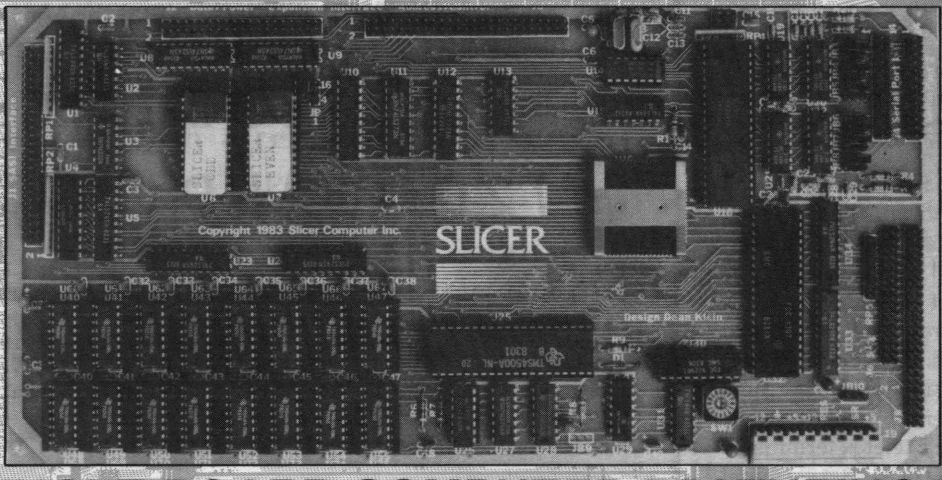
GET REAL 16 BIT POWER



A SINGLE BOARD COMPUTER FEATURING THE INTEL 80186

(see Byte magazine April '83)

THE SLICER

- Full 8MHz 16-bit microprocessor having complete software compatibility with the 8086 and 8088
- 256K Bytes of RAM plus 32K Bytes of EPROM memory capacity on board
- Floppy disk controller can run the combination of 8", 5-1/4", 3-1/2" drives simultaneously
- SASI port for hard disk controller
- Two full function RS232C serial ports with individually controlled baud rates from 50 to 38.4K baud
- 8K of EPROM contains drivers for peripherals, commands for hardware checkout and software testing
- Source for monitor and bios included on disk
- Bios for CP/M 86* supports 8", 5-1/4", 3-1/2" drives and the Xebec 1410 controller for hard disks
- Board size 6" x 12" power requirements +5 @ 3A, +12V @ 60mA, −12V @ 50mA
- Complete documentation included.

Sold in various forms:

Assembled and tested	$1,075.00
Full Kit	895.00
Easy Kit (hard to get parts)	450.00
Bare Kit (board, Rom, doc, and disk)	150.00
CP/M 86* available	85.00

Regular U.P.S. shipping within continental U.S. is included.

THE NEW SLICER EXPANSION BOARD

- 256K bytes dynamic RAM using the TMS4500 DRAM controller
- 2 RS232C serial asynchronous ports using the Signetics SC2681 serial communications circuit with baud rates selectable from 38.4K baud to 50 baud
- 2 RS232C serial asynchronous ports using the Zilog 8530 SCC with header connectors for synchronous communications personality cards
- Real Time Clock with battery back-up using the CDP6818 RTC circuit, on-board or off-board battery
- Parallel printer port for Centronics-type printers
- Same size as the SLICER, Power: 5V @ 1.5A, +12 @ 200 mA − 12V @ 100 mA

Sold in various forms:

Assembled and tested	$850.00
Full Kit	650.00
Memory Board	450.00
3 Port Board and Clock	225.00
Bare Board and Doc	95.00

Regular U.P.S. shipping within continental U.S. is included.

SLICER™

Choose 8 MHz CPU for speed
6 MHz CPU for speedy delivery

SLICER COMPUTERS INC. 2543 Marshall St. N.E.
Minneapolis, MN 55418 (612) 788-9481

Mastercard, Visa, check, money order or C.O.D. orders accepted.
Please allow 4 weeks for delivery.

*CP/M a trademark of Digital Research, Inc.

SLICER COLUMN

(continued from page 16)

the source code with it so you can customize it.

If you are running Siemens drives you need to set up the Slicer for a 6 ms step rate, a slower step rate makes them unreliable.

They've had a lot of OEM enquiries lately and have just received a 100-board order from England. There are several OEMS in the U.S. already basing new systems on the Slicer. One outfit is putting the Slicer inside a teletex terminal.

Jade advertises a TTX-3000 terminal which Jade says contains a power supply sufficient to support a single board computer like the Slicer and two drives. Otto says that the TTX-3000 supply cannot support a Slicer and two drives.

He says that the Wyse 50 terminal works very well on the Slicer and it costs only about $500. It lets you select between an 80 and 132 character screen, and it has a sharp, crisp display.

I have a Visual 50 terminal on my Slicer and I'm really happy with the display and the keyboard. It emulates a bunch of terminals including the VT 52, the Heath terminal, and, of course, the ADM 3A. It only has an 80 column screen, however and it costs about $650.

Slicer Articles and Software Wanted

I know it takes a while for you to get rolling on a new machine and I know that the Slicer is unique because of the active support from Slicer Inc. However, once you are rolling, please keep us in mind for letters, articles, software and what not.

■ ■ ■

C'ing Clearly

By Tony Ozrelic

6708 Melrose
Los Angeles, CA 90038

Not too long ago I had a blinding flash of stupidity (as opposed to inspiration) and wound up destroying the directory of my documentation disk. The one with about 100k of typescript on it.

Needless to say, I did not look forward to re-typing the entire disk even though I knew what was on it! You see, I had not made any backups. Shame on me! I put the disk aside for a few days while I considered having a mental breakdown.

It soon occurred to me, however, that I might look at the undamaged portions of the disk and recover the text directly. The wheels in my head began to turn, and I put together two programs, RDISK and CDISK to help me undo the mess.

RDISK.C

This program is designed to display a disk a sector at a time. Basically, you give it a drive (a thru p) and the beginning track (0 thru 77) and it shows you the sectors which have text on them. Other sectors are skipped (see Figure 1).

RDISK also shows you the current track and sector. Hitting any key except "q" will take you to the next displayable sector. This way you can step thru the disk sector by sector, writing down the track and sector numbers of the text you need and then use CDISK to copy the text to a good disk. To use it, type:

rdisk b 3

To read the disk in drive B: starting with track 3.

CDISK.C

CDISK takes the track/sector numbers you have copied and reads them into a file on a good disk. See Figure 2. Use your text editor to generate a file with a track and a sector number on each line:

```
12  20
12  21
12  22
12  23
12  24
```

and so on. The first number is the track number, the second the sector.

While using RDISK, I noticed that CP/M uses a lot of consecutive sectors for text storage. To make typing easier, I put in a loop generator—if a line begins with

Figure 1 - Read Disk Routine in C

```
/*
    rdisk - read disk sectors from a damaged disk. Hitting any key
    will read the next sector on the disk, and hitting 'q' will
    quit the program. Use Manx Aztec C compiler on this program
*/

#include  "libc.h"


/*
    maximum, minimum tracks, sectors
*/

#define   MINTRK        0
#define   MAXTRK        77
#define   MINSEC        1
#define   MAXSEC        26
#define   YES  1
#define NO     0


/*
    macro definitions for the BIOS routines needed to read the disk
*/

#define   home()        bios(8,0,0)
#define   select(n) bioshl(9,n,0)
#define   seek(n)       bios(10,n,0)
#define   setsec(n) bios(11,n,0)
#define   setdma(n) bios(12,n,0)
#define   read()        bios(13,0,0)
#define   write()       bios(14,0,0)


/*
    ivect is the index vector used to "straighten out" CP/M's sector
    mapping so tracks & sectors can be incremented by one
*/

int ivect[]={
    0,1,7,13,19,25,5,1,17,23,3,9,15,21,
    2,8,14,20,26,6,12,18,24,4,10,16,22
    };


/*
    main program - type

    rdisk <drive> <trk>

    where <drive> is a letter a thru p for the drive on which the bad
    disk is mounted. <trk> is the first track you want to look at
*/

main(argc,argv)
int argc;
char *argv[];
    {
    char buf[128];
    int disk,strk,trk,sec,i,tflag;

    if(argc>1) disk=*argv[1]-'A'; /* get the drive number */
    else disk=0;
    if(disk<0 || disk>15) {
         printf("?\nBad disk drive %c -should be A thru P\n",*argv[1]);
         exit(1);
         }
    if(argc>2) strk=atoi(argv[2]);
    else strk=MINTRK;
    printf("\n drive %d track %d selected\n",disk,strk);
```

(Listing continued next column)

```
for(trk=strk;trk<MAXTRK;trk++) {
     for(sec=MINSEC;sec<=MAXSEC;sec++) {
          select(disk);seek(trk);setdma(buf);
          setsec(ivect[sec]);
          read();
          tflag=YES;
          for(i=0;i<128;i++) {
               if(!istext(buf[i])) {
                    tflag=NO;
                    break;
                    }
               }
          if(tflag) {
               printf(
                    "\n\n>>>>>>> %d %d <<<<<<<\n\n",
                    trk,sec
                    );
               for(i=0;i<128 && buf[i]!=0x1a;i++) {
                    putchar(buf[i]);
                    }
               if(getchar()=='q') exit(0);
               }
          }
     }
}

/*   istext - is it regular ascii text?          */

istext(c)
char c;
     {
     if(c=='\t' || c=='\n' || c=='\r' || c==0x1a) return c;
     else if(c>=' ' && c<='~') return c;
     else return 0;
     }
```

Figure 2 - Copy Disk Routine in C

```
/*
     cdisk - copy disk sectors as indicated by input file

     This program is written to compile under Manx Aztec C and is
     used to recover blocks of text from a damaged disk. This is
     done by reading a file which indicates which blocks are to
     be read in. These blocks are then written to a file on drive A:
*/

#include  "libc.h"


/* minimum, maximum track and sector sizes                */

#define   MINTRK         0
#define   MAXTRK         77
#define   MINSEC         1
#define   MAXSEC         26


/* CP/M buffer size, line size, etc.                      */

#define MAXBUF 128
#define MAXLINE     80
#define   YES  1
#define NO     0
```

(Listing continued next page)

a dot, the next line is used as the end of a loop, so that the above sequence would be written as:

```
12  20
.
12  24
```

In many cases, you will find that reconstructing a file consists of just a few lines of typing. This file is given to CDISK along with the letter of the drive containing the bad disk and a filename on A: which will hold the copied sectors:

cdisk b sectors dyna.doc

Is what I used to recover DYNA.DOC from drive B: using the numbers in the file SECTORS. Recovering this file alone saved me about 29 Kbytes of typing!

A few Words of Warning

These programs work fine if you have Manx Aztec C and standard single-sided single density 8″ disk drives.

Not all compilers can call BIOS routines directly, so consult your manual to see if this type of call is available. Also, I have noticed that not all compilers accept such sophisticated macros as #define select(n) bioshl(9,n,0), so you may need to make each of these macros a function call.

Another thing to watch out for is disk formatting. Your disk may be formatted to speed up data access so the sectors on the disk may not be numbered 1,2,3,.. but rather, offset by a constant, such as six.

This offset allows the controller to access the next sector more quickly since a slow controller might not be able to keep up with consecutively numbered sectors and would wait a full revolution before fetching the next one. All this means that ivect[ ] must be altered according to the way your sectors are mapped on the disk.

Also, I have found that on some minifloppies the sectors are numbered from zero, but 8″ disks are numbered starting at one. Consult your friendly neighborhood BIOS listing or wizard to find out how your disk is formatted.

(C'ing Clearly continued next page)

Bells and Whistles

These programs are about as bare-bones as you can get. Some possible enhancements are:

Fix RDISK so you can move from sector to sector easily—perhaps n for next, p for previous, g for goto.

Make RDISK log tagged sectors in a text file for use by CDISK, or, better yet, transfer tagged sectors directly to a text file.

That's about it for now; study the listings and try to get them to work on your computer/compiler. If you improve on these programs, or discover your disk is formatted differently from what's here, send us your stuff so we can build up a better program for all.

C Users' Group Meetings

Somebody has finally taken the initiative and started a C Users' Group meeting here in LA. The meeting will be held the second Tuesday of every month at Shakey's Pizza, 5321 Laurel Canyon, North Hollywood. The meeting starts at 7 PM and lasts until 9 PM, or whenever.

■ ■ ■


```
/*
     these are definitions for each of the CP/M bios operations needed to
     access the disk.
*/
#define   home()          bios(8,0,0)
#define   select(n) bioshl(9,n,0)
#define   seek(n)         bios(10,n,0)
#define   setsec(n) bios(11,n,0)
#define   setdma(n) bios(12,n,0)
#define   read()          bios(13,0,0)
#define   write()         bios(14,0,0)


/*
     ivect is the index vector used to map the linear address (1,2,3...26)
     into CP/M's sector address (1,7,13,...22). This index varies from
     system to system depending on how your disk sectors are mapped. You
     may need to change this map if you use the program on double density
     or minifloppy drives
*/

int ivect[]={
     0,1,7,13,19,25,5,1,17,23,3,9,15,21,
     2,8,14,20,26,6,12,18,24,4,10,16,22
     };


/*
     main program - type

     cdisk <drive> <numfile> <datfile>

     where <drive> is ranges from a thru p. The bad disk is on this drive.
     <numfile> is the text file of track and sector numbers needed to
     be recovered. <datfile> is the file in which the text will be stored.
*/

main(argc,argv)
int argc;
char *argv[];
     {
     char buf[MAXBUF],s[MAXLINE];  /*buf is gonna be CP/M's disk buffer*/
     int disk,trk,sec,i,endtrk,endsec;
     FILE *fnum,*fdat;

     if(argc<4) {
          puts("?\nUsage:\ncdisk driveno numfile copyfile\n");
          exit(1);
          }
     disk=*argv[1]-'A';        /*get the drive we're gonna read */
     if(disk<0 || disk>15) {
          printf("?\nBad disk drive %c\nShould be A thru P\n",*argv[1]);
          exit(1);
          }
/*
     fnum is the file with the track/sector numbers we need to read
     fdat is the text file we're putting the text into

*/
     if((fnum=fopen(argv[2],"r"))==NULL) cantopen(argv[2]);
     if((fdat=fopen(argv[3],"w"))==NULL) cantopen(argv[3]);

/*
     until we run out of numbers, let's read in blocks from the disk
*/
     while(fgets(s,MAXLINE,fnum)!=NULL) {
          if(*s=='.') {       /*if a dot, get next number*/
               fgets(s,MAXLINE,fnum);
               getnum(s,&endtrk,&endsec);
               }
          else {
               getnum(s,&trk,&sec);
               endtrk=trk;endsec=sec;
               }
          while(1) {
          printf("\n\n>>>>>> trk %d   sec %d <<<<<<\n\n",trk,sec);

               select(disk);  /*now read disk*/
               setdma(buf);
               seek(trk);
               setsec(ivect[sec]);
               read();
               puts(buf);
               select(0);     /*go back to A: to write out sector*/
               fputs(buf,fdat);
```

```
                    if(trk==endtrk && sec==endsec) break;
                    else {
                         sec++;
                         if(sec>MAXSEC) {
                              sec=1;
                              trk++;
                         }
                    if(trk>MAXTRK-1) trk=0;
                    }
               }
          }
     fclose(fnum);
     fclose(fdat);
     exit(0);
     }


/*   cantopen - squwak about file opening             */

cantopen(s)
char *s;
     {
     printf("Can't Open: %s\n",s);
     exit(1);
     }


/*   getnum - get 2 nos from line                    */

getnum(s,n1,n2)
char *s;
int *n1,*n2;
     {
     *n1=atoi(s);
     while(isdigit(*s)) s++;
     *n2=atoi(s+1);
     }
```

END

FORTHwords

Column by Arne A. Henden

7415 Leahy Road
New Carrollton, MD 20784
301-552-1295

I just got back from a most enjoyable FORTH convention and thought you might like to hear about what I saw. In addition, this column will give you the latest benchmarks on the new processors, and the User's Corner will discuss printer graphics.

16-bit Microprocessors

The October 13, 1983 issue of EDN magazine (p. 193) has one of the more interesting comparisons of the currently available 8 and 16 bit processors. Instead of using benchmark speeds, Kuare Christian compares processor architecture (number of registers, stacks, addressing modes, etc.). I feel this is a much more valid comparison than benchmark times.

However, I presented simple benchmarks in Micro Cornucopia in issue #5 (April 1982), comparing FORTH implementations on various processors. Since then, UNIFORTH has been ported to several new 16-bit processors. Figure 1 lists these new implementations, along with a Z80 base version for comparison. The general operations tested are listed beside the test name; the full coding of the test can be found in issue #5. The times listed are in microseconds for one execution of the loop.

I'm impressed with the 8088/8087 pair. While a considerable amount of coding is required to fully implement the 8087's transcendental functions, the speed advantage is remarkable. UNIFORTH uses the data stack for storing both integer and floating point numbers. If you use the 8087's stack for the floating point storage, tests T6-T16 would be 10-30% faster.

The PDP-11 tests indicate the speed advantage between various models. T3A is slow because the PDP-11 instruction set does not include unsigned multiply/divide, so these functions must be emulated. The new LSI-11/73 card from DEC using the J-11 chip will benchmark right at the 11/44 times. For $2000, this card will be an easy way to upgrade your LSI-11.

The 68000 times indicate that FORTH on the 68K is within a factor of two of FORTH on a PDP-11/44. However, the listed 68K times are for a 16-bit implementation. A full 32-bit version is about the same speed on the 68K, but would be much slower (if even possible) on the 11/44. A 4MHz 68000 is not all that different in speed from a 4MHz Z80. The advantages of upgrading to a 68K are the larger address space, numeric coprocessor support, and the faster clock speeds.

The 16032 timings are not really indicative of the ability of this processor. The benchmarks were run on a DB16000 development board which has wait-stated memory, so any operation that requires memory access (such as T4) is significantly slower than would be the case in an actual system. I anticipate the true times would be almost as good for the 16032 as for the 68000. The 16032 is the only processor in this group that provides 32-bit integer multiply and divide, making a true 32-bit FORTH possible.

The FIG National Convention

I attended the National Convention (held in Palo Alto, CA, October 14 and 15) for the first time as an exhibitor. Unfortunately, duties at the booth prevented me from taking in all of the sights, so my report is going to be one-sided. If someone else who attended the meeting wants to report on the sessions, please drop me a line.

For a very specialized field, the turnout was impressive. Figures quoted to me were about 600 people on Friday and 800 on Saturday, making this year's convention turnout double last year's figures. I found the experience mix interesting: about half of the people passing the booth had extensive FORTH knowledge (usually having implemented their own version of FORTH), and the other half were interested but had never used FORTH before. A large number had Commodore 64's and wanted a faster language than BASIC.

Laboratory Microsystems had a large display, showing off color graphic world maps and real-time graphics on an IBM-PC. Mountain View Press was the cornerstone of the exhibit area, with some 50 feet of counter space. They were displaying their latest publications and MVP-FORTH versions.

New Micros Inc demonstrated the Rockwell R65F11 FORTH chip development system. This is a 2.5-inch square board with the FORTH chip, an RS232 port, 2 parallel ports and up to 24K EPROM/RAM, all for $250!

Hartronix demonstrated a single board FORTH engine based on a microprogrammed bit-sliced processor. The board included a clock, up to 32K RAM, two RS-232 ports, a double density floppy controller and much more, for a $995 base price.

These two single-board FORTH products might make excellent controller boards.

Other products included a multiuser FORTH from Shaw Labs (CP/M, Micropolis and North Star), tFORTH/68K

Figure 1 - FORTH Benchmarks

TEST	Z80 4MHz SWFP	8088 5MHz SWFP	8088 5MHz 8087	11/2 FIS	11/23 3MHz FP-11	11/44 10MHz FP-11	68K 8MHz	16K 7MHz
T1 (loop)	55	35	35	38	19	8	14	24
T2 (I,+,-)	131	82	82	78	42	17	31	56
T3 (I,*,/)	739	141	141	250	128	86	58	77
T3A (I,U*,U/MOD)	697	169	169	1237	571	282	66	105
T4 (@,!)	113	73	73	90	48	22	33	61
T5 (C@,C!)	108	73	73	99	52	23	35	67
T6 (D+)	108	68	68	91	49	21	24	45
T7 (FLOAT,FIX)	420	632	70	990	109	40		
T8 (F*)	690	950	152	200	136	47		
T9 (F/)	1780	2740	173	320	182	48		
T10 (F+)	460	550	146	180	127	40		
T11 (2dup loop)	118	79	79	88	47	20	29	52
T12 (FSQRT)	10500	14000	140	2290	1110	480		
T13 (FLOG10)	35800	46700	340		4230	1360		
T14 (FEXP)	26000	33900	350		3560	1080		
T15 (FATAN)	38200	51200	400		5010	1440		
T16 (FSIN)	27300	36900	510		3010	910		
T18 (CMOVE)	1170	846	846	4064	2200	918	616	686
Sieve	84	53	53	64	37	15	21	

from Talbot Microsystems, and Telos X (from the inventors of Valdocs on the QX-10).

For those of you interested in implementing FORTH-83 on your computer, No Visible Support Software, P.O. Box 1344, Berkeley, CA 94704 is supplying FIG-like disk versions for the 8080 CP/M, 8086 CP/M, and 8086 MSDOS for $25 a copy, an unbeatable bargain. These versions were implemented by Henry Laxen and Mike Perry, both good fellows to know.

The Institute for Applied Forth Research Inc. publishes the Rochester FORTH Conference proceedings. One of their latest publications is *A Bibliography of FORTH References* which lists all magazine/book articles up to 1983 by author and subject. For $15, it's one book that we all should own. You can order it from them at 70 Elmwood Avenue, Rochester, NY 14611 or from Mountain View Press. I picked up my copy at the meeting, and it made the long flight home much more enjoyable.

Forth User's Corner

One of the advantages to buying a dot-matrix printer is the capability of performing dot-addressable graphics. The Epson MX-80 equipped with the Graftrax PROMs is one of the cheapest graphics hardcopy devices available.

Epson Graphics

The MX-80 has two graphics modes: low resolution, 60 dots per inch (480 dots/line), and high resolution, 120 dots/inch (960 dots/line). The vertical spacing is always 72 dots/inch. The high-res mode overlaps dots and makes much nicer-appearing lines. For that reason, we recommend that you use the high-res mode if possible.

Before using the MX-80 in graphics mode, you need to set the line spacing. The line spacing in graphics mode is equal to n/72, where n is the number of pins in the printhead that you are going to be using.

For example, if you can send 8-bit data to the MX-80, n should be 8; if you are using 7-bit serial communication, n will be 7; if you just want to print one line at a time using 1 pin, set n to 1. The word NDOTS, shown in Figure 2, will send the proper sequence to the printer.

When in graphics mode, 8 lines of dots are printed in one pass of the printhead. When you send a byte of data to the printer, the most significant bit controls the topmost pin, and the least significant bit fires the 'bottom' pin (actually, one up from the bottom, since there are 9 pins and only 8 can be used). If you send the printer 'FF' (hex), all 8 pins will be fired; a '01' will just fire the bottommost pin.

To put the Epson in graphics mode, you need to send the sequence:

```
ESC  K   n1  n2
27   72  xx  xx
```

where n1 and n2 are combined by the printer to determine the number of horizontal dots to be printed on the current printhead pass. The algorithm is:

```
m = n1 + 256 * n2
```

(that is, m is a 16-bit integer with its upper and lower bytes separated and sent to the printer as 'characters'). You can specify up to 2047 dots per line, though the MX-80 is limited to 960, so be careful. The word TRAN uses the 16-bit number on top of the stack and tells the Epson to print that number of dot columns.

After this sequence, you enter the actual byte-wide bit patterns for the printhead. When m bytes have been sent to the printer, it will revert back to character mode.

As a simple example of plotting, look at the word BAR. It sends a user-specified number of 192's to the printer. In hex, 192=C0, which means that the upper two pins in the printhead will be fired. BAR will then create a horizontal line 2 dots high and a user-specified number of dots long, when used in conjunction with TRANS to initially set the printer in graphics mode. The word HLINE performs the entire function: sets the line spacing to 8/72 inches, puts the MX-80 in graphics mode, outputs the line, carriage returns, and then resets the line spacing to the default 12/72.

Beyond this, you're on your own. The simplest method of plotting would be to use a single pin and plot one horizontal line at a time. The method used in the UNIFORTH plotting package is to print 8 lines at once by filling a 960-byte buffer with zeroes, calculating each of the 8 lines by OR'ing in the appropriate bits, and then shipping the buffer to the printer. This method is much faster and less wearing on the printhead, but involves a lot more software. Read the graphics chapters in your printer manual for more information.

Figure 2 - Screen of Epson Printer Commands

```
SCR # 10
( SIMPLE EPSON PRINTER PLOTTING )
: NDOTS  ( n --- set line spacing to n/72)
   27 PREMIT 65 PREMIT  PREMIT ;
: TRAN  ( m --- put Epson in graphics mode for m bytes)
   960 MIN 256 /MOD SWAP 27 PREMIT 76 PREMIT PREMIT PREMIT ;
: BAR  ( n --- draw 2-dot horiz line n columns wide)
   192 SWAP 960 MIN 0 DO DUP PREMIT LOOP DROP ;
: PCR  ( send carriage return to the printer)
   13 PREMIT 10 PREMIT ;
: HLINE  ( n --- draw horizontal line)
   8 NDOTS DUP TRAN BAR PCR 12 NDOTS ;

( for the uninformed, PREMIT works just like EMIT)
( except outputs to the printer only)

;S
```

Next issue

We will cover one of the topics dearest to my heart: error handling. Any user contributions will be incorporated into the column with deepest appreciation. Enjoy your spring!

■ ■ ■

HAZELTINE 1410
Brand New

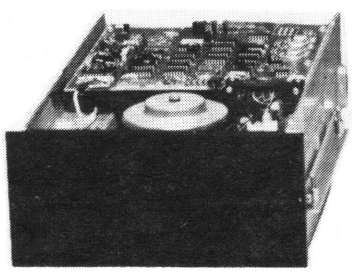
- 24 x 80 characters
- 5 x 7 dot matrix, block cursor
- 64 displayable ASCII characters
- White on black background
- 2048 x 8 Random Access Memory
- EIA RS232C at 110, 300, 600, 1200, 1800, 2400, 4800 or 9600 baud (switch selected)
- Odd, Even, One or Zero (switch selected)
- Half duplex or full duplex (switch selected)
- 15½" wide 13½" high 20½" deep, 28 lbs.
- Fully addressable cursor
- $300.00 f.o.b. our warehouse

HAZELTINE 1421
Brand New

- 12 inch - P4 phos
- 24 x 80 characters
- 5 x 8 dot matrix, block cursor
- 95 displayable ASCII characters
- White on black background, two intensities, blink or blank
- 2048 x 8 Random Access Memory
- EIA RS232C at 110, 300, 600, 1200, 1800, 2400, 4800 or 9600 baud (switch selected)
- Odd, Even, One or Zero (switch selected)
- Half duplex or full duplex (switch selected)
- 15½" wide 13½" high 20½" deep, 28 lbs.
- ADM-3A Emulation
- Consul 580 Emulation
- Fully addressable cursor
- $400.00 f.o.b. our warehouse

HAZELTINE 1500
Brand New

- 12 inch - P4 phos
- 24 x 80 characters
- All 128 ASCII codes
- 94 displayable characters
- Hi-res characters using 7 x 10 dot matrix
- ANSI std. keyboard w/numeric keypad
- Dual intensity - std. & reverse video
- Cursor addressing & sensing
- EIA & 20MA interface
- Auxiliary EIA output
- Baud rates to 19.2KB
- Remote editing commands
- Shipping wt. 40 lbs.
- $450.00 f.o.b. our plant



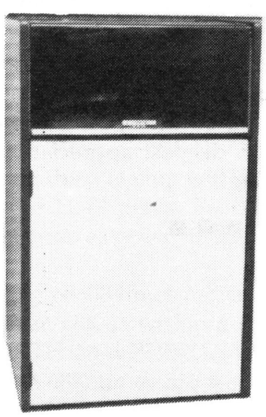
SHUGART 8" DISK DRIVE

- SS/DD
- Model 800-2
- Requires 115VAC (24VDA, + 5VDC, - 5VDC)
- $140.00 (new)
- $100.00 (used) limited useage
- Shipping wt. 16 lbs.
- All prices f.o.b. our warehouse

MFE DISK DRIVE

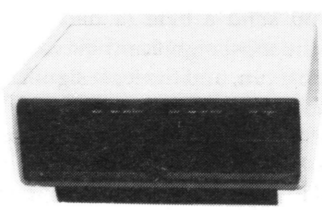
- DS/DD
- Shugart 800 series compatable
- Reconditioned
- Shipping wt. 16 lbs.
- $175.00 f.o.b. our plant

GATES SOUND PROOF CABINET



Brand New

- 40"h x 25"w x 24"d
- Two compartments
- 25"h x 23"w x 21"d (inside lower comp)
- 115 VAC line - fan - elec. conn. & switches in top section
- smoke colored plastic top on hinges
- 155 lbs. shipping wt.
- $125.00 f.o.b. our plant



DUAL DISK DRIVE CABINET

- Fits ALL Shugart 800 series
- 115 VAC motor supply & all cables
- Shipping wt. 30 lbs.
- $50.00 f.o.b. our warehouse.

PRINTERS

DATA ROYAL

- Serial I/O
- Bi-di. - 132 chrs. line
- 120 cps - 7 x 9 dot matrix
- Sprocket feed (2½" to 15")
- 110, 300, 1200 bps
- Shipping wt. 80 lbs.
- $400.00 f.o.b. our plant

CENTRONICS 101A

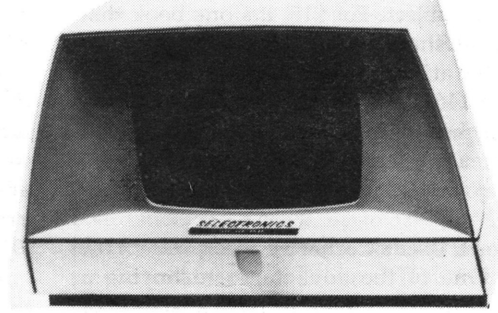
- Centronics parallel
- 164 cps - 7 x 9 dot matrix
- Tractor feed (adj. to 15")
- $225.00 upper case only
- $350. - u/l case & graphics
- All prices f.o.b. our plant

PERKIN ELMER 310

- Letter quality printer terminal
- Microprocessor controlled
- Tractor & friction feed
- 40 cps - 132 char. buffer
- Excellent condition
- Shipping wt. 100 lbs.
- $600.00 f.o.b. our plant

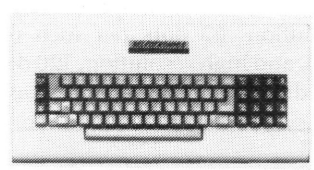
CABLES & MISC.

- DB-25 m/m, m/f, f/f, 10+' - $10.00 ea. 3/$25.00
- DB-25 m/blank 5+' - $ 5.00 ea. 3/$10.00
- "Muffin" fans (4½") - $ 3.00 ea. 6/$15.00
- 3" quiet fan (steel frame) - $ 3.00 ea. 6/$15.00



CONRAC MONITOR

- 9 inch, P4 phos
- 80 x 24 characters
- Composite video in (RCA phono)
- Controls in front panel
- Shipping wt. 30 lbs.
- $45.00 f.o.b. our warehouse



KEYTRONIC KEYBOARDS

- ASCII encoded
- +5 volts
- New - never used
- Shipping wt. 7 lbs.
- $45.00 f.o.b. our warehouse

SELECTRONICS

VISA & MasterCard Accepted

1229 S. Napa Street
Philadelpia, PA 19146
Phone: (215) 468-4645 • (215) 468-7891

Pa. residents add 6% sales tax. All prices f.o.b. our warehouse. All products carry a replacement warranty. All merchandise accurate as to description to the best of our knowledge.