# HEXDOS NEWS

Ah, the long-promised newsletter at last. I hope to make this the first in a series of quarterly newsletters to let all of you know what is happening with HEXDOS, as well as providing a way for you to share ideas, application programs, etc. among yourselves. If you live in an area without an OSI users' group (like the booming metropolis of Enid, OK), it seems like no one has ever even heard of Ohio Scientific, much less has anything that is compatible with it. I hope to at least provide a medium for exchanging ideas and programs. I'll continue to publish it for the remaining 3 quarters of this year for $1 per issue, and then see if enough people want me to continue it to be worthwhile. If you want to receive the next 3 issues, send $3 by May 15. Also, my interests and yours will be different; if you have a neat application, notes on how to get around some restriction of HEXDOS, etc. I'll be happy to include it in future editions of this newsletter. Let me know what you want and I'll try to oblige. If nothing else, I enjoy talking to other OSI users.

Without getting into a formal table of contents (since it will change by the time I finish writing this), here are some of the things in this newsletter. I have a new program package for use with HEXDOS, containing an editor, assembler, and other programs, described later. Several of you have asked about merging programs and converting programs from OS65D for use with HEXDOS; I'll show how to do that. Much has been written about the "garbage collector" problem and I'll add my two cents. Some notes on the character generator and a "shape table" program are included.

## NEW PROGRAM PACKAGE

This package will be available from the 6502 Program Exchange in the very near future. All the programs are written and working; we are just working out some final details on compatibility with other assemblers. Exact price will be announced soon.

Several users have been crying for an assembler to go with HEXDOS so they could give OS65D a decent burial once and for all. I was the first - I keep getting lost switching disks around, since the source code for HEXDOS is on a 65D disk and I have to assemble it with their assembler, then juggle things around in memory to put the object code where it belongs, switch to a HEXDOS disk, use the monitor to bring up the new code, and then use it to save itself on the disk. And if you aren't confused by now, you're doing better than I am. In any case, I finally got around to writing an assembler (in BASIC) to run under HEXDOS. Yes, it's slow since it's written in BASIC, but not as slow as I expected. However, offsetting that is the fact that the editor is lightening fast. No more going to get a sandwich after typing a line in the middle of a long program, as with OS65D's assembler/editor. The basic idea is to treat the source file just like any BASIC program while editing, and then assemble it from disk. There are some interesting spin-offs to this set-up, too.

## EDITOR

How would you like to have an editor that lets you edit full ASCII files just like you edit a BASIC program, lets you access all BASIC's commands in the immediate mode while you're using it, and fits in 156 bytes of RAM? Well, what I did was to write a BASIC program that copies the core of the editor from ROM to high RAM and makes a few patches. It disables the leading space deletion and the reserved word compression, forcing the editor to store every line just as you typed it. This way, without having to learn a whole new set of editing commands, you can edit a file using all the facilities of BASIC and HEXDOS, save and load the file, and be able to do things like PEEKs or POKEs and arithmetic calculations while you are editing a file. I am using that editor to write this newsletter, as a matter of fact.

## STRIPPER

No, this newsletter is not going to be X-rated. What I'm talking about are line numbers. With such a powerful editor available, I could very neatly write letters with it, but if I just LIST my letters to the printer, I get the line numbers with them. Of course, most people would probably read right on through them, but the numbers are a little distracting and make what could be a very neat letter rather a mess. No doubt you have already taken the leap from how I wrote the editor to how I wrote the routine to list a file without line numbers - same idea: Just copy the LIST routine to RAM, disable the part that prints the line numbers, and call it through USR, leaving LIST available for normal use. As a little bonus, I replaced the number-printer with an optional tab routine to adjust the left margin on printers that don't do that for you.

## ASSEMBLER

Finally, the real meat of the package. This assembler includes the features they should have put in the 65D assembler, and works with an operating system that's fit to use.

One of the features I missed most on OSI's assembler was the ability to assemble a program longer than would fit in RAM with the assembler all at one time. HEXDOS in particular is too big to assemble with that assembler without pulling some fast ones like writing the object code over the source code after the assembler is finished with a portion of it. There really should be no need for that sort of shenanigans with an 80K disk just sitting there spinning all the while. Consequently, my assembler reads the source code from disk during assembly.

Another feature lacking in OSI's assembler which I included in this one is the ability to get a symbol table at the end of the assembly listing, giving the addresses of labels, values of constants, etc. If you want to assemble a REALLY big file, you can chain from one file to the next, allowing the source code to be as big as you can squeeze onto a single disk. And for those of you accustomed to big-system features, it can even handle macros, including up to ten parameters.

## $RENUMBER

Since there don't seem to be very many renumbering routines floating around for this machine, there's one in this package. As you can see by the $ in the name, it is in machine language. This makes it fast but creates the problem of where to put it in RAM, since it's hard to make anything useful for this processor position-independent. I got around that by providing four different copies of it, assembled to fit at the end of 12K, 16K, 24K, and 32K.

## COPIERS

A number of you have asked about copying disks to make backup copies. One way is of course to use FORMAT to set up a disk and then copy program files one at a time, but data files were a little cumbersome. I have one routine to make an exact copy of a disk on the second drive of a dual-drive system, and another program to make a copy by filling RAM with several tracks, then prompting you to switch disks to write it, and so on back and forth for as many swaps as it takes to copy as much of the disk as you ask for.

## DISK TESTER

This routine will write several different patterns on every track of a disk and insure that they read back correctly. Presumably, if it gets back each pattern correctly, the disk and the drive are OK. This one is dangerous to leave lying around, since it erases the disk, though I did build in plenty of queries to make sure you really intend to use it.

## MERGING PROGRAMS

Is that anything like merging traffic? Well, sort of. If you want to combine two programs into one, or have a set of subroutines you'd like to include in several programs without retyping them in every one, here's how to do it. It works very well under HEXDOS, though it can also be done under ROM BASIC. First, the two programs to be merged should probably not have overlapping line numbers unless you're intentionally playing silly games and know just what you're about. Unless you really understand BASIC's innards you will get some strange effects if you later try to insert or delete lines in a file where the line numbers are not strictly increasing, though it will run just fine until you try to GOTO or GOSUB an out-of-sequence number.

After creating both (or all) of the programs you want to merge and saving them on disk, load the one which comes first in the merged file as if it were the only one you wish to load. Now comes the tricky part. BASIC keeps a pointer to the start of the program at PEEK(121) (low order part) and PEEK(122) (high order part), and a pointer to the end of the program at 123 and 124. Now we fool BASIC into loading one program on top of another: PEEK the values at 123 and 124 (do not use any variables for them). POKE the value that was at 123, minus 2, into 121. If this value is negative, add 256 before POKEing it. Similarly, POKE the value that was at 124 into 122, first subtracting 1 if you added 256 to the other value. Now, load the next program. Do NOT attempt to list it yet! There are pointers within it that are messed up right now because it is not loaded where it normally is. If you have a later version of HEXDOS, it will patch the pointers automatically when it loads the program, but older ones will not.

Now, if you want to merge some more programs, continue PEEKing and POKEing as for the second program. When you finish loading all the programs, POKE 121,1 and POKE 122,11, resetting the start-of-program pointer to the start of the first program. If your version of HEXDOS does not automatically patch the line pointers (you get garbage when you try to list more than the first program of the merged file), just do any editing to the first program. The easiest way is to delete a non-existent line. If your program has no line 0, type just a 0 <carriage return>. BASIC normally patches up the pointers after you insert or delete a line, and this fools it into cleaning up the mess you just made. Now you should be able to list, save, or run the combined file just as if you had typed it all in as one.

## CONVERTING PROGRAMS FOR HEXDOS

The HEXDOS manual gives some pointers on converting programs, but here is some additional information. If you can get the BASIC program into memory anywhere that does not conflict with HEXDOS, the conversion is simple, though you will need to know a little about the internal storage of the program under whatever system you wrote it under (from what I've heard, most of you will - HEXDOS users seem to be a bunch of real hackers who can get right down into the nuts and bolts if need be). If you can get the start-of-program and end-of-program pointers to point to the program as above, just SAVE it under HEXDOS as you would any other program. Replace the normal start-of-memory pointer or reboot HEXDOS and then LOAD the program. As I said above, newer versions of HEXDOS (2.4 or 3.0) should patch up the pointers automatically. With older versions, you can force it by editing the file (deleting a non-existent line as above). If you need to patch the pointers via brute force, use the monitor to jump to $A31C. This will come back to the normal BASIC prompt. Use the hardware reset (BREAK key) to get back to the monitor, and check the values at $71 and $72. t back to the monitor, and check the values at $71 and $72. Put that 2-byte value plus 2 into $7B and $7C, and then warm start. Now regardless of what operating system or lack thereof you are using the pointers should be OK.

Of course, there will be some minor changes to the program, especially disk accesses, in changing from another operating system to HEXDOS. In general, remember that HEXDOS uses LOAD where other systems use OPEN and SAVE where others use CLOSE. You'll have to know what the program is trying to do with the disk to make a complete conversion. Some functions that may have been done in BASIC are provided in machine language in HEXDOS, like keyboard functions, screen clear, etc.

## ROMS

The next couple of topics involve changes to ROMs on the system. Since I've added a ROM programmer, I've felt a new freedom to make the system work the way I want it to. If any of you have changes you want to make to your ROMs, I can program 2716s for you. Just send me the code you want in them on one track of a HEXDOS disk, and I'll burn a ROM from it and send it back. $10 will cover my costs for the ROM and $1 will cover the return postage. Fair warning - most ROM changes will require changing the chip-enable line to the socket involved. A 2716 needs an active-low chip-enable, while most of the standard ROMs used by OSI need it active-high. You will have to put an unused inverter somewhere on the board into the line. If you aren't sure you can handle this, don't buy ROMs from me or anyone else.

## GARBAGE COLLECTOR

When you run a program which does a lot of manipulations on string arrays, it will seemingly at random hang up, just blinking the screen at you and not responding to ctrl-C. Sometimes even a warm start isn't enough to rescue your program, as it gets garbled. Here's where the problem originates.

Since strings can be any length from 0 the 255 characters, BASIC cannot allot them a fixed space in memory to exist. When you assign a value to a string variable, the string is stored at the high end of available RAM and the variable is pointed to it. If you later assign a new value, BASIC discards the old string and allots a new space for it. The old string remains in memory for the time being. Since BASIC is being wasteful with the memory, it eventually runs out as more and more space is discarded from high memory and the strings work their way down until they conflict with other variables. At this point, BASIC calls the now infamous "garbage collector" which runs around and figures out what memory has been discarded, and then rearranges the strings so that they use all the memory starting at the top of memory down just as far as is necessary. BASIC goes back on its merry way until it runs out of space again, and calls the garbage collector again. This can go on indefinitely until memory gets so full that the garbage collector can't find find enough space to compress the strings into, and you get an OM (out of memory) error.

This process works just fine, except for one minor problem. It seems that when they wrote the garbage collector routine, one hand didn't know what the other was doing (which doesn't seem too uncommon for OSI!). Each string descriptor requires 3 bytes for storage, but since normal variable take 4, string pointers are stored in 4 bytes, with the last one set to 0. However, they forgot to tell the garbage-collector-writer that, because he assumed they would only take up 3 bytes, and the routine gets lost as it steps from one pointer to the next in string arrays. This is why the fix of dimensioning string arrays to have a multiple of 3 elements sometimes works (the actual dimension must be 3N-1 since there is a 0-th element). In any case, PEEK(65) published a fix some months ago which seems to solve the problem completely except for triple concatenations (A$+B$ +C$), and I haven't heard of any fix that handles that. I have programmed the fix into a 2716 and replaced the BASIC 3 ROM with it and it has worked perfectly ever since. If anyone wants a copy, I'll burn you a ROM and mail it as I noted above.

Lest you get the idea that all the troubles with the garbage collector are over, let me point out the it still has a conflict with HEXDOS. HEXDOS data file buffers are in the same space as strings and HEXDOS moves the free memory pointer accordingly, but the garbage collector of course isn't smart enough to go check the disk file headers when it is deciding what memory is not in use, so it tends to clobber disk file buffers. The solution is still as printed in the HEXDOS manual: either do all your disk file access, then all your string manipulations, and then reopen the disk files for final processing, or else open the files first and move the end-of-memory pointer ahead of them so that the garbage collector never gets to them. To do this, PEEK the values at 129 and 130 (decimal) and POKE them to 133 and 134, respectively.

## CHARACTER GENERATOR

This one is not really related to HEXDOS, but it fits in here anyway. I've been tinkering with a changed character set to include some more useful graphics shapes like electronics symbols, chess pieces, etc., and reorganized the characters which fit together to make bargraphs, lines, etc. I also added the missing pieces for TRS-80 style graphics - the original character set included characters for half-size blocks, and included every combination except for 3 of the 4 within a character cell being on at once. I've placed the complete set at CHR$(0) thru CHR$(15) so that bit 0 of the character turns the upper left block on, bit 1 is the upper right, bit 2 is the lower left, and bit 3 is the lower right.

The most interesting thing about the character generator change is in the change in the chip-enabling logic. The change for the BASIC 3 ROM involves cutting some traces and inserting an inverter into one line, but this one is much simpler. The original character generator is a 2316, with both chip selects (pins 18 and 20) being active-high. A 2716, on the other hand, needs both of those pins to be low. I simply cut the ties to +5 on those pins, tied them high with a 2.2K resistor, and added a switch to short them to ground. To add the new character generator, I just bent its pins inward enough to grip the pins of the original one and stuck it on piggy-back fashion - now the switch lets me select either character set, and without even the bother of finding a spot to mount the alternate ROM!

## PROGRAM OF THE QUARTER
## SHAPE TABLES

The problem with doing animation on this system is speed. Even though the BASIC is fast, it's not really fast enough for smooth effects of any complexity. Listed below is a machine language routine which you can type in with the monitor, and which becomes a part of your program as a comment in the first line. This is a useful technique for any short machine language accessory to a BASIC program, whether you are using straight ROM BASIC or using HEXDOS. With ROM BASIC, change the addresses in the $0Bxx page to $03xx. The only restriction is that there can be no 0 bytes in it. If you must load an immediate value of 0, load X or Y with 1 and then decrement. This takes an extra byte, but how many extra bytes would your program need to load the machine language by conventional means?

When loading a program this way, start the machine code at $0B06. At the end, place 3 consecutive zeroes. Then go back and place the address of the second of those zeroes in $0B01 and $0B02, in the usual low-high form. Put 0 into $0B03 and $0B04 (the line number) and put $8E (REM) into $0B05. Last but not least, put the address of the byte after the final triple-0 into $007B and $007C (end-of-program pointer). Now do a warm start and LIST the program. What's all THAT garbage!?$*"! That's what BASIC makes of your machine language, but it ignores it as part of the REMark. Unless you slipped a 0 into it, it will still be intact. Now add line 10 POKE240,6:POKE 241,11 to link USR(-7) to the machine language.

Here's the data in hex for a routine which will place a multi-piece graphic figure anywhere on the screen:

```
0B00  00 6E 0B 00 00 8E 20 AE B3 E0 20 90 03 4C 88 AE
0B10  A0 01 88 84 FE 86 FF 46 FF 66 FE 46 FF 66 FE 46
0B20  FF 66 FE 20 02 B4 8A 18 65 FE 85 FE A5 FF 69 D0
0B30  85 FF 20 01 AC 20 C1 AA A0 FF C8 B1 AE AA 96 AC
0B40  C0 02 D0 F6 88 88 84 5F B1 AD 91 FE E6 AD D0 02
0B50  E6 AE C6 AC D0 01 60 B1 AD 18 65 FE 85 FE 90 02
0B60  E6 FF E6 AD D0 02 E6 AE C6 AC D0 DC 60 00 00 00
007B  70 0B
```

USR(-7) is now the shape table function. The format for using it looks like: T=USR(-7)X,Y,A$. X is the row number where the first character should appear, counting from 0 at the top of the screen (not visible on most monitors) to 31 at the bottom (ditto). Y is the column number, from 0 at the far left to 31 at the far right. A$ is the description of the figure. The first character appears as the leftmost character in the uppermost line of the figure. The next byte specifies the number of bytes to the next non-blank character of the figure, followed by the character, then another number of bytes and a character, etc. The following program will build a descriptor for a box.

```
20 DATA 210             upper left corner
30 DATA 1,207           space over 1 and place the upper right
40 DATA 31,209          drop down and left for the lower left
50 DATA 1,208           1 to the right for the lower right
60 FOR I=0 TO 6
70 READ T
80 A$=A$+CHR$(T)
90 NEXT
```

After adding this to the program above and running it, you should be able to put a box anywhere on the screen with the USR call above. The advantage to this method is that you can completely describe a shape in one string, and place that shape anywhere on the screen almost instantly with the USR call. Remember that a string may be only 255 bytes long, so the most complex shape you can describe this way may not have more than 128 non-blank characters. If you need a bigger shape, you can of course split it into two or more sections. If you need a gap of more that 255 bytes between characters (8 vertical lines), just place a blank (decimal 32) somewhere between them.

## HEXDOS REVISION

Version 4.0 of HEXDOS is now available. If you wish to update yours, just send a check for $11 ($10 plus $1 postage) and I'll send you a new disk with version 4.0 and corrected utility programs. I hate to ask that much for an update, but I don't break even if I do it for less than that. Rather than you mailing back your disk, I'll provide a new disk - it seems more sensible to spend a little on a new disk than extra postage to mail back the old one.

The biggest reason for the update is to make HEXDOS work with the standard monitor ROM and the C1S and C1E without needing separate versions. Another major reason for the change is to make filenames be strings - you can now INPUT a string and then open a data file named in the string. The error messages are slightly different - similar to BASIC's error messages and easier to remember. So many people have asked for an ON ERROR GOTO type of function that I added that also, with provisions for your error-handling routine to detect the type of error and the line number in which it occurred. The time-delay routine of the real-time clock can now be written in BASIC rather than in machine language.

The new version provides for a program to run automatically on boot-up. The program must be short, since it resides in leftover space on track 0, but it may be long enough to provide a short menu and RUN a regular program. Not only can you now have the capabilities of OSI's BEXEC* program, but you can have it with 10K more memory and without taking up space on disk that you could use for other programs.

There are a few changes to the page zero memory map with this version, mostly additions for the new features. The changed locations would not normally be accessed directly from any BASIC program. The only change which will likely require changes to programs is the fact that filenames are now strings. This requires placing the filename in quotes if you type it directly in the line, but you can also place a string variable or expression in its place. I will include a short supplement to the manual explaining the changes. If you don't already have a manual for version 2.4 or later, there may be some changes which will not be explained in your current manual. The current manual is professionally printed and costs me $10, so include that if you need a copy.

## PROBLEMS WITH USR

When it happened, I don't know, but I somehow managed to clobber part of the code for the USR function, and at least one copy sneaked out before I found it. The correct code appears below. Addresses vary with different versions, so to find the exact location of the USR code on your version, use the monitor to look at $000B and $000C. That is the low byte - high byte (in that order) of the label USR. If you want to try to see how this code works, more power to you. It's one example of the tangle it took to cram all this into 2K. This is a slightly tighter version than the original, so you may have some space left over. Anything between the second JMP $AFD0 and the label TONE should be changed to NOPs ($EA).

```
2012B4 USR  JSR $B412
AA          TAX
D009        BNE *+11
84E2        STY IOPTR
20EBFF      JSR $FFEB
A8          TAY
4CD0AF      JMP $AFD0
101F        BPL TONE ;you can change this to skip the
                     ;NOPs if you want
98          TYA
C9FB        CMP #$FB
D008        BNE *+10
20A506      JSR GETVAL ;the address is approximate
85AD        STA $AD
6CAD00      JMP ($AD)
B00A        BCS *+12
C9F9        CMP #$F9
F003        BEQ *+5
4C00FE      JMP $FE00
6CF000      JMP ($00F0)
AA          TAX
B4EC        LDY CLOCK+4,X
4CD0AF      JMP $AFD0
   ;insert as many NOPs as necessary here
A958  TONE  LDA #$58
85DB        STA TEMP3
A900        LDA #0
    ;and so forth
```

The label GETVAL is noted as an approximate location. If you can't find the correct address in your copy of HEXDOS, look near the given location for code that looks like:

```
20C1AA GETVAL JSR $AAC1
2005AE        JSR $AE05
A5AF          LDA $AF
60            RTS
```

As always, if there is something wrong I will fix it, but this patch might save you some postage and time waiting for the US Snail. Of course, if your version of USR works, better just leave well enough alone!

# HEXDOS NEWS

Volume 1 Number 2

Second Quarter 1982

The response to the first newsletter has kept me hopping to answer questions, mail updates, etc. However, I haven't seen exactly a flood of articles for publication here (not even a trickle - not a one). I have to assume then, that I'm filling the entire need for information out there (and you know what happens when you assume!). All the comments and suggestions are a big help, but I think this would be a much better newsletter if it weren't a one-man show. How about it?

By the way, from what I've been hearing, a lot of you are producing some pretty neat applications software to run under HEXDOS. The 6502 Program Exchange, 2920 W. Moana, Reno NV 89509 is interested in marketing software to go with HEXDOS. If you have something the rest of us could use, this is a good way of sharing your work with other users. Everybody wins - they don't have to reinvent the wheel, and you get a little compensation for your efforts.

One big news item is an address change. You may want to note this in the HEXDOS manual where my address currently appears. By the time you get this, my address will be:

Steven P. Hendrix
Route 1 Box 81E
New Braunfels, TX 78130

When you write to me, it would be a big help if you include the serial number of your copy of HEXDOS. My mailing list is indexed that way, and it's a lot quicker printing an address label. I'm lazy about typing them out myself and error-prone.

## HEXASM

As I promised in issue #1, the assembler for HEXDOS is finally ready. The price is $38.50 for a disk with the assembler and the support programs I described last quarter. This package runs only with version 4.0. The DOS itself is included on track zero of course, but you will run into some incompatibilites with your other software and the support programs for HEXDOS if you try to mix with older versions. If you need an update, the price is as before - $10 for a new disk with HEXDOS and all the programs which go with it, and $10 for the newer manual if you still have the mini-printer copy. Sorry about the charge but I lose money at this game if I do it for less.

## ENTRY POINTS FOR DISK ROUTINES

A number of you have asked about tying other language processors, etc. to HEXDOS. The main routines you will need are LDTRK and SVTRK, "load a track to memory" and "save memory to a track", respectively. Version 4.0 is now tied down firmly enough that I feel safe in giving out locations without risk of changing them in later copies of what is supposedly the same version. LDTRK starts at $05B0 (1456) and SVTRK starts at $0754 (1876). These address track number A and the 2K bytes of memory starting at X:Y, with the high byte of the address in X and the low byte in Y. A seek error will print the normal message and do a JSR $0003 before returning to BASICs immediate mode, so you can intercept it by changing the JMP at $0003.

## HIGH MEMORY ADDRESSES

Apparently some people are adding RAM above OSI's limit of 32K and doing some other strange things in the high half of the address space. I've heard from several users who were having difficulty doing disk track LOADs and SAVEs up there. If HEXDOS won't accept a number above 32768 in any given context, just subtract 65536 from the number. The difficulty is that some of the ROM routines read bit 15 of a two-byte number as a sign bit, while others assume it's an unsigned number from 0 to 65535.

## RANDOM ACCESS FILES

I've had some questions about RANFILE which indicate that maybe I didn't explain random access well enough. Here's an attempt at a different description.

Think of a normal sequential disk file as a cassette tape. Opening the file (LOAD *<file number>,<name>) loads the tape and rewinds it. Every time your program does an INPUT#<file number>, the tape gets advanced by enough characters to satisfy the INPUT statement, which keeps reading until it sees a carriage return. This keeps everything simple, but finding something which is a long way down the tape is a nuisance, even at the speed a floppy goes.

A random access file works the same way, except now your tape player has a character counter. The RANFILE subroutine lets you do an instant "zap" to any given character by knowing what number the character appears at, without having to wind the tape past all the intervening characters. In practice, the letters are group into records, and the tape counter counts records instead of characters. RANFILE is set up for 64 characters per record, but this may easily be changed. Calling RANFILE for record number X "zaps" the tape to character number 64 * X, and the next INPUT or PRINT statement to that file will start there.

It's important to understand that INPUT and PRINT don't know anything about random files, and RANFILE can't affect what happens if they get into trouble. If you PRINT past the end of a record, for instance, the effect is just like it would be on the video screen - the extra gets typed on the next line, or in this case, the next record. Also, if there is no carriage return in a record, INPUT keeps blindly eating characters until it finds one, even if that means running into the end of the file.

For all their versatility, random access files do require a little extra care in your programming. Always make sure your program PRINTs a carriage return on the end of each record. Your program must trap any attempt to print past the end of a record, if that will cause problems. The POS function will tell you how many characters you have printed since the last carriage return, and you can use that number unless you are doing IO with other devices intermixed with your IO on the random file.

## ROM VERSION OF HEXDOS

A few people have asked for ROM versions of HEXDOS. At one time, I had planned to go that route altogether, but updates on disk are so much more convenient than ROMs. You would have to start it by entering the monitor and executing the boot routine, and you lose the autostart program capability altogether. It would also create problems because track 0 would then be difficult to use, being "trapped" below the directory. Moving the directory itself to track zero would make an incompatible disk format. And if you want to do that to save memory, you can gain more by adding one of the new 6116 CMOS RAMs instead of a ROM, opening more memory for general-purpose use. All in all, I don't think a ROM version is going to be practical, but if anyone insists on it, we can work something out.

## LINE EDITOR BUG

It seems a few of the first copies of version 4.0 to go out had a glitch which crippled the forward-space function (esc) under the standard monitor ROM. If the esc key types an extra character over every other character as you try to forward-space the following program should fix it:

```
10 P=PEEK(536)+256*PEEK(537)
20 Q=PEEK(538)+256*PEEK(539)
30 FOR I=P TO Q
40 IF PEEK(I)<>201 OR PEEK(I+1)<>32 THEN NEXT:PRINT"BAD"
50 POKE I+1,27
```

If this program prints "BAD" then it did not find the problem. This will happen if the forward space already works. Otherwise, let me know if you still have a problem.

## PARALLEL PRINTER PORT

Some of you have asked about implementing the parallel printer port at $D900, named as device #1 in HEXDOS. Here is a schematic for approximately the way I've done it. You may be able to take some shortcuts, because the address decoders I use are very general and in fact I use them for several other things on the same board. And, to forestall a bunch of letters - no, I'm not in the business of producing hardware, at least for the present. If anyone out there wants to make up a board to sell, let me know and I'll put your name and address in the next newsletter.

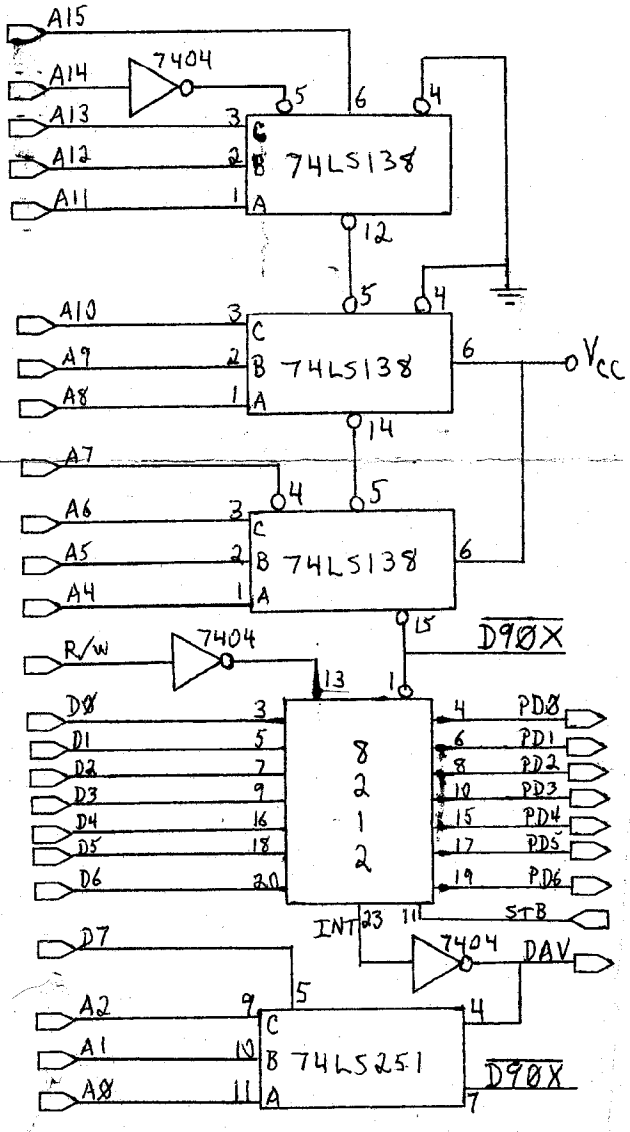


Figure 1. Printer Port

The three 74LS138s function as address decoders, pulling the selected output line low when CS0 is high and CS1 and CS2 are low. One of eight possible output lines is selected by the binary number presented at C, B, and A. When the processor does a store to $D900, the 8212 is selected, data is latched from the data bus onto the output lines, and INT goes high, making DAV low and signalling the printer that data is available and valid (other devices could be connected to this port - I'm assuming a printer). After the printer accepts the character, it drops BUSY to a low level, triggering STB and causing the 8212 to drop INT and thus raise DAV. The processor tests to see if the port is ready for another character by reading from $D900, causing the 74LS251 to place the value of DAV on the data bus as bit 7. Note that this happens on the write to the port as well, so bit 7 of the output port is unusable.

The driver routine in HEXDOS simply tests bit 7 of $D900 (the DAV line) and waits for it to go high (signalling that the printer is ready for a character). It then stores the character at $D900, placing it on the output lines and signalling (through DAV) that data is ready.

## About my telephone ...

I deliberately don't publish my phone number with my address. A lot of you must have called information for it, because I get a lot of calls. I don't mind them and actually I enjoy talking to fellow users, but please don't call past 9:00 PM Central time. Also, I can usually give you a more coherent and complete answer by mail, and your phone bill doesn't get so high. The basic idea is, I don't mind talking if you don't mind paying the phone bill, but don't interrupt my sleep.

# HEXDOS NEWS

Better late than never - here's edition number three of the HEXDOS newsletter. This issue contains the first (and so far, only) article sent in by a user. Jim Ford has some ideas and programs for use under HEXDOS. If anyone else has anything to publish in the newsletter, you'd better get it to me soon. Based on the response so far, I plan to follow through on my promise of four quarterly issues and then let it rest. HEXDOS continues to be a viable product, of course, and its user base is growing if slowly. I prefer to get back to the business of writing software and get out of the newsletter editing business. I may still put out notices from time to time if I see a need to get some information out, but I'm not promising any more regular newsletters.

Also in this issue are fixes for a couple of minor bugs in HEXDOS and some notes on HEXASM and HEXDOS. I will reproduce one of my articles which has been published in PEEK(65), for those of you who do not subscribe to that, and have several other miscellanous short items.

## ----------)))))))) ADDRESS CHANGE ((((((((----------

If any of you have tried to write to me at my new address since about July, you probably haven't had much luck. It seems the postperson learned to count higher than 1, and they renumbered the routes here. Unfortunately, that happened about a year ago and the former owner neglected to tell me the correct address to use. Sorry if that caused any inconvenience for anyone. If you've sent me letters which haven't been answered, give me another chance at this address:

Hx Computer Products
Route 8 Box 81E
New Braunfels, TX 78130

By the way, be sure to put the "E" on the box number -- there are only about 20 box 81's out here! (Texans seem to have some strange customs all around with addresses!)

## Quick Reference Card

I've had a HEXDOS quick reference card printed. For those who bought HEXDOS before I had that, I'll send you one on request. Just send a self-addressed stamped envelope. The card is 5-1/2 x 8-1/2 inches, so the size envelope you send will determine how much it gets folded. It shows the added commands available under HEXDOS and the exact syntax for each, keyboard functions, device numbers, error messages, and an abbreviated memory map.

## HEXDOS Bugs

For anyone who missed the fix for the ESC key, a few earl copies of version 4.0 have a bug in the forward-space routin corresponding to that key. The byte at $00CD is $20 and shou be changed to $1B. This will prevent the double echo of the ES key.

Another bug has recently turned up in the SAVE routine. If you have difficulty SAVEing a program to another file (in th form SAVE &(filename)), check $07E1 for a $36. This byte shou be changed to $32. If you try to save to a non-existent file you may get an "ID" error (illegal direct). Of course, you cou Just think of this as an "IDentification" error, but to correc it to the proper error message, change the two bytes at $07D from $A9 $42 to $A2 $EA.

If you know of any other errors in HEXDOS which I haven't yet addressed, let me know as soon as you can so I can devise patch and publish it in the final newsletter. Hopefully, ther aren't any left after 3 years of development and testing

## HEXASM

If you don't have the garbage collector patch (BASIC 3 ROM installed, HEXASM will run afoul of the old one when you try to assemble an assembly language program of any great size. Even if you have the patch, it can get into trouble with the fil buffers. It seems that every time a label is inserted into the symbol table, moving the entries below it to make room involve discarding the memory allocated to them and taking up some more memory. If your program has very many labels, this eats up al the memory in a big hurry and trips the garbage collector. If your copy seems to have this problem, send it back for a revise version which does not do so much string shuffling.

Another minor problem with HEXASM is that it cannot use memory above 32K. If you have added that much memory, you wil have to limit BASIC to 32K on boot-up (answer 32768 to the "Memory Size" prompt).

The revised version of HEXASM is capable of assemblin very large programs, though it's not exactly a speed demon (what do you expect from a program written in BASIC, anyway?). The source code for HEXFORTH (more on this later) is now about 50 lons and assembly time is approaching an hour. On the other hand, the OSI assembler isn't much faster and couldn't handle that large a program to begin with. It won't even give you symbol table so you can assemble such a thing in pieces an link sections together by hand!

## HEXFORTH

I'm making steady progress on Forth for this machine. At this time, I have the inner interpreter fully working, and all the words necessary for compiling new words working. Now I'm defining the rest of the words which are required under the full Forth-79 language definition. I hope to have it finished within the next couple of months.

Some of the outstanding features of this implementation of Forth will include: Full floating point arithmetic, with a complete set of functions such as trig and log functions; disk functions permitting access to the normal Forth screen arrangement, HEXDOS files, or OS65D disks; later, I may also add yet another option for disk format, which will be truly soft-sectored, ignoring the index hole altogether and making it easier to use both sides of a disk, and allowing slightly more storage on a disk. It appears that the full implementation will fit in about 6K of RAM, making the minimum system about a 10K system. I hope to have it ready to announce in the next newsletter.

## LIST "Bug"

Several of you have written that LISTing to a printer looses the last line or that it isn't printed until you print something else to the printer. This is not a bug of HEXDOS or your hardware; just a quirk of BASIC. On any LIST, including one to the screen, BASIC does not print a carraige-return line-feed pair at the end of the last line of the program (it does do a return after the last line of a listing if that is not the last line of the program, however). You may wish to experiment with this on the screen by entering about a three-line program and trying all the forms of LIST with various combinations of a starting line number and an ending line number. The reason the OK is not printed on the same line as the last line of the program is that the prompt includes a CR-LF pair both before and after the letters "OK".

To resolve this problem, simply do a PRINT#1 or PRINT#2 after LISTing a program to your printer or tape. You cannot do it in the same line as the LIST command, however, because BASIC goes not finish that line. It returns to the immediate mode after executing any LIST or NEW command, which means any following commands will be lost.

## Formatting a Disk To Include the Auto-Directory

Because the automatic-directory program resides only on track 0 of the HEXDOS 4.0 disk, you may have had difficulty transferring this feature to formatted disks. Here is a very simple method. After formatting your new disk as usual, re-boot the master disk, holding ctrl-C, repeat, or break as appropriate to your system after responding to the "Terminal Width" query. This should result in a "BREAK IN xxxxx" message. You can now list the directory program if you wish. Replace your newly-formatted disk in drive A and type SAVE#0,768. Now your disk should give the automatic directory every time it is booted up.

## Memories Available

I'm branching out into the hardware arena. I now have available a 16K byte x 8 bits memory on a 28-pin dip. This memory is fully static, with no clocks or timing strobes. Access time is under 250ns and maximum power consumption is less than 200 ma. It has three device selects, one active high and two active-low, which allows full decoding of 48K (three memories) from a 16-bit address buss with no external parts. It is ideal for the 6502 buss. In fact, as I edit this newsletter it resides in one of these memories. I replaced some of my 2114's with it to cut down power consumption so my power supplies could handle the whole system without so much ripple, and moved one 8K block on the 610 board to the area at $8xxx and $9xxx, giving me access to 40K of RAM.

Current prices for the RAMs are $79 for single units. In quantities of 10, they are $69 and $59 in 100 quantities. If you want a free data sheet, send a self-addressed stamped envelope to the address shown on page 1. I'm not planning to offer any boards for this or any other system which will accomadate these memories, but the wiring is very straightforward.

## ON ERROR GOTO

Perhaps the explanation of the error-handler under v4.0 needs to be supplemented by an example. Suppose you wish to have your program print an error message similar to the normal error message, but then restart the program whenever an error occurs. In this example, we will start the program at line 10 and place the error-handler at line 2560.

```
10  POKE  237,10  ; Line #2560 / 256
.........
1000  END
2560  EL=PEEK(135)+256*PEEK(136)
2570  PRINT  "ERROR  #";USR(-1);"IN";EL
2580  GOTO  10
```

The following table gives the complete list of error numbers which can be returned by either HEXDOS or BASIC. (The list in the HEXDOS manual covers only HEXDOS errors.)

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | NF | 8 | FC | 18 | DD | 26 | LS | 38 | OR* |
| 2 | SN | 10 | OV | 20 | /0 | 28 | ST | 42 | IN* |
| 4 | RG | 12 | OM | 22 | ID | 30 | CN | 57 | EA* |
| 6 | OD | 14 | US | 23 | DT* | 32 | UF | 104 | PX* |
| 7 | DF* | 16 | BS | 24 | TM | 37 | RO* | 234 | Fd* |

* Indicates HEXDOS errors. All others are from BASIC.

# COPYING AND MERGING

James A. Ford
3712 W. 90th Street
Bloomington, MN 55431

Since I received HEXDOS 4.0 I've been very busy. I am going to report on two of the things I've been writing and working on with HEXDOS. The first is a program I call MERGE which does in a BASIC program all of the things Steve described in his first HEXDOS newletter. The listing for MERGE is given in listing #1.

Things to be remembered in using this program:

1) Programs should be merged in line number order - otherwise I'm not sure what will happen - lowest line numbered program first, etc.

[ Merging out-of-order programs will cause difficulties with editing lines, GOTOs, and GOSUBs - anything requiring a search for a line by number. Hx ]

2) After merging, the merged program can be SAVEd as you would any other program.

3) The beginning of BASIC pointers have been incremented further up in memory and can be restored to their normal values by re-booting the system.

This program requres some expanation since some tricks had to be used to make it work. I will explain it by the line numbers.

Line 53010 - Peeks the end of BASIC pointers after MERGE is loaded.
Line 53011 - Pokes the pointers into safe page 0 locations.
Line 53017 - Pokes N into a safe page 0 location.
Line 53020 - Asks for the first/next program filename - F$.
Line 53025 - Again peeks the end of BASIC pointers - A & B.
Line 53026-53030 - Subtracts two from end of BASIC pointer.
Line 53040 - Pokes A & B into beginning of BASIC pointers in preparation for loading next program.
Line 53050 - Loads next program and clobbers all variables at the same time - this is why they've all been poked into page 0 locations.
Line 53051 - Retreives N from page 0 and restores beginning of BASIC pointers to the start of MERGE - if you don't do this BASIC gets lost [ in seeking line 53017 from line 53052 ].
Line 53052 - Decrements N and tests for 0 to determine whether to load more programs.
Line 53054 - Retrieves original end of BASIC pointers from page 0.
Line 53055-53056 - Subtracts 2 from pointer.
Line 53060 - Pokes A0 & B0"into" beginning of BASIC pointers.

MERGE provides for HEXDOS the capability of OS65D's indirect file operations when used with what I call a strippine routine that I'll describe next. Steve Hendrix gave me this idea over the phone.

The routine given here will strip all lines greater than a line with two consecutive ctrl-G's after the line number. So type in a line number between the parts you want to strip with ctrl-G's.

Now for the stripper:

```
62410 FOR I = 2816 TO 40960
62420 IF PEEK(I) <> 7 OR PEEK(I+1) <> 7 THEN NEXT
62430 POKE I-4,0 : POKE I-3,0
   [ Strips everything after and including the line with
     the ctrl-G's ]
 or
62430 T=PEEK(I-4) : POKE 121,I-4 : T=PEEK(I-3) : POKE 122,I-
   [ Strips everything before and including the line with
     the ctrl-G's ]
```

You should re-boot the system after you save the remaining program to get things back to normal. This is real handy after you MERGE and run a renumbering program. You will find that it's a lot easier than OS65D's indirect file handling ritual. Caution: don't use a line number of 1799 in your program - it'll look like two ctrl-G's.

The other program is one that will copy programs from an OS65D disk to a HEXDOS disk. The listing is given in listing #2. The following procedure should be followed in conjunction with the 65D COPY program.

1) Boot up HEXDOS.
2) Create a file of sufficient track length.
3) Boot up OS65D.
4) Do a DISK!"LO <pgm>"
5) Peek locations 120, 121, 122 and 123 and record the values.
6) Boot up HEXDOS - type in your memory size when asked - otherwise your program will be clobbered by the memory test routine.
7) RUN"65D COPY"
8) Enter the PEEKed values when asked.
9) Re-boot HEXDOS to restore pointers.

For some reason SAVE"new file name" won't work but SAVEing to a previously named file does. (???) Maybe Steve will explain this.

Jim Ford

I SAVEing to a new file doesn't work because of the directory modification part of HEXDOS. HEXDOS can must load the directory into memory to modify it and uses the first 2K bytes of the program space for this after saving the program, and then reloads the first 2K bytes of the program. This is not a problem if the file already exists because HEXDOS need not change the directory - it can just read the directory "on the fly" as it goes past the disk head without loading it into memory. Hx 1

Listing # 1

```
53000 REM ***** MERGING PROGRAM *****
53010 A0=PEEK(123):B0=PEEK(124)
53011 POKE 245,A0:POKE 246,B0
53015 PRINT:PRINT"**** PROGRAM MERGER ****":PRINT
53016 INPUT"HOW MANY PROGRAMS";N:PRINT
53017 POKE 244,N
53020 PRINT:PRINT:INPUT"NEXT PROGRAM";F$
53025 A=PEEK(123):B=PEEK(124)
53026 A=A-2
53030 IF A<0 THEN A=A+256:B=B-1
53040 POKE 121,A:POKE 122,B
53050 LOAD F$
53051 N=PEEK(244):POKE 121,1:POKE 122,11
53052 N=N-1:IF N>0 THEN53017
53054 A0=PEEK(245):B0=PEEK(246)
53055 A0=A0-2
53056 IF A0<0 THEN A0=A0+256:B0=B0-1
53060 POKE 121,A0:POKE 122,B0
53080 END
```

Listing # 2

```
10 REM OS65D PROGRAM COPIER
20 INPUT"65D PEEK(120,121,122,123)";L,H,LO,HI
30 POKE 121,L:POKE 122,H
40 POKE 123,LO:POKE 124,HI
50 CLEAR:INPUT"SAVE IN FILE";F$:SAVE&F$:LIST
```

## BASIC's Scanner

This key routine is one which I'll call "GETBYTE". Its main function is to get the next byte of a BASIC program or immediate line, and it also does some of the work of determining what type of byte it is passing back to the BASIC interpreter. Nearly every other routine in BASIC calls this routine sooner or later. The fact that it is copied into RAM and called from there makes it vulnerable to some tinkering, permitting you to change the way almost any part of BASIC works. More on this later.

```
00BC GETBYTE  INC  ADDRLO
00BE          BNE  REGETBYTE
00C0          INC  ADDRHI
00C2 REGETBYTE LDA $xxxx  ; I refer to xxxx as ADDR
                          ; the low byte as ADDRLO,
                          ; and the high byte as ADDRHI.
00C5          CMP  #$3A
00C7          BCS  RETURN
00C9          CMP  #$20
00CB          BEQ  GETBYTE
00CD          SEC
00CE          SBC  #$30
00D0          SEC
00D1          SBC  #$D0
00D3 RETURN   RTS
```

ADDR, composed of two bytes I'm calling ADDRHI and ADDRLO, always points to the character currently being processed. When BASIC calls GETBYTE, the three instructions from $00BC to $00C1 do a two-byte increment of the value in ADDR. This is an example of self-modifying code, which is usually bad practice, but which runs faster in this case than equivalent code which is not self-modifying. Since the 6502 has no simple indirect addressing mode for data, it would be necessary to save the contents of an index register, load it with zero, load the desired byte using an indexed indirect mode, and then restore the original value of the index register. Since this routine is used so heavily by BASIC, what appears to be a slight speed advantage turns out to be quite significant.

The instruction at REGETBYTE loads the A register with the actual value of the next byte to be interpreted. REGETBYTE is an alternate entry point used, if anything, more than GETBYTE. REGETBYTE gets the current character without stepping ADDR, but still sets the flags for the type of byte which it returns. The flags signal conditions which you would expect BASIC to need to check for: an end-of-statement mark or ASCII digits. The processor's Z flag signals end-of-statement if it is set, while the C flag is used to signal an ASCII digit if it is cleared. In order to understand this routine, you may need to walk through it with various values, but here's a play-by-play covering the operation of the routine.

With the byte in the accumulator, the CMP at $00C5 tests against the ASCII character for a colon (:). If the actual byte is a colon or "higher" (in the ASCII sequence), it jumps to the RTS at $00D3. Note that the C (carry) flag is set, indicating that the byte is not a digit. The ASCII characters for digits are $30 thru $39, just "below" the $3A for the colon. Also, if the byte is exactly a colon, the Z (zero or equal) flag will be set, indicating the end of a statement.

Next the CMP at $00C9 tests to see if the byte is the ASCII character for a space. This routine will not return a space but rather discards it and gets the next character. The BEQ at $00CB implements this by jumping back to GETBYTE, which increments the pointer and falls back into REGETBYTE. This is why BASIC totally ignores spaces, even spaces imbedded within numbers. (The actual characters in a string are picked up differently, allowing spaces to appear there).

Now the plot thickens. Before, it was a simple matter of a compare to set the carry flag if the byte was too high to be a digit. Now, we must also set the C flag if the byte is too low to be any of the digits. A comparison with the value of an ASCII "0" ($30) will leave the flag in the opposite state from the way it was set above - that is, set will indicate a digit, and cleared will indicate a non-digit. While there are many ways to reverse this, most obvious ways would disturb some registers' contents, requiring some tricky coding to preserve them. The code from $00CD thru $00D2 is a very elegant way of making the flag mean what it is supposed to.

The SEC at $00CD and $00D0 just prevent the processor from doing a borrow during the subtractions, which would obviously change the results. Notice that subtracting $30 and then subtracting $D0 is equivalent to subtracting $0100, and since we are discarding the borrow from this operation, the net result is that of subtracting $00, except for the flags. Starting with a byte greater than or equal to $30 in the accumulator, the first subtraction leaves a value from $00 thru $CF. The second subtraction must then necessarily involve a borrow, which leaves the C flag clear. Starting with a byte less than $30, the first subtraction leaves a byte from $D0 thru $FF, with a borrow which is ignored by the following SEC. Then the second SBC ($D0 from a value in the range $D0 - $FF) cannot involve a borrow, leaving the C flag set. Ta-dah!! Notice that if the final result is a zero, the Z flag will be set, indicating end-of-statement like the colon above. (BASIC uses a colon to separate statements on the same line but uses a null [$00] to indicate the end of a line, which must also be the end of a statement.)

Notice that throughout this code, great pains are taken not to disturb any registers. Also, the sequence of the tests is optimized for speed. The large majority of the bytes in a BASIC program are greater than the colon, so most often the first branch will be taken, saving the time required by the other tests.

Since this routine is called so often by BASIC and because it is run in read-write memory instead of ROM, it makes BASIC susceptible to some tinkering. For instance, suppose you want to implement an additional command. A very simple way to do this is to make the command consist of a single character which is not normally used by BASIC, such as #, %, &, or '. You could insert the instruction JMP PATCH in place of the CMP #$3A and the first byte of the CMP instruction at $00C5. PATCH could look something like this:

```
PATCH   CMP  #(your selected character)
        BEQ  (to machine code executing the command)
        CMP  #$3A   ; to substitute for the destroyed
                    ; instructions in REGETBYTE
        BCC  NORMAL
        RTS
NORMAL  JMP  $00C9  ; to finish the other tests
```

This idea is carried to an extreme to implement most of the commands and modifications that HEXDOS adds to BASIC. There are some other key spots to intercept such things as the LOAD and SAVE commands, but all the I/O routing, for instance (PRINT #, etc.) is done through a mechanism similar to this.

Obviously, this would impose a slight speed penalty, since this patched routine is called for every byte of your BASIC program, but that's a tradeoff for the convenience of having the extra command / commands. On the other hand, by implementing the new command in machine language, you may gain more speed than you lose by adding this patch.

Summary

GETBYTE -- increments BASIC's current-byte pointer and then falls into REGETBYTE

REGETBYTE -- loads the current byte into the A register, does not disturb any other registers, and returns flags to indicate the type of byte:

End of statement ($00 or $3A) - Z and C flags both set.
ASCII digit ($30 - $39) - Z flag and C flag cleared.
ASCII space - gets the next character and sets the flags accordingly.
Any other character - Z flag cleared and C flag set.

The N and V flags are both affected by this routine but have no particular significance to BASIC.

The original copy of this routine is in ROM starting at $BCEE.