17 Luglio 2015 **VOLUME 1, N°7**

Questa newsletter italiana nasce da un accordo avuto con Chris Schneider che ha acconsentito alla traduzione italiana della sua *Newsletter Shift838*. Vuole essere una nuova risorsa per i computer **TI-99/4A** e **GENEVE 9640**, in modo da divulgare e mantenere aggiornati tutti gli utenti che ancora oggi si interessano a questi computer.

In questa edizione della newsletter avremo un colloquio con ***Harry Wilhelm***, progettista e creatore di alcuni strumenti per lo sviluppo in linguaggio Extended Basic molto validi come ad esempio: ***il Compiler*** (compilatore da Basic/XB in Assembly ndt), l'***XB256*** (variante dell'Extended BASIC ndt) e ***The Missing Link*** (un ottimo potenziamento dell'Extended BASIC ndt). Questi strumenti hanno migliorato l'effettiva realtà di sviluppo dei programmatori. Oltre ad una spiegazione dettagliata dei suoi strumenti, Harry, è stato così gentile da entrare nel dettaglio su ciò che accade dietro le quinte del suo COMPILER 256.

Altri argomenti trattati in questo numero:

- **Aggiungiamo un pulsante di Reset al nostro TI-99/4A;**
- **Softworx BBS v1.0;**
- **Rileviamo una portante con un piccolo programmino;**

Buona Lettura !

Ciro Barile

Intervista a:
Harry Wilhelm

D: Puoi spiegare in dettaglio cosa permette di far fare ai programmatori ciascuno dei programmi sotto riportati?

Harry: Prima di tutto, voglio ringraziarti per avermi dato l'opportunità di parlare di questi programmi. Come è mia abitudine, la mia filosofia nello scrivere per l'XB è sempre stata quella di sviluppare programmi che potessero essere eseguiti su un sistema TI "standard" cioè con un TI99/4A con l'Extended BASIC, 32K di memoria e un disk drive. Non è necessaria alcuna versione speciale di XB, nessun potenziamento della scheda grafica (come la scheda F18A), nessuna espansione di memoria oltre alla solita da 32K. Mi piace da matti vedere cosa attualmente si può fare con le apparecchiature che c'erano nel 1981. Ora, risponderò alle tue domande partendo dall'inizio:

The Missing Link

Nel 1989 circa, quando tutti mi dicevano che non era possibile, feci qualche esperimento per verificare la possibilità di poter usare la modalità grafica bitmap con l'Extended BASIC. L'idea era di creare un programma che utilizzasse la routine di interrupt alla locazione >83C4 per rilevare automaticamente all'avvio di un programma XB l'impostazione dei registri VDP per bitmap e cancella lo schermo.
La routine di interrupt tiene inoltre traccia degli stack pointer e li ripristina in base alle esigenze. Un'altra complicazione è la presenza di due locazioni di memoria che sono pesantemente utilizzate dall'XB (>0370 e >0820). Queste si posizionano nel mezzo della color table o nella screen image table senza possibilità di essere spostate, così è necessario mappare la screen image table in un modo inusuale per evitare di scompaginare l'immagine.
La prima subroutine è stata scritta in assembly semplicemente per tracciare pixel sullo schermo, ma è stata una vera emozione vedere un programma XB tracciare una linea in diagonale sullo schermo, ritornare al graphic mode quando il programma si interrompe, poi riprendere il plot con CON. Dopo molte prove, quando ero ormai sicuro che fosse stabile e affidabile, decisi di ampliarlo con altre subroutine in assembly tese a riempire interamente gli 8K della low memory. Nel 1990 *Texaments* pubblicò il mio ***The Missing Link***.

I programmi sono scritti in Extended BASIC, il che li rende facili da scrivere e facili da capire, non è richiesta la conoscenza del linguaggio assembly per utilizzare pienamente The Missing Link. Poiché nessuno dei normali modi nell'XB permettono di accedere allo schermo in modalità bitmap, queste subroutine sostituiscono i soliti comandi dell'XB per accedere allo schermo.

The Missing Link comprende subroutine di testo che ti consentono di immettere un informazione o di visualizzarla sullo schermo. Vi è un a capo automatico quando il testo viene visualizzato e lo stesso testo può anche essere visualizzato in verticale. Non c'è il limite ai caratteri del tipo 8x8 o 6x8; pertanto possono essere utilizzati font di diverse dimensioni fino al carattere 4x6 che permette di avere sullo schermo 32 righe per 60 colonne. Poiché è **tutto completamente in bitmap**, possono essere visualizzati contemporaneamente sullo stesso schermo testi di varie dimensioni.

I grafici cartesiani ti permettono di tracciare punti, linee, cerchi e quadri. La Turtle Graphic (v. LOGO) può essere utilizzata senza nessuna delle restrizioni di inchiostro e colore che si trovano in LOGO. La grafica degli sprite può gestire fino a 32 sprite in movimento sullo schermo. Non ci sono limiti quando si combinano grafica e testo sullo schermo.
E' permesso definire una finestra da utilizzare come casella di testo. Il testo viene sempre visualizzato all'interno della finestra; i grafici possono essere visualizzati all'interno o all'esterno della finestra.
Le immagini possono essere caricate o salvate nel formato standard TI-Artist ed è sempre disponibile un dump della grafica dello schermo.

Per avere un'idea di ciò che è possibile, con il comando RUN "DSK1.TMLDEMO" viene eseguito un programma dimostrativo che mostra tutto ciò che il TML (The Missing Link) è in grado di fare.
Un'altra parte del pacchetto è un tutorial chiamato "***Potatohead***" che mostra come scrivere un semplice programma simile a "*Facemaker*" per il TI o "*Mr. Potatohead*" per coloro che amano giocare in cucina. In caso si decida di provare il TML, mi raccomando di studiare *Potatohead* per ottenere una maggior e più profonda comprensione di come si usa. Se invece si vuole partire comunque senza leggere il manuale, ecco un semplice programma che disegna una serie di cerchi che si allargano via via sullo schermo:

RUN "DSK1.TML" - TML è caricato ed attivato (Schermo verde e cursore Texas).

```
10 FOR I = 1 TO 100 STEP 5 :: CALL LINK("Circle",I,I,I) :: NEXT I
20 GOTO 20
```

XB256

XB256 condivide la sua filosofia di design con TML. Come TML, è una raccolta di subroutine in linguaggio assembly, ma piuttosto che la grafica bitmap, con XB256 l'attenzione è rivolta sfruttando appieno la modalità Normal Graphic Mode. Anche in questo caso, non è necessaria nessuna conoscenza del linguaggio assembly per utilizzare *XB256.*
XB256 consente di scegliere tra due schermi indipendenti. Lo Screen1 è la schermata di default, che è la schermata normalmente usata dall'Extended BASIC. Vi si accede utilizzando le solite chiamate alla grafica dell'XB. Lo Screen2 è completamente indipendente da Screen1. Consente di definire 256 caratteri, rispetto ai 112 normalmente disponibili a XB. Inoltre, è possibile utilizzare fino a 28 sprite di dimensioni doppie utilizzando i pattern disponibili per lo Screen1. È possibile passare da una all'altra delle due schermate quando lo si desidera, preservando la grafica su ognuno dei due schermi. Ciò potrebbe essere utile per usare una schermata di aiuto o di menu. Quando si utilizza Screen2 ci sono delle subroutine in assembler che sostituiscono CHAR, CHARPAT, COLOR, e CHARSET. Fatta eccezione per queste subroutine, tutti gli accessi allo schermo in Screen2 sono eseguiti con le solite dichiarazioni dell'extended BASIC come PRINT, SPRITE, ACCEPT, ecc.

Ci sono routine di scrolling che permettono di far scorrere i caratteri sullo schermo a sinistra, a destra, in alto o in basso. È possibile specificare una area della finestra per lo scrolling e lasciare il resto dello schermo invariato. Altre routine consentono di far lo scrolling agevolmente di un pixel alla volta verso sinistra, verso destra, su o giù. Ancora un'altra routine permette di fare una ricerca per far "scivolare" un testo come nella scena del titolo di "Star Wars". C'è una miscellanea di varie subroutine che consentono di evidenziare il testo, impostare lo sprite per l'orologio, la stampa sullo schermo utilizzando tutte le 32 colonne, leggere o scrivere sulla RAM VDP, scrivere stringhe compresse sul VDP, spostare tabelle sonore nel VDP, riprodurre un elenco dei suoni, congelare e scongelare il movimento dello sprite, catalogare un disco e altro ancora.

E' incluso un programma di utilità (COMPRESS) che permette di salvare le aree selezionate della memoria VDP in stringhe compresse che possono poi essere fuse in un programma. È possibile salvare le definizioni di carattere, le tabelle suoni, le immagini dello schermo, ecc. in una forma compatta che può essere caricata molto più rapidamente.

Un'altra caratteristica utile è la possibilità di eseguire liste sonore. Le liste dei suoni sono un modo molto compatto per salvare la musica e gli effetti sonori in un

formato compresso che può essere caricato direttamente nella memoria VDP e riprodotto automaticamente mentre il programma fa altre cose. La lista suoni del player nell'XB256 può esegire due liste sonore contemporaneamente, così si può avere un sottofondo musicale continuo in un gioco e si possono aggiungere effetti speciali come esplosioni sopra una musica. Non c'è nessuna necessità di imparare l'assembly per creare la lista sonora. Partendo da un programma XB che crea la musica con dichiarazioni CALL SOUND, ci sono due programmi di utilità, SLCOMPILER e SLCONVERT che possono essere usati per convertire le dichiarazioni CALL SOUND in una lista sonora e che possono poi essere fuse (con il comando MERGE) in un programma XB.

COMPILER 256

Anche se gli appassionati di Forth potrebbero essere in disaccordo, il linguaggio Extended BASIC è probabilmente il più versatile dei linguaggi disponibili per il TI99/4A. I programmi sono facili da scrivere, relativamente comprensibili e semplici da modificare ed editare, con un sacco di controlli degli errori che facilitano lo sviluppo del programma. Lo svantaggio principale è che la natura di doppio interprete dell'Extended BASIC (BASIC e GPL) lo rende estremamente lento. L'intento nello scrivere il mio compilatore per l'Extended BASIC è stato quello di permettere di trarre il massimo vantaggio dallo sviluppo semplice del programma offerto da XB, poi trovare una scappatoia per le limitazioni alla velocità. L'obiettivo era quello di implementare sull'Extended BASIC quanto più possibile entro i limiti di tempo del programmatore e di memoria della macchina. Ci sono dei limiti e probabilmente sarà necessario che modifichiate un po' il vostro stile di programmazione, ma in generale, tutte le caratteristiche principali dell'XB sono incluse nel compilatore. Ciò significa che è possibile concentrarsi sulla scrittura del codice XB e fare i test sempre nell'ambiente XB. Dopo che il programma è stato messo a punto in Extended BASIC, può in seguito essere compilato in un codice equivalente che funziona ad una velocità prossima a quella del linguaggio assembly. Un programma medio in extended BASIC verrà eseguito circa **20** volte più velocente dopo essere stato compilato e alcune operazioni diventeranno fino a **70** volte più veloci.

Esistono diversi metodi con cui il compilatore raggiunge questo incremento di velocità. In primo luogo, l'Extended BASIC deve eseguire una lunga operazione di scansione preliminare prima che un programma possa iniziare. Questa operazione viene fatta in anticipo dal compilatore e diventa parte del codice compilato. In secondo luogo, un programma XB è interpretato due volte dal computer; una volta dall'interprete dell'Extended BASIC e una seconda volta dall'interprete GPL. Il compilatore genera un "*codice sequenziale*" che ha bisogno di un proprio interprete

(chiamato *runtime routine*), ma così è coinvolto un solo interprete, ed è più veloce! In terzo luogo, viene dovunque utilizzata l'aritmetica dei numeri interi invece dell'aritmetica in virgola mobile. Questa caratteristica da sola fa eseguire il codice almeno 5 volte più velocemente, certamente senza la versatilità delle 13 cifre della precisione dei numeri in virgola mobile. Quarto, per aumentare la velocità ancora di più, praticamente non si fa diagnostica per gli errori. Eventuali segnalazioni di errori che vengono evidenziati non sono molto utili in ogni caso, perché non si conosce il numero di riga dove è avvenuto l'errore. Pertanto è indispensabile che sul programma in Extended BASIC sia stato accuratamente eseguito il debug prima di tentare di compilarlo!

Il compilatore è stato ampliato per includere tutte le estensioni in linguaggio assembly dell'XB256.

È possibile verificare il programma nell'ambiente XB o XB256, quindi utilizzare il compilatore per ottenere un enorme aumento delle prestazioni. Il compilatore è stato testato con un vero e proprio TI-99/4A, con Classic 99 e con Win994A ed è compatibile con tutti. Un programma può essere salvato come programma XB o, in alternativa, come programma in EA5.

L'XB256 e il COMPILER256 danno il meglio quando vengono utilizzati insieme e la combinazione è l'XB Program Developer's Package. Anche se sono stati progettati per completarsi a vicenda, entrambi questi sono delle utility standalone. I programmi sviluppati utilizzando XB256 non devono essere compilati. Se il tempo di esecuzione è adeguato, verranno ben eseguiti in XB e si dovrebbe avere accesso alla matematica a virgola mobile, all'accesso al disco, ecc. Allo stesso modo, i programmi XB non devono usare XB256; possono ancora essere compilati per incrementare la velocità.

Ora devo dare una notizia sgradevole: se si dovesse decidere di utilizzare uno di questi programmi, mantenete il manuale chiuso a portata di mano e fate riferimento ad esso spesso. Se lo si farà, si potrà avanzare molto più velocemente!

D: Ci sono stati parecchi interrogativi sul compilatore e come effettivamente funziona. Puoi descrivere (ad un buon livello) in pratica cosa sta facendo quando genera il nuovo codice?

Harry: Idealmente un compilatore dovrebbe prendere un programma in XB, convertirlo direttamente in un programma in linguaggio assembly pronto per essere eseguito e salvarlo su disco. Ma con questo approccio il debug sarebbe molto difficile da scrivere ed da eseguire, almeno per me. Per cercare di preservare

la mia sanità mentale, il mio approccio è stato quello di suddividere il compilatore in due parti. Il primo è il compilatore effettivo che esegue la conversione. Questo è un programma ibrido XB/assembly che converte un programma XB in formato MERGE in quello che viene chiamato "codice oggetto sequenziale (threaded object code)". Questo codice deriva strettamente dal programma XB originale. Come esempio di come funziona, il programmino qui sotto stampa "HELLO WORLD!!" sullo schermo utilizzando il metodo TI BASIC; quindi segue la versione compilata:

```
10 A$="Hello World!!"
15 CALL CLEAR
17 FOR R=1 TO 19
20 FOR I=1 TO LEN(A$):: CALL HCHAR(R,R+I,ASC(SEG$(A$,I,1))):NEXT I
30 NEXT R
40 GOTO 40
```

Ecco ciò che viene fuori quando il compilatore converte il programma. I miei commenti e il programma XB sono in corsivo. Notate come il compilatore suddivide un comando complicato come CALL HCHAR in elementi semplici:

```
DEF RUN,CON
RUNEA  B @RUNEA5                    Usato solo da un programma EA5
FRSTLN                              La prima riga del programma.
L10                                 10 A$="Hello World!!"
     DATA LET,SV1,SC1               A$ (SC1) è memorizzato in SV1
L15                                 15 CALL CLEAR
     DATA CLEAR
L17                                 17 FOR R=1 TO 19
FOR1
     DATA FOR,NV1,NC1,NC2,ONE,0,0   set up per il ciclo for/next. R è ora
                                     NV1
L20                                 20 FOR I=1 TO LEN(A$):: CALL
                                     HCHAR
                                     (R,R+I,ASC(SEG$(A$,I,1))):NEXT I

     DATA LEN,SV1,NT1               Lunghezza di A$ (SV1), messa in NT1
                                     Per utilizzarla nel ciclo
FOR2
     DATA FOR,NV2,NC1,NT1,ONE,0,0   Set up per il ciclo For/Next; I è ora
                                     NV2
```

(Qui il compilatore suddivide la linea in passi più semplificati:)

```
     DATA SEGS,SV1,NV2,NC1,ST1      Prende SEG$ di A$, lo memorizza in ST1
     DATA ASC,ST1,NT1               Memorizza ASC di ST1 in NT1
     DATA ADD,NV1,NV2,NT2           somma NV1 a NV2, memorizza in NT2
     DATA HCHAR,NV1,NT2,NT1         Ora è semplice abbastanza fare HCHAR
     DATA NEXT,FOR2+2
L30                                 30 NEXT R
     DATA NEXT,FOR1+2
L40                                 40 GOTO 40
     DATA GOTO,L40
LASTLN DATA STOP
OPTBAS DATA 0
NC0
ZERO   DATA 0
ONE    DATA 1
PI     DATA 3
```

```
RND    DATA 0
NC1    DATA 1
NC2    DATA 19
NV0
NV1    DATA 0 R
NV2    DATA 0 I
NT1    DATA 0
NT2    DATA 0
SC0
SC1    DATA SC1+2                    Converte "Hello World!!" in bytes; in questo
                                     modo qualsiasi codice ASCII può essere utilizzato
                                     in una stringa.
       BYTE 13,72,101,108,108,111,32,87,111,114
       BYTE 108,100,33,33
       EVEN
SV0
SV1    DATA 0 A$
ST1    DATA 0
SA0
NA0

FRSTDT
LASTDT
       EVEN
       COPY "DSK1.RUNTIME1"          copia le runtime routines
       COPY "DSK1.RUNTIME2"
       COPY "DSK1.RUNTIME3"
       COPY "DSK1.RUNTIME4"
       END
```

Si può vedere che il programma compilato segue da vicino il programma XB originale, il che rende più facile il debug. Ora è il momento di fare qualcosa con il codice che è stato creato. C'è un ciclo principale che è la chiave per eseguire il codice:

```
STAR8  LI R13,FRSTLN       R13 punta all'indirizzo nel codice compilato, si parte dalla prima linea
********************       poi in seguito:
RTN    LIMI 2              ciclo principale, tutte le subs tornano qui
       LIMI 0
       MOV *R13+,R12       sposta l'indirizzo del codice in R12 e somma 2 a R13
       B *R12              branch al codice puntato da R12 ovvero GOTO, Sprite ecc.
                           Ritorno è fatto con B @RTN
```

I sottoprogrammi nelle routine di runtime aggiornano automaticamente R13 così quando ritornano puntano direttamente all'istruzione successiva. Ecco la subroutine LEN che fa parte di RUNTIME2:

```
LEN    BL @GET2            recupera le due parole seguenti LEN, e le mette in R5 e R6,
                           aggiunge 4 to R13
       MOV *R5,R5          ora R5 punta alla stringa (il primo byte è la lunghezza in byte)
LEN1   MOVB *R5,R4         muove il byte lunghezza nel MSB of R4
       SRL R4,8            sposta il byte lunghezza in LSB
       MOV R4,*R6          riporta la lunghezza all'indirizzo puntato da R6
       B @RTN              tutto fatto, torna indietro. GET2 aggiorna R13 to puntare
                           alla prossima istruzione.
```

Quindi, in pratica le routine di runtime recuperano l'indirizzo di ogni istruzione compilata (SEG$, LEN, PRINT,CALL HCHAR e tutto il resto). L'istruzione fa un "branch". Se i parametri sono stati tarsmessi allora l'R13 deve essere aggiornato in modo che, quando viene eseguito B@RTN (il branch al RETURN) ritorna con R13 puntato sull'istruzione successiva. Questa è una breve panoramica di come lavora il compilatore; si spera che sia abbastanza semplice da seguire. Come ci si potrebbe aspettare le routine di runtime possono ottenere gestire operazioni molto più complesse rispetto al semplice esempio di cui sopra, ma tutte sono connesse (con un'istruzione di branch) al il ciclo principale e ivi ritornano.

D: Sono certo che ci sono sicuri vantaggi sulla velocità con il compilatore. Con un programma tipico in che percentuale si potrebbe valutare il miglioramento della velocità di esecuzione?

Harry: Come menzionato sopra, l'aumento di velocità è solitamente di almeno 20 volte e talvolta 50 volte o più. Inoltre, i programmi si avviano immediatamente senza aspettare che avvenga la scansione preliminare. Il mio TI99 reale ha 32K sul bus a 16 bit, in modo che i programmi compilati funzionino ancora più velocemente dal 20 al 25%. Durante la scrittura di giochi, di solito, si devono aggiungere loop di ritardo in modo da rallentarli abbastanza per rendere il gioco giocabile.

D: So che ci sono limitazioni sul file di I/O e per il compilatore c'è l'intenzione di affrontarle e se sì, quando?

Harry: Il compilatore è stato aggiornato e ora può gestire l'accesso al disco. Solo il comando DISPLAY, VARIABLE è per ora supportato, ma è possibile utilizzare qualsiasi lunghezza da 1 a 254 caratteri. Altri tipi di file potrebbero essere supportati; qualcuno del framework c'è già. Dovrebbe essere solo una questione di espansione delle routine di runtime (RUNTIME7 per l'esattezza). Ricordate che le routine di runtime diventano parte del programma compilato. Ogni byte aggiunto per loro significa un byte in meno disponibili per il vostro programma. Sto cercando di trovare un equilibrio tra completezza e l'avere spazio sufficiente per il programma.

D: Ci sono problemi di compatibilità per qualcuno dei programmi citati precedentemente alla domanda #1, e se sì quali?

Harry: L'unico problema di compatibilità di cui sono a conoscenza è con *The Missing Link,* con cui non è possibile avere un qualsiasi accesso al disco usando l'espansione CF7. Se avessi una CF7 il problema potrebbe probabilmente essere risolto, ma i miei sforzi per farlo senza questo device non hanno avuto successo.

D: Ci sono delle restrizioni che un utilizzatore deve conoscere prima di utilizzare uno di questi programmi?

Harry: Le maggiori restrizioni sono nel compilatore. Non tutto l'XB è stato implementato, per cui, se si tentasse di compilare un programma XB con istruzioni non supportate, avreste dei problemi. Si è limitati ad usare solo l'aritmetica integrata, non è possibile utilizzare i sottoprogrammi del tipo CALL [*name*], matrici nidificate, funzioni trigonometriche, le CALL SPEECH, il comando DISPLAY, DEF e poche altre cose. L'istruzione IF THEN ELSE lavora con numeri di riga (come BASIC), non con la più complessa forma possibile in XB. Queste cose comunque sono tutte dettagliate nel manuale. E' meglio durante lo sviluppo di un programma da compilare, mantenere a mente queste restrizioni, piuttosto che convertire un programma già esistente. In caso di una compilazione di un programma già scritto in XB, si dovrà probabilmente analizzarlo e vedere se sono presenti, nel codice, istruzioni non supportate. Alcuni programmi sono più facili da modificare rispetto ad altri - ho modificato con successo e ho compilato il programma XB "PORTAL", che ora ha una bella risposta scattante. Gli attuali programmi in TI BASIC sono più semplici da compilare senza modifiche rispetto ai programmi XB.

L'intero programma compilato viene caricato nella memoria alta così come le routine di runtime e le variabili stringa, che normalmente sono contenute nella RAM VDP. A compensazione di tutto ciò c'è che il codice compilato è di solito circa 2/3 della dimensione del programma XB originale. Quindi, se avete un grande programma di XB, dovrebbe essere meglio compilarlo.
The Missing Link utilizza la maggior parte della RAM VDP per la grafica. La RAM disponibile per la programmazione è la stessa, ma lo spazio di stack nella RAM VDP è molto limitata. Le stringhe sono conservate nello stack e se ne avete tante sarà possibile eseguirle facilmente fuori dallo stack. C'è una sezione nel manuale che fornisce informazioni su come risparmiare spazio dello stack.

D: Dove si possono ottenere tali programmi?

Harry: Questi sono disponibili in due siti:

Atariage TI99 forum
http://atariage.com/forums/forum/119-ti-994a-development/
(presenti nella pagina TI-99/4A Development Resources quella in evidenza)

http://www.99er.net
(in: Download Database → Utilities)

Aggiungiamo un pulsante RESET al TI-99/4A

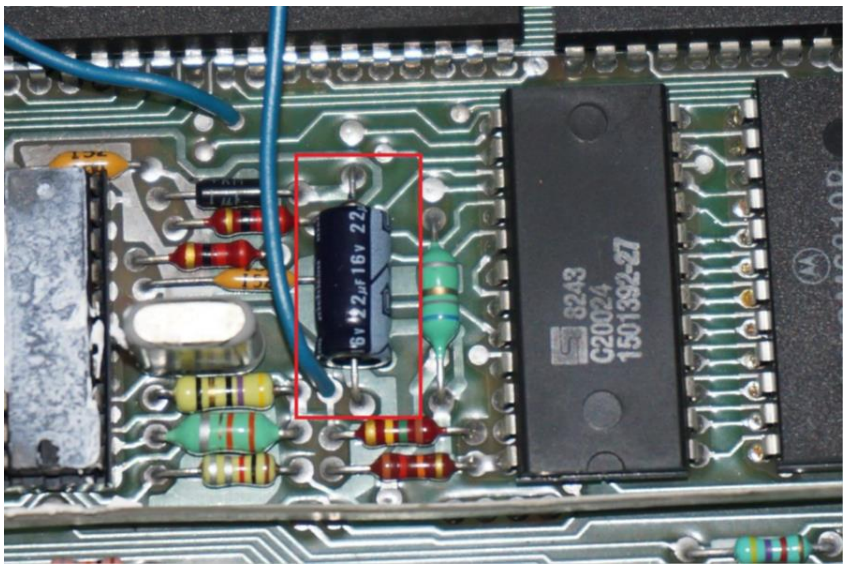
Questo argomento è stato sollevato più volte così è stato deciso di inserirlo in questa newsletter. Gli utenti del TI-99/4A, come me, si sono spesso chiesti perché Texas Instruments non abbia progettato questo computer con un tasto reset. Infatti è risaputo che in caso di un blocco del sistema, l'unica soluzione a disposizione è quella di spegnerlo con l'interruttore di alimentazione. Naturalmente questo può causare un'usura anomala dell'interruttore stesso come già è capitato ad alcuni utilizzatori.
Dopo molte discussioni e la ricerca di come si sarebbe potuto fare la modifica al meglio, descrivo di seguito quella che secondo me è una buona soluzione per tutti gli utenti più e non; Infatti ho scelto questa soluzione perché come tale non richiede alcuna saldatura alla CPU o taglio di tracce sulla scheda madre ed è un aggiornamento facile per chiunque, anche per un principiante.

Elenco dei componenti:

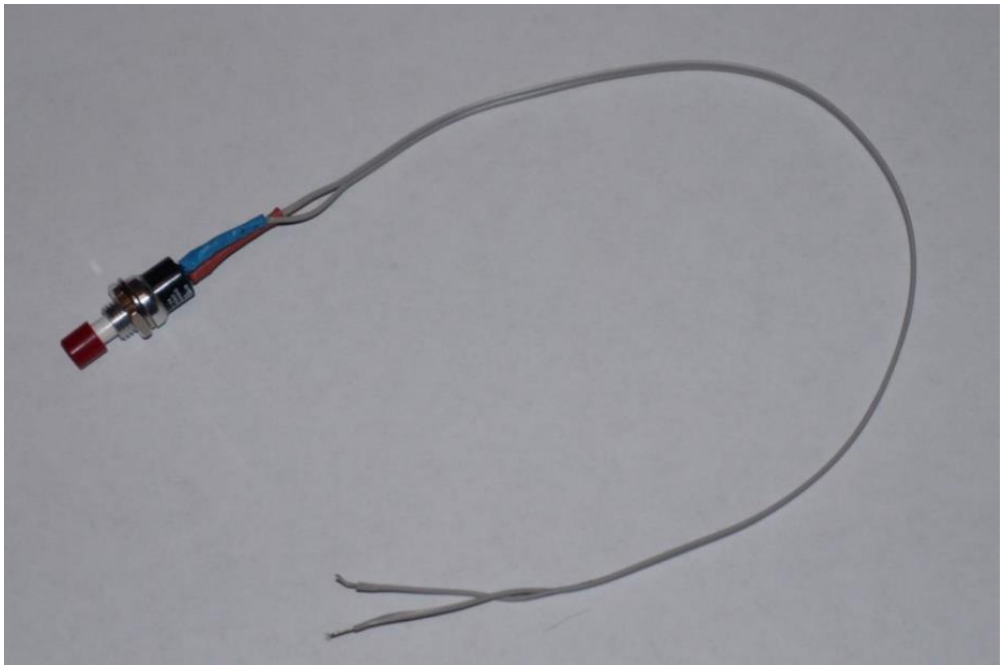
1 x interruttore a pulsante (normalmente aperto)
2 x 30cm di filo elettrico doppio
… e un pò di tubo termorestringente o di nastro isolante

Procedura di installazione:

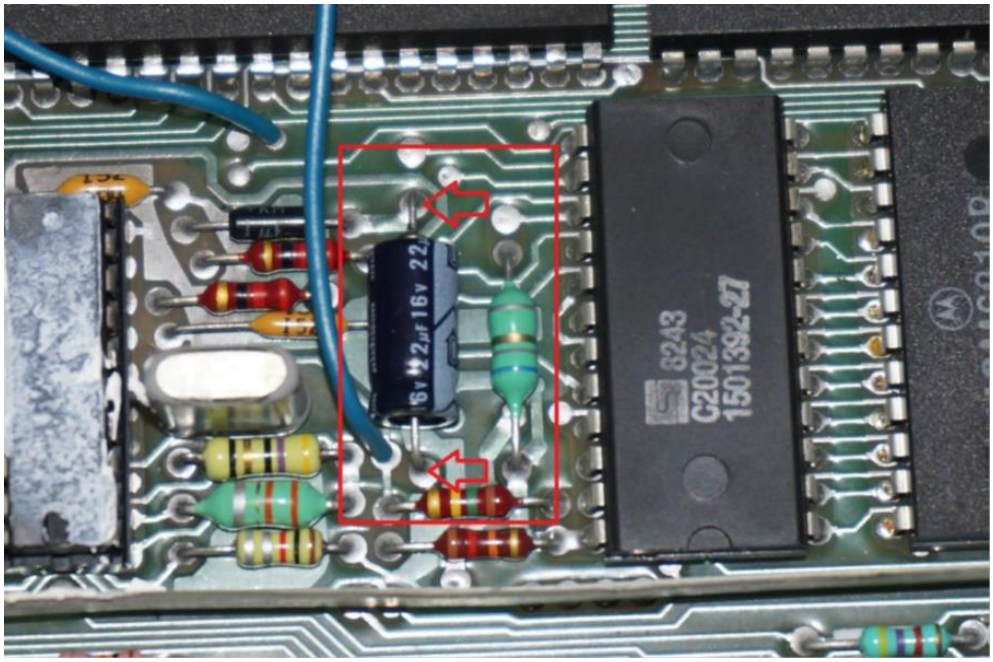
Fase 1: Smontare il computer
Fase 2: Individuare il condensatore C606 - *Questo condensatore si trova sul lato destro del cristallo grigio sotto il processore TMS9900*



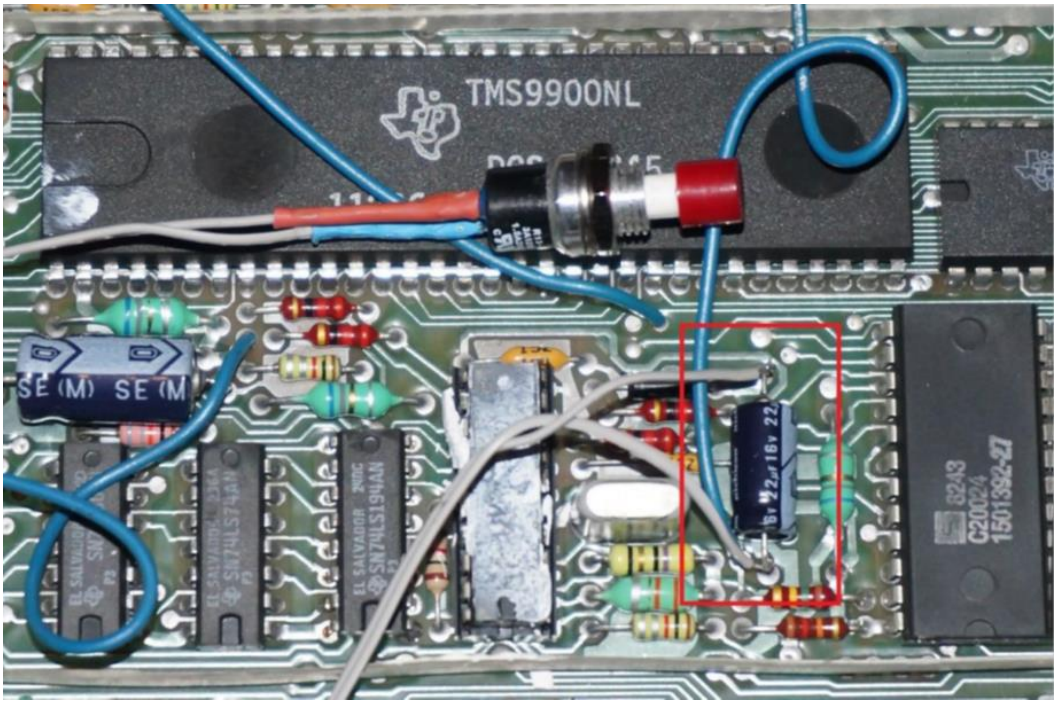
Fase 3: Salda i cavi al pulsante e usa 2 pezzi di tubo termorestringente per coprire i terminali.



Fase 4: Salda quindi i fili del pulsante al condensatore C606 (uno su ciascun lato del condensatore)



Una volta saldato dovrebbe apparire come l'immagine qui sotto.



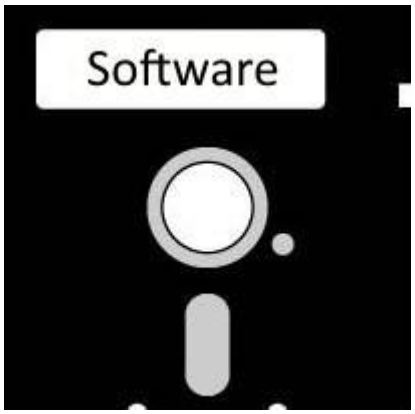
Fase 5: Fai un foro centrale sulla parte superiore posteriore destra della console per montare il pulsante. (Potrebbe essere necessario utilizzare una lima tonda per ottenere un foro su misura).

Fase 6: Monta il pulsante avvitando la ghiera

Fase 7: Rimonta la console

Si dispone ora di un pulsante di reset. È possibile montare l'interruttore ovunque e può essere di qualsiasi tipo purché 'normalmente aperto'. Se si dovesse scegliere di montarlo in un altro posto assicurarsi di utilizzare la giusta quantità di filo e che il pulsante non interferisca con la console in nessun modo.

Premendo l'interruttore si resetta la console senza dover usare l'interruttore di alimentazione.



Softworx BBS V1.0 - Questa è stata la prima BBS che ho gestito ed è stata scritta da *Mark Shields* che si occupava della USS Enterprise a Houston, TX. Attualmente sto cercando di rilanciare la sua ultima versione di **Softworx**.

Questa BBS deriva dall'originale Linguaggio Macchina Zyolog codificato da *Bryan Wilcutt*. Il codice XB fu riscritto da *Mark Shields* partendo da zero. Questa BBS supportava messaggi, file di testo e trasferimenti XMODEM e file ASCII.

La capacità di codifica di Mark fu eccellente e crebbe maggiormente con la progressiva comprensione del funzionamento del TI-99/4A.

Ho pensato che nei primi anni '80 Mark utilizzasse un metodo molto particolare per la rilevazione di una portante. Usò la porta joystick collegata al pin #8 del modem (DCD) al pin #9 della porta Joystick. Questo metodo non riporta nessun valore quando viene rilevata una portante ma restituisce un valore >0 se una portante non viene rilevata. Per una più approfondita delucidazione vedi la sezione ''Coding" qui di seguito.

Ricorda di non collegare il cavo del joystick finché non viene richiesto.

Questa BBS deve essere eseguita su un sistema TI reale. Ho verificato che non funzionerà con tutti gli emulatori né lavorerà con la scheda NanoPEB. Può essere eseguita da unità floppy, da RAMDisk e da dischi rigidi.

Il software può essere scaricato al seguente link:

ftp://ftp.whtech.com/Users/Chris_Schneider/BBS/SOFTWORX_V1.0

Ho anche messo assieme un piccolo documento per spiegare i *CALL LOAD* così come i *CALL LINK* che vengono utilizzati nel codice. Ho dovuto fare qualche prova per capire esattamente cosa stessero facendo visto che non ho più "pasticciato" con questo programma dalla metà degli anni '80.

Si può “giocare” con questa BBS, e ci sono tutti i presupposti per divertirsi. In caso aveste domande non esitare a contattare Chris Schneider che sarà lieto di rispondervi.

Puoi approfondire ulteriormente l’argomento “software BBS d’annata” esistenti per il TI-99/4A al link qui di seguito:

https://en.wikipedia.org/wiki/BBS_software_for_the_Texas_Instruments_TI-99/4A

L'esempio di codice seguente mostra un programma semplice per rilevare una portante. Ho verificato che funziona sia con un vero e proprio modem che con un dispositivo UDS usando hardware originale TI.

E’ necessario fare un cavo che collega il pin #9 della porta joystick al pin #8 sul retro del modem o del dispositivo UDS. Questo è il DATA CARRIER DETECT pin. Quando una portante è presente il valore sarà uno '0'. Quando i programmi non rilevano una portante verrà riportato un valore maggiore di zero.

Le linee da 100 a 170 in realtà servono solo per abbellire il programma cambiando il carattere “ – “ facendolo apparire come una linea continua su ogni riga.

NB: Nessun modem deve essere collegato e attivo durante l'esecuzione del programma.

```
100! MODEM CARRIER DETECT
105! ORIGINAL CONCEPT BY MARK SHIELDS
110! (C)OPYRIGHT 2015
120! CHRIS SCHNEIDER
130! SHIFT838
140 CALL CHAR (45,"00000000FF")
150 CALL CLEAR
```

Non dovrebbe essere presente nessuna portante a questo punto.

```
160 PRINT "PLUG IN JOYSTICK DCD CABLE!"
170 CALL KEY (0, K, S) :: IF K=49 OR K=80 THEN 170 ELSE 180
180 CALL CLEAR
190 DISPLAY AT (6,1): "----------------------------"
200 DISPLAY AT (8,1): "----------------------------"
```

Il codice qui sotto riportato comincia a verificare la presenza di una portante in continuazione. Ora è possibile collegarsi al modem da un'altra macchina e vedere il cambiamento di stato da **'NO CARRIER DETECTED**' a **'CARRIER DETECTED'**. Quando si sgancia la portante tornerà a **'NO CARRIER DETECTED'**

```
210 CALL JOYST(1, X, Y)
220 IF X>0 THEN 230 ELSE 250
230 DISPLAY AT (7,5): ""
240 DISPLAY AT (7,5): "NO CARRIER DETECTED" :: GOTO 210
250 DISPLAY AT (7,5): ""
260 DISPLAY AT (7,5): "CARRIER DETECTED"
270 GOTO 210
```

RESOURCES

Informazioni

Per contattarmi non esitate a visitare il mio sito e fare clic sulla scheda '**Contatti**'.

Argomenti per la Newsletter

Se volete partecipare alla stesura di questa newsletter e fornire argomenti per questa newsletter vi prego di contattarmi tramite il mio sito web.

Siti

Qui di seguito trovate le risorse in una manciata di siti che supportano i computer TI-99/4A e/o Geneve 9640. Non è certamente un elenco completo. Questa sezione sarà inclusa e aggiornata in tutte le prossime newsletter.

Siti Web / siti FTP

http://www.ti99iuc.it
http://www.atariage.com
http://shift838.wix.com/shift838
http://www.99er.net
http://www.harmlesslion.com
http://www.mainbyte.com
http://www.ninerpedia.org/
http://www.ti99-geek.nl/
http://www.turboforth.net/
ftp://ftp.whtech.com

new http://www.ti99hof.org/index.html
new http://www.ti99ers.org/unsung/

new http://ti99ers.org/modules/Inspire/remember.htm
contiene tutti gli storici TI-99ers che sono deceduti.

Lista Gruppi Yahoo

https://groups.yahoo.com/neo/groups/TI99-4A/info
https://groups.yahoo.com/neo/groups/TI994A/info
https://groups.yahoo.com/neo/groups/Geneve9640/info
https://groups.yahoo.com/neo/groups/turboforth/info

BBS attive

HeatWave BBS

Accesso: Dial-Up e Telnet
Sistema: Geneve 9640
Software: S&T BBS Software
Località: Arizona
Contenuto: biblioteche di file TI e Geneve, messaggi di base, porte giochi ed e-mail.
Telnet: www.heatwavebbs.com port 9640 dialup: 602-955-4491 @ 8-N-1

The Reef Hidden

Accesso: Dial-Up
Sistema: TI-99/4A modificato
Software: S&T BBS Software
Località: New York
Contenuto: biblioteche di file TI e Geneve, messaggi di base, porte giochi ed e-mail.

The Keep

Accesso: HTTP e Telnet
Sistema: Pentium 4 con sistema operativo Windows 2000
Software: Worldgroup BBS Software (fino a 256 connessioni utente)
Località: Tigard, Oregon
Contenuto: biblioteche di file TI e Geneve, messaggi di base, porte giochi, multi-utente e giochi multiplayer ed e-mail.
Telnet: www.thekeep.net porta 23 Web browser per http://www.thekeep.net
The Keep dispone di librerie di file TI, messaggi di base, e-mail, giochi porte, multi-utente e giochi multiplayer. The Keep ha anche una linea modem collegata con tutti coloro che desiderano contattare The Hidden Reef BBS da internet attraverso The Keep.
Semplicemente Telnet alla www.thekeep.net sulla porta 23, accedi a The Keep e quindi digita **/GO DIALOUT** nel menu principale, quindi D1 per la composizione verso The Hidden Reef. E' molto semplice.

Venditori

SHIFT838 – Fornisce componenti TI usati come li ha acquistati. Controllate spesso cosa ho disponibile. Un sacco di articoli possono essere riutilizzati da altri utenti TI.

Arcade Shopper - fornisce attrezzature TI vecchie e nuove, aggiornamenti e nuove piste PCB a www.arcadeshopper.com

Centri di riparazione

Richard Bell

Riparazioni disponibili su base limitata, si prega di contattare Richard a swim4home@verizon.net per conoscere i tempi di attesa prima di inviare qualsiasi componente da riparare

Tim

Riparazioni su hardware Myarc disponibili su base limitata. Contattare Tim a insane_m@hotmail.com per itempi di attesa o per richiedere il servizio.

TI-99 Italian User Club

www.ti99iuc.it