USEFUL PROGRAMS

The next few pages contain programs designed to help you in your everyday computing. We've made them fairly simple, so you can make any changes appropriate to your work. Have fun!

3.1 ERROR HANDLING

"The first step is the hardest" is a well-known maxim. A fellow named Murphy said it best: "If something *can* go wrong, it *will*." Both these sayings seem to apply to the programmer. In the beginning phases of programming, we say to ourselves, "What are all these error messages?" The more advanced programmer has the advantage of knowing some of the error messages, but he still ends up doing a lot of debugging.

Some BASIC versions have a HELP function, and the C-128 is no exception. This function finds the errant line, and shows the error in reverse video (40-column mode) or underscoring (80-columns). You can also use the command within a BASIC program.

Although the C-128 has no ON ERROR GOTO command, it does have a TRAP command. Type the command, followed by a line number, and an error will cause the system to jump to that line.

USEFUL PROGRAMS

The next few pages contain programs designed to help you in your everyday computing. We've made them fairly simple, so you can make any changes appropriate to your work. Have fun!

3.1 ERROR HANDLING

"The first step is the hardest" is a well-known maxim. A fellow named Murphy said it best: "If something *can* go wrong, it *will*." Both these sayings seem to apply to the programmer. In the beginning phases of programming, we say to ourselves, "What are all these error messages?" The more advanced programmer has the advantage of knowing some of the error messages, but he still ends up doing a lot of debugging.

Some BASIC versions have a HELP function, and the C-128 is no exception. This function finds the errant line, and shows the error in reverse video (40-column mode) or underscoring (80-columns). You can also use the command within a BASIC program.

Although the C-128 has no ON ERROR GOTO command, it does have a TRAP command. Type the command, followed by a line number, and an error will cause the system to jump to that line.

Next, we have the ERR$ variable. The variable ER stores the numbers of error, and PRINT ERR$(ER) prints the corresponding error messages. Finally, there's the variable EL, which contains the line number of the "bad" line.

If the program stops due to a TRAP command, there are three possible methods of continuing the program, all by means of the RESUME command. Typing in RESUME alone causes the computer to continue at the line where the error occurred. Giving a line number with RESUME causes the program to continue from the line specified. Finally, you can type in NEXT after RESUME; then the system starts from the line after the "bad" line.

If you want to watch your program step-by-step, if only to see what the errors are doing, you have TRON and TROFF at hand. These control the TRACE function.

Now for a some applications using these commands:
1. For Beginners

Type in this program.

```
10 TRAP 1000
20 :
30 PRINT:PRIND: REM PRIND NOT PRINT
40 PRINT CHR$(147)"(C) 1985 BY WERNER
50 FOR T=1 TO 10:PRINT:NEXT Y
60 PRINT"OK"
70 END
80 :
1000 REM ERROR JMP
1010 PRINT"(YEL 1)"ERR$(ER)" ERROR IN "EL;
1020 HELP
```



```
1030 PRINT"(CRSR DOWN)(CYAN)'G'O ON OR 'E'ND?"
1040 GET KEY A$
1050 IF A$="E" THEN END
1060 IF A$="G" THEN RESUME NEXT
1070 GOTO 1040
```

The program jumps to line 1000 whenever it runs into an error (e.g., at line 30) as a result of the TRAP command at line 10. The error trap routine gives you the type of error and the line number at which it occurred. That's all the routine does; it's up to you to find the error (HELP key) and correct it.

A tip for machine language programmers: You can develop an "error statement" program for machine language errors. The vector for printing BASIC errors is at locations $0300/$0301, and the error number is given in the X-register. The X-register presents the value $80 when no errors exist. On the next page are the errors and their values:

NUMBER (hex)	NUMBER (dec)	ERROR
$01	1	TOO MANY FILES
$02	2	FILE OPEN
$03	3	FILE NOT OPEN
$04	4	FILE NOT FOUND
$05	5	DEVICE NOT PRESENT
$06	6	NOT INPUT FILE
$07	7	NOT OUTPUT FILE
$08	8	MISSING FILENAME
$09	9	ILLEGAL DEVICE NUMBER
$0A	10	NEXT WITHOUT FOR
$0B	11	SYNTAX
$0C	12	RETURN WITHOUT GOSUB
$0D	13	OUT OF DATA
$0E	14	ILLEGAL QUANTITY
$0F	15	OVERFLOW
$10	16	OUT OF MEMORY
$11	17	UNDEF'D STATEMENT
$12	18	BAD SUBSCRIPT
$13	19	REDIM'D ARRAY
$14	20	DIVISION BY ZERO
$15	21	ILLEGAL DIRECT
$16	22	TYPE MISMATCH
$17	23	STRING TOO LONG
$18	24	FILE DATA
$19	25	FORMULA TOO COMPLEX
$1A	26	CAN'T CONTINUE
$1B	27	UNDEF'D FUNCTION
$1C	28	VERIFY
$1D	29	LOAD
$1E	30	BREAK
$1F	31	CAN'T RESUME
$20	32	LOOP NOT FOUND
$21	33	LOOP WITHOUT DO
$22	34	DIRECT MODE ONLY
$23	35	NO GRAPHICS AREA
$24	36	BAD DISK
$25	37	BEND NOT FOUND
$26	38	LINE NUMBER TOO LARGE
$27	39	UNRESOLVED REFERENCE
$28	40	UNIMPLEMENTED COMMAND
$29	41	FILE READ

2. The Use of RESUME

Typing RESUME without a line number or NEXT can result in an endless loop, which probably wasn't what the designers of BASIC 7.0 had in mind. Try this program out:

```
1     TRAP 1000
10    CATALOG
20    END
1000  IF ER=5 THEN PRINT CHR$(19);"PLEASE TURN
      DISK DRIVE ON": RESUME
1010  END
```

Turn your disk drive off and run the program, which looks for the disk directory. Since the drive is off, the system responds with "DEVICE NOT PRESENT" and jumps to line 1000. Seeing error number 5, the program asks you to switch the disk drive on. The RESUME command causes the program to continue displaying this message until the drive is finally turned on.

Now for a sample of RESUME with a line number:

```
1     TRAP 1000
10    INPUT"FILENAME";NA$
20    INPUT"DEVICE NUMBER";DV
30    LOAD NA$,DV
40    END
1000  IF ER=4 THEN PRINT"THERE IS NO SUCH
      FILENAME":RESUME 10
1010  IF ER=5 THEN PRINT"PLEASE TURN DISK DRIVE
      ON":RESUME
1020  IF ER=8 THEN PRINT"PLEASE GIVE ME A
      FILENAME":RESUME 10
1030  IF ER=9 THEN PRINT"PICK A DEVICE NR FROM
      8 - 15": RESUME 20
1040  END
```

Essentially, this little routine can make a program "foolproof". That is, it won't stop if you give it a bad filename, have your disk drive turned off, press <RETURN> on the filename prompt, or give it a "non-disk" device address -- and it will tell you what the problem is, in simple English.

3.2 *LISTER* - A LISTING UTILITY

You've just spent your last dollar on a computer magazine. You run home, switch on the computer, and start typing in page upon page of program listing. Wait a minute; there's something wrong here. The problem is, as in so many cases, the printout in the magazine is from a Commodore printer. You know what that means: Unreadable control characters -- what's this reversed heart? And this reverse-video cross? Eventually you give up and either buy the companion disk, or just stop buying the magazine.

Some computer magazines have heard the cries of their readers. These publishers use a special listing technique to alter the program listing so that, instead of control characters, a descriptive term or abbreviation appear (e.g., "{CLR}" or "{5 DOWN}") in the listing.

There's an advantage to this: It increases the marketability of the magazine. It's a lot easier for a user to just punch CRSR DOWN from the listing, rather than look at the graphic symbol, flip through the manual, figure out what that symbol actually does, then type it in. On the other hand, there's a greater frustration that isn't cured. How many spaces do I put here? Are there 39 or 40 (or 38)? How many CRSR DOWNs? And so on....

Our *LISTER* fixes that problem: Any spaces and control characters in quotes and in number will be printed out with their amounts. Plus, each command will be separated by a space. This can be a godsend to the beginner. It's a lot easier to read;

```
FOR ES = 1 TO P: W = PEEK (ES) AND NOT 5 OR 8
```

than it is to read:

```
FORES=1TOP:W=PEEK(ES)ANDNOT5OR8.
```

The design of the printout is familiar to those of you who are also Apple users: commands are indented, variables are not.

Now for the program: Type it in and SAVE it with the name *LISTER* before trying it out.

```
1     REM  LISTING OF LISTER CHAPTER 3.2.A
5     PRINT "{CLR HOME}PLEASE WAIT..."
10    DIM TK$(255),FE$(29),CE$(8),SZ$(159)
20    FOR Z = 0 TO 125: READ WE$,A$: TK$( DEC
      (WE$)) = A$:  NEXT Z
22    FOR Z = 0 TO 27:  READ WE$,A$: FE$( DEC
      (WE$)) = A$:  NEXT Z
24    FOR Z = 0 TO 7:  READ WE$,A$: CE$( DEC (WE$))
      = A$:  NEXT Z
26    FOR Z = 0 TO 25:  READ WE$,A$: SZ$( DEC
      (WE$)) = A$:  NEXT Z
30    AN = 7168
40    PRINT "{CLR HOME}'S'CREEN OR 'P'RINTER?"
42    DR = 0:  GET  KEY A$:  IF A$ = "D" THEN DR =
      1:  GOTO 50
44    IF A$ <  > "B" THEN 40
50    PRINT "{CRSR DOWN}'U'PPEROR'L'OWERCASE?"
```

```
52    GET  KEY A$:  IF A$ = "L" THEN  PRINT
      CHR$ (14):  GOTO 60
54    IF A$ <  > "U" THEN  PRINT "{CRSR UP}";:
      GOTO 50
60    INPUT "{LINE FEED}NAME OF THE PROGRAM";NA$
70    IF DR THEN  CLOSE 4:  OPEN 4,4,0
72    PRINT :  PRINT NA$:  PRINT
73    IF DR THEN  PRINT# 4:  PRINT# 4,NA$: PRINT# 4
100   FL = 0: AN = AN + 3
102   LO = PEEK (AN): AN = AN + 1: HI = PEEK (AN):
      ZE = LO + 256 * HI
105   PRINT ZE;" ";
106   IF DR THEN  PRINT# 4,ZE;" ";
110   AN = AN + 1
115   WE =  PEEK (AN):  IF WE = 0 THEN 1000
120   IF WE = 32 THEN 2000
122   IF WE = 34 THEN 7000
123   IF WE = 58 THEN 8000
125   IF WE =  DEC ("FE") THEN 3000
130   IF WE =  DEC ("CE") THEN 4000
135   IF FF THEN 6000
140   GOTO 5000
999   :
1000  REM  NEW LINE
1009  IF DR THEN  PRINT# 4
1010  PRINT : IF PEEK (AN + 1) = 0 AND PEEK (AN +
      2) = 0 THEN  CLOSE 4:  END
1020  GOTO 100
1999  :
2000  REM  EMPTY LINE
2010  IF FF THEN 6030
2020  IF FL THEN  PRINT " ";
2021  IF DR AND FL THEN  PRINT# 4," ";
2030  GOTO 110
2999  :
3000  REM  $FE
3010  AN = AN + 1: WE =  PEEK (AN)
3020  PRINT FE$(WE);" ";
3021  IF DR THEN  PRINT# 4,FE$(WE);" ";
3030  GOTO 110
3999  :
4000  REM  $CE
4010  AN = AN + 1: WE =  PEEK (AN)
```



```
4020  PRINT CE$(WE);" ";
4021  IF DR THEN  PRINT# 4,CE$(WE);" ";
4030  GOTO 110
4999  :
5000  REM  BEGINNING OF STRING
5005  IF WE = 143 THEN FL = 1
5010  IF TK$(WE) <  > "" THEN 5040
5020  PRINT  CHR$ (WE);
5021  IF DR THEN  PRINT# 4, CHR$ (WE);
5030  GOTO 110
5040  PRINT " ";TK$(WE);" ";
5041  IF DR THEN  PRINT# 4," ";TK$(WE);" ";
5050  GOTO 110
5999  :
6000  REM  INNER PART OF STRINGS
6005  IF WE <  > 255 THEN 6010
6006  PRINT "π";
6007  IF DR THEN  PRINT# 4,"π";
6008  GOTO 110
6010  IF SZ$(WE) <  > "" THEN 6030
6020  PRINT  CHR$ (WE);
6021  IF DR THEN  PRINT# 4, CHR$ (WE);
6025  GOTO 110
6030  ZA = 1
6040  IF  PEEK (AN + 1) <  > WE THEN 6060
6050  AN = AN + 1: ZA = ZA + 1:  GOTO 6040
6060  PRINT  CHR$ (18);"[";SZ$(WE); STR$ (ZA);"]";
      CHR$ (146);
6061  IF DR THEN  PRINT# 4," [";SZ$(WE); STR$
      (ZA);"] ";
6070  GOTO 110
6999  :
7000  REM  QUOTATION MARK
7010  FF = XOR (FF,255)
7020  PRINT  CHR$ (34);:  POKE 244,0
7021  IF DR THEN  PRINT# 4, CHR$ (34);
7030  GOTO 110
7999  :
8000  REM  COLON
8010  PRINT  CHR$ (58);" ";
8011  IF DR THEN  PRINT# 4, CHR$ (58);" ";
8020  GOTO 110
9999  :
```

```
10000 DATA "80","END","81","FOR","82","NEXT",
      "83","DATA","84","INPUT#"
10010 DATA "85","INPUT","86","DIM","87","READ",
      "88","LET","89","GOTO"
10020 DATA "8A","RUN","8B","IF","8C","RESTORE",
      "8D","GOSUB","8E","RETURN"
10030 DATA "8F","REM","90","STOP","91","ON",
      "92","WAIT","93","LOAD","94","SAVE"
10040 DATA "95","VERIFY","96","DEF","97","POKE",
      "98","PRINT#","99","PRINT"
10050 DATA "9A","CONT","9B","LIST","9C","CLR",
      "9D","CMD","9E","SYS","9F","OPEN"
10060 DATA "A0","CLOSE","A1","GET","A2","NEW",
      "A3","TAB(","A4","TO","A5","FN"
10070 DATA "A6","SPC(","A7","THEN","A8","NOT",
      "A9","STEP","AA","+","AB","-"
10080 DATA "AC","*","AD","/","AE","^","AF","AND",
      "B0","OR","B1",">","B2","="
10090 DATA "B3","<","B4","SGN","B5","INT",
      "B6","ABS","B7","USR","B8","FRE"
10100 DATA "B9","POS","BA","SQR","BB","RND",
      "BC","LOG","BD","EXP","BE","COS"
10110 DATA "BF","SIN","C0","TAN","C1","ATN",
      "C2","PEEK","C3","LEN","C4","STR$"
10120 DATA "C5","VAL","C6","ASC","C7","CHR$",
      "C8","LEFT$","C9","RIGHT$"
10130 DATA "CA","MID$","CB","GO","CC","RGR",
      "CD","RCLR","CF","JOY","D0","RDOT"
10140 DATA "D1","DEC","D2","HEX$","D3","ERR$",
      "D4","INSTR","D5","ELSE"
10150 DATA "D6","RESUME","D7","TRAP","D8","TRON",
      "D9","TROFF","DA","SOUND"
10160 DATA "DB","VOL","DC","AUTO","DD","PUDEF",
      "DE","GRAPHIC","DF","PAINT"
10170 DATA "E0","CHAR","E1","BOX","E2","CIRCLE",
      "E3","GSHAPE","E4","SSHAPE"
10180 DATA "E5","DRAW","E6","LOCATE","E7","COLOR",
      "E8","SCNCLR","E9","SCALE"
10190 DATA "EA","HELP","EB","DO","EC","LOOP",
      "ED","EXIT","EE","DIRECTORY"
10200 DATA "EF","DSAVE","F0","DLOAD","F1",
      "HEADER","F2","SCRATCH"
```



```
10210 DATA "F3","COLLECT","F4","COPY","F5",
      "RENAME","F6","BACKUP","F7","DELETE"
10220 DATA "F8","RENUMBER","F9","KEY","FA",
      "MONITOR","FB","USING","FC","WHILE"
10230 DATA "FD","UNTIL","FF","π"
11000 DATA "02","BANK","03","FILTER","04",
      "PLAY","05","TEMPO","06","MOVSPR"
11010 DATA "07","SPRITE","08","SPRCOLOR",
      "09","RREG","0A","ENVELOPE"
11020 DATA "0B","SLEEP","0C","CATALOG","0D",
      "DOPEN","0E","APPEND","0F","DCLOSE"
11030 DATA "10","BSAVE","11","BLOAD","12",
      "RECORD","13","CONCAT","14","DVERIFY"
11040 DATA "15","DCLEAR","16","SPRSAV","17",
      "COLLISION","18","BEGIN"
11050 DATA "19","BEND","1A","WINDOW","1B",
      "BOOT","1C","WIDTH","1D","SPRDEF"
12000 DATA "00","RWINDOW","02","POT","03","BUMP",
      "04","PEN","05","RSPPOS"
12010 DATA "06","RSPRITE","07","RSPCOLOR",
      "08","XOR"
13000 DATA "03","RUN STOP","05","WHT",
      "0A","LINE FEED","11","CRSR DOWN"
13010 DATA "12","RVS ON","13","HOME",
      "1C","RED","1D","CRSR RIGHT","1E","GRN"
13020 DATA "1F","BLU","20","SPACE","81","ORNG",
      "90","BLK","91","CRSR UP"
13030 DATA "92","RVS OFF","93","CLR HOME",
      "95","BRN","96","L RED"
13040 DATA "97","D GRY","98","M GRY","99","L GRN",
      "9A","L BLU"
13050 DATA "9B","L GRY","9C","PUR",
      "9D","CRSR LEFT","9E","YEL","9F","CYN"
```

Now RUN it. After a short wait, you'll be asked for some input parameters. For the moment, press "S" for screen and "U" for uppercase. Then enter "*LISTER*"; this isn't for loading from disk -- just a heading for the listing. If you typed the program in correctly, the system will print "*LISTER*" on the screen and the program will be listed to the screen;

control characters should be spelled out in reverse on the screen. Next, switch on your printer and run the program again to get a hardcopy listing of the program.

When you want to use this program to print other listings, you'll need to do the following:

A) Move the start of BASIC - set pointers $2D (45) and $2E (46) higher than the length of the program to be listed and place a zero in the first location of the new start of BASIC. Here are the commands we used:

```
POKE  46, 192 : POKE 49152, 0: NEW
```

This moves the start of BASIC to $C000. Try the FRE(0) function after typing in these commands, less than 16K is free. This is more than enough memory to run our converter program.

B) LOAD the listing converter program with `LOAD"LISTER",8`. The { ,8 } is important! This command loads the listing converter program into memory at the new start of BASIC.

C) Load in the program you wish listed with the command LOAD "program-name" ,8,1. The { ,8,1 } is very important here. This command loads the program into memory at the normal start of BASIC. Because you have moved the start of BASIC the *LISTER* program is not overwritten. LIST the program to be sure.

D) RUN the *LISTER* to get a formatted listing.

You could revise the *LISTER* program so that it reads the program directly from disk, which would save the trouble of moving the start-of-BASIC. We tried that with the program listings in this book. They were transferred to this book with a modified version of the above program that used the RS-232 adapter to send the listing to another computer. The memory resident version is much faster.

A few things you should know about the memory resident *LISTER*. It looks at and interprets every byte from the normal start of BASIC. If it finds a zero, it checks the next few bytes; a set of three zeroes indicates the end of the program, and the *LISTER* halts. Every character is tested to see if it is a token; if not, the character is tested for whether it is within quotes or not. Tokens are printed out as commands. Strings with color graphic symbols will have their colors "spelled out".

Here are a few of the variables used in the program:

DR:	If DR=0, the listing goes to the screen only; if DR=1, then it's sent to the printer and the screen.
FL:	If equal to 1, the characters are placed after a REM.
AN:	Address of bytes read.
WE:	Contents of the above address.
FF:	If 0, the byte is not in quotes; 1, the byte is a string (within quotes).
ZA:	Counter -- counts number of consecutive characters (lines 6040-6070). Determines number of spaces in a string.
TK$(255):	Contains command strings of individual tokens.
FE$(29):	Contains command strings which have a value of $FE in the first byte.

CE$(8): Same as FE$, but for commands beginning with $CE.
SZ$(159): Graphic character codes are held here.

3.3 OLD

What do you do if you accidentally type NEW while a program you haven't saved is in memory? No more program, right? Wrong. All is not lost. You'll remember the programs running around for the C-64 known by various titles (OLD or UNNEW); these programs renewed the bytes at the start of a program that had been zeroed out by an errant NEW command. These two bytes point to the next line number of a program (the address of the next line number, to be more specific). In addition to finding and resetting the starting address, the start-of-variables must be reset as well before the program can be completely recovered.

These OLD routines are fairly long and inaccurate most of the time. There's a much easier way of going about it:

A) Put any number into the two skipped bytes (unequal to 0, of course). This works best when you use the high-byte of start-of-BASIC.

B) Next, call the BASIC interpreter routine that restores the BASIC program lines, and is usually used for inserting program lines in editing. It changes all the vectors (line links) and recalculates appropriately. The link to the first program line, which were zeroes a moment ago, are re-inserted into memory, and recalculated.

C) Start of variables is set at the end of the program.

This process is much simpler on the C-128: Step C) can be eliminated, since variables are stored in Bank 1, and don't collide with Bank 0. You can activate OLD by doing this:

```
POKE PEEK(45)+256*PEEK(46)+1,28:SYS DEC("4F4F")
```

Addresses 45/46 point to start-of-BASIC. Calling this routine restores the pointers. This is what it looks like in machine language.

```
0B00 A0 01    LDY #$01       ;Offset 1
0B02 A9 1C    LDA #$1C       ;28 for BASIC RAM start
0B04 91 2D    STA ($2D),Y    ;and in link high-byte
0B06 20 4F 4F JSR $4F4F      ;Determine links
0B09 60       RTS            ;End
```

You can put the routine anywhere in memory, not just at $0B00.

OLD also works if you've changed the BASIC pointers (say, for the conversion program in chapter 3.2). Not even a RESET can outsmart this program.

There is one exception -- when start-of-BASIC is moved up *and* a RESET is executed; then OLD goes for the normal starting address of BASIC, and won't find it there. If you know where the starting address is, just change the parameters accordingly, and you should be able to recover the program.

3.4 A LITTLE MUSIC ON THE SIDE....

Some folks like music while they're typing. That's great, unless there's no radio or stereo nearby, and most of us aren't born singers. Hiring an in-house pianist can be expensive, and typewriters don't make music. The C-128, however, CAN make music. The next program actually plays a piece of music while you're working with the computer. You can listen through your television or monitor, or run it through your stereo, etc.

The BASIC loader is a long one; the main reason is the 99-note tune within. To play a note, the SID chip needs two numbers per note (high-byte/low-byte) and a duration, which gives us a total of 297 DATA elements. You'll find this DATA at the end of the program.

```
0     REM  IRQ MUSIC
10    FOR I = 5120 TO 5198
20    READ A
30    POKE I,A
40    S = S + A
50    NEXT I
60    IF S <> 8891 THEN PRINT "?ERROR IN DATA":END
70    DATA 120,169,22,141,20,3,169,20
80    DATA 141,21,3,88,160,0
90    DATA 140,243,20,200,140,241,20,96
100   DATA 206,241,20,208,49,72,141,0
110   DATA 255,141,18,212,138,72,174,243,20
120   DATA 189,0,21,141,14,212,189,0
130   DATA 22,141,15,212,189,0,23,141
140   DATA 241,20,169,65,141,18,212,232
150   DATA 56,224,99,144,2,162,0,142
160   DATA 243,20,104,170,104,76,101,250
170   BANK 15
180   POKE 54296,15
190   POKE 54288,255
200   POKE 54289,0
```



```
210   POKE 54291,15
220   POKE 54292,0
230   N = 99:  REM  99 NOTES
240   FOR I = 0 TO N - 1
250   READ A
260   POKE 5376 + I,A
270   READ A
280   POKE 5632 + I,A
290   READ A
300   POKE 5888 + I,A
310   NEXT I
320   POKE 5185,N
330   SYS 5120
340   DATA 10,13,5,10,13,5,10,13,10
350   DATA 10,13,5,162,14,5,162,14,10
360   DATA 247,10,10,10,13,5,10,13,5
370   DATA 103,17,5,137,19,13,237,21,5
380   DATA 237,21,10,237,21,5,237,21,10
390   DATA 137,19,10,137,19,10,103,17,10
400   DATA 103,17,20,237,21,5,237,21,13
410   DATA 59,23,10,237,21,5,237,21,5
420   DATA 237,21,13,137,19,30,237,21,8
430   DATA 137,19,8,137,19,15,103,17,15
440   DATA 103,17,30,103,17,13,10,13,5
450   DATA 10,13,5,10,13,10,162,14,10
460   DATA 103,17,5,10,13,5,10,13,10
470   DATA 10,13,10,103,17,5,137,19,13
480   DATA 137,19,5,237,21,13,237,21,10
490   DATA 137,19,5,137,19,5,137,19,5
500   DATA 103,17,10,103,17,5,103,17,20
510   DATA 237,21,5,237,21,13,59,23,10
520   DATA 237,21,5,237,21,5,237,21,13
530   DATA 137,19,30,237,21,10,137,19,10
540   DATA 137,19,10,103,17,15,103,17,30
550   DATA 237,21,5,137,19,10,103,17,5
560   DATA 103,17,20,237,21,5,20,26,10
570   DATA 69,29,5,69,29,10,20,26,10
580   DATA 20,26,5,20,26,10,237,21,5
590   DATA 237,21,5,137,19,5,103,17,10
600   DATA 162,14,5,10,13,10,237,21,5
610   DATA 237,21,10,103,17,30,237,21,5
620   DATA 237,21,10,59,23,5,59,23,10
630   DATA 237,21,5,237,21,5,237,21,10
```

```
640   DATA 137,19,30,237,21,10,137,19,10
650   DATA 137,19,10,137,19,13,103,17,13
660   DATA 103,17,30,247,10,5,10,13,5
```

This tune may bore you after a while, so we'll now give you some ground rules for putting in your own tunes. A song can have up to 255 notes, with every note entered in this format:

Frequency	(low-byte)
Frequency	**(high-byte)**
Duration	(how long it is to be held)

You can choose your own notes. For example, we called up a frequency of 3338 (in low/high format, 10 and 13) for a G. Our duration was 10 (like an average quarter note). Line 230 will have to be changed to adjust for your song length. When you want to change waveforms, put POKE 7226,WF (waveform) at the end of the program. WF has the following functions:

```
17    TRIANGLE WAVE
33    SAWTOOTH WAVE
65    PULSE WAVE
129   NOISE
```

This program operates voice three ONLY. You could convert the program to include the first and second voices of the SID chip. If you want to program this routine in machine language, we've included that listing below. This routine works with the IRQ vectors:

$1400 (5120)	Routine which moves IRQ vectors to $1415 (5141)
$1416 (5142)	Start of music/IRQ routine till $144E (5198)
$14F1 (5361)	Count duration; when 1CF1=0, go to next note
$14F3 (5363)	Note counter
$1500 (5376)	Memory for low-byte of note to $15FF (5631)
$1600 (5632)	Memory for high-byte of note to $16FF (5887)
$1700 (5888)	Memory for all note values to $17FF (6143)

Once you have a look at the commented listing, you'll have a good idea of the program's inner workings. You can use our BASIC loader, or type this in on your monitor:

```
1400 78        SEI           ;Hinder interrupt
1401 A9 16     LDA #$16      ;Set IRQ vector to
1403 8D 14 03  STA $0314     ;start of the
1406 A9 14     LDA #$14      ;music routine
1408 8D 15 03  STA $0315
140B 58        CLI           ;Enable interrupt
140C A0 00     LDY #$00
140E 8C F3 14  STY $14F3
1411 C8        INY
1412 8C F1 14  STY $14F1
1415 60        RTS           ;Return to BASIC
1416 CE F1 14  DEC 14F1      ;Decrement counter
1419 D0 31     BNE 144C      ;Counter >0, then
                              continue count
141B 48        PHA           ;Accu on stack
141C 8D 00 FF  STA $FF00     ;switches I/O
141F 8D 12 D4  STA $D412     ;Waveform = 0
1422 8A        TXA           ;X-reg into accumulator
1423 48        PHA           ;Accu to stack
1424 AE F3 14  LDX $14F3     ;Number for new note
1427 BD 00 16  LDA $1500,X   ;Get note low byte
142A 8D 0E D4  STA $D40E     ;Set low byte
142D BD 00 16  LDA $1600,X   ;Get high byte
1430 8D 0F D4  STA $D40F     ;Set high byte
1433 BD 00 17  LDA $1700,X   ;Get note value
```

```
1436 8D F1 14 STA $14F1     ;Set counter
1439 A9 41    LDA #$41      ;Load waveform
143B 8D 12 D4 STA $D412     ;Set waveform
143E E8       INX           ;Raise note counter
143F 38       SEC
1440 E0 63    CPX #$63      ;Last note?
1442 90 02    BCC $1446     ;NO--then $1C46
1444 A2 00    LDX #$00      ;Note counter at 0
1446 8E F3 14 STX $14F3     ;Store note counter
1449 68       PLA           ;Accu from stack
144A AA       TAX           ;X-reg = accu
144B 68       PLA           ;Accu from stack
144C 4C 65 FA JMP $FA65     ;Goto normal IRQ
```

3.5 REAL-TIME CLOCK ON THE C-128

These days, time is important to all of us. Perhaps this is on reason why most computers now have built-in clocks. By making use of the built-in variable TI$, your C-128 can become a clock. Be forewarned that TI$ runs unevenly at times; because of this we have developed a real-time clock using CIA #1:

```
20    REM  0
30    DATA 31,248,,63,252,
40    DATA 127,254,,255,255,,240,15,
50    DATA 224,7,,224,7,,224,7,,224,7,
60    DATA 224,7,,224,7,,224,7,,224,7,
70    DATA 224,7,,224,7,,224,7,,224,7,
80    DATA 240,15,,255,255,,127,254,
90    DATA 63,252,,
100   REM  1
110   DATA 3,252,,7,254,,7,254,
120   DATA 7,254,,7,254,,3,254,
130   DATA ,126,,,126,,,126,,,126,,,126,
140   DATA ,126,,,126,,,126,,,126,,,126,
150   DATA ,126,,,126,,,126,,,126,,,60,,
```



```
160   REM  2
170   DATA 63,254,,127,255,,127,255,
180   DATA 124,31,,120,15,,48,15,,,15,
190   DATA ,31,,,63,,,126,,,252,,1,248,
200   DATA 3,240,,7,224,,15,160,,31,134,
210   DATA 63,7,,126,7,,252,7,,255,255,
220   DATA 127,254,,
230   REM  3
240   DATA 63,254,,127,255,,127,255,
250   DATA 124,31,,120,15,,48,15,,,31,
260   DATA ,63,,,127,,,255,,1,255,
270   DATA 1,255,,,255,
280   DATA ,127,,,63,,96,31,,240,15,
290   DATA 240,15,,255,255,
300   DATA 255,255,,127,254,,
310   REM  4
320   DATA ,124,,,252,,1,252,,1,252,
330   DATA 3,252,,3,252,,7,252,,7,252,
340   DATA 15,188,,15,188,,31,60,,31,60,
350   DATA 62,60,,124,60,,248,60,
360   DATA 255,255,,255,255,,255,255,
370   DATA 255,255,,,60,,,60,,
380   REM  5
390   DATA 127,252,,255,255,,255,255,
400   DATA 240,15,,224,15,,224,6,,240,,
410   DATA 255,,,255,224,,127,248,
420   DATA 1,254,,,127,,,31,,,31,,,31,
430   DATA ,31,,96,127,,241,254,,255,248,
440   DATA 255,224,,127,128,,
450   REM  6
460   DATA 127,252,,255,255,,255,255,
470   DATA 240,15,,224,15,,224,6,,240,,
480   DATA 255,,,255,224,,255,248,248,
490   DATA 254,,240,127,,224,31,,224,31,,224,31,
500   DATA 224,31,,240,127,,248,254,,255,248,
510   DATA 255,224,,127,128,,
520   REM  7
530   DATA 63,254,,127,255,,127,255,
540   DATA 124,31,,120,15,,48,15,,,15,
550   DATA ,31,,,63,,,126,,,252,,1,248,
560   DATA 3,240,,7,224,,15,160,,31,128,
570   DATA 63,,,126,,,252,,,252,,,252,,,
580   REM  8
```

```
590   DATA 127,254,,255,255,,255,255,
600   DATA 240,15,,224,7,,240,15,
610   DATA 255,255,,255,255,,127,254,
620   DATA 127,254,,254,127,,240,15,
630   DATA 224,7,,224,7,,224,7,,224,7,
640   DATA 240,15,,127,254,,31,248,
650   DATA 15,224,,1,128,,
660   REM  9
670   DATA 1,254,,7,255,,31,255,,127,143,
680   DATA 254,15,,240,15,,240,15,
690   DATA 240,15,,240,15,,254,15,
700   DATA 127,143,,31,254,,7,255,,1,255,
710   DATA ,15,,96,7,,240,7,,240,15,
720   DATA 255,255,,255,255,,127,254,,
730   POKE 56334, PEEK (56334) OR 128
740   FOR I = 3456 TO 4095
750   READ A
760   POKE I,A
770   NEXT I
780   REM  SET TIME
790   INPUT "HOURS{SPACE}{SPACE}{SPACE}";S
800   IF S > 12 THEN S = S - 12:  GOTO 800
810   IF S < 0 THEN 790
820   POKE 56331,S +  INT (S / 10) * 6
830   INPUT "MINUTES{SPACE}";M
840   IF M < 0 OR M > 59 THEN 830
850   POKE 56330,M +  INT (M / 10) * 6
860   INPUT "SECONDS{SPACE}";S
870   IF S < 0 OR S > 59 THEN 860
880   POKE 56329,S +  INT (S / 10) * 6
890   S = 0
900   FOR I = 5120 TO 5180
910   READ A
920   S = S + A
930   POKE I,A
940   NEXT I
950   IF S <> 6300 THEN PRINT "?ERROR IN DATA":END
960   PRINT "{CLR HOME}HIT RETURN TO START"
965   GET A$:  IF A$ <  >  CHR$ (13) THEN 965
970   POKE 56328,0
980   SYS 5120
990   DATA 120,169,13,141,20,3,169
1000  DATA 20,141,21,3,88,96,160
```



```
1010  DATA 3,173,11,220,41,31,76
1020  DATA 26,20,185,8,220,41,240
1030  DATA 74,74,74,74
1040  DATA 72,185,8,220,41,15
1050  DATA 72,136,208,237,173,8
1060  DATA 220,72,160,7,104
1070  DATA 24,105,54,153,247,7,136
1080  DATA 208,246,76,101,250
1090  FOR I = 1 TO 7
1100  SPRITE I,1,1,1,0,1,0
1110  MOVSPR I,30 + I * 20 + INT((I-1)/2) * 10,100
1120  NEXT I
```

The individual numbers are actually sprites. In fact, most of the listing is just to define the sprites. If you prefer, you can run the program without sprites; just type in the program beginning at line 780 and ending at line 1080. You will need to alter the checksum (to "6298"), and change line 1070 to:

```
1070 DATA 24,105,48,153,255,3,136
```

Now for the machine language listing:

```
1400 78        SEI          ;Interrupt off
1401 A9 0D     LDA #$00     ;Set IRQ vector
1403 8D 14 03  STA $0314    ;to beginning of
1406 A9 14     LDA #$14     ;this routine
1408 8D 15 03  STA $0315
140B 58        CLI          ;Enable interrupt
140C 60        RTS          ;Return to BASIC
140D A0 03     LDY #$03
140F AD 0C DC  LDA $DC0B    ;Hour setting
1412 29 1F     AND #$1F     ;only first 5 bits
1414 4C 1A 14  JMP $141A
1417 B9 08 DC  LDA $DC08,Y;Time
141A 29 F0     AND #$F0     ;only decimal point
141C 4A        LSR          ;Shift bit 4-7
141D 4A        LSR          ;to bits 0-3
```

```
590   DATA 127,254,,255,255,,255,255,
600   DATA 240,15,,224,7,,240,15,
610   DATA 255,255,,255,255,,127,254,
620   DATA 127,254,,254,127,,240,15,
630   DATA 224,7,,224,7,,224,7,,224,7,
640   DATA 240,15,,127,254,,31,248,
650   DATA 15,224,,1,128,,
660   REM  9
670   DATA 1,254,,7,255,,31,255,,127,143,
680   DATA 254,15,,240,15,,240,15,
690   DATA 240,15,,240,15,,254,15,
700   DATA 127,143,,31,254,,7,255,,1,255,
710   DATA ,15,,96,7,,240,7,,240,15,
720   DATA 255,255,,255,255,,127,254,,
730   POKE 56334, PEEK (56334) OR 128
740   FOR I = 3456 TO 4095
750   READ A
760   POKE I,A
770   NEXT I
780   REM  SET TIME
790   INPUT "HOURS{SPACE}{SPACE}{SPACE}";S
800   IF S > 12 THEN S = S - 12:  GOTO 800
810   IF S < 0 THEN 790
820   POKE 56331,S +  INT (S / 10) * 6
830   INPUT "MINUTES{SPACE}";M
840   IF M < 0 OR M > 59 THEN 830
850   POKE 56330,M +  INT (M / 10) * 6
860   INPUT "SECONDS{SPACE}";S
870   IF S < 0 OR S > 59 THEN 860
880   POKE 56329,S +  INT (S / 10) * 6
890   S = 0
900   FOR I = 5120 TO 5180
910   READ A
920   S = S + A
930   POKE I,A
940   NEXT I
950   IF S <> 6300 THEN PRINT "?ERROR IN DATA":END
960   PRINT "{CLR HOME}HIT RETURN TO START"
965   GET A$:  IF A$ <  >  CHR$ (13) THEN 965
970   POKE 56328,0
980   SYS 5120
990   DATA 120,169,13,141,20,3,169
1000  DATA 20,141,21,3,88,96,160
```



```
1010  DATA 3,173,11,220,41,31,76
1020  DATA 26,20,185,8,220,41,240
1030  DATA 74,74,74,74
1040  DATA 72,185,8,220,41,15
1050  DATA 72,136,208,237,173,8
1060  DATA 220,72,160,7,104
1070  DATA 24,105,54,153,247,7,136
1080  DATA 208,246,76,101,250
1090  FOR I = 1 TO 7
1100  SPRITE I,1,1,1,0,1,0
1110  MOVSPR I,30 + I * 20 + INT((I-1)/2) * 10,100
1120  NEXT I
```

The individual numbers are actually sprites. In fact, most of the listing is just to define the sprites. If you prefer, you can run the program without sprites; just type in the program beginning at line 780 and ending at line 1080. You will need to alter the checksum (to "6298"), and change line 1070 to:

```
1070 DATA 24,105,48,153,255,3,136
```

Now for the machine language listing:

```
1400 78       SEI          ;Interrupt off
1401 A9 0D    LDA #$00     ;Set IRQ vector
1403 8D 14 03 STA $0314    ;to beginning of
1406 A9 14    LDA #$14     ;this routine
1408 8D 15 03 STA $0315
140B 58       CLI          ;Enable interrupt
140C 60       RTS          ;Return to BASIC
140D A0 03    LDY #$03
140F AD 0C DC LDA $DC0B    ;Hour setting
1412 29 1F    AND #$1F     ;only first 5 bits
1414 4C 1A 14 JMP $141A
1417 B9 08 DC LDA $DC08,Y;Time
141A 29 F0    AND #$F0     ;only decimal point
141C 4A       LSR          ;Shift bit 4-7
141D 4A       LSR          ;to bits 0-3
```

```
141E 4A       LSR
141F 4A       LSR
1420 48       PHA          ;Put number on stack
1421 B9 08 DC LDA $DC08,Y;Time
1424 29 0F    AND #$0F     ;Seconds
1426 48       PHA          ;Put time on stack
1427 88       DEY
1428 D0 ED    BNE $1417
142A AD 08 DC LDA $DC08    ;Tenths
142D 48       PHA          ;Put tenths on stack
142E A0 07    LDY #$07
1430 68       PLA          ;Get count from stack
1431 18       CLC
1432 69 36    ADC #$36     ;Add start-of-sprites
1434 99 F7 07 STA $07F7,Y;Sprite on block
1437 88       DEY
1438 D0 F6    BNE 1430
143A 4C 65 FA JMP $FA65    ;Goto normal IRQ
```

The "other" version (without sprites) requires changing only two lines:

```
1432 69 30    ADC #$30     ;Convert count into ASCII
1434 99 FF 03 STA $03FF,Y;and put onscreen
```

As you can see from the listing, you could set the clock using the IRQ. The clock will shut down when <RUN STOP/RESTORE> is pressed.

3.6 ANALOG CLOCK

To emphasize the importance of time, here's another clock program. Since not everyone likes digital clocks, this is an analog clock, from which most of us learned to tell time in the first place. This program combines a time program and the C-128's graphic commands. This program only works on a 40-column screen.

Now on to the program. It's divided into two parts. The first part should look familiar to you, since it was used for setting the clock in the previous chapter. The second section of the program begins at line 150. This is where the clock face and hands are drawn, and where the second hand is controlled. Every minute, the screen is redrawn. The hands are all drawn using an extension of the CIRCLE command.

```
10    REM  SET CLOCK
20    INPUT "HOURS{SPACE 3}";S
30    IF S > 12 THEN S = S - 12:  GOTO 30
40    IF S < 0 THEN 20
50    POKE 56331,S +  INT (S / 10) * 6
60    INPUT "MINUTES{SPACE}";M
70    IF M < 0 OR M > 59 THEN 60
80    POKE 56330,M +  INT (M / 10) * 6
90    INPUT "SECONDS{SPACE}";S
100   IF S < 0 OR S > 59 THEN 90
110   POKE 56329,S +  INT (S / 10) * 6
120   PRINT "HIT RETURN TO START CLOCK"
125   GET A$:  IF A$ <>  CHR$ (13) THEN 125
130   POKE 56334, PEEK (56334) OR 128
140   POKE 56328,0
150   REM  DRAW CLOCK
160   GRAPHIC 1
170   SCNCLR
180   COLOR 1,2
```

```
200   CIRCLE ,160,100,74
210   CHAR ,19,4,"12"
220   CHAR ,20,21,"6"
230   CHAR ,28,12,"3"
240   CHAR ,11,12,"9"
270   REM  DRAW HOUR AND MINUTE HAND
280   COLOR 1,12
290   M =INT(PEEK(56330)/16)*10+(PEEK(56330)AND15)
300   COLOR 1,1
310   CIRCLE ,160,100,0,60,40,90,M * 6
320   H = ((16 AND  PEEK(56331))/16)*10 +
      (PEEK (56331) AND 15)
330   CIRCLE ,160,100,0,60,60,90,H * 30 + M / 2
340   REM   DRAW SECOND HAND
350   WAIT 56328,8
360   COLOR 1,12
370   CIRCLE ,160,100,0,110,0,40,S * 6
380   COLOR 1,2
390   S=INT(PEEK(56329)/16)*10 +(PEEK(56329)AND15)
400   CIRCLE ,160,100,0,110,0,40,S * 6
410   IF S = 0 THEN 170
420   GOTO 350
```

3.7 LLIST

There are times when you'd like a quick printout of a BASIC program. Commodore BASIC doesn't make a hardcopy listing easy. You must enter all of the following commands:

```
OPEN4,4:CMD 4:LIST:CLOSE4
```

Other versions of BASIC have the command LLIST. The C-128 has to enable printer output only, jump to LIST, and turn back to the screen.

The following LLIST routine will not go directly to the LIST command at the beginning, but a little later. It won't check for which lines to list (you'd want the entire program in hardcopy anyway). Vectors $61/$62 come into play here (pointing to the beginning of the first line to be listed). Here's the assembler listing:

```
1A00 A9 OO     LDA  #$00     :Configuration
1A02 8D 00 FF  STA $FF00     :Set configuration
1A05 A9 OO     LDA #$00      :File name
1A07 20 BD FF  JSR $FFBD     :Set file name
1A0A A9 04     LDA #$04      :Logical file number
1A0C AA        TAX           :Device number
1A0D A0 00     LDY #$07      :Secondary address
1A0F 20 BA FF  JSR $FFBA     :Set file parameters
1A12 20 C0 FF  JSR $FFC0     :Kernal routine OPEN
1A15 A2 04     LDX #$04      :Logical file number
1A17 20 C9 FF  JSR $FFC9     :Set output device
1A1A A5 2D     LDA $2D       :BASIC RAM start--low
1A1C 85 61     STA $61       :store
1A1E A5 2E     LDA $2E       :BASIC RAM start--high
1A20 85 62     STA $62       :store
1A22 20 E5 50  JSR $50E5     :LIST routine (list all)
1A25 A9 OO     LDA  #$00     :Configuration
1A27 8D 00 FF  STA $FF00     :Set configuration
1A2A A9 04     LDA #$04      :Logical file number
1A2C 20 C3 FF  JSR $FFE7     :Kernal routine CLALL
1A2F 60        RTS           :Return
```

Run this routine by entering SYS DEC("1A00") and your program will be listed to the printer.

Here is the BASIC loader:

```
100 FOR X = 6656 TO 6703
110 READ A : CS = CS + A: POKE X, A
120 NEXT X
130 DATA 169,0,141,0,255,169,0,32,189,255
```

```
140 DATA 169,4,170,160,7,32,186,255,32,192
150 DATA 255,162,4,32,201,255,165,45,133,97
160 DATA 165,46,133,98,32,228,80,169,0,141
170 DATA 0,255,169,4,32,231,255,96
```

3.8 DO-IT-YOURSELF WORD PROCESSING

Writing about full-fledged word processing doesn't make a lot of sense in a "Tricks and Tips" book. Instead, we offer you a short idea which will do for throwing a quick memo on paper, that is, in effect, a cheap imitation of a word processing program.

First, type in AUTO 10 <RETURN>. Now type in your first line number (10), followed by a line of text and <RETURN> (you'll have to do this at the end of every line, but it's a small price to pay). Watch your line length (60 characters (1 1/2 lines) will work), since the printer will receive this verbatim. Here's a sample of text:

```
10 This is the new word processor which
20 is using built-in commands.  The quick
30 brown fox jumped over the lazy warthog.
40 ...
```

Great, but how do we get it on paper? Simple; type in this line in direct mode when you're through entering text:

```
POKE 24,37:OPEN4,4:CMD 4:LIST:PRINT#4:CLOSE4
```


The text prints out -- without the line numbers!

You can SAVE this material to diskette or cassette as you would a normal program, and print it out later. Sooner or later, though, you may end up with your own professional word processing software.

Address 24 can do an amazing number of things. For example:

```
POKE 24,37:LIST
```

lists a program on the screen without line numbers. If you run into a "FORMULA TOO COMPLEX ERROR", just try

```
POKE 24,27
```

Values other than 27 used in address 24 can make some changes to BASIC listings. Be careful. Error messages will reset the contents of location 24.

3.9 MODIFIED INPUT

BASIC 7.0 has all those input commands you "grew up with" on the C-64; GET and INPUT, and a new one: GETKEY.

INPUT, however, isn't especially aesthetic -- you get a question mark at every prompt onscreen. Sometimes a question mark isn't appropriate, as in

```
10 INPUT"INPUT YOUR NAME!";N$
```

One method of getting around this is to open your screen as a file. Try this out instead of the normal INPUT statement:

```
10 OPEN 1,0 :REM (screen as a file)
20 INPUT#1,A$: REM (input w/o ?)
30 CLOSE 1
```

Now, if the INPUT command has to be in a particular spot on the screen, add the following:

```
20 CHAR,column, row
25 PRINT" ";:
27 INPUT#1,A$
```

CHAR positions the cursor much like a PRINT AT statement.

3.10 WARNING TONE

When the first typewriters appeared, they had a warning bell to let you know that you were getting close to the right margin. This can still be useful on computers. Here's a machine language program to create such a function:

```
142D A9 FF    LDA #$FF      ;Musical values
142F 8D 06 D4 STA $D406
1432 8D 18 D4 STA $D418
1435 A9 09    LDA #$09
1437 8D 05 D4 STA $D405
143A 78       SEI           ;Interrupt off
143B A9 47    LDA #$47      ;Reset IRQ vector
143D 8D 14 03 STA $0314
1440 A0 14    LDA #$14
```



```
1442 8D 15 03 STA $0315
1445 58       CLI           ;Enable IRQ
1446 60       RTS           ;Return to BASIC
1447 48       PHA           ;Accu on stack
1448 A5 E7    LDA $E7       ;Load right margin
144A E5 EC    SBC $EC       ;Draw cursor-column
144C C9 02    CMP #$02      ;=2?
144E D0 11    BNE $1461     ;NO--then $1461
1450 CD 80 14 CMP $1480     ;Column by last
1453 F0 0F    BEQ $1464     ;IRQ also 2, then $1464
1455 8D 80 14 STA $1480     ;Store 2
1458 A9 A0    LDA #$A0      ;Play tone
145A 8D 01 D4 STA $D401
145D A9 21    LDA #$21
145F D0 05    BNE $1466
1461 8D 80 14 STA $1480     ;Store column
1464 A9 00    LDA #$00      ;Turn tone off
1466 8D 01 D4 STA $D401
1469 8D 00 D4 STA $D400
146C 8D 04 D4 STA $D404     ;Set waveform
146F 68       PLA           ;Accu from stack
1470 4C 65 FA JMP $FA65     ;Jump to IRQ
1480 Pointer for previous column.
```

Two characters from the end-of-line, a tone will sound. This is standard; it doesn't change if you switch from TV to a monitor. Basically, the program counts from the right margin of a window -- this is where the current cursor position is drawn.

```
10    REM  WARNING TONE
20    FOR I = 5165 TO 5234
30    READ A
40    S = S + A
50    POKE I,A
60    NEXT I
70    IF S <  > 8167 THEN BEGIN
80    PRINT "?ERROR{SPACE}IN{SPACE}DATA"
90    END
100   BEND
110   SYS 5165
```

```
120    DATA 169,255,141,6,212,141,24,212
130    DATA 169,9,141,5,212,120,169,71
140    DATA 141,20,3,169,20,141,21,3,88,96
150    DATA 72,165,231,229,236,201,2,208
160    DATA 17,205,128,20,240,15,141,128
170    DATA 20,169,160,141,1,212,169,33
180    DATA 208,5,141,128,20,169,0,141
190    DATA 1,212,141,0,212,141,4,212
200    DATA 104,76,101,250
```

You can change the value to anything *except* the last column. Just bear in mind that this routine is counting from right to left:

```
POKE 5197,N
```

N will give you your new "warning" location.

This routine would be great for word processing, or for other data input especially when used in combination with the "Keyboard Beep" program in Chapter 7.

CHAPTER 4

SOFTWARE PROTECTION ON THE C-128

Software publishers invest a lot of time and money in copy protection. It's really pretty sad that software piracy is running rampant. Pirates are taking money out of the author's hand. The author is writing software to make a profit, but if the pirate is stealing software, then this is a disincentive to the author to produce more quality software.

You can install copy protection on your own disks. The protection isn't unbreakable, but it will make matters difficult for the curious. One form of protection is based on a staple of BASIC programming: the LIST command. We'll talk about other methods of getting in the "back door" of protection.

4.1 PROTECTION WITH COLONS

We did a lot of research and experimentation into protection routines. We found a few small BASIC routines for you to try. For example, you can hide individual lines from nosy folks with five colons! Type this:

```
450 :::::PRINT "(C) 1985 by 'The Team'"
```

Next, add this routine to your programs with five colons inserted in the lines you wish to hide:

```
60000   LI = 45
60005   BA = LI
60010   LI = PEEK (LI) + 256 * PEEK (LI+1)
60020   IF LI = 0 THEN PRINT "DONE!" : END
60030   PRINT CHR$(145); "PASS ";
        PEEK(BA+2)+256*PEEK(BA+3);
60040   PRINT X$
60050   BA = LI
60060   FOR X = 4 TO 8
60070   IF PEEK (LI+X) <> ASC(:) THEN X=8: FL=1
60080   NEXT X
60090   IF FL = 1 THEN FL = 0 : GOTO 60010
60100   POKE LI + 4, 0 : X$ = X$ + "*"
60110   GOTO 60010
```

The routine is activated with RUN 60000. It tests for program lines to be ignored (PASS). When it finds one, the line will be covered with asterisks. The system will say READY when it's done; all that remains is for you to DELETE 60000- . Run the program; works fine. Now LIST it; the line numbers of the protected lines appear, but the contents of the lines are invisible.

The Program:

60000 : Start-of-BASIC is stored in variable LI. If you want to use this program in C-64 mode, you'll have to change the parameters in locations 45 and 43.

60010 : Rather than look at *every* byte in the program, the protection routine checks line links for colons; the routine runs faster that way.

60020 : Program ends when line link LI=0 .

60030 : Current line number given.

60040 : Number of protected lines is stored in X$.

60050 : Basis of new line starting range is equal to the contents of link LI.

60060 : Routine looks at first five characters of a line.

60080 : Each "normal" line sets flag FL, and X to 8.

60090 : If FL=1 the flag is reset, and the next line observed.

60100 : Line is protected; first colon is set to 0. When the program sees the 0, it's interpreted as an end-of-line, and the line is treated as if it were non-existent.

4.2 LINE NUMBER ROULETTE

A prime characteristic of BASIC is its sequential line numbers, which act as a guide for the programmer. Line numbers larger than 64000 cannot be used. We can use this little fact to our advantage: Type in this routine.

```
60000   BA = PEEK (45) + 256 * PEEK (46)
60010   LI = PEEK (BA) + 256 * PEEK (BA + 1)
60020   IF LI = 0 THEN PRINT "DONE!" : END
60030   PRINT"FIND LINE:"; PEEK(BA+2)+256*PEEK(BA+3)
60040   PRINT "(C)HANGE  (G)O ON  (E)ND  ?"
60050   GETKEY A$
```

```
60060   IF A$ = "E" THEN PRINT "OK" : END
60070   IF A$ = "G" THEN 60140
60080   INPUT "NEW LINE NUMBER"; ZN$
60090   IF ZN < 0 OR ZN > 65535 THEN 60080
60100   HI = INT(ZN/256) : LO= ZN-(256*HI)
60110   POKE BA+2, LO
60120   POKE BA+3, HI
60130   PRINT CHR$(145);
60140   BA = LI
60150   PRINT CHR$(145);CHR$(145);
60160   GOTO 60010
```

The first line probably looks familiar to you. Start the routine; the system will go through each line with prompts. You have three options: (C)HANGE, (G)O ON, or (E)ND. C alters the program line; G looks for the next line; and E ends the program. There are a few limitations:

a) GOTO and GOSUB line numbers are unaltered.

b) Line numbers larger than 64000 cannot be deleted.

c) All line numbers less than one cannot be jumped to.

The Program:

The principle of this routine is pretty simple: It looks at all the BASIC program lines. If the line number is changed, the new line number is stored in ZN and in line 60100 in low/high-byte format. Then, the new value is POKEd over the old in BASIC memory.

This protection will be automatically saved on disk or cassette with the program itself.

One other thing: type in a line, say, 64000. Now run the routine, and change it to 65535. This produces two effects. First, a line with the number 65535 is essentially considered to be the end of the program. Second, that high number is not shown if it's a REM, so you could hide your copyright notice there.

4.3 MANIPULATING LINE-LINKS

The preceding routine worked on the so-called line-links to change line numbers. Let's take another shot at "bending" these line-links, specifically, adjusting the link pointing to the next line. When we do this, the next line is invisible. Or, if you wish, we could fix a line-link to list the same line over and over. Usually, this won't bother the program run, unless you've really gone wild with these adjustments. Type this in by hand:

```
60000 INPUT "1. LINE NUMBER"; Z1
60020 INPUT "2. LINE NUMBER"; Z2
60020 BA = 45
60030 BA = PEEK (BA) + 256*PEEK(BA+1)
60040 IF BA=0 THEN PRINT "LINE DOES NOT EXIST!":END
60050 ZN = PEEK(BA+2) +256*PEEK(BA+3)
60060 IF FL = 0 THEN IF ZN =Z1 THEN
      L1=BA:L2=BA+1:FL=1
60070 IF FL = 0 THEN 60030
60080 IF ZN=22 THEN POKE L1,PEEK(BA) :
      POKE L2,PEEK(BA+1):END
60090 GOTO 60030
```

The routine will ask you for the numbers of the first and second program lines. The links in the first line pointing to both lines will be altered. All lines become invisible. One thing to remember with this routine: any

alterations (adding or deleting lines) recalculate the line-link pointers, so it's a far from infallible protection trick.

4.4 CREATIVE CONTROL CHARACTERS: MAKING GREMLINS

Control characters are a familiar sight; for example, this one clears the screen: "♥". We see them in quote mode, they change character position and color. There is another use for these characters, though: we can use them on the screen to foil LISTing. Type this in first in direct mode:

```
KEY1,CHR$(27)+CHR$(79)+"{RVSON}{SHIFT-M}{RVSOFF}"+
CHR$(34)+CHR$(20)
```

This turns <F1> into an instant <SHIFT-M> (this activates control characters). Let's have some fun with a few BASIC lines (whenever an asterisk (*) appears, press <F1>).

The Gremlin and PRINT

```
PRINT"This is how it works*in PRINT statements*!"
```

All <SHIFT-M> does here is perform a carriage return (C/R). This has nothing to do with software protection; just showing you what it does.

The Gremlin and REM

```
10 REM"*(CRSRUP)(RVSON)This is a test line
```

List this line, things are getting interesting now! The <SHIFT-M> causes the two control characters to execute, so the line moves up one line and goes into reverse video. In other words, the REM acts like a PRINT statement when listed!

The possibilities are almost limitless! We can change colors...

```
1000 REM"*(CTRL+1to8)
1000 REM"*(C= + 1to8)
```

...format listings...

```
1000 REM"*(CRSRDOWN)(CRSRDOWN)
```

...or simply mask lines out:

```
1000 REM"*{CRSRUP}900 SYS (4096) CRACKED BY RUSS T
```

(Indicates cracked machine language program)

```
1000 PRINT"(C) BY RUSS T":REM"*{CRSRUP}
```

(Line 1000 overwritten by next line)

Have a good time!

4.5 PROTECTION WITH POKES

POKEs can do so much; they even give you the ability to ward off saving, listing or running a program. One disadvantage: The program has to be RUN before any of this will take effect, but you can sidestep this somewhat by using an autoboot program.

4.5.1 LIST DISABLING

Page 3 has a pointer that points to the vector for the LIST input. This vector has the addresses 774/775 ($0306/$0307). Try POKE 774,61:POKE 775,255. Now LIST. The computer acts as if you've requested a warm-start, and goes to the opening screen. Now try POKE 774,38:POKE 775,160, which points to the vector for "RTS". Only line numbers will be given.

As you can see, changing pointers can do remarkable things.

4.5.2 DISABLING RUN-STOP/RESTORE

Early in computing, people got the idea of putting code words into programs. Commodore computers can be circumvented, just by pressing <RUN-STOP/RESTORE>, and looking up the code word in the program. We can foil that with POKE 808,PEEK(808)-3 which disables both keys.

Let's put it into practice:

```
10 POKE 808,PEEK(808)-3
20 PRINT"This is a test ";
30 GET A$:IF A$=CHR$(3) THEN PRINT:PRINT"BREAK"
```

POKE 792,51:POKE 793,255 disables RESTORE alone; <RUN-STOP> alone can be done in BASIC:

```
10 TRAP 1000
20 PRINT"This is a test ";
30 GOTO 20
1000 ?"*BREAK*"
1010 RESUME NEXT
```

Here we used the message "BREAK" to signify the <RUN-STOP> key. See the chapter on "Error Handling" (Chap. 3). One final word: This routine can be skirted by pressing the <RUN-STOP> key quickly several times.

4.5.3 DISABLING SAVE

The SAVE vector also resides in page 3 of memory (818/819 [$0332/$0333]). We can pretty much pull the same stuff that we did in the LIST disable: RTS, reset, etc.

4.6 DISK COPY PROTECTION

So far, we've looked at general in-program protection. Next, we're going to look at what's involved in protecting a disk from copying.

Perhaps you've heard of "destroying" the tracks on a disk. We can create a READ ERROR, which will at least slow down disk copying. Name the track you wish to change in the variable TR.

```
 0 REM READ ERROR ON DISK
10 OPEN 1,8,15
20 OPEN 2,8,2,"#"
30 PRINT#1,"U1 2 0";TR;0
40 PRINT#1,"M-E"CHR$(163)CHR$(253)
```

Before starting, make sure there's no valuable disk in the drive.

4.7 LOAD "$"

There are times when you may want to try stopping anyone from reading the disk directory, since the directory gives important information to the would-be copier. Best way around this is to wipe the directory altogether!

```
10 REM DIRECTORY SCRATCH
20 OPEN 1,8,3,"#"
30 OPEN 2,8,15,"B-P3,144"
40 PRINT#1,CHR$(20)CHR$(20)CHR$(20)CHR$(0)CHR$(0)
   CHR$(0)
50 PRINT#2,"U2:3,0,18"
60 PRINT#2,"I"
70 END
```

Use this program with disks from which you've memorized the program names. Also, keep in mind that you won't know what programs have been changed, scratched, or whatever.

Essentially, when the directory is loaded with LOAD "$",8 it is handled like a BASIC program. We simply write a zero to the beginning of the directory, and the LIST routine interprets the zero as the end-of-program. This only works for LOAD"$",8 and not the DIRECTORY command.

CHAPTER 5

SELF-MODIFYING PROGRAMS

5.1 LINE INSERTION

Programs are usually designed with just one application in mind, partially due to the fact that you can't change programs as they're running. In the next few pages, we're going to show you how to insert lines into a running program. We'll also give you some sample applications for this useful feature.

Inserting lines in an running program opens up a new spectrum of programming, and can be accomplished with a few simple POKEs. Remember, though, this little routine is a big step from a program generator, which lets you create a BASIC (or other) program.

Technically, this line addition is very complicated. So, the normal procedure is to stop the program, insert the line and restart the program. The key to our work in this chapter is the keyboard buffer. Try this test program:

```
10 TP=842
20 POKE   TP,ASC("L")
30 POKE TP+1,ASC("I")
40 POKE TP+2,ASC("S")
50 POKE TP+3,ASC("T")
60 POKE TP+4,13: REM RETURN
70:
80 POKE 208,5
90 END
```

RUNning this program makes it LIST itself by writing LIST into the keyboard buffer (lines 20-50) and <RETURN> (line 60). Line 80 tells the operating system that five characters are in the keyboard buffer. One application would be for INPUT or GET, or for a program end. This is what happens: the program ends at line 90, but the computer "INPUTS" the characters that were "stuck" in the keyboard buffer. So, the system interprets LIST<RETURN> as a direct command. Remember that the computer gets the characters when the program ends. Experiment -- put in your own commands (e.g., RUN; but remember to put in <RETURN>s (13)).

5.2 FORMULA ENTRY

Now, let's experiment with changing lines in a running BASIC program:

```
10 PRINT"FORMULA ENTRY"
20 PRINT:PRINT
30 PRINT"ENTER AN EQUATION"
40 PRINT"(E.G. Y=2*X+SQR(X/34.5)*INT(5.6*X^4)
50 INPUT V$
```

This program section needs no detailed explanation. Basically, the equation of your choice is typed in, and stored in V$. Our example has the format Y=....X... , which leaves X open for any variable. But how do we get the equation into the program?

```
60 PRINT CHR$(147)
70 PRINT : PRINT"1000";V$
80 PRINT : PRINT"RUN 500"
```


This creates a new line number -- line 1000 with your formula.

```
90 PRINT CHR$(19);
```

When you RUN this program, the cursor blinks on the new line. If the line is correct, press the <RETURN> key. Here's more:

```
100 FOR A=0 TO 3
110 POKE 842+A,13
120 NEXT A
130 POKE 208,4
140 END
500 PRINT CHR$(147)
510 INPUT "X-VALUE";X
1000 REM HERE'S WHERE THE EQUATION GOES
1010 PRINT"RESULT IS:";Y
1020 END
```

The keyboard buffer is filled with the ASCII code for <RETURN> . The program ends, the <RETURN>s are sent, and the routine automatically starts with RUN 500. Be sure you make the proper changes when a larger number of variables are used; this is a matter of altering line 500.

5.3 DATA STATEMENT GENERATOR

We're not through with this subject and the next program will show that. This is a "real" application program: a DATA generator, which you'll find to be useful. Although this program lends itself to alterations, it generates a lot of lines!

```
10    PRINT "{CLR HOME}D A T A - G E N E R A T O R"
20    PRINT :  PRINT :  PRINT
30    PRINT "THIS PROGRAM CONVERTS THE ";
40    IF  PEEK (215) < 128 THEN  PRINT
50    PRINT "CONTENTS OF MEMORY TO A BASIC-LOADER!"
60    PRINT :  PRINT
70    PRINT "I N P U T - F O R M A T :  "
80    PRINT :  PRINT
90    INPUT "MEMORY CONFIGURATION (BANK)";BK
100    INPUT "BEGINNING DATA ADDRESS (HEX)";A$
110    INPUT "ENDING DATA ADDRESS (HEX)";B$
120    A =  DEC (A$) - 1: B =  DEC (B$)
130    :
140    INPUT "BEGINNING LINE NUMBER";Z: O = Z
150    INPUT "LINE NUMBER INCREMENT";S
160    INPUT "AMOUNT OF DATA PER LINE";DZ: XY = DZ
170    :
180    GOSUB 1000
190    :
200    LD = B - A
210    PRINT "{CLR HOME}{CRSR DOWN}{CRSR DOWN}"
220    IF LD < DZ THEN DZ = LD: FL = 1
230    LD = LD - DZ
240    PRINT Z;"DATA ";
250    IF DZ > 1 THEN  FOR W = 1 TO DZ - 1:
       ELSE  GOTO 290
260    BANK BK: P =  PEEK (A + W + (XY * ZE)):
       PRINT P;"{CRSR LEFT},";
270    CS = CS + P
280    NEXT W
290    ZE = ZE + 1
300    BANK BK: P =  PEEK (A + (XY * (ZE-1)) + DZ):
       PRINT P: CS = CS + P
310    :
320    Z = Z + S
330    IF FL = 0 THEN  PRINT "GOTO 210":
       ELSE  PRINT "GOTO 900"
340    GOTO 800
800    REM  FILL KEYBOARD BUFFER
810    PRINT "{HOME}";
820    FOR X = 842 TO 851
830    POKE X,13
840    NEXT X
```



```
850    POKE  DEC ("D0"),10
860    END
900    REM  EDITOR
910    PRINT "{CLR HOME}{CRSR DOWN}{CRSR DOWN}";
920    PRINT O+3*S;"IF CS<>";CS;
       " THEN PRINT CHR$(7);: LIST"
930    PRINT "DELETE -1300"
940    BANK 15:  GOTO 800
1000   REM  INIT
1010   PRINT "{CLR HOME}{CRSR DOWN}{CRSR DOWN}"
1020   PRINT Z;"FOR X=";A + 1;"TO";B
1030   PRINT Z + S;"READ A: CS=CS+A: POKE X,A"
1040   PRINT Z + 2 * S;"NEXT X"
1050   Z = Z + 4 * S
1060   PRINT "GOTO 200"
1070   GOTO 800
```

Save this program as *DATAGEN* before running!!! Here's what it does:

The program generates a BASIC loader for you. You'll need to know the starting and ending addresses of the machine language program and the memory configuration (normally Bank 0). In addition, you choose the line number to start, increment, etc. The first line number should be larger than 1300; otherwise the DATA generator program will be overwritten. Once this is complete, the DELETE command erases the DATA generator program, after which you may want to renumber the program.

C-128	C-64	MEANING
208	198	Number of characters to be given by the keyboard.
842-851	631-640	Keyboard buffer (normal); 10 bytes to be given are stored in keyboard buffer .
2592	649	Maximum size of keyboard buffer (10).

Important: "Real-time" line insertion is also possible in C-64 mode, but it's a lot easier to manage in C-128 mode. When you change lines in 64 mode, all the variables are cleared. This is not the case in C-128 mode, because a special memory bank is set aside for variables (Bank 1).

The maximum size of the keyboard buffer is normally 10, which is usually sufficient. If need be, command words can be inserted in "short form" (see the back of the C-128 User's manual for abbreviations). Otherwise, the C-128 lets you change the keyboard buffer to a maximum of 20 elements. This overwrites the TAB-bitmap memory, which means that the TAB key won't work properly. The enlarged keyboard buffer covers memory locations 842 to 861. Change the size of the keyboard buffer with this command:

```
POKE 2592,20
```


THE DATASETTE

In this chapter, we're going to show you a few neat tricks for your Datasette (or whatever compatible tape drive you may have). We imagine that most of the C-128 owners have opted for the 1571 disk drive. By the same token, if you can be patient with the tape drive's speed, it's a good, cheap alternative to a disk drive (beats having nothing for storing programs).

6.1 SOFTWARE CONTROL FOR DATASETTE

Those who use the Datasette usually don't notice that the cassette motor stops on its own when the drive is through saving or loading programs. This motor is controlled by the operating system, but can also be controlled by software (i.e., through programming). The important addresses are locations 1 (processor port) and 192 (cassette motor flag).

Try this: press the PLAY button on your recorder; the motor runs and the tape spins. Now type this in direct mode:

```
POKE 192,1:POKE1,PEEK(1) OR 32
```

The motor stops without your touching the recorder. Let's get the motor running again:

```
POKE 1, PEEK(1) AND 39:POKE 192,0
```

The key to this is the processor port, address 1. Bit 3 (CASS WRT), bit 4 (CASS SENS) and bit 5 (CASS MOTOR) of this address deal with cassette operation. To switch the motor on, the bit CASS MOTOR must be turned on.

Now you can control the cassette drive using a timing loop to create a catalog of your programs, or even set up function keys to switch the motor on and off.

6.2 SENSING THE DATASETTE KEYS

Address 1 does more! Bit 4 checks to see if a key is pressed on the Datasette, which is part of the operating system ("PRESS PLAY(&RECORD) ON TAPE" messages are controlled here). Unfortunately, the computer is unable to discern *which* Datasette key is pressed. This can be a nuisance if the user presses PLAY and forgets to press RECORD as well; the system won't catch the error.

This bit checks for the tape switches:

```
WAIT 1,16       (Wait until STOP is pressed on recorder)
WAIT 1,32,32    (Wait until a key has been pressed on the recorder)
```

We can also make the system test for RUN-STOP:

```
IF PEEK(1)=99 THEN PRINT "OOPS"
```


6.3 DOING UNUSUAL THINGS WITH THE DDR

Okay, what is a "DDR"? And what sort of "unusual things" can you do with it?

A lot, we think. The processor port is basically an I/O (Input/Output) port, handled by the CIAs. This port has six lines, three of which are dedicated to the Datasette. The lines act as both output and input; these directions are determined by the DDR (Data Direction Register) for the processor port, found at address 0. The layout:

bit 3 : Data direction CASS WRT
bit 4 : Data direction CASS SENS
bit 5 : Data direction CASS MOTOR

0 = Input
1 = Output

CASS WRT and CASS MOTOR are output. They affect the recorder when writing to tape. CASS SENS is an input line. It checks if a tape drive key down or not.

6.4 COPY PROTECTION WITH THE DATASETTE

What happens if we change the bits in the DDR? The POKE below turns CASS SENS into an output line:

```
POKE 0,PEEK(0) OR 2^4
```

The result: CASS SENS will not register cassette status. The system will say, "PRESS PLAY(& RECORD) ON TAPE", and even if you do so, the operating system won't acknowledge "OK", or LOAD, or SAVE. Changing things back to normal is a simple matter of typing:

```
POKE 0,PEEK(0) AND NOT 2^4
```

By changing CASS WRT, we can produce a method of copy protection:

```
POKE 0, PEEK(0) AND NOT 2^3
```

The prompt will come up, the keys will be sensed, and the SAVE routine will *appear* to work...but it won't. To return to normal:

```
POKE 0,PEEK(0) OR 2^3
```

One last possibility is to alter the motor control. Re-enter:

```
POKE 0,PEEK(0) AND NOT 2^4
```

The solution to this problem: `POKE 0,PEEK(0) OR 2^4`

6.5 "LO-FI" -- THE DATASETTE AS MUSIC BOX

Stereo systems and video disc players have been developed with computer interfaces built in. This means that in a few years, your stereo and video equipment will be computer controlled. Right now, some computers allow you to play audio cassettes through the computer's data system. We give you a program that does just that on the next page. A word of warning: the sound quality -- well, the word "rotten" would describe it best.

The program below manipulates the three I/O lines of the processor port. There is another, unused line between the recorder and the computer (CASS READ, the equivalent of CASS WRT). All "incoming" data goes through this line. The problem: The line doesn't end at the processor, like the other lines -- it continues on to CIA1 (putting the serial port at our disposal). So register 13 of CIA1 will have to be set:

REG 13 BIT 4:1= Signal tripped on pin FLAG

Here's the machine code listing:

```
1A00 A0 00    LDY #$00      :VOLUME 0
1A02 AD 0D DC LDA $DC0D     :Check FLAG pin on CIA1
1A05 C9 00    CMP #$00      :No signal?
1A07 F0 02    BEQ $1A0B     :YES--go on
1A09 A0 0F    LDY #$0F      :NO--turn off volume
1A0B 8C 18 D4 STY $D418     :and SID register 24
1A0E A5 D5    LDA $D5       :Read keyboard
1A10 C9 58    CMP #$58      :88=no key pressed
1A12 F0 EC    BEQ $1A00     :No keys -- go on
1A14 60       RTS           :Return to BASIC
```

And the BASIC loader:

```
5000  FOR X = 6656 TO 6676
5010  READ A: CS = CS + A:  POKE X,A
5020  NEXT X
5030  IF CS <> 2799 THEN  PRINT  CHR$ (7);: LIST
5040  DATA 160,0,173,13,220,201,0,240,2,160,15,140
5050  DATA 24,212,165,213,201,88,240,236,96
```

Start this program with the command: SYS DEC ("1A00"). The principle of this program is simple: It looks for a signal at FLAG. If that's the case, the speaker is turned on; if not, the speaker remains off. The switching on and off occurs within the smallest loop possible in machine language and a horrible tone comes out.

If you put a music cassette into the system, it will probably sound pretty mangled; the tone is filtered as much as possible.

6.6 SAVING TO CASSETTE -- SORT OF

The SAVE command needs no introduction; you know what it is, and how it works. But what are those funny sounds made by a cassette (did you listen, using the program in the last chapter?)? The peeps are made by the CASS WRT line:

bit 3 CASS WRT
1=impulse

It's fairly simple to make your own tape impulse:


```
10  REM TONES ON CASSETTE
20  PRINT "PRESS PLAY AND RECORD ON TAPE"
30  WAIT 1,32,32
40  FOR W= 1 TO 20
50  FOR K= 0 TO 20
60  POKE 1,PEEK(1) OR 8:REM IMPULSE
70  FOR T = 1 TO W:REM DELAY
80  NEXT T
90  POKE 1,PEEK(1) AND NOT 8:REM SET BACK
100 NEXT K
110 NEXT W
120 END
```

This short program saved different high and low tones to your cassette. Use the previous program to listen to the various sounds created. It should demonstrate the principle.

CHAPTER 7

THE KEYBOARD

7.1. KEYBOARD ASSIGNMENT

There are many ways to read the keyboard besides the BASIC commands GET and INPUT, especially for machine language programmers. In zero page location 213 ($D5) the computer stores the value of the currently pressed key. This value is not the ASCII value, it is derived from the keyboard table in the ROM of the computer.

The C-64 stores the currently pressed key in zero-page location 203 ($CB). Due to the additional keys on the C-128 the values are not always the same. Below is a chart showing what value is returned for each key pressed.

Key	Value	Key	Value
Left arrow	57	1	56
2	59	3	8
4	11	5	16
6	19	7	24
8	27	9	32
0	35	+	40
-	43	O	48
CLR HOME	51	INST DEL	0
CTRL	58	Q	62
W	9	E	14
R	17	T	22
Y	25	U	30
I	33	o	38
P	41	@	46
*	49	^(up arrow)	54
RUN-STOP	63	A	10
S	13	D	18
F	21	G	26

H	29	J	34
K	37	L	42
:	45	;	50
=	53	RETURN	1
Z	12	X	23
C	20	V	31
B	28	N	39
M	36	'	47
.	44	/	55
CRSR down	7	CRSR right	2
SPACE	60		

There are differences between the C-64 and C-128. No key pressed returns the value 88 on the C-128. On the C-64 no key returns the value 64.

The following keys are brand new on the C-128:

Help-Button	64
Tab-Button	67
ESC-Button	72
Line Feed	75

In addition to the cursor keys below the <RETURN> key (which return the same values in both machines) there are four new cursor keys. These have different values!

Cursor up	83
Cursor down	84
Cursor left	85
Cursor right	86

The C-128 also includes a numeric keypad. Just like the separate cursor keys, the values returned by the keypad keys are not equal to the values returned for the numbers on the alphanumeric keyboard. Here are the values for both keyboards on the C-128:

Key	Keyboard	Keypad
1	56	71
2	59	68
3	8	79
4	11	69
5	16	66
6	19	77
7	24	70
8	27	65
9	32	78
0	35	81
.	44	82
RETURN	1	76
+	40	73
-	43	74

The joystick of port 1 can also affect location 213 ($CD). Unfortunately this address is not useful for reading the joystick. When you push the joystick upward, the address will not change. You could use this fact to connect another keyboard to port 1.

Joystick up	88	Effect like ALT
Joystick in the middle	88	No effect
key pushed	92	Effect like shift

Joystick left	90	Effect like CTRL
Joystick right	91	
Joystick down	89	

With the following program you can simulate the BASIC 7.0 command GETKEY:

```
10 IF PEEK(213) = 88 THEN 10
20 GET A$
30 PRINT A$
40 GOTO 10
```

Of course, this command sequence in BASIC programs doesn't make much sense because the GETKEY command already exists. This program can be converted for machine language programs. In assembly language the program would be very short and could look as following:

```
Loop  LDA $D5
      CMP $58
      BEQ Loop
```

After leaving the loop the accumulator contains the value of the key pressed for further processing.

7.2. CHANGING KEY ASSIGNMENTS

Every computer programer and user has their own ideas on how to use a computer. Many people became interested in computers through computer games. For this group the computer indusrty designed their products to be user-friendly, with easily connected joysticks, a lightpen, a trackball, or paddles.

Another group of computer users are people who got involved with computers because of their jobs. The C-128 will probably be more popular with this group than was the C-64, because the C-128 can also run CP/M. CP/M is an operating system that runs on a wide variety of computers. The ability of CP/M to run on many different computers has made it a very popular operating system. To expand the performance of the C-128 it would be very useful to change the key assignments, depending on the requirements of the user.

This is not as easy in the C-128 as it is in the C-64. The keyboard decoder table is located in the ROM:

64128 ($FA80) for ASCII operation
64809 ($FD29) for DIN operation (international models only)

On the international models of the C-128, two character decoding tables are included in ROM. This is to give the international models the correct foreign character sets for the countries in which they are sold. We have checked the U.S. models and the German models (DIN stands for Deutsch (German) Industry Norm). On the international models the <CAP-LOCK>

key is replaced by an <ASCII-DIN> key. When this key is pressed the operating system loads the foreign character (DIN) set into memory. Minor changes were made to the operating system to accommodate this; for more details please see our book *Commodore 128 Internals*.

You can read the key assignments (with the monitor use the command MFFA80), but you can't change the assignments because they are in ROM. There is another way. In zero page there is an pointer to the keyboard decoder table. It is at locations 830 ($033E) and 831 ($033F). To change these values on the international models a little trick is needed. On both the U.S. and the international models you will be able to change the low-byte, but you will not be able to enter commands anymore because the keyboard will not be decoded properly.

On the international models, as soon as you change the high-byte of the pointer (at 831) you will notice that the computer has reset these locations back to their normal values.

This doesn't happen because the address is in the ROM. It is caused by the permanent check of the <ASCII-DIN> key. On the international models when the high-byte does not equal the value which is preset for the current mode, the address is reset. You can prevent the permanent check of the <ASCII-DIN> key, by setting the seventh bit of address 2757 ($OAC5).

```
POKE 2757, PEEK(2757) or 128
```

Now you can change the pointer to the decoder table on the international models. It will not reset automatically again. This is not necessary on the U.S. models.

When you alter this pointer you will no longer be able to enter any commands from the keyboard. When you push a key a character will appear but it is seldom the one you pressed. You must press the reset button to return the computer back to normal operation; not even a <RUN-STOP/RESTORE> helps. For this reason, you should copy one of the character set tables to RAM before switching. Example at 6912 ($1BOO):

```
10 REM Copy and switch
20 FOR I=O TO 88
30 POKE 6912+I, PEEK(64128+I)
40 NEXT I
50 REM INTERNATIONAL MODELS ONLY:
      POKE 2757,PEEK(2757) OR 128
60 POKE 830,0
70 POKE 831,27
```

On the international models if you had the <ASCII-DIN> key on, you will not recognize a big difference. If you started the program when the ASCII character set was activated, you will have the <CAPS-LOCK> character set, which is normal in the American version. When the <CAPS-LOCK> key is on, all letters appear as capital letters, just like the <SHIFT> or <SHIFT-LOC>K key. The difference is, the numbers appear just like before and not the upper character, obtained with <SHIFT>. If <CAPS-LOCK> is not pushed, you have the normal character set.

Now back to changing the keyboard assignments. In the previous section is a chart showing keyboard assignment values, which you can use now. Take the value of the key you would like to change and add it to 6912. In this location you store the ASCII value of the new assignment.

Example: You would like to change Y and Z. Just add these lines to the previous program:

```
80 POKE 6912+12,89
90 POKE 6912+25,90
```

After the program finishes running the Y is the Z key and Z the Y. It is really that easy.

Another tip that will help you to enter machine language listings, in the form of DATA statements, faster into your computer. In the keypad is a period key. This character is useless for entering DATA statements for machine language programs. Instead of this period a comma would make entering DATA statements much easier. So we'll change the keypad, to a comma instead of the period:

```
10 REM COMMA INSTEAD POINT
20 FOR I=0 TO 88
30 POKE 6912+I,PEEK(64128+I)
40 NEXT I
50 POKE 2757,PEEK(2757) OR 128
60 POKE 830,0
70 POKE 831,27
80 POKE 6994,44
```


7.3. HEX-KEYBOARD FOR THE C-128

If you work with hexadecimal numbers, then you know the problems of entering these numbers from the normal keyboard.

Above the keypad are the eight function keys. You only have to assign those with the six necessary letters. After this change you still have two keys left which are not assigned to any function. But we'll also change those. We assign those two keys to the function DEC(" and HEX$(, which are used often. Often, during programming you have only one hand free. Your second hand is busy holding books, lists and other things. For this reason, some of the letters that are reachable only with the <SHIFT> key are awkward to use. But there are still plus and minus keys which are not really necessary for entering hex-numbers. Thats why we'll assign those keys to the letters E and F.

And here is an example how you can enter hex-decimal numbers with one hand:

```
10 REM Hex-keyboard on the C-128
20 KEY 1,"A"
30 KEY 3,"B"
40 KEY 5,"C"
50 KEY 7,"D"
60 KEY 2,"E"
70 KEY 4,"F"
80 KEY 6,"PRINT DEC("+CHR$(34)
90 KEY 8,"PRINT HEX$("
100 For I=0 TO 88
110 POKE 6192+I.PEEK(64128+I)
120 NEXT I
130 POKE 2757,PEEK(2757) or 128
```

```
140 POKE 6994, 32
150 POKE 6985, 69
160 POKE 6986, 70
```

You reach the letters "E" and "F" when you push the keys <SHIFT> and <F1> or <SHIFT> and <F3>, or with the plus key (E) and the minus key (F).

When you would like to convert a hexadecimal number to a decimal number you only have to push the keys <SHIFT> and <F5>. You can convert a decimal number to a hexadecimal number with SHIFT and <F7>.

7.4.SHIFT, C=, CTRL, ALT KEY ASSIGNMENTS

When you used the program where we changed the Y and Z, you probably saw, that the shifted values of these keys remain unchanged. For assignments using the <SHIFT>, the <C=> key, the <ALT> key and the <CTRL> key, there are separate tables. For the ASCII character set (U.S. models) the tables are located as follows:

64128 ($FA80)	First assignment
64217 ($FAD9)	With SHIFT
64306 ($FB32)	With COMMODORE-Key
64395 ($FB8B)	With CTRL
64128 ($FA80)	With ALT (same meaning without shift)

On the international models the C-128 has a second foreign language character set. When the <ASCII-DIN> key is pressed, you can change the

keyboard assignment. Therefore you have four completely different tables containing the keyboard decoder tables and the shift assignments. Here are the current addresses:

64809 ($FD29)	Initial value
64898 ($FD81)	With SHIFT-KEY
64987 ($FDDB)	With COMMODORE-KEY
65076 ($FE34)	With CTRL
64128 ($FA80)	With ALT

Pointers to the respective table for each of the five keys are located in zero page. On the international models there is also a pointer for the <ASCII/DIN> key:

830-831 ($033E-$033F)	Standard key press
832-833 ($0340-$0341)	With <SHIFT>
834-835 ($0342-$0343)	With Commodore (<C=>)
836-837 ($0344-$0345)	With <CTRL>
838-839 ($0346-$0347)	With <ALT>
840-841 ($0348-$0349)	With <ASCII-DIN> or <CAPS-LOCK> key

Each table has 89 bytes, equals to the 88 keys, plus the possibility that no key is pressed.

To replace two keys, you have to change the values in all four tables. The following program will do this:

```
10 REM Z and Y change
15 BANK 15
20 FOR I=0 to 90*4
30 POKE 6656*I,PEEK (64128+I)
40 NEXT I
50 POKE 2757,PEEK(2757) or 128
60 REM normal table
70 POKE 6656+12,89
80 POKE 6656+25,90
90 REM Shift-Chart
100 POKE 6656+89+12,89+32
110 POKE 6656+89+25,90+32
120 REM Commodore-Chart
130 POKE 6656+2 * 89+12,183
140 POKE 6656+2 * 89+25,184
150 REM CTRL
160 POKE 6656+3 * 89+12,25
170 POKE 6656+3 * 89+25,26
180 REM Change pointer
190 POKE 830,0
200 POKE 831,26
210 POKE 832,89
220 POKE 833,26
230 POKE 834,89 * 2
240 POKE 835,26
250 POKE 836,11
260 POKE 837,27
```

Now you can change the shifted key assignments to anything you want. The new key assignment does not have to be at address 6656. Remember, you also have to change the pointer (830 - 841).

7.5. THE AUXILIARY KEYS

The computer needs extra keys to make it possible for a key to have more than one value. The C-64 had three extra keys: <SHIFT> (both <SHIFT> and <SHIFT LOCK> have the same function), <C=> and <CTRL>.

The C-128 has these, as well as three additional keys: <ESC>, <ALT> and the <CAPS-LOCK> (<ASCII-DI> on the international models).

ESC means escape. This key is not a new invention from Commodore. It has been used for many years in other computers. With the <ESC> key on the C-128 you can do functions like switching from the 40 to the 80 column screen. Every key has a special function in combination with the <ESC> key. But <ESC> is different than the <SHIFT>, <C=> and <CTRL> keys: <ESC> and another key are not pressed simultaneously, but one after each other.

7.5.1. USING THE AUXILIARY KEYS

The auxiliary keys can be read at memory location 211 ($D3). If one of the <SHIFT> keys or <SHIFT-LOCK> key is pressed, the first bit is set. With the <C=> key the second bit is set. The third bit is for <CTRL>, and he fourth for <ALT>. The next bit is set by pressing the <CAP-LOCK> (<ASCII-DIN>) key. Because these keys can be pressed at the same time, there are 31 possible combinations. The following table shows a few of

those possibilities. It is not complete and only intended as an example. Once you understand the principle, a complete table is not necessary.

0 - No Extra keys	1 - SHIFT
2 - COMMODORE - KEY	3 - SHIFT + COMMODORE
4 - CTRL - KEY	5 - SHIFT + CTRL
8 - ALT	10 - ALT + COMMODORE
15 - SHIFT+COMMODORE+CTRL+ALT	16 - CAPS LOCK (ASCII/DIN-KEY)

7.6. EIGHT ADDITIONAL FUNCTION KEYS

When the eight existing function keys are not enough, the following program will solve your problems.

To any of the new function keys you can assign text that is sixteen characters long, similar to the KEY command, and you can enter all the control commands. You get the additional function keys with the <ALT> key. You get the first function key, when you press ALT only, the second with <ALT> and <SHIFT>. The <C=> key and <CTRL> key in combination with <ALT> key activates the third and fourth. As a by-product you get four more function keys but those are much harder to produce. You would have to push several extra keys and <ALT> key at the same time.

The following BASIC loader is a program that lets you assign the eight new <ALT> function keys. You can't edit the text for the extra function keys

directly on the screen. Only after you are finished entering the text will it be displayed. You are then asked if everything is correct. When you program your own use for the <ALT> key you can use the same principles as we did in the following machine language program.

```
10    REM  8 EXTRA FUNCTION KEYS
20    FOR I = 5120 TO 5190
30    READ A
40    S = S + A
45    POKE I,A
50    IF A <>  PEEK (I) THEN  PRINT A,I, PEEK (I)
60    NEXT I
70    IF S <  > 7375 THEN BEGIN
80    PRINT "?ERROR IN DATA  "
90    END
100   BEND
110   BANK 0:  SYS 5120
120   DATA 120,169,13,141,20,3,169,20,141
130   DATA 21,3,88,96,72,152,72,165
140   DATA 211,168,41,8,240,29,152,205
150   DATA 96,20,240,23,141,96,20,10
160   DATA 10,10,10,168,185,0,20,240
170   DATA 13,201,13,240,15,32,210,255
180   DATA 200,208,241,140,96,20,104,168
190   DATA 104,76,101,250,141,74,3,169
200   DATA 1,133,208,76,55,20
210   REM  INPUT OF THE FUNCTION KEYS
220   FOR I = 0 TO 7
230   B$ = "                ":  REM  15 SPACES
240   PRINT "ALT-";I + 1
250   PRINT "INPUT TEXT:  END WITH  LINE FEED"
260   FOR J = 0 TO 14
270   GET  KEY A$
280   IF A$ =  CHR$ (10) THEN 330
290   IF A$ =  CHR$ (13) THEN 330
300   MID$ (B$,J + 1,1) = A$
310   POKE 5248 + J + I * 16, ASC (A$)
320   NEXT J
330   POKE 5248 + J + I * 16,0
340   PRINT B$
350   INPUT "ALL OK  (Y/N)";A$
```

```
360   IF A$ = "N" THEN 230
370   NEXT I
```

Here is the machine language portion of the above routine:

```
1400   78        SEI         ;Prevents Interrupt
1401   A9 OD     LDA #$OD    ;Sets Interrupt vector
1403   8D 14 03  STA $0341
1406   A9 14     LDA #$14
1408   8D 15 03  STA $0315
140B   58        CLI         ;Permit IRQ
140C   60        RTS         ;Back to Basic
140D   48        PHA         ;Accumulator on Stack
140E   98        TYA         ;Y-Register in Accu.
140F   48        PHA         ;Accumulator on Stack
1410   A5 D3     LDA $D3     ;Flag for ALT
1412   A8        TAY         ;Accu. in Y Register
1413   29 08     AND #$08    ;Bit 3 set?
1415   F0 1D     BEQ $1434   ;No, then $1434
1417   98        TYA         ;Value back in Accu.
1418   CD 60 14  CMP $1460   ;Previous Key ALT ?
141B   F0 17     BEQ $1434   ;Yes, then $1434
141D   8D 60 14  STA $1460   ;Stores 8
1420   0A        ASL         ;Move four Bits
1421   0A        ASL         ;To the Left
1422   0A        ASL
1423   0A        ASL
1424   A8        TAY         ;Accu. in Y register
1425   B9 00 14  LDA$1400,Y;Character
1428   F0 0D     BEQ $1437   ;$1437 whenzero
142A   C9 0D     CMP #$0D    ;Return ?
142C   F0 0F     BEQ $143D   ;Yes, then $143D
142E   20 D2 FF  JSR $FFD2   ;Give out Character
1431   C8        INY
1432   D0 F1     BNE $1425
1434   8C 60 14  STY $1460   ;Stores Helpbutton
1437   68        PLA         ;Accumulator from Stack
1438   A8        TAY         ;Accu. to Y Register
1439   68        PLA         ;Accumulator from Stack
143A   4C 65 FA  GMP $FA65   ;To normal Interrupt
143D   8D 4A 03  STA $034A   ;Return in Key Buffer
1440   A9 01     LDA #$01
```



```
1442   85 D0     STA $D0
1444   4c 37 14  JMP $1437
```

7.7. KEYBOARD BEEP

Here is another little program that gives your C-128 programs a more professional appearance: press any key and a low volume beep will sound.

The program changes the IRQ vector, so it reacts very fast to every key press. For this reason it is not written in BASIC, but you can enter it in your computer with the BASIC loader. To make programing easier your computer will also give out a different tone whenever the <RETURN> key is pressed. Below is the assembly language listing for those of you who are familiar with machine language.

```
1400 A9 FF     LDA #$FF      ; Set Value For
1402 8D 06 D4  STA $D406     ; the Beeptone
1405 8D 18 D4  STA $D418     ;
1408 A9 09     LDA #$09
140A 8D 05 D4  STA $D405
140D 78        SEI           ; Prevents Interrupt
140E A9 1A     LDA #$1A
1410 8D 14 03  STA $0314     ; IRQ-Vector open
1413 A9 14     LDA #$14
;Beep-Routine

1415 8D 15 03  STA $0315     ; set
1418 58        CLI           ; Permits Interrupt
1419 60        RTS           ; Back in BASIC
141A 48        PHA           ; Accu. on Stack
141B A5 D5     LDA $D5       ; Val. pushed key
141D C9 58     CMP #$58      ; key pushed?
141F F0 24     BEQ $1443     ; Yes
```

```
1421 C9 4C      CMP #$4C       ; RETURN?
1423 D0 0D      BNE $1431      ; no, then $1431.
1425 A9 67      LDA #$67       ; stores Frequences.
1427 BD 00 D4   STA $D400      ; Tone by RETURN
142A A9 11      LDA #$11
142C 8D 01 D4   STA $D401
142F D0 14      BNE $1445
1431 C9 01      CMP #$01       ; RETURN key?
1433 F0 F0      BEQ $1425      ; Yes, then $1425.
1435 A9 67      LDA #$67       ; set Frequence for
1437 8D 01 D4   STA $D401      ; the Beeptone
143A A9 21      LDA #$21
143C 8D 00 D4   STA $D400
143F A9 11      LDA #$11       ; Waveform
1441 D0 02      BNE $1445      ; Skip Next Line
1443 A9 00      LDA #$00       ; Waveform, when no
                               key pushed
1445 8D 04 D4   STA $D404      ; Store Waveform
1448 68         PLA            ; Accum. from Stack
144A 4C 65 FA   JMP $FA65      ; Normal IRQ
```

Here is the BASIC loader:

```
10   REM  KEY SOUND
20   FOR I = 5120 TO 5195
30   READ A
40   S = S + A
50   POKE I,A
60   NEXT I
70   IF S <  > 8932 THEN BEGIN
80   PRINT "?ERROR{SPACE}IN{SPACE}DATA"
90   END
100   BEND
110   SYS 5120
120   DATA 169,255,141,6,212,141,24,212
130   DATA 169,9,141,5,212,120,169,26
140   DATA 141,20,3,169,20,141,21,3
150   DATA 88,96,72,165,213,201,88,240
160   DATA 34,201,76,208,12,169,103,141
170   DATA 0,212,169,17,141,1,212,208
```



```
180   DATA 20,201,1,240,240,169,103,141
190   DATA 1,212,169,33,141,0,212,169
200   DATA 17,208,2,169,0,141,4,212
210   DATA 104,76,101,250
```

7.8. PROGRAM PAUSE

With this routine you can assign a program pause function to any key, you can stop program execution with the key of your choice. You can answer the door or the telephone without missing any computing.

If you would like to assign the program pause function to a key other than the <NO-SCROLL> key, you have to add the following POKE commands to the end of the BASIC loader:

```
POKE 5134, N
POKE 5140, N
```

The parameter N must contain the value of the key you choose. You can find the exact values in the chapter 7.1.

The interrupted program resumes when you press any other key. This key stays in the keyboard buffer and is read by the next input statement.

```
10   REM  PAUSE-FUNCTION
20   FOR I = 5120 TO 5165
30   READ A
40   S = S + A
50   POKE I,A
```

```
60    NEXT
70    IF S <  > 5361 THEN BEGIN
80    PRINT "?ERROR{SPACE}IN{SPACE}DATA"
90    END
100    BEND
110    SYS 5120
140    DATA 169,11,141,8,3,169,20,141
150    DATA 9,3,96,165,212,201,87,208
160    DATA 10,165,212,201,87,240,250,201
170    DATA 88,240,246,169,15,133,2,169
180    DATA 74,133,3,169,162,133,4,169
190    DATA 0,133,5,76,227,2
```

The routine in machine language:

```
1400 A9 0B     LDA #$0B          : Changes Vector
1402 8D 08 03  STA $0308         ; For "execute
1405 A9 14     LDA #$14          ; next BASIC-Line"
1407 8D 09 03  STA $0309
140A 60        RTS               ; Back to BASIC
140B A5 D4     LDA $D4           ; Value of Key
140D C9 40     CMP #$57          ; NO SCROLL?
140F D0 0A     BNE $141B         ; No
1411 A5 D4     LDA $D4           ; Value of Key
1413 C9 40     CMP #$57          ; NO SCROLL?
1415 F0 FA     BEQ $1411         ; Yes, then wait
1417 C9 58     CMP #$58          ; No Key?
1419 F0 F6     BEQ $1411         ; Yes, then wait
141B A9 0F     LDA #$0F          ; Next BASIC-Line
141D 85 02     STA $02           ; execute
141F A9 4E     LDA #$4A
1421 85 03     STA $03
1423 A9 A2     LDA #$A2
1425 85 04     STA $04
1427 A9 00     LDA #$00
1429 85 05     STA $05
142B 4C E3 02  JMP $02E3
```


7.9. HELP & RUN/STOP KEY ASSIGNMENT

The C-128 has a total of ten function keys, that you can assign yourself. In addition to the eight function keys above the numeric keypad, there are two more: one of them is the <HELP> key, which is actually function key ten. **You can reach the ninth function key with the <SHIFT> and** <RUN-STOP> key combination. This key is assigned with

```
DLOAD"*
RUN
```

These lines load the first program from the disk drive and then run the program. This text can be changed, but not with the KEY Command. If you enter KEY 9 or KEY 10 the computer will print out " ? ILLEGAL QUANTITY ERROR ".

But a little program can take care of that problem:

```
10    REM  FUNCTION KEY  9 + 10
20    REM  [SHIFT] RUN/STOP + HELP
30    FOR I = 4096 TO 4103
40    A = A +  PEEK (I)
50    NEXT I
60    PRINT "^ = RETURN"
70    PRINT  CHR$ (27);"F"
80    FOR I = 0 TO 1
90    PRINT "FUNCTION KEY # "9 + I
110   GET  KEY B$
120   IF B$ <  >  CHR$ (13) THEN BEGIN
130   PRINT B$;
140   B =  ASC (B$)
150   IF B = 94 THEN B = 13
160   POKE 4106 + A + Z,B
```

```
170   Z = Z + 1
180   IF A + Z + 4106 > 4351 THEN  END
190   GOTO 110
200   BEND
210   PRINT
220   POKE 4104 + I,Z
230   A = A + Z
240   Z = 0
250   NEXT I
```

With this program you can change the assignment of these two keys. If you'd like to execute a <RETURN> (CHR$(13)) after the text, you have to enter an up arrow (↑). To make it possible to enter commas we used the GET command.

After minor changes, this program can be also used to change the assignment of the other function keys. This would be very useful for function key assignments with text of more than 128 characters. In this case you would obtain an error using the KEY command.

The assignment of the function keys is in memory at 4096 ($1000) to 4AA2 ($1100). In the first ten bytes, the length of the function key text is stored. With these values you can find out the start address of the text. When you add the first five bytes to each other, you get the start address to the text of F6. You can assign a total of 245 characters to the 10 function keys. The program will not allow you to enter a longer text.

Of course you can use different control characters other than <RETURN>. You can enter these directly. Example: If you like to scroll up your screen ten times you just push the <ESC> and the <W> key 10 times.

If you entered a wrong character you should not use the <INST-DEL> key. This key seems to work alright, but the entering of the <INST-DEL> key will then be stored in the function key definition.