

THE COMPUTER JOURNAL

For Those Who Interface, Build, and Apply Micros

Vol II. No. 5

Issue Number 9

\$2.50 US

**Reading PC DOS Diskettes
with the Morrow Micro Decision** page 2

Write Your Own Threaded Language pages

Interfacing Tips and Troubles:

Build a DC-to-DC Converter page 12

Multi-user:

C-NET page 14

LSTTL Reference Chart page 17

DOS Wars Oakc20

Building a Code Photoreader page 23

Editor's Page

Formulating Our Product Review Policy

Recently we received a letter that said:

"I don't think you should do reviews for new products. This is what has hurt all the other magazines—their need to be current, trendy, with-it. *The Computer Journal* should be behind the times by supporting the end user after he or she has bought their system and then says "What Now?"

It was a thought-provoking letter with some good points, and it made the staff here at *The Computer Journal* think a lot about our then-unstated product review policy.

We feel that one of the purposes of a journal is to bring information on new product developments to the attention of the readers, but it is true that some magazines have reached the point where new product releases and product reviews take up the majority of the space.

There is a difference between a product announcement, which is usually written from the vendor's literature, and a product review, which requires working with the product in an actual application long enough to find out how it really performs. A so-called "review" written from the manual with a quick run on the computer is useless.

Our product information should follow in the same vein as the majority of our articles by providing specific technical information that will be of use to those who "build, interface, and apply micros." As a general rule, if it is covered in *Byte* and *Info World*, we do not need to rehash the same information by covering it in *The Computer Journal*.

The areas we intend to cover in our product reviews are utilities, languages, sensors, boards, interfaces, peripherals, etc. We may occasionally report on products which have been covered elsewhere, but we will concentrate on the technical aspects of the products and not just repeat what you have already read.

A good review involves a lot of work, and we don't have the time to spend on products which do not interest our readers. We would appreciate your feedback on the subject of product reviews. Some of

the products we have for review are: Conix (a CP enhancement), ACNAP (an AC circuit analysis program), PLOTPRO (a plotting program which works with ACNAP), SPP (a signal processing program), FORTH-83, and Condor-3.

Tell us which ones we should eliminate, and suggest items that you think we should add to our list. We would also like to incorporate your experience with the product into the reviews, or use them as added comments in a future issue.

Our policy henceforth will be to review technical products which are of interest to our readers, but to avoid reviewing systems and general interest items which are covered in other publications. We don't want to use an excessive amount of space for new product announcements and product reviews, but since other publications are not presenting the detailed technical reviews our readers demand, we will concentrate on presenting needed information.

Editor/Publisher..... Art Carlson
Art Director..... Joan Thompson
Technical Editor..... Lance Rose
Production Assistant..... Judie Overbeek
Contributing Editor..... Ernie Brooner

*The Computer Journal** is published 12 times a year. Annual subscription is \$24 in the U.S., \$30 in Canada, and \$48 airmail in other countries.

Entire contents copyright © 1984 by *The Computer Journal*

Postmaster: Send address changes to: *The Computer Journal* P.O. Box 1697, Kalispell, MT 59903-1697.

Address all editorial, advertising and subscription inquires to: *The Computer Journal*, P.O. Box 1697, Kalispell MT 59903-1697.

READING PCDOS DISKETTES WITH THE MORROW MICRO DECISION

by Lance Rose, Technical Editor

With all the different 5¹/₄ inch diskette formats around today, it's not surprising that there is little compatibility from one manufacturer's equipment to another, even within the realm of CP/M systems. Add to this the fact that aside from CP/M systems there is the IBM PC and its lookalikes with their own operating system known as PCDOS, and you have a real mess.

Here at *The Computer Journal* we are constantly faced with the problem of potential articles being submitted on a variety of diskette formats and not always having the right machine around to read them on. Fortunately, we do have an 8-inch CP/M system and an Apple which helps quite a bit. Nevertheless, until recently we have been dependent on other facilities to read any diskette that came in written on a PC.

Recently we acquired a Morrow Micro Decision with double-sided minifloppy drives. Morrow has thoughtfully provided several utility programs with the system that can enable the machine to read diskettes made on the Osborne, Xerox and IBM disk systems. The catch is that only the CP/M-86 format is supported for the PC. Unfortunately for us, PCDOS is the de-facto standard for the PC and almost no one uses CP/M-86 on it. The question was, how do we use this utility to read files from a PCDOS diskette?

Luckily, it turns out that PCDOS and CP/M-86 for the IBM PC both use the same type of physical diskette formatting, namely 512-byte sectors. Depending on which version of PCDOS we're talking about, there are eight (Version 1) or nine (Version 2) sectors per track. In addition, for the double-sided formats of PCDOS, the diskette is organized in cylinders. This means that track 0 side 0 is used first, then track 0 side 1, followed by track 1 side 0, etc. Fortunately, this is the same arrangement used in the Micro Decision double-sided format. By simply running the [IBM.COM](#) utility to set drive B to an IBM drive and changing a byte in the drive description table we can also read double sided PCDOS diskettes.

With the physical format compatibility problems handled by [IBM.COM](#), we can turn our attention to the logical layout of PCDOS diskettes and how they differ from CP/M diskettes. In CP/M, each directory entry is 32 bytes long and contains the user number, file name and type, number of 128-byte records in the file, and a list of up to 8 "allocation block numbers" which point to the areas of the diskette where the file is to be found. Since in many systems these blocks are 2K bytes each, the directory entry can represent files up to 16K in size. The problem here is that if the file is over 16K bytes long, we need a second directory entry to hold the additional allocation block numbers for the file. For example, a file that was 87K long would require 6 entries

(192 bytes) in the directory to represent it.

This isn't a very efficient use of directory space and it may be one reason that Microsoft chose a different directory format for their MSDOS/PCDOS operating system. In this system, there is a file allocation table (FAT) on track 0 of each diskette. This table contains linked lists of the sectors allocated to each file. The directory, which is located on track 0 immediately following the FAT, consists of 32-byte entries, like CP/M. Each entry contains the file name and type, file size in bytes, and a pointer into the file allocation table. This pointer identifies the first sector allocated to a file. The succeeding pointers are in the FAT itself, of course. The advantage here is that only two bytes are necessary in the directory entry itself to point to the linked list as opposed to 16 bytes for each CP/M entry. There are two immediate advantages to this. First, the additional bytes in the 32-byte directory entry can be used for time and date stamping which PCDOS supports. Secondly, only one entry is required no matter how large the file.

With this in mind, and the PCDOS manual in hand, one can read the FAT and directory into memory, display the contents of the directory, and/or read a file from the diskette by following the linked list of allocated sectors using the mapping algorithm described in the PCDOS manual.

Listing 1 is an assembly listing of a program that will read a PCDOS directory and display its contents in a format similar to the CP/M DIR command. In addition, the size of each file is also displayed, rounded up to the nearest 1k bytes. This allows the user to examine the PCDOS diskette in drive B and see what files, if any, he wants to transfer to the CP/M diskette in drive A. The program is run by typing: DIRPC (return).

Listing 2 is an assembly listing of a program that searches the PCDOS directory for a given file and then, if found, copies it to the CP/M diskette in drive A. For example, if the user wants to copy a file named LETTER.TXT, he would type: READPC LETTER.TXT (return). A message is displayed if the file isn't found; otherwise the copy is made. When finished reading PCDOS diskettes, the program [MORROW.COM](#) can be run to set drive B back to a Morrow drive.

These programs will work on both the Micro Decision MD-2 (single sided drives) and MD-3 (double sided drives) as long as the PCDOS diskette is single sided. In addition, if you have an MD-3, the program will detect double-sided PCDOS diskettes and read them as well. We plan to make good use of it here at *The Computer Journal* office.

Although not our immediate goal, it would also be possible to go the other way, i.e. transfer CP/M files to a

PCDOS diskette. To do this, one would have to search the FAT for empty sectors and build the linked list as the file is transferred. Should any of our ambitious readers come up with such a program, we would very much like to hear about it. (Listings 1 and 2 follow.)

Listing 1

```

; DIRPC.ASM
; Program to display directory and file size of an
; IBM PCDOS diskette in Drive B of a Morrow Micro
; Decision, assuming IBM.COM has already been run
; to set Drive B to an IBM type diskette
; Version of 4/24/84
;
BOOT EQU 0000B ;CP/M reboot address
BDOS EQU 0005H ;(BDOS entry point
TBUFF EQU 0080B ;Transient disk buffer
;
;
ORG 100H
;
LXI SP,STACK ;Set up local stack
LDA BOOT*2
STA SELDSK+2 ;Patch for BIOS calls
STA SETTRK+2
STA SETSEC+2
STA SETDMA+2
STA REACH 2
MVI C, 1
CALL SELDSK ;Select P drive
MVI C, 0
CALL SETTRK ;Set track 0
MVI C, 5
CALL SETSEC ;Set logical sector 5
CALL REAP ;Read first part of FAT
ORA A
JNZ BOOT ;Read error
LHLD BOOT+1 ;Locate drive table (MTAB)
LX 1 D, 41H
DAD D
MOV E, M
I NX H
MOV D, M
LXI H, 9
DAD D ;HL points to drive B entry
LX 1 D, FMterr ;In case unreadable
LDA TBUFF ;Get first byte of FAT
LXI B, 1C0DH ;Sector count 6 first sector
INR A
JZ DOUBLE ;DOS 1.0 double sided
MVI B, 10H
A
INR A
JZ SINGLE ;(DOS 1.0 single sided
LXI B, 1C15B
INR A
JZ DOUBLE ;DOS 2.2 double sided
MVI B, 10H
A
INR A
JNZ EXIT ;Not DOS 2.0 single sided
SINGLE: I NX B
MOV A, M
AN I 0FFH ;Set media bit to single sided
MOV M, A
JMP RDDIR
DOUBLE: MOV A, M
ANI 20H ;Test double sided drive bit
JZ EXIT ;Single sided drive, no read
I NX H
MOV A, M
ORI 84H ;Mark media double sided
MOV M, A
RDDR1: LXI H, DIRBUF ;Point at directory buffer
RDDR1: PUSH B ;Save sector
PUSH H ;Save DMA address
CALL SETSEC
POP B
PUSH B
CALL SET DMA
CALL READ
POP H
POP B
ORA A
JNZ BOOT ;(Exit if disk error
PUSH H
LXI H, -0920H
DAD B
MOV A, H
ORA L
JZ RDDR2 ;Go to next side
LXI H, -0D24B
DAD B
MOV A, H
ORA L
JNZ RDDR3 ;(Stay on same side
RDDR2: PUSH B
MVI C, 1 ;Go to track 1
CALL SETTRK
POP B
MVI C, 0 ;(Start with sector 1
RDDR3: POP A
LXI D, 128
DAD D ;(Go to next DMA address
INR C ;Went sector
DCR B ;Decrease sector count
JNZ RDDR1
MOV A, H ;(Calculate directory size
LXI B, DIRBUF
SUB H
ADD A ;8 entries per page
ADD A
ADD A
MOV C, A ;Marimum entries in C
NXTENT: CPI A, M
JC ;Not printing ASCII
CPI 7FH
JNC EMPTY ;Same, assume empty
PUSH B ;(Save count
PUSH H ;(Save pointer

```

```

MOV A, B
ANI 83B
JNZ MIDLIN
LXI D, CRLP
CALL DSPLY ;Do a CRLP at beginning
JMP DSPEM ;(Skip separator
MIDLIN: MVI E, ' ' ;(Display separator between
CALL WRTCON
CALL SPACE ;(Add a space
DSPENT: POP H ;(Restore entry pointer
PUSH H
MVI C, 8 ;Filename characters
CALL NAME ;(Display filename
CALL SPACE ;(Separate with space
MVI C, 3
CALL NAME ;(Display filetype
CALL SPACE ;(Trailing space
LXI D, 17
DAD D ;(Point at file size
MOV E, M ;(Get size in registers
I NX H
MOV D, M
INX H
MOV A, M
LXI H, 3FFH ;(Round up to nearest 1k
DAD D
ACT 0
MOV L, H ;(Shift right 10 bits
MOV A, A
MVI C, 2
SHIFTR: XRA A
NOV A, H
RAR
MOV H, A
MOV A, L
RAR
MOV L, A
DCR C
JNZ SHIFTR
XRA A
STA NONZERO ;Non-zero flag
LXI D, -166
CALL DIGIT ;(Display hundreds
LXI D, -10
CALL DIGIT ;(Display tens
MOV A * L
ADI '0'
MOV E, A ;(Display ones
CALL WRTCON ;(Add 'k' to size
MVI E, 'k'
CALL WRTCON
CALL SPACE ;(Add a trailing space
POP H
POP B
INR B
EMPTY: LXI D, 32 ;Go to next entry
DAD D
DCR C ;(More entries to check
JNZ NEXTENT
MOV A, B
ORA A
LXI D, NOFILE ;No entries
JZ EXIT ;(At least one entry
LXI D, CRLP ;(Display message
CALL DSPLY ;(Reboot
DSPLY: MVI C, 9 ;(Display string function
JMP BDOS
SPACE: MVI E, ' ' ;(Display space
WRTCON: PUSH B
PUSH H
MVI C, 2 ;(Write console function
CALL BDOS
POP H
POP B
RET
NAME: MOV E, M
CALL WRTCON
INX A
DCR C
JNZ NAME
RET
DIGIT: MVI A, '0' ;(Initialize digit
DIGIT: DAD D ;(Divisor in DE
INR A ;(ASCII digit in A
JC DIGIT1
PUSH PSW ;(Save digit
MOV A, L
SUB E ;(Correct for overflow
MOV I, B
MOV A, H
DAD D
MOV H, A
POP PSW
MOV E, A ;(Digit in E
CPI '0' ;(Zero in this place
JZ DIGIT2
MVI A, 0FF8
STA NONZERO ;(Set flag
JMP WRTCON
DIGIT2: LDA NONZERO
ORA A
JNZ WRTCON ;(Display embedded zero
MVI E, ' ' ;(Suppress leading zeros
JMP WRTCON
SELDISK: JMP 001BH ;(Values are filled in
SETTRK: JMP 001EH
SETSEC: JMP 0021R
SET DMA: JMP 0024H
READ: JMP 00278
;
; FMterr: DB ;*** UNREADABLE PCDOS FORMAT ***
; DB 0DH, 0AH, 07H, ' $ '
; NOFILE: DB BDH, 8AH, 'NO FILE '
; CRLP: DB SDH, BAH, ' $ '
; NONZERO: DS 1 ;(Flag for non-zero value
; DS 48 ;(Stack area
; EQU $
;
; ORG ($+BFF) AND 0FF00H
;
; DIRBUF: DS 7*200R ;(PCDOS directory
;
END

```

Listing 2

```

; READPC.ASM
; Program to copy a file from an IBM PCDOS diskette
; in Drive B to a CP/M file on Drive A of a Norrow
; Micro Decision, assuming IBM.COM has already
; been run to set Drive B to an IBM type diskette
; Version of 4/24/84
;
;
; BOOT EQU MBBS (CP/M reboot address)
; BDOS EQU BBSH (BDOS entry point)
; DSTFCB EQU BBSCH (Destination FCB)
;
;
; ORG 186H
;
; LXI SP, STACK (Set up local stack)
; LDA BOOT*2
; STA SELDSK*2 (Patch for BIOS calls)
; STA SETTRK*2
; STA SETSEC*2
; STA SETDMA*2
; STA READ*2
; MVI C, 1
; CALL SELDSK (Select B drive)
; MVI C, B
; CALL SETTRK (Set track 0)
; LXI B, B4B5H (Read FAT)
; LXI H, FAT
; RDFAT: PUSH B (Save sector)
; PUSH H (Gave DMA address)
; CALL SETSEC
; POP B
; PUSH B
; CALL SET DMA
; CALL REAL
; POP H
; POP B
; ORA A
; JNZ BOOT (Reboot if disk error)
; LXI D, 126
; DAD D ;Go to next DMA address
; INF C (Next sector)
; DCR B (Decrease sector count)
; JNZ RDFAT
; LHLD BOOT+1 ;*Locate drive table (MTAB)
; LXI D, 41H
; DAD D
; MOV E, M
; INX H
; MOV D, M
; LXI H, 9
; DAD D
; XCHG ;DF points to drive P entry
; LDA FAT (Get first byte of FAT)
; LXI B, 1C0DH (Sector count 4 first sector)
; LXI H, B608H (CLOFF and SPT)
; INR A
; JZ DOUBLE ;D05> 1.8 double sided
; MVI B, 1BH
; LXI H, B110H
; INN A
; JZ SINGLE (DOS 1.0 single sided)
; LXI B, 1C15H
; LXI H, 08B9H
; INF A
; JZ DOUBLE (DOS 2.0 double sided)
; MVI B, 10H
; LXI H, B212H
; INR A
; JZ SINGLE ;DOS 2.0 single sided
; NOREAD: LXI D, FMTEKR ;Unreadable
; SINGLE: JMP EXIT
; INX D
; LDAX D
; ANI 0FBH ;Set media bit to single sided
; STAX D
; MVI A, 03H (Mask)
; JMF RDDIR
; DOUBLE: LDAX D
; ANI 20H ;Test double sided drive bit:
; JZ NOREAD (Single sided drive, no read)
; INX D
; LDAX D
; ORI 04H (Mark media double sided)
; STAX D
; MVI A, 07H
; RDDIR: STA CLMASK (Double sided mask)
; SHLD SPT (Save format info)
; LXI H, DIRBDF ; Point at directory buffer
; RDDIR1: PUSH B (Save sector)
; PUSH H (Save DMA address)
; CALL SETSEC
; POP B
; PUSH B
; CALL SET DMA
; CALL READ
; POP H
; POP B
; ORA A
; JNZ BOOT (Exit if disk error)
; PUSH B
; LXI H, -B92BH
; DAD B
; MOV A, H
; ORA A
; JZ RDDIR2 ;Go to next side
; LXI H, -0D24H
; DAD B
; MOV A, H
; ORA L
; JNZ RDDIR3 (Stay on same side)
; RDDIR2: PUSH B
; MVI C, 1 (Go to track)
; CALL SETTRK
; POP B
; MVI C, 0 ; Start with sector 1
; RDDIR3: POP H
; LXI D, 128
; LXI D, 128
; DAD D (Go to next DMA address)
; INR C (Next sector)
; DCR B (Decrease sector count)
; JNZ RDDIR1
; NOV A, B (Calculate directory site)
; LXI B, DIRBUF
; SUB H
; ADD A (B entries per page)
; ADD A
;
;
; ADD A
; MOV B, A ;Mazimus entries in B
; LXI D, DSTFCB+1 (Point at filename)
; SRARCH: PUSH D (Save pointers)
; PUSH B
; MVI C, 11 ;Length of filename/type
; COMPAR: LDAX D
; CMP M
; JNE DIFFER ;Not the same
; D INX D
; IMX H
; DCR C
; JNZ COMPAR
; JMP FOUND (Entry matched)
; DIFFER: POP B
; LXI D, 32
; DAD D ;Go to next entry
; POP D
; DCF B
; JNZ SEARCH
; LXI D, FNFERR ; (Not found)
; JMP EXIT
; FOUND: POP D
; XTHL
; MVI C, B
; CALL SELDSK
; LXI D, DSTFCB
; MVI A, 1
; STAX D (Destination is always A)
; PUSH D
; MVI C, 19
; CALL BDOS (Delete old file if present)
; POP D
; MVI C, 22
; CALL BDOS (Make new file)
; A
; LXI D, DDFERR
; JZ EXIT (Disk full)
; POP H (Restore pointer)
; LXI D, 15
; DAD D (Point to 1st cluster)
; MOV E, M
; INX H
; MOV D, M
; INX H
; SHLD CLUSTR (Save next cluster)
; XCHG
; MOV E, M ;Get file size
; INX B
; MOV D, M
; INX H
; NOV A, M
; LXI H, 7PH ;*Convert to logical records
; DAD D
; ACI 6
; DAD H
; RAL
; MOV L, H
; MOV H, A
; LXI D, DATABUF (Record count in HL)
; MVI C, 0 (Buffer pointer in DE)
; LOOP: MOV A, H (Buffer records in C)
; ORA L
; JNZ MORE ;More records in file
; CALL FLUSH ;Flush data buffer
; LXI D, DSTFCB
; MVI C, 16
; CALL BDOS ;Close file
; JMP BOOT ;Done
; MORE: PUSH B ;Save parameters
; PUSH B
; PUSH D
; LDA CLMASK ;Get mask
; ANA C
; JNZ CURRNT ;Disk and track current
; LHLD CLUSTR
; XRA A
; MOV A, H
; RAR
; MOV D, A
; MOV A, L
; RAR
; MOV E, A
; PUSH PSW (Save odd/even status)
; DAD D
; LXI D, FAT
; DAD D
; MOV E, M
; INX H
; MOV D, M (DE have next cluster number)
; POP PSW
; MVI C, 4 (Bit count)
; JC ODD (Previous cluster was odd)
; MOV A, D
; ANI BFH ;Reep only lowest 12 bits
; NOV D, A
; JMP UPDATE
; ODD: XRA A (Clear carry)
; MOV A, D (Keep highest 12 bits)
; RAR
; MOV D, A
; MOV A, E
; RAR
; MOV E, A
; DCR C
; JNZ ODD
; UPDATE: LHLD CLUSTR (Get current cluster)
; XCHG
; SHLD CLUSTR (Replace with next cluster)
; LHLD CLOFF
; MVI H, 0
; XCHG
; DAD H (Cluster*?)
; DAD D (Compute track/sector from BL)
; LDA SPT
; CMA
; INR A
; NOV E, A
; MVI D, BFFB
; NOV C, D
; DIVIDE: DAD D (Divide HL by apt)
; INR C (Track in C)
; JC DIVIDE
; LDA CLMASK
; IMR A
; RC

```

continued

Listing 2, continued

```

RRC      B,A
NOV      A,L
NOV      E
MULT:   ADD      ;Remainder in A
NOV      A      ;Multiply by mask*1
NOV      E,A
NOV      A,B
RRC      B,A
NOV      A,E
NOV      A
JNC      MULT
10       A      ;First sector is 91
STA      SECTOR  ;Update sector
NOV      A,C
STA      TRACK   ;Update track
CALL     SETTRK  ;Set new track
MVI      C,2
CALL     SELDSK
CURRMT: LDA     SECTOR ;Select next sector
CFI      37      ;No track overflow
JC       NOOVF
LDA      TRACK
I NR     A
NOV      C,A
CALL     SETTRK ;Go to next track
MVI      A,1    ;Start with sector 1
NOOVF:  MOV     C,A
IBR      A
STA      SECTOR
CALL     SETSEC
POP      B
PUSH     B
CALL     SETDMA ;Set next DMA address
CALL     READ
ORA      A
JNZ      BOOT   ;Read error
POP      D
LX 1    H,128
DAD      D      ;Go to next buffer area
XCHG
POP      H
DCX      B      ;Restore record counter
POP      B
INR      C
CM       FLUSH  ;Flush data buffer if 128
JMP      LOOP
FLUSH:  INR     C ;Flush data buffer
DCR      C
R2      H
PUSH     B
PUSH     C,8
CALL     SELDSK
POP      B
LX 1    D,DATBUP
PUSH     D
FLUSH1: PUSH     D
MVI      C,26
CALL     BDOS ;Set DMA address
LXI     D,DSTPCB
MVI      C,21
CALL     EDOS ;Write the record
LXI     D,DDPERR
ORA      A
JNZ      EXIT  ;Disk full
POP      D
POP      B
LXI     H,128
DAD      D      ;Go to next record
XCHG
DCR      C
JNZ     FLUSH1
POP      D
POP      H
RET
EXIT:   MVI     C,9
CALL     BDOS ;Display message
JMP      BOOT ;Reboot
SELDISK: JMP    0010H ;Values are filled in
SETTRK:  JMP    0010H
SETSEC:  JMP    0021H
SETDMA:  JMP    0024H
READ:    JMP    0027H
;
FMERR:  DB     '*** UNREADABLE PCDOS FORMAT ***'
DB
FWERR:  DB     '*** FILE NOT FOUND ***.SDH,BAH,07H, '$'
DDPERR: DB     '*** DISK OR DIRECTORY FULL ***'
DB
SPT:    DS     1 ;2*Sectors/track
CLOFF:  DS     1 ;2*Cluster offset
CLMASK: DS     1 ;Cluster mask
CLUSTR: DS     2 ;Next cluster
TRACK:  DS     1 ;Current logical track
SECTOR: DS     1 ;Next logical sector
STACK  EQU 64 ;Stack area
EQU     $
;
ORG     ($+PPH) AND 0FF00B
;
FAT:    DS     1*2608 ;File allocation table
DIRBUP: DS     79206H ;PCDOS directory
DATBUP: DS     40008 ;Data buffer
;
END

```

Customer Support Survey

In order to improve customer support in the microcomputer industry, TCJ will publish user experiences with vendors. Send us your candidates for the best and worst vendor, along with your supporting information.

MicroMotion

MasterFORTH

It's here — the next generation of MicroMotion Forth.

- Meets all provisions, extensions and experimental proposals of the FORTH-83 International Standard.
- Uses the host operating system file structure (APPLE DOS 3.3 & CP/M 2.x).
- Built-in micro-assembler with numeric local labels.
- A full screen editor is provided which includes 16 x 64 format, can push & pop more than one line, user definable controls, upper/lower case keyboard entry, A COPY utility moves screens within & between lines, line stack, redefinable control keys, and search & replace commands
- Includes all file primitives described in Kemigan and Plauger's Software Tools.
- The input and output streams are fully redirectable
- The editor, assembler and screen copy utilities are provided as relocatable object modules. They are brought into the dictionary on demand and may be released with a single command.
- Many key nucleus commands are vectored. Error handling, number parsing, keyboard translation and so on can be redefined as needed by user programs. They are automatically returned to their previous definitions when the program is forgotten.
- The string-handling package is the finest and most complete available.
- A listing of the nucleus is provided as part of the documentation.
- The language implementation exactly matches the one described in FORTH TOOLS by Anderson & Tracy. This 200 page tutorial and reference manual is included with MasterFORTH
- Floating Point & HIRES options available.
- Available for APPLE 11/II+/Ile & CP/M 2.x users.
- MasterFORTH-\$100.00. FP & HIRES-\$40.00 each
- Publications
 - FORTH TOOLS - \$20.00
 - 83 International Standard - \$15.00
 - FORTH-83 Source Listing 6502, 8080, 8086 - \$20.00 each.



=V Contact:

MicroMotion
12077 Wilshire Blvd., Ste. 506
Los Angeles, CA 90025
(213) 821-4340

WRITE YOUR OWN THREADED LANGUAGE

by Douglas Davidson

The FORTH language presents a method remarkable for its simplicity, economy, and power; it should be remembered, however, that FORTH proper is but one example of the method referred to as *threaded language*. The characteristics of threaded languages make them particularly useful to hobbyists and all those who like to get close to the machine they are working with; it is my contention that the project of writing such a language is useful, educational, very much in the spirit of the language, and at the same time not very difficult. I will present in these articles a practical guide to the writing of one particular threaded language with the intention that it be customized and altered at will. This language is very similar to FORTH, but varies from it in a number of ways for the purposes of simplification.

A threaded language consists essentially of a collection of tools and some facilities for using and adding to these tools. The basic tools are simple routines to do things such as simple arithmetic operations, input and output, etc. The tools communicate by means of a LIFO (last in, first out) stack which forms the heart of the apparatus; most routines are defined in terms of their effects on the stack, and arithmetic operations are presented in RPN form (for an explanation of RPN, see the tint box on page 30.) The power of the language comes from its extensibility: new routines are defined in terms of old ones, and then become part of the tool collection, and can in turn be used in the definition of still more routines. This is the origin of the term "threaded:" routines are stored essentially as lists of references to other routines, which are "threaded" together. Before introducing specifics, I would like to present a list of definitions.

- **Word:** A named routine, consisting of a header followed by machine language code or data of some sort. See Figure 1.
- **Primary:** A word written in machine language; more generally, a word without a higher-level definition.
- **Secondary:** A word defined in higher-level terms—that is, in terms of other words. In this presentation the distinction between primary and secondary will be somewhat blurred by the fact that secondaries will actually consist of machine code, but this is not always the case, and the difference will

still be clear.

- **Dictionary:** The linked list of words. Words will be stored one after the other in memory; each one will contain a link to the previous one, and a record will be kept of the location of the last one.
- **Header:** The housekeeping information at the head of a word, here consisting only of the word's name and a link to the previous word in the dictionary. The name will be stored as its first three characters in ASCII form, preceded by its length—this is somewhat standard, and should be sufficient. The link follows the name and is two bytes, the NFA of the previous word.
- **NFA:** The name field address of a word—the address of the first byte of the header.
- **LFA:** The link field address of a word—the address of the fifth byte of the header, equal to NFA + 4.
- **PFA:** The parameter field address of a word—the address of the first byte following the header, equal to NFA + 6.

I wish to present the language from the bottom up; I will start with the simplest routines, the ones that must be coded first, and proceed through the more complex, developing details as needed. The routines will be presented in several ways—a functional description will be given in the text, and the 6502 assembly and machine code for an Apple II implementation will follow. The presentation here may be conformed to on at least four different levels: the machine code level, preserving all memory locations, and using an Apple; the assembly level, for another 6502 computer; the logic of the routines, translated into a different assembly language; or the functional descriptions of the routines. Obviously, it will be more of a personal language and easier to customize if it conforms less strictly to what is presented here, but the choice is yours. What is presented here is a bare-bones version, with all of the interfaces specified and the bugs ironed out. The memory map, at least for now, will be relatively simple. The dictionary will start near the bottom of memory and grow up; the stack will start near the top of memory (some room will be needed above it) and grow down; see Figure 2. Some space will also be needed for system variables. We will need a two-byte variable, S, to point to the head of the stack; as

each stack entry will consist of two bytes, S will be incremented or decremented in multiples of two. I now postpone details of more complex things to get right to some basic routines.

The first necessity is a

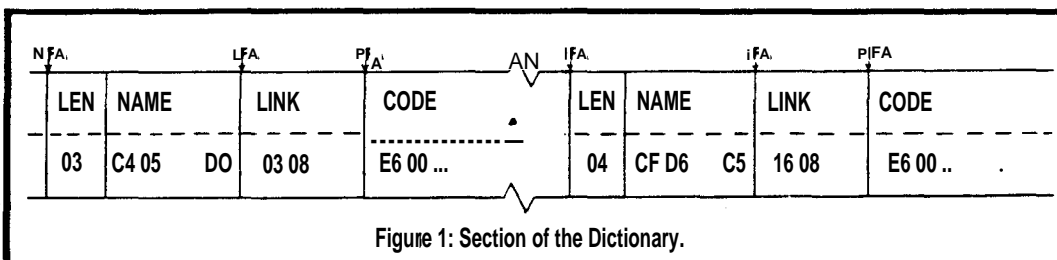


Figure 1: Section of the Dictionary.

start-up routine, but since we do not yet know all that needs to be done, it would be best to simply put a jump instruction at the start of the program and leave its destination for later. Immediately after that we start adding words to the dictionary; each consists of a header —one byte of name length, three bytes of name, and two bytes of link— followed by the machine code, ended with a return instruction. The link of the first word should be zero.

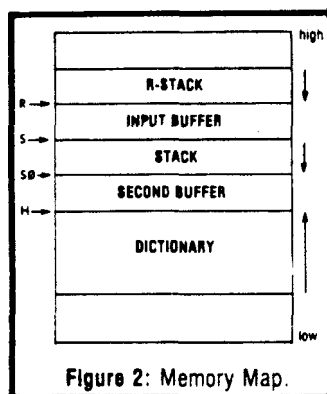


Figure 2: Memory Map.

Stack Manipulation:

DROP removes the top stack element; it simply increments *S* by two.

DUP duplicates the top stack element; it decrements *S* by two, then moves the two bytes starting at *S* + 2 to the two bytes starting at *S*.

OVER places a copy of the second stack entry on top of the stack; it decrements *S* by two, then moves the two bytes starting at *S* + 4 to the two bytes starting at *S*.

SWAP exchanges the top two stack elements; it exchanges the two bytes starting at *S* with the two bytes starting at *S* + 2.

ROT cyclically exchanges the top three stack elements; it exchanges the two bytes starting at *S* + 2 with the two at *S* + 4, then the two at *S* with the two at *S* + 2.

> R It will be useful for several reasons to maintain a second stack, the "return" stack, similar to but smaller than the first, and starting at the top of available memory and growing down. A two-byte variable, *R*, will point to the head of the return stack. The word **> R** moves the head of the main stack to the return stack; it decrements *R* by two, moves the two bytes at *S* to the two at *R*, and increments *S* by two.

R> This word is the opposite of **> R**; it decrements *S* by two, moves the two bytes at *R* to the two at *S*, then increments *R* by two.

I copies the head of the return stack to the main stack; it decrements *S* by two, then moves the two bytes at *R* to the two at *S*.

Comparison:

0= We will use a "flag" to indicate logical values; \$0001 = true, \$0000 = false. All comparison operators will return these values, but for other purposes any non-zero value will be considered true. Also, it is conventional for words to destroy their operands; thus, the word **0** checks the value at the head of the stack and replaces it with a true flag if it is zero, or a false flag otherwise.

0< Stack values will be considered for different purposes as logical values, as two's complement values, as ASCII values, or as unsigned binary values. The word **0<** takes the top stack value as a two's complement signed integer; it

replaces the top stack entry with a true flag if its most significant bit is high, otherwise with a false flag.

= Words, as I have said, destroy their operands; the word **=** determines a flag, a true flag if the top two stack entries (i.e., the two bytes at *S* and the two bytes at *S* + 2) are equal and a false flag otherwise, then increments *S* by two and replaces the two bytes starting at *S* by the flag previously determined.

< considers the two top stack entries as two's complement signed binary integers and determines a flag, true if the first one (the two bytes starting at *S* + 2) is less than the second (the two bytes starting at *S*) and false otherwise, then increments *S* and replaces the two bytes starting at *S* with the flag previously determined. **>** is similar to **<**, if "less than" is replaced by "greater than".

Logical:

AND forms this bitwise logical AND of the two bytes starting at *S* with the two bytes starting at *S* + 2, increments *S* by two, then replaces the two bytes starting at *S* with the result of the logical operation.

OR is similar to **AND**, but the operation is an OR.

XOR is similar to **AND**, but the operation is an exclusive-or.

NOT replaces the two bytes starting at *S* with their logical complement.

Arithmetic:

+ finds the sum of the two top stack entries, increments *S* by two, and replaces the two bytes starting at *S* with the sum.

- is similar to **+**, but the difference rather than the sum is calculated; note that no overflow checking is performed.

NEGATE replaces the top stack entry by its two's complement additive inverse; essentially it exclusive-ors the two bytes with \$FFFF and then increments them by one.

ABS replaces the top stack entry by its absolute value; if the MSB is high, it calls **NEGATE**, otherwise it does nothing.

***** It is wise, for the multiplication and division words, to reserve about ten bytes for accumulators, and to have headerless routines (a) to move stack values into these accumulators, taking their absolute values first and saving the final sign elsewhere, (b) to move a value from the accumulator to the stack, giving it the saved sign, and (c) to perform operations on the accumulators. The ***** word takes the two top stack entries, considered as two's complement binary integers, removes them from the stack, forms their product, and places it on the stack.

/ is similar to *****, but it forms the quotient (of the first value divided by the second) rather than the product.

/MOD considers the two top stack values as unsigned binary integers, takes them off the stack, forms their quotient and remainder, and puts first the remainder, then the quotient on the stack.

MOD is similar to **/MOD**, but it discards the quotient and returns just the remainder.

***/** takes the three top stack entries, considered as two's

complement binary integers, off the stack, forms the 32-bit product of the first two, and divides this 32-bit product by the third. It then places the resulting 16-bit value back on the stack.

/MOD is similar to */, but it considers the entries as unsigned binary values, and it returns first a remainder and then a quotient.

Memory:

! takes the value in the two bytes starting at S + 2 and stores it in the two-byte location pointed to by the two bytes starting at S; it then increments S by four.

+ ! takes the value in the two bytes starting at S + 2 and adds it to the two-byte value pointed to by the two bytes starting at S, then stores the sum in the location pointed to by the two bytes starting at S; it then increments S by four.

C! takes the value in the one byte starting at S + 2 and stores it in the one byte pointed to by the two bytes starting at S; it then increments S by four.

@ replaces the two bytes starting at S with the two bytes they point to.

C@ replaces the two bytes starting at S with the one byte they point to and a \$00 for the upper byte.

The next installment will present more complex routines, including most of the input-output and dictionary management. For now, test each of these simple routines independently, and perhaps add some more — anyone familiar with FORTH will know several more. The purpose of some of these routines will become clearer as things progress.

```

*****
** CODE FOR SAMPLE THREADED LANGUAGE **
** APPLE/11650? **
*****
0000: 4c 16 13 JMF STARTUP
*
** DROP **
*
0003: 04 C4 D2 CF 00 00 INC SL ; increment S by 2
0009: E6 00 BNE 00
000B: D0 02 BNE 00
000D: E6 01 INC SH
000F: E6 00 OK1 INC SL
0011: D0 02 BNE 00
0013: E6 01 INC SH
0015: 60 OK2 RTS
*
** DUP **
*
0016: 03 C4 D5 D0 03 00 INC SL ; decrement S by 2
001C: E6 00 DEC SL
001E: C6 00 BNE 00
0020: D0 02 BNE 00
0022: C6 01 DEC SH
0024: C6 00 OK1 DEC SL
0026: D0 02 BNE 00
0028: C6 01 DEC SH
002A: C6 00 OK2 DEC SL
002C: A0 03 LDY 0003 ; move the two bytes
002E: B1 00 LDA (S),Y ; at S+2
0030: A0 01 LDY 0001 ; to the two bytes
0032: 91 00 STA (S),Y ; at S
0034: C8 INY
0035: B1 00 LDA (S),Y
0037: A0 00 LDY 0000
0039: 91 00 STA (S),Y
003B: 60 RTS
*
** OVER **
*
003C: 04 CF D6 C5 16 00 INC SL ; decrement S by 2
0042: E6 00 DEC SL
0044: C6 00 BNE 00
0046: D0 02 BNE 00
0048: C6 01 DEC SH
004A: C6 00 OK1 DEC SL
004C: D0 02 BNE 00
004E: C6 01 DEC SH
0050: C6 00 OK2 DEC SL
0052: A0 05 LDY 0005 ; move the two bytes
0054: B1 00 LDA (S),Y ; at S+4
0056: A0 01 LDY 0001 ; to the two bytes
0058: 91 00 STA (S),Y ; at S
005A: A0 04 LDY 0004
005C: B1 00 LDA (S),Y
005E: A0 00 LDY 0000
0060: 91 00 STA (S),Y
0062: 60 RTS

```

```

*
** SWAP **
*
0063: 04 D3 D7 Ct SC 06 LDY 9*03 I exchange the two
0069: A0 03 LDA (S),Y ; bytes
006B: 81 00 LDA (S),Y ; at S+2 with
006D: AA TAX ; the two bytes at 5
006E: A0 01 LDY 4401
0070: B1 00 LDA S*,Y
0072: A0 03 LDY #003
0074: 91 00 STA (S),Y
0076: 80 01 LDY 4501
0078: BA TZA
0079: 91 00 STA <S>,Y
007B: C8 INY
007C: 81 00 LDA (S),Y
007E: AA TAX
007F: A0 00 LDY ssee
0081: 81 00 LDA <S>,Y
0083: A0 02 LDY 0=02
0085: 91 00 STA (S),Y
0087: 80 00 LDY 0600
0089: 6A TZA
008A: 91 00 STA S*,Y
008C: 60 RTS
*
** ROT **
*
008D: 03 D2 CF D4 63 08 INC SL ; swap the two bytes
0093: E6 00 BNE 01 ; at S+4 with the
0095: D0 02 INC SH ; two bytes at S+2
0097: C6 01 INC SL
0099: E6 00 OK1 BNE 02
009B: D0 02 INC SH
009D: E6 01 JSR SWAP
009F: 20 69 08 Or 2 JMC SL ; then swap the two
00A2: E6 00 DEC SL ; 5 bytes at S+2
00A4: C6 00 BNE 03 ; with the two bytes
00A6: D0 02 DEC SH ; at S
00A8: C6 01 DEC SL
00AA: C6 00 0 3 BNE 04
00AC: D0 02 BNE 04
00AE: C6 01 DEC SH
00B0: C6 00 Or 4 DEC SL
00B2: 4C 69 08 JMP SWAP
*
** >R **
*
0065: 02 BE D2 A0 80 08 INC RL ; decrement R by 2
006B: E6 02 DEC RL
006D: C6 02 BNE 01
006F: D0 02 DEC RH
0071: C6 03 DEC RH
0073: C6 02 OK1 BNE 02
0075: D0 02 DEC RH
0077: C6 03 DEC RL
0079: 06 02 02 LDY 4501 ( move the two bytes
007B: A0 01 LDA (S),Y ; at S to the
007D: 61 00 STA (R),Y ; two bytes at R
007F: 91 02 DEY
0081: 68 LDA (S),Y
0083: 61 00 STA <R>,Y
0085: 91 02 INC SL $ increment S by 2
0087: E6 00 BNE 03
0089: D0 02 INC SH
008B: E6 01 03 INC SL
008D: D0 02 BNE 04
008F: E6 01 INC SH
0091: 60 04 RTS
*
** R> **
*
00E3: 02 D2 DE A0 65 08 INC SL 1 decrement S by 2
00E9: E6 00 DEC SL
00EB: 06 00 BNE 01
00ED: D0 02 DEC SH
00EF: 06 01 DEC SH
00F1: 06 00 0 1 DEC SL
00F3: D0 02 BNE 02
00F5: 06 01 DEC SH
00F7: C6 00 02 LDY 0=01 ( move the two bytes
00F9: A0 01 LDA <R>,Y 1 at R to the
00FB: 81 02 STA <S>,Y ; two bytes at S
00FD: 91 00 DEY
00FF: 88 LDA <R>,Y
0900: B1 02 STA (S),Y
0902: 91 00 INC RL
0904: E6 02 BNE 03 { increment R by 2
0906: D0 02 INC RH
0908: E6 03 INC RH
090A: E6 02 03 INC RL
090C: D0 02 BN 04
090E: E6 03 INC RH
0910: 60 04 RTS
*
** 1 **
*
0911: 01 09 A0 A0 e3 06 INC SL | decrement 6 by 2
0917: E6 00 DEC SL
0919: 06 00 BNE 01
091B: D0 02 BNE 01
091D: 06 01 DEC SH
091F: C6 00 01 DEC SL
0921: D0 02 BNE 02
0923: 06 01 DEC SH
0925: 06 00 02 DEC SL
0927: A0 01 LDY <*01
0929: B1 02 LDA (R),Y t at R to the
092B: 91 00 STA <S>,Y ; two bytes at S
092D: 88 DEY
092F: B1 02 LDA <R>,Y
0931: 91 00 STA S*,Y
0933: 60 RTS
*
** 0- **
*
0933: 02 B0 BD A0 11 09 LDY 0000 ; are the two bytes
0939: A0 00 LDA (S),Y ; at S both zero?
093B: B1 00 BNE NZ
093D: D0 00 INY
093F: C8 INY
0941: B1 00 LDA (S),Y
0943: D0 03 BNE NZ
0945: 98 TYA ; yes, flag=00001

```

continued

```

*945: De' * 2      BN ZERO
*947: A9 00      NZ   UDA **00 ( no, flag=sesee
0949: Ae' 00      ZERO LDY * 00
*94B: 91 00      STA (S), Y
*94D: 98          TYA
*94E: C8          I NY
094F: 91 **      STA (S), Y
e951: 60          RTS

*
* << * < **
*
*952: 02 Be BC A 33 '09 ' LDY **01 | is the MSB high''
*956: A0 01      LDA (S),Y
#95A: B1 00      BMI MINUS
0950: 30 01      DEY f no, 41*2" 0000
*95E: 88          MINUS TYA ( ves, flag=sepen
*95F: 98          LDY ****
*960: Ae' *0     STA (S>,Y
*962: 91 00      TYA
*964: 98          I NY
*965: C8          STA (S), Y
*966: 91 00      RTS
*966: 60          RTS

*
* * = **
*
*969: 01 ED A0 A0 52 09 LDY **00 i are the two bytes
*96F: Ae' se'    LDA (S),Y I at S equal to the
*971: B1 00      LDY **2 J two bytes at S+27
*973: A0 *2     CMP (S), Y
*975: D1 00      BNE NEO
*977: De 0D     DEY
*979: 88          LDA (S), Y
097A: B1 *0     LDY eset.
0970: A0 03      CMP (S), Y
097E: D1 00      BNE NEC
0982: A 9 01    LDA * *01 ( ye*, flag=seesi
0984: 00 02     BNE EO
0986: A9 00     NEO LDA esee' t no, 4 lag=0seee
*988: Ae 02     EO LDY **02
098A: 91 00     STA (S), Y
0980: C8          I NY
*980: A9 00     LDA * *00
09BF: 9 1 00   STA <S>, Y
*991: E6 00     INC SL 5 increment S by 2
*993: D 02     BNE OKI
*995: E6 01     INC SH
*997: E6 00     OKI INC SL
*999: D0 02     BNE OK2
*998: E6 01     INC SH
*99D: 60          OK 2 RTS

*
* * < **
*
0996: 01 BC A0 A0 69 09 LDY **02 : compare the two byte*
09A4: A0 02     SEC 1 at S+7 with the
*9A5: 38          LDA (S), Y ; two byte* at S
*9A7: B1 00     LDY ****
09A9: A0 00     SBC (S), Y
09AB: F1 00     LDY * *03
*9AD: A0 03     LDA (S), Y
*9AF: B1 00     LDY * *01
*9B1: A0 01     SBC (S >, Y
*9B3: F1 00     BVC OK 1
*9B5: 50 02     BOR **80
*9B7: 49 80     OK 1 AND * *80
89B9: 29 80     ASL
*9BB: *A        ROL ; and produce a 41 ag
*9BD: CB        I NY
*9BE: 91 00     STA (S), Y
09C0: A9 00     LDA **0
0902: C8          I NY
*9C3: 91 00     STA <S>,Y
*9C5: E6 00     INC SL $ increment S by 2
*9C7: De 02     BNE DK2
*9C9: E6 01     INC SH
*9CB: E6 00     OK2 INC SL
*9CD: D0 02     BNE 0.3
*9CE: E6 01     INC SH
*9D1: 60          OK3 RTS

a
* * 2 * <
*
*9D2: 01 BE A0 A0 9E 09 LDY esee' 1 compare the two byte*
09 D8: A 0 00

*9DA: 38          SEC ( at S with the two
09DB: B1 00     LDA (S),Y ; byte* at S+2
*9DD: A *2     LDY **02
99 DF: F1 M     SBC (S), Y
*9E1: 86          DEY
(92: B1 *0     LDA (6), Y
094: As 03     LDY * *03
*9E6: F1 *0     SBC (S), Y
*9EB: 50 02     BVC OKI
*9EA: 49 80     BOR *480
DEC: 29 80     AND **80
*9EE: *A        ASL
DEF: A9 *0     LDA esee'
09FL: 91 *0     STA <S>,Y
*9F3: 88          DEY
(9F4: 2A        ROL 1 and produce a flag
*9F5: 91 so'    STA (S),Y
D9F7: E6 *0     INC SL 1 increment S by 2
(9F9: De' 02     BNE OK2
(9FB: E6 01     INC SH
*9FD: E6 se'    OK2 INC SL
09FF: 00 *2     BE OK3.
BA01: E6 *1     INC SH
*A03: 60          OK3 RTS

a
* * AND **
*
M4: 03 C1 CE C4 D2 99 LDY **00 1 AMD the two byte*
BABA: AP *0     LDA (S),Y I starting at S with
*ARC: B1 os     LDY 0102' 1 the two at S+2
ME: A0 *2     AMD <S>,Y
BA10: 31 *0     STA <S>,Y
*AL2: 91 *0     DEY
*AL4: 88          LDY
*AL5: B1 **     LDA (S), Y
*AL7: A0 03     LDY **03
0019: 31 M     AND (S),Y
* AIB: 91 *0     STA (B),Y

```

continued

Which... would YOU buy?

**Global Specialties
QT-59S Socket**

- 590 Tie Points
- 8-I.C. Capacity
- No Grid Labeling
- Horizontal Expansion only
- Screw-Down Mounting



\$1250

**HANDY
HB-1000 Socket**

- 640 Tie Points
- 9-I.C. Capacity
- Alphanumeric Grid Labeling
- Both Horizontal and Vertical Expansion
- Self-Adhesive Mounting



NEW!

\$995

If you have two similar products, both designed for the same function...and one offers you MORE features for LESS money... which would YOU buy? The answer is obvious!

Just look at all these EXTRAS built into every HANDY test socket and buss strip...

- **Total contact labeling...** simplifies circuit design/layout
- **Self-adhesive backing...** for one-step simplified alignment and mounting
- **Full 9 14-Pin I.C. Capacity**
- **Expands both horizontally and vertically...** interlocks can't break or twist off
- **High temperature plastic housing...** to 80° C... no warping or melting ever!
- **Prices always up to 25% less than other leading brands**

To all these add: Long Life, low resistance and wide range contacts that accept combinations of resistors, capacitors, diodes, transistors, I.C.s, etc. with leads from .012 - .032" or 20 - 29 AWG. Clear, easy-to-read-and-identify contact markings simplify layout, wiring and documentation. Socket rows are labeled 1-to-64. and columns are marked A-to-E and F-to-J. Mating buss strip rows are labeled

A-to-D and consist of 25 contacts each. Bold red and blue lines show where contact strips begin and end.

Finally, we have a full line of breadboarding equipment, from discrete sockets and buss strips to multi-board assemblies, available at comparable lower-than-low prices.

Let's face it. If you get all this...and the prices are ALWAYS to 25% less... there's no doubt which you'll buy. HANDY. It's our name...and it tells you what we do.

What are you waiting for? Order your HANDY breadboarding products today. A toll-free call is all it takes!

Here's how to order...

HANDY Sockets and Buss Strips

Part Number	Socket Strips	Buss Strips	Ground Plate	Tie Points	14 pin IC Capty.	Price
HB-0100	N/A	1	no	100	N/A	2.25
HB-1000	1	N/A	no	640	9	9.95
HB-1110	1	1	yes	740	9	11.95
HB-1210	1	2	yes	840	9	13.95

HANDY Breadboard Assemblies

Part Number	Socket Strips	Buss Strips	Binding Posts	Tie Points	14 pin IC Capty.	Price
HB-2112	2	1	2	1380	18	25.95
HB-2313	2	3	3	1980	18	31.00
HB-3514	3	5	4	2420	27	47.95
HB-4714	4	7	4	3260	36	63.95

Mall Orders: Please add \$3 (Canada & Int'l add \$5) to cover cost of shipping/handling. Sorry! No COD orders.
Charge Cards: (Min \$15) Please include Acct No., Exp. Date, Bank No. (M/C only) and your signature.
Checks: Drawn in U.S. Dollars on US banks only. Connecticut Residents: Add 7.4% Sales Tax.



To order...call 1-800-34-HANDY

...charge with VISA, MasterCard or American Express.
All items off-the-shelf for Immediate Shipment!

a division of RSP Electronics Corp.

7 Business Park Drive • P.O. Box 699 • Branford, CT 06405 • (203) 488-6603 • TWX: (910) 997-0684
Easy Link Mail Box: 62537580 • CompuServe: 71346,1070
U.S. and Canadian Distributor inquiries welcomed.

Interfacing Tips and Troubles

A Column by Neil Bungard I

Introduction

Every new column begins with a particular theme in mind. Perhaps an appropriate theme for "Interfacing Tips and Troubles" would be "get down to brass tacks and make the interface work." This column will take a practical and workable approach to connecting computers to the real world. We will look at the troubles which you will likely encounter when attempting to connect interface circuits to your computer, and we will review some handy tips which will make your interfacing tasks a little less work. As you may have already discovered, conceptualizing a circuit, creating a logic diagram, and wiring the circuit are only part of an interfacing task. Once all of these things have been done, you've still got to make the interface work. Sometimes you can connect your interface to the computer and it works the first time, but in many cases it does not. You may trace your circuit and find that it has been wired correctly, so you go back to the schematic diagram to see if you've made a logic error. Upon discovering that your logic was correct you disconnect the interface from the computer, connect wires to the buses and the control signal inputs of your circuit, and check the interface under static conditions. Surprisingly, it works! Where to next? That's part of what this column is all about —determining the source of problems which don't appear on the logic diagrams. Such problems can be caused by power supply noise, lack of sufficient (or proper) decoupling, improper cable termination, mismatched transfer timing, and a number of other conditions which are rarely discussed in interfacing literature. I will address a number of these issues myself and I hope to solicit help from you, the readers, who have encountered and solved such problems in your interfacing experiences.

Although the interfacing techniques are straightforward, each personal computer has its own unique idiosyncrasies which may add a "twist" to the interfacing process. As you interface your computer and discover its particular quirks, drop us a letter; we would like to share the benefits of your experience with others who own similar machines.

"Interfacing Tips and Troubles" also hopes to provide a number of helpful tips that will assist you in your interfacing tasks. These "tips" will range from software suggestions to trick circuits which will hopefully help you as you connect your computer to the real world. Here again, your experience can be of benefit to us. Everybody has a trick which has made interfacing a little easier for you. Drop us a line; we would like to share your trick with others.

DC to DC Converter

We will begin this column with a handy little circuit which

eliminates the need for multiple output power supplies in applications where the supply current requirements are low. Some of the applications for such a circuit include: voltage levels for the RS 232 serial interface, programming voltages for EPROMs and CPUs, voltage references for analog to digital converters, etc. Typically, if a circuit requires more than one voltage, a multiple output power supply is used. A few disadvantages of the multiple output power supply are that it is physically large, relatively expensive, and in many cases is hard to find for the specific voltages that you are interested in. Another problem is adding ICs that require voltages other than five volts to a circuit with only a five volt power supply. At this point, if you are going to use a multiple output power supply, you must replace the existing supply. Of course, you would like to avoid that if possible.

I have managed to avoid multiple output power supplies in many applications by using a little circuit shown to me by Cam "Cannonball" Reid. Cam is always good for a trick circuit when I need one. This circuit is a DC to DC converter that you can build and place right on the board to generate needed voltage levels. A general circuit diagram of the DC to DC converter is shown in Figure 1. NPN transistors Q1

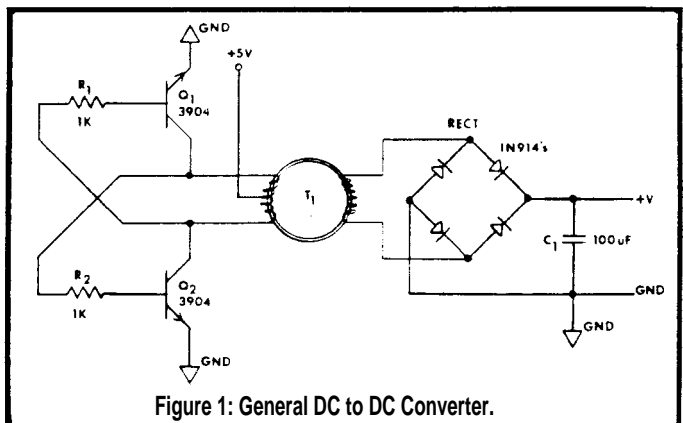


Figure 1: General DC to DC Converter.

and Q2, resistors R1 and R2, and the primary of transformer T1 form an astable multivibrator. The frequency of the multivibrator is somewhat dependent upon the physical characteristics of T1, but in general it will oscillate at about 1000hz. As the multivibrator oscillates, it induces a voltage into the secondary of T1. The voltage in the secondary winding is stepped up according to the relationship:

$$V_s = \frac{N_s}{N_p} \times V_p$$

V_s = Voltage Secondary
 V_p = Voltage Primary
 N_s = Turns Secondary
 N_p = Turns Primary

A bridge rectifier converts the secondary oscillation to an unfiltered direct current and capacitor C1 smooths the waveform. Since the frequency of the unfiltered DC is relatively high (compared to 120hz for a conventional power supply), capacitor C1 can be small (about 100mf). As shown in the photograph in Figure 2, the DC to DC converter can be made very small so that little board space will be dedicated to accomplishing your voltage conversion.

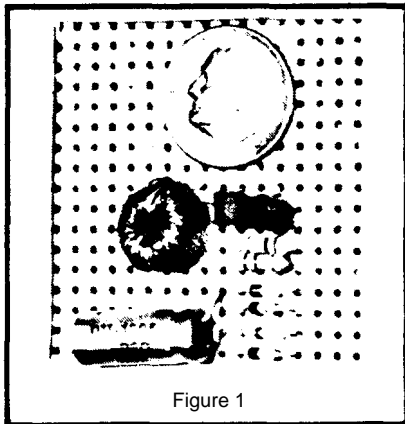


Figure 1

This DC to DC converter can be used to generate either a positive or a negative voltage by simply connecting the appropriate leg of the bridge rectifier to ground. Development of the DC to DC converter can be carried further to incorporate a three terminal voltage regulator (like the

7905) to ensure a stable voltage level. A plus and minus output is also possible from the DC to DC converter. By center tapping the secondary winding and connecting the center tap to ground, a positive voltage can be obtained from one leg of the bridge rectifier and a negative voltage can be obtained from the other leg. Circuits representing these applications will be presented in the following discussions.

The following circuits represent a couple of applications for which I have personally found the DC to DC converter useful. In the first circuit (Figure 3), the DC to DC converter was used to generate the programming voltage for a 2716 EPROM. It was very important that the programming voltage did not exceed 25 volts, so the output of the supply was regulated with a zener diode. The circuit in Figure 3 works exactly like the general DC to DC converter circuit in Figure 1. T1, a toroidal core transformer, can be purchased from Mouser Electronics for about a dollar, and they can be ordered in various sizes. The size (in inches) used in the converter above was: 0.50 (outside diameter) by 0.30 (inside diameter) by 0.19 (height). The primary winding consists of about 20 turns of number 26 enameled copper wire. The number of turns on the secondary winding was calculated according to the equation:

$$N_s = \frac{V_p}{X} \times N_p$$

These windings were then placed on the core and T1 was wired into the converter circuit. The converter circuit was to be regulated at 25 volts under a loaded condition so that the unregulated voltage was adjusted to about 26.5 volts. To adjust the unregulated output, remove the zener diode from the circuit, place a voltmeter on the secondary of T1 until the desired voltage is obtained. Once the 26.5 volts is obtained, place the zener diode back into the circuit — the DC to DC converter is now ready for service.

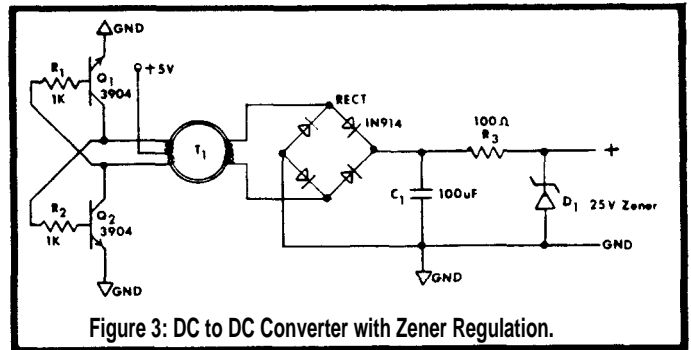


Figure 3: DC to DC Converter with Zener Regulation.

The circuit in Figure 4 is an example of generating a negative 12 volt reference using the DC to DC converter and a standard three terminal voltage regulator (7912). Before adjusting the secondary voltage, consult the specifications for the particular regulator that you are using. These three terminal regulators require a minimum input to output voltage difference to operate properly. Also keep in mind that the voltage difference between the input and the output will be dropped across the voltage regulator and dissipated as heat, so stay as close to the minimum difference specification as possible. The capacitors, C2 and C3, on either side of the voltage regulator in Figure 4 ensure that oscillations do not occur within the voltage regulator itself.

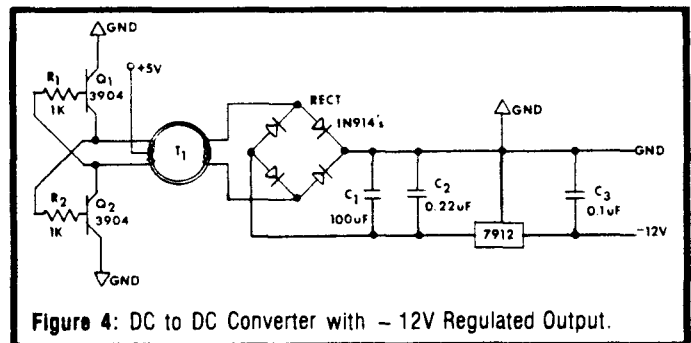


Figure 4: DC to DC Converter with -12V Regulated Output.

My favorite application for the DC to DC converter circuit is shown in Figure 5. This circuit configuration generates the plus and minus voltages required for an RS 232 serial interface. As you can see, the secondary is center tapped. With the center tap grounded, a negative voltage is obtained from one leg of the bridge rectifier and a positive voltage is obtained from the other. The RS 232 voltage

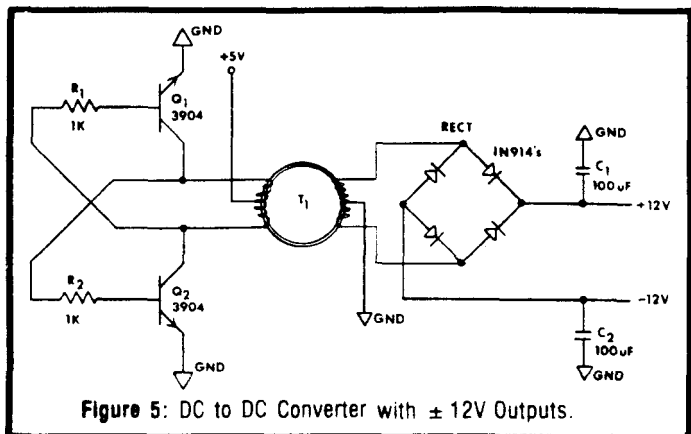


Figure 5: DC to DC Converter with ±12V Outputs.

Multi-user

A Column by E.G. Brooner

While talking about networks and other multi-user systems, we should keep in mind that most of the concepts originated with large mainframe or minicomputer companies, and are just now being adapted to micros. This is for the same reason that micros themselves are popular, namely that they are more cost effective in the instances where they are adequate—as they indeed are for a great many applications.

As one of the pioneer microcomputer makers, Cromemco is in a position to take advantage of the current multi-user boom—they have done this with C-NET. C-NET is a true network, and exhibits most of the finer features that distinguish such systems from lesser lookalikes.

This system features a relatively high-speed data rate of 0.5 megabytes per second, true collision detection, and packet message construction. It uses the "bus" topology (in which all stations are effectively in parallel across a common transmission medium) and permits up to 255 users to be connected at any one time. It generally follows the OSI 7-level protocol system; the three lower, or network levels are hardware on a plug-in S-100 board and the transport (routing, etc.) is handled by associated software. Levels five, six, and seven of the OSI model are necessarily machine-dependent in any network operation and are not considered a function of the network itself.

Figure 1 is a photo (courtesy Cromemco) of their plug-in network board, which is most of what you need to adapt a Cromemco computer for network use. This is a standard S-100 (IEEE-696) board and although it was designed with their computers in mind, there is no fundamental reason it could not be used in any other S-100 bus computer. Figure 2 is Cromemco's illustration of a typical C-NET configuration.

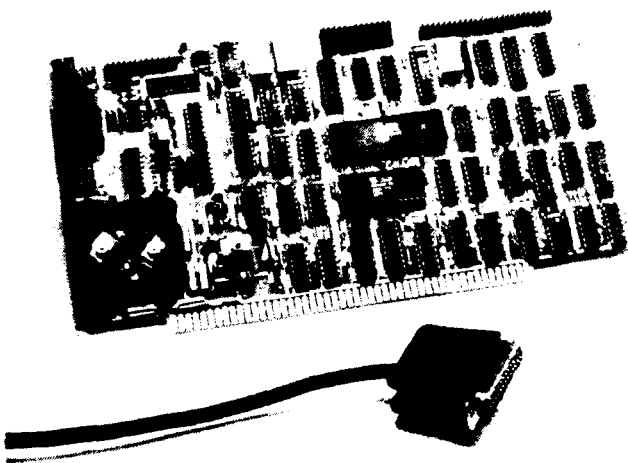


Figure 1

The functions they depict here are perhaps atypical of the installation that would be made by most small users; one of the interesting features they show is a *gateway* to a distant point. We have not previously discussed gateways in this column.

Earlier discussions described LANs as being very local in nature and confined, generally, to a building or relatively small complex. These limitations are imposed by the high speed at which networks usually operate. Long-distance communication is inherently at a slower rate. A gateway is a means of connecting a network, or some of its users, to a distant point. It may also connect two similar networks which, for some reason, cannot be totally combined. Communication with the distant point does not take place at high network speeds. The medium associated with a gateway may be telephone, satellite, or any other means of connecting two digital devices. Essentially, the gateway is an external connection, at which translations of speed and/or protocol may have to take place.

To understand gateways, we will have to discuss network protocols in a little more detail than we have in previous columns.

The International Standards Organization (ISO) has developed a plan known as the OSI, or Open Systems Interconnection model. This plan attempts to define all interactions between digital equipment, and between equipment and applications, as seven families of standards and protocols. The term "7-level protocols" is often used to describe this scheme. The RS-232 standard, the Centronics parallel standard, and the IEEE-488 bus are all level one protocols when viewed in this context—in other words, they are means of connecting equipment at the lowest level.

All methods of formatting data for transmission, such as SDLC, HDLC, and the various "packet" configurations, are level two protocols. It is important to note that the OSI model does not attempt to standardize any single protocol at any level, but only defines the functions that will be performed.

Needless to say, manufacturers do not always (as yet) follow this scheme. Ethernet, as an example, lumps levels one and two together and calls it "Ethernet;" this poses no problems unless one would try to interface something into an Ethernet between levels one and two. IBM and other mainframe makers usually have some kind of "network architecture" (such as IBM's SNA) that combine all of the low-level protocols in an inseparable manner.

Using the OSI concept, each level provides a transition to the next higher and next lower layers. It is apparent that if one network uses a certain type of level three or network control protocol, and another uses a different protocol for

COMPUTER CONTROLLED ROBOTICS

1. DRIVER BOARD 5005 DB \$75 *
4. 5" x 3.8" x 0.5". TTL/CMOS COMPATIBLE, OPTICALLY INSULATED. FOR 4 PHASE MOTORS 2AMPS/50 VOLTS
2. LINEAR ACTUATOR 601 AM \$ 75
12V 12W, 16 02. .001" STEP SIZE
.19 LBS HOLDING FORCE, 3 IN TRAVEL
3. LINEAR ACTUATOR 501AM \$43
12V/3.5W, 1.5 02. .002" STEP SIZE
40 02 HOLDING FORCE, 1.88 IN TRAVEL
4. STEPPER MOTOR 201SM \$ 16
5V 2W, 1.0 02, 15° STEP SIZE
0.8 02 IN HOLDING TORQUE
5. STEPPER MOTOR 301 SM \$ 59
12V, 21.5 02, 1.8° STEP SIZE
80 02 IN HOLDING TORQUE
6. MOTOR MOUNT FOR 301 SM \$25
7. MOTOR MOUNT FOR 501 AM \$ 12
- & MOTOR MOUNT FOR 501 AM \$ 13
- * EDGE CONNECTOR \$ 3.50

U AMSI CORR (516) 361-9499
Me BOX 651, SMITHTOWN, LI., N.Y. 11787

TERMS Check, Money Order, C.O.D. VISA, MasterCard, Purchase Orders from Accredited Institutions

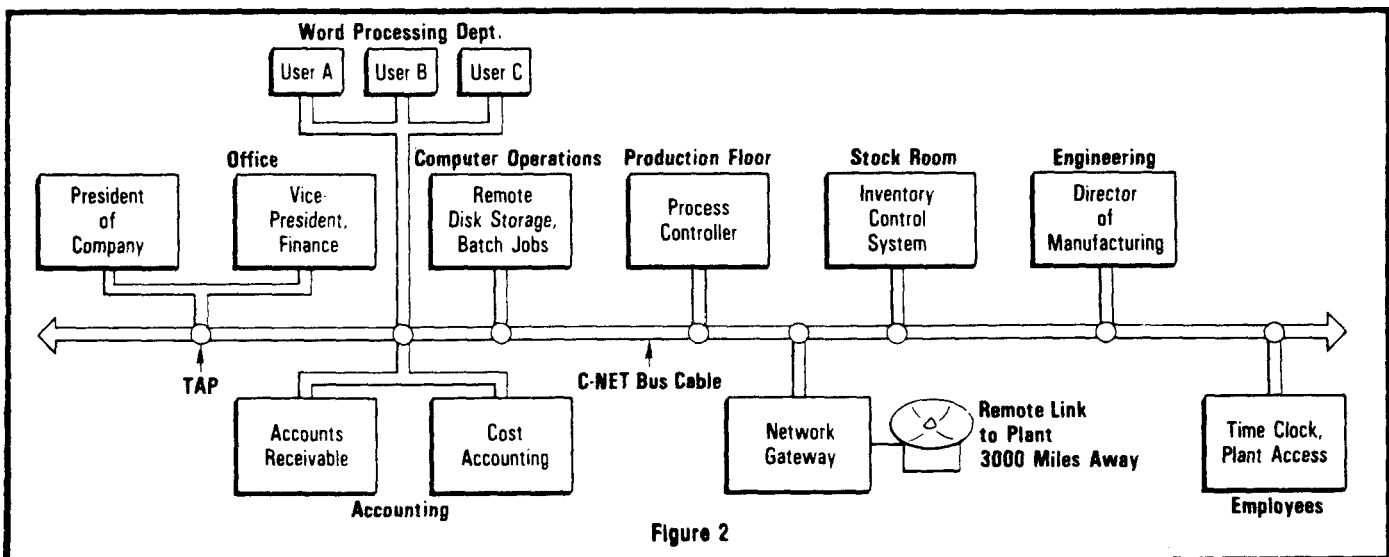
the same purpose, they are not compatible.

Finally, we get back to the concept of a "gateway." If at any point in a communication system it is necessary to translate from one protocol to another, at an equal level, a gateway is necessary. The gateway might change a format, a code, the speed of transmission, or the type of medium that is being used.

Cromemco does not advertise the details of the gateway shown in their functional diagram and, in fact, they may

have several different configurations for different purposes.

Now that we are on the subject of the protocol model, we should go ahead and define the rest of them as well as the state of the art permits. As explained earlier, level one is the direct connection of equipment, level two is the format of the transmission, and level three is the network control. There is no general agreement on the distinction between three and four —both are concerned with "transporting" data between points after it is *continued on page 22*



Searching for Useful Information?

The Computer Journal is for those who interface, build, and apply micros. No other magazine gives you the fact filled, how-to, technical articles that you need to ; use micros for real world applications. Here is a list of recent articles.

Volume 1, Number 1:

- The RS-232-C Serial Interface, Part One
- Telecomm@ngs" the Apple): Transferring

Bin:

- BeMMNers:cRumn, Part One: Getting Started
- Build an "Epram"

Volume 1, Number 2:

- File Transfer Programs for CP/M
- The RS-232-C Serial Interface, Part Two
- Build a Hardware Print Spooler, Part One: Background and Design
- A Review of Floppy Disk Formats
- Sending Morse Code With an Apple]
- Beginner's Column, Part Two: Basic Concepts and Formulas in Electronics

Volume 1, Number 3:

- Add an 8087 Math Chip to Your Dual Processor Board
- Build an A/D Converter for the Apple]
- ASCII Reference Chart
- Modems for Micros
- The CP/M Operating System
- Build a Hardware Print Spooler, Part Two: Construction

Volume 1, Number 4:

- Optoelectronics, Part One: Detecting, Generating, and Using Light in Electronics
- Multi-user: An Introduction
- Making the CP/M User Function More Useful
- Build a Hardware Print Spooler, Part Three: Enhancements
- Beginner's Column, Part Three: Power Supply Design

Volume 2, Number 1:

- Optoelectronics, Part Two: Practical Applications
- Multi-user: Multi-Processor Systems
- True RMS Measurements
- Gemini-10X: Modifications to Allow both Serial and Parallel Operation

Volume 2, Number 2:

- Build a High Resolution S-100 Graphics Board, Part One: Video Displays
- System Integration, Part One: Selecting System Components
- Optoelectronics, Part Three: Fiber Optics
- Controlling DC Motors
- Multi-User: Local Area Networks
- DC Motor Applications

Volume 2, Number 3:

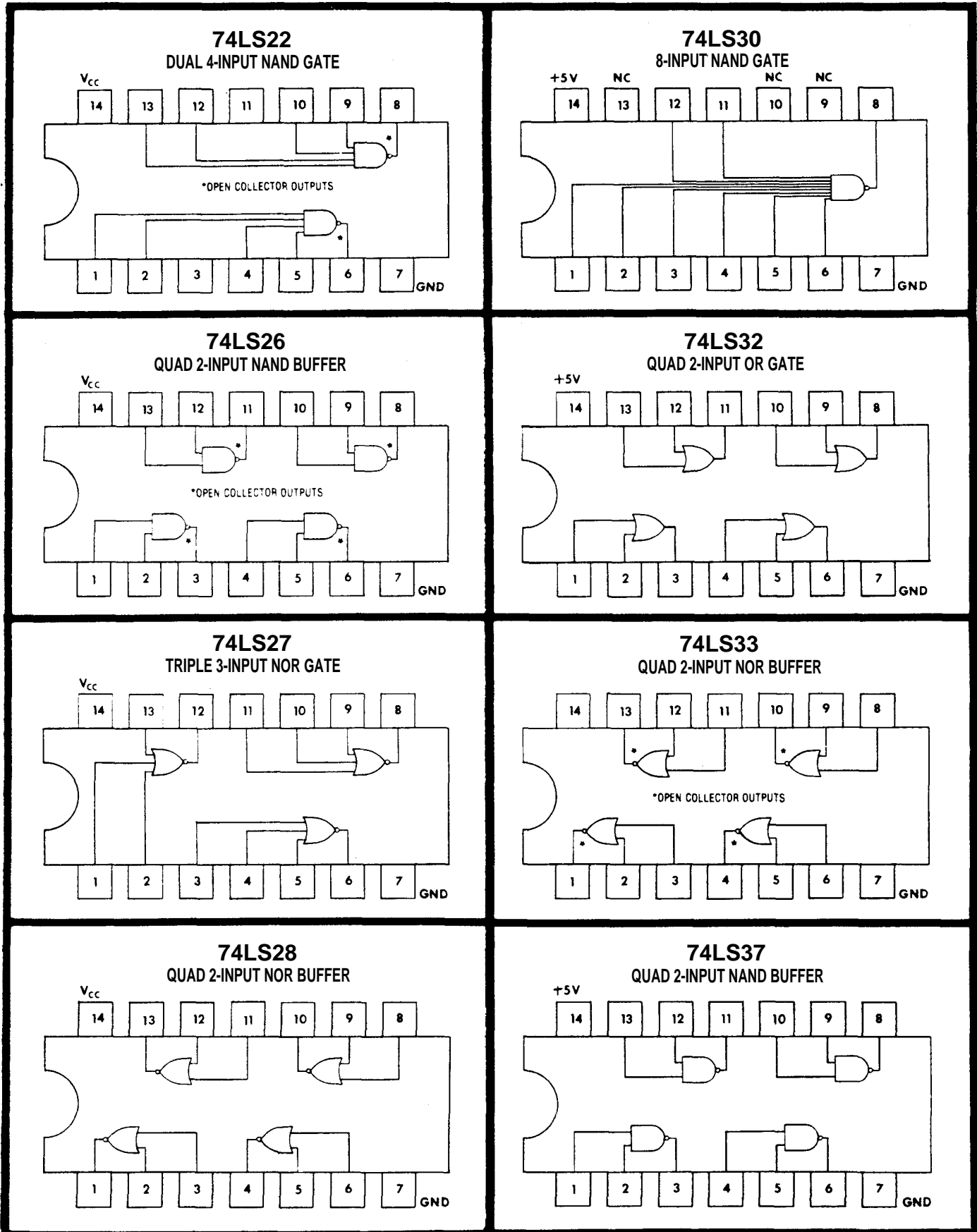
- Heuristic Search in Hi-Q
- Build a High-Resolution S-100 Graphics Board, Part Two: Theory of Operation
- Multi-user: Etherseries
- System Integration, Part Two: Disk Controllers and CP/M 2.2 System Generation

Volume 2, Number 4:

- Build a VIC-20 EPROM Programmer
- Multi-user: CP/Net
- Build a High-Resolution S-100 Graphics Board, Part Three: Construction
- System Integration, Part Three: CP/M 3.0
- Linear Optimization with Micros
- LSTTL Reference Chart

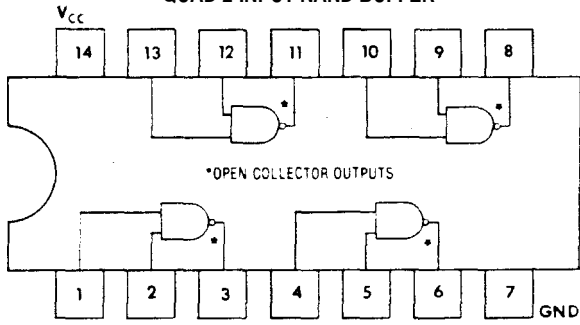
Back issues: \$3.25 in the U.S and Canada, \$5.50 in other countries (air mail postage included). Send payment along with your complete name and address to The Computer Journal, PO Box 1697, Kalispell, MT 59903. Allow 3 to 4 weeks for delivery.

LSTTL Reference Chart



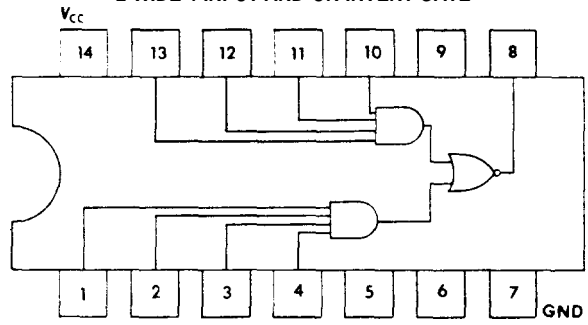
74LS38

QUAD 2-INPUT NAND BUFFER



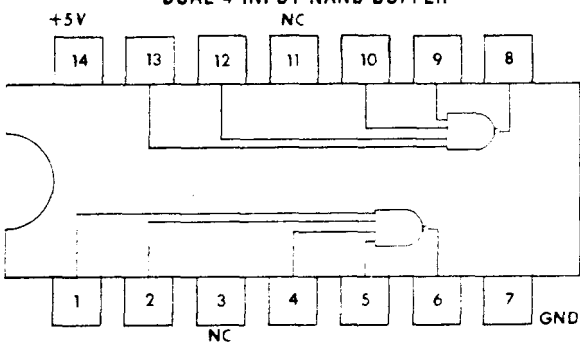
74LS55

2-WIDE 4-INPUT AND-OR-INVERT GATE



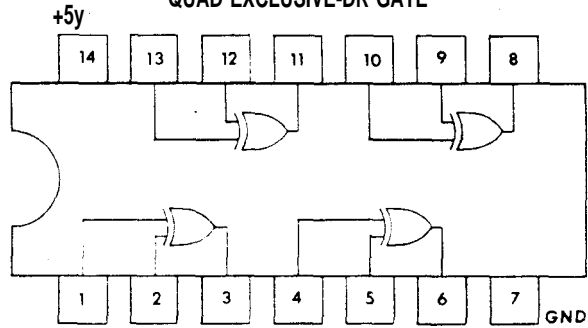
74LS40

DUAL 4-INPUT NAND BUFFER



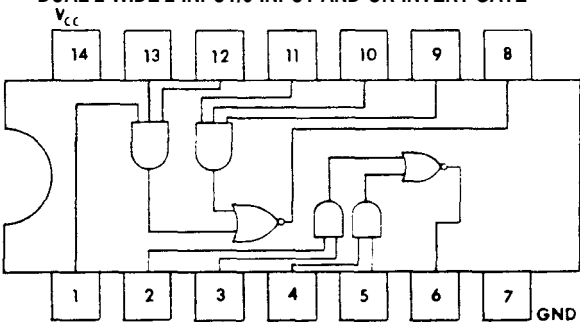
74LS86

QUAD EXCLUSIVE-OR GATE



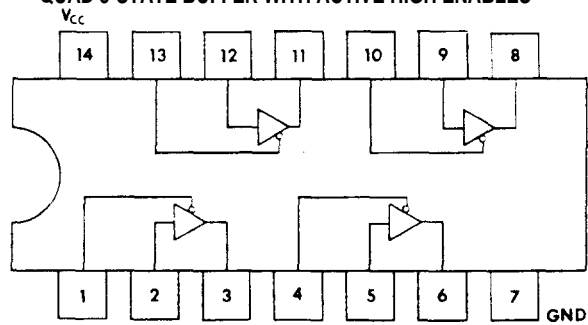
74LS51

DUAL 2-WIDE 2-INPUT/3-INPUT AND-OR-INVERT GATE



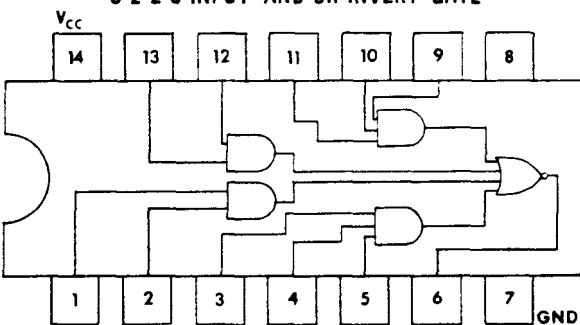
74LS125

QUAD 3-STATE BUFFER WITH ACTIVE HIGH ENABLES



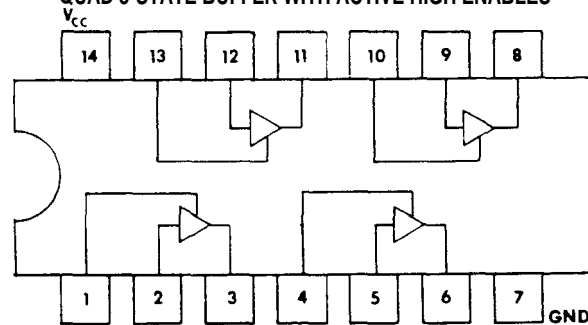
74LS54

3-2-2-3-INPUT AND-OR-INVERT GATE



74LS126

QUAD 3-STATE BUFFER WITH ACTIVE HIGH ENABLES



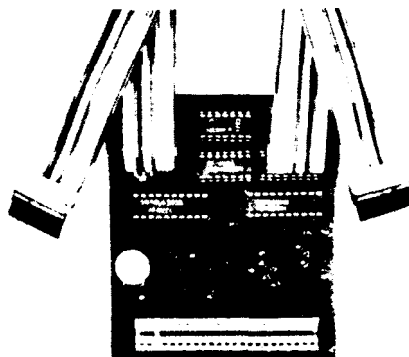
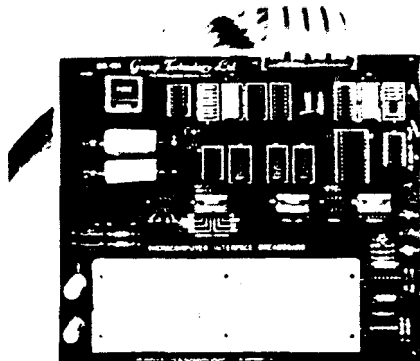
LEARN MICROCOMPUTER INTERFACING VISUALIZE SCIENCE PRINCIPLES

Using GROUP TECHNOLOGY BREADBOARDS with your
APPLE® ...COMMODORE64® ...TRS-80® ...TIMEX-SINCLAIR® ...VIC-20®

Versatile breadboards and clearly written texts with detailed experiments provide basic instruction in interfacing microcomputers to external devices for control and information exchange. They can be used to provide vivid illustrations of science principles or to design interface circuits for specific applications. Fully buffered address, data, and control buses assure safe access to decoded addresses. Signals brought out to the breadboards let you see how microcomputer signals flow and how they can be used under BASIC program control to accomplish many useful tasks.

Texts for these breadboards have been written by experienced scientists and instructors well-versed in conveying ideas clearly and simply. They proceed step-by-step from initial concepts to advanced constructions and are equally useful for classroom or individual instruction. No previous knowledge of electronics is assumed, but the ability to program in BASIC is important.

The breadboards are available as kits or assembled. Experiment component packages include most of the parts needed to do the experiments in the books. Connecting cables and other accessory and design aids available make for additional convenience in applying the boards for classroom and circuit design objectives. Breadboard prices range from \$34.95 to \$350.00



The INNOVATOR' BG-Boards designed by the producers of the highly acclaimed Blacksburg Series of books have gained wide acceptance for teaching microcomputer interfacing as well as for industrial and personal applications. Detailed, step-by-step instructions guide the user from the construction of device address decoders and input/output ports to the generation of voltage and current signals for controlling servo motors and driving high-current, high-voltage loads. BG-Boards are available for the Apple II, II+, IIe; Commodore 64 and VIC-20; TRS-80 Model 1 with Level II BASIC and at least 4K read/write memory. Models III and 4. The books, *Apple Interfacing* (No. 21862) and *TRS-80 Interfacing Books 1 and 2* (21633, 21739) are available separately.

The FD-ZX1 I/O board provides access to the Timex-Sinclair microcomputer for use in automated measurement, data acquisition, and instrument control applications. A number of science experiments have been developed to aid teachers in illustrating scientific principles. The operating manual contains instructions for constructing input and output ports. A complete text of the experiments will be available later in 1984. The FD-ZX1 can be used with Models 1000, 1500, 2068, ZX81, and Spectrum.

The Color Computer Expansion Connector Breadboard (not shown) for the TRS-80 Color Computer makes it possible to connect external devices to the expansion connector signals of the computer. Combined with a solderless breadboard and the book *TRS-80 Color Computer Interfacing, With Experiments* (No. 21893), it forms our Model CoCo-100 Interface Breadboard providing basic interfacing instructions for this versatile computer. Experiments in the book show how to construct and use a peripheral interface adapter interface, how to input and output data, and how digital-to-analog and analog-to-digital conversion is performed.

Our new Spring Catalog describes the interface breadboards, dozens of books on microcomputer interfacing, programming, and related topics including the famous Blacksburg Continuing Education Series, a resource handbook for microcomputers in education, and a comprehensive guide to educational software; utility software for the TRS-80, scientific software for the Apple II, and other topics. We give special discounts to educational institutions and instructors. Write for the catalog today.

Apple II, II+, and IIe are registered trademarks of Apple Computer Inc.; Commodore 64 and VIC-20 are registered trademarks of Commodore Business Machines; TRS-80 is a registered trademark of Radio Shack, a Tandy Corporation; Timex/Sinclair is a registered trademark of Timex Computer Corporation.

**PUTTING
HANDS
AND
MINDS
TOGETHER**



**Group Technology, Ltd.
P.O. Box 87N
Check, VA 24072
703-651-3153**

DOS WARS

by Bill Kibler

in the three-part series on integrating systems (Vol. II, nos. 2,3, and 4), I discussed operating systems in general. Recently I was involved in a discussion at my computer club about the different types of operating systems and their merits. Many users, I am sure, still have some questions about which operating systems to use and why. It occurred to me that I should explain more about DOSs (disk operating systems) both for beginners and for those who work with them but sometimes forget what they're for. Understanding DOSs is becoming more important as a larger variety of systems is pushed by the manufacturers.

DOS Designs

Digital Research Inc. made its name on the CP/M operating system. This system is the interface between the user and his computer. There are many different types of interfaces around. Some were even out before CP/M, but none provided such an open form of interface for independent writers of programs to use. Most interfaces before CP/M were company specific—that is, only software written by the company could run on their system. CP/M is an open standard which allows programs to move from system to system and is, for all practical purposes, independent of hardware considerations. This design opened up the microcomputer market and set the stage for the current DOS war.

CP/M was designed around the Intel 8080 device. This means that the machine code which comprises the program will run only on the 8080. The Zilog Z80 is a superset of the 8080 (contains the same machine instructions plus more special high speed functions) and will also run almost all CP/M programs unchanged. When used with an 8080 or Z80, CP/M is referred to as CP/M-80, as many newer versions of CP/M now exist. CP/M-86 is for the Intel 8086088, CP/M-68K for the Motorola 68000, CP/M-8K for Zilog's Z8000, and soon CP/M-16K for National's 16000 series of CPUs. All of these CPUs (central processing units) have 16 bit accumulators, whereas the 8080/Z80's is 8 bits. The other CPU enhancements concern the number of internal registers (temporary storage and pointers), the number of bits in each register, special register functions (indirect memory addressing), and how many address lines each can manipulate. Computers using the 16 bit CPUs can use more memory, perform larger calculations, and are supposed to be faster.

Before IBM, microcomputers were considered by many to be just hobby items. Although a lot of people were doing very meaningful things with their computers, if IBM didn't

do it, large business considered it a toy. When IBM entered the market, micros stopped being toys and many people bought their first computer (made by IBM). Due to financial considerations, IBM chose their operating system based on Microsoft's MSDOS. This program was originally written by Seattle Computers, and was adapted by Microsoft for the 8086088 system. It has the same open interface (CP/M-like) that allows programs to run independently of hardware. With IBM behind the system, MSDOS is now becoming the 16 bit standard, like it or not.

The War Starts

As one can probably guess, war between lovers of CP/M and MSDOS has been raging in many publications. The companies have countered with updated versions and even newer systems (concurrent CP/M). AT&T recently jumped into the fray with their UNIX operating system (originally for CPUs with a 32 bit design) and many 16 bit versions of it. The hour-long discussion at my computer club was on which one of these systems is best, and which one should be bought and learned. The general feeling was that none of the new systems merit buying now, but that they should be watched closely. This position is based on the current capabilities and prices of these products.

In my articles on system integration I tried to stress the need for defining clearly in your mind why you are getting a computer. The same is true when dealing with operating systems. Choosing the wrong system will limit the number of off-the-shelf programs you can run. I use mostly public domain software and pay about \$6.00 per 8" disk of programs. However, if I didn't use a CP/M system many of these programs would have cost me several hundred dollars per disk (my library has about 40 user group disks). There are currently several 16 bit programs in the normally 8 bit public domain, and separate MSDOS user groups are being formed (one group already has 135 IBM PC DOS disks). This wealth of programs has almost eliminated program availability as a deciding factor in purchasing systems.

In part three of *System integration*, I tried to point out why I felt CP/M 3.0 was not necessarily a good buy. I still feel that users of a 2.2 system are better off installing a RAM DRIVE than upgrading to 3.0. This also holds for upgrading to MSDOS or CP/M-86 even if the hardware is cheap. My personal home computer is a new Heath Z-100, bought so that I would have access to all possible worlds (S-100 bus, CP/M-80, MSDOS, and CP/M-86). Since writing articles is 80% of its use, CP/M-80 gets used 80% of the time. This has also given me a chance to test system speeds, and again CP/M-80 wins.

System Design

When the IBM PC first appeared in the trade journals, I studied the articles closely, and decided that it was a poorly designed machine. Since then, several articles have appeared showing how slow and cumbersome its design is. These facts have not stopped first time users from spending 14000 on the name. I would guess that many users are unaware of the system's poor features. Dedicated 2.2 users who use IBM usually become frustrated with its slow speed, and return to their favorite system. Why? CP/M 2.2 is a straightforward design, the DOS is located at high memory, and programs run from low memory just above a fixed entry jump table (8080 interrupt vectors set this design condition). When writing programs, little variation occurs between different systems to cause problems, and a lot of reference material is present to answer questions at non-technical levels. With 3.0, things start to get complex. Despite the fact that the same interface to the user is maintained, the operating system is in several parts, and data is moved into unknown parts using unknown algorithms.

MSDOS starts right out differently, with the DOS in low memory and the user program area after the DOS. Large memory is now necessary, offsets are used to determine which bank of memory is active, and 64K is still the largest continuous segment of program space usable. What this means to the user is a more complex system to understand, forcing the user to purchase programs. True, more complex programs can be run, but writing your own programs in anything other than BASIC is almost impossible. This fact is borne out by the IBM user disks which contain about 60-70% BASIC programs (including the disk utilities).

Not only should an operating system let you use programs (and be able to modify or understand them easily) but it must allow you to back them up (make copies), check density, catalog them, sort files, and many other disk utility functions. This is really where the battle is being fought—over utility functions. CP/M 2.2 has several built-in functions, and the open design allows for many utilities to support non-resident functions. The full operating system usually resides in less than 16K and can be used with only 10K of utilities beyond the system tracks. Approximately 170K is the normal free space on an 8" disk after I load my word processor and support utilities (250K after formatting). CP/M 3.0 uses a 32K system file plus system tracks (banked system) and may require up to 20K of support utilities. The same word processor and support programs would leave approximately 120K of 250K free. My favorite RAM DRIVE program only requires 4K of disk space and improves the speed to more than that of CP/M 3.0.

When looking at speed, the 16 bit systems have a major drawback; they do 16 bit moves on an 8 bit wide system. IBM uses the Intel 8088, a 16 bit wide device that talks to the system 8 bits at a time. This equates to two accesses for each full accumulator function. Since most PC programs are only conversions of 8 bit programs, optimization to use the 8 bit word moves is not always implemented. This means that two read or write functions must happen to load the accumulator. In true 8 bit systems only one read is needed

for many of the same operations, or the 8 bit systems will do it in half the number of program steps. Consider again that text is in ASCII code (my 80% word processing) which is 7 bits, and if 16 bit words are moved around, that wastes time. The only true test of speed is to try the system, since design is not the only thing to consider when comparing CPU capability.

Dos Usage

Having taken a diversion to discuss the different types of CPUs, let's get back to the main problem—that of which operating system to use. When 2.2 came out, most programmers were new at writing DOSs, and users helped upgrade the system as much as did the manufacturer. Things are different now. Competition is fierce, and professional programmers abound. Like some doctors, many programmers want all the work for themselves with as little intervention from the user as possible. TURBODOS*, a high speed version of CP/M 2.2 and MP/M, is a good example of some programmers' positions. The authors are not available, and hide out somewhere unknown to all but their agents. Very tight registration procedures are used (to prevent freebee systems), and little how-to information is given. Although this program works fine, I would rather use one of the look-alikes of CP/M that provides the source listing. Although I don't plan or recommend changes in DOSs for anyone but experts, my experience has shown that such lack of information will only cause problems in the long run.

In order to make comparisons, you, the user, need to define the system's use. Unix is a programmer's operating system, best suited to professional programmers who use the operating system more than they do canned programs. When I make changes in a BIOS, the steps go something like this: 1) make change in BIOS, 2) ASM BIOS, 3) find errors, 4) change errors, 5) run program and find errors, 6) go back to 1. Doing that 20 to 30 times in an afternoon can get pretty frustrating with slow, cumbersome systems. My Z-100 takes about a minute to boot, and if testing the system involves booting, that makes the loop about 10 to 15 minutes long. Small, short BIOSs that can be loaded quickly are great for developing new systems, as are programs that provide outer and inner rings for programs to run in. In the 32 bit versions, UNIX takes 250k of memory and several megabytes of disk space, which is definitely not a home system. Although this system provides a lot of tools (programs for special functions) for the programmer, and has those rings that help develop new programs more quickly, these functions are all at the operating system level. Once inside a running program, most of the extra bells and whistles are lost.

Not Getting Lost

What I have been attempting to do in this article is to give enough background on operating systems to keep you, the user, from buying useless programs. Most of these programs are not useless in the right hands, but advertising agencies can make you believe a child could run their

BUILDING A CODE PHOTOREADER

by R.O. Whitaker

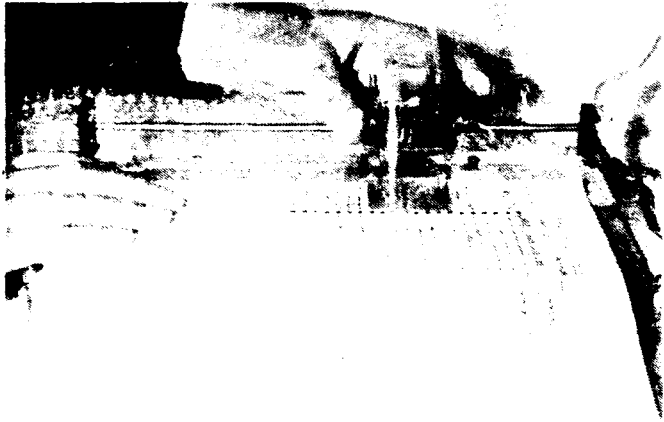


Figure 1: Reading the code into the computer.

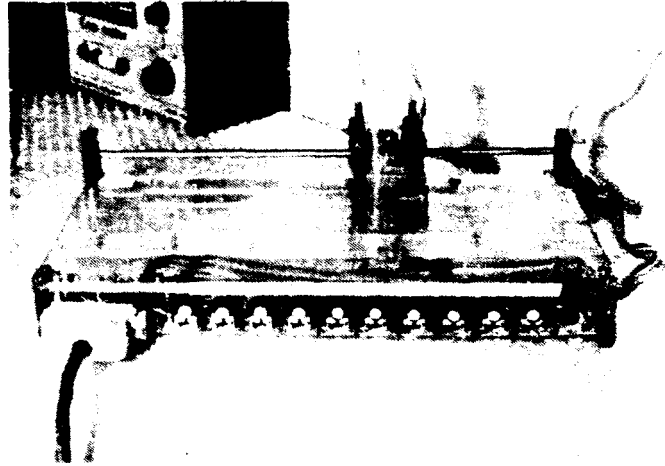


Figure 2: The reader. Electronics mounted beneath forward edge of platen. Seven zeroing pots shown in photo.

Introduction

I don't key programs into my computer any more. Instead, I put the code sheets in my photoreader, pull the scanner down the code column, and in goes the code. But it is really not all that easy, because the code has to be written in "Computer Compatible Digits" (CCD). First, I had to learn to read and write them. What is a "Computer Compatible Digit?" Look at Figure 3. The elements of the code are binary weighted. Since each element corresponds to a respective bit of a computer word, the digit is "Computer Compatible," and no encoding is required for going to or coming from the computer. The digit is inherently hexadecimal, as indicated at the bottom of Figure 3.

The first CCD was proposed by a gentleman over in Prague about 1955. Since then, several others have proposed digits of various configurations. The one I use was discussed in a paper published in an electronics magazine several years ago. All of the digits except six and seven can be written without lifting the pencil from the paper. All elements are equally conspicuous, and the elements of each

digit are mutually contiguous.

Figure 4 is a coding sheet with the code column on the left. The CCDs are formed by connecting the appropriate dots with a dull #2 pencil.

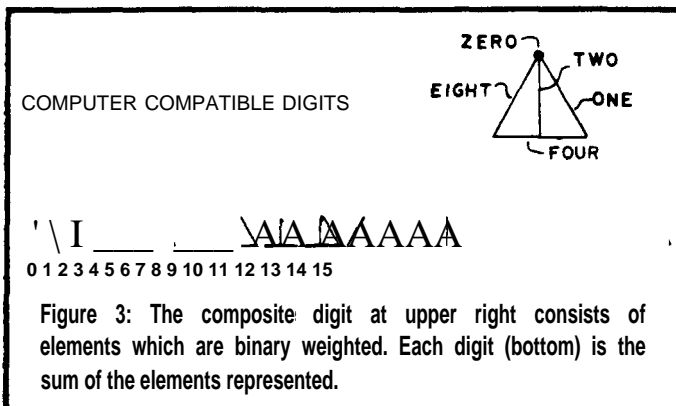
Figure 5 shows the basic structure of the photoreader with a Code Sheet on the platen. The left edge of the sheet is placed against the stop to assure proper horizontal registry. The scanner slides on the guide rod. The optical fibers pass through holes in the faceplate (which slides over the code column).

The scanner has a carrier associated with it which moves under the platen and keeps a pair of pilot lamps opposite the fiber ends. Light shines up through the transparent plastic platen and through the paper. Consequently, each fiber receives light except when it passes over a line of one of the digits. The other ends of the fibers feed to a series of photodetectors which are mounted on top of the scanner. The signals are then led by the ribbon cable to the electronics mounted under the forward edge of the platen (see Figure 2).

To operate the photoreader, insert the code sheet and move the scanner to the top of the column. Lift sense switch 80 on the computer. Draw the scanner down the page. Drop SSW 80, remove the code sheet, and insert the next one.

The fiber optic sensors scan along the dotted lines of Figure 6. Note that the ID (indicium) sensor senses the presence of indicia in the column at the right. The action is as follows:

1. When the ID sensor leaves an indicium, it causes the "B" register of the computer to be nulled. The word being scanned will be assembled in this register.
2. While the ID sensor is in the "A" region of Figure 6, the other six sensors look for their respective lines. Each one



that sees its line sets its respective bit in register B.

3. While the ID sensor is in region B, the 8-sensor looks for the 4-line. If it sees the 4-line, it sets the 4-bit in register B. Similarly, the 80-sensor looks for the 40-line and sets the 40-bit.

4. When the ID sensor drops from the indicium of region B, the word in register B is delivered to memory and to the display. Register B is reset before scanning the next word.

The "2-lines" were not included in Figure 6, but appear in Figure 4. Note that each 2-line bears a crossbar, which causes the 2-sensor to see the line even if the paper is off a millimeter or so in horizontal registry or if I was too sloppy in writing it.

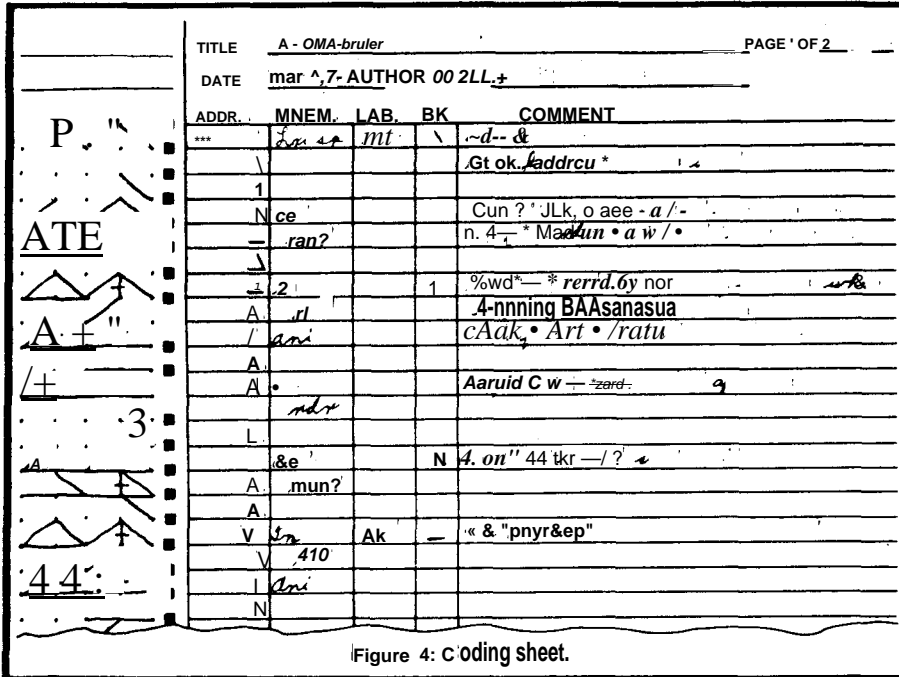


Figure 4: Coding sheet.

The Flowchart

Refer to the flowchart in Figure 7. Blocks one to four initialize the system.

Blocks five to end constitute the RUN loop in which the computer operates while characters are being read.

Blocks six to nine constitute the A loop, where the computer operates as long as the ID sensor is in region A.

Blocks ten to thirteen are the B loop where the computer operates while the ID sensor is in the B region.

Blocks fourteen to end terminate the reading of a pair of digits.

The blocks will now be discussed in order.

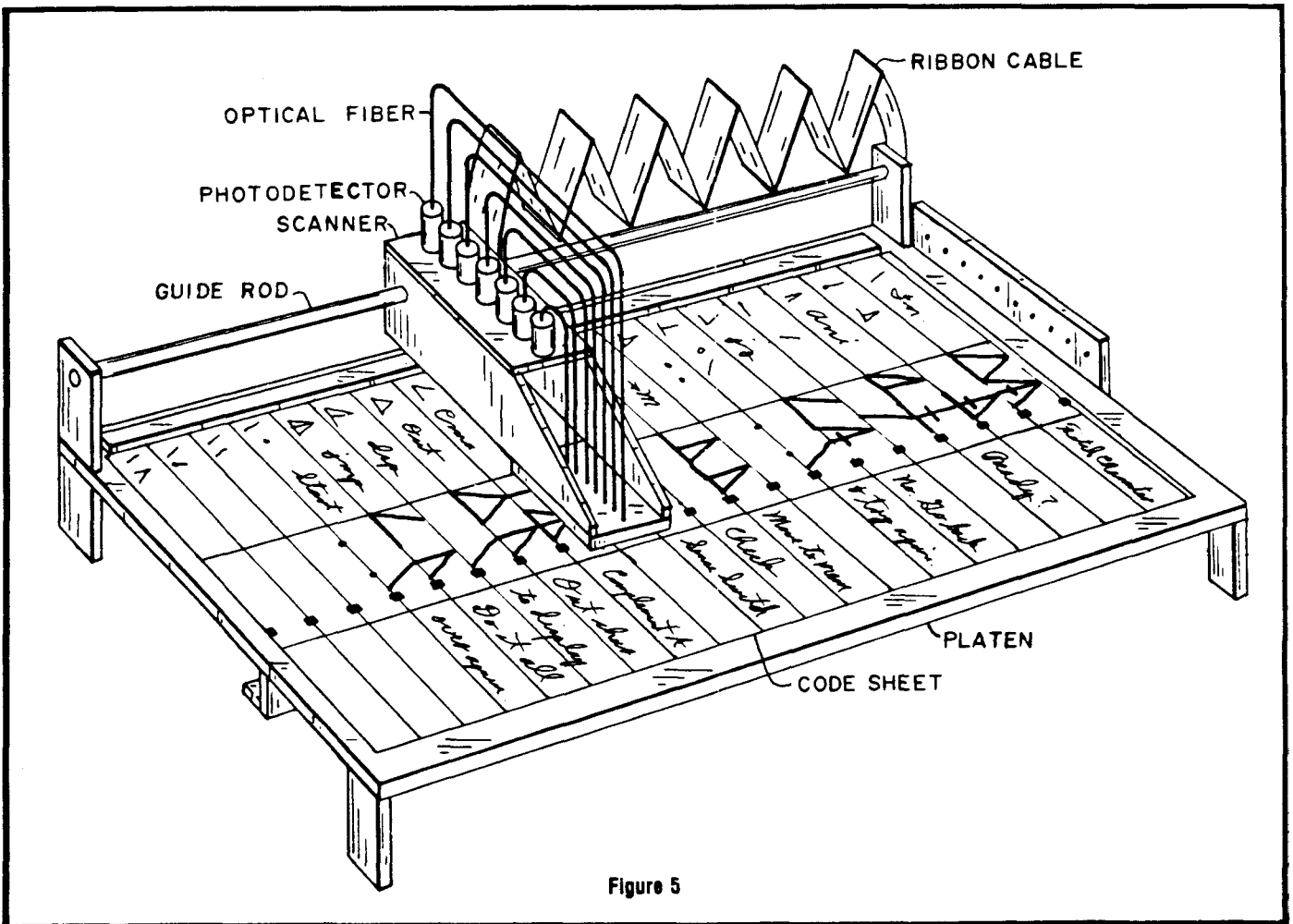
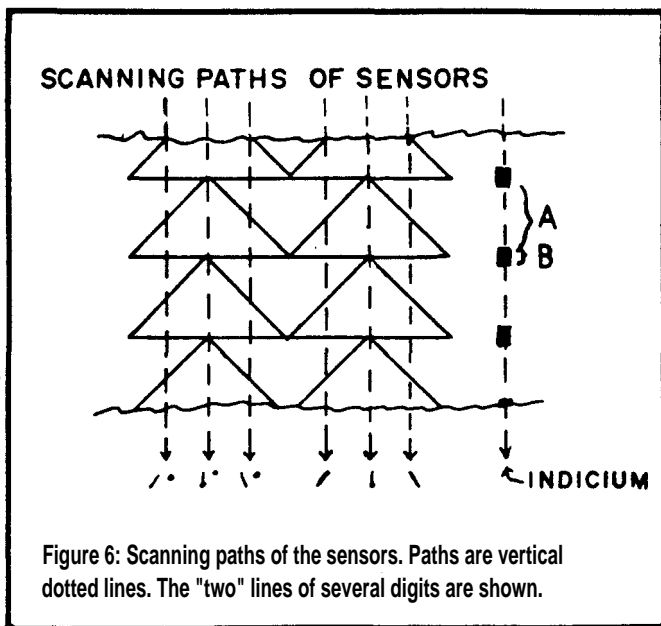


Figure 5



1. The computer twiddles until SSW 80 is lifted. This prevents anything from being read until I have zeroed the pots, given the computer a loading address, have the code sheet in place, and am ready to read.
2. As soon as SSW 80 is raised, the computer drops to here. If the first indicium is not yet seen by the ID sensor, the computer continues to twiddle.
3. As soon as the top indicium (see Figure 3) is sensed by the ID sensor, the computer drops to here. If the first indicium is not yet seen by the ID sensor, the computer continues to twiddle.
4. If SSW 80 is still up, the computer drops to here. It checks to see if the indicium is still visible. Yes. It twiddles in a new loop.
5. As soon as the ID sensor drops off the top indicium, the computer drops to here. This block nulls the B register before the word being read is assembled into that register. The "PWD" stands for "Processor Word." The word is assembled in register B and moved to memory.
6. Checks to see if SSW 80 is still up. No. It aborts the scan.
- 7, 8, and 9. Checks to see if the SSW 80 is still up. Yes. Brings in the "Read Word" (RWD) sensed by the reader at any particular instant and dependent upon where the scanner happens to be in relation to the digit pair. Merges it into B. Repeats Steps six through nine until the ID sensor senses the presence of the second indicium. Note from Figure 6 that this loop will operate for region A. It will cause the 1, 2, 8,10, 20, and 80 lines to be read. For each line present, it will cause its respective bit in register B to be set. Note from Figure 6 that a 1 or 8 line can hardly be mistaken for a 2-line.
10. Once the ID sensor sees the second indicium, the computer drops to here.
- 11, 12, and 13. This is the B loop. The computer stays here as long as the ID sensor sees the second indicium —as long as the sensors are in region B of Figure 6. The 80 and 8 sensors look for the 40-line and the 4-line respectively. The other sensors do nothing. If the 80-sensor sees the 40-line,

the 40-bit is set in register B. If the 8-sensor sees a 4-line, the 4-bit is set.
 14,15, and last. When ID drops off the second indicium, the word stored in register B is sent to memory and to the display. The memory pointer is incremented.

At the completion of scanning, I drop SSW 80 to prevent any more reading. A copy of the program (written for an 8080 computer) will be forwarded upon request. Send three 20 cent stamps to the author.

Electronics

Study Figure 8 for a moment. The transistors and associated components were mounted on a piece of perfboard under the platen, as shown in Figure 2. Pots were mounted so that they could be easily adjusted from the front. The LEDs were placed beside their respective pots. Since there is no 4-sensor, the ID sensor occupies the 4 slot.

The Schmitt trigger (7413) was installed on the output of the ID sensor to eliminate multiple readings associated with noise signals when the sensor moved over the edge of an indicium. The pots permit the channels to be "zeroed." G.E. specifies three volts for proper operation of the photodarlington. Zener Z-1 drops the five volt supply to this level.

Construction

I used scrap plastic obtained from a local plastics dealer for the frame parts and the platen. It seems that plastic is transparent to infrared radiation of the wavelength to which the sensors are sensitive. Hot glue was used to hold minor components in place. The guide rod on which the scanner slides was an iron rod taken from an old Selectric typewriter. It was already ground smooth, permitting the scanner to slide easily upon it. Any smooth steel rod about 4mm in diameter should do. Plastic Crofon optical fibers 1mm in diameter were used. The General Electric GF0D-1B photodarlington was designed to work with these fibers.

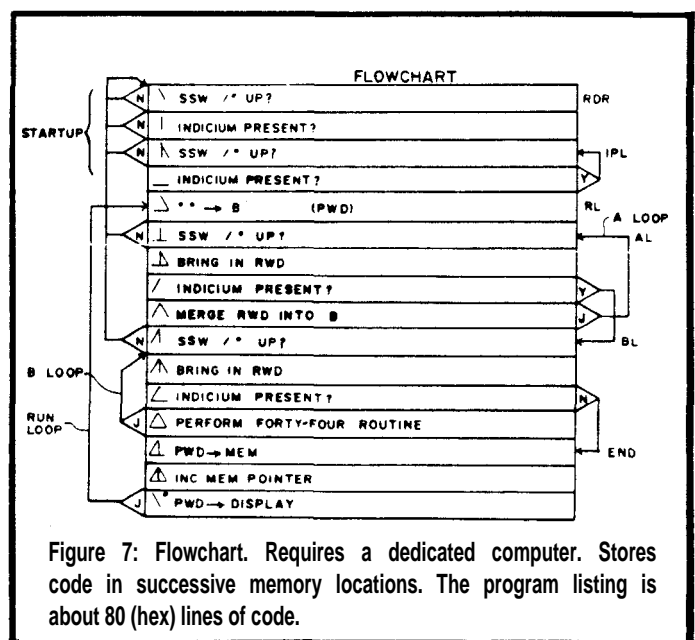


Figure 7: Flowchart. Requires a dedicated computer. Stores code in successive memory locations. The program listing is about 80 (hex) lines of code.

The open ends were polished with jeweler's rouge, and fit flush with the bottom surface of the faceplate. A piece of Scotch tape protected the exposed ends from abrasion. The holes for the fibers were drilled with a Unimat lathe, which permitted them to be positioned very precisely. However, that should not be necessary. If the digits are about 1.5cm in width, then a scale and centerpunch should allow the holes to be positioned with sufficient accuracy.

I used two six volt, 200ma pilot lamps for the light source, and surrounded them with a reflector made from a beer can. The two bulbs are powered by five volts regulated. I used a five volt bench supply to power the reader. Power could also be drawn from the host computer, or a supply could be mounted on the reader.

Zeroing the System

To aid in zeroing the system, I put two horizontal lines across the top of the code column (see Figure 4). The first is so thin that the sensors do not trip on it. The second is just barely dark enough to trip the sensors. The pots were then adjusted so that their respective LEDs did not light for the first line and did light for the second. The only requirement for proper reading is that the lines of each digit be darker than the second line.

Paper

At first I used a pad of dime store paper that was cross-hatched into 5mm squares. Any paper used should be free of clay—clay is opaque to infrared. I now use preprinted forms. They are more expensive and don't work a bit better, but they do make a more favorable impression upon the impressionable. Dots on the form are positioned so that the procedure of writing the code is reduced to connecting the appropriate dots with fairly straight lines.

Future Models

Small diameter sensors are now available which can be mounted in the read head, eliminating the optical fibers. Should I build another reader, I will go that route. It would probably be better to replace the lamps with infrared emitters of the semiconductor type designed to work with the detectors. The two ordinary bulbs I used worked fine, but ideally, an emitter with a lens would focus radiation upon the paper directly opposite its respective sensor.

Where to Get the Parts

The detectors should be procurable from your local GE distributor. The transistors and Schmitt triggers can be purchased from any Radio Shack or electronic parts store, or from a mail order company such as Jameco. I got my

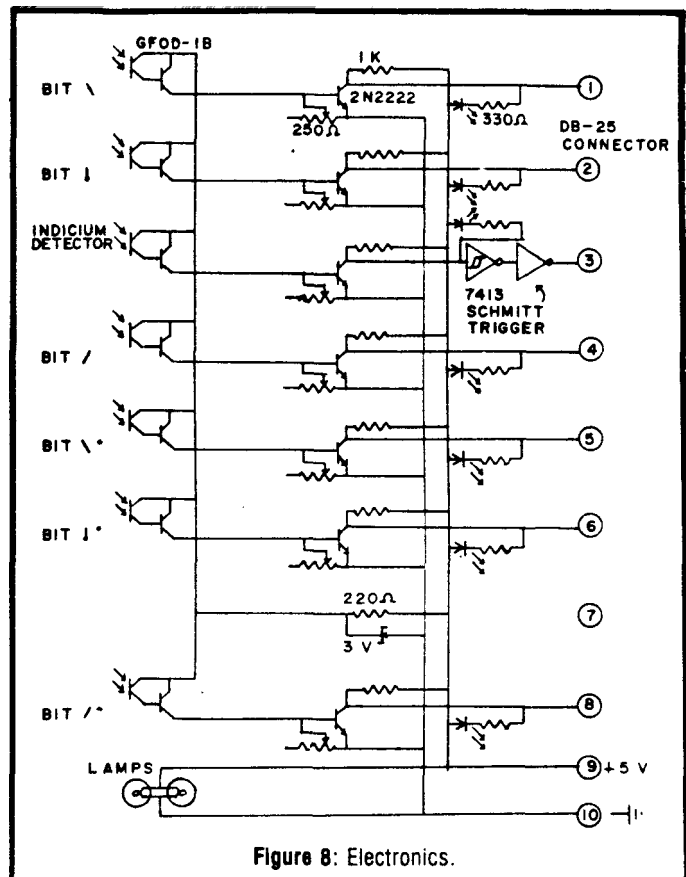


Figure 8: Electronics.

Crofon optical fibers from Edmund Scientific. Crofon is made by DuPont, and you can probably obtain it from a local plastics distributor. Check with GE. The matching connectors for the GFOD-1B were the most difficult parts to obtain. My source made me buy far more than I needed, and at an astronomical price. Check with your GE distributor regarding matching connectors, or check with me. If there is enough interest, I'll stock parts for resale at a nominal price. Hot glue or cement are viable alternatives to the connectors, but are not as fancy looking. The connectors are also considerably more convenient if you have to take the rig apart.

I have applied for a patent on the reader, and will sell you a non-exclusive license to build one unit for your own use. The cost to you will be one stamped, self-addressed envelope. The stamp should be of the 20 cent variety. And uncanceled. If you don't have one, let it go. Maybe I can find one around here.

R.O. Whitaker
4719 Squire Drive
Indianapolis, IN 46241

Customer Support Survey

In order to improve customer support and aid users in their shopping within the microcomputer industry, TCJ will publish user experiences with vendors. Send us your candidate for the best and worst vendor, along with your supporting information.

HELP!

The Readers' Column

Readers, this is your column! We encourage you to communicate with other readers by using this space to ask for their help with your problems, and to reply to the problems presented here. Where possible, the editors will respond to specific questions regarding TCJ articles. Otherwise, you can provide the "HELP" by sending your solutions for publication to PO Box 1697, Kalispell, MT 59903. We will try to keep the lead time short for a rapid exchange of information. Let us hear from you!

Dear Computer Journal,

I need information on interfacing a TI 99YA with a Teletype model 28.

Wilbur Kespert
Florida

Ed: Readers, can you help?

Dear Computer Journal,

Do you know how I could upgrade a TRS-80 MC-10 micro color computer to 64K using 4164 chips? Radio Shack makes a 16K plug-in module, but no 64K module.

Bryan Lepkowski
New York

Ed: Readers, can you help?

Dear Mr. Rose,

I have read your three part article on building a print spooler with great interest. I am seriously considering the project. I do have one question. I have an Apple II + and

some experience programming the 2716. I would like to know how I can handle the Z80 op codes with my "Big Mac" assembler. Could I enter the machine code directly into memory and then into the 2716? If so, could you provide me (and perhaps other readers) with a memory dump for the operating program?

Thanks,
Hugh McEntire
California

Ed- Although I don't have an Apple II + myself, I asked someone local here about the "Big Mac" assembler and he informs me that it can only handle the 6502 op codes and will not assemble those for the Z80. However, the program is not a long one, and your idea of entering it directly in machine language should be quite feasible. I made a hex dump of the operating program and am including it for you here.

Thanks for writing and good luck with your project.

Regards,
Lance Rose
Technical Editor

```
0000 01 00 00 11 00 08 21
0010 E6 01 CA 29 00 DB 01
0020 13 7A FE 80 C2 29 00
0030 DB 00 77 03 23 7C FE
0040 C2 09 00 79 FE F0 C2
0050 01 CA 75 00 DB 01 E6
0060 7A FE 80 C2 68 00 16
0070 D3 01 C3 09 00 DB 01
0080 92 00 DB 00 77 03 23
0090 4D 00 76
```

```
00 08 78 B1 CA 29 00 DB 01
E6 80 CA 29 00 1A D3 00 0B
16 08 DB 01 E6 02 CA 09 00
80 C2 3D 00 26 08 78 FE 77
09 00 3E 00 D3 01 DB 01 E6
80 CA 75 00 1A D3 00 0B 13
08 79 FE E0 C2 75 00 3E 02
E6 02 CA 4D 00 79 FE FF CA
7C FE 80 C2 4D 00 26 08 C3
```

Classified

The Computer Journal will carry Classified Ads. The rate is \$.50 per word, minimum charge \$7.50. All Classified Ads must be paid in advance, and will be published in the next available issue. No checking copies or proofs are supplied, so please type your ad or print legibly.



The Bookshelf

CP/M Primer

Helps microcomputer veterans and novices alike find the answers about CP/M in a complete, one-stop sourcebook that's a Sams best-seller! Gives you complete CP/M terminology, hardware and software concepts, startup details, and more for this popular 8080/8085/2-80 operating system. Helps you begin using and working with CP/M immediately, and includes a list of compatible software, too. By Stephen Murtha and Mitehd Waite. 96 pages, BVixll. comb. ©1980..... \$16.95

Soul of CP/M: Using and Modifying CP/M's Internal Features

Teaches you how to modify BIOS, use CP/M system calls in your own programs, and more! Excellent for those who have read *CP/M Primer* or who otherwise understand CP/M's outer-layer utilities. By Mitchell Waite. Approximately 160pages. 8x9", comb. ©1983..... \$18.95

The S-100 and Other Micro Buses (2nd Edition)

Examines microcomputer bus systems in general and 21 of the most popular systems in particular, including the S-100. Helps you expand your computer system through a better understanding of what each bus includes and how you can interface one bus with another. By Elmer C. Poe and James C. Goodwin, II. 208 pages, 5"x8", soft. ©1981/59,95

Interfacing & Scientific Data Communications Experiments

This book introduces you to the principles involved in transferring data using the asynchronous serial data-transfer technique. It focuses on using the universal asynchronous receiver/transmitter (UART) chip in order to help your understanding of communication chips. Explores operation of teletype-writer interfaces and serial transmission circuits. With experiments and circuit details. By Peter R. Rony. 160 pages, 5"x8", soft. ©1979..... \$7.95

Active-Filter Cookbook

A practical discussion of the many active-filter types and uses, written by one of Sams' most popular authors. Teaches you how to construct filters of all types, including high-pass, low-pass, and bandpass having Bessel, Chebyshev, or Butterworth response. Easy to understand — no advanced math or obscure theory. Can also be used as a reference book for analysis and synthesis techniques for active-filter specialists. By Don Lancaster. 240 pages, 5"x8", soft. ©1975..... \$14.95

Regulated Power Supplies (2nd Edition)

Newest, most comprehensive discussion you'd find of regulated power supplies, including their internal architecture and operation. Thoroughly explains how to use regulation in your designs and projects when the need arises, and discusses practical circuitry and components. A valuable book for any technician or engineer involved in servicing or design. By Irving M. Gottlieb. 424 pages. 5.3"x8", soft. ©1981..... \$19.95

TTL Cookbook

Popular Sams author Dan Lancaster gives you a complete look at TTL logic circuits, the most inexpensive, most widely applicable form of electronic logic. In no-nonsense language, he spells out just what TTL is, how it works, and how you can use it. Many practical TTL applications are examined, including digital counters, electronic stopwatches, digital voltmeters, and digital tachometers. By Don Lancaster. 336 pages, 5"x8", soft. ©1974..... \$12.95

IC Op-Amp Cookbook

An informal, easy-to-read guide covering basic op-amp theory in detail, with 200 practical, illustrated circuit applications to reflect the most recent technology. JFET and MOSFET units are shown in both single and multiple formats. Includes manufacturers' data sheets, and lists addresses of the companies whose products are featured. By Walter G Jung. 480 pages, 5"x8", soft. ©1980..... \$15.95

IC Converter Cookbook

Discusses and explains data conversion fundamentals, hardware, and peripherals. A valuable guide to help you understand and use d/a and a/d converter applications. Includes manufacturers' data sheets. By Walter G. Jung. 576 pages, 5"x8", soft. ©1978..... \$14.95

IC Timer Cookbook (Second Edition)

Learn more ways to use the IC timer in this easy to use second edition that includes many new IC devices with ready to use applications in practical, working circuits. AU circuits and component relationships are fully defined and discussed for clarity. By Walter C. Jung. 384 pages, 5"x8", soft. ©1983..... \$17.95

The Programmer's CP/M Handbook

An exhaustive coverage of CP/M-80*, its internal structure and major components is presented. Written for the programmer, this volume includes subroutine examples for each of the CP/M system calls and information on how to customise CP/M — complete with detailed source codes for all examples. A dozen utility programs are shown with heavily annotated C-language source codes. An invaluable and comprehensive tool for the serious programmer. By Andy Johnson-Laird, 750 pages, 7"x9", softbound..... \$21.95

Interfacing to S-100 (IEEE 696) Microcomputer*

This book is a must if you want to design a custom interface between an S-100 microcomputer and almost any type of peripheral device. Mechanical and electrical design is covered, along with logical and electrical relationships, bus interconnections and more. By Sol Libes and Mark Garetz, 322 pages. 6"x9", softbound..... \$16.95

Microprocessor* for Measurement and Control

You'd learn to design mechanical and process equipment using microprocessor-based "real time" computer systems. This book presents plans for prototype systems which allow even those unfamiliar with machine or assembly language to initiate projects. By D.M. Auslander and P. Sagues, 310 pages, 7 3/8"x9 1/4, softbound..... \$15.99

Osborne CP/M* User Guide (Second Edition)

A new revised edition which includes expanded sections on CP/M* 86 and CP/M* 80, as well as CP/M* 's relationship to assembly language programming, MP/M* and CP/NET® operating environments. By Thom Hogan, 292 pages. 6"hx9", softbound..... \$15.95.

Discover FORTH

Whether you are a beginner seeking information on this multi-faceted programming language or a serious programmer already using FORTH, this book is a reference that should not be overlooked. Long considered a computer language of building blocks, FORTH has been optimised for speed and requires little computer support. By Thom Hogan, 146 pages, 6"x9", softbound..... \$16.95

68000 Assembly Language Programming

Each of the 68000's instructions is individually presented and fully explained in this assembly language tutorial. For experienced programmers, this book is also a complete reference to the 68000 instruction set and programming techniques. By Lance A Leventhal, 614 pages. 6-x9, softbound..... \$18.95

Z8000® Assembly Language Programming

This book is filled with real-world programming examples, sample problems, and troubleshooting hints that will guide the reader to mastery of this powerful new 16-bit "super chip". The entire Z8000* instruction set is described in detail. By Lance A Leventhal, Adam Osborne, and Chuek Collins. 928 pages. 6"hx9", softbound..... \$19.99

The 8086 Book

Anyone using, designing, or simply interested in an 8086-based system will be delighted by this book's scope and authority. As the 16-bit microprocessor gains wider inclusion in small computers, this book becomes invaluable as a reference tool which covers the

Books of Interest

Interfacing Microcomputers to the Real World.

by Murray Sargent III and Richard L. Shoemaker.

Published by Addison-Wesley, Inc.

288 pages, 6'4 x 9', soft bound, \$15.55.

This book covers interfacing using the Z80 microprocessor. It starts with the elementary principles of microelectronics and then uses them to explain more complex interfacing concepts. The examples are useful, practical applications including interrupts, real time clocks, stepper motors, A/D and D/A conversion, an interface board, transmitter and receiver circuits (garage door opener), and BSR carrier wave control circuits.

The contents are as follows:

- Chapter 1 Introduction to Digital Logic. The Diode, TTL Gates-AND, NAND, OR, NOR, XOR, The transistor as a Switch, TTL Input/Output Characteristics, Flip-flops, Clocks, Counters, and One shots and Shift Registers.
- Chapter 2 Programming the Z80 Microprocessor. Machine and Assembly Languages, Moving Data in 8-bit Registers and Memory, Manipulating Data —INC, SET, ADD, AND, 16-bit Registers and Memory Pointers, Jumps, Conditional Jumps and Subroutines, Shifty Registers, Input/Output, The Stack and A Tiny Operating System.
- Chapter 3 Processor-Input/Output Interfacing. The Forty Pins, Input/Output Ports, Input/Output Address Decoding, Interrupts, Real Time Clock Interrupt Scheme, Direct Memory Access (DMA), and I/O Example: Multiplexed Keypad/Display.
- Chapter 4 Controlling/Monitoring Various Real World Devices. Switch Closures, Input and Output, Digital to Analog Conversion, Analog to Digital Conversion, Signal Averaging and Lock-in Detection, Waveform Generation and Recognition, Motor Control, and Raster Displays.
- Chapter 5 Serial Input/Output. Parallel/Serial Conversion: the USART, RS232 and Current-loop Conventions, Modems, Computer-Computer Communication Methods, and Rf, Fiber Optics, and Power-line Carriers.
- Chapter 6 Microcomputer Systems. RAM, EPROM, Hard and Floppy Disks, Small Controller Systems, Z80 Computer Busses, and Examples of Larger Microcomputers.

- Chapter 7 Software. Editors, Assemblers, Linkers, Debuggers, Interpreters and Compilers, Disk Operating Systems, and Computer Hierarchies.

- Chapter 8 Hands-On Experience. TTL Logic, Assembly Language Programming, Building an Interface Board, Looking at CPU Signals, Interrupts and Real-time Clocks, Using I/O Ports as Device Controllers, Experiments with DACs and ADCs, Stepper Motors, and Serial and Computer-computer Communications.

- Appendices include the following: Z80 Buses —S-100 Bus Definition, TRS-80 Bus Definition, STD Bus Definition; Computer-Computer Communications— TRSCOM; The DEMON Monitor; Tiny Operating System; Keyboard/Display Routines; and Z80 Instruction Codes.

The entire book is very good, but I found the chapter on hands-on experience especially helpful. This chapter starts with two quotations, "Experiments should be reproducible; they should all fail in the same way," and "Experience is directly proportional to the amount of equipment ruined." These are attributed to Murphy, but anyone who has spent much time at the bench should agree. An example of the practical advice is found on page 190, where they say "Always wire up your circuits with the power OFF. When you are finished wiring, double check that the circuit is wired correctly BEFORE you turn the power on. The alternative is to check the circuit after the power is on and to observe which components have smoke rising from them."

This book is a good blend of theory and practical applications for anyone who wants to interface their Z80 microcomputer to the real world.

Art Carlson

RPN (Reverse Polish Notation)

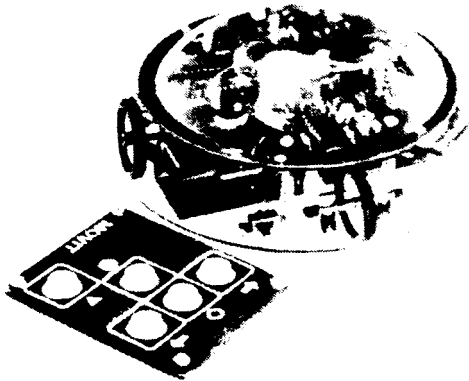
RPN (also known as postfix notation) is commonly used on stack oriented machines, and makes complicated algebraic expressions easier to enter. Using RPN, the operator is listed after the operands instead of between them. For example:

Std. notation	RPN
3×2	3 2 x
$4 + 5$	4 5 +
$7 / 3$	7 3 /
$2 + 4 / 3$	2 4 3 / +

Mew Products

Programmable 4K RAM Robot

A new, low cost, programmable 4K RAM robot introduced by Stock Model Parts is controlled through its seven function teach pendant. The robot includes an on-board CMOS static RAM 256x4 sequencer. Applications include school science projects, robotics courses or personal enjoyment. Of special significance is the fact that the robot is programmable through any of the popular microcomputers (with parallel interface) to gain experience about how real robots are controlled. Buttons on the teach pendant can be used to program the robot to go forward, right, left, pause, sound a buzzer, light a LED lamp, or repeat a program continuously.



The three wheeled robot, called the Memcon Crawler, is offered in kit form. Included are four pages of easy to follow instructions which make it possible to complete the mechanical assembly in about two hours. All electronic elements are contained in two pre-soldered and pre-tested printed circuit boards. The 5 1/2' diameter, 2 1/2' high robot consists of 51 major components plus over 140 fasteners. Among the major parts are two DC motors, rugged blue tinted molded body parts, plus two geartrain assemblies. It is powered by one nine volt and two AA batteries.

The kit is available for \$79.97 postpaid from Stock Model Parts, Division of Designatronics, Inc., 54 South Denton Ave, New Hyde Park, New York 11040.

MicroSolutions' New UniForm

MicroSolutions has developed a product called UniForm, which makes it possible to read and write 5 1/4" CP/M disks in a number of popular formats. UniForm is a program that allows the user to redefine the operating format of the "B" drive of their CP/M computer, opening up a whole new avenue of communication between systems with varying disk formats.

With UniForm, the user has the ability to take a blank

5 1/4" disk and initialize it to any of the supported formats. For example, you could initialize a disk on your KayPro II in the Morrow MD2 format, set your operating format to MD2, write a letter using WordStar, and mail it to someone with an MD2 who could use WordStar on their machine to read it. UniForm works with any files, text, data, or machine code. It does nothing to alter the files at all, and is transparent to the user or program while in operation. Programs with no video attribute or hardware dependent routines can be transferred between machines and run (such as PIP, or MBASIC programs if properly written).

UniForm is completely menu driven and self prompting, with English messages in response to invalid entries. The program was written with a heavy emphasis on ease of use, and as a result, is so straightforward that a manual is practically unnecessary. A manual is provided with UniForm. It explains the program simply and accurately, and provides a complete walk-through of all functions for those who desire it.

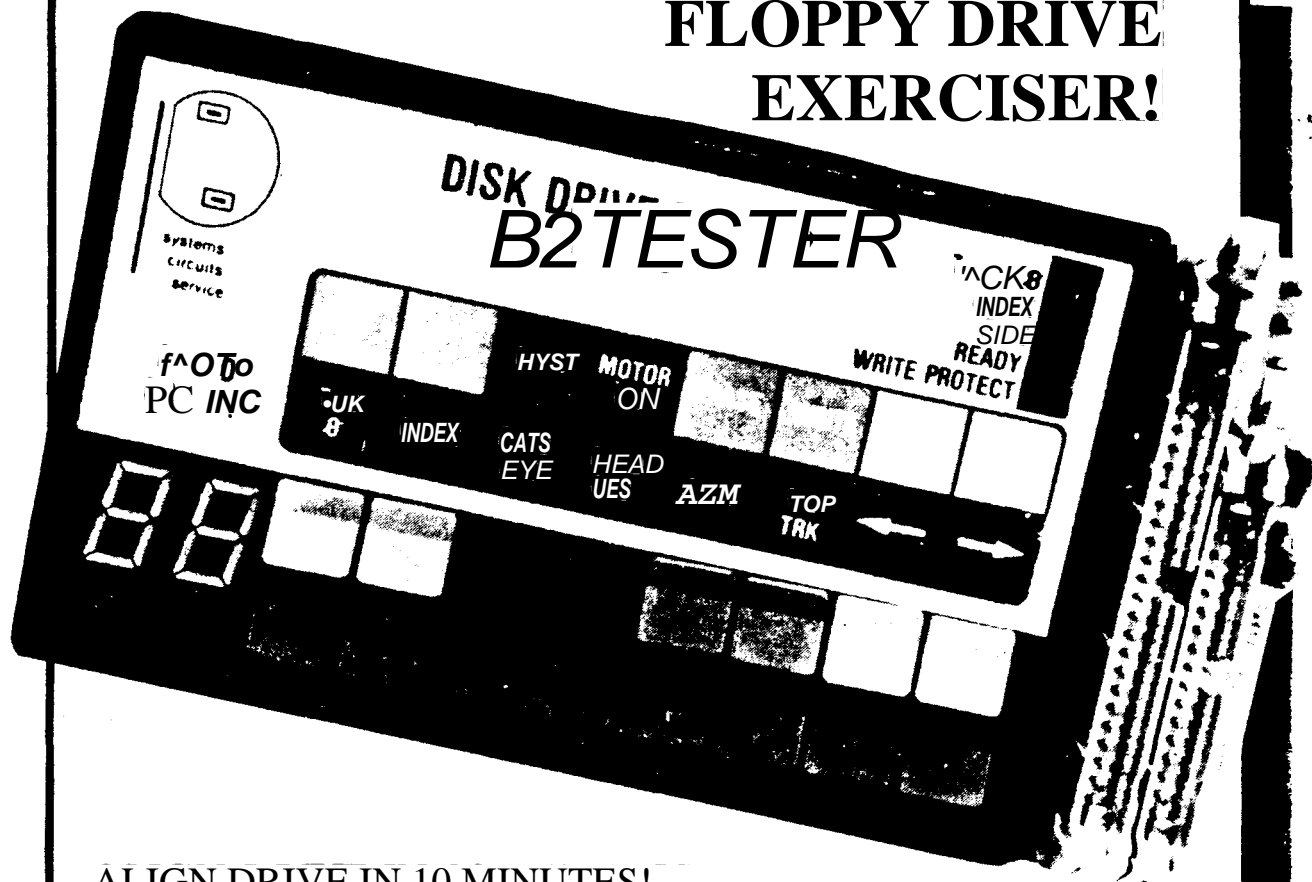
UniForm is able to support almost all 48 track per inch soft sectored CP/M formats. Since it is hardware dependent, versions are available for several of the most popular computers. Currently, there are eleven versions of UniForm: Actrix; Epson QX-10, KayPro II, 4, 4-84, and 10; Morrow Micro Decision; NEC PC-8801; Osborne I (DD); Televideo TS 803/TPC-1; and Zenith Z-100 CP/M-85. Some computers can support more formats than others because of internal hardware characteristics; for example, the Morrow MD UniForm supports 20 single sided formats, 18 double sided formats, and one non-CP/M format (the MSDOS/PCDOS). For a complete list of the formats available with any of these versions, write to MicroSolutions, 126 South Fourth St, DeKalb, IL 60115. UniForm is available from MicroSolutions for \$69.95.

AUTHORS WANTED!

The Computer Journal is interested in technical articles. Query with SASE or send for our Author's Guide.

P0 Box 1697, Kalispell, MT 59903

FLOPPY DRIVE EXERCISER!



ALIGN DRIVE IN 10 MINUTES!

Use with scope and alignment disk (SS \$49, DS \$75)

- SINGLE KEYSTROKE FOR ALL ALIGNMENT TRACKS
- JOG KEYS-MOVE TO ANY TRACK
- INCLUDES "OSBORNE" TYPE POWER HOOKUP
- RUNS ANY STANDARD 34 PIN (5") OR 50 PIN (8") DRIVE
- SHOWS SPEED AND SPEED AVERAGE!
- HYSTERESIS CHECK BUILT IN
- SELECT 5" 48, 96, 100 TPI, OR 8" 48, TPI
- POWER "Y" CABLE=\$10
- DRIVE DATA CABLE \$20

USED BY: IBM, ARMY, NAVY, RCA, ETC...

EX 2000 **\$299**

FREE Air Freight on Prepaid Orders. COD: Add \$5 Plus Shipping

PROTO PC inc. CALL NOW! 612-644-4660

2439 Franklin, St. Paul, MN 55114