Compiler General Description
8K Version

The Compiler version 1.3 occupies 5.8 K and was originally written on an OSI Superboard with 8K. The program is sized to run on machines with 8K or more and will have sufficient memory on an 8K machine to produce a 2 page object code. The compiler can produce relocatable object code and the USR(X) routine allows linkage of these object codes such that even on an 8k machine large machine language routines can be generdted and used.

Both the object code location and the variable table location are chosen by the user, thus allowing multiple machine language routines to utilize the same or different variable tables. The object code is stand alone. It uses 16 bit arythmetic stored LSB, MSB and uses the ACC for the LSB ant the X register for the MSB. Only positive integers are used but the user can utilize two's complement to create dummy negative integers. The Y register is used as the offset for the location of the variables. The only working locations are the first 10 bytes in variable table. Self modifying code is used for the PEEK, USR, and POKE compilations. During the first pass the line #'s for GOTO and GOSUB are stored as addresses for the JMP and JSR. Later this is replaced by the absolute address using vectors contained in the string variables L$, L2$, and L3$. The arithmetic routines are from William Barden's book 'How to Program Microcomputers', Howard W. Sams & Co., Indianapolis, Indiana, 1977.

The code generated by the compiler is not as efficient as the experienced programmer can write using assembly code, however it is much easier to have the compiler do the dirty work than to dig up that dusty Assembly Language routine and interface it to your other assembly code. The speed has been compared to the interpreter for nested FOR loops and the object code was found to be about 40 times faster than the BASIC Interpreter. At this speed, some game program subroutines may require waiting loops.

When compiling, decimal object code is printed out on the screen, however jumps to subroutines have line #'s in place of absolute adresses until the jump table is used. Also the compilation of POKES, PEEKS, and USR(X) generates self-modifying code. Locations to be modified in this manner are filled with zeros (look for 2 adjacent zeros) until the object code is run. Dimensioned variables will generate more object code and run slower than non-dimensioned variables because of the need to save and retrieve the status register, ACC and x-register when calculating the addresses. They should be used only when their use actually simplifies the program and thereby probably makes up for the difference.

Just a word about errors upon compilation. Most errors are due to the user violating the limited syntax of the Tiny Compiler. My most common errors are generated by the following incorrect code. Take a look at the available commands and see if you can identify the problems.

```
FORI=1TO20:......:NEXT
POKE57089,3
IFA=BTHEN110
```

TINY COMPILER COMMANDS

Legal variables A thru Z (positive integers 0 to 64K) Dimensioned variables A thru Z, each having subscripts 0 -127 Subscripts in dimensioned variables may be a variable or integer. Dimensioned variables may be used anywhere except as a subscript; however, they may not be used on the left side of "=" under multiplication and division. DIMZ(nnn),G(mmm) multiple DIM statements allowed.

```
A = nnn (where 0 <= nnn <= 64K)
A = B
A = B + C,      A = B + nnn      A = B OR C,      A = B OR nnn
A = A + C,      A = A + nnn      A = B AND C,     A = B AND nnn
A = B - C,      A = B - nnn
A = PEEK(B),    A = PEEK(nnn)
POKE A,B        POKE A,nnn (where 0 <= nnn <= 255)
GOSUB nnn,      GOTO nnn
A = D * B,      A = D * nnn      0<=D<=255, 0<=B<=64K, 0<=nnn<=64K
A = B / C,      A = B / nnn,     0<=C<=128, 0<=B<=64K, 0<=nnn<=128
IF A=number THEN GOTO nnn
IF C=number THENGOSUB nnn
```

```
IF D<>number THENGOTO nnn
IF E<>number THENGOSUB nnn
IF A=B THEN GOTO nnn,
IF A=B THEN GOSUB nnn,          RETURN
IF A<>B THEN GOTO nnn,          STOP
IF A<>B THEN GOSUB nnn,         REM
IF A<B THENGOTO nnn,            X=USR(X), Does not pass arguement,
                                          use as call.
IF A<B THEN GOSUB nnn,          END (Only one END statement per
                                program, use to terminate compilation)
FOR I = A TO B,    (up to 9 nested FOR loops)
FOR I = nnn TO B,
FOR I = A TO B STEP nnn  (nnn + or -)
FOR I = PEEK(nnn) TO B STEP mmm,
FOR I = PEEK(C) TO B STEP nnn,
NEXTX (X optional),
```

Object code is relocatable, i.e. the code can be compiled for relocation to other regions of RAM. Object code must be moved before execution in this case.

Multiple statements per line are OK except IF A = .. THEN GOSUB nnn which must be located on the end of a line.

TINY COMPILER
INSTRUCTIONS

1) Load program (5.8K)

2) Type program to be compiled - lines 0 thru 10 (10 must be an END *). The compiler currently limits the source program to 10 lines. Add a DIM statement for the string variables L$, L2$, and L3$ in Line 200 to to increase the number of lines.

3) Run the source program usngthe BASIC interpreter for checkout, Type "RUN".

4) To compile a program type "RUN 200".

5) Beginning of Object code should be about 1K above the top of BASIC to give the compiler room to store data. Leave about 1/2 page,(128 bytes) for the compiler to store strings from the top of memory. The compiler uses 9 string variable bytes for each line compiled and 5 bytes for each GOSUB or GOTO. Thus a typical program with 10 lines and 5 GOSUBS would use 115 bytes of string storage.

6) Variable table - Only used after compilation, 62 locations needed if 26 variables A-Z are used. If dimensioned variables are used add 2 bytes for each subscript (remember 0).

7) If you want to compile the program in one location and later move it to another location then answer "YES" or "Y" to the question "RELOCATE OBJECT CODE". Give decimal address as answer to next question.

8) During compilation P1 is checked for 58 (colon) or 0, if it isn't then the compiler is out of sync and will give an error message. Most errors occur upon compilation because the Tiny Compiler syntax is a subset of BASIC and the programmer forgets and uses the complete BASIC interpreter syntax. Usually the following diagnostics will occur because of this:

   a) ERROR LINE #...

   b) FC ERROR IN 12 (error on NEXT, usually due to incorrect FOR)

9) After compilation type "Control - C" to exit to save the Object code, or hit shift to execute the Object code.

* Note only one END statement allowed per program, use STOP for interior termination.

Should you need more room for object code, you can selectively remove parts of the compiler not needed. Use the following tables to remove macro codes (such as multiply) that are not needed.

Tiny Compiler
General Layout

Line #	Description
12	Poke Object Code
14 - 26	Peek Source Code (Codes addresses for variables)
28 - 40	Setup Integer, error check
44 - 144	Poke Instruction codes
	(122 - 138 Decodes addresses for variables)
200 - 220	Initialization
224 - 270	MAIN LOOP
272 - 286	JUMP Calculations
288 - 296	Run Machine Code & Stop
298 - 600	Macro Codes

Organization of Macro Codes

298 - 326	A = #, check for +, -, *, /, PEEK, and USR(X), perform addition and subtraction
328 - 342	EEK
344 - 366	Multiplication
368 - 390	Division
394 - 434	IF THEN
436 - 442	USR(X)
444 - 448	GOSUB, GOTO
450 - 456	POKE
458 - 466	Self Mod. Code for POKE & PEEK & USR(X)
470 - 481	FOR
482 - 489	NEXT
584 - 600	DIMENSION

Tiny Compiler

Machine Dependent Locations

Statement #

200 Efficient use of memory is made by minimizing the number of lines in the program. The DIM for the string variables may be increased on machines with more memory The current program needs about 1K between the BASIC program and the Object code.

204 Location of the Top of the BASIC program 123,124

206 Location of the beginning of BASIC program 121,122

208 Default Location of Object Code. In the current configuration about 1K is needed between the top of the BASIC program and the bottom of the Object code.
Set default location of object code.

212 Location of the Variable table (62 locations are needed, the first ten of which are working locations, followed by A, B, C, etc., each using two locations in 'word order', LSB,MSB). The dimensioned variables follow in the order they are dimensioned.

288 PEEK(57088) is used to detect shift keys for running object program.

292 LSB and MSB for USR(X) jump to subroutine, 11,12

438 F = 11, LSB for USR(X)

** Note Control C must be enabled so that the user can exit the compiler and save the Object code.

Detailed Description of Tiny Compiler

Line # Description

12 Poke machine code, increment location to be poked, M

14 Call peeking routine, check for non subscripted variables.

16 Set counter for variables (A=1000, B=2000), read (and value within ( ), set F for variable or integer.

18 If variable is within ( ), then add ASC of variable e.g. A(B) would be 1066.

20 If variable is within ( ), then add value and negate. e.g. A(10) would be -1010.

22 Peek BASIC source program, print token, ASC and location peeked (Q), increment Q, do again if character is a blank (32)

24 Check for end of line (0), set flag for end of line

28-30 Check for alpha character

32 Indicate error line # and stop

34 If not integer, then exit reading loop

36 Build a string of integers, peek source code, check for integer; used for assembling addresses, etc.

38 set starting equal to a null, check for integer, if string is null set flag (F) equal to minus one indicating a variable not an integer. 38 is the common entry point. First character of string must be in P upon entry otherwise a double read will occur.

40 Convert a string to number (line number of value of a variable)

42 Calculate MSB and LSB for storage in object code; often used in LDX #, LDA #

44 Get location of variable in variable table, load X register with MSB, and Accumulator with LSB.

48 Load accumulator immediate with LSB, load X register with MSB.

50 LDY#

52 Get location in variable table, store accumulator in LSB, and X register in MSB of variable in table. Commands are absolute indexed by Y.

54 INY

56 DEY

58-64 Sets up sign (S) as plus, minus, OR, AND. Sign is set as code for ADC, th%n checks for SBC(164), AND(168), ORA(169). Pokes command and address. All commands are absolute indexed with Y.

66-68 Performs CLC for ADC or SEC for SBC. Usual,y called before 58 is called.

70 RTS

72 BPL

74 Find variable, Roll variable to left.

76 ROL

78 ASL A

82 PHA

84 PLA

86 PHP

88 PLP

90 DEX

92 Store Accumulator, absolute inde8ed by Y

94 Store Accumulator into variable table address

96 BEQ

98 Poke LSB and MSB

100 Load Accumulator from vaiabe tale address

102 Load Accumulator, absolute indexed by Y

104 BCC

106 BCS

108 BNE

110 LDA#

112 LDX#

114 TXA

116 TAX

118 TAY

120 Compare variable table address to accumulator

122 Set Base addresses for regular variables A-Z

124 Load Y index register with variable table address (A-Z)

126 Get dimensioned variable offset, set base addresses IF V1 is negative then access A(nnn) routine

128 Check for no dimension on subject variable.

130 Beginning of A(variable) routine. Save Status register the ACC and X register on stack.

132 Load Y register with regular variable offset Set base addr for regular variable.

134 LDA with LSB of subscript, multiply by 2, put in Y get X index register from stack

136 Get ACC and status from stack, set base addr for appropriate subscripted variable & return.

138 Entrance for A(nnn), LDY # with subscript, return

140 Pokes command, and LSB and MSB

142 Pokes command, and variable table address

144 Pokes 2 zeros-used for filler on self mod code.

146-148 Check for failure to Dimension variable B.

150-154 Finds location of variable for ROL·(used in * and /)

200 During compilation the line # is stored in L$( ), its location in L2$( ).The location in the object code where a call to a subroutine occurs is stored in L3$( ), and N is the index for L3$( ). Strings are used to decrease storage requirements.

202 title

204 Print Top of BASIC (source program and compiler) so that the user can judge where to place object code and variable table.

206 Source code pointer (Q) is initialized at bottom of BASIC workspace.

208 User chooses locat)on of object code.
Default set up for object code location.

212 Save beginning location of the object code (MM).

214 Initialize pointer for jumps (N), lines (L), FOR (J), relocate vector (R). Default location is set up for variable table.

216 Query user for object code relocation. If not "Y" assume no relocation.

218 If relocation desired, read in address (R) on hag
rent object location.

220 Calculate max location in Variable table, increment later as DIM's are encountered

222 Initialize pointers for base of possible dimensioned variables to XX, ZZ

224 Read location of next BASIC line in RAM (M1), and current line #.

226 Print line # and the location in RAM to next be poked with object code. Increment counter for line (L). Add 4 to location of pointer in source code since line 224 has already read 4 bytes.

228 Save string containing line # and location.
If line # of next line is greater than 10 goto jump table.

230 Reset flag for end of line (C),read a character, do again if end of line (C=2).

232 Go to "A =" routine if P = alphabetic character or if P>999 (A(var)) or if P<0 (A(###))

234 If P = 128 (END) then RTS and GOTO execution phase

236 Set X = 76 (JMP) for GOTO in line 260. If P=REM then skip remainder of line and go to next line of BASIC.

260 Y=-1 if P =129 (FOR)
Y=-2 if P =130 (NEXT)
Y=-3 if P =133 (DIM)
Y=-4 if P =136 (GOTO)
Y=-5 if P =138 (IF)
Y=-6 if P =140 (GOSUB)
Y=-7 if P =141 or P =143 (RETURN, STOP)
Y=-8 if P =150 (POKE)

262 Go to subroutine indicated by ABS(Y)

264 PEEK next character

266 If last character was a colon (58) then cont to peek

268 If last character was not 0 then print it, go to error.

270 Set Q=next line, go read next line header (4 bytes).

272 Check to see if there aren't any jump vectors, if not go to shift detect & run (286).

274-284 Subtract 1 from jump index (N).
Look thru variable table for line L$, poke L2$ + Relocation factor (R) into location L3$.

286 Print # pages of object code, Top location of object code and print message.

288 Detect shift keys.

290 Clear variable table.

292 Print message set up USR(X).

Line # Description

294 - 296 Run Object code, and print variable table and stop.

298 Entry point for all A= commands. Save variable & check for "=".

300 If P=PEEK goto peek routine.

302 If USR, goto USR routine.

304 Check for variable, if variable skip a line.

306 If integer Load ACC & X register # and store in variable V1 location.

308 Check next character for +, -, *, /, AND, OR, if not one of these goto 286 after resetting Q for reread.

310 Save operand (S), save variable or integer V3 following if operand = * then 344.

312 Check operand for division, transfer to division routine.

314 Check flag (F) for variable. If variable skip next line.

316 If integer, load ACC & X immediate with integer and store into working location 0,1.

318 Load second variable V2 into ACC register. Get variable table address, do either CLC or SBC depending on S, then either add or subtract depending on S.

320 Get variable table address for result (V4) & store ACC.

322 - 324 Add most significant bytes as above without CLC or SBC.

326 A = B, load B into ACC & X register. Store into A. RETURN.

328 Entry for PEEK, check for "(".

330 Check for variable or integer if variable skip 2 lines.

332 Load Y with 0, load contents of address (LB & MB) into ACC, Load X with zero.

334 Store into result variable space (V4), read another token ")", return.

336 Read ")" and check for correct syntax.

340 Set X=10 go to self-modifying code, X indicate # of lines below current M that the STA will modify. Load Accumulator with absolute address poked as zeros now which will be loaded by STA located ten locations earlier in code.

342 Store into result location (V4).

344 Entry Point for multiplication prior to 344 A = B * C had been read. In case the multiplier was an integer 38 was called in V4, B in V2 and C in V3. S equals the token * (165). When this routine is called, the multiplier (V2) is loaded in the accumulator, only the lower byte is used and it is put on the stack.

346 F is checked to see whether the multiplier is a variabble or an integer. If it is a variable then the ACC & X register are loaded with the variable and they are stored in locations 0 and 1 of the variable table. Skip next line.

348 If F is an integer then the Accumulator and X register are loaded with LSB and MSB respectively, and they are stored in locations 0 and 1 of the variable table.

350 The Accumulator and X register with zero. Which is loaded into the result location V4. The multiplier is pulled from the stack and the X index register which serves as a counter is loaded with 8.

354 - 366 The following code is poked:

```
 LOOP     CLC
          ROL   V4
          ROL V4 + 1
          ASL   A
          BCC   NOC1 (33)
          PHA
          LDY   V4 location
          LDA   XXZZ, Y
          CLC
          LDY   #0
          ADC   XXZZ, Y
          LDY   V4 location
          STA   XXZZ, Y
          INY
          LDA   XXZZ, Y
          LDY   #1
          ADC   XXZZ, Y
          LDY   V4 location
          INY
          STA   XXZZ, Y
          PLA
 NOC1     DEX
```

```
            BNE    LOOP
```

368 Entry point for Division. Prior to 368 , A = B / C, where A is the quotient, B is the dividend, and C is the divisor, had been read. In case the divisor was an integer, 38 was called and the integer stored in F (MB and LB). A was stored in V4, B in V2, an C in V3. S equals the token "/". When this routine (368) is called, F is checked to see whether the divisor was an integer or variable; if a variable, the ACC and X register are loaded immediate & next line skipped.

Line #	Description
368	If the divisor is an integer then the ACC & X register are loaded immediate with the LSB and MSB respectively.
372	The divisor is used as an 8 bit #, so the LSB is transferred to the X register and the Accumulator is loaded with 0. Then those values are stored at location 0, and 1 in the variable table. Location 0 being the remainder and 1 being the divisor.
374	The dividend is loaded into the Accumulator and X register and stored in the location of the quotient which is then used as a working register. The X register is then loaded with 17 to serve as a counter.
376 - 390	F is used as a working variable to set up the jump to the start of the division routine. F is offset by R in order to be relocatable. V4 is the address of the quotient, xxzz is the location of the bottom of the variable table. The remainder of the routine is :

```
        JMP   START
        LOOP        LDA   xxzz, Y
        SEC
        INY
        SBC xxzz, Y
        BPL NREST
        START CLC
      JP MERGQ
NREST LDY #0
        STA xxzz, Y
        SEC
MERGQ ROL V4
        ROL V4+1
        DEX
```

```
BEQ RTN
ROL RMDR
JMP LOOP
RTN
```

394 Entry point for IF THEN. Peek character, check for alphabetic. Peek character, check for less than (<) or equal (=), if not, indicate error.

396 If "less than" then peek character for greater than (>) if not go to "less than" code at 420.

398 If F=-1 then Variable so continue to 416

400 - 402 If F=number then jump to patch at 416.

404 Go to "THEN" code at 428.

406 Set P = 7, if V$ is an "=" then P = 10. Different branching for "=" and "not equals".

408 TXA, INY, compare MSB.

If V4 is "less than" then BEQ and skip a line.

412 BNE

414 P = 3, go to 446 (GOTO,GOSUB routine).

416 Convert F to MSB+LSB, LDA #$ LB LDX #$ MB

418 Fix PEEK counter and back to normal routine

420 Beginning of "less than" portion of IF THEN macro. Check alphabetic on second variable into ACC and X register.

422 GOTO 428, load ACC and X register compare and BCC 11.

424 BNE 12, compare.

426 BEQ 5, BCS 3, goto 446.

428 Check for "GOTO" or "GOSUB".

430 If not gosub or goto then error

432 - 434 Set GOTO (76) or GOSUB (32).

436 Entry for USR(X), PEEK "(X)", check for ")".

438 Load contents of 11 and 12.

440 - 442 Store contents into absolute address of JSR 00, USR(X) enters self-modifying code at 464.

444 Set x=32 for gosub

446 Entry to "GOSUB", "GOTO", PEEK code, check for legal address.

448 Poke JSR or JMP, store location in memory for absolute address, increment pointer (N), poke line # for temporary address, reset Q by 1, return. Whenever 38 is called one more read will occur than needed, reset upon exit is required.

450 Entry to Poke, check for alphabetic, check for comma, sv address of variable V1.

452 Read integer or variable. Skip a line if a variable.

454 POKE A,nnn. Save integer (V4) call self-modifying

code routine, so as to get the contents of variable V1 and store it in the object code as an address following a STA command.

456 LDA with LSB, STA 00,Y, Reset Q by 1. Return.

458 Poke A,B. Entry to portion of poke macro where value to be poked is contained in variable. GOSUB 462, load A and X with value to be poked, LDY #0, STA 00,Y. Zero's will be replaced by address upon running. RETURN is achieved through line 144.

462 Entrance to self-modifying code for PEEK, POKE,USR(X). Load A and X offset by Y register.

464 - 466 STA; INY; TXA; STA MBLB,Y; Return. MBLB is calculated from X, M and R, where X is relative delta from M to the place in the object code to be modified, M is the machine location in object code, and R is the relocation factor.

470 Entrance to FOR Macro. Add one to nesting counter.

472 Peek variable and save as V7( ), use "A =" subroutine at 298, reduce Q by 1, reread, check for "TO".

474 V6(J) is reentrance LOOP pointer, V%(J) is variable for testing completion of FOR LOOP, Set step (T(J)) equal to 1, set sign V4 as plus.

476 Check for "STEP" (162) if not then Decrement Q, return

478 Check for minus, set V$ equal to 164 if minus.

480 Read step value, if negative step use 2's complement for step.

482 Entrance to NEXT macro. Check for alpha character following NEXT.

483 Load ACC and X register with variable for testing completion, BNE 1

484 TXA, INY, compare MSB.

485 BNE 3, JMP to address plus 26.

486 Load ACC and X register with step, set sign to plus.

487 CLC, ADD subject variable V7(J), store LSB back into subject variable TXA, INY.

488 Add, STA MSB into subject variable, JMP to reentrance location of "FOR".

489 Subtract 1 from counter for nested FOR loops.

594 Entrance for DIM statement. Read character, check for alpha. Save ASC-64 in X as counter (A=1, B=2...)

596 Calculate and store MSB and LSB in XX(X) and ZZ(X) respectively. Read # locations in Dimension.

598 Calculate new top of Table (FM). If end of command or end of line, then return.

600 If not end of Dimension, read character and continue.

```
1 REM TINY COMPILER V1.3 DAVID PITTS JAN 14,1982
2 REM 8K ROM VERSION
3 A=10:DIMA(10)
4 A(10)=1000
5 A(A)=A(10)+1
7 STOP
10 END
12 POKEM,P:M=M+1:RETURN
14 GOSUB22:IFPEEK(Q)<>40ORP<65ORP>90THENRETURN
16 V5=1000j(P-64):GOSUB22:GOSUB22:GOSUB38
18 IFF=-1THENV5=V5+P:GOSUB22:P=V5:RETURN
20 P=-(V5+F):RETURN
22 P=PEEK(Q):Q=Q+1:IFP=32THEN22
24 IFP=0THENC=2
26 RETURN
28 IF(P<65ANDP>0)OR(P>90ANDP<999)THEN32
30 RETURN
32 PRINT:PRINT"ERROR LINE #";L$(L):END
34 YFP<48ORP>57THENRETURN
36 C$=C$+CHR$(P):GOSUB22:GOTO34
38 C$="":GOSUB34:IFC$=""THENF=-1:RETURN
40 F=VAL(C$)
42 MB=INT(F/PG):LB=F-MB*PG:RETURN
44 GOSUB122:GOSUB54:GOSUB100:GOSUB116:GOSUB56:GOSUB100:RETURN
48 GOSUB110:P=LB:GOSUB12:GOSUB112:P=MB:GOSUB12:RETURN
50 P=160:GOSUB12:RETURN
52 GOSUB122:GOSUB94:GOSUB54:GOSUB114:GOSUB94:RETURN
54 P=200:GOSUB12:RETURN
56 P=136:GOSUB12:RETURN
58 P=121:IFS=164THENP=249:GOTO64
60 IFS=168THENP=57:GOTO64
62 IFS=169THENP=25
64 GOSUB142:RETURN
66 P=24:IFS=164THENP=56
68 GOSUB12:RETURN
70 P=96:GOSUB12:RETURN
72 P=16:GOSUB12:RETURN
74 V1=V4:GOSUB150:F=P+ZZ+PG*XX:GOSUB42:GOSUB76:RETURN
76 P=46:GOSUB140:RETURN
78 P=10:GOSUB12:RETURN
82 P=72:GOSUB12:RETURN
84 P=104:GOSUB12:RETURN
86 P=8:GOSUB12:RETURN
88 P=40:GOSUB12:RETURN
90 P=202:GOSUB12:RETURN
92 P=153:GOSUB12:RETURN
94 P=153:GOSUB142:RETURN
96 P=240:GOSUB12:RETURN
98 P=LB:GOSUB12:P=MB:GOSUB12:RETURN
100 P=185:GOSUB142:RETURN
102 P=185:GOSUB12:RETURN
104 P=144:GOSUB12:RETURN
106 P=176:GOSUB12:RETURN
108 P=208:GOSUB12:RETURN
110 P=169:GOSUB12:RETURN
112 P=162:GOSUB12:RETURN
114 P=138:GOSUB12:RETURN
116 P=170:GOSUB12:RETURN
118 P=168:GOSUB12:RETURN
120 P=217:GOSUB142:RETURN
122 XX=XX(0):ZZ=ZZ(0)
124 IFV1>59ANDV1<91THENGOSUB50:P=(V1-60)*2:GOSUB12:RETURN
126 B=INT(ABS(V1)/1000):XX=XX(B):ZZ=ZZ(B):GOSUB146
128 IFV1<0THEN138
130 GOSUB86:GOSUB82:GOSUB114:GOSUB82:GOSUB50:P=(V1-B*1000-60)*2
```

```
132 GOSUB12:XX=XX(0):ZZ=ZZ(0):GOSUB100:GOSUB78:GOSUB116
136 GOSUB84:GOSUB116:GOSUB84:GOSUB88:XX=XX(B):ZZ=ZZ(B):RETURN
138 GOSUB50:P=2*(ABS(V1)-B*1000):GOSUB12:RETURN
140 GOSUB12:P=LB:GOSUB12:P=MB:GOSUB12:RETURN
142 GOSUB12:P=ZZ:GOSUB12:P=XX:GOSUB12:RETURN
144 P=0:GOSUB12:GOSUB12:RETURN
146 IFXX=XX(0)ANDZZ=ZZ(0)THENPRINT"NO DIM FOR";CHR$(B+64):GOTO32
148 RETURN
150 XX=XX(0):ZZ=ZZ(0):IFV1>59ANDV1<91THENP=(V1-60)*2:RETURN
152 B=INT(ABS(V1)/1000):P=2*(ABS(V1)-B*1000):IFV1>999THENP=P-120
154 XX=XX(B):ZZ=ZZ(B):RETURN
200 DIMXX(26),ZZ(26):PG=256
202 PRINT:PRINT:PRINT"     TINY COMPILER 1.3":PRINT:PRINT
204 X=PEEK(123)+PG*PEEK(124)-5:PRINT"TOP OF BASIC PRGM=";X:PRINT
206 Q=PEEK(121)+PG*PEEK(122):L=1:PRINT"FOR DEFAULT ENTER '0'"
208 INPUT"LOC(DEC) OF OBJ CODE(7500 DEFAULT)";M:IFM<XTHENM=7500
212 MM=M:INPUT"LOC OF VARIABLE TABLE (8000 DEFAULT)";VT
214 J=0:N=1:L=0:L3$(1)="0":R=0:IFVT<XTHENVT=8000
216 INPUT"RELOCATE OBJ CODE";C$:IFASC(C$)<>89THEN220
218 INPUT"DEC ADDR";R:R=R-M
220 F=VT:GOSUB42:XX=MB:ZZ=LB:FM=ZZ+PG*XX+62
222 FORX=0TO26:XX(X)=XX:ZZ(X)=ZZ:NEXT
224 M1=PEEK(Q)+PG*PEEK(Q+1):X=PEEK(Q+2)+PEEK(Q+3)*PG
226 PRINT"LINE=";X;"LOC=";M:L=L+1:Q=Q+4
228 L$(L)=STR$(X):L2$(L)=STR$(M):IFX>10THEN272
230 C=0:GOSUB14:IFC=2THEN230
232 IF(P>64ANDP<91)ORP>999ORP<0THENGOSUB298:GOTO266
234 IFP=128THENGOSUB70:GOTO272
236 X=76:IFP=142THENQ=M1:GOTO224
260 Y=(P>128)+(P>129)+(P>132)+(P>135)+(P>137)+(P>139)+(P>140)+(P>149)
262 ONABS(Y)GOSUB470,482,594,446,394,444,70,450
264 GOSUB14
266 X=PEEK(Q-1):IFX=58THEN230
268 IFX<>0THENPRINT"P1=";X:GOTO32
270 Q=M1:PRINT:GOTO224
272 PRINT"JUMP VECTORS":IFVAL(L3$(1))<1THEN286
274 N=N-1:FORY=1TON:C=VAL(L3$(Y)):XX=PEEK(C)+PG*PEEK(C+1):ZZ=0
276 FORX=1TOL:V2=VAL(L2$(X)):V1=VAL(L$(X))
278 IFXX=V1THENZZ=V2+R:PRINT"JUMPTO";V1;"ADDR=";ZZ
280 NEXT:IFZZ=0THENPRINT"NO ADDR FOR ";XX:GOTO284
282 MB=INT(ZZ/PG):LB=ZZ-MB*PG:POKEC,LB:POKEC+1,MB
284 NEXT
286 PRINT(M-MM)/PG;"PAGES":PRINT"TOP=";M
287 PRINT"SHIFT TO RUN, CNTRL C TO EXIT"
288 X=PEEK(57088):IFX<>250ANDX<>252THEN288
290 FORX=VTTOFM:POKEX,0:NEXT
292 PRINT"RUNNING":X=INT(MM/PG):Y=MM-X*PG:POKE12,X:POKE11,Y
294 X=USR(X):FORX=10TO388STEP2:M=VT+X:Y=PEEK(M):Q=PEEK(M+1)
296 PRINTCHR$(X/2+60);Y+PG*Q:NEXT:STOP
298 GOSUB28:V1=P:GOSUB14:IFP<>171THEN32
300 GOSUB14:IFP=187THEN328
302 IFP=176THEN436
304 GOSUB38:IFF=-1THEN308
306 GOSUB48:GOSUB52:RETURN
308 V2=P:V4=V1:GOSUB14:IFP<163ORP>172THENQ=Q-1:GOTO326
310 S=P:GOSUB14:GOSUB38:V3=P:IFS=165THEN344
312 IFS=166THEN368
314 IFF=-1THENV8=P:GOTO318
316 V8=60:GOSUB48:V1=V8:GOSUB52:Q=Q-1
318 V1=V2:GOSUB44:V1=V8:V2=V8:GOSUB122:GOSUB66:GOSUB58
320 V1=V4:GOSUB122:GOSUB94:GOSUB114
322 V1=V2:GOSUB122:GOSUB54:GOSUB58
324 V1=V4:GOSUB122:GOSUB54:GOSUB94:GOSUB14:RETURN
326 V1=V2:GOSUB44:V1=V4:GOSUB52:GOSUB14:RETURN
328 GOSUB14:IFP<>40THEN32
330 GOSUB14:GOSUB38:V4=V1:V1=P:IFF=-1THEN336
```

```
334 P=0:GOSUB12:V1=V4:GOSUB52:GOSUB14:RETURN
336 GOSUB14:IFP<>41THEN32
340 X=10:GOSUB462:GOSUB50:P=0:GOSUB12:GOSUB102:GOSUB144
342 GOSUB112:P=0:GOSUB12:V1=V4:GOSUB52:GOSUB14:RETURN
344 S=163:V1=V2:GOSUB44:GOSUB82
346 IFF=-1THENV1=V3:GOSUB44:V1=60:GOSUB52:GOSUB14:GOTO350
348 GOSUB48:V1=60:GOSUB52
350 F=0:GOSUB42:GOSUB48:V1=V4:GOSUB52:GOSUB84:GOSUB112:P=8:GOSUB12
354 P=24:GOSUB12:GOSUB74:F=F+1:GOSUB42:GOSUB76:GOSUB78:GOSUB104
356 P=33:GOSUB12:GOSUB82:V1=V4:GOSUB122:P=185:GOSUB142:P=24:GOSUB12
358 GOSUB50:P=0:GOSUB12:XX=XX(0):ZZ=ZZ(0):GOSUB58:V1=V4:GOSUB122
362 GOSUB94:GOSUB54:GOSUB100:GOSUB50:P=1:GOSUB12:XX=XX(0):ZZ=ZZ(0)
364 GOSUB58:V1=V4:GOSUB122
366 GOSUB54:GOSUB94:GOSUB84:GOSUB90:GOSUB108:P=210:GOSUB12:RETURN
368 S=164:IFF=-1THENV1=V3:GOSUB44:GOSUB14:GOTO372
370 GOSUB48
372 GOSUB116:GOSUB110:P=0:GOSUB12:V1=60:GOSUB52
374 V1=V2:GOSUB44:V1=V4:GOSUB52:GOSUB112:P=17:GOSUB12
376 F=M+R+15:GOSUB42:P=76:GOSUB140:GOSUB50:P=0:GOSUB12
378 XX=XX(0):ZZ=ZZ(0):GOSUB100:GOSUB66:GOSUB54:GOSUB58:GOSUB72:P=4
382 GOSUB12:P=24:GOSUB12:F=M+R+9:GOSUB42:P=76:GOSUB140:GOSUB50
384 P=0:GOSUB12:XX=XX(0):ZZ=ZZ(0):GOSUB94:GOSUB66:GOSUB74:F=F+1
386 GOSUB42:GOSUB42:GOSUB76:GOSUB90:GOSUB96:P=6:GOSUB12:P=46
390 XX=XX(0):ZZ=ZZ(0):GOSUB142:F=M+R-34:GOSUB42:P=76:GOSUB140:RETURN
394 GOSUB14:GOSUB28:V1=P:GOSUB14:IFP>172ORP<171THEN32
396 V4=P:IFP=172THENGOSUB14:IFP<>170THEN420
398 V2=V1:GOSUB14:V1=P:GOSUB38:IFF<>-1THEN416
400 GOSUB44
402 V1=V2:GOSUB122:GOSUB120:GOSUB108
404 GOSUB428
406 P=7:IFV4=171THENP=10
408 GOSUB12:GOSUB114:GOSUB54:GOSUB120:IFV4=172THENGOSUB96:GOTO414
412 GOSUB108
414 P=3:GOSUB12:GOTO446
416 GOSUB42:P=169:GOSUB12:P=LB:GOSUB12:P=162:GOSUB12
418 P=MB:GOSUB12:Q=Q-1:GOTO402
420 GOSUB28:V2=P:GOSUB122:GOSUB100:GOSUB116:GOSUB54
422 GOSUB428:GOSUB100:V1=V2:GOSUB122:GOSUB54:GOSUB120:GOSUB104:P=11
424 GOSUB12:GOSUB108:P=12:GOSUB12:GOSUB114:GOSUB56:GOSUB120
426 GOSUB96:P=5:GOSUB12:GOSUB106:P=3:GOSUB12:GOTO446
428 GOSUB14:IFP<>160THEN32
430 GOSUB14:IFP<>136ANDP<>140THEN32
432 X=76:IFP=140THENX=32
434 RETURN
436 GOSUB14:GOSUB14:GOSUB14:IFP<>41THEN32
438 GOSUB14:GOSUB50:P=1:GOSUB12:GOSUB102:F=11:GOSUB42:GOSUB98
440 GOSUB116:GOSUB56:GOSUB102:GOSUB98:X=8:GOSUB464:P=32:GOSUB12
442 GOSUB144:RETURN
444 X=32
446 GOSUB14:GOSUB38:IFF<1ORF>10THEN32
448 P=X:GOSUB12:L3$(N)=STR$(M):N=N+1:GOSUB98:Q=Q-1:RETURN
450 GOSUB14:GOSUB28:V1=P:GOSUB14:IFP<>44THEN32
452 GOSUB14:GOSUB38:IFF=-1THEN458
454 V4=LB:X=14:GOSUB462:GOSUB50:P=0:GOSUB12
456 LB=V4:MB=0:GOSUB48:GOSUB92:GOSUB144:Q=Q-1:RETURN
458 X=21:V2=P:GOSUB462:V1=V2:GOSUB44:GOSUB50:P=0:GOSUB12
460 GOSUB92:GOTO144
462 GOSUB44
464 GOSUB50:P=0:GOSUB12:GOSUB92:F=M+X+R:GOSUB42
466 GOSUB98:GOSUB54:GOSUB114:GOSUB92:GOSUB98:RETURN
470 J=J+1
472 GOSUB14:V7(J)=P:GOSUB298:Q=Q-1:GOSUB14:IFP<>157THEN32
474 V6(J)=M-1:GOSUB14:V5(J)=P:GOSUB14:T(J)=1:V4=163
476 IFP<>162THENQ=Q-1:RETURN
478 GOSUB14:IFP=164THENV4=P:GOSUB14
```

```
481 RETURN
482 GOSUB14:IFP<65ORP>90THENQ=Q-1
483 V1=V7(J):GOSUB44:V1=V5(J):GOSUB122:GOSUB120:GOSUB108
484 P=10:GOSUB12:GOSUB114:GOSUB54:GOSUB120
485 GOSUB108:P=3:GOSUB12:P=76:GOSUB12:F=M+26+R:GOSUB42
486 GOSUB98:F=T(J):GOSUB42:GOSUB48:S=163:V1=V7(J)
487 GOSUB122:GOSUB66:GOSUB58:GOSUB94:GOSUB114:GOSUB54
488 GOSUB58:GOSUB94:P=76:GOSUB12:F=V6(J)+1+R:GOSUB42:GOSUB98
489 J=J-1:RETURN
594 GOSUB22:GOSUB28:X=P-64:GOSUB22
596 F=FM:GOSUB42:XX(X)=MB:ZZ(X)=LB:GOSUB22:GOSUB38:GOSUB40
598 FM=2*F+FM:IFPEEK(Q)=58ORPEEK(Q)=0THENRETURN
600 GOSUB22:GOTO594
```

Additions to Tiny Compiler

SAVE statement

SAVE n1,n2,n3...
e.g. SAVE 32,0,253

Description: Inserts the listed values directly into the compiled code.

Line Description

9139-Intercepts a SAVE token (Dec. 148) and goes to line 10000
10000-Translate a number from the line.
10005-If the character was a comma, get another line
10010-If the character is a ending null, or a colon, go to the next line
10015-If none of the above, call an error
10020-Put into the compiled code the LSB of the number. Get another number.
Warning: Multiple Statement lines are not allowed with SAVE statements.

Hexadecimal constants

e.g. $23, $FE, $A9

Description: Any number preceeded by a dollar sign ('$') will be translated as a hexadecimal constant.

Line Description

8045-Intercept dollar signs and call the routine at 9670 to translate to decimal if necessary.
8047-The line normally at 8045 for normal decimal numbers.
9001-H$ is used in the hexadecimal to decimal translation routine.
9670-Set the result (F) to 0.
9673-Get a character and find it's location in H$.
9675-If not found, return.
9680-Multiply the result so far by 16, adding the position-1 from H$. This does the actual hex-to-dec conversion. Then loop back to 9673 for more characters.