CONTENTS

Getting started 1
Key functions 2
LOAD 3
SAVE 4
INPUT, PRINT, and LIST 5
USR 6
Debugging aids 7
Real-time clock 8
Program conversions 9
Editing 10
Drive select 10
FORMAT 11
CREATE 12
DELETE 13
DISASSEMBLER 14

APPENDIX

Error codes 16
Diskette format 17
Memory map 18
Hardware modifications 19
Reserved IO devices 22
Using disk data files 23

GETTING STARTED

We'll skip the usual congratulations on your purchase and get right down to business. Turn on your system, press BREAK, insert the HEXDOS diskette, and press D. HEXDOS will boot up using some parts of the cold start routine, including the MEMORY SIZE and TERMINAL WIDTH functions, which run normally. It then provides a prompt of HEXDOS rather than the OK prompt used by BASIC-in-ROM. This prompt indicates that HEXDOS is still alive and presumably well.

This manual will assume that you are familiar with BASIC-in-ROM and in particular with the requirement to press RETURN after typing each line. If not, please read the manuals which came with your system and become familiar with BASIC-in-ROM before proceding.

After loading HEXDOS as in the first paragraph above, you can treat the system as if you were using BASIC-in-ROM, because in fact you are. HEXDOS gets its incredible compactness by using the power built into the system as much as possible.

Please avoid the USR function until you read the section describing its special powers, and use LOAD and SAVE only as described below until you read the appropriate sections. If you leave the write-protect tape on the disk, you need not fear damaging it just by tinkering with the new commands at your fingertips. Once you remove this protection you are on your own.

To get a directory, type LOAD/ . This will load the disk directory as a BASIC program which you can LIST. To load a program, for instance FORMAT, type LOAD FORMAT . To save a program after editing, type SAVE. To create a new copy and call it NAME (for example) type SAVE NAME.

SPECIAL KEYS

All normal key functions of BASIC-in-ROM are available under HEXDOS, along with the following additional features. Any word enclosed in ( ) refers to a single key. Where two keys are in [ ] hold the first key (usually CTRL or SHIFT) while pressing the second key.

Key names appearing in CAPITALS should by pressed while holding the shift key.

(ctrl) suspend output and wait until the key is released.
(rub out) move the cursor nondestructively backward.
(esc) move the cursor nondestructively forward.
(line feed) cancel the current line
(RETURN) allows editing of the resultant output (see EDITING)
(repeat) break a BASIC program (identical to [(ctrl) c]). If change is made as in appendix D, repeats the last command entered in immediate mode.
[(ctrl) L] instant screen clear (from a BASIC program, PRINT CHR$(12);). (Note that output is suppressed while you hold the ctrl key; the screen-clear character will not be printed until you release the ctrl key.)

SAVE

SAVE functions similarly to LOAD with a few exceptions. See the description of LOAD for general comments. SAVE in its normal function dealing with tape is unnecessary under HEXDOS. LIST #2 will perform the same function as the old SAVE:LIST without writing extraneous data on the tape. (See also LIST).

SAVE save the current BASIC program to the same tracks it came from. Note that the same disk must still be in the drive, since it would write over whatever was on the corresponding tracks on another disk.

SAVE (filename) check to see that (filename) does not already exist, create a new file named (filename) large enough to hold the current program, and save the program in the new file.

SAVE &(filename) save the current program in the existing file (filename). The old contents of (filename) are lost. Example: SAVE &TEST

SAVE #(aex) close output file (aex) and write the last buffer to the disk (generates F ERROR if used on an input file).

SAVE #(aex1),(aex2) save the 2K bytes starting at (aex2) on track number (aex1). USE WITH EXTREME CAUTION, PREFERABLY ON AN EMPTY DISK !!!

Transfer"O"
Save#0, 6000

Filenames may be any length and may contain any character except :. They must not start with #,#,&,or /. These characters would be interpreted as part of the LOAD or SAVE commands. A filename which starts with $ will be interpreted as a machine language file with load address information; see USR, CREATE, and DISK FORMAT.

LOAD

HEXDOS adds the following additional forms of LOAD. Items enclosed in < > refer to an item that should appear there (don't type the < >'s).

<aex> any arithmetic expression valid in BASIC. A number such as 5, a variable such as I, a subscripted variable such as A(I), or even an expression such as A(I)^N-LOG(SIN(Q/2))*43.787 would all be valid. The expression must result in a value in the range -32768 to 32767.

<filename> any valid filename. (See SAVE for filename rules.)

LOAD/ load the directory. Use LIST to view it after loading.

LOAD <filename> load <filename> as a BASIC program. Example: LOAD FORMAT. If <filename> starts with $, it is assumed to refer to a machine language file containing load address information as noted under USR.

If executed in a program, also RUNs the resulting BASIC program. Note that using LOAD $name in a program will clear all variables and re-start the BASIC program at the first line; this is not true of LOAD $name in the immediate mode.

LOAD #<aex>,<filename> open <filename> as the data file specified by <aex>. Example: LOAD #5,MYFILE
Each use of this form allocates a 2K byte buffer from BASIC's string space. A call to FRE(X) deallocates all file buffers.

LOAD #<aex1>,<aex2> directly loads track number <aex1> to the 2K bytes starting at <aex2>. USE WITH CAUTION, preferably with nothing in memory which is not saved. Example: LOAD #1,8192 Load #0, 6000

INPUT, PRINT, AND LIST

Input and output are routed by HEXDOS using the forms INPUT#n, PRINT#n, and LIST#n, where n selects the appropriate device from the table below. For instance, to save a program on tape in same format as SAVE:LIST would save it normally, simply LIST#2 . This will list the program but send the listing to the tape rather than the screen. Such a tape may be loaded by BASIC-in-ROM or by HEXDOS just as you would load any other tape.

Similarly, INPUT can get its data from any device listed below, and PRINT can send its output to any device. If any of these verbs are used without # , device 0 is assumed. Thus programs dealing only with the keyboard and screen will run without modification.

Use of the #n form of any of these verbs immediately after THEN requires a : between THEN and PRINT, INPUT, or LIST.

Input devices

- 0 keyboard
- 1 reserved for expansion (printer)
- 2 6850 ACIA (tape interface, printer, or modem depending on your hardware)
- 3 reserved for expansion (modem)
- 5 thru 25 (odd) disk files

Output devices

- 0 video screen
- 1 reserved for expansion (printer)
- 2 6850 ACIA
- 3 reserved for expansion (modem)
- 4 thru 24 (even) disk files

USR

The USR function is automatically defined for a number of uses by HEXDOS. The usual way to access it from BASIC is T=USR(X) or PRINT USR(X), where X selects the function:

X)= 256 tone generator (you will need to connect a speaker as noted in the hardware section). X is 256*length+pitch.

0 (= X (256 input a character from device X. This form may not be used in the immediate mode nor in a PRINT statement.

-4 (= X (0 returns the value of one of the four bytes of the real-time clock. USR(-4) returns the least-significant byte, while USR(-1) returns the most-significant byte.

X = -5 operates like SYS on the TRS-80 or PET; that is, jumps to a machine language routine at the location specified by an arithmetic expression following USR. That is, T=USR(-5) 0 would jump to location $0000 (warm start), while T=USR(-5) -1024 (i.e., 64512) would jump to the disk boot routine at $FC00. (CAUTION: once you pass control to a machine language program, you lose the protection provided by BASIC. That is, it may erase your program, write on the disk, or just go away and not come back. You may recover from the latter by doing a warm start with BREAK followed by W.)

X = -6 Jump to the ROM monitor (return to BASIC via a warm start, which may also be done by a GO at $0000).

X = -7 Jump to the routine last loaded by LOAD $. You may also set this up for your own USR function just as in the BASIC manual except that the vector is at $F0 or decimal 240 and a parameter to be passed must appear after the right parenthesis. Example: USR(-7) 2*B(I)+J. To create a machine language file, see CREATE. PRINT USR(-7) X

DEBUGGING AIDS

HEXDOS adds some powerful program analysis features to BASIC. It retains the simple break on ctrl-C, and will also stop on the REPEAT key or the BREAK key (see hardware section). This action may be disabled just as originally by POKE 530,1 .

TRACE

To have a full trace of your BASIC program as it executes, POKE 530,2 before you RUN or CONT it. This will cause BASIC to print the line number in [ ] after each statement is executed. Program output is intermixed with this trace, allowing you to see the action of each line as it executes. Using this option with a program which does input from devices other than the keyboard may cause errors because the system will attempt to print the trace information to the selected input or output device.

SINGLE-STEP

The single-step mode uses tracing as above but pauses and waits for a keystroke after each statement. POKE 530,3 enters the single-step mode. After each line, LINE FEED will step to the next statement, while RETURN will disable the single-step and trace options and continue normal execution. To break out of the program to print variables, list the program, or correct the problem, hold REPEAT or BREAK (depending on hardware) and press LINE FEED. You may also activate trace or single-step by placing the appropriate POKEs in your program. POKE 530, 0 disables any of these options and returns to normal execution.

REAL-TIME CLOCK

HEXDOS includes precisely the type of real-time clock supporting software that OSI mentions in their literature on the 610 board. Two jumpers must be added to the 610 board at regular tie points to activate this option as noted in the hardware section.

With the hardware activated, there is a 4-byte clock which counts seconds since the last cold or warm start. The contents of this clock are available through USR.

The clock also includes an option to designate a machine language program to be called at some time in the future (up to 65536 seconds or about 18 hours later). This facility is used by HEXDOS to turn off the disk drive after a short idle period.

To have a routine called on a time delay, first load the machine language into memory via POKEs, the monitor, or an assembler. Then poke the two's complement of the time delay into 238 ($EE) and 239 ($EF) with the high byte in 239. Last, poke the start address of the routine into 236 ($EC) and 237 ($ED) with the high byte in 237. Now continue with programming or running other programs or just sit back and watch. After the appropriate delay, your machine language program will take over. It may use any registers but should avoid using page 0 below $F0. It may return to whatever program was active before it was called by a RTS.

PROGRAM CONVERSIONS

BASIC programs saved on tape will be compatible with HEXDOS. Load them just as you normally would, with LOAD.

Programs which use USR should be changed to use USR(-7) by moving the start vector to 240 ($F0) and placing the parameter to be passed immediately after the) rather than within ( ). Values returned by USR are handled normally.

Programs which deal with tape will run as is, except that the old function of SAVE to switch output to tape is replaced by PRINT #2, ... You probably will prefer to change your programs to work with data on disk rather than tape, now that you paid all that money for the disk. See LOAD %, SAVE %, and the section on INPUT and PRINT.

Programs written under another DOS may be loaded most easily by saving them on tape and then loading the tape under HEXDOS. To save a program on tape from OS65D, type the following:

```
DISK!"IO ,03":LIST
```

(The space between IO and the comma is essential in this line!) Programs using disk under OS65D will need some minor changes to the input and output sections. OPEN must be changed to LOAD and CLOSE must be changed to SAVE (see LOAD and SAVE).

EDITING

HEXDOS adds the capability to edit a line which you have already entered without retyping the entire line, and also lets you edit a line with true backward and forward cursor control.

To edit a line, for instance line 40, simply list the line but hold the shift key while pressing return. That is, type

LIST 40 [(shift) (return)]

Line 40 will list on the screen and be placed in the edit buffer just as if you had typed the line but had not yet pressed (return). Now you may edit it just as you may edit a line when first typing it by using (rubout) to backspace the cursor and (esc) to move the cursor forward. Simply type over any part of the line you wish to change. Note that you must use (esc) to move the cursor to the end of the line and then press (return) to enter the line when you have finished editing it.

DRIVE SELECT

If your system has dual disk drives, you may select the drives alternately with LOAD! . This command selects the drive which is currently idle and homes its head to track 0. A program may detect which drive is active as in this example: IF (PEEK(49152) AND 64) THEN PRINT "DRIVE A IS ACTIVE" . This quantity is true if drive A is selected and false if drive B is selected. WARNING - HEXDOS does not keep track of drive selection. Before doing any disk write (program SAVE, write to a data file, or track write), ensure the correct drive is selected.

FORMAT

This utility program provides a way to format new or used disks so they will be compatible with HEXDOS. It does so by disabling the normal error traps in the disk handlers and then writing nulls on each track. It also names and dates the disk and sets up an empty directory. Please note that this process

ERASES

the entire disk.

The only interaction required from the user is to load the disk to be erased and provide the name and date for the disk. HEXDOS is automatically copied to track 0 of the new disk so it will boot up with (break) D even when using copies of the original disk. Please note that this feature is provided as a convenience for your own use and is not a license to produce copies for others.

****
**
** THIS PROGRAM MAY BE HAZARDOUS TO YOUR DISK'S HEALTH !! **
**
****

CREATE

This utility program will create an empty data file of a specified length. It is not needed for program files, because they are created automatically by SAVE (filename).

The only user input required is the name of the file and its length in pages (a page is 256 bytes or characters - 1K requires 4 pages). Since the length must be specified in advance and may not be changed, it should be set up somewhat longer than you expect to need. Unused space will be filled out with nulls.

To make a machine language file, first use CREATE to make a file N tracks long, where N is the number of 2048 byte tracks required. Then load the machine language routine into memory (not necessarily at its normal location), and place the proper start address in the two bytes immediately preceding its first byte, low-order byte first. Now use SAVE # to save an image of memory starting at (start address)-2 on consecutive tracks of the file you have created for it. Now using LOAD $ will load the machine language to its proper location (with the address in the two bytes immediately before it) and set up the USR(-7) vector to start USR(-7) at the start address of the routine.

DELETE

This utility program provides a way to delete obsolete files from a disk (HEXDOS itself does not allow you to delete a file once it has been created). DELETE will list the files in the directory one at a time and pause for a command to either retain the file or delete it. To delete the file, press D three times. To retain the file, press any other character, such as the space bar. Do not interrupt this program, since that might leave the directory disagreeing with the actual contents of the disk.

DISASSEMBLER

This program interactively disassembles the machine code in an area of memory specified by the user. It first requests the address of the desired area. Respond with the address in hex using any appropriate number of digits.

The disassembler will print one screen full of information and then pause. Typing a C (for continue) will resume the listing for another page, while any other character will return to a request for a new start address.

Instructions are printed in standard 6502 mnemonics, but the addressing mode is specified by appending a letter from the table below. The operand is printed in hex. Each byte is also printed in ASCII to the right of a ; after the instruction (the high-order bit of each byte is ignored).

I - immediate
M - memory (direct, absolute)
Z - zero page
  - accumulator, implied, or relative
V - (ind,X)
W - (ind),Y
S - zero page,X
X - absolute,X
Y - absolute,Y

APPENDIX

ERROR MESSAGES

Error messages from BASIC consist of a ? followed by two letters and the word ERROR. Messages from HEXDOS consist of a single letter from the following table and the word ERROR.

A - seek error (the track header indicated a different track than intended. To ignore this, POKE 229,255. Loading a track will load its formatting information as well which may aid in locating the problem. POKE 229,0 to restore normal operation.
B - file not found in the directory.
C - read or write attempted past end of file.
D - the file you are attempting to create already exists.
E - disk is protected against writing.
F - attempt to write to an input file.
G - directory has an error. LOAD/:LIST will let you view it.

OSI made an error in the routine which prints error messages which causes the second character to be printed as a graphic character (bit 7 is a 1). HEXDOS prints the correct two-letter error identifier by masking off bit 7. If you wish to print graphic characters (decimal 128 thru 255) to any output device (screen, tape, or disk file), use the statement POKE 227,255 to disable this feature. POKE 227,127 will restore it.

Error messages generated during IO will be printed to the device being accessed by that IO. During input from a disk file, this will cause an F ERROR. To see the actual error message, temporarily change the INPUT or PRINT statement to use device 0 (keyboard or screen) so that the message will be printed normally.

DISKETTE FORMAT

Track zero has its own unique format to comply with the requirements of the bootstrap firmware in ROM on the 600 board. The first byte is the high-order byte of the load address, followed by the low-order byte, and the number of pages of data on track zero (always 8 under HEXDOS).

The remaining tracks start with a sync character ($57) and the track number in binary, followed by 2048 bytes of data. The directory resides on track 1 as a BASIC program. The line number of the name of each file is the starting track number of that file. It includes all tracks up to but not including the first track of the next file. The last line of the directory contains a *, indicating the first available track.

BASIC programs are stored in core-image form, with the first byte of the program on the first track being loaded to the beginning of BASIC's program space. (Under HEXDOS, this is $0B01.)

A machine language file is specified by a filename beginning with $. The first two bytes in a machine language file are the load and start address of the program, with the low-order byte first. These two bytes will be loaded into the two bytes immediately preceeding the address they specify, and also into $F0, linking the machine language routine to USR(-7). A machine language file may consist of more than one track just as may a BASIC program, but only the first track need contain the load address (the following tracks are assumed to be contiguous).

MEMORY MAP

00 warm start JUMP
03 prompt JUMP
0A USR JUMP
0D NULL
0E POS
0F terminal width
13-59 text buffer
79-7A start of program
7B-7C start of variables
7D-7E start of arrays
7F-80 end of arrays
81-82 start of strings
83-84 end of strings
85-86 end of memory
87-88 current line number
89-8C start of current stmt
8F-90 data pointer
BC GETCHR code
C2 REGETCHR code
D8 current track number
D9-DD temporaries
DE start track of program
DF end track of program - 1
E0-E1 temporaries
E2 IO device number
E4 editting flag
E5 ignore seek error
E8-EB clock
EC-ED timed routine vector
EE-EF time until activate EC-ED
F0-F1 vector to USR(-7)
0200 cursor
0201 character at cursor
0202 character being output
0212 debug control
0222-02FF data file headers
0300-0AFF HEXDOS
0B00- BASIC workspace

(2816)

HARDWARE MODIFICATIONS

None of the following mods is essential to use HEXDOS, but each adds a bit of usefulness to the system. If you are not familiar with procedures for working on integrated circuits, please have someone who is make the solder connections. For temporary use of the mods which just require jumpers between provided tie points, you can simply place wires (#22 or smaller) in the holes without soldering, but you may have some difficulty with intermittent operation using this method.

REAL-TIME CLOCK

This requires that you add jumpers at tie points provided by OSI to activate the interrupt-driven clock. The two jumpers necessary are added to the 610 board.

The square pad nearest pin 9 of U11 is the source for a pulse every second. There is a row of four square pads across the ends of U10 and U11. These pads connect to the interrupt inputs of U72. The second pad from the end nearest U10 is the interrupt used by HEXDOS. Connect this pad to the one-second pulse line.

Near pin 40 of U72 are 3 square pads arranged in a triangle. Connect the one nearest U72 (U72's interrupt output) to the pad nearest J3 (the NMI line on the bus). The clock will be active the next time HEXDOS is used.

DISK MOTOR CONTROL

OSI made an error in the polarity of the MOTOR-ON signal, which they fixed by permanently tying the line to ground. There is a spare inverter on the 610 board which will correct the polarity. First, on the adapter board which connects the disk cable to the 610 board, remove the jumper between pins 4 and 12. If a land has been cut at pin 4, use the old wire to replace it.

Back on the 610 board, cut the land running from pin 14 of U72. Connect pin 14 to pin 3 of U5, and the separated land to pin 4 of U5. You will now have motor control with the real-time clock of HEXDOS, which will deselect the drive and turn off the motor if the disk is not accessed for a short time. The disk is automatically reactivated by any subsequent access.

BREAK KEY

The break key is wired to the hardware reset line of the computer. It is, alas, in a location where it is often accidentally pressed. You may alleviate this problem and make this single key do the work of ctrl-C as follows.

Disconnect the break key from the reset line and ground by cutting those lands. (Be careful in cutting the ground land as it must still continue past the disconnected key). You will need to add a normally open pushbutton somewhere out of the way to replace the function of the BREAK key. I suggest just behind the keyboard where it is accessible but not hazardously so.

Now connect the break key to R0 and C7 (the lines to the repeat key), and connect the repeat key to R5 and C2, which will change it to the repeat-last-command function as noted under special keyboard features.

AUTOMATIC POWER-ON RESET

This mod simply takes advantage of the automatic reset provided to U72 and routes it to the 600 board via an unused pin in the ribbon cable.

On the 610 board, tie pin 34 of U72 to pin 11 of J1. Then, on the 600 board, tie pin 11 of J1 to pin 40 of U8. Now the system will automatically reset and give the D/C/W/M prompt any time it is turned on.

TONE GENERATOR

This uses the RTS line of the 6850 on the 610 board (the floppy ACIA at $C010, not the cassette ACIA at $F000), toggled rapidly by software, to drive an audio amplifier and speaker. Use an LM386 or equivalent for the amplifier. A convenient place to mount it is between U11 and U67 on the 610 board. Connect the input to pin 5 of U71, and the output to your choice of speaker. The BASIC program below will generate a continuous tone of about 800 Hz for testing purposes.

```
10 T=USR(2860)
20 GO TO 10
```

RESERVED IO DEVICES

Device selection is done by a skip chain rather than a table of driver addresses. The only convenient way to implement the additional devices is by intercepting the existing routine.

The input and output routines are called through vectors at $0218 and $021A respectively, just as the CIP manual shows. Just change the vectors to point to your drivers (a warm start restores the values used by HEXDOS). If you wish the provided devices to remain active, you must test the device number (see memory map) for the number you wish to assign to your device. If the device number is not one of yours, the routine should do a JMP to the address which was originally in the vector. Your routine must preserve all register values and should not tamper with page 0 below $F2 unless you know what you're doing. It can terminate with an RTS or simply jump to the normal IO routine (It accepts 0 or 1 for the keyboard and 2 or 3 for the cassette interface!).

Because of the large number of devices (including all the data files), it was impractical to assign each device a single bit, so there is no provision for simultaneous output to more than one device. One possible way to implement this would be to set up device 1 to set the device pointer to 0, JSR OUTPUT, set the pointer to 2, JSR OUTPUT, reset the pointer to 1, and RTS. Note that you must preserve the registers during this sequence.

The character to be output is passed in the A register, and an input character is expected in the A register. The X register will sometimes have the current POS. It may always be obtained at $0E, and the terminal width at $0F.

I suggest removing the disk or using a blank one during all machine language testing to preclude disasters!

USING DISK DATA FILES

A HEXDOS data file may be visualized as a video screen which may be written to and read from. A PRINT statement writes the same characters to a disk file as it would write on the screen. This includes control characters such as carriage return and line feed, which are not normally visible on the screen. An INPUT statement accessing a disk file will react exactly as if you typed the characters it contains on the screen by hand.

The following short example should help to clarify disk data file use for those who are not familiar with the concepts involved. This example will assume a file named MYDATA already exists (use the CREATE utility to make such a file 1 track long for this example). First, we must associate a file number with the file MYDATA and prepare it for writing. This process is commonly referred to as "opening" the file. HEXDOS uses the BASIC verb LOAD for this purpose:

```
LOAD #4,MYDATA
```

This sets the disk file for writing and effectively clears the screen so that writing will start at the beginning of the file. Print a line into the file just as you would on the screen:

```
PRINT #4,"HELLO THERE!!"
```

You may also print numbers. To print several numbers on one line so that they may be read properly, you must print commas between them just as you would if you were typing on the screen:

```
A=12.5:B=23
PRINT #4,A;",";B
```

After writing to the file, it must be properly closed to insure that all data is correctly entered on the disk. HEXDOS uses the BASIC verb SAVE to close a file:

```
SAVE #4
```

Now the data file is actually on the disk, and may be read by any program.

Since BASIC does not permit using the INPUT statement from the keyboard, we will have to create a short program to show how to read from the data file we just created:

```
NEW
10  LOAD  #5,MYDATA
20  INPUT  #5,A$
30  PRINT  A$
40  INPUT  #5,X,Y
50  PRINT  X,Y
```

Line 10 opens the file as before, but uses an odd file number to designate an input file. Line 20 reads the first line of the file as if you had typed the line on the keyboard. Line 30 shows the results. Line 40 reads the next line, on which we printed the numbers 12.5 and 23 previously. Line 50 shows the results. Since we are not changing the file, there is no need to close it.

You may output on the same line with several different PRINT statements by ending them with a semicolon, just as you can on the screen. For the most compact data files, use NULL 0 to prevent BASIC from placing nulls after each line (they take up space but do not appear in the data when it is read back). HEXDOS selects this setting during boot-up, so it need not be changed unless you change the setting elsewhere.

WARNING FOR STRING MANIPULATION ROUTINES

HEXDOS does not defeat the infamous "arbage collector" routine and its buss (see March 1981 issue of PEEK(65)). Data buffers share the string space, but the garbage collector considers them free space, so it will destroy all file buffers which are open when it is called. The FRE(X) function calls this routine, and it is also called automatically anytime free memory is less than 256 bytes.

If your program does extensive string operations, the garbage collector may interfere with disk operations. There are two ways to get around this:

a) input your data, do your string manipulations, and then reopen all data files. This does not work if you must manipulate strings as they are read from disk.

b) put the file buffers in a protected memory area. At the beginning of your program, prior to using any strings, open all data files. After opening all files, include the statement

```
T=PEEK(129):POKE 133,T:T=PEEK(130):POKE 134,T
```

This decreases the apparent memory size by the amount used for buffers, but protects them from BASIC. If you run the program repeatedly, the end-of-memory pointer will eventually work its way down until you have no free memory left. Either re-boot HEXDOS from disk or POKE the original values back into 133 and 134.

UTILITY PACKAGE

The programs included in this package are useful or entertaining in their own right, but they are intended also to demonstrate some techniques for fully using the power in your system which is made accessible by HEXDOS. CHECKBOOK is an example of the use of serial disk files, requiring that the entire file be read into memory and then manipulated by the program. ADDRESSBOOK, on the other hand, makes use of random-access files to allow use of a bigger data structure than can exist in main memory by bringing in only the desired record at any given time. RANFILE provides the necessary subroutines and a short program to demonstrate how a random access file allows a program to save or recover a piece of information without reading an entire file.

SURROUND demonstrates a means for accessing the keyboard in a real-time program which must continue running whether or not a key is pressed. BACKGAMMON, on the other hand, shows how to use HEXDOS to reduce the keyboard debouncing, decoding, and status detection requirements in your program. Both SURROUND and BACKGAMMON use the tone generator for sound effects.

$LIFE is a machine language program which may be loaded by HEXDOS by name. BSR shows another way of using machine language by using a BASIC program to set up the machine language and link it to USR. This also shows how to use the USR(-7) function under HEXDOS.

CHECKBOOK

This program uses a serial data file to maintain the ledger for your personal checking account. It records the check number, date, amount, and the payee or other notation used on the check. Deposits, service charges, regular monthly deductions, etc. are best handled by assigning them a unique series of "check numbers". The program is menu-driven, which means that you simply select from a menu of possible actions at any given time. Entering the checks as you write them may seem somewhat tedious at the time, but the real benefit comes with the arrival of your monthly statment. Simply select "balance statement" from the menu and enter the check and deposit numbers which have cleared the bank during the month. The program provides the balance which should currently appear in your checkbook as well as the balance which should appear on your bank statement.

ADDRESSBOOK

This program is similarly menu-driven, but it uses random-access files and actually accesses the disk only when necessary. Record lookup is accomplished using the Soundex code to find a given last name, which allows for slight misspellings. Since such a system can find several similar entries, it produces a query after finding any record which is close to the desired last name, requiring a yes/no response. Since there may be several entries under the same last name, even an exact match requires a response.

SURROUND

This is a real-time video game in which two players compete by attempting to avoid running into anything on the screen: the border, their opponent's trail, or even their own trail. The player who starts in the left-hand side of the screen uses A, W, S, and Z; the player who starts on the right uses L, P ; (semicolon), and . (period) to turn his piece in the direction shown:

```
    ^                   ^
    W                   P
( A   S )           ( L   ; )
    Z                   .
    v                   v
```

It is not necessary to press the key hard (this may be hard to remember in the heat of the battle), but you must lead your turns slightly. A quick stab at a key is not very effective; there is no effect by holding a key after the turn is made, so the most effective strategy is to press and hold the key until you see that it has taken effect.

At some time ittwill become apparent that the two players arrive at the same point at exactly the same time and not cause a collision. They may cross trails or even run as one for a time after such an encounter, which adds some interesting twists to the strategy of the game.

BACKGAMMON

This is the standard game of backgammon designed as an automated board and dice-roller for two players. The inner table is on the left side of the screen, the bar is in the center, and pieces removed from the inner table at completion are not visible.

When the dice are rolled they are displayed below the table, with the player to move indicated at the right. To move a piece, point to it by pressing the key marked on that point, and then the key on the destination point. The program will ignore illegal moves. You may return a piece to the point from which it came with no penalty until it is placed on another valid point, at which time the move is permanent. More than 5 pieces may be placed on a point and are accounted for but will be stacked up so that the screen will never appear to have more than 5 pieces on any one point. This program uses the tone generator for additional feedback to indicate when a move or capture has been accepted.

FIFTEEN

In this game, you and the computer alternate picking without replacement from the set of numbers 1 thru 9. When either player has collected any three numbers which add to fifteen, he wins.

REVERSI

This is a board game in which one person plays against the computer. Instructions are given at the beginning of the game. The tone generator is again used for sound effects and to warn against attempting illegal moves.

$LIFE

This is a simulation of the birth and death of organisms living in a square grid simulated on the screen. Each cell has eight total neighbors, counting the four which share its edges and the four which share corners. If a cell is touching only one or no living neighbors, it dies in the next generation due to isolation. A cell which has four or more living neighbors dies due to overcrowding. Exactly three live cells touching a given cell will produce a living cell in the next generation.

To run this simulation, first use LOAD $LIFE to load the machine language file and link it to BASIC. Then use ?USR(-7) to jump to the program. To set up your initial pattern, use A, S, W, and Z as in SURROUNDDto move the cursor around the screen (it is invisible). To place a live cell, press M; to kill a cell, press the space bar. You may see where the cursor is by pressing M and then the space bar at any time. To start the simulation, press G (go) with either (shift lock) or the left shift key depressed. The simulation will run continuously while the shift lock is locked depressed, or may be stepped through single generations by releasing the shift lock and pressing a shift key for each step.

BSR CONTROLLER

This program allows control of lights and appliances using the BSR home control system marketed by Sears and Radio Shack. Load the program and run it, then load your program which uses these controls. After having run this program, until loading another machine-language program or re-booting the system, USR(-7) will access the BSR system.

For the hardware part of the interface, simply set up the tone generator as suggested in the HEXDOS manual, but substitute a 40KHz (approx) ultrasonic transducer for the speaker. (A suitable transducer is part #MM 1002 available for $6 from The MicroMint Inc, 917 Midway, Woodmere NY 11598.) Place this transducer very close to the front of the control console, just below the LED.

After setting up both the hardware and software, you may enter commands from the keyboard as T=USR(-7)X where X is a number from 1 to 22 which selects a key on the console to press. Numbers 1 thru 16 are the numbered keys, and 17 thru 22 are the control keys:

17 - ALL OFF.
18 - ALL LIGHTS ON
19 - ON
20 - OFF
21 - DIM
22 - BRIGHT

Note that the keys are only held for a short time with each call; thus the dim or bright functions require multiple calls. To activate ALL OFF, as an example, you would type T=USR(-7) 17 This will work either directly from the keyboard or when executed in a program. You may also use any valid arithemtic expression to designate the key, as long as it reults in a value from 1 to 22 (out-of-range values are ignored). Thus, if A is 3 and B is 9, T=USR(-7) A^3-4*(B-3) is a valid way to activate key number 3. Simply have your program or your inputs on the computer keyboard set up to generate the same console actions as you would require to do the job manually.