The First Book of Tiny BASIC Programs

Copyright (C) 1981 by Tom Pittman

scanned & OCRed by Lee Hart, Richard Peters, Chuck Yakym, others
Last edit Herb Johnson Jan 16 2020

CONTENTS

Chapter 1 -- INTRODUCTION

Tiny Basic was originally conceived by the dragons at People's Computer Company as a simple language to teach kids programming. It turned out to have applications far exceeding this modest object. So much so that I doubt that it was ever used much for the original purpose. The biggest use of Tiny Basic turned out to be its ability to make small computers programmable in something higher than binary absolute.

Yet, for all its lofty intentions, the language remains arcane, and meaningful programs are hard to write in it. Certain kinds of programs turned out to be much easier than was ever thought; small control applications. These programs are satisfied with 16-bit integer arithmetic and limited variable space. Thus variants of Tiny BASIC turned up in several 1-chip CPUs for precisely this purpose. I have included in this book a small program to illustrate this kind of application.

The main purpose of this book, however, is to make available to users of Tiny BASIC a broader selection of programs that are useful in the home and perhaps even (in the case of Tiny Calc) in a small business. The purpose of these programs is not just to entertain and do a job (for programs written in machine language will do that much better), but rather to demonstrate programming techniques so that after you have enjoyed the programs for a while, you can adapt the ideas to your own programs, and thus make your computer truly personal.

When I first conceived of doing a book of Tiny BASIC programs, I came up with maybe a dozen ideas. I figured I would spend a couple of weeks getting each one up. I should know better. Six weeks later the first program was still only half done, so I lowered my sights a little. That is why this is "The First Book of Tiny BASIC Programs". Some of those other ideas are still good, and it may be worth doing another book (perhaps next year? It seems to take about that long to get it all together). I guess it depends on how well this book is received. Many of the earlier users of Tiny BASIC have moved up to bigger and better BASICs or other computers.

1.1 -- For the Record

America is rapidly becoming a consumer-oriented society. To the extent that this protects us from unscrupulous vendors who knowingly sell defective products for a greater profit, I favor the trend. Unfortunately the laws to support this protection do not adequately distinguish between fraudulent negligence and unavoidable mistakes, probably because so many of us use the increasingly sophisticated technology to conceal the differences.

One important law in the United States controls exactly what may and may not be promised under the general term "warranty". Alas! the state of the art in computer software does not support the level of confidence required to make such promises as are required at a price you would be willing to pay for something as esoteric as a Tiny BASIC program. So (listen, all you lawyers), THERE IS NO WARRANTY, EXPRESS OR IMPLIED associated with any program in this book. The purpose of this book is expressly educational. If, after studying it or the programs in it, you know a little more about programming in Tiny BASIC, or about writing big programs such as "Adventure" or "Calc", the book has served its purpose. If you succeed in using Tiny Calc in your business, I'm pleased as peaches, but that is not the purpose of the program.

Now obviously, I do not stay in business by selling a bunch of junk. If you find any problems in any program I sell, I want to hear about it. I intend to correct all errors I know about, and if you are nice enough to tell me about them, I'll surely be nice enough to tell you what the fixes are.

Please notice also that some of the programs in this book bear that little circle (C) that means "Copyright". I spent a lot of time writing and checking out these programs, and I hope I can get some money for that effort. You may make copies for your own use. Especially this means that you can make a copy on tape or some such, so that you do not have to type the whole thing in again. What you are not allowed to do is make a copy for a friend. Now some of these programs are veerrry long, and there is not much incentive for both you and your friend to individually type them into your own computers. So, I will not sue anybody who gives a machine-readable copy of one of these programs to another person who has already bought his own copy of this book. I do not know the exact legal status of doing that, but I consider it to be "fair use".

1.2 -- Configurator

Each program in this book comes with a description of how to use it. One of the important parts of that description is how the peculiarities of your computer affect its operation. Several of the programs use the "peek" and "poke" USR functions, so they need to know where they are. Most of the programs in this book are written for ease of reading, and as such they take much more memory than they could if

crunched to minimum space. It is rumored that most people have lots of memory ("memory is cheap" they say). I don't exactly believe it, so I've put in notes on how to crunch the big programs.

When I started to write these programs the only Tiny BASIC I knew they would run on was what I was selling for the 1802, the 6502, and the 6800. Since that time, an article in Byte announced the availability of a Z8 microcomputer with Tiny BASIC in its ROM. Since I know where that Tiny came from, I can confidently say that most of these programs will run on that computer also, with the indicated modifications. Except for the random number problem (see below), this is something on the order of three lines per program.

The details on how to modify (if needed) the programs to run in different computers is collected in a section for each program with the heading "Configurator".

There are a number of Tiny BASICs around that I did not write. One of the most popular a while back was Li-Chen Wang's Palo Alto Tiny BASIC. Because the source code was published, several variations of this appreared for various computers. Another independent Tiny BASIC came out of National Semiconductor, in the ROM of their one-chip CPU. Since I was not involved in these, I am not in a position to determine whether the programs in this book will run (or can be modified to run) on them. You are on your own.

1.3 -- Program Notes

Since this is an educational book, every program has a complete source listing with program sections labelled with REMarks. Also, every program comes with a section of the documentation labelled "Program Notes". These are a running description of the program, explaining how it works (to the extent it is not obvious from the listing), and sometimes why it was written the way it was. It is intended that these notes should help you understand the program enough to modify it. On other occasions I have been accused of writing cryptic code, so if you are still lost after reading these notes; well, at least I tried.

One thing I discovered while testing the Tiny Adventure game is that big programs run too slow. I came up with a way to speed up Tiny BASIC by about an order of magnitude. The patches to do this are in the Appendix. All the fix does is make correct programs run faster. There is a problem, however. If you have in your program in memory, any lines with only one character (there are no legal Tiny BASIC statements of only one character), then this may cause a GOTO to fail, even though the line number is there. If you are using a single period for debugging (as suggested in the Experimenter's Kit), you will have to go to two periods, or else leave the fix out. The fix takes a few extra bytes, and if your Tiny is in ROM, you are out of luck.

1.4 -- Random Numbers for the Z8

The Z8 Tiny BASIC does not have a built-in random number generator like the other three CPU versions that these programs are able to run on. For games, it is important to have this element of chance, so in the three programs in this book that use the random number function RND, Z8 users will have to substitute something that does about the same thing.

The easiest way to get something approximately random is to use one of the timer counts. T0 is used for the Serial I/0 (your terminal), and is unlikely to be very large (especially with high baud rates). I recommend setting up T1 with a very small prescale, counting continuously through from 99 to 0:

```
LET @243 = 7

LET @242 = 99

LET @241 = 14
```

You only need to set this up once at the beginning of your session (after reset). Then for each occurrence of a random number function call of the form

```
... RND(n) ...
```

you need to replace it with the following calculation:

```
... n*@242/100 ...
```

When the random number routine is called several times in a row without any input statements, the effect is likely to be less random but probably still acceptable.

Chapter 2 -- DATA LOGGER

There have been a flurry of articles in the recent press about using Tiny BASIC in industrial applications. Here is a simple program that demonstrates some of the control application capabilities of Tiny.

The application here is a simple time-keeping function, with the simultaneous logging of data input changes. Clock functions are not part of Tiny BASIC, and accuracy here requires external hardware. In this program I assume that a hardware clock is providing a periodic interrupt once every millisecond (1000 times per second). Other periods are easily accomodated by setting a different constant in variable A (line 540). A machine language routine is needed to respond to the

interrupt and tweak Tiny BASIC. This example is coded for the 6800, but you should be able to read the comments to adapt the idea to your own CPU.

The program is very simple. In a tight loop, the variable U (which is counting the interrupts) is compared to variable T (which tracks it under software control). When the difference exceeds 1000 (or whatever the interrupt rate is), then one second has passed, and the clock time is updated. Also within that tight loop, the input value is compared to its previous state, and if it changes, the new state is printed out with the current time. Printing the state of the data may (I should say probably) take more than one second, but the timer can get as much as 32 seconds ahead of the software clock and still catch up.

I/O on the 6800 and 6502 is memory-mapped; the Z8 I/O is register-mapped, but Tiny sees the registers as memory. For the 1802, you will need to use the built-in I/O USR function at (Cold Start)+38 (i.e. hex 0126): Input takes an argument from 9 to 15, and output is from 1 to 7 (like the N field of the INP and OUT opcodes). Thus for the four different processors, only the setup differs.

2.1 -- Operation

Running this program is not as easy as the other programs in this book. First you must select the input port where the data is to be logged. The hardware clock must be set up to interrupt the CPU (fortunately, Tiny does not care if it is being interrupted, as long as the interrupt service routine operates correctly). An interrupt service routine must be set up in memory to respond to the hardware clock interrupts, and to increment memory locations 00AA-00AB (Tiny BASIC variable U; in the Z8 it is in a different location). Then finally the program can be (loaded and) run.

The first thing the program does is ask for the time of day. After this is entered, the program begins logging data. Here is a sample of the program output:

```
:RUN
TIME (H,M,S)? 3,17,50
00000000 3:17:51
00100000 3:18:10
00000000 3:18:15
00000001 3:18:16
```

2.2 -- Configurator

I'm sorry that I cannot tell you everything you need to run this program on your computer. Every computer has different I/O hardware, and all I can do is give suggestions. The best thing to do is read the program listings.

This program is intended not so much to be useful by itself, as to give you ideas on how you can use Tiny in control applications.

2.3 -- Program Notes

There is not much to say about this program that is not obvious. The initialization is done at the end with a subroutine so that it does not take space at the front (which makes GOTOs slow down).

Since speed is not one of Tiny BASIC's strong points, the main loop of the program is designed to be fast so that the clock will keep good time. Thus the main loop is near the front (so GOTOs go faster), and the frequently used constants are preset into variables (which compute faster than constants).

The binary printout of the data byte is a little obscure. In this case there is only one byte to print, so it is moved to the left byte of variable R (by multiplying it by 256). The sign of variable R is now in the leftmost bit of the datum, so if R is negative then the bit is necessarily a one. Otherwise, it is zero. This is printed (variable B), and then the byte is moved over (left shift by adding R to itself), so that the sign tests the next bit. This is repeated eight times for the whole byte.

If printing a log line has taken more than one second, then the very next time through the time test will trip, and bump the seconds counter. If the clock got several seconds behind, then it will catch up slowly, one second per iteration. Thus it is important that the main loop be much less than one second in total transit. If you have a very slow Tiny (like 1802), you may have to handle seconds in tens.

The variable D retains the last known value of the input data. It is initialized with the one's complement of the initial port value, so that the first time it is compared, all bits are likely to be wrong, forcing it to print.

2.4 -- Interrupt Service

Let's assume that interrupts are vectored somehow to the memory location corresponding to the label INTS below. I assume also (though probably incorrectly for your clock hardware) that clock interrupts are self-clearing. That is, no extra instructions are needed to clear the interrupt after responding to it. The only function of the service routine is therefore to increment memory. Note that the 6800 Interrupt saves the entire CPU state and the Return restores it. Other CPUs may need to do that in the service routine explicitly, as in the 1802 example.

I have used the IEEE proposed standard mnemonics and syntax in the assembly language, but the hex object code is the same 6800 or 1802 code that you know and love.

One thing to watch for is the byte sex of the variable U. In the 6800 and the 1802 (as well as the Z8), the most significant byte is in the low address; in the 6502 it is reversed.

Careful programmers will want to consider what happens in the event of a conflict of access to variable U. One important result of this consideration is that only one routine ever modifies it. I could have written the program so that the interrupt incremented it, and the main routine bumped it back down. Then there would be the problem of what to do if halfway through the statement that bumped it down, an interrupt came to increment it. This is unsafe code.

The question still remains; what if an interrupt increments U while the main program is fetching it to look at it? If the interrupt occurs either before or after the two bytes have been both fetched, then this is no different than if it occurred remotely; there is no problem. It is when the interrupt occurs between the fetching of the two bytes that poses the potential problem. Two cases need to be considered.

Suppose after fetching one byte of the variable, the main program is interrupted and the low byte is incremented, but no carry occurs. Then, if the main program has already fetched that byte, the result is the same as if the interrupt occurred after both fetches. If the main program fetched the high byte first, then the effect is the same as if the interrupt had occurred before both fetches. Again there is no problem.

Now suppose that the interrupt service routine propagates a carry out of the low byte of variable U into the high byte, between the fetches of the two bytes in the main program. This means that the main program will get the upper half of the old value and the lower half of the new value. Both halves have changed. The high byte is one greater, and the low byte went from 255 to 0. So, depending on the order in which the two bytes are fetched, the number seen by the main program could be 255 too large or 255 too small. What are the consequences of this error?

If the number is too small, then the main program will fail the comparison in line 120, and go around for another try, at which point the same fault is unlikely. This is no more serious than if the interrupt came marginally too late to affect the test in this iteration. Again, no problem.

If the number is too large, then there is one chance in three (given that the interrupt occurred in this precise window, and that a carry was propagated) that the test in line 120 will pass too soon. That is, U will look like it has a value of 1023 when it actually is in transition between 767 and 768. The test will pass, and the seconds counter will be bumped approximately one quarter second too soon. Considering that every time data is logged, the resolution is only to the second, and that the internal (software) clock may lag the true time by several seconds after logging the data, I do not consider a lead of 255 milliseconds to be serious. It is important, however, to consider the problem. It is not adequate to note simply that the probability of this happening is one in a million and therefore ignore it.

Another question to consider; what happens if the count in U overflows? The answer; nothing. It will overflow every 65.5 seconds, and so also will the value in T. There is no problem with comparing them, since we are actually not comparing U and T, but the difference between them, which will normally be a small number between zero and maybe a thousand or so (perhaps several thousand, if the software clock falls behind because of logging). It is important to realize, however, that if this difference grows past 32K, it will go negative and the software clock will suddenly lose 65 seconds. You need to consider the capacity of your data port in the light of the interrupt rate of the hardware clock and the baud rate of your terminal.

2.5 -- Program Listing

```
90 GOSUB 500
100 LET R=USR(I,X)
110 IF R<>D GOSUB 300
120 IF U-T<A GOTO P
130 LET T=T+A
140 LET S=S+1
150 IF S<60 GOTO P
160 S=0
170 M=M+1
180 IF M<60 GOTO P
190 M=0
200 H=H+1
210 IF H<24 GOTO P
220 H=0
230 GOTO P
300 LET D=R
320 LET N=0
330 LET R=R*256
340 LET B=0
350 IF R<0 LET B=1
360 PRINT B;
370 LET N=N+1
380 LET R=R+R
390 IF N<8 GOTO 340
410 PRINT " ";H;":";M;":";S
420 RETURN
500 REM INITIALIZE THE OPERATION
510 REM VARIABLE P=LINE NUMBER 100
520 P=100
530 REM VARIABLE A=NUMBER OF INTERRUPTS PER SECOND
540 A=1000
550 REM VARIABLE I=PEEK FOR INPUT
560 I=276
570 REM VARIABLE X=PORT ADDRESS (ASSUME FBC6 HERE)
580 X=-1082
590 REM INITIALIZE & ENABLE INTERRUPTS
620 REM GET INITIAL PORT VALUE (WRONG, SO IT PRINTS)
630 D=-USR(I,X)-1
640 REM SET TIME OF DAY
650 PRINT "TIME (H,M,S)";
660 INPUT H,M,S
670 REM OK, GO DO IT
680 LET T=U
690 RETURN
```

2.6 -- Interrupt Service Routine

```
                ; Interrupt Service Routine -- 6800
                ;
7C 00 AB        INTS:   INC     171     ; increment low byte of U
26 03                   BNZ     $RETN   ; done if no carry out
7C 00 AA                INC     170     ; increment high byte of U
3B              RETN:   RETI            ; return from interrupt



                ; Interrupt Service Routine -- 1802
                ;
42              EXIT:   POP             ; restore state
F6                      SHR             ;   DF
42                      POP             ;   D
70                      EI              ; and exit
22              INTS:   DEC     .2      ; save state
78                      PUSH    .T      ;   T
22                      DEC     .2
73                      PUSH            ;   D
7E                      ROLC
52                      ST      @.2     ;   DF
E0                      LD      #0,.X   ; assume R0 is OK to use
F8 00                   LD      #0
B0                      STHI    .0
F8 AB                   LD      #U+1    ; point R0 to U
A0                      STLO    .0
F8 01                   LD      #1      ; add 1
F4                      ADD             ;   low byte
73                      PUSH
90                      LD      #0
74                      ADDC            ;   high byte
73                      PUSH
E2                      LD      #2,.X
30 00                   BR      EXIT    ; then done.
```

Chapter 3 -- The KINGDOM of EUPHORIA

Imagine that you are the king of the ancient European Kingdom of Euphoria. Each year your loyal ministers report to you on the state of your holdings, and you are responsible for the welfare of your peasants, to give them grain to eat, protect them from the ravages of war, etc. And of course, you want to increase your wealth and influence.

As a game, this program does not need much introduction. It is an ideal pastime to keep the kids busy, since the rules are simple. It is not particularly easy to get rich in this game, however, so you might find it challenging yourself.

The hardest thing about this program is figuring out how much to feed the peasants, plant, or keep. I recommend playing once or twice with no other objective than to find out what the parameters are. Then you will not be so

disappointed when three quarters of your population dies off from starvation. After that you can get down to developing a serious strategy.

Because of the limitations in Tiny BASIC, if you try to hoard or deal in too much grain (over 30,000 bushels), strange things may happen. I tried to check for most hazards, but let's face it: This is Tiny BASIC.

This game has been around in various versions for a long time. This one is neither original with me, nor particularly well-designed. I only translated it into Tiny BASIC. It is, however, interesting to play for a while, and it has been known to keep kids (and some adults!) busy for hours.

3.1 -- Configurator

This program is pure Tiny BASIC (no USR calls), and should run as-is. As listed, it is 5791 bytes long. If you are hard up for space, remove lines 3600-3950, which only give instructions to the player. You can also do the normal crunching things like remove blanks, LET and THEN, shorten PRINT, take out REMarks, etc. Z8 users should note that this game does use the random number generator.

3.2 -- Program Notes

I translated this program from an "Extended BASIC" version of the program I had laying around. I've had that program stashed away for some eight years. I got it from a friend, who got it for his minicomputer, but I don't know where it came from originally. There was no Copyright notice on it or in it. (There is a proverb or something about people who live in glass houses throwing stones.)

The original program was full of arrays and PRINT USING statements, but on analysis, it became evident that the arrays were only used as scalars: except for the initialization FOR loop, every array reference had a constant subscript. Substituting ordinary (Tiny BASIC) variables was natural. As for the PRINT USING statements; well, their only function was to line up the numbers into neat columns. As you see, I abandoned that.

As with all significant Tiny BASIC programs, the first task was to see if the 26 variables will do the job. I always try to get the data organized for a program before starting to write it. In this case, I made a list of all the variables in the original program, cross-referenced by usage, then tentatively assigned Tiny BASIC variables to them. Two or three of the variables were used to hold running sums. I was running out of variables, so I replaced their usage with the actual sums. I also packed two variables into one (the current year, and the seven-year locust cycle). After all that, it turned out that one of the variables I thought I needed, I didn't, so there is a spare (variable I). Most of the data in this program are integers, but a few

of the calculations involved percentages and fractions. I scaled these calculations, so that they could be done in Tiny's integer arithmetic.

3.3 -- Variables

```
A - Acres of land last year
B - Births
C - Crop yield
D - Natural deaths
E - Grain eaten by rats
F - Grain eaten for food
G - Grain last year (bushels)
H - Hire for mercenaries
I
J - Cumulative war and starvation
K - War casualties
L - Land deals (in grain)
M - Looting victims (murders?)
N - Input
O - Looting losses
P - Population last year
Q - Grain planted
R - A random number
S - Starvations
T - Acres traded
U - Acres won or lost
V - Disease victims
W - Probability of war (x100)
X - Crop yield/acre
Y - Current year and locust cycle
Z - Fruits of war
```

Variable Y is packed so that the remainder mod 7 is the year the locusts will hit, and Y/7 is the current year. This is tested in the rather obscure IF expressions in lines 2340 and 3010. Basically,this compares the remainder mod 7 of Y to the remainder mod 7 of Y/7. Remainder is evaluated by the formula

```
Remainder = Value - Value / Modulus * Modulus
```

So the test wants to establish

```
(Y-Y/7*7) =? ((Y/7)-(Y/7)/7*7)
```

A little algebraic manipulation gives

```
Y-Y/7*7 =? Y/7 - Y/49*7
Y =? Y/7 + Y/7 *7 - Y/49*7
Y =? (Y/7)*(1+7) - Y/49*7
```

Note that in this analysis we do not apply certain obvious algebraic rules to the division (such as collapsing it with the multiply), since (mathematically) "A/B" really means "IntegerPart(A/B)".

Please don't ask me to explain the probability calculations for the war -- I only copied the algorithm. The rest of the program is pretty obvious.

3.4 -- Program Listings

```
1000 PRINT "THE KINGDOM OF EUPHORIA"
1010 PRINT
1020 GOSUB 3510
1100 REM PRINT STATE OF THE KINGDOM
1110 PRINT
1120 PRINT "YEAR ";Y/7
1130 PRINT "POPULATION: ";P
1140 IF B>0 THEN PRINT B;" BIRTHS"
1150 IF D>0 THEN PRINT D;" DEATHS BY NATURAL CAUSES"
1160 IF S>0 THEN PRINT S;" DEATHS BY STARVATION"
1170 IF K>0 THEN PRINT K;" WAR CASUALTIES"
1180 IF V>0 THEN PRINT V;" VICTIMS OF DISEASE"
1190 IF M>0 THEN PRINT M;" VICTIMS OF LOOTING"
1210 LET D=D+S+K+V+M
1220 LET P=P+B-D
1230 IF B+D>0 THEN PRINT "TOTAL: ";P
1250 PRINT "LAND (ACRES): ";A
1260 IF T=0 THEN IF U=0 THEN GOTO 1350
1270 IF T>0 THEN PRINT T;" ACRES BOUGHT"
1280 IF T<0 THEN PRINT -T;" ACRES SOLD"
1290 IF U<>0 THEN PRINT "FRUITS OF WAR: ";U;" ACRES"
1310 LET A=A+T+U
1320 LET T=0
1330 LET U=0
1340 PRINT "TOTAL: ";A
1350 PRINT "GRAIN (BUSHELS): ";G
1360 IF C<0 THEN GOTO 1530
1370 IF C=0 THEN GOTO 1430
1380 LET R=X
1390 PRINT "CROP YIELD ";C;" AT ";
1410 GOSUB 3220
1420 PRINT
1430 IF F>0 THEN PRINT F;" BUSHELS USED FOR FOOD"
1440 IF Q>0 THEN PRINT Q;" BUSHELS PLANTED"
1450 IF L<>0 THEN PRINT "LAND DEALS: ";L;" BUSHELS"
1460 IF H>0 THEN PRINT "MERCENARY HIRE: ";H
1470 IF E>0 THEN PRINT E;" BUSHELS LOST TO RATS"
1480 IF Z<>0 THEN PRINT "FRUITS OF WAR: ";Z;" BUSHELS"
1490 IF O>0 THEN PRINT "LOOTING LOSSES: ";O;" BUSHELS"
1510 LET G=G+C-F-Q+L-H-E+Z-O
1520 PRINT "TOTAL: ";G
1530 GOSUB 3310
1600 REM NOTICE IF GAME ENDED
1610 IF J<100 THEN GOTO 1690
1620 PRINT "THE PEASANTS TIRE OF WAR AND STARVATION"
1630 PRINT "YOU ARE DEPOSED."
1640 PRINT "DO YOU WISH TO PLAY AGAIN";
1650 LET N=Y-1
1660 INPUT N
1670 IF Y<>N THEN END
1680 GOTO 1000
1690 IF P>1 THEN GOTO 1750
1710 PRINT "YOU AND THE REMAINING POPULATION"
1720 PRINT "RETIRE IN THE SWISS ALPS."
1730 GOTO 1640
```

```
1740 REM MAKE LAND DEALS
1750 LET R=23+RND(8)
1760 GOSUB 3250
1770 PRINT "BUY AT ";
1780 GOSUB 3220
1790 INPUT T
1810 IF T<0 THEN GOTO 1760
1820 IF T=0 THEN GOTO 1860
1830 IF R*T<=G THEN GOTO 2060
1840 GOSUB 3270
1850 GOTO 1760
1860 LET R=R-1
1880 GOSUB 3250
1890 PRINT "SELL AT ";
1910 GOSUB 3220
1920 INPUT T
1930 IF T<0 THEN GOTO 1880
1940 IF T=0 THEN GOTO 2060
1950 IF T<=A THEN GOTO 1990
1960 PRINT "BUT THERE IS INSUFFICIENT LAND"
1970 GOTO 1880
1990 IF T<A/10 THEN GOTO 2050
2010 LET R=R-1
2020 PRINT "FOR SELLING SO MUCH YOU CAN ONLY GET ";
2030 GOSUB 3220
2040 PRINT
2050 LET T=-T
2060 LET L=-R*T
2100 REM DISTRIBUTE GRAIN
2110 GOSUB 3250
2120 PRINT "PLANT";
2130 INPUT Q
2140 IF Q<0 THEN GOTO 2110
2150 IF Q<=A+T THEN GOTO 2180
2160 GOSUB 3270
2170 GOTO 2110
2180 IF Q<=P*10 THEN GOTO 2220
2190 PRINT "BUT THERE ARE INSUFFICIENT PEOPLE"
2210 GOTO 2110
2220 PRINT "HOW MANY BUSHELS DO YOU WISH TO ";
2230 PRINT "USE AS FOOD";
2240 INPUT F
2250 IF F<0 THEN GOTO 2220
2260 IF G+L-Q-F>=0 THEN GOTO 2310
2270 GOSUB 3270
2280 GOTO 2220
2310 IF F<=40*P THEN LET S=P-F/40
2320 LET J=J+S
2330 LET X=5+RND(4)
2340 IF Y/7*8-Y/49*7=Y THEN LET X=X/2-1
2350 LET C=X*Q
2360 IF G+L-Q-F+C>=0 THEN GOTO 2390
2370 LET X=(32767-G-L+Q+F)/Q
2380 GOTO 2350
2390 IF RND(99)<25 THEN LET E=(G+L-F+C)/10
2400 REM WAR AND PLAGUE
2410 IF RND(99)>15 THEN GOTO 2880
2420 LET W=25
2430 PRINT "A NEARBY KINGDOM THREATENS WAR"
2440 PRINT "DO YOU WISH TO MAKE A";
2450 PRINT " PRE-EMPTIVE STRIKE";
```

```
2460 LET N=Y-1
2470 INPUT N
2480 IF N=Y-1 THEN GOTO 2520
2490 LET W=100
2510 LET J=J+5
2520 PRINT "HOW MANY MERCENARIES WILL YOU";
2530 PRINT " HIRE AT 80 BUSHELS EACH";
2540 INPUT N
2550 IF N<0 THEN GOTO 2520
2560 LET D=G+L-Q-F+C-E
2570 IF RND(99)<W THEN GOTO 2610
2580 PRINT "PEACE NEGOTIATIONS SUCCEED"
2590 GOTO 2810
2610 IF W=25 THEN LET W=150
2620 LET R=P-S
2630 IF N>R/10 THEN GOTO 2660
2640 LET R=3*W/5*N*N/R*100/R
2650 GOTO 2680
2660 LET R=3*W/5+N*100/R
2680 LET K=(P-S)/2
2690 LET U=-(A+T)/2
2710 LET Z=-D/2
2720 IF R>RND(99) THEN GOTO 2760
2730 PRINT "YOU HAVE LOST THE WAR"
2740 LET J=J+5000/W
2750 GOTO 2810
2760 PRINT "YOU HAVE WON THE WAR"
2770 LET K=K/2
2780 LET U=-U
2790 LET Z=D/4
2810 IF N*80<=D+Z THEN GOTO 2870
2820 GOSUB 3280
2830 PRINT "TO PAY THE MERCENARIES"
2840 LET M=3*(P-S-K)/4
2850 LET O=3*(R+Z)/4
2860 GOTO 2880
2870 LET H=N*80
2880 LET N=P-S-K-M
2890 IF RND(99)>4 THEN GOTO 2940
2910 PRINT "THE BLACK PLAGUE STRIKES"
2920 LET V=N/2
2930 GOTO 2970
2940 IF RND(99)>20 THEN GOTO 2980
2950 PRINT "A POX EPIDEMIC BREAKS OUT"
2960 LET V=N/20
2970 LET N=N-V
2980 LET B=(N*RND(5)+9)/100+1
2990 LET D=(N*RND(3)+4)/100
3010 IF Y/7*8-Y/49*7<>Y THEN GOTO 3030
3020 PRINT "SEVEN YEAR LOCUSTS REDUCE CROP YIELD"
3030 IF E>0 THEN PRINT "RATS INFEST YOUR SILOS"
3040 LET Y=Y+8-(Y-Y/7*7)/6
3050 GOTO 1110
3200 REM UTILITY PRINT ROUTINES
3220 PRINT R;" BUSHELS/ACRE";
3230 RETURN
3250 PRINT "HOW MANY ACRES DO YOU WISH TO ";
3260 RETURN
3270 PRINT "BUT ";
3280 PRINT "THERE IS INSUFFICIENT GRAIN"
3290 RETURN
```

```
3300 REM INITIALIZE POPULATION AND GRAIN VECTORS
3310 LET B=0
3320 LET D=0
3330 LET K=0
3340 LET M=0
3350 LET S=0
3360 LET V=0
3410 LET C=0
3420 LET E=0
3430 LET F=0
3440 LET H=0
3450 LET L=0
3460 LET O=0
3470 LET Q=0
3480 LET Z=0
3490 RETURN
3500 REM INITIALIZATION
3510 LET Y=RND(6)+1
3520 LET J=0
3530 LET P=100
3540 LET A=1500
3550 LET G=5000
3560 LET T=0
3570 LET U=0
3580 GOSUB 3310
3590 LET C=-1
3600 REM PRINT INSTRUCTIONS (MAY BE OMITTED)
3610 PRINT "DO YOU NEED INSTRUCTIONS (Y OR N)";
3620 LET N=P
3630 INPUT N
3640 IF N=P THEN RETURN
3650 IF N=Y THEN GOTO 3710
3660 PRINT "PLEASE TYPE Y FOR YES, N FOR NO. OK";
3670 GOTO 3620
3710 PRINT "YOU HAVE INHERITED THE THRONE TO THE MEDIEVAL"
3720 PRINT "KINGDOM OF EUPHORIA, SOMEWHERE IN EUROPE."
3730 PRINT "AT THIS TIME IT IS ABOUT ";A;" ACRES IN SIZE,"
3740 PRINT "WITH ";P;" LOYAL PEASANTS TO SERVE YOU."
3750 PRINT "IN YOUR ROYAL SILOS YOU HAVE ";G;" BUSHELS OF"
3760 PRINT "NUTRITIOUS GRAIN WITH WHICH TO FEED THE PEOPLE"
3770 PRINT "AND TRANSACT INTERNATIONAL TRADE."
3780 PRINT "CLOSE BY YOUR SIDE ARE JEALOUS NEIGHBORING"
3790 PRINT "KINGDOMS, BUT YOU CAN PROTECT YOURSELF BY"
3810 PRINT "HIRING EVIL MERCENARIES."
3820 PRINT "EACH YEAR YOUR MINISTERS WILL PRESENT YOU"
3830 PRINT "WITH A SUMMARY OF YOUR CURRENT STATUS,"
3840 PRINT "AND THEN ASK YOU FOR DECISIONS ON WHAT"
3850 PRINT "TO DO FOR THE NEXT YEAR. PLEASE TYPE Y OR N"
3860 PRINT "FOR YES OR NO, OR A WHOLE NUMBER FOR"
3870 PRINT "NUMERIC ANSWERS. IF YOU CHANGE YOUR MIND"
3880 PRINT "BEFORE HITTING RETURN, HOLD THE 'CTRL' KEY"
3890 PRINT "DOWN WHILE TYPING THE LETTER 'X', THEN"
3910 PRINT "RETYPE YOUR ANSWER ON THE NEXT LINE."
3920 PRINT "ARE YOU READY, YOUR HIGHNESS";
3930 LET N=P
3940 INPUT N
3950 IF N<>Y THEN GOTO 3920
3980 RETURN
3990 END
```

Chapter 4 -- FLASHCARD - A Math Drill

They always say that a computer can help educate your children. This is a noble goal, but most of us never get any of the round "tuits" that we need to actually pull this off (as in "I'll write that program when I get a round tuit"). With Flashcard, some of that excuse disappears, and you have to think of better excuses to hoard your computer to yourself.

Flashcard is a graded arithmetic drill program, designed to drill the arithmetic skills of elementary school children. It is intended to relate to the level of difficulty being taught, so that the problems presented are neither too hard nor too easy. It does however, require a certain amount of hand-eye coordination (to use the keyboard and display). Also, since it is written in Tiny BASIC, it is not particularly bullet-proof. This latter problem may be best solved by a special keypad with the 10 number keys and <return>. Or, use the program as inspiration to write your own version in assembler with all the protection.

Flashcard lets you choose from one of fourteen levels of difficulty, according to the skill level of your child. These begin with one-digit sums and continue through products and dividends of two digits. All problems are presented with two operands; no multiple sums are included. The levels are as follows:

```
 1. One-digit sums
 2. One-digit differences
 3. Sums of 1-digit operands (may be 2-digit answer)
 4. Differences with one-digit subtrahend and result
 5. Review: 3 & 4 combined
 6. Two-digit sums, no carries
 7. Two-digit differences, no borrows
 8. Two-digit sums, including carries
 9. Two-digit differences, including borrows
10. Review: 8 & 9 combined
11. Product of one-digit operands
12. One-digit quotient with one-digit divisor
13. Two-digit products
14. Review: 8 & 9 & 12 & 13 combined
```

The problems at each level of difficulty have been chosen for a good distribution in the range, to the extent that the random number generator in Tiny BASIC allows (it is known to have consistent patterns: sorry about that).

Several categories of errors are specifically tested for, and appropriate responses generated to suggest to the student the nature of the errors. In particular, errors in carry/borrow, digit transposition, and "closeness" are tested. Numbers radically out of range are also rejected.

Each problem is presented up to three times, with special praise heaped on the correct response the first time. One of a variety of responses is selected each time to help prevent the drill from becoming tiresome.

A note of disclaimer: I am not a professional educator, nor do I have elementary school age children. The levels of difficulty were suggested to me by a teacher working in these grade levels. The text of the responses was entirely my own doing, and you will have to establish the appropriateness of the various responses for your children, as well as the quality of the problems. When I tried this on some school-age kids, the problems seemed to be the right difficulty but often they misunderstood the responses. You may have to reduce the "cuteness" of the responses to correspond to the reading level of your students.

4.1 -- Operation

This program is pretty much self-explanatory. It is pure Tiny BASIC, so no information about the host computer is necessary. The first question requests a level number for the run. Problems are displayed either in the one-line form or operands one above the other, depending on a program parameter (see the Configurator section). Examples:

```
2 + 2 =? __
```

```
   2
+  2
----
? __
```

Ten problems are presented and a statistical summary is given, then another problem set is presented. The score on each subsequent problem set is compared to a running average to note improvement. This repeats indefinitely. The only way to stop or change level of difficulty is to error off (type in something illegal, like a period) and re-run.

4.2 -- Configurator

As this program is pure Tiny BASIC (no USR calls), there is not much to configure. It does use the random number generator, however, to generate problems and select responders.

Those using this program on a video display terminal will probably prefer to keep the problem presentation in the classical operand-over-operand form. Users with only a one-line display or a hard-copy terminal (all that paper!) may prefer the problems to be presented on a single line. This is easily accomplished by setting variable V in line 40 to equal 1.

As listed, the program is 5556 bytes. If you are short on memory, you may wish to reduce each response set down to a single short sentence (with no choices). This requires that the selector statement also be modified to a simple GOTO (or GOSUB).

Each set of messages is collected into a range of statement numbers comprising one block of 100. For example, the 3900s are the messages chosen when the student fails the third attempt. In general, the message selector is a computed GOTO (or GOSUB) which adds a random number multiplied by 20 to the base message line number. To eliminate all but the first message in case of three failures, line 3320 must be changed to remove the RND call:

```
3320 GOSUB 3900
```

A few more bytes can be saved in the cases where it is a GOSUB (rather than GOTO), by removing all the messages from the message block (e.g. remove all lines 3900-3950) and replace the GOSUB by a simple PRINT:

```
3320 PRINT "PFOOEY"
```

The lines that need to be modified for the various message sets are

```
3320 (3900) All three attempts failed
3170 (4000) Correct answer, but not at first
3190 (4100) Correct answer on first try
3220 (4200) Invalid response
3230 (4200) Invalid response
3240 (4300) Carry/borrow error
3250 (4400) Answer is "close", but incorrect
3260 (4400) Answer is close
3280 (4500) Answer is wrong
3720 (4600) One digit correct
3730 (4600) One digit correct
3710 (4700) Digits reversed
3430 (4800) Problem set below average
3440 (4900) Problem set above average
```

4.3 -- Program Notes

The overall structure of this program is fairly simple. A problem is generated by the routine associated with the selected level of difficulty, then it is printed and the answer typed in is compared to the various expected responses and the score tallied. This is repeated ten times, the summary printed and it begins again.

The problem selection is done by a computed GOTO on variable L (the level number). Une number 200 corresponds to level 1; 400 to level 2, and so on to line 2800 for level 14. The operands of the problem, the operation, and the correct answer, as well as several supposed wrong answer syndromes, are set up in program variables, so that from line 3000 a single routine can print all problems and evaluate all responses. From line 3800 to the end of the program are only the quips f rom which are selected a response to the student's answer.

There are not many parts of the program that are difficult to follow. The few that are there are discussed in the next section.

4.4 -- Variables

```
A - First operand of problem
B - Second operand of problem
C - Number of correct answers (so far)
D - Probable answer, if dropped carry
E - Number of errors so far
F - Function: 1 if +, 2 if -, 3 if X, 4 if /
G - Total number of attempts
H - High limit for guessing
I
J
K - Cumulative correct count
L - Level of difficulty
M - A possible reply nearly correct
N - Another nearly correct possibility
O
P
Q - Cumulative average
R - Correct result (computed)
S
T - Try counter
U
V - Format: 0=horizontal, 1=vertical
W
X - Temporary
Y - Temporary
Z
```

4.5 -- Obscure Code

Line 220

This line is for the purpose of determining the function code in all cases except the review. Problem levels 1-5 all come here, but level 5 has already selected the function code. A similar expression occurs in line 1310.

Lines 1640 and 1840

These two lines determine what the result would have been if a carry or borrow was wrong (ie. if the problem caused it, D is the result of suppressing it; if the problem did not, D forces one). The expression inside the parentheses evaluates the sum (or difference) of the units digit only, which divided by ten gives the carry (0 or 1). Multiply by 20 and subtract from the correct result plus ten to get the incorrect result. This is most easily seen with a couple examples:

```
Problem:         12+34   56+28
Correct result     36      84
Units digits      2+4     6+8
Sum                06      14
Carry (tens)        0       1
Times -20           0     -20
Plus 10            10     -10
Variable D         46      74
```

Lines 2600-2640

The selection of the operands for the general multiply problem bears a little explaining. First a provisional product is selected (weighted toward the high tens). Then a multiplier is chosen (with a slightly lower probability of multiplier being 1). This is divided into the provisional product to get the the multiplicand. The two factors are finally multiplied together to get the actual product.

Line 2810

This complicated expression selects one of four lines to go to, based on the four possible values of F: 1 goes to line 1600, 2 to line 1800, 3 to 2600, and 4 to 2400. Dividing F by 3 gives a zero if F is less than 3, one if 3 or 4. Zero effectively eliminates the right half of the expression from consideration. That half can therefore serve as a correction to the value computed by the left half, just in the case of multiply and divide.

Lines 2900-2980

This subroutine is designed to generate a non-linear distribution of random numbers (heavily weighted in the high end). This becomes the sum. A random number within this range is one of the terms, and the difference becomes the other term. The comment (line 2920) shows the distribution of sums. The subroutine is used to generate both digits of a two-digit sum with no carry, and the tens digits of the two-digit sum where carry is allowed. Differences use the same numbers, of course, but re-arrange them.

Lines 3270 and 3710-3740

If both the correct answer and the incorrect student response are greater than 9 (i.e. both have two digits), then perhaps the error is one of the two syndromes tested here. Perhaps the digits are reversed (line 3710), or perhaps one digit is correct and the other is wrong (line 3720 for left digit, 3730 for right digit).

Lines 3430-3450

Because Tiny BASIC cannot cope with fractions, the performance figure (as a fraction of problems answered correctly on the first try) is not directly representable. It is scaled here by 100 to overcome that limitation. This allows a 10% resolution and 327 correct results (at least 32 problem sets) before the scaled value overflows.

4.6 -- Program Listing

```
10 PRINT "ITTY BITTY FLASHCARD PROGRAM"
20 REM COPYRIGHT (C) 1981 T.PITTMAN
30 PRINT
40 LET V=0
50 LET Q=0
60 LET G=0
70 LET K=0
```

```
80 PRINT "LEVEL OF DIFFICULTY";
90 INPUT L
100 X=RND(2)
110 LET E=0
120 LET C=0
130 IF L>0 IF L<15 THEN GOTO L*200
140 PRINT "PLEASE CHOOSE FROM 1 TO 14"
150 GOTO 80
190 REM
190 REM         SET UP PROBLEM
200 GOSUB 2950
210 LET H=9
220 IF L<5 THEN LET F=L/2*2+2-L
230 LET D=-1
240 LET M=R-1
250 LET N=R+1
260 GOTO 3010
400 GOSUB 2950
410 LET A=R
420 LET R=A-B
430 GOTO 210
600 LET A=RND(9)+1
610 LET B=RND(9)+1
620 LET R=A+B
630 LET H=19
640 GOTO 220
800 LET B=RND(9)+1
810 LET R=RND(9)+1
820 LET A=R+B
830 GOTO 210
1000 LET F=RND(8)/4+1
1010 GOTO (F+L)*200-600
1200 GOSUB 2950
1210 LET X=A*10
1220 LET Y=B*10
1230 GOSUB 2950
1240 LET A=A+X
1250 LET B=B+Y
1260 LET R=A+B
1270 LET D=-1
1280 LET H=99
1290 LET M=-1
1300 LET N=-1
1310 IF L<10 THEN LET F=L-L/2*2+1
1320 GOTO 3010
1400 GOSUB 2950
1410 LET X=R*10
1420 LET Y=B*10
1430 GOSUB 2950
1440 LET A=R+X
1450 LET B=B+Y
1460 LET R=A-B
1470 GOTO 1270
1600 GOSUB 2950
1610 LET R=R*10+RND(10)
1620 LET A=RND(R-1)+1
1630 LET B=R-A
1640 LET D=R+10-(A-A/10*10+B-B/10*10)/10*20
1650 GOTO 1280
1800 GOSUB 2950
1810 LET A=R*10+RND(10)
```

```
1820 LET B=RND(A-1)+1
1830 LET R=A-B
1840 LET D=R+10-(A-A/10*10-B+B/10*10+10)/10*20
1850 GOTO 1280
2000 GOTO 1000
2200 LET A=RND(9)+1
2210 LET B=RND(9)+1
2220 LET R=A*B
2230 LET M=R/10*10+A
2240 LET N=R/10*10+B
2250 IF M=R THEN LET M=R-10
2260 IF N=R THEN LET N=R+10
2270 LET F=3
2280 LET H=99
2290 LET D=-1
2300 GOTO 3010
2400 LET R=RND(9)+1
2410 LET B=RND(9)+1
2420 LET A=R*B
2430 LET M=R+1
2440 LET N=R-1
2450 LET F=4
2460 LET D=-1
2470 LET H=9
2480 GOTO 3010
2600 GOSUB 2950
2610 R=R*10+RND(10)
2620 LET B=9-RND(25)/3
2630 LET A=R/B
2640 LET R=A*B
2650 LET D=R-(A-A/10*10)*B/10*10
2660 LET M=R+10
2670 LET N=D-10
2680 IF R-D>10 THEN LET M=R-10
2690 IF R-D>20 THEN LET N=D+10
2700 LET F=3
2710 LET H=99
2720 GOTO 3010
2800 LET F=RND(8)/2+1
2810 GOTO F*200+1400+F/3*(1800-F*400)
2900 REM
2900 REM        SELECT 1-DIGIT SUM, NO CARRY
2910 DISTRIBUTION OF RESULT (IN 81 CASES):
2920 REM 2-1 3-2 4-5 5-7 6-9 7-14 8-19 9-25
2950 LET R=9-(RND(9)+1)*(RND(9)+1)/11
2960 LET A=RND(R-1)+1
2970 LET B=R-A
2980 RETURN
3000 REM
3000 REM        DISPLAY PROBLEM & ACCEPT ANSWER
3010 LET T=0
3020 IF V>0 THEN GOTO 3100
3030 LET X=A
3040 GOSUB 3510
3050 GOSUB 3610
3060 LET X=B
3070 GOSUB 3520
3080 PRINT "----"
3090 GOTO 3130
3100 PRINT A;" ";
3110 GOSUB 3610
```

```
3120 PRINT B;" =";
3130 INPUT X
3140 LET T=T+1
3150 IF X<>R THEN GOTO 3210
3160 LET C=C+1
3170 IF T>1 THEN GOTO RND(4)*20+4000
3180 LET K=K+1
3190 GOTO RND(5)*20+4100
3200 REM
3200 REM        IDENTIFY ERROR TYPE
3210 LET E=E+1
3220 IF X<0 THEN GOTO RND(3)*20+4200
3230 IF X>H THEN GOTO RND(4)*20+4200
3240 IF X=D THEN GOTO RND(3)*20+4300
3250 IF X=M THEN GOTO RND(4)*20+4400
3260 IF X=N THEN GOTO RND(4)*20+4400
3270 IF R>9 THEN IF X>9 THEN GOTO 3710
3280 GOSUB RND(5)*20+4500
3300 REM
3300 REM        ADVANCE TO NEXT TRY
3310 IF T<3 THEN GOTO 3020
3320 GOSUB RND(3)*20+3900
3330 PRINT "THE ANSWER IS ";R
3340 LET G=G+1
3350 PRINT
3360 IF G>G/10*10 THEN GOTO L*200
3370 PRINT "YOU GOT ";C;" ANSWERS CORRECT"
3380 IF E>0 THEN PRINT "BUT GAVE ";E;" WRONG ANSWERS."
3390 IF G=10 THEN GOTO 3450
3410 PRINT "IN THE LAST ";G;" LEVEL ";L;" PROBLEMS,"
3420 PRINT "YOU GOT ";K;" ANSWERS PERFECT."
3430 IF K*100/G<Q THEN GOSUB RND(2)*20+4800
3440 IF K*100/G>Q THEN GOSUB RND(3)*20+4900
3450 LET Q=K*100/G
3460 PRINT "LET'S GO AGAIN."
3470 GOTO 110
3500 REM
3500 REM        PRINT ONE NUMBER ON A LINE
3510 PRINT "  ";
3520 IF X<10 THEN PRINT " ";
3530 PRINT X
3540 RETURN
3600 REM
3600 REM        PRINT OPERATION SYMBOL
3610 IF F=1 THEN PRINT "+ ";
3620 IF F=2 THEN PRINT "- ";
3630 IF F=3 THEN PRINT "X ";
3640 IF F=4 THEN PRINT "/ ";
3650 RETURN
3700 REM
3700 REM        TRY FOR 2-DIGIT ERRORS
3710 IF R/10+(R-R/10*10)*10=X THEN GOTO RND(3)*20+4700
3720 IF R/10=X/10 THEN GOTO RND(2)*20+4600
3730 IF R-R/10*10=X-X/10*10 THEN GOTO RND(2)*20+4600
3740 GOTO 3280
3800 REM
3800 REM        SELECTED QWIPS
3900 PRINT "LET'S FACE IT: YOU'RE NOT GOING TO GET THIS ONE."
3910 RETURN
3920 PRINT "THIS ONE IS TOO HARD FOR YOU, EH?"
3930 RETURN
```

```
3940 PRINT "LET'S NOT WASTE ANY MORE TIME ON THIS ONE."
3950 RETURN
4000 PRINT "NOT BAD FOR ";T;" TRIES."
4010 GOTO 3340
4020 PRINT "WELL, IT'S ABOUT TIME YOU GOT IT RIGHT."
4030 GOTO 3340
4040 PRINT "WHEW! I THOUGHT FOR A MINUTE IT WAS TOO HARD."
4050 GOTO 3340
4060 PRINT "DO YOU ALWAYS WAIT SO LONG TO GET IT RIGHT?"
4070 GOTO 3340
4100 PRINT "FANTASTIC! YOU GOT IT THE FIRST TRY."
4110 GOTO 3340
4120 PRINT "WE OUGHT TO DO THIS MORE OFTEN."
4130 GOTO 3340
4140 PRINT "GREAT! DO YOU GET BROWNIE POINTS FOR NO MISTAKES?"
4150 GOTO 3340
4160 IF C<E THEN PRINT "YOU'RE GETTING BETTER."
4170 IF C>E THEN PRINT "HEY, YOU'RE SHARP TODAY!"
4180 PRINT "RIGHT ON!"
4190 GOTO 3340
4200 PRINT "AW, C'MON, THAT'S NO ANSWER!"
4210 GOTO 3310
4220 PRINT "YOU'RE SUPPOSED TO TYPE A NUMBER FROM 1 TO ";H
4230 GOTO 3310
4240 PRINT "WHAT A SILLY ANSWER! NOW STOP PLAYING."
4250 GOTO 3310
4260 PRINT "THAT'S WAY TOO BIG. TRY AGAIN."
4270 GOTO 3310
4300 PRINT "YOU MISSED THE";
4310 GOTO 4370
4320 PRINT "WATCH OUT FOR THE";
4330 GOTO 4370
4340 PRINT "ALMOST RIGHT, EXCEPT FOR THE";
4370 IF F=2 THEN PRINT " BORROW."
4380 IF F<>2 THEN PRINT " CARRY."
4390 GOTO 3310
4400 PRINT "YOU'RE GETTING WARM!"
4410 GOTO 3310
4420 PRINT "CLOSE, BUT NO CIGAR."
4430 GOTO 3310
4440 PRINT "NOT BAD, BUT NOT GOOD EITHER."
4450 GOTO 3310
4460 PRINT "ALMOST ONLY COUNTS IN HORSESHOES AND HAND GRENADES."
4470 GOTO 3310
4500 PRINT "AW, YOU'RE JUST GUESSING."
4510 RETURN
4520 PRINT "YOU'RE NOT TRYING AT ALL."
4530 RETURN
4540 PRINT "YOU CAN DO BETTER THAN THAT!"
4550 RETURN
4560 PRINT "THAT'S NOT EVEN CLOSE!"
4570 RETURN
4580 PRINT "THINK A LITTLE HARDER."
4590 RETURN
4600 PRINT "ONE DIGIT RIGHT IS BETTER THAN NONE."
4610 GOTO 3310
4620 PRINT "THAT'S ONLY HALF RIGHT."
4630 GOTO 3310
4700 PRINT "YOU AREN'T CROSSEYED, ARE YOU?"
4710 GOTO 3310
4720 PRINT "RIGHT NUMERALS, WRONG PLACES."
```

```
4730 GOTO 3310
4740 PRINT "NO, THAT'S BACKWARDS."
4750 GOTO 3310
4800 PRINT "YOU MUST BE GETTING TIRED."
4810 RETURN
4820 PRINT "THAT'S NOT SO GOOD."
4830 RETURN
4900 PRINT "HEY, YOU'RE GETTING BETTER!"
4910 RETURN
4920 PRINT "GOOD GOING! I'M PROUD OF YOU!"
4930 RETURN
4940 PRINT "KEEP THAT UP, AND WE CAN GRADUATE TO THE NEXT LEVEL."
4950 RETURN
5000 END
```

Chapter 5 -- TINY CALC - A Spreadsheet Calculator

Tiny BASIC is limited to 16-bit integers for its arithmetic (about 4.5 digits). This is inadequate for most financial computations, and in particular it is incapable of coping with decimal numbers (dollars and cents), percentages, and the like. Nevertheless an important function of computers in the home and business is financial computations. Tiny Calc lets you perform calculations to 12-digit precision and two decimal places, and saves its results to over eight significant digits. Thus Tiny Calc can carry out financial calculations to a result exceeding $1,000,000.00 accurate to the nearest cent.

A further requirement of financial calculations is that the same procedure must be carried out on a (possibly small) amount of data every day or week or month or year. The calculation usually involves many intermediate values, which for auditing purposes must be printed out. Human bookkeepers often do their calculations on ledger sheets with 40 or so lines of six or eight columns, so that the intermediate values show up as line items. Tiny Calc displays its results in the same format, so little or no readjustment is needed to convert bookkeeping methods. Up to 1791 items, organized in up to 199 lines (a.k.a.rows) of up to 9 columns each may be calculated and displayed by Tiny Calc.

Handwritten spreadsheets normally require only one or two calculations per item. Tiny Calc allows up to four, with facilities for extending this indefinitely.

Other features of Tiny Calc include optional rounding to whole numbers or one decimal place, optional non-printing of selected intermediate results, and the ability to "do the same thing as" some other item's calculation.

5.1 -- How It Works

A Tiny Calc spreadsheet consists of many items, organized into rows and columns, with one intermediate or final result per item. For example, the 1980 Federal

income tax form 1040 has lines numbered from 1 to 66, generally with only one number on each line. Some of the lines used for intermediate calculations are offset to the left, so Tiny Calc might be set up with two columns to reflect this offset. A few lines have a), b), and c) parts, so you might wish to define three columns in all, and leave the unused items blank. If no calculation is specified for an item, nothing is printed there.

To set up Tiny Calc for a new job, you first specify the number of rows and columns to be calculated. You may also give the printout column headings and row labels. The number of rows and columns is specified by the variable L (for Last), which is the row and column number of the last item. For example, a form 1040 spreadsheet with 3 columns and 66 rows would set L=663, where the units digit (3) is the number of columns, and the digits to the left (66) are the number of rows. The highest number of rows allowed is 199. Too many rows, or zero rows, or zero columns are all errors. The last row and column limits are set in line number 3 of the Tiny Calc program:

```
          ____last row is 66
         //
3 LET L=663
           \__last column is 3
```

At every occupied column position in every row, you provide a "calculation specification", and if desired, its rounding and printing qualifications. You may type these in any order, since they are entered with row and column identifications, and stored in memory together with the particular item.

Calculation specifications are coded as assignment statements (LET) in Tiny BASIC, as part of the Tiny Calc program. Each item sets up any or all of nine variables (letters A to I), which define the specifications for one item's calculation. When it comes time to calculate that item, these lines are called as a subroutine from the main part of the Tiny Calc program, which then uses the values to direct the calculation. You do not type in the actual arithmetic expressions that are used, but rather a sequence of operation-operand codes. Any variables that are not given values for a particular item are assumed to have a default value (usually zero, which stands for no operation). The order in which the variables are assigned in an item's specification is not significant, but the values given to the variables are.

The tens digit of the line number on an operation specification defines the column, and the digits to the left (hundreds, thousands) define the row. The units digit is used to separate the different specifications. Thus the following line is the 4th specification for row 12, column 3.

```
  ___________row 12
 //
1234 B=1041
   \\_________specification 4 for row 12, column 3
```

```
        \_________column 3
```

In general, there are two kinds of operations. One kind refers to another item for its operand value, and the other kind has the operand value encoded in the operation number. Both kinds are numbers, where the right-hand digit defines the operation. The digits to the left in the one case specify the row and column where the operand is to be found, and in the other case specify a percentage value to be applied.

Five variables A-E are used to specify the operations to calculate the value of an item. Variable A is assigned a number that specifies the first operation to be performed (or sets up the initial value). Variable B specifies the next operation; variable C the next after that, then D, and finally E is the last operation to be performed. If less than five operations are needed, the excess variables may be ignored. If more than five are needed, a subroutine call is required (see operation code 0 below).

5.2 -- Operators

There are ten Operators, numbered 0 to 9. The Operator is placed in the rightmost position of an Operation Code number. Operators 0 to 6 are used with a row and column to refer to some other item. Operators 7 to 9 are used with a percentage value.

A reference to a row and column is by number: RRC. The tens digit defines the column (1 to 9), and the digits to the left of that define the row (1 to 199). A row or column specified as zero (0) is taken to mean the same row or column as the item being calculated. If Row and column are both zero, it has special significance, and normally refers to the value of an entered constant (see below).

```
              ___operator 1 (Add)
             /
1234 B=1041 <--Operation Code
        \\\____column 4
         \\____row 10
```

Percents are entered as percent and tenths: nn.n%, written as nnn. The tenths of % go in the tens digit of the Operation Code, the units of % goes in the hundreds digit of the Operation Code, etc. Thus to add 6.5%, the percentage code 650 is added to Operator 7 (see Add Percent, below), for a total Operation Code of 657. Note that percentages greater than 327.5% cannot be specified by this method, since that would require an Operation Code larger than the largest (Tiny BASIC) number, 32767.

```
              ___Operator 7 (Add Percent)
             /
1235 C=657 <--Operation Code
        \\____6.5%
```

The Operators:

1 = Add

Adds the specified item (row and column) to the current item. If this is the first operation (i.e. assigned to variable A), then the meaning is equivalent to "start with item..." If row 0 column 0 is specified, the value of the constant for this item is used. A row greater than that of the item being calculated, or a column greater than the item being calculated on the same row, is considered to be in error (since that item has not yet been calculated).

2 = Subtract

Subtracts the specified item from the current item's running value.

3 = Multiply

Multiplies the current item's running value by the the specified item.

4 = Divide

Divides the current item's running value by the specified item.

5 = Minimum

Compares the specified item to the current item's running value, and then sets the item's running value to whichever is smaller.

6 = Maximum

Compares the specified item to the current item's running value, and then sets the item's running value to whichever is larger.

7 = Add Percent

Calculates the specified percentage of the current item, and then adds it to the current item's running value.

8 = Subtract Percent

Calculates the specified percentage of the current item, and then subtracts it from the current item's running value.

9 = Multiply Percent

Replaces the current item with the specified percentage of it.

0 = Same As (Subroutine Call)

The calculation of this item follows the same procedure as the specified row and column. This should be the last item in the sequence, since any unprocessed operations (variables higher than the one assigned to this operation code) may be lost or erroneous in the context of the operations referred to. This operation need not result in the same calculated value as the referenced item, since any operations already done will have affected the

initial value, and since any operations referring to row or column zero are understood to refer to the row or column of the item being calculated, not necessarily the same as the item where the specifications are.

The row and column specifications in the subroutine call operation are not as limited as for item value reference. There is no requirement that the item be lower in row or column number. In fact, it may be outside the computed range of the calculator entirely, just so long as there are specifications at the appropriate line numbers (if not, then Tiny BASIC will error off). Calculations specified for items outside the row and column boundaries of Tiny Calc specifications can only be referenced by a subroutine call. A reference to the row and column of the item being calculated (i.e. row 0 and column 0) is not meaningful. Row 0, column 0, operation 0 is therefore considered to be "no operation" (ignored).

The two main uses of operation zero are to save on entering duplicate calculations, and to enable the calculation of an item to have more operations than the five in variables A to E. Consider an invoice, where each line item is calculated by multiplying the quantity in column 2 times the price in column 3 to give the total price in column 4. Row 1, column 4 is given the whole calculation, and each successive row in the column is "the same":

```
...
141 A=021
142 B=033
...
241 A=140
...
341 A=140
...
```

In row 1 (lines 141 and 142) the row number is implied (zero); i.e. use the present row (row 1). Thus, when the same calculation is used in row 2 (line 241 has an Operation Code of row 1, column 4, operator 0 "same as"), the calculations will be the same as row 1, but using the values of the items from row 2.

5.3 -- Constants

Numbers are entered into Tiny Calc as constants. A constant may be as large as 32,767,999.99 though the final value of an item is limited to one tenth that value. The constant is entered in three parts, and any parts that are zero need not be specified. The three parts are the High part, in variable H; the Integer part, in variable I; and the Fractional part, in variable F. The High part specifies the thousands, for numbers greater than 999. The Integer part is normally 1 to 999, but can be as large as 32,767. The Fractional part specifies the hundredths (cents), but for small dollar amounts, the dollar part may also be specified in F for convenience. Thus F=1342 is the same as I=13 and F=42. Similarly, small thousands (to a maximum of 32,767) may be specified in the Integer part I.

Negative constants are entered by making any of the parts negative. The following examples all define the same constant, -2,047.90:

```
1. H=-2 I=47    F=90
2.      I=-2047 F=90
3. H=2  I=-46   F=190
4. H=2          F=-4790
5. H=2  I=-7    F=4090
```

If an item in a particular row/column position consists solely of the entered constant, then none of the Operation variables need to be specified. When variables A-E are all zero (unspecified) and a constant is specified (H, I, and/or F not zero), then the constant is taken for the item value.

If two or more operations specify a constant in the calculation of the same item, then the first operation uses the value of the specified constant (or zero, if none is specified), and all subsequent operations use the value zero. Thus a single item can be specified to be "not more than some number, and not less than zero". For example:

```
A=...   (compute value to be limited)
H=1     (set High limit; constant 1000)
C=5     (5=Minimum; returns current value or 1000, whichever is less)
D=6     (6=Maximum; returns current value or 0, whichever is greater)
```

5.4 -- Print Format

Unless otherwise specified, all numbers are printed to two decimal places. This is easily overridden by assigning a value to variable G. If G=-1 the item is not printed. If G=0, 1, or 2, the item is printed with that many decimal places. If you specify 0 decimal places, no decimal point is printed. The latest value of G prevails, so if row 5 column 2 sets G=-1 (don't print) and then calls row 2 column 3 with Operation 0 (same as), and that item sets G=2, then the item in row 5 column 2 will be printed to two decimal places.

The value of an item is always rounded to the number of places printed (or to two places if not printed), so any reference to it from another item gets exactly the same value as printed. Rounding is by the "banker's rule": Round to the nearest unless there is a tie; ties are rounded up or down so that the last digit comes out even. Thus 12.345 is rounded down to 12.34 and 12.355 is rounded up to 12.36.

The column headers are specified with print statements in program lines 9 to 89. Each column item should take exactly ten positions (BASIC "comma tabbing" does not work here). In addition, each column header should leave space over the line labels. As many header lines may be printed as desired.

Row labels are defined as subroutines in the position of column 0 (i.e. line numbers ending in 00). If two-line row labels are desired, with the second line on

the same printed line as the line items, then both lines of labels can be printed together. If a third line label is desired below the line of the line items, then the third line's PRINT should be on line numbers ending with 05 and have their own RETURN. If no label is to be printed below the line of numbers, the RETURN for the line labels should be on the line numbers ending in 05. In the example below, row 1 (lines 100-105) has only a one-line label, row 2 (lines 200-205) has a two-line label, and row 3 has a three-line label:

```
100 PRINT "LINE 1",
105 RETURN

200 PRINT "TWO LINES OF"
202 PRINT "LINE 2",
205 RETURN

300 PRINT "THREE LINES OF"
302 PRINT "LINE 3",
304 RETURN
305 PRINT
306 PRINT "THIRD LINE";
308 RETURN
```

Notice that the part of the label printed on the line with the line items must end in a comma or semicolon. If a semicolon , then you must be careful to ensure that all line labels are exactly the same length, or the columns will not line up correctly. Ending with a comma makes it easier, since then the labels only need to be about the same length. Label lines above the number line should not end in either a comma or semicolon. Label lines below the line with the printed items should end in a semicolon, unless you want an extra blank line between lines. If the extra blank PRINT for the third line of a label is left out, the label will print on the right-hand end of the number line.

If all the lines are to have labels that extend below the item line (and none on the right end), or if double-spacing is desired, then these extra PRINTs may be omitted by setting the last row-column variable L negative. A negative value in L forces an extra PRINT at the end of each line of items, before jumping to line number xxx05.

5.5 -- Putting It All Together

To use Tiny Calc, first layout your spreadsheet on a piece of paper. Show where items will be printed, and what the row and column labels will be. Then for each item (i.e. each column of each row where there is a number), write down where that number comes from or how it is computed. By doing this ahead of time on a piece of paper, it is easier to clarify your thinking, and possibly change your mind about line numbers, etc. You should then load the Tiny Calc main program and its default line items. You need to load as many defaults as there are rows times columns specified in variable L; these must be loaded even if that part of your spreadsheet is to be blank. Then type in the specifications for your application,

including any data values (as constants). Finally, type RUN, and sit back and relax or go get yourself a cup of coffee (Tiny Calc is rather slow)! It will stop after it finishes, or if it finds an error. In either case you may need to change some of the specifications if it did not do exactly what you wanted. Just retype the lines with the numbers or operations that need changing, then type RUN again.

5.6 -- Error Summary

```
ROW nn?         Row nn is outside the range specified by L
COLUMN n?       Column n is outside the range specified by L
SYNTAX          Attempt to subtract 100% or more
OVERFLOW        Result is too large
0 DIVISOR       Attempt to divide by 0
```

5.7 -- Special Effects

Often it is desirable to create a running total, or otherwise make a computation that depends on an adjacent column or row. This is easy to do, by reference to its row and column number. However, if we want to use the same computation in several places without retyping the specifications every time, it would be nice to have a convenient relative notation. Its lack is a minor limitation of Tiny Calc, but we can get around it by appeal to the Tiny BASIC that underlies it.

The variable N normally holds the current row and column number, in the same format as L (but not negative). Thus N-1 refers to the previous column, and N-10 refers to the previous row. N*10+1 refers to the current row and column, for the operation ADD. Therefore, a running column total can be specified by

```
B=(N-10)*10+1
```

or equivalently,

```
B=N*10-99
```

As an example, Pascal's Triangle can be generated by setting column 1 of every row to the constant 1, and every other number except row 1 to the sum of the two numbers above and to the above left of it. Here is the complete specification for the first four rows:

```
3 L=49
10 PRINT "PASCAL'S TRIANGLE"

100 REM
105 RETURN
110 I=1
111 G=0
119 RETURN
120 REM
130 REM
140 REM
150 REM
```

```
160 REM
170 REM
180 REM
190 RETURN
200 REM
205 RETURN
210 A=110
219 RETURN
220 A=N*10-9
221 B=N*10-99
228 G=0
229 RETURN
230 REM
240 REM
250 REM
260 REM
270 REM
280 REM
290 RETURN
300 REM
305 RETURN
310 A=110
319 RETURN
320 REM
330 A=220
339 RETURN
340 REM
350 REM
360 REM
370 REM
380 REM
390 RETURN
400 REM
405 RETURN
410 A=110
419 RETURN
420 REM
430 REM
440 A=220
449 RETURN
450 REM
460 REM
470 REM
480 REM
490 RETURN
```

5.8 -- Rounding

Tiny Calc is programmed to round results according to the so-called "banker's rule". This eliminates a tendency of numbers to creep up through repeated roundings. The rule is that when the quantity to be rounded is exactly halfway between possible answers the answer with the even last digit is chosen.

Consider a calculation (admittedly absurd): X+Y-Y+Y-Y+Y-Y. Let us say X is 10 and Y is 0.5 and we are rounding to whole numbers after each step. The result of this calculation is the original value with banker's rule rounding, but it creeps up to 13 with the usual "add 0.5 and throw away the fraction" rounding:

	Round by Adding 0.5		Round by Banker's Rule		True
Step	Before	After	Before	After	Value
1 +Y	10.5	11	10.5	10	10.5
2 -Y	10.5	11	9.5	10	10
3 +Y	11.5	12	10.5	10	10.5
4 -Y	11.5	12	9.5	10	10
5 +Y	12.5	13	10.5	10	10.5
6 -Y	12.5	13	9.5	10	10

This example shows that banker's rule rounding (on the average) is likely to give answers more nearly correct. However, it is more complicated, and for this reason it is not always used in the real world. One important exception is in IRS forms, where the rounding method is specified as "Add 0.5". Therefore, for doing your income tax on Tiny Calc, you may want to disable the banker's rule. To do this, remove three lines from the program: 21350, 21380, and 21420. With this change, rounding will be strictly "Add 0.5 and Discard Fraction".

5.9 -- Data Work Space

All of the computed values in Tiny Calc are stored in memory by pokes (USR calls). As written, the area used is taken out of the GOSUB stack. This is a little slow, and if you have an available block of memory somewhere else, you can change it to use that without all the computation to allocate stack space. Four bytes are needed for every position, so for a 15x9 matrix the memory size needed is 15x9x4=540 bytes. Smaller arrays need less; larger need more.

To program your own data memory space, remove all the program lines from 20210-20330 and replace them with a single line

```
20210 M=nnnn
```

where nnnn is the decimal address of the first byte of the memory block.

5.10 -- Program Size

A typical Tiny Calc calculation needs about 30 bytes of program memory space for every position on the spreadsheet that you calculate; complicated calculations will need more. These bytes are used for the two or three lines of program statements required to specify the calculations to be performed. For example, an income tax calculation will require about 60x2x30=3600 bytes of program storage in addition to the Tiny Calc program. If you code it carefully with only one column, only half as much may be needed. This may still tax the size of your computer system, and further steps to reduce the program size may be needed.

One important way to get more space is to eliminate the unnecessary bytes of the executive program. Ten lines may be eliminated if you isolate a place in memory for the data (see above, Data Work Space). The program was written to be readable, with spaces, comments, etc. Remove all the comment lines, spaces,

keywords LET and THEN, and shorten PRINT (important: do not remove spaces inside quotes in Print statements). Also remove the three lines that do banker's rule rounding (see above). This will save about 2700 bytes in all.

If you are really hard up for space, and are willing to be satisfied with only 19 rows (down from a maximum of 199), you can get 78 more bytes by taking the final zero off all the line numbers, then fixing all the GOTOs and GOSUBs to match (simply remove the final zero from every line with "GO" in it -- 71 in all). There are seven lines that need special handling, marked in the listing by two spaces after the line number instead of one. Three of these take an extra zero or two off in unobvious places (as shown below), and the other four should not be changed (take no zeros off).

```
2016 IF L/10<>0 THEN IF L<200 THEN IF L>-200 THEN GOTO 2021
2223 IF O>6 THEN GOTO O*5+2226
2229 IF J<=N THEN IF O>0 THEN GOTO O*40+2201
```

With all these changes, the program should weigh in at a little under 4400 bytes (plus whatever is needed for row/column specifications). This allows enough space in an 8K memory to do a small spreadsheet (like the example following), but probably not enough to do your income taxes.

5.11 -- Configurator

Tiny Calc consists of two parts. The first part is the executive, and is necessary for Tiny Calc to work. The second part has the default row/column table definitions, and you only need to enter as much as you are going to use. If you have enough memory, you will generally put in all the defaults for as big a spreadsheet as you will ever want with the executive, then save them both together. For individual runs, the saved version is loaded and the specifications entered in over it. If memory is limited (and for most of us that is always true), you may want to make several default saves (say, a long narrow one, a square one, etc.). Row/column information outside the bounds set in variable L on line 3 are not used, but do not hurt anything. However, you must have row labels and row/column defaults for every position in the table within the bounds specified by line 3, or Tiny BASIC will stop on a missing line number. The default listing in this book only covers the first 15 rows; for more rows, you need to make up your own additional defaults.

Tiny Calc accesses its data by peek/poke USR calls. Line 98 defines the address for peek in your computer. It is printed as 276 (for a Cold Start of 256); you should replace that line with one appropriate to your system. Byte sex is determined automatically in lines 20210-20240, and is only used to calculate the actual address of the data area. If you supply an address, this calculation is unnecessary.

Four lines do the actual data fetch and storage. If variables M and P are set up correctly (line 98 above, and lines 20210 and following as printed, or your

replacement), then these need not concern you. For the record, they are lines 21460-21470 and 24440-24450.

Z8 Tiny BASIC has different reference addresses, so extra modifications are required. Change the four references to "P+4" in lines 21450-21460 to "129", then set P=71 in line 98 and allocate your own data area (line 20210; remove 20220-20330 as described in the section on memory usage). If you want Tiny Calc to automatically allocate its data area out of the stack, then the following lines must also be replaced with the new code shown below:

```
20210 GOSUB 20250
20250 LET M=^6+2
20280 IF M+J<^10 THEN GOTO 21010
20330 LET M=^6+2
```

5.12 -- A Business Example

Probably the best way to see how Tiny Calc works is to see a realistic problem done with it. Here is an imaginary sales tax budget for a small company doing mail-order business, with some taxable and some non-taxable revenues. The tax rate in this example is 6.5% and the businessman is required to make quarterly payments.

Six columns are required for this report; one each for taxable and non-taxable receipts, one for the gross, one to show the calculated tax, one to show the payments made to the government, and finally a running total of the taxes due (shown as a credit balance, which perhaps by "creative accounting" is mostly negative). Fourteen rows allow for one row for each month, a yearly total, and some space for neatness.

To run this example, first load the executive program, with the default row and column information for 14 rows and 6 columns. Then type in the following lines. Line 3 specifies the array size at 14 rows by 6 columns. Lines 9, 10, 40, and 70 define the header across the top. Notice that the column headings are carefully spaced in fields of 10 characters.

```
3 L=146
9 PRINT "SALES",
10 PRINT " OUT-STATE  IN-STATE";
40 PRINT "     GROSS      TAX ";
70 PRINT "   DEPOSIT    CREDIT"
```

Now type in the row labels. Notice that each one ends with a comma, like the beginning of the header line (line 9). We will print no data on row 13, so its row label is extended to provide underlines for the columns.

```
100 PRINT "JAN",
200 PRINT "FEB",
300 PRINT "MAR",
```

```
400 PRINT "APR",
500 PRINT "MAY",
600 PRINT "JUN",
700 PRINT "JUL",
800 PRINT "AUG",
900 PRINT "SEP",
1000 PRINT "OCT",
1100 PRINT "NOV",
1200 PRINT "DEC",
1300 PRINT "","  --------  --------";
1301 PRINT "  --------  --------";
1302 PRINT "  --------  --------"
1400 PRINT "TOTALS",
```

Column 3 is the gross receipts; that is, the sum of the quantities in columns 1 and 2. This is specified for row 1 by the Add operation (units digit 1), columns 1 and 2 (tens digit), and "current row" (0 in the hundreds digit).

```
130 A=011
131 B=021
```

Column 4 calculates the tax on the taxable receipts in column 2. This is done by getting current row (0xx) column 2 (x2x), Add (xx1) in the first step (variable A), then multiplying (xx9) by 6.5% (65x) in the next step (variable B).

```
140 A=21
141 B=659
```

Row 1 column 6 is the difference of column 5 (add: 1) minus column 4 (subtract: 2). Row 2 column 6 adds to that the previous row's value. This is computed by the method mentioned under special effects (above), that is, N*10-100 with the Add code 1 (total N*10-99) for the first step, then do the same thing as row 1 column 6 (xx0 + 1xx + x6x = 160), namely, add column 5 of the current row (this time row 2) and subtract column 4.

```
160 A=51
161 B=42
260 A=N*10-99
261 B=160
```

Column 5 is for quarterly payments, which are first calculated in the fourth month (row 4). The payment is the running total due at the end of the previous month (column 6, previous row), or $50, whichever is greater. To get the running total, we could specify 361, but we want this same specification to be used also for the next two quarterly payments so we compute it as N*10 (current row/column) minus the row/column where we happen to be figuring this (450), plus the row/column where its data is (360), plus finally the function (2 for subtract, so the signs come out right). This gets the negative of the running balance. Then we compute the maximum of that value and constant 50 (in variable I, specified by row/column 00) with function 6.

```
450 A=N*10-450+360+2
451 B=6
452 I=50
```

Now we are ready to figure the totals at the bottom (row 14). The column total for 12 rows exceeds the capacity of Tiny Calc, so we will define a subroutine to do it, and put it in the non-existent row 15. "Column 1" of this subroutine adds the first four rows of the current column (we use zero here so that the same subroutine works for all columns), then jumps to the next part of the subroutine in "column 2". There the next four rows are added before jumping to the third part. The end of the subroutine is whenever there is nothing else to do, which occurs after the last four rows are added. The subroutine is called by a jump to row 15/column 1, so we code that for each of the first five columns of row 14. Column 6 of row 14 is just the same as the running total in row 12, and does not need the subroutine to compute it.

```
1510 A=101
1511 B=201
1512 C=301
1513 D=401
1514 E=1520
1520 A=501
1521 B=601
1522 C=701
1523 D=801
1524 E=1530
1530 A=901
1531 B=1001
1532 C=1101
1533 D=1201
1410 A=1510
1420 A=1510
1430 A=1510
1440 A=1510
1450 A=1510
1460 A=1261
```

Finally, we need to fill in all the computations that are the same as the ones we defined. Deposits need to be figured in months 7 and 10, using the same computation as in month 4 (row 4 column 5 function 0). Columns 3 and 4 of every row from 2 to 12 are the same as row 1, and column 6 is the same as row 2.

```
750 A=450
1050 A=450
230 A=130
240 A=140
330 A=130
340 A=140
360 A=260
430 A=130
440 A=140
460 A=260
530 A=130
540 A=140
560 A=260
```

```
630 A=130
640 A=140
660 A=260
730 A=130
740 A=140
760 A=260
830 A=130
840 A=140
860 A=260
930 A=130
940 A=140
960 A=260
1030 A=130
1040 A=140
1060 A=260
1130 A=130
1140 A=140
1160 A=260
1230 A=130
1240 A=140
1260 A=260
```

Now the calculation part of the program is complete. It remains only to put in the data for each month and year, and admire the results. Let's assume an initial payrnent of $50 (row 1 column 5). The actual sales figures are coded as constants in columns 1 and 2 of each row. For this illustration, assume that December figures are not in yet. The other numbers are quite unrealistic, to illustrate various features. For example, July was a bad month and returns exceeded sales in the taxables. October taxable sales were identical to September (yes, I know it does not happen that way, but this is only an example), so we used a "same as" operator (0); but since now there is some operation specified, the constant value must be explicitly referenced, thus the Add Constant (001) in variable B. November taxables are split up strangely, with overlapping constants. Tiny Calc adds them.

```
150 I=50
110 I=349
111 F=83
120 I=123
121 F=17
210 I=1266
211 F=59
220 H=30
221 I=528
222 F=16
310 F=1098
320 I=206
321 F=87
410 I=10460
420 I=176
421 F=13
510 I=593
511 F=37
520 I=33
521 F=176
610 I=984
611 F=29
620 I=29
```

```
710 F=385
720 I=-4
721 F=52
810 I=396
811 F=69
820 I=693
821 F=30
910 H=1
911 F=8
920 I=335
921 F=55
1010 I=28
1020 A=920
1021 B=1
1110 I=367
1111 F=88
1120 I=1884
1121 F=15398
```

If you have put in all the program codes and data correctly, you should get the table below printed out. To help you understand how the functions work, try varying things one at a time, and compare the results.

```
SALES      OUT-STATE  IN-STATE     GROSS       TAX   DEPOSIT
CREDIT

JAN           349.83    123.17    473.00      8.01     50.00
41.99
FEB          1266.59  30528.16  31794.75   1984.33           -
1942.34
MAR            10.98    206.87    217.85     13.45           -
1955.79
APR         10460.00    176.13  10636.13     11.45   1955.79    -
11.45
MAY           593.37     34.76    628.13      2.26             -
13.71
JUN           984.29     29.00   1013.29      1.88             -
15.59
JUL             3.85     -4.52     -0.67     -0.29     50.00
34.70
AUG           396.69    693.30   1089.99     45.06             -
10.36
SEP          1000.08    335.55   1335.63     21.81             -
32.17
OCT            28.00    335.55    363.55     21.81     50.00     -
3.98
NOV           367.88   2037.98   2405.86    132.47           -
136.45
DEC                                0.00      0.00           -
136.45
           --------  --------  --------  --------  --------  -------
-

TOTALS      15461.56  34495.95  49957.51   2242.24   2105.79   -
136.45
```

5.13 -- Program Notes

In overall structure, this program is a simple loop that computes each value in the array once, then quits. For each computation, a GOSUB to the line number where the parameters are sets up which calculations are to be performed, and these are processed iteratively until there are no more, then the resulting value is rounded, stored into memory, and printed. In advancing to the next item, when the last column has been finished on any row, the row-end processing is done before beginning the next item on the new row.

Some of the complications of this program are due to the limited arithmetic capabilities of Tiny BASIC, and some to the generality of Tiny Calc itself. As with the other programs in this book, the notes here concentrate on the more obscure and complicated code. The simple parts are left to the reader as an exercise.

The user's application fills the lower line numbers of Tiny Calc (because they are easier to type than big numbers, and more mnemonic). The entire executive part of Tiny Calc is in the line numbers from 20000 to 24990. Program lines numbered 20xxx are concerned with initialization. Lines 21xxx are the main service loop, processing one item, then advancing to the next, until all have been processed. The rest of the program is organized as subroutines to support this main loop.

5.14 -- Variables

```
A - First operation, temp 100
B - Second operation
C - Third operation
D - Fourth operation
E - Fifth operation
F - Constant fraction
G - Decimal position
H - Constant high digits
I - Constant integer part
J - Temporary
K - Number of columns
L - Lower right corner (last)
M - Address of data block
N - Current row/column
O - Current operation code
P - Peek address
Q - Temporary (multiply/ divide)
R - Temporary (multiply/ divide)
S - Accumulator sign
T - Operand sign
U - Operand upper digits
V - Operand integer part
W - Operand fraction
X - Accumulator upper digits
Y - Accumulator integer part
Z - Accumulator fraction
```

5.15 -- Data Space

Tiny Calc uses four bytes of Poked memory for each computed value. This has to be memory not otherwise in use. The easiest thing to program here would require the user to set aside the memory and tell the program where it is. This is rather a nuisance, so I put the extra code in to allocate the space out of the GOSUB stack automatically. This way, if there is not enough memory for everything, a memory overflow error will occur (rather than perhaps trashing the program or data).

Some Tiny BASICs store their numbers low byte first. Given that I have to look at the stack pointer anyway, it is a small thing to tell which byte sex the machine is: do one GOSUB, then compare the old stack top with the new. A GOSUB pushes exactly two bytes, so the difference is either two (if I looked at it in the right order) or 512 (if I have the bytes swapped). Line 20240 makes this test.

Now, since this memory allocation process is quite slow, another optimization is included (line 20280); if the stack is already big enough, don't push any more. This would happen if instead of stopping with an END statement, the last statement of the main program (line 21790 and 22390) were something illegal, like "...". Then after printing the spreadsheet, Tiny BASIC would print some error number and stop' but you would know (by the line number) that it is not an error, and that next time you RUN it won't take so long to get going.

Otherwise, variable J is the number of bytes needed, and a GOSUB-to-self loop is executed, decrementing J by two until it hits zero. The new stack pointer at this point is the beginning of the data block. No RETURNs will ever be executed for these GOSUBs, so the data space is available.

5.16 -- Main Loop

When the program gets up to line 20000 it has already printed the heading. The next order of business is the label on the first row, which is done with a subroutine call to the appropriate line: N*10 will always point to the user specifications (label or item) under current consideration.

First the variables A-H and the accumulator are initialized. Variables B-E are left zero by the previous computation, so no further zeroing is needed. Then the user specifications for this item are fetched by another GOSUB. The user specifications supply the differences from the default values in these variables.

If variables A-E are not all zero, then this item has some calculations specified. If they are all zero, but H, I, and F are not all zero, then this is a data entry, and operation 001 is forced into variable A. Otherwise the calculation section is skipped and the zero in the accumulator is stored directly.

The calculations are performed in six bytes (three variables, plus sign), but only stored in four. This means that the calculated result must have its high and low

byte discarded. The subroutine at 24510 does a shift by two digits to effect this. The number is then rounded according to the specifications in variable G, and stored. The packing for storing the number is relatively simple; two digits per byte (binary 0-99, or seven bits) for the low four decimal digits, and then just cut the upper half of the number into bytes (but since there is no sign, it is limited to 32767, i.e. 15 bits). The sign of the number is packed into the third byte, extending its range to 0-199.

Printing the result (if G is not negative) is a little tricky. Only ten digit positions have been allowed, so if the result is a full nine digits (somewhere between 999,999.99 and 3,276,799.99; any higher gives an error stop), then when it is printed with two decimal places there is no room for a minus sign in front. I cheated. Since this is not an advertised capability of Tiny Calc, I decided to let you get away with it by printing the minus sign in place of the decimal point. But notice, two large numbers in adjacent columns will not leave any blank space for readability. This is why Tiny Calc is specified for eight digits only.

Line 21520 notices if the number is very large. Thus, for printing, S takes on one of four values: 0 for normal positive numbers, 1 for normal negative numbers, -1 for positive 9-digit numbers, or -2 for negative 9-digit numbers. The rest of the printing is straightforward.

Line 21740 checks for end of row. Line 21780 checks for end of run. This looks a little peculiar, since N is always a positive number, but L could be negative. If it is, it would have been nice to have an "absolute value" function; multiplying it times itself has the same effect, but gets too big if there are over 181 rows (182x182=33,124, which is bigger than 32767 and thus goes negative). Dividing by ten brings the product into range.

5.17 -- Calculations

In the main calculation routine, each successive variable is transferred to the current operation (variable O), and the variable it came from is cleared. The fields are unpacked (lines 22210-22290) and the operation selected is jumped to. Variable J takes on the row/column number of the operand for this, and if either row or column is zero (for operation digit less than 7), the current row or column is substituted. Operation 0 jumps to 22010, where the selected row/column parameters are fetched by a subroutine call, replacing the current values, and the whole calculation is restarted (except that the accumulator value is carried over).

The arithmetic is not overly obscure. The sum or difference is formed from the low order variable parts (the fractions) first, then carries (or borrows) are propagated to higher variables in the accumulator. By limiting the lower variable parts to four decimal digits each, there is room in the 15 (unsigned) bits of the variable to hold the full sum and its carry; the carry is of course stripped away after it has been

propagated to the next four digits. In the case of the difference, the result may have a different sign than was expected, so it must be recomplemented (one of the artifacts of sign-magnitude arithmetic) by subtracting it from zero (with borrow, which shows up as the "1" in 10000).

Choosing the maximum or minimum is a little sneaky. A normal subtract is done, then the operation is compared to the resulting sign by adding them:

Operation	Sign	Sum	Action
< (5)	+	5	Take new
< (5)	-	6	Take old
> (6)	+	6	Take old
> (6)	-	7	Take new

Since the old value was destroyed in taking the difference, it must be restored (if selected) by adding the new value back on.

All of the percent calculations simply multiply by the proper factor.

Multiplying is a pain, but not very complicated. The multiplier and multiplicand are each considered to be a string of two-digit parts. Each part of each number is multiplied times each part of the other number, and the partial products are lined up according to their relative decimal points. Some of multiplications are skipped because they have no effect on the result. For example the low half of Z times the low half of W gives a result that is necessarily less than 0.0001. Conversely the two high variables, if both greater than zero, will give an answer of at least 100,000,000 which is clearly an overflow condition.

Also, I ran out of variables to collect the answer in, so I had to build it on the fly, re-using the temporaries; this gives the code a little less clarity (sorry 'bout that). Note that from time to time the partial sums of the partial products run the risk of overflowing themselves, and carries must be propagated into higher parts of the result. Generally, three partial products can be summed before worrying about the carry out. The variable A is used here as an abbreviation for 100 (less bytes of program, and it executes faster).

You remember how they taught you to do "long division" in grammar school? Maybe they don't teach that any more; I just missed the so-called "new math" and had to learn it the old way -- I think a lot of schools are going back to it. Anyway, that is the method used for division. It is actually the same as the method used in computer division hardware, but in binary it is much easier. Divide routines like this are like Byron's sonnet (in his words): "When that was written, only God and the author understood it. Now only God understands it."

The general principle is that the division is done two digits at a time. A trial quotient part is chosen (in variable S; the sign has been saved in the GOSUB stack -- see below), and the divisor is multiplied times that and subtracted from the

remaining dividend. This is repeated while summing the quotient parts until the divisor is greater than the remaining divided. The sum of quotient parts becomes the next two digits of the quotient. Then the dividend and partial quotient is shifted left two digits and the process repeated. It quits when the decimal point of the quotient lines up.

There are some optimizations in the loop. If a whole variable is zero (a likely occurrence for small numbers), the divide step is bypassed, and the number shifted left four digits (lines 23790-23950).

If the multiply ran out of variables, the divide did so in spades. Variable A (the first operation) is known to be unused (after the first operation starts calculating). Variable O contained the operation code, now unused. Variable S normally contains the sign of the accumulator (watch this one); we calculate ahead of time whether the sign of the quotient is positive or negative, then do a GOSUB to the divide proper in such a way that when it returns we know (from where it came back to) what the sign must be. This amounts to keeping the sign in the GOSUB stack while we use the variable. The method is explained in the Experimenter's Kit, available from Itty Bitty Computers.

Fetching a previously computed value is just a matter of unpacking the number from the storage format. It needs to be shifted over two places to put it into the accumulator format.

The constant is not quite in the internal accumulator format, so it needs a little manipulation to add the parts together, extract the sign, and align the digits. Also the constant variables (H, I, and F) must be cleared so it can be used as zero a second time around.

5.18 -- Program Listing -- Defaults

```
2 REM TINY CALC DEFAULT (15X9) SETTINGS
3 LET L=159
4 PRINT
9 PRINT "CORNER",
10 PRINT " COLUMN 1. COLUMN 2. COLUMN 3.";
40 PRINT " COLUMN 4. COLUMN 5. COLUMN 6.";
70 PRINT " COLUMN 7. COLUMN 8. COLUMN 9.";
90 PRINT
100 PRINT "ROW 1",
105 RETURN
110 REM
119 RETURN
120 REM
129 RETURN
130 REM
139 RETURN
140 REM
149 RETURN
150 REM
```

```
159 RETURN
160 REM
169 RETURN
170 REM
179 RETURN
180 REM
189 RETURN
190 REM
199 RETURN
200 PRINT "ROW 2",
205 RETURN
210 REM
219 RETURN
220 REM
229 RETURN
230 REM
239 RETURN
240 REM
249 RETURN
250 REM
259 RETURN
260 REM
269 RETURN
270 REM
279 RETURN
280 REM
289 RETURN
290 REM
299 RETURN
300 PRINT "ROW 3",
305 RETURN
310 REM
319 RETURN
320 REM
329 RETURN
330 REM
339 RETURN
340 REM
349 RETURN
350 REM
359 RETURN
360 REM
369 RETURN
370 REM
379 RETURN
380 REM
389 RETURN
390 REM
399 RETURN
400 PRINT "ROW 4",
405 RETURN
410 REM
419 RETURN
420 REM
429 RETURN
430 REM
439 RETURN
440 REM
449 RETURN
450 REM
459 RETURN
```

```
460 REM
469 RETURN
470 REM
479 RETURN
480 REM
489 RETURN
490 REM
499 RETURN
500 PRINT "ROW 5",
505 RETURN
510 REM
519 RETURN
520 REM
529 RETURN
530 REM
539 RETURN
540 REM
549 RETURN
550 REM
559 RETURN
560 REM
569 RETURN
570 REM
579 RETURN
580 REM
589 RETURN
590 REM
599 RETURN
600 PRINT "ROW 6",
605 RETURN
610 REM
619 RETURN
620 REM
629 RETURN
630 REM
639 RETURN
640 REM
649 RETURN
650 REM
659 RETURN
660 REM
669 RETURN
670 REM
679 RETURN
680 REM
689 RETURN
690 REM
699 RETURN
700 PRINT "ROW 7",
705 RETURN
710 REM
719 RETURN
720 REM
729 RETURN
730 REM
739 RETURN
740 REM
749 RETURN
750 REM
759 RETURN
760 REM
```

```
769 RETURN
770 REM
779 RETURN
780 REM
789 RETURN
790 REM
799 RETURN
800 PRINT "ROW 8",
805 RETURN
810 REM
819 RETURN
820 REM
829 RETURN
830 REM
839 RETURN
840 REM
849 RETURN
850 REM
859 RETURN
860 REM
869 RETURN
870 REM
879 RETURN
880 REM
889 RETURN
890 REM
899 RETURN
900 PRINT "ROW 9",
905 RETURN
910 REM
919 RETURN
920 REM
929 RETURN
930 REM
939 RETURN
940 REM
949 RETURN
950 REM
959 RETURN
960 REM
969 RETURN
970 REM
979 RETURN
980 REM
989 RETURN
990 REM
999 RETURN
1000 PRINT "ROW 10",
1005 RETURN
1010 REM
1019 RETURN
1020 REM
1029 RETURN
1030 REM
1039 RETURN
1040 REM
1049 RETURN
1050 REM
1059 RETURN
1060 REM
1069 RETURN
```

```
1070 REM
1079 RETURN
1080 REM
1089 RETURN
1090 REM
1099 RETURN
1100 PRINT "ROW 11",
1105 RETURN
1110 REM
1119 RETURN
1120 REM
1129 RETURN
1130 REM
1139 RETURN
1140 REM
1149 RETURN
1150 REM
1159 RETURN
1160 REM
1169 RETURN
1170 REM
1179 RETURN
1180 REM
1189 RETURN
1190 REM
1199 RETURN
1200 PRINT "ROW 12",
1205 RETURN
1210 REM
1219 RETURN
1220 REM
1229 RETURN
1230 REM
1239 RETURN
1240 REM
1249 RETURN
1250 REM
1259 RETURN
1260 REM
1269 RETURN
1270 REM
1279 RETURN
1280 REM
1289 RETURN
1290 REM
1299 RETURN
1300 PRINT "ROW 13",
1305 RETURN
1310 REM
1319 RETURN
1320 REM
1329 RETURN
1330 REM
1339 RETURN
1340 REM
1349 RETURN
1350 REM
1359 RETURN
1360 REM
1369 RETURN
1370 REM
```

```
1379 RETURN
1380 REM
1389 RETURN
1390 REM
1399 RETURN
1400 PRINT "ROW 14",
1405 RETURN
1410 REM
1419 RETURN
1420 REM
1429 RETURN
1430 REM
1439 RETURN
1440 REM
1449 RETURN
1450 REM
1459 RETURN
1460 REM
1469 RETURN
1470 REM
1479 RETURN
1480 REM
1489 RETURN
1490 REM
1499 RETURN
1500 PRINT "ROW 15",
1505 RETURN
1510 REM
1519 RETURN
1520 REM
1529 RETURN
1530 REM
1539 RETURN
1540 REM
1549 RETURN
1550 REM
1559 RETURN
1560 REM
1569 RETURN
1570 REM
1579 RETURN
1580 REM
1589 RETURN
1590 REM
1599 RETURN
```

5.19 -- Program Listing -- Executive

```
1 REM TINY CALC -- COPYRIGHT(C) 1981 T.PITTMAN
2 REM L IS DIMENSION: RRC
3 LET L=11
4 PRINT
5 REM PRINT HEADING
90 PRINT
98 LET P=276
99 GOTO 20010
100 PRINT "ROW 1",
105 RETURN
110 REM
119 RETURN
```

```
20000
20000 REM INITIALIZATION
20010 LET A=0
20020 LET B=0
20030 LET C=0
20040 LET D=0
20050 LET E=0
20110 LET K=L-L/10*10
20130 LET N=10
20140 IF K<0 THEN LET K=-K
20150 IF K=0 THEN GOTO 20170
20160 IF L/10<>0 THEN IF L<2000 THEN IF L>-2000 THEN GOTO 20210
20170 PRINT "ROW/COLUMN?"
20180 END
20200
20200 REM COMPUTE DATA WORK SPACE (M)
20210 LET S=1
20220 LET M=USR(P,38)*256+USR(P,39)
20230 GOSUB 20240
20240 IF M=USR(P,38)*256+USR(P,39)+2 THEN LET S=0
20250 IF S=1 THEN LET M=USR(P,39)*256+USR(P,38)+2
20260 LET J=L/10*K*4+2
20270 IF J<0 THEN LET J=-J+4
20280 IF M+J<USR(P,34+S)*256+USR(P,35-S) THEN GOTO 21010
20310 LET J=J-2
20320 IF J>0 THEN GOSUB 20310
20330 LET M=USR(P,38+S)*256+USR(P,39-S)+2
21000
21000 REM MAIN CELL SERVICE LOOP
21010 IF L<0 THEN PRINT
21030 GOSUB N*10
21070 LET H=0
21080 LET I=0
21090 LET F=0
21110 LET N=N+1
21120 LET G=-2
21130 LET S=0
21140 LET X=0
21150 LET Y=0
21160 LET Z=0
21170 LET A=0
21180 GOSUB N*10
21190 IF A+B+C+D+E>0 THEN GOTO 21230
21200
21200 REM COMPUTE RESULT, ROUND, AND STORE IT
21210 IF I=0 THEN IF H=0 THEN IF F=0 THEN GOTO 21450
21220 LET A=1
21230 IF G=-2 THEN LET G=2
21240 GOSUB 22020
21250 LET U=X
21260 LET V=Y
21270 LET W=Z
21280 GOSUB 24510
21290 LET J=0
21310 IF G<0 THEN GOTO 21410
21320 IF G>1 THEN GOTO 21410
21330 IF G=1 THEN GOTO 21370
21340 LET J=(V+50)/100*100
21350 IF W=0 THEN IF V-V/200*200=50 THEN LET J=J-100
21360 GOTO 21430
21370 LET J=(V+5)/10*10
```

```
21380 IF W=0 THEN IF V-V/20*20=5 THEN LET J=J-10
21390 GOTO 21430
21410 LET J=V+W/5000
21420 IF W=5000 THEN IF V=V/2*2 THEN LET J=J-1
21430 LET X=U+J/10000
21440 LET Y=J-J/10000*10000
21450 LET J=(N/10*K+N-N/10*10-K-1)*4+M
21460 LET Z=USR(P+4,J,X/256)+USR(P+4,J+3,Y-Y/100*100)
21470 LET Z=USR(P+4,J+2,Y/100+S*100)+USR(P+4,J+1,X)
21500
21500 REM PRINT RESULT
21510 IF G<0 THEN GOTO 21720
21520 IF G>1 THEN IF X>9999 THEN LET S=-S-1
21530 IF X<10000 THEN IF G<2 THEN PRINT " ";
21540 IF X<1000 THEN PRINT " ";
21550 IF X<100 THEN PRINT " ";
21560 IF X<10 THEN PRINT " ";
21570 IF X<1 THEN PRINT " ";
21580 IF X=0 THEN IF Y<1000 THEN PRINT " ";
21590 IF S=0 THEN PRINT " ";
21610 IF S>0 THEN PRINT "-";
21620 IF X>0 THEN PRINT X;
21630 IF X>0 THEN IF Y<1000 THEN PRINT 0;
21640 PRINT Y/100;
21650 IF G=0 THEN GOTO 21730
21660 IF S=-2 THEN PRINT "-";
21670 IF S>-2 THEN PRINT ".";
21680 PRINT (Y-Y/100*100)/10;
21690 IF G>1 THEN PRINT Y-Y/10*10;
21710 GOTO 21740
21720 PRINT "        ";
21730 PRINT "  ";
21740 IF N-N/10*10<K THEN GOTO 21070
21750 PRINT
21760 GOSUB 5+N/10*100
21770 LET N=N+10-K
21780 IF N/10*N<L/10*L THEN GOTO 21010
21790 END
22000
22000 REM PERFORM A CALCULATION
22010 GOSUB J*10
22020 LET O=A
22030 LET A=0
22040 IF O>0 THEN GOSUB 22210
22050 LET O=B
22060 LET B=0
22070 IF O>0 THEN GOSUB 22210
22110 LET O=C
22120 LET C=0
22130 IF O>0 THEN GOSUB 22210
22140 LET O=D
22150 LET D=0
22160 IF O>0 THEN GOSUB 22210
22170 LET O=E
22180 LET E=0
22190 IF O<=0 THEN RETURN
22200
22200 REM DECODE ONE OPERATION
22210 LET J=O/10
22220 LET O=O-J*10
22230 IF O>6 THEN GOTO O*50+22260
```

```
22240 IF J<10 THEN LET J=J+N/10*10
22250 IF J=J/10*10 THEN LET J=J+N-N/10*10
22260 IF J<>N THEN IF O=0 THEN GOTO 22010
22270 IF J-J/10*10>K THEN GOTO 22330
22280 IF J<=N THEN IF O>4 THEN GOTO 23010
22290 IF J<=N THEN IF O>0 THEN GOTO O*400+22010
22300
22300 REM ERROR MESSAGES
22310 PRINT " ROW ";J/10;"?"
22320 GOTO 22360
22330 PRINT " COLUMN ";J-J/10*10;"?"
22340 GOTO 22360
22350 PRINT " SYNTAX"
22360 PRINT "ERROR IN CALCULATING ROW ";N/10;
22370 PRINT ", COLUMN ";N-N/10*10
22380 PRINT
22390 END
22400
22400 REM SUM
22410 GOSUB 24410
22420 IF S<>T THEN GOTO 22840
22430 LET Z=Z+W
22440 LET Y=Y+V+Z/10000
22450 LET X=X+U+Y/10000
22510 LET Y=Y-Y/10000*10000
22520 LET Z=Z-Z/10000*10000
22530 LET A=0
22540 IF X+Y+Z<>0 THEN RETURN
22550 LET S=0
22560 RETURN
22600
22600 REM ADD/SUBTRACT PERCENT
22610 LET W=(J-J/1000*1000)*10
22620 LET V=J/1000+1
22630 GOTO 22770
22660 IF J>999 THEN GOTO 22350
22670 LET W=10000-J*10
22680 LET V=0
22690 GOTO 22770
22700
22700 REM MULTIPLY PERCENT
22710 LET W=(J-J/1000*1000)*10
22720 LET V=J/1000
22770 LET U=0
22780 LET A=100
22790 GOTO 23250
22800
22800 REM DIFFERENCE
22810 GOSUB 24410
22830 IF S<>T THEN GOTO 22430
22840 LET Z=Z-W+10000
22850 LET Y=Y-V+9999+Z/10000
22860 LET X=X-U+Y/10000-1
22870 IF X>=0 THEN GOTO 22510
22880 GOSUB 22510
22910 LET S=1-S
22920 LET Z=10000-Z
22930 LET Y=9999-Y+Z/10000
22940 LET X=Y/10000-X-1
22950 GOTO 22510
23000
```

```
23000 REM CHOOSE MAX/MIN
23010 GOSUB 22810
23040 IF O+S=6 THEN GOTO 22420
23050 LET S=T
23060 LET X=U
23070 LET Y=V
23080 LET Z=W
23090 RETURN
23200
23200 REM MULTIPLY (BY PARTS)
23210 GOSUB 24410
23220 LET S=S+T
23230 IF S>1 THEN LET S=0
23240 IF X>0 THEN IF U>0 THEN GOTO 23580
23250 IF Y>=A THEN IF U>=A THEN GOTO 23580
23260 LET Q=Y/A*(V-V/A*A)+V/A*(Y-Y/A*A)
23270 IF W>0 THEN LET Q=Q+X/A*(W-W/A*A)+W/A*(X-X/A*A)
23280 IF U>0 THEN LET Q=Q+Z/A*(U-U/A*A)+U/A*(Z-Z/A*A)
23290 LET R=Y/A*(V/A)+Z/A*(U/A)+Q/A
23310 IF X>0 THEN LET R=R+X/A*(W/A)+(X-X/A*A)*(V-V/A*A)
23320 IF U>0 THEN LET R=R+(Y-Y/A*A)*(U-U/A*A)
23330 LET Q=(Q-Q/A*A)*A+(Y-Y/A*A)*(V-V/A*A)
23340 IF W+Z>0 THEN LET Q=Q+Y/A*(W/A)+Z/A*(V/A)
23350 IF W>0 THEN LET Q=Q+(X-X/A*A)*(W-W/A*A)
23360 IF U>0 THEN LET Q=Q+(Z-Z/A*A)*(U-U/A*A)
23370 IF Q<0 THEN LET R=R+3
23380 IF Q<0 THEN LET Q=Q-30000
23390 LET X=X/A*(V-V/A*A)+V/A*(X-X/A*A)
23410 IF U>0 THEN LET X=X+Y/A*(U-U/A*A)+U/A*(Y-Y/A*A)
23420 IF X>=A THEN GOTO 23580
23430 IF X>=0 THEN LET X=R+X*A
23440 LET J=0
23450 LET R=Z/A*(V-V/A*A)+V/A*(Z-Z/A*A)
23460 IF W=0 THEN GOTO 23510
23470 LET J=Z/A*(W-W/A*A)+W/A*(Z-Z/A*A)
23480 LET J=(J+50)/A+Z/A*(W/A)
23490 LET J=J+(Y-Y/A*A)*(W-W/A*A)
23510 LET J=J+(Z-Z/A*A)*(V-V/A*A)
23520 LET R=R+Y/A*(W-W/A*A)+W/A*(Y-Y/A*A)
23530 LET Q=Q+R/A+J/10000
23540 LET Z=J-J/10000*10000+(R-R/A*A)*A
23550 LET Y=Q+Z/10000
23560 LET U=0
23570 IF X>=0 THEN GOTO 22450
23580 PRINT " OVERFLOW"
23590 GOTO 22360
23600
23600 REM DIVIDE (CLASSICAL LONG DIVISION)
23610 GOSUB 24410
23620 IF U+V+W=0 THEN GOTO 23680
23630 IF S+T<>1 THEN GOTO 23710
23640 GOSUB 23710
23650 LET S=1
23660 RETURN
23680 PRINT " 0 DIVISOR"
23690 GOTO 22360
23710 LET Q=0
23720 LET R=0
23730 LET T=X/10000
23740 LET X=X-T*10000
23750 LET J=20
```

```
23760 IF U>2 THEN GOTO 23810
23770 GOSUB 24510
23780 LET J=J+2
23790 IF U<3 THEN GOTO 23770
23810 LET O=Q
23820 LET Q=R
23830 LET R=T
23840 LET T=X
23850 LET X=Y
23860 LET Y=Z
23870 LET Z=0
23880 LET J=J-4
23890 IF J<5 THEN GOTO 23960
23910 IF O>0 THEN GOTO 23960
23920 IF U>Q THEN GOTO 23810
23930 IF Q>U THEN GOTO 23960
23940 IF V>R THEN GOTO 23810
23950 IF V=R THEN IF W>T THEN GOTO 23810
23960 IF U>O THEN GOTO 24210
23970 LET S=O/(U+1)
23980 IF S=0 THEN GOTO 24150
23990 LET Z=Z+S
24010 LET R=R-(W-W/100*100)*S+20000
24020 LET A=W/100*S
24030 LET R=R-(A-A/100*100)*100
24040 LET Q=Q-(V-V/100*100)*S+19998-A/100
24050 LET A=V/100*S
24060 LET Q=Q-(A-A/100*100)*100+R/10000
24070 LET O=O-U*S-A/100-2+Q/10000
24090 LET R=R-R/10000*10000
24110 LET Q=Q-Q/10000*10000
24120 IF O>U THEN GOTO 23970
24130 LET S=0
24150 IF U>O THEN GOTO 24210
24160 IF Q<V THEN GOTO 24210
24170 IF Q=V THEN IF R<W THEN GOTO 24210
24180 LET S=1
24190 GOTO 23990
24210 LET A=100
24220 LET O=O*A+Q/A
24230 LET Q=(Q-Q/A*A)*A+R/A
24240 LET R=(R-R/A*A)*A+T/A
24250 LET T=(T-T/A*A)*A+X/A
24260 LET X=(X-X/A*A)*A+Y/A
24270 LET Y=(Y-Y/A*A)*A+Z/A
24280 LET Z=(Z-Z/A*A)*A
24290 IF J<4 THEN IF T>0 THEN GOTO 23580
24310 LET J=J-2
24320 IF J>0 THEN GOTO 23890
24330 LET A=0
24340 RETURN
24400
24400 REM FETCH VALUE FROM CELL
24410 IF J=N THEN GOTO 24610
24420 LET J=(J/10*K+J-J/10*10-K-1)*4+M
24430 LET U=0
24440 LET V=USR(P,J)*256+USR(P,J+1)
24450 LET W=USR(P,J+2)*100+USR(P,J+3)
24460 LET T=W/10000
24470 LET W=W-T*10000
24510 LET A=100
```

```
24520 IF U>300 THEN GOTO 23580
24530 LET U=U*A+V/A
24540 LET V=(V-V/A*A)*A+W/A
24550 LET W=(W-W/A*A)*A
24560 RETURN
24600 REM FETCH & CLEAR CONSTANT
24610 LET U=H
24620 LET V=I
24630 LET W=F
24640 LET T=1
24650 IF W>=0 THEN IF V>=0 THEN IF U>=0 THEN LET T=0
24660 IF W<0 THEN LET W=-W
24670 IF V<0 THEN LET V=-V
24680 IF U<0 THEN LET U=-U
24690 IF U>3000 THEN GOTO 23580
24710 LET A=100
24720 LET V=V+W/A+(U-U/10*10)*1000
24730 LET U=U/10+V/10000
24740 LET V=V-V/10000*10000
24750 LET W=(W-W/A*A)*A
24760 LET H=0
24770 LET I=0
24780 LET F=0
24790 RETURN
24990 END
```

Chapter 6 -- TINY ADVENTURE

Some computer games have complicated rules and need a great deal of skill to be enjoyed. Others are best played with as little introduction as possible. Tiny Adventure (TA) is in this latter category. Unfortunately, because it is only a tiny adventure game, there are some peculiarities you should know about.

The original Adventure game (available for most popular computers with disks) provided the inspiration for TA, but if you ever played the original game you will notice a number of significant differences. For one thing, your orientation in TA is significant, and you will not automatically see things off to one side or behind you -- you have to Look. Also, you can only hold one thing at a time in your hands. This is quite a nuisance when you want to open a locked door in a dark room, because you cannot hold both the lantern and the keys at the same time. And, unlike the original game, TA. keeps no score; you play for the pleasure of exploring, or set your own goals. For those achievement-oriented people like myself who need goals, there are some suggestions (read on).

Most of the instructions you need are given at the beginning of the game. My comments here are to clarify some common misunderstandings. TA has a very limited vocabulary, and it may be that you asked it to do something using words TA does not know, but that have the same initials as words it does know. The

result is that what TA did may not be what you expected at all. If you have any doubt, it usually does not hurt to take Inventory and to Look around.

Another common mistake (or maybe it is a failing in TA) is to Look Left then Look Right in order to get a panorama of the situation -- but it does not work that way, because when you Look in some (horizontal) direction, you become turned in that direction. Similarly, if you try to Go in some direction, you will usually get turned that direction, even if you cannot go that way.

One common complaint I've heard from several people who played this game is that it does not follow standard Euclidean geometry. That is not true. A map (on a flat piece of paper) was drawn of the area before a single line of code was written, and it is faithful to the map. What happens is that in crawling, climbing, or otherwise moving from one place to another, you got turned around, and the way out may not be behind you. Or, the divisions between places (such as rooms) may not fall on cartesian boundaries. This is true to life, and the game is consistent.

6.1 -- Help

The vocabulary in TA is small, and there is a (partial) list of the words it recognizes available to you at any point in the game. The list changes depending on circumstances (for example, if there is nothing nearby to Open, that word will not be listed). TA will print out the list if you type in a letter that is not recognized in the situation. Usually Help will cause the list to be printed out without complaining about an error.

Important: When you ask for Help, or if you typed an unrecognized letter and get a list of words to "CHOOSE FROM", you must retype the whole command. For example, if you type in GI (thinking "Go In"), TA will complain and give you the list of directions you can Go (and "In" is not one of them), then ask for a "COMMAND?". Do not now type a direction. If, after reviewing the directions you can Go, you still want to do so, type the G again, followed by the first letter of the direction. Of course, you can change your mind about going, and do something else instead.

6.2 -- Some Goals

The first time you play TA, you probably will just want to wander around and get comfortable, see what there is to see, learn where the keys and lantern are, what mistakes to avoid, etc. Then you might try to do some of the following (no fair figuring it out from the listing!):

Can you rescue the maiden and her jewels without killing the troll (leave him locked in his den)? What is the least number of turns to do this?

There are two ways into the dragon's lair, but you cannot get back out by one of them. Can you find it?

Can you discover what the "magic dragon tears" do for you? Can you undo it? Can you get more, after you use them up?

This is a hard one: If you get lost in the forest, can you get out? Hint: You need to head off in the direction of the ravine, but you must get your bearings before you get lost. Crashing through the underbrush of the forest tends to get you turned around, and you usually end up going around in circles.

Once you solve the forest problem, you might want to take the maiden on a moonlight boat ride around the island. Watch out for the riptide!

How many turns does it take you to visit every place? There are 17 places in all, counting both ends of the tunnel as one place. Usually you can tell you're in a different place if the scenery is different, or if something you Putdown is no longer visible.

The troll will under certain circumstances, wander around on his own. Can you coax him into the bedroom? Harder yet, can you lock him in the bedroom without the maiden being there to look on?

6.3 -- Configurator

There are three lines in Tiny Adventure that need to be configured for your system. All three are inset in the listing by two spaces after the line number instead of one.

Line 110 defines the "peek" USR address in your Tiny BASIC interpreter. This number is equal to the Cold Start + 20, and is shown in the listing as 276 (for the most common Cold Start address of 256). You should change this to correspond to your version of Tiny BASIC. This value is only used in lines 1660 and 1710 (see below).

Lines 1660 and 1710 depend on the "byte sex" of your machine. Both are concerned with getting the pointer to the input line buffer, in the one case to see if there is more input, in the other to stop the input line short. The listing shows both forms, with the normal form second. If you have a 6502 (which puts the least significant byte first), use the first form. The 6800 and 1802 use the second form.

The Z8 Tiny BASIC handles input slightly differently, and three lines need to be recoded:

```
1660 X=@^14
1710 X=0:INPUTX:GOTO1740
1730 INX
```

Half the fun of playing TA is in discovering what you can and cannot do. Much of this fun would be spoiled if the choices were obvious from the program listing, so it has been deliberately written in an obscure manner. Actually, that was only part of the reason - just trying to nuke it small accounts for much of the obscurity. There are, however, comments marking program section boundaries. If the program is entered into Tiny BASIC as printed, the comments will be eliminated, saving space. Or, you can omit typing them in. You can save some more memory space (771 bytes) by eliminating the instructions that print out at the beginning.

Because TA is so large (19,703 bytes), I found that execution became excruciatingly slow, simply due to the memory scan for GOTOs, GOSUBs, and RETURNs. A simple patch to the interpreter converts it to a binary search, for about an order of magnitude speedup in execution time. The necessary changes are listed in the Appendix.

A note for Z8 Tiny BASIC users. Unlike the other programs in this book, TA has already been crunched. Alas, Z8 Tiny crunches PRINT statements differently than the standard, so you will have to go through and change every occurrence of "PR" to a simple quote ('), and every occurrence of PR alone to a pair of quotes (""). Sorry about that.

You will need to fix the RND function calls for the Z8 as well. Ten of them can be fixed as indicated in the introduction (Chapter 1, section 1.4), but three work with large values. The following lines need to be replaced:

```
2270 IF(H/500+1)*@242>X/100*@242GOTO2410
4040 IF99*@242>PRETURN
7330 IFM+8=UIFN+7<>UIFM+7<>TIFN/300*@242<P/4GOTO7345
```

6.4 -- Program Notes

STOP!

Do not read any further before you play, or you will spoil the fun.

I am not going to tell you in this section what the map of the terrain is, but I would be remiss in my documentation if I did not give you enough information to figure it out from the listing. For you to do so, however, would spoil the fun. But once you have played the game out and tried everything there is to try, you may want to see if you can change the terrain or add objects or other citizens.

All of the features of the environment are encoded in the program code. Variables mostly retain state information: Where you are, which way you are pointing, where each object or citizen is, the state of the doors and windows, etc. If you want to stop a game in the middle and resume later, it is necessary only to note the contents of the variables and restore them to resume. The low two digits (decimal) of each

variable are used to decode the text input; but variables B, Q, V, X, and Z are exceptions to this rule.

Line numbers less than 1000 are strictly setup, and may be omitted to save a few hundred bytes (for whatever good that may do), provided that the variables are correctly initialized before the game starts.

From 1000 to 1999, the program consists of the main command loop and assorted utility routines. There are also some utilities scattered in between other sections of the program.

From line 2000 to 11500, every block of 500 line numbers represents one command code, positioned by the name of the command. Thus line 2000 corresponds to the letter "A" (for "Attack"), line 3000 is letter "C" (for "Close"), 3500 is "D" (for "Drink"), etc. up to line 11500, which is letter "T" (for "Take"). Non-existent commands jump to line 16384 (variable V), which reports the error.

From line 14000 to 30000, every block of 1000 line numbers represents one place in the environment (room or cave or region of the meadow, etc.). The places are effectively numbered from 14 to 30.

Each object or citizen (the variable whose name begins that object's name) maintains the place number in its thousands digits. Thus, variable R retains the location of the Rock, which may, for example, be in the ravine (place 14), so its value there is 14018. The hundreds digit is used only in variable Y (for "You"), where it records your orientation. The units and tens of course are the encoding of which variable it is (R is the 18th letter of the alphabet).

There are three implied places in the game, where an object can be. Place 1 is "in your hand"; place 2 is "in your knapsack"; and place 13 is "dead" (out of the game).

6.5 -- Variables

```
A - Location of the Axe
B - Composite record (bit mask) of seven booleans
C - Not used, except for input
D - Location of the Dragon
E - Not used
F - Location of the Flask
G - Not used
H - Temporary, to compute object in your Hand
I - Temporary
J - Location of jewels
K - Location of Keys
L - Location of Lantern
M - Location of Maiden
N - Orientation; free copy of Y (where you are Now)
O - Not used
P - Used to compute foe and state of light, etc.
```

```
Q - States of things that can be Open
R - Location of Rock
S - Location of Sword
T - Location of Troll
U - Location of you on previous turn
V - Constant 16384, to save space
W - Used to compute Where you are looking
X - Temporary used for input and certain bit masks
Y - Location and orientation of You (the player)
Z - Not used
```

The bit masks in variables B and Q are not fully utilized. Originally they were intended to be, but the program got too large, so some of the bits were disabled. The bits are given in the table in section 6.10, which also shows the bit positions in X for possible directions. The bit positions in X for things that can be Opened or Closed is the same as Q.

The bits in X, B, and Q are tested by multiplying the variable by an appropriate power of two, then seeing if the result is negative. This depends on the fact that overflow in multiplication simply wraps around, modulo 65536, without error.

6.6 -- Main Program

Lines 1000-1290 define the main program command processor. First the current environment is described. Variable W (the thousands digit) defines how much to describe: 1 is Front only; 2 is Right, Left, and Front; 3 is all four directions; and 0 is none of the above. There are several entry points into the beginning of the loop here, corresponding to whether a full panorama is required, if bit 12 in variable B needs to be cleared, or if no description of the place is required. Lines 1120-1160 step variable W through all the values down to 0023 to describe the scene; X here is used to hold the corresponding compass direction with the place number (i.e. the entry to the particular place or direction to be described).

After the full description, each of the other citizens is given a chance to act.

If the Dragon is awake, the subroutine at 6520 is called to determine whether he will change rooms, or go back to sleep. While awake, he meanders aimlessly between rooms 23, 24, and 25. If this results in his entering or leaving the place where you are, then notice is made of the fact ("enters" or "exits" if there is light to see; "scuffling" if not). If he enters the place where the maiden is, then she will respond in some way (her hand turns cold if you are holding it; if the troll is not present and you are not far away, then you hear her scream). If the dragon stays in the room where you are, then note is made of that fact (depending on whether there is light, it "fills your view" or "you hear breathing").

If the Troll happens to be in the place where the keys are, they are taken out of the game (sent off to place 13). If you are in the same room and there is light you are notified of the loss. The keys can only be recovered by killing the troll.

The troll also wanders about, though not always aimlessly. When the maiden is in the same place (note that "in your hand" is not the same as in the room you are in), then he makes a beeline for his den; the maiden follows unless you are in the room and it is light, and sometimes even then. If the maiden is not with him, the troll will wander about aimlessly throughout the enclosed areas (places 23-30), but in no case will he go through a closed door. When the troll enters, exits, or remains in a lighted room with you, you are notified of the fact.

Then in lines 7310-7390, the maiden has an opportunity to act on her own. If you are invisible and she is in the same room, mention is made of it. If she could see you and is in the place you just left (variable U) and the troll is not with her then she has a chance of following you out (better if there is light). Notification of this is made by the "footsteps". Finally, if you and she are both in the same place and there is light (and you are not invisible), she holds out her hand.

The actions of the three other citizens of the game go on independently of what you do. The technical word used for this kind of activity is "demon"; I don't particularly like the spiritual implications of the term , but it can't be much worse than when I used a "Diablo" (Spanish for "Devil") printer to print this book. Anyway, after the three demons are processed, you get prompted for another command, and a computed GOTO executes it.

6.7 -- Places

The seventeen actual places are divided into functions. Every place has responses for the six cartesian directions, plus a general introduction and an appendix. There are three entry points for the introduction, one for the appendix, and two for each cartesian direction. Each entry point is associated with a fixed line number value in the low three digits. Since the listing clearly identifies each place (in the REM header), there is no need to repeat that information here. The following notes deal only with the parallel entry points that are the same for all places.

Line xx010

> This is the name of the place. It is called as a subroutine whenever the game needs to tell you where you are.

Line xx040

> This is a subroutine that returns in variable X, a bit mask of the available places to Go (see Go command for details).

Line xx070

> This is a subroutine that returns in variable X, a bit mask of the available things that can be Opened or Closed. At most one of each category of thing to be opened or closed may be present in a given place. The Bedroom has

the maximum of one Door, one Window, and one Chest. For details, see the Close command.

Line xx110

This is a subroutine that prints what may be seen by Looking North.

Line xx140

This is not a subroutine. It takes the necessary action to cause you to Go North, and eventually jumps to the command processor.

Lines xx210, xx240

These two entry points are concerned with Looking and Going East.

Lines xx310, xx340

These two entry points are concerned with Looking and Going South.

Lines xx410, xx440

These two entry points are concerned with Looking and Going West.

Lines xx510, xx540

These two entry points are concerned with Looking and Going Up.

Lines xx610, xx640

These two entry points are concerned with Looking and Going Down.

Line xx910

This is a subroutine that may do or print anything associated with this place. Often it is used to draw attention to some object, such as the Sword in the forest. In some places, the objects there are listed at this time; in other places you are required to Look (usually down) to see them.

There are a few places that are not completely straightforward. Most of them can be deciphered by following the circuitous GOTOs and GOSUBs, and need no further comment. The remarks below deal with the hard ones.

Forest

In both forest places, every direction but out gets you lost. This includes both a random re-orientation, and projection into the middle of place 19. Two turns of correctly heading towards the ravine are necessary to get out of the inner forest, and another to actually get to the ravine. Of course, each of the first two turns got you all turned around, so you need to check your bearings again.

At Sea

You can get into the boat and head off in any direction. But if you do not stay on course towards known land, the same thing happens as in the forest. When you are lost at sea, the only back is to make a beeline for mainland (again, two turns required, and since it is dark out there, you also get a random re-orientation).

In an early version of the program, when things were Putdown in the boat in one place, then you moved to another place, the things did not follow, but they magically re-appeared if you went back. This is unrealistic, so there is a special call to a routine (line 4695) to move all the objects in the boat with you to the new place.

Between the island and the mainland, the boat can be in any of three places: beached on the mainland, at sea between, and beached on the island. Two bits in variable B encode these three states (bits 12 and 13). The states are ordered, so that a simple add or subtract of 4096 (V/4) correctly changes state.

Tunnel

The tunnel is divided into two regions with different scenery and things you can do -- it's almost as if they were two places. Bit 12 of variable B distinguishes them. If I had it to do over, I would probably make two places out of it.

Crashing into the walls anywhere underground wakes the dragon up (see line 25440).

6.8 -- Commands

There are eleven defined commands. The other 15 letters of the alphabet are rejected. I had intended to include three or four others, but space got out of hand so they were eliminated. One you might want to consider adding on your own is Break, since much of the structure is there for it (an extra bit in Q for each window and mirror, to show that it has been broken).

Each command must determine on its own criteria, whether it can be done or not. If it is not possible, a suitable error message must be printed out. If the command can be executed, then that is done, and the command routine should jump finally to the command processor to take the next command. There are several points to jump back to in the command processor, depending on whether a new place was entered (and must be described), of when an action was taken in the same place (and print "OK"), etc. This is why the exit from a command is a GOTO, not a RETURN.

In the notes that follow for each command, you may find it helpful to follow in the program listing, and where subroutines are called (there are many of them), it may

be necessary to refer to the descriptions of the subroutines (which come after the command descriptions).

Attack

> In general, the Attack command determines if there are any citizens around to be attacked, ranking them in order of villainy, and assigns a probability of being able to kill them (hundreds digit of variable P; thousands digits are the name of the foe). I suppose I could have had it ask you whom to attack. Oh. well. A second probability figure is computed for the weapon. If what is in your hand is not a weapon, the command aborts. If it seems unlikely that you really want to complete the attack (i.e. kill the fair young maiden), you are asked for confirmation.

On with the battle. The probability factor for the foe is extracted into variable X, and reduced to half if you are invisible (he is less likely to win if he can't see you). Two random numbers are generated, and if yours is bigger, you win, killing the foe. If you failed, the troll (if he was your foe) has a small chance of dumping your knapsack in the melee (with interesting consequences). You also have a certain probability of dropping your weapon (if you have one). If you win, of course your foe is sent off to place 13. Note that the dragon must be put to sleep when he dies, or he will creep back into the game and wander around (as he did in an early version).

Close/Open

> There is essentially one routine to deal with both opening and closing things, since they are basically the same kinds of operation, distinguished only by the sign of Q (negative for Open).

First it must be determined if there is anything to open or close. This involves a tricky little subroutine at 9310. Variable H holds a temporary value that has ones in those bit positions that can be opened or closed. If Q is positive, then it is used directly, because the command is to Close, and the ones in Q are those things that are open. If negative, then its complement is used. Note: I really want "one's complement", but that differs from the arithmetic negative in only the lowest non-zero bit, which is always bit 0 (and it does not hold useful data). Then X and H are shifted left together one bit at a time (lines 9350-9380) until the first non-zero bit comes up in X. The low bit of H is tagged (set to one), then if its bit (corresponding to the bit in X just tested) is also one, we are done. Otherwise the process is repeated until we run out of bits in X. On exit, both H and X are negative if there is something to open or close. H is odd if any bits in X were passed, that is, we tested them. X is not exactly -32768 if there are more things we did not look at.

When all is said and done, if there is more than one thing to open or close in the place, then it is necessary to ask which one. Otherwise, the command can just

proceed with the one item. This is determined by looking at the remaining bits in X (the things not tested) and the low bit in H (things that were tested). If clarification is needed, and the input line is empty, more input is requested. Then another letter is read. It must be "C", "D", or "W", which selects a range of bits to be examined from X (and expecting only one bit in that range). The range is selected by remaindering (which is what you must do if you do not have a logical AND operation).

Now it is but a simple matter to test whether that bit in variable Q is one or zero, check the sign of Q (the two should be different), and perform the appropriate action (turn bit on or off). Of course, if the command was to Open, and if the selected bit is a door or the troll's chest, the keys must be in your hand to complete the command. Also, if you opened the troll's chest and you are visible, the troll is in the room, and there is light to see, the troll quickly closes the chest again.

Drink

This one is fairly simple. If the flask is in your hand and not empty, then it is emptied and you are set to be invisible, with appropriate response from the maiden (if she is in the same place and it's not dark).

Go

The Go command is not very complicated. A second letter is required to specify direction, and if the input line is empty, a prompt is issued. All 23 possible inputs are considered, and only a few are rejected. The selection is computed on the value of the input (in X) with a GOSUB into the range 5100-5330. The returned value is (in the hundreds digit) the correct cartesian direction to go. Relative directions are computed from the difference between Y (which has your current orientation) and N (that has a zero in that digit), by subtractinging an offset (100 for each 90 degrees of turn to the left), and if the value goes negative, normalizing it. If the direction is one of the four compass points, your orientation is set to match. Finally, the command jumps off to the respective entry point in the place code to do it.

Help

The only difference between Help and an error is that Help does not ask "WHAT?". The choices given are the same. First, if it is dark, several important commands are not mentioned (you cannot see to do them). Otherwise, if there is a potential foe around, Attack is mentioned. If something can be closed, Close is mentioned. If the flask is in your hand and not empty, Drink is mentioned. You always get Go, Help, and Inventory. Look, if not dark. Open is like close, but the sign on Q has to be toggled. If there is something in your hand, Keep and Putdown are mentioned (but you cannot Keep the maiden, so that is excepted). If there is nothing in your

hand but your knapsack has something, or there is some object in the room you can take, then Take is mentioned.

Inventory
: This one is pretty obvious.

Keep
: This is also pretty much obvious.

Look
: Look works something like Go. If there is no second letter waiting, prompt for it, then a computed GOTO on the input letter selects a cartesian direction to point you. For Up and Down, a GOSUB to the appropriate entry in the current place (N-496 or N+596) does it. Otherwise, you are pointed in the selected direction and the command processor displays what you see.

Put
: With only four lines, what is there to say?

Take
: If your hands are full, you cannot take anything. Otherwise if the input line is empty, another input is prompted. As in Look and Go, a computed GOSUB based on the input letter selects the appropriate action. If the letter names no object, that is an error. If a known object is named, then it must either be in the knapsack or in the place, or that is also an error. If the jewels are mentioned, they must not only be available, but if they are in either the bedroom or the troll's den, then they cannot be taken if the chest is closed. The keys work the same way in the bedroom only (since they are needed to unlock the troll's chest, it would not do to force them into it).

6.9 -- Subroutines

Tiny Adventure is full of little subroutines that do obvious little things. For example, line 1380 is a one-line subroutine to print a short message. Making it a subroutine saved a few bytes by eliminating its duplication in the several places it is used. The remarks to follow deal with the routines whose function or operation is less than obvious.

Line 1480
: This is one of several routines to set or clear a bit in variable B. This one clears bit 12. The bit is tested by shifting it over to the sign position by multiplying by the appropriate power of 2; in this case, a 3-bit shift is done by multiplying by 8, which is 2^3. If the bit is one (the product is negative), it can be cleared by subtracting the appropriate power of 2 (for bit 12, subtract V/4=4096=2^12).

Line 1650

This routine examines the input line to see if there is any more input on it. X is returned zero if not. This is used to decide whether to prompt for more input.

Lines 1700-1840

This routine sets the input line to empty, and reads a value, which is assumed to be the value of a letter-variable. Because variable B is a bit vector, not a decimal string as the others, it must be treated separately. Similarly, X is set to zero, so that an input of "X" will read a well-defined value, namely zero. If the input is "Y", this also is converted to zero. All other letters, A-W, are stripped of the higher digits so that the value returned in X is a number between 0 and 23. Any value outside this range is rejected, which is signified by a returned value of -1. A second entry point is line 1720, which accepts any remaining (previous) input.

Line 1870

This routine calls on the input routine to get a yes or no answer. If not one of these, the message in line 1860 is printed to accept another input.

Lines 2700-2880

There are several situations where another citizen moves into or out of the same place as you, so this routine relates the adjacent room involved to your orientation to specify which relative direction that is. For example, if you are in the Dragon's lair, there is only one connecting passageway, to the south (hundreds digit 3 in variable P). Otherwise, if you are not in the cave (place 23), then if the adjacent room is 24 (the tunnel) then the direction must be east (2), since that is the direction to the tunnel from both the troll's den and the wine cellar (see line 2750). After all the options have been exhausted, P contains not only the place number of the adjacent place (that it started with), but also a direction. The difference between this and variable Y reflects the relative direction of that passageway, and is used to compute a GOTO (line 2810) to select the appropriate print statement.

Line 2900

This routine determines if the chest to be considered is the one in the bedroom, and if so, if the keys are also in the room ("yes" sets X to 1). Then if either the keys or jewels are in it, that is printed out. Finally, the fact that the chest is open is printed. If the chest is not open, then of course its contents cannot be seen, so this subroutine is not called. Line 2970 is also used to print the word "OPEN" associated with the command of that name.

Lines 3930-3990

This short routine is used in the forest and out in the open sea to get you disoriented. It also clears bit 12 in variable B, so that it will still take two turns to get out.

Lines 4030-4590

This routine is concerned with selecting the next room the troll will be in. On entry, the thousands digit of P has a probability figure to determine if any change is likely. Then, depending on where he is (in X), a computed GOTO into the range 4110-4180 selects the room parameters, where there may be a choice of directions (selected randomly), or if a door is closed the troll must have the keys to go through. A second entry point (at line 4060) uses a different selection algorithm, namely, the fastest route to his den, by computing a GOTO in the range 4520-4590.

Line 4610

This prints "CAN'T".

Line 4630

If it is dark, it prints "YOU SEE NOTHING". Otherwise it jumps to the line whose number is in variable X to print what you see.

Line 4660

This routine is called if the troll has bested you in battle. Then, if the knapsack is not empty (X=0) its contents are dumped out into the room (see following routine).

Lines 4695-4790

For each object that is in the same location specified by variable P, this routine moves them into the place where you are (variable N). It is used to dump the knapsack out onto the floor (P=2016), and to move objects in the boat from one place in the sea to the other (P is the place you came from).

Line 4840

This routine prints out that you cannot do something. Variable X contains the line number with the print statement of what it is you cannot do, and that is called as a subroutine after printing "CAN'T". Finally, it jumps back to the command processor to accept another command. The entry at line 4830 also sets bit 12 in variable B to one.

Lines 4920-4990

This routine checks for sufficient light to see. The answer is yes if P is greater than N, and no if less. In general there is enough light to see if the lantern is in your hand or in the room, or if you are not underground and not lost at sea. A second entry point at line 4910 answers yes only if the troll is

also in the same room with you, and is used to select actions that need you to see the troll.

Line 5600

This routine takes a bit mask in variable X corresponding to the cartesian (compass) directions, and prints out the the corresponding words, relative to the direction you are facing. The non-relative directions are tested (by shifting the respective bit into the sign with a multiply) and printed directly. The computed GOTO in line 5690 selects one of the following four lines, depending on the direction you are facing, which shifts X 0-3 bits left (by adding X to itself). The bits have been duplicated (in line 5680) so that what was originally in bit positions 1,2,3,4 is now also in 5,6,7,8 thus:

```
Bit Position  ......9876543210
Original      ..........ABCD.
Final         .......ABCDABCD.
```

Line 6210

This routine tests to see if the knapsack is empty. If any object is in the knapsack, then its location (the thousands digits) will be 2; subtract 2, and multiply times the rest of this stuff, and the answer is zero. If nothing is in the knapsack, then all these values will not be two, and X will be the product of seven numbers none of them zero, so it should not come out zero. Now, actually, this is not a perfect test. If you have four things in the forest (place 18), then you will multiply 16 times itself four times, which is 65536, which Tiny BASIC finds indistinguishable from zero. Similarly, five things in the troll's den (place 26), and at least one other thing in any even-numbered place, will do the same thing (five multiples of eight, times one multiple of two, gives a multiple of 65536). However, if this failure occurs, you will be told something like "you have kept" instead of "you have nothing kept". It is not that serious a problem, but if you wanted, you could fix it by re-coding it to work like the room test (lines 6240-6280).

Line 6240

This routine tests for the presence of an object in the same room with you, and if so, variable X is returned zero. If there is nothing there with you, X is returned non-zero (the location of the maiden). Thus a single IF condition on X can test objects other than the maiden (X=0) or objects including the maiden (X*(N-X-1)=0).

Line 6440

This routine tests the contents of the knapsack, then prints either that list, or the word "nothing".

Lines 8520-8590

This prints the tunnel scene. If it is dark, of course "you see nothing." If variable X is zero, then there is nothing to see in that direction, so you see "a dank wall". Otherwise the direction is selected by the bit in X (Up, Down, and neither=Off).

Line 8950

There are many many ways to look where there is nothing to see but a plain wall. I tried to liven it up with some (otherwise useless) descriptive adjectives. The line number of the print statement for the adjective is in variable X, and it is called as a subroutine at the appropriate place in the phrase.

Lines 9770-9890

This is a very useful routine for finding out what object (if any) is in your hand. H is returned with the object number (the ordinal value of its alphabet letter name) in the thousands digits, or if your hand is empty, with zero there. There is no convenient way to get this information but by testing each object to see if it is in your hand.

Line 10100

This is actually eight subroutines that are normally called with a computed GOSUB, indexed on the name of the object (that is, line number 10100 plus 20 times the letter value). Only the selected object is moved. Variable P contains the destination (room number, or knapsack or hand).

Lines 12490-12965

This is a set of sixteen subroutines that are normally called with a computed GOSUB on the letter value of the object whose name is to be printed. Each subroutine has two entry points, one for the simple word (at line number 12500 plus 20 times letter value) and an entry (line number less 5) that prints a more descriptive phrase. Zero is allowed as a letter value in this set, and refers to empty hands (used in Attack as a weapon name).

Lines 13540-13730

This is three routines, with indexed entry points corresponding to the three other citizens of the land. The selected routine is called with a computed GOSUB (line number 13500 plus 10 times letter value) from Attack, when the selected party is slain in battle. Note (line 13540) that when the dragon is killed he is also put to sleep. When the troll is killed, if he has the keys (line 13720), they fall on the floor.

Lines 13770-13990

This subroutine prints all the objects in the place specified by variable P. There are several entry points: Line 13770 assumes the place where you are

is desired, and a full description of the objects is to be printed; line 13810 asks for a short description (used when P is the knapsack, etc.). Each object is tested to see if its place is the same as P, and if so, it is printed.

Some objects, notably the keys and jewels, have restrictions. These are in the chest if they are in the troll's den or the bedroom, so for a room description they are not printed (this is distinguished by X=0 if not the room description). If this is to be a list of objects that can be Taken, then the respective chest must be open. Line 13860 is perhaps the most obscure of these tests. If P is the bedroom, then P/29000 is 1 and the product is (1+1)*64*Q, which tests the open bit for the bedroom chest; otherwise the product is 64*Q to test the troll's chest.

6.10 -- Bit Vector Table

Bit	Value	Variable B	Variable Q	Variable X
0	1	Always 1	Always 1	Not used
1	2	Invisible	Not used	North
2	4	Not used	Bedroom Window	East
3	8	Not used	Not used	South
4	16	Dragon awake	Mirror over table	West
5	32	Not used	Not used	Up
6	64	Not used	Troll's mirror	Down
7	128	Flask empty	Not used	Not used
8	256	Flask described	Chest in bedroom	Not used
9	512	Troll described	Troll's chest	Not used
10	1024	Not used	Outside door	"Off"
11	2048	Not used	Bedroom door	Not used
12	4096	Alternate position	Wine cellar door	Not used
13	8192	Boat at island	Troll's door	Not used
14	16384	Not used	Not used	Not used
15	-32768	Always 1	1 if in Open	Not used

6.11 -- Program Listing

```
100 REM TINY BASIC ADVENTURE GAME COPYRIGHT (C) 1981 T.PITTMAN
110 Z=276
180 V=16384
210 A=23001
220 B=-24575
230 C=3
240 D=25004
250 E=5
260 F=30006
270 G=7
```

```
280 H=8
290 I=9
300 J=26010
310 K=29011
320 L=30012
330 M=26013
340 N=14
350 O=15
360 P=16
370 Q=2129
380 R=14018
390 S=18019
400 T=26020
410 U=21
430 W=23
440 X=24
450 Y=15325
510 PR"WELCOME TO TINY ADVENTURE!"
520 PR"YOU ARE FREE TO WALK AROUND AND"
530 PR"LOOK AT WHAT YOU MAY SEE."
540 PR"THERE ARE OTHER INHABITANTS,"
550 PR"AND VARIOUS OBJECTS YOU CAN PICK UP"
560 PR"AND CARRY. YOU CAN ONLY HOLD ONE"
570 PR"THING AT A TIME IN YOUR HANDS,"
580 PR"BUT YOU CAN KEEP SEVERAL THINGS"
590 PR"IN YOUR KNAPSACK. BE CAREFUL AS YOU"
610 PR"GO FROM PLACE TO PLACE -- IT IS"
620 PR"EASY TO GET TURNED AROUND OR LOST."
630 PR
640 PR"I'M SLOW, BUT AFTER I TYPE '?',"
650 PR"YOU TELL ME WHAT YOU WANT TO DO."
660 PR"PLEASE TYPE ONLY THE FIRST LETTER"
670 PR"OF EACH IMPORTANT WORD, AND LEAVE"
680 PR"THE OTHER WORDS OUT."
690 PR
750 PR"FOR HINTS ON WHAT YOU CAN DO,"
760 PR"TYPE H (FOR HELP). HIT 'RETURN'"
770 PR"KEY TO ENTER YOUR SELECTION."
780 PR
790 PR"HAPPY HUNTING!"
880 PR"OK";
890 GOSUB1710
900
900 REM REMOVE TO HERE FOR MORE MEMORY SPACE
900
1000 REM DESCRIBE SURROUNDINGS
1000
1010 W=3023
1020 GOTO1060
1050 GOSUB1480
1060 PR"YOU ARE IN A ";
1070 U=N+7
1080 N=Y/1000*1000+14
1090 GOSUBN-4
1110 IFW<99GOTO1180
1120 X=Y-W/10+87
1130 IFX<NX=X+400
1140 GOSUBW/10+1208
1150 GOSUBX
1160 IFW>99GOTO1120
1165 GOSUBN+896
```

```
1170
1170 REM PROCESS DEMONS
1170
1180 IFN<23000IFB*V<0B=B-2
1185 IFV/8*B<0GOSUB6520
1190 IFT>14000GOSUB3610
1195 IFM>14000GOSUB7310
1200
1200 REM ACCEPT NEXT INPUT LINE
1200
1210 H=8
1230 P=16
1270 GOSUB1710
1280 IFX>0IFX<21GOTOX*500+1510
1290 GOTOV
1300
1300 REM PANORAMA SETUP & SMALL PRINTS
1300
1310 W=23
1320 PR"IN FRONT OF";
1330 GOTO1530
1340 PR"ON YOUR ";
1350 RETURN
1380 PR"FLOWERS BLOOM"
1390 RETURN
1410 W=4023
1420 GOSUB1340
1430 PR"LEFT ";
1440 RETURN
1450 PR"GO THRU WALL"
1460 RETURN
1480 IFB*8<0B=B-V/4
1490 RETURN
1510 W=2023
1520 PR"BEHIND";
1530 PR" YOU ";
1540 RETURN
1550 PR"OK";
1560 GOTO1870
1610 W=1023
1620 GOSUB1340
1630 PR"RIGHT ";
1640 RETURN
1650
1650 REM TEST END OF INPUT LINE (0= YES)
1650
1660   X=USR(Z,USR(Z,46))-13
1660   X=USR(Z,USR(Z,47))-13
1670 RETURN
1700
1700 REM INPUT SUBROUTINES
1700
1710   X=USR(Z+4,USR(Z,46),13)
1710   X=USR(Z+4,USR(Z,47),13)
1720 X=0
1730 INPUTX
1740 IFX=BX=2
1760 IFX<1GOTO1810
1770 IFX=YX=0
1780 X=X-X/1000*1000
1790 IFX<24RETURN
```

```
1810 PR"WHAT?"
1820 PR"CHOOSE FROM:"
1830 X=-1
1840 RETURN
1850
1850 REM INPUT YES/NO ANSWER
1850
1860 PR"YES NO"
1870 GOSUB1710
1880 IFX*(X-14)<>0GOTO1860
1890 RETURN
1910 PR"THERE IS NOTHING ";
1920 RETURN
1930 PR"ATTACK ";
1940 RETURN
1950 PR"YOU HAVE ";
1960 RETURN
1970 PR"IN YOUR HAND"
1980 RETURN
2000
2000 REM ATTACK PROCESSOR
2000
2010 GOSUB2340
2020 IFP>99GOTO2110
2030 X=1930
2040 GOTO4840
2060 IFH>99H=H+4000
2070 IFH/2000=11H=H+H/23000*1000-21000
2080 IFP=4916IFH>4000H=H-2000
2090 RETURN
2110 GOSUB1930
2120 GOSUBP/1000*20+12500
2130 PR"WITH ";
2140 GOSUB9770
2150 GOSUBH/50+12500
2160 GOSUB2060
2170 IFH<6000GOTO2210
2180 PR"IS SILLY"
2190 GOTO1210
2210 IFH>99IFP<>13116GOTO2250
2220 PR
2230 GOSUB1550
2240 IFX>0GOTO1210
2250 GOSUB2480
2260 X=P-P/1000*1000
2266 IFB*V<0X=X/2
2270 IFRND(H/5+99)>RND(X)GOTO2410
2280 IFP=20416IFRND(9)>6GOSUB4660
2285 IFH<99GOTO7070
2290 IFRND(9)>3GOTO7070
2310 GOSUB9610
2320 PR"YOU DROPPED WEAPON"
2330 GOTO1180
2340 P=16
2350 IFM+1=NP=13116
2360 IFD+10=NP=4516
2370 IFT=N+6P=20416
2380 IFD+10=NIFV/8*B<0P=4916
2390 RETURN
2410 GOSUBP/1000*10+13500
2420 PR
```

```
2430 GOTO1060
2480 IFY<28000IFY>23000IFV/8*B>0B=B+16
2490 RETURN
2500
2500 REM NO B
2500
2510 GOTOV
2700
2700 REM PRINT DIRECTION OF MOVE
2700
2710 P=P/1000
2720 IFN=25014P=25316
2730 IFN=23014P=23416
2740 IFP=28IFN=27014P=27116
2750 IFP=24P=N+202
2770 IFN/28000>0P=N+302-N/30000*100+P/30*300-P/29*200
2780 IFN=24014P=24316-P/25*100+P/26*200-P/27*300
2790 IFP/1000=24IFB*8<0P=24416-P/24200*100
2810 GOTO(Y-P+28491)/10
2820 GOTO1420
2830 GOTO1520
2840 GOTO1620
2850 GOTO1320
2860 GOTO1420
2870 GOTO1520
2880 GOTO1620
2900
2900 REM PRINT CHEST CONTENTS
2900
2910 IFX=128IFK=N-3X=1
2920 IFX<>1IFJ<>N-4GOTO2960
2930 PR"WITH ";
2940 IFX=1GOSUB12715
2945 IFJ=N-4GOSUB12695
2950 PR"IN IT ";
2960 PR"IS ";
2970 PR"OPEN ";
2980 IFX>0PR
2990 RETURN
3000
3000 REM OPEN/CLOSE PROCESSOR
3000
3010 GOSUB9310
3020 IFX=0GOTO9110
3025 IFH<0H=H+V+V
3030 X=X+H-H/2*2
3035 H=0
3040 IFX+X=0GOTO3150
3050 GOSUB1660
3060 IFX>0GOTO3080
3070 GOSUB9260
3075 PR"WHAT";
3080 GOSUB1720
3090 H=0
3110 IFX=4H=1024
3120 IFX=3H=256
3130 IFX=23H=1
3140 IFH=0GOTO9210
3150 GOSUBN+56
3160 IFH>0X=X/H*H
3170 IFH=256X=X-X/1024*1024
```

```
3180 IFH=1X=X-X/256*256
3190 IFX=0GOTO4610
3210 IFV/X*2*Q<0GOTO3340
3220 IFQ<0GOTO3250
3230 PR"ALREADY ";
3235 GOSUB3360
3240 GOTO4860
3250 IFX>256IFK<>1011GOTO3310
3255 IFX=512IFB*V>0GOTO3450
3260 Q=Q+X
3270 GOSUB3380
3280 GOTO7070
3310 GOSUB8390
3320 GOSUB3370
3330 GOTO1180
3340 IFQ<0GOTO3230
3350 Q=Q-X
3355 GOTO7070
3360 GOSUB9260
3370 IFQ>=0RETURN
3380 Q=Q+V+V
3390 RETURN
3410 X=8650
3420 GOTO4840
3450 GOSUB4910
3455 IFP<NGOTO3260
3460 PR"TROLL SEES YOU, CLOSES CHEST AGAIN"
3470 GOTO3320
3480 IFB*8>0B=B+V/4
3490 RETURN
3500
3500 REM DRINK PROCESSOR
3500
3510 IFF=1006IFB*256>0GOTO3550
3520 X=8610
3530 GOTO4840
3550 B=B+128
3560 IFB*V<0GOTO7070
3565 B=B+2
3570 GOSUB4920
3580 IFP>NIFM+1=NPR"MAIDEN LOOKS & GASPS"
3590 GOTO7070
3600
3600 REM TROLL PROCESSOR
3600
3610 GOSUB4910
3620 IFK+9<>TGOTO3650
3630 K=13011
3640 IFP>NPR"TROLL TAKES KEYS"
3650 IFM+7=TGOTO5810
3660 X=T
3670 P=8016
3680 IFM=1013P=2016
3690 GOSUB4040
3710 GOSUB4920
3720 IFP<NGOTO3850
3730 IFX=TGOTO3880
3740 IFX-6=NGOTO3810
3750 IFT-6<>NGOTO3850
3760 P=X
3770 GOSUB8630
```

```
3780 GOSUB2710
3785 PR
3790 GOTO3850
3810 P=T
3820 GOSUB2710
3830 PR"TROLL ENTERS"
3850 T=X
3860 RETURN
3880 IFT-6=NPR"TROLL SCOWLS"
3890 RETURN
3930
3930 REM ROTATE DIRECTION RANDOMLY
3930
3950 GOSUB1480
3960 Y=N+111+RND(4)*100
3990 GOTO1060
4000
4000 REM NO E
4000
4010 GOTOV
4030
4030 REM CHANGE ROOM FOR TROLL
4030
4040 IFRND(9999)>PRETURN
4050 GOTO4410-X/100
4060 GOTO4290+X/100
4080 X=26020
4090 RETURN
4110 GOTO4220
4120 GOTO4210
4130 GOTORND(3)*20+4420
4140 GOTORND(2)*20+4220
4150 IFQ*4>0GOTO4250
4160 GOTO4180
4170 GOTORND(4)*20+4320
4180 X=24020
4190 RETURN
4210 IFQ*16>0IFK<>13011RETURN
4220 X=28020
4230 RETURN
4240 IFQ*8<0GOTO4180
4250 IFK=13011GOTO4180
4260 RETURN
4320 X=23020
4330 RETURN
4340 X=25020
4350 RETURN
4360 IFQ*4>0IFK<>13011RETURN
4370 GOTO4080
4380 IFQ*8>0IFK<>13011RETURN
4420 X=27020
4430 RETURN
4440 X=30020
4450 RETURN
4460 IFQ*16>0IFK<>13011RETURN
4470 X=29020
4480 RETURN
4500
4500 REM NO F
4500
4510 GOTOV
```

```
4520 GOTO4180
4530 GOTO4360
4540 GOTO4180
4550 RETURN
4560 GOTO4240
4570 GOTO4420
4580 GOTO4210
4590 GOTO4220
4600
4600 REM TROLL DUMPS KNAPSACK
4600
4610 X=1670
4620 GOTO4840
4630 GOSUB4920
4640 IFP<NGOTO20110
4650 GOTOX
4660 PR"HE HITS YOU"
4670 GOSUB6210
4680 IFX=0PR"& SPILLS KNAPSACK"
4690 P=2016
4695 IFM=P-3M=N-1
4710 IFA=P-15A=N-13
4720 IFF=P-10F=N-8
4730 IFJ=P-6J=N-4
4740 IFK=P-5K=N-3
4750 IFL=P-4L=N-2
4760 IFR=P+2R=N+4
4770 IFS=P+3S=N+5
4790 RETURN
4800
4800 REM NO CAN DO
4800
4810 X=8390
4820 GOTO4840
4830 GOSUB3480
4840 GOSUB7160
4850 GOSUBX
4860 PR
4870 GOTO1210
4880 GOSUB1480
4890 GOTO4840
4900
4900 REM CHECK IF LIGHT ENOUGH TO SEE
4900
4910 IFT-6<>NGOTO4970
4920 P=N+2
4930 IFL+2=NRETURN
4940 IFL=1012RETURN
4950 IFN<24000IFN<>20014RETURN
4960 IFN>28000RETURN
4970 P=13016
4990 RETURN
5000
5000 REM GO PROCESSOR
5000
5010 W=1023
5020 GOSUB1660
5025 IFX=0PR"GO WHERE";
5030 GOSUB1720
5035 IFX>=0GOSUBX*10+5100
5040 IFX<0GOTO5060
```

```
5045 X=X+N
5050 IFX<N+500Y=X-15
5055 GOTOX
5060 X=0
5065 GOSUBN+26
5070 GOSUB5620
5075 PR
5080 PR"COMMAND";
5090 GOTO1210
5100 REM
5110 GOTO1810
5120 X=Y-N-185
5125 GOTO5350
5130 GOTO1810
5140 X=626
5145 RETURN
5150 X=226
5155 RETURN
5160 X=Y-N+15
5165 GOTO5350
5170 GOTO1810
5180 GOTO1820
5190 REM
5200 REM
5210 GOTO1810
5220 X=Y-N-85
5225 GOTO5350
5230 GOTO1810
5240 X=126
5245 RETURN
5250 REM
5260 REM
5270 GOTO1810
5280 X=Y-N-285
5285 GOTO5350
5290 X=326
5295 RETURN
5300 GOTO1810
5310 X=526
5315 RETURN
5320 GOTO1810
5330 X=426
5335 RETURN
5350 IFX<99X=X+400
5360 RETURN
5500
5500 REM HELP PROCESSOR
5500
5510 GOSUB1820
5520 GOTO13110
5600
5600 REM PRINT AVAILABLE DIRECTIONS
5600
5610 IFX*32<0PR"OFF ";
5620 IFX*1024<0PR"UP ";
5630 IFX*512<0PR"DOWN ";
5670 IFX<0RETURN
5680 X=(X-X/32*32)*17
5690 GOTO(Y-N)/10+5699
5710 X=X+X
5720 X=X+X
```

```
5730 X=X+X
5740 IFV/8*X<0PR"FORWARD ";
5750 IFX*512<0PR"BACK ";
5760 IFX*256<0PR"LEFT ";
5770 IFX*1024<0PR"RIGHT";
5780 PR
5790 RETURN
5800
5800 REM CONTINUING TROLL PROCESSOR
5800
5810 X=T
5815 IFP<NGOTO5850
5820 IFB*64<0GOTO5850
5825 B=B+512
5830 GOSUB12895
5835 PR"OBVIOUSLY DOMINATES ";
5840 GOSUB12755
5845 PR
5850 GOSUB4060
5860 IFX=TIFP>NGOTO3880
5870 IFX-6=NRETURN
5880 T=X
5890 IFP<NGOTO5980
5910 GOSUB8630
5920 P=T
5930 GOSUB2710
5940 IFRND(9)<3GOTO7270
5970 PR"WITH MAIDEN"
5980 M=T-7
5990 RETURN
6000
6000 REM INVENTORY PROCESSOR
6000
6010 GOSUB1950
6030 GOSUB6440
6040 PR"KEPT"
6050 GOSUB9770
6060 IFH<99GOTO6130
6070 GOSUB1950
6080 IFH=13008H=8008
6110 GOSUBH/50+12495
6120 GOTO6140
6130 GOSUB1910
6140 GOSUB1970
6160 GOTO1060
6200
6200 REM TEST KNAPSACK/ROOM EMPTY
6200
6210 X=(A/1000-2)*(F/1000-2)*(J/1000-2)*(K/1000-2)
6220 X=(L/1000-2)*(R/1000-2)*(S/1000-2)*X
6230 RETURN
6240 X=0
6250 IFN<>A+13IFN<>F+8IFN<>J+4IFN<>K+3X=M
6260 IFN<>L+2IFN<>R-4IFN<>S-5IFX=MRETURN
6270 X=0
6280 RETURN
6440 GOSUB6210
6450 P=2016
6460 IFX=0GOTO13810
6480 PR"NOTHING ";
6490 RETURN
```

```
6500
6500 REM DRAGON PROCESSOR
6500
6510 GOTOV
6520 X=D
6525 IFF-2=DIFB*256<0B=B-128
6530 IFD<25000IFRND(9)>3D=D+1000
6540 IFD>24000IFRND(9)>5D=D-1000
6550 GOSUB4920
6560 IFX=DGOTO6790
6580 IFP<NGOTO6930
6590 IFD+10=NGOTO6660
6610 IFX+10<>NGOTO6960
6620 PR"DRAGON EXITS ";
6630 P=D
6640 GOSUB2710
6650 GOTO6950
6660 GOSUB12575
6670 PR"ENTERS ";
6680 P=X
6690 GOSUB2710
6710 PR
6720 IFM+1=NGOTO7230
6730 IFM<>1013GOTO6960
6740 PR"THE ";
6750 GOSUB12660
6760 PR"TURNS COLD"
6770 RETURN
6790 IFD+10<>NGOTO6910
6810 IFP<NGOTO6880
6820 GOSUB12575
6830 PR"FILLS YOUR VIEW"
6840 IFV/8*B>0PR"IT SLEEPS"
6850 IFM+1<>NGOTO6730
6860 PR"MAIDEN SOBS"
6870 RETURN
6880 PR"YOU HEAR BREATHING"
6890 RETURN
6910 IFD=25004IFRND(9)>6B=B-16
6920 RETURN
6930 IFD+10<>NIFX+10<>NGOTO6960
6940 PR"YOU HEAR SCUFFLING";
6950 PR
6960 IFD+9=MIFM+7<>TGOTO7210
6960 RETURN
7000
7000 REM KEEP PROCESSOR (NO LIMIT)
7000
7010 GOSUB9770
7020 IFH<999GOTO6130
7030 IFH=13008GOTO7110
7040 P=2016
7050 GOSUB9680
7070 PR"OK"
7090 GOTO1180
7110 GOSUB7160
7120 GOSUB7180
7130 GOSUB12655
7140 PR
7150 GOTO1210
7160 PR"CAN'T ";
```

```
7170 RETURN
7180 PR"KEEP ";
7190 RETURN
7200
7200 REM MAIDEN PROCESSOR
7200
7210 IFN<22000RETURN
7220 IFN>28000RETURN
7230 PR"A SCREAM ";
7240 IFM+1=NGOTO7280
7250 PR"ECHOS ";
7260 GOSUB8020
7270 PR
7275 RETURN
7280 PR"PIERCES AIR"
7290 RETURN
7310 GOSUB4920
7320 IFB*V<0GOTO7380
7330 IFM+8=UIFN+7<>UIFM+7<>TIFRND(N/3)<P/4GOTO7345
7335 IFM+1=NGOTO7360
7340 RETURN
7345 M=N-1
7350 PR"YOU HEAR FOOTSTEPS"
7360 IFP>NPR"MAIDEN HOLDS OUT HAND"
7370 RETURN
7380 IFM+1=NIFP>NPR"MAIDEN DOES NOT SEE YOU"
7390 RETURN
7500
7500 REM LOOK PROCESSOR
7500
7510 GOSUB1660
7520 IFX=0PR"LOOK WHERE";
7530 GOSUB1720
7540 IFX>=0GOTOX*10+7600
7550 X=126
7555 GOSUB5620
7560 GOTO5080
7600 GOSUB1810
7605 GOTO7550
7610 GOTO1010
7620 Y=Y-200
7625 GOTO7655
7630 GOTO7600
7640 GOSUBN+596
7645 GOTO1180
7650 Y=N+211
7655 IFY<N+100Y=Y+400
7660 W=1023
7665 GOTO1060
7670 REM
7680 REM
7690 REM
7700 REM
7710 GOTO7600
7720 Y=Y-100
7725 GOTO7655
7730 GOTO7600
7740 Y=N+111
7745 GOTO7660
7750 REM
7760 REM
```

```
7770 GOTO7600
7780 Y=Y-300
7785 GOTO7655
7790 Y=N+311
7795 GOTO7660
7800 GOTO7600
7810 GOSUBN+496
7815 GOTO1180
7820 GOTO7600
7830 Y=N+411
7835 GOTO7660
8000
8000 REM SMALL PRINTS
8000
8010 GOTOV
8020 PR"IN THE DISTANCE ";
8030 RETURN
8040 PR"A BOAT IS ";
8045 PR"BEACHED"
8050 RETURN
8060 PR"YOU FELL INTO ";
8070 RETURN
8080 PR"A WATERFALL"
8090 RETURN
8100 PR"A STEEP BLUFF "
8110 RETURN
8370 PR"DARK ROOM"
8380 RETURN
8390 PR"IT IS LOCKED"
8400 RETURN
8500
8500 REM NO N
8500
8510 GOTOV
8520 GOSUB4920
8530 IFP<NGOTO20110
8540 IFX=0GOTO23110
8550 X=X-V-V
8560 PR"A CORRIDOR LEADS ";
8570 GOSUB5610
8580 PR"INTO THE GLOOM"
8590 RETURN
8610 PR"DRINK ";
8620 RETURN
8630 PR"TROLL EXITS ";
8640 RETURN
8650 PR"FLY"
8660 RETURN
8710 PR"SWIM"
8720 RETURN
8730 PR"DANK";
8740 RETURN
8750 PR"MISTY";
8760 RETURN
8770 PR"SOOTY";
8780 RETURN
8790 PR"PLAIN";
8800 RETURN
8810 PR"DINGY";
8820 RETURN
8830 PR"HANDS FULL"
```

```
8840 RETURN
8950
8950 REM WALL DESCRIPTOR
8950
8960 PR"IS A ";
8970 GOSUBX
8980 PR" WALL"
8990 RETURN
9000
9000 REM OPEN PROCESSOR
9000
9010 GOSUB3380
9020 GOTO3010
9110 X=3360
9150 GOTO4840
9210 PR"ONLY ";
9220 GOSUB9260
9230 PR"DOOR WINDOW CHEST"
9240 GOSUB3370
9250 GOTO5080
9260 X=0
9270 IFQ<0GOTO2970
9280 PR"CLOSE ";
9290 RETURN
9310 X=0
9320 GOSUBN+56
9325 X=X+X
9330 H=Q+Q
9340 IFQ<0H=-H
9350 H=H+H
9355 IFX<0H=H+1
9360 IFH/4*4<>HIFX>0H=H+1
9365 X=X+X
9370 IFX>0GOTO9350
9380 IFH>0IFX<0GOTO9350
9390 RETURN
9500
9500 REM PUT PROCESSOR
9500
9510 GOSUB9770
9520 IFH<99GOTO6130
9530 GOSUB9660
9540 GOTO7070
9600
9600 REM RELEASE HELD OBJECT
9600
9610 GOSUB9770
9660 P=N+2
9670 IFH<99RETURN
9680 GOTOH/50+10100
9700
9700 REM GET IN HAND
9700
9770 H=8
9810 IFA=1001H=1008
9820 IFF=1006H=6008
9830 IFJ=1010H=10008
9840 IFK=1011H=11008
9850 IFL=1012H=12008
9860 IFM=1013H=13008
9870 IFR=1018H=18008
```

```
9880 IFS=1019H=19008
9890 RETURN
10000
10000 REM NO Q
10000
10010 GOTOV
10100
10100 REM MOVE AN OBJECT
10100
10120 A=P-15
10125 RETURN
10220 F=P-10
10225 RETURN
10300 J=P-6
10305 RETURN
10320 K=P-5
10325 RETURN
10340 L=P-4
10345 RETURN
10360 M=P-3
10365 RETURN
10460 R=P+2
10465 RETURN
10480 S=P+3
10485 RETURN
10500
10500 REM NO R
10500
10510 GOTOV
11000
11000 REM NO S
11000
11010 GOTOV
11500
11500 REM TAKE PROCESSOR
11500
11510 GOSUB9770
11520 X=8830
11550 IFH>99GOTO4840
11560 GOSUB1660
11570 IFX=0PR"TAKE WHAT";
11580 GOSUB1720
11585 P=1016
11590 IFX>0IFX<20GOSUBX*20+12100
11600 IFX>0IFX<20GOTO7070
11610 PR"CAN TAKE:"
11615 P=2016
11620 GOSUB13810
11630 PR
11640 GOSUB13780
11660 GOTO5075
12120 IFA<>2001IFA+13<>NGOTO6270
12125 GOTO10120
12140 REM
12160 REM
12180 REM
12200 GOTO6270
12220 IFF<>2006IFF+8<>NGOTO6270
12225 GOTO10220
12240 GOTO6270
12260 GOTO12360
```

```
12280 GOTO6270
12300 IFJ<>2010IFJ+4<>NGOTO6270
12305 IFJ=26010IFQ*64>0GOTO6270
12310 IFJ=29010IFQ*128>0GOTO6270
12315 GOTO10300
12320 IFK<>2011IFK+3<>NGOTO6270
12325 IFK=29011IFQ*128>0GOTO6270
12330 GOTO10320
12340 IFL<>2012IFL+2<>NGOTO6270
12345 GOTO10340
12360 IFM<>2099IFM+1<>NGOTO6270
12365 GOTO10360
12380 REM
12400 REM
12420 REM
12440 GOTO6270
12460 IFR<>2018IFR-4<>NGOTO6270
12465 GOTO10460
12480 IFS<>2019IFS-5<>NGOTO6270
12485 GOTO10480
12490
12490 REM PRINT OBJECT NAME
12490
12495 PR"YOUR ";
12500 PR"BARE HANDS ";
12505 RETURN
12515 PR"AN ";
12520 PR"AXE ";
12525 RETURN
12555 PR"A ";
12560 PR"CHEST ";
12565 RETURN
12575 PR"A HUGE ";
12580 PR"DRAGON";
12585 RETURN
12615 GOSUB12630
12620 PR"FLASK ";
12625 RETURN
12630 PR"A";
12635 IFB*256<0PR"N EMPTY";
12640 PR" ";
12645 RETURN
12655 PR"A ";
12660 PR"FAIR MAIDEN'S HAND ";
12665 RETURN
12695 PR"SOME ";
12700 PR"JEWELS ";
12705 RETURN
12715 PR"SOME ";
12720 PR"KEYS ";
12725 RETURN
12735 PR"A ";
12740 PR"LANTERN ";
12745 RETURN
12755 PR"A FAIR ";
12760 PR"MAIDEN ";
12765 RETURN
12775 PR"A ";
12780 PR"MIRROR ";
12785 RETURN
12855 PR"A ";
```

```
12860 PR"ROCK ";
12865 RETURN
12875 PR"A ";
12880 PR"SWORD ";
12885 RETURN
12895 PR"AN EVIL ";
12900 PR"TROLL ";
12905 RETURN
12955 PR"A ";
12960 PR"WINDOW ";
12965 RETURN
13080
13080 REM HELP COMMAND
13080
13090 IFX>0GOSUB1810
13110 GOSUB4920
13115 IFP<NGOTO13180
13120 GOSUB2340
13125 IFP>99GOSUB1930
13160 GOSUB9310
13170 IFH<0IFX<0GOSUB9270
13180 IFF=1006IFB*256>0GOSUB8610
13190 PR"GO HELP INVENTORY ";
13210 GOSUB4920
13220 IFP<NGOTO13280
13230 PR"LOOK ";
13240 GOSUB3380
13250 GOSUB9310
13260 GOSUB3380
13270 IFH<0IFX<0GOSUB2970
13280 GOSUB9770
13290 IFH<99GOTO13340
13310 IFH<>13008GOSUB7180
13320 PR"PUTDOWN"
13330 GOTO1210
13340 GOSUB6210
13350 IFX=0GOTO13380
13360 GOSUB6240
13370 IFX>0IFX<>N-1GOTO4860
13380 PR"TAKE"
13390 GOTO1210
13500
13500 REM DESTROY AN OBJECT (D,M,T)
13500
13510 PR"YOU KILLED THE ";
13520 RETURN
13540 IFV/8*B<0B=B-16
13600 D=13004
13610 GOSUB13510
13620 GOTO12580
13630 M=13013
13680 GOSUB13510
13690 GOTO12760
13700 T=13020
13710 GOSUB13510
13720 IFK=13011K=N-3
13730 GOTO12900
13740
13740 REM LIST OBJECTS
13740
13745 PR"NEARBY IS ";
```

```
13750 GOSUB13770
13760 GOTO7270
13770 X=-5
13780 P=N+2
13790 GOTO13820
13810 IFP>NGOTO13770
13815 X=0
13820 IFA+15=PGOSUBX+12520
13830 IFF+10=PGOSUBX+12620
13840 IFJ+6<>PGOTO13910
13850 IFP<>26016IFP<>29016GOTO13890
13860 IFX=0IF(P/29000+1)*64*Q<0GOTO13890
13870 GOTO13910
13890 GOSUBX+12700
13910 IFK+5<>PGOTO13960
13920 IFP<>29016GOTO13950
13930 IFX=0IFQ*128<0GOTO13950
13940 GOTO13960
13950 GOSUBX+12720
13960 IFL+4=PGOSUBX+12740
13970 IFR-2=PGOSUBX+12860
13980 IFS-3=PGOTOX+12880
13990 RETURN
14000
14000 REM THE STREAM IN A RAVINE
14000
14010 PR"RAVINE"
14020 PR"A BROOK AT YOUR FEET"
14030 RETURN
14040 X=254
14070 RETURN
14110 IFB*8<0GOSUB8020
14115 GOSUB8080
14120 PR"CRASHES DOWN ";
14130 GOTO16110
14140 IFB*8<0GOTO1050
14145 GOSUB8060
14150 PR"A CREVICE BEHIND ";
14160 GOSUB8080
14170 Y=25225
14180 GOSUB2480
14185 PR"YOU TUMBLE FAR"
14190 GOTO1060
14210 PR"IS A ";
14220 GOTO18010
14240 Y=18225
14250 GOTO1050
14310 IFB*8>0GOSUB8020
14315 IFB*4>0GOSUB8040
14320 IFB*4>0PR"WHERE ";
14325 PR"STREAM MEETS SEA"
14330 RETURN
14340 IFB*8>0GOTO15470
14350 IFB*4<0GOTO21640
14360 Y=21325
14370 GOTO1050
14410 PR"IS A ";
14420 GOTO15010
14440 Y=15425
14450 GOTO1050
14510 GOTO14110
```

```
14540 Y=14125
14550 GOTO14140
14610 PR"THERE ARE ROCKS"
14620 RETURN
14640 Y=14325
14650 GOTO14340
14910 GOSUB6240
14920 IFX>0RETURN
14930 GOTO13745
15000
15000 REM INITIAL POSITION IN MEADOW
15000
15010 PR"MEADOW"
15040 X=30
15050 IFB*8<0X=14
15070 RETURN
15110 GOSUB16110
15120 PR"SHIMMER ";
15130 GOTO7260
15140 Y=16125
15150 GOTO1060
15210 PR"SOUNDS RUNNING WATER ";
15220 RETURN
15240 IFB*8>0Y=14225
15250 GOTO1050
15310 GOTO1380
15340 Y=17325
15350 GOTO1060
15410 IFB*8>0GOSUB8020
15420 PR"IS A COTTAGE"
15430 RETURN
15440 IFB*8<0GOTO23140
15470 GOSUB3480
15490 GOTO1060
15510 PR"FLEECY CLOUDS IN BLUE SKY"
15530 RETURN
15540 GOTO3410
15610 GOSUB6240
15615 IFX>0GOTO1380
15620 PR"HIDDEN IN GRASS:";
15630 GOTO22625
15640 GOTO4610
15910 RETURN
16000
16000 REM NORTH MEADOW
16000
16010 GOTO15010
16040 X=12
16045 IFB*8<0X=14
16050 IFV/2*Q>0X=X-X/14*8
16060 RETURN
16070 IFB*8<0X=4
16090 RETURN
16100 PR"CLIMB ";
16110 PR"SHEER CLIFFS"
16120 RETURN
16140 X=16100
16155 IFB*8<0GOTO1050
16160 GOTO4840
16210 GOTO1380
16240 Y=15225
```

```
16250 GOTO1060
16310 GOSUB15410
16320 GOTO29110
16340 IFB*8>0GOTO15470
16360 IFV/2*Q>0GOTO23140
16370 Y=29325
16380 GOTO1050
16384 GOTO13090
16390 PR"ENTER ";
16410 PR"A THICK HEDGE"
16430 RETURN
16440 X=16390
16460 GOTO4840
16510 GOTO15510
16540 GOTO3410
16610 GOTO15610
16640 GOTO4610
16910 RETURN
17000
17000 REM SOUTH MEADOW
17000
17010 GOTO15010
17040 X=6
17050 IFB*8<0X=14
17055 IFQ*1024>0X=X-X/7
17060 RETURN
17070 IFB*8<0X=1024
17080 RETURN
17110 GOSUB15410
17120 GOTO30310
17140 IFB*8>0GOTO15470
17160 IFQ*32>0GOTO4810
17170 Y=30125
17180 GOTO1050
17210 GOTO1380
17240 GOTO16240
17310 PR"A BLUFF OVERLOOKS SEA"
17315 IFB*4>0RETURN
17320 GOSUB8040
17325 PR"ON AN ISLAND"
17330 RETURN
17340 IFB*8<0GOTO1050
17350 GOTO4610
17410 GOTO16410
17440 GOTO16440
17510 GOTO15510
17540 GOTO3410
17610 GOTO15610
17640 GOTO4610
17910 RETURN
18000
18000 REM WEST FOREST
18000
18010 PR"FOREST"
18040 X=30
18070 RETURN
18110 PR"ARE TREES"
18120 RETURN
18140 N=19014
18150 GOTO3950
18210 GOTO18110
```

```
18240 GOTO18140
18310 GOTO18110
18340 GOTO18140
18410 GOTO15210
18440 Y=14425
18450 GOTO1050
18510 GOSUBY-N+1209
18520 PR"MOSS IS GREENER"
18535 RETURN
18540 GOTO3410
18610 GOTO22610
18640 GOTO4610
18910 IFS<>18019RETURN
18920 GOSUB12875
18930 PR"CATCHES YOUR EYE"
18935 RETURN
19000
19000 REM LOST IN THE FOREST
19000
19010 GOSUB18010
19020 PR"EVERY DIRECTION LOOKS SAME"
19040 X=30
19070 RETURN
19110 GOTO18110
19140 GOTO3950
19210 GOTO18110
19240 GOTO3950
19310 GOTO18110
19340 GOTO3950
19410 GOTO18110
19440 IFB*8<0GOTO19480
19450 B=B+V/4
19460 GOTO3960
19480 N=18014
19490 GOTO3960
19510 GOTO18510
19540 GOTO3410
19610 GOTO22610
19640 GOTO4610
19910 RETURN
20000
20000 REM LOST AT SEA
20000
20010 GOSUB21010
20020 PR"IT'S DARK"
20040 X=30
20070 RETURN
20110 PR"YOU SEE NOTHING"
20120 RETURN
20140 IFB*8>0GOTO19450
20150 P=N+2
20160 N=21014
20180 GOSUB4695
20190 GOTO3960
20210 GOTO20110
20240 GOTO3950
20310 GOTO20110
20340 GOTO3950
20410 GOTO20110
20440 GOTO3950
20510 GOSUBY-N+1209
```

```
20520 PR"BIG DIPPER TWINKLES"
20530 RETURN
20540 GOTO3410
20610 GOTO21610
20640 GOTO21640
20910 GOSUB4920
20920 IFP>NGOTO21910
20930 RETURN
21000
21000 REM IN BOAT IN WATER
21000
21010 PR"BOAT ";
21015 IFN>21000IFB*8>0GOTO21030
21020 PR"AT SEA"
21025 RETURN
21030 GOSUB8045
21035 IFB*4<0GOTO21990
21040 X=30
21070 RETURN
21110 IFB*8>0IFB*4>0GOTO21120
21115 GOSUB8020
21120 PR"IS MAINLAND"
21125 RETURN
21140 IFB*8>0IFB*4>0GOTO21180
21150 B=B-V/4
21170 GOTO1060
21180 Y=14125
21190 GOTO15470
21210 PR"HORIZON IS LOST ";
21220 GOSUB8020
21230 GOTO7270
21240 P=N+2
21250 N=20014
21260 GOTO20180
21310 IFB*4>0GOSUB8020
21320 PR"IS A ";
21330 GOTO22010
21340 IFB*4<0GOTO21380
21350 B=B+V/4
21370 GOTO1060
21380 Y=22325
21390 GOTO1050
21410 GOTO21210
21440 GOTO21240
21510 GOTO15510
21540 GOTO3410
21610 PR"WATER IS CLEAR"
21620 RETURN
21640 X=8710
21650 GOTO4840
21910 GOSUB6240
21915 IFX>0IFN-1<>MRETURN
21920 PR"WITH YOU IS"
21925 IFM/2000=10GOSUB12755
21930 GOTO13750
21990 PR"AT A ";
22000
22000 REM ON AN ISLAND
22000
22010 PR"SMALL ISLAND"
22040 X=16
```

```
22050 IFB*4<0X=18
22070 RETURN
22110 IFB*4>0GOTO21115
22115 GOTO21010
22140 IFB*4>0GOTO21640
22150 Y=21125
22160 GOTO1050
22210 GOTO21210
22240 GOTO21640
22310 GOTO21210
22340 GOTO21640
22410 GOTO22990
22440 Y=23425
22450 GOTO1050
22510 GOTO15510
22540 GOTO3410
22610 PR"ON THE GROUND IS ";
22620 GOSUB6240
22625 P=N+2
22630 GOSUB6460
22635 GOTO7270
22640 GOTO4610
22910 GOSUB6240
22920 IFX>0RETURN
22930 GOTO22610
22990 PR"IS A ";
23000
23000 REM IN THE CAVE
23000
23010 PR"CAVE"
23040 X=84
23070 RETURN
23110 X=8730
23120 GOTO8960
23140 X=1450
23150 GOTO4840
23210 PR"SUN SHINES IN"
23220 RETURN
23240 Y=22225
23250 GOTO1050
23310 PR"ARE GRAFFITI"
23320 RETURN
23340 GOTO23140
23410 PR"A TUNNEL SLOPES DOWN"
23420 RETURN
23440 Y=24125
23450 GOTO1050
23510 GOTO20110
23540 GOTO4610
23610 PR"TUNNEL IS DARK"
23620 Y=23425
23630 RETURN
23640 GOTO23440
23910 GOTO22910
24000
24000 REM IN THE TUNNEL
24000
24010 GOSUB4920
24020 IFP<NPR"DARK ";
24030 PR"TUNNEL"
24040 GOSUB24070
```

```
24050 IFV/X*2*Q<0X=X+1
24060 X=72+X/8000*38+X*16
24065 RETURN
24070 X=V/4
24080 IFB*8>0X=X+X
24090 RETURN
24110 X=32
24120 IFB*8<0X=0
24130 GOTO8520
24140 IFB*8<0GOTO23140
24160 GOTO15470
24210 X=64
24220 GOTO24120
24240 IFB*8<0GOTO23140
24250 Y=25225
24260 GOTO1050
24310 X=1024
24320 IFB*8<0X=64
24330 GOTO8520
24340 IFB*8<0GOTO24650
24350 Y=23225
24360 GOTO1050
24410 X=24420
24415 GOTO4630
24420 GOSUB24070
24425 X=V/X*2
24430 GOTO29320
24440 GOSUB24070
24450 IFV/X*2*Q>0GOTO4610
24460 Y=26425
24470 IFB*8<0Y=27425
24490 GOTO1050
24510 X=24520
24515 GOTO4630
24520 IFB*8<0GOTO20110
24525 Y=24125
24530 GOSUB1520
24535 GOTO24110
24540 Y=24125
24550 GOTO24140
24610 Y=24225
24615 IFB*8>0GOTO24210
24620 Y=24325
24625 GOTO24310
24640 IFB*8>0GOTO24350
24650 Y=24225
24660 GOTO1050
24910 GOSUB4920
24920 IFP>NIFB*8>0GOTO22910
24930 RETURN
25000
25000 REM IN DRAGON'S LAIR
25000
25010 GOSUB4920
25015 IFP<NGOTO8370
25020 PR"CAVERN"
25040 X=40
25070 RETURN
25110 X=25120
25115 GOTO4630
25120 X=8750
```

```
25130 GOTO8960
25140 GOTO25440
25210 X=25220
25215 GOTO4630
25220 X=8770
25230 GOTO8960
25240 GOTO25440
25310 X=25320
25315 GOTO4630
25320 X=32
25330 GOTO8520
25340 Y=24425
25350 GOTO1050
25410 X=23110
25420 GOTO4630
25440 IFD>VIFV/8*B>0B=B+16
25450 GOSUB4920
25460 IFP<NGOTO4610
25470 GOTO23140
25510 GOTO20110
25540 GOTO25340
25610 X=25630
25620 GOTO4630
25630 GOTO22610
25640 GOTO4610
25910 X=D
25920 IFV/8*B>0GOSUB6550
25930 GOTO24910
26000
26000 REM IN THE TROLL'S DEN
26000
26010 GOSUB4920
26015 IFP<NGOTO8370
26020 PR"TROLL'S DEN"
26040 X=0
26050 IFQ*4<0X=16
26060 RETURN
26070 X=8704
26090 RETURN
26110 GOTO25410
26140 GOTO25440
26150 IFQ*4>0GOTO20110
26160 PR"IT'S DRAFTY"
26170 RETURN
26210 GOSUB4920
26220 IFP<NGOTO26150
26230 X=4
26235 GOTO29320
26240 IFQ*4>0GOTO4810
26250 Y=24225
26260 GOTO1050
26310 X=26320
26315 GOTO4630
26320 X=12775
26325 IFB*V<0GOTO26350
26330 PR"YOU SEE YOURSELF IN ";
26335 GOTO26350
26340 GOTO25440
26350 GOSUBX
26360 GOTO7270
26410 X=26420
```

```
26415 GOTO4630
26420 X=64
26430 GOTO29420
26440 GOTO25440
26510 X=26520
26515 GOTO4630
26520 X=8810
26530 GOTO29520
26540 GOTO4610
26610 X=26620
26615 GOTO4630
26620 PR"NOT MUCH AROUND.."
26630 GOTO13750
26640 GOTO4610
26910 RETURN
27000
27000 REM IN THE WINE CELLAR
27000
27010 PR"WINE CELLAR"
27020 GOSUB4920
27030 IFP<NGOTO20020
27040 X=34
27050 IFQ*8<0X=38
27060 RETURN
27070 X=V/4
27090 RETURN
27110 PR"STEPS UP ARE FAINTLY LIT"
27120 RETURN
27140 Y=28225
27150 GOTO1050
27210 X=8
27220 GOTO29320
27240 IFQ*8>0GOTO4810
27250 Y=24325
27260 GOTO15470
27310 GOTO25410
27340 GOTO25440
27410 X=27420
27415 GOTO4630
27420 PR"IS A WINE RACK"
27430 RETURN
27440 GOTO25440
27510 GOTO26510
27540 GOTO27140
27610 GOTO26610
27640 GOTO27240
27910 GOTO22910
28000
28000 REM IN THE HALL/STAIRWELL
28000
28010 PR"HALLWAY"
28020 RETURN
28040 X=82
28050 RETURN
28070 X=V/8
28090 RETURN
28110 GOTO29310
28140 IFQ*16>0GOTO4810
28150 Y=29125
28160 GOTO1050
28210 X=8790
```

```
28220 GOTO8960
28240 GOTO23140
28310 PR"A STAIRWAY DESCENDS"
28320 GOTO8580
28340 Y=27325
28350 GOTO1050
28410 PR"OPENS TO NEXT ROOM"
28420 RETURN
28440 Y=30425
28450 GOTO1050
28510 PR"A SKYLIGHT"
28520 RETURN
28540 GOTO4610
28610 GOTO28310
28640 GOTO28340
28910 GOSUB6250
28920 IFX>0RETURN
28930 GOTO26620
29000
29000 REM IN BEDROOM
29000
29010 PR"BEDROOM"
29040 X=10
29050 RETURN
29070 X=2308
29090 RETURN
29110 X=V/2
29120 GOSUB12955
29125 IFX*Q<0GOTO2960
29130 PR"IS CLOSED"
29135 RETURN
29140 IFV/2*Q>0GOTO23140
29150 Y=16125
29160 GOTO15470
29210 PR"IS A BED"
29220 RETURN
29240 GOTO23140
29310 X=16
29320 PR"A DOOR ";
29330 GOTO29125
29340 IFQ*16>0GOTO4810
29350 Y=28325
29360 GOTO1050
29410 X=128
29420 GOSUB12555
29425 IFX*Q>0GOTO29130
29430 GOTO2910
29440 GOTO23140
29510 X=8790
29520 PR"THE CEILING IS";
29530 GOSUBX
29535 GOTO7270
29540 GOTO4610
29610 GOTO26620
29640 GOTO4610
29910 GOSUB6240
29920 IFX=0GOTO26620
29930 RETURN
30000
30000 REM IN THE TABLE ROOM
30000
```

```
30010 PR"ROOM WITH TABLE"
30040 X=4
30050 IFQ*32<0X=12
30060 RETURN
30070 X=1024
30090 RETURN
30110 GOTO28210
30140 GOTO23140
30210 GOTO28410
30240 Y=28225
30250 GOTO1050
30310 X=32
30320 GOTO29320
30340 IFQ*32>0GOTO4810
30350 Y=17325
30360 GOTO15470
30410 GOTO26310
30440 GOTO23140
30510 GOTO29510
30540 GOTO4610
30610 PR"THE FLOOR IS CLEAN"
30620 RETURN
30640 GOTO4610
30910 GOSUB6240
30915 IFX<>0RETURN
30920 PR"ON THE TABLE ARE"
30925 GOSUB13750
30930 IFB*128<0RETURN
30935 PR"FLASK CONTAINS MILKY FLUID"
30940 PR"LABELLED 'MAGIC DRAGON TEARS'"
30950 B=B+256
30955 RETURN
32000 END
```

Chapter 7 -- IEEE DISASSEMBLER

Everybody has a disassembler for their own computer. It is one of the first programs you get, so that you can start hacking on acquired code. This disassembler is different. First, it is written in Tiny BASIC, so you can easily modify it. More important, it handles three machines, not just the one you have. And most significant, it uses the new proposed standard IEEE mnemonics and syntax.

The IEEE Computer Society has been involved in standards activities for microprocessors since 1977. One of the early projects was a uniform representation of similar functions by the same mnemonic in microprocessor instruction sets (project P694). This work climaxed in a draft standard published in Computer magazine in December 1980. Responses to that publication resulted in some changes to the proposed standard, which have been included in this disassembler. The changes are mostly syntactic, not having to do with mnemonics.

7.1 -- Operation

Running the disassembler is self-explanatory. Entries are made in a special form of hexadecimal, where the letter "O" is used instead of the digit zero (like in the hex dump at the back of the Tiny BASIC User Manual). It will first ask for the address of the cold start of your interpreter, which you type in this funny-hex. Each funny-hex number must also end with the letter "X", because Tiny BASIC keeps reading digits across spaces, commas, and carriage returns.

Then you are given a choice of CPU type: 1802, 6502, or 6800 (sorry, no Z8).

The code to be disassembled is assumed to be already in memory (getting it there is your problem), but possibly at a different location that it is intended to execute. The "offset" input will change line addresses and relative jump addresses by the offset value so that the disassembly listing reflects the true execution addresses. A positive offset means that the location in your memory is greater than the intended (and listed) address. For example, if you load into memory locations starting from hex "4OOOX" a program intended to execute at hex "O1OOX", then the offset is the difference, "3FOOX". For programs loaded into their correct locations, just type in zero "OX".

Now you will be asked for a sequence of starting and ending addresses. If the starting address you give is less than the ending address, then the program will go back to ask you for "Which CPU?" Otherwise the program between the starting and ending addresses (and possibly one or two bytes farther, if in a 3-byte instruction) will be disassembled. Note that these addresses are the logical address values, not the physical place in memory where the program is. If the offset is not zero, it is added to the starting address to get the actual location of the code. After completing one section of disassembly, the disassembler asks for another starting and ending address. Important: Because addresses in the upper 32K of memory are negative, you cannot disassemble in one segment a program that crosses address "8OOOX" -- two starting and ending address sequences are required, the first ending at "7FFFX" and the second starting at "8OOOX" (or just after, if an instruction crosses the boundary).

The form of the listing is a hexadecimal (logical) address, then the opcode byte in hex, followed by zero, one, or two bytes of operand, then the mnemonic in the next column, and any operands in the next column after that.

7.2 -- Operand Syntax

The IEEE standard makes two consistent syntactical requirements. First, the location or addressing mode of the operands is to be specified in the operand field, rather than in the mnemonic. Second, when there is more than one operand they are ordered in a uniform Source-Destination sequence (insofar as that is meaningful).

7.3 -- Addressing Modes

The proposed standard calls for a unique prefix character to identify every common addressing mode in a microprocessor. The addressing modes used in this disassembler are as follows:

Mode	Form	Example
Immediate	prefixed "#"	#0D
Register	prefixed "."	.X
Relative	prefixed "$"	$01FC
Base Page ("Direct")	prefixed "!"	!20
Absolute ("Extended")	prefixed "/"	/0100
Indirect	prefixed "@"	@126
Indexed	parentheses	2(.X)
Post-Indexed Indirect	postfixed "@"	120(.X)@

Of course, as some of the examples show, certain of the addressing modes may be combined.

7.4 -- Mnemonics

The IEEE mnemonics were carefully chosen to reflect common usage and perspicuity. For example, the first letter of each mnemonic is normally the first letter of the action verb that describes the function of the opcode.

The 1802 presented a little bit of a dilemma, since the address register loading instructions (in RCA's notation "GLO", "GHI", "PLO", and "PHI") have no exact correspondence in the standard. I tend to view the machine in a hierarchical fashion, something like this:

```
CPU accumulator, carry
 |__ address registers
 |__ memory
```

The other way to view it would put the address registers up at the same level as the accumulator and carry. The point of view has a profound effect on the mnemonics chosen here, since memory-to-register transfers require the standard mnemonics "LD" and "ST", while register-to-register transfers require the standard mnemonic " MOV". The "MOV" mnemonic is a horizontal operator, and requires the specification of both the source and destination. The problem here is that there is not an adequate name for the 1802 accumulator. RCA uses "D" in its documentation, but that is too easily confused with the hex number 13. The IEEE register name ".D" can't be used, as it is the IEEE standard the name of 1802 address register R13. I opted for my favorite model (vertical) so I could use the "LD" and "ST" mnemonics, which require no specification of the accumulator (there is only one, so it is implied). The justification is that the register file can be thought of as a 32-byte scratchpad RAM with some special functions. Indeed, many programs I write for the 1802 treat it exactly that way.

The register loading and storing operations are distinguished from the memory references by a qualifier tacked onto the mnemonic (as per the standard recommendation) to say whether the high or low byte is accessed. This is not a necessary distinction, since there would be no ambiguity of reference: Register references use register addressing in the operand field, while memory accesses use register indirect or implied addressing.

The table on the next two pages gives the standard mnemonics (as augmented for the 1802) and the usual manufacturer-defined mnemonics for reference. Note that in many cases there are several manufacturer-defined mnemonics corresponding to one IEEE mnemonic. The reverse is also true, but the disassembler must choose one. For example, the IEEE mnemonic SETC (set carry) corresponds to a 2-byte instruction in the 1802 (FFOO), but it is usually represented as SUB #0. Many of the blanks in the table, however, represent instructions not available in the respective processor.

A number of IEEE mnemonics have not been included in this table because they are not printed out by the disassembler. Several of these are branch condition names in the standard but not represented here, though in fact they do have opcodes in the respective machines. For example, the standard mnemonic BEQ is the same as BZ for all three CPUs; BL (Branch on unsigned Low) is BNC in the 1802 and 6502, and BC in the 6800.

The last 19 entries in the table are mnemonics I made up to cover deficiencies in the standard. They are constructed according to the guidelines specified in the standard, and are believed to be consistent with it. I do not mean to suggest here, however, that they are in fact "standard" mnemonics, as are the first 62 entries.

7.5 -- Configurator

Any standard version of Tiny BASIC should have no trouble running this program. The address of the Cold Start is requested only to compute the peek address in line 160. This address is used only in line 3060. Though I cannot see much value in it, Z8 users could run this program by changing either of these two lines.

The disassembler is quite large (10426 bytes as listed), but if you are interested in only one CPU, the code for the other two is easily removed. Of course, the usual byte-saving techniques can be used also. The line numbers involved are (note there is some overlap):

```
1802: 1140, 1600-1780, 10000-14960
6502: 1130, 1400-1570, 18010-21890
6800: 1200-13409 15000-18040
```

7.6 -- Opcode Table

```
IEEE RCA MOS-Tech Motorola
ADD  ADD,ADI ABA,ADD
ADDC ADC,ADCI ADC ADC ADJ DAA AND AND,ANI AND
AND BRK BRK swi BR BR,LBR imp B R A , J M P BC BDF,LBDF BCS BCS BGE
BGE BGT
BGT BH BHI BLE BLE BLT BLT BN BMI BMI BNC BNF,LBNF B C C B C C BNH
BLS BNV
BVC BVC BNZ BNZ,LBNZ BNE BNE BP BPL BPL BV BVS BVS BZ BZ,LBZ BEQ BEQ
CALL S
E P J S R B S R J S R C L R REQ CLR C L R C C L C CLC CLRV CLV CLV
CMP CMP,
CPX,CPY CBA,CMP,CPX DEC DEC DEC,DEX,DEY DEC,DES,DEX DI DIS SEI SEI EI
RET
CLI CLI IN INP INC INC,IRX INC,INX,INY INC,INS,INX LD LDA,LDI,LDN,
LDA,LDX,
LDY LDA,LDS,LDX L D X , S E X Mov TAX,TAY,TSX, TAB,TAP,TBA,
TXA,TXS,TYA
TPA,TSX,TXS NEG NEG NOP NOP NOP NOP NOT COM OR OR,ORI ORA ORA OUT OUT
POP
LDXA PLA,PLP PUL PUSH MARK,SAV,STXD PHA,PHP PSH RET SEP RTS RTS RETI
RTI RTI
ROLC SHLC ROL ROL RORC SHRC ROR ROR

IEEE RCA MOS Technology Motorola S E
T SEQ SETC SEC SEC SETV SEV SHL SHL ASL ASL SHR SHR LSR LSR SHRA ASR
SKIP
LSKP,SKP SKC LSDF SKNC LSNF SKNZ LSNZ SKZ LSZ ST STR STA,STX,STY
STA,STS,STX
SUB sm'smi SBA,SUB SUBC SMB,SMBI S B c SBC SUBR SD,SDI SUBRC SDB,SDBI
TEST
BIT BIT,TST WAIT IDL WAI XOR XOR,XRI EOR EOR BNQ BNQ,LBNQ BNX1 BN1
BNX2 BN2
BNX3 BN3 BNX4 BN4 BQ BQ,LBQ BX1 Bl BX2 B2 BX3 B3 BX4 B4 CLRD CLD LDHI
GHI
LDLO GLO SETD SED SKIE LSIE SKNQ LSNQ SKQ LSQ STHI PHI STLO PLO
```

7.7 -- Program Notes

Originally I intended to make this an interpreter (simulator) as well as a disassembler. Now wouldn't that be a trip! A Tiny BASIC program on your computer interpreting a Tiny BASIC interpreter interpreting a Tiny BASIC program (perhaps the same program, for yet another iteration); can you imagine how slow that would be? But, as with some of the other objectives of this book, my sights got lowered. If there is sufficient interest, I'll try to do it in Volume 2.

In principle, a disassembler is very simple. You look at the next byte, and find it in your opcode table; if it is a multi-byte instruction, you get the extra bytes. All the bytes are printed in hex, then the opcode mnemonic is printed and the operands are decoded and printed out. Then you repeat, until you reach the ending address.

In this disassembler, the initialization is in the line numbers less than 1000. The lines 1xxx analyse the opcode. 2xxx print it, and the operand bytes in hex, then jump to the mnemonic printer. Lines 8xxx input and print funny-hex numbers. The mnemonic tables and operand representations are encoded in the program from line

10000 on. There is almost nothing in here that is not rather plain and understandable. A few places use long complicated arithmetic expressions to get a fancy result. I'll try to explain what they are doing. Otherwise almost anything I could say would be trivial.

7.8 -- Variables

```
A Hex input.
B Hex input.
C Hex input.
D Hex input.
E Hex input.
F Hex input.
G Peek address.
H Hexadecimal temporary.
I Temporary.
J Upper limit.
L M N 0 P Q R Operand location, CPU number, Opcode, Hex input,
Instruction address.
S T U V Memory value.
W Number of words.
X Hex In and Out.
Y
Z Memory offset.
```

7.9 -- Opcode Maps

Part of the analysis of the opcode necessary to determine how many bytes are involved, and what the operand format is, is to determine which class of instruction it is. Most CPUs have a (usually small) number of fixed formats, and the trick is to discover these and use them to advantage in a disassembler. Nice CPUs like the 8080 and 1802 put their fields on hexadecimal or octal boundaries.

Then there is the 6502. Actually the 6502 fields are reverse octal (3-3-2 instead of 2-3-3). The 6800 and 1802 have a well-defined major field, then sub-fields within that. The 6502 is much messier.

Let's start with an easy one. The 1802 comes in two formats, determined by the upper hex digit: 1111 RRRR and 1111 N SSS. The four major instruction bits select one of sixteen groups of instructions. Some of these are an opcode by themselves, and the lower four bits are a register number (an operand); this is the first format above. The others generally have two subfields, where the low three bits select one of eight functions and bit 3 selects an option or variation, or bit 3 defines a major group and the low three bits determine a port number (in the case of 1/0). For the disassembler, I linearized the sequence by giving every instruction of the first format an opcode number between 0 and 15, then (generally, with special-case exceptions) assigning the second format to opcodes of a full eight bits.

The 6800 has more formats, but fewer special cases: 00 11 SSSS, 01 AA SSSS, and 1 R AA SSSS. The most significant bit or two selects the format, the next two

bits determine an operation class or addressing mode, and the low four bits select the function within that class. These are linearized by using the full eight bit value for the first quadrant (high two bits zero), then taking the other two formats as a sequence of 16 to 87 opcodes each (on the low four bits, offset to not overlap).

The 6502 has basically two formats: SSS AAA I I and SSS SSS 00. The low two bits are the major distinction, with the two non-zero cases most well-defined (all opcodes with 11 in the low two bits are illegal). If the low two bits are zero, the fields are very ill-defined. In one case (the middle three bits 100) the upper three bits define the instruction. In another case (most significant bit 1, middle three bits not 100), it follows the first format. It is obvious that this CPU was not a top-down design like the others. I linearized the opcode set by making the low two bits major, and the high three minor within that. The exceptions then got tacked on to the end of the sequence. In all cases, the addressing mode bits of the opcode are saved out (not considered part of the opcode) to affect the format of the operand printing but not the mnemonic, as per the IEEE requirements. The linearized opcode is used to compute the GOTO in line 2180; so peculiarities in the computation of it can be understood in the light of the correspondence of line numbers to mnemonics in the mnemonic table section. For example, line 1530 looks complicated enough: if it is a one-byte instruction (W=O), we are interested in the case that the low five bits are 01010 and the most significant bit is 0. All others get a new sequence value (variable 0) that moves them out of the way. Y'know, now that I look at this I gotta say, there must be a better way to do this. Maybe I should offer a prize to the first person who can provide a lucid description of how this works. How about a free copy of Volume 2 (if I do a Volume 2)?

The moral of the story is that Tiny BASIC is the worst of all possible programming languages for stuff like like this, except maybe binary absolute is probably worse.

Line 8170 is used to convert a number read in as decimal into the binary number that should have come in if it had been read in as hexadecimal. For example, if you type in "1234", the digit "31" is weighted by the decimal input routine with a value of 10; but, as hexadecimal, it should be weighted with a value of 16, so the difference (six) times that digit's value must be added on. Similarly, the "2" was weighted 100 (i.e. came in as 200), but we want it to be weighted by 256. The correction to the tens digit has added another weighting of 60 to this digit (bringing it to 160), so there remains yet 96 to add. And so on. Try working a number through.

In line 8820, the objective is to do a right shift by division, but negative numbers do not shift correctly. Therefore, if the number in H is negative, we remove the sign bit (by subtracting off 32768), do the divide, then re-insert the removed bit (which as shifted, now has the value 8).

Line 10820 does exactly the same thing.

7.10 -- Program Listing

```
[this code re-formated by Richard Peters]

100 PRINT "ITTY BITTY DISASSEMBLER"
110 PRINT "COPYRIGHT 0 1981 T.PITTMAN"
120 PRINT
130 PRINT "COLD START (HEX, USE LETTER OH FOR 0, END WITH X)";
140 GOSUB 8010
150 IF V/256*256<>V THEN GOTO 120
160 LET G=V+20
210 PRINT
220 PRINT "WHICH CPU (1802, 6502, 6800)";
230 INPUT M
240 IF M<>1802 IF M<>6502 IF M<>6800 THEN GOTO210
250 PRINT
260 PRINT "OFFSET (HEX, 0X IF NONE)";
270 GOSUB 8080
280 LET Z=V
320 PRINT
330 PRINT " HEX STARTING ADDRESS";
340 GOSUB 8010
350 LET P=V
360 PRINT " HEX ENDING ADDRESS";
370 GOSUB 8080
380 LET J=V
390 IF P>J THEN GOTO210
1000 REM MAIN INSTRUCTION DECODE
1010 PRINT
1020 IF P>J THENGOTO320
1030 LET H=P
1040 GOSUB 8810
1050 GOSUB 3060
1060 LET O=V
1070 LET H=V
1080 PRINT "  ";
1090 GOSUB 8880
1110 PRINT " ";
1120 LET W=0
1130 IF M=6502 THEN GOTO1410
1140 IF M=1802 THEN GOTO1610
1200 REM 6800 DECODE
1210 IF O=33 THEN GOTO 9970
1220 IF O=141 THEN LET O=33
1230 LET I=O/16
1240 IF I=2 THEN LET W=1
1250 IF I<4 THEN GOTO 1330
1260 LET O=O-I*16+I/8*16+64
1270 IF I>5 THEN LET W=1
1280 IF I/4*4+3=I THEN LET W=2
1290 IF V=140 THEN LET W=2
1310 IF V=142 THEN LET W=2
1320 IF V=206 THEN LET W=2
1330 LET O=O+250
1340 GOTO2010
1400 REM 6502 DECODE
1410 LET I=O-O/4*4
1430 IF I=3 THEN GOTO 9970
1440 IF O=32 THEN LET W=2
1450 IF O-O/16*16>11 THEN LET W=2
1460 IF O-O/32*32=25 THEN LET W=2
```

```
1470 IF O-O/16*16-I/2*2=8 THEN GOTO 1510
1480 IF O/32<4 THEN IF O=O/32*32 THEN GOTO 1510
1490 IF W<>2 THEN LET W=1
1510 LET O=O/32+I*8
1520 IF V-V/32*32=16 THEN LET O=O+24
1530 IF W=0 IF V-V/32*32-V/128<>10 THEN LET O=(V/16+I)*8-V/32*15+32
1540 LET I=(V-V/32*32) /4
1550 IF V=V/32*32 IF V<128 THEN LET O=V/32+48
1560 LET O=O+500
1570 GOTO2010
1600 REM 1802 DECODE
1610 LET O=V/16
1620 LET I=V-O*16
1630 IF O=12 THEN IF (V-V/8*8)/4=0 THEN LET W=2
1640 IF O=3 THEN IF V<>56 THEN LET W=1
1650 IF V=200 THEN LET W=0
1660 IF O=7 THEN IF V<>126 THEN LET W=I/12
1670 IF O=15 THEN IF V<>254 THEN LET W=I/8
1680 IF O<16 THEN IF O-O/8*8=7 THEN LET O=V-W*8
1690 IF V=0 THEN LET O=192
1710 IF W=2 THEN LET O=V-144
1720 IF V=96 THEN LET O=V
1730 IF V/2=106 THEN LET O=V
1740 IF V=254 THEN LET O=160
1750 IF O=3 THEN LET O=V
1760 IF O=12 THEN LET O=V
1770 IF O=6 THEN LET O=O+I/8
1780 IF O=212 THEN LET W=2
2000 REM DISPLAY OPERANDS
2010 IF W=0 THEN GOTO2160
2020 GOSUB 3060
2030 LET L=V
2040 LET H=V
2050 IF W=1 THEN GOTO2140
2060 GOSUB 8880
2070 GOSUB 3060
2080 IF M=6502 THEN GOTO2120
2090 LET L=L*256+V
2110 GOTO2140
2120 LET L=L+V*256
2140 LET H=V
2150 GOSUB 8880
2160 PRINT "",
2180 GOTO O*20+10010
3000 REM FETCH OPCODE BYTE
3060 LET V=USR(G,P+Z)
3070 LET P=P+1
3080 RETURN
8000 REM INPUT HEX VALUE
8010 LET A=-10
8020 LET B=-11
8030 LET C=-12
8040 LET D=-13
8050 LET E=-14
8060 LET F=-15
8070 LET X=-999
8080 LET O=0
8090 LET V=0
8110 INPUT H
8120 IF H=X THEN RETURN
8130 IF H>999 THEN LET V=V*16
```

```
8140 IF H>99 THEN LET V=V*16
8150 IF H>9 THEN LET V=V*16
8160 LET V=V*16
8170 IF H>=0 THEN LET V=V+H+H/10*6+H/100*96+H/1000*1536
8180 IF H<0 THEN LET V=V-H
8190 GOTO 8110
8600 REM PRINT HEX VALUE
8620 PRINT "A";
8630 RETURN
8650 PRINT "B";
8660 RETURN
8680 PRINT "C";
8690 RETURN
8710 PRINT "D";
8720 RETURN
8740 PRINT "E";
8750 RETURN
8770 PRINT "F";
8780 RETURN
8810 LET X=H/4096
8820 IF H<0 THEN LET X=(H-16384*2)/4096+8
8830 LET H=H-X*4096
8840 GOSUB8950
8850 LET X=H/256
8860 LET H=H-X*256
8870 GOSUB 8950
8880 LET X=H/16
8930 GOSUB 8950
8940 LET X=H-X*16
8950 IF X>9 THEN GOTO X*30+8320
8960 PRINT X;
8970 RETURN
9900 REM ILLEGAL OPCODE
9970 PRINT "",
9980 PRINT "***"
9990 GOTO 1020
10000 REM 1802 MNEMONICS
10010 PRINT "LD","@";
10020 GOTO 10330
10030 PRINT "INC",
10040 GOTO 10330
10050 PRINT "DEC",
10060 GOTO 10330
10090 PRINT "LD","(";I;")+"
10100 GOTO 1020
10110 PRINT "ST","@";
10120 GOTO 10330
10130 PRINT "OUT",I
10140 GOTO 1020
10150 PRINT "IN",I-8
10160 GOTO 1020
10170 PRINT "LDLO",
10180 GOTO 10330
10190 PRINT "LDHI",
10200 GOTO 10330
10210 PRINT "STLO",
10220 GOTO 10330
10230 PRINT "STHI",
10240 GOTO 10330
10270 PRINT "CALL","@";
10280 GOTO 10330
```

```
10290 PRINT "LD","#";I;",.X"
10300 GOTO 1020
10330 PRINT ".";I
10340 GOTO 1020
10810 IF W=2 THEN GOTO 10850
10820 IF P<0 THEN LET L=L-(-P-256)/256*256-256
10830 IF P>0 THEN LET L=L+P/256*256
10840 GOTO 10860
10850 PRINT "/";
10860 LET H=L
10870 GOSUB 8810
10880 GOTO 1010
10970 PRINT "BR",
10980 GOTO 10810
10990 PRINT "BQ",
11000 GOTO 10810
11010 PRINT "BZ",
11020 GOTO 10810
11030 PRINT "BC",
11040 GOTO 10810
11050 PRINT "BX1",
11060 GOTO 10810
11070 PRINT "BX2",
11080 GOTO 10810
11090 PRINT "BX3",
11100 GOTO 10810
11110 PRINT "BX4",
11120 GOTO 10810
11130 PRINT "SKIP",1
11140 GOTO 1020
11150 PRINT "BNQ",
11160 GOTO 10810
11170 PRINT "BNZ",
11180 GOTO 10810
11190 PRINT "BNC",
11200 GOTO 10810
11210 PRINT "BNX1",
11220 GOTO 10810
11230 PRINT "BNX2",
11240 GOTO 10810
11250 PRINT "BNX3",
11260 GOTO 10810
11270 PRINT "BNX4",
11280 GOTO 10810
11930 PRINT "INC","@.X"
11940 GOTO 1020
12250 PRINT "EI"
12260 GOTO 1020
12270 PRINT "DI"
12280 GOTO 1020
12290 PRINT "POP"
12300 GOTO 1020
12310 PRINT "PUSH"
12320 GOTO 1020
12330 PRINT "ADDC",
12340 GOTO 12560
12350 PRINT "SUBRC",
12360 GOTO 12560
12370 PRINT "RORC",
12380 GOTO 1020
12390 PRINT "SUBC",
```

```
12400 GOTO 12560
12410 PRINT "PUSH",".T"
12420 GOTO 1020
12430 PRINT "PUSH",".XP"
12440 GOTO 1020
12450 PRINT "CLR",".Q"
12460 GOTO 1020
12470 PRINT "SET","A"
12480 GOTO 1020
12530 PRINT "ROLC"
12540 GOTO 1020
12560 IF W=0 THEN GOTO 1010
12570 PRINT "#";
12580 LET H=L
12590 GOSUB 8880
12610 GOTO 1010
13210 PRINT "SHL"
13220 GOTO 1020
13850 PRINT "WAIT"
13860 GOTO 1020
13930 PRINT "NOP"
13940 GOTO 1020
13950 PRINT "SKNQ"
13960 GOTO 1020
13970 PRINT "SKNZ"
13980 GOTO 1020
13990 PRINT "SKNC"
14000 GOTO 1020
14010 PRINT "SKIP",2
14020 GOTO 1020
14090 PRINT "SKIE"
14100 GOTO 1020
14110 PRINT "SKQ"
14120 GOTO 1020
14130 PRINT "SKZ"
14140 GOTO 1020
14150 PRINT "SKC"
14160 GOTO 1020
14250 PRINT "CALL",
14260 GOTO 10810
14270 PRINT "RET"
14280 GOTO 1020
14810 PRINT "LD",
14820 GOTO 12560
14830 PRINT "OR",
14840 GOTO 12560
14850 PRINT "AND",
14860 GOTO 12560
14870 PRINT "XOR",
14880 GOTO 12560
14890 PRINT "ADD",
14900 GOTO 12560
14910 PRINT "SUBR",
14920 GOTO 12560
14930 PRINT "SHR"
14940 GOTO 1020
14950 PRINT "SUB",
14960 GOTO 12560
15000 REM 6800 MNEMONICS
15010 GOTO 9980
15030 PRINT "NOP"
```

```
15040 GOTO 1020
15050 REM
15070 REM
15090 REM
15110 GOTO 9980
15130 PRINT "MOV",".A,.P"
15140 GOTO 1020
15150 PRINT "MOV",".P,.A"
15160 GOTO 1020
15170 PRINT "INC",".X"
15180 GOTO 1020
15190 PRINT "DEC",".X"
15200 GOTO 1020
15210 PRINT "CLRV"
15220 GOTO 1020
15230 PRINT "SETV"
15240 GOTO 1020
15250 PRINT "CLRC"
15260 GOTO 1020
15270 PRINT "SETC"
15280 GOTO 1020
15290 PRINT "EI"
15300 GOTO 1020
15310 PRINT "DI"
15320 GOTO 1020
15330 PRINT "SUB",".B,.A"
15340 GOTO 1020
15350 PRINT "CMP",".B,.A"
15360 GOTO 1020
15370 REM
15390 REM
15410 REM
15430 GOTO 9980
15450 PRINT "MOV",".A,.B"
15460 GOTO 1020
15470 PRINT "MOV",".B,.A"
15480 GOTO 1020
15490 GOTO 9980
15510 PRINT "ADJ"
15520 GOTO 1020
15530 GOTO 9980
15550 PRINT "ADD",".B,.A"
15560 GOTO 1020
15570 REM
15590 REM
15610 REM
15630 GOTO 9980
15650 PRINT "BR",
15660 GOTO 18010
15670 PRINT "CALL",
15680 GOTO 18010
15690 PRINT "BH",
15700 GOTO 18010
15710 PRINT "BNH",
15720 GOTO 18010
15730 PRINT "BNC",
15740 GOTO 18010
15750 PRINT "BC",
15760 GOTO 18010
15770 PRINT "BNZ",
15780 GOTO 18010
```

```
15790 PRINT "BZ",
15800 GOTO 18010
15810 PRINT "BNV",
15820 GOTO 18010
15830 PRINT "BV,
15840 GOTO 18010
15850 PRINT "BP",
15860 GOTO 18010
15870 PRINT "BM",
15880 GOTO 18010
15890 PRINT "BGE",
15900 GOTO 18010
15910 PRINT "BLT",
15920 GOTO 18010
15930 PRINT "BGT",
15940 GOTO 18010
15950 PRINT "BLE",
15960 GOTO 18010
15970 PRINT "MOV",".S,.X"
15980 GOTO 1020
15990 PRINT "INC",".S"
16000 GOTO 1020
16010 PRINT "POP",".A"
16020 GOTO 1020
16030 PRINT "POP",".B"
16040 GOTO 1020
16050 PRINT "DEC",".S"
16060 GOTO 1020
16070 PRINT "MOV",".X,.S"
16080 GOTO 1020
16090 PRINT "PUSH",".A"
16100 GOTO 1020
16110 PRINT "PUSH",".B"
16120 GOTO 1020
16130 GOTO 9980
16150 PRINT "RET"
16160 GOTO 1020
16170 GOTO 9980
16190 PRINT "RETI"
16200 GOTO 1020
16210 REM
16230 GOTO 9980
16250 PRINT "WAIT"
16260 GOTO 1020
16270 PRINT "BRK"
16280 GOTO 1020
16290 PRINT "NEG",
16300 GOTO 16970
16310 REM
16330 GOTO 9980
16350 PRINT "NOT",
16360 GOTO 16970
16370 PRINT "SHR",
16380 GOTO 16970
16390 GOTO 9980
16410 PRINT "ROR",
16420 GOTO 16970
16430 PRINT "SHRA",
16440 GOTO 16970
16450 PRINT "SHL",
16460 GOTO 16970
```

```
16470 PRINT "ROL",
16480 GOTO 16970
16490 PRINT "DEC",
16500 GOTO 16970
16510 GOTO 9980
16530 PRINT "INC",
16540 GOTO 16970
16550 PRINT "TST",
16560 GOTO 16970
16570 PRINT "BR",
16580 GOTO 16970
16590 PRINT "CLR",
16600 GOTO 16970
16610 PRINT "SUB",
16620 GOTO 17010
16630 GOTO 16850
16650 PRINT "SUBC",
16660 GOTO 17010
16670 GOTO 9980
16690 PRINT "AND",
16700 GOTO 17010
16710 PRINT "TST",
16720 GOTO 16950
16730 GOTO 16890
16750 GOTO 16910
16770 PRINT "XOR",
16780 GOTO 17010
16790 PRINT "ADDC",
16800 GOTO 17010
16810 PRINT "OR",
16820 GOTO 17010
16830 PRINT "ADD",
16840 GOTO 17010
16850 PRINT "CMP",
16860 GOTO 16950
16870 PRINT "CALL",
16880 GOTO 17050
16890 PRINT "LD",
16900 GOTO 17010
16910 PRINT "ST",
16950 GOSUB 17110
16960 PRINT ",";
16970 IF I>5 THEN GOTO 17050
16980 GOSUB I*20+17050
16990 GOTO 1010
17010 GOSUB 17310
17020 PRINT ",";
17030 GOSUB 17110
17040 GOTO 1010
17050 GOSUB 17310
17060 GOTO 1010
17100 REM PRINT SELECTED REGISTER
17110 IF O>=342 THEN GOTO 17170
17120 IF I>11 THEN GOTO 17150
17130 PRINT ".A";
17140 RETURN
17150 PRINT ".B";
17160 RETURN
17170 IF O>342 THEN IF I<12 THEN GOTO 17210
17180 PRINT ".X";
17190 RETURN
```

```
17210 PRINT ".S";
17220 RETURN
17300 REM PRINT ADDRESSING MODE
17310 LET H=L
17320 GOTO (I-I/4*4)*100+17410
17410 PRINT "#";
17420 GOTO 8950-W*70
17510 PRINT "!";
17520 GOTO 8810
17610 PRINT L;"(.X)";
17620 RETURN
17710 PRINT "/";
17720 GOTO 8810
18010 PRINT "$";
18020 LET H=P+L-L/128*256
18030 GOSUB 8810
18040 GOTO 1010
20000 REM 6502 MNEMONICS
20010 GOTO 9980
20030 PRINT "TST",
20040 GOTO 21310
20050 PRINT "BR",
20060 GOTO 21310
20070 PRINT "BR","@";
20080 GOTO 21310
20090 PRINT "ST",".Y,";
20100 GOTO 21310
20110 PRINT "LD",
20120 GOTO 21310
20130 PRINT "CMP",".Y,";
20140 GOTO 21310
20150 PRINT "CMP",".X,";
20160 GOTO 21310
20170 PRINT "OR",
20180 GOTO 21310
20190 PRINT "AND",
20200 GOTO 21310
20210 PRINT "XOR",
20220 GOTO 21310
20230 PRINT "ADDC",
20240 GOTO 21310
20250 PRINT "ST",".A,";
20260 GOTO 21310
20270 PRINT "LD",
20280 GOTO 21310
20290 PRINT "CMP",".A,";
20300 GOTO 21310
20310 PRINT "SUBC",
20320 GOTO 21310
20330 PRINT "SHL",
20340 GOTO 21310
20350 PRINT "ROL",
20360 GOTO 21310
20370 PRINT "SHR",
20380 GOTO 21310
20390 PRINT "ROR",
20400 GOTO 21310
20410 PRINT "ST",".X,";
20420 GOTO 21310
20430 PRINT "LD",
20440 GOTO 21310
```

```
20450 PRINT "DEC",
20460 GOTO 21310
20470 PRINT "INC",
20480 GOTO 21310
20490 PRINT "BP",
20500 GOTO 18010
20510 PRINT "BM",
20520 GOTO 18010
20530 PRINT "BNV",
20540 GOTO 18010
20550 PRINT "BV,
20560 GOTO 18010
20570 PRINT "BNC",
20580 GOTO 18010
20590 PRINT "BC",
20600 GOTO 18010
20610 PRINT "BNZ",
20620 GOTO 18010
20630 PRINT "BZ",
20640 GOTO 18010
20650 PRINT "PUSH",".P"
20660 GOTO 1020
20670 PRINT "POP",".P"
20680 GOTO 1020
20690 PRINT "PUSH",".A"
20700 GOTO 1020
20710 PRINT "POP",".A"
20720 GOTO 1020
20730 PRINT "DEC",".Y"
20740 GOTO 1020
20750 PRINT "MOV",".A,.Y"
20760 GOTO 1020
20770 PRINT "INC",".Y"
20780 GOTO 1020
20790 PRINT "INC",".X"
20800 GOTO 1020
20810 PRINT "CLRC"
20820 GOTO 1020
20830 PRINT "SETC"
20840 GOTO 1020
20850 PRINT "EI"
20860 GOTO 1020
20870 PRINT "DI"
20880 GOTO 1020
20890 PRINT "MOV",".Y,.A"
20900 GOTO 1020
20910 PRINT "CLRV"
20920 GOTO 1020
20930 PRINT "CLRD"
20940 GOTO 1020
20950 PRINT "SETD"
20960 GOTO 1020
20970 PRINT "BRK"
20980 GOTO 1020
20990 PRINT "CALL",
21000 GOTO 21550
21010 PRINT "RETI"
21020 GOTO 1020
21030 PRINT "RET"
21040 GOTO 1020
21050 PRINT "MOV",".X,.A"
```

```
21060 GOTO 1020
21070 PRINT "MOV",".A,.X"
21080 GOTO 1020
21090 PRINT "DEC",".X"
21100 GOTO 1020
21110 PRINT "NOP"
21120 GOTO 1020
21130 REM
21150 REM
21170 REM
21190 GOTO 9980
21210 PRINT "MOV",".X,.S"
21220 GOTO 1020
21230 PRINT "MOV",".S,.X"
21240 GOTO 1020
21250 REM
21270 GOTO 9980
21300 REM ADDRESSING MODES
21310 LET H=L
21320 GOSUB I*50+21410
21330 IF O-O/8*8=1 THEN GOSUB O/4*10+20090
21340 GOTO 1010
21350 PRINT ",.Y";
21360 RETURN
21370 PRINT ",.A;
21380 RETURN
21390 PRINT ",.X;
21400 RETURN
21410 IF (O-500)/8=1 THEN GOTO 21860
21420 PRINT "#";
21430 GOTO 8880
21460 PRINT "!";
21470 GOTO 8880
21510 IF O<516 THEN GOTO 21420
21520 PRINT ".A";
21530 RETURN
21550 LET H=L
21560 PRINT "/";
21570 GOTO 8810
21610 PRINT "@!";
21620 GOTO 21770
21660 PRINT "!";
21670 GOTO 21770
21710 REM
21760 PRINT "/";
21770 GOSUB 8810
21780 IF I/2*2=I THEN GOTO 21810
21790 PRINT "(.X)";
21800 RETURN
21810 PRINT "(.Y)";
21820 RETURN
21860 PRINT "!";
21870 GOSUB 8880
21880 PRINT "(.X)@";
21890 RETURN
```

APPENDIX -- Binary Search Speedup Code

1802 [this code re-formated by Richard Peters]

```
0000 ;       .. TINY BASIC BINARY SEARCH SPEEDUP -- 81 MAY 9
0000 :       ... NOTE: one-byte Lines with Line in [3328..3583] bomb.
05DA C000C0;       LBR SRCH
00C0 ;             ORG 192                .. ASSUME THIS IS VACANT
00C0 BB;     SRCH: PHI 11
00C1 4D;           LDA 13
00C2 AB;           PLO 11
00C3 1D;           INC 13
00C4 1D;           INC 13
00C5 4D;           LDA 13                 .. GET PROG END
00C6 FF01;         SMI 1
00C8 B8;           PHI 8
00C9 E2;     HALF: SEX 2                  .. BISECT THE REGION
00CA 9B;           GHI 11
00CB 52;           STR 2
00CC 98;           GHI 8
00CD F7;           SM
00CE CB05DD;       LBNF FLINE
00D1 F6;           SHR
00D2 C205DD;       LBZ FLINE              .. TOO SMALL ...
00D5 F4;           ADD
00D6 BF;           PHI 15
00D7 F800;         LDI 0
00D9 AF;           PLO 15
00DA 1F;           INC 15
00DB 4F;     FLNO: LDA 15                 .. FIND A LINE NUMBER
00DC FBOD;         XRI 13
00DE 3ADB;         BNZ FLNO
00E0 1F;           INC 15
00E1 7C01;         ADCI 1
00E3 F6;           SHR
00E4 33DA;         BDF FLNO-1             .. (SKIP TWO TO SYNC)
00E6 EF;           SEX 15
00E7 4F;           LDA 15                 .. END?
00E8 Fl;           OR
00E9 32F8;         BZ HIGH                .. YES.
00EB 8A;           GLO 10
00EC F7;           SM
00ED 2F;           DEC 15
00EE 9A;           GHI 10
00EF 77;           SMB
00F0 3BF8;         BNF HIGH
00F2 8F;           GLO 15                 .. TOO LOW
00F3 AB;           PLO 11
00F4 9F;           GHI 15
00F5 BB;           PHI 11
00F6 30C9;         BR HALF
00F8 9F;     HIGH: GHI 15                 .. TOO HIGH
00F9 30C8;         BR HALF-1
```

6502 [this code re-formated by Richard Peters]

```
           ; BINARY SEARCH SPEEDUP
0200=      COLD    EQU     $200
0476               ORG     COLD+$276
0476 4CC70A        JMP     BINARY
047C=      FLINE   EQU     *+3
0414=      CHAR    EQU     COLD+$214
```

```
0020=       BASIC   EQU     32
0024=       MEND    EQU     36
00BC=       TEMP    EQU     $BC
002C=       BP      EQU     44
00C9                ORG     $C9
00C9=       LO      EQU     *              ;3 BYTES SCRATCHPAD
00CA=       HI      EQU     *+1
0AC7                ORG     COLD+$8C7      ;NEED 73 BYTES ANYWHERE
0AC7 85C8   BINARY  STA     LO-1
0AC9 A521           LDA     BASIC+1
0ACB 85C9           STA     LO
0ACD A525           LDA     MEND+1
0ACF 85CA   SLICE   STA     HI
0AD1 38             SEC
0AD2 E5C9           SBC     LO
0AD4 4A             LSR     A
0AD5 F030           BEQ     DONE
0AD7 18             CLC
0AD8 65C9           ADC     LO
0ADA 852D           STA     BP+1
0ADC A5C8           LDA     LO-1
0ADE 852C           STA     BP
0AE0 201404         JSR     CHAR
0AE3 D0FB           BNE     *-3
0AE5 201404         JSR     CHAR
0AE8 201404         JSR     CHAR
0AEB 201404         JSR     CHAR
0AEE D0FB           BNE     *-3
0AF0 A5BC           LDA     TEMP
0AF2 D12C           CMP     (BP),Y
0AF4 C8             INY
0AF5 A5BD           LDA     TEMP+1 ;TEMP0 corrected by Lee Hart
0AF7 F12C           SBC     (BP),Y
0AF9 A52D           LDA     BP+1
0AF8 90D2           BCC     SLICE
0AFD 85C9           STA     LO
0AFF A52C           LDA     BP
0B01 85C8           STA     LO-1
0B03 A5CA           LDA     HI
0B05 D0CB           BNE     SLICE+3
0B07 A5C8   DONE    LDA     LO-1
0B09 852C           STA     BP
0B0B A5C9           LDA     LO
0B0D 4C7A04         JMP     FLINE-2
0285                ORG     COLD+$85
0285 A9 010         LDA     16             ; NO OVERLAP USER
```

6800 [this code re-formated by Richard Peters]

```
                    *  BINARY SEARCH SPEEDUP
                           ORG     COLD+$425
0525 7E 0900               JMP     BINARY
                    FLINE  EQU     COLD+$429
                    TEMP   EQU     $BC
                    BP     EQU     $2C
                    BASIC  EQU     $20
                    MEND   EQU     $24
                    LO     EQU     $C8           4 BYTES SCRATCH
                    HI     EQU     $CA
```

```
                            ORG     $0900         NEED 65 BYTES
ANYWHERE
0900 DE 20          BINARY  LDX     BASIC
0902 DF C8                  STX     LO
0904 96 24                  LDA A   MEND
0906 97 CA                  STA A   HI
0908 90 C8          SLICE   SUB A   LO
090A 44                     LSR A
090B 27 2D                  BEQ     DONE
090D 9B C8                  ADD A   LO
090F 97 2C                  STA A   BP
0911 7F 002D                CLR     BP+l
0914 DE 2C                  LDX     BP
0916 86 0D                  LDA A   #13
0918 08                     INX
0919 Al 00                  CMP A   X
091B 26 FB                  BNE     *-3
091D 08                     INX
091E 08                     INX
091F 08                     INX
0920 Al 00                  CMP A   X
0922 26 FB                  BNE     *-3
0924 08                     INX
0925 96 BD                  LDA A   TEMP+l
0927 A0 01                  SUB A   1,X
0929 96 BC                  LDA A   TEMP
092B A2 00                  SBC A   X
092D 96 2C                  LDA A   BP
092F 25 03                  BCS     TOOHI
0931 DF C8                  STX     LO
0933 8C                     SKIP    2
0934 DF CA          TOOHI   STX     HI
0936 96 CA                  LDA A   HI
0938 20 CE                  BRA     SLICE
093A DE C8          DONE    LDX     LO
093C DF 2C                  STX     BP
093E 7E 0529                JMP     FLINE
                            ORG     $0200          IF IN USER SPACE,
0200 CE 0950                LDX     #USER          MOVE IT OVER
```