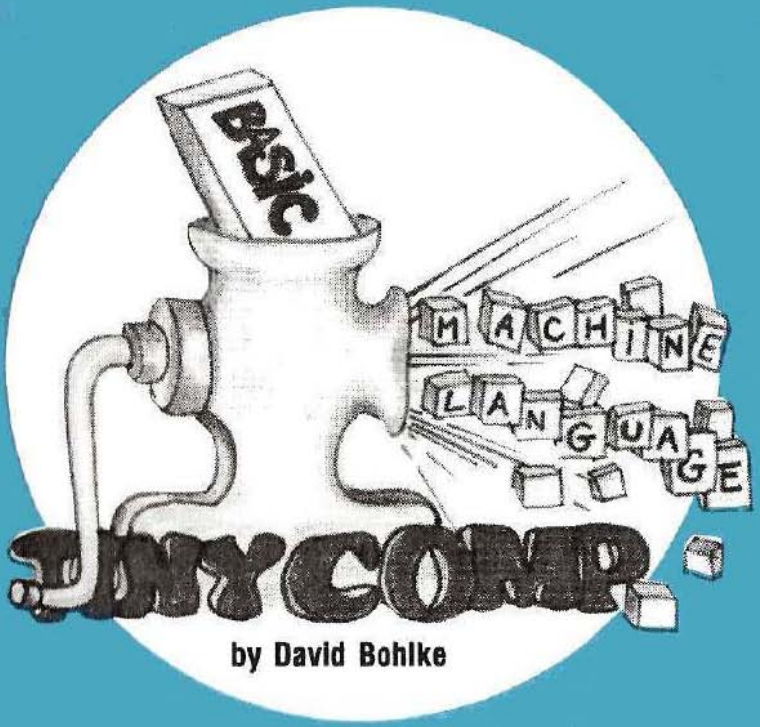

by David Bohlke

RAMWARE™

TINY COMP 3.13

by David Bohlke

TINY COMP is a Basic compiler program written in Level II BASIC for a 16K TRS 80. The compiler program (line numbers 800-5210) occupys less than 5000 bytes of memory; leaving over 6500 free bytes for the source compiled object (Z80 machine) code. (Disk: 2600 for source ;16 or 32K for object code). The source program, written from line numbers 1-799, must use only the statements which TINY COMP can compile. The object code will be POKEd into high memory beginning at location 26000 (disk 32669). Either the BASIC source program or the machine language object code can be RUN under user control. A source code program using all the TINY COMP statements (line numbers under 800) is included on the TINY COMP tape.

In this TINY COMP manual you will find operating instructions, statement definitions, operating system particulars, a line listing description, and several additional sample programs. It is intended that these will aid you in experimenting with the BASIC and machine code.

OPERATING INSTRUCTIONS:

MEMSIZE? 26000 on powerup (32669 and 0 files on disk)

CLOAD program

ENTER program to be compiled, using the TINY COMP statement set and line numbers 1-799. A sample program using all the TINY COMP commands is provided on the tape.

Type RUN to execute the source program to check for proper execution before you compile it. The object code should

provide the same results, with the exception of INKEY$ as explained later. The source program may be edited as under Level II BASIC.

Type RUN1000 to compile the source program. The decimal line number, POKE address, and each machine language code will be displayed for each line compiled. After the compiler is finished, type ENTER when the prompt appears to execute the object code.

Type BREAK to return to the edit/compile mode. If the object code has an endless loop, you'll have to press the RESET button.

An LNF (line not found) error message will appear during the compile mode if the program cannot find a branch location (for GOTO etc.). An ERROR LINE # message will appear if the compiler does not recognize a statement.

After the source program is compiled and executed, it may be run additional times by typing RUN1300.

The source code and compiler may be CSAVED for use at a later time. Make note of the beginning POKE address and the last POKE address to save the object file under DOS, ESF, etc., if you don't want to re-compile it.

OPERATING SYSTEM

Variables (scratch, A-Z) are stored in memory from 32700-32753. VT points to the beginning of the variable table, and V1 is the offset.

L1(L),L2(L) are the current maximum (200) number of lines allowed in the source program. This may be adjusted, as there should be sufficient free memory for over 350 single statement lines.

All variables are set equal to zero when RUN is issued; Variables have the integer range — 32768 to 32767.

STATEMENT SET

A,B,C . . . Z in all statements represent the legal TINY COMP variable names. nnn in the statements is an allowable line number (1-799); and ccc is a positive (or zero) integer constant (less than 32767).

GOTO nnn
GOSUB nnn . . . RETURN
END is required
REM not compiled
A = RND(B); RND(0) or RND (1) will return a zero or a one, respectively.
LET A = ccc The LET is always optional.
LET A = B
LET A = B + C
LET A = B – C Variables cannot be assigned as negative using A = ccc; but they can be stored and printed as negatives. For example, C = 100 B = 70 and A = B – C will result in A containing a negative 30.

TAPE VERSION ONLY:

(LET A = B*C LET A = B/C)

IF A = B THEN nnn
IF A<B THEN nnn

IF A>B THEN nnn This statement will branch on a greater than or equal condition.

A$ = INKEY$ This statement will not return a $ character as in Level II. Rather, the ASCII value of the key pressed will be stored in the integer variable A. To simulate this statement when you wish to test RUN the source program, enter a routine similar to the one below and the source program will execute in the same manner as the compiled machine code. The compiler will ignore the second statement in line number 667 of the example:

```
666 A =0
667 A$ = INKEY$:IFA$"THEN668ELSEA = ASC(A$)
668 continue
```

CLS

PRINT@A,B;
PRINT@A, "STRING"; The maximum length of STRING is sixtyfour. Both "quotes" are required.

TAPE VERSION ONLY:

POKE A,B
A = PEEK(B)

PEEK and POKE can be used to build DATA tables and ARRAYS in high memory, above the object code. They may also be used for graphics; and PEEK can be used to scan the keyboard for input.

800	POKEs machine code in 26000+
805	PEEKs BASIC source program
810	checks for variable type
815	prints ERROR message
820	finds integer value of a constant
830	checks for constant type
850	acknowledgement print
860-870	LOAD variable types (V1,V2)
890	checks for legal line number
900-990	Compiler codes to be POKEd
900	LD HL(nnn)
905	LD D nn, LD E nn
920	EXC HL, DE
925	LD(nnn),HL
930	JP nn
940	CALL
990	computes (nnnn) for 900,925
1000-1015	sets parameters
1020-1045	saves decimal and POKE line numbers
1050	gets statement code
1090-1200	branch on statement codes
1300-1333	executes machine language program
1800-1860	POKEs for print CALL routines

The remaining routines compile the selected statements:

2000-2300	LET
2500-2530	PRINT@A,CHR$(C)
2600-2640	PRINT@A,B
2700-2770	PRINT@A, 'MESSAGE'
3000-3050	IF . . . THEN
3500	GOTO
3700-3800	GOSUB . . . RETURN
4000-4020	INKEY$
4500-4530	RND
5000	CLS

TAPE VERSION ONLY:

5100	PEEK
5200	POKE

SAMPLE PROGRAMS:

This program illustrates the speed of the compiled machine code (crude as it may be). Enter the source below, then type RUN to get an idea of the BASIC Interpreter speed. Then execute RUN1000 for a blistering surprise!

```
10 A=33:B=91:E=1
20 C=1023:D=0
30 PRINT@D,CHR$(A);
40 D=D+E
50 IF D > C THEN 70
60 GOTO 30
70 A=A+E
80 GOTO 20
799 END
```

In this example, INKEY$ and the arrow keys are used to move a cursor. Don't go off the top or the bottom of the screen!

```
20 P=350:Y=64:R=3:D=191
30 PRINT@P,CHR$(D);
35 A=0
40 A$=INKEY$:IFA$=""THEN42ELSEA=ASC(A$)
42 F=8:G=9:H=10:I=91
50 IF A=F THEN 60
52 IF A=G THEN 70
54 IF A=H THEN 80
56 IF A=I THEN 90
58 GOTO 30
60 P=P-R
62 GOTO 30
70 P=P+R
72 GOTO 30
80 P=P+Y
82 GOTO 30
90 P=P-Y
92 GOTO 30
799 END
```

DISK USERS ONLY:

For multiplying unsigned variables, try the routine at 700. Division can also be performed by successive subtraction. Remember to stay within the integer range.

```
5 CLS
10 N=150:A=0:B=10:C=17
20 X=RND(N)
22 Y=RND(N)
30 PRINT@A,X
32 PRINT@B,"TIMES"
34 PRINT@C,Y
40 GOSUB700
50 A=128:B=140:C=320
52 PRINT@A,"PRODUCT IS"
54 PRINT@B,P
56 PRINT@C,"PRESS ANY KEY FOR ANOTHER EXAMPLE ?"
60 S=0:B=0
62 S$=INKEY$:IFS$=""THEN64ELSES=ASC(S$)
64 IF S=B THEN 60
66 GOTO 5
700 Z=0:P=0:A=1
702 P=P+X
704 Z=Z+A
706 IF Z=Y THEN 710
708 GOTO 702
710 RETURN
```

SPLAT

The sample source code (line numbers 10-799) included on disk with TINY COMP is a game called SPLAT. An asterisk (*) will appear on the screen, and you must use four arrow keys to move the cursor block and run over the target. Pressing any other than an arrow key will halt the cursor. To illustrate the graphics capabilities, there are ten cursor speeds to choose from. The object code for SPLAT requires about 1400 compiled bytes; with over 4000 free bytes remaining. There is also more than 4500 free bytes left for source code.

SPLAT Line Listing Summary

10-30	Initialization
110-250	Main game loop
110	Print splatter, delay
120-124	Check for hit
130	Loop to continue
200-250	Increment points, reset screen
600-602	Set new target
650-654	White screen
700-737	Adjust cursor
740-746	Delay loop, check for end of game
750-752	Print splatter

NOTE TO DISK USERS:

The author of this remarkable program is one of the few remaining tape users. For that reason, the tape version of TINY COMP actually has more features than the disk version! Specifically, multiplication, division, PEEK and POKE. In addition, the sample source program supplied with the disk version (SPLAT) is different from the program which comes in the tape version (3-D TIC-TAC-TOE).

So that you will have access to the above enhancements to TINY COMP, the tape version is supplied along with the disk version on the accompanying diskette. To use the tape version you must load the tape version (filename: TINYCOMP/TAP), CSAVE it onto tape, and CLOAD it under Level II (hold down 'BREAK' while you press 'RESET').

TINY COMP SYSTEM CSAVE (for tape version only)

TINY COMP has made it possible for anyone who can write in BASIC to get efficient machine language code from their programs. Although the statement set for TINY COMP is a subset of Level II BASIC, complex programs can be written. One example is the three dimensional TIC TAC TOE game provided with TINY COMP.

Since the compiler is written in BASIC, execution time to compile is relatively slow. The TIC TAC TOE game takes fourteen minutes to compile about 4000 bytes. Thanks to a routine offered by George Blank in the April 1980 issue of PROG/80, it is now possible to save the TINY COMP object code buffer on a SYSTEM-compatible tape. Disc spinners already have this ability with their version of TINY COMP.

The SYSTEM CSAVE program is recorded after TINY COMP on your cassette. This routine could have been incorporated with the TINY COMP compiler program; since it takes over 1000 bytes, it is better to have it as a separate program. The program is fairly well documented. If you would like more information on the specifics, or a

look at the machine code source listing, refer to the April issue of PROG/80.

Directions for CSAVEing and CLOADing the object code buffer are given below. Because the entire object code buffer is recorded, it will always take two minutes to CSAVE and CLOAD the SYSTEM tape. This, I believe, is quite acceptable when compared to the compile time. As a precaution, your BASIC source program should have an endless loop if it is to be CSAVEd on a SYSTEM tape.

OPERATING INSTRUCTIONS

CSAVE

1. Power-up your system, answering MEMORY SIZE?-with 26000. CLOAD the TINY COMP compiler with the source program. Compile the source program by typing RUN1000.
2. Type NEW. This will not affect the object code buffer.
3. CLOAD the TINY COMP SYSTEM CSAVE program, and RUN. First you will need to enter the file name (up to six characters) of your program. Then ready the cassette in the CSAVE mode, and press ENTER on the prompt. When the recorder stops, additional copies may be saved by typing RUN, and following the same procedures.

CLOAD

To execute the SYSTEM tape, ready the cassette in the CLOAD mode and type SYSTEM followed by an ENTER. When the prompt appears (*?), enter the filename used to CSAVE above. When you press ENTER, the object code will be loaded into the machine from the tape. On completion, a second (*?) prompt will appear. Type a slash (/) followed by 26000, press ENTER,and the program will execute.

SPLAT — AN INTRODUCTION TO TINY COMP by David Bohlke

Have you ever wanted to create games or routines with the speed of machine language and the ease of programming in BASIC? This is now possible with the use of the Level II BASIC compiler program TINY COMP. SPLAT is presented in this article as a sample program to illustrate some of the commands which can be converted to machine language by TINY COMP.

SPLAT is very simple in concept. An asterisk will appear on the screen, and the player must use the four arrow keys to move the block cursor and intercept the target. The sequence is timed and scored, and there are ten speed levels from which to select.

All the program lines, once compiled, will execute exactly as in BASIC — except they will run with machine language speed. The only exception is the INKEY$ function, as illustrated in line number 22 of the program listing. Q$=INKEY$ will return the ASC value of the key pressed in the variable Q. It does not return a String variable. The second statement in line 22 (IFQ$=""THEN22ELSEQ=ASC(Q$)) is added so that the program will RUN under BASIC with the same result as the compiled code. This second statement is NOT compiled by TINY COMP.

Since the compiled code executes the same as the BASIC code, it is fairly easy, using BASIC, to write and debug the programs before they are compiled. The compiler, source BASIC code, and the object code buffer all reside in a 16K Level II machine at the same time. There is no need to switch tapes back and forth every time you want to adjust the program listing. The BASIC code for SPLAT is a little over 1000 bytes; and the compiled code is 1142 bytes. In a 16K machine there is sufficient memory to compile a program over 6000 bytes in length.

Although not included in SPLAT, TINY COMP (tape version) can also compile the PEEK and POKE commands (as well as * and /). These can be used to build and access data tables and arrays. Also, the PEEK command can be used for scanning the keyboard for input.

To get the SPLAT program up and running, you must first CLOAD the TINY COMP compiler after answering MEMORY SIZE? with 26000. Next, key in the SPLAT listing as presented with this text. A line listing description is included so you can identify the SPLAT routines. If you would like a comparison of the BASIC speed, you can type RUN to execute SPLAT under Level II BASIC. In addition, when you are writing a program from scratch, you will be able to check and adjust program flow under BASIC execution. Remember, your BASIC source program must use line numbers 1-799, and it MUST contain an END statement.

When you are ready to compile SPLAT, type RUN1000. The compiler will display each line number of the current line being compiled, as well as the current location in the object code buffer, and the decimal equivalents of the compiled object code. TINY COMP can compile only the statements illustrated in the statement set table. Similar statements with the same key words (PRINT, IF, etc) which cannot be compiled will produce an ERROR LINE # message. Lines with other key words, like DEFINT A-Z, will be skipped by the compiler. Compile time is about 1000 bytes every 3-4 minutes.

After your program is compiled, just press ENTER when the prompt appears to execute the machine language code. If your program has a logical end, control will return to BASIC after execution is completed. To run the program additional times, type RUN1300. When the program has an endless loop, as in SPLAT, you will have to press the RESET button to return to BASIC.

The BASIC source program SPLAT, along with TINY COMP, can be CSAVEd for future use. The object code cannot be SAVEd in the same fashion, so it will be necessary to recompile (RUN1000) SPLAT on power-up.

The disk version of TINY COMP functions in a fashion essentially similar to the tape version. Some rearrangement of object code routines was necessary, since source code and compiler will not fit into the 6K left by disk BASIC (in a 16K machine).

Even though the writing of a BASIC compiler began as an experiment for me, I believe it has developed into a useful configuration. SPLAT is presented as a demonstration of the high speed capabilities of the compiled code in games. A more complex source program, 3D TIC TAC TOE, is included with the tape version of the TINY COMP compiler. Whether you believe a BASIC compiler can be useful for gaming, or just experimentation, I hope TINY COMP can be a learning experience for you as it has been for me.

```
 S P L A T   (SOURCE CODE)

10 X=1:Y=3:W=64:V=48:U=896:G=1000:H=0
18 CLS:PRINT@2,"S P L A T     BY  DAVID BOHLKE"
20 PRINT@W,"ENTER SPEED (0-9) ?"
22 Q$=INKEY$:IFQ$=""THEN22ELSEQ=ASC(Q$)
23 K=48:IFQ<KTHEN22
24 K=58:IFQ>KTHEN22
26 K=47:Q=Q-K:K=150:L=0:M=0:O=Q-X
30 L=L+K:M=M+X:IFQ=MTHEN90
32 GOTO30
90 Q=L:GOSUB650:CLS:GOSUB600
100 A=X:B=W
110 T=191:GOSUB750:GOSUB740:IFK=XTHEN790
115 P=821:PRINT@P,"POINTS";
116 P=885:PRINT@P,H;
117 K=501:PRINT@K,"SPLAT";
118 K=245:PRINT@K,"SPEED";:K=310:PRINT@K,O
120 P=A+B:IFP=CTHEN200
122 P=P+X:IFP=CTHEN200
124 P=P+X:IFP=CTHEN200
130 GOTO380
200 H=H+X:GOSUB650:CLS:GOSUB650:CLS:GOSUB600
380 T=128:GOSUB750:GOSUB700
400 GOTO110
600 K=14:D=RND(K):K=44:E=RND(K):K=0:F=0
```

```
602 K=K+X:IFK>DTHEN610
604 F=F+W:GOTO602
610 C=E+F:K=42:PRINT@C,CHR$(K);:RETURN
650 K=0:L=1023:M=191
652 PRINT@K,CHR$(M);
654 K=K+X:IFK<LTHEN652
656 RETURN
700 T=128:GOSUB750
702 R=S:S$=INKEY$:IFS$=""THEN703ELSES=ASC(S$)
703 IFS>XTHEN705
704 S=R
705 J=8:K=9:L=10:M=91
706 IFS=JTHEN720
707 IFS=KTHEN725
708 IFS=LTHEN730
710 IFS=MTHEN735
715 RETURN
720 IFA<YTHEN715
722 A=A-Y:RETURN
725 IFA>VTHEN715
727 A=A+Y:RETURN
730 IFB=UTHEN715
732 B=B+W:RETURN
735 IFB<WTHEN715
737 B=B-W:RETURN
740 K=53:PRINT@K,"TIME";:K=0
742 K=K+X:IFK<QTHEN742
743 G=G-X:K=117:PRINT@K,G;:IFG<XTHEN746
744 RETURN
746 K=1:RETURN
750 P=A+B:PRINT@P,CHR$(T);:P=P+X:PRINT@P,CHR$(T);:P=P+X
752 PRINT@P,CHR$(T);:RETURN
790 K=629:PRINT@K,"<ENTER>";
792 K$=INKEY$:M=13:IFK=MTHEN10
794 GOTO792
799 END
```

```
T I N Y C O M P   (TAPE VERSION)

800 POKEM,P:PRINTP; :M=M+1:RETURN
805 P=PEEK(Q):Q=Q+1:IFP=32THEN805ELSERETURN
810 IFP<650RP>90THEN815ELSERETURN
815 PRINT:PRINT"ERROR LINE #";L1(L-1):END
820 IFP<480RP>57THENRETURN
822 C$=C$+CHR$(P):GOSUB805:GOTO820
830 C$="":GOSUB820:IFC$=""THENC1=-1:RETURN
832 C1=VAL(C$)
834 D1=C1/256:E1=C1-D1*256:RETURN
850 PRINT@0,"TINYCOMP 3.13     DAVID BOHLKE - COGGON,IA":RETURN
860 GOSUB805:GOSUB810:V1=P-64:GOSUB805:RETURN
870 GOSUB805.GOSUB810:V2=P-64:RETURN
890 GOSUB805:GOSUB830:IFC1<10RC1>799THEN815ELSERETURN
900 P=42
902 GOSUB800:GOSUB990:GOSUB800:P=P1:GOSUB800:RETURN
905 P=22:GOSUB800:P=D1:GOSUB800:P=30:GOSUB800:P=E1:GOSUB800:RETU
RN
920 P=235:GOSUB800:RETURN
925 P=34:GOSUB800.GOSUB990.GOSUB800:P=P1:GOSUB800:RETURN
930 P=195
932 GOSUB800.P=E1:GOSUB800:P=D1:GOSUB800:RETURN
940 P=205:GOTO932
990 P=VT+V1*2-INT((VT+V1*2)/256)*256:P1=(VT+V1*2)/256:RETURN
1000 DEFINTA-Z:DIM L1(200),L2(200)
1010 Q=17129:L=1:VT=32700:VS=VT:M=26800:MS=M:CLS
1015 GOSUB850:GOSUB1800
1020 M1=PEEK(Q)+PEEK(Q+1)*256:L1(L)=PEEK(Q+2)+PEEK(Q+3)*256
1030 GOSUB850:IFM>32650PRINT"M =";M;" : OUT OF MEMORY":END
1040 PRINT:PRINT@960,"*";L1(L);M;":"; :L2(L)=M:L=L+1:Q=Q+4
1045 IFL1(L-1)>799THEN1220
1050 GOSUB805
1090 IFP>64ANDP<91THENQ=Q-1:GOSUB2000
1100 IFP=140GOSUB2000ELSEIFP=178GOSUB2500
1110 IFP=141GOSUB3500ELSEIFP=143GOSUB3000
1120 IFP=145GOSUB3700ELSEIFP=146GOSUB3800
1130 IFP=177GOSUB5200ELSEIFP=132GOSUB5000
1140 IFP=128THENP=201:GOSUB800
```

```
1190 IFP=0ORPEEK(Q-1)=0THEN1200
1192 IFPEEK(Q-1)=58THEN1050
1195 GOSUB805:GOTO1190
1200 Q=M1:PRINT:GOTO1020
1220 GOSUB850:PRINT@960,"ADJUSTING JUMP'S . . .";
1250 FORI=MSTOM:IP=PEEK(I)
1252 IFIP=195ORIP=202ORIP=242ORIP=250ORIP=194THEN1253ELSE1290
1253 IFPEEK(I+2)>127THEN1290
1260 DN=PEEK(I+1)+256*PEEK(I+2):DH=0
1262 FORJ=1TOL:IFDN=L1(J)THENDH=L2(J):PRINTL1(J);
1264 NEXT:IFDN>799THEN1290
1266 IFDH=0PRINT" LNF =";DN; :GOTO1290
1270 MB=DH/256:LB=DH-MB*256:POKE(I+1),LB:POKE(I+2),MB
1290 NEXT:PRINT
1300 POKE16526,144:POKE16527,101:GOSUB850:PRINT@960,"";
1302 FORI=32700TO32754:POKEI,0:NEXT
1305 INPUT"<ENTER> TO  =RUN=  MACHINE CODE . . .";A$:CLS
1310 X=USR(0)
1333 GOTO1333
1800 FORI=32673TO32698:READX:POKEI,X:NEXT
1810 DATA 235,213,22,60,30,0,25,209,235,213,34,33,65,205,189,15
1820 DATA 209,126,183,200,18,19,35,195,178,127
1830 FORI=32755TO32764:READX:POKEI,X:NEXT
1832 DATA 235,213,22,60,30,0,25,209,115,201
1850 FORI=32652TO32669:READX:POKEI,X:NEXT:RETURN
1855 DATA 213,22,60,30,0,25,209,235
1860 DATA126,254,34,200,18,19,35,195,148,127
2000 GOSUB860:IFP<>213THEN4000
2010 GOSUB805:IFP=222THEN4500ELSEIFP=229THEN5100
2015 GOSUB830:IFC1=-1THEN2100
2020 GOSUB905:GOSUB920:GOTO2150
2100 V2=P-64:GOSUB805
2110 IFP>204ANDP<209THEN2300
2120 V4=V1:V1=V2:GOSUB900:V1=V4
2150 GOSUB925:RETURN
2300 S=P:V4=V1:V3=V2:GOSUB870
2310 V1=V3:GOSUB900:GOSUB920:V1=V2:GOSUB900
2320 IFS=208THEND1=36ELSED1=11
2330 IFS=208THENE1=144ELSEIFS=207THENE1=242ELSEIFS=206THENE1=199
ELSEE1=210
2340 GOSUB940:IFS=208THEND1=10:E1=127:GOSUB940
```

```
2350 V1=0:VT=16673:GOSUB900:VT=V5:V1=V4:GOTO2150
2500 GOSUB805:GOSUB860:GOSUB805
2510 IFP<>247THEN2600ELSEGOSUB805:GOSUB870
2520 GOSUB900:GOSUB920:V1=V2:GOSUB900
2530 D1=127:E1=243:GOSUB940:RETURN
2600 IFP=34THEN2700
2610 GOSUB810:V2=P-64:GOSUB900:GOSUB920:V1=V2:GOSUB900
2640 D1=127:E1=161:GOSUB940:RETURN
2700 Z=1:Q1=Q
2710 IFPEEK(Q1)=34THEN2740ELSEIFZ>64THEN815
2720 Q1=Q1+1:Z=Z+1:GOTO2710
2740 GOSUB900:D1=(M+9)/256:E1=M+9-D1*256:GOSUB905
2750 E1=140:D1=127:GOSUB940:P=24:GOSUB800:P=Z:GOSUB800
2760 P=PEEK(Q):Q=Q+1:IFP=34GOSUB800:RETURN
2770 GOSUB800:GOTO2760
3000 GOSUB860:SG=P:IFP=36THENP=0:RETURN
3010 GOSUB870:GOSUB805:IFP<>202THEN815
3020 GOSUB890
3030 V4=V1:V1=V2:GOSUB900:V1=V4:GOSUB920:GOSUB900
3035 P=138 GOSUB800:P=237:GOSUB800:P=82:GOSUB800
3040 IFSG=212THENP=242ELSEIFSG=213THENP=202ELSEIFSG=214THENP=250
ELSE815
3050 GOSUB932:RETURN
3500 GOSUB890:GOSUB930:RETURN
3700 GOSUB890
3710 D2=D1:E2=E1:C1=M+8:GOSUB834:GOSUB905:P=213:GOSUB800
3720 D1=D2:E1=E2:GOSUB930:RETURN
3800 P=225:GOSUB800:P=233:GOSUB800:RETURN
4000 IFP<>36THEN815
4010 GOSUB805:GOSUB805:IFP<>201THEN815
4020 D1=0:E1=43:GOSUB940:P=38:GOSUB800:P=0:GOSUB800
4030 P=111:GOSUB800:GOSUB925:RETURN
4500 GOSUB805:GOSUB870:V3=V1
4510 V1=V2:GOSUB900:VT=16673:V1=0:GOSUB925:VT=V5:V1=V3
4520 D1=20:E1=201:GOSUB940:D1=10:E1=127:GOSUB940
4530 GOSUB925:RETURN
5000 D1=1:E1=201:GOSUB940:RETURN
5100 V3=V1:GOSUB805:GOSUB870:V1=V2:GOSUB900:P=126:GOSUB800
5110 V1=V3:P=50:GOSUB902:RETURN
5200 GOSUB860:GOSUB900:GOSUB870:V1=V2:P=58:GOSUB902
5210 P=119:GOSUB800:RETURN
```

THE LAZY MAN'S SHORTCUT TO MACHINE LANGUAGE!

A BASIC **Compiler** in BASIC! Run your source program in BASIC, compile it into FAST Z-80 Code and execute the compiled version — all without reloading. 26 Integer variables, GOTO, GOSUB, END, REM, RND, LET, + * , / , IF,THEN. ELSE,<, = ,> INKEY$, CLS, PRINT @,CHR$, PEEK, POKE. Compiled programs may be saved via TAPEDISK.

Supplied with game program, "3D TIC TAC TOE", which uses all of the TINY COMP Statement set and is ready to compile.

6 SOUTH ST., MILFORD, N.H. 03055 (603) 673-5144