarie richiedono una o due cifre decimali a seconda del valore delle cifre binarie. Questo tende semplicemente ad essere disordinato e scomodo.

Bit, Byte, Nibble e Word

Dato che tutte queste diverse quantità venivano utilizzate regolarmente, dovevano avere dei nomi. Un Bit è una singola quantità digitale, che può essere descritta in binario da una cifra (0 o 1). Un Nibble è un gruppo di quattro cifre binarie, che può essere rappresentato da una cifra esadecimale (da 0 a F). Un byte è un gruppo di otto cifre binarie, che può essere rappresentato da due cifre esadecimali. Un Word è un gruppo di sedici cifre binarie, che può essere rappresentato da quattro cifre esadecimali.

Il processore Z80 gestisce 8 bit di dati contemporaneamente, ovvero un byte, e utilizza un indirizzo di memoria a 16 bit, ovvero un word.

Evitare confusione

Si potrebbe aver notato che la maggior parte delle volte in cui un numero veniva menzionato era necessario prefissarlo con la parola decimale, binaria o esadecimale. In mancanza, come si fa a sapere se il numero "10" è 10 decimale, 10 binario (decimale 2) o 10 esadecimale (decimale 16)? Spesso si può indovinare in base al contesto, ma non si può esserne sicuri.

Per evitare questa confusione e per evitare di dover scrivere continuamente "decimale", "binario" o "esadecimale" prima di ogni numero, nell'industria informatica sono stati utilizzati diversi sistemi di notazione. Purtroppo non tutti usano lo stesso sistema! I vari sistemi utilizzano un prefisso o un suffisso per identificare la base numerica. Ad esempio, mettendo "\$" prima di un numero (es. \$10) indica che il numero è espresso in esadecimale.

Alcuni identificatori comunemente utilizzati sono:

Identificatore	Base	Esempio
\$	Esadecimale	\$1FC
0x	Esadecimale	0x1FC
H	Esadecimale	1FCH
0b	Binario	0b101
%	Binario	%101
B	Binario	101B
+	Decimale	+123
D	Decimale	123D

Poiché Small Computer Monitor viene utilizzato principalmente per manipolare direttamente la memoria, i registri e le porte, che sono di natura digitale, l'impostazione predefinita per tutti i numeri è esadecimale. Pertanto i numeri esadecimali di solito non richiedono identificatori speciali, ma ci sono alcune eccezioni.

Il disassemblatore mostra le costanti come numeri esadecimali preceduti da "\$". Il prefisso è qui necessario per chiarezza perché anche i registri B, C, D ed E del processore sono numeri esadecimali validi. L'assemblatore accetta numeri esadecimali senza alcun identificatore. Per specificare un numero decimale è necessario precederlo dal simbolo "+". Sono accettati anche i numeri esadecimali con il prefisso "\$" o "0x".

All'interno di questa guida il prefisso "\$" viene normalmente utilizzato laddove sono richiesti chiarimenti.

Caratteri ASCII

Nei computer digitali tutti i caratteri dell'alfabeto e tutti gli altri simboli sono memorizzati come numeri. Lo standard di allocazione dei caratteri utilizzato dal Small Computer Monitor è l'American Standard Code for Information Interchange (ASCII).

Le tabelle seguenti mostrano il sottoinsieme di ASCII utilizzato dal Small Computer Monitor.

ASCII	Hex.	Char.	ASCII	Hex.	Char.	ASCII	Hex.	Char.	ASCII	Hex.	Char.
0	00	NUL	16	10		32	20	(spc)	48	30	0
1	01		17	11		33	21	!	49	31	1
2	02		18	12		34	22	"	50	32	2
3	03		19	13		35	23	#	51	33	3
4	04		20	14		36	24	\$	52	34	4
5	05		21	15		37	25	%	53	35	5
6	06		22	16		38	26	&	54	36	6
7	07		23	17		39	27	'	55	37	7
8	08	BS	24	18		40	28	(	56	38	8
9	09		25	19		41	29	)	57	39	9
10	0A	LF	26	1A		42	2A	*	58	3A	:
11	0B		27	1B	ESC	43	2B	ES+C	59	3B	;
12	0C		28	1C		44	2C	,	60	3C	<
13	0D	CR	29	1D		45	2D	-	61	3D	=
14	0E		30	1E		46	2E	.	62	3E	>
15	0F		31	1F		47	2F	/	63	3F	?

ASCII	Hex.	Char.	ASCII	Hex.	Char.	ASCII	Hex.	Char.	ASCII	Hex.	Char.
64	40	@	80	50	P	96	60	`	112	70	p
65	41	A	81	51	Q	97	61	a	113	71	q
66	42	B	82	52	R	98	62	b	114	72	r
67	43	C	83	53	S	99	63	c	115	73	s
68	44	D	84	54	T	100	64	d	116	74	t
69	45	E	85	55	U	101	65	e	117	75	u
70	46	F	86	56	V	102	66	f	118	76	v
71	47	G	87	57	W	103	67	g	119	77	w
72	48	H	88	58	X	104	68	h	120	78	x
73	49	I	89	59	Y	105	69	i	121	79	y
74	4A	J	90	5A	Z	106	6A	j	122	7A	z
75	4B	K	91	5B	[	107	6B	k	123	7B	{
76	4C	L	92	5C	\	108	6C	l	124	7C	
77	4D	M	93	5D	]	109	6D	m	125	7D	}
78	4E	N	94	5E	^	110	6E	n	126	7E	~
79	4F	O	95	5F	_	111	6F	o	127	7F	

Quindi, per fare riferimento alla lettera "A" quando si lavora sul monitor, di solito è necessario utilizzare il valore decimale 65 o il valore esadecimale 41.

Connettersi

Se il tuo sistema è già attivo e funzionante con una ROM fornita dal produttore, puoi saltare questa sezione.

Il sistema (LiNC80 o RC2014) deve essere collegato a un terminale o a un computer che esegue il software di emulazione terminale. Il terminale fornisce l'input da tastiera e l'output visualizzato.

La connessione fisica è un cavo seriale dal sistema di destinazione al terminale.

Seriali LiNC80

LiNC80 contiene due porte seriali, una RS232 (port A) e una TTL 5 volt (port B). Port B è la porta predefinita per Small Computer Monitor.

Dei ponticelli permettono la selezione della fonte del clock e il segnale CTS. Small Computer Monitor presume che i ponticelli siano impostati ai valori predefiniti del produttore, quindi port A usa il CTC come fonte di clock e velocità 9600 baud, mentre port B usa il clock del processore e lavora a 115200 baud.

Per usare l'interfaccia RS232 è necessario un cavo da DTK/Intel a DB9. Sarà inoltre necessario un cavo null modem per collegare il connettore di tipo D a 9 pin del cavo adattatore a quello del terminale (o del computer).

Per utilizzare l'interfaccia TTL da 5 volt (stile FTDI) è necessario un cavo USB-seriale FTDI (o compatibile). Ce ne sono di vario tipo. Assicurarsi che il cavo sia progettato per segnali a 5 volt.

Consultare la sezione "Parti e fornitori" per i dettagli sui 'cavi' compatibili noti.

Seriali RC2014

La porta seriale RC2014 può essere il modulo I/O seriale ufficiale, il modulo seriale doppio ufficiale (SIO/2), la porta seriale dell'RC2014 Mini o moduli SIO/2 non ufficiali che seguono l'indirizzamento del registro di Grant Searle (ad es. SIO del Dr Scott M Baker /2 modulo). Sono disponibili anche altri moduli che potrebbero funzionare.

Il modulo I/O seriale e RC2014 Mini utilizzano un chip MC68B50 o ACIA (Asynchronous Communications Interface Adapter) compatibile. Il modulo I/O seriale ha l'opzione di un'interfaccia seriale in stile FTDI da 5 volt o

un'interfaccia seriale RS232 di tipo D a 9 pin, mentre il Mini ha solo l'interfaccia in stile FTDI da 5 volt.

Il modulo seriale doppio utilizza un chip Zilog Z80 SIO/2 e fornisce due porte seriali. Il monitor per piccoli computer richiede solo la porta A. Questo modulo fornisce solo l'interfaccia seriale in stile FTDI da 5 volt, non la tradizionale interfaccia seriale RS232.

Per utilizzare l'interfaccia stile RS232 è necessario un cavo null modem per collegare il connettore tipo D a 9 pin della scheda RC2014 a quello del terminale (o computer).

Per utilizzare l'interfaccia stile FTDI da 5 volt è necessario un cavo USB-seriale FTDI (o compatibile). Sono disponibili diversi design. Nota: alcuni utenti RC2014 hanno riscontrato problemi con alcuni cavi con determinati valori di resistori sulle schede RC2014. Assicurati inoltre che il cavo sia progettato per segnali a 5 volt.

Consultare la sezione “Parti e fornitori” per i dettagli sui ‘cavi’ compatibili noti.

Programmi terminale

A meno che non si utilizzi un terminale originale, è necessario installare un programma di emulazione terminale su un PC o simile.

Di seguito vengono descritti in dettaglio diversi programmi di emulazione terminale disponibili gratuitamente, ma in generale queste sono le impostazioni richieste:

Il numero di porta sarà determinato dall'hardware del tuo computer. In ambiente MS Windows, le porte seriali sono normalmente chiamate "COM#", dove "#" è un numero come 3 (es. COM3). Quando si collega un cavo USB-Seriale adatto, sul computer sarà disponibile una porta COM. Se il tuo computer dispone di una porta RS232 legacy, avrà anche un numero di porta COM.

In ambiente GNU/Linux le porte seriali native sono normalmente identificate con /dev/ttyS0, /dev/ttyS1, eccetera, mentre gli adattatori USB-Seriale sono identificate come /dev/ttyUSB0, /dev/ttyUSB1 oppure /dev/ttyACM0 eccetera.

Le impostazioni della porta seriale per Small Computer Monitor sono normalmente:

- Baud rate = 115200
- Data bits = 8
- Parity = None
- Stop bits = 1
- Flow control = None (potrebbe anche essere impostato a Hardware)
- Carattere New line = Carriage Return
- Local echo = Off
- Backspace key sends = Control+H
- Terminal type = VT100 (non critico, basta un terminale semplice)
- Transmission delay(s) = 0

Il software del terminale solitamente dispone di opzioni per aggiungere ritardi ai caratteri inviati dal terminale. Di solito questi possono essere impostati su zero, ma se si verifica una perdita di caratteri è possibile aggiungere un piccolo ritardo.

Tera Term

A meno di avere già una preferenza, il mio suggerimento è usare Tera Term.

Con Tera Term v4.96, usare il menù Setup, voce Terminal e voce Serial ports per configurarla come descritto sopra. Lasciare Answerback vuoto e Auto switch off.

Putty

Putty è un programma molto completo, ma ha molte opzioni complesse.

Con Putty v0.70, utilizzare la finestra di dialogo Configurazione che si apre al caricamento del programma.

Nella pagina Sessione selezionare Tipo di connessione = Seriale.

Seleziona la Categoria = Seriale. Immettere la linea seriale a cui connettersi = COM# (dove # è il numero della porta seriale), velocità = 115200, bit di dati = 8, bit di stop = 1, parità = nessuna, controllo di flusso = nessuno.

Utilizzare la pagina Terminale per assicurarsi che “CR implicito in ogni LF” sia deselezionato, insieme a “LF implicito con ogni CR”. Anche l'eco locale e la modifica della linea locale sono disattivati.

Utilizza la pagina Tastiera per impostare Backspace su Control+H.

Ritorna alla pagina Sessione, seleziona Impostazioni predefinite e fai clic su Salva. Quindi fare clic su Apri.

Una volta aperta la finestra del terminale è possibile tornare alle opzioni di configurazione facendo clic sull'icona del terminale in alto a sinistra della finestra e selezionando Modifica impostazioni.

HyperTerminal

Per gli utenti di installazioni Windows precedenti, come XP, probabilmente è già installato HyperTerminal. Per HyperTerminal v5.1, utilizzare il menu File, voce Proprietà per configurare le impostazioni come descritto sopra.

Installare Small Computer Monitor

Su alcuni sistemi è possibile eseguire Small Computer Monitor caricandolo nella RAM, ma in questo caso si presuppone che lo si stia eseguendo da un chip ROM (Read Only Memory).

Per programmare il chip, si può trovare il programma fornito come file Intel Hex adatto per l'apertura con la maggior parte dei prodotti di programmazione dei chip.

Il metodo di programmazione dipende dal programmatore che si sta utilizzando e va oltre lo scopo di questo tutorial. Vedere la sezione "Parti e fornitori" per i dettagli sui programmatori compatibili noti.

Esistono numerose configurazioni del monitor per piccoli computer. Il Monitor da solo occupa 8k byte, ma può includere BASIC e un caricatore CP/M in 16k byte.

Il monitor per piccoli computer, da solo, può trovarsi in un chip ROM con 8k byte per 8 bit di memoria (spesso descritto come memoria da 64k bit) o in parte di un chip più grande. Ogni chip utilizzato dal programma Monitor deve essere posizionato nella mappa di memoria dello Z80 a partire dall'indirizzo \$0000.

Un chip adatto, per il solo Monitor, è l'AT28C64B-15PU. Questo è un chip molto comodo se si vuole riprogrammarlo di nuovo in seguito, poiché è una memoria di sola lettura programmabile e cancellabile elettricamente (EEPROM). Può essere cancellato e riprogrammato in pochi secondi con un moderno programmatore economico come il MiniPro TL866.

Le esatte istruzioni di montaggio dipendono dal chip ROM e dal circuito stampato a cui è collegato. Di seguito viene dettagliato l'utilizzo dell'AT28C64B-15PU nelle schede più comuni.

Per Small Computer Monitor è disponibile anche una guida all'installazione più approfondita, che descrive dettagliatamente le impostazioni dei ponticelli per le combinazioni comuni di schede, chip PROM e configurazione del monitor. Anche le impostazioni dei ponticelli sono tutte mostrate con illustrazioni!

Scheda LiNC80 SBC1

La ROM standard del kit LiNC80 contiene Small Computer Monitor, per cui è sufficiente seguire le istruzioni del produttore per impostare i ponticelli.

Comunque, se si vuole installare un AT28C64B, entrambi i ponticelli della ROM devono essere impostati a "HIGH".

Scheda originale RC2014 8k

Questa scheda ROM accetta solo un chip ROM da 8k byte e non ha ponticelli da configurare. Quindi inserire il chip programmato e proseguire.

Scheda RC2014 con ROM configurabile

Questa scheda ROM può accettare chip di varia capacità e tramite dei ponticelli si può impostare l'indirizzamento della ROM. Usando un chip da 8k byte come AT28C64B-15PU i ponticelli vanno impostati in questo modo:

Page selection: ponticelli A13, A14 e A15 = Vcc (5 volt)

Per chip più grandi, come il 27C512, i ponticelli da A13 ad A15 devono essere impostati per coincidere con la posizione del codice all'interno del chip.

Scheda RC2014 con ROM paginata

Questa è una scheda abbastanza flessibile e ha un numero di ponticelli che devono essere impostati. Se stai utilizzando un chip da 8k byte come l'AT28C64B-15PU i ponticelli dovrebbero essere impostati come segue:

Page size: 8k (ponticelli come indicato dalla legenda indicata sul PCB)

Page selection: A10, A11 e A12 = no ponticelli, A13, A14 e A15 = 1

Per chip più grandi, come il 27C512:

Page size: 8k (ponticelli come indicato dalla legenda indicata sul PCB)

Page selection: A10, A11 e A12 = no ponticelli, A13, A14 e A15 devono corrispondere alla posizione del codice all'interno del chip.

Scheda RC2014 Mini

Questa scheda ROM può accettare chip di varia capacità e tramite dei ponticelli si può impostare l'indirizzamento della ROM. Usando un chip da 8k byte come AT28C64B-15PU i ponticelli vanno impostati in questo modo:

Page selection: ponticelli A13, A14 e A15 = Vcc (5 volt)

Per chip più grandi, come il 27C512:

Page selection: A13, A14 e A15 devono corrispondere alla posizione del codice all'interno del chip.

Accensione

Se tutto va bene, quando si accende il sistema, dovrebbe comparire qualcosa di simile sullo schermo del terminale:

```
Small Computer Monitor  
*
```

Se non si riceve un messaggio di accesso simile a quello sopra, consultare la sezione “Ricerca guasti” che si trova verso la fine di questo documento.

Il carattere “*” visualizzato sul terminale è il prompt del monitor. Ciò indica che il monitor è pronto per accettare un nuovo comando. Provare a digitare il carattere del punto interrogativo e quindi a premere il tasto Invio.

Nel testo seguente, l'input dell'utente è in un carattere grassetto italico, mentre i risultati vengono visualizzati in un carattere normale. La pressione di tasti speciali, come Escape, viene mostrata racchiusa tra parentesi graffe. Pertanto, l'esempio seguente significa che l'utente digita “*b 5000*” sulla tastiera del terminale seguito dalla pressione del tasto Invio e il monitor visualizza “Punto di interruzione impostato” sullo schermo del terminale.

```
b 5000 {invio}  
Breakpoint set
```

Se non diversamente indicato, i parametri sono numeri esadecimali, come FF12. Non è necessario anteporre loro un identificatore esadecimale o un carattere numerico. L'eccezione a questa regola sono gli operandi nell'assembler.

I parametri nominati sono mostrati, in questo documento, racchiusi da “<” e “>”, e ulteriormente racchiusi da “[” e “]” se il parametro è facoltativo.

I nomi dei comandi e tutti i parametri sono delimitati da un carattere spazio.

I comandi del monitor non fanno distinzione tra maiuscole e minuscole, quindi possono essere digitati in maiuscolo o minuscolo o in qualsiasi combinazione di maiuscolo e minuscolo.

Primi comandi

Di seguito sono riportati alcuni semplici comandi del Monitor da provare.

? oppure Help

Sintassi: HELP

Oppure sintassi: ?

Viene visualizzato un elenco dei comandi del monitor insieme alla loro sintassi.

Ad esempio:

help {invio}

Small Computer Monitor by Stephen C Cousins (www.scc.me.uk)

Version 1.0.0 configuration L1 for Z80 based LiNC80 systems

Monitor commands:

A [<address>]	= Assemble		D [<address>]	= Disassemble
M [<address>]	= Memory display		E [<address>]	= Edit memory
R [<name>]	= Registers/edit		F [<name>]	= Flags/edit
B [<address>]	= Breakpoint		S [<address>]	= Single step
I <port>	= Input from port		O <port> <data>	= Output to port
G [<address>]	= Go to program			
BAUD <device>	<rate>		CONSOLE <device>	
FILL <start>	<end> <byte>		API <function>	[<A>] [<DE>]
DEVICES, DIR,	HELP, RESET			

Ripristino

Sintassi: Reset

Questo comando esegue un ripristino del software, simile alla pressione del pulsante di reset.

Non può eseguire un ripristino fisico dell'hardware sull'elettronica, ma esegue lo stesso software del ripristino dell'hardware.

Visualizzazione memoria

Sintassi: M [<indirizzo memoria>]

Viene visualizzato un blocco di memoria, in cui ciascuna riga mostra l'indirizzo di memoria in esadecimale, il contenuto di sedici posizioni di memoria in esadecimale e il contenuto di tali sedici posizioni di memoria in ASCII. I caratteri ASCII non stampabili vengono visualizzati come punti.

Il parametro dell'indirizzo di memoria è facoltativo. Se fornito, la memoria verrà visualizzata a partire dall'indirizzo specificato. In caso contrario, la visualizzazione della memoria inizia dall'ultimo indirizzo referenziato.

Ad esempio:

```
m 1600 {invio}
1600: 7C FF CB CE 21 15 16 CD 93 15 20 08 21 7C FF CB |...!..... .!|..
1610: C6 21 1B 16 C9 DD 15 E6 15 F1 15 A9 15 B2 15 BD .!.....
1620: 15 11 27 16 C3 34 02 5A 38 30 20 62 61 73 65 64 ..'.4.Z80 based
1630: 20 52 43 32 30 31 34 20 73 79 73 74 65 6D 73 05 RC2014 systems.
1640: 00 21 7C FF 11 55 16 CB 46 C4 34 02 11 62 16 CB .!|..U..F.4..b..
1650: 4E C4 34 02 C9 53 65 72 69 61 6C 20 41 43 49 41 N.4..Serial ACIA
1660: 05 00 53 65 72 69 61 6C 20 53 49 4F 2F 32 05 00 ..Serial SI0/2..
1670: FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF .....
```

Premendo {invio} nuovamente, viene visualizzato il successivo blocco di memoria.

Premendo {esc} si esce dalla modalità di visualizzazione della memoria e si torna al prompt del monitor.

In alternativa si può inserire un nuovo comando senza prima tornare al prompt del monitor.

Visualizzazione e modifica dei registri

Sintassi: R [<nome del registro>]

Questo comando può visualizzare i registri correnti del processore o modificare il valore di un registro del processore.

Quando non viene immesso alcun parametro, vengono visualizzati i valori correnti dei registri.

Ad esempio:

```
r {return}
PC:0001 AF:0002 BC:0003 DE:0004 HL:0005 IX:0006 IY:0007 Flags:-----N-
SP:0011 AF'0012 BC'0013 DE'0014 HL'0015 (S)0016 IR:0017 Flags'---H--N-
```

Altro riguardo ai registri più avanti.

Ricezione da periferica

Sintassi: I <indirizzo porta>

L'indirizzo della porta di ingresso specificato viene letto e il risultato visualizzato in formato esadecimale.

Ad esempio:

```
I F0 {invio}  
00
```

Se disponi di un dispositivo di input digitale, all'indirizzo della porta \$00, puoi leggere gli input con il comando:

```
I 0 {invio}
```

I sistemi RC2014 utilizzano in genere la porta \$00, mentre LiNC80 utilizza in genere la porta \$30.

Se il dispositivo I/O è dotato di pulsanti, premerne uno e tenerlo premuto mentre si preme il tasto Invio alla fine del comando di input. Tenendo premuto il pulsante sul bit 0, si dovrebbe vedere questo:

```
I 0 {invio}  
01
```

Invio a periferica

Sintassi: O <indirizzo porta> <byte dati>

Il byte di dati specificato viene scritto nell'indirizzo della porta di uscita specificata.

Ad esempio:

```
O F0 55 {invio}
```

Se disponi di un dispositivo I/O digitale, all'indirizzo della porta \$00, puoi scrivere sulle uscite con il comando:

```
O 0 <byte output> {invio}
```

I sistemi RC2014 utilizzano in genere la porta \$00, mentre LiNC80 utilizza in genere la porta \$30.

Se il dispositivo I/O dispone di LED per l'uscita, questo comando accenderà i LED collegati ai bit 0 e 2.

```
O 0 5 {invio}
```

Il valore 5 viene scritto nelle uscite LED. In binario il numero 5 è 00000101, quindi i bit 0 e 2 sono ON. Ciò fa sì che il LED sul bit 0 e il LED sul bit 2 si accendano.

Scrittura di programmi

I programmi sono costituiti da istruzioni. Al livello più basso, il microprocessore Z80 comprende solo le istruzioni, o codice macchina, che sono una serie di bit binari. Questi bit sono divisi in byte, con una singola istruzione lunga da uno a quattro byte. Ogni byte può essere descritto come un numero decimale da 0 a 255, o un numero esadecimale da 00 a FF. Quindi un programma è solo un elenco di numeri. Non troppo amichevole davvero.

Quindi quali numeri compongono un programma?

Per rispondere a questa domanda si deve prima decidere cosa farà il programma. Dovrebbe essere qualcosa che si potrebbe scrivere e leggere a chiunque in modo che possa capire cosa fa.

Il passaggio successivo consiste nel suddividere l'attività in semplici passaggi, ad esempio un elenco numerato di cose da fare o un diagramma di flusso. Ancora una volta questo dovrebbe essere nel linguaggio di tutti i giorni in modo da poter facilmente far capire a chiunque altro.

Ogni fase dell'attività può quindi essere convertita in istruzioni comprensibili al computer.

Linguaggi di Alto Livello

È possibile utilizzare linguaggi di programmazione di alto livello, in cui le istruzioni sono abbastanza simili ai normali linguaggi scritti. Ciò rende relativamente semplice convertire il progetto in istruzioni comprensibili al computer.

Sui classici computer a 8 bit, il linguaggio di alto livello di solito significava il linguaggio BASIC (Beginner's All-purpose Symbolic Instruction Code). Date le limitazioni di quei primi computer, il BASIC era una buona soluzione. Molte persone scartano il classico BASIC ma con solo poche migliaia di byte di memoria e forse solo un registratore a cassette per l'archiviazione, non era facile fornire un linguaggio di programmazione più desiderabile.

Questo documento è un tutorial per Small Computer Monitor, quindi stiamo parlando di codice macchina, non di linguaggi di programmazione di alto livello. Il codice macchina è veloce ed efficiente, ma non è così facile da scrivere come il codice di un linguaggio di alto livello.

È possibile utilizzare Small Computer Monitor per scrivere istruzioni binarie in codice macchina direttamente nella memoria del computer. Questo è il

vero codice macchina binario hard-core. Ma ci sono modi leggermente più semplici per farlo.

Per evitare di lavorare direttamente in binario abbiamo già stabilito che i numeri esadecimali sono più facili da usare per le persone, ma è comunque solo per i pazzi. Invece le istruzioni sono solitamente scritte in linguaggio Assembly.

Linguaggio Assembly

Il linguaggio assembly è un modo di scrivere istruzioni in codice macchina utilizzando una serie di mnemonici e operandi che sono molto più facili da leggere, scrivere e ricordare rispetto ai numeri esadecimali. Questi mnemonici e operandi vengono convertiti in codice macchina da un assemblatore.

Small Computer Monitor può convertire le istruzioni in linguaggio assembly in codice macchina, ma solo un'istruzione alla volta. Per scrivere programmi di grandi dimensioni è preferibile utilizzare un programma chiamato Assembler per compilare il codice macchina.

Il primo programma

È una tradizione nel mondo della programmazione realizzare il tuo primo programma con la scritta “Hello World!”, ma qui iniziamo con qualcosa di più semplice.

Iniziamo provando a scrivere il carattere “!” alla schermata del terminale.

Come spiegato nella sezione precedente, la prima cosa da fare è definire l'attività. Bene, l'abbiamo fatto nella riga sopra.

Successivamente si deve suddividere l'attività in semplici passaggi. In un programma più ampio sarebbe una fortuna anche solo vedere menzionato un passaggio piccolo quanto l'intero compito, ma poiché si tratta di un tutorial approfondirò il progetto e lo analizzerò completamente.

Questi sono i passaggi necessari per scrivere una lettera sullo schermo del terminale:

1. Memorizzare il valore ASCII in una variabile (un registro del processore)
2. Chiamare la funzione di sistema che scrive un carattere sul terminale
3. Terminare il programma

Passo 1

Il valore ASCII per il carattere “!” è decimale 33 o esadecimale 21. Il processore ha un numero di variabili speciali chiamate registri. Per preparare un carattere per la visualizzazione, il valore ASCII del carattere deve essere memorizzato nel registro A. L'istruzione in linguaggio assembly per fare ciò è:

```
LD    A, $21
```

Questa istruzione può essere letta come “Carica il registro A con il valore esadecimale 21”.

Si noti che l'assembler del Monitor imposta per impostazione predefinita l'operando, in questo caso 21, in formato esadecimale, quindi non è necessario inserire il simbolo del dollaro. È stato mostrato qui solo per chiarezza.

Passo 2

Small Computer Monitor fornisce una serie di funzioni per aiutare i programmi a eseguire attività comuni. Questa è chiamata API (Application Programming Interface). Per utilizzare l'API per visualizzare un carattere è necessario memorizzare il valore 2 nel registro C e quindi chiamare l'API. Le istruzioni in linguaggio assembly per fare ciò sono:

```
LD    C, 2  
CALL  $30
```

La subroutine API inizia all'indirizzo esadecimale 30. Il numero della funzione API per inviare un carattere al terminale è 2.

Passo 3

Per terminare un programma si utilizza l'istruzione Return:

```
RET
```

L'intero programma scritto in linguaggio assembly è:

```
LD    A, $21
LD    C, 2
CALL $30
RET
```

L'assemblatore converte queste istruzioni in linguaggio assembly in istruzioni in codice macchina binario. Espresso in esadecimale diventa:

```
3E 21 0E 02 CD 30 00 C9
```

Il processo di conversione è chiamato assemblaggio. Quando viene assemblato in memoria a partire dall'indirizzo esadecimale 8000, l'elenco di output dell'assemblatore sarà solitamente simile a questo:

```
8000: 3E 21      LD    A, $21
8002: 0E 02      LD    C, 2
8004: CD 30 00  CALL $30
8007: C9         RET
```

Uso dell'assemblatore

Per inserire questo programma in memoria utilizzando Small Computer Monitor si utilizza il comando Assemble (A). Per assemblare il codice macchina all'indirizzo esadecimale 8000 si digita il comando:

```
A 8000 {invio}
```

Il monitor mostra quindi l'istruzione corrente all'indirizzo 8000, che assomiglia a questa:

```
8000: 00          .      NOP          >
```

In questo caso l'istruzione attualmente in memoria all'indirizzo 8000 è NOP, ma a questo punto potrebbe essere qualsiasi cosa. Puoi quindi digitare l'istruzione che desideri assemblare lì:

```
8000: 00          .      NOP          > LD A,21 {invio}
```

L'assembler del Monitor utilizza per impostazione predefinita il formato esadecimale, quindi non è necessario inserire il simbolo del dollaro prima del numero 21.

Il programma monitor visualizza quindi l'istruzione immessa nello stesso formato elenco, seguita dall'istruzione successiva attualmente presente in memoria:

```
8000: 3E 21      >! LD A,$21
8002: 00         .   NOP                >
```

È quindi possibile inserire l'istruzione successiva: LD C,2

Proseguire così fino all'inserimento dell'intero programma.

Per uscire dall'assemblatore premere il tasto Esc.

Si può verificare che l'intero programma sia corretto elencando il programma con il comando Disassembla (D):

D 8000 {invio}

Dovrebbe quindi essere visualizzato quanto segue:

```
8000: 3E 21      >! LD A,$21
8002: 0E 02      .. LD C,$02
8004: CD 30 00  .0. CALL $0030
8007: C9         .   RET
```

Premere il tasto Esc per terminare l'elenco dei programmi o il tasto Invio per visualizzare le istruzioni successive.

I caratteri dopo i byte del codice macchina sono i caratteri ASCII per ciascuno di questi byte. Pertanto la prima istruzione 3E 21, mostra ">!".

Ora che abbiamo in memoria il programma in codice macchina possiamo eseguirlo. Per questo usiamo il comando Go (G):

G 8000 {invio}
!*

L'output, illustrato sopra, è il carattere "!" visualizzato dal programma seguito dal carattere di richiesta del monitor "*".

Debugging

Si spera che il programma semplice scritto qui sopra funzioni senza problemi. Ma se così non fosse?

A volte si può semplicemente guardare un programma per anni e non capire perché ha fallito. In questi casi è necessario utilizzare gli strumenti di debug integrati in Small Computer Monitor.

Ci sono due strumenti a disposizione:

- Breakpoint (punto di interruzione)
- Single step (passo singolo)

Un punto di interruzione consente di arrestare il programma ad un indirizzo specificato e di visualizzare lo stato del processore. Usando il programma sopra come esempio, provare questo:

```
B 8004 {invio}  
Breakpoint set
```

```
G 8000 {invio}
```

Ora si dovrebbe vedere qualcosa di simile a questo:

```
Breakpoint  
PC:8004 AF:21B7 BC:6902 DE:BDBA HL:BCDE IX:5B5A IY:AEAB Flags:S--H-PNC
```

Quanto sopra indica che il programma si è fermato al punto di interruzione e mostra i registri e i flag del processore in quell'istante.

Nota: alcuni valori saranno probabilmente diversi da quelli illustrati sopra. Tuttavia, quelli utilizzati dal programma di esempio dovrebbero essere quelli specificati di seguito.

In questo caso i registri significativi sono questi:

PC	Questo è il contatore del programma. Questo dice a che punto del programma si trova il processore. In realtà è l'indirizzo della prossima istruzione da eseguire. In questo caso è 8004.
AF	AF è una coppia di registri, in questo caso contenente 21B7. Le prime due cifre di questo valore sono il registro A, mentre le seconde due sono il registro F. Quindi A è 21.
BC	BC è una coppia di registri, quindi C è 02.

Le coppie di registri verranno spiegate più dettagliatamente in seguito.

Questo è il nostro programma e dove si è fermato.

```
8000: 3E 21      >!   LD A,$21  
  
8002: 0E 02      ..   LD C,$02  
  
8004: CD 30 00   .0.  CALL $0030  
  
8007: C9        .    RET
```

Fermati qui!

Appena prima di fermarsi al punto di interruzione, il programma ha eseguito le istruzioni LD A,\$21 e LD C,\$02. Pertanto A contiene \$21 (il valore ASCII di "!") e C contiene \$02 (il numero di chiamata API).

Se si desidera continuare il programma da quel punto, inserire semplicemente:

```
G {invio}
```

Il programma dovrebbe quindi completarsi come prima e viene visualizzato il carattere "!".

Il comando Go senza un indirizzo specificato fa sì che il programma venga eseguito a partire dall'indirizzo visualizzato nel PC del display del registro. In questo caso 8004.

In alternativa, è possibile modificare i valori del registro prima di continuare il programma con il comando Go.

Ripeti il processo sopra descritto:

```
B 8004 {invio}
```

Breakpoint set

```
G 8000 {invio}
```

Breakpoint

PC:8004 AF:21B7 BC:6902 DE:BDBA HL:BCDE IX:5B5A IY:AEAB Flags:S--H-PNC

Per cambiare il carattere da stampare dobbiamo modificare il registro A. Per fare ciò inseriamo:

```
R A {invio}
```

A: 21 _

Il comando R seguito dal nome del registro fa sì che venga visualizzato il contenuto del registro specificato. In questo caso il valore 21. Il cursore di immissione dovrebbe trovarsi alla destra del valore corrente invitandoti a inserire un nuovo valore. Prova a inserire il valore 41, che è il valore ASCII della lettera "A".

Per verificare che il valore del registro è stato modificato, inserire:

```
R {invio}
```

PC:8004 AF:41B7 BC:6902 DE:BDBA HL:BCDE IX:5B5A IY:AEAB Flags:S--H-PNC
SP:FE4E AF'B0E6 BC'2176 DE'F789 HL'B354 (S)00B1 IR:801E Flags'SZ---PN-

Questo mostra che il registro A ora è 41. Il comando R mostra più registri di quelli mostrati nel punto di interruzione, ma per ora questi possono essere ignorati.

Per continuare il programma con il nuovo valore nel registro A, inserire:

```
G {invio}
```

Il programma dovrebbe quindi completarsi come prima, ma visualizzerà il carattere "A" invece del carattere "I".

Se il programma fosse più lungo, potrebbe valere la pena impostare il punto di interruzione più avanti nel programma prima di continuare. Il programma continuerà quindi dal primo punto di interruzione al nuovo punto di interruzione.

Nei programmi più complessi la possibilità di arrestare il programma in una posizione specifica e visualizzare i registri e i flag del processore può essere preziosa. Essere in grado di modificare i valori del registro prima di continuare è un modo molto utile di interagire con il programma per impostare condizioni di test e provare idee.

Passo singolo

L'altro strumento di debug è il passo singolo (Single Step).

Il punto di interruzione consente di arrestare il programma in una posizione specifica, mentre la funzionalità di passo singolo consente al programma di eseguire un'istruzione alla volta. Il passo singolo è proprio come impostare il punto di interruzione dopo l'esecuzione di ciascuna istruzione.

Per eseguire un singolo passo nell'esempio precedente, immettere:

```
S 8000 {invio}
```

```
PC:8000 AF:41B7 BC:6902 DE:BDBA HL:BCDE IX:5B5A IY:AEAB Flags:S--H-PNC
```

Lo stato del processore viene visualizzato subito prima dell'esecuzione dell'istruzione all'indirizzo specificato. Nota: il messaggio del monitor "*" non viene visualizzato.

Per eseguire l'istruzione all'indirizzo visualizzato, premere il tasto Invio. Dovrebbe essere visualizzato quanto segue.

```
8000: 3E 21      >!    LD A,$21
```

```
PC:8002 AF:21B7 BC:6902 DE:BDBA HL:BCDE IX:5B5A IY:AEAB Flags:S--H-PNC
```

Come accennato in precedenza per i punti di interruzione, alcuni valori del registro saranno probabilmente diversi da quelli illustrati sopra.

La prima riga mostra l'istruzione all'indirizzo corrente, la seconda riga mostra lo stato del processore dopo l'esecuzione dell'istruzione. In questo caso il registro A è ora 21 e il program counter (PC) è ora l'indirizzo della prossima istruzione da eseguire (8002).

Premendo il tasto Invio si esegue l'istruzione successiva e così via.

L'esecuzione dell'intero programma un passo alla volta dovrebbe risultare in una visualizzazione simile a questa:

```
s 8000 {return}
PC:8000 AF:41B7 BC:6902 DE:BDBA HL:BCDE IX:5B5A IY:AEAB Flags:S--H-PNC
8000: 3E 21      >!    LD A,$21
PC:8002 AF:21B7 BC:6902 DE:BDBA HL:BCDE IX:5B5A IY:AEAB Flags:S--H-PNC
8002: 0E 02      .    LD C,$02
PC:8004 AF:21B7 BC:6902 DE:BDBA HL:BCDE IX:5B5A IY:AEAB Flags:S--H-PNC
8004: CD 30 00    .0.   CALL $0030
Stepping over code in ROM or in monitor
!PC:8007 AF:2180 BC:1B02 DE:BDBA HL:021B IX:5B5A IY:AEAB Flags:S-----
8007: C9        .    RET
Stepping over code in ROM or in monitor
```

Laddove il display mostra “Stepping over code in ROM or in monitor” all'indirizzo 8004, la modalità passo singolo ha saltato tutte le istruzioni nel monitor che vengono eseguite quando si invia un carattere al terminale.

Quando il programma termina con l'istruzione RET all'indirizzo 8007, viene nuovamente visualizzato il messaggio “Stepping over code in ROM or in monitor”, mentre il ritorno ritorna al programma monitor. La procedura a passo singolo termina e viene visualizzato il prompt del monitor.

Quando si scorre un programma è possibile premere il tasto Esc per uscire dal passaggio singolo o semplicemente iniziare a digitare un nuovo comando. Non è necessario prima uscire dalla modalità passo singolo.

È possibile combinare punto di interruzione e passaggio singolo, ad esempio:

```
b 8004 {invio}
Breakpoint set

g 8000 {invio}
Breakpoint
PC:8004 AF:2180 BC:1B02 DE:BDBA HL:021B IX:5B5A IY:AEAB Flags:S-----

s {invio}
PC:8004 AF:2180 BC:1B02 DE:BDBA HL:021B IX:5B5A IY:AEAB Flags:S-----
8004: CD 30 00    .0.   CALL $0030
Stepping over code in ROM or in monitor
!PC:8007 AF:2180 BC:1B02 DE:BDBA HL:021B IX:5B5A IY:AEAB Flags:S-----
8007: C9        .    RET
Stepping over code in ROM or in monitor
```

Quando si esegue il debug di un programma di grandi dimensioni, eseguire i passaggi dall'inizio può essere molto noioso, quindi è meglio impostare un punto di interruzione all'inizio del codice sospetto e procedere da lì al singolo passaggio.

Hello World!

Ora passiamo al classico programma “Hello World!”.

Abbiamo già mostrato come visualizzare un singolo carattere sul terminale.

```
8000: 3E 21      >! LD A,$21
8002: 0E 02      .. LD C,$02
8004: CD 30 00   .0. CALL $0030
8007: C9         . RET
```

Per visualizzare “Hello World!” è sufficiente ripetere questo:

```
LD A,$48      ;"H"
LD C,$02
CALL $0030
```

```
LD A,$65      ;"e"
LD C,$02
CALL $0030
```

```
LD A,$6C      ;"l"
LD C,$02
CALL $0030
```

RET

Questo porterebbe a termine il lavoro ma è molto inefficiente. Un modo migliore sarebbe quello di avere i caratteri del messaggio archiviati in memoria e avere un piccolo programma che scansiona il messaggio un carattere alla volta visualizzandoli man mano che procedono.

Per scrivere caratteri in memoria è possibile utilizzare il comando Modifica (E) come segue.

```
e 8100 {invio}
8100: 74      t LD (HL),H      > "Hello world!
810C: 3F      ? CCF           > 0
810D: D1      . POP DE       >
```

Premere il tasto Esc per uscire dalla modifica della memoria.

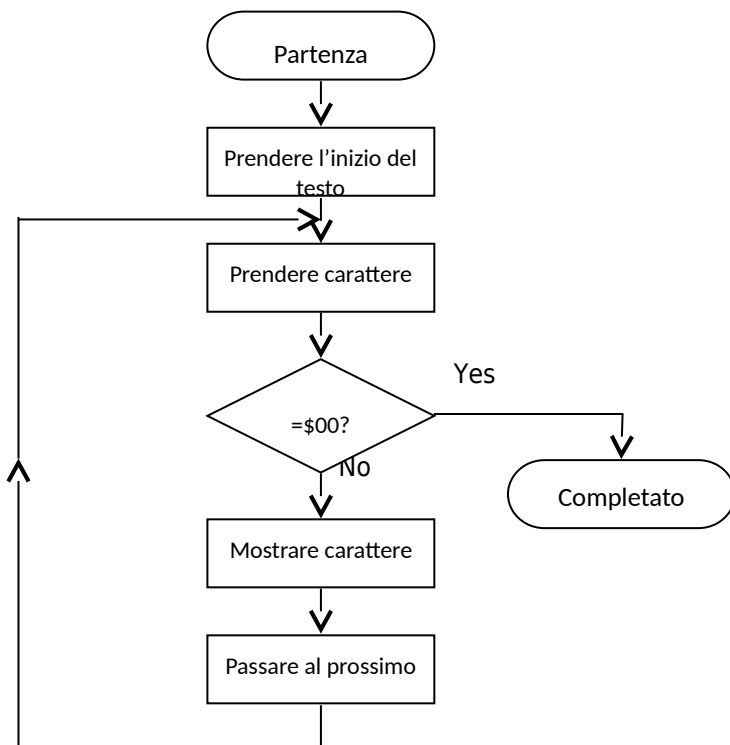
Il comando di modifica mostra il contenuto corrente della memoria e invita a inserire nuovi contenuti. Accetta numeri esadecimali, come F3, o una stringa di caratteri ASCII preceduti da un carattere Citazione. Notare che non si inserisce un carattere di doppie virgolette alla fine dei caratteri che inseriti. Quando si inseriscono numeri esadecimali si possono inserirne più di uno, basta inserire uno spazio tra di loro.

Nell'esempio sopra, lo zero viene inserito come indicatore di fine testo.

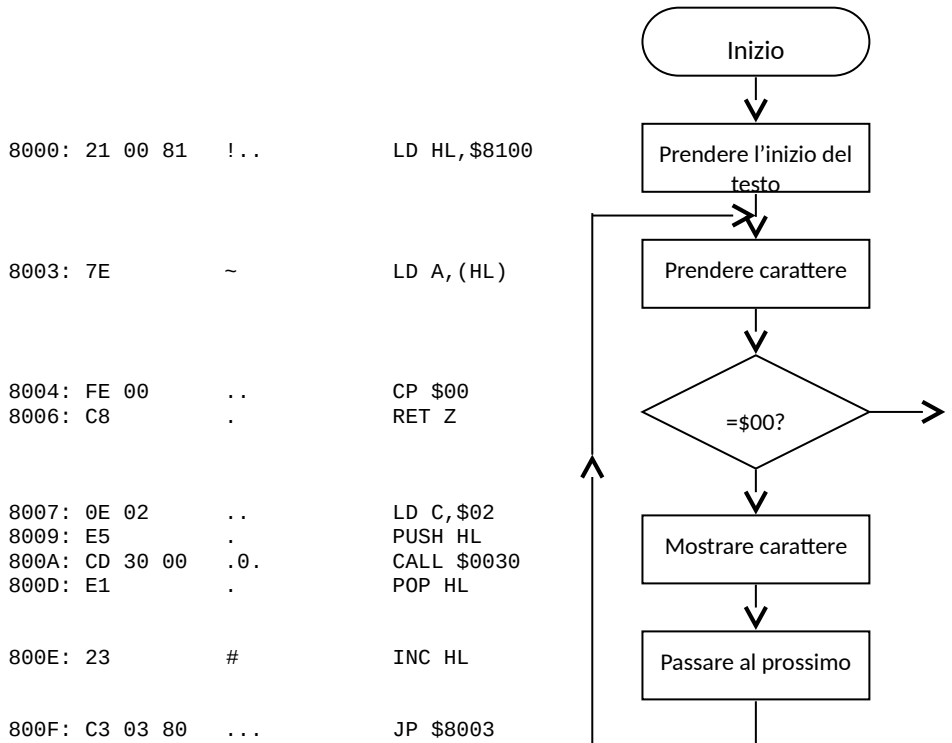
Si può verificare che i caratteri siano inseriti correttamente con il comando Memoria (M).

```
m 8100 {invio}
8100: 48 65 6C 6C 6F 20 77 6F 72 6C 64 21 00 D1 94 E3 Hello world!....
8110: 9A 82 B7 15 BA 13 B8 01 96 53 42 57 2E D7 9C BD .....SBW....
8120: E4 FD D7 A9 D7 F2 8B 77 DD 89 9D 7F B7 9C 7E 24 .....w.....~$
8130: 22 83 7B 12 12 92 64 AB 42 02 10 A8 80 B0 02 A6 ".{...d.B.....
8140: 61 6A D8 42 44 B8 45 26 D8 AF B6 BF E9 78 58 6B aj.BD.E&.....xXk
8150: 2A F7 85 D7 50 34 27 9B 6F 11 49 A3 85 DB 94 90 *...P4'.o.I.....
8160: DD 5D C6 4B 34 63 83 7B 99 EF ED 90 B1 E7 7D C2 .].K4c.{.....}.
8170: 72 12 D3 DC 72 19 56 29 CA 86 7D 32 9C 8C 03 4E r...r.V)..}2...N
```

Ora abbiamo i caratteri in memoria, quindi abbiamo bisogno di un modo semplice per leggere ogni carattere a turno e visualizzarlo, fermandosi quando viene raggiunto il marcatore zero. Ecco come funzionerà:



La conversione del diagramma di flusso in istruzioni assembler ci dà questo:



Considerando singolarmente ogni fase del programma, ecco una spiegazione dettagliata.

LD HL, \$8100

La stringa di caratteri che vogliamo visualizzare è in memoria a 8100. L'istruzione "LD HL, \$8100" imposta la coppia di registri HL al valore 8100, la posizione della stringa di caratteri. Pertanto HL punta all'inizio della stringa di caratteri.

LD A, (HL)

Il contenuto della memoria indicato da HL viene indicato con "(HL)". Pertanto "LD A,(HL)" prende il contenuto della memoria indicato da HL e lo memorizza nel registro A. Quindi A ora contiene un carattere della stringa.

CP \$00

Questa istruzione confronta il valore nel registro A con il valore specificato, in questo caso \$00. Quindi l'istruzione pone la domanda: A = 0? Se il valore

in A è uguale al valore specificato, viene impostato il flag Zero (Z) del processore. In altre parole, la differenza tra loro è Zero. Se non sono uguali, il flag Zero del processore viene cancellato. Questo è indicato come Not Zero o NZ. Questa istruzione controlla se abbiamo raggiunto il carattere terminatore della stringa che ha il valore ASCII \$00.

RET Z

Questa è una variazione dell'istruzione Return (RET). Ritorno significa ritorno dalla subroutine. Un programma è scritto come una subroutine, quindi il ritorno dalla subroutine termina il programma. Questa versione dell'istruzione è RET Z, che significa Ritorno se è impostato il flag Zero. Pertanto, se l'istruzione precedente rileva che il carattere è \$00 e quindi imposta il flag Zero, questa istruzione terminerà la subroutine e quindi terminerà il programma.

LD C,\$02

Questo memorizza il valore 2 nel registro C. Questo è in preparazione per chiamare l'Application Programming Interface (API) con la funzione numero 2.

PUSH HL

In precedenza, quando chiamavamo l'API, non eravamo preoccupati di quali fossero i valori del registro al termine della funzione API, ma eccoci qui. Dobbiamo ricordare il valore di HL poiché ne abbiamo ancora bisogno per trovare il carattere successivo nella stringa. Dobbiamo quindi ricordare il valore di HL nel caso in cui la funzione API lo modifichi. L'istruzione "PUSH HL" mette una copia di HL nello Stack. La pila è come una pila di carta. Se metti un nuovo pezzo di carta, con un numero scritto sopra, nella pila, rimarrà lì. Quando avrai nuovamente bisogno del numero, potrai toglierlo dallo stack. Puoi avere molti pezzi di carta in pila ma puoi solo metterli in cima e toglierli dall'alto, quindi non puoi cambiare l'ordine.

CALL \$0030

Questo chiama l'API. Il valore del registro C contiene il numero della funzione, in questo caso 2. La funzione 2 visualizza il carattere ASCII in A sul display del terminale.

A titolo di ulteriore spiegazione, l'istruzione CALL funziona inserendo il valore corrente del contatore del programma nello stack e quindi caricando il contatore del programma con l'indirizzo specificato. Alla fine della subroutine, l'istruzione RET estrae un valore dallo stack e lo scrive nel registro del contatore del programma. Ciò fa sì che il programma ritorni all'istruzione immediatamente successiva alla CALL.

POP HL

Per recuperare un valore dallo stack si usa l'istruzione "POP". Questo preleva un valore dallo stack e lo memorizza nella coppia di registri specificata. In questo caso ripristiniamo HL all'indirizzo del carattere nella stringa di caratteri. Per usare l'analogia della pila di carta, prendi il primo foglio di carta dalla pila.

INC HL

Incrementa il valore della coppia di registri specificata. Incremento significa semplicemente aggiungerne uno. Quindi $HL = HL + 1$. Poiché HL punta a un carattere nella stringa di caratteri, ora punta al carattere successivo nella stringa.

JP \$8003

Ciò fa sì che il programma salti all'indirizzo specificato. In realtà memorizza il valore specificato nel registro PC. Il programma quindi torna indietro per considerare il carattere successivo nella stringa.

E questo è tutto. Il programma "Hello World!" è fatto.

Un programma migliore

In realtà non è tutto finito perché ci sono alcuni miglioramenti da apportare e alcune lezioni da imparare.

Il programma di visualizzazione delle stringhe sopra è così utile che vale la pena organizzarlo come una subroutine da utilizzare nei programmi futuri.

Ecco la routine di visualizzazione delle stringhe riutilizzabile:

8200:	7E	~	LD A, (HL)
8202:	FE 00	..	CP \$00
8203:	C8	.	RET Z
8004:	0E 02	..	LD C, \$02
8006:	E5	.	PUSH HL
8007:	CD 30 00	.0.	CALL \$0030
800A:	E1	.	POP HL
800B:	23	#	INC HL
800C:	C3 00 82	!..	JP \$8200

Ora per visualizzare la stringa:

8000:	21 00 81	!..	LD HL, \$8100
8003:	CD 00 82	...	CALL \$8200
8006:	C9	.	RET

Ma ci sono anche ulteriori miglioramenti.

Si può sostituire "CP \$00" con "OR A". Anche l'istruzione "OR A" imposterà il flag Zero se A vale \$ 00, ma è più piccola e più veloce.

È anche possibile sostituire l'istruzione CALL \$0030. Il processore Z80 ha alcune istruzioni CALL speciali che sono più piccole e più veloci. Queste sono istruzioni di riavvio, dove RST 30 equivale a CALL 0030.

Questo tutorial non vuole essere una guida completa alla programmazione dello Z80. Ci sono molti altri materiali da consultare su Internet, quindi qui non vengono fornite ulteriori spiegazioni sui miglioramenti di cui sopra.

C'è tuttavia un miglioramento specifico per Small Computer Monitor e quindi necessita di essere menzionato. La subroutine per visualizzare una stringa di caratteri è così utile e utilizzata così spesso che il Monitor fornisce questa funzione come una delle sue chiamate API. La subroutine riutilizzabile di cui sopra non è effettivamente necessaria nel kit di strumenti di codice personale. Per visualizzare il testo "Hello World!", scrivere semplicemente questo:

```
8000: 11 00 81    ...    LD DE,$8100
8003: 0E 06      ..     LD C,$06
8005: F7         .      RST 30
8006: C9         .      RET
```

Alcuni direbbero che è una buona pratica trasformare questo codice specifico del sistema operativo in una subroutine:

```
8300: 0E 06      ..     LD C,$06
8302: F7         .      RST 30
8303: C9         .      RET
```

E chiamarla con:

```
8000: 11 00 81    ...    LD DE,$8100
8003: CD 00 83    ...    CALL $8300
8306: C9         .      RET
```

In questo modo, la maggior parte del codice è indipendente dal sistema operativo su cui è in esecuzione. Anche in questo caso ci sono molte risorse disponibili riguardanti le tecniche di programmazione.

Registri e Flag

Abbiamo già visto come esaminare i valori di registro del processore.

```
R {invio}
```

```
PC:8004 AF:41B7 BC:6902 DE:BDBA HL:BCDE IX:5B5A IY:AEAB Flags:S--H-PNC  
SP:FE4E AF'B0E6 BC'2176 DE'F789 HL'B354 (S)00B1 IR:801E Flags'SZ---PN-
```

La prima riga di registri mostra:

- Program Counter (PC)
- Coppie di registri (AF, BC, DE, HL)
 - Il registro A è l'accumulatore del processore
 - Il registro F contiene i flag di stato del processore
- Registri indice (IX, IY)
- Flag mostrati come bit simbolici (S, Z, H, P, N, C)

La seconda linea di registri mostra:

- Stack Pointer (SP)
- Set di registri alternativi (AF', BC', DE', HL')
- Il contenuto referenziato dallo stack (S)
- La coppia di registri vettore Interrupt e Refresh (IR)
- Registro flag alternativo mostrato come bit simbolici (S, Z, H, P, N, C)

Ci sono molte risorse in internet per spiegare questi registri e flag.

Abbiamo visto come modificare il valore di un singolo registro.

```
R A {invio}
```

```
A: 41 _
```

Possiamo modificare il valore di una coppia di registri allo stesso modo.

```
R HL {invio}
```

```
A: BCDE _
```

Con la stessa modalità è possibile modificare anche il registro dei Flag.

```
r f {invio}
```

```
F: 01 _
```

Sebbene questo metodo di modifica del registro Flags funzioni, non è molto conveniente. Ciascun flag è un singolo bit all'interno del registro a 8 bit, quindi manipolare i bit del flag inserendo un byte esadecimale richiede un po' di riflessione. Small Computer Monitor dispone quindi di un comando separato per modificare i singoli bit di flag.

Digitando solo il comando Flag (F) vengono visualizzati i registri, inclusi i bit di flag.

f {invio}

PC:7000 AF:6597 BC:1B02 DE:BDBA HL:1234 IX:5B5A IY:AEAB Flags:S--H-PNC

Digitando il comando Flag (F) seguito dal nome di un flag si imposterà quel bit di flag e si visualizzerà il valore aggiornato.

f z {invio}

PC:7000 AF:65D7 BC:1B02 DE:BDBA HL:1234 IX:5B5A IY:AEAB Flags:SZ-H-PNC

Per cancellare un flag, digitare il comando Flag (F) seguito dalla versione NOT del nome del flag:

f nz {invio}

PC:7000 AF:6597 BC:1B02 DE:BDBA HL:1234 IX:5B5A IY:AEAB Flags:S--H-PNC

La Guida per l'utente di Small Computer Monitor fornisce i dettagli di tutti i nomi di flag e condizioni che possono essere utilizzati con questo comando.

Altri programmi di esempio

Questa sezione contiene alcuni programmi di esempio progettati per illustrare come utilizzare le varie funzionalità del monitor per piccoli computer e anche per mostrare ulteriori tecniche di programmazione dello Z80.

Si potrebbe voler consultare la Guida per l'utente di Small Computer Monitor che contiene informazioni di riferimento relative ad alcuni di questi esempi.

Questi esempi sono scritti solo come codice sorgente e non includono i byte esadecimali del codice macchina assemblati mostrati negli esempi precedenti.

È buona norma quando si scrive il codice sorgente aggiungere commenti per spiegare il codice. I commenti iniziano con un punto e virgola e si estendono fino alla fine della riga corrente. I commenti possono trovarsi su una riga separata o alla fine di una riga di istruzioni. Quando si utilizza un assembler completo, questi commenti fanno parte del codice sorgente e aiutano a documentarne il funzionamento.

Quando si utilizza l'assembler in linea di Small Computer Monitor non si inseriscono commenti.

Mostrare una nuova riga

Gli esempi precedenti che mostravano come visualizzare i caratteri sullo schermo di un terminale evitavano di produrre una fine di riga e quindi di iniziare qualsiasi ulteriore visualizzazione su una nuova riga.

Questo codice mostra come visualizzare un carattere, quindi una nuova riga, quindi un altro carattere.

```
; Example showing output on a new line
LD  A,31      ;ASCII character "1"
LD  C,2       ;API call 2 = Output character
RST 30        ;API call outputs "1"
LD  C,7       ;API call 7 = Output new line
RST 30        ;API call outputs new line
LD  A,32      ;ASCII character "2"
LD  C,2       ;API call 2 = Output character
RST 30        ;API call outputs "2"
RET
```

L'output dovrebbe assomigliare a questo.

1
2*

È possibile generare una nuova riga emettendo un carattere di ritorno a capo (\$0D) ed eventualmente anche un carattere di avanzamento riga (\$0A). Il problema è la parola “forse”. Alcuni dispositivi utilizzano il carrello per una nuova riga, altri utilizzano il ritorno a capo più l'avanzamento riga e altri ancora utilizzano solo l'avanzamento riga. Utilizzando la chiamata API per generare una nuova riga, non è necessario codificare nel programma i caratteri esatti richiesti. Pertanto, quando il programma viene spostato su un sistema con uno schema di nuova linea diverso, continuerà a funzionare correttamente.

Non è sempre conveniente chiamare l'API ogni volta che è richiesta una nuova riga. Ad esempio, si potrebbe voler avere una stringa di caratteri (come nell'esempio Hello World) che includa una nuova riga. Small Computer Monitor consente ciò espandendo il carattere \$05 a qualunque nuova sequenza di righe per la quale è configurato il monitor.

Ingresso da tastiera

Questo esempio attende che l'utente prema un tasto sulla tastiera del terminale, quindi attende un secondo prima di visualizzare il carattere sullo schermo del terminale.

```
LD    C,1      ;Selezione API 1 = Input carattere
RST   $30      ;Chiamata API inputs carattere
PUSH  AF       ;Memorizzazione del carattere
LD    C,+10    ;Selezione API 10 = Attesa di DE ms
LD    DE,+1000 ;Ritardo in millisecondi
RST   $30      ;Chiamata API attesa 1 secondo
POP   AF       ;Richiamo carattere
LD    C,2      ;Selezione API 2 = Output carattere
RST   $30      ;Chiamata API outputs character
RET
```

Lampeggio di un LED

Se si dispone di un modulo I/O digitale, con LED per le uscite, è possibile scrivere un semplice programma per far lampeggiare un LED.

Esempio LiNC80: (porta output ipotizzata all'indirizzo \$30)

```
8000  Loop:    LD    A,1      ;LED bit zero valore ON
                OUT  ($30),A  ;Accende il LED
                LD    C,+10    ;API call 10 = Ritarda DE millisecondi
                LD    DE,+500  ;Ritardo in millisecondi
                RST   $30      ;Esegue API ritardo 0.5 secondi
```

```

LD    A,0      ;LED bit zero valore OFF
OUT ($30),A    ;Spegne il LED
LD    C,+10    ;API call 10 = Ritarda DE millisecondi
LD    DE,+500  ;Ritardo in millisecondi
RST   $30      ;Esegue API ritardo 0.5 secondi
JP    Loop     ;Ripete all'infinito

```

Esempio RC2014: (porta output ipotizzata all'indirizzo \$00)

```

8000  Loop:    LD    A,1      ;LED bit zero valore ON
                        OUT ($00),A    ;Accende il LED
                        LD    C,+10    ;API call 10 = Ritarda DE millisecondi
                        LD    DE,+500  ;Ritardo in millisecondi
                        RST   $30      ;Esegue API ritardo 0.5 secondi
                        LD    A,0      ;LED bit zero valore OFF
                        OUT ($00),A    ;Spegne il LED
                        LD    C,+10    ;API call 10 = Ritarda DE millisecondi
                        LD    DE,+500  ;Ritardo in millisecondi
                        RST   $30      ;Esegue API ritardo 0.5 secondi
                        JP    Loop     ;Ripete all'infinito

```

Si suggerisce come posizione del programma l'indirizzo 8000.

Gli esempi precedenti presuppongono che il modulo I/O digitale sia configurato per l'indirizzo della porta di uscita predefinita per il sistema host.

Quando questo programma viene eseguito, il LED lampeggia, lampeggia e lampeggia. Non termina finché il processore non si ripristina. In realtà è piuttosto difficile scrivere un programma che possa fare qualcosa di apparentemente semplice come far lampeggiare una luce mentre fa anche qualcos'altro.

Utilizzando un assembler completo, "Loop" è un'etichetta che può essere utilizzata per fare riferimento all'indirizzo di memoria di Loop. In questo caso l'istruzione "JP Loop" viene assemblata come un salto alla locazione di memoria contenente l'istruzione "LD A,1".

Quando si utilizza l'assemblatore in linea di Small Computer Monitor, viene considerata solo una linea alla volta, e le etichette su altre linee non sono supportate. Pertanto è necessario inserire l'indirizzo del salto, come hanno mostrato gli esempi precedenti. Presto si vorrà avere un assemblatore completo di tutte le funzionalità!

Miglioriamo il lampeggio!

L'esempio "Lampeggio di un LED" può essere migliorato. Ogni volta che si vedono blocchi di codice ripetuti, vale la pena valutare se il codice può essere scritto per evitare la duplicazione.

Se estraiamo il blocco comune di codice e lo trasformiamo in una subroutine "Led" otteniamo questo:

```
8000  Loop:      LD      A,1      ;LED bit zero valore ON
                   CALL Led      ;Invia ai LED e attendi
                   LD      A,0      ;LED bit zero valore OFF
                   CALL Led      ;Invia ai LED e attendi
                   JP       Loop    ;Ripete all'infinito

8100  Led:       OUT (0),A      ;Cambia lo stato on/off dei LED
                   LD      C,+10    ;API call 10 = Ritarda DE millisecondi
                   LD      DE,+500  ;Ritardo in millisecondi
                   RST 30          ;Esegue API ritardo 0.5 secondi
                   RET             ;Ritorna da questa subroutine
```

Si suggerisce come posizioni del programma gli indirizzi 8000 e 8100.

Ora il primo Loop può essere espanso per includere ulteriori modifiche alla luce, come:

```
Loop:      LD      A,1      ;LED bit zero valore ON
            CALL Led      ;Invia ai LED e attendi
            LD      A,2      ;LED bit uno valore ON
            CALL Led      ;Invia ai LED e attendi
            LD      A,3      ;LED bit due valore ON
            CALL Led      ;Invia ai LED e attendi
            LD      A,4      ;LED bit tre valore ON
            CALL Led      ;Invia ai LED e attendi
            JP       Loop    ;Ripete all'infinito
```

Ora sembra che anche il ciclo sopra possa essere migliorato.

Per perfezionare ulteriormente questo codice e adattarlo meglio a uno spettacolo di luci più lungo, utilizzeremo una tabella dati. Le tabelle dati sono dispositivi molto potenti.

Nell'esempio sopra ripetiamo la sequenza:

```
LD A,x
CALL Led
```

Ad ogni cambio di luce.

Una soluzione migliore, soprattutto per sequenze più lunghe, sarebbe quella di avere un elenco (o tabella) dei valori di uscita dei LED e avere una piccola routine che legge ogni elemento nell'elenco e lo trasmette ai LED.

Per modificare la sequenza delle luci è quindi sufficiente modificare la tabella.

Una tabella per l'esempio precedente sarebbe:

DB 1,2,3,4

In questo caso la direttiva assembler DB sta per Definire Byte e fa sì che i valori numerici elencati vengano scritti in memoria come una serie di byte.

Il seguente programma è stato inserito nell'assemblatore on line su www.asm80.com. Vedere www.scc.me.uk per un breve tutorial su asm80.com.

```
.ORG    $8000

; Carica nel registro B la lunghezza della lista (14 decimale)
; Carica nel registro HL l'indirizzo dell'inizio della lista (TABLE)
START:  LD      B,14
        LD      HL, TABLE

; Ricomincia da qui per ogni valore da inviare ai LED
LOOP:

; Carica nel registro A il contenuto della memoria indirizzato da HL
; Questo sarà il valore inviato ai LED
        LD      A,(HL)

; Chiama la subroutine che invia il valore alla porta dei LED e
; attende.
; Dato che la subroutine LED potrebbe modificare il valore dei
; registri B e HL, usati per conteggiare e puntare alla tabella,
; è necessario memorizzare il contenuto di questi registri nello
; stack (istruzione PUSH) e in seguito ripristinare questi valori
; dallo stack (istruzione POP)
        PUSH    BC
        PUSH    HL
        CALL    LED
        POP     HL
        POP     BC

; Incrementa HL per puntare al successivo valore nella tabella
        INC     HL

; Ripeti fino al raggiungimento della fine della tabella (il
; registro B diventa zero)
; DJNZ è un'istruzione intelligente che decrementa il registro B
; e se non è zero salta alla posizione specificata
        DJNZ    LOOP

; Per ripetere la sequenza all'infinito, occorre tornare
; all'inizio del programma
        JR      START

; In alternativa si può tornare al Monitor
        RET
```

```

; Questa subroutine invia il valore del registro A alla porta dei
; LED, quindi attende prima di ritornare al chiamante.
; Attenzione: la chiamata API (RST $30) può cambiare il valore di
; alcuni registri, quindi è necessario memorizzare i loro valori
; prima della chiamata e ripristinarli in seguito
LED:      OUT      ($00),A
          LD        C,10
          LD        DE,500
          RST       $30
          RET

; Questa tabella è un elenco di valori da inviare ai LED
TABLE:    DB        1,2,4,8,16,32,64,128
          DB        64,32,16,8,4,2

```

Spesso le tabelle hanno un indicatore di fine, quindi non è necessario conoscere in anticipo la lunghezza della tabella. La tabella può essere semplicemente letta finché non viene trovato il valore del contrassegno finale. In questo caso tuttavia non è possibile riservare alcun valore byte come indicatore di fine poiché per lo spettacolo di luci potrebbe essere necessario qualsiasi modello LED.

Una tabella con un NULL (\$00) come indicatore di fine potrebbe assomigliare a questa:

```

DB        64,32,16,8,4,2
DB        0          ; Marcatore di fine

```

L'assemblatore online produce un file esadecimale (HEX) da inviare a Small Computer Monitor come documentato nel tutorial di Asm80.com. Produce anche un file di elenco come mostrato di seguito. In questo caso i commenti sono stati esclusi.

```

8000                                     .ORG $8000
8000 06 0E                                LD    B,14
8002 21 1C 80                            LD    HL, TABLE
8005 7E                                LOOP: LD    A, (HL)
8006 C5                                PUSH   BC
8007 E5                                PUSH   HL
8008 CD 13 80                            CALL   LED
800B E1                                POP    HL
800C C1                                POP    BC
800D 23                                INC    HL
800E 10 F5                            DJNZ   LOOP
8010 18 EE                            JR     START
8012 C9                                RET
8013 D3 00    LED:                      OUT   ($00),A
8015 0E 0A                                LD     C,10
8017 11 F4 01                            LD     DE,500
801A F7                                RST    $30
801B C9                                RET

```

```

801C  01 02 04 08 10 20 40 80 40 20 10 08 04 02
      TABLE:      DB   1, 2, 4, 8, 16, 32, 64, 128
                   DB   64, 32, 16, 8, 4, 2

```

In questo esempio la tabella dati è un semplice elenco di numeri. Va considerato come un foglio di calcolo con una sola colonna. Utilizzando più di una colonna è possibile creare tabelle molto più complesse. Codificando attentamente valori e funzioni in combinazioni di bit e byte, problemi piuttosto complessi possono essere ridotti a piccoli programmi che traggono gran parte della loro apparente sofisticazione dalla tabella dei dati. L'assemblatore e il disassemblatore integrati in Small Computer Monitor utilizzano questa tecnica. Cercare di scrivere codice per gestire direttamente ciascuna istruzione Z80 risulterebbe in un programma enorme e ingombrante.

Altri modi di divertirsi con I LED

Come possiamo fare qualcosa di apparentemente semplice come accendere una luce o eseguire uno spettacolo di luci mentre facciamo anche qualcos'altro? Esistono varie tecniche per raggiungere questo obiettivo.

Si potrebbe utilizzare gli interrupt se il piccolo computer ha un orologio hardware. Purtroppo i kit standard RC2014 non li hanno di serie, ma il LiNC80 sì.

Un orologio (timer) hardware consentirebbe di interrompere il sistema a intervalli di tempo regolari. Qualunque software sia in esecuzione quando si verifica l'interruzione hardware viene sospeso e viene eseguita la routine di gestione dell'interruzione designata. Il gestore dell'interruzione dovrebbe essere una breve subroutine che, in questo caso, modifica lo stato dei LED prima di ritornare. Al rientro si riprende l'attività precedente. Normalmente ciò accadrebbe così rapidamente che l'utente non noterebbe alcun ritardo.

Senza un orologio hardware, una subroutine simile potrebbe essere utilizzata per sequenziare i LED, ma dovrebbe essere chiamata regolarmente tramite polling. Ciò significa che è necessario che il software in esecuzione sul sistema di destinazione richiami regolarmente la subroutine LED. Questo è un vero fastidio e può, nella migliore delle ipotesi, portare solo alla chiamata della subroutine LED approssimativamente all'intervallo richiesto.

Poiché la capacità di eseguire facilmente semplici attività in background, come i LED lampeggianti, è così desiderabile, Small Computer Monitor fornisce una soluzione di polling. Lo fa nel miglior modo possibile senza gravare eccessivamente sull'utente e sui suoi programmi. È una soluzione molto compromessa ma è comunque piuttosto utile da avere. Il meccanismo adottato si basa sul fatto che la maggior parte delle volte un computer attende semplicemente che l'utente prema un tasto. Aspettare la pressione di un tasto è un processo abbastanza prevedibile e consente di tenere il tempo ragionevolmente accurato quando questa è l'unica cosa che succede. La precisione diventa molto meno prevedibile in altre condizioni.

L'esecuzione di semplici attività in background, come il lampeggiamento dei LED, è un meccanismo estremamente importante e porta in definitiva ai moderni sistemi operativi basati sugli eventi.

I computer a otto bit sono relativamente limitati e di solito funzionano grazie al programma che prende il controllo del sistema (a parte gli interrupt hardware) e di solito sono strutturati in un ciclo o una serie di cicli,

che aspettano qualche input e poi fanno qualcosa con esso prima di ripetere il processo. Nei sistemi guidati dagli eventi, il programma dell'utente è più simile a uno schiavo del sistema operativo e dà un'occhiata solo quando il sistema operativo richiede che il programma dell'utente risponda a qualche evento.

Come descritto in precedenza, con un interrupt del timer hardware è necessario scrivere una subroutine per sequenziare i LED di un passo ogni volta che viene richiamata la subroutine. Lo stesso vale per il timer interrogato. In questo caso tuttavia chiamiamo la subroutine un gestore di eventi.

Ad esempio, convertirlo in un precedente spettacolo di luci LED basato su tabella in un programma basato su eventi. Ciò richiede che il programma ricordi il punto della sequenza raggiunto l'ultima volta che è stato chiamato.

Nella precedente implementazione Small Computer Monitor non eseguiva nessun altro codice, quindi i registri del processore contenevano la posizione corrente nella sequenza. Ora è necessario memorizzare questi valori in variabili in modo che i valori dei registri possano essere ripristinati alla successiva chiamata della subroutine.

Il programma seguente è stato creato utilizzando www.Asm80.com. Utilizza gli eventi di Small Computer Monitor per sequenziare lo spettacolo di luci.

```
.ORG    $8000

; Avvia il programma impostando lo stato iniziale
START:

; Attiva il gestore eventi di Small Computer Monitor
        LD     A,1
        LD     C,$13
        RST    $30

; Imposta il timer 2 per chiamare la subroutine "POLL"
; ogni 50 x 10 millisecondi
        LD     A,50
        LD     DE,POLL
        LD     C,$15
        RST    $30

; Impostazioni per iniziare o ripetere la sequenza dello
; spettacolo luminoso
REPEAT:
        LD     A,14
        LD     (COUNT),A
        LD     HL, TABLE
        LD     (POINT),HL
        RET
```

```

; Questa subroutine è chiamata regolarmente da Small Computer Monitor
; Usa le variabili POINT e COUNT per tenere traccia della posizione
; all'interno della sequenza di luci
POLL:

```

```

; Prima si ripristina il puntatore alla tabella dei dati,
; poi si legge il valore dalla tabella e lo si invia ai led,
; infine si incrementa il puntatore al valore successivo e lo si
; memorizza

```

```

LD      HL,(POINT)
LD      A,(HL)
OUT     ($00),A
INC     HL
LD      (POINT),HL

```

```

; Si decrementa il contatore della tabella
; Se il contatore raggiunge il valore zero, si salta alle istruzioni
; di impostazione per ripetere la sequenza

```

```

LD      A,(COUNT)
DEC     A
JR      Z,REPEAT
LD      (COUNT),A

```

```

RET

```

```

; Tabella con i valori dei LED

```

```

TABLE:  DB      1,2,4,8,16,32,64,128,64,32,16,8,4,2

```

```

; Variabili usate per ricordare la posizione usata all'interno
; della tabella

```

```

COUNT: DS      1
POINT:   DS      2

```

Avviare il programma come al solito con il comando:

```
G 8000 {invio}
```

Tenere presente che mentre è in esecuzione, Small Computer Monitor è ancora attivo, quindi si può digitare comandi ed eseguire programmi.

Mentre lo spettacolo di luci è in esecuzione, provare ad accelerarlo modificando l'intervallo del timer a 2 x 10 millisecondi con il comando API del monitor:

```
API 15 2 8019 {invio}
```

Attenzione: funzionerà solo se il programma è esattamente come mostrato sopra, poiché questo comando API imposta il gestore eventi all'indirizzo \$8019.

L'API \$15 imposta il timer dell'evento sul multiplo specificato di 10 millisecondi e il gestore dell'evento sull'indirizzo specificato. Consulta la guida utente di Small Computer Monitor per ulteriori dettagli sulle funzioni API.

Assemblatori

Small Computer Monitor ha un assemblatore integrato in linea, che è davvero comodo, ma non è adatto alla scrittura di programmi di grandi dimensioni. Esistono due approcci di base per lo sviluppo di programmi in codice macchina di grandi dimensioni:

- Installare un editor e assemblatore completo sul computer di destinazione
- Utilizzare un editor e assemblatore su un PC (o simile)

Il primo approccio è tutto ciò che la maggior parte delle persone aveva in passato. Spesso si trattava di un software di sviluppo eseguito sotto CP/M. Questa opzione è disponibile anche oggi. Il software e la documentazione possono essere trovati online, ma per eseguirlo è necessario un piccolo computer di fascia alta.

La seconda opzione è molto interessante oggi poiché abbiamo facile accesso a computer moderni relativamente potenti. Il software per modificare il codice sorgente e per assemblarlo nel codice macchina Z80 è disponibile online. Esistono anche simulatori per consentire di testare la maggior parte del software su un computer moderno prima di installarlo sul computer di destinazione.

Dare un'occhiata a “www.Asm80.com”, che fornisce un editor e assemblatore in linea adatto all'uso con Small Computer Monitor. Un tutorial per aiutarti a iniziare a scrivere programmi con Asm80.com da utilizzare con il monitor per piccoli computer può essere trovato su www.scc.me.uk.

Una volta scritto e assemblato il programma su un computer moderno, si può scaricarlo sul computer di destinazione per eseguirlo e testarlo su hardware reale. Gli assemblatori di solito possono produrre un file Intel Hex contenente il programma in codice macchina. Questo file può essere caricato sul piccolo computer di destinazione utilizzando lo stesso programma terminale che già utilizzato. Cercare semplicemente un'opzione come “Invia file di testo”. Small Computer Monitor riconosce un file Intel Hex non appena arriva e lo decodifica, scrivendo il codice macchina in memoria mentre lo fa. Una volta scaricato completamente, si può eseguire il programma con il comando Go. Quindi si avranno a disposizione tutte le funzionalità di Small Computer Monitor per testare ed eseguire il debug del programma.

Questi assemblatori non sono specifici di Small Computer Monitor, pertanto non sono documentati in dettaglio qui.

Trovare errori

Se non si vede il messaggio di accesso del monitor sul terminale quando si accende il sistema, ecco alcune cose da provare:

Premere il pulsante di ripristino del sistema.

Controllare l'alimentazione, controllare tutti i ponticelli, controllare che nessun chip abbia i piedini piegati e quindi non faccia contatto con lo zoccolo, ispezionare attentamente tutte le saldature, controllare che tutti i chip siano inseriti nel verso giusto, controllare che tutti i componenti siano al posto giusto. Controllare che la connessione seriale sia corretta e che il terminale sia impostato correttamente. Quindi piangere!

Non dimenticare di seguire la guida alla risoluzione dei problemi fornita dal produttore.

LiNC80

Ricontrollare i ponticelli per i segnali ROM A14 e A15, in particolare se si utilizza un chip come il 28C256 che ha una piedinatura inusuale per questi segnali. Inoltre ricontrollare i ponticelli seriali poiché la sorgente del clock determina la velocità di trasmissione iniziale.

RC2014

Se il sistema RC2014 non è stato precedentemente testato con la ROM BASIC fornita, se possibile verificare che funzioni con la ROM BASIC. Se ciò non è possibile, eseguire tutti i consueti processi di ricerca guasti, descritti di seguito.

Se è noto che il sistema RC2014 funziona con la ROM BASIC fornita, verifica che la ROM di Small Computer Monitor contenga il codice corretto e controlla i ponticelli relativi all'indirizzamento della ROM (specialmente se il chip ha una capacità diversa da quello contenente BASIC). A parte questo, sembrerebbe che ci sia uno strano problema poiché la ROM del monitor dovrebbe, in teoria, funzionare se la ROM BASIC standard RC2014 funziona.

Va notato che sono disponibili numerosi moduli seriali diversi per RC2014 e non tutti sono compatibili. Attualmente Small Computer Monitor funziona solo con moduli seriali ufficiali RC2014 o con moduli totalmente compatibili con questi, e anche con moduli SIO/2 seguendo l'ordine di indirizzamento progettato da Grant Searle.

Parti e fornitori

Di seguito è riportato un elenco di parti e fornitori utilizzati durante lo sviluppo di Small Computer Monitor.

Moduli ufficiali LiNC80

È possibile trovare informazioni sul LiNC80 e sui suoi accessori alla pagina del negozio in cui è possibile acquistare i kit a <http://linc.no/go/linc80>.

Moduli ufficiali RC2014

Informazioni a www.rc2014.co.uk

Parti acquistate tramite Tindie:

[https://www.tindie.com/stores/Semachthemonkey/?](https://www.tindie.com/stores/Semachthemonkey/?ref=offsite_badges&utm_source=sellers_Semachthemonkey&utm_medium=badges&utm_campaign=badge_medium)

[ref=offsite_badges&utm_source=sellers_Semachthemonkey&utm_medium=badges&utm_campaign=badge_medium](https://www.tindie.com/stores/Semachthemonkey/?ref=offsite_badges&utm_source=sellers_Semachthemonkey&utm_medium=badges&utm_campaign=badge_medium)

Programmatori di chip

WINGONEER TL866CS Universal USB MiniPro EEPROM FLASH BIOS
Programmer AVR GAL PIC SPI

Amazon ASIN: B071H5XGR7

<https://www.amazon.co.uk/gp/product/B071H5XGR7/>

[ref=oh_aui_detailpage_o00_s00?ie=UTF8&psc=1](https://www.amazon.co.uk/gp/product/B071H5XGR7/?ref=oh_aui_detailpage_o00_s00?ie=UTF8&psc=1)

EEPROM 8k x 8 bit

Microchip Technology AT28C64B-15PU Parallel EEPROM Memory, 64kbit, 150ns, 4.5 → 5.5 V PDIP 28-Pin

RS part number: 127-6572

<http://uk.rs-online.com/web/p/eprom-memory-chips/1276572/>

EEPROM 32k x 8 bit

Microchip Technology AT28C256-15PU Parallel EEPROM Memory, 256kbit, 150ns, 4.5 → 5.5 V PDIP 28-Pin

RS part number: 127-6570

<https://uk.rs-online.com/web/p/eprom-memory-chips/1276570/>

FTDI cable

TTL-232R-5V - USB to Serial Converter Cable, 5V, 6Way, 1.8m

Farnell order code: 2419945

http://uk.farnell.com/ftdi/ttl-232r-5v/usb-to-serial-converter-cable/dp/2419945?ost=2419945&isrfrnonsku=false&ddkey=http%3Aen-GB%2FElement14_United_Kingdom%2Fsearch

FTDI 'cable'

HALJIA FT232RL FTDI USB to TTL Serial Converter Adapter Module Mini USB 3.3V 5.5V Board for Arduino
Amazon ASIN: B06XDH2VK9
https://www.amazon.co.uk/gp/product/B06XDH2VK9/ref=oh_aui_detailpage_o00_s00?ie=UTF8&psc=1

USB-RS232 cable

UGREEN 20210 USB Serial Cable, USB to RS232 DB9 9 pin Converter Cable
Amazon ASIN: B00QUZY4UG
https://www.amazon.co.uk/gp/product/B00QUZY4UG/ref=oh_aui_search_detailpage?ie=UTF8&psc=1
Nota: è necessario anche un cavo null modem tra questo e LiNC80 o RC2014.

Informazioni di contatto

Per contattarmi riguardo Small Computer Monitor usare la pagina di contatto a www.scc.me.uk.

Stephen C Cousins, Chelmsford, Essex, United Kingdom.

LiNC80

I problemi relativi al LiNC80 possono essere pubblicati sul gruppo Google "LiNC80".

È possibile trovare informazioni sul LiNC80 e sui suoi accessori alla pagina del negozio in cui è possibile acquistare i kit all'indirizzo <http://linc.no/go/linc80>

RC2014

Questioni relative all'RC2014 possono essere pubblicati sul gruppo Google "RC2014-Z80".

Informazioni riguardo RC2014 si trovano a www.rc2014.co.uk

Kits sono disponibili da www.tindie.com