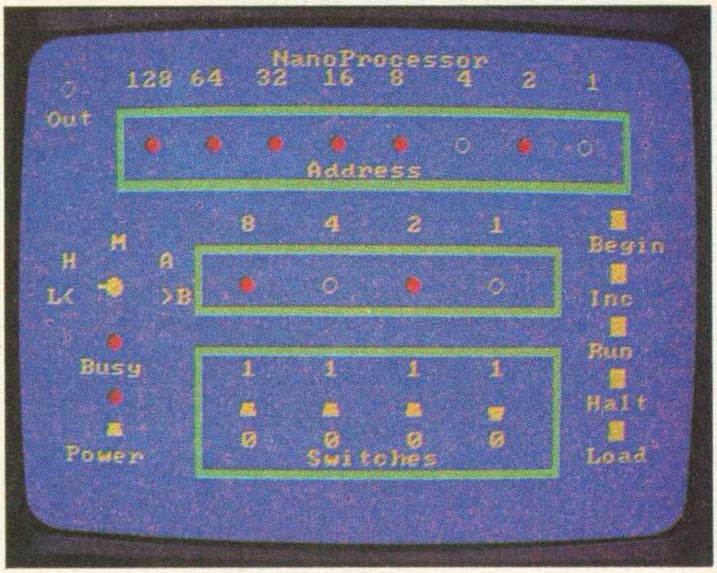
The NanoProcessor

by Roger Wood
and
Wayne Koberstein
HCM Staff

With this simple simulation of the machine's inner workings, you can discover how easy (and fun!) it is to communicate with computers in their own language.



Since the premiere of the movie *Tron*—in which the hero has to fight his way out of a computer's microcircuits—many people have held a fascination for the inner workings of this "thinking machine." Are you one of them? Perhaps your interest has always been there, but you have not yet "taken the plunge" into machine-level programming. Or perhaps you know a great deal about this subject already, but would appreciate a very clear and simple demonstration of how computers "think." If so, you're ready for *NanoProcessor*—a program that emulates the computer at its most fundamental level.

At the heart of a computer, there is nothing but an immense set of on and off switches. But how can such a simple foundation foster such a complex information-handling system? In short, how are all these switches organized? A "real" computer, such as the one you have at home, is such a large system that it would be difficult to see the forest for the trees. But, with *NanoProcessor*, you have a chance to operate and see a much-simplified model of how a computer performs its tasks.

Brain Central

All computers—including the *NanoProcessor*—have a central "brain." It's called the CPU (Central Processing Unit). This brain recognizes and responds to different sets of numbers as instructions. These instructions direct the CPU to carry out certain operations—much as our brains store, handle, and act on information encoded in switch-like neurons. In a computer, information travels along parallel paths of wires and printed circuits called "buses."

As humans, we may think in English, Spanish, or any other language—some subtle, some exact. Computers also "think" in languages—such as BASIC and LOGO. CPU's like our own brains, must translate these high-level languages into encoded information. In computers, this information takes the form of machine language—a set of codes and numerical values expressed as binary numbers. Binary means "two," and implies two choices: on or off; or, in purely numerical terms, 1 or 0.

People tend to think in terms of a ten-based number system because they have ten fingers—but a switch has only two "fingers." (For a detailed look at converting between these number systems see the sidebar "Numbers To Bits And Back.") When you RUN *NanoProcessor* you will notice the row of switches at the bottom of the screen—your only means of shuttling information through this simulated computer (See Photos 1, 2, 3). Each switch only has two positions—up for *on* (1), or down for *off* (0). A switch is therefore the perfect means for conveying binary information.

Banking on Memory

Every computer has a memory area, called "Random Access Memory" (RAM), and a Central Processing Unit (CPU). Memory is the computer's capacity to store information, and is measured in terms of "bytes." A byte generally consists of 8 *bits* of information—where a bit is one binary (on or off) condition.

A CPU performs all the arithmetic that manipulates the numerically-encoded data—the ones and zeros—stored in a computer's RAM. This memory is made up of discrete "locations" in the machine, each of which has an "address." It helps to think of each memory location as a mailbox that not only has an address attached to it, but also a place to put the mail. This mail is the data stored at that location. Each "mailbox" has a limited amount of space that depends on the machine design. Because each of *NanoProcessor*'s memory locations can only store 4 bits, (one nibble), we say it is "nibble-addressable." By simply requesting a particular address, the CPU can immediately find what is contained at that address. This direct addressability of memory by the CPU is what gives a computer the power of *random access*.

The CPU and RAM are connected by three buses: the address bus (8 parallel wires), the data bus (4 parallel wires), and the control bus (See Figure 2). The first provides *access* to each memory location; the second simply moves data to and from each location; and the third carries control signals which control the flow of data between the CPU and memory. Furthermore, the CPU is organized into a system of discrete "registers" that serve as temporary stations for storing and shuffling data.

Look at the *NanoProcessor* front panel. On the middle-left side of the screen is a "rotary switch" with various letters positioned around it. The letters on the right-hand side of this switch—A and B—stand for the A and B registers in the CPU. It is between these two registers that the actual "arithmetic" and logic operations take place. The A register is also called the Accumulator because this is where the answers to many of the commands end up—or *accumulate*.

NUMBERS TO BITS AND BACK

One of the most important aspects of machine language programming (but sometimes most confusing for the novice) is converting digital numbers to binary and vice versa. To make this as easy as possible, we have employed two aids: 1) Whenever we list a *binary* number, we precede it with a percent (%) sign; and 2) *NanoProcessor* displays the decimal equivalent of each bit above the address and data windows of the front panel (see diagram below). We refer to these decimal equivalents as the "weight" of the bits.

To quickly convert a binary number to a decimal number, simply add up the weights of the "1" (on) bits. For example, to convert %1111 1010, refer to the following diagram:

	128	64	32	16	8	4	2	1
	*	*	*	*	*	*	*	*
%	1	1	1	1	1	0	1	0

Then add 128+64+32+16+8+2 and you can easily arrive at the correct decimal equivalent: 250. (Also, see Figure 1 for converting the numbers 0—15 to binary.)

Figure 1

Decimal	Binary
0	%0000
1	%0001
2	%0010
3	%0011
4	%0100
5	%0101
6	%0110
7	%0111
8	%1000
9	%1001
10	%1010
11	%1011
12	%1100
13	%1101
14	%1110
15	%1111

Turning On

First, press **P** to turn on the Power to your *NanoProcessor*. Make sure the rotary switch is pointing to the letter M, for Memory. You move this switch left (counter-clockwise)with the < (less than) key, and right (clockwise) with the > (greater than) key.

At the top of the screen, you should see an address box containing a long row of "lights" with numbers across the top. This is the "location counter" shown inside the CPU of Figure 2. It displays the 8-bit address of the location currently being interrogated by the CPU. Notice the vertical row of buttons at the right side of the screen. These buttons represent *NanoProcessor's* functions. Press the **B** (for Begin) key on your keyboard. This effectively turns off all the lights in the address box, indicating that you have returned to the first address in memory: the 0 (zero) location. Now press the **I** key, for Increment. This moves you to the next address: location 1. If you repeatedly press **I**, you will continue to step through successive locations.

Notice that, as you step through each location, the row of 8 lights in the address box changes. These lights display the *address* of the "mailbox." To view the *contents* of this mailbox, look at the row of 4 lights directly above the toggle switches. This shows the value stored at the current location. If you were to move the rotary switch pointer to A, you would see the contents of the A register. To examine the B register, point the switch to the letter B. Now, move the pointer to the letters H or L at left. These access the "high nibble" (the first or left-most 4 bits) and the "low nibble" (the last or right-most 4 bits) in the 8-bit address.

Entering Data

The next step is to "fill" these locations so that the processor has something to process. With the rotary switch in the M position, try toggling the switches in the switch box. Nothing happens? Don't worry; turn some of these switches "up" and then press **L**, for Load. Now you have something. Any switch that is *on* has a corresponding light glowing just above it.

You have just entered your first "data" into the *NanoProcessor*. Now move the rotary switch to the H position and try the same exercise. This time, when you press **L**, lights not only come on in the "contents" box, but the same pattern of lights appears in the high (left-most) nibble of the address box. Moving the rotary switch to L (for Low nibble) and loading a value affects the low nibble (right-most) half of the 8-bit address in the same way. Once you have thus designated a full 8-bit address, move the pointer to the M position again to view the contents of *that same address*. By doing this, you have, in effect, moved to this address location, and can enter data there.

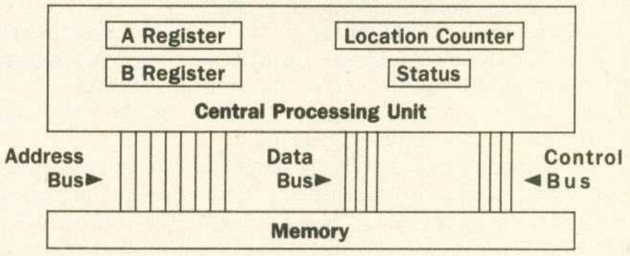
If you next move the rotary switch pointer to the A or B position and try to enter data, you will not be able to—because whatever goes in or out of these registers has to do so while the *NanoProcessor* is running instructions encoded into memory. You will also notice a small Output light (labeled "Out") at the upper left of the screen. We will explain the use of this in the *NanoAssembler* program next issue.

Your next job is to enter your first machine-language program on the *NanoProcessor*.

Programming The Machine

A CPU executes commands *sequentially*. As it runs a program, it steps through this sequence in much the same way you "incremented" through each memory location. However, the program may instruct the CPU to take other paths—"branching" to many different locations before completing its task. You are able to program this processor by entering three different kinds of data: 1) encoded commands; 2) pure numbers; and 3) addresses. As with any program, it is the *logic* of this sequence that determines what the processor will do.

Figure 2
Simplified Block Diagram of the NanoProcessor



NanoProcessor understands 16 different commands—its "instruction set." Although initially expressed in one nibble, some commands require additional memory locations to hold the data necessary to execute the command. Figure 3 lists these 16 commands, showing each corresponding binary code; how many nibbles in a program the instruction requires; its "mnemonic"; which (if any) flags in the status register the instruction affects; and a brief explanation of the command function. As you develop more complicated programs, you will have to understand and use more of these commands. But, for now, try a very short routine—one that simply adds two small numbers together.

Figure 3: Instructions Set

Dec.	Binary	Nibbles per instr.	Mnemonic	Flags* affected C Z	Function
0	%0000	1	ADD	Y Y	Add the contents of B register to the contents of A register—result in A.
1	%0001	2	LDA #	N Y	Load A with number following instruction.
2	%0010	3	LDA *addr*	N Y	Load A with number at location specified by *addr*.
3	%0011	3	STA *addr*	N N	Store the contents of A at location specified by *addr*.
4	%0100	1	TAB	N N	Transfer contents of A to B.
5	%0101	1	TBA	N Y	Transfer contents of B to A.
6	%0110	1	RRC	Y Y	Rotate A right one bit through carry.
7	%0111	1	RLC	Y Y	Rotate A left one bit through carry.
8	%1000	1	AND	Y Y	Logically AND A and B—Result in A.
9	%1001	1	OR	Y Y	Logically OR A and B—Result in A.
10	%1010	1	XOR	Y Y	Logically XOR A and B—Result in A.
11	%1011	3	BZ *addr*	N N	Branch to *addr* if Zero flag is set.
12	%1100	3	BNZ *addr*	N N	Branch to *addr* if Zero flag is not set.
13	%1101	3	BCS *addr*	N N	Branch to *addr* if Carry flag is set.
14	%1110	3	BCC *addr*	N N	Branch to *addr* if Carry flag is not set.
15	%1111	3	JMP *addr*	N N	Branch to *addr* unconditionally.

****Flags affected refers to whether or not the instruction has any effect on the flags in the status register. The C column stands for the Carry flag (did the operation result in a carry being generated?), and the Z stands for the Zero flag (did the operation result in a zero?). A Y appears in the column if the flag is affected by the instruction. An N indicates the flag is not changed by the instruction.**

Sample Program 1

Addr	Code	Mnemonic	Remark
0	%0001	LDA #3	;Get first number
1	%0011		
2	%0100	TAB	;Move to B
3	%0001	LDA #7	;Get second number
4	%0111		
5	%0000	ADD	;Figure sum
6	%1111	JMP 6	;Jump self to stop
7	%0110		
8	%0000		

Sample Program 2

Addr	Code	Mnemonic	Remark
0	%0010	LDA 240	;Get first number
1	%0000		
2	%1111		
3	%0100	TAB	;Move to B
4	%0010	LDA 241	;Get second number
5	%0001		
6	%1111		
7	%0000	ADD	;Figure sum
8	%0011	STA 248	;Put low nibble in memory
9	%1000		
10	%1111		
11	%1110	BCC 19	;Only one nibble answer
12	%0011		
13	%0001		
14	%0001	LDA #1	
15	%0001		
16	%1111	JMP 21	;All done
17	%0101		
18	%0001		
19	%0001	LDA #0	;Zero A
20	%0000		
21	%0011	STA 249	;Put high nibble in memory
22	%1001		
23	%1111		
24	%1111	JMP 24	;Jump self to terminate
25	%1000		
26	%0001		

Sample Program 3

Addr	Code	Mnemonic
0	%0001	LDA #2
1	%0010	
2	%0100	TAB
3	%1000	AND
4	%0110	RRC
5	%0011	STA 254
6	%1110	
7	%1111	
8	%0000	ADD
9	%0011	STA 254
10	%1110	
11	%1111	
12	%0000	ADD
13	%0011	STA 254
14	%1110	
15	%1111	
16	%0001	LDA #6
17	%0110	
18	%0011	STA 254
19	%1110	
20	%1111	
21	%0000	ADD
22	%0011	STA 254
23	%1110	
24	%1111	
25	%0000	ADD
26	%0011	STA 254
27	%1110	
28	%1111	
29	%0000	ADD
30	%0011	STA 254
31	%1110	
32	%1111	
33	%0001	LDA #13
34	%1101	
35	%0011	STA 254
36	%1110	
37	%1111	
38	%1111	JMP 38
39	%0110	
40	%0010	

Roundabout Addition

Sample Program 1 will add the numbers 7 and 3, and the answer will end up in the Accumulator. If you haven't already, turn on the power by pressing **P**. Now, press **B** for **B**egin, and confirm that the rotary is pointing at M (Memory). Now "key-in" this program with the following procedure:

1. Toggle the switches to the on and off positions corresponding to the bits of the number identified as Code in the program—up (or on) for 1, and down (or off) for 0. Notice that each binary code is preceded by a % (percent) sign to make it easy to distinguish binary numbers from decimal quantities (See "Numbers To Bits And Back" for details).
2. Check that the address indicated by the location counter is the correct one for that Code, and then Press **L** for Load.
3. Press **I** for Increment. This will take you to the next address.
4. Repeat steps 1 through 3, loading the correct nibble into each address, and move on to the next set until you've loaded all the nibbles in the proper order.
5. Once you have completed loading the program, press **B** again to return to address 0. Then step through each memory location with the **I** key to be certain the program is entered properly.
6. Now press **B** for **B**egin once more, then **R** for **R**un. Note that you may **H**alt the program at any time (by pressing **H**) and continue again by pressing **R**.

Let's go over Sample Program 1 step-by-step to see exactly what it does when **L**oaded and **R**un. First it uses the "LoaD Accumulator immediate" instruction (abbreviated LDA #) to load the number stored at the address immediately following the instruction code (address 1) into the Accumulator. This number (in this case a %0011 or decimal 3) is one of the two to be added. At address 2 is an instruction to Transfer the number from the Accumulator into register B (TAB). Address

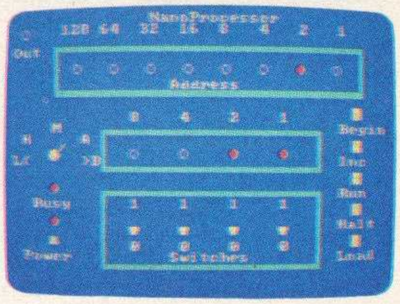


Photo 1: This shows the contents of the A register in the initial step of Sample Program 1. First, the program moves one number (3 or %0011) of an addition problem into A.

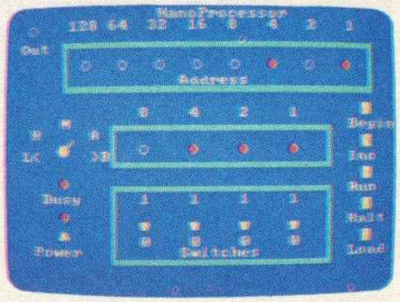


Photo 2: Next, after the first number moves to the B register, the second number (7 or %0111) is loaded in A.

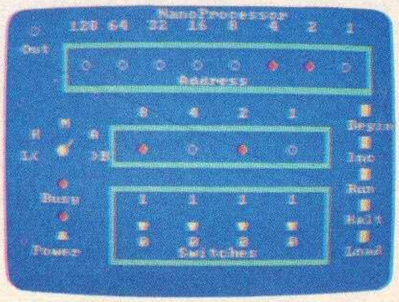


Photo 3. The A register now shows the result (10 or %1010) after the contents of A and B have been added together.

3 contains another LDA# instruction to Load a %0111 (7 decimal) from address 4 into register A. The instruction at address 5 actually ADDs the number in register B to the number in A, and places the answer in A. Address 6 contains a JuMP instruction (JMP *addr*), that tells the machine to jump to the address specified at the next two memory locations—7 and 8. All addresses are two nibbles, and the *NanoProcessor* follows a procedure standard to many microprocessors where the low nibble of the address is in the next location (7 in this case) and the high nibble in the following one (8). We call this a "jump self" because we specify address 6 (%0000 0110) as the place to jump to.

When you **R**un this program, the "busy light" remains on and both rows of lights flash different patterns as the CPU steps through the program. The *Nanoprocessor* has been made to **R**un slowly so that you can track each instruction as it is executing. When the program "hangs-up" at location 6, press **H** (for Halt) to make the busy light go off. Now turn the rotary switch to point at A. Here you find the answer to the addition problem: %1010 or 10 decimal. Keep the pointer in this position and run the program again, after pressing **B**egin. Watch the A register change values—first 3 (%0011), then 7 (%0111), then the answer, 10 (%1010). Photos 1 through 3 show this sequence.

Moving On

In Sample Program 1, the machine added two numbers and got an answer that it could express in one 4-bit nibble. But, what if this answer had been larger than one 4-bit nibble—say, a number like 23 (%0001 0111)? Fifteen (%1111) is the largest number that one nibble can express. When a processor adds two numbers together whose answer is bigger than its registers can hold, the answer "overflows" the register. When this happens in *NanoProcessor*, a "carry flag" is set to 1 in a special Status register of the CPU. (This register is not directly accessible to the user.) The program has to contain commands that recognize the condition of this flag (either 1 when an overflow has occurred, or 0 when there is no overflow) and take appropriate action. You can determine which instructions cause changes in the carry flag by studying the C column (under "Flags affected") of Figure 3. If there is a Y in the C column, the instruction will affect the carry flag —i.e., set it to 1 if an overflow occurs, or reset it to 0 if no overflow occurs.

Sample Program 2 adds the numbers 11 (%1011) and 12 (%1100) to arrive at 23 (%0001 0111). Not only does the program have to check the carry flag, but because the answer doesn't fit in one register, it has to place the answer someplace else. The solution is to designate certain memory locations as data areas—two for input and two for output. Program 2 fetches the two numbers to be added from memory locations 240 (%1111 0000) and 241 (%1111 0001). These addresses are input areas. This means that before you **R**un the program, you must manually Load the numbers to be added at these locations—place 11 at address 240, and 12 at address 241.

Similarly, the output area is at locations 248 (%1111 1000) and 249 (%1111 1001). The low nibble of the answer (%0111 in our example above) appears at 248, and the high nibble (%0001) at address 249.

This program also handles the overflow condition described above. If the answer does overflow a nibble, the program places a 1 in the accumulator and stores it as the answer's high nibble. If, however, the answer is less than 15 (and fits into one nibble), the program branches to another address, where it loads a 0 into A and stores that instead. This introduces one of 4 "conditional jump commands," which we will explore more fully in next issue's companion "utility," *NanoAssembler*.

Program 3 is a "mystery program" that actually accesses the "sound chip" we've built into the *NanoProcessor*. Watch next issue for an explanation of how this program works. Or perhaps, in the meantime, you will learn enough by playing with *NanoProcessor* to figure this one out yourself. The best way to learn the details of operating the the *NanoProcessor* is to use it and experiment by creating your own machine-language programs.

Saving and Loading

With *NanoProcessor*, you can Save and Load the entire 256 memory locations (%0000 0000 through %1111 1111) to disk (and/or tape on The C-64, Atari, and TI-99/4A). Use the Save command listed in your Control Capsule and type in a file name in response to the prompt. To Load, use the Load command and type in the name of the file you wish to load.

HCM Glossary terms: CPU, bus, machine language, binary numbers, Random Access Memory (RAM), byte, address, nibble, location counter, accumulator, register, instruction set, mnemonic, branch, jump, conditional jump, status register, zero flag, carry flag, overflow, weight (of bits).

HCM

For your key-in listings, see HCM PROGRAM LISTINGS Contents.

CONTROL CAPSULE
NanoProcessor

Key	Function
B	Set address to zero.
I	Increment address by 1.
R	Run program.
H	Halt program.
L	Load location.
<	Move rotary switch counter-clockwise.
>	Move rotary switch clockwise.
P	Toggle Power switch.
E	End program (only when Power is off)
1-4	Toggle panel switch 1 = left-most bit, 4 = right-most bit.

CONTROL CAPSULE
NanoProcessor

Key	Function
OPTION	Save file.
SELECT	Load file.

CONTROL CAPSULE
NanoProcessor

Key	Function
F1	Save file.
F3	Load file.

CONTROL CAPSULE
NanoProcessor

Key	Function
CONTROL W	Save file.
CONTROL Q	Load file.

CONTROL CAPSULE
NanoProcessor

Key	Function
FN 6	Save file.
FN 7	Load file.

CONTROL CAPSULE
NanoProcessor

Key	Function
FCTN 6	Save file.
FCTN 8	Load file.

NANOPROCESSOR *Continued*

IBM PC/PCjr, TANDY 1000

```
F 3070 FOR IT=44 TO 260 STEP 30:PUT (IT,32),OFFLIT,PSET:NEXT IT
Y 3080 PUT(25,120),OFFLIT,PSET:PUT(25,160),DN,PSET:PUT(25,145),OFFLIT,PSET:PUT(2,6),OFFLIT,PSET:AD%(253)=0:RETURN
A 3090 DATA "0000","0001","0010","0011","0100","0101","0110","0111","1000","1001","1010","1011","1100","1101","1110","1111"
N 3100 DATA "O1L2B","O2L2C","O2L2C#","O2L2D","O2L2D#","O2L2E","O2L2F","O2L2F#","O2L2G","O2L2G#","O2L2A","O2L2A#","O2L2B","O3L2C","O3L2C#","O3L2D"
N 3110 DATA "O3L2D#","O3L2E","O3L2F","O3L2F#","O3L2G","O3L2G#","O3L2A","O3L2A#","O3L2B","O4L2C","O4L2C#","O4L2D","O4L2D#","O4L2E","O4L2F","O4L2F#","O4L2G"
K 3120 DATA "BL3L6UR7","BH3H4DF5","BU3U7RD7"
S 3130 DATA "BE3E5DG5","BR3R6UL7"
L 3140 DATA "UL3D2RD3R5U3RU2L6","BD2R3U2LU3L5D3LD2R4"
X 3150 DATA "BL3D4R7U8L7D5R","BL2D3R5U6L5D5"
Y 3160 LOCATE 5,1:PRINT "INPUT FILE NAME:";:ROW=5:COL=18:MAXLEN=8:SELECT$="ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789!@#$%&(){}'`-_":IN$="":GOSUB 3190:IF IN$="" THEN RETURN ELSE FL$=IN$
B 3170 LOCATE 7,1:PRINT "WHICH DRIVE? A":ROW=7:COL=14:MAXLEN=1:SELECT$="AaBb":IN$="A":GOSUB 3190:IF IN$="" THEN FL$="A:"+FL$:RETURN ELSE FL$=IN$+":"+FL$:RETURN
F 3180 ' INPUT SUBROUTINE *****
X 3190 PT=1
P 3200 LOCATE ROW,COL,0:PRINT IN$;SPACE$(MAXLEN-LEN(IN$));:LOCATE ROW,COL+(PT-1),1:K$="":WHILE K$="":K$=INKEY$:WEND
C 3210 IF K$=CHR$(13) THEN RETURN
X 3220 IF INSTR(SELECT$,K$) THEN IN$=LEFT$(IN$,PT-1)+K$+MID$(IN$,PT+1):PT=PT+1:IF PT>MAXLEN THEN PT=MAXLEN:GOTO 3200
K 3230 IF K$=CHR$(8) AND PT>1 THEN IN$=LEFT$(IN$,PT-2)+MID$(IN$,PT):PT=PT-1:GOTO 3200
Z 3240 IF K$=CHR$(0)+CHR$(83) THEN IN$=LEFT$(IN$,PT-1)+MID$(IN$,PT+1):GOTO 3200
J 3250 IF K$=CHR$(0)+CHR$(82) AND LEN(IN$)<MAXLEN THEN IN$=LEFT$(IN$,PT-1)+" "+MID$(IN$,PT):GOTO 3200
H 3260 IF K$=CHR$(0)+CHR$(77) AND LEN(IN$)>=PT THEN PT=PT+1:IF PT>MAXLEN THEN PT=MAXLEN:BEEP:GOTO 3200
I 3270 IF K$=CHR$(0)+CHR$(75) AND LEN(IN$)>1 THEN PT=PT-1:IF PT<1 THEN PT=1 :GOTO 3200
N 3280 GOTO 3200
M 3290 DIM ERCD(14),ERM$(14):RESTORE 3350:FOR I=1 TO 14:READ ERCD(I):READ ERM$(I):NEXT:RETURN
E 3300 ' ERROR ROUTINES *****
U 3310 CLOSE:LOCATE 24,1,0:R=ERR:L=ERL:FOR Z=1 TO 14:IF ERCD(Z)=R THEN 3330
O 3320 NEXT:PRINT "ERROR #";R;" IN LINE #";L;:GOTO 3340
H 3330 PRINT ERM$(Z);"-- #";R;
N 3340 SOUND 110,20:FOR TD=1 TO 4000:NEXT:LOCATE 24,1:PRINT SPACE$(39);:IF MD1=1 THEN RESUME 660 ELSE RESUME 710
N 3350 DATA 64,BAD FILE NAME,69,COMMUNICATIONS BUFFER OVERFLOW,25,DEVICE FAULT,57,DEVICE I/O ERROR,24,DEVICE TIMEOUT,68,DEVICE UNAVAILABLE,61,DISKETTE IS FULL,72,DISK MEDIA ERROR,71,DISK NOT READY,70,THIS DISK IS WRITE PROTECTED
O 3360 DATA 53,FILE IS NOT ON THE DISK,14,DATA STORAGE AREA FULL--START NEW FILE,67,TOO MANY FILES ON THIS DISK,52,BAD FILE NUMBER OR NAME
```

HCM

NANOPROCESSOR

TI-99/4A

```
P  100 REM ******************
Q  110 REM * NANOPROCESSOR *
P  120 REM ******************
M  130 REM COPYRIGHT 1985
H  140 REM EMERALD VALLEY PUBLISHING CO.
A  150 REM BY ROGER WOOD
R  160 REM HOME COMPUTER MAGAZINE
G  170 REM VERSION 5.5.1
T  180 REM TI BASIC OR EXTENDED BASIC
A  190 DIM AD(255)
N  200 DIM D$(15)
O  210 DIM TN(31)
C  220 GOSUB 3940
Y  230 SW=2
N  240 CALL SCREEN(4)
E  250 GOSUB 3880
F  260 ON SW+1 GOSUB 710,730,750,770,790
X  270 CALL KEY(0,K,S)
N  280 IF S<1 THEN 270
F  290 C=POS("EP<,>.1234",CHR$(K),1)
K  300 IF C=0 THEN 270
W  310 IF C=1 THEN 4270
A  320 IF C<>2 THEN 390
F  330 CALL SOUND(1,-5,0)
M  340 CALL HCHAR(20,5,128)
U  350 CALL HCHAR(21,5,92)
O  360 GOSUB 1260
Y  370 GOSUB 1360
L  380 GOTO 430
B  390 IF C<7 THEN 410
R  400 CALL SOUND(1,-5,0)
K  410 ON C-2 GOSUB 2300,2300,2380,2380,2190,2190,2190,2190
J  420 GOTO 260
Q  430 GOSUB 810
X  440 IF C>1 THEN 490
P  450 CALL HCHAR(20,5,40)
N  460 CALL HCHAR(21,5,91)
G  470 GOSUB 1480
G  480 GOTO 250
B  490 ON C-1 GOSUB 1570,2130,2130,2150,2150,2190,2190,2190,2190,1640,1820,1880
R  500 IF RF=1 THEN 510 ELSE 430
A  510 CALL KEY(0,K,S)
Y  520 IF S=0 THEN 650
T  530 IF K<>72 THEN 610
M  540 CALL SOUND(1,-5,0)
D  550 CALL HCHAR(19,29,94)
N  560 CALL HCHAR(17,5,40)
U  570 GOSUB 3850
B  580 CALL HCHAR(19,29,93)
V  590 RF=0
E  600 GOTO 430
Z  610 IF K<>80 THEN 650
E  620 CALL HCHAR(17,5,40)
U  630 RF=0
A  640 GOTO 450
M  650 ON AD(CA)+1 GOSUB 2720,2790,2830,2910,3040,3060,3080,3170,3270,3330,3390,3450,3500,3550,3600,3650
D  660 ON SW+1 GOSUB 710,730,750,770,790
A  670 GOSUB 1260
G  680 GOSUB 1360
B  690 IF RF=0 THEN 560
A  700 GOTO 510
M  710 TEMP=16*(CA/16-INT(CA/16))
J  720 RETURN
J  730 TEMP=INT(CA/16)
X  740 RETURN
J  750 TEMP=AD(CA)
D  760 RETURN
M  770 TEMP=AR
B  780 RETURN
R  790 TEMP=BR
O  800 RETURN
B  810 CALL KEY(0,K,S)
Z  820 IF S<1 THEN 810
R  830 C=POS("PB<,>.1234IRL",CHR$(K),1)
X  840 IF (K=6)+(K=12)THEN 890
A  850 IF C=0 THEN 810
X  860 IF (C>2)*(C<7)THEN 880
O  870 CALL SOUND(1,-5,0)
C  880 RETURN
U  890 CALL CLEAR
X  900 CALL COLOR(13,2,4)
H  910 IF K=6 THEN 1020
C  920 PRINT TAB(8);"SAVE FILE": : : :"DEVICE AND/OR FILE NAME:"
S  930 INPUT "":FL$
J  940 IF FL$="" THEN 1110
C  950 OPEN #1:FL$,INTERNAL,OUTPUT,FIXED 64
T  960 FOR IT=0 TO 245 STEP 7
L  970 PRINT #1:AD(IT);AD(IT+1);AD(IT+2);AD(IT+3);AD(IT+4);AD(IT+5);AD(IT+6)
F  980 NEXT IT
K  990 PRINT #1:AD(252);AD(253);AD(254);AD(255)
N 1000 CLOSE #1
U 1010 GOTO 1110
J 1020 PRINT TAB(8);"LOAD FILE": : :"DEVICE AND/OR FILE NAME:"
W 1030 INPUT "":FL$
P 1040 IF FL$="" THEN 1110
D 1050 OPEN #1:FL$,INTERNAL,INPUT ,FIXED 64
D 1060 FOR IT=0 TO 245 STEP 7
M 1070 INPUT #1:AD(IT),AD(IT+1),AD(IT+2),AD(IT+3),AD(IT+4),AD(IT+5),AD(IT+6)
N 1080 NEXT IT
C 1090 INPUT #1:AD(252),AD(253),AD(254),AD(255)
W 1100 CLOSE #1
Q 1110 CALL CLEAR
S 1120 CALL COLOR(13,2,2)
K 1130 GOSUB 4090
```

Continued

```
A 1140 CALL HCHAR(13,4,32)
X 1150 CALL HCHAR(12,5,32)
K 1160 CALL HCHAR(13,6,32)
O 1170 ON SW+1 GOSUB 710,730,750,770,790
E 1180 GOSUB 2460
G 1190 GOSUB 1260
I 1200 CALL HCHAR(20,5,128)
E 1210 CALL HCHAR(21,5,92)
X 1220 GOSUB 1360
B 1230 CALL COLOR(13,7,4)
G 1240 CALL SCREEN(4)
U 1250 GOTO 810
N 1260 PL$=D$(TEMP)
Z 1270 FOR IT=10 TO 22 STEP 4
R 1280 CH=INT(IT/4)-1
Y 1290 IF SEG$(PL$,CH,1)="1" THEN 1320
S 1300 CH2=40
F 1310 GOTO 1330
Z 1320 CH2=128
J 1330 CALL HCHAR(12,IT,CH2)
E 1340 NEXT IT
B 1350 RETURN
T 1360 AD$=D$(INT(CA/16))
Z 1370 AD$=AD$&D$(CA-16*INT(CA/16))
O 1380 REM WRITE ADDRESS TO SCREEN
O 1390 FOR IT=6 TO 27 STEP 3
G 1400 CH=INT(IT/3)-1
A 1410 IF SEG$(AD$,CH,1)="1" THEN 1440
X 1420 CH2=40
V 1430 GOTO 1450
G 1440 CH2=128
D 1450 CALL HCHAR(6,IT,CH2)
D 1460 NEXT IT
U 1470 RETURN
G 1480 FOR IT=10 TO 22 STEP 4
L 1490 CALL HCHAR(12,IT,40)
U 1500 NEXT IT
H 1510 FOR IT=6 TO 27 STEP 3
U 1520 CALL HCHAR(6,IT,40)
N 1530 NEXT IT
Y 1540 CALL HCHAR(2,3,40)
E 1550 AD(253)=0
S 1560 RETURN
X 1570 CALL HCHAR(10,29,94)
R 1580 CA=0
R 1590 GOSUB 1360
Y 1600 ON SW+1 GOSUB 710,730,750,770,790
R 1610 GOSUB 1260
A 1620 CALL HCHAR(10,29,93)
M 1630 RETURN
U 1640 CALL HCHAR(13,29,94)
B 1650 CA=CA+1
Y 1660 GOSUB 1790
C 1670 IF SW>2 THEN 1760
C 1680 IF SW<2 THEN 1710
T 1690 TEMP=AD(CA)
I 1700 GOTO 1750
O 1710 IF SW=0 THEN 1740
I 1720 GOSUB 730
F 1730 GOTO 1750
I 1740 GOSUB 710
S 1750 GOSUB 1260
S 1760 GOSUB 1360
T 1770 CALL HCHAR(13,29,93)
M 1780 RETURN
Q 1790 IF CA<256 THEN 1810
S 1800 CA=CA-256
Q 1810 RETURN
L 1820 CALL HCHAR(16,29,94)
N 1830 CALL HCHAR(17,5,128)
Z 1840 RF=1
Y 1850 GOSUB 3850
Q 1860 CALL HCHAR(16,29,93)
C 1870 RETURN
S 1880 CALL HCHAR(22,29,94)
S 1890 CT=0
D 1900 TEMP=0
S 1910 PL$=""
U 1920 FOR IT=22 TO 10 STEP -4
B 1930 CALL GCHAR(19,IT,M)
Y 1940 PL$=CHR$(M-43)&PL$
A 1950 TEMP=TEMP+((M-91)*2^CT)
B 1960 CT=CT+1
F 1970 NEXT IT
G 1980 ON SW+1 GOSUB 2010,2030,2070,2070,2070
R 1990 CALL HCHAR(22,29,93)
U 2000 RETURN
I 2010 CA=INT(CA/16)*16+TEMP
N 2020 GOTO 2040
F 2030 CA=(CA-(INT(CA/16)*16))+(TEMP*16)
F 2040 GOSUB 1260
F 2050 GOSUB 1360
G 2060 RETURN
C 2070 AD(CA)=TEMP
N 2080 IF SW>2 THEN 2100
N 2090 GOSUB 1270
O 2100 IF CA<253 THEN 2120
Y 2110 ON CA-252 GOSUB 2620,2680,2700
D 2120 RETURN
N 2130 GOSUB 2300
H 2140 GOTO 2160
R 2150 GOSUB 2380
O 2160 ON SW+1 GOSUB 710,730,750,770,790
B 2170 GOSUB 1260
N 2180 RETURN
K 2190 C=C-6
V 2200 CALL GCHAR(19,C*4+6,M)
V 2210 IF M=91 THEN 2240
B 2220 CALL HCHAR(19,C*4+6,91)
T 2230 GOTO 2250
A 2240 CALL HCHAR(19,C*4+6,92)
X 2250 RETURN
P 2260 ON SW+1 GOSUB 710,730,750,770,790
S 2270 GOSUB 1260
E 2280 GOSUB 1360
L 2290 RETURN
W 2300 IF SW=0 THEN 2370
F 2310 SW=SW-1
C 2320 CALL SOUND(1,-5,0)
Q 2330 GOSUB 2460
N 2340 CALL KEY(0,K,S)
O 2350 IF S>=0 THEN 2370
F 2360 GOTO 2300
W 2370 RETURN
X 2380 IF SW=4 THEN 2450
D 2390 SW=SW+1
Z 2400 CALL SOUND(1,-5,0)
F 2410 GOSUB 2460
A 2420 CALL KEY(0,K,S)
I 2430 IF S>=0 THEN 2450
W 2440 GOTO 2380
N 2450 RETURN
A 2460 ON SW+1 GOSUB 2480,2500,2530,2570,2600
D 2470 RETURN
Z 2480 CALL HCHAR(13,4,123)
B 2490 RETURN
N 2500 CALL HCHAR(13,4,41)
N 2510 CALL HCHAR(12,5,32)
J 2520 RETURN
U 2530 CALL HCHAR(13,4,32)
X 2540 CALL HCHAR(12,5,124)
O 2550 CALL HCHAR(13,6,32)
Z 2560 RETURN
U 2570 CALL HCHAR(12,5,32)
G 2580 CALL HCHAR(13,6,42)
G 2590 RETURN
Z 2600 CALL HCHAR(13,6,125)
N 2610 RETURN
G 2620 IF AD(253)<1 THEN 2650
Y 2630 CH2=128
J 2640 GOTO 2660
Z 2650 CH2=40
X 2660 CALL HCHAR(2,3,CH2)
L 2670 RETURN
F 2680 CALL SOUND(500,TN(AD(254)),0)
J 2690 RETURN
N 2700 CALL SOUND(500,TN(AD(255)+16),0)
O 2710 RETURN
D 2720 AR=BR+AR
G 2730 IF AR<16 THEN 2770
G 2740 AR=AR-16
B 2750 CF=1
N 2760 GOTO 2780
G 2770 CF=0
E 2780 GOTO 3780
T 2790 CA=CA+1
B 2800 GOSUB 1790
Z 2810 AR=AD(CA)
H 2820 GOTO 3780
H 2830 CA=CA+1
P 2840 GOSUB 1790
Y 2850 TEMP=AD(CA)
L 2860 CA=CA+1
W 2870 GOSUB 1790
B 2880 TEMP=TEMP+16*AD(CA)
N 2890 AR=AD(TEMP)
A 2900 GOTO 3780
T 2910 CA=CA+1
A 2920 GOSUB 1790
H 2930 TEMP=AD(CA)
U 2940 CA=CA+1
V 2950 GOSUB 1790
A 2960 TEMP=TEMP+16*AD(CA)
W 2970 AD(TEMP)=AR
N 2980 AD(TEMP)=AR
E 2990 IF TEMP<253 THEN 3010
H 3000 ON TEMP-252 GOSUB 2620,2680,2700
B 3010 CA=CA+1
M 3020 GOSUB 1790
U 3030 RETURN
H 3040 BR=AR
B 3050 GOTO 3780
L 3060 AR=BR
X 3070 GOTO 3780
U 3080 TEMP=0
H 3090 IF CF=0 THEN 3110
H 3100 TEMP=8
C 3110 IF INT(AR/2)=AR/2 THEN 3140
J 3120 CF=1
H 3130 GOTO 3150
U 3140 CF=0
W 3150 AR=INT(AR/2)+TEMP
F 3160 GOTO 3780
M 3170 TEMP=0
L 3180 IF CF=0 THEN 3200
D 3190 TEMP=1
N 3200 IF AR<8 THEN 3240
R 3210 CF=1
F 3220 AR=AR*2-16+TEMP
A 3230 GOTO 3260
L 3240 CF=0
A 3250 AR=AR*2+TEMP
K 3260 GOTO 3780
J 3270 GOSUB 3730
```

Continued

NANOPROCESSOR *Continued*


```
C 3280 FOR IT=1 TO 4
L 3290 IF (SEG$(AR$,IT,1)="1")*(SEG$(BR$,I
       T,1)="1")THEN 3300 ELSE 3310
A 3300 AR=AR+(2^(4-IT))
T 3310 NEXT IT
L 3320 GOTO 3780
E 3330 GOSUB 3730
D 3340 FOR IT=1 TO 4
Z 3350 IF (SEG$(AR$,IT,1)="1")+(SEG$(BR$,I
       T,1)="1")THEN 3360 ELSE 3370
M 3360 AR=AR+(2^(4-IT))
D 3370 NEXT IT
N 3380 GOTO 3780
K 3390 GOSUB 3730
A 3400 FOR IT=1 TO 4
R 3410 IF (SEG$(AR$,IT,1)="1")<>(SEG$(BR$,
       IT,1)="1")THEN 3300 ELSE 3310
F 3420 AR=AR+(2^(4-IT))
W 3430 NEXT IT
W 3440 GOTO 3780
D 3450 IF ZF=0 THEN 3470
C 3460 GOTO 3650
Z 3470 CA=CA+3
U 3480 GOSUB 1790
Q 3490 RETURN
E 3500 IF ZF=1 THEN 3520
Q 3510 GOTO 3650
H 3520 CA=CA+3
M 3530 GOSUB 1790
S 3540 RETURN
Q 3550 IF CF=0 THEN 3570
D 3560 GOTO 3650
W 3570 CA=CA+3
Z 3580 GOSUB 1790
V 3590 RETURN
D 3600 IF CF=1 THEN 3620
P 3610 GOTO 3650
W 3620 CA=CA+3
P 3630 GOSUB 1790
V 3640 RETURN
J 3650 CA=CA+1
W 3660 GOSUB 1790
T 3670 TEMP=AD(CA)
G 3680 CA=CA+1
U 3690 GOSUB 1790
D 3700 TEMP=TEMP+16*AD(CA)
M 3710 CA=TEMP
O 3720 RETURN
T 3730 AR$=D$(AR)
A 3740 BR$=D$(BR)
L 3750 AR=0
Q 3760 CF=0
X 3770 RETURN
M 3780 IF AR=0 THEN 3810
U 3790 ZF=0
Z 3800 GOTO 3820
W 3810 ZF=1
W 3820 CA=CA+1
P 3830 GOSUB 1790
V 3840 RETURN
Z 3850 FOR D=1 TO 300
O 3860 NEXT D
W 3870 RETURN
U 3880 RANDOMIZE
A 3890 AR=INT(RND*15)+1
T 3900 BR=INT(RND*15)+1
I 3910 CA=INT(RND*253)+1
R 3920 AD(CA)=INT(RND*15)+1
N 3930 RETURN
B 3940 CALL CLEAR
P 3950 PRINT TAB(7);"The NanoProcessor"::
       :::"        PLACE ALPHA LOCK DOWN":
       :"   PLEASE WAIT WHILE I SET UP"
V 3960 PRINT :::::
Y 3970 FOR IT=0 TO 18
A 3980 READ CH,CH$
W 3990 CALL CHAR(CH,CH$)
E 4000 NEXT IT
L 4010 FOR IT=0 TO 15
W 4020 READ D$(IT)
F 4030 NEXT IT
A 4040 FOR IT=0 TO 31
N 4050 READ TN(IT)
O 4060 NEXT IT
A 4070 CALL CLEAR
D 4080 CALL COLOR(13,7,4)
K 4090 CALL SCREEN(2)
U 4100 PRINT "(            NanoProcessor":"ut":
       "  128 64 32 16 8  4  2  1":"  $!!!!
       !!!!!!!!!!!!!!!!!!!%"
H 4110 PRINT " # (  (  (  (  (  (  (  (  `"
       :" #         addr           `":"  &!
       !!!!!!!!!!!!!!!!!!!!!!'"
R 4120 PRINT :"          8    4    2    1";TAB(2
       7);"]":"    m  $!!!!!!!!!!!!!!!!% Begi
       n"
L 4130 PRINT "h | a# (   (   (   (  `":"   ~
          #       `         ]":"l<  >b&!!
       !!!!!!!!!!!!'    Inc"
M 4140 PRINT :TAB(27);"]":"   (   $!!!!!!!!
       !!!!!!%  Run":"busy  # 1   1   1   1
        `"
C 4150 PRINT TAB(6);"# [   [   [   [ `
       ]":"  (  # 0   0   0   0 `  Halt":"
          [   #    switches     `"
S 4160 PRINT "Power&!!!!!!!!!!!!!!!'    ]"
       :TAB(25);"Load"
Y 4170 CALL HCHAR(3,2,79)
P 4180 RETURN
I 4190 DATA 33,00FFFF,35,0303030303030303,
       36,0003030303030303,37,00C0C0C0C0C0
       C0C,38,030303,39,C0C0C
B 4200 DATA 41,E0FE7F0F,42,077FFEF
E 4210 DATA 96,C0C0C0C0C0C0C0C,128,001C3E3
       E3E1C,40,001C3E3E3E1C,91,00000000FE
       7C7C7C
P 4220 DATA 92,007C7C7CFE,93,7F7F7F7F7F7F7
       F7F,94,003E3E3E3E3E3E
V 4230 DATA 123,0000000077FFEF,124,001818
       18181818,125,00000000E0FE7F07,126
       ,3C7EFFFFFFFF7E3C
M 4240 DATA 0000,0001,0010,0011,0100,0101,
       0110,0111,1000,1001,1010,1011,1100,
       1101,1110,1111
M 4250 DATA 247,262,277,294,311,330,349,37
       0,392,415,440,466,494,523,554,587,6
       22,659,698,740,784,831,880,932
L 4260 DATA 988,1047,1109,1175,1245,1319,1
       397,1480
M 4270 CALL CLEAR
F 4280 END
```

HCM

PLAINS OF SALISBURY

APPLE // Family

```
T 100  REM ******************
K 110  REM *   THE PLAINS    *
A 120  REM *  OF SALISBURY   *
E 130  REM ******************
B 140  REM COPYRIGHT 1985
E 150  REM EMERALD VALLEY PUBLISHING CO.
Q 160  REM BY WILLIAM K. BALTHROP
Y 170  REM AND THE HCM STAFF
X 180  REM HOME COMPUTER MAGAZINE
N 190  REM VERSION 5.5.1
G 200  REM APPLE // FAMILY APPLESOFT
S 210  TEXT : HOME : FLASH : FOR Z = 6 TO
       8: VTAB Z: HTAB 6: PRINT SPC( 31)
       : NEXT : NORMAL : VTAB 7: HTAB 7: P
       RINT "    THE PLAINS OF SALISBURY
       "
T 220  IF PEEK (104) = 64 THEN 250
M 230  POKE 104,64: POKE 16384,0
M 240  PRINT CHR$ (4);"RUN PLAINS"
T 250  DIM L(2,6,5),P$(2),M$(4),R(2),M(2)
       ,S%(24),C%(6): ONERR GOTO 2100
I 260  DEF FN F1(N) = L(P,N,1) + 1: DEF
       FN F3(N) = L(P,N,2) - (Q - 1) * 40
B 270  P = 1:P2 = 2:Q = 0:M(1) = 0:M(2) =
       0:PRNT = 2048:OFF = 2051:SOUND = 20
       54:D$ = "IMJK" + CHR$ (27) + CHR$
       (13):M$(1) = "LEFT":M$(2) = "MIDDL
       E":M$(3) = "RIGHT"
A 280  FOR Z = 1 TO 23: VTAB Z: HTAB 1:S%
       (Z) = PEEK (41) * 256 + PEEK (40)
       - 1: NEXT
N 290  PD = 0: IF PEEK (48905) = 76 AND
       PEEK (48911) = 0 THEN PD = 1
V 300  IF PEEK (64435) = 6 THEN VTAB 11
       : HTAB 4: PRINT "* MAKE SURE CAPS L
       OCK KEY IS DOWN *"
W 310  FOR Z = 1 TO 900: READ A$: IF VAL
       (A$) < > 9999 THEN NEXT
W 320  FOR Z = 2048 TO 2245: READ K: POKE
       Z,K: NEXT : READ K: IF K < > 9999
       THEN PRINT "DATA ERROR" CHR$ (7):
       END
C 330  FOR Z = 2326 TO 2374: READ K: POKE
       Z,K: NEXT Z: RESTORE
R 340  VTAB 13: PRINT "     DO YOU WANT SO
       UND EFFECTS ?Y (Y/N)"
N 350  VTAB 13: HTAB 32: GET A$: PRINT A$
       : POKE 2064,48 * (A$ < > "N"): IF
       A$ < > "Y" AND A$ < > "N" AND A$
       < > CHR$ (13) THEN 350
R 360  VTAB 15: PRINT "    LOAD AN OLD GAM
       E FROM DISK ?N (Y/N)";
C 370  VTAB 15: HTAB 32: GET A$: PRINT A$
       : IF A$ < > "Y" AND A$ < > "N" AN
       D A$ < > CHR$ (13) THEN 370
Q 380  IF A$ = "Y" THEN GOSUB 1930: GOSU
       B 1640: GOSUB 1440:S = 100: GOTO 49
       0
B 390  N = 1: VTAB 11: HTAB 1: CALL - 958
       : PRINT : HTAB 7: INPUT "BLACK KNIG
       HT'S NAME : ";A$:P$(1) = LEFT$ (A$
       ,10): IF A$ = "" THEN PRINT CHR$
       (7);: GOTO 390
```

Continued

TYPE-IN LISTINGS