Wycove Forth

for the Texas Instruments
99/4A Home Computer*

Produced by Wycove Systems Limited
P. O. Box 499,
Dartmouth, Nova Scotia
B2Y 3Y8 Canada

Table of Contents

Introduction

Wycove Systems Limited is proud to bring the Forth language to owners of the Texas Instruments 99/4A home computer. We have attempted to unravel the mysteries of the 99/4A to the extent of being able to provide you with a sophisticated programming language at a low cost. We ask your cooperation in helping us recoup the money we spent on developing this product. Please do not give away or sell copies of this program or manual. Aside from the fact that it is an infringement of the law, it also hurts you as well as us. Unless we are able to expect a reasonable profit from developing software for you, we will be unable to enhance what you have now or to bring new products on the market. We have included order forms for your friends who wish to start real computing with Forth.

We cannot at this time offer a complete tutorial on the Forth language. We recommend that beginners start by reading the book Starting Forth by Lee Brodie. The language is easy to learn and yet can be extended in so many bewildering ways that it is hard to conceive of a complete description of the language. When learning Forth you will find the S. word to be invaluable to print out the contents of the stack at any time.

Wycove Systems Limited retains the right to charge a licence fee for products developed which include Forth as an integral part. We intend to be reasonable about the required fee, which will vary depending upon the access to Forth allowed of the purchaser.

Running Wycove Forth

Wycove Forth is available for running through three host modules: Editor/Assembler, Mini Memory and Extended Basic. The capabilities of all three versions are similar. The only difference is the size of the programs. Please refer to the section on Memory Usage for details. The cassette versions are also just as powerful as the diskette systems, although somewhat less convenient to run.

Editor/Assembler Disk Version

This version of Wycove Forth is run through the fifth option on the Editor/Assembler menu, Run Program File. Insert the Editor/Assembler module. Your system must include the Memory Expansion and a disk controller. Select the Editor/Assembler program, then select option 5. You will be prompted for a file name. Type in DSK1.FORTH (or DSK2 if that's where your file is). Forth will load and start up automatically.

Editor/Assembler Cassette Version

This version of Wycove Forth is run through the fifth option on the Editor/Assembler menu, Run Program File. Insert the Editor/Assembler module. Your system must include the Memory Expansion. Select the Editor/Assembler program, then select option 5. You will be prompted for a file name. Type in CS1.A and Forth will load. There are at least three files to be loaded. Do not rewind the tape between files as instructed. Simply keep pressing the ENTER key until Forth starts up automatically.

Mini Memory Disk Version

This version of Forth is normally loaded through the use of the assembler routine supplied with your copy of Mini Memory Forth. On the diskette version, you can load the routine by using option one (Load and Run) of the Mini Memory menu. You may have to re-initialise the Mini Memory module, using option three (Re-Initialize) of the Mini Memory menu. The name of the routine on the disk is DSK1.FORTHLOAD . The Mini Memory will load and automatically run this program, which will result in the Forth system being completely loaded and started.

Mini Memory Cassette Version

This version of Forth is normally loaded through the Easy Bug option of the Mini Memory. Prepare the tape and enter the Mini Memory Easy Bug option. To exit from the Easy Bug menu type a character. Then type in the L command to load from the tape. There will be at least two files to be loaded (the distributed version has two files). After the first file is loaded type the L command again, remembering not to rewind the tape between files. When the last file is loaded use the Easy Bug execute command in the form EA000 (that's E,A,zero,zero,zero) to start up Forth.

Extended Basic Disk Version

To start the Extended Basic disk version of Wycove Forth it is only necessary to have the disk in drive number one before selecting Extended Basic from the main program selection menu. Extended Basic will automatically execute the LOAD program on the disk, which in turn reads in the FORTHLOAD assembly language program, which in turn loads and starts Forth.

Extended Basic Cassette Version

To start the Extended Basic cassette version of Forth you must load and run the Extended Basic program which is included first on your cassette. This program will then read the Forth system from cassette and automatically start Forth. There are at least three files to be read from the tape, so do not rewind the cassette tape between files as instructed.

Saving Forth

After you have created several new words you may wish to have those words available immediately when Forth loads. For instance, suppose you modify the screen editor supplied and decide to make it a resident part of the system, available immediately whenever you load Forth. This can be accomplished through use of the SAVE-SYSTEM word. It will write out a new copy of the language to the currently selected device. Thus, to create a cassette copy of Forth type CS1 SAVE-SYSTEM. To create a new load copy on disk use DSK1 SAVE-SYSTEM (or DSK2 or DSK3). Note that you cannot create an Editor/Assembler copy from a Mini Memory or Extended Basic copy and vice-versa.

The SAVE-SYSTEM word is very powerful. It allows you to customize your system to suit your needs. Should you wish to protect the extra words that you have added to the basic Wycove Forth, you may wish to change the value stored in the FENCE cell, making it impossible to FORGET the new words.

New copies of the diskette versions of Forth should be written to an initialized diskette. The screens that you wish to use with the new system should be copied with the Disk Manager module. For the Mini Memory and Extended Basic versions there are additional files that should also be copied. The Forth program itself is written to files FORTH, FORTI, FORTJ etc. with the last letter of each file increasing by one.

Editor/Assembler Version

The cassette version will require at least three files to be written to cassette, so ignore the extra prompts to rewind your tape (except for verification if desired). The diskette version also uses at least three files. Should you wish to rename the files on the disk, make sure that the second and third files are renamed so that all characters except the last are the same and the last character increases by one for each file.

If you originally purchased the cassette Editor/Assembler version of Wycove Forth and you wish to convert to the disk system you can do so simply by using the SAVE-SYSTEM word to save a copy of Forth on a diskette.

Mini Memory Version

The Mini Memory Version can be saved to disk or cassette. The new system can be loaded as detailed in the previous section. Note that an assembler routine is required to be in the Mini Memory for the diskette version. This is because the Mini Memory has no provision for loading program files from disk. To save the system the proper device must be selected. For a cassette copy type in CS1 SAVE-SYSTEM and you will be instructed to prepare the cassette for two or more files. Do not rewind the tape between files (except for verification if desired). To save a copy of the system to disk type in DSK1 SAVE-SYSTEM creating at least two files on the disk.

If you ordered the cassette system and you have acquired disk drives, you can type in the FORTHLOAD program following to load Forth through the Mini Memory module. Note that it must be assembled by the Editor/Assembler module, or translated for use with the line-by-line assembler provided with the Mini Memory module.

Extended Basic Version

The Extended Basic version of Forth will require at least three files to be saved. On cassette you should first leave room for the program that copies Forth into memory. This program will have to be copied by using the OLD CS1 and SAVE CS1 statements in Extended Basic. Note that the program should not be edited. It contains embedded assembly language code and editing may destroy the program. While saving Forth on cassette do not rewind the tape between files (except to verify if desired). The Extended Basic disk version writes three or more files to the disk but cannot be run until the LOAD and FORTHLOAD files are copied from the original diskette to the new diskette

If you originally purchased the cassette version of Forth and have acquired disk drives you can switch to a disk version by using the SAVE-SYSTEM word to write a new system to the disk, typing in the LOAD program listed below and typing in and assembling the Extended Basic FORTHLOAD program also listed below. You will need the assistance of someone who has the Editor/Assembler module to assemble the FORTHLOAD routine.

Stopping Wycove Forth

Forth can be exited either through pressing the QUIT key or by executing the words 32 GPLLNK , which returns to the main title screen. If the system is in bit-map mode for screen display it may be necessary to turn the computer off for a few seconds to allow the display to reset. The function of the QUIT key can be disabled as detailed in the section on the Interface to 99/4A Hardware.

The method of storing screen buffers in Forth requires that to ensure that all newly updated screens are written out to the proper device the word SAVE-BUFFERS must be called before quitting. If SAVE-BUFFERS is not called, some screens may not be properly updated.

Typing Conventions

Wycove Forth operates in two modes for typing of Forth text. Originally it is in interactive mode. Whenever the screen editor is invoked it changes to edit mode. Some of the keys have slightly different actions in each case.

In the normal interactive mode there are special actions defined for only four keys. The LEFT ARROW key (FCTN S) performs as a backspace key, destroying the previous character in the input line. The RIGHT ARROW key (FCTN D) acts as if it were a space key, moving to the right in the line. The CLEAR key (FCTN 4) can be used to delete the input line, allowing a new one to be started. Often the CLEAR key is used to exit other commands, such as HELP, LIST and INDEX. The final special key is the ENTER key, which tells Forth to process the words entered.

The input for each Forth line can be up to 80 characters. This allows at least two screen lines to be typed before an ENTER is needed. When defining new words using the : word, a definition can be continued over several lines before the ; is reached. Each individual input line must be at most 80 characters.

All pre-defined Forth words are in capital letters. You can use both upper case and lower case letters to name your new words. Forth will treat upper and lower case letters as completely different. Special characters can also be embedded in word names and definitions. Care must be exercised in using these special characters.

The Screen Editor

Supplied with your diskette if you have a disk system is the screen editor. If you have a cassette system you will have to type it in from the listing in the sample screens section. You should decide what items to include in a complete Forth system, then load them and use the SAVE-SYSTEM word to create a new master copy.

To start editing it is necessary to have the editor screens loaded. To do this type in the two words 30 LOAD and Forth will load the editor. Normally this would be done immediately upon starting up Forth, or the Editor would be saved in the Forth system using SAVE-SYSTEM. To edit a screen type the number of the screen you want to edit, then the word EDIT itself. If there is no such screen a blank one will be created for you.

The screen editor was designed to be very similar to the editor used by Basic. The arrow keys, insert character and delete character keys work exactly the same as they do in Basic. The delete line key erases the line, exactly as in Basic. Three other keys are defined: the BACK key is used to end editing. Once you get out of the editor, you should consider typing a SAVE-BUFFERS command to make your changes permanent. The PROCEED key continues editing onto the next screen. The REDO key continues editing onto the previous screen.

There are some problems associated with this editor. Foremost is the problem of displaying a 1024 character Forth buffer on the 960 character text-mode graphics screen. There obviously is no best solution to this problem. The screen size of 1024 was selected for compatibility with most Forth systems in existence. It is possible to program a bit-map mode editor, but the extra work required is severe. Should you find that the text of a screen causes the display to overflow, you may still edit the screen, most easily by using the RIGHT ARROW key to space over and show the line currently being edited.

The screens developed on disk are in a form that allows editing by the Editor/Assembler or TI-Writer if desired. Note that any character past column 64 will be ignored and there should be 16 lines of 64 characters. Cassette screens are handled using the LOAD/SAVE option of the file management package. This leads to more compact storage and much quicker loads, but means that you will not be able to edit those screens except through the Forth editor.

Cassette/Diskette Operation

Storage of the text forming Forth words is on cassette or diskette as desired. Diskette operation is straightforward with all screens being read and written automatically to disk. If a screen is requested which does not exist on the disk, it will be created as a blank screen (but will not be written to disk unless altered). Disk screens are written in a form similar to that used by TI-WRITER and the Editor/Assembler, which allows use of those modules for editing if desired.

Cassette screens are a little more difficult to operate with than diskette screens. Whenever a cassette screen must be read or written the monitor display is saved during prompting of the transfer. Note that Forth may not save new screens in the order that they were created. The screen number should always be checked. To create a new, blank screen in cassette versions, attempt to read a non-existent screen. When the NO DATA FOUND message is issued select the E (EXIT) option. A blank screen will be created for editing.

Screens may be transferred from cassette to diskette in the following manner. First, read in the cassette screen with the commands CS1 88 LIST (assuming the screen number is 88). Change the residence of the screen with the commands 88 PREV @ ! UPDATE and the command SAVE-BUFFERS will then write the screen to disk unit DSK1.

Memory Usage

Wycove Forth resides mainly in the upper address block of system RAM memory in the Memory Expansion peripheral of the 99/4A. Addresses (given in hexadecimal) >A000 to the value left on the stack by the word LIMIT (approximately >FC00) are reserved for the Forth system. In particular, the addresses under the value left on the stack by the HERE word are the start-up routines and the resident dictionary. Addresses from HERE to the value held at the S0 word are the remaining dictionary space and the parameter or data stack. The dictionary grows upward in this area and the stack grows downward. The return stack is located above the parameter stack to the value held at the R0 word. The terminal input buffer shares this area, growing upwards while the return stack grows downwards. Above the return stack are the I/O buffers, starting at FIRST and extending to LIMIT. Memory cells from LIMIT and above to HI are available for use by the programmer.

In the lower block of RAM, from >2000 to >3FFF, the Editor/Assembler version of Wycove Forth uses system routines that extend from >2000 to >23BA. The remainder of the block is available to the programmer. In the Mini Memory version this whole block is free. In the Extended Basic version, addresses >2000 to >26EA are reserved.

In the scratchpad area of RAM, from >8300 to >83FF, Forth uses the area >8300 to >831F for its workspace registers. This area must not be altered by any GPLLNK routines. The remaining scratchpad area is in general used by the 99/4A system routines, and should be used in programming only with extreme caution.

In the Video Display Processor address space of 0 to >3FFF screen tables, I/O peripheral access blocks, peripheral control blocks and sound lists are stored. The top part of this space is reserved for device service routines. The lowest address available is stored in the main memory at address >8370. Writing to VDP RAM above this address may cause damage to your diskettes, and should only be done with extreme care.

Forth stores a peripheral access block and buffer for internal use in the area stored in the PAB word, initially >1000. Note that as cassette screens are written as a 1024 byte block, addresses up to almost >1500 are used when reading and writing screens from cassette. This space is only used temporarily, moving a block at a time from the main RAM I/O buffers to VDP and then out to the device.

The printer control words use VDP RAM starting at >1300 for a printer peripheral access block, which accumulates characters one at a time and must not be overwritten while the printer is open.

The SAVE-SYSTEM word uses addresses >1500 to >34FF in VDP RAM for storage of the blocks of data to be written out. Thus, this word destroys almost all of VDP RAM.

When in bit-map mode, none of the routines above should be used. If they are to be used after the bit-map graphics are completed the WARNING flag should be set to zero and the Device Service Routine memory should be written to low RAM (starting at the same main memory address as the VDP address, using the lower limit found in >8370). When the screen is returned to graphics mode or text mode those blocks can then be copied back into the VDP RAM space, restoring use of the diskette system.

In the bit-map mode the following addresses are suggested for the screen control blocks: Pattern Descriptor Table from 0 to >17FF, Screen Image Table from >1800 to >1AFF, Sprite Attribute List from >1C00 to >1C80, Color Table from >2000 to >37FF and Sprite Descriptor Table from >3800 to >3FFF. Refer to the bit-map mode demonstration screens for an example.

In the graphics mode the Screen Image Table is stored in addresses 0 to >2FF. The Sprite Attribute List extends from there to >37F, followed by the Color Table. The Sprite Descriptor Table uses addresses >400 to >77F (refer to the Editor/Assembler Manual for details on this table) and is followed by the Pattern Generator Table, starting at address >800. In this table are stored the bit patterns of the 256 characters that can be put on the screen.

Forth starts out in graphics mode, but text mode is more convenient for screen editing. In text mode the screen extends from 0 to >3BF. Refer to example screen one for words that switch between text mode and graphics mode.

Interface to 99/4A Hardware

This section contains a few examples of words allowing interface to the hardware of the 99/4A. Futher examples will be found in the Sample Screens Appendix.

Reading the Joysticks

The following words can be used to read the joysticks. To use these words, provide the joystick number on the stack. Returned will be a flag indicating if the FIRE button is pressed and the Y and X values for the joystick position.

```
HEX
: JBYTE C@ DUP 8 > IF 0FF00 OR ENDIF ;
: JOYSTICK ( n JOYSTICK fire y x : read joystick n)
  8374 C@ SWAP 8374 C!
  ?KEYBOARD DROP 012 =
  SWAP 8374 C!
  8376 JBYTE
  8377 JBYTE ;
DECIMAL
```

Controlling the Interrupts

The byte at hexadecimal >83C2 controls the interrupt routines in the 99/4A console. The top bit of this byte is used to control whether interrupt processing is done at all. If set, there is no processing beyond testing for and performing the user interrupt routine.

The second bit in the byte at >83C2 selects whether the sprites are to be moved. If set, the sprites will not be moved.

The third bit in the byte at >83C2 selects whether the sound processing is to be done. Setting that bit turns off processing of the sound lists. Note that the sound generators should be silenced before setting this bit.

The fourth bit at >83C2 controls whether the QUIT key will cause a return to the main menu. If set, the QUIT key will be ignored by the interrupt routine, and then can be used just as any other key is used. For example, to turn off recognition of the QUIT key enter:

```
HEX 83C2 C@ 10 OR 83C2 C!
```

User Interrupt

It is possible to write an assembler routine and to have it called at every interrupt by storing its address in the word at address >83C4. The routine should start by saving the return address in R11 and loading its own set of registers. This user interrupt routine will be performed after all other interrupt actions (sprite moves, sound list operation, screen blanking, etc.) every sixtieth of a second except when device service routines turn off the video interrupt.

CRU Access

The following words are a sample of code using the Communications Register Unit control words. The purpose of the word DEVICES is to list all devices connected to the computer.

```
HEX

: LIST-NAMES 4008 @ IF 4008 BEGIN
    @ DUP 4 + CR COUNT TYPE DUP @ 0= UNTIL DROP ENDIF ;

: LIST-ROM 4000 C@ 0AA = IF LIST-NAMES ENDIF ;
: LIST-DEV 1 OVER 1 !CRU
           LIST-ROM
           0 SWAP 1 !CRU ;
: DEVICES 0F80 800 DO I LIST-DEV 80 +LOOP ;

DECIMAL
```

Sample Screens

Screen 1 contains a word to clear the screen and words to switch between graphics mode display and text mode display. None of these words require any values on the stack before they are called.

```
( Select Text or Graphics Mode Display)
BASE @ HEX

: CLS 0C EMIT ;

: TEXTMODE 0F0 DUP 83D4 C! 1 !VDPREGISTER
  28 SCREEN-WIDTH ! 13 7 !VDPREGISTER CLS
  84FC 8484 8484 FC84 8 0 DO 1F 8 * 800 + I + !VDP 2 +LOOP ;

: GRAPHICSMODE 0E0 DUP 83D4 C! 1 !VDPREGISTER
  20 SCREEN-WIDTH ! CLS
  3A0 380 DO 13 I C!VDP LOOP
  380 300 DO 0 I C!VDP LOOP ;

BASE !
```

Screens 2 and 3 contain words to ring the bell, and to request either a Y or N answer, or to read in a number.

```
( Utility words: BELL, Y/N )
BASE @ DECIMAL
: BELL 7 EMIT ; ( Ring the bell)
```

```
( Y/N flag : request Y, N, y or n)
( n2 flag n TEST n2 flg2 : TEST KEY) : TEST >R OVER R> = OR ;
: Y/N   0 BEGIN DROP KEY 0
 78 TEST 89 TEST 110 TEST 121 TEST
 UNTIL 0 89 TEST 121 TEST SWAP DROP ;
BASE !
```

```
( #IN d : Read a double number to the stack, setting DPL)
BASE @ DECIMAL
: (#IN)  BLK @ >R   IN @ >R   0 BLK !
  QUERY   BL WORD   HERE 0 0
  ROT DUP 1+ C@   45 = DUP >R +
  -1 DPL ! (NUMBER)
  DUP C@ 46 = IF 0 DPL ! (NUMBER) ENDIF
  C@ ROT ROT   R> IF DMINUS ENDIF
  ROT   32 = IF BL WORD HERE 1+ C@ ELSE 1 ENDIF
  R> IN !   R> BLK ! ;

: #IN TIB @ >R PAD TIB ! BEGIN CURSOR-POS @ (#IN)
  WHILE DROP DROP DUP CURSOR-POS ! 54 GPLLNK DROP
  SCREEN-WIDTH @ OVER OVER MOD - COLUMN-SKIP @ - SPACES
  CURSOR-POS ! REPEAT ROT DROP R> TIB ! ;
BASE !
```

Screen 6 contains a benchmark that was run to compare the speed of Wycove Forth against Basic. This program is to solve what is called the house numbers problem. A four-digit house number falls from a house and breaks into two parts of two digits each. It is observed that the original number was equal to the square of the sum of the two parts. What four-digit numbers have this property? This problem takes 220 seconds to solve with the Extended Basic program that follows. The Forth version takes about 5.3 seconds. A similar program in normal TI Basic also runs in about 220 seconds. To run the problem once type the word HOUSE#S. To run the problem several times (to allow timing) type the word TIMEIT.

```
100 N=1000
110 FOR X=10 TO 99
120 FOR Y=00 TO 99
130 IF (X+Y)*(X+Y)=N THEN PRINT N
140 N=N+1
150 NEXT Y
160 NEXT X
```

```
( SCREEN 6: HOUSE#S BENCHMARK)
BASE @ DECIMAL : TASK ;
2 LOAD 3 LOAD ( #IN and BELL)
( HOUSE#S : run benchmark once)
: HOUSE#S 1000 100 10 DO 100 0 DO DUP J I + DUP * =
  IF DUP . ENDIF 1+ LOOP LOOP DROP ;

( TIMEIT : run benchmark many times)
: TIMEIT CR CR ." House Numbers Benchmark"
      CR CR ." How many times? "
      #IN DROP 1+
      1 DO CR I . HOUSE#S LOOP
      BELL ;

BASE !  ( autostart)  TIMEIT
```

Screens 8 and 9 provide words that use the floating point routines built into the 99/4A. These routines work with 8-byte variables. The workings of the routines are quite similar to the basic arithmetic words of Forth except that the addresses should be on the stack instead of the values. The address of a temporary holding area is left on the stack and the value should then be stored into a variable. Thus, to divide 2 by 3 the following is needed:

```
FVARIABLE X 2 S->F X F!
FVARIABLE Y 3 S->F Y F!
FVARIABLE Z
X Y F/ Z F!
Z F.
```

```
( SCREEN 8: FLOATING POINT ARITHMETIC, SCREEN 1 OF 2)
BASE @ HEX : TASK ;

: FVARIABLE <BUILDS 0 0 0 0 , , , ,
             DOES> ;

: F! 8 CMOVE ;
: F+ 834A 8 CMOVE 835C 8 CMOVE 600 XMLLNK DROP 834A ;
: F- 834A 8 CMOVE 835C 8 CMOVE 700 XMLLNK DROP 834A ;
: F* 834A 8 CMOVE 835C 8 CMOVE 800 XMLLNK DROP 834A ;
: F/ 834A 8 CMOVE 835C 8 CMOVE 900 XMLLNK DROP 834A ;
: F. 834A 8 CMOVE 0 8355 C! 14 GPLLNK DROP 8355 C@ 8300 +
     8356 C@ TYPE ;
: F.R.D 8357 C! SWAP 834A 8 CMOVE DUP 2- 8355 C! 14 GPLLNK
  DROP 8356 C@ SWAP OVER - SPACES 8355 C@ 8300 + SWAP TYPE ;
-->
```

```
( SCREEN 9: FLOATING POINT ARITHMETIC, SCREEN 2 OF 2)
: CIF 6000 @ -DUP IF 2 = IF          8000 ( Extended Basic) ELSE
 7200 ( Mini Memory) ENDIF ELSE 2300 ( E/A) ENDIF ;

: S->F 834A ! CIF XMLLNK DROP 834A ;

: F< F- @ 0< ;
: F= F- @ 0= ;
: F> F- @ 0> ;

: FSQR 834A 8 CMOVE 26 GPLLNK DROP 834A ;

  1000 836E ! ( SET VALUE STACK ADDRESS) BASE !
```

Screens 12 through 14 control a printer, allowing all output that would normally be sent to the screen to be written out on the printer. The user will probably have to alter the printer specification as given after the PAB" word in screen 13 (the second one here) and may wish to alter the line length set just above that. To use the printer the word TO-PRINTER is called. Then all output will be copied to the printer as well as the screen. To restore normal operation use the word TO-SCREEN. Note that the file control words must be loaded before these words. A typical sequence would be 12 LOAD TO-PRINTER.

```
( Printer Control words, screen 1 of 3)
BASE @ DECIMAL
: PRINTER ;
15 LOAD ( File Control words)



: ?PRINT -DUP IF
   CR ." ERROR IN USING PRINTER"
   CR ." DSR RETURN CODE " . DROP

  ELSE DROP ENDIF ;


BASE ! -->
```

```
( Printer Control words, screen 2 of 3)
BASE @
HEX 3100 VARIABLE PRINT-PAB DECIMAL        80 VARIABLE LINE-LEN
: OPEN-PRINT PRINT-PAB @ DUP
  PAB" RS232.LF.BA=1200.PA=N.DA=8"
  DUP 0 VAR OUTPUT + 0 SETPAB              LINE-LEN @ 1+ OVER 4 +
  C!VDP DUP DEVICE-CALL      ?PRINT ;
: CLOSE-PRINT PRINT-PAB @ 1 OVER C!VDP DUP DEVICE-CALL ?PRINT ;
: PRINT-EMIT1 DUP 13 =
  IF DROP PRINT-PAB @ 3 OVER C!VDP
     DUP DEVICE-CALL 0 PRINT-PAB @
     5 + C!VDP     ?PRINT
  ELSE PRINT-PAB @ DUP 5 + DUP C@VDP
     1+ DUP ROT C!VDP SWAP 2+ @VDP +
     1- C!VDP ENDIF ;
BASE !  -->
```

```
( Printer Control words, screen 3 of 3)
BASE @ DECIMAL

: PRINT-EMIT PRINT-PAB @ 4 + DUP 1+
  C@VDP SWAP C@VDP 1- = OVER 13 = 0=
  AND IF 10 PRINT-EMIT1
  13 PRINT-EMIT1 ENDIF PRINT-EMIT1 ;
: EMIT-BOTH DUP SCREEN-EMIT PRINT-EMIT ;

: TO-PRINTER OPEN-PRINT
        ' EMIT-BOTH 2- ' EMIT ! ;

: TO-SCREEN CLOSE-PRINT
        ' SCREEN-EMIT 2- ' EMIT ! ;

BASE !
```

Screens 15 and 16 provide words to allow access to the file system of the 99/4A. The only word that the user will need in the first screen is the PAB" word. On the stack before this word is called should be the VDP address of the desired peripheral access block. The buffer for the block will immediately follow it in VDP memory. After the PAB" word is the file specifier. Refer to the printer control words for an example of normal use. The file specifier must be followed by a double quote mark. Screen 16 defines some of the flag words to be used with SETPAB (which is in the main system, and accordingly can be found in the glossary). For full use of these words the Editor/Assembler manual section on file management should be understood. There are two examples of use of these words to control files: the printer control screens (12 to 14) and the directory words (17 and 18). The user is of course free to define any file manipulation words he/she desires.

```
( File Control Words, screen 1 of 2)
: FILE-WORDS ; BASE @ DECIMAL

: COPY-PAB  1+ ROT DUP 9 0 DO 0 OVER          C!VDP 1+ LOOP DROP
  OVER OVER 9 + + OVER 2+ !VDP
  ROT 1- SWAP 9 + ROT RAM->VDP  ;

: (PAB") R COUNT DUP 1+ =CELLS R> +          >R COPY-PAB ;

: PAB"    34    STATE @
  IF COMPILE   (PAB") WORD HERE C@ 1+     =CELLS ALLOT
  ELSE WORD HERE COUNT COPY-PAB ENDIF    ; IMMEDIATE

BASE ! -->
```

```
( File Control Words, screen 2 of 2)
BASE @ HEX
0 CONSTANT FIXED                          10 CONSTANT VAR
0 CONSTANT DISPLAY                         8 CONSTANT INTERNAL
0 CONSTANT UPDATE-MODE                     2 CONSTANT OUTPUT
4 CONSTANT INPUT                           6 CONSTANT APPEND
0 CONSTANT SEQUENTIAL                      1 CONSTANT RELATIVE

BASE !
```

Screens 17 and 18 contain a sample application of the file control words. This application lists the directory from a specified disk drive. The calling sequence is 2 DIR for instance to list the directory from disk drive number two. Note that the file control words (screens 15 and 16) must be loaded by these screens.

```
( Disk Directory, screen 1 of 2)
BASE @ DECIMAL 15 LOAD ( File control)
HEX
: HEADER 0C EMIT ." Directory of disk "  . CR ;

: READ-A-LINE 2 OVER C!VDP
              DEVICE-CALL
              -DUP IF CR ." Error code:" .
              ENDIF ;

: PRINT-OUT 2+ @VDP ( buffer address)
            DUP C@VDP 1+   HERE SWAP VDP->RAM ( copy to ram)
            HERE COUNT DUP IF CR TYPE 0 ELSE DROP DROP 1 ENDIF ;


BASE ! -->


( Disk Directory, screen 2 of 2)
BASE @ HEX

( FRII n : return code flags)
: FRII FIXED RELATIVE INPUT INTERNAL       + + + ;

( disk# DIR : show a disk directory)
: DIR ( set up pab) DUP HEADER                  1200 DUP PAB" DSK1."
 ( fix disk#) SWAP 1- OVER 0D + DUP            C@VDP ROT + SWAP C!VDP
 ( open pab) DUP 0  FRII
             0 SETPAB DUP DEVICE-CALL
 -DUP IF ." Can't open the file - error code " .
 ELSE DUP READ-A-LINE BEGIN DUP READ-A-LINE DUP PRINT-OUT UNTIL
 DUP 1 OVER C!VDP DEVICE-CALL DROP       ENDIF DROP ;

BASE !
```

Screen 20 is an example of the processing needed to use sprites. The words should be examined to see the actions that are necessary. The user can write words to control sprites in any manner he/she desires. In this example four sprites are created and sent around the screen. This application is similar to the program given in the Editor/Assembler Manual, section 21.7.2. In addition the word REPEATX performs a coincidence check on the sprites. To run 10000 coincidence checks type in 10000 REPEATX and it will print out whenever the red sprite hits another one.

```
( SPRITE EXAMPLE)
BASE @ HEX
: BBL 3C7E 400 !VDP CFDF 402 !VDP FFFF 404 !VDP 7E3C 406 !VDP ;
: SAL 6178 8006 6178 8002 6178 8004 6178 800B D000
  12 0 DO 10 I - 300 + !VDP 2 +LOOP ;
: SML 0404 0000 F808 0000 0CF4 0000 F0F0 0000
  10 0 DO E I - 780 + !VDP 2 +LOOP ;
: SETNUM 837A C! ;

: HITRED 300 @VDP 4 1 DO I 4 * 300 + @VDP OVER = IF
: REPEATX 0 DO HITRED LOOP ;

( ACTIVATE) BBL SAL SML 4 SETNUM

BASE !
```

Screens 24, 25 and 26 are examples of the use of bit-map mode for screen display. Words TO-BIT-MAP and FROM-BIT-MAP switch between bit-map mode and graphics mode. BLANK-SCREEN does exactly that and BLANK-COLORS sets the color table to black on white. The main word used here for drawing is TOGGLE, which expects the x and y bit positions of a screen point on the stack. It then turns that point black if it was white, and vice-versa. The routine DRAW takes directions from the arrow keys and draws on the screen. This obviously is not a very useful example, but the individual user will see the potential. For high speed animation it will be necessary to customize the TOGGLE word. The GUNSIGHT word displays crosshairs on the screen, once again moving them in response to the arrow keys. To run the demonstrations load these screens then type in DEMO1 or DEMO2. The arrow keys are used for controlling the direction of the action (use the FCTN D key to go right for instance). To stop the demonstration press the CLEAR key (FCTN 4).

```
( BIT-MAP GRAPHICS EXAMPLE, SCR 1)
BASE @ HEX : TASK ;

: TO-BIT-MAP 1A01 1800 DO 100 0 DO I J I + C!VDP LOOP 100 +LOOP
  3 4 !VDPREGISTER   FF 3 !VDPREGISTER     38 5 !VDPREGISTER
  6 2 !VDPREGISTER 2 0 !VDPREGISTER        7 6 !VDPREGISTER
  4000 8370 @ DO I C@VDP I C! LOOP
  1C80 1C00 DO D000 I !VDP I 1C00 -  800 * 1 + I 2+ !VDP 4 +LOOP
  4000 3800 DO 0 I !VDP 2 +LOOP ;

: FROM-BIT-MAP  380 300 DO 0 I C!VDP LOOP 3A0 380 DO 13 I C!VDP
  LOOP 4000 8370 @ DO I C@ I C!VDP LOOP
     0 0 !VDPREGISTER 0 2 !VDPREGISTER 0E 3 !VDPREGISTER
  1 4 !VDPREGISTER 900 834A ! 18 GPLLNK 4A GPLLNK DROP DROP ;
BASE ! -->


( BIT-MAP GRAPHICS EXAMPLE, SCR 2)
BASE @ HEX
: BLANK-SCREEN 1800 0 DO 0 I C!VDP LOOP ;
: BLANK-COLORS 1800 0 DO 13 I 2000 + C!VDP LOOP ;
: BIT-TABLE <BUILDS 0 DO C, LOOP          DOES> + C@ ;
  01 02 04 08 10 20 40 80 8 BIT-TABLE BIT

: TOGGLE-BIT DUP 7 AND BIT >R F8 AND OVER 7 AND + SWAP F8 AND
    20 * +                                  DUP C@VDP R> XOR
  SWAP C!VDP ;

: DRAW BLANK-SCREEN BLANK-COLORS 80 80 BEGIN ?KEYBOARD DROP
  DUP 02 - WHILE DUP 9 = IF SWAP 1+ FF MIN SWAP ENDIF
  DUP 8 = IF SWAP 1- 0 MAX SWAP ENDIF DUP 0A = IF ROT 1+ BF MIN
  ROT ROT ENDIF 0B = IF SWAP 1- 0 MAX SWAP ENDIF OVER OVER
 TOGGLE-BIT REPEAT ; BASE ! -->
```

```
( BIT-MAP MODE DEMO)
BASE @ HEX
: DO-BIT OVER OVER TOGGLE-BIT ;     : Y-BIT ROT + SWAP DO-BIT ;
: CROSSHAIRS 2+ DO-BIT 1- DO-BIT 1- DO-BIT 1- DO-BIT 1- DO-BIT
  2+ 2 Y-BIT -1 Y-BIT -1 Y-BIT -1 Y-BIT -1 Y-BIT DROP DROP ;


: DRAW-IT OVER OVER CROSSHAIRS ;
: GUNSIGHT 80 80 DRAW-IT
  BEGIN ?KEYBOARD DROP DUP 02 - WHILE
  DUP 9 = IF DROP DRAW-IT 1+ 0FC MIN DRAW-IT 0 ENDIF
  DUP 8 = IF DROP DRAW-IT 1- 2 MAX DRAW-IT 0 ENDIF
  DUP 0A = IF DROP DRAW-IT SWAP 1+ 0BC MIN SWAP DRAW-IT 0 ENDIF
  DUP 0B = IF DROP DRAW-IT SWAP 1- 2 MAX SWAP DRAW-IT 0 ENDIF
  DROP REPEAT DROP DROP ;
BASE ! -->


( BIT-MAP MODE DEMO, SCR 4)


: DEMO1 TO-BIT-MAP DRAW FROM-BIT-MAP ;

: DEMO2 TO-BIT-MAP GUNSIGHT FROM-BIT-MAP ;
```

Screens 30 to 34 are a screen editor. They are not included in the basic system in order that the user may alter them if desired. As written here the editor is very similar to the editing commands available in Basic. To edit a particular screen, for example screen number twenty, use the sequence 20 EDIT, giving the screen number first. The screen will be displayed starting at line number 8. The arrow keys move up and down in the screen as well as along any particular line, just as in Basic. The insert character and delete character keys also work as in Basic, as does the delete line key. The BACK key completes editing. The PROCEED key moves on to the next screen and the REDO key moves to the previous screen. When editing is complete the user should type the SAVE-BUFFERS word to make the changes permanent.

```
( FORTH SCREEN EDITOR, SCREEN 1 OF 5)
: EDITOR ; BASE @ DECIMAL

: SHOWOTHER DUP 64 -TRAILING TYPE ;

: SHOWCURRENT CURSOR-POS @ ROT ROT SWAP OVER
        DUP 64 TYPE 30 EMIT ;
: SHOWSCREEN SWAP BLOCK 12 EMIT 16 0 DO

  I 0> IF CR ENDIF I 2 .R ." >" OVER I =

  IF SHOWCURRENT ELSE SHOWOTHER ENDIF

  64 + LOOP DROP DROP ;

-->


( FORTH SCREEN EDITOR, SCREEN 2 OF 5)
  ( ALL COMMANDS GET CURSOR-POS RAMADDR CHAR# KEY AND SHOULD
    LEAVE THE STACK UPDATED)
: EDIT-RIGHT >R DUP 63 = IF
  7 EMIT ELSE OVER OVER + C@ EMIT 1+ ENDIF R> ;
: EDIT-LEFT >R DUP 0= IF 7 EMIT ELSE 1- 8 EMIT OVER
 OVER + C@ CURSOR-POS @ SWAP EMIT CURSOR-POS ! ENDIF R> ;


: EDIT-ERASE >R DROP DUP 64 BLANKS OVER CURSOR-POS ! 64 0 DO
  SPACE LOOP 30 EMIT OVER CURSOR-POS ! 0 R> UPDATE ;
: EDIT-DEL >R CURSOR-POS @ >R 63 OVER DO OVER I + DUP 1+ C@ DUP
  EMIT SWAP    C! LOOP SPACE OVER 63 + BL SWAP C! R>
  CURSOR-POS ! R> UPDATE ;

    -->
```

```
( FORTH SCREEN EDITOR, SCREEN 3 OF 5)
  ( EDIT-CHAR GETS CURSOR-POS RAMADDR CHAR# KEY EDIT-MODE AND
    LEAVES THE STACK UPDATED)

: EDIT-CHAR IF ( INSERT MODE)
  >R CURSOR-POS @ >R DUP 63 - IF DUP 1- 62 DO OVER I + DUP C@
  SWAP  1+ C! -1 +LOOP 64 OVER DO
  OVER I + C@ EMIT LOOP ENDIF R> CURSOR-POS ! R>

  ENDIF ( REPLACE MODE)
    >R OVER OVER + R SWAP C! CURSOR-POS @ R EMIT OVER 63 =
    IF 7 EMIT CURSOR-POS ! ELSE DROP 1+ ENDIF R> UPDATE  ;



    -->


( FORTH SCREEN EDITOR, SCREEN 4 OF 5)
: EDIT-LINE OVER CURSOR-POS ! 0 0 >R BEGIN KEY
DUP 31 > OVER 128 < AND IF R EDIT-CHAR 0 ELSE
 DUP 4 = R> DROP DUP >R IF 0 ELSE
  DUP 7 = IF EDIT-ERASE ENDIF
  DUP 9 = IF EDIT-RIGHT ENDIF
  DUP 8 = IF EDIT-LEFT ENDIF
  DUP 3 = IF EDIT-DEL ENDIF
0 OVER 13 = IF 1+ 0 ROT ROT 1 ROT ROT ENDIF
  OVER 15 = IF 1+ 0 ROT ROT 0 ROT ROT ENDIF
  OVER 12 = IF 1+ 1 ROT ROT 0 ROT ROT ENDIF
  OVER 6 = IF 1+ -1 ROT ROT 0 ROT ROT ENDIF
  OVER 11 = IF 1+ 0 ROT ROT -1 ROT ROT ENDIF
  OVER 10 = IF 1+ 0 ROT ROT 1 ROT ROT ENDIF
ENDIF ENDIF SWAP DROP UNTIL ROT DROP ROT DROP ROT DROP R> DROP ;
    -->


( FORTH SCREEN EDITOR, SCREEN 5 OF 5)



: EDIT-SCREEN 8 BEGIN OVER OVER SHOWSCREEN
  EDIT-LINE DUP WHILE SWAP DROP + 0 MAX 15 MIN
  REPEAT DROP ROT ROT DROP DROP ;

HEX 1 LOAD  ( Text  Graphics modes)
: EDIT 83D4 @ SWAP TEXTMODE 0 COLUMN-SKIP DUP @ >R !
  BEGIN DUP EDIT-SCREEN
  DUP ROT + SWAP 0= OVER 0= OR UNTIL
  DROP R> COLUMN-SKIP !
  1000 AND 0= IF GRAPHICSMODE ENDIF 0C EMIT ;

BASE !
```

Screens 40 to 47 are a sample game. This game is given only to demonstrate the speed of Forth as compared to Basic. Imagine trying to get a Basic program to work as fast as this game operates! Games written for the 99/4A can use sprites to speed up the action even more than the character graphics used here. When these screens are loaded the game starts automatically.

```
( DEMO GAME BREAKOUT, SCREEN 1 OF 8)
: TASK ; BASE @ DECIMAL
2 LOAD ( BELL and Y/N)
1 LOAD ( CLS, GRAPHICSMODE)
3 LOAD ( #IN)
 0 VARIABLE SPEED 0 VARIABLE SPVAR 0 VARIABLE SCORE
 0 VARIABLE XPOS 0 VARIABLE YPOS 2 VARIABLE PPOS
 1 VARIABLE YDIR 1 VARIABLE XDIR 0 VARIABLE BEST
 0 VARIABLE BALLS

 0 COLUMN-SKIP ! GRAPHICSMODE
```

-->

```
( BREAKOUT, SCREEN 2)
  ( CLEAR LINE)
: LINE SCREEN-WIDTH @ SWAP OVER * DUP CURSOR-POS ! SWAP
  SPACES CURSOR-POS ! ;
```

-->

```
( BREAKOUT, SCREEN 3)
: INIT1 CLS
  0 LINE ." Enter speed (1-10): "
  #IN DROP 1 MAX 10 MIN 10 * SPEED !
  2 LINE ." How many balls (1-99): "
  #IN DROP 1 MAX 99 MIN BALLS ! ;


: INIT2   CLS 2 LINE 32 0 DO 36 EMIT 36 EMIT LOOP 24 4 DO I
  LINE 36 EMIT 36 EMIT 28 SPACES 36 EMIT 36 EMIT LOOP 10 4 DO I
  LINE 36 EMIT 36 EMIT 28 0 DO 35 EMIT LOOP 36 EMIT 36 EMIT LOOP
  0 SCORE ! 0 LINE ."    Score:     0    Best:" BEST @ 6 .R
  1 LINE   ."              Ball: "  ;

    -->


( BREAKOUT, SCREEN 4)

: PCLR PPOS @ 23 32 * + CURSOR-POS ! ."         " ;
: PSET PPOS @ 23 32 * + CURSOR-POS ! ." ATTTTA" ;

: PADDLE ?KEYBOARD DROP DUP 83 = IF PCLR PPOS @ 1- 2 MAX
 PPOS ! PSET ENDIF DUP 68 = IF PCLR PPOS @ 1+ 24 MIN PPOS ! PSET
  ENDIF 80 = IF BEGIN ?KEYBOARD SWAP DROP 1 = UNTIL ENDIF ;

      -->


( BREAKOUT, SCREEN 5)

: XCHK XPOS @ 2 < IF XDIR @ MINUS XDIR ! 2 XPOS ! BELL ENDIF
  XPOS @ 29 > IF XDIR @ MINUS XDIR ! 29 XPOS ! BELL ENDIF ;

: YCHK YPOS @ DUP 4 < IF 1 YDIR ! 4 YPOS ! 1 SPVAR ! BELL ENDIF
  DUP 23 < IF SPVAR @ 4 MIN SPVAR ! ENDIF
  DUP 19 < IF SPVAR @ 3 MIN SPVAR ! ENDIF
  15 < IF SPVAR @ 2 MIN SPVAR ! ENDIF ;

      -->
```

```
( BREAKOUT, SCREEN 6)
: PCHK 0 YPOS @ 22 > IF
  22 YPOS ! XPOS @ PPOS @ - DUP 6 < OVER -1 > AND
  IF -1 YDIR ! BELL
    DUP 2 > 3 * + 4 - 2 / XDIR !
  ELSE DROP 1+
  ENDIF ENDIF ;

BASE @ HEX
: RND 8379 C@ * 100 / ;
BASE !



      -->


( BREAKOUT, SCREEN 7)
: SCRPOS YPOS @ 32 * XPOS @ + ;
: CLR 32 SWAP C!VDP
  10 YPOS @ - SCORE +! 9 CURSOR-POS ! SCORE @ 6 .R BELL
  YDIR @ MINUS YDIR ! ;

: BALLCHK YDIR @ YPOS +! XDIR @ XPOS +! XCHK YCHK PCHK
  SCRPOS DUP C@VDP 32 = 0= IF CLR ELSE DROP
  ENDIF ;

: BALL 32 SCRPOS C!VDP BALLCHK
  DUP 0= IF 42 SCRPOS C!VDP ENDIF ;
: GAMECHK SCORE @ 700 MOD 588 =
  IF 4 LINE 10 4 DO 31 EMIT 31 EMIT 28 0 DO 35 EMIT LOOP 31 EMIT
  31 EMIT LOOP 112 SCORE +! ENDIF ;
      -->


( BREAKOUT, SCREEN 8 OF 8)

: DELAY SPEED @ SPVAR @ * 10 * 0 DO LOOP ;
: BREAKOUT INIT1 BEGIN INIT2 PSET BALLS @ 0
  DO 2000 SPEED @ / 0 DO DELAY PADDLE LOOP
  49 CURSOR-POS ! I 1+ 2 .R  5 SPVAR !
  2 RND 1 = IF 1 ELSE -1 ENDIF XDIR ! -1 YDIR !
  28 RND 2+ XPOS ! 22 YPOS ! BEGIN 3 0 DO PADDLE LOOP BALL
  GAMECHK ?TERMINAL OR UNTIL
  LOOP SCORE @ BEST @ MAX BEST !
  23 LINE ." Press REDO or BACK "
  0 BEGIN DROP KEY DUP 15 = OVER 6 = OR UNTIL 15 = UNTIL ;


BASE ! BREAKOUT
```

Screens 90 to 92 contain a sample use of the sound generator. This code plays a set of chimes. The S word loads sound list information into the sound list table. Note that the bytes are put on the stack in the opposite order than you would expect. To hear the sound after loading these screens type in the word CHIMES. Note that once the sound list is started it will continue independently of the rest of the program. If you wish to build a loop ringing the chimes more than once you will have to put in a wait loop, that tests the byte at >83CE and waits until it has turned to zero before restarting the sound list.

```
( Chimes - Sound List Example)
BASE @ HEX : TASK ;

3000 83CC !

: S DUP 2+ 0 DO 83CC @ C!VDP 1 83CC +! LOOP ;

DECIMAL 91 LOAD HEX

: CHIMES 3000 83CC !
  83FD C@ 1 OR 83FD C!
  1 83CE C! ;

BASE !
```

```
( Chimes - Sound List Example)
BASE @ HEX
1 E3 FF DF BF 9F 5 S
6 D3 B6 90 1 C5 2 A4 1 8E 9 S
5 D4 B7 91 3 S
4 D5 B8 92 3 S
5 D6 B0 93 04 A7 5 S
6 D7 B1 94 03 S
7 D8 B2 95 3 S
6 D0 B3 96 02 CA 05 S
5 D1 B4 97 3 S
4 D2 B5 98 3 S
5 D3 B6 90 03 85 5 S
6 D4 B7 91 3 S

 -->



7 D5 B8 92 3 S
6 D6 B0 93 02 A4 5 S
5 D7 B1 94 3 S
4 D8 B2 95 3 S
5 D0 B3 96 1 C5 5 S
6 D1 B4 97 3 S
7 D2 B5 98 3 S
0 DF BF 9F 3 S
BASE !
```

Implementation Notes

Wycove Forth is based on FIG Forth, version 1.0, with several minor differences and extensions for the 99/4A environment. This section is intended to be a guide to some of the areas of conflict in Forth systems.

The keyboard control word ?KEYBOARD, while essential for implementing game programs, is non-standard. For portability the word KEY should be used.

Words such as INPUT-LINE, CURSOR-POS, COLUMN-SKIP, !VDP, !CRU etc. are specific to the 99/4A hardware and should not be considered portable. In addition for portability of disk files only the BLOCK word should be used.

The DO ... step +LOOP construct is not standard throughout all Forth systems when used with a negative step value. Wycove Forth follows the convention of not allowing the index to reach the lower limit.

Two words in the FIG Forth system have different operation in Wycove Forth. The word M/MOD in Wycove Forth operates with signed numbers (as does M/ in FIG Forth). The word U/ is not available, but can be constructed using the word U/MOD with care.

The word HELP is used to print out the dictionary. Some systems use VLIST or CATALOG.

Errors and Messages

In the disk versions of Forth screens 4 and 5 contain the error messages and informative messages. When running the cassette system you will only get the error number. In addition, certain errors generated by the disk system are given as numbers (the error may recur in trying to read the text of an error message). The following is an explanation of the meaning of these errors:

Error 1: The stack is empty.
At the end of execution of a line, the interpreter discovered that one of the words you used had insufficient items on the stack to handle. For instance, if you typed 5 + there might be no number on the stack to add to the five.

Error 2: The dictionary is full.
The stack and dictionary have grown so much that there is no longer any free space between them.

Message 4: is not unique.
The name of the word you are defining is already in use. This message is merely a warning that you may not have intended to create a new word by the same name. The new word will render the old one unusable, unless it is removed via the FORGET word.

Error 6: Illegal block number requested.
The block number requested is illegal. Legal values are from 1 to 1999. (Internal words add an offset to this number in order to access different devices.)

Error 7: The stack is full.
This error is similar to error 2, with either the stack or the dictionary getting too large.

Error 10:
Read error - this can include many problems, but it generally indicates some problem in reading screens.

Error 11:
This error is normally issued when the writing of a screen cannot be completed for some reason. (Not enough space?)

Error 12:
This error indicates that the file could not be closed.

Error 13:
This error indicates that a file could not be opened. The file will probably exist but have different characteristics than that required by Forth (Display, Variable 80).

Message 15: Wycove Forth for the TI 99/4A
This message is issued by TRIAD at the bottom of each page of three screens.

Error 17: is legal only within a colon definition.
The operation you are attempting can only be done in a colon definition. This refers to control structures such as those starting with DO, IF and BEGIN.

Error 18: is not legal within a colon definition.
The operation you are attempting can not be done in a colon definition. This generally refers to the inclusion of a second colon in the definition.

Error 19: The expression contains unpaired conditionals.
The control structures in your colon definition are not properly nested. That is, you have overlapped different control structures such as in IF BEGIN ENDIF UNTIL.

Error 20: The definition has not been finished.
A control structure in your colon definition is unfinished. This means that you have started an IF, DO or BEGIN structure but have not finished it.

Error 21: is within the protected dictionary.
The words you are trying to forget are in the protected dictionary (i.e. are located at an address less than the value stored in the FENCE word). You should not delete these words, as they are part of the resident system.

Error 22: should only be used while loading.
You can only use the --> word while you are loading screens.

Error 24: May destroy a vocabulary.
If you forget the words you desire, you will destroy links in another vocabulary.

Message 27:
This message contains some abbreviations commonly used in forming dates.

The actual error screens are:

```
*** Error Messages for Wycove TI 99/4A Forth ***
The stack is empty.
The dictionary is full.
Error 3
is not unique.
Error 5
Illegal block number requested.
The stack is full.
Error 8
Error 9
Read error on device.
Can't write to device.
Error on closing file.
There was no such screen.
Error 14
   Wycove Forth for the TI 99/4A


*** Message screen 2 for Wycove TI 99/4A Forth ***
is legal only within a colon definition.
is not legal within a colon definition.
The expression contains unpaired conditionals.
The definition has not been finished.
is within the protected dictionary.
should only be used while loading.
Error 23
May destroy a vocabulary.
Error 25
Error 26
JanFebMarAprJunJulAugSepOctNovDecMonTueWedThuFriSatSun
Error 28
Error 29
Error 30
Error 31
```

Glossary of Words

This section contains a glossary of the words that are contained in the basic release of Wycove Forth. Other words are contained on the sample screens distributed.

Each word definition consists of a prototype definition on the first line, followed by explanatory text. The prototype line shows the effect of the word on the data or parameter stack. It consists of three parts. For example, take the definition below:

n2 n1 **/MOD** n3 n2/n1

The word itself is printed in an expanded typeface. It is preceded by an indication of the contents of the stack when it is called. It is followed by the contents of the stack when it has completed. For this word there would be two 16-bit signed numbers on the stack at the time it is called. The top number on the stack is n1, which is the item on the right of the list of stack items. This is also the manner in which stack values are displayed by the S. word. The second item on the stack is n2. The purpose of this word is to divide n1 into n2 and to return the quotient and the remainder of the division. The first thing to notice in examining the result of the word is that the input values n1 and n2 are removed from the stack. Left in their place are two new values. Second on the stack is the remainder value, n3. (The descriptive text that accompanies the word would say that it was the remainder.) On the top of the stack is the quotient. In this word the value of the result can be expressed consisely with aritmetic notation. Often another name (n4 for instance) would be chosen and explained in the text.

The word being defined is expressed in capital letters. All predefined words are in capitals, although you can use lower case letters for your own words if desired. The items detailing what is on the stack are in lower case letters in the prototype. This is intended to convey the idea that they are to be replaced by whatever is desired. These items follow a standard form as in the following table.

Stack item	Meaning
n, n1, n2	a 16-bit signed number.
d, d1, d2	a 32-bit signed number.
u, u1, u2	a 16-bit unsigned number.
ud, ud1, ud2	a 32-bit unsigned number.
char	an ASCII character code, stored as a 16-bit number.
count	a 16-bit signed number, used as a byte count.
bitcount	a 16-bit signed number, used as a bit count.
flag	a 16-bit signed number, used as a logical flag.
addr	a 16-bit address in the main computer memory.
vdpaddr	a 16-bit address in Video Display Processor memory.
cruaddr	a 16-bit address in the Communications Register Unit address space.
text	any amount of text, up to the next delimiter.
NAME	indicates that a word name should follow the word.
(n)	values in parentheses are not left on the stack if the flag is false.

The addresses used come in three varieties. First there are the main memory addresses. These refer to ROM (Read Only Memory) or RAM (Random Access Memory that can be read or written to) in the 99/4A. They are usually quoted in hexadecimal. Addresses 0 to hex >1FFF and >4000 to >6FFF are ROM and cannot be written into. Addresses >2000 to >3FFF, >8300 to >83FF and >A000 to >FFFF are RAM memory and can be altered (if you know what you're doing). Addresses >8000 to >82FF are just duplicate addresses for the >8300 to >83FF block. Addresses >8400 to >9FFF are used for control of system peripherals and should not be accessed unless you are familiar with the internal workings of the machine. Finally, addresses >7000 to >7FFF are RAM if you are using the Mini Memory module, or ROM if you are using the Editor/Assembler or Extended Basic module. To determine which areas of RAM memory are used by Forth, refer to the Memory Usage section of this manual.

The Video Display Processor memory addresses range from 0 to >3FFF hexadecimal. The screen, sound list, color and sprite tables and disk control blocks are all stored in this area. The user should refer to the Editor/Assembler manual before writing into this space. The screen control tables used by Forth are in the same place as used by the Editor/Assembler, even in the Extended Basic version.

The Communications Register Unit addresses are bit addresses. The values used by Forth are multiplied by two before being used in instructions, so they refer to actual pin numbers. Please refer to the appropriate system documents for a complete explanation of CRU access.

n addr !
Store the second item on the stack, a 16-bit number, into the RAM address given. On the 99/4A RAM addresses range from (hexadecimal) >2000 to >3FFF, >8300 to >83FF, >A000 to >FFFF. In addition, with the Mini Memory version addresses >7000 to >7FFF are also RAM. The RAM address specified should be even. If it is odd the address one less than it is used.

n cruaddr bitcount !CRU
Write the number of bits specified of the data value n into the CRU address space starting at the CRU address given. The low order bit of n will be written to the first CRU bit, the second lowest to the second CRU bit, etc. The bitcount can be from 1 to 16.

!CSP
Internal to Forth. Store the current stack pointer into the CSP word. This word is used by the colon definition words.

n vdpaddr !VDP
Store the 16-bit value n into the two bytes of VDP RAM starting at the Video Display Processor address given. The address need not be even. Valid VDP addresses for use with this word range from 0 to (hexadecimal) >3FFE.

n2 n1 !VDPREGISTER
Store the 8-bit value n2 into the VDP register specified by the top value on the stack, n1 (which should be a number from 0 to 7). This command is used to change display attributes. Refer to the Editor/Assembler manual for details on VDP register contents.

ud1 # ud2
In formatted numeric output, removes the least-significant digit of the 32-bit number on the top of the stack and puts it into the hold area. If the number is zero, a zero is stored.

ud #> addr count
Completes formatted numeric output, leaving the address of the formatted number and the length on the stack in a form suitable for printing via use of the TYPE word.

ud1 #S ud2
In formatted numeric output, puts the remaining digits of the 32-bit number that is on top of the stack into the hold area. At least one digit is always produced.

' NAME addr
The tic word, a single quote mark, leaves on the stack the address of the parameter field of the word that follows it.

(text)
The left parenthesis tells FORTH to ignore everything up to the next occurrence of a right parenthesis and is normally used to add comments. There must be a blank after the left parenthesis, but need not be one before the right parenthesis.

n (+LOOP)
Internal to Forth. Used to implement the +LOOP word internally. The value on the stack is used as an increment to the do loop index. If the loop is not complete the code offset in the following memory cell is used to return to the start of the loop. Otherwise the loop is terminated.

(_")
Internal to Forth. Used to type the following in-line text, which is preceded by a byte count.

(;CODE)
Internal to Forth. Causes the latest word defined to use the assembler code which follows immediately in-line.

(ABORT)
Internal to Forth. Performs a warm start, clearing the stack, resetting the number base to decimal and resetting the context and current vocabulary to Forth.

n2 n1 (DO)
Internal to Forth. Performs the processing required at execution time for the DO word.

addr2 addr1 (FIND) (addr3) (count) flag
Internal to Forth. Given the starting address of the context vocabulary on top of the stack (addr1) and the address of a string containing a word name (preceded by a byte count), finds the dictionary address of the word and returns a flag indicating success of the search. If the word cannot be found, only the flag (0 or false) is left on the stack. If the word is found the parameter field address and the count field are left under the true flag.

n2 n1 (LINE) addr count
Internal to Forth. Given a screen number n1 and the line desired, n2, in the screen, assures the block is in memory and returns the address and length of the line.

(LOOP)
Internal to Forth. This word does the execution time work required from the colon definition word LOOP.

d1 addr (NUMBER) d2 addr2
Internal to Forth. Used in decoding numbers. Examines the characters in the number that starts at addr+1 and adds them to the 32-bit number on the stack. The address left is the first non-digit encountered.

n2 n1 * n2*n1
Multiplies the top two 16-bit signed numbers on the stack, leaving a 16-bit product.

n3 n2 n1 */ n4
Multiplies the second and third values on the stack, generating a 32-bit intermediate product, then divides by the top element on the stack. Leaves the result, a 16-bit number.

n3 n2 n1 */MOD n5 n4
Multiplies the second and third values on the stack, generating a 32-bit intermediate product, then divides by the top element on the stack. The top item left on the stack is the result of the division, referred to as the quotient. The second item on the stack is the remainder. The remainder will have the same sign as the dividend, the product of n3 and n2.

n2 n1 + n2+n1
Adds the top two numbers on the stack, leaving the sum.

n addr +!
Adds the second value on the stack to the 16-bit value in memory at the RAM address specified. The RAM address should be even. If it is odd the even address one less than it is used.

n2 n1 +— n3
Leaves n2 if n1 is positive or zero and -n2 if n1 is negative. Used internally to generate absolute values.

addr +BUF addr2 flag
Given a buffer address, returns the next buffer address in memory and the distance that address is from the address in PREV. This distance is normally used as a flag to indicate whether the circular search through the buffers has returned to PREV yet.

n +LOOP
In the ... n2 n1 DO ... step-value +LOOP ... construction of the colon definition, uses the number on the stack as an increment for the do loop counter. If the number is positive the loop continues if the new loop index is less than the limit (n2). If the number is negative the loop continues if the new loop index is greater than the limit.

n ,
The comma puts the 16-bit value on the stack into the next memory cell in the dictionary, advancing the dictionary pointer by two bytes.

n2 n1 — n2-n1
Subtracts the top number on the stack from the second number on the stack.

——>
Used only in loading screens. When this word is encountered in a screen, loading continues to the next screen.

n —DUP n (n)
Duplicates the top value on the stack only if it is non-zero. Otherwise the stack is not changed.

—FIND NAME (addr) (count) flag
Internal to Forth. Attempts to find the word next in the input in the context vocabulary. If it cannot be found in the context vocabulary, the current vocabulary is searched. If the word is not found a flag of 0 (false) is left on the stack. Otherwise the parameter field address of the word and the count field are left under a true flag (1).

addr count —TRAILING addr count2
Examines the string whose starting address and length are on the stack and adjusts the count so as to delete all trailing blanks.

n .
The period prints the top number on the stack as a signed 16-bit value in the current base.

." text"
This word consists of a period and a double quote mark. It prints all the text that follows it up to the next double quote. There must be a blank between the ." and the text to be printed.

n2 n1 .LINE
Prints out the given line number (n2) from the desired screen (n1).

n2 n1 .R
Prints out the signed 16-bit number n2 in a field n1 characters long. If the number is smaller than the field it will be preceded by enough blanks to make it right justified in the field. If the number is larger than the field width it will be printed out in full, exceeding the field width.

n2 n1 / n2/n1
Performs the integer division of n2 by n1. Any remainder from the division is discarded.

n2 n1 /MOD n3 n2/n1
Divides n1 into n2 returning both the integer result of the division and the remainder. The sign of the remainder, which is the second item on the stack, is the same as the sign of the dividend n2.

0 n
Leaves the number zero on the stack. This word can be used with the CASE: word.

n 0< flag
Tests the top value on the stack to see if it is less than zero, returning a logical flag, 1 (true) if the number is negative and 0 (false) if it is zero or positive.

n 0= flag
Tests the top value on the stack to see if it is zero, returning a logical flag. If the number on the stack is zero a value of one (true) will be left in its place. If the number on the stack is non-zero a zero (false) will replace it. This word can be used as a logical complement (not) operation.

n 0> flag
Tests the top value on the stack to see if it is greater than zero. If the number on the stack is negative or zero it will be replaced with zero (false), otherwise a value of one (true) will be left on the stack.

flag 0BRANCH
Internal to Forth. Used to branch to a code address whose offset is in-line if the top element on the stack is zero.

1 n
Leaves the value 1 on the stack. This word can be used with the CASE: word.

n 1+ n+1
Adds one to the value on top of the stack.

n 1- n-1
Subtracts one from the value on top of the stack.

2 n
Leaves the value 2 on the stack. This word can be used with the CASE: word.

n 2+ n+2
Adds two to the top value on the stack.

n 2- n-2
Subtracts two from the top value on the stack.

3 n
Leaves the value 3 on the stack. This word can be used with the CASE: word.

: NAME words ;
Used to define new words. The name of the new word follows the colon. Words following the name are the actions to be taken when the new word is used, up to the ; word that terminates the definition.

;
Used to end the definition of a new word, as started with the : word.

;CODE
Used to end the definition of an assembler coded new word. There is no asembler available currently for Wycove Forth.

;S
Internal to Forth. Completes execution of a colon definition by returning to execution of the calling word at the address stored on the return stack.

n2 n1 < flag
Tests to see if the second element on the stack is less then the top element on the stack. Returns a logical flag, 1 (true) if the second element on the stack is less than the top element on the stack and 0 (false) otherwise.

(n) ud <## (n) ud
Starts formatted output of a 32-bit number d. If the SIGN word is to be used there must be a 16-bit value n under the 32-bit number to provide the sign. The form of the output is determined by use of the words #, #S, HOLD and SIGN before the terminating #>. The necessary values for formatted output can be produced on the stack in the following manners:

Value_on_stack_is	Print_with
signed 16-bit (n)	DUP S->D <# ... SIGN #>
unsigned 16-bit (u)	0 <# ... #>
signed 32-bit (d)	SWAP OVER DABS <# ... SIGN #>
unsigned 32-bit (ud)	<# ... #>

<BUILDS
This word is used in defining new words in colon definitions. It is usually paired with the word DOES> in a definition of the form ... : name <BUILDS words1 DOES> words2 ; ... where the words between <BUILDS and DOES> are executed to create a new object and the words after DOES> are executed whenever the object is invoked. When the word is invoked the address of its parameter field is left on the stack for the words following DOES> to work with.

n2 n1 = flag
Tests the top two numbers on the stack to see whether they are equal. If so a true value (1) is left on the stack. If not a false value (0) is left. To test for unequal values in most cases the - word can be used, as the difference will be the proper flag.

n =**CELLS** n2
Forces the number on the stack, usually an address, to be an even number. If the number is already even no change is made. Otherwise the number is incremented by one.

n2 n1 > flag
Tests to see whether the second number on the stack is greater than the top number on the stack. If so a true value (1) is left on the stack. Otherwise a false value (0) is put on the stack.

n **>R**
Moves the top element on the data stack to the return stack. This word can be used in conjunction with the words R and R> to temporarily remove a number that is in the way on the stack. It can only be used in a colon definition. The user must ensure that the numbers placed on the return stack do not interfere with the operation of DO loops and are removed before exiting the word that put them there.

addr **?**
Gets the signed 16-bit number located in main memory at the given address. The number is then printed out. Note that accessing addresses hexadecimal >8400 to >9FFF with this word may cause Forth to lose control of the computer.

?COMP
Internal to Forth. Tests to see that Forth is currently doing a colon definition. If not, issues error number 17.

?CSP
Internal to Forth. Tests the stack to make sure that all control structures in the colon definition have been completed. If not, issues error number 20.

n2 n1 **?DISKERROR**
Internal to Forth. Causes the disk error message n1 to be issued if the status n2 on the stack is non-zero. The status itself is also printed.

flag error# **?ERROR**
Causes the error message number given to be issued if the flag on the stack is true. In addition, a warm start is performed (see the word ABORT).

?EXEC
Internal to Forth. Tests to see that Forth is not in compile mode, that is, is not performing a colon definition. If not in execution mode, error 18 is issued.

?KEYBOARD char status
Scans the keyboard and returns the status and the key pressed. If no key is currently pressed status will be Ø and the key value will be 255. If a new key press is detected status will be 1 and the ASCII value of the key pressed is stored on the stack. If the key detected previously is still pressed the status returned will be -1 and the ASCII key value will be on the stack.

This word can also be used to read the joystick positions. If the proper joystick number is written to the byte at hexadecimal address >8374 after a call to ?KEYBOARD the position can be read from addresses >8376 (Y) and >8377 (X). After reading the position the byte at >8374 should be restored to zero.

?LOADING
Internal to Forth. Checks to see whether the LOAD word is currently operating. If not, error 22 is issued.

n2 n1 **?PAIRS**
Internal to Forth. Checks to see whether the control structures in colon definitions are properly nested. The code values associated with the control structures are compared and if not equal, error message 19 is issued.

?STACK
Internal to Forth. Checks to see whether the stack is empty or has overwritten the dictionary. If so issues error messages 1 or 2 respectively.

?TERMINAL flag
Leaves a flag on the top of the stack indicating whether the CLEAR key (FCTN 4) is pressed. If so the flag returned will be true.

addr **@** n
Reads a 16-bit number from the main memory address space. The address given should be even, otherwise the value will be retrieved from the address one less than the specified address. Addresses from hexadecimal >8400 to >9FFF are used to control specific hardware on the 99/4A and reading from these addresses may cause Forth to lose control of the machine.

cruaddr bitcount **@CRU** n
Reads information from the Communications Register Unit address space. The address specified is a CRU starting bit number, which is doubled before being passed to the assembler instructions. The number of bits of information specified by count (from 1 to 16 bits) are read and stored as a single value on the stack. The first bit read will be stored in the least significant bit position of the top stack element.

vdpaddr **@VDP** n
Gets the 16-bit value stored in the Video Display Processor memory at the address taken from the top of the stack. Valid VDP addresses for this word are 0 to hexadecimal >3FFE. The address need not be even for access to VDP RAM.

ABORT
Causes Forth to clear the stack, reset the number base and vocabularies and return control to the user. This is often referred to as a warm start.

n **ABS** n2
Takes the absolute value of the number on top of the stack. If the number n is negative the result is -n, otherwise the stack is unchanged.

AGAIN
Internal to Forth. Used to implement the REPEAT word. Can also be used to create a loop in the form ... BEGIN words AGAIN ... that cannot be exited except via the QUIT or ABORT words.

count **ALLOT**
Allots the desired number of bytes from the dictionary for storage. This word is often used to allocate storage for variables.

n2 n1 **AND** n3
Performs the logical and operation on the two 16-bit values on the top of the stack. Each bit in n3 is set independently. If both corresponding bits in n2 and n1 are ones, the bit in n3 is set to one. Otherwise that bit is set to zero.

B/BUF n
Leaves on the stack the number of bytes in each buffer.

B/SCR n
Leaves on the stack the number of buffers in each screen.

addr **BACK**
Internal to Forth. Subtracts the current dictionary pointer from the address on the stack and stores the result into the dictionary, incrementing the pointer.

BASE addr
Returns the address of the memory cell in which the current number base is stored.

BEGIN
Used in colon definitions to create a block of words that is repeated until a certain event happens or while a certain event has not happened. In the sequence ... BEGIN words test UNTIL ... the words are executed until the test becomes true. In the sequence ... BEGIN words1 test WHILE words2 REPEAT ... the words are repeated while the test returns a true (non-zero) value. When the test becomes false, the block is exited without doing the words between the WHILE and the REPEAT. The sequence ... BEGIN words AGAIN ... can be used to create an infinite loop that can only be left via the QUIT or ABORT words.

BL char
Puts the ASCII character code value for a blank space on the stack.

addr count **BLANKS**
Fills with blanks the block of RAM memory starting at the address given and of the length in bytes specified.

BLK addr
Leaves the address of a memory cell used by the LOAD word to store the block number of the screen currently being loaded. Input from the keyboard is treated as coming from block number zero.

n **BLOCK** addr
The top element on the stack is a block number or screen number. This word checks to see whether that block is currently in memory. If not, it is read in. Then the address of the start of the block is left on the stack. In Wycove Forth all blocks are 1024 characters long.

BRANCH
Internal to Forth. Branches to the address whose offset is stored in-line in the code of the word.

BUFFER addr
Internal to Forth. Used in reading in disk and cassette storage blocks. This word returns the address of a free buffer after writing out the old contents if necessary.

n addr **C!**
Stores the single byte value that is the second element on the stack to the RAM address given. Valid RAM addresses are hexadecimal >2000 to >3FFF, >8300 to >83FF and >A000 to >FFFF. In addition in the Mini Memory version, addresses >7000 to >7FFF are also available.

n vdpaddr **C!VDP**
Stores the single byte value that is the second element on the stack to the Video Display Processor memory address specified. Valid VDP addresses range from hexadecimal 0 to >3FFF. Note that many tables are stored in VDP memory, including some used by peripherals attached to the computer. In particular VDP addresses larger than the value stored in RAM address >8370 should not be altered.

n **C,**
Allocates a single byte of dictionary space and puts the top element of the stack into that byte. Note that this word should always be used in pairs. The dictionary pointer should be left at an even address.

addr **C@** n
Reads the 8-bit value from the main memory address specified. Note that accessing addresses in the range >8400 to >9FFF may cause Forth to lose control of the computer.

vdpaddr **C@VDP** n
Reads the byte value from the Video Display Processor memory address specified. Valid VDP addresses are 0 to >3FFF.

n **CASE:** NAME word0 word1 ... wordn-1 ;
Used in a word definition of the type given above. The word created allows choice of execution of one of a number of words based on the value on top of the stack. If the word defined above is called with the value 1 on the stack then word1 would be executed. Note that the choices all have to be single words, they cannot be numbers. The value n used when defining the word is the number of choices available. If it is desired to use numbers, the numbers 0, 1, 2 and 3 are already defined as words and other numbers can be made into words (instead of literals) via a definition of the form ... : 8 8 ; ... which makes 8 a word instead of a literal.

addr **CFA** addr2
Internal to Forth. Used to get the code field address addr2 given the parameter field address of a defined word.

addr1 addr2 count **CMOVE**
Moves the number of bytes specified as the top value on the stack from the main memory source address (the third item on the stack) to the RAM address given as the second item on the stack. The source and destination blocks should not overlap except that the source block can overlap the destination block if it starts at a higher address in memory.

COLD
Performs a cold start. All user-defined words are discarded, the stack is cleared, etc. The effect is to leave the system in the same state as when first started up.

COLD-START
Resets several words in the Forth system to the effect that a new cold start will start the system up including the words defined at the time that COLD-START is executed. This word is used in conjunction with SAVE-SYSTEM to allow the user to save the Forth system with whatever words he or she wishes to include. The initial values of the dictionary pointer, vocabulary link and fence are affected.

COLUMN-SKIP addr
Returns the address of the memory cell used to tell Forth how many columns to skip on each side of the video display. For systems operated with a television set the value stored in this address should be 1 or 2 to ensure that all characters typed are visible. Systems running with a monitor may use this word to use the whole screen for display. This word can also be used when writing programs that are to write on the whole display, for instance game programs.

COMPILE
Internal to Forth. Used to compile the in-line address that follows this word.

n **CONSTANT** NAME
Defines a constant whose value is preset to the value of the element on the top of the stack. The name of the constant follows this word. Whenever the name is invoked at a later time, the value of the constant will be put on top of the stack.

CONTEXT addr
Leaves on the stack the address of the memory cell that is used to store the name field address of the first word in the context vocabulary, that is, the set of words that are scanned to determine what definitions are in effect.

addr **COUNT** addr+1 count
Internal to Forth. Used to split the length of a string away from the text of the string. The first byte in the string should be the byte count.

CR
Outputs a line feed and carriage return. This will cause subsequent printing to start on a new line.

CREATE NAME
Internal to Forth. Creates a new dictionary entry, taking the new name from the input string.

CS1
Sets the screen offset so that future references to screens 1 to 1999 will specify screens that are on cassette unit one.

CSP addr
Leaves on the stack the address of the compiler stack pointer cell, used to store the stack pointer when the compiler is entered.

CURRENT addr
Leaves on the stack the address of the memory cell used to store the name field address of the first word of the current vocabulary, that is the vocabulary to which new word definitions are added.

CURSOR addr
Leaves on the stack the address of the memory cell that is used to store the ASCII character code for the cursor. The cursor character can be changed or eliminated by changing the value stored at this address.

CURSOR-POS addr
Leaves on the stack the address of the memory cell used to store the location of the cursor on the screen. By changing the value at this address printing to the screen can start at any character position and input can be requested at any location. Screen locations are numbered from 0 to 767 (32 * 24 - 1) in graphics mode and 0 to 959 (40 * 24 - 1) in text mode. Given a row number R from 0 to 23 and a column number C from 0 to 31 in graphics mode, the screen position would be R * 32 + C.

d2 d1 **D+** d2+d1
Adds the two 32-bit numbers on the top of the stack.

d n **D+-** d2
If the 16-bit number on top of the stack is negative, multiplies the 32-bit number by -1. Otherwise the double number is left alone. Used internally to generate the absolute value of a 32-bit number.

d **D.**
Prints out the 32-bit signed double number on the top of the stack.

d n **D.R**
Prints out the 32-bit signed number given in a field width specified as the top element on the stack. If the number printed does not need as much space to print as requested blanks will be added in front of the number so that it is right justified in the field width. If the number is longer than the desired field width it will be printed out in full, exceeding the field width.

d **DABS** d2
Takes the absolute value of the 32-bit number on top of the stack.

DECIMAL
This word sets the number base so that all numbers read in and printed out will be done in decimal or base ten format. This word is usually used to switch back to decimal notation after use of the HEX word, which causes hexadecimal (base 16) notation to be used.

DEFINITIONS
Makes the current vocabulary the same as the context vocabulary. This means that any new word definitions will be added to the specified vocabulary.

vdpaddr **DEVICE-CALL** n
Internal to Forth. Used to call the appropriate system routine (GPLLNK or DSRLNK) whether the device being used is cassette or disk. The address given is a peripheral access block address. The value left on the stack is the status byte returned by the system routine. If the device chosen is a cassette unit, all auxiliary operations required prior to a call to GPLLNK are performed.

char n **DIGIT** (n2) flag
Internal to Forth. Used to decode a single digit of a number being read in, taking into account the present number base. The number base is the top item on the stack. If the character is in fact a valid digit for the base system currently in effect the value of the character is left on the stack along with a true flag (1). If the character is not a valid digit a flag of false (0) is left on the stack alone.

d **DLITERAL** (d2)
Internal to Forth. Used in decoding double numbers from the input stream. If the system is in compilation mode the literal is compiled and removed from the stack. Otherwise it is left on the stack.

d **DMINUS** -d
Multiplies the 32-bit number on top of the stack by -1, making it positive if it was negative, and vice-versa.

n2 n1 **DO**
Used in colon definitions to construct loops. In the sequence ... n2 n1 DO words LOOP ... the words will be executed n2-n1 times. The first time through the do loop index will be equal to n1. In each subsequent iteration the loop index is increased by one while it remains less than the maximum index n2. The index value never is allowed to actually reach the maximum value n2. The value of the loop index can be accessed through use of the I word.
In the sequence ... n2 n1 DO words step-value +LOOP ... the words are again executed repeatedly while the do loop index is incremented (or decremented) by the value on top of the stack when +LOOP is reached. When the step-value is positive, the loop will be continued if the new loop index is still less than the maximum loop index n2. When the step-value is negative the loop will be continued if the new index is greater than the specified minimum loop index value n2.

DOES> addr
Used in a colon definition of the form ... : name <BUILDS words1 DOES> words2 ; ... this word precedes the executable portion of the definition. Refer to the explanation of the <BUILDS word for further details.

DP addr
Leaves on the stack the address of the memory cell used to store the dictionary pointer (the pointer to the next free memory cell in the dictionary space).

DPL addr
Leaves on the stack the address of the memory cell used to store the decimal point location in the last number read in. That cell is set to -1 if there was no decimal point in the last read number. In that case the number will be stored on the stack as a 16-bit number. If there was a decimal point in the number last read the decimal point location cell records the number of digits that followed the decimal point and the number is stored as a 32-bit value on the stack.

n **DROP**
Discards the top element from the the stack.

DSK1
Sets the screen offset so that screens 1 to 1999 will be found on disk unit one.

DSK2
Sets the screen offset so that screens 1 to 1999 will be found on disk unit two.

DSK3
Sets the screen offset so that screens 1 to 1999 will be found on disk unit three.

vdpaddr n **DSRLNK** n2
Calls the system routine DSRLNK to link to device service routines for peripheral access. The stack must contain a peripheral access block address (in VDP ram) and a code-value (n) of 8 for device access and 10 for subroutine access. The device service routine status is left on the stack as n2. If the device could not be found the status value will be 8. Otherwise the status value will be as listed in the Editor/Assembler manual, with zero meaning that there was no error.

n **DUP** n n
Duplicates the top item on the stack.

ELSE
Used in colon definitions in the sequence ... test IF words1 ELSE words2 ENDIF ... to specify that if the flag value tested by the IF word is false or zero the words between the ELSE and the ENDIF will be executed.

char **EMIT**
Prints out the ASCII character corresponding to the code value specified by the top item on the stack. For output to the screen the code value 7 causes a beep, 8 causes a backspace, 10 causes a line feed, 12 causes the screen to be cleared and 13 causes a carriage return. This routine calls the word SCREEN-EMIT to do the work. The printer control words change this word to copy the output to the printer as well as the screen. Thus, if you want the output never to go to the printer use the word SCREEN-EMIT instead.

EMPTY-BUFFERS
Clears the disk and cassette screen buffers, discarding all current contents.

addr char **ENCLOSE** addr n3 n2 n1
Internal to Forth. Used to break down the input line into separate words. The next word is found in the string starting at addr. The ASCII character char is used as the delimiter. The numbers left on the stack are the starting character of the word (n3), the ending character (n2) and the character to be scanned next for the next word (n1).

flag **END**
Used in colon definitions in the sequence ... BEGIN words test END ... to repeat the words until the test becomes true. This word is equivalent to the word UNTIL.

ENDIF
Used in colon definitions in the sequence ... test IF words ENDIF ... or ... test IF words1 ELSE words2 ENDIF ... to signal the end of the conditional block. See the description of the IF word for more details. This word is equivalent to the word THEN.

addr count **ERASE**
This word sets to zero the main memory locations in the block starting at the given address, with the byte count given on top of the stack.

n **ERROR**
Internal to Forth. Used to issue an error message, the number of which is on the top of the stack.

addr **EXECUTE**
Executes the word specified by the code field address given.

addr count **EXPECT**
Reads in characters from the keyboard until either the ENTER key is pressed or the buffer area provided is filled to the maximum length given. The characters read in are stored starting at the address supplied, followed by two zero bytes. The space allowed should be sufficient to account for the extra two characters.

FENCE addr
Leaves on the stack the address of the memory cell in which is stored the limit of the protected dictionary. The FORGET word will not operate on words whose addresses are less than the contents of this cell.

addr count n **FILL**
Fills the specified area of RAM memory with the 8-bit data value given. Each byte receives the same value. This word will work on main memory RAM only. It cannot write to VDP RAM.

FIRST addr
Leaves on the stack the address at which the screen storage buffers start.

FLD addr
Leaves on the stack the address of the word in which the output field length is stored.

FORGET NAME
Tells FORTH to forget all words defined since the latest definition of the NAME that follows. Words stored in the dictionary under the fence cannot be forgotten.

FORTH
FORTH is the main vocabualry. Using this word makes FORTH the context vocabulary, that is, the vocabulary used first to get the definitions of words.

n **GPLLNK** n2
Calls the TI 99/4A system routine GPLLNK. The stack must contain the code value to be passed to GPLLNK. Refer to the Editor/Assembler manual for the routines and code values. The GPLLNK status return byte is left on the stack.

HELP
Lists all words currently accessible. The listing can be stopped by pressing the CLEAR key (FCTN 4).

HERE addr
Leaves on the stack the contents of the DP word, that is, the address of the next available word in the dictionary space.

HEX
Sets the number base to hexadecimal (base 16) for all subsequent conversions of numbers, both for input and for output.

HI addr
Leaves on the stack the address of the top of the RAM space.

HLD addr
Leaves on the stack the address of the memory cell that stores the address of the output area during formatted numeric output.

char **HOLD**
Inserts the ASCII character from the stack into the formatted output area. This word is used between <# and #> to add special characters to formatted output.

I n
Leaves on the stack the value at the top of the return stack. In most cases this word is used to get the index value of the innermost do loop. Note that this word will not get that index value if the do loop is defined in a different word.

addr **ID.**
Prints the name of the word whose name field address is on the stack.

flag **IF**
Used only in colon definitions. In the sequence ... test IF words ENDIF ... the words will be executed if the test yields a true (non-zero) value. In the sequence ... test IF words1 ELSE words2 ENDIF ... if the test is true the words between the IF and the ELSE will be executed. Otherwise the words between the ELSE and the ENDIF will be executed.

IMMEDIATE
Marks the latest word defined as a word to be executed immediately when in colon definition mode. These words are used to implement control structures such as IF and BEGIN.

IN addr
Leaves on the stack the address of the memory cell in which is stored the next input character offset in the input buffer area.

n2 n1 **INDEX**
Lists the first lines of all screens from n2 to n1. (n2 should be less than or equal to n1). The listing can be stopped by pressing the CLEAR key (FCTN 4).

INPUT-LINE addr
Leaves on the stack the address of the memory cell in which is stored the line to be used for input on the video monitor. The value at this address may be altered if it is determined that the last line on the monitor is too hard to read. Values stored at this address should be from 0 to 23.

INTERPRET
Internal to FORTH. Used in interpreting the keyboard input.

J n
Leaves on the stack the third value on the return stack. This word is normally used to retrieve the index value of the outer of a pair of nested do loops. Note that in a word called inside the nested loops the index value cannot be accessed via this word.

KEY char
Reads a keypress from the user and leaves the ASCII value of the key on the stack. When this word is called the cursor appears on the screen and a key must be entered. Compare with the ?KEYBOARD word.

LATEST addr
Leaves on the stack the code field address of the latest word defined in the current vocabulary.

LEAVE
Adjusts the index value of the innermost do loop to make the index value equal to the limit value. This ensures that the loop will exit when the LOOP or +LOOP word is next reached. Should only be used inside DO loops in colon definitions.

addr **LFA** addr2
Takes the parameter field address of a word (addr) and converts it to the link field address (addr2).

LIMIT addr
Leaves on the stack the address of the end of the I/O buffer space.

n **LIST**
Lists screen number n. The listing can be stopped by pressing the CLEAR key (FCTN 4).

LIT n
Internal to FORTH. Puts on the stack the in-line literal value following in the code.

n **LITERAL** (n)
Internal to FORTH. Used in converting literal input. If the system is compiling the number is compiled as a literal into the code. Otherwise the stack is unaffected.

LO addr
Leaves on the stack the address of the first word free after the I/O buffer space. The RAM space between LO and HI is available to the user.

n **LOAD**
Loads the screen specified. If that screen contains the LOAD word or --> word itself, additional screens may be loaded.

LOOP
Used in the ... n2 n1 DO words LOOP ... construction. Refer to the DO word for futher information.

n2 n1 **M*** d
Multiplies the top two 16-bit signed numbers on the stack and leaves a 32-bit product.

d n **M/** d2
Divides the 16-bit signed value on the top of the stack into the 32-bit signed value underneath it on the stack. The result is a 32-bit quotient.

d n **M/MOD** n2 d2
Divides the 16-bit signed value on the top of the stack into the 32-bit value underneath it. Left on the stack are the 32-bit quotient (left on top) and the 16-bit remainder. The sign of the remainder is the same as the sign of the dividend (d).

n2 n1 **MAX** n3
Compares the two numbers on top of the stack and leaves the largest of the two.

n **MESSAGE**
Produces the error message whose number is on top of the stack. If the warning flag is zero, the message number itself will be printed out. Otherwise, assuming the message number is not zero, the message will be read from the message screens (4 and 5) and printed.

n2 n1 **MIN** n3
Compares the top two numbers on the stack and leaves the smallest.

n **MINUS** -n
Multiplies the number on the top of the stack by -1.

n2 n1 **MOD** n3
Leaves on the stack the remainder of a division of the top number, n1, into the second number n2. The remainder will have the same sign as n2.

addr **NFA** addr2
Given a parameter field address of a defined word on the top of the stack, finds the name field address.

addr **NUMBER** d
Converts the input number, stored in RAM memory starting at the byte after the given address and ended with a blank, from character form to binary form. The current number base is used. The 32-bit result is left on the stack. The DPL word is also updated. If the string at address is not a proper number an error message is issued.

OFFSET addr
Leaves on the stack the address of the memory cell used to store the screen offset. The contents of this address are added to the screen number of such words as LIST, LOAD, INDEX and TRIAD to force their residence from one device to another. An offset of zero means that DSK1 is the prime file source, an offset of 2000 means that DSK2 is the prime file source, 4000 means that DSK3 is the prime source and 6000 means that CS1 is the prime source.

n2 n1 **OR** n3
Performs the logical or operation on the two 16-bit values on top of the stack, leaving the result. Each bit in n3 is set independently, with the bit being set to one if either of the corresponding bits of n2 or n1 is one.

OUT addr
Leaves on the stack the address of the memory cell in which is stored a pointer into the output buffer area.

n2 n1 **OVER** n2 n1 n2
Makes a copy of the second item on the stack and puts it on top of the stack.

PAB addr
Leaves on the stack the address of the memory cell in which is stored the address in VDP RAM of the peripheral access block used internally by Forth.

PAD addr
Leaves on the stack the address of an area that is used for temporary storage of strings during formatted numeric output.

addr **PFA** addr2
Given the name field address of a defined word on the top of the stack, finds the parameter field address of the word.

PREV addr
Leaves on the stack the address of a memory cell that contains the address of the I/O buffer most recently used.

QUERY
Internal to FORTH. Receives input from the keyboard for interpreting, putting it into the terminal input buffer.

QUIT
Internal to FORTH. Drives the interpreter. Can be used to exit applications in mid-stream, returning control to the user at the keyboard.

R n
Copies the value that is on top of the return stack and puts it onto the data stack.

R# addr
Leaves on the stack the address of the memory cell in which is stored the edit cursor position on some FORTH systems. This cell is not used by the Wycove Forth editor.

addr n flag **R/W**
Reads or writes the given block (n) to or from the specified RAM address. This word is internal to the disk system of FORTH. The flag should be zero to write out block number n and non-zero to read the block.

R0 addr
Leaves on the stack the address of the memory cell in which is stored the address of the bottom of the return stack. Since the return stack grows downward in memory, this is the highest address it uses.

R> n
Removes the top value from the return stack and puts it onto the data stack. This word should only be used in colon definitions and should be paired with the word >R.

addr vdpaddr count **RAM->VDP**
Copies a block of memory locations from main memory to Video Display Processor RAM. The top item on the stack is the number of bytes to move. The second item is the VDP address to move the data to and the third item on the stack is the RAM or ROM address from which the data is to be taken.

vdpaddr addr count n **READLINE** flag
Internal to FORTH. Used in reading disk blocks. The VDP address given is the peripheral access block address. Also specified are the RAM address to receive the line, the number of bytes desired and n, the DSRLNK code value and file flags. Left on the stack is the status flag from DSRLNK.

REPEAT
Used only in colon definitions in the sequence ... BEGIN words1 test WHILE words2 REPEAT ... in which the two sets of words are executed repeatedly until the test before the WHILE word leaves a false value or zero on the stack. At that point the words between the WHILE and the REPEAT are skipped and words after the REPEAT are executed.

n3 n2 n1 **ROT** n2 n1 n3
Rotates the top three values on the stack, pushing the first and second values down one and moving the original third item on the stack to the top.

RP!
Stores the return pointer so that it points to the bottom of the return stack. Used internally.

n **S->D** d
Takes the signed 16-bit value on top of the stack and converts it into a signed 32-bit value. If the value to be converted is an unsigned 16-bit number it can be converted to 32-bit format simply by putting a zero on the stack.

S.
Prints out the contents of the data or parameter stack. The values are printed as 16-bit signed numbers in the current base. The first number printed is at the bottom of the stack, the last is at the top.

S0 addr
Leaves on the stack the address of the memory cell in which is stored the address of the bottom of the data or parmeter stack. Since this stack grows downward in memory, this is actually the highest address it uses.

addr count n2 n1 **SAVE-BLOCK**
This word is used by SAVE-SYSTEM to write a block of memory to the selected device. The blocks to be written can be no longer than >1FFA to be used with the load/save option of the file management system. The block specified by addr and count is written out. The top item on the stack, n1, is a file index starting at zero for the first block and increasing by one for each block. The other item on the stack, n2, is a flag of zero for the last block and -1 for all others.

SAVE-BUFFERS
Instructs FORTH to write out for permanent storage all screens in memory that have been updated. This word should be used to retain changes to programming screens before turning the computer off.

SAVE-SYSTEM
Instructs FORTH to create a new load copy of itself, containing all words currently defined, and to write that copy to the currently selected device. To protect the words in the new copy, the value stored in the FENCE cell can be updated.

SCR addr
In some Forth systems leaves on the stack the address of the memory cell used to store the number of the last screen accessed. Not used in Wycove Forth.

char **SCREEN-EMIT**
This word prints the ASCII character given to it onto the screen. This word is normally called indirectly through the word EMIT, which can then be altered to allow output to be redirected to a printer. Refer to EMIT for further information.

SCREEN-WIDTH addr
Leaves on the stack the address of the memory cell is which is stored the width of the monitor screen. The value in this cell should be 32 in graphics mode and 40 if the display is in text mode. The value is needed to allow proper scrolling of the screen.

vdpaddr n3 n2 n1 **SETPAB**
Internal to FORTH. Used to set certain parts of the peripheral access block at vdpaddr for use with system I/O utilities. Values supplied are the record length cell n3, the code and flag cell n2 and the record number cell n1.

n ud **SIGN** ud
Used in formatted numeric output (i.e. with the <# ... #> sequence). When this word is encountered it inserts the sign into the formatted numeric output. When this word is included in the output sequence the 32-bit value being converted on the stack must be preceded by a 16-bit value from which the sign is taken. If the value is positive no action results. If it is negative a minus sign is inserted.

SMUDGE
Toggles a bit in the latest word defined. Used internally to make improperly defined words inaccessible. To delete a badly defined word SMUDGE can be used before FORGET.

SP!
Resets the stack pointer to the bottom of the data stack, discarding all values. This word is used internally.

SP@ addr
Leaves on the stack the address of the top of the stack prior to execution of this word.

SPACE
Prints out a single blank space.

n **SPACES**
Prints out the number of blank spaces given on top of the stack.

STATE addr
Leaves on the stack the address of the memory cell in which is stored the current state of the FORTH machine. Normal values stored in this cell are zero for interpreting and 192 (hexadecimal C0) for compiling.

n2 n1 **SWAP** n1 n2
Swaps the top two items on the stack.

THEN
Used only in colon definitions in the sequences ... test IF words THEN ... or ... test IF words1 ELSE words2 THEN ... to provide conditional execution of code. This word is equivalent to the word ENDIF. Please refer to the IF word for more details.

TIB addr
Leaves on the stack the address of the memory cell used to store the address of the terminal input buffer.

addr n **TOGGLE**
Performs an exclusive or operation on the byte located in main RAM memory at the address specified. The bottom 8 bits of the data value on top of the stack are used for the exclusive or operation. Compare with the XOR word, which acts on 16-bit values.

addr n **TRAVERSE** addr2
Steps through the string located at addr in the direction given by n (usually 1 or -1). When a byte is reached whose top bit is set that address is left on the stack.

n **TRIAD**
Converts n to the multiple of three closest to n but not greater. Then the three screens starting at that number are printed out. The printing can be stopped by pressing the CLEAR key (FCTN 4).

addr count **TYPE**
Types out the string located at the address given and of the length specified by the top value on the stack.

u2 u1 **U*** ud
Multiplies the two 16-bit unsigned numbers on the top of the stack, creating a 32-bit product.

u **U.**
Prints out the unsigned 16-bit number from the top of the stack. The current number base is used in formatting the number.

ud u **U/MOD** u3 u2
Divides the unsigned 32-bit number on the stack by the unsigned 16-bit number on top of the stack. Left on the stack is the 16-bit remainder, u3 and the 16-bit result u2. This word works only if the value u is greater than the top half of the double number ud and should be used with extreme care.

flag **UNTIL**
Used only in colon definitions after the word BEGIN. Please refer to BEGIN for futher details.

UPDATE
Marks the screen most recently used as having been updated. This means that it will be written out to disk or cassette before its buffer space is reused, or whenever SAVE-BUFFERS is used.

USE addr
Leaves on the stack the address of the memory cell used to store the address of the I/O buffer next to be used (once FORTH is finished with the PREV buffer).

n **USER** NAME
Defines a system variable, giving it an initial value that is the distance from the bottom of the user variable table. When the name is used subsequently the proper address is returned.

n **VARIABLE** NAME
Defines a 16-bit variable whose name follows. The initial value of the variable must be on the stack when it is defined. Whenever the NAME is used subsequently the address of the variable will be put on the stack. Then the words ! and @ can be used to store into it and to read from it.

vdpaddr addr count **VDP->RAM**
Writes the block of memory from the specified Video Display Processor address to the given main memory RAM address. The byte count is the top item on the stack.

VOC-LINK addr
Leaves on the stack the address of the memory cell in which is stored the pointer to the first vocabulary in the chain of vocabularies.

VOCABULARY NAME
Starts definition of a new vocabulary, that is a new set of words that can redefine other words without conflicting. The NAME given is the name of the new vocabulary. Whenever that name is called subsequently the new vocabulary will become the context vocabulary, which means it will be searched first for the definitions of any words encountered in the input stream. To start defining words in this new vocabulary the word DEFINITIONS must be used.

WARNING addr
Leaves on the stack the address of the memory cell used to store the value of the error message warning option. When the value in this cell is zero, error messages are issued simply by giving the error number. When the value is one the error message is read from disk for a more comprehensible message. When the value is negative any error will cause a system reset, using the (ABORT) word.

flag **WHILE**
Used only in colon definitions in the sequence ... BEGIN words1 test WHILE words2 REPEAT ..., for more information refer to the BEGIN word.

WIDTH addr
Leaves on the stack the address of the memory cell used to store the maximum width of any defined word.

char **WORD**
Internal to FORTH. Processes the input stream, finding the next word delimited by the ASCII character value on the stack. Copies the word into the dictionary space starting at the value of the HERE word, preceding it with a byte count.

vdpaddr addr count n **WRITELINE** n2
Internal to the FORTH I/O system. Writes out a single line to disk using the VDP peripheral access block given. The line is in the RAM address specified, with the length given by count. The value n is the code and file flags word to be used for DSRLNK. Left on the stack is the status returned by DSRLNK.

n **XMLLNK** n2
Calls the system routine XMLLNK using the stack value as the code value for XMLLNK (refer to the Editor/Assembler manual for valid code values). The status value returned by XMLLNK is left on the stack. The code value to be used for the CIF (convert integer to floating point) function is >2300 for the Editor/Assembler module, >7200 for the Mini Memory module and >8000 for the Extended Basic module. Otherwise the values to use for Extended Basic, Editor/Assembler and Mini Memory are identical.

n2 n1 **XOR** n3
Performs the exclusive or operation on the two 16-bit values on the top of the stack. That is, in the result value n3 a particular bit is set if the corresponding bit positions of n1 and n2 are different (one is set to one and the other is zero).

[
Leaves compile mode. This word can be used to issue executable words in the middle of a colon definition. It is normally used to change number bases or make calculations. To restart compile mode use the] word.

[COMPILE] NAME
Compiles the code field address of the word that follows it. Used only in colon definitions and mainly to access internal system words.

]
Enters compile mode. This word is normally used after the [word to continue definition of a new word after some intervening executable words.