VOLUME 2 ISSUE 5 $1.75 OCTOBER 1986

"Serving 99'ers Since 1984"

THE SMART PROGRAMMER

Well, this month we begin with the second installment of D.C. Warren's Interrupts series. Each segment is even better than the one before and next month's article will be another that programmers will find both useful and enlightening.

Interrupts

Part 2

by D.C. Warren

The interrupt interface of the TMS9900 µP, as discussed in the Sept. '86 issue of *The Smart Programmer*, has one pin to signal interrupts and four other pins used for developing sixteen interrupt vector levels (remember, level zero is reserved for reset). For the sake of completeness we must mention the other two pins on the µP, which are specialized interrupt pins. The first is the RESET pin, which executes a level zero reset when the system is turned on or a cartridge is inserted. The second pin is the Non-Maskable Interrupt (NMI or also known as the LOAD pin) which is used by currently available load interrupt buttons and single-step hardware. The NMI vectors reside at the far end of memory at >FFFC. These latter two interrupts won't be discussed anymore since they aren't related to the Interrupt Service Routine (ISR) in console ROM.

Although the TMS9900 makes available fifteen of sixteen external interrupt levels to the system, only level one is implemented in the 99/4A. That means that the only mask instructions necessary to work with interrupts on the 4A are LIMI 1 and LIMI Ø. This may be confusing because TI recommends using LIMI 2 throughout much of their literature when interrupts are needed. Also, there is a set of level two interrupt vectors at >ØØØ8 in ROM. I will only *speculate* that the recommendation to use the level two mask was to maintain software compatibility with future products (such as a 99/7 or 99/8, perhaps). Also, the level two interrupt vectors in ROM may be there for software to use if screen-blanking is desired without having to enable interrupts (by using BLWP @>ØØØ8). It will be clear later that this is one of the functions of the level one and two routines.

The single interrupt implemented in the 4A is attached to the TMS99Ø1 Programmable Systems Interface chip. This IC performs several functions in the 4A but the function of interest to us is the interrupt interface section. The Video Display Processor (VDP) interrupt and the peripheral interrupts are both attached to the 99Ø1. Also, The 99Ø1 has an internal timer which can cause an interrupt. When any one of these three items cause an interrupt in the 99Ø1, the 99Ø1 in turn signals a level one interrupt to the 9900 µP. It is then the duty of the ISR in console ROM to interrogate the 99Ø1 in order to find out if the timer, VDP or peripheral caused the interrupt. Once this is resolved then a specific part of the ISR is executed.

Now that the hardware aspects of the interrupt interface have been discussed, it is time to take a look at the software part (the ISR) that we've been talking about. The following listing is documented source code for the ISR. One comment is that some sections of the code are hard to follow, notably the Auto Sprite Motion and Auto Sound Processing. This is because little information from TI is available explaining just how these pieces of code function. A good way to understand the Sprite Motion and Sound Processing parts of the ISR is to set up a small program in XBASIC and watch what happens to Sprites and Sound with the MG EXPLORER.

```
*******************************
* ISR Source Code             *
* Commented by D.C. Warren    *
*******************************

GPLWS   EQU  >83E0              GPL workspace
R3LB    EQU  GPLWS+7            LSB of GPL R3
R8LB    EQU  GPLWS+17           LSB of GPL R8
INTWSP  EQU  >83C0              Interrupt workspace
AR12LB  EQU  INTWSP+25          LSB of interrupt R12

SOUND   EQU  >8400              Sound generator port address
VDPRD   EQU  >8800              VDP read data port
VDPWD   EQU  >8C00              VDP write data port
VDPWA   EQU  >8C02              VDP write address port
WRVDP   EQU  >4000              Write bit mask for VDP writes
VDPSTA  EQU  >FC00              Offset to VDP status port [@VDPSTA(R15)=>8802]

TIME    EQU  >8379              VDP interrupt timer byte
MOTION  EQU  >837A              Number of sprites in motion
VDPST   EQU  >837B              Copy of VDP status byte
INTFLG  EQU  >83C2              Interrupt control byte
INTPTR  EQU  >83C4              ISR hook
SNDADD  EQU  >83CC              Address of sound list
STFLGS  EQU  >83CE              Sound byte counter

RSMOT   EQU  >0780              Sprite Motion List address in VDP
QSAML   EQU  >0480              Offset from SML to Sprite Attribute list in VDP
VDELTA  EQU  >2000              Constant for checking invalid sprite positions

PUTSTK  EQU  >0864 [These vary] Routine which stacks GROM address on SRAM stack
GETSTK  EQU  >0842 [on other  ] Routine which pops GROM address off SRAM stack
TIMER   EQU  >1404 [consoles!!] 9901 timer handler within cassette routine

H4000   EQU  >4000              Address of peripheral card validation byte
H400C   EQU  >400C              Address of peripheral card interrupt link

HAA     EQU  $+2                Validation flag comparison byte
H00     DATA >00AA              Zero mask byte
C1      DATA >0001              VDP/GROM sound processing bit mask
H20     DATA >0020              Bit mask comparison byte
FNCEQ   DATA >1100              Function and = key bit mask
*
*===============================================================
* LEVEL ONE INTERRUPT SERVICE ROUTINE
*  Check to see if bit 10 of GPLWS R14 is set.  If so then assume interrupt was
*  caused by the 9901 timer.  If not then check bit two of 9901 to see if
*  VDP caused interrupt.  If VDP did not cause interrupt then assume that a
*  peripheral did.
*
```

```
REMOTE LIMI 0                   Set mask to zero for non-interrupt entries
       LWPI GPLWS               Load GPL workspace
       CLR  R12                 Clear CRU base to point to 9901
       COC  @H20,R14            Is the 9901 timer flag set?
       JNE  TIM1                NO!
       B    @TIMER              YES! Branch to cassette routine area
TIM1   TB   2                   Did VDP cause the interrupt?
       JNE  VDPINT              YES! Jump to service VDP interrupt
*
*===============================================================
* PERIPHERAL INTERRUPT HANDLER
*  Service all peripheral interrupt routines in CRU base range >1000 to >1F00.
*  Peripheral interrupt routines may use GPL registers R1 through R10. Reserved
*  registers are used as follows:
*  R0  contains next interrupt routine link
*  R11 contains the return address for the interrupt service routine
*  R12 contains the CRU base of the peripheral
*  R13 contains the current GROM library read data address
*  R14 contains >01xx where xx is used by the GPL interpreter
*  R15 contains the VDP write address address
*  Also, the allocation of memory from >834A to >836D may be split between the
*  interrupt routine and the DSR.  Interrupt routine returns with a B *R11.
*
       LI   R12,>0F00           Initialize CRU base to start checking periphs.
       SBO  1                   Turn 9901 external interrupt bit off
ILOOP  SBZ  0                   Turn last peripheral off
       AI   R12,>0100           Increment CRU base to next peripheral
       CI   R12,>2000           Are we finished with all peripherals?
       JEQ  EMERG2              YES! Exit interrupt service routine
       SBO  0                   NO! Turn next peripheral on
       CB   @H4000,@HAA         Do we have a validation flag here?
       JNE  ILOOP               NO! Look at next peripheral slot
       MOV  @H400C,R2           Grab interrupt entry list pointer
LOOP1  JEQ  ILOOP               If no interrupt routine here then skip
       MOV  R2,R0               Save next entry pointer
       MOV  @>0002(R2),R2       Grab interrupt entry address
       BL   *R2                 Branch to interrupt routine
       MOV  *R0,R2              Restore next entry pointer
       JMP  LOOP1               Check for more interrupt routines in this card
EMERG2 B    @EMERGE             Exit interrupt service routine
*
*===============================================================
* VDP INTERRUPT HANDLER
*  Perform housekeeping on all VDP interrupt related items.  These include:
*  auto-sprite motion, auto-sound processing, quit key check, storage of VDP
*  status byte, screen blank function and user defined ISR hook routine.  The
*  first three; sprites, sound and quit are controlled by an interrupt flag
*  byte at >83C2.  If the MSBit (bit 0) of this flag is a one then servicing
*  of the above three items is skipped.  Otherwise bits 1,2,3 govern whether
*  or not each individual routine respectively is executed.  If the bit is a
*  zero then the routine is executed otherwise it is skipped and the next bit
*  is checked.  The remainder of the first mentioned items are within the level
*  two interrupt code and are always executed on a VDP interrupt.
*
VDPINT SBO  2                   Reset the VDP interrupt bit in 9901
       MOVB @INTFLG,R1          Fetch interrupt flag byte from Scratch RAM
       SLA  R1,1                Check first bit
       JNC  TSTMOT              If not set then service sound, sprites and quit
       B    @VSTAT              Else update some values and check screen-blank
*
```

```
*****************************************************************
* AUTO-SPRITE MOTION
*  For auto-sprite motion the Sprite Motion List (SML) must always be located
*  at >780  in VDP.  The first and second bytes of each entry in the list
*  represents the Y and X motion of the corresponding sprite.  The values >00
*  to >7F are the positive directions of down or right.  Values >80 to >FF are
*  the negative directions of up or left.  The next two bytes are used
*  to scale the speed of motion of the sprites.  A motion value of one, for
*  example, will be scaled to move the sprite one pixel every 16 VDP interrupts.
*  A motion value of 16 moves the sprite one pixel every VDP interrupt, etc.
*  The Sprite Attribute List (SAL) must be located at >300 in VDP, also, in
*  order to use this auto-sprite motion routine.
*
TSTMOT SLA   R1,1                  Shift 2nd bit off
       JOC   TSTSND                If set then skip auto-sprite motion processing
       MOVB  @MOTION,R12           Grab number of sprites in motion from SRAM
       JEQ   TSTSND                If zero then go check for auto-sound processing
       SRL   R12,8                 Make number of sprites a word value
       LI    R2,VDPRD              Init. R2 to VDP read data address
       LI    R3,VDPWD              Init. R3 to VDP write data address
       LI    R8,RSMOT              Init. R8 to sprite motion table address in VDP
MLOOP  MOVB  @R8LB,*R15            Set VDP address to sprite motion table
       MOVB  R8,*R15
       CLR   R4                    Clear out R4
       MOVB  *R2,R4                Read in delta Y byte
       CLR   R6                    Clear out R6
       MOVB  *R2,R6                Read in delta X byte
       SRA   R4,4                  Shift delta Y over a digit
       MOVB  *R2,R5                Read in temp. Y scaler
       SRA   R5,4                  Shift temp. Y scaler over a digit
       A     R4,R5                 Add delta Y to temp. Y scaler
       MOVB  *R2,R7                Read in temp. X scaler
       SRA   R6,4                  Shift delta X over a digit
       SRA   R7,4                  Shift temp. X scaler over a digit
       A     R6,R7                 Add delta X to temp. X scaler
       AI    R8,-QSAML             Back address up to sprite attribute list
       MOVB  @R8LB,*R15            Set VDP address to sprite attribute list
       MOVB  R8,*R15
       CLR   R4                    Clear out R4 again
       MOVB  *R2,R4                Read in present Y location
       A     R5,R4                 Add motion to Y location
*
*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
*+++ The following segment was added to fix a sprite flicker problem as sprites
*+++ rolled off the screen (values >C0 to >E0 are invalid vertical positions).
*+++ It was removed from consoles dated 03/16/81 and reinstalled in consoles
*+++ dated 07/29/81.
*
       CI    R4,6*VDELTA+255       Is vertical position greater than >C0?
       JLE   ONSCRN                NO! So skip rest of checks
       CI    R4,7*VDELTA           Is vertical position less than >E0?
       JH    ONSCRN                NO! So skip rest of segment
       MOV   R5,R5                 Is vertical motion negative?
       JGT   $+6 >>>>>>>>>>!       NO! So don't add VDELTA*7 to position
       AI    R4,6*VDELTA    !      Bump negative value past forbidden values
       AI    R4,VDELTA <<<<!      Bump positive value past forbidden values
*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
*
ONSCRN CLR   R6                     Clear out R6 again
       MOVB  *R2,R6                Read in present X location
```

```
A     R7,R6          Add motion to X location
ORI   R8,WRVDP       Set MSBit of sprite attribute list address
MOVB  @R8LB,*R15     Set VDP address for a write to the
MOVB  R8,*R15         sprite attribute list area
MOVB  R4,*R3         Install new Y location
AI    R8,QSAML+2     Reset address to sprite motion list plus two
MOVB  R6,*R3         Install new X location
SWPB  R5             Move LSB of temp. Y scaler to MSB
MOVB  @R8LB,*R15     Set VDP address to temp. Y scaler location
MOVB  R8,*R15         in sprite motion table
SRL   R5,4           Adjust temp. Y scaler value
MOVB  R5,*R3         Update temp. Y scaler in VDP
SWPB  R7             Move LSB of temp. X scaler to MSB
SRL   R7,4           Adjust temp. X scaler value
MOVB  R7,*R3         Update temp. X scaler in VDP
AI    R8,>C002       Set to sprite motion list and strip write bit
DEC   R12            Decrement the number of sprites in motion count
JGT   MLOOP          Continue until all sprites are serviced
```

5th 1- =FORTH by Mariusz Stanczak

This month we have lots of FORTH code for you. There is the continuing saga of PSEUDO83 (see Listing, Part 2, lines 99-181), and then an implementation of one of many run-time overlay techniques. The latter set of words that comprise MODULE require some explanation, but I'll try to be brief. Instead of repeating what you can find in other places (see references), let's go over the glossary of words that make up the solution presented here with emphasis on the MODULE lexicon words. But, first, the introductory material.

Being a system extension utility, MODULE relies on the structure of the system dictionary. Here, we need to know about DP (dictionary pointer), LATEST, LFA (link field address), NFA (name field address), and some concepts employed in implementation of VOCABULARY.

The value stored in DP tells the outer interpreter, INTERPRET, the next free memory cell for a new dictionary entry. By modifying this value, we can tell INTERPRET to compile definitions at any arbitrarily chosen address. The word LATEST leaves on the PS the address of location which points to the NFA of the topmost word in the current vocabulary, LFA converts PFA (parameter field address) of a word into its LFA. LFA chains each word in the dictionary to a word below, it points to NFA of the preceding word. OK! That was all simple, and most of you knew it already. Now, the vocabularies. There aren't many sources that undertake the task of explaining the full scope of the concept, and the most thorough treatment (that I found) of vocabularies in FORTH was in FORTH Encyclopedia. In general terms, the word VOCABULARY is FORTH's way of implementing a multi-dimensional dictionary structure. In FIG-FORTH, one vocabulary (parent) may contain another (its child), which in turn may contain its own children vocabularies. The word VOCABULARY is a defining word, and new vocabulary roots are created by typing:

VOCABULARY Vname

which creates a dictionary entry, Vname, having the following structure:

```
+-------------------------------+
| LFA | NFA | CFA | VLF | PNF | CLF |
+-------------------------------+
```

LFA and NFA are the same as for any other definition

- CFA points to the run-time portion of VOCABULARY (the part that follows DOES> in the definition of VOCABULARY)

- VLF (vocabulary link field) points to the last definition added to the root Vname. VLF is to this location to which LATEST is the pointer.

- PNF (pseudo name field) contains >81AØ, an impossible (a blank, so it can never be found by -FIND) but legal name. To this location points, initially, VLF, but as definitions are added to vocabularies, the LFA of the first word will point to PNF, and VLF will always point to the word added last.

- CLF (chronological link field) points to CLF of parent vocabulary. This location of the most recently created Vname root is referenced by VOC-LINK.

By typing Vname, the run-time part of VOCABULARY is executed, which changes the few system pointers as described above and makes Vname the context vocabulary by also stuffing the address of VLF into CONTEXT. That means that only the words in Vname vocabulary, and the ones contained in vocabularies to which Vname was chained, will be searched by -FIND. This in effect limits/changes the scope of words known to the system. The word DEFINITIONS makes the context vocabulary current and all newly defined words will be appended to that vocabulary's chain of words.

If the above sounds complicated, don't worry. None of it is really needed to use MODULE in your application, but if you do figure out the trick, it will be easier for you to understand the do's and don't's imposed on MODULE's usage.

With vocabularies, MODULE is mainly concerned with links between them, and to assure that this is the only thing that should be of concern, MODULE makes the last word of each overlay a part of FORTH vocabulary. This bypasses the complicated process of properly relinking words located within an overlay that might belong to some other vocabulary chains, so restriction #1, and the only one there is, is:

- overlay may only contain vocabularies that are fully enclosed by its physical boundaries.

*** Note - as long as a given overlay is in memory, you are free to select any of the vocabularies it may contain and add more definitions to it, but those definitions should be considered temporary, if you're in interactive mode, and be deleted from dictionary before the overlay they are connected to is deleted. There is one big unless, and that's: when the newly defined words are a part of your application, in which case you control when the words are executed, and of course if the application is sealed so your exclusive control is assured. End-of-note ***

As you rightly suspect, all this fuss has to do with linking of each word to its predecessor through LFA and the word's association to its vocabulary through VLF, as well as with chronological linking of vocabularies through CLF. Your system is definitively in trouble if either points to something that does not exist. The same, of course, goes for system variables CURRENT and CONTEXT, but it is much less likely to happen.

Glossary:

TH - a tip word from Thinking FORTH. TH indexes the nth item in a word array. This is taken to the extreme through redefinitions of TH to ST, RD

and ND.

ADR - a + redefined for linguistic clarity.

MODULE_BUFFER - a pointer to overlay area.

>FORTH - makes FORTH vocabulary current and context.

DP>MODULE_BUFFER - aims DP past the parameter array located at the beginning of each overlay area.

POINTER@ and POINTER! - words that fetch and store values from the parameter array.

HI>LO - aims DP into overlay area, preserving nfa of last word in FORTH vocabulary residing in high memory along with vocabulary link pointer.

MODULE_LOAD - brings the saved code into overlay area from disk and restores vocabulary and dictionary links. It is, basically, a simplified BLOAD, but it does not have any error checking built in so watch out; the correct floppy had best be in the correct drive!!! *** A NOTE - for any serious application it is highly recommended to include extensive error checking, especially in view of the many ways one can refer to the location of a screen on a disk. - END OF NOTE ***

MODULE_LINK: - a defining word used to create an overlay loading word.

LO>HI - preserves the new DP in low memory and vocabulary link pointer. Restores the old dictionary pointer value into DP and builds the overlay loading word.

MODULE - the main word of the overlay utility. At the same time, it is the one of the two words that constitute this utility's lexicon (the other one is SAVE_MODULE). MODULE automates the process of creating an overlay; it needs on PS a list of screen numbers, which are to be included in an overlay, and it has to be followed by an inline word. This word will be the overlay's name, and by this name the overlay will be called into memory from disk. *** A TIP - before using MODULE type SP! to clear PS, and then put on PS the screen numbers.

DICTIONARY_RELINK - patches lfa of module_name to point to old_latest.

VOCABULARY_RELINK - restores the content of VOC-LINK variable to its value prior to creation of an overlay.

SAVE_MODULE - this word wraps up the process of creation of an overlay. It preserves the last few bits of information about the state of environment, writes the overlay to disk, and forgets it. From this point on you will use module_name to bring the needed code back to memory for further use.

The overlay techniques had been used extensively on early computers, which lacked large internal memory space, for program segmentation. Even today, in the era of mega everything (not the case with our 99/4A's), as the complexity of software rises along with its hunger for memory, program segmentation still gets a fair share of use (just take the UNIX and VMS, or any other virtual memory operating system). The MODULE utility was written mainly as an exercise in that idea; although, I found one use for it, that by far exceeded the usefulness of the original purpose in the interactive environment of Forth. As a matter of fact, I have built my whole system around this side benefit. I've

segmented all of FORTH outside of the resident portion into logical chunks of code, and swap a given utility set in and out as needed without disturbing the high memory portion of the dictionary with my efforts "in progress". Very useful during development, when things are far from being concrete and need a lot of patching or redoing, which I can do without wiping out any of the loaded "helpers" that are located out of the way, in low memory. As for its intended purpose, most of program code can be segmented, leaving the high memory space for data structures and for global to each segment entities. Have fun, and good words to you.

References:

FORTH Encyclopedia, Mitch Derick and Linda Baker, MVP, Inc., 1982.

FORTH Dimensions, Volume V, number 3, page 5, "FIG-FORTH Vocabulary Structure", by Evan Rosen

*** Note for TI-FORTH users - prior to using this utility on your system, the system has to be reconfigured to make room for overlays. There are total of five buffers in the system. The minimum number of buffers on the system, without disturbing its proper functioning, is two, so up to three buffers can be taken away from the system for the overlay area; unless, you are keeping messages memory resident, in which case only two buffers can be had (message takes one). The following reconfiguration words have to be executed only once upon booting the system up, so, preferably, they should be located on screen number 3.

```
: BUF-   ( n --- adr )
   EMPTY-BUFFERS   B/BUF 4 +   *   LIMIT$ @   SWAP -   DUP LIMIT$ ! ;

3 BUF- CONSTANT MODULE_BUF
```

End of note ***

*** Note for Wycove-FORTH users - There is about 2.5K bytes free starting at >2000 so if you reconfigure your system down to two buffers (using CONFIG word), you get a healthy amount of space for MODULE_BUF. Define this constant to have the value of >2000. - End of note ***

*** Note for both systems - Make sure that the overlay code you are creating does not exceed the size of the buffer made for it by loading the wanted screens to high memory first, and checking the value left on PS by HERE before and after the compilation. - End of note ***

Listing, Part 1

```
 1. : DEPTH ( --- n )
 2.      S0 @ SP@ - 2- 2 / ;
 3. : TH   ( n --- n )
 4.      2 * 2- ;
 5. : ST    TH ;    : RD    TH ;    : ND    T
 6. : ADR    + ;
 7. : POINTER@    ( n --- n )
 8.      MODULE_BUF + @ ;
 9. : POINTER!    ( n --- n )
10.      MODULE_BUF + ! ;
11. : DP>MODULE_BUF    ( --- )
```

```
    MODULE_BUF 10 TH ADR  DP ! ;
: >FORTH
    [COMPILE] FORTH DEFINITIONS ;
: HI>LO   ( --- <DP> )
    HERE  >FORTH
    LATEST 1 ST POINTER!
    VOC-LINK @ 5 TH POINTER!
    DP>MODULE_BUF
: MODULE_LOAD   ( scr# --- )
    MODULE_BUF
    BEGIN
      >R  DUP 1+ SWAP BLOCK
      DUP 4 TH ADR @  R - R>  DUP B/ >R
      SWAP B/BUF  MIN CMOVE
      R
      4 TH POINTER@  R>  <
    UNTIL  DROP DROP
    3 RD POINTER@  2 ND POINTER@  !
    7 TH POINTER@ -DUP
    IF
      6 TH POINTER@  SWAP  !
    THEN ;
: MODULE_LINK:   ( --- ) \ followed line name
    >FORTH
    [COMPILE] :  \ in Wycove's  [COM (:)
    COMPILE OFFSET  COMPILE DUP  COM@
    COMPILE LIT  HERE 9 TH POINTER!
    COMPILE 3  COMPILE PICK  COMPILE
    COMPILE LIT  HERE 8 TH POINTER!
    COMPILE MODULE_LOAD  COMPILE SWAMPILE !
    [COMPILE] ;  \ in Wycove's  [COM (;)
;
: LO>HI   ( <DP> --- )
    HERE 4 TH POINTER!  VOC-LINK @ 6OINTER!  DP !
    MODULE_LINK:  LATEST PFA CFA DUPRD POINTER!  2 ND POINTER! ;
: MODULE   ( any_#_of_screen_#s --- followed by inline name
    HI>LO  DEPTH 1
    DO  SWAP LOAD
    LOOP  LO>HI ;
: DICTIONARY_RELINK ( --- )
    1 ST POINTER@  2 ND POINTER! ;
: VOCABULARY_RELINK ( --- )
    5 TH POINTER@  6 TH POINTER@ OVER -
    IF
      VOC-LINK
      BEGIN  DUP @ 3 PICK OVER -
      WHILE  SWAP DROP
      REPEAT
      DROP SWAP DROP DUP 7 TH POINTE
    ELSE
      DROP DROP  0 7 TH POINTER!
    THEN ;
: MODULE_SAVE   ( scr# --- scr# )
    DUP  8 TH POINTER@ !  OFFSET @  POINTER@ !
    DICTIONARY_RELINK VOCABULARY_REL
    MODULE_BUF
    BEGIN
      >R  DUP 1+ SWAP  OFFSET @  +  R UPDATE
      DUP B/BUF ERASE
      4 TH POINTER@ R -  R>  DUP B/B >R
```

```
 72.        SWAP >R SWAP R>  B/BUF MIN CMO
 73.        R
 74.        4 TH POINTER@  R> <
 75.      UNTIL  DROP  FLUSH ; ;S
 76.
 77. Environment info parameter array.  <ans the content of
 78. 1 ST - old_latest's nfa (in FORTH voary)
 79. 2 ND - module_name's lfa
 80. 3 RD - module_name's <lfa>
 81. 4 TH - low memory (DP)
 82. 5 TH - old (VOC-LINK)
 83. 6 TH - new (VOC-LINK)
 84. 7 TH - adr of new (VOC-LINK)
 85. 8 TH - adr in parameter field of modame.
 86. 9 TH - adr in parameter field of modame
```

Listing, Part2, PSEUDO83

```
 99. ( comparators, not, sign, etc.)                    \ ver 2.1  gtjul86
100. \ comparators to leave tf=-1, this screen not copyrighted
101. : =   = MINUS ;      : 0=  0= MINUS ;      : >  > MINUS ;
102. : <  < MINUS ;       : U<  U< MINUS ;      : 0>  0 > ;
103. : 0<  0 < ;       : <>  = 0= ;       : =>  1+ < ;       : =<  1+ > ;
104. : NOT83  ( n --- ~1 )  \ 1's complement
105.     -1 XOR ;
106. : NIP  ( n1 n2 --- n2 )
107.     SWAP DROP ;
108. : TUCK  ( n1 n2 --- n2 n1 n2 )
109.     SWAP OVER ;
110. : ON  -1 SWAP ! ;
111. : OFF  0 SWAP ! ;
112. : INCR  1 SWAP +! ;      : INCR2  2 SWAP +! ;
113. : DECR  -1 SWAP +! ;      : DECR2  -2 SWAP +! ;
114. : ?ENOUGH  ( n --- ) \ aborts if PS too shallow  mapOct83
115.     DEPTH 1- > IF  ." Not enough parameters" ABORT  THEN ;
116. : SIGN83  3 ?ENOUGH  ROT SIGN ;
117. ( span, expect )
118.   VARAIBLE83 SPAN
119. : (SPAN)  ( cnt --- )
120.     SPAN ! ;
121. : LENGTH  ( adr --- cnt )
122.     82 0 DO  DUP  \ dup buf adr
123.           I + C@  \ get succeeding chars
124.           0=  IF  I LEAVE  THEN  \ if null leave count
125.         LOOP  NIP ;
126. : EXPECT83  ( text_buf_adr n --- cnt )
127.     2 ?ENOUGH  ?DUP  \ text buf needs n+2 bytes for delimiter
128.     IF  ( n>0 )
129.       OVER >R  \ save adr
130.       EXPECT  R> LENGTH (SPAN)
131.     ELSE
132.       DROP ( adr )  SPAN OFF
133.     THEN ;
134. ( defer, is ) \ lines 135-    not copyrighted
135. \ modified from Will Baden, FD V.6/5, also see Laxen&Perry's F83
136. \ model and Mastering Forth, Anderson, et al, 1984
137. : DEFER  ( --- )
138.     <BUILDS  2 ALLOT  DOES>  @ EXECUTE ;
139. : (IS)  ( cfa --- )  R @  >BODY !  R> 2+ >R ;
140. : IS  ( cfa --- )   STATE @
```

```
    IF  COMPILE (IS)
    ELSE  '83 >BODY !
    THEN ; IMMEDIATE
( query, doer/make )
  VARIABLE83 #TIB
: QUERY83
    TIB @ 80  2DUP BLANKS  >IN OFF
    EXPECT83
    IN OFF  BLK OFF
    SPAN @  #TIB ! ;
\ DOER/MAKE L. Brodie, Thinking Forth, 1984, p279
\ modified, gt
: DOER
    <BUILDS  ['] NOP >BODY , DOES>  @ >R ;
  VARIABLE83 MARKER
: (MAKE)  R>  DUP 2+ DUP 2+ SWAP @ >BODY !  @ ?DUP IF  >R  THEN ;
( query, doer/make )
: MAKE  STATE @
    IF  COMPILE (MAKE)  HERE MAKER !  0 ,
    ELSE  1  17 ?ERROR ABORT
    THEN ; IMMEDIATE
: ;AND  [COMPILE] ;S  HERE MAKER @ ! ; IMMEDIATE
: UNDO  ['] NOP >BODY  [COMPILE] '83 >BODY ! ;
( name, save-buffers )
  '83 ;S CFA CONSTANT DEF-END
: NAME  ( pfa --- ) \ displays the name of a word from any adr in
\ its pfa field.  a handy addition to debugging tools, as it allows
\ you to id the numbers on the return stack.
  CR
  BEGIN  2- DUP @ DEF-END =
  UNTIL  4+ ID. ;
  BASE->R  HEX
: SAVE-BUFFERS  ( --- ) \ flush updated buffers to disk but do not
\ deallocate the buffers.  an 83 version
  LIMIT$ @  FIRST$ @
  DO  I @ 0<
    IF  I @ 8000 - DUP 2+ SWAP  0 R/W  I !
    THEN  404
  +LOOP ;
  R->BASE
```

NEWS AND NOTES

Chicago Faire

The Chicago TI-Faire, held November 1, was again a big success. About 1,000 99'ers braved the rain for an outstanding show. Users from Canada and at least 20 states of the U.S. were in attendance. As 30 vendors participated, a lady, sporting a smile, was heard to observe, "Until now, I'd only dreamed of a day like this. It's unbelievable, the variety of products available!". Indeed, products ranged from the not-yet-released to those not produced in years (items such as TI stand-alone products were not uncommon!).

A bright new star of the 99/4A world emerged in Chicago, J. Peter Hoddie. Hoddie had already received considerable acclaim in 1986 and his talents are quite well-known in the Boston area, where he resides, but the Chicago show is

he gained the respect of an international audience. Hoddie offered two recent software efforts, Gram Packer (Genial Computerware) and Font Writer (Asgard). Hoddie also gave a very entertaining musical presentation, which included a selection with a 99/4A playing music while controlling music through another 99/4A, with the two linked via the cassette ports, and a number featuring Hoddie on the cello accompanied by the 99/4A on "piano". Hoddie also announced his participation in a joint venture, Genial Computerware, with Corson Wyman, another talented Boston-area 99'er, and Barry Traver, Genial's charter owner. Anticipated projects from Genial include both software and hardware, in addition to continuation of the firm's *Genial TRAVelER* diskazine.

Many outstanding products and lectures vied for attention in Chicago. Among the most notable were a 99/4A compatible computer prototype (now known as the MYARC 964Ø) from MYARC, an instantly popular program called Printer's Apprentice from McCann Software, products from both Europe and North America available from Ryte Data, Joypaint 99 from Great Lakes Software, and a PC-type keyboard interface from Rave 99. Clint Pulley provided a very informative talk on c99.

Addresses for the above vendors are:

Genial Computerware, P.O. Box 183, Grafton, MA Ø1519, *Genial TRAVelER* address: 835 Green Valley Drive, Philadelphia, PA 19128.

Asgard, POB 1Ø3Ø6, Rockville, MD 2Ø85Ø

MYARC, Inc., P.O. Box 14Ø, Basking Ridge, NJ Ø792Ø

Clint Pulley, 38 Townsend Avenue, Burlington, Ontario, Canada L7T 1Y6

McCann Software, P.O. Box 3416Ø, Omaha, NE 6Ø134

Ryte Data, 21Ø Mountain Street, Haliburton, Ontario, Canada KØM 1SØ

Great Lakes Software, P.O. Box 241, Howell, MI 48843

Rave 99, 23 Florence Rd., Bloomfield, CT Ø6ØØ2

Editor's Note

Well, there's plenty in store for you in upcoming months!

Next month, we'll have a really important announcement! We'll be releasing a software package that many of you have been asking for!

Yes, things have slowed to a snail's pace with our recent peak renewal period and a few unexpected matters. Things are looking much better now, though. We've got some great material lined up for upcoming issues. And, we'll have another memory map for you next month -- it was a bit long for this issue.

RM

Gram Packer

Notes from Richard Mitchell

Wow! Gram Packer (Genial Computerware, address above) is a fantastic package! Gram Packer works with Gram Kracker™ or Maxi-Mem or Gram Karte to allow you to create your own custom menu of programs and modules that you use most often. The files load into GRAM space. Programs must be Memory Image Programs (the ones that load from E/A Option 5, TI-Writer Option 3, etc.). Using our June Gram Kracker™ tip for 16 items on a menu, you can really create some very elaborate menu schemes and also chain from one menu to the next, so you can reduce the number of filenames to memorize to only one! Though I've only just begun working with the package, I've already created a menu that includes Turbo, Multiplan™, Mini Memory, Disk Manager II, TE II (12ØØ bps), Navarone DBM, DM-1ØØØ 3.5, Fast-Term 1.16rj8, Editor/Assembler (Regular), EA/XB (created with MG's GK Utilities I), TI-Writer GRAM Disk and Next Menu! Gram Packer allows choosing

from as many files as you can access from your system's disks, including hard disks, RAM disks and floppies! Priced at only $10, Gram Packer is certainly a program that should be in the library of every GRAM device owner! We'll bring you more info on Gram Packer in the future.

FILE OVERLAYS

by Richard M. Mitchell

Files are normally useful in the format in which originally created. However, occasionally there are files which cannot be quickly and easily modified. For instance, to depict boxes with questionnaire responses, a primary file with the boxes could be created, followed by directing the responses to an overlay file, then the overlay file could be overlaid onto the primary file, as shown below.

PRIMARY FILE:

```
 -   -   -   -
| | | | | | | |
 -   -   -   -
```

OVERLAY FILE:

```
 A   B   C   D
```

RESPONSES OVERLAID ONTO PRIMARY FILE:

```
 -   -   -   -
|A| |B| |C| |D|
 -   -   -   -
```

The program listed below will replace a character in the primary file with any non-space (space is ASCII 32) character in the overlay file. Of course, the output device name can be a disk file name, as well as a printer name, etc. Because there might be many possible variations on this theme, the program is one I wrote rather quickly as an example. The idea came from a request for a method of printing a teacher's gradebook and attendance sheet, output from Multiplan™ into D/V 80 files, into a single output. If you find other uses for the program, please drop me a note in the mail.

```
100 DISPLAY AT(3,1)ERASE ALL
:"D/V 80 Output Overlays":"T
he Smart Programmer":"1986"
110 DISPLAY AT(8,1):"Primary
 Filename:":"DSK1.":"":"Over
lay Filename:":"DSK1.":"":"O
utput Device Name:":"PIO"
120 ACCEPT AT(9,1)BEEP SIZE(
-15):A$
130 ACCEPT AT(12,1)BEEP SIZE
(-15):B$
140 ACCEPT AT(15,1)BEEP SIZE
(-28):P$
150 OPEN #1:A$
160 OPEN #2:B$
170 OPEN #3:P$
180 LINPUT #1:S$
190 IF EOF(2)=0 THEN LINPUT
#2:T$
200 L=MIN(80,MIN(LEN(S$),LEN
(T$)))
210 FOR I=1 TO L
220 IF SEG$(T$,I,1)=" " THEN
 240
230 IF I>1 THEN S$=SEG$(S$,1
,I-1)&SEG$(T$,I,1)&SEG$(S$,I
+1,LEN(S$)-I)ELSE S$=SEG$(T$
,I,1)&SEG$(S$,I+1,LEN(S$)-I)
240 NEXT I
250 IF LEN(T$)>LEN(S$)THEN S
$=S$&SEG$(T$,LEN(S$)+1,LEN(T
$)-LEN(S$))
260 PRINT #3:S$ :: PRINT S$
270 IF EOF(1)=0 THEN 180
280 IF EOF(2)=0 THEN S$="" :
: GOTO 190
290 CLOSE #1 :: CLOSE #2 ::
CLOSE #3 :: END
```

Q & A

In the September, 1986, issue, you referred to using DISkASSEMBLER™ to determine all addresses in a source file. Why not simply AORG the file?

Using the AORG directive to load the object code at a specific address is an alternative method. However, when more than one file is to be loaded or various combinations of files are to be tested, the AORG method can become cumbersome. Additionally, since AORG

code does not update program memory allocations (such as XB's FFALM, the First Free Address in Low Memory), non-AORG files may be the final preference and one may find having both AORG and non-AORG files on disk confusing. When an option is available for accomplishing a task, we try to present the method that covers the widest range of possible cases.

Could you supply the addresses of public domain software authors?

No. In many instances, software is released into the public domain because the author has no interest in providing support functions, such as reading and answering mail. Most public domain software is available through telecommunications services and user groups. One of the many groups that makes available to its members many of the latest public domain releases for a nominal library fee is LA 99'ers Computer Group, P.O. Box 3547, Gardena, CA 90247-7247. LA 99'ers also offers many other services to its members, including a monthly newsletter, within which new library acquisitions are listed.

Is there a quick way to update a Multiplan™ cell while Recalc is off?

Yes. Because updating an entire spreadsheet is a very slow process, it is generally best, as we've covered before, to select Options Recalc (No) so that the sheet will not update all of its formulae with each input. However, it would often be convenient to update a particular cell. To do so, simply select Edit and press enter! Of course, you must proceed with caution, as the cell may be dependent upon other cells that might not be updated by this procedure (in such a case, Recalc (Yes) must be selected).

The mention of a reward in your article on piracy insulted my good intentions. Why did you offer the article?

Certainly there are no easy answers to the piracy issue. The article points up the need for the private sector to aggressively monitor itself, before government intervention results in excessive laws governing telecommunications. Currently, there is great latitude in establishing a telecommunications service, such as a BBS, and in disseminating information through such services, affording computer users an avenue for grassroots support for both computer-related and other issues. Blatant disregard of copyright laws by a few individuals is one of many factors endandering the liberties enjoyed by many.

Can you offer advice on printers?

We receive many requests for specific printer information. Though we'd really like to help, it would be impossible for us to try every possible combination of printers, parallel interfaces, serial interfaces, print buffers, print spoolers, software, etc. Essentially, printer support is the function of a dealer. And, do remember that while a good dealer will charge more than a discount mail-order firm, the service and support available from the good dealer is often the real long-term bargain! As a basic guideline for purchasing a printer, because the TI Impact Printer was the early standard for the 99/4A and it is a 9-pin dot-matrix printer that uses Epson print codes, most 99'ers have purchased printers compatible with it. For typewriter quality print, a daisy-wheel printer is required, but such printers have practically no graphics capabilities. We use an NEC P6 to produce *The Smart Programmer* and it is a 24-pin printer that handles most of the Epson print codes. The P6 is an excellent printer within its price range, often discounted at under $500, plus about $25 to $75 for uni-directional or bi-directional tractor. Printers adequate for most home users begin at about $200. If you have already purchased a printer and your dealer cannot or will not assist you, then you need to locate a user with a configuration similar to your own, with the best avenues being your

local user group and/or the major telecommunications networks.

What should I do to test a program that I want to market?

The program should be tested in-house first, of course. Next, the program needs to be Beta tested, tested by individuals who have never before seen the package. Beta testers should receive proposed final versions of both the software package and the package's documentation. Beta testers should be individuals with unquestionable morals, of course. Also, at least some of the Beta testers should assume the role of a typical customer and not have easy phone access to the author. Beta testing should be done by "Joe Bright" and "Joe Dim". Most folks who go the Beta test route seek Joe Bright; he's the guy with an incredible IQ, he's completely computer literate and an English grammar whiz -- he can tell nearly everything a package does that it shouldn't and nearly everything it doesn't that it should. Unfortunately, few firms employ the services of Joe Dim, who is somewhat of a dim-wit when it comes to computers. If Joe Dim can run the package properly, though, anyone can! If the program is updated, subsequent rounds of Beta testing would be advisable, with at least some new testers selected in each round. If the program is based on a useful idea and Joe Bright and Joe Dim approve the package, then with a good marketing strategy the program should become a sales success. It usually takes a few trials before a firm recognizes the benefits of Beta testing and learns the best Beta test procedures, but most eventually get around to it (or fail)!

SHOW SCHEDULE

Fest-West '87, May 1 and 2, 1987, contact LA 99'ers, P.O. Box 3547, Gardena, CA 90247-7247.

Texas Instruments Computer Owners' Fun Fest (TICOFF '87), March 28, 1987, contact TICOFF '87, c/o Bob Guellnitz, Roselle Park High School, 185 W. Webster Ave., Roselle Park, NJ 07204.

TI-Writer Tips

by Richard M. Mitchell

In using TI-Writer, there are several simple but often overlooked ways to save keystrokes. And, of course, saving keystrokes means saving time!

Except as noted below, all of the commands of TI-Writer can be accessed directly. For instance, to use LF (Load File), it is not necessary to use the F (File) command first.

L (Lines), F (Files) and SH (SearcH) serve no function and simply provide a list of Command options.

Q (Quit) also serves no direct function, but offers access to three options, S (Save file), P (Purge) and E (Exit). Q is rather peculiar in that the only unique acronym following use of Q is P (Purge). S (Save file) is otherwise used as S (Show Line) and E (Exit) is otherwise used as E (Edit). Use of unique acronyms would have been less confusing, but that's not the way it works.

A tip that is more easily overlooked is that using LF (Load File) without line parameters automatically purges the current document. So, P (Purge) is generally not necessary before using LF.

Special Offer

We have located a limited number of new TI-Writer packages. The price is $20.00, C.O.D., U.S. only. The price includes all fees, etc. To obtain one, send a self-addressed stamped postcard (so you can be notified if quantities aren't adequate to fill your order). Due to the limited supply, do not send any money!

BYTEMASTER ORDER FORM

NAME_____

ADDRESS_____

CITY_____

STATE_____

ZIP CODE_____

COUNTRY_____

The Smart Programmer

SP1	$18.00	U.S. AND CANADA FIRST CLASS
SP2	$15.00	U.S. THIRD CLASS (no back issues)
SP3	$20.00	FOREIGN SURFACE (no back issues)
SP4	$32.00	FOREIGN AIRMAIL
SP5A-D	$ 1.75	U.S. JUNE - SEPT. 1986, ea.
SP6A-D	$ 2.75	FOREIGN JUNE - SEPT 1986, ea.
SP7A-G	$ 2.50	U.S. JAN 84-AUG 84,photostats,ea.
SP8A-G	$ 3.50	FOREIGN JAN 84 - AUG 84, ea.

Super 99 Monthly

SM1	$18.00	Complete set of 18 back issues
SM2A-R	$ 1.00	Back issues - ea. (U.S. Third Class)
SM3A-R	$ 1.50	Back issues - ea. (Canada and U.S. First Class)
SM6A-R	$ 2.50	Back issues - ea. (Foreign Air Mail)
SM4	$12.00	Programs on disk (non-FORTH)
SM5	$15.00	Super 99 Handicapper (req. XB, 32K, Disk, Printer)

Payments accepted by check or money order in U.S. funds, coded for processing through the U.S. Federal Reserve Banking System. No billings or credit sales. Dealer inquiries invited. Discounts available on quantity orders.

ITEM #	QTY	EACH	AMOUNT	New	Renewal
				+-+	+-+
				| |	| |
				+-+	+-+

Louisiana residents must add 4% sales tax. Calcasieu 1%. Sulphur 2%

The Smart Programmer is published monthly by Bytemaster Computer Services, 171 Mustang Street, Sulphur, LA 70663.

Bytemaster Computer Services
171 Mustang Street
Sulphur, LA 70663-6724
U.S.A.

FIRST CLASS MAIL

Bu
U.
Su 70663
Pe 141

POSTMASTER: ADDRESS CORRECTION REQUESTED
RUSH -- TIME DATED MATERIAL

FIRST CLASS MAIL