NOTES FOR CURSOR 10
May 1979
Ron Jeffries, Publisher
Glen Fisher, Editor

CURSOR™
Programs for PET™ Computers

"Everything should be made as simple as possible, not simpler."
- Einstein

THE CODE WORKS
Box 550
Goleta, CA 93116
805-683-1585

A CURSORY GLANCE

My impression is that Commodore has very bright guys in their engineering department, and that they have designed a good system. However, I think that the problems of getting a product as complex as a floppy disk system fully ready for the consumer market have been seriously underestimated. Many of the best technical people at Commodore are quite young, and do not have the "seasoning" that comes from having your head bloodied by a product that was introduced too soon. Another major problem with the disk system is that the manual doesn't give enough information for either the new user or the experienced "hacker". I can only guess at what sort of tricks advanced users will be able to make the disk perform, as the manual is frustratingly devoid of technical details. All that I can report is that the little bit of information that is given about the 2040 disk utility commands such as BLOCK EXECUTE, MEMORY WRITE, and MEMORY EXECUTE are tantalizing clues about what may be under the covers. There is 4K bytes of memory out in that system, with a full blown 6502, as well as a substantial set of stuff in the disk ROM. Sounds like those pieces could be made to do some pretty fancy tricks when we get enough information.

Evidently I was not the only person that was unhappy with the way that the Commodore Disk Operating System treated the user. According to Commodore, a new function will soon be available called "The Wedge". (The names comes from the fact that it prints a wedge ">" as its prompt character). As I understand things, the wedge will be the first program on each diskette. When you bring the system up "cold", you will do the following: LOAD "*",8. (In the Notes for Cursor #9, I criticized the very awkward sequence of commands that are needed to look at the directory of a newly inserted disk. At that time I wasn't aware of this shorthand method.) The command "LOAD "*",8 goes to the disk and brings into memory the first program on the disk. (I wish that it brought in a program with a certain name, such as STARTUP...) So, if you have the new "Wedge" program as the fisrt thing on your disk, you will be able to load it into memory automatically. From what I know, the Wedge is a machine language program that will make many of the DOS commands available from the keyboard. Most importantly, it will allow you to look at the directory of a disk without destroying the program that is currently in memory.

My experience with the 2040 for a little over one month has been mixed. I'm not sure why, but my disk has not performed as it should. One of my disks works quite well (although not perfectly.) On that drive, I find that diskettes sometimes won't "initialize" correctly. However, once they start working, I have almost no trouble. But, if I take a disk from drive zero and put it in drive one, I don't have very good luck. The most common thing that happens is that I won't be able to read the disk at all on drive one. In my opinion, Commodore will probably solve these problems. I know that they are trying to decide what to do about the fact that the 2040 runs hot. I just hope that those of us with early units will get an upgrade once they find the problems.) The reason that I think that Commodore will solve the disk problems is very simple: if they do not succeed in making the 2040 work reliably, they will not be able to stay in the computer business. Since I assume that they plan to continue in the business, I also believe that they will solve their problems. (Incidentally, since CURSOR is sold only to Pet owners, I have a strong economic incentive to see Commodore do well.)

CURSOR 10 HAS THESE PROGRAMS: (Programs ending in '!' use CB2 sound.)

COVER!	Another musical cover. Hook up your Pet for sound and enjoy!
TITRATE	Practice titration by turning your Pet into a chemistry lab. By Garry Flynn.
FINANCE	Calculate mortgages, pension plans, savings, etc. By T.M. Wagner
COURSE	An interesting obstacle course, with varying degrees of difficulty. By Glen Fisher.
ASM	A simple assembler for the 6502. By Glen Fisher.
READER	This program turns machine code into DATA statements. By Glen Fisher.

Distributed in Japan by:
SYSTEMS FORMULATE Corp.
Shin-Makicho Bldg., 1-8-17
Yaesu, Chuo-Ku, Tokyo 103

Distributed in England by:
AUDIOGENIC Ltd.
P.O. Box 88
Reading, Berkshire

MORE ABOUT THE PROGRAMS

COVER... To appreciate this cover, you will need to have your Pet attached to a small amplifier. (Actually, a big amplifier is even nicer, but it doesn't matter very much.) CAP Electronics, 8462 Hillwood Ln. Tuscon, AZ 88715 sells a modified radio and a demonstration tape for about $30. You plug their edge connector into the back of your Pet, and you are all set for the style of sound that CURSOR and several other vendors use. (It is called "CB2 Sound", as that is the function of the 6522 that is used.) You can also use the Radio Shack 200mw Speaker/Amplifier, Catalog Number 277-1008. Channel Data, 4141 Hollisterv Ave Santa Barbara, CA 93111 sells the Radio Shack amplifier and the appropriate edge connector for about $20. If you want to do a little work yourself, you will need to get an edge connector, and solder two wires to the connections to pin 12 (ground), and pin M (sound). See Notes for CURSOR 3 for a diagram. However you do it, DO IT! If you have a Pet, and you don't have sound, you are missing one of life's little pleasures.

TITRATE... This is a beautiful example of how an educational excercise can also be a lot of fun. Garry Flynn wrote this simulation of the process of titration, which is a common procedure in chemical analysis. Remember, you are trying to get the solution to just turn color, which is shown as grey on the Pet. If you go too far, it will turn white, which means that you went past the end-point, and wasted the sample.

FINANCE... This is a program that will assist you in several routine calculations, such as compound interest, mortgage payments, pension plans, etc.
COURSE... You select the degree of dificulty that you want for the obstacle course that the program builds for you. After the course is displayed, use the number keys to move the cursor from the upper left corner to the lower right corner in the minimum amount of time. Hint: don't forget that you can move diagonally! The course is a bit tricky, but we assure you that there is always a path that will take you to the finish.

ASM... Please see the article that begins on Page 3 of these Notes. Our purpose in publishing this simple assembler is to make it possible for a large number of people to experiment with 6502 machine language by using an assembler. The CURSOR assembler has one design goal: get as much capability into as little memory as possible. There are far better assemblers around, but in one case, by the time you get the assembler into an 8K Pet, you have almost no room left for your code! Please don't think for a minute that we are suggesting that you should quit using Basic. But in some cases, you will want to experiment with the "guts" of your system, and to do that you'll have to face machine code. Also, there are times when the extra overhead of the Basic interpreter makes a function too slow to be useful.)

READER... After you have written and debugged your assembly language program, what next? You probably want to use Basic as the easiest way to load the code into the machine. So, first you assemble the code (quite likely into the second cassette buffer that starts at 826 decimal). Next, you load in the READER program, and run it, giving a starting and ending memory location. READER will print DATA statements on your screen, which you can then enter into the program by pressing [RETURN] on each line. If it won't all fit on one screen, just keep repeating the process until you get it all entered in. Then, you can delete the Reader program, and save your machine code as a Basic file. This all sounds harder than it is! But believe me, if you have ever tried doing the same thing the hard way by copying the stuff down by hand, you can appreciate what a nice utility this is.

"There's no sense being precise about something
when you don't even know what your are talking about."
- John von Neumann

THE CURSOR ASSEMBLER

A prerequisite to using the assembler is a knowledge of 6502 machine language. We don't have room to go into it now, but there are a number of books available on the subject. At the least, you should have a copy of the MOS Technology 6502 Programming Manual.

Why an assembler? For that matter, what IS an assembler? As is our usual fashion, we shall answer the second question first. An assembler is a program that reads an assembler program and translates it to machine language.

(Before we take any more questions, let us clarify a confusing point: the word 'assembler' can refer to either the program to be translated or to the program doing the translating. This is unfortunate, but true. Henceforth, the program doing the translating will always be 'THE assembler' or 'AN assembler', while the program being translated will just be 'assembler'. We now return to the lecture already in progress.)

Assembler programs are machine-language programs written in a way that makes them halfway comprehensible to people. Real machine language is actually pure numbers, and is completely indecipherable except to the computer (and REALLY dedicated hackers). Assembler uses names for all those things for which numbers don't make sense to people. For example, the number $AD, to the 6502, means 'load a number into the accumulator'. To people, it means nothing at all. So the assembler lets people write 'LDA' (LoaD into Accumulator) instead, which is more easily remembered. (A note: anything, and ONLY those things, starting with a '$' are hexadecimal, or base-16, numbers.) Another advantage of using names instead of numbers is that programs written using names tend to have fewer errors in them than programs written using numbers only.

The 6502 computer has 56 different kinds of instructions, each coming in several styles. To match that variety, the assembler has 56 different names, one per instruction, with styles enough to match all of the 6502's. We won't waste space by listing all the instructions, but it is worthwhile to look at the styles they come in. The proper name for the styles that instructions come in is 'addressing modes'. The 6502 has thirteen different addressing modes. The addressing modes control just how and where each instruction gets hold of the number it's going to play with. Things aren't as bad as they sound: no instruction uses all thirteen modes; many use fewer than four, and a number of instructions are restricted to one mode only. Along with the list of modes, we'll tell how it is indicated in the MOS Technology assembler (as that is how many programs are published) and in our assembler (as that's how you'll have to write it).

Mode	Them		Us	
Implied	BRK		BRK	
Accumulator	LSR	A	LSRA	
Relative	BCS	NEMO	BCS	+NEMO
Immediate	LDA	#10	LDA#	#10
Absolute	JSR	QUIX	JSR	;QUIX
X-indexed absolute	STA	FOO,X	STAX	FOO
Y-indexed absolute	ADC	ABC,Y	ADCY	ABC
Page zero	BIT	ZPG	BIT.	.ZPG
X-indexed page zero	INC	SPOT,X	INC.X	SPOT
Y-indexed page zero	SBC	PLUGH,Y	SBC.Y	PLUGH
Indirect	JMP	(THERE)	JMP@	THERE
Pre-(X-)indexed indirect	AND	(MASK,X)	AND@X	MASK
Post-(Y-)indexed indirect	CMP	(NUM),Y	CMP@Y	NUM

As a bonus, we threw in a sampling of instructions. All those funny symbols under 'Us' will be explained later.

The 6502 can refer to up to 65536 different spots in which it can remember numbers. The exact number available depends on how much memory your Pet has in it, however. Each one of those spots in memory has a number (called its 'address') which it can be referred to by. For example, the first spot in the 2nd cassette buffer is spot number 826. Clearly, it is just as annoying to have to keep track of those numbers as it is to remember the numbers for instructions. The assembler provides things called 'labels' to aid you in that. Whenever you happen upon a spot you want to refer to later, you can tell the assembler the spot, and what name you're going to call it. After that, you use your name for the spot, and the assembler will plug in the proper number in place of the name. The sample program below has several labels in it, which ought to help clear things up.

How to use the assembler

In this assembler, as in many assemblers, there are three classes of things you can say to it: instructions, operands, and directives. Instructions are just the names of the 6502 instructions (with a little extra tacked on the end). Operands are the names of the places from which the instructions get the numbers they play with. Directives are commands to the assembler itself, which don't get translated to machine language. For example, the command to attach a name to an address is a directive.

The instructions can be any legal 6502 instructios. The addressing mode of the instruction is indicated by a suffix of one or two characters. The characters are as follows ('bl' means 'blank' (there is no suffix)):

bl absolute	X absolute,X	Y absolute,Y
. page zero	.X page zero,X	.Y page zero,Y
@ indirect	@X indirect,X	@Y indirect,y
bl implied	bl relative	
# immediate	A accumulator	

The assembler comes with a broad selection of operands as well:

<u>One byte operands:</u>

#10	decimal literal (means 'literally, a ten')
$F2	hexadecimal literal ($F2 equals 242 in decimal)
'X	character (ascii) literal (represents the ascii value of the character. Thus, the operand 'R is the same as the number 82. If the character is shifted or is ',' or ':', it should be in quotes.)
.PLINTH	a page zero address, or a named constant
+LOOP	a relative branch destination

<u>Two byte operands</u>

;FOONLY	absolute address, or a named constant
;#59468	decimal address or two-byte integer
;$FFD2	hexadecimal address or two-byte integer

The assembler also has two <u>directives:</u> '@' and '='.

@826 tells the assembler that anything assembled from the directive on down should be put in memory spots 826 on down. In other words, the assembler's 'location counter' is set to 826. The location counter keeps track of where the assembled code is going to be put. It is also used to provide a value for named constants, as will be explained below.

@$33A does the same thing as the other @ directive, but takes a hexadecimal address, instead of decimal.

=HERE tells the assembler that you are going to call the current memory spot 'HERE'. The assembler will save the current value of the location counter under the name 'HERE', and look it up again whenever you use 'HERE' somewhere else (like in a branch instruction).

The '=' directive is also used to give names to constants. Giving names to constants has the same value that giving names to everything else does: it makes things easier to remember. Suppose you're writing a text editor (a popular pastime among hackers). For some reason you've decided to use a '1' to mean move up one line, and a '2' to mean move down. Compare the two program fragments, and decide which is more understandable:

```
                       |          @1
                       |   =UP                  (1 means UP one line)
                       |          @2
                       |   =DOWN                (2 means DOWN a line)
       CMP#   #1       |          CMP#   .UP    (did he say UP?)
       BEQ    +GOUP    |          BEQ    +GOUP
       CMP#   #2       |          CMP#   .DOWN  (how about DOWN?)
       BEQ    +GODOWN  |          BEQ    +GODOWN
```

Most people will prefer the stuff to the right, since it is more apparent what the intention of the code is. (The comments apply to either side.)

The assembler code should be typed into the assembler as DATA statements, starting at line 11000. Separate the instructions, operands, and directives from each other with commas. (You could enter them one per line, but that gets rather wasteful of space.) After you've typed in all your program, <u>save it</u> (along with the assembler). It would be a shame to have to retype all that code. After the save is finished, run the assembler. It will print a listing on the screen as it assembles the program. (It does no good to route the listing to a printer; the cursor control keys are used in it, and they don't print very well. The result would be a goodly amount of scratch paper.) The assembler reads your program twice: first to find out what all the labels are, and again to produce the machine code. While your program is being read the first time, the assembler will print the labels as it finds them, to let you know what it's doing. When your program is read the second time, the main listing is done. The listing shows all of your program, and what values are put into what memory locations. Below is a part of the listing from the program that comes with the assembler:

```
836    162        LDX#
837      0              #0
838          =OUTER
838    160        LDY#
839      0              #0
840          =INNER
840    177        LDA@Y
841      1              .PTR
842    201        CMP#
843     32              '
844    240        BEQ
845      4              +SKIP
```

What Pet Owners Can Do

Recognize that you are the key person responsible for controlling your pet.

Make a commitment to control your pet.

KEEP YOUR PET LEASHED, FENCED OR CONFINED.

The numbers with a box around them represent reverse-video (the thing we do these notes on can't print that way). The number to the left is the value of the location counter, and tells where the number to its right was put. The second number is the number produced by the assembler in place of the instruction or operand to the far right. If the location counter is reverse-video (black on white) then nothing was produced by the assembler for that line; it only made a note for itself about the line. Last, the instruction, operand, or directive is listed. Its position on the line is adjusted in an effort to keep the listing somewhat readable, and is similar to the listing of the program below. When the assembler is done, it will say 'ASSEMBLY COMPLETE', and stop. If there were any errors, a message will have been printed beside the line in error, telling what the error was. (If you get 'BYTE TOO BIG' on a branch instruction, the label you're branching to is more than 128 bytes away, and can't be reached from the branch. If you get 'TOO MANY LABELS', increase the value of SIZ on line 100.)

When at last the assembler gives no error messages, you are ready to test it. Before you do, SAVE THE ASSEMBLER AND THE PROGRAM YOU WROTE! Assembler programs have a nasty habit of making the Pet go off into limbo, where you must turn the Pet off and on again to bring it back. If that happens, and you didn't save your program, you'll have to type it in all over again. To run the program, use the SYS command to make the Pet start running the machine language program. If your program starts at location 826, type

```
        SYS 826
```

If you're lucky, your program will run the first time. If not, you'll have to debug it. Debugging programs is a whole subject unto itself, and will have to wait for another time. One suggestion, however: use the machine-language monitor. It isn't perfect, but is a help in finding where the program goes wrong, especially with the new Pets, when it stays around all the time.

The assembler as it appears in Cursor comes with the following program built in. The program looks at the Pet's screen and changes all non-blanks to reverse video (or back, if they were already reversed). We list it here side-by-side with the same thing written using MOS Technology style, both so you can see how programs are written using the assembler, and for further help in reading MOS Technology style programs.

```
        MOS Technology    |          Us
                          |          @32768
CRT     EQU  32768        |  =CRT                       (note where the screen buffer is)
                          |          @1
PTR     EQU  1            |  =PTR                       (handy spot on page zero)
                          |          @$80               (named constant here)
BIT7    EQU  $80          |  =BIT7                      (high-order bit of a byte)
        ORG  826          |          826                (put code into the 2nd cassette buffer)
        LDA  CRTADR       |          LDA    ;CRTADR     (set PTR to start of CRT buffer)
        STA  PTR          |          STA.   .PTR
        LDA  CRTADR+1     |          LDA    ;CRTADR+1
        STA  PTR+1        |          STA.   .PTR+1
        LDX  #0           |          LDX#   #0          (X counts quarter-screens)
OUTER   LDY  #0           |  =OUTER  LDY#   #0          (Y counts characters)
INNER   LDA  (PTR),Y      |  =INNER  LDA@Y  .PTR        (get a char from the screen)
        CMP  #'           |          CMP#   '           (is it a blank?)
        BEQ  SKIP         |          BEQ    +SKIP       (if so, leave it alone.)
        EOR  #BIT7        |          EOR#   .BIT7       (otherwise, reverse it)
        STA  (PTR),Y      |          STA@Y  .PTR        (and put it back)
SKIP    INY               |  =SKIP   INY                (on to next char)
        CPY  #250         |          CPY#   #250        (end of quarter-screen?)
        BNE  INNER        |          BNE    +INNER      (no - go flip next char)
        CLC               |          CLC                (yes - advance to next quarter-screen)
        LDA  PTR          |          LDA.   .PTR        (by adding 250 to start pointer)
        ADC  #250         |          ADC#   #250
        STA  PTR          |          STA.   .PTR
        LDA  PTR+1        |          LDA.   .PTR+1      (add carry to high byte)
        ADC  #0           |          ADC#   #0
        STA  PTR+1        |          STA.   .PTR+1
        INX               |          INX
        CPX  #4           |          CPX#   #4          (finished last quarter?)
        BNE  OUTER        |          BNE    +OUTER      (if not, do next quarter-screen)
        RTS               |          RTS                (all done - go away)
CRTADR  DW   CRT          |  =CRTADR ;CRT
```

You can try out the program by running the assembler after it has loaded. The assembler will print a listing, as described above, and stop when it's finished. (If you get an error, you must have had a load error, as we carefully exterminated all the bugs before we let it loose.) After the assembler is done, type 'SYS 826', and watch what happens. You can re-run the program as many times as you like.

The observant among you will have noticed that the program doesn't do the whole screen at once; it carves it up into quarters, and does one quarter at a time. The reason is that the 6502 does all its work in byte-sized chunks (no humor intended - honest!). The largest number that one byte can hold is 255. Unfortunately, the screen is 1000 characters long, which is clearly more than 255. After some deep and abstruse calculations, we find that one-quarter of the screen is 250 characters long, and 250 WILL fit in one byte. Taking advantage of that, the program has a small inner loop to reverse one quarter of the screen, and an outer loop to count how many quarter-screens have been reversed. As you can see, programming in Basic does have its advantages.

WHERE TO LEARN 6502 ASSEMBLER

There are several books on the market intended to teach 6502 machine language, of varying quality. The better ones are rather biased towards the KIM, and away from the Pet, and so are of limited usefulness. The books we have been able to see ourselves are:

Programming a Microcomputer: 6502 by Caxton C. Foster - covers the subject fairly well, with many different applications. However, those applications are mostly for the KIM. As a source of information, rather than programs, it could be useful.

Programming the 6502 by Rodney Zaks - a rather thick book, 305 pages. That isn't as much as you might think, as a quarter of the book is given over to a compendium of descriptions of the instructions. On the whole, you would do better with something else (the MOS Technology manual covers most of what's in here all by itself.)

The First Book of KIM by Butterfield, Ockers, and Rehnke - almost useless for the Pet, as it is chock-full of many fascinating goodies for the KIM. Nearly all of those goodies will work only on the KIM, however, and not the Pet, as the Pet and the KIM have different built-in software.

The MOS Technology Programming Manual for the 6502 - While not the best, it is useful for the Pet, as it makes very few assumptions about how you are running your programs, or where. Also, since it is put out by MOS Technology, which makes the 6502, it can be considered to be the final source of information, to which all others much conform. (Since they make it, they BETTER know how it works!)

There are other books on 6502 machine language available. The ones mentioned above are just the ones we have seen. If we sound somewhat harsh on them, it's that we have as high standards for books as for programs. By the time you read this, Adam Osborne's 6502 book should be out. We haven't seen it, but he has an excellent reputation in such matters. It is called 6502 Assembly Language Programming by Lance A. Leventhal, and can be bought from your friendly local computer store. If you don't have a friendly local computer store, or are harboring a grudge against it, you can write to:

Osborne McGraw Hill Associates
630 Bancroft Way
Berkeley, California 94710

Those of you acquainted with other 6502 assemblers will have noticed that the CURSOR assembler requires that the programmer type more than the usual 6502 assembler. The reason for that is to keep the assembler small, so that larger programs may be assembled. Since the assembler can know exactly what code to put out without having to know about other parts of the program, it is simpler, and therefore smaller. For example, it can produce the proper number for a specific instruction without having to decide what kind of operand is being dealt with. Similarly, it can handle the operand correctly without having to remember which instruction the operand went with. The extra typing is an inconvenience, true. Having used it as our production assembler for a while, we have found that the inconvenience isn't as much as would be thought at first. Also, since your program is part of the assembler (which is in Basic), you have the full power of the Pet's screen editor available to correct your mistakes.

ABOUT PROGRAMS LARGER THAN 8K

The mail is running about even on the issue of CURSOR publishing 16K programs (such as GAMMON in CURSOR #9). Since some people feel strongly that we should NOT publish large programs, we are going to proceed very carefully. One possibility will be that when we want to publish a 16K program, that we will include it as a sixth program on the tape, so that our loyal 8K subscribers don't feel cheated.