S-100
$2.00
U.S.A.

MICROSYSTEMS™

MAY/JUNE 1980 VOL. 1/NO. 3

SORTING PROGRAMS FOR MICROCOMPUTERS

See Pages 35-47

Also in this Issue

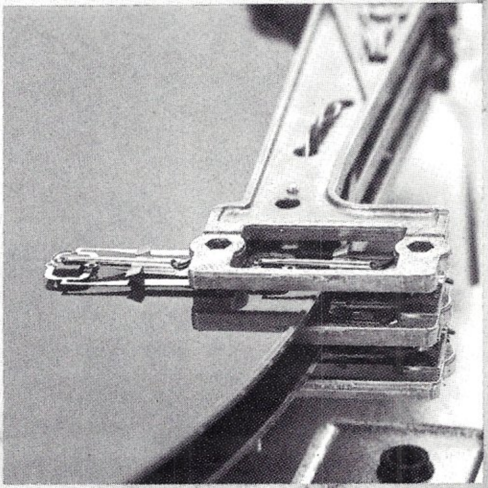
*Registered trademark of Digital Research

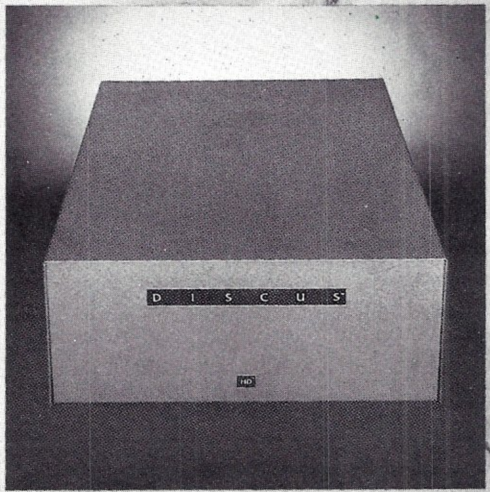
and more

Complete Table of Contents on Page 3

26 MEGABYTES
$4995.

DRIVE A HARD BARGAIN!





Suddenly, S-100 microcomputer systems can easily handle 100 million bytes. Because Morrow Designs™ now offers the first 26 megabyte hard disk memory for S-100 systems—the DISCUS M26™ Hard Disk System.

It has 26 megabytes of useable memory (29 megabytes unformatted). And it's expandable to 104 megabytes.

The DISCUS M26™ system is delivered complete—a 26 megabyte hard disk drive, controller, cables and operating system—for just $4995. Up to three additional drives can be added, $4495 apiece.

The DISCUS M26™ system features the Shugart SA4008 Winchester-type sealed media hard disk drive, in a handsome metal cabinet with fan and power supply.

The single-board S-100 controller incorporates intelligence to supervise all data transfers, communicating with the CPU via three I/O ports (command, status, and data). The controller has the ability to generate interrupts at the completion of each command to increase system throughput. There is a 512 byte sector buffer on-board. And each sector can be individually write-protected for data base security.

The operating system furnished with DISCUS M26™ systems is the widely accepted CP/M* 2.0.

See the biggest, most cost-efficient memory ever introduced for S-100 systems, now at your local computer shop. If unavailable locally, write Morrow Designs,™ 5221 Central Avenue, Richmond, CA 94804. Or call (415) 524-2101, weekdays 10-5 Pacific Time.

*CP/M is a trademark of Digital Research.

MORROW DESIGNS™
Thinker Toys™

S-100 MICRO SYSTEMS™

Volume 1 Number 3 May/June 1980

Editorial Correspondence should be sent to: S-100 MICROSYSTEMS, BOX 1192, Mountainside, NJ 07092.

STAFF

Sol Libes
publisher/editor

Russell Gorr
executive editor

Jacob Epstein
CP/M* editor

Jon Bondy
Pascal editor

Don Libes
assistant editor

Lennie Libes
Susan Libes
subscriptions/office manager

S-100 MICROSYSTEMS is seeking articles on S-100 software, hardware and applications. Program listings should be typed on white paper with a new ribbon. Articles should be typed 40 characters/inch at 10 pitch. Author's name, address and phone number should be included on first page of article and all pages should be numbered. Photos are desirable and should be black and white glossy.

Commercial advertising is welcomed. Write to S-100 MICROSYSTEMS, Box 1192, Mountainside, NJ 07092, or phone Sol Libes at 201-277-2063 after 4 PM EST.

*TMK Digital Research

IN THIS ISSUE

S-100 MICROSYSTEMS (USPS 529-530) is published six times per year for $9.50 per year (U.S.A.) by LIBES, INC., 995 Chimney Ridge, Springfield, N.J. 07081. Controlled circulation postage paid at Westfield, N.J.

POSTMASTER: Send address changes to **S-100 MICROSYSTEMS**, P.O. Box 1192, Mountainside, N.J. 07092.

The Editor's Page
by Sol Libes

The S-100 Bus: Past, Present, and Future

Part I

Reprinted, with permission, from March 17, 1980 issue of INFOWORLD, 530 Lytton Ave., Palo Alto, CA 94301 Subscription $18/yr.

By Sol Libes

This is the first of a two-part article analyzing the S-100-based computer systems picture. The S-100 bus is currently the most widely used microcomputer system bus and hence, I feel, is deserving of an in-depth analysis of where it came from, where it is presently, and what its future looks like. I would like to thank the following individuals who have spoken to me at great length on this topic: Dr. Bob Stewart, IEEE; Bill Godbout, Godbout Electronics; George Morrow, Morrow/Thinker Toys; Steve Edelman, Ithaca InterSystems; and Larry Stein, Computer Mart of New Jersey.

LATE IN 1974, Ed Roberts, then President of a small Albuquerque, New Mexico company by the name of Micro Instrumentation and Telemetry Systems (better known as MITS) called Les Solomon, Technical Editor of *Popular Electronics* magazine. Ed told Les that he had designed a microcomputer system using the new Intel 8080 microprocessor IC, and that MITS wanted to produce it as a kit aimed at hobbyists. MITS was than a small company of about a dozen people who had previously attempted, unsuccesfully, to make and sell radio telemetry kits for model rockets and programmable calculator kits. *Popular Electronics* had helped Ed in promoting these failures in the past; therefore, he turned to them again with his new computer kit project. Two earlier kits based on the Intel 8008 microprocessor had met with some limited acceptance.* MITS, in late 1974, was doing poorly, and the microcomputer kit was, as Ed himself later admitted "a sort of last hope."

Ed projected that they could sell 300 of these computer kits in 1975. Les Solomon thought the project was great and asked Ed to bring the working prototype to New York City for a demonstration and a photo session. *PE* agreed to run a feature construction type article, including schematic diagrams. Ed, with Les' help, dream'nt up a name for the computer. They called it the "Altair 8800." Ed brought the prototype unit to NYC, but something happened in transit, and the unit would not work. Les had faith, and decided to run the article anyway.

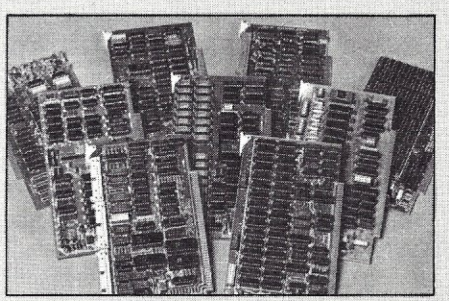
THE REST OF THE STORY is pretty well known. The article appeared in the January 1975 issue of *Popular Electronics,* which was actually published and distributed in December, 1974. At the end of the article it was mentioned that MITS was offering a parts kit for the Altair 880 for $395. At the time, Intel was charging $350 for a single 8080 IC. The Altair price seemed like an absolute steal. Further, MITS offered a complete PC board set for the Altair for only $77, and a complete set of parts (less the cabinet, power supply and front panel switches) for only $189. How cheap could you get?

It was like opening the flood gates. Within one week after the article appeared, MITS had received 200 orders for the Altair; later that year, they received 300 orders in one afternoon. By the end of February, they had 2000 orders and still all they had was one prototype Altair. Working day and night, with the phones constantly jammed, they managed to ship some board sets by early April; in May, they started shipping complete kits.

Those interested in the early history of Personal Computing (1964 to 1974) should consult my article, "The First Ten Years of Amateur Computing," which appeared in the July, 1978, issue of *Byte* magazine.

At Intersystems, "dump" is an instruction. Not a way of life.

(Or, when you're ready for IEEE S-100, will your computer be ready for you?)



We're about to be gadflies again.

While everyone's been busy trying to convince you that large buses housed in strong metal boxes will guarantee versatility and ward off obsolescence, we've been busy with something better. Solving the *real* problem with the first line of computer products *built from the ground up to conform to the new IEEE S-100 Bus Standard*. Offering you extra versatility in 8-bit applications today. And a full 16 bits tomorrow.

We call our new line Series II.™ And even if you don't need the full 24-bit address for up to 16 megabytes (!) of memory right now, they're something to think about. Because of all the performance, flexibility and economy they offer. Whether you're looking at a new mainframe, expanding your present one or upgrading your system with an eye to the future. (Series II boards are compatible with most existing S-100 systems and *all* IEEE S-100 Standard cards as other manufacturers get around to building them.)

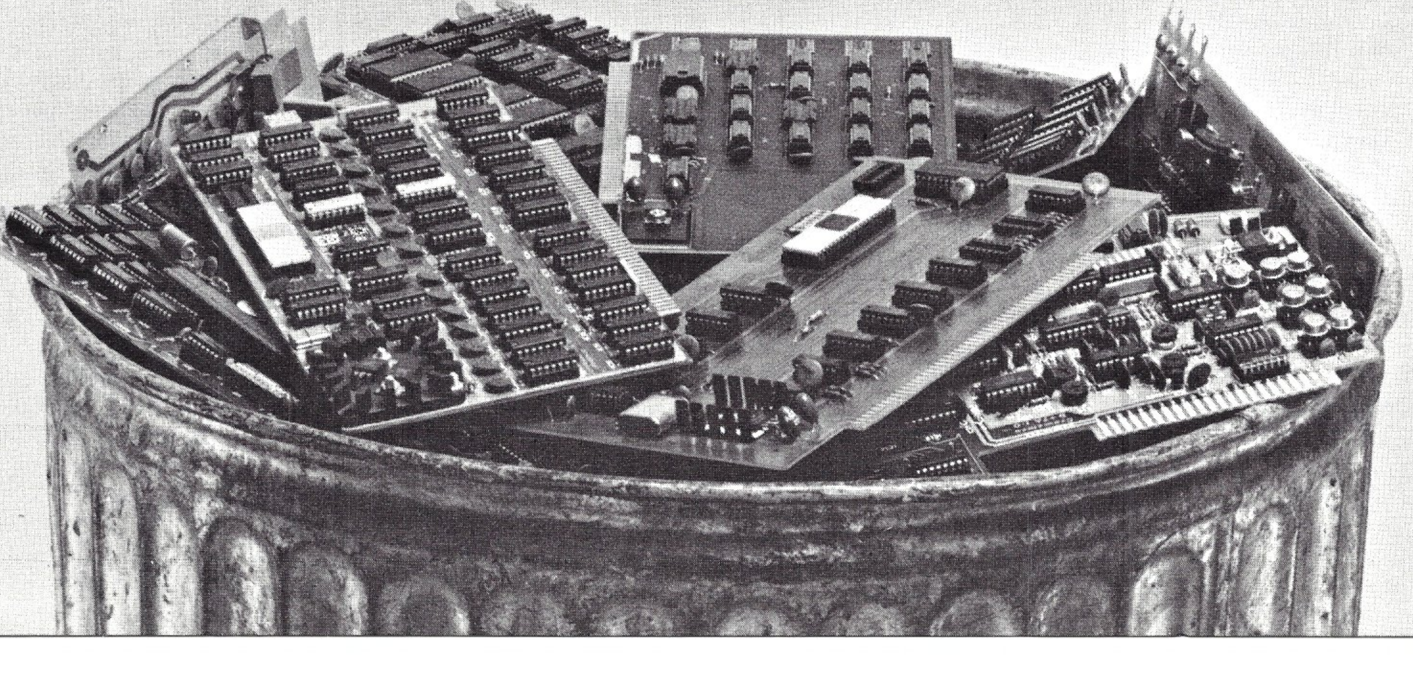
Consider some of the features: Reliable operation to 4MHz and beyond. Full compatibility with 8- and 16-bit CPUs, peripherals and other devices. *Eight* levels of prioritized interrupts. Up to 16 individually-addressable DMA devices, with IEEE Standard overlapped operation. User-selectable functions addressed by DIP-switch or jumpers, eliminating soldering. And that's just for openers.

The best part is that all this heady stuff is available *now*! In our advanced processor—a full IEEE Bus Master featuring Memory Map™ addressing to a full megabyte. Our fast, flexible 16K Static RAM and 64K Dynamic RAM boards. An incredibly versatile and economical 2-serial, 4-parallel Multiple I/O board. 8-bit A/D-D/A converter. Our Double-Density High-Speed Disk Controller. And what is undoubtedly the most flexible front panel in the business. Everything you need for a complete IEEE S-100 system. Available separately, or all together in our new DPS-1 Mainframe!

Whatever your needs, why dump your money into obsolete products labelled "IEEE timing compatible" or other words people use to make up for a lack of product. See the future now, at your Intersystems dealer or call/write for our new catalog. We'll tell you all about Series II and the new IEEE S-100 Bus we helped pioneer. Because it doesn't make sense to buy yesterday's products when tomorrow's are already here.

InterSystems™

Ithaca Intersystems Inc.,
1650 Hanshaw Road/P.O. Box 91,
Ithaca, NY 14850
607-257-0190/TWX: 510 255 4346



THE ALTAIR-8080 used a 100-pin bus that was created by an anonymous draftsman, who selected the connector from a parts catalog and arbitrarily assigned signal names to groups of connector pins. Originally known as the "Altair BUS," its name was changed by other manufacturers of compatible products to the "S-100 Bus." The Altair-8800 came with a 1K-RAM card and promises from MITS of additional boards for I/O interfacing, memory expansion and the like. But the owners of Altairs were desperate for these boards so that they could get their systems to do something.

This led to the introduction of S-100 peripheral plug-in boards by other suppliers. The first company to introduce these boards was a small 2-man operation in a 1,000 square foot shop in Berkeley, California. Named Processor Technology Company, it was run by Gary Ingram and Bob Marsh. Most of their boards were designed by Lee Felsenstein, an independent electronics consultant. Lee's designs included the 3P+S I/O board, which allowed the interfacing of interminals, printers, etc., to the Altair, and the VDM-1, an amazing device which permitted the use of a television monitor to provide alphanumeric and graphics output at very low cost. PTCo also introduced RAM and ROM boards, as well as a software package (appropriately called "Package #1") which made the Altair a real computer rather than just a toy. PTCo also experienced incredible growth.

IN LATE 1975, Bill Gates, Paul Allen, and some cohorts wrote a small Basic language interpreter program in 8080 code with a cross-assembler on a large computer system. They took a paper tape of the program from their location in Seattle down to MITS in Albuquerque, loaded it into an Altair, and with only a few patches, got it to work the same day. Later, Bill and Paul formed MicroSoft, Inc., to market their software directly. The result was that by the end of 1975, a person could build a CPU mainframe for a little over $1,000, to which he could attach a terminal and printer and run Basic, Assembler, Debugger, and even do basic word processing. This enabled MITS, during 1975, to sell about 8,000 Altair-8800 computers.

At the end of 1975, Imsai Manufacturing Corporation introduced their own 8080 CPU which also used the same bus structure and a similar operator panel. More rugged, with a larger power supply, it was considered more professional than the Altair so that, although it cost $100 more, it started to out-sell the Altair to both hobbyists and professional users. Imsai was also started as a garage-type operation by Bob Millard, a consulting electronics engineer.

OVER A DOZEN MORE S-100 board vendors came onto the scene in 1976. Cromemco (started in Harry Garland's garage), building the Dazzler," a color television controller for the Altair and Imsai computers. TDL (Technical Design Labs - later to become XITAN) started in mid 1976 in Roger Amidom's basement with a Z-80 CPU card and a powerful monitor software program. By the end of 1976, there were a half dozen different S-100 mainframes being sold, and close to 30 suppliers of S-100 plug-in boards. Over 30,000 S-100 systems were sold in 1976.

Meanwhile, MITS began thinking of themselves as "the IBM of the microcomputer business." They began to redirect their marketing to commercial and business users; and started to set up a dealer network that sold Altair products exclusively. They introduced a system package for business users.

BUT, MITS LEARNED the hard way that there was a big difference between a hobbyist system and a business system. Now with over 100 employees, MITS was having difficulty developing reliable memory, I/O, and disk storage systems. Even worse, the development of business software was proving an even more formidable task than they had envisioned. By early 1977, Ed Roberts realized that MITS did not have the financial wherewithall for the task. Further, Imsai's better mainframe and the new PTCo Sol computer (named after Les Solomon), Cromemco, and TDL computers were having an impact on the sales of the Altair. An attempt by MITS to broaden its product line with an Altair-6800 computer was a failure. In addition, several products (e.g. a 4K dynamic RAM board) proved very unreliable, and caused an incredible number of returns to the factory.

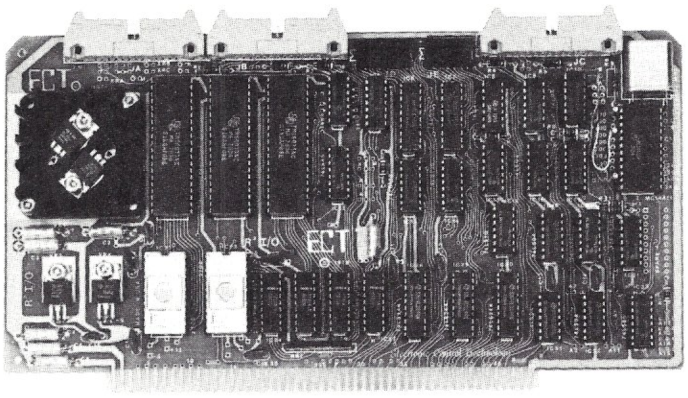
AS MITS SOUGHT to move from kits to assembled business systems, they found that their production problems, restrictive marketing organization (limited to less than 60 dealers by early 1977), and the increased competition were causing financial problems. Therefore, in July 1977, Ed Roberts sold MITS to Pertec Computer Corp. Pertec, a conglomerate with high overhead, raised the Altair prices, stopped all kit production, and ceased all promotion to the personal computer market.

Imsai, PTCo, TDL, Cromemco, North Star, Vectorgraphics, and several other S-100 computer system makers were now selling systems through over 500 personal computer stores world-wide. As Pertec sought to turn the Altair into a serious business oriented system, a dark cloud appeared in the form of low-cost integrated computer systems from Commodore (PET) and Radio Shack (TRS-80), which had become available by the end of 1977. By year-end, Pertec gave up the ghost and ceased production of all Altair products. Despite this event, over 60,000 S-100 systems were sold in 1977

AS MORE AND MORE manufacturers introduced S-100 mainframes and peripheral plug-in boards, it became apparent that compatibility problems were developing. In many cases, S-100 boards would operate in some S-100 systems and not in others. The problems derived from the fact that MITS had only loosely defined the electrical specifications of the bus and had left 19 of the 100 pins undefined. Also, S-100 manufacturers started to look to the future. They realized that some redesign of the S-100 bus was required to accommodate the new 16-bit microprocessors and expanding systems capabilities, i.e., multiprocessing, higher speed operation, and enhanced interrupt vectoring.

The result was that in mid 1978, several companies (most notably Mor-

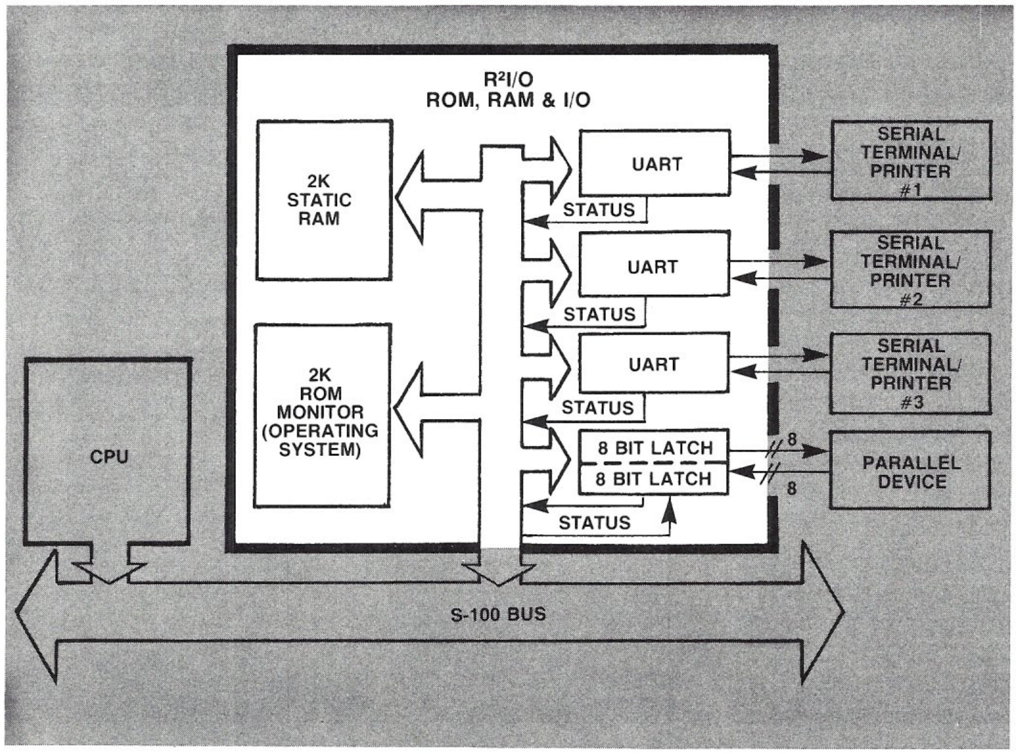
ECT R²I/O... The S-100 ROM, RAM & I/O Board



- S-100 BUS
- 2K ROM
- 2K RAM
- 3 Serial I/O Ports
- 1 Parallel I/O Port
- 4 Status Ports
- ROM Monitor (Operating System)

ELECTRONIC CONTROL TECHNOLOGY's R²I/O is an S-100 Bus I/O Board with 3 Serial I/O Ports (UART's), 1 Parallel I/O Port, 4 Status Ports, 2K of ROM with Monitor Program and 2K of Static RAM. The R²I/O provides a convenient means of interfacing several I/O devices, such as – CRT terminals, line printers, modems or other devices, to an S-100 Bus Microcomputer or dedicated controller. It also provides for convenient Microcomputer system control from a terminal keyboard with the 8080 Apple ROM monitor containing 26 Executive Commands and I/O routines. It can be used in dedicated control applications to produce a system with as few as two boards, since the R²I/O contains ROM, RAM and I/O.

The standard configuration has the Monitor ROM located at F000 Hex with the RAM at F800 Hex and the I/O occupies the first block of 8 ports. Jumper areas provide flexibility to change these locations, within reason, as well as allow the use of ROM's other than the 2708 (e.g. 2716 or similar 24 pin devices). Baud rates are individually selectable from 75 to 9600. Voltage levels of the Serial I/O Ports are RS-232.



8080 APPLE MONITOR COMMANDS

A	Assign I/O
B	Branch to user routine A-Z
C	Undefined
D	Display memory on console in Hex
E	End of file tag for Hex dumps
F	Fill memory with a constant
G	GOTO an address with breakpoints
H	Hex math sum & difference
I	User defined
J	Non-destructive memory test
K	User defined
L	Load a binary format file
M	Move memory block to another address
N	Nulls leader/trailer
O	User defined
P	Put ASCII into memory
Q	Query I/O ports: QI (N)-read I/O; QO(N,V)-send I/O
R	Read a Hex file with checksum
S	Substitute/examine memory in Hex
T	Types the contents of memory in ASCII equivalent
U	Unload memory in Binary format
V	Verify memory block against another memory block
W	Write a checksummed Hex file
X	Examine/modify CPU registers
Y	'Yes there' search for 'N' Bytes in memory
Z	'Z END' address of last R/W memory location

ECT™

Specializing in Quality Microcomputer Hardware
Building Blocks for Microcomputer Systems, Control and Test Equipment
Card Cages, Power Supplies, Mainframes, CPU's, Memory, I/O

ELECTRONIC CONTROL TECHNOLOGY **(201) 686-8080**

763 Ramsey Ave., Hillside, N.J. 07205

No.12: Gourmet Goodies

Software for most popular 8080/Z80 computer disk systems including* **NORTH STAR, iCOM, MICROPOLIS, DYNABYTE DB8/2 & DB8/4, EXIDY SORCERER, SD SYSTEMS, ALTAIR, VECTOR MZ, MECA, 8" IBM, HEATH H17 & H89, HELIOS, IMSAI VDP42 & 44, REX, NYLAC, INTERTEC SUPER-BRAIN, VISTA V80 and V200, TRS-80* MODEL I and MODEL II, ALTOS, OHIO SCIENTIFIC, DIGI-LOG, KONTRON PS180, IMS 5000 DISKETTE** *formats and* **CSSN BACKUP** *cartridge tapes.*

Genuine CP/M for Apple II coming soon! Call for details.

Prices reflect distribution on 8" single density diskettes. If a format is requested which requires additional diskettes, a surcharge of $8. per additional diskette will be added. A surcharge of $25 will be added for software on CSSN format DC 300XL cartridges.

CP/M® VERSION 2 FOR TRS-80 MODEL II NOW AVAILABLE

All Lifeboat programs require CP/M, unless otherwise stated.

Software with Manual / Manual Alone

☐ **CP/M* FLOPPY DISK OPERATING SYSTEM** — Digital Research's operating system configured for many popular micro-computers and disk systems:

System	Version	Price
North Star Single Density	1.4	145/25 v
North Star Double Density	1.4	145/25
North Star Double/Quad	2.x	170/25
iCOM Micro-Disk 2411	1.4	145/25
iCOM 3712	1.4	170/25 v*
iCOM 3812	1.4	170/25 *
Mits 3202/Altair 8800	1.4	145/25
Heath H8 + H17	1.4	145/25 Ⓜ
Heath H89	1.4	145/25 Ⓜ
TRS-80 Model I	1.4	145/25 Ⓜ
TRS-80 Model II	2.x	170/25
Processor Technology Helios II	1.4	145/25
Cromemco System 3	1.4	145/25
Intel MDS Single Density	1.4	145/25
Intel MDS Single Density	2.x	170/25
Intel MDS 800 Double Density	2.x	200/25
Intel MDS 230 Double Density	2.x	200/25
Micropolis Mod I	1.4	145/25 v
Micropolis Mod II	1.4	145/25 v

The following configurations are scheduled for release during the first half of 1980:

System	Version	Price
North Star Double/Quad + Corvus	2.x	250/25
North Star Horizon HD-1	2.x	250/25
Ohio Scientific C3	2.x	200/25
Ohio Scientific C3-B	2.x	250/25
Ohio Scientific C3-C	2.x	250/25
Micropolis Mod II	2.x	200/25
Mostek MDX STD Bus System	2.x	350/25 **
iCOM 3812	2.x	225/25 *
iCOM 4511/Pertec D3000	2.x	375/25 * +
TRS-80 Model II + Corvus	2.x	250/25

*Software consists of the operating system, text editor, assembler, debugger and other utilities for file management and system maintenance. Complete set of Digital Research's documentation and additional implementation notes included. Systems marked * and ** include firmware on 2708 and 2716. Systems marked + include 5440 media charge. Systems marked Ⓜ require the special Ⓜ versions of software in this catalog. Systems marked v have minor variants available to suit console interface of system. Call or write for full list of options.*

☐ **MP/M*** — Intel MDS single density only (Documentation includes CP/M 2.0 manuals) ....**$300/$50**

☐ Ⓜ **Z80 DEVELOPMENT PACKAGE** — Consists of: (1) disk file line editor, with global inter and intra-line facilities; (2) Z80 relocating assembler, Zilog/Mostek mnemonics, conditional assembly and cross reference table capabilities; (3) linking loader producing absolute Intel hex disk file ....**$95/$20**

☐ Ⓜ **ZDT** — Z80 Monitor Debugger to break and examine registers with standard Zilog/Mostek mnemonic disassembly displays. $35 when ordered with Z80 Development Package ....**$50/$10**

☐ **XASM-68** — Non-macro cross-assembler with nested conditionals and full range of pseudo operations. Assembles from standard Motorola MC6800 mnemonics to Intel hex ....**$200/$25**

☐ **XASM-65** — As XASM-68 for MOS Technology MCS-6500 series mnemonics ....**$200/$25**

☐ **DISTEL** — Disk based disassembler to Intel 8080 or TDL/Xitan Z80 source code, listing and cross reference files, Intel or TDL/Xitan pseudo ops optional. Runs on 8080 ....**$65/$10**

☐ Ⓜ **DISILOG** — As DISTEL to Zilog/Mostek mnemonic files. Runs on Z80 only ....**$65/$10**

☐ Ⓧ **SMAL/80** Structured Macro Assembler Language — Package of powerful general purpose text macro processor and SMAL structured language compiler. SMAL is an assembler language with IF-THEN-ELSE, LOOP-REPEAT-WHILE, DO-END, BEGIN-END constructs ....**$75/$15**

☐ Ⓧ **tiny C** — Interactive interpretive system for teaching structured programming techniques. Manual includes full source listings ....**$105/$40**

New Features

☐ Ⓜ Ⓘ **BDS C COMPILER** — Supports most features of language, including Structures, Arrays, Pointers, recursive function evaluation, overlays. Includes linking loader, library manager, and library containing general purpose, file I/O, and floating point functions. Lacks initializers, statics, floats and longs. Documentation includes "The C PROGRAMMING LANGUAGE" by Kernighan and Ritchie ....**$125/$20**

☐ Ⓛ Ⓧ **WHITESMITHS C COMPILER** — The ultimate in systems software tools. Produces faster code than a pseudo-code Pascal with more extensive facilities. Conforms to the full UNIX* Version 7 C language, described by Kernighan and Ritchie, and makes available over 75 functions for performing I/O, string manipulation and storage allocation. Linkable to Microsoft REL files. Requires 60K CP/M ....**$630/$30**

Everything on #12 runs on 64K TRS-80 Model II

MICROSOFT

☐ Ⓛ Ⓜ **BASIC-80** — Disk Extended BASIC, ANSI compatible with long variable names, WHILE/WEND, chaining, variable length file records ....**$325/$25**

☐ Ⓛ Ⓜ **BASIC COMPILER** — Language compatible with BASIC-80 and 3-10 times faster execution. Produces standard Microsoft relocatable binary output. Includes MACRO-80. Also linkable to FORTRAN-80 or COBOL-80 code modules ....**$350/$25**

☐ Ⓛ Ⓜ **FORTRAN-80** — ANSI 66 (except for COMPLEX) plus many extensions. Includes relocatable object compiler, linking loader, library with manager. Also includes MACRO-80 (see below) ....**$425/$25**

☐ Ⓛ Ⓜ **COBOL-80** — Level 1 ANSI '74 standard COBOL plus most of Level 2. Full sequential, relative, and indexed file support with variable file names. STRING, UNSTRING, COMPUTE, VARYING/UNTIL, EXTEND, CALL, COPY, SEARCH, 3-dimensional arrays, compound and abbreviated conditions, nested IF. Powerful interactive screen-handling extensions. Includes compatible assembler, linking loader, and relocatable library manager as described under MACRO-80 ....**$700/$25**

☐ Ⓛ Ⓜ **MACRO-80** — 8080/Z80 Macro Assembler. Intel and Zilog mnemonics supported. Relocatable linkable output. Loader, Library Manager and Cross Reference List utilities included ....**$149/$15**

☐ Ⓛ **XMACRO-86** — 8086 cross assembler. All Macro and utility features of MACRO-80 package. Mnemonics slightly modified from Intel ASM86. Compatibility data sheet available ....**$275/$25**

☐ Ⓛ **EDIT-80** — Very fast random access text editor for text with or without line numbers. Global and intra-line commands supported. File compare utility included. ....**$89/$15**

Price Slashed

☐ Ⓘ **PASCAL/M*** — Compiler generates P code from extended language, implementation of standard PASCAL. Supports overlay structure through additional procedure calls and the SEGMENT procedure type. Provides convenient string handling capability with the added variable type STRING. Untyped files allow memory image I/O. Requires 56K CP/M ....**$150/$20**

☐ Ⓘ **PASCAL/Z** — Z80 native code PASCAL compiler. Produces optimized, ROMable re-entrant code. All interfacing to CP/M is through the support library. The package includes compiler, Microsoft Compatible relocating assembler and linker, and source for all library modules. Variant records, strings and direct I/O are supported. Requires 56K CP/M and Z80 CPU. ....**$395/$25**

New Version New Features

☐ Ⓜ Ⓧ **PASCAL/MT** — Subset of standard PASCAL. Generates ROMable 8080 machine code. Symbolic debugger included. Supports interrupt procedures, CP/M file I/O and assembly language interface. Real variables can be BCD, software floating point, or AMD 9511 hardware floating point. Version 3 includes Enumeration and Record data types. Manual explains BASIC to PASCAL conversion. Source for the runtime package requires Digital Research's MAC. Requires 32K ....**$250/$30**

☐ **ALGOL-60** — Powerful block-structured language compiler featuring economical run-time dynamic allocation of memory. Very compact (24K total RAM) system implementing almost all Algol 60 report features plus many powerful extensions including string handling direct disk address I/O etc. Requires Z80 CPU ....**$199/$20**

☐ Ⓜ **CBASIC-2** Disk Extended BASIC — Non-interactive BASIC with pseudo-code compiler and run-time interpreter. Supports full file control, chaining, integer and extended precision variables, etc. ....**$120/$15**

MICRO FOCUS

☐ Ⓛ **STANDARD CIS COBOL** — ANSI '74 COBOL standard compiler fully validated by U.S. Navy tests to ANSI level 1. Supports many features to level 2 including dynamic loading of COBOL modules and a full*ISAM file facility. Also, program segmentation, interactive debug and powerful interactive extensions to support protected and unprotected CRT screen formatting from COBOL programs used with any • dumb terminal ....**$850/$50**

☐ Ⓛ **FORMS 2** — CRT screen editor. Output is COBOL data descriptions for copying into CIS COBOL programs. Automatically creates a query and update program of indexed files using CRT protected and unprotected screen formats. No programming experience needed. Output program directly compiled by CIS COBOL (standard) ....**$200/$20**

Tasty lower prices!

EIDOS SYSTEMS

☐ Ⓛ **KISS** — Keyed Index Sequential Search. Offers complete Multi-Keyed Index Sequential and Direct Access file management. Includes built-in utility functions for 16 or 32 bit arithmetic, string/integer conversion and string compare. Delivered as a relocatable linkable module in Microsoft format for use with FORTRAN-80 or COBOL-80, etc. ....**$335/$23**

☐ Ⓛ **KBASIC** — Microsoft Disk Extended BASIC with all KISS facilities, integrated by implementation of nine additional commands in language. Package includes KISS. REL as described above, and a sample mail list program ....**$585/$45**
To licensed users of Microsoft BASIC-80 (MBASIC) ....**$435/$45**

☐ **XYBASIC** Interactive Process Control BASIC — Full disk BASIC features plus unique commands to handle bytes, rotate and shift, and to test and set bits. Available in Integer, Extended and ROMable versions.
Integer Disk or Integer ROMable ....**$295/$25**
Extended Disk or Extended ROMable ....**$395/$25**

☐ Ⓜ **BASIC UTILITY DISK** — Consists of: (1) CRUNCH-14 — Compacting utility to reduce the size and increase the speed of programs in Microsoft BASIC and TRS-80 BASIC. (2) DPFUN — Double precision subroutines for computing nineteen transcendental functions including square root, natural log, log base 10, sin, arc sin, hyperbolic sin, hyperbolic arc sin, etc. Furnished in source on diskette and documentation ....**$50/$35**

☐ Ⓜ **STRING/80** — Character string handling plus routines for direct CP/M BDOS calls from FORTRAN and other compatible Microsoft languages. The utility library contains routines that enable programs to chain to a COM file, retrieve command line parameters, and search file directories with full wild card facilities. Supplied as linkable modules in Microsoft format. ....**$95/$20**

STRING/80 source code available separately **$295/n.a.**

☐ **THE STRING BIT** — FORTRAN character string handling. Routines to find, fill, pack, move, separate, concatenate and compare character strings. This package completely eliminates the problems associated with character string handling in FORTRAN. Supplied with source ....**$65/$15**

NEW

☐ Ⓜ **VSORT** — Versatile sort/merge system for fixed length records with fixed or variable length fields. VSORT can be used as a stand-alone package or loaded and called as a subroutine from CBASIC-2. When used as a subroutine, VSORT maximizes the use of buffer space by saving the TPA on disk and restoring it on completion of sorting. Records may be up to 255 bytes long with a maximum of 5 fields. Upper/lower case translation and numeric fields supported. ....**$175/$20**

☐ **CPM/374X** — Has full range of functions to create or re-name an IBM 3741 volume, display directory information and edit the data set contents. Provides full file transfer facilities between 3741 volume data sets and CP/M files ....**$195/$10**

☐ Ⓜ **BSTAM** — Utility to link one computer to another also equipped with BSTAM. Allows file transfers at full data speed (no conversion to hex), with CRC block control check for very reliable error detection and automatic retry. We use it! It's great! Full wildcard expansion to send ★.COM, etc. 9600 baud with wire. 300 baud with phone connection. **Both** ends need one. Standard and Ⓜ versions can talk to one another. ....**$150/$5**

☐ **WHATSIT?*** Interactive data-base system using associative tags to retrieve information by subject. Hashing and random access used for fast response. Requires CBASIC-2 ....**$175/$25**

Lower Price!

☐ Ⓜ † **SELECTOR III-C2** — Data Base Processor to create and maintain multi Key data bases. Prints formatted sorted reports with numerical summaries or mailing labels. Comes with sample applications, including Sales Activity, Inventory, Payables, Receivables, Check Register, and Client/Patient Appointments, etc. Requires CBASIC-2. Supplied in source ....**$295/$20**

☐ **GLECTOR** — General Ledger option to SELECTOR III-C2. Interactive system provides for customized COA. Unique chart of transaction types insure proper double entry bookkeeping. Generates balance sheets, P&L statements and journals. Two year record allows for statement of changes in financial position report. Supplied in source. Requires SELECTOR III-C2, CBASIC-2 and 52K system ....**$250/$25**

NEW

☐ Ⓜ **CBS** — Configurable Business System is a comprehensive set of programs for defining custom data files and application systems without using programming language such as BASIC, FORTRAN, etc. Multiple key fields for each data file are supported. Set-up program customizes system to user's CRT and printer. Provides fast and easy interactive data entry and retrieval **with transaction processing.** Report generator program does complex calculations with stored and derived data, record selection with multiple criteria, and custom formats. Sample inventory and mailing list systems included. **No support language required** ....**$295/$25**

Prices and specifications subject to change without notice.

MICRO DATA BASE SYSTEMS

☐ **HDBS** — Hierarchical Data Base System. CODASYL oriented with FILEs, SETs, RECORDs and ITEMs which are all user defined. ADD, DELETE, UPDATE, SEARCH, and TRAVERSE commands supported. SET ordering is sorted, FIFO, LIFO, next or prior. One to many set relationship supported. Read/Write protection at the FILE level. Supports FILEs which extend over multiple floppy or hard disk devices.

☐ **MDBS** — Micro Data Base System. Full network data base with all features of HDBS plus multi-level Read/Write protection for FILE, SET, RECORD and ITEM. Explicit representation of one to one, one to many, many to many, and many to one SET relationships. Supports multiple owner and multiple record types within SETs. HDBS files are fully compatible.

☐ **MDBS-DRS** — MDBS with Dynamic Restructuring System option which allows altering MDBS data bases when new ITEMs, RECORDs, or SETs are needed without changing existing data.

HDBS-Z80 version ....**$250/$40**
MDBS-Z80 version ....**$750/$40**
MDBS-DRS-Z80 version ....**$850/$50**
8080 Version available at $75. extra.

When ordering, specify one of the languages listed below.

NOTE

HDBS and MDBS manuals purchased alone come without specific language interface manuals. Manuals are available for the following Microsoft languages: 1) MBASIC 4.51, 2) BASIC-80, 5.0, 3) Compiled BASIC-80 or FORTRAN-80, 4) COBOL-80, 5) MACRO-80 ....**$NA/$10**

all Micropro prices are

MICROPRO discounted!

☐ Ⓛ **SUPER-SORT I** — Sort, merge, extract utility as absolute executable program or linkable module in Microsoft format. Sorts fixed or variable records with data in binary, BCD, Packed Decimal, EBCDIC, ASCII, floating & fixed point, exponential, field justified, etc. Even variable number of fields per record! .**$225/$25**

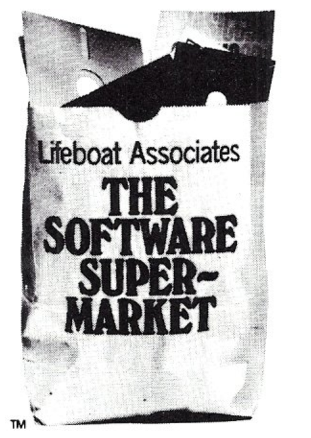
☐ Ⓛ **SUPER-SORT II** — Above available as absolute program only ....**$175/$25**

☐ Ⓛ **SUPER-SORT III** — As II without SELECT/EXCLUDE ....**$125/$25**

☐ Ⓛ **WORD-STAR** — Menu driven visual word processing system for use with standard terminals. Text formatting performed on screen. Facilities for text paginate, page number, justify, center and underscore. User can print one document while simultaneously editing a second. Edit facilities include global search and replace, Read/Write to other text files, block move, etc. Requires CRT terminal with addressable cursor positioning ....**$445/$40**

☐ **WORD-STAR Customization Notes** — For sophisticated users who do not have one of the many standard terminal or printer configurations in the distribution version of WORD-STAR ....**NA/$95**

☐ Ⓛ **WORD-MASTER** Text Editor — In one mode has superset of CP/M's ED commands including global searching and replacing, forwards and backwards in file in video mode, provides full screen editor for users with serial addressable-cursor terminal ....**$125/$25**



Lifeboat Associates, 2248 Broadway, N.Y., N.Y. 10024 **(212) 580-0082** Telex: 220501 TM

Software with Manual / Manual Alone

☐ Ⓜ **POLYVUE/80** — Full screen editor for any CRT with XY cursor positioning. Includes vertical and horizontal scrolling, interactive search and replace, automatic text wrap around for word processing, operations for manipulating blocks of text, and comprehensive 70 page manual .... **$135/$15**

☐ Ⓜ **POLYTEXT/80** — Text formatter for word processing applications. Justifies and paginates source text files. Will generate form letters with custom fields and conditional processing. Support for Daisy Wheel printers includes variable pitch justification and motion optimization .... **$85/$15**

☐ Ⓜ **TEXTWRITER III** — Text formatter to justify and paginate letters and other documents. Special features include insertion of text during execution from other disk files or console, permitting recipe documents to be created from linked fragments on other files. Has facilities for sorted index, table of contents and footnote insertions. Ideal for contracts, manuals, etc. Now compatible with Electric Pencil* prepared files. .... **$125/$20**

New Features

Now-applications software for Microsoft's BASIC interpreter!

PEACHTREE SOFTWARE

☐ Ⓛ † **GENERAL LEDGER** — Records details of all financial transactions. Generates a balance sheet and an income statement. Flexible and adaptable design for both small businesses and firms performing client writeup services. Produces reports as follows: Trial Balance, Transaction Registers, Balance Sheet, Prior Year Comparative Balance Sheet, Income Statement, Prior Year Comparative Income Statement and Department Income Statements. Interactive with other PEACHTREE accounting packages. Supplied in source code for Microsoft BASIC .... **$990/$30**

☐ Ⓛ † **ACCOUNTS PAYABLE** — Tracks current and aged payables and incorporates a check writing feature. Maintains a complete vendor file with information on purchase orders and discount terms as well as active account status. Produces reports as follows: Open Voucher Report, Accounts Payable Aging Report and Cash Requirements. Provides input to PEACHTREE General Ledger. Supplied in source code for Microsoft. BASIC .... **$990/$30**

☐ Ⓛ † **ACCOUNTS RECEIVABLE** — Generates invoice register and complete monthly statements. Tracks current and aged receivables. Maintains customer file including credit information and account status. The current status of any customer account is instantly available. Produces reports as follows: Aged Accounts Receivable, Invoice Register, Payment and Adjustment Register and Customer Account Status Report. Provides input to PEACHTREE General Ledger. Supplied in source code for Microsoft BASIC .. **$990/$30**

☐ Ⓛ † **PAYROLL** — Prepares payroll for hourly, salaried and commissioned employees. Generates monthly, quarterly and annual returns. Prepares employee W-2's. Includes tables for federal withholding and FICA as well as withholding for all 50 states plus up to 20 cities from pre-computed or user generated tables. Will print checks, Payroll Register, Monthly Summary and Unemployment Tax Report. Provides input to PEACHTREE General Ledger. Supplied in source code for Microsoft BASIC .... **$990/$30**

☐ Ⓛ † **INVENTORY** — Maintains detailed information on each inventory item including part number, description, unit of measure, vendor and reorder data, item activity and complete information on current item costs, pricing and sales. Produces reports as follows: Physical Inventory Worksheet, Inventory Price List, Departmental Summary Report, Inventory Status Report, The Reorder Report and the Period-to-Date and Year-to-Date reports. Supplied in source code for Microsoft BASIC .... **$1,190/$30**

☐ Ⓛ † **MAILING ADDRESS** — Keeps track of name and address information and allows the selective printing of this information in the form of mailing lists or address labels. Allows the user to tailor the system to his own particular requirements. User-defined format and print-out system uses a special format file which tells programs how to print the mailing list or address labels. Standard format files are included with system. Automatic sorting of data uses indexed file management routines which allow the name and address information to be sequentially retrieved and printed without file sorting. Supplied in source code for Microsoft BASIC .... **$790/$30**

GRAHAM-DORIAN SOFTWARE SYSTEMS

☐ Ⓛ Ⓜ † **GENERAL LEDGER** — An on-line system; no batching is required. Entries to other GRAHAM-DORIAN accounting packages are automatically posted. User establishes customized C.O.A. Provides transaction register, record of journal entries, trial balances and monthly closings. Keeps 14 month history and provides comparison of current year with previous year. Requires CBASIC-2. Supplied in source ... **$995/$35**

☐ Ⓛ Ⓜ † **ACCOUNTS PAYABLE** — Maintains vendor list and check register. Performs cash flow analysis. Flexible — writes checks to specific vendor for certain invoices or can make partial payments. Automatically posts to GRAHAM-DORIAN General Ledger or runs as stand alone system. Requires CBASIC-2. Supplied in source .... **$995/$35**

☐ Ⓛ Ⓜ † **ACCOUNTS RECEIVABLE** — Creates trial balance reports, prepares statements, ages accounts and records invoices. Provides complete information describing customer payment activity. Receipts can be posted to different ledger accounts. Entries automatically update GRAHAM-DORIAN General Ledger or runs as stand alone system. Requires CBASIC-2. Supplied in source .... **$995/$35**

☐ Ⓛ Ⓜ † **PAYROLL SYSTEM** — Maintains employee master file. Computes payroll withholding for FICA, Federal and State taxes. Prints payroll register, checks, quarterly reports and W-2 forms. Can generate ad hoc reports and employee form letters with mail labels. Requires CBASIC-2. Supplied in source .... **$590/$35**

☐ Ⓛ Ⓜ † **INVENTORY SYSTEM** — Captures stock levels, costs, sources, sales, ages, turnover, markup, etc. Transaction information may be entered for reporting by salesman, type of sale, date of sale, etc. Reports available both for accounting and decision making. Requires CBASIC-2. Supplied in source ... **$590/$35**

☐ Ⓛ Ⓜ † **JOB COSTING** — Designed for general contractors. To be used interactively with other GRAHAM-DORIAN accounting packages for tracking and analysing expenses. User establishes customized cost categories and job phases. Permits comparison of actual versus estimated costs. Automatically updates GRAHAM-DORIAN General Ledger or runs as stand alone system. Requires CBASIC-2. Supplied in source **$995/$35**

Orders must specify disk systems and formats: e.g. North Star single, double or quad density, IBM single or 2D/256, Altair, Helios II, Micropolis Mod I or II, 5¼" soft sector (Micro iCOM/SD Systems Dynabyte), etc.

Prices F.O.B. New York. Shipping, handling and C.O.D. charges extra.

Manual cost applicable against price of subsequent software purchase.

The sale of each proprietary software package conveys a license for use on one system only.

VISA · master charge

Software with Manual / Manual Alone

☐ Ⓛ Ⓜ † **APARTMENT MANAGEMENT SYSTEM** — Financial management system for receipts and security deposits of apartment projects. Captures data on vacancies, revenues, etc. for annual trend analysis. Daily report shows late rents, vacancy notices, vacancies, income lost through vacancies, etc. Requires CBASIC-2. Supplied in source .... **$590/$35**

☐ Ⓛ Ⓜ † **CASH REGISTER** — Maintains files on daily sales. Files data by sales person and item. Tracks sales, over-rings, refunds, payouts and total net deposits. Requires CBASIC-2. Supplied in source ... **$590/$35**

☐ Ⓜ **POSTMASTER** — A comprehensive package for mail list maintenance that is completely menu driven. Features include keyed record extraction and label production. A form letter program is included which provides neat letters on single sheet or continuous forms. Compatible with NAD files. Requires CBASIC-2 .... **$150/$15**

STRUCTURED SYSTEMS GROUP

☐ † **GENERAL LEDGER** — Interactive and flexible system providing proof and report outputs. Customization of COA created interactively. Multiple branch accounting centers. Extensive checking performed at data entry for proof, COA correctness, etc. Journal entries may be batched prior to posting. Closing procedure automatically backs up input files. Now includes Statement of Changes in Financial Position. Requires CBASIC-2 .... **$1250/$25**

☐ † **ACCOUNTS RECEIVABLE** — Open item system with output for internal aged reports and customer-oriented statement and billing purposes. On-Line Enquiry permits information for Customer Service and Credit departments. Interface to General Ledger provided if both systems used. Requires CBASIC-2. .... **$1250/$25**

☐ † **ACCOUNTS PAYABLE** — Provides aged statements of accounts by vendor with check writing for selected invoices. Can be used alone or with General Ledger and/or with NAD. Requires CBASIC-2 .... **$1250/$25**

☐ † **PAYROLL** — Flexible payroll system handles weekly, bi-weekly, semi-monthly and monthly payroll periods. Tips, bonuses, re-imbursements, advances, sick pay, vacation pay, and compensation time are all part of the payroll records. Prints government required periodic reports and will post to multiple SSG General Ledger accounts. Requires CBASIC-2 and 54K of memory .... **$1250/$25**

☐ † **INVENTORY CONTROL SYSTEM** — Performs control functions of adding and depleting stock items, adding new items and deleting old items. Tracks quantity of items on hand, on order and back-ordered. Optional hard copy audit trail is available. Reports include Master Item List, Stock Activity, Stock Valuation and Re-order List. Requires CBASIC-2 **$1250/$25**

☐ † **ANALYST** — Customized data entry and reporting system. User specifies up to 75 data items per record. Interactive data entry, retrieval, and update facility makes information management easy. Sophisticated report generator provides customized reports using selected records with multiple level break-points for summarization. Requires CBASIC-2 .... **$250/$15**

☐ **LETTERIGHT** — Program to create, edit and type letters or other documents. Has facilities to enter, display, delete and move text, with good video screen presentation. Designed to integrate with NAD for form letter mailings. Requires CBASIC-2 .. **$200/$25**

☐ **NAD** Name and Address selection system — interactive mail list creation and maintenance program with output as full reports with reference data or restricted information for mail labels. Transfer system for extraction and transfer of selected records to create new files. Requires CBASIC-2 .... **$100/$20**

☐ **QSORT** — Fast sort/merge program for files with fixed record length, variable field length information. Up to five ascending or descending keys. Full back-up of input files created .... **$100/$20**

★★★★★★★★

CONDIMENTS

☐ **HEAD CLEANING DISKETTE**—Cleans the drive Read/Write head in 30 seconds. Diskette absorbs loose oxide particles, fingerprints, and other foreign particles that might hinder the performance of the drive head. Lasts at least 3 months with daily use. Specify 5" or 8".

NEW
Single sided .... **$20 each/$55 for 3**
Double sided .... **$25 each/$65 for 3**

☐ **FLIPPY DISK KIT** — Template and instructions to modify single sided 5¼" diskettes for use of second side in single sided drives **$12.50**

☐ **FLOPPY SAVER** — Protection for center holes of 5" and 8" floppy disks. Only 1 needed per diskette. Kit contains centering post, pressure tool and tough 7 mil mylar reinforcing rings for 25 diskettes.

NEW
5", Kit **$14.95**
5", Rings only **$7.95**
8", Kit **$16.95**
8", Rings only **$8.95**

☐ **PASCAL USER MANUAL AND REPORT** — By Jensen and Wirth. The standard textbook on the language. Recommended for use by Pascal/Z, Pascal/M and Pascal/MT users **$10**

☐ **THE C PROGRAMMING LANGUAGE** — By Kernighan and Ritchie. The standard textbook on the language. Recommended for use by BDS C, tiny C, and Whitesmiths C users **$12**

NEW
☐ **STRUCTURED MICROPROCESSOR PROGRAMMING** — By the authors of SMAL/80. Covers structured programming, the 8080/8085 instruction set and the SMAL/80 language **$20.00**

NEW
☐ **ACCOUNTS PAYABLE & ACCOUNTS RECEIVABLE—CBASIC** — By Osborne/McGraw-Hill **$20**

☐ **GENERAL LEDGER—CBASIC** — By Osborne/McGraw-Hill **$20**

★★★★★★★★

Hearty Appetite.

*CP/M and MP/M are trademarks of Digital Research.
Z80 is a trademark of Zilog, Inc.
UNIX is a trademark of Bell Laboratories.
WHATSIT? is a trademark of Computer Headware.
Electric Pencil is a trademark of Michael Shrayer Software.
TRS-80 is a trademark of Tandy Corp.
Pascal/M is a trademark of Sorcim.

†Recommended system configuration consists of 48K CP/M, 2 full size disk drives, 24 x 80 CRT and 132 column printer.

Ⓜ Modified version available for use with CP/M as implemented on Heath and TRS-80 Model I computers.

Ⓛ User license agreement for this product must be signed and returned to Lifeboat Associates before shipment may be made.

Ⓘ Ⓧ This product Includes/eXcludes the language manual recommended in Condiments.

™The Software Supermarket is a trademark of Lifeboat Associates

row/Thinker Toys, Parasitic Engineering, and Ithaca InterSystems) began development, under the aegis of the IEEE (Institute of Electrical & Electronic Engineers), of an S-100 Bus Standard**. All in all, 1978 was another glorious year for S-100 system producers as nearly 100,000 S-100 systems were manufactured.

THE YEAR 1979 WAS DISASTROUS for three of the leading S-100 manufacturers. A tight money market combined with bad marketing decisions and manufacturing problems led to Imsai going bankrupt and PTCo closing their doors (even though they were financially solvent). Polymorphics filed for bankruptcy but was able to get additional financing, go through reorganization, and by mid-year, turned around and came out of bankruptcy. Imsai was purchased by the Fisher-Freitas Corporation, who have resumed manufacturing and marketing of the entire Imsai product line.

On the other hand, 1979 proved to be another excellent year for most S-100 system suppliers: six new S-100 mainframes were introduced, making a total of 17 companies manufacturing S-100 mainframes. Nearly 60 companies were manufacturing S-100 plug-in boards, and over 140 companies offered S-100 software packages. Although the increase in the sales of S-100 hardware was significant in 1979, it was not the dramatic 100% to 200% increases of prior years. On the other hand, S-100 software sales skyrocketed. Total number and dollar figures are difficult to obtain, since so many manufacturers are involved. However, there is little doubt that presently, S-100 type computer systems are more widespread than any other type of computer system.

In the second, and concluding, part of this article, I will analyze the present state and future prospects of the S-100 marketplace. ■

**This proposed standard is 25 pages long, nearing adoption, and has been printed in *Computer* (July 1979) and *S-100 Microsystems* (Jan-Feb 1980) magazines.

NEWS & VIEWS

by Sol Libes

S-100 PASCAL MICROENGINE BOARD SET SOON

Digicomp Research Corp., Terrace Hill, Ithaca, NY 14850, will shortly start shipping an S-100 plug in board set (two boards) using the Western Digital PASCAL MICROENGINE™ chipset. It will directly execute the UCSD™ P-code (version III.0) and promises 7 to 12 times speedup over software PASCALs. A Z80 mpu is included to run CP/M™ and I/O. DRC is selling pre-production units for $750 and the actual production units to be available later this year should be twice this price.

PL-1 COMPILER ANNOUNCED

Digital Research, the people who created CP/M™, MP/M, etc., have announced a full implementation of the PL-1 language for 8080/Z80 based systems. They claim that the compiler will generate compiled code which takes fewer bytes and runs faster than the same program written in PASCAL.

FULL UNIX™ RUMORED

Microsoft has disclosed that they are very close to signing a contract with Bell Laboratories to distribute UNIX™. It will include a C-compiler. They will write versions to run on 8086, Z8000 and 68000 based systems.

™UCSD PASCAL is a trademark of the Regents of the University of California.
™CP/M is a trademark of Digital Research Corporation.
™PASCAL MICROENGINE is a trademark of Western Digital Corporation.
™UNIX is a trademark of Bell Laboratories.

PASCAL NEWS

UCSD did do one thing to pacify all those PASCAL owners and clubs whose software license was arbitrarily terminated. It provided an offer to owners of Version 1.4 of UCSD PASCAL an upgrade to Version II at a charge of $95 instead of the usual $300. However, clubs and owners received very short notice, about 4 weeks, and therefore the offer expired before several clubs were able to notify their members.

Softech, the distributor of UCSD PASCAL, is starting a national user's group for UCSD PASCAL. They will distribute software (a la CP/M User Group) and expect to publish a newsletter. They are talking about a meeting of UCSD Pascal users sometime this summer in La Jolla, CA.

Jim McCord, publisher of the UCSD PASCAL HOBBY NEWSLETTER, has now released disk #3 (LIB.3) of UCSD PASCAL software. You can get a copy for $5 + disk or $9 if he supplies disk. Write to: Jim McCord, 330 Vereda Leyenda, Goleta, CA 93017.

ANNOUNCEMENTS

5th ANNUAL CALIFORNIA COMPUTER SWAP MEET
SUNDAY, JUNE 1st - 10 AM to 6 PM
Santa Clara County Fairgrounds, Gateway Hall
344 Tully Road (West on Tully Rd off 101)
San Jose, California
Selling Spaces: $25 & $55 (non-commercial)
$60 & $130 (commercial)
Admision: Free
Consignment Table: 8% fee
Free Literataure Table
For information call: John Craig (415) 324-2404
or write: Box 52, Palo Alto, CA 94302

3rd ANNUAL PERSONAL COMPUTER ARTS FESTIVAL
SATURDAY & SUNDAY - August 23 & 24
Philadelphia, PA
Call for computer musicians and artists to participate. Write to:
PCAF'80, c/o Philadelphia Area Computer Society
Box 1954, Philadelphia, PA 19105

3rd ANNUAL PERSONAL COMPUTER FAIR
NOVEMBER 8 & 9
Pacific Science Center
Seattle, Washington
For information call: (206) 284-6109
or write: Northwest Computer Society, Box 4193, Seattle, WA 98119

BACK ISSUES OF S-100 MICROSYSTEMS

Did you miss the previous issues of **S-100 Microsystems**? They are still available. Vol. 1, No. 1 is already a collector's item; we only have a small supply which we expect will be exhausted by the end of April (we are considering reprinting it).

The price is $2.00 each or $3.50 for both. Add $1.00 to cover postage and handling.

Vol. 1, No. 1 - Jan/Feb 1980
The IEEE S-100 Standard (complete)
An Introduction to CP/M, Part I
Modifying the SDS VDB-8024 Display Card
Computerized Bulletin Board Systems
An 8080 Disassembler (complete source code)

Vol. 1, No. 2 - March/April 1980
North Star Topics, Part I
Linear Programming in Pascal, Part I
Introduction to CP/M, Part II
Addressing the Cursor, Part I
S-100 Bus - New versus Old
Product Review - CGS-808 Color Graphics Board
Tarbell Disk Controller Mods

NEW! TPM* for TRS-80 Model II
NEW! System/6 Package
Computer Design Labs

Z80* Disk Software

We have acquired the rights to all TDL software (& hardware). TDL software has long had the reputation of being the best in the industry. Computer Design Labs will continue to maintain, evolve and add to this superior line of quality software.

— Carl Galletti and Roger Amidon, owners.

Software with Manual/Manual Alone

All of the software below is available on any of the following media for operation with a Z80 CPU using the CP/M* or similar type disk operating system (such as our own TPM*).

for TRS-80* CP/M (Model I or II)
for 8" CP/M (soft sectored single density)
for 5¼" CP/M (soft sectored single density)
for 5¼" North Star CP/M (single density)
for 5¼" North Star CP/M (double density)

BASIC I

A powerful and fast Z80 Basic interpreter with EDIT, RENUMBER, TRACE, PRINT USING, assembly language subroutine CALL, LOADGO for "chaining", COPY to move text, EXCHANGE, KILL, LINE INPUT, error intercept, sequential file handling in both ASCII and binary formats, and much, much more. It runs in a little over 12 K. An excellent choice for games since the precision was limited to 7 digits in order to make it one of the fastest around. $49.95/$15.

BASIC II

Basic I but with 12 digit precision to make its power available to the business world with only a slight sacrifice in speed. Still runs faster than most other Basics (even those with much less precision). $99.95/$15.

BUSINESS BASIC

The most powerful Basic for business applications. It adds to Basic II with random or sequential disk files in either fixed or variable record lengths, simultaneous access to multiple disk files, PRIVACY command to prohibit user access to source code, global editing, added math functions, and disk file maintenance capability without leaving Basic (list, rename, or delete). $179.95/$25.

ZEDIT

A character oriented text editor with 26 commands and "macro" capability for stringing multiple commands together. Included are a complete array of character move, add, delete, and display function. $49.95./$15.

ZTEL

Z80 Text Editing Language - Not just a text editor. Actually a language which allows you to edit text and also write, save, and recall programs which manipulate text. Commands include conditional branching, subroutine calls, iteration, block move, expression evaluation, and much more. Contains 36 value registers and 10 text registers. Be creative! Manipulate text with commands you write using Ztel. $79.95/$25.

TOP

A Z80 Text Output Processor which will do text formatting for manuals, documents, and other word processing jobs. Works with any text editor. Does justification, page numbering and headings, spacing, centering, and much more! $79.95/$25.

MACRO I

A macro assembler which will generate relocateable or absolute code for the 8080 or Z80 using standard Intel mnemonics plus TDL/Z80 extensions. Functions include 14 conditionals, 16 listing controls, 54 pseudo-ops, 11 arithmetic/logical operations, local and global symbols, chaining files, linking capability with optional linker, and recursive/reiterative macros. This assembler is so powerful you'll think it is doing all the work for you. It actually makes assembly language programming much less of an effort and more creative. $79.95/$20.

MACRO II

Expands upon Macro I's linking capability (which is useful but somewhat limited) thereby being able to take full advantage of the optional Linker. Also a time and date function has been added and the listing capability improved. $99.95/$25.

LINKER

How many times have you written the same subroutine in each new program? Top notch professional programmers compile a library of these subroutines and use a Linker to tie them together at assembly time. Development time is thus drastically reduced and becomes comparable to writing in a high level language but with all the speed of assembly language. So, get the new CDL Linker and start writing programs in a fraction of the time it took before. Linker is compatible with Macro I & II as well as TDL/Xitan assemblers version 2.0 or later. $79.95/$20.

DEBUG I

Many programmers give up on writing in assembly language even though they know their programs would be faster and more powerful. To them assembly language seems difficult to understand and follow, as well as being a nightmare to debug. Well, not with proper tools like Debug I. With Debug I you can easily follow the flow of any Z80 or 8080 program. Trace the program one step at a time or 10 steps or whatever you like. At each step you will be able to see the instruction executed and what it did. If desired, modifications can then be made before continuing. It's all under your control. You can even skip displaying a subroutine call and up to seven breakpoints can be set during execution. Use of Debug I can pay for itself many times over by saving you valuable debugging time. $79.95/$20.

DEBUG II

This is an expanded debugger which has all of the features of Debug I plus many more. You can "trap" (i.e. trace a program until a set of register, flag, and/or memory conditions occur). Also, instructions may be entered and executed immediately. This makes it easy to learn new instructions by examining registers/memory before and after. And a RADIX function allows changing between ASCII, binary, decimal, hex, octal, signed decimal, or split octal. All these features and more add up to give you a very powerful development tool. Both Debug I and II must run on a Z80 but will debug both Z80 and 8080 code. $99.95/$20.

ZAPPLE

A Z80 executive and debug monitor. Capable of search, ASCII put and display, read and write to I/O ports, hex math, breakpoint, execute, move, fill, display, read and write in Intel or binary format tape, and more! on disk $34.95/$15.

APPLE

8080 version of Zapple $34.95/$15.

NEW! TPM now available for TRS-80 Model II!

TPM*

A NEW Z80 disk operation system! This is not CP/M*. It's better! You can still run any program which runs with CP/M* but unlike CP/M* this operating system was written specifically for the Z80* and takes full advantage of its extra powerful instruction set. In other words its not warmed over 8080 code! Available for TRS-80* (Model I or II). Tarbell, Xitan DDDC, SD Sales "VERSA-FLOPPY", North Star (SD&DD), and Digital (Micro) Systems. $79.95/$25.

SYSTEM MONITOR BOARD (SMB II)

A complete I/O board for S-100 systems. 2 serial ports, 2 parallel ports, 1200/2400 baud cassette tape interface, sockets for 2K of RAM, 3-2708/2716 EPROM's or ROM, jump on reset circuitry. Bare board $49.95/$20.

ROM FOR SMB II

2KX8 masked ROM of Zapple monitor. Includes source listing $34.95/$15.

PAYROLL (source code only)

The Osborne package. Requires C Basic 2.
5" disks $124.95 (manual not included)
8" disks $ 99.95 (manual not included)
Manual $20.00

ACCOUNTS PAYABLE/RECEIVABLE (source code only)

By Osborne, Requires C Basic 2
5" disks $124.95 (manual not included)
8" $99.95 (manual not included)
Manual $20.00

GENERAL LEDGER (source code only)

By Osborne. Requires C Basic 2
5" disks $99.95 (manual not included)
8" disks $99.95 (manual not included)
Manual $20.00

C BASIC 2

Required for Osborne software. $99.95/$20.

SYSTEM/6

TPM with utilities, Basic I interpreter, Basic E compiler, Macro I assembler, Debug I debugger, and ZEDIT text editor.
Above purchased separately costs $339.75
Special introductory offer: Only $179.75 with coupon!!

$160.

This Coupon Is Worth One Hundred And Sixty Dollars Toward The Full Price Of The SYSTEM/6 Package System/6 with this coupon is only $179.95. This is a limited time offer.

$160.00

ORDERING INFORMATION

Visa, Master Charge and C.O.D. O.K. To order call or write with the following information.

1. Name of Product (e.g. Macro I)
2. Media (e.g. 8" CP/M)
3. Price and method of payment (e.g. C.O.D.) include credit card info. if applicable.
4. Name, Address and Phone number.
5. For TPM orders only: Indicate if for TRS 80, Tarbell, Xitan DDDC, SD Sales (5¼" or 8"). ICOM (5¼" or 8"), North Star (single or double density) or Digital (Micro) Systems.
6. N.J. residents add 5% sales tax.

Manual cost applicable against price of subsequent software purchase in any item except for the Osborne software.

For information and tech queries call
609-599-2146

For phone orders ONLY call toll free
1-800-327-9191
Ext. 676
(Except Florida)

OEMS

Many CDL products are available for licensing to OEMs. Write to Carl Galletti with your requirements.

* Z80 is a trademark of Zilog
* TRS-80 is a trademark for Radio Shack
* TPM is a trademark of Computer Design Labs. It is not CP/M*
* CP/M is a trademark of Digital Research

Prices and specifications subject to change without notice.

DEALER INQUIRIES INVITED.

COMPUTER DESIGN LABS

342 Columbus Avenue
Trenton, N.J. 08629

AN INTRODUCTION TO CP/M—Part 3

by

Jake Epstein
Box 571
Pittsfield, Ma. 01201

CCP FUNCTIONS

In this month's article, the third in a series on the CP/M operating system, I will be discussing the practical matter of console operation of CP/M. I have also included a section on mass-storage configurations available to CP/M users.

Once the CP/M operating system is 'booted up', the user has two options that can exercised. One is to execute the various commands inherent in the CCP, (CONSOLE COMMAND PROCESSOR). The other is to execute a program that has been stored as a file on the disk. While functioning in the CCP mode, the syntax of CP/M, as discussed in Article II, will prevail, but once a program is executed, then console syntax may change.

The 7 commands built into the CCP are shown in Table 1:

TABLE 1 - CCP COMMANDS

COMMAND	TYPE	FUNCTION
ERA	Alter	Erase a FCB in the directory
DIR	Non-alter	List files in the directory
REN	Alter	Rename a file
SAVE	Alter	Save memory image as a file
TYPE	Non-alter	Type contents of a file
(LOAD FILE-EXECUTE)	Non-alter	Load file in TPA then execute code at 100h
USER	Non-alter	Set user number, ver. 2.0 only

In the above list, functions that alter will change contents of a disk, and thus, care must used when exercizing commands that do so or data may be lost. Once data has been erased, it cannot be recovered so an important chore that users must do is make backup copies of files that are important in case of accident or mistake in command usage. More on this later.

Before explaining each built-in command, I will first describe disk log-in commands. As described in Article I, when the system is initially booted up, the prompt A> appears. This indicates that as far as the operating system is concerned, the storage device named A is online and ready to function as commanded by the user via the CCP. In the computer field, two terms are used to describe I/O devices: LOGICAL and PHYSICAL. Physical is a term referring to the device as it actually ocurrs in the real world. Logical refers to devices as they are seen by software. The following list should clarify the differences.

PHYSICAL	LOGICAL
8 inch floppy disk	A:
5.25 inch floppy disk	B:
1600 bpi mag-tape	C:
CRT	CON
ASR 33 teletype	LST
Paper tape	RDR

When there are several physical devices of the same type, then numbers are used beginning with 0. In other words, drive 0, drive 1, drive 2, and drive 3 would be the physical devices in a computer system with 4 floppy disk units. On the other hand, when the the user wants to access any of these via the operating system, then the logical device name is used. The value of this is that physical matters are taken care of by hardware/ software interfaces found in the operating system leaving the user free to concentrate on other functions that use the logical devices.

In CP/M 1.4, BDOS (BASIC DISK OPERATING SYSTEM) and BIOS (BASIC INPUT/OUTPUT System) both contain software that is dependent of disk type, density, and size. As discussed last month, sector skew is a function determined in BDOS thus CP/M for 5.25 inch disks will not function with 8 inch and vice-versa. Also, all disks in a system have to be compatible with the mixing of disk types impossible. A big advantage of CP/M 2.0 is that a section of BIOS contains tables that are used to describe each physical device in the system. Thus any number and/or type of mass storage device could be utilized as long as hardware and software interfacing is implemented for each device in the BIOS. The following mass storage list is feasible with CP/M 2.0:

LOGICAL	**PHYSICAL**	**APPROX CAPACITY IN BYTES**
A:	Double density floppy disk 0	500k
B:	Double density floppy disk 1	500k
C:	Double density 5 inch floppy	150k
D:	Hard disk	20meg
E:	Single density floppy disk 0	256k
F:	Single density floppy disk 1	256k

In the above list, there is an example of one physical device, floppy disk 0, having two logical names, A: and E:. This was done because dual density floppy disk controllers can read/write in either single or double density. This implementation gives a means for easily transfering information from single to double density or vice versa.

When using any version of CP/M, disk drives are logged-in at the CCP by simply typing the logical name followed by a colon and carriage return (cr). In the above system, to log-in floppy disk 0 in single density mode the following is typed:

A>E:	User types E: (cr)
E:	System response

When naming files, the logical device where the file is located is indicated by placing the device name in front of the file name:

B:STAT.COM	File STAT.COM on device B:

If the logical device is not given, then the logged-in device is used.

In this article, I will limit the discussion of other I/O devices to just the console (logical-CON:) and the hardcopy device (logical-LST:). When I discuss user implementation of BIOS functions and advanced uses of the STAT and PIP utility program, then I will describe other physical-logical device pairings available in CP/M.

In order to determine which files have fcb (file control block) entries in the directory, the DIR command is used. In ver. 1.4 typing DIR(cr) will give a listing of all the files that have fcbs. In ver 1.4 these files are simply listed in order vertically on the console device. In version 2.0, however, file names are listed in rows of 4 names on the console. By using file names, wild card functions, and logical device names, the following command string variations are possible:

DIR TEST.COM	Find and list file name
DIR B:DDT.*	List all files on device B: with primary name DDT
DIR *.??M	List all files that have M as last character of secondary name
DIR E:	List all files on E:
DIR A???.COM	Find COM files with primary name of 4 characters with A as first character

In naming files, remember that secondary names are not necessary, but primary names are. Also, one space is used between names and commands. The prompt, NO FILE, is printed when the DIR command does not find a file or group of files. Finally, ver. 2.0 allows the user to designate files as SYS (System) files so that when the DIR function is given, they will not be listed in the directory. The ability to implement this option is a function of the STAT utility program and will be discussed later.

The TYPE function will read a specified file from a disk and print it on the console device. Since console devices interpret information sent to them as ASCII data, only ASCII format files will give proper print although any file type can be used. This function will read and print an entire file up to the EOF (End of file) delimiter which is cntr-z (1Ah) in CP/M. Wild card functions are not permitted. Typing a 'space' while a file is being listed will abort the TYPE function and return control to the CCP. This is also true of the DIR command.

The REN function is used to change the name of a file. The command syntax is:

```
REN HELLO.COM=TEST.ASM
```

In this case, file name TEST.COM is changed to HELLO.COM. Wild card functions are not allowed.

The ERA function is used to erase fcb entries in the directory on a disk. The data itself is not erased but the space that it occupies on the disk may be used when other files are created at a later time. If a fcb is removed, it is normally impossible to retrieve the data unless directory information is stored elswhere. In a later article I will discuss deciphering fcb information so that the user can reconstruct files when directory entries are lost. The ERA function uses wild cards so the following variations are possible:

ERA *.ASM	Erase all ASM files
ERA C:DUMP.COM	Erase file on device C:
ERA TEST?.*	Erase all TEST files with extra character in primary name
ERA *.*	Erase all files.

When using the *.* file name, the CCP will ask for verification by typing 'ALL FILES (Y OR N)?' in which case the user has to type Y for the function to occur. Any other character causes the function to abort.

The SAVE command is used to store an image of memory starting at location 100h, start of TPA (Transient Program Area), as a COM file. Article I in this series contains a description of the TPA. Although the beginning location of the data to be saved is always 100h, the user signifies the size of the memory image.

CP/M uses three terms that signify differing amounts of memory. The record as described in previous articles is given as 128 (80h) bytes and is equal to the size of a single sector on a single density floppy disk. A page of memory is equal to 256 (100h) bytes and is thus two records in length. Remembering that location 00h is a position, the first page of memory is from 00-FFh, the second page is from 100-200h and so on. Thus in a computer whose address bus is 16 bits, (2 bytes), each page is addressed by all of 8 bit combinations of the lower byte with one value of the upper byte. Thus there are 256 pages in a 16 bit machine. The term block is used to describe 2 records or 256 bytes of data. Since block and page in this context have the same value, it is important to remember that page refers to memory addresses but block refers to an amount of data. Page almost always is equal to 256 but block as well as record can have other sizes when working with different operating systems. A final point is that when dealing with data in these sizes as determined by hardware, the user is working with physical concepts. Records, pages, and/or blocks can take on differing values when one is dealing in logical concepts. For example, a record in a data base system could be made up of a person's name, his/her pay scale, and address. This logical unit may need one or more records of physical space on disk.

The syntax of the SAVE command is as follows:

```
SAVE 12 D:HELP.TEX
```

In this case 12 is the number of blocks that are to be saved, and is entered in decimal values. The user has to convert hexadecimal locations into decimai blocks. Only an even number of sectors are used, so there will be times when even though one sector of data needs to be saved, the file will be 2 sectors long. Actually this does not prove to be wasteful of disk space, because as discussed in Article II, the smallest unit that can be handled by BDOS is a cluster of 8 sectors or 400h (1024) bytes. When working with hexadecimal addresses, conversion from memory locations to blocks of memory in decimal can be accomplished using the following steps:

1: Round the final address in the memory to the next highest page value. (xx00h)

2: Subtract 100h. Page 0, 00-FFh, is not saved.

3. Convert the most signficant nibble to decimal and then multiply by 16 (16 pages in 1000h).

4. Convert the second most significant nibble to decimal and then add to value computed in 3.

5. The result is number of pages needed to save memory image.

Here is example of a memory image from 100h to 2E6Ah:

```
1: 2E6Ah := 2F00h
2: 2F00h – 100h := 2E00h
3: 2h := 2 dec, 2 * 16 = 32
4: Eh := 14 dec, 14 + 32 = 46
5: 46 pages is the size of memory image
```

When using the SAVE function for files longer than 16k bytes, areas of the TPA will be destroyed when using CP/M ver. 1.4 because the CCP uses this area when building extension file control blocks (See Article II). Thus only one SAVE can safely be done. CP/M 2.0 uses areas outside the TPA for this function allowing multiple saves of the same memory image.

The final built-in command of the CCP is the LOAD file and execute function. This function is implemented by simply typing in the primary name of the file to be loaded and then a carriage return. Only COM files will work and any other file type will generate an error prompt and the system will return to the CCP. The file is loaded at 100h and then the computer jumps to this location. Programs that are run can have differing interactions with CP/M depending on their coding. Programs can be totally independent or they can use functions and subroutines available in BDOS and BIOS via a group of SYSTEM calls. These functions will be the topics of susequent articles on CP/M. Also, the term transient program is often used for files as loaded and executed in the TPA.

A function found only in CP/M 2.0 is USER. With this command, the operator can specify a user number of 0 to 15. The result of this is that only files as previously stored under that number can be accessed by the operator. Thus all the CCP commands are effected. When the system is initially booted up, the user number is 0 which is where files stored under ver 1.4 are found. To change the user number the following is typed: USER <0-15>. To copy files from one area to another, the PIP 2.0 utility is needed although the SAV and USER functions can be used with memory images. Last of all, the function ERA *.* will not erase the entire directory in ver 2.0; the quickest way to erase the disk is to use a utility such as a disk format program that clears all sectors.

All input of the console is buffered in the 128 bytes of memory from 80h to FFh as is disk I/O when the system is at the CCP level. After a program is loaded, the CCP will save all the information in the command line excluding the original entry.

```
RUN TEST EMPTY.BAS $L HEX
```

would be stored as:

```
TEST EMPTY.BAS $L HEX
```

beginning at 81h with the number of characters (21) being stored at 80h.

The transient program can read up to 128 characters of information from this area using string handling routines. Also, the second entry (TEST in the example) is place at the default fcb location tfcb (5Ch) while the third entry (EMPTY.BAS) is placed at tfcb + 16 (6Ch). Since the full fcb is 33 bytes long, the user program must move the second file name. The use of these functions will also be discussed with system calls in a future article.

CCP CONTROL CODE OPERATION

Since console I/O is buffered, the user can edit text strings by typing control characters. The carriage return code instructs the CCP to execute the command string typed in just previous to it. If a (cr) is typed when no other information has been input, then the disk prompt is printed. Control codes are selected on the keyboard of the console by first depressing the control key and then the desired character. Certain keyboards have function keys that are substitutes for control codes. The control key functions by forcing bit 6 (40h) of the alphanumeric key depressed to zero, thus only those codes that have bit 6 set (1) will be effected:

CHAR-ACTER	ASCII CODE	CONTROL CODE	FUNCTION
M	100 1101	000 1101	CARRIAGE RET
J	100 1010	000 1010	LINE FEED
H	100 1000	000 1000	BACK-SPACE
I	100 1001	000 1001	TAB

The codes used by the CCP are shown in Table 2:

TABLE 2 - CCP CODES

CHARACTER	FUNCTION KEY	ASCII CODE	FUNCTION
ctl-U		15h	delete line from buffer but do not erase from console screen; # is printed at end old line to indicated deleted line
ctl-X		18h	same as ctl-U but erases line from screen
	RUBOUT (RUB) DELETE (DEL)	7Fh	delete last character in the console buffer but echo it on screen (command string is typed backwords as DEL is depressed
ctl-H	BACK-SPACE	08h	same as rubout but last character is deleted from screen implemented as CCP function in ver 2.0; user option installed in BIOS in ver 1.4
ctl-R		12h	retype console buffer; used with DEL to give clear display of string; # is printed at console at end of old line before printing to indicate deleted text
ctl-E		05h	breaks line at console by sending (cr)(lf) to console without entering (cr) in console buffer; allows line of up to 128 characters to entered on console that allows lines of shorter length
ctl-M	CR, RET RETURN	0Dh	(cr)(lf) sent to console then command string is interpreted and executed by CCP
ctl-J	LINE FEED LF	0Ah	same as ctl-M
ctl-C		03h	CP/M system reboot (see discusion below)
ctl-Z		1Bh	not a CCP function; used to indicate end of console input in utility programs
ctl-S		13h	used to stop printout to console during DIR, TYPE, or similar functions in transient programs; typing any key will cancel ctl-S
ctl-P		10h	text printed on console device will also be printed on list device; if function is active then ctl-P cancels effect

While in CCP mode, inputing ctl-C causes a 'warm-boot'. When this occurs, CP/M executes a routine in BIOS that brings in the CCP and BDOS. If implemented while in CCP mode, the net effect is that the system logs in device A: and is ready to begin operation as if the system was initially booted on power up. Many transient programs implement a ctl-C option to return to CCP mode so care must be used not to execute this function accidently causing a loss of work and/or data. Also, when programs return control to CP/M, they usually do so by jumping to location 0 or by using the reset system call of BDOS which directs the computer via jumps to the warm boot routine in BIOS. When the warm boot function occurs or when a new device is logged-in for the first time after a warm boot, the disk is checked for read/write status. Using the STAT utility, disks can be software protected, and the CCP can also tell when a disk has been placed in a drive that has been initialized with another disk. As a result of both software write protection or swapping of disks, an error code will be generated when data is written to the disk. Thus whenever changing disks a ctl-C must be typed. Also, a warm boot will not change the contents of the TPA so that programs that have been developed using one disk can be saved after swapping disks in the same drive. When the CCP

cannot alter disk contents because of write protection then the following statement is printed on the console:

BDOS ERROR ON A: R/O

A can be any logical device and R/O means Read Only.

ONE, TWO or THREE DRIVES?

Many computer users when first researching mass storage alternatives ask the quetion: 'How many drives are needed for my application?' Although alternatives can vary depending on application, my experiences have given the following conclusions. First of all, the two drive system is the minimal configuration for intensive work. As mentioned above, file duplication on different disks is a necessity for protection against loss of data, but even though this can be done with one drive, it can be quite time consuming. The PIP (Peripheral Interchange Utility) is used to copy files from one disk to another. In one drive systems, two different floppy diskettes can be used by swapping disks when required by the system. When the system requires a change of disk, it will print the command 'MOUNT B:' or 'MOUNT A:' depending on whether information is to be read from A: or written to B:. This procedure can be very confusing, and can be costly when copying original files and errors occur. It should be noted that this facility is implemented in BIOS, and it may or may not be present depending on the BIOS in the system. Also, some BIOS' have this function as an option during assembly of the BIOS source code while other systems use the prompt during system boot up of: 'HOW MANY DISK DRIVES?'. With two or more storage devices, however, file duplication using PIP is a simple chore.

Probably the best configuration in terms of number of units is three. One of the areas needing more development is multi-tasking software. Multi-tasking hardware/software systems have the abiltity to perform two or more functions at same time. This is accomplished through procedures that allow routines to share computer time. Several programs have been developed that use multi-tasking, and for the most part, these have been based on SPOOL or DESPOOL functions. In the early days of computing, when computers could only accomplish one task at a time, having the computer spend time printing information on list device or entering data from card readers could be both expensive and/or problematical due to scheduling considerations. A simple solution was to write (SPOOL) the information to be printed on a mass storage device which usually was magnetic tape; hence the term SPOOL. At a later time, the information could be printed (DESPOOLED) onto a printer which was either on-line (connected to and controlled by the original computer) or off-line (not connected to the original computer).

In CP/M programs, time that is spent while the computer waits for input from the console is used to output information on a disk file to the list device. This can prove to be a great time saver in installations that require a lot of printing. One problem, however, is that the disk containing the file that is being printed cannot be removed from its drive until completion of despooling. With a two drive system, this causes problems if two disks are required for an operation, for even though space on the despooling disk can be used, the non-despooling disk is the only free disk. With the three drive system, one drive can be dedicated as in the above example while two drives are left free.

A second advantage of having three drives is that one of the drives can be write-protected while the other two are free for both reading and/or writing. This allows the user to protect important files from possible loss due to mistake or accident. Another point is that one drive can be dedicated to holding the system diskette and various utilities while the other two are free for disk swapping.

A final advantage, and in my mind the most important, is hardware backup. In situations where the computer is a necessity for operation, failure of hardware can prove disasterous, and due to this, entire computer manufactering firms have been built or broken by the ability of users to get quick and effective maintenance. At the present time, this is by far the biggest problem in the microcomputer industry. Although microcomputers have proven to be very reliable, many tales have been circulating about failures of equipment and days, weeks, and even months of computer 'down' time. Since the disk unit is a device with moving parts that can wear out or lose adjustment, it is one of the first devices to fail and due to its nature one of the most difficult to repair. With the three drive system, if one drive malfunctions, then the other two are still available while the third is off-line. In most cases, the user will not need to alter hardware except in that case where drive-0 (the SYSTEM drive) is effected.

WHICH DISK SIZE, TYPE & DENSITY?

Another question commonly asked is: 'What size, type, and/or density format do I need?' My opinion on type of drive for most micro-computer installations, at the start, is 8 inch single density format. The reason is that this is the most time proven and standard media for microcomputing. Other systems such as tape, hard disk, and even 5.25 inch floppy disk although viable have problems due to price, avaiability, capacity, and most importantly, dependability. The reason I maintain single density is that the standard in the industry for the transferring of data is still single density. Although the bugs seem to have been worked out of double density hardware/software in the 8 inch drive, I suggest than when purchasing or updating to this type system, that it be thorougly tested before purchase and use. Users should also beware that many disk drives are rated for both single and double density use, so when purchasing a single density system, check the drives so that update to dual density at a later time can be done without change of drives, the most expensive component. Another consideration is that when purchasing dual density systems, (can perform single and double density operations), check the software and documentaion for clearness and ease of single vs. double density operation. Although 5.25 inch disks have proven dependable, cost effective, and advantageous over larger devices in physical size and weight, they have been used mostly in microcomputers or stand-alone devices such as smart terminals or word processors. The 8 inch variety has been used widely in the entire computer industry, and when disk formats are standardized for the interchange of data between different systems, the 8 inch disk will probably be used.

HARD DISK SYSTEMS

Small, high capacity, cost effective hard disk alternatives have developed quickly over the last year. Also, S-100 controllers have appeared for older hard disk designs. Capacities range from 5 megabyte on up for single units with multi unit sytems controlled by CP/M 2.0 getting into the 100 megabyte range. Of importance to the average CP/M user is the fixed disk alternatives that are becoming competitive with floppy disks. Some floppy disk manufactures are building units that are hard disks within 8 inch floppy disk housings, have similar if not identical signal connections, and have the same power requirements as their flexible counterpart. As a result of this, the new idea is to mix hard disks with floppies using one controller and CP/M 2.0 software.

There are two reasons why these disks are cost effective, smaller, and more energy efficient. One, Winchester Technology, allows very high densities of data per track and tracks per disk. Secondly and most important to CP/M users is that the storage medium is non-removable. This allows the manufacturer a lot more mechanical freedom than in systems where movement of the disk due to physical support becomes a problem. As a result, these new 8 inch hard disks although offering large capacity do not offer disk backup. As long as the user does not use up his/her disk space, need to transfer data on mass storage media, need to get new data onto his/her disk systems, or have an accident, hard disks are fine.

In other words, unless the media is removable, having a second floppy is a necessity. Even if all or part of the media is removable, CP/M software will still be distributed on 8 or 5.25 inch floppy unless the software distributor has hardware that is identical to the user's. The real value of the hard disk is in using its storage capacity to greatly expand computer memory. Since data transfer on hard disks is much faster than floppies and much larger files can be maintained, operations such as searching and sorting or storage and retrieval of system memory images become quite feasible on 8 bit and 16 bit (8086 or Z8000) CP/M systems. When backup storage on floppy disk becomes a problem due file length, then magtape units based on digital cartridges become a feasible alternative and as disk technology develops, this area will also expand.

IN CONCLUSION

A few final remarks. If you are new to the mass storage market, do not be afraid to buy now for fear that your purchase will quickly become obsolete. Try to buy equipment with the philosophy that if expansion is needed at a later date, then hardware should be supplemented rather than replaced. Microcomputer equipment is like stereo equipment: once purchased its resale value drops quickly, thus replacement can prove quite costly. As far as obsolesence is concerned, as long S-100 bus systems are used, the user has a world of manufacturers and products to draw from. If one device needs to be replaced, the entire system need not be replaced. This philosophy is quite unique to the S-100 industry for a great majority of manufactures still viable today have survived because they have used industry compatibility as a major marketing point. The same can be said of CP/M and CP/M compatible operating systems.

In the next article in this series, I will list the various utility programs that are included by Digital Research with CP/M and give a brief overview of the functions they provide. I will also begin to describe the BIOS giving its structure and possible modifications that the user can implement.

APL
APL
APL
APL
APL

©1979 **Erik T. Mueller**

SOFTRONICS μAPL for the 8080/8085/Z-80

APL is an interactive general-purpose programming language with powerful primitive functions. **SOFTRONICS** APL runs under the CP/M* operating system. It is 'ready-to-go' in ASCII, using CP/M standard I/O. The interpreter runs in a variety of character set configurations. In addition to the standard ASCII mnemonic representations, it supports typewriter and bit-pairing ASCII-APL character sets. It can run with user-supplied I/O drivers.

FEATURES:

- Most of the functions and operators of full APL, including n-dimensional inner and outer product, reduction, compression, general transpose, reversal, take, drop. Execute and format.
- The interpreter resides in 30K bytes of memory, leaving remaining memory for the workspace and disk operating system.
- Shared-variable mechanism for CP/M disk input and output, system functions and variables, system commands.
- Abrams' descriptor calculus and shared data storage are the advanced optimization techniques employed by the interpreter. This saves memory space and execution time. Values are stored internally in a variety of formats for efficient memory utilization.
- Optional driver program for video display with programmable character generator.

$350 on CP/M* disk COMPLETE WITH USER'S MANUAL

NJ residents please add 5% sales tax.

$30 FOR USER'S MANUAL ALONE
(Refundable With Order)

SOFTRONICS

36 Homestead Lane
Roosevelt, N.J. 08555

* CP/M is a registered trademark of Digital Research

NORTH ★ STAR TOPICS

by
Randy Reitz
26 Maple St.
Chatham Township, N.J. 07928

A General Purpose Permuted Keyword Index Program

I have been interested in PASCAL ever since the August 78' BYTE magazine feature. I purchased Kenneth L. Bowles's book Microcomputer Problem Solving Using PASCAL some time later and quickly became sold on the ease of expressing algorithms in this language. By this time I had already been experimenting with a "structured" language using Tom Gibson's Tiny-c, so I knew that BASIC was a thing of the past for me. When North Star announced the availability of the UCSD PASCAL development system for only $50 on their disk system, I couldn't resist any longer. For $50, UCSD PASCAL on North Star has to be one of the best software bargains ever offered. I'm surprised that North Star wasn't swamped with orders. This is one piece of good news that seems to travel very slowly.

I was anxious to try out my "new" software toy; and by this time I was all the way up to chapter 7 in Bowles's book. There was a problem that caught my eye. The problem had to do with removing "noise" words from a character string in preparation for using the string in a keyword in context -KWIC -program. I had seen this type of index also called a permuted keyword index. Since a title will be entered into the index once for each keyword it contains, the title is permuted so the keyword always starts in the same column. I always wished I had such a program to keep track of all the articles contained in the 5 monthly computer publications I receive. It is very frustrating when I can remember reading an article but have great difficulty finding the publication. I decided it was time to apply PASCAL power to build a permuted keyword index program that I could use to easily search for articles in my rapidly growing volume of computer publications.

A fully capable permuted keyword index system can get quite complicated, so I wanted to decide on some limited goals before I got carried away. Remember, at this time I believed that my new PASCAL system could express any algorithm with the greatest of ease. Indeed it can, but no language can handle foggy thinking by its programmer. I found just the simple algorithm I was looking for in a personal filing system a friend of mine was using at Bell Labs. Consider the following data taken from the index of several BYTE publications:

```
.Distributed.networks/Horton/78 11
.Graphic input of wheather.data/Smith/79 7
Quest;.games/Chaffee/79 7
.Subroutine.parameters;.data/Maurer/79 7
A spacecraft.simulator/Sirvak/79 11
The Intel.8086;system design.kit/Ciarcia/79 11
The Cherry pro.keyboard/Parker/79 11
```

The title of the article, author and date of publication are listed along with some unusual punctuation. The punctuation is used to indicate the following:

1) A period is placed in front of a keyword
2) The author is enclosed in "slashes" (/).
3) The date of publication is always after the last / and is in year month format.

All other punctuation is superfluous to the algorithm. Since the "filing system" for magazine publications is constrained to be ordered by date, the permuted keyword index program should produce an alphabetically sorted listing of each keyword found in a title (identified by a period) along with the remaining title, author and date. If all of the keywords found are listed left justified, you can simply scan down the list for the keyword of interest and presto find all the articles which contain this keyword in the title. This simple idea can be extended to sort by author or date as well. Also, since I was using a video terminal, I wanted to add the capability to specify the range of keywords, authors or dates that were displayed so I could leisurely read the results before they disappeared from the screen. The UCSD PASCAL program that follows implements this simple idea using the North Star disk system that I am running on my "antique" IMSAI 8080. I call it a general purpose permuted keyword index program because I can easily think of many more applications other than a magazine publication index.

I must warn you that the program I am about to describe must be considered unfinished. Also, since this was my first PASCAL experience, I used as many of the language features I could. You will find string manipulation using the UCSD string intrinsics, record data structures and pointers, sorting with binary trees, variable arguments and more. All of the "modern" stuff that makes PASCAL so much more exciting than BASIC. Unfortunately the result isn't as "clean" as it could be.

All PASCAL programs begin with a "program" statement and a declaration of global variables:

```
PROGRAM KWIC;
CONST
 N=10;
 BLANKS=' (72 blanks here) ';
TYPE
 INDEXES=ARRAY[1..N] OF INTEGER;
 STRING1=STRING[1];
 LINKS  =^ENTRY;
 ENTRY  =RECORD
            STUFF      :STRING;
            RLINK,LLINK:LINKS
        END;
VAR
 LINE,LOW,HIGH                 :STRING;
 TITLE,AUTHOR,DATE,ABL,DBL     :STRING[72];
 ERROR                         :BOOLEAN;
 F                             :TEXT;
 PLOC,SLOC                     :INDEXES;
 I,J,NUM,SORT,MAX              :INTEGER;
 ROOT                          :LINKS;
```

I find this part of structured programming the most difficult to get used to. You have to have well laid out plans to begin a program by defining all of the variables and types. First, two constants are defined. I cheated in the definition of the constant BLANKS since I can't type 72 blanks in one of these columns. The next section defines variable types. These items are called type identifiers. They are not variables but are used to define variables in the next section. The capability that PASCAL offers to define variable types to suit the needs of the algorithm is an extremely valuable feature which I think sets PASCAL apart from the other "modern" languages. The type identifier "INDEXES" will be used to define variables that are arrays of 10 integers. "STRING1" will define variables that are strings of only one character. In a strongly typed language like PASCAL, a string of one character is quite different than a variable of type character. Finally, "LINKS" will define a pointer type variable that points to a data structure of type record defined by "ENTRY". Each variable of type "ENTRY" will contain "STUFF" and two pointers to variables of the same type as "ENTRY". This data structure elegantly implements a linked list that will be used in a binary tree sort algorithm.

The variables are defined next. The type STRING is pecular to UCSD PASCAL. The default string length is 80 characters but can be specified to any value less than 256 using a number in brackets. The variable F is of type "TEXT" which is a synonym for "FILE OF CHARACTERS". The input data will be read from this file. Finally, the variable ROOT will serve as the root of the binary tree so it is of type "LINKS". All of these variables are global and can be used by the main program as well as all functions and procedures defined below.

The next feature in a PASCAL program is the definition of the functions and procedures used in the program.

```
FUNCTION UPPERCASE(CH:CHAR):CHAR;
 BEGIN
 IF CH IN ['a'..'z'] THEN
   UPPERCASE:=CHR(ORD(CH)-32)
   ELSE
   UPPERCASE:=CH
 END;
```

This function is used to be sure a character is upper case ASCII only. Notice that functions which return values must be given types just like variables. Also notice the use of the set constant ('a'..'z'). The meaning is self explanatory and is certainly preferable to arithmetic comparisons. The ORD function is built in and is similar to the BASIC ASC function. The CHR function is similar to the BASIC CHR$ funciton.

```
PROCEDURE FINDR(PAT:STRING1; VAR S:STRING;
          VAR WHERE:INDEXES; VAR CNT:INTEGER);
 VAR J,CUM:INTEGER;
 BEGIN
   CUM:=0; CNT:=0; WHERE[1]:=0;
   REPEAT
     J:=POS(PAT,COPY(S,CUM+1,LENGTH(S)-CUM));
     CUM:=CUM+J;
     IF J>0 THEN
       BEGIN
       S[CUM]:=' ';
       CNT:=CNT+1;
       WHERE[CNT]:=CUM;
       WHERE[CNT+1]:=0
       END
   UNTIL (J=0) OR (CUM=LENGTH(S))
 END;
```

A subroutine which doesn't return any explicit value is called a procedure. This procedure finds the punctuation used to define keywords and the author. When the punctuation defined in argument "PAT" is found in argument "S", the punctuation is replaced by a blank and the location is noted in the next argument "WHERE". The final argument "CNT" returns the number of punctuations found. Notice that this procedure really returns values in three of it's four arguments. That's why these arguments are prefixed with "VAR" to identify that they are to be passed to the procedure by address rather than value. This may seem overly tedious but PASCAL keeps you aware of what variables a procedure is free to change and what variables it can't change. In a long program, this feature can help you to avoid those really hard to find bugs. The procedure uses two local variables, "J" and "CUM". Even though "J" is also a global variable since it can only access the local variable. "POS" and "COPY" are two built in UCSD string intrinsics. "POS" returns the position of the first occurrence of the pattern (first argument) in the second argument. "COPY" returns a string which is a copy of the first argument starting with the character position defined by the second argument for the number of character defined by the third argument. For example,

```
 STUFF:='TAKE THE BOTTLE WITH A METAL CAP';
 PATTERN:='TAL'
 WRITELN(POS(PATTERN,STUFF));
```

will print 26. Also,

```
 WRITELN(COPY(STUFF,POS('B',STUFF),6));
```

will print "BOTTLE". The next two procedures implement the binary tree:

```
 PROCEDURE ENTER(NEW:LINKS);
  VAR THIS,NEXT:LINKS;
  BEGIN
    NEW^.STUFF[1]:=UPPERCASE(NEW^.STUFF[1]);
    IF ROOT=NIL THEN ROOT:=NEW
    ELSE
    BEGIN
      NEXT:=ROOT;
      REPEAT
        THIS:=NEXT;
        IF NEW^.STUFF<=THIS^.STUFF THEN
          NEXT:=THIS^.LLINK
        ELSE
          NEXT:=THIS^.RLINK
      UNTIL NEXT=NIL;
      IF NEW^.STUFF<=THIS^.STUFF THEN
        THIS^.LLINK:=NEW
      ELSE
        THIS^.RLINK:=NEW
    END
  END;
```

```
PROCEDURE TRAVERSE(PTR:LINKS);
 BEGIN
   IF (PTR^.LLINK<>NIL) AND (PTR^.STUFF>=LOW)
     THEN TRAVERSE(PTR^.LLINK);
   IF (PTR^.STUFF>=LOW) AND (PTR^.STUFF<HIGH)
     THEN BEGIN
     WRITELN(PTR^.STUFF);
     J:=J+1;
     IF J>20 THEN
       BEGIN
       J:=0;
       WRITE('Type <ret> to continue');
       READLN
       END
     END;
   IF (PTR^.RLINK<>NIL) AND (PTR^.STUFF<HIGH)
     THEN TRAVERSE(PTR^.RLINK)
 END;
```

The ENTER procedure will take a data structure of type "ENTRY" and link it into the appropriate node in the binary tree. The binary tree is implemented using a linked list data structure defined as type "ENTRY" above. Each entry is a record which contains 3 items: 1) STUFF which is a string, 2) RLINK which is a pointer to the next "ENTRY" record with STUFF greater than this record's STUFF and 3) LLINK which is a pointer to the next "ENTRY" record with STUFF less than or equal to this record's STUFF. The procedure works with these pointers which are of type "LINKS". PASCAL allows the items of a record to be accessed using the construction "record variable.item variable". I do not have any variables of type "ENTRY", which is the record variable type. I only use pointers to these record variables so I access the variables contained in a record using the construction "pointer variable.item variable". The ENTER procedure first makes sure the first character of STUFF is upper case. Next, if ROOT is empty, it will contain the special value NIL and will be initialized to point to the NEW record. If ROOT contains a valid pointer, the search of the tree is begun to find the proper node for the NEW record. The search will follow either the left link (LLINK) or right link (RLINK) depending on the relationship between STUFF in the NEW record and STUFF in the current (THIS) record. UCSD PASCAL allows strings of different lengths to be compared. The search continues until the end of the tree is found (a pointer value of NIL). The NEW record is entered by making the current (THIS) record point to the NEW record.

The TRAVERSE procedure is used to retrieve in a sorted fashion STUFF from the tree. This procedure is really simple; but is difficult to understand if you are not familiar with recursion. The main program below will define the LOW and HIGH search strings and start TRAVERSE at the ROOT of the tree. TRAVERSE procedes down the left link (LLINK) until it finds either the end of the tree or a record with STUFF less than LOW. Remember that STUFF was entered with lesser

North Star Horizon– COMPUTER WITH CLASS

The North Star Horizon computer can be found everywhere computers are used: business, engineering, home — even the classroom. Low cost, performance, reliability and software availability are the obvious reasons for Horizon's popularity. But, when a college bookstore orders our BASIC manuals, we know we have done the job from A to Z.

Don't take our word for it. Read what these instructors have to say about the North Star Horizon:

"We bought a Horizon not only for its reliability record, but also because the North Star diskette format is the industry standard for software exchange. The Horizon is the first computer we have bought that came on-line as soon as we plugged it in, and it has been running ever since!"
— Melvin Davidson, Western Washington University, Bellingham, Washington

"After I gave a ½ hour demonstration of the Horizon to our students, the sign-ups for next term's class in BASIC jumped from 18 to 72."
— Harold Nay, Pleasant Hill HS, Pleasant Hill, California

"With our Horizon we brought 130 kids from knowing nothing about computers to the point of writing their own Pascal programs. I also use it to keep track of over 900 student files, including a weekly updated report card and attendance figures."
— Armando Picciotto, Kennedy HS, Richmond, California

"The Horizon is the best computer I could find for my class. It has an almost unlimited amount of software to choose from. And the dual diskette drives mean that we don't have to waste valuable classroom time loading programs, as with computers using cassette drives."
— Gary Montante, Ygnacio Valley HS, Walnut Creek, Calif.

See the Horizon at your local North Star dealer.

NorthStar

North Star Computers, Inc.
1440 Fourth Street
Berkeley, CA 94710
(415) 527-6950
TWX/Telex 910-366-7001

STUFF on the left link. When the trip down the left link stops with a record with STUFF between LOW and HIGH, the record is printed on the terminal. The global variable J keeps track of the number of records printed and stops at 20 so the CRT screen can be leisurely read. Now TRAVERSE starts down the right leg if it exists and if the STUFF down there is less than HIGH. This defines a new "subtree" which is searched in similar fashion. The resulting listing will have STUFF sorted from low to high.

The final procedure creates a record and the variable STUFF:

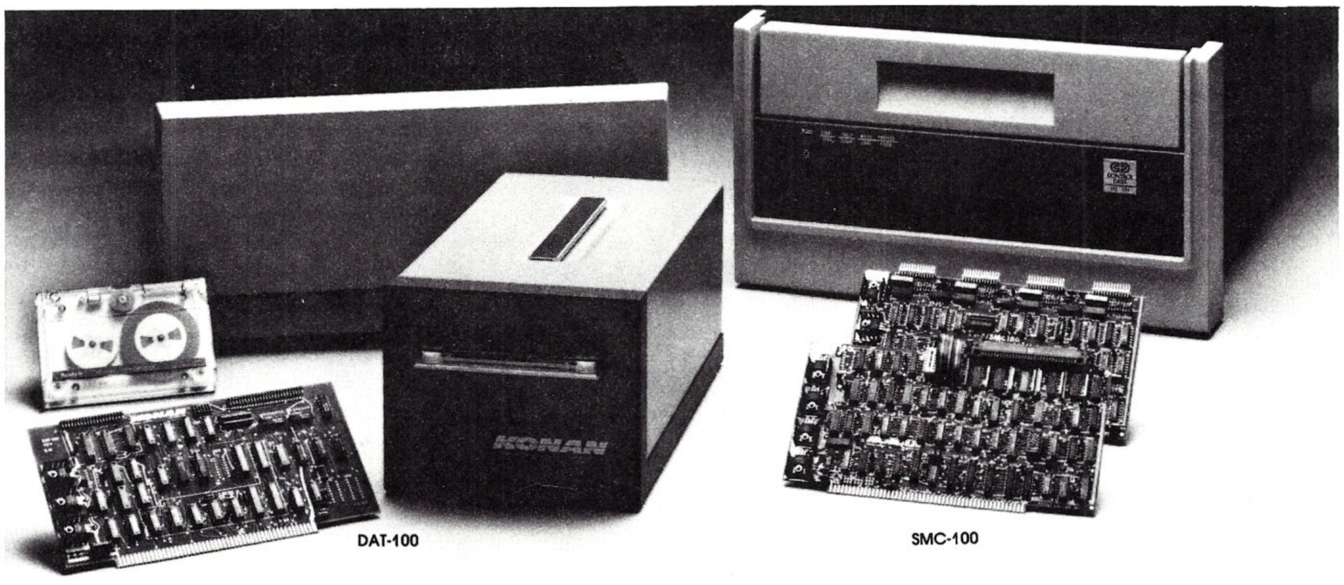
```
PROCEDURE CREATIT;
 VAR P:LINKS;
 BEGIN
   NEW(P);
   CASE SORT OF
     1: TITLE:=CONCAT(COPY(LINE,PLOC[I]+1,
              SLOC[I]-PLOC[I]),COPY(LINE,
              1,PLOC[I]));
     2,3: IF LINE[1]=' ' THEN
              TITLE:=COPY(LINE,2,SLOC[I])
              ELSE
              TITLE:=COPY(LINE,1,SLOC[1]);
   END;
   TITLE:=COPY(CONCAT(TITLE,BLANKS),1,56);
   AUTHOR:=COPY(LINE,SLOC[I]+1,
              (SLOC[2]-SLOC[1])-1);
   IF LENGTH(AUTHOR)>14 THEN
     BEGIN
     AUTHOR:=COPY(AUTHOR,1,14);
     ABL:=' '
     END
     ELSE
     ABL:=COPY(BLANKS,1,15-LENGTH(AUTHOR));
   DATE:=CONCAT('19',COPY(LINE,SLOC[2]+1,
              LENGTH(LINE)-SLOC[2]));
   DBL:=COPY(BLANKS,1,8-LENGTH(DATE));
   CASE SORT OF
     1: P^.STUFF:=CONCAT(TITLE,ABL,AUTHOR,
              ' ',DATE);
     2: P^.STUFF:=CONCAT(AUTHOR,ABL,TITLE,
              ' ',DATE);
     3: P^.STUFF:=CONCAT(DATE,DBL,TITLE,
              ABL,AUTHOR)
   END;
   P^.LLINK:=NIL;
   P^.RLINK:=NIL;
   ENTER(P)
 END;
(* Begin Main program *)
BEGIN
  ROOT:=NIL; J:=0;
  WRITE('Enter data file name ->');
  READLN(LINE);
  RESET(F,LINE);
  REPEAT
  WRITE('Sort by 1) TITLE, 2) AUTHOR ',
        'or 3) DATE? Enter 1,2 or 3 ->');
  READLN(SORT)
UNTIL SORT IN [1,2,3];
READLN(F,LINE);
  WHILE NOT EOF(F) DO
    BEGIN
    FINDR('/',LINE,SLOC,NUM);
    ERROR:=NUM<>2;
    FINDR('.',LINE,PLOC,NUM);
    ERROR:=ERROR OR (NUM=0);
    IF SORT IN [2,3] THEN NUM:=1;
    IF NOT ERROR THEN
      FOR I:=1 TO NUM DO
        BEGIN
        CREATIT;
        J:=J+1
        END
    ELSE
      BEGIN
      WRITELN('**BAD LINE**',CHR(7));
      WRITELN(LINE)
      END;
    READLN(F,LINE)
  END;
  WRITELN('Sort complete with ',J,
          ' records entered.  Enter range',
          ' for output.');
  REPEAT
    WRITE('Low string (<etx> to quit)->');
    READLN(LOW);
    IF NOT EOF THEN
      BEGIN
      LOW[1]:=UPPERCASE(LOW[1]);
      WRITE('High string->');
      READLN(HIGH);
      IF NOT EOF THEN
        BEGIN
        HIGH[1]:=UPPERCASE(HIGH[1]);
        J:=0;
        TRAVERSE(ROOT)
        END
      END
  UNTIL EOF
END.
```

The main program asks for the name of the data file and the type of sorting to do. Records are read from the data file and the position of the two slashes are saved in SLOC. The position of the periods are saved in PLOC. The CREATIT procedure is called once if the sort is by author or date. CREATIT is called for each keyword if the sort is by title.

The CREATIT procedure creates a new record with pointer in "P". If the sort is by title, the title is permuted using the value of the index "I". Strings for title, author and date are created with the proper lengths. Then STUFF is put together depending on the type of sort requested. Finally, ENTER is used to link the new record into the tree.

The main program finishes by requesting the values for the low and high strings. If control-C is not entered, the first character of each string is converted to uppercase and TRAVERSE is started at the ROOT. You can repeatedly query the data by entering new low and high strings. I have 56K of memory which will hold one year's worth of a publication's titles.

If you try this program, I hope you will find it as interesting and useful as I have.



DAT-100 SMC-100

Hard disk and hardtape™ control

Up to 2400 Megabytes of hard disk control for the S-100 bus.

Konan's SMC-100 interfaces S-100 bus micro computers with all hard disk drives having the Industry Standard SMD Interface. It is available with software drivers for most popular operating systems. Each SMC-100 controls up to 4 drives ranging from 8 to 600 megabytes per drive, including most "Winchester" drives -- such as Kennedy, Control Data, Fujitsu, Calcomp, Microdata, Memorex, Ampex, and others.

SMC-100 is a sophisticated, reliable system for transferring data at fast 6 to 10 megahertz rates with onboard sector buffering, sector interleaving, and DMA.

SMC-100's low cost-per-megabyte advanced technology keeps your micro computer system micro-priced. Excellent quantity discounts are available.

Konan's HARDTAPE™ subsystem . . . very low cost tape and/or hard disk Winchester backup and more.

Konan's new DAT-100 Single Board Controller interfaces with a 17½ megabyte (unformatted) cartridge tape drive as well as the Marksman Winchester disk drive by Century Data.

The DAT-100 "hardtape" system is the only logical way to provide backup for "Winchester" type hard disk systems. (Yields complete hard disk backup with data verification in 20-25 minutes.)

Konan's HARDTAPE™ subsystem is available off the shelf as a complete tape and disk mass storage system or an inexpensive tape and/or disk subsystem.

Konan controllers and subsystems support most popular software packages including FAMOS™, CP/M® version 2.X, and MP/M.

Konan, first (and still the leader) in high-reliability tape and disk mass storage devices, offers OEM's, dealers and other users continuing diagnostic support and strong warranties. Usual delivery is off the shelf to 30 days with complete subsystems on hand for immediate delivery.

Call Konan's TOLL FREE ORDER LINE today:

800-528-4563

Or write to Bob L. Gramley
Konan Corporation, 1448 N. 27th Avenue
Phoenix, AZ 85009. TWX/TELEX 9109511552

CP/M® is a registered trade name of Digital Research,
FAMOS™ is a trade name of MVT Micro Computer Systems.
HARDTAPE™ is a trade name of Konan Corporation.

KONAN

LINEAR PROGRAMMING PART 2

by
W.M. Yarnall
19 Angus Lane
Warren, N.J. 07060

Setting Up & Solving A Problem

INTRODUCTION

In Part 1, the UCSD PASCAL implementation of the Revised Simplex Algorithm was presented, together with the output from a sample problem. In this part, four example problems will be taken up, one in each of the four problem classes mentioned in Part 1:

*The PRODUCT MIX problem,

*The TRANSPORTATION problem,

*the DIET problem, and

*GAMING STRATEGY.

The program shown in Listing 1 of Part 1 (LINEARP) provides very voluminous output, including an echo of all input data, as well as a list of the status of the solution at each iteration.

For this part, since the problems are longer, we prefer to suppress some of the output, leaving only the data at the end of the problem. The program LINPROG we will use is derived from LINEARP by deleting the procedures PRINTC and PRINTD (lines 55 thru 90), their references in lines 161 and 186, and three calls on the procedure PRINTX in lines 189, 302, and 401. This will reduce the output to more manageable proportions for publication.

In the solution of any problem by Linear Programming techniques, there are several necessary steps (in common with any problem solution by any other technique):

*STATEMENT OF THE PROBLEM -- what problem do we wish to solve?

*GATHERING OF DATA -- what data are available for the solution, and what are their values?

*FORMULATION OF THE MODEL -- construct the equations describing the problem and its data.

*Enter the data into the data file, and run the program.

When we take up each of the example problems, we will discuss each of these four steps, include a listing of the data file, and output from the computer program.

GENERAL

The format of the data file, as can be seen by the declarations of the program listing (see Listing 1, part 1) in lines 13-21, is a collection of records of variant types. This file can be constructed using the EDITFILE program shown in Listing 1. This program provides the capability to build a new file, or to modify/list an existing file. Upon execution of the program, you are prompted by

```
EDIT: L(IST, B(UILD, M(ODIFY, Q(UIT (1.0)
```

and the program will wait for a command, followed by (CR). A response of one of the letters L, B, M or Q will proceed to execute the command; Q exits to the PASCAL system. If a new file has been created (via the B or M command), then the file is LOCKED onto the disk. Each of the other commands prompts for the data to be entered at each stage.

When the action is L(ist or M(odify, the program will ask for a file name, and the record number at which the requested action is to start. Record numbers begin with 0 for the first record. If you are M(odifying a file, you are also asked for the name of the new file -- which will contain the results of the edit. Each record, starting with 0, until the designated record, is copied from the old file to the new file. Then the designated record is printed out, and a prompt is given for the action to be taken on the record. The options are:

```
K(EEP, C(HANGE, I(NSERT, D(ELETE.
```

If K or D is selected, the next record is displayed. If C or I is selected, a new record is requested by prompting for each element of the record. The first item requested is the TAG. The valid values are:

0 - Must be the first record in the file. It identifies the size of the problem.
1 - Optional. It provides the text to name the problem
2 - Identifies a row name and index, and the RHS data.
4 - Identifies a column name, index and OBJ (objective) data.
6 - Identifies an element of the ABAR matrix.
99 - Identifies the logical EOF.

Except for the first (TAG 0) and the last (TAG 99), records may be in any order; it is recommended, however, that they be grouped to make it easier to proofread a listing to make sure your data are correct.

When B(uilding a file, the program continuously prompts for a new record. The action is continued until a TAG greater than 100 is entered (as an escape). I use 999. In the other modes, the editor returns to its command level when the end of the input file is seen.

EXAMPLE PROBLEMS

In each of the four areas, we will present the problem, and carry thru the formulation of the model, and provide listings of the data file and the program run output. Now, on to the problems--

PROBLEM 1 -- A PRODUCT MIX PROBLEM

This problem is also sometimes called a production balance problem.

Problem Statement

A Manufacturer of a product with a very seasonal demand decides to carry out an analysis of his production strategy to minimize production costs. You volunteer to do the job on your home micro --

It appears that there are two alternatives: extra help can be hired or overtime used (or both) to meet the needs of high peak demand, laying off the extra help when demand is slow, or an attempt may be made to level the work force, and to stock the excess produced during the slow demand periods.

Each of these alternatives has cost factors associated with them; it is desired to minimize the production cost. The Sales Department has analyzed the demand for the product for the next year, and feels that customer demand for each of six two-month periods will be

Period 1	100 units
2	250 units (spring sales)
3	100 units
4	200 units (early Christmas orders)
5	400 units (mail Christmas orders)
6	500 units (refills of stock at retail)

It has been determined that the cost of stocking a unit of prior production in 2.0. (Note - all costs are given in units of standard production unit costs). Workforce can be augmented by use of overtime, and by hiring of temporary help. Because of the cost of hiring and training, and the time-and-a-half overtime rule, each unit of augmented production costs 1.75; moreover, when personnel cutbacks are made, the cost of decreasing production capacity by one unit is 1.25 (partly due to unemployment compensation). It is estimated that the current work force can produce for unit (1.0) cost. Since this is a new model, there is no prior stock to start. (Note - we missed last year's holiday sales because the computer-aided design program didn't work too well.

Problem Formulation

The variables we will use are:

X_i - Quantity of standard production in period 1
Y_i - Quantity of productive capacity increase at the start of the i-th period
Z_i - Units of productive capacity to be dropped, either by layoff or discontinuation of overtime at the start of period i
S_i - Units produced for stock during the i-th period, to be used to fill orders during period (i+1).

The constraining equations are

$$\begin{aligned} X_1 - S_1 &= 100 \\ X_2 + S_1 - S_2 &= 250 \\ X_3 + S_2 - S_3 &= 100 \\ X_4 + S_3 - S_4 &= 200 \\ X_5 + S_4 - S_5 &= 400 \\ X_6 + S_5 &= 500 \end{aligned}$$

for the production balance equations, and

$$\begin{aligned} -X_1 + X_2 - Y_2 + Z_2 &= 0 \\ -X_2 + X_3 - Y_3 + Z_3 &= 0 \\ -X_3 + X_4 - Y_4 + Z_4 &= 0 \\ -X_4 + X_5 - Y_5 + Z_5 &= 0 \\ -X_5 + X_6 - Y_6 + Z_6 &= 0 \end{aligned}$$

for the manpower balance equations. These equations reflect the fact that you can't have both a net increase AND a net decrease in capacity for a production period.

The cost function to be minimized is:

COST = 1.0*(sum of X's, i=1 to 6)
+ 1.75*(sum of Y's, i=2 to 6)
+ 1.25*(sum of Z's, i=2 to 6)
+ 2.0*(sum of S's, i=1 to 5).

Listing 2 shows the data file listing for this problem. Rows 1 thru 6 are the sales constraints, and rows 7 thru 11 are the manpower balance constraints. These 11 values are the RHS of the equations. Column data (TAG = 4) are the objective (Cost) items: columns 1 - 6 are the X's, columns 7 - 11 are the Y's, columns 12 - 16 are the Z's, and columns 17 - 21 are the stocking quatities. Since the equations above have unity coefficients for the variables, only 1 or -1 will show up in the non-zero elements of ABAR (TAG 6 items).

```
PROGRAM EDITFILE;

TYPE
 FREC = RECORD
        CASE TAG:INTEGER OF
         0: (NAME:STRING[6]; N1,N2:INTEGER);
         1: (HEADER:STRING[64]);
         2: (RNAME:STRING[6]; RINDEX:INTEGER; RHS:REAL);
         4: (CNAME:STRING[6]; CINDEX:INTEGER; OBJ:REAL);
         6: (R,S:INTEGER; T:REAL);
        99: ()
        END;

VAR
 OLDF,NEWF : FILE OF FREC;
 OBUF,NBUF : FREC;
 OFLAG,NFLAG : BOOLEAN;
 OFIL,NFIL : STRING;
 EDITING : BOOLEAN;
 COMMAND : CHAR;

FUNCTION INREC : INTEGER;
 VAR J:INTEGER;

 BEGIN
  WRITE(' ENTER TAG ');
  READ(J);
  READLN;
  NBUF.TAG:=J;
  WITH NBUF DO
   CASE TAG OF
    0: BEGIN
        WRITE(' PROGNAME ');
        READ(NAME);
        READLN;
        WRITE(' NO. ROWS ');
        READ(N1);
        READLN;
        WRITE(' NO. COLS ');
        READ(N2);
        READLN
       END;
    1: BEGIN
        WRITELN(' HEADER: ');
        READ(HEADER);
        READLN
       END;
    2: BEGIN
        WRITE(' ROW NAME ');
        READ(RNAME);
        READLN;
        WRITE(' ROW NO. ');
        READ(RINDEX);
        READLN;,
        WRITE(' RHS ');
        READ(RHS);
        READLN
       END;
    4: BEGIN
        WRITE(' COL NAME ');
        READ(CNAME);
        READLN;
        WRITE(' COL NO. ');
        READ(CINDEX);
        READLN;
        WRITE(' OBJ ');
        READ(OBJ);
        READLN
       END;
    6: BEGIN
        WRITE(' ROW NO. ');
        READ(R);
        READLN;
        WRITE(' COL NO. ');
        READ(S);
        READLN;
        WRITE(' ABAR[R,S] ');
        READ(T);
        READLN
       END;
   99: ;
   END;    (* CASE *)
  IF J > 100 THEN J:=200;
  INREC:=J
 END;    (* INREC *)

PROCEDURE PRINT(F:FREC; N:INTEGER);
 BEGIN
  WRITELN(' ');
  WRITELN(' REC',N:4,' TAG:',F.TAG:5);
  WITH F DO
   CASE TAG OF
    0: BEGIN
        WRITELN(' NAME: ',NAME);
        WRITELN(' NO ROWS: ',N1);
        WRITELN(' NO COLS: ',N2)
       END;
    1: BEGIN
        WRITELN(' HEADING:');
        WRITELN(HEADER)
       END;
    2: BEGIN
        WRITELN(' ROW: ',RNAME);
        WRITELN(' INDEX: ',RINDEX);
        WRITELN(' RHS: ',RHS)
       END;
    4: BEGIN
        WRITELN(' COL: ',CNAME);
        WRITELN(' INDEX: ',CINDEX);
        WRITELN(' OBJ: ',OBJ)
       END;
    6: WRITELN(' ABAR[',R,',',S,']: ',T);
   99: WRITELN(' --- LOGICAL EOF --- ');
   END;    (* CASE *)
  WRITELN(' ')
 END;    (* PRINT *)

PROCEDURE BUILD;
 VAR N:INTEGER;

 BEGIN
  WRITELN(' ');
  WRITE(' BUILD WHAT FILE? ');
  READ(NFIL);
  READLN;
  REWRITE(NEWF,NFIL);
  NFLAG:=TRUE;
  N:=0;
  WHILE N < 100 DO
  BEGIN
   N:=INREC;
   IF N < 100 THEN
   BEGIN
    NEWF^:=NBUF;
    PUT(NEWF)
   END
  END    (* WHILE *)
 END;    (* BUILD *)

PROCEDURE LIST;
 VAR REC:INTEGER;

 BEGIN
  IF NOT OFLAG THEN
  BEGIN
   WRITELN(' ');
   WRITE(' LIST WHAT FILE? ');
   READ(OFIL);
   WRITELN(' ');
   READLN;
   RESET(OLDF,OFIL);
   OFLAG:=TRUE
  END;
  WRITE(' STARTING AT WHAT RECORD? ');
  READ(REC);
  READLN;
  WRITELN(' ');
  SEEK(OLDF,REC);
  GET(OLDF);
  WHILE NOT EOF(OLDF) DO
```

Listing 1: Data File Editor Program

```
  BEGIN
   OBUF:=OLDF^;
   WRITE(REC:5,': ');
   WITH OBUF DO
    CASE TAG OF
     0: WRITELN(TAG:3,NAME:8,N1:7,N2:7);
     1: WRITELN(TAG:3,' ';HEADER);
     2: WRITELN(TAG:3,RNAME:8,RINDEX:7,RHS:14:8);
     4: WRITELN(TAG:3,CNAME:8,CINDEX:7,OBJ:14:8);
     6: WRITELN(TAG:3,' ROW',R:3,' COL',S:3,T:14:8);
    99: WRITELN(TAG:3,' LOGICAL EOF'); _
    END;    (* CASE *)
   REC:=REC+1;
   GET(OLDF)
  END   (* WHILE *)
 END;   (* LIST *)

PROCEDURE MODIFY;
 VAR REC,J:INTEGER; ANS:CHAR;

 BEGIN
  IF OFLAG
  THEN     (* OLD FILE IS OPEN *)
   SEEK(OLDF,0)
  ELSE
  BEGIN
   WRITE(' MODIFY WHAT FILE? ');
   READ(OFIL);
   RESET(OLDF,OFIL);
   OFLAG:=TRUE;
   READLN;
  END;
   WRITE(' NAME OF NEW FILE? ');
   READ(NFIL);
   IF NFLAG
   THEN
    CLOSE(NEWF,LOCK)
   ELSE
   BEGIN
    REWRITE(NEWF,NFIL);
    NFLAG:=TRUE
   END;
   READLN;
   WRITE(' STARTING AT WHICH RECORD? ');
   READ(J);
   READLN;
   IF J>0 THEN
    FOR REC:=0 TO (J-1) DO
    BEGIN
     SEEK(OLDF,REC);
     GET(OLDF);
     IF NOT EOF(OLDF) THEN
     BEGIN
      NEWF^:=OLDF^;
      PUT(NEWF)
     END
  END;
  REC:=J;
  WHILE NOT EOF(OLDF) DO
  BEGIN
   SEEK(OLDF,REC);
   GET(OLDF);
   IF NOT EOF(OLDF) THEN
   BEGIN
    PRINT(OLDF^,REC);
    WRITELN(' PROCESS THIS RECORD?');
    WRITELN(' K(EEP, C(HANGE, I(NSERT, D(ELETE');
    READ(ANS);
    READLN;
    CASE ANS OF
    'K': BEGIN
          NEWF^:=OLDF^;
          PUT(NEWF);
          REC:=REC+1
         END;
    'C': BEGIN
          J:=INREC;
          IF J<100 THEN
          BEGIN
           NEWF^:=NBUF;
           PUT(NEWF);
           REC:=REC+1
          END ELSE REC:=REC+1
         END;
```

HAYDEN HAS THE BOOKS FOR TOMORROW'S

PROSUMER*

New! S-100 BUS HANDBOOK
(Bursky). Explains all the details of commonly available S-100 systems and how they are organized. Covers computer fundamentals, basic electronics, and each section of the computer. Schematic drawings and illustrations are provided for reference. **#0897-X, $12.95**

New! DESIGNING MICROCOMPUTER SYSTEMS
(Pooch and Chattergy). Describes three of the most popular microcomputer families: the Intel 8080, Zilog Z80, and Motorola 6800 in terms of microprocessor architecture, timing, control and clock signals, interrupt handling, etc. Timing diagrams are included as well as information on building microcomputer systems from kits. **#5679-6, $8.95**

New! PASCAL WITH STYLE: Programming Proverbs
(Ledgard, Nagin, & Hueras). Introduces superior methods for program design and construction. Stresses overall program organization and "logical thinking." A special chapter shows you how to use the top down approach with PASCAL. Includes samples of PASCAL programs. **#5124-7, $6.95**

*Prosumers are those who measure their personal success by their ability to independently produce goods and services for and by themselves... and their computers.

Available at your local computer store.

Or write to:
Hayden Book Company, Inc.
50 Essex Street, Rochelle Park, N.J. 07662

Call (201) 843-0550, ext. 307 TO CHARGE YOUR ORDER TO: Master Charge or Visa!

Minimum order is $10.00; customer pays postage and handling.

```
      'D': REC:=REC+1;
      'I': BEGIN
             J:=INREC;
             IF J<100 THEN
             BEGIN
              NEWF^:=NBUF;
              PUT(NEWF)
             END
            END;
     END     (* CASE *)
    END
   END     (* WHILE *)
  END;     (* MODIFY *)

BEGIN    (* MAIN *)
  OFLAG:=FALSE;
  NFLAG:=FALSE;
  EDITING:=TRUE;
  WHILE EDITING DO
  BEGIN
   WRITELN(' ');
   WRITE(' EDIT: L(IST, B(UILD, M(ODIFY, Q(UIT [1.0] ');
   READ(COMMAND);
   READLN;
   CASE COMMAND OF
    'L': LIST;
    'B': BUILD;
    'M': MODIFY;
    'Q': EDITING:=FALSE;
   END     (* CASE *)
  END;     (* WHILE *)
  IF NFLAG THEN CLOSE(NEWF,LOCK)
END.
```

CATCH THE S-100 INC. BUS!

	LIST PRICE	OUR SPECIAL CASH PRICE
Godbout, Econoram XIV 16K Static Ram w/Extended Addressing 4 MHz Assembled & Tested	349.00	298.00
Godbout Econoram X 32K 4 MHz Static Memory Board — "Unkit"	599.00	512.00
S.D. Systems VDB 80x24 Video Board Kit	370.00	309.00
S.D. Systems Z-80 Starter Kit w/PIO	340.00	275.00
Sanyo Video Monitor 9"	240.00	160.00
Intertec Intertube Terminal U/L Case 80x25	995.00	779.00

Subject to Available Quantities • Prices Quoted Include Cash Discounts. Shipping & Insurance Extra.

We carry all major lines such as S.D. Systems, Cromemco, Ithaca Intersystems, North Star, Sanyo, ECT, TEI, Godbout, Thinker Toys, Hazeltine, IMC For a special cash price, telephone us.

Hours: Mon.-Fri. 10 A.M.-6 P.M.

Bus .... **S-100, inc.**
Address ..... **7 White Place Clark, N.J. 07066**
Interface **201-382-1318**

```
EDIT: L(IST, B(UILD, M(ODIFY, Q(UIT [1.0] L

LIST WHAT FILE? BALANCE.DATA

STARTING AT WHAT RECORD? 0

    0:   0  BALPRD       11      21
    1:   1 PRODUCTION BALANCE EXAMPLE
    2:   2  SALES1        1   100.000
    3:   2  SALES2        2   250.000
    4:   2  SALES3        3   100.000
    5:   2  SALES4        4   200.000
    6:   2  SALES5        5   400.000
    7:   2  SALES6        6   500.000
    8:   2  BAL2          7     0.00000
    9:   2  BAL3          8     0.00000
   10:   2  BAL4          9     0.00000
   11:   2  BAL5         10     0.00000
   12:   2  BAL6         11     0.00000
   13:   6 ROW  1 COL  1     1.00000
   14:   6 ROW  2 COL  2     1.00000
   15:   6 ROW  3 COL  3     1.00000
   16:   6 ROW  4 COL  4     1.00000
   17:   6 ROW  5 COL  5     1.00000
   18:   6 ROW  6 COL  6     1.00000
   19:   6 ROW  7 COL  1    -1.00000
   20:   6 ROW  7 COL  2     1.00000
   21:   6 ROW  8 COL  2    -1.00000
   22:   6 ROW  8 COL  3     1.00000
   23:   6 ROW  9 COL  3    -1.00000
   24:   6 ROW  9 COL  4     1.00000
   25:   6 ROW 10 COL  4    -1.00000
   26:   6 ROW 10 COL  5     1.00000
   27:   6 ROW 11 COL  5    -1.00000
   28:   6 ROW 11 COL  6     1.00000
   29:   6 ROW  7 COL  7    -1.00000
   30:   6 ROW  8 COL  8    -1.00000
   31:   6 ROW  9 COL  9    -1.00000
   32:   6 ROW 10 COL 10    -1.00000
   33:   6 ROW 11 COL 11    -1.00000
   34:   6 ROW  7 COL 12     1.00000
   35:   6 ROW  8 COL 13     1.00000
   36:   6 ROW  9 COL 14     1.00000
   37:   6 ROW 10 COL 15     1.00000
   38:   6 ROW 11 COL 16     1.00000
   39:   6 ROW  1 COL 17    -1.00000
   40:   6 ROW  2 COL 17     1.00000
   41:   6 ROW  2 COL 18    -1.00000
   42:   6 ROW  3 COL 18     1.00000
   43:   6 ROW  3 COL 19    -1.00000
   44:   6 ROW  4 COL 19     1.00000
   45:   6 ROW  4 COL 20    -1.00000
   46:   6 ROW  5 COL 20     1.00000
   47:   6 ROW  5 COL 21    -1.00000
   48:   6 ROW  6 COL 21     1.00000
   49:   4  PROD1         1     1.00000
   50:   4  PROD2         2     1.00000
   51:   4  PROD3         3     1.00000
   52:   4  PROD4         4     1.00000
   53:   4  PROD5         5     1.00000
   54:   4  PROD6         6     1.00000
   55:   4  HIRE2         7     1.75000
   56:   4  HIRE3         8     1.75000
   57:   4  HIRE4         9     1.75000
   58:   4  HIRE5        10     1.75000
   59:   4  HIRE6        11     1.75000
   60:   4  FIRE2        12     1.25000
   61:   4  FIRE3        13     1.25000
   62:   4  FIRE4        14     1.25000
   63:   4  FIRE5        15     1.25000
   64:   4  FIRE6        16     1.25000
   65:   4  STOCK1       17     2.00000
   66:   4  STOCK2       18     2.00000
   67:   4  STOCK3       19     2.00000
   68:   4  STOCK4       20     2.00000
   69:   4  STOCK5       21     2.00000
   70:  99 LOGICAL EOF

EDIT: L(IST, B(UILD, M(ODIFY, Q(UIT [1.0] Q
```

Listing 2: Data File, Product Mix Example

The output of the run is shown in Listing 3, and shows that if the following production strategy is used:

	Period					
	1	2	3	4	5	6
Std. production	175	175	150	150	400	500
Produce for stock	75		50			
Add capacity					250	100
Drop capacity			25			

then the total cost of this program is 2443.75 (units). For the 1550 units produced and sold, the average production unit costs are 1.577. This is a minimum cost for the assumptions made on the cost elements. Other assumptions on stocking, hiring and firing costs would give a different production program and cost.

If, for example, the storage costs were lower, a more uniform production would have resulted. (Try it yourself).

Listing 3: Product Mix Program Run

```
ENTER DATA FILE NAME ---> BALANCE.DATA


PROG. NAME = BALPRD
NO. ROWS   =      11
NO. COLS   =      21


START PHASE 1

ITERATION 1 OF BALPRD
ITERATION 2 OF BALPRD
ITERATION 3 OF BALPRD
ITERATION 4 OF BALPRD
ITERATION 5 OF BALPRD
ITERATION 6 OF BALPRD
ITERATION 7 OF BALPRD
ITERATION 8 OF BALPRD
ITERATION 9 OF BALPRD
ITERATION 10 OF BALPRD
ITERATION 11 OF BALPRD
END OF PHASE 1 FOR BALPRD AFTER 11 ITERATIONS

LIST & X ARRAYS

      1  STOCK3      19         200.000
      2  STOCK4      20         300.000
      3  PROD1        1         175.000
      4  STOCK1      17          75.0000
      5  HIRE3        8         125.000
      6  STOCK5      21         200.000
      7  PROD2        2         175.000
      8  PROD3        3         300.000
      9  PROD4        4         300.000
     10  PROD5        5         300.000
     11  PROD6        6         300.000
     12  M+1         33       -3318.75
     13  M+2         34           0.00000

START PHASE 2

ITERATION 1 OF BALPRD
ITERATION 2 OF BALPRD
ITERATION 3 OF BALPRD
END OF PHASE 2 FOR BALPRD AFTER 3 ITERATIONS

LIST & X ARRAYS

      1  STOCK3      19          50.0000
      2  FIRE3       13          25.0000
      3  PROD1        1         175.000
      4  STOCK1      17          75.0000
      5  HIRE5       10         250.000
      6  HIRE6       11         100.000
      7  PROD2        2         175.000
      8  PROD3        3         150.000
      9  PROD4        4         150.000
     10  PROD5        5         400.000
     11  PROD6        6         500.000
     12  M+1         33       -2443.75
     13  M+2         34          -0.00001788

PRODUCTION BALANCE EXAMPLE
```

PROBLEM 2 -- A TRANSPORTATION PROBLEM

This type of problem is concerned with the shipment of goods from M sources to N destinations. The ABAR matrix, X(i,j), has M*N columns and M+N rows. X(i,j) then represents the amount shipped from the i-th source to the Jth destination. Let us set up a problem with three sources and four destinations.

Problem Statement

In this problem, we have 3 sources, with availabilities of 6, 8 and 10 units respectively. We have 4 4 destinations with requirements for 4, 6, 8 and 6 units. (Note that the total of the availabilities MUST equal the total of the requirements; nothing is created or lost enroute).

Costs of shipment of one unit, C(i,j), between source i and destination j are:

$$\begin{array}{llll} C(1,1)=1 & C(1,2)=2 & C(1,3)=3 & C(1,4)=4 \\ C(2,1)=4 & C(2,2)=3 & C(2,3)=2 & C(2,4)=0 \\ C(3,1)=0 & C(3,2)=2 & C(3,3)=2 & C(3,4)=1 \end{array}$$

Problem Formulation

The constraints on availabilities can be expressed by:

$$\begin{aligned} X(1,1)+X(1,2)+X(1,3)+X(1,4) &= 6 \\ X(2,1)+X(2,2)+X(2,3)+X(2,4) &= 8 \\ X(3,1)+X(3,2)+X(3,3)+X(3,4) &= 10 \end{aligned}$$

and for the requirements:

$$\begin{aligned} X(1,1)+X(2,1)+X(3,1) &= 4 \\ X(1,2)+X(2,2)+X(3,2) &= 6 \\ X(1,3)+X(2,3)+X(3,3) &= 8 \\ X(1,4)+X(2,4)+X(3,4) &= 6 \end{aligned}$$

The data file is shown in Listing 4, and the program RUN output is shown in Listing 5.

We can see that Source 1 ships all 6 of its units to Dest. 2, thereby filling 2's requirements. Source 2 ships 2 units to Dest. 3 and 6 to Dest. 4. Source 3 ships 4 units to Dest. 1, and 6 to Dest. 4. The total cost for the problem is 28.

There are manual techniques available for solving small transportation problems such as this more quickly than through the use of the computer; when the problem is only a little larger than this one, then the computer is much faster.

—Continued on Page 50—

ADDRESSING THE CURSOR

by
Larry Stein
Computer Mart of New Jersey, Inc.
501 Route 27
Iselin, N.J. 08830

PART II - An Analysis of the BASIC program presented in Part I

This is the second part of an article describing the structure of a basic program, the first part being published in the March /April 1980 issue of S-100 Microsystems. In the first part, I concentrated primarily on the cursor positioning aspects of programming in BASIC. In this part I will discuss some very specific features of BASIC as well as some standards for writing programs in BASIC.

The program being described was written in Microsoft Basic version 4.51 and running under the CP/M operating system version 2.0. Other BASICs and operating systems may have different syntax and different results. It is up to the reader to identify the differences, if any, for himself/herself. However, the general concepts probably apply to all programming, in general.

This program allows the operator to specify a mailing label of any size up to 66 characters wide by 20 lines deep, enter data into that label on a formatted screen and then print out any number of these labels. The program is very useful for club meeting notices, by printing the information on pressure sensitive labels and then applying the labels to the message side of a postcard. The address side of the postcard can be likewise addressed by using one of the many available mailing label programs.

Most likely, when you sit down to write a program, it is to perform specific function and you do not intend to make it your life's work. However, any program worth writing is worth writing with some structure, so that if you need to go back to modify it, or if someone else would like to use it, the job won't have to be started from scratch.

This leads to the area of program comments. Each routine or sub-routine within your program should have a title with enough description so as to alert you where to find all of the areas of the logic of the program. If you look at the accompanying program, you will see one method of titling subroutines. Now, you do not need to make all the pretty boxes, but they do serve as targets for your eyes as you scan down the listing looking for some special routine within your program. 'Nuff said.

Within most programs there will be certain instructions or sets of instructions, called subroutines, that will be used more than just one time. These subroutines should be identified within the program and whenever they are required, they should be entered with a GOSUB statement. The LABEL program described here uses many such subroutines. The most frequently used subroutine is the cursor positioning routine located between lines 2170 and 2560. As you can see, this subroutine is GOSUBed from lines 320, 330, 350 and many other places by the statement GOSUB 2320. This method of programming makes the program shorter by not duplicating instructions and also easier to change.

Let's now look at the program in some detail and see some of the techniques employed.

Line 150 clears 2000 bytes of string space for the program variables. BASIC normally allows a fixed amount of string space, each version of BASIC allowing a different number. If you do not know how much string space is normally allowed, you can assign some arbitrary amount, say 100, and if the program needs more, you will get some message such as 'OUT OF STRING SPACE' which alerts you to allocate more. Not very scientific, but it works. If you want a more scientific method, consult your BASIC manual for the method of calculating string space.

Line 160 is a dimension statement for A$ and contains a comment indicating that it is a dummy statement. This is for reasons of consistency. Later in the program, it is necessary to create the dimension of A$ depending data being entered from the console and we may do in more than one time. In order to dimension an array that has been previously dimensioned, we must first ERASE the array. The first time this is encountered, line 2680-2690 or line 3030-3040, unless the array has already been dimensioned, an error will occur.

Lines 170-260 will present the program title and adjust the screen display depending upon the terminal selected. Note that the SOL screen is only 64 characters wide while the other two choices are 80 characters wide.

Line 220 is a special input statement that allows the user to enter data from the keyboard without using the return or enter key. This instruction accepts one character (1) into the variable Z$. It can be used whenever the programmer knows exactly how many characters are needed from the console.

Lines 270-380 continue the program sign-on messages.

Line 390 is called a program switch. The first time the program encounters this statement, BG is equal to 0, therefore the program does not GOTO 490 and will execute the following statements. Line 480 sets BG equal to a 1, so that the next time line 390 is encountered, it will skip the questions asked in lines 400-470. This is known as a one-time switch.

Lines 400-470 allow the user to define which characters on the keyboard will be used to backspace, forward space, insert and delete. Depending on the terminal used, the operator may select any keys which are convenient. These questions utilize the sub-routine at line 2830 for data input because any characters are allowed, including control characters which most BASICs reject.

Lines 490-540 allow the user to select a label previously stored on diskette or a standard label defined elsewhere in the program. Note that all YES/NO questions allow both upper or lower case answers. Upper/lower case translations can be accomplished using a more sophisticated subroutine (line 1280 converts Z$ to upper case), however for single character entry, this method seems acceptable.

Line 600 uses the subroutine at 2620 to determine the label size. The instructions from line 2620-2760 could have been inserted here at line 600 instead of using the GOSUB; your preference.

Lines 660-720 will display the label format on the screen depending on the size of the label you specify. It will number the lines from 1 to the size specified and will show the left and right boundaries of the label.

Lines 780-810 will display the current contents of the label. On first input, these lines will be blank, but later if the label is to be changed, these lines will re-display the current label for the A$ array.

Lines 820-990 are used to accept input from the console into the proper line of the A$ array. Note that if the input statements in lines 890-940 encounter certain characters, namely those entered as the cursor moving commands, special subroutines are executed to handle the cursor moving and the aligning of the data in A$. Also, in line 940 if a backspace character (ASCII value 8) or a delete character (ASCII value 127) are entered, they will be ignored by the program. As valid characters are entered, they are placed into the current line buffer, L$, in the proper place. This is what allows the user to use the forward and backward space instructions and still maintain the correct data. When the data is entered in its entirety, it is then placed into the proper line of the array A$ in statement 1000.

Note: these 18 lines of code along with the subroutines at 1860, 2050 and 2130 should be completely studied to understand the operation of the cursor moving aspects of the data entry of this program if you wish to use this code in another program.

Lines 1040-1080 allow the user to make any changes to the label by simply repositioning the cursor to the beginning of the label, and going back to the data entry routine at 660.

Lines 1140-1390 allow the user to save the label on diskette for future use. The program stores the labels on the diskette with the file suffix (.LAB). First, the directory of the diskette selected is displayed, showing those files which have the suffix (.LAB). Then the user is asked to supply a new name. This name is converted to upper case characters in line 1280. When the label is stored on diskette, the first record of the file contains the width and length of the label and the remaining lines are the data entered into the label.

Lines 1450-1570 allow alignment of the labels by printing X's.

Lines 1630-1700 print the number of labels requested and ask if more labels are desired. If so, either the same label or a different label can be printed.

Lines 1860-1900 keep the position of the data within the current label line when using the backspace and forward space keys. The screen position is automatically adjusted in the data entry routine.

Lines 1960-1990 handle the end of line condition. When the cursor is at the end of the label line, the only allowable characters are the carriage return and the backspace character.

Lines 2050-2160 handle the deletion and insertion of characters into the label text. This is done by readjusting the position within the current line and redisplaying the line on the screen.

Lines 2170-2560 handle the cursor positioning. This was described in detail in the previous article.

Lines 2620-2770 determine the label size. The label parameters are stored in the variables WD, LN, SK and NB. The variables WD and LN are also stored in the disk file if the label is stored on the diskette, so when the label is redisplayed, it is the correct size.

Lines 2830-2870 provide for direct input from the port of the computer. If the standard input statement in BASIC is used, then no control characters will be allowed as input. Since this program allows the insert, delete, backspace and forward space characters to be any characters, including control characters, some other method of input had to be used. This must be configured for the computer you are using. If you do not know how to directly input from your computer, the following routine may be substituted for lines 2830-2870:

```
2830 IN$=INPUT$(1)
2840 IN=VAL(IN$)
2850 REM
2860 REM
2870 RETURN
```

The purpose for the REM at lines 2850 and 2860 are only to maintain the line numbering consistent. They may be removed.

```
10 REM    *********** PROGRAM NAME "LABELS"  11/6/79 **************
20 REM    *                                                        *
30 REM    *************** WRITTEN BY LARRY STEIN *****************
40 REM    *                                                        *
50 REM    **********************************************************
60 REM    *                                                        *
70 REM    *          PROGRAM FOR DISKETTE LABEL PREPARATION        *
80 REM    *                                                        *
90 REM    **********************************************************
100 REM   **********************************************************
110 REM   *                                                        *
120 REM   *     INITIALIZATION ROUTINE FOR SPECIFIC TERMINALS      *
130 REM   *                                                        *
140 REM   **********************************************************
150 CLEAR 2000
160 DIM A$(2) : REM DUMMY DIMENSION SO THAT ERASE WILL WORK LATER
170 PRINT:PRINT "THIS PROGRAM IS DESIGNED FOR ANY OF THE FOLLOWING:"
180 PRINT:PRINT "1 - LEAR SIEGLER ADM-3A"
190 PRINT:PRINT "2 - HAZELTINE 1500"
200 PRINT:PRINT "3 - SOL TERMINAL COMPUTER"
210 PRINT:PRINT "ENTER THE NUMBER OF THE ONE YOU ARE USING ";
220 Z$=INPUT$(1):PRINT Z$
230 IF Z$="1" THEN AM=1:WIDTH 80:GOTO 320
240 IF Z$="2" THEN AM=2:WIDTH 80:GOTO 320
250 IF Z$="3" THEN AM=3:WIDTH 64:GOTO 320
260 GOTO 170
270 REM   **********************************************************
280 REM   *                                                        *
290 REM   *            BEGINNING OF PROGRAM - TITLE                *
300 REM   *                                                        *
310 REM   **********************************************************
320 Y=0:X=0:GOSUB 2320
330 Y=11:X=14:GOSUB 2320
340 PRINT "DISKETTE LABEL PREPARATION PROGRAM - NOVEMBER 6, 1979"
350 Y=15:X=32:GOSUB 2320
360 PRINT "LARRY STEIN"
370 FOR Z=1 TO 1000:NEXT Z : REM SET FOR DELAY OF TITLE ON SCREEN
380 Y=0:X=0:GOSUB 2320
390 IF BG=1 THEN 490
400 PRINT "ENTER THE CHARACTER YOU WANT TO USE FOR BACKSPACE ";
410 GOSUB 2830:BS=IN:PRINT CHR$(BS)
420 PRINT "ENTER THE CHARACTER YOU WANT TO USE FOR FORWARD SPACE ";
430 GOSUB 2830:FS=IN:PRINT CHR$(FS)
440 PRINT "ENTER THE CHARACTER YOU WANT TO USE FOR INSERTING ";
450 GOSUB 2830:IT=IN:PRINT CHR$(IT)
460 PRINT "ENTER THE CHARACTER YOU WANT TO USE FOR DELETING ";
470 GOSUB 2830:DT=IN:PRINT CHR$(DT)
480 BG=1
490 PRINT "DO YOU WANT TO USE A PREVIOUSLY SAVED LABEL (Y/N) ";
500 Z$=INPUT$(1):PRINT Z$
510 IF Z$="Y" OR Z$="y" THEN 3280
520 PRINT "DO YOU WANT STANDARD PRODIGY LABELS (Y/N) ";
530 Z$=INPUT$(1):PRINT Z$
540 IF Z$="Y" OR Z$="y" THEN 2930
550 REM   **********************************************************
560 REM   *                                                        *
570 REM   *                 GET LABEL PARAMETER                    *
580 REM   *                                                        *
590 REM   **********************************************************
600 GOSUB 2620
610 REM   **********************************************************
620 REM   *                                                        *
630 REM   *             DISPLAY LEFT SIDE OF SCREEN                *
640 REM   *                                                        *
650 REM   **********************************************************
660 Y=0:X=0:GOSUB 2320
670 PRINT
680 FOR N=1 TO LN
690 Y=N:X=1:GOSUB 2320
700 N$=STR$(N):IF LEN(N$)=2 THEN N$=" "+N$
710 PRINT "LINE ";N$;"   ";TAB(WD+14);" "
720 NEXT N
730 REM   **********************************************************
740 REM   *                                                        *
750 REM   *                GET LABEL INFORMATION                   *
760 REM   *                                                        *
770 REM   **********************************************************
780 FOR N=1 TO LN
790 Y=N:X=12:GOSUB 2320
800 PRINT A$(N)
810 NEXT N
820 FOR N=1 TO LN
830 I=1
840 L$=A$(N)
850 FOR M=1 TO WD
860 IF I=0 THEN 880
870 Y=N:X=11+M:GOSUB 2320
880 GOSUB 2830
890 IF IN=13 THEN M=WD:GOTO 990 : REM CARRIAGE RETURN
900 IF IN=BS THEN 1860 : REM MOVE CURSOR TO THE LEFT
910 IF IN=FS THEN 1860 : REM MOVE CURSOR TO THE RIGHT
920 IF IN=DT THEN 2050 : REM DELETE CHARACTER
930 IF IN=IT THEN 2130 : REM INSERT CHARACTER
940 IF IN=8 OR IN=127 GOTO 880 : REM CHARACTERS TO BE NOT CONSIDERED
                                                               AT ALL
950 I=0
960 MID$(L$,M,1)=IN$
970 PRINT IN$;
980 IF M=WD THEN 1960
990 NEXT M
1000 A$(N)=L$
1010 PRINT
1020 NEXT N
1030 PRINT
1040 PRINT "DO YOU WANT TO MAKE ANY CHANGES (Y/N) ";
1050 Z$=INPUT$(1):PRINT Z$
1060 IF Z$="N" OR Z$="n" THEN 1140
1070 Y=0:X=0:GOSUB 2320
1080 GOTO 660
1090 REM **********************************************************
1100 REM *                                                        *
1110 REM *            ROUTINE TO SAVE LABELS ON DISK              *
1120 REM *                                                        *
1130 REM **********************************************************
1140 PRINT "DO YOU WANT TO SAVE THIS LABEL ON DISK (Y/N) ";
1150 Z$=INPUT$(1):PRINT Z$
1160 IF Z$="N" OR Z$="n" THEN 1450
1170 PRINT "ENTER THE DRIVE ON WHICH LABEL IS TO BE STORED (A,B,C,D) ";
1180 D$=INPUT$(1):PRINT D$
1190 D$=CHR$(ASC(D$) AND &HDF)
1200 IF D$["A" OR D$|"D" THEN 1170
1210 D$=D$+":"
1220 F$=D$+"*.LAB"
1230 PRINT
1240 FILES F$
1250 PRINT:PRINT
1260 PRINT "ENTER A FILE NAME *** NOT *** IN THE ABOVE LIST"
1270 LINEINPUT "USE FILE NAME ONLY, NO EXTENSION ";Z$
1280 FOR N=1 TO LEN(Z$):MID$(Z$,N,1)=CHR$(ASC(MID$(Z$,N,1)) AND &HDF):NEXT N
1290 F$=D$+Z$+".LAB"
1300 OPEN "O",1,F$
1310 PRINT#1,WD$+","+LN$
1320 FOR N=1 TO LN
1330 PRINT#1,A$(N)
1340 NEXT N
1350 CLOSE
1360 PRINT
1370 F$=D$+"*.LAB"
1380 FILES F$
1390 PRINT
1400 REM **********************************************************
1410 REM *                                                        *
1420 REM *                ROUTINE TO ALIGN LABELS                 *
1430 REM *                                                        *
1440 REM **********************************************************
1450 PRINT "READY THE LABELS IN THE PRINTER AND PRESS RETURN ";
1460 Z$=INPUT$(1):PRINT
1470 PRINT "DO YOU WANT TO ALIGN THE LABELS (Y/N) ";
1480 Z$=INPUT$(1):PRINT Z$
1490 IF Z$="N" OR Z$="n" THEN 1630
1500 FOR N=1 TO LN
1510 LPRINT STRING$(WD,88)
1520 NEXT N
1530 FOR N=1 TO SK
1540 LPRINT
1550 NEXT N
1560 PRINT "DO YOU NEED MORE ALIGNMENT (Y/N) ";
1570 Z$=INPUT$(1):PRINT Z$:GOTO 1490
1580 REM **********************************************************
1590 REM *                                                        *
1600 REM *                ROUTINE TO PRINT LABELS                 *
1610 REM *                                                        *
1620 REM **********************************************************
1630 FOR M=1 TO NB
1640 FOR N=1 TO LN
1650 LPRINT A$(N)
1660 NEXT N
1670 FOR N=1 TO SK
1680 LPRINT
1690 NEXT N
1700 NEXT M
1710 PRINT "DO YOU WANT TO PRINT MORE LABELS (Y/N) ";
1720 Z$=INPUT$(1):PRINT Z$
1730 IF Z$="Y" OR Z$="y" THEN 1760
1740 STOP
1750 GOTO 1740
1760 PRINT "DO YOU WANT TO PRINT THE SAME LABEL (Y/N) ";
1770 Z$=INPUT$(1):PRINT Z$
1780 IF Z$="N" OR Z$="n" THEN 380
1790 GOSUB 2750
1800 GOTO 1040
1810 REM **********************************************************
1820 REM *                                                        *
1830 REM *        ROUTINE TO MOVE CURSOR RIGHT AND LEFT           *
1840 REM *                                                        *
1850 REM **********************************************************
1860 I=1
1870 IF IN=BS AND M[|1 THEN M=M-1:GOTO 870
1880 IF IN=FS AND M[|WD THEN M=M+1:GOTO 870
1890 I=0
1900 GOTO 870
1910 REM **********************************************************
1920 REM *                                                        *
1930 REM *       ROUTINE TO HANDLE CURSOR AT END OF FIELD         *
1940 REM *                                                        *
1950 REM **********************************************************
1960 GOSUB 2830
1970 IF IN=BS THEN I=1:GOTO 870
1980 IF IN$=CHR$(13) THEN 990
1990 GOTO 1960
2000 REM **********************************************************
2010 REM *                                                        *
2020 REM *             ROUTINE TO DELETE A CHARACTER              *
2030 REM *                                                        *
2040 REM **********************************************************
2050 MID$(L$,M,WD-M+1)=MID$(L$,M+1,WD-M)+" "
2060 PRINT MID$(L$,M,WD-M+1)
2070 GOTO 870
2080 REM **********************************************************
2090 REM *                                                        *
2100 REM *             ROUTINE TO INSERT A CHARACTER              *
2110 REM *                                                        *
2120 REM **********************************************************
2130 L1$=MID$(L$,M,WD-M)
2140 MID$(L$,M,WD-M+1)=" "+L1$
2150 PRINT MID$(L$,M,WD-M+1)
2160 GOTO 870
```

```
2170 REM *********************************************************
2180 REM *                                                       *
2190 REM *           UNIVERSAL CURSOR POSITIONING ROUTINE        *
2200 REM *                                                       *
2210 REM *********************************************************
2220 REM *                                                       *
2230 REM * THE FOLLOWING INSTRUCTION CORRECTS FOR THE FACT THAT *
2240 REM * BASIC WILL OUTPUT A CR/LF AUTOMATICALLY, AFTER A      *
2250 REM * CERTAIN NUMBER OF CHARACTERS ARE SENT TO THE SCREEN  *
2260 REM * SINCE THESE CURSOR POSITIONING ROUTINES SEND MANY    *
2270 REM * CHARACTERS TO THE SCREEN WITHOUT A CR, WE WILL       *
2280 REM * ARBITRARILY SEND A CR (PRINT) TO THE SCREEN EACH TIME*
2290 REM * WE EXECUTE THIS ROUTINE 5 TIMES.                     *
2300 REM *                                                       *
2310 REM *********************************************************
2320 D=D+1:IF D=5 THEN D=0:PRINT
2330 ON AM GOTO 2390,2530,2460
2340 REM *********************************************************
2350 REM *                                                       *
2360 REM *          CURSOR POSITIONING FOR ADM-3A TERMINAL       *
2370 REM *                                                       *
2380 REM *********************************************************
2390 IF Y+X=0 THEN PRINT CHR$(26)
2400 PRINT CHR$(27)+CHR$(61)+CHR$(32+Y)+CHR$(32+X);:RETURN
2410 REM *********************************************************
2420 REM *                                                       *
2430 REM *   CURSOR POSITIONING FOR THE SOL TERMINAL COMPUTER    *
2440 REM *                                                       *
2450 REM *********************************************************
2460 IF Y+X=0 THEN PRINT CHR$(11);:RETURN
2470 PRINT CHR$(27)+CHR$(2)+CHR$(Y-1)+CHR$(27)+CHR$(1)+CHR$(X-1);:RETURN
2480 REM *********************************************************
2490 REM *                                                       *
2500 REM *    CURSOR POSITIONING FOR HAZELTINE 1500 TERMINAL     *
2510 REM *                                                       *
2520 REM *********************************************************
2530 IF Y+X=0 THEN PRINT CHR$(126)+CHR$(28);:RETURN
2540 IF X[32 THEN X=X+96
2550 Y=Y+96
2560 PRINT CHR$(126)+CHR$(17)+CHR$(X-1)+CHR$(Y-1);:RETURN
2570 REM *********************************************************
2580 REM *                                                       *
2590 REM *              ROUTINE TO DETERMINE LABEL SIZE          *
2600 REM *                                                       *
2610 REM *********************************************************
2620 LINEINPUT "ENTER LABEL WIDTH (IN CHARACTERS) ";WD$
2630 WD=VAL(WD$)
2640 IF WD|65 OR WD[1 THEN PRINT "LABEL WIDTH OUT OF RANGE (1-65) ":GOTO 2620
2650 LINEINPUT "ENTER NUMBER OF PRINT LINES PER LABEL ";LN$
2660 LN=VAL(LN$)
2670 IF LN|20 OR LN[1 THEN PRINT "LABEL LENGTH OUT OF RANGE (1-20) ":GOTO 2650
2680 ERASE A$
2690 DIM A$(LN)
2700 FOR N=1 TO LN
2710 A$(N)=STRING$(WD,32)
2720 NEXT N
2730 LINEINPUT "ENTER NUMBER OF LINES TO SKIP BETWEEN LABELS ";SK$
2740 SK=VAL(SK$)
2750 LINEINPUT "ENTER TOTAL NUMBER OF LABELS TO BE PRINTED ";NB$
2760 NB=VAL(NB$)
2770 RETURN
2780 REM *********************************************************
2790 REM *                                                       *
2800 REM *        ROUTINE FOR DIRECT INPUT FROM TERMINAL         *
2810 REM *                                                       *
2820 REM *********************************************************
2830 OUT 29,1
2840 WAIT 29,1,0
2850 IN=INP(28)
2860 IN$=CHR$(IN)
2870 RETURN
2880 REM *********************************************************
2890 REM *                                                       *
2900 REM *     ROUTINE TO GENERATE PRODIGY DISKETTE LABELS       *
2910 REM *                                                       *
2920 REM *********************************************************
2930 Y=0:X=0:GOSUB 2320
2940 PRINT "ENTER DISKETTE NUMBER XXXX";STRING$(4,8);
2950 LINEINPUT DS$
2960 PRINT "ENTER UNIT NUMBER     XXXX";STRING$(4,8);
2970 LINEINPUT UN$
2980 PRINT "ENTER DATE (MM/DD/YY) XX/XX/XX";STRING$(8,8);
2990 LINEINPUT DT$
3000 PRINT "ENTER DEALER NAME     XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX";STRING$(30,8)
3010 LINEINPUT DL$
3020 IF LEN(DL$)|30 THEN PRINT "DEALER NAME TOO LONG !":GOTO 3000
3030 ERASE A$
3040 DIM A$(8)
3050 A$(1)=" P R O D I G Y   S Y S T E M S,  I N C."
3060 A$(2)=" DISKETTE #"+DS$+"   UNIT #"+UN$+"  DATE "+DT$
3070 A$(3)=" "
3080 A$(4)=" DEALER: "+DL$
3090 A$(5)="  MASTER DISKETTE - RETURN IMMEDIATELY"
3100 A$(6)=" "
3110 A$(7)="COPYRIGHT (C) 1979 PRODIGY SYSTEMS, INC."
3120 A$(8)="        ALL WORLDWIDE RIGHTS RESERVED"
3130 LINEINPUT "ENTER TOTAL NUMBER OF LABELS TO BE PRINTED ";NB$
3140 NB=VAL(NB$)
3150 WD$="40"
3160 WD=VAL(WD$)
3170 LN$="8"
3180 LN=VAL(LN$)
3190 SK$="1"
3200 SK=VAL(SK$)
3210 GOTO 1450
3220 STOP
```

THE MM-103 DATA MODEM AND COMMUNICATIONS ADAPTER

S-100 bus compatible

FCC APPROVED

Both the modem and telephone system interface are FCC approved, accomplishing all the required protective functions with a miniaturized, proprietary protective coupler.

HIGH QUALITY

-50 dBm sensitivity. Auto answer. Auto originate. Auto dialer with computer-controlled dial rate. 61 to 300 baud (anywhere over the long-distance telephone network), rate selection under computer control. Flexible, software-controlled, maskable interrupt system.

WARRANTY

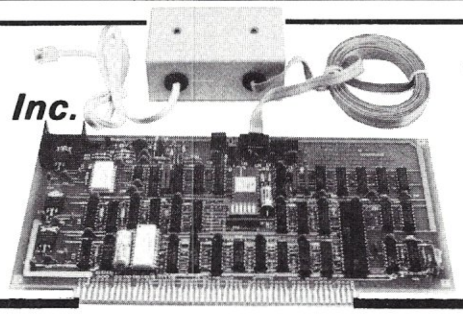
One year limited warranty. Ten-day unconditional return privilege. Minimal cost, 24-hour exchange policy for units not in warranty.

ASSEMBLED & TESTED

Not a kit! (FCC registration prohibits kits)

LOW PRICE—$359.95— For Modem AND Coupler plus shipping & handling

VISA master charge



Potomac Micro-Magic, Inc.

Write for brochure:
First Lincolnia Bldg., Suite B1
4810 Beauregard St.
Alexandria, Va. 22312

Call for further information:
VOICE: (703) 750-3727
MODEM: (703) 750-0930 (300 baud)

Micro Magic

SOFTWARE DIRECTORY

In each issue of S-100 MICROSYSTEMS we will have this catalog listing of S-100 system software. If you have a software package you are offering for sale and want to be listed then send us the information in the format shown. All information must be included. We reserve the right to edit and/or reject any submission.

Program Name: APL
Hardware System: 8080/8085/Z80 CP/M
Minimum Memory Size: 44K
Description: Implementation of most of the APL functions and functions of full APL, including n-dimensional inner and outer product, reduction, compression, general transpose, reversal, take, drop; execute and format, system functions and variables, system commands. Runs in either ASC II or bit-pairing ASC II-APL character sets. Can run with user-supplied I/O drivers. Shared variable mechanism allows CP/M disk I/O. Uses Abranis descriptor calculus and shared data storage to save memory space and execution time. Comes with optional driver program for video display with programmable character generator.
Release: October 1980
Price: $350 (NJ residents add 5% sales tax)
Included with price: CP/M disk and Users Manual
Author: Erik T. Mueller
Where to purchase it:
Softronics
36 Homestead Lane
Roosevelt, NJ 08555

Program Name: MDBS.DRS: A Dynamic Restructuring System for MDBS Data Bases
Hardware System: Z-80, 8080,6502
Minimum Memory Size: 19K plus approximately 3K for buffers (Z-80)
23K plus approximately 3K for buffers (8080)
29K plus approximately 3K for buffers (6502)
Language: Written is assembly language; interfaces with BASIC, COBOL, FORTRAN and assembly language.
Description: MDBS.DRS is a system which can be used to alter the structure of an existing MDBS data base. Its primary use is to permit an MDBS user to include new data fields in existing data records, to define new data records or set relationships in the data base or to delete existing fields, records or sets from a data base. These functions can all be performed without the need to dump the data base contents and reload it, saving much time for the data base user.
Release: Currently available
Price: $100.00 (Manual only: $5.00)
Included with price: MDBS.DRS system and manual with sample application program
Author: Micro Data Base Systems
Where to purchase it:
Micro Data Base Systems
PO Box 248
Lafayette, IN 47902

Program Name: Diagnostics I
Hardware System: CP/M 5" & 8"
Minimum Memory Size: 24K
Language: Supplied as object only
Description: Comprehensive set of CP/M compatible system check-out programs. Finds hardware errors in system, confirms suspicions, or just gives system a clean bill of health. Tests: Memory, Disk, CPU (8080/8085/Z80), CRT, and printer.
Release: now
Price: $50
Included with price: Complete user manual and Discette.
Author: SuperSoft Associates
Where to purchase it: Direct from us or dealers everywhere.
SuperSoft
Box 1628
Champaign, IL 61820

Program Name: MDBS: A Full Network Data Base Management System
Hardware System: Z-80, 8080, 6502
Minimum Memory Size: 17K plus approximately 3K for buffers. (Z-80)
20K plus approximately 3K for buffers. (8080)
26K plus approximately 3K for buffers. (6502)
Language: Written in assembly language; interfaces with BASIC, COBOL, FORTRAN and assembly language.
Description: MDBS is a full network data base system expressly designed for microcomputer use. Details of physically storing, sorting, updating and retrieving data are handled by the MDBS system, freeing the programmer from the tedium and complexity of data management tasks. The amount of data stored is limited only by the amount of on-line disk storage available. Up to 254 different types of data records may be processed, each of which can contain up to 255 data fields. Read/Write access protection is provided at the record, field and set levels. Use of the MDBS system can significantly reduce the cost of developing and maintaining data oriented applications programs.
Release: Currently available
Price: $750.00 - $825.00 (Manual only: $35.00)
Included with price: 260 page User's Manual, MDBS.DDL Data Definition Language, MDBS.DMS Data Management System and a sample program
Author: Micro Data Base Systems
Where to purchase it:
Micro Data Base Systems
PO Box 248
Lafayette, IN 47902

Program Name: Encode/Decode I & II
Hardware System: CP/M 5" & 8" disks
Minimum Memory Size: 24K CP/M
Language: Supplied as object only
Description: Complete software security system for CP/M. Transforms data stored on disk into coded text which is completely unrecognizable. Encode/decode supports multiple security levels and passwords. A user defined combination (one billion possible) is used to code and decode a file. Encode/decode is available in two versions: Level I provides a level of security for normal use. Level II provides enhanced security for the most demanding needs.
Release: Now
Price: $50/$100
Included with price: User manual and diskette
Author: SuperSoft Associates
Where to purchase it: Direct from us or dealers everywhere
SuperSoft
Box 1628
Champaign, IL 61820

Program Name: HDBS: An Extended Hierarchical Data Base Management System
Hardware System: Z-80, 8080,6502
Minimum Memory Size: 17K plus approx. 3K for buffers (Z-80)
20K plus approximately 3K for buffers (8080)
26K plus approximately 3K for buffers (6502)
Language: Written in assembly language; interfaces with BASIC, COBOL, FORTRAN and assembly language.
Description: HDBS is a data base management system similar to the MDBS system, except that the data structures which can be handled by HDBS are limited to hierarchics. For many applications a hierarchical system will suffice. A limited read/write protection is available in HDBS at the data base file level. HDBS is designed for use by hobbyists and applications programmers with relatively straight-forward data representation needs.
Release: Currently available
Price: $250.00 - $375.00 (Manual only: $35.00)
Included with price: 260 page User's Manual, HDBS.DDL Data Definition Language, HDBS.DMS Data Management System and a sample program
Author: Micro Data Base Systems
Where to purchase it:
Micro Data Base Systems
PO Box 248
Lafayette, IN 47902

IS YOUR COMPUTER OUT OF SORTS?

by
Chris Terry
324 E. 35th St.
New York, NY 10016

Use These Guidelines to Choose a Tonic For It – the sorting method that best suits both your system and your application.

From time to time I get asked 'What is the best sorting method?' If you search the literature, you find hundreds of sorting methods, each of which has some attraction and gains an ounce or two of efficiency for particular types of data, but they all fall into a few general classes, and all the methods in a given class have similar general characteristics. Each class has advantages and disadvantages of its own. Thus, in its broad form, the question is almost meaningless. Best from what point of view? Simplicity? Speed? Ease of using the result? Economy of memory space? You have to consider all these things, and more. There really is no 'best' method that gives a clear-cut advantage under all circumstances and for all types of data encountered.

Quite a number of articles on sorting have appeared in the personal computing journals, most of which extol one, or perhaps two, sorting methods, and there is an overwhelming mass of material in textbooks and professional journals, but nobody in the personal computing field has so far assembled in one place the basic information that is needed to make an intelligent choice of sorting algorithm. This article is an attempt to plug that gap. It is not intended for end-users who buy complete software packages -- one hopes that the sort/merge routines included in such packages are already optimized for the application. Rather, it is intended for hobbyists who need a sort for their own system or application programs but don't know how to make a choice.

I have therefore chosen to test and compare five common sorting algorithms, all of which are classed as INTERNAL sorts -- that is, all of the items to be sorted are available in main memory. Three of these methods (Bubble, Shell-Metzner, and Heap) can be further classified as **exchange sorts;** when two items are compared and found to be out of order, they are physically swapped. The Tree Sort does not swap items, but constructs (in a separate area of memory) an ordered list of pointers to the original items. The remaining method (Quicksort) is an example of a **partitioning** sort. These terms will be explained later, in the comments on the individual methods.

The general characteristics of these five methods are summarized in Table 1. Table 2 lists execution times for three file sizes in each method on an Altair 8800a (8080A CPU), an Apple II (6502 CPU), and a TRS-80 (Z-80 CPU), using a number of different BASIC interpreters.

The books and articles on sorting that I have found most readable and most generally useful for my own microcomputer applications are listed in the bibliography. Knuth, of course, is the classic source of information. However, his approach is highly mathematical, and his programming examples are in MIX, an assembly language for a hypothetical machine which does not resemble any current microcomputer. If you are not mathematically inclined, you will find Lorin's book much more readable and rewarding. It is written (in beautiful and lucid English) "for a programmer who desires a complete but pragmatic knowledge of sorting and sort systems, and does not wish to learn a specific programming language, advanced statistics, or a hypothetical machine in order to obtain that knowledge." The book fully lives up to this promise. It discusses all of the factors affecting sort performance, as well as the mechanisms of both simple and complex methods. The extremely clear descriptions are enhanced by really excellent diagrams and trace examples.

GENERAL CONSIDERATIONS

FILE SIZE. For small files with fewer than 50 records, execution speed may be less important than simple coding. As file size grows, differences between execution speeds become more noticeable and carry more weight in the choice of method.

RECORD SIZE. If record size is large in comparison to the sort key length, it may be worth while to build a table containing only sort keys and pointers to the associated records, and to sort this table instead of the records. This procedure becomes worth while when the time spent in building the key/pointer table is significantly less than the time that would be spent in moving large records around during the sort. Moving large records may never present a problem in Z-80 machines which have an efficient block-move instruction, but experimentation along these lines should certainly be done if an 8080 machine is used.

RECORD ORGANIZATION. All of the methods described, except the Tree sort, require that items to be sorted should be of exactly the same length. A single record of abnormal length can cause total destruction of the file by the sort routine. If the file was created from the keyboard, record length should be checked by the computer before entering the sort, to ensure that no invisible control characters crept in. If variable-length records are to be sorted, the keys MUST be extracted and put into a table for sorting.

LANGUAGE. The BASIC interpreters tested on microcomputers are all abominably slow in sorting (see Table 2). If the application program is written in BASIC, IT SHOULD CALL A MACHINE-language sort routine which will run the same algorithm 70-100 times faster than the BASIC interpreter can do it. However, if the sort routine must be written in BASIC, try to match the sort method to the peculiarities of your BASIC interpreter. For example, the Processor Technology interpreter runs Tree sort about 4.5 percent faster than Quicksort. Also, you may obtain some speed increase by concatenating multiple statements per line, if your interpreter allows this. For the sake of portability and simplicity, no attempt was made to optimize the test program in this way.

If your application must sort files with unknown or very widely varying data distribution, Shell-metzner may be better, although slower, because its performance is more consistent. Heapsort is said (by Knuth and others) to be inefficient for small files; my experimental timings do not support that idea, unless "small" is taken to mean "less than 10 items" -- and for such tiny lists Bubble is the obvious choice because of its simple and compact coding.

MEMORY USAGE. Some methods, such as the Tree sort and all insertion methods, require a work space equal to or larger than the unsorted list. If the available memory space is limited, such methods may not be feasible for large files.

NATURE OF THE DATA. Some methods (notably the Quicksort) are extremely sensitive to the distribution of the data. If your application (like many of mine) involves adding records to the end of a file and then resorting the file, be very cautious in using Quicksort. Versions that are optimized for randomly distributed data become very slow when they encounter nearly-ordered data; versions that are optimized for nearly-ordered data become slow when they encounter random data.

EXECUTION SPEED. Execution time for a given sort run is determined by two groups of computer operations: 1) Array/String compares and Array/String exchanges, which have a non-linear relationship to file size; and 2) overhead operations such as address computation, or the addition, subtraction, and comparison of simple variables, which have a linear relationship to file size. In Table 1, overhead operations are represented by the variable K. As file size increases, the linear increase in K has much less influence on total run time than the exponential growth of comparisons and exchanges.

Table 1. General Characteristics of Five Sorting Methods

METHOD	BASIC STMTS*	EXTRA WORK SPACE	SPEED FACTOR	PRINTING OF OUTPUT	REMARKS
Bubble	8	1 record, for swaps	K*(N**2) Very Slow	Linear dump of sorted list	Intolerably slow for large files
Shell-Metzner	16	1 record, for swaps	K*N*log2(N) Fast	Linear dump of sorted list	Very consistent and reliable -- no pathological cases
Heap	22	1 record, for swaps	K*N*log2(N) Very Fast	Linear dump of sorted list	On small files (<50) may run slower than Shell-M, but generally faster on large files
Tree	65	Array for N+log2(N) pointers	Super Fast; some BASICs run it faster than Quicksort	Print routine must access original records from the linkage list	Too complex to be worth while for small files; excellent for large files of integers or for files with long or variable-length records.
Quick	34	log2(N)+1	Slow to Super Fast	Linear dump of sorted list	Very sensitive to data; best case approaches K*N, worst case approaches K*(N**2), average around K*log2(N)

*Executable statements only; does not include REMARKs

Optimizing overhead code can give only small increases in speed. Reduction of the exponential factors is the only way to obtain a substantial speed increase; it is more difficult to do, however, and increases the complexity of the code. All of the work in this field has been aimed at finding the best way to accomplish a reduction of comparisons and exchanges without nullifying the benefits by excessive code complexity. To take the concrete example of a 200-item file to be sorted by a Processor Tech BASIC routine, 7.5 seconds gained by shifting from Shell-Metzner to Tree may not be worth the entry, checking, and memory space entailed by 50 extra BASIC statements -- but for a list of 5000 items, the gain may be several minutes, and so be worth while.

THE FIVE METHODS

BUBBLE SORT. See Flow Chart 1. This is the simplest of all sorts to implement (no more than 8 BASIC statements), but is also the most inefficient by a whole order of magnitude. The execution time is proportional to the SQUARE of the number of items to be sorted, because each item is compared to every other item, not once but many times. A bubble sort of 1000 numbers logged nearly half a million comparisons and a quarter of a million swaps. Using a switch to terminate the run after a pass in which no swaps took place requires more code and only reduced execution time by about 10 percent. There is no reason to use this method for any list of more than 20 items, since the Shell-Metzner sort, with only 16 BASIC statements, can do the job 30-50 times as fast on a large computer with a good BASIC interpreter, and at least 3-4 times as fast on an 8080 with a merely moderate BASIC. The only additional space required by the bubble sort is enough to hold one record (or key) during swaps.

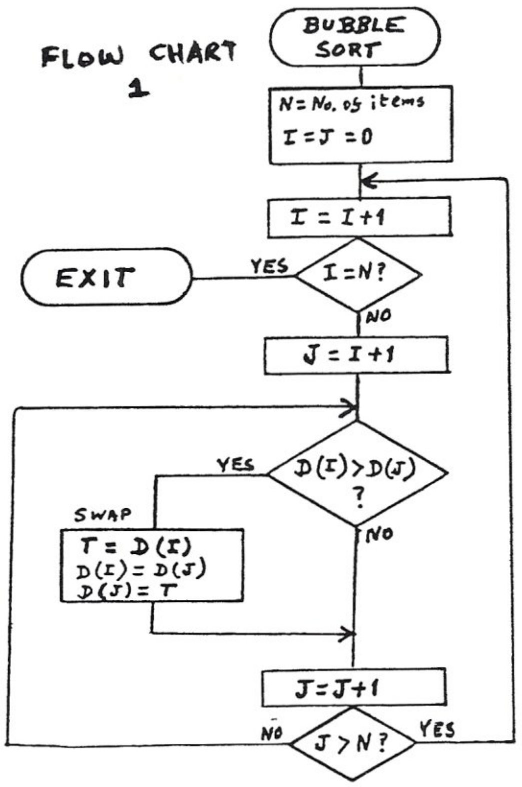
HEAPSORT. See Flow Chart 3. Knuth remarks that this is a very inefficient method for small files,

Table 2. Comparative Timings (in Seconds) for Sorting Methods on Various Machines

File Size	SORTING METHOD Bubble	Heap	Shell-M	Tree	Quick	Comments
Xerox Sigma 9, Xerox BASIC						
1000	50.3	1.8	1.8	1.2	<1	Sigma 9 is the Xerox equivalent of IBM 370-158
2000	180.5	3.0	4.0	2.3	1.8	
3000	-	4.6	6.0	4.0	2.5	
8080A, Processor Tech. Extended Cassette BASIC						
50	25	14	14.5	12	14.5	This interpreter runs
200	403.5	75	74.5	67	72.5	Treesort 4.5% faster
400	-	179	177.5	149.5	156.5	than Quicksort.
8080A, BASIC-E Compiler/Interpreter, Thinker Toys Disk with CP/M						
50	39	13	11	9	8	
200	-	68	78	48	47	
400	-	156	198.5	105	95	
8080A, Machine-language Sort						
210	-	-	1.5	-	-	Assembler Symbol Table containing 210 7-character strings
6502 Processor, APPLE Integer BASIC						
50	19	12	8	-	-	
200	316	59	57	-	-	Fastest microcomputer
400	-	146	130	-	-	BASIC tested.
Z-80 Processor, TRS-80 Level II BASIC						
50	47	23	22	28	18	
200	700	118	221	119	97	Slowest microcomputer
400	2867	269	346	256	230	BASIC tested.

NOTE: Xerox timings were measured by the program from the system calendar/clock (resolution of 1 second). All other timings were taken manually with a digital stop watch (resolution of 1 second).

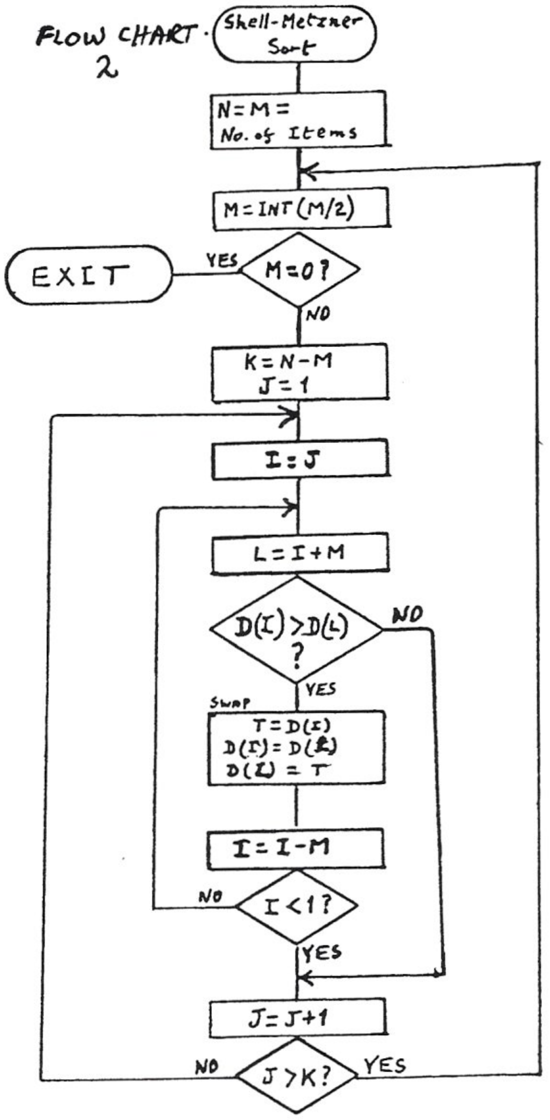
because the large numbers get moved to the left of the array before being shifted to their final positions on the right, but says that for large files it is nearly as fast as the Quicksort. The implementation by Geoffrey Chase which I tested confirms this for Processor Tech BASIC, where Heap is about 12% slower than Quick. For the other BASICs, the difference is 20-30%. The overhead is not much greater than for the Shell-metzner (22 BASIC statements). The only additional space re-



quired is sufficient to hold one record (or key) during comparisons/swaps. One big advantage of this method is that execution time is guaranteed to be of the order of N*log2(N), and the worst case time is not very much longer than the best case time.

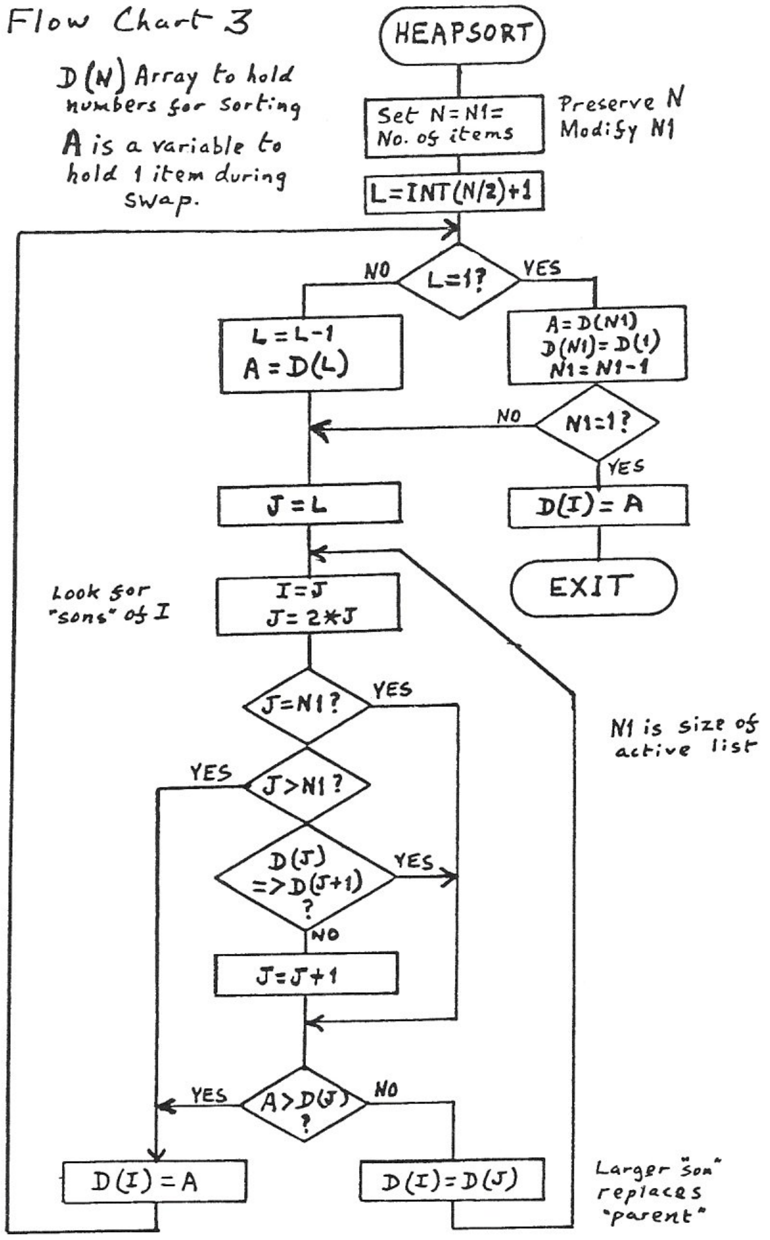
SHELL-METZNER SORT. See Flow Chart 2. This method, which requires only 16 BASIC statements to implement, is my favorite. Although there are five variables, the arithmetic is simple (no multiplication and only one divide-by-2). For a full explanation of the mechanism, refer to my article in Interface Age of November, 1978. The only extra space required is enough to hold one record (or key) during swaps. Here, too, execution time is guaranteed to be of the order of N*log2(N).

TREE SORT. The implementation which I tested is by Richard Hart, who modified the Woodrum sort for minimum number of comparisons and minimum number of steps between comparisons. The coding is complex, but execution goes like greased lightning in spite of a very large overhead (65 BASIC statements, with quite a few multiplications and divisions). There is an additional overhead in the form of an array to hold linked lists. This array must be large enough to hold N+log2(N) items, where N is the number of items to be sorted. This is larger than the file itself if the items to be sorted are integers; however, if the file contains records 100 bytes long the linked lists array becomes a much smaller proportion of the entire space needed. The method has the added advantage that the pointers in the linked list array avoid the need to move the records themselves. One possible disadvantage is that random access to a given item is not possible after sorting; you must start at the head of the linkage list and work downward until the desired item is found. One possible way around this would be to write the items, in sorted order, to a new file. If stored on a disc, the DOS could then give random access; if it must be resident in core, the sorted file could overwrite the original unsorted file, and a binary search could be used to find a given item.



Flow Chart 3

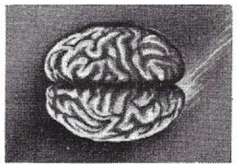
D(N) Array to hold numbers for sorting
A is a variable to hold 1 item during swap.



QUICKSORT. The implementation which I tested is by Steven Harrington. The overhead is moderate: 34 BASIC statements and an additional array to hold pointers to the beginning and end of segments of the main array that are to be individually sorted and then merged. This is a partitioning sort, which works on the premise that it is usually quicker (and never slower) to sort M lists of N/M elements each than to sort one list of N elements. Successive division of the list into smaller and smaller segments is not just a question of finding the center array position of the segment; for best performance, the "pivot point" should, rather, be the median value found in the segment. The accuracy with which the pivot point selected corresponds to the true median value is crucial to execution speed, especially in early phases. An enormous amount of work has been devoted to the search for the most efficient partitioning methods, culminating in the 1978 publication of an algorithm by Dobosiewicz which is reputed to run at least twice as fast as any previous version of quicksort, and involves a complex and elegant method of finding medians during early phases of the sort.

The Harrington version has no such sophistication; it merely picks the value at the center of the array as the first pivot point. Even so, it generally runs faster than any of the other methods tested.

The method used for partitioning affects not only the AVERAGE execution time, but also the worst-case time. KNUTH and HARRINGTON both caution that for pathological cases, execution time will be of the order of N**2, whereas for the average case the time is of the order of N*log2(N)*K, where K is linearly proportional to overhead operations; K for the Quicksort is often considerably smaller than the K for other methods. For the version tested, the pathological case is the concatenation of two nearly ordered lists. If randomization is introduced into the partitioning, then nearly ordered lists become the best case and completely random lists the worst case. The Dobosiewicz algorithm is aimed at optimizing partitioning for average cases in such a way that the average time factor approaches

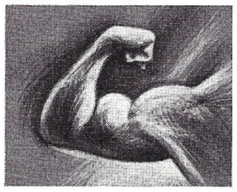


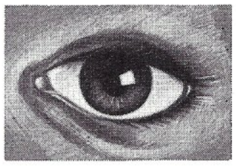
8086 Boards

CPU with $650.
Vectored Interrupts
PROM-I/O $495.
RAM $395.
8K x 16/16K x 8

ANALOG Boards

A/D 16 Channel, **$495.**
12 Bit, High Speed
D/A 4 Channel, **$395.**
12 Bit, High Speed





VIDEO DIGITIZATION

Real Time Video **$850.**
Digitizer and Display
Computer Portrait
System **$4950.**

S-100 Boards

Video and/or Analog Data Acquisition Microcomputer Systems

TECMAR INC.

The High Performance S-100 People

TECMAR, INC.
23414 Greenlawn • Cleveland, OH 44122
(216) 382-7599

```
90 *              XEROX BASIC CONVENTIONS
91 *     '*'='REM'
92 *     '&' concatenates multiple statements on a line
93 *     Array subscripts MUST start at 1 (not 0)
94 *     Multiple assignments are separated by a comma,
95 *          e.g., Y6=1,Y7=5  OR C,S9=0
96 *     IF...THEN must be followed by a line number
97 *       ':' indicates a print image for PRINTUSING
100 * SORTTEST -- Tests sorting algorithms
101 * by Chris Terry, 15 Feb 1979
102 ***************************************************
105 * ARRAYS: D hold list to be sorted
106 *         F saves unsorted list for re-use
107 *         B hold segment pointers for Quicksort
108 *         L holds N+log2(N)+2 linkages for Tree sort
110 DIM D(2500) & DIM F(2500)
115 DIM B(100) & DIM L(2050)
116 ************************************************
117 *        File size and Sort selection
120 PRINT 'HOW MANY NUMBERS'TAB(0)
130 INPUT N$ & X2=LEN(N$)
135 IF N$='' THEN 230
140 N=VAL(N$)
150 FOR X=1 TO N
160   X1=RND(0) & X3=INT(X1*10000)
170   D(X),F(X)=X3
180 NEXT X
190 PRINT 'BUBBLE (1), HEAP (2), SHELL-METZNER (3),'
195 PRINT 'TREE (4), OR QUICK (5)'TAB(0)
200 INPUT Z
210 Y1=TIM(1) & Y1=Y1*3600
220 ON Z GOTO 240,400,700,1000,2000
230 STOP
240 ****************** BUBBLE SORT *******************
245 C,S9=0
250 FOR A=1 TO N-1
260   FOR B=A+1 TO N
265     C=C+1
270     IF D(A)<D(B) THEN 300
275       S9=S9+1
280       T=D(A) & D(A)=D(B)
290       D(B)=T
300   NEXT B
310 NEXT A
320 GOTO 4000
400 ****************HEAP SORT ************************
402 *           Implementation by G. Chase
405 C,S9=0
410 N1=N
420 L=INT(N/2)+1
430 IF L=1 THEN 470
440 L=L-1
450 A=D(L)
460 GOTO 510
470 A=D(N1)
480 D(N1)=D(1)
490 N1=N1-1
500 IF N1=1 THEN 610
510 J=L
520 I=J
530 J=2*J
540 IF J=N1 THEN 580
550 IF J>N1 THEN 600
560 C=C+1 & IF D(J)=>D(J+1) THEN 580
570 J=J+1
580 C=C+1 & IF A>D(J) THEN 600
590 D(I)=D(J) & GOTO 520
600 D(I)=A & GOTO 430
610 D(I)=A
620 GOTO 4000
700 *************** SHELL-METZNER SORT ****************
702 *          Implementation by J. Grillo
705 C,S9=0
710 M=N
720 M=INT(M/2)
730 IF M=0 THEN 4000
740 K=N-M
750 J=1
760 I=J
770 L=I+M
775 C=C+1
780 IF D(I)<=D(L) THEN 810
785 T=D(I) & D(I)=D(L) & D(L)=T
786 S9=S9+1
790 I=I-M
800 IF I=>1 THEN 770
810 J=J+1
820 IF J>K THEN 720
830 GOTO 760
1000 ****************TREE SORT********************
1002 *          Implementation by R. Hart
1005 C,S9=0
1020 K1,I,M1,T2,T4=0
1030 J=N+1  & *HEAD OF SEQUENCE 1
1040 L(1+1),L(1+J),K2=1
1050 IF N<=1 THEN 1780& *NOTHING TO SORT -- EXIT
1060 S1=N  & *NUMBER OF LEAVES
1070 ******* Climb the tree *****
1080 IF S1<4 THEN 1140& *Low order twig value
1090 K2=K2*2 & *Total number of twigs
1100 B2=S1/2
1110 S1=INT(B2)
1120 T4=T4+(B2-S1)*K2
1130 GOTO 1070
1140 ********** Initial calculations ***********
1150 T4=K2-T4 & *Number of low-order twigs
1160 B2=K2/2 & *High bit value of binary counter
1170 *********** Next twig ***************
1180 IF K1=K2 THEN 1780& *SORT COMPLETE -- EXIT
1190 K1,T1=K1+1 & *Twig number
1200 B1=B2 & *High bit value
1210 T3=T2 & *Previous reflected twig number
1220 *********** Add 1 to reflected binary counter and carry **
1230 T1=T1/2
1240 IF INT(T1)<T1 THEN 1300& *No more carries
1250 M1=M1+1 & *Number of merges
```

```
1260 T2=T2-B1
1270 B1=B1/2 & *Next bit value
1280 GOTO 1220
1290 *(CARRY ONE)
1300 *************Twig calculations*************
1310 T2=T2+B1 & *Reflected twig number
1320 IF S1=2 THEN 1380& *2-twigs and 3-twigs
1330 *********** 3-twigs and 4-twigs *************
1340 IF T3<T4 THEN 1390& *Low-order twig (3-twig)
1350 ****** 4-twig **********
1360 M1=-M1 & *Disengage number of merges
1370 GOTO 1460
1380 IF T3<T4 THEN 1440& *Low-order twig (2-twig)
1390 *********** 3-twig ***********
1400 M1=M1+1 & * Number of merges
1410 I=I+1 & *Next Leaf
1420 L(1+I),L(1+J)=I & *Generate a leaf
1430 J=J+1 & *Next sequence head
1440 *********** 2-twig ***********
1450 M1=M1+1 & *Number of merges
1460 I=I+1 & *Next leaf
1470 L1,L(1+I),L(1+J)=I & *Generate a leaf
1480 L9=J & *Head of older leaf (last line)
1490 J=J+1 & *Head of lates leaf (next two lines)
1500 I=I+1 & *Next leaf
1510 L2,L(1+I),L(1+J)=I & *Generate a leaf
1520 GOTO 1590
1530 *(Merge leaves)
1540 ************** Merge twigs and branches ************
1550 J=J-1 & *Head of lates branch or twig
1560 L9=J-1 & *Head of older branch or twig
1570 L1=L(1+L9) & *Head of sequence 1
1580 L2=L(1+J) & *Head of sequence 2
1590 C=C+1 & IF D(L1)<=D(L2) THEN 1660 & *Stay in sequence 1
1600 L(1+L9)=L2 & * Switch to sequence 2
1610 L9=L2 & *Top leaf in sequence 2
1620 L2=L(1+L9) & *Next leaf in sequence 2
1630 IF L2=L9 THEN 1710& *End of sequence 2
1640 C=C+1 & IF D(L1)>D(L2) THEN 1610 & * Stay in sequence2
1650 L(1+L9)=L1 & *Switch to sequence 1
1660 L9=L1 & *Top leaf in sequence 1
1670 L1=L(1+L9) & *Next leaf in sequence 1
1680 IF L1<>L9 THEN 1590& *Not end of sequence 1
1690 L(1+L9)=L2 & *Switch to sequence 2)
1700 GOTO 1720
1710 L(1+L9)=L1 & *Switch to sequence 1
1720 M1=M1-1 & *Number of merges
1730 IF M1>0 THEN 1540
1740 IF M1=0 THEN 1170
1750 ******** Generate 2nd half of a 4-twig **********
1760 M1=1-M1 & *Re-engage number of merges
1770 GOTO 1460
1780 *********** EXIT *************
1790 GOTO 4000
2000 *********************************************************
2010 *                          QUICKSORT ROUTINE
2015 *            Implementation by S.Harrington
2020 *Initialize begin and end points to entire array
2030 L=1 & C,S=0
2040 B(L)=N+1
2050 M=1
2060 *Set end of array segment
2070 J=B(L)
2080 *Set start of array segment
2090 I=M-1
2100 *If only 2 or 3 elements, then handle specially
2110 IF (J-M)<3 THEN 2350
2120 M1=INT((I+J)/2)
2130 *Find a large element among the small ones
2140 I=I+1
2150 IF I=J THEN 2250
2160 IF D(I)<=D(M1) THEN 2140
2170 *Fine a small element among the large ones
2180 J=J-1
2190 IF I=J THEN 2250
2200 IF D(J)=>D(M1) THEN 2180
2210 *Swap elements
2220 T=D(I) & D(I)=D(J) & D(J)=T
2230 GOTO 2140
2240 *Array segment now divided; move compare element between
2250 IF I<M1 THEN 2270
2260 I=I-1
2270 IF J=M1 THEN 2300
2280 T=D(I) & D(I)=D(M1) & D(M1)=T
2290 *Save starting point for segment of large elements
2300 L=L+1
2310 B(L)=I
2320 *Repeat Quicksort on segment of small elements
2330 GOTO 2070
2340 *Special handling for 1- and 2-element cases
2350 IF (J-M)<2 THEN 2390
2360 IF D(M)<D(M+1) THEN 2390
2370 T=D(M) & D(M)=D(M+1) & D(M+1)=T
2380 *Set begin and end points for segment of large elements
2390 M=B(L)+1
2400 L=L-1
2410 IF L>0 THEN 2070
2420 *End of Sort; EXIT
2430 C='-' & S9='-' & GOTO 4000
2440 ******************************************************
4000 *END ROUTINE
4010 Y2=TIM(1) & Y2=Y2*3600
4020 Y3=Y2-Y1
4030 IF Y3=>1 THEN 4050
4040 Y3='<1'
4050 PRINT 'SORT TIME = 'Y3' SECONDS'
4060 IF C+S9<1 THEN4070
4065 PRINT 'COMPARISONS: 'C,'SWAPS: 'S9
4070 PRINT & PRINT
4080 PRINT 'WANT TO PRINT SORT RESULT'TAB(0)
4090 INPUT P$ & IF P$<>'Y' THEN 4110
4100 W=1 & GOSUB 4200
4110 PRINT 'WANT UNSORTED ARRAY'TAB(0)
4120 INPUT P$ & IF P$<>'Y' THEN 4140
4130 W=2 & GOSUB 4200
4140 PRINT 'RE-USE UNSORTED ARRAY'TAB(0)

4150 INPUT P$ & IF P$<>'Y' THEN 120
4155 *          Copy array F into Array D
4160 FOR X=1 TO N
4170   D(X)=F(X)
4180 NEXT X
4190 GOTO 190
4200 ***********************************************
4210 *    PRINT 1ST 100 ELEMENTS OF SORTED OR UNSORTED ARRAY
4220 L9=N+1
4230 FOR X=1 TO 100 STEP 10
4240 IF W=2 THEN 4280
4250 IF Z=4 THEN 4300
4260 PRINTUSING 4370,D(X),D(X+1),D(X+2),D(X+3),D(X+4),D(X+5),D(X+6),D(X+7),D(X+8),D(X+9)
4270 GOTO 4350
4280 PRINTUSING 4370,F(X),F(X+1),F(X+2),F(X+3),F(X+4),F(X+5),F(X+6),F(X+7),F(X+8),F(X+9)
4290 GOTO 4350
4295 *     Print Tree-sorted numbers
4300 FOR Y=1 TO 10
4310   L9=L(1+L9)
4320   PRINT D(L9);
4330 NEXT Y
4340 PRINT
4350 NEXT X
4360 RETURN
4370 :####  ####  ####  ####  ####  ####  ####  ####  ####  ####
4380 ************************************************
```

K*N (with a K between 3 and 6), and the worst-case time factor does not exceed K*N*log2(N).

From curiosity, I tried two runs of the Harrington version on a 1000-element sorted array in which I had manually disordered a few pairs of numbers. The sorting time in each run was no more than double the sorting time for a random array. Nevertheless, various authors have produced abundant evidence that under worst-case conditions Quicksort **can** run nearly as slowly as a bubble sort. If you find that it consistently runs slowly on the type of file that you most often sort, introduce or remove randomization in the manner suggested by Harrington.

THE TEST RUNS

To ensure portability, only DARTMOUTH BASIC statements were used. Where possible, the code of the original implementer was used without change; where translation from his dialect was necessary, it was done as straightforwardly as possible.

To provide comparison with a large machine, and for ease of debugging, the first runs were made on a Sigma 9 with a very powerful and efficient BASIC interpreter -- The Sigma 9 is the Xerox equivalent of the largest IBM System/370.

Table 2 summarizes the results of the timing tests I have run on various machines, for various file sizes. It is quite evident that most microcomputer BASIC interpreters are pretty slow, and that a machine-language sort routine would be advantageous for large files; my article in Interface Age describes a machine language Shell-Metzner sort routine that can handle strings or integers with sort keys in any position. Although the version published is limited to 255 items, I later modified it to sort up to 65K strings or integers on 8080/Z80 machines, and the documentation is adequate to allow adaptation to other machines.

All runs listed in Table 2 were performed on an array of random numbers generated by the test program.

The times shown are the Averages of several runs -- 10 runs per file size for each method in the case of the Xerox Sigma 9 machine, and 3 runs per file size for each method on all other machines. Times are shown in seconds.

REFERENCES:

CHASE, Geoffrey. Heapsort. **Creative Computing**, Nov/Dec 1977

DOBOSIEWICZ, Wlodzmierz. Sorting by Distributive Partitioning. **Information Processing Letters**, Vol. 7 No. 1, January 1978. (North-Holland Publishing Co., Netherlands).

GRILO, John P. A Comparison of Sorts. **Creative Computing** Nov/Dec 1977.

HARRINGTON, Steven. Quicksort! **Kilobaud / Microcomputing**, April, 1979.

HART, Richard. Tree Sort. **Creative Computing**, Jan/Feb 1978.

KNUTH, Donald E. The Art of Computer Programming, Volume 3: Sorting and Searching. Addison-Wesley, 1973.

LORIN, Harold. Sorting and Sort Systems. Addison-Wesley, 1975.

RERKO, Andrew J. Sorting Routines. **Kilobaud** No. 4, April 1977.

RICH, Robert. Internal Sorting Methods Illustrated with PL/1 Programs. Prentice-Hall, 1972.

TERRY, Chris. A Generalized 8080 String Sorting Routine. **Interface Age**, Nov 1978.

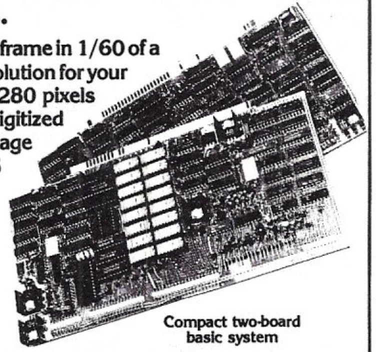
ACKNOWLEDGEMENTS

My sincere appreciation and thanks go to John Grillo, whose lively article in **Creative Computing** started me exploring a field that I have found completely fascinating; to Bob St. Hilaire, of Dun & Bradstreet, Inc., and Dave Zernoske of the New York Amateur Computer Club, who did the TRS-80 runs; and to Rick Auricchio (formerly of Dun & Bradstreet, now of Apple Computer Co.) who did the Apple runs.

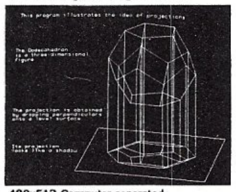
CAT-100 FULL COLOR GRAPHICS

The original 256-color imaging system with high resolution video FRAME GRABBER for the S-100 bus.

Capture and digitize a video frame in 1/60 of a second. Select the best resolution for your application, from 256 to 1280 pixels per TV line. Display your digitized or computer processed image with 256 gray levels or 256 colors on standard B&W, NTSC or RGB color TV monitors.


Compact two-board basic system

240x256 Digitized image, 16 levels


480x512 Computer-generated

Features:

- Highest possible quality 480x512x8 digital video image presently available on the market
- Input capability from TV camera or other sources
- Variety of synchronization choices
- 2 selectable video A/D conversion circuits
- Choice of 1, 2, 4, 8, 16 or 32 bits per pixel
- 32K-byte image memory on the basic system
- 32, 64, 128 & 256K byte system capacity
- Lightpen input
- Photographic trigger control input
- Software selectable system parameters
- Interfaces for TRS-80 and other processors
- Comprehensive line of accessories, monitors and support software

SEND FOR FREE CATALOG

DIGITAL GRAPHIC SYSTEMS
441 California Ave., Palo Alto, CA 94306 415/494-6088

NO MORE WAITING FOR SORTS

by
Robert L. Sheffield
4505 Apache Road
Boulder, CO 80303

You have a list of names to put in alphabetical order. So you sit down at your favorite computer and load your bubble sort routine (written in BASIC). You enter the names you want alphabetized - about a hundred, say. You enter the sort command. And you go away and read the newspaper while you computer sits there grinding away on your list of names. In 10 to 20 minutes you may have your list in RAM ready to print.

YOU DON'T HAVE TO TAKE IT ANY MORE!

My First Sort Routine

My wife Janet, who is into businesses, clubs, and children's activities, always has lists of people's names she wants to put in alphabetical order. Soon after I got my Poly 88 up and running reliably enough to complete a losing same of Star Trek, Janet had names of about 100 of her summer swim club members scattered in random order over several lists. Could my computer (she wanted to know) put her names in alphabetical order? My son Bob (who knew BASIC) and I (who knew the alphabet) sat down with Poly and, going by a magazine article on sorting algorithms wrote a bubble sort routine along with the other programming necessary to let Janet enter her names and print out the alphabetized list.

Sure (I said) my computer could sort her list. She entered her names and entered the command to sort. Janet doesn't claim to be patient. It only took her a couple of minutes to wonder where her alphabetical list was. I consider myself to be very patient, but after a few more minutes I was convinced Poly had somehow failed. When I had tested the sorter with only a dozen or so entries, Poly had finished immediately. To see what was going on this time, I stopped the program and looked at the sort area. It was sorting. We let it finish. It took more than 15 minutes. When we ran it again without stopping, it took just under 15 minutes. Janet was happy enough. Knowing the sort would take a while, she could start it and go do something else while it ran.

I figure there had to be a better way. There is. Trees. Binary trees.

My Last Sort Routine

Rejoice! With the program on these pages, you may never have to wait for a sort again. There isn't even a sort command. You simply enter the things you want alphabetized; as soon as you are through entering them, list them - immediately - in alphabetical order!

This sample program is actually a utility program that Janet now uses to create and maintain many of her ordered lists. Look at the menu on lines 1200 through 1240. She may enter the things she wants ordered (1) and delete them (4). She may list them, in ASCII code order, on the CRT screen (2) or on the printer (3). And she can save it away for another day - list, program, BASIC, and all - to tape (5). (I didn't have disks yet when I wrote this program).

This particular version takes a lot of RAM. I allow for 200 63-character entries requiring 12600 bytes for the list itself. There are also 3 directory entries created by the program for each of the 200 data entries. In my 8-digit precision BASIC, each directory entry requires 5 bytes; therefore the directory requires 3000 bytes. All the other variables add another 500 bytes. The source code as you see it here takes 9100 bytes, but densely packed it only takes 2800 bytes. The total RAM required then is 19000 bytes plus any required by your BASIC and your save and print routines. You can cut the whole thing down the size and number of data entries allowed. For example, 100 20-character entries would take 12300 less bytes in my BASIC than the 200 63-character version.

Binary Trees - Planting, Growing, and Climbing

This program is not really a sort program because it does not sort anything. It just stores the user's entries in the order he enters them, but as they come in it makes a directory in the form of a binary tree. When it lists the entries back at the terminal or on the printer, it uses the directory to determine the order.

For example, look at Figure 1. Assume the list of letters down the side - MNO, DEF, TUV, and so forth - are the entries to be alphabetized and that they are entered in the order shown. The columns show what the directory for each entry looks like after each of the subsequent entries is entered. The 3 digits in each column represent 3 indexes into the list of entries. The indexes are 0 (for MNO) through 7 (for JKL). The first digit in each directory element is the back pointer - that is, the index of the entry that this entry is attached to. The second digit is the low pointer - the index of the next entry made which was lower alphabetically than this entry. The third digit is the high pointer - the index of the next entry made which was higher. Note that the indexes start with 0 like BASIC counts.

Figure 2 shows graphically the logical organization of the directory - the binary tree. It is called a binary tree because there can be two branches from each node. Each line which connects two boxes represents a low or high pointer and a back pointer. The line between DEF and GHI, for example, is DEF's high pointer and GHI's back pointer. When you ask for a list, the program first goes down the leftmost legs until it gets to the end (ABC) and lists it. It then starts looking for a right leg, listing each node as it works its way back up. In this case, it finds a right leg after it lists DEF. It follows the right leg only to the next node (GHI). It would next go all the way down the leftmost legs from this node, but their aren't any, so it lists the node and checks for a right leg. And so on.

Anyway, you don't have to wait for a sort. The program makes the directory so fast, as you make the entries, you don't know it's happening. When you ask for a list, it starts listing immediately. On the tube, the list comes out slower than just a straight list would, but even with only three characters per entry, it comes out faster than you can read it. On my printer, it comes out just as fast as any other listing.

You may notice some slowing as you enter data under certain circumstances. Janet was entering a list the other day and noticed she was able to enter several characters of an entry before the characters began to appear on the screen. It so happened that the lists she was entering from were already almost completely alphabetized. Now the response time from the program as it is taking entries increases as the depth of the tree increases. While it can branch at each level and thereby avoid comparing the new entry with most of the old entries, it must compare the new entry with exactly one entry at each level until it finds a place to attach the new entry. The worst thing you can do from the standpoint of response time is to enter your list either in alphabetical order or in the reverse of alphabetical order. The effect would be to create a very tall tree with no branches.

Planting the Tree

The data areas required to define and manage the list and the directory are set up in lines 1040 through 1130 of the sample program. Line 1040 establishes the space for the list - 200 entries of 63 characters each. Line 111 is the directory - 200 entries (starting with 0) with 3 pointers each (back pointer, low pointer, and high pointer). I1 gives the index for the next entry coming in. It is easier to save it each time than to calculate it. I2 is used to prevent overrunning the end of the index. Lines 1070 through 1100 set up the input record and provide for padding it with blanks. Lines 1120 and 1130 provide an easy way to delete an entry. The program does not really delete an entry; it just marks it deleted and then skips those marked deleted when listing the entries in alphabetical order.

```
          EXAMPLE OF DIRECTORY AFTER EACH ENTRY

        ---------INDEXES RELATED TO EACH ENTRY--------

           : MNO : DEF : TUV : GHI : QRS : ABC : XYZ : JKL :
           : -0- : -1- : -2- : -3- : -4- : -5- : -6- : -7- :
           :     :     :     :     :     :     :     :     :
E   MNO    :0 0 0:     :     :     :     :     :     :     :
N   DEF    :0 1 0:0 0 0:     :     :     :     :     :     :
T   TUV    :0 1 2:0 0 0:0 0 0:     :     :     :     :     :
R   GHI    :0 1 2:0 0 3:0 0 0:1 0 0:     :     :     :     :
I   QRS    :0 1 2:0 0 3:0 4 0:1 0 0:2 0 0:     :     :     :
E   ABC    :0 1 2:0 5 3:0 4 0:1 0 0:2 0 0:1 0 0:     :     :
S   XYZ    :0 1 2:0 5 3:0 4 6:1 0 0:2 0 0:1 0 0:2 0 0:     :
    JKL    :0 1 2:0 5 3:0 4 6:1 0 7:2 0 0:1 0 0:2 0 0:3 0 0:
```

Fisure 1

Growing the Tree

Lines 1450 through 1790 grow the tree. Lines 1450 through 1480 create the root when the user enters his first entry. Line 1460 puts the entry in the list. Line 1470 sets the next entry to be at index 1. The first entry, of course, is at index 0, the way BASIC counts. In figures 1 and 2, the first entry is the entry MNO. The indexes relating to the MNO entry are all zeroes because, being the "root" entry, it has no back pointer to a previous entry and no other entries are yet attached to it.

Lines 1490 through 1510 compare each new entry with existing entries. Line 1490 causes the comparisons to start with the first, or "root" entry. E1 (not the same variable as dimensioned variable E1) will contain the starting position of the next entry in the list to be compared with the new entry less one. (BASIC starts counting with one when it is counting characters in a string.) Lines 1500 and 1510 do the comparison and take the appropriate branch when the new entry is either higher or lower than the old one it is being compared with.

If the new entry is equal to the old entry, the program drops through to lines 1520 through 1540. These lines just turn off the deleted flag for the entry. This has the effect of reinstating the entry if it had been previously deleted. If it had not been deleted, the deleted flag would already be off and the effect of these lines would be to simply leave the list and the index as is.

Lines 1580 through 1630 and lines 1670 through 1720 handle the situations where the new entry is lower or higher, respectively, than the old entry. Take the DEF entry in the example. Line 1500 compares DEF with MNO, the root, and branches to line 1560 because DEF is less than MNO. Line 1580 calculates the index of the old entry just compared - 0 since it is the root and since the program just set E1 to 0 at line 1490. Line 1590 checks to see if the low pointer of

MNO is 0 indicating that no entry lower than MNO has as yet been entered. That is the case in this instance, so the program branches to line 1620. Line 1620 puts the index of the new entry from I1 - in this case, 1 - into MNO's low pointer. Look at Figure 1. MNO's low pointer (in the MNO column) after the DEF entry (in the DEF row) is now a 1 indicating that there is now at least one entry in the list lower than MNO and that one of them is the one represented by index 1.

Line 1630 branches to the common routine - lines 1760 through 1790 - for adding the new entry to the list. Line 1760 sets the new entry's back pointer - in this case 0 since it is attached to the root. Line 1770 tacks the new entry onto the end of the list. Line 1780 indicates the index of the next new entry. Line 1790 goes to set the next entry.

The entry of TUV works the same way except it goes through line 1670 through 1740 instead of lines 1580 through 1630. In Figure 1, after TUV has been entered, MNO's index has a high pointer of 2 indicating that there is now at least one entry in the list which is higher than MNO and that one of them is represented by the index at 2.

Now let's see what happens when GHI is added. Line 1490 starts the comparisons at the root, MNO. Line 1500 finds GHI lower than MNO and branches to 1560. Line 1590 this time finds that MNO's low pointer is not 0 - that there is already an entry lower than MNO in the list - and falls through to line 1600. Line 1600 calculates where this entry that is lower than MNO is in the entry list and line 1610 branches to line 1500 for another compare. Line 1500 compares GHI with the entry lower than MNO which we know is DEF. Since GHI is not lower than DEF, the program falls through to line 1510. Line 1510 branches to line 1650 since GHI is greater than DEF. Line 1680 finds that DEF's high pointer is empty and goes to line 1710 where GHI's index is put in DEF's high pointer and then to 1740 where GHI is added to the list. In Figure 1, row GHI,

```
GRAPHICAL REPRESENTATION OF THE DIRECTORY

              A BINARY TREE

                 +---+
                 :MNO:
                 ++-++
                  : :
          +--------+ +--------+
          :                   :
        +-+-+               +-+-+
        :DEF:               :TUV:
        ++-++               ++-++
         : :                 : :
     +---+ +---+         +---+ +---+
     :         :         :         :
   +-+-+     +-+-+     +-+-+     +-+-+
   :ABC:     :GHI:     :QRS:     :XYZ:
   +---+     +--++     +---+     +---+
                :
              +-+-+
              :JKL:
              +---+
```

Figure 2

column DEF, see that DEF's high pointer now shows 3 which is GHI's index. In column GHI, see that GHI's back pointer is 1 which is DEF's index.

Climbing the Tree

Lines 2820 through 3150 find the entries in ASCII code order and list them at the console or on the printer. There are three rules for finding the nodes in the binary tree in Figure 2 in the proper order:

1. If we came down from above, find the next node to the left, if any.
2. If we came up from the left or there is no left pointer, find the next node to the right, if any.
3. If we came up from the right or there is no right pointer, go up unless we are at the root, in which case quit.

There are two rules for determining whether it is time to print a node:

1. If we came down from above and there is no left pointer, print.
2. If we came up from the left, print.

The variable L1$ on line 2820 records where we just came from. The variable E on line 2840 is the index of the node we are at in the tree. To start the climb, L1$ is set to "A" indicating we are coming from above, and E is set to 0 starting us at the root. Lines 2850 through 2880 take us from the root down to ABC following search rule 1 above. Neither MNO nor DEF were printed since at the time we passed them the circumstances matched neither of the two print rules. At both MNO and DEF there was a left pointer failing print rule 1, and at both we were coming down, not up from the left as required by print rule 2. But at ABC we have just come down from above and there is no left pointer, so we should print it. Line 2850 finds that ABC has no left pointer and branches to line 2890. Line 2890 finds that we did not come up from the right. Since there was no left pointer we could not have come up from the left. That leaves that we came from above so we fall through to line 2920. Lines 2920 through 3050 print entries not marked deleted. Since there is no left pointer, search rule 2 says look for a node on the right. Lines 3060 through 3090 would find a node on the right, but in this case there is none, so line 3060 branches us to line 3100. If we were at the root node, line 3100 would end the search in accordance with search rule 3, since there is no right pointer. But since we are not at the root, rule 3 and lines 3110 through 3140 back us up to DEF after setting L1$ to indicate we are coming up from the left. This time line 2860 finds we did not come from above and sends us to line 2890 which finds we did not come from the right either - leaving that we must have come from the left - so we print DEF in accordance with print rule 2. After the print, line 3060 finds that there is a right pointer from DEF and lines 3070 and 3080 find it and flag that we are coming from above. At GHI line 2850 finds no left pointer and line 2890 finds that we did not come from the right leaving that we must have come from above, and in accordance with print rule 1 we print GHI. Lines 3060 through 3090 find JKL in accordance with search rule 2 and lines 2850 and 2890 cause it to print in accordance with print rule 1. Lines 3110 through 3150 work us all the way back up to MNO in accordance with search rule 3. Notice that when we arrive at MNO, L1$ = L causing MNO to print. Lines 3060 through 3090 send us down the right leg from MNO in accordance with search rule 2. The same things happen on the right branch from MNO as happened on the left except that when we get back to MNO this time, L1$ = R. When line 2890 sees the R it branches to line 3100 which finds we are at the root and quits the search in accordance with search rule 3.

For those who understand decision tables, I offer Figure 3 without comment to help clarify the tree-climbing process.

```
        TREE CLIMBING DECISION TABLE

 CONDITIONS                     RULES
-------------      -------------------------------
From above         Y Y Y Y Y Y Y N N N N N N N N Y
Left pointer       Y N N N N N N - - - - - - - - -
From right         N N N N N N N N N N N N N Y Y Y
Deleted            - Y N Y N Y N Y N Y N Y N - - -
Right pointer      - Y Y N N N N Y Y N N N N - - -
Top node           - - - Y Y N N - - Y Y N N Y N -

   ACTIONS
-------------
Go left            X - - - - - - - - - - - - - - -
Print              - - X - X - X - X - X - X - - -
Go right           - X X - - - - X X - - - - - - -
Quit               - - - X X - - - - X X - - X - -
Go back            - - - - - X X - - - - X X - X -
Impossible         - - - - - - - - - - - - - - - X

                   Figure 3
```

Variations

There are many ways to improve on the sample program. Most of the ones I was aware of when I started this project would severely complicate the basic message of the article - the way to a fast alphabetical listing. Furthermore, this project would become one of those never-ending ones were I to follow where each idea or misgiving leads me. I will, however, discuss briefly some of the more useful variations and extensions that have occured to me as the project has developed. Be warned that I have not tested any of these ideas. You may find them useful. You also may find they will not work.

Pruning the Tree

As indicated earlier, the program does not really delete entries. When the routine at lines 2140 through 2420 determines what the user wants to delete, it simply puts a "D" in the string D1$ (defined in line 1120) at the point corresponding to the position in the main list of the entry to be deleted. When line 2930 in the list output routine finds a "D" corresponding to an entry, it skips listing that entry. This, of course, means that you may find your list space used up even though you have less than the allowable number of entries active. A way to reclaim the space is to logically remove the index of the deleted element and make that index available for the next entry to be added. Suppose, for example, we wanted to remove DEF from the structure shown in Figure 2. We could do this by changing MNO's low pointer to point to either ABC or GHI. Let's choose GHI, for example. We complete rechaining by pointing GHI's low pointer to ABC. If there had already been something attached to GHI's low pointer we would have traced down GHI's low pointer until we found an empty one. This effectively removes DEF from the list, but it does not make its space available for the next entry. To do this would require complicating the tree-growing routine somewhat. First we would probably initialize the entire list with blanks and insert each new entry instead of just tacking on each new entry as the sample program does. We would initialize the index list so that each set of indexes pointed to the next set; for example, in any as yet unused index, one of its pointers, say the back pointer, could be used to point to the next index. I1, the index of the next available entry, would initially point to the start of the list. Each time an entry is added, I1 would be updated with the back pointer of the index used for the new entry. Now, when we delete an entry, the value of I1 is put in the back pointer of the index of the deleted entry and the index of the deleted entry is put in I1.

The result is a chain of available entries starting with the most recently deleted entry, passing through all previously deleted entries in reverse order of their deletion, to the first never-used space in the list, and finally through to the end of the list. We would also need a root pointer to provide for deleting the root entry. The sample program assumes that the first entry is always the root entry.

Multiple Lists with One Index

One binary tree directory can be used with any number of lists. You can, for example, have a list of names, a list of street addresses, and a list of cities, states, and zip codes using one directory to tie them all together. You then print or display any of or combination of the lists ordered according to the index.

Multiple Indexes

Several binary tree directories can index the same set of several lists allowing for ordering in as many different ways as there are directories. Continuing the example of the address list, you could have two indexes, one by name and one by city and provide for listing them in either order.

```
1000REM - SORTER USING BINARY TREE DIRECTORY
1010REM
1020REM - ENTRY LIST
1030REM
1040 DIM E1$ (12600)           \REM - LIST OF ENTRIES AS ENTERED
1050 I1 = 0                    \REM - INDEX OF NEXT ENTRY
1060 I2 = 199                  \REM - MAX NUMBER OF ENTRIES
1070 DIM E2$ (63)              \REM - ENTRY FROM TERMINAL
1080 DIM E3$ (63)              \REM - BLANKS TO FILL OUT E2$
1090 FOR I = 1 TO 7\E3$ = E3$ + "         "\NEXT
1100 E3 = LEN (E3$)            \REM - CONSTANT LEN OF 1 ENTRY
1110 DIM E1 (199,2)            \REM - DIRECTORY OF ENTRIES
1120 DIM D1$ (200)             \REM - "DELETED" FLAGS FOR EA ENT
1130 FOR I = 1 TO 10\D1$ = D1$ + "                    "\NEXT
1140REM
1150REM - PRIMARY MENU OF FUNCTIONS
1160REM
1170 PRINT CHR$(12)            \REM - CLEAR CRT SCREEN
1180 GOSUB 3220\PRINT          \REM - PRINT TAPE FILE NAME
1190 PRINT "SELECT:"           \REM - OFFER SELECTION
1200 PRINT " 1 ENTER DATA"
1210 PRINT " 2 DISPLAY ENTRIES ON SCREEN IN ALPHABETICAL ORDER"
1220 PRINT " 3 PRINT ENTRIES IN ALPHABETICAL ORDER"
1230 PRINT " 4 DELETE ENTRIES"
1240 PRINT " 5 SAVE TO TAPE"
1250 INPUT " ->",A$            \REM - GET FUNCTION SELECTION
1260 IF LEN(A$)<>1 THEN 1300   \REM - LENGTH NOT 1 IS ERROR
1270 IF A$ < "1" THEN 1300     \REM - SELECT LESS THAN 1 IS ERROR
1280 IF A$ > "5" THEN 1300     \REM - SELECT MORE THAN 5 IS ERROR
1290 ON VAL(A$) GOTO 1340,1810,1940,2120,2440
1300 PRINT A$," NOT VALID"     \REM - SEND ERROR MESSAGE
1310 GOSUB 2760                \REM - HOLD SCREEN TO SHOW MESSAGE
1320 GOTO 1150                 \REM - REDISPLAY MENU
1330REM
1340REM - ENTER DATA TO BE SORTED
1350REM
1360 PRINT CHR$(12)           \REM - CLEAR CRT SCREEN
1370 PRINT "ENTER ONE LINE (63 CHARACTERS) PER ENTRY"
1380 IF I1 < I2+1 THEN 1420   \REM - IF OUT OF SPACE...
1390 PRINT "OUT OF SPACE"     \REM - ...SEND MESSAGE,...
1400 GOSUB 2760               \REM - ...HOLD SCREEN,...
1410 GOTO 1150                \REM - ...AND REDISPLAY MENU
1420 INPUT"",E2$              \REM - GET ENTRY
1430 IF E2$ = "" THEN 1150    \REM - NULL ENTRY MEANS DONE
1440 E2$ = E2$ + E3$          \REM - PAD RIGHT WITH BLANKS
1450 IF I1 <> 0 THEN 1490     \REM - IF THIS IS FIRST ENTRY...
1460 E1$ = E2$                \REM - ...JUST PUT IT IN LIST,...
1470 I1 = 1                   \REM - ...BUMP INDEX,...
1480 GOTO 1380                \REM - ...AND GET NEXT ENTRY
1490 E1 = 0                   \REM - INDEX OF NEXT TO COMPARE
1500 IF E2$ < E1$(E1+1,E1+E3) THEN 1560
1510 IF E2$ > E1$(E1+1,E1+E3) THEN 1650
1520 E = (E1/E3) + 1          \REM - NEW SAME; JUST REINSTATE...
1530 D1$(E,E) = " "           \REM - ...IT IF IT WAS DELETED,...
1540 GOTO 1380                \REM - ...AND GET USER'S NEXT ENTRY

1550REM
1560REM - NEW ENTRY LESS THAN PREVIOUS
1570REM
1580 E = E1/E3                \REM - INDEX OF COMPARED ENTRY
1590 IF E1(E,1)=0 THEN 1620   \REM - IF STILL LOWER ENTRY,...
1600 E1 = E1(E,1)*E3          \REM - ...FIND START IN LIST...
1610 GOTO 1500                \REM - ...AND COMPARE IT WITH NEW
1620 E1(E,1) = I1             \REM - ELSE, POINT TO NEW ENTRY...
1630 GOTO 1740                \REM - ...AND PUT IT IN LIST
1640REM
1650REM - NEW ENTRY GREATER THAN PREVIOUS
1660REM
1670 E=E1/E3                  \REM - INDEX OF COMPARED ENTRY
1680 IF E1(E,2)=0 THEN 1710   \REM - IF STILL HIGHER ENTRY,...
1690 E1 = E1(E,2) * E3        \REM - ...FIND START IN LIST...
1700 GOTO 1500                \REM - ...AND COMPARE IT WITH NEW
1710 E1(E,2) = I1             \REM - ELSE, POINT TO NEW ENTRY...
1720 GOTO 1740                \REM - ...AND PUT IT IN LIST
1730REM
1740REM - PUT NEW ENTRY IN LIST
1750REM
1760 E1(I1,0) = E             \REM - POINT TO PREVIOUS ENTRY
1770 E1$ = E1$ + E2$          \REM - PUT NEW ENTRY IN LIST
1780 I1 = I1 + 1              \REM - BUMP INDEX OF NEXT ENTRY
1790 GOTO 1380                \REM - GET NEXT ENTRY FROM USER
1800REM
1810REM - DISPLAY ENTRIES ON SCREEN IN ALPHABETICAL ORDER
1820REM
1830 IF I1 <> 0 THEN 1870     \REM - IF NOTHING IN LIST...
1840 PRINT "NO ENTRIES TO DISPLAY"\REM - ...SEND MESSAGE,...
1850 GOSUB 2760               \REM - ...HOLD SCREEN,...
1860 GOTO 1150                \REM - ...AND REDISPLAY MENU
1870 PRINT CHR$(12),          \REM - ELSE, CLEAR SCREEN,...
1880 O1$ = "D"                \REM - ...FLAG DISPLAYING,...
1890 GOSUB 2800               \REM - ...DISPLAY LIST,...
1900 PRINT "END OF LIST; ",   \REM - ...TELL USER WE ARE DONE,..
1910 GOSUB 2760               \REM - ...HOLD SCREEN,...
1920 GOTO 1150                \REM - ...AND REDISPLAY MENU
1930REM
1940REM - PRINT ENTRIES IN ALPHABETICAL ORDER
1950REM
1960IFI1=0THEN!"NO ENTRIES TO PRINT"\GOSUB2760\GOTO1150
1970 IF I1 <> 0 THEN 2010     \REM - IF NO ENTRIES IN LIST,...
1980 PRINT "NO ENTRIES TO PRINT"\REM - ...TELL USER,...
1990 GOSUB 2760               \REM - ...HOLD SCREEN,...
2000 GOTO 1150                \REM - ...AND REDISPLAY MENU
2010 PRINT CHR$(12),          \REM - ELSE, CLEAR SCREEN,...
2020 PRINT "TURN PRINTER ON AND ALIGN PAPER TO TOP OF PAGE"
2030 GOSUB 3170               \REM - ...WAIT TILL USER'S READY,.
2040 IF R1$<>"R" THEN 1150    \REM - ...(QUIT IF HE WANTS TO),..
2050 O1$ = "P"                \REM - ...FLAG PRINTING,...
2060 PRINT CHR$(17),          \REM - ...START PRINTER,...
2070 GOSUB 2800               \REM - ...PRINT LIST,...
2080 PRINT CHR$(12),          \REM - ...EJECT LAST PAGE,...
2090 PRINT CHR$(19)           \REM - ...STOP PRINTER,...
2100 GOTO 1150                \REM - ...AND REDISPLAY MENU"
2110REM
2120REM - DELETE ENTRIES
2130REM
2140 IF I1 <> 0 THEN 2180     \REM - IF NO ENTRIES,...
2150 PRINT "NO ENTRIES TO DELETE"\REM - ...TELL USER,...
2160 GOSUB 2760               \REM - ...HOLD SCREEN,...
2170 GOTO 1150                \REM - ...AND REDISPLAY MENU
2180 PRINT CHR$(12),          \REM - ELSE, CLEAR SCREEN
2190 PRINT "ENTER UNIQUE CHARACTER STRING FROM ENTRY TO DELETE"
2200 INPUT " ->",E2$          \REM - GET STRING IN ENTRY TO DELETE
2210 IF E2$ = "" THEN 1150    \REM - QUIT IF USER WANTS TO
2220 PRINT "SEARCHING FOR ENTRY TO DELETE..."
2230 I0 = LEN (E2$) - 1       \REM - GET END OF STG TO COMPARE
2240 FOR I = 1 TO LEN (E1$) - I0\REM - SEARCH LIST FOR MATCH
2250 IF E2$ = E1$(I,I+I0) THEN EXIT 2280\REM - MATCH FOUND
2260 NEXT\PRINT "NOT FOUND"   \REM - IF NOT FOUND, SEND MESSAGE...
2270 GOTO 2190                \REM - ...AND GET NEXT STRING
2280 I = INT(I/E3)            \REM - GET INDEX OF MATCHED ENTRY
2290 I0 = (I * E3) + 1        \REM - FIND START OF MATCHED ENTRY
2300 PRINT E1$(I0,I0+E3-1)    \REM - DISPLAY ENTRY TO DELETE
2310 INPUT "ENTER 'D' TO DELETE OR RETURN TO LEAVE AS IS ->",A$
2320 IF A$ <> "" THEN 2380    \REM - IF NOT TO DELETE,...
2330 I = I + 1                \REM - ...(GET INDEX INTO FLAGS),...
2340 IF D1$(I,I) <>"D" THEN 2190\REM - ...IF ALREADY DELETED,...
2350 D1$(I,I) = " "           \REM - ...RESTORE ENTRY,...
2360 PRINT "RESTORED"         \REM - ...TELL USER,...
2370 GOTO 2190                \REM - ...AND GET NEXT STRING
2380 IF A$ <> "D" THEN 2190   \REM - IF NOT RECOGNIZED, ASK AGAIN
2390 I = I + 1                \REM - ELSE (DELETE), INDEX FLAGS
2400 D1$(I,I) = "D"           \REM - DELETE ENTRY
2410 PRINT "DELETED"          \REM - TELL USER
2420 GOTO 2190                \REM - GET NEXT STRING
2430REM
2440REM - SAVE TO TAPE
2450REM
2460 PRINT CHR$(12),          \REM - CLEAR SCREEN
2470 PRINT "NAME OF FILE TO SAVE IS ",
2480 GOSUB 3220\PRINT         \REM - NAME OF LAST FILE LOADED
2490 PRINT "IF NAME OK, ENTER SPACE; ",
2500 PRINT "TO CHANGE, ENTER NEW NAME; "
2510 PRINT "TO QUIT, RETURN",
2520 INPUT " ->",A$           \REM - GET USER'S CHOICE
2530 IF A$ = " " THEN 2600    \REM - IF NEW FILE NAME,...
2540 A$ = A$ + "        "     \REM - ...PAD NAME WITH BLANKS,.
2550 FOR I = 1 TO 8           \REM - ...LOAD NEW NAME IN...
2560 POKE 4795+I,ASC(A$(I,I))\REM - ...TAPE FILE HEADER,...
2570 NEXT                     \REM - ...AND...
2580 PRINT "SAVING ",         \REM - ...CONFIRM NEW...
2590 GOSUB 3220\PRINT         \REM - ...NAME
2600 PRINT "TURN TAPE ON; ",\REM - LET USER TURN TAPE ON
2610 GOSUB 3170               \REM - WAIT FOR READY SIGNAL
2620 IF R1$="R" THEN 2660     \REM - IF USER NOT READY,...
2630 PRINT "NOT SAVED"        \REM - ...WARN HIM,...
2640 GOSUB 2760               \REM - ...HOLD SCREEN,...
2650 GOTO 1150                \REM - ...AND REDISPLAY MENU
2660 N=CALL(4352,61440-FREE(0))\REM - PUT IT ALL ON TAPE
2670 IF N <> 1 THEN 2700      \REM - IF FILE JUST DUMPED,...
2680 GOSUB 3220               \REM - ...TELL USER WE...
2690 PRINT " SAVED"           \REM - ...SAVED IT
2700 IF N <> 2 THEN 2730      \REM - IF FILE JUST LOADED,...
2710 GOSUB 3220               \REM - ...TELL USER WE...
2720 PRINT " LOADED"          \REM - ...LOADED IT
2730 GOSUB 2760               \REM - HOLD SCREEN
2740 GOTO 1150                \REM - REDISPLAY MENU
2750REM
2760REM - HOLD SCREEN UNTIL USER GIVES CONTINUE SIGNAL
2770REM
2780 INPUT "RETURN TO CONTINUE ",X1$\RETURN

2790REM
2800REM - DISPLAY OR PRINT LIST IN ASCII ORDER
2810REM
2820 L1$ = "A"                \REM - FLAG COMING FROM ABOVE
2830 L = 1                    \REM - SET LINE COUNTER
2840 E = 0                    \REM - INDEX OF FIRST ENTRY
2850 IF E1(E,1)=0 THEN 2890   \REM - IF THERE'S A LOWER ENTRY...
2860 IF L1$<>"A" THEN 2890    \REM - ...AND WE CAME FROM ABOVE,.
2870 E = E1(E,1)              \REM - ...FIND THE...
2880 GOTO 2850                \REM - ...LOWER ENTRY
2890 IF L1$="R" THEN 3100     \REM - IF FROM RIGHT, DON'T PRINT
2920 E0 = E + 1               \REM - INDEX INTO DELETE TABLE
2930 IF D1$(E0,E0) = "D" THEN 3060
2940                           REM - IF ENTRY NOT DELETED,...
2950 E0 = E * E3              \REM - ...FIND ENTRY TO SEND,...
2960 PRINT E1$(E0+1,E0+E3)    \REM - ...SEND IT,...
2970 L = L + 1                \REM - ...COUNT LINES SENT,...
2980 IF O1$<>"D" THEN 3020    \REM - ...IF DISPLAYING...
2990 IF L < 16 THEN 3020      \REM - ...AND SCREEN FULL,...
3000 GOSUB 2760               \REM - ...HOLD SCREEN...
3010 GOTO 3040                \REM - ...AND RESET PAGE, OR...
3020 IF O1$<>"P" THEN 3060    \REM - ...IF PRINTING...
3030 IF L < 55 THEN 3060      \REM - ...AND PAGE FULL,...
3040 L = 1                    \REM - ...RESET LINE COUNTER AND...
3050 PRINT CHR$(12)           \REM - ...GET NEXT SCREEN OR PAGE
3060 IF E1(E,2)=0 THEN 3100   \REM - IF THERE IS A RIGHT LEG,...
3070 E = E1(E,2)              \REM - ...FIND IT,...
3080 L1$ = "A"                \REM - ...FLAG FROM ABOVE,...
3090 GOTO 2850                \REM - ...AND CHECK NEXT NODE
3100 IF E = 0 THEN RETURN     \REM - IF AT TOP NODE, QUIT
3110 IF E=E1(E1(E,0),2) THEN L1$="R" ELSE L1$="L"
3120                           REM - ELSE, FLAG IF WE ARE ON...
3130                           REM - ...A RIGHT OR LEFT LEG,...
3140 E = E1(E,0)              \REM - ...GET PARENT NODE,...
3150 GOTO 2850                \REM - ...AND CHECK IT OUT
3160REM
3170REM - WAIT FOR 'READY' OR 'QUIT' SIGNAL
3180REM
3190 INPUT "TYPE 'R' WHEN READY OR RETURN TO QUIT ->",R1$
3200 RETURN
3210REM
3220REM - DISPLAY CURRENT TAPE FILE NAME
3230REM
3240 FOR I = 1 TO 8
3250 PRINT CHR$(PEEK(4795+I)),
3260 NEXT
3270 RETURN
```

Lists on Disks

Those of you who have direct access mass storage systems can build a binary tree directory to order a list of record identifiers. This would allow you to order records stored on your direct access device. You could do "instant sorts" on large quantities of data.

Recursive Languages

Those of you who are lucky enough to be using a high level language that supports recursion or who are patient enough to code in assembler or machine language can eliminate back pointers and the indicators telling which direction the tree climber came from and all the code associated with them. Recursive tree handling routines call themselves instead of looping to the beginning of themselves. At each call, they save their activation records - all values of variables at the time of the call - in a push down stack. The index of the node they are on is in the activation record. Each return pops the activation record restoring the values it was working with at that level. The program keeps track of the node it was working with by the level it is at. It keeps track of the direction it was coming from by the point of the call and return. Recursive routines look much neater - academics say more elegant - than the clumsy loops and branches in the example.

Final Note

A simulated sort using a binary tree index is not the answer for all sorting problems. Some applications require that the entries really be sorted. Well, the binary tree index can be used to really sort something by simply writing the records to storage instead of to the printer or terminal. And it will be faster than most methods for that purpose if the input is well disordered. But take note: Not counting the index or the extra program space, to do a real sort, the binary tree sort will take nearly twice the space that a bubble sort will take. That can be devastating for the microcomputer owner.

But for many purposes the binary tree sort is effective and provides much better overall response to the user.

But most of all, it is fun to climb trees.

S-100 COMPUTERS

AT SENSIBLE PRICES

NORTHSTAR SPECIALIST
HAZELTINE, IMS, TELEVIDEO, NEC, ANADEX

BUSINESS SOFTWARE

COMPUTERS BY
FREEPORT UTILITIES COMPANY
40 NORTH MAIN STREET
FREEPORT, NEW YORK 11520
(516) 379-2400

LETTERS TO THE EDITOR

Sol:

Please publish the new number for CBBS/Chicago. Randy is moving, so a new number is necessary. Effective 3/31/80, (312) 528-7141 will be disconnected, and the CBBS installed at the new number:
(312) 545-8086

Ward Christensen
CACHE

Dear Sol & Russ:

I'm sure I am not alone when I say that it's about time for a magazine such as yours.

We do get a little tired of sifting through TRS-80, Apple, Pet, etc. looking for something applicable to our S-100 systems.

Best of luck to you.

Lee Osborne
Orange, Calif.

Dear Mr. Libes:

Please enter my subscription to S-100 Microsystems, starting with issue No. 1, if possible. A check for $7.50 is enclosed. Mailing address is given above.

I would like to see an article directed at a specific area that, I suspect, impacts many of us. That is, how can one buy new, state of the art, S-100 boards with **reasonable** confidence?

I have purchased most of my hardware through a local dealer or through a well established mail-order source. There are, however, many new interesting boards using the 8086, 6809, color graphics, and new CPU's waiting in the wings which are advertized in the computer magazines. Local dealers usually don't handle such items for some time.

Perhaps your publication or some unbiased and impartial person would be willing to evaluate some of these interesting items and indicate their availability, the response of the manufacturer, the difficulty of getting them up and running, etc. Perhaps the manufacturer would supply a sample or a loan, in return for a review.

I suppose the bottom line is, who is going to do this and how is he going to get paid? Perhaps you or your readers would have some ideas.

Stanley W. Haskell
Arlington, Mass.

Dear Sirs:

Post haste enter my subscription to your S-100 Microsystems for 3 yrs. so I can start dropping my others as they have turned to TRS etc. Newsletters. My check for 3 yrs. (21.50) enclosed.

John W. Neel
Apopka, Fla.

Dear Mr. Libes:

I recently subscribed to your magazine in the hope that it would, in its concentration on the S-100, deal with the information and 'need to know' problem that I have been experiencing in this area.

My cumulative frustrations were the source of my expectations for the first issue, unfortunately I received very little in the way of relief when I received my first copy, hence this letter.

For the sake of brevity my list of 'wants' follows.

1. I am only interested in products which are **fully** compatible with the new IEEE S-100 standard.
2. I need help in identifying specific products and their manufacturers, and in identifying from any one source those products which meet the standard and those which do not.
3. I need critical review information on specific products and comparisons within classes of product (e.g. mainframes, memory boards, processor boards, etc.)
4. I need information on the future direction of the IEEE S-100 product industry, with future product intentions of individual companies.
5. I need software information particularly in the Operating System/ Utilities areas, with feature comparisons, family variant identification, and rational critiques.

One could go on, but no doubt others will give you reactions in other ways. For the moment then my best wishes for your success in filling a real need in the micro spectrum.

Derek Grieves
Chestnut Hill, Mass.

MIDWEST AFFILIATION OF COMPUTER CLUBS

&

FRANKLIN UNIVERSITY

•CO-SPONSORS•

5th ANNUAL COMPUTERFEST

Franklin University
301 E. Rich Columbus, Ohio

JUNE 20-22

- See the latest in small business and personal computers
- Learn from stimulating seminars
- Participate in technical work-shops
- Browse through the large, flea market
- Pre-register
- Park free in a convenient, location

To Pre-register, send $3.50 to:
COMPUTERFEST '80
c/o Paul Pittinger
215 Delhi Ave., Apt. J
Columbus, Ohio 43202

For more information: (614) 461-9825 • "All roads lead to Columbus - Heart of the Midwest"

COMPUTERFE
80 COMPUTERFEST 80
COMPUTERFEST 80

n

Linear Programming -

Continued From Page 29

```
EDIT: L(IST, B(UILD, M(ODIFY, Q(UIT [1.0] L

LIST WHAT FILE? DIET.DATA

STARTING AT WHAT RECORD? 0

    0:    0  DIET          4     14
    1:    1 DIET PROBLEM EXAMPLE
    2:    2  NUTR.         1    74.2000
    3:    2  PROT.         2    14.7000
    4:    2  CALC.         3     0.140000 )
    5:    2  PHOS.         4     0.550000 )
    6:    4  CORN          1     2.40000
    7:    4  OATS          2     2.52000
    8:    4  MAIZE         3     2.18000
    9:    4  BRAN          4     2.14000
   10:    4  MIDDL.        5     2.44000
   11:    4  LINSD.        6     3.82000
   12:    4  COTTON        7     3.55000
   13:    4  SOYML.        8     3.70000
   14:    4  GLUTEN        9     2.60000
   15:    4  GRITS        10     2.54000
   16:    4  ENUTR.       11     0.00000
   17:    4  EPROT.       12     0.00000
   18:    4  ECALC.       13     0.00000
   19:    4  EPHOS.       14     0.00000
   20:    6 ROW  1 COL  1   78.6000
   21:    6 ROW  1 COL  2   70.1000
   22:    6 ROW  1 COL  3   88.1000
   23:    6 ROW  1 COL  4   67.2000
   24:    6 ROW  1 COL  5   78.9000
   25:    6 ROW  1 COL  6   77.0000
   26:    6 ROW  1 COL  7   70.6000
   27:    6 ROW  1 COL  8   78.5000
   28:    6 ROW  1 COL  9   76.3000
   29:    6 ROW  1 COL 10   84.5000
   30:    6 ROW  1 COL 11   -1.00000
   31:    6 ROW  2 COL  1    6.50000
   32:    6 ROW  2 COL  2    9.40000
   33:    6 ROW  2 COL  3    8.80000
   34:    6 ROW  2 COL  4   13.7000
   35:    6 ROW  2 COL  5   16.1000
   36:    6 ROW  2 COL  6   30.4000
   37:    6 ROW  2 COL  7   32.8000
   38:    6 ROW  2 COL  8   37.1000
   39:    6 ROW  2 COL  9   21.3000
   40:    6 ROW  2 COL 10    8.00000
   41:    6 ROW  2 COL 12   -1.00000
   42:    6 ROW  3 COL  1    0.0200000)
   43:    6 ROW  3 COL  2    0.0900000)
   44:    6 ROW  3 COL  3    0.0300000)
   45:    6 ROW  3 COL  4    0.140000 )
   46:    6 ROW  3 COL  5    0.0900000)
   47:    6 ROW  3 COL  6    0.410000 )
   48:    6 ROW  3 COL  7    0.200000 )
   49:    6 ROW  3 COL  8    0.260000 )
   50:    6 ROW  3 COL  9    0.480000 )
   51:    6 ROW  3 COL 10    0.220000 )
   52:    6 ROW  3 COL 13   -1.00000
   53:    6 ROW  4 COL  1    0.270000 )
   54:    6 ROW  4 COL  2    0.340000 )
   55:    6 ROW  4 COL  3    0.300000 )
   56:    6 ROW  4 COL  4    1.29000
   57:    6 ROW  4 COL  5    0.710000 )
   58:    6 ROW  4 COL  6    0.860000 )
   59:    6 ROW  4 COL  7    1.22000
   60:    6 ROW  4 COL  8    0.590000 )
   61:    6 ROW  4 COL  9    0.820000 )
   62:    6 ROW  4 COL 10    0.710000 )
   63:    6 ROW  4 COL 14   -1.00000
   64:   99 LOGICAL EOF

EDIT: L(IST, B(UILD, M(ODIFY, Q(UIT [1.0] Q
```

Listing 6: Data File For Diet Problem

Listing 7: Diet Problem Run

```
ENTER DATA FILE NAME ---> DIET.DATA

PROG. NAME = DIET
NO. ROWS   =      4
NO. COLS   =     14

START PHASE 1

ITERATION 1 OF DIET
ITERATION 2 OF DIET
ITERATION 3 OF DIET
ITERATION 4 OF DIET
ITERATION 5 OF DIET
END OF PHASE 1 FOR DIET    AFTER 5 ITERATIONS

LIST & X ARRAYS

       1  EPHOS.      14          0.144202 )
       2  LINSD.       6          0.332113 )
       3  GRITS       10          0.575471 )
       4  ECALC.      13          0.122770 )
       5  M+1         19         -2.73037
       6  M+2         20          0.00000063

START PHASE 2

ITERATION 1 OF DIET
ITERATION 2 OF DIET
ITERATION 3 OF DIET
ITERATION 4 OF DIET
ITERATION 5 OF DIET
END OF PHASE 2 FOR DIET    AFTER 5 ITERATIONS

LIST & X ARRAYS

       1  MAIZE        3          0.187710 )
       2  GLUTEN       9          0.170195 )
       3  MIDDL.       5          0.585280 )
       4  EPHOS.      14          0.0614213)
       5  M+1         19         -2.27980
       5  M+2         20         -0.00000030

DIET PROBLEM EXAMPLE
```

PROBLEM 4 -- GAMING STRATEGY

If you started here, go back to the beginning, and start there. After all, I had to save a carrot to induce reading this, didn't I?

In the theory of games, the principle characteristic of a game is the PAYOFF MATRIX. It is the expression of the gains of Player 1 for all combinations of his possible plays and those of his opponent (Player 2). We choose Player 1 (P1), and say we desire to maximize his payoff; the variables are the playing strategies which he and his opponent may select. Usually, these strategies are given as a vector of probabilities, since if his strategy is fixed, it's not a very interesting game. As an example, in the game of calling 'heads' or 'tails' on the flip of a fair coin, there are two 'pure' strategies -- always calling 'heads' or always calling 'tails'. We generally want to calculate the best 'mixed' strategy, in which the player uses each of the pure strategies some fraction of the time. If the game has two players, it is called a TWO-PERSON game. In addition, if the sums of the wins and losses balance, it is also called a ZERO-SUM game. A game of poker, in which the 'house' cuts the pot, is NOT a Zero-Sum game.

Another parameter of the game is the value -- this is the expected (in a statistical or probabilistic sense) gains by P1 if both he and his opponent adopt their optimal strategies. The game is 'fair' if this value is zero; otherwise the game is said to be biased in favor of one of the players.

If the payoff matrix is given by A, P1 selects a strategy X = (X1,X2,...Xn), and P2 selects as his strategy Y = (Y1,Y2,...Yn), then the game's value, V, is given by:

$$V = X * A * Y$$

where matrix multiplication is indicated.

Let us now examine a game, and ask 'is it fair?'. As an example, let us use the 'skin game', which has the following rules:

> The two players are each provided with an Ace of Clubs and an Ace of Diamonds. P1 is also given the Deuce of Diamonds, and P2 the Deuce of Clubs.
>
> In the first move, P1 selects one of his cards, playing it face down. P2 then selects one of his cards, and the two cards are compared with the following payoff -- P1 wins if the suits match; P2 wins if they do not. The amount of the payoff is the numerical value of the card shown by the winner. If the two Deuces are shown, however, the payoff is zero.

From these rules, we can construct the payoff matrix. P2's possible selections are shown across (the top), and P1's are shown at the (left) side:

	AD	AC	2C
AD	1	-1	-2
AC	-1	1	1
2D	2	-1	0

It can be shown that the optimum strategy, X, for P1 is (0,3/5,2/5); for P2, his strategy, Y, is (2/5,3/5,0). These strategies mean the P1 plays the Ace of Diamonds never (0), the Club Ace 60% of the time, and the Diamond Deuce 40% of the time. When we compute V for this game, we find that its value is not zero -- it is 1/5, which means the game is not fair, but is 'rigged' in favor of P1 (since the value is positive). We say that P1 has an advantage.

Problem Statement and Problem Formulation

Given a payoff matrix, A, and stragegies for the players P1 : (X1,X2,X3) and P2 : (Y1,Y2,Y3), where the strategies are expressed in terms of probabilities, then the payoff, V, to P1 is:

$$\begin{aligned} X_1 - X_2 - 2X_3 &= V \text{ if P2 selects Y1,} \\ -X_1 + X_2 - X_3 &= V \text{ if P2 selects Y2, and} \\ -2X_1 + X_2 &= V \text{ if P2 selects Y3.} \end{aligned}$$

NEW! from ads

6809 SINGLE-BOARD COMPUTER
S-100 bus

- IEEE S-100 Proposed Standard
- 2K RAM
- 4K/8K/16K ROM
- PIA, ACIA Ports
- adsMON; 6809 Monitor Available

P.C. Board & Manual Presently Available

ALL PC BOARDS FROM ADS ARE SOLDER MASKED, WITH GOLD CONTACTS, & PARTS LAYOUT SILK SCREENED ON BOARD.
Add 50¢ postage & handling per item.
Ill. residents add sales tax.

Sound Effects . . . Sound Effects . . . !!!
© NOISEMAKER * & © NOISEMAKER II **
S-100 bus Apple IItm bus

ADD "SPACESHIP" SOUNDS, PHASERS, GUNSHOTS, TRAINS, MUSIC, SIRENS, ETC.!
UNDER SOFTWARE CONTROL!!!

- Soundboards Use GI AY 3-8910 I.C.'s to Generate Programmable Sound Effects.
- On Board Audio Amp. Breadboard Area With +5 & GND.
- Noise Sources • Envelope Generators • I/O Ports

PCB & Manual: *39.95 (NM); **34.95 (NM II)

!!!!!ATTENTION APPLE II USERS!!!!!!
Assembled and Tested NM II Units Now Available!!!

Call or Write for Details.

ads

Ackerman Digital Systems, Inc., 110 N. York Road, Suite 208, Elmhurst, Illinois 60126 (312) 530-8992

Listing 4: Data File Transportation Problem Example

```
EDIT: L(IST, B(UILD, M(ODIFY, Q(UIT [1.0] L

LIST WHAT FILE? TRANSPRT.DATA

STARTING AT WHAT RECORD? 0

     0:   0  TRANSP        7       11
     1:   1  TRANSPORTATION PROBLEM EXAMPLE
     2:   2  SHIP1         1      6.00000
     3:   2  SHIP2         2      8.00000
     4:   2  SHIP3         3     10.0000
     5:   2  RECV1         4      4.00000
     6:   2  RECV2         5      6.00000
     7:   2  RECV3         6      8.00000
     8:   2  RECV4         7      6.00000
     9:   4  C11           1      1.00000
    10:   4  C12           2      2.00000
    11:   4  C13           3      3.00000
    12:   4  C14           4      4.00000
    13:   4  C21           5      4.00000
    14:   4  C22           6      3.00000
    15:   4  C23           7      2.00000
    16:   4  C24           8      0.00000
    17:   4  C31           9      0.00000
    18:   4  C32          10      2.00000
    19:   4  C33          11      2.00000
    20:   4  C34          12      1.00000
    21:   6 ROW  1 COL  1      1.00000
    22:   6 ROW  1 COL  2      1.00000
    23:   6 ROW  1 COL  3      1.00000
    24:   6 ROW  1 COL  4      1.00000
    25:   6 ROW  2 COL  5      1.00000
    26:   6 ROW  2 COL  6      1.00000
    27:   6 ROW  2 COL  7      1.00000
    28:   6 ROW  2 COL  8      1.00000
    29:   6 ROW  3 COL  9      1.00000
    30:   6 ROW  3 COL 10      1.00000
    31:   6 ROW  3 COL 11      1.00000
    32:   6 ROW  3 COL 12      1.00000
    33:   6 ROW  4 COL  1      1.00000
    34:   6 ROW  4 COL  5      1.00000
    35:   6 ROW  4 COL  9      1.00000
    36:   6 ROW  5 COL  2      1.00000
    37:   6 ROW  5 COL  6      1.00000
    38:   6 ROW  5 COL 10      1.00000
    39:   6 ROW  6 COL  3      1.00000
    40:   6 ROW  6 COL  7      1.00000
    41:   6 ROW  6 COL 11      1.00000
    42:   6 ROW  7 COL  4      1.00000
    43:   6 ROW  7 COL  8      1.00000
    44:   6 ROW  7 COL 12      1.00000
    45:  99 LOGICAL EOF

EDIT: L(IST, B(UILD, M(ODIFY, Q(UIT [1.0] Q
```

Listing 5: Transportation Problem Run

```
ENTER DATA FILE NAME ---> TRANSPRT.DATA


PROG. NAME = TRANSP
NO. ROWS   =      7
NO. COLS   =     11


START PHASE 1

ITERATION 1 OF TRANSP
ITERATION 2 OF TRANSP
ITERATION 3 OF TRANSP
ITERATION 4 OF TRANSP
ITERATION 5 OF TRANSP
ITERATION 6 OF TRANSP
ITERATION 7 OF TRANSP
ITERATION 8 OF TRANSP
ITERATION 9 OF TRANSP
END OF PHASE 1 FOR TRANSP AFTER 9 ITERATIONS
```

```
LIST & X ARRAYS

        1  C31          9           4.00000
        2  C22          6           0.00000
        3              14           0.00000
        4  C13          3           0.00000
        5  C14          4           6.00000
        6  C23          7           8.00000
        7  C32         10           6.00000
        8  M+1         19         -52.0000
        9  M+2         20           0.00000

START PHASE 2

ITERATION 1 OF TRANSP
ITERATION 2 OF TRANSP
ITERATION 3 OF TRANSP
END OF PHASE 2 FOR TRANSP AFTER 3 ITERATIONS

LIST & X ARRAYS

        1  C31          9           4.00000
        2  C12          2           6.00000
        3              14           0.00000
        4  C33         11           6.00000
        5  C24          8           6.00000
        6  C23          7           2.00000
        7  C32         10           0.00000
        8  M+1         19         -28.0000
        9  M+2         20           0.00000

TRANSPORTATION PROBLEM EXAMPLE
```

PROBLEM 3 -- THE DIET PROBLEM

Several models of the diet problem have been constructed in the past, some with small success. They all have in common that the minimum cost diet consists mainly of only a few cheap food elements; the diets would not appeal to very many. In fact, they have been described as poor in variety even for a slave labor camp. One may extend the model, including in the cost factors such other considerations as taste, ethnic preferences, and food fads. These extensions invariably increase the cost of the minimum-cost diet. One area in which considerable success has been achieved, however, is in the analysis of the minimum-cost feed mixes for farm animals. The reason for this success is probably due to the fact that they don't complain about a monotonous diet.

Problem Statement

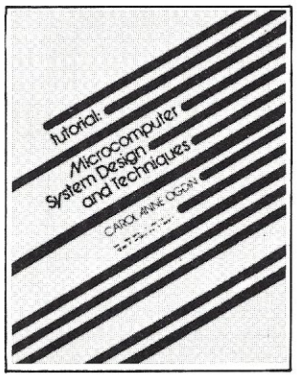
Our problem is to determine the minimum-cost mix of feeds for dairy cattle. There are four nutrients considered essential in the following minimum daily amounts:

Digestible nutrients	: 74.2 lbs/day
Digestible protein	: 14.7 lbs/day
Calcium	: 0.14 lbs/day
Phosphorous	: 0.55 lbs/day

There are 10 commonly available feeds, with the amounts (in lbs) of nutrient, and the cost -- data in Table 1. Nutrient content is given in pounds per 100 lbs. of feed; cost is based on 100 lbs.

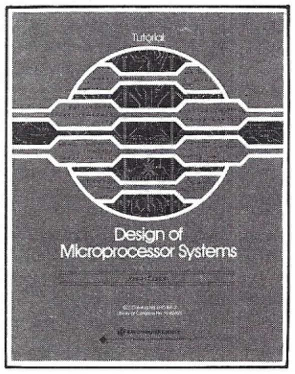
DON'T MISS these important new publications by the IEEE Computer Society:

259 Tutorial: Microcomputer System Design and Techniques *Carol Anne Ogdin*



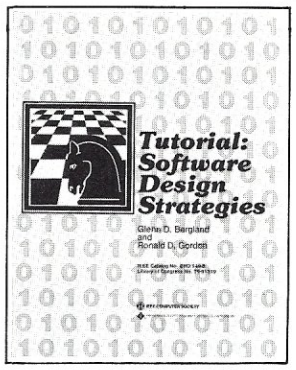
(432 pp.) *$12.00/$16.00*
The purpose of this anthology is to capitalize on the programmer's experience with systems and software in order to introduce and clarify the differences among micros. The volume is composed of 53 different papers divided into seven topics, covering micros and their applications, microprocessor architecture, microcomputer buses and systems, storage technology, input/output interfacing, programming and languages, management, and tools. Text is aimed primarily at programmers, analysts, and technicians as well as system engineers and project managers who may become responsible for or participate in the implementation of microcomputer systems.

260 Tutorial: Design of Microprocessor Systems
(December 1979), John H. Carson



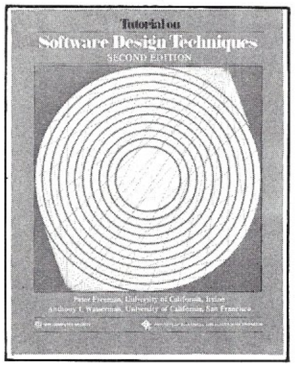
(262 pp.) *$12.00/$16.00*
This tutorial is intended for those involved in the design of microprocessor-based systems. Presents the entire design effort, with emphasis on system configuration, software development, and system testing. Stresses the wide range of available microprocessor products and the development tools for microprocessor-based design. Topics covered: review of microprocessor-based systems, design steps, testing and development tools, design alternatives, and current trends affecting design. Contains 23 reprints.

268 Tutorial: Software Design Strategies
(November 1979)
Edited by Glenn D. Bergland and Ronald D. Gordon


Tutorial: Software Design Strategies
Glenn D. Bergland and Ronald D. Gordon

(430 pp.) *$12.00/$16.00*
This tutorial begins with the Jackson design methodology and delves into logical construction programs and systems, as well as structured design and stepwise refinement based on functional decomposition. It displays numerous methodologies and techniques and concludes with PSL/PSA structured documentation and analysis, program design languages, and other tools for software design strategies. Contains 32 reprints, with examples and exercises.

075 Tutorial: Software Design Techniques
(Second Edition, April 1977)
Edited by Peter Freeman and Anthony I. Wasserman



(294 pp.) *$9.00/$12.00*
Intended for both beginning and experienced designers, this book contains 23 key papers as well as original material explaining design concepts. The contents include the following: introduction, framework of design, elements of design techniques, design tools, design methodologies, examples, and an annotated bibliography.

272 Microprocessors and Microcomputers
(Second Edition), Edited by Portia Isaacson

MICROPROCESSORS & MICROCOMPUTERS
2nd EDITION
Selected reprints from COMPUTER

(298 pp.) *$9.00/$12.00*
Selected papers from COMPUTER, organized and introduced by the technical editor. Sections on architecture, software, and applications include the standard specification for S-100 bus interface devices and special articles on modular programming in PL/M, microprocessor networks, and microprocessors in automation and communications.

Return to: Order Desk, IEEE Computer Society, 5855 Naples Plaza #301, Long Beach, CA 90803

ORDER NO.	TUTORIAL TITLES	QTY.	MEMBER PRICE	NON-MEMBER PRICE
075	Tutorial: Software Design Techniques		$9.00	$12.00
259	Tutorial: Microcomputer System Design and Techniques		12.00	16.00
260	Tutorial: Design of Microprocessor Systems		12.00	16.00
268	Tutorial: Software Design Strategies		12.00	16.00
272	Microprocessors and Microcomputers		9.00	12.00

Overseas purchases:
Remit U.S. dollars on U.S. Bank.

☐ Check Enclosed
☐ Bill Visa/BankAmericard
☐ Bill Master Charge

California residents add 6% sales tax ____________
☐ Bill me and add $3.00 billing charge ____________
Total ____________
Optional Shipping Charge (4th class, no charge) ____________
Total ____________

Charge Card Number ______ Expiration Date ______ Signature ______

Name (please print) ______ Member No. ______

Address ______

City/State/Zip ______ Country ______

Table 1
Nutrient content and cost of dairy feeds

No.	Feed	Nutr.	Prot.	Calc.	Phos.	Cost
1	Corn	78.6	6.5	0.02	0.27	$2.40
2	Oats	70.1	9.4	0.09	0.34	2.52
3	Maize	80.1	8.8	0.03	0.30	2.18
4	Bran	67.2	13.7	0.14	1.29	2.14
5	Middlings	78.9	16.1	0.09	0.71	2.44
6	Linseed Meal	77.0	30.4	0.41	0.86	3.82
7	Cottonseed Meal	70.6	32.8	0.20	1.22	3.55
8	Soybean Meal	78.5	37.1	0.26	0.59	3.70
9	Gluten	76.3	21.3	0.48	0.82	2.60
10	Hominy Meal	84.5	8.0	0.22	0.71	2.54

We need four additional variables (having no cost), which represent excess nutrients in each of the four categories. These 'slack' or dummy variables are required because the four constraints on daily nutritional requirements are given as "at least"; we need to include them to convert the inequalities into equalities.

Problem Formulation

Let X_1 through X_{10} represent the amount of the ten feeds shown in table 1 that are included in the mix, and X_{11} thru X_{14} the amounts of excess nutrients. Then,

$$(78.6)*X_1 + (70.1)*X_2 + (80.1)*X_3 + (67.2)*X_4 + (78.9)*X_5 + (77.0)*X_6 + (70.6)*X_7 + (78.5)*X_8 + (76.3)*X_9 + (84.5)*X_{10} - X_{11} = 74.2 \text{ (Digestible Nutrients)}$$

$$(6.5)*X_1 + (9.4)*X_2 + (8.8)*X_3 + (13.7)*X_4 + (16.1)*X_5 + (30.4)*X_6 + (32.8)*X_7 + (37.1)*X_8 + (21.3)*X_9 + (8.0)*X_{10} - X_{12} = 14.7 \text{ (Digestible Protein)}$$

$$(.02)*X_1 + (.09)*X_2 + (.03)*X_3 + (.14)*X_4 + (.09)*X_5 + (.41)*X_6 + (.20)*X_7 + (.26)*X_8 + (.48)*X_9 + (.22)*X_{10} - X_{13} = 0.14 \text{ (Calcium)}$$

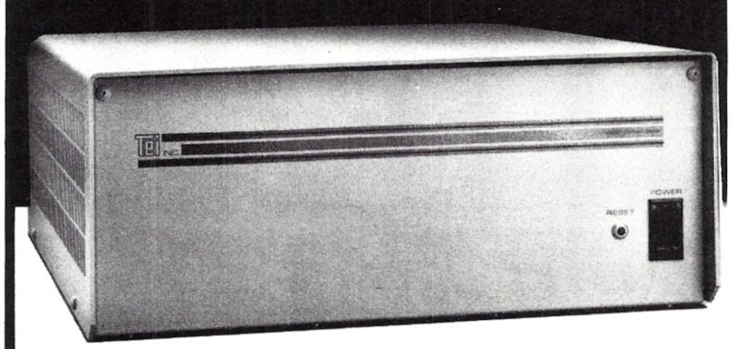
$$(.27)*X_1 + (.34)*X_2 + (.30)*X_3 + (1.29)*X_4 + (.71)*X_5 + (.86)*X_6 + (1.22)*X_7 + (.59)*X_8 + (.82)*X_9 + (.71)*X_{10} - X_{14} = 0.55 \text{ (Phosphorous)}$$

The data file is shown in Listing 6. We use 6-character abbreviations for the feeds and excess nutrients. Listing 7 shows the program run, with the following results (note that feed quantities are in units of 100 lbs.):

> Three basic feeds are included -- 18.77 lbs of Maize, 17.02 lbs of Gluten, and 58.53 lbs of Middlings. They provide 0.0614 lbs of Excess Phosphorous, and the cost is $2.2798 for this mix.

OK for cattle, I guess, but I wouldn't want to live on it.

The Mainframe.
(or how to get a good night's sleep)



There is no other mainframe that compares with the performance and reliability of a TEI mainframe. Its unique design enhances substantially the reliability of any S-100 computer system by providing high efficiency power, brown out protection, line noise rejection and a sophisticated high-speed bus packaged in a durable enclosure.

TEI manufactures the broadest selection of S-100 mainframes . . . 8, 12 and 22 slot, desk top and rackmount models. Whether your requirements are standard or custom, TEI's extensive manufacturing capacity and know-how can solve your mainframe problems today!

Successful OEM's, system integrators and computer dealers worldwide rely on TEI mainframes and enjoy a good night's sleep knowing that their systems are still running. Call TEI to day . . . you too can enjoy a good night's sleep!

TEI **More than a decade of reliability.**

5075 S. LOOP E., HOUSTON, TX. 77033 (713) 783-2300 TWX. 1 910-881-3639

"THE ORIGINAL"
Personal Computing® 80

Presents:

Personal Computing and Small Business Computer Show

The Largest Personal Computing Show in 1980

August 21, 22,23, 24th at the Philadelphia Civic Center

- **Major exhibits by the industries leading companies**
- **Thursday, Aug. 21st, Dealer Day ——— 12 Noon to 6 P.M.**
- **Friday and Saturday, Aug. 22, 23rd ——— 9 A.M. to 6 P.M.**
- **Sunday, Aug. 24th ——— 10 A.M. to 5 P.M.**
- **Free Seminars** ● **Robotics Contest** ● **Antique Computer Display**
- **Special Seminars and Tutorials about Computer Music,** Saturday, Aug. 23rd
- **3rd Annual Computer Music Festival,** Saturday Evening, Aug. 23rd
  (Computer Music Festival is sponsored by the Philadelphia Area Computer Society-Tickets on sale at show)
- **Computer Visual Arts Festival,** Sunday, Aug. 24th

Advanced Registration
Saves Time & Money

☐ Send______Dealer-Retailer (4 days) Registrations at $10. each, $12. at door for Thursday-Sunday, Aug. 21, 22, 23, 24

☐ Send______Regular Registrations (3 days) at $8. each, $10. at door for Friday-Sunday, Aug. 22, 23, 24 only.

Advanced Registrations will be mailed late July - early August. No Advanced Registrations accepted after Aug. 8th.

☐ Send Exhibitor information or Phone 609-653-1188

COMPANY NAME ______________________

NAME ______________________

STREET ______________________

CITY ______________ STATE ________ ZIP ________

PHONE ______________________

Send To:

PERSONAL COMPUTING 80

Rt. 1, Box 242, Warf Rd., ● Mays Landing, NJ 08330

Since P1 chooses his plays in some manner, not yet known to us, we can say that P1 will gain, in these cases, at least the value, V. These equations indicate that P1 can maximize his gain by selecting a set of plays (X_1,X_2,X_3) which will maximize his payoff for any choice that P2 may make. This is a classical problem for linear programming, if we add one additional constraint:

$$X_1 + X_2 + X_3 = 1$$

since we are defining the strategy in terms of the probability that P1 makes any one of the plays available to him. Let us take a look at another game, with the following payoff matrix:

$$A = \begin{array}{ccc} 3 & -2 & -4 \\ -1 & 4 & 2 \\ 2 & 2 & 6 \end{array}$$

We would like the answers to the following questions:

1. Is the game fair?
2. What strategy would P1 use if he desires to maximize his gains (or minimize his losses)?

Our equations are:

$$\begin{aligned} 3{*}X_1 - {*}X_2 + 2{*}X_3 &>= V \\ -2{*}X_1 + 4{*}X_2 + 2{*}X_3 &>= V \\ -4{*}X_1 + 2{*}X_2 + 6{*}X_3 &>= V \\ X_1 + \quad X_2 + \quad X_3 &= 1 \end{aligned}$$

We must convert the inequalities in the first three equations to equalities by including 'slack' or dummy variables.

$$\begin{aligned} 3{*}X_1 - \quad X_2 + 2{*}X_3 - X_4 \qquad\qquad &= V \\ -2{*}X_1 + 4{*}X_2 + 2{*}X_3 \qquad - X_5 \qquad &= V \\ -4{*}X_1 + 2{*}X_2 + 6{*}X_3 \qquad\qquad - X_6 &= V \\ X_1 + \quad X_2 .+ \quad X_3 \qquad\qquad\qquad &= 1 \end{aligned}$$

Take the first equation, since it is an equation for V, and take it as a statement of our objective:

$$\text{Maximize } 3X_1 - X_2 + 2X_3 - X_4$$

and eliminate it from the equations above. Subtracting this equation from the second and third equations, we get our set of problem constraints:

$$\text{Maximize } 3{*}X_1 - X_2 + 2{*}X_3 - X_4$$

subject to

$$\begin{aligned} -5{*}X_1 + 5{*}X_2 \qquad\quad + X_4 - X_5 \qquad &= 0 \\ -7{*}X_1 + 3{*}X_2 + 4{*}X_3 + X_4 \qquad - X_6 &= 0 \\ X_1 + \quad X_2 + \quad X_3 \qquad\qquad\qquad &= 1 \end{aligned}$$

Since the standard form for the program is to minimize, we convert our objective (multiply by -1) to give

$$\text{MInimize } -3{*}X_1 + X_2 - 2{*}X_3 + X_4$$

subject to the above constraints. We have a problem in 3 equations (rows) with six variables (columns) of which 3 are slack variables.

The data file for this problem is shown in Listing 8, and the program run in Listing 9.

Listing 8: Data File For Game Problem

```
EDIT: L(IST, B(UILD, M(ODIFY, Q(UIT [1.0] L

LIST WHAT FILE? GAME.DATA

STARTING AT WHAT RECORD? 0

  0:   0  GAMES        3      6
  1:   1 GAMES STRATEGY EXAMPLE
  2:   2  S1           1      0.00000
  3:   2  S2           2      0.00000
  4:   2  S3           3      1.00000
  5:   4  C1           1     -3.00000
  6:   4  C2           2      1.00000
  7:   4  C3           3     -2.00000
  8:   4  C4           4      1.00000
  9:   4  C5           5      0.00000
 10:   4  C6           6      0.00000
 11:   6 ROW  1 COL  1    -5.00000
 12:   6 ROW  1 COL  2     5.00000
 13:   6 ROW  1 COL  4     1.00000
 14:   6 ROW  1 COL  5    -1.00000
 15:   6 ROW  2 COL  1    -7.00000
 16:   6 ROW  2 COL  2     3.00000
 17:   6 ROW  2 COL  3     4.00000
 18:   6 ROW  2 COL  4     1.00000
 19:   6 ROW  2 COL  6    -1.00000
 20:   6 ROW  3 COL  1     1.00000
 21:   6 ROW  3 COL  2     1.00000
 22:   6 ROW  3 COL  3     1.00000
 23:  99 LOGICAL EOF

EDIT: L(IST, B(UILD, M(ODIFY, Q(UIT [1.0] Q
```

Listing 9: Sample Run of Game

```
ENTER DATA FILE NAME ---> GAME.DATA


PROG. NAME = GAMES
NO. ROWS   =     3
NO. COLS   =     6


START PHASE 1

ITERATION 1 OF GAMES
ITERATION 2 OF GAMES
ITERATION 3 OF GAMES
END OF PHASE 1 FOR GAMES  AFTER 3 ITERATIONS

LIST & X ARRAYS

     1  C1          1          0.333333 )
     2  C2          2          0.333333 )
     3  C3          3          0.333333 )
     4  M+1        10          1.33333
     5  M+2        11          0.00000012

START PHASE 2

ITERATION 1 OF GAMES
END OF PHASE 2 FOR GAMES  AFTER 1 ITERATIONS

LIST & X ARRAYS

     1  C6          6          4.00000
     2  C2          2          0.00000
     3  C3          3          1.00000
     4  M+1        10          2.00000
     5  M+2        11         -0.00000012

GAMES STRATEGY EXAMPLE
```

P1's strategy is that he should select play 1 (X_1) 0% of the time, play 2 (X_2) 0% of the time, and play 3 (X_3) 100% of the time. His payoff value (shown in the M + 1 row) is 2.0. A zero value would have indicated no bias, but here, P1 has the game rigged. If the value here had been negative, it would indicate bias toward P2.

If we wish to calculate the strategy and payoff for P2, we would set up our equations using rows, instead of columns of the payoff matrix as we did above. If we did this, (and did not make any mistakes), we would expect a change in sign of the computed game's value.

CONCLUSIONS

In general, the problems which can be solved by Linear Programming techniques are those in which we desire to optimize some quantity which is subject to constraints. The suggested reading referenced in part 1 give hundreds of applications in which it has been successfully used.

I am willing to provide these programs and data files on North Star UCSD PASCAL disk. Anyone wishing a disk may get it from me by sending a Postal Money Order for $20 (US) to my address. This will cover cost of disk and handling/mailing costs.

the microcomputer people®

computer mart of new jersey

THE VITAL INGREDIENT: **EXPERTISE**

Before you buy your new microcomputer, chances are you have a lot of questions. Important questions that could mean the difference between a working system and a wasted system. The vital ingredient is expertise. The microcomputer people at Computer Mart are expert at answering your questions and helping you put together the best system for your application. Whether it's for business, the home, or the laboratory, come see the experts at Computer Mart of New Jersey. We have the vital ingredient.

Computer Mart of New Jersey
501 Route 27
Iselin, N.J. 08830
(201) 283-0600

HOURS:
Open at 10 am,
Tuesday through Saturday

ADDRESSING THE CURSOR

Continued From Page 33

```
3230 REM ********************************************************
3240 REM *                                                      *
3250 REM *       ROUTINE FOR RETRIEVING LABELS FROM DISK        *
3260 REM *                                                      *
3270 REM ********************************************************
3280 PRINT "ENTER DRIVE ON WHICH LABEL IS STORED (A,B,C,D) ";
3290 Z$=INPUT$(1):PRINT Z$
3300 Z$=CHR$(ASC(Z$) AND &HDF)
3310 IF Z$["A" OR Z$|"D" THEN 3280
3320 F$=Z$+":*.LAB"
3330 PRINT
3340 FILES F$
3350 PRINT:PRINT
3360 PRINT "ENTER A FILE NAME FROM THE ABOVE LIST"
3370 LINEINPUT "USE FILE NAME ONLY, NO EXTENSION ";Z$
3380 FOR N=1 TO LEN(Z$):MID$(Z$,N,1)=CHR$(ASC(MID$(Z$,N,1)) AND &HDF):NEXT N
3390 F$=Z$+".LAB"
3400 OPEN "I",1,F$
3410 INPUT#1,B$,C$
3420 WD$=B$:WD=VAL(WD$)
3430 LN$=C$:LN=VAL(LN$)
3440 ERASE A$
3450 DIM A$(LN)
3460 FOR N=1 TO LN
3470 LINEINPUT#1,D$
3480 A$(N)=D$+STRING$(WD-LEN(D$),32)
3490 NEXT N
3500 CLOSE
3510 GOSUB 2730
3520 GOTO 660
3530 REM
3540 REM
3550 REM ********************************************************
3560 REM *                                                      *
3570 REM *     IN THIS LISTING, "|" MEANS "IS GREATER THAN      *
3580 REM *                      "[" MEANS "IS LESS THAN"        *
3590 REM *                                                      *
3600 REM ********************************************************
```

If the above routine is used, then no control characters may be used as input to this program.

Lines 2930-3210 are used to enter preformatted labels for any special purpose. The program here uses this label to prepare diskette labels for Prodigy Systems, Inc. When the labels are prepared, certain questions are asked to provide the variable information. If the label is stored on diskette, when they are recalled, the standard method of display is used. Notice, when using this part of the program, the label size is set to a width of 40 and a length of 8. You may substitute any format of label here or delete this code entirely. If you delete this section of the program, be sure to delete the question at lines 520-540.

Lines 3280-3520 provide for reusing labels stored on the diskette. This routine is essentially the same as the routine for storing labels on the diskette in lines 1140-1390.

The remarks at the end of the program, lines 3550-3600 may seem to be unnecessary. However, some printers do not print the 'less than sign' (<) and 'greater than sign' (>). This statement will show the characters the way your printer will print them so that as you look through the program listing, you can see what the characters are. For example, if your printer prints a '>' as a %, then by looking at the lines 3550-3600 you will be able to see how a '>' is represented in the rest of the program listing.

PROFESSIONAL 8080/Z80 SOURCE CODE
COMMUNICATIONS..GRAPHICS..LANGUAGE SYS
HAWKEYE GRAFIX..213/348-7909
23914 MOBILE..CANOGA PARK..CA..91307

NEW PRODUCTS

Z-80 CPM Softcard For Apple

Microsoft Consumer Products has announced the Z-80 SoftCard™, a new plug-in processor for the Apple II that allows the Apple to run software written for Z-80 based computers.

In addition to the plug-in card, the SoftCard package includes the two most widely used microcomputer system software packages, the CP/M operating system from Digital Research and Microsoft Disk BASIC, ready to run on the Apple II.

The SoftCard allows the user to use either the Apple's 6502 processor or the Z-80 processor as needed to run a program. A command is used to switch between the two processors. The SoftCard is compatible with existing Apple software and peripherals.

Versions of Microsoft's FORTRAN, COBOL and BASIC Compiler for the Apple II with Z-80 SoftCard will be available separately. In addition, CP/M applications software written for Z-80 based computers can be converted to run on the Apple with minimal alteration.

The package includes the card, CP/M and BASIC on diskette and full documentation. Suggested retail price for the Z-80 SoftCard with Microsoft BASIC and CP/M is $349.00. For the name of the nearest dealer, contact Microsoft Consumer Products, 10800 Northeast Eighth, Suite 507, Bellevue, WA 98004. Telephone 206/454-1315.

S-100 Direct-Language Execution Processor

The DLX-10 from Alasda Computer Systems, is a direct-language execution processor for S-100 bus systems. It executes BASIC directly in high-speed hardware from five to ten times faster than 8080 systems or two to five times faster than Z-80 systems. The DLX-10 is a single-board computer that operates as an additional processor on the S-100 bus. It does not replace the CPU but functions as a separate, dedicated BASIC computer. It can boost an S-100 bus microcomputer system into the performance range of a minicomputer. The DLX-10 is recommended for scientific or business microcomputer systems which need increased speed or precision.

The DLX-10 is built of high-speed bipolar devices and uses a unique combination of hardware, firmware and state logic to provide extraordinary performance for small systems. As a separate processor, it runs independently of the main CPU and accesses memory as a DMA device. It runs in parallel to the existing CPU and accesses memory as a DMA device. It runs in parallel to the existing CPU and does not interfere with existing software. It has a stack architecture and utilizes high-speed on-board RAM to hold intermediate computations.

The DLX-10 supports full-feature BASICs with multi-dimensional arrays, string handling and print formatting. BASIC source language programs are first translated by software to relocatable BASIC stack-machine object code. This compact code is then executed by the DLX-10.

Typical one-byte operations are "floating multiply" and "string compare." Because the computer directly executes many of these time-consuming operations, the DLX-10 can execute many programs faster than equivalent machine-language code for the 8080 or Z80. In addition, the programs are more compact, thus permitting more efficient use of memory. Numbers are represented as floating-point decimal. The precision can be selected by the user to be from 2 digits to 20 digits plus exponent (10**±63). The accuracy may be set to different values for different programs.

Programmer productivity can be improved by using the DLX-10. Because efficient programs can be written is BASIC rather than assembly language, the labor cost of programming, debugging and maintaining programs is drastically reduced. The user does not need to retreat to assembly language to satisfy performance requirements.

Manufacturers of application-specific systems can use the DLX-10 in their systems to provide the power of a minicomputer in micro system. This optional power boost extends the range of price-performance options of microcomputer systems. A custon option permits the object code to be "scrambled" to provide built-in protection for proprietary software.

The DLX-10 is available assembled at $1250 (quantity 1). It comes with software to run Northstar BASIC or CBASIC. Delivery is 60-90 days ARO. The manufacturer is Alasda Computer Systems, 12759 Poway Road, Poway, CA 92064. (714) 748-8640.

S-100 Card Adds Sound Dimension

The NOISEMAKER sound board by Ackerman Digital Systems Inc, 110 North York Road, Suite 208, Elmhurst, Illinois 60126, generates sound effects under software control.

The board provides six tone generators, two noise sources, two envelope generators, and four 8-bit I/O ports which are software controlled using four switch selectable 8080 I/O addresses. A multitude of sound effects and noises may be created to add the sound dimension to graphics and computer games. An on-board audio amplifier (0.2 watts) and breadboard area allows easy interfacing of this product into any system. Three I/O ports, the amplifier output, and the supply voltage are brought out to a standard 44 pin plugboard connector.

The "Noisemaker" is available currently as a blank, solder-masked printed circuit board with the component layout silk screened in white. All connector contacts are gold plated. Included with the PCB is a parts list, schematic, construction notes and information on how to use the AY 3-8910.

P.C. Board and notes; $34.95. Add 50¢ for postage and handling.

TI9900-16 bit S-100 Microcomputer System

A powerful Scientific/Business Microcomputer based on the 16 bit TI9900 CPU and the S-100 bus is announced by Interface Technology, Box 745, College Park, MD 20740. This system can be viewed as a high end personal computer, or a small business/research system. Two versions are presented; both feature a 9900 16-bit CPU by Marinchip Systems, 32K bytes of memory, and two 8 inch floppy disks. Included as standard are a disk operating system, BASIC, word processor software, Editor, assembler, linker, and utilities. The scientific machine features PASCAL and a floating point package as standard. The business version substitutes extended commercial BASIC, General Ledger, Accounts Payable and Receivable, and Payroll. A Network Operating System for multi-user environments is available. The system is complete in one cabinet with power supply, fan, and power line filter. Scientific system: $4495. Business system: $4895. CRT and Printer, additional/faster memory, Network Operating System available at extra cost. Support of hard disk available.

ADVERTISER INDEX

Advertiser	Page
Ackerman Digital System	51
Computer Design Labs	11
Computerfest-80/M.A.C.C.	49
Computer Mart of New Jersey	57
Digital Graphic Systems	42
Electronic Control Technology	7
Freeport Utilities Co.	48
Godbout Electronics	60
Hayden Book Co.	27
Hawkeye Grafix	57
I.E.E.E. Computer Magazine	53
Ithaca Intersystems	5
Konan Corp.	23
Lifeboat Associates	8-9
Microcomputer Investor	58
Morrow Designs/Thinker Toys	2
North Star	21
Personal Computing-80	55
Potomac Micro-Magic	25
S-100 Inc.	28
S-100 Microsystems	59
Softronics	17
Tec-Mar	39
TEI Inc.	54

THE JOURNAL OF THE MICROCOMPUTER INVESTORS ASSOCIATION

THE **MICROCOMPUTER INVESTOR**

INVESTORS WHO USE MICROCOMPUTERS

FOR AN INFORMATION PACKET, SEND $2.00 TO:
MCIA, 902 ANDERSON DR., FREDERICKSBURG, VA 22401
BACK ISSUES ARE NOW AVAILABLE

THE ONLY MAGAZINE BY AND FOR S-100 SYSTEM USERS!

S-100 MICROSYSTEMS™

At last there is a magazine written exclusively for S-100 system users. No other publication is devoted to supporting S-100 system users. No longer will you have to hunt through other magazines for an occasional S-100, CP/M* or PASCAL article. Now find it all in one publication. Find it in **S-100 MICROSYSTEMS.**

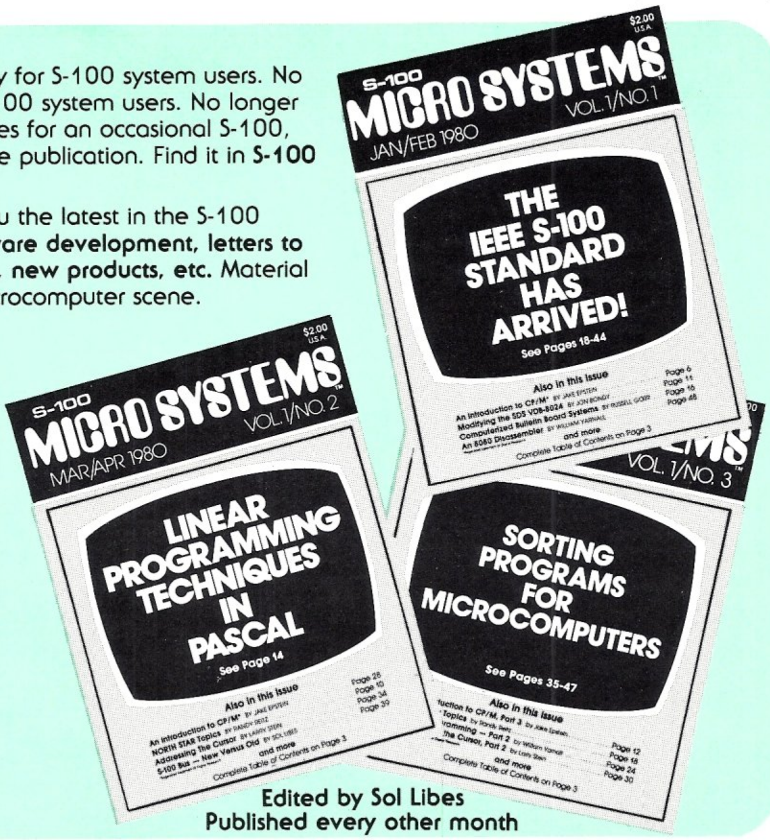
Every issue of **S-100 MICROSYSTEMS** brings you the latest in the S-100 world. **Articles on applications, tutorials, software development, letters to the editor, newsletter columns, book reviews, new products, etc.** Material to keep you on top of the ever changing microcomputer scene.

SOFTWARE
CP/M*
Assembler
BASIC
PASCAL
applications
and lots more

SYSTEMS
Cromemco
North Star
IMSAI
SOL
Polymorphics
and lots more

HARDWARE
8 bit & 16 bit CPUs
interfacing
hardware mods
bulletin board systems
multiprocessors
and lots more

*TMK Digital Research



Edited by Sol Libes
Published every other month

S-100 MICROSYSTEMS SUBSCRIPTION RATES
(effective May 1, 1980)

USA	Canada	Europe/ So. Amer.	Other Foreign
ONE YEAR (6 issues)			
$ 9.50	$12.50	$21.50	$23.50
TWO YEARS (12 issues)			
$17.00	$20.00	$41.00	$45.00
THREE YEARS (18 issues)			
$23.00	$26.00	$59.50	$65.00

Europe, So. America, and other foreign sent air mail.
Payments must be in U.S. Funds.
BACK ISSUES
U.S.A., $2.50 each*, 3 for $6*
Foreign, add $1/issue
Subscriptions start with next mailing

Send me ☐ 6 ☐ 12 ☐ 18 Issues of S-100 MICROSYSTEMS™

Send me the following back issues ($2.50 each; 3 for $7):
☐ 1-1 Jan/Feb 1980
☐ 1-2 Mar/Apr 1980
☐ 1-3 May/Jun 1980

Total enclosed $ ____________

Name ____________

Address ____________

City ____________

State ____________ Zip ____________

CPU Make: ____________ Disk System Make: ____________

IT'S HERE . . . AND CPU BOARDS WILL NEVER BE THE SAME AGAIN.

The **CompuPro Dual Processor Board** gives true 16 bit power with an 8 bit bus, is downward compatible with the vast library of 8080 software, is upward compatible with hardware and software not yet developed, accesses 16 Megabytes of memory, meets all IEEE S-100 bus specifications, runs 8085 and 8086 code **in your existing mainframe** as well as Microsoft 8086 BASIC and Sorcim PASCAL/M™, and runs at 5 MHz for speed as well as power.

The **Dual Processor Board** has two CPUs that "talk" to each other; the 8088 CPU is an 8 bit bus version of the 8086 16 bit CPU, while the 8085 is an advanced 8 bit CPU that can run existing software such as CP/M.

Amazingly enough, all this flexibility won't break your budget: **introductory prices** are **$385 unkit**, **$495 assembled**, and **$595** qualified under the **Certified System Component** high-reliability program. **Don't need 16 bit power yet?** Then select our **single processor** version which does not inlcude the 8088 for **$235 unkit**, **$325 assembled**, and **$425 CSC**.

The **Dual Processor Board** is built to the same stringent standards that have established our leadership in S-100 system components . . . and starting June 1st, you'll be able to plug it into your mainframe to experience computing power that, until now, you could only dream about. CPU boards will truly never be the same again.

THINKING GRAPHICS? THINK "SPECTRUM" COLOR GRAPHICS BOARD.

The **CompuPro Spectrum board** is actually three sophisticated products in one: a fast (5 MHz), low power 8K × 8 IEEE compatible memory board with extended addressing; an I/O board with full duplex bidirectional parallel port (including latched data along with attention, enable, and strobe bits), capable of interfacing with keyboards, joysticks, or similar parallel peripherals; and a 6847-based graphics generator board that can display all 64 ASCII characters. Put these together, and you've got 10 modes of operation — from alphanumeric/semi-graphics in 8 colors to ultra-dense 256 × 192 full graphics. Includes a 75 Ohm RS-170 compatible line output and video output for use with FCC approved video modulators. **Introductory pricing is $339 unkit**, **$399 assembled**, and **$449** qualified under the high-reliability **CSC** program. **Looking for graphics software?** Sublogic's **2D Universal Graphics Interpreter** (normally $35) is yours for **$25 with the purchase of a Spectrum board in any configuration.**

No longer must you settle for B&W graphics, or stripped down color graphics boards; starting June 1st, you'll be able to plug one of the industry's most cost-effective and full-feature color graphics boards into your S-100 system.

OUTSTANDING COMPUTER PRODUCTS:

MEMORY

All boards are static, run in 5 MHz systems, meet all IEEE standards, include a 1 year limited warranty, and feature low power consumption. Choose from **unkit** (sockets, bypass caps pre-soldered in place), **assembled**, or boards qualified under our high-reliability **Certified System Component (CSC)** program (200 hour burn-in, 8 MHz operation, and extremely low power consumption.

Name	Buss & Notes	Unkit	Assm	CSC
8K Econoram* IIA	S-100	**$169**	**$189**	**$239**
16K Econoram XIV	S-100 (1)	**$299**	**$349**	**$429**
16K Econoram X-16	S-100	**$329**	**$379**	**$479**
16K Econoram XIIIA-16	S-100 (2)	**$349**	**$419**	**$519**
16K Econoram XV-16	H8 (3)	**$339**	**$399**	n/a
24K Econoram XIIIA-24	S-100 (2)	**$479**	**$539**	**$649**
32K Econoram X-32	S-100	**$599**	**$689**	**$789**
32K Econoram XIIIA-32	S-100 (2)	**$649**	**$729**	**$849**
32K Econoram XV-32	H8 (3)	**$649**	**$749**	n/a
32K Econoram XI	SBC/BLC	n/a	n/a	**$1050**

* **Econoram** is a trademark of **Bill Godbout Electronics.**

(1) Extended addressing (24 address lines). Addressable on 4K boundaries.
(2) Compatible with all bank select systems (Cromemco, Alpha Micro, Etc.); addressable on 4K boundaries.
(3) Bank select option for implementing memory systems greater than 64K.

SPECIAL PRICE! TRS-80* -I or -II MEMORY EXPANSION CHIP SET: $69!

We've done it again . . . 8 low power, 250 ns 16K dynamic RAMs at a trendsetting price. Don't be impressed with fancy packaging or four color ads; our chip set gives all the performance you want at a price you can afford. **Offer good while supplies last.** Add $3 for TRS-80 compatible DIP shunts and complete installation instructions.

*TRS-80 is a trademark of the Tandy Corporation.

TERMS: Cal res add tax. Allow 5% for shipping, excess refunded. VISA®/Mastercharge® call our 24 hour order desk at **(415) 562-0636**. COD OK with street address for UPS. Sale prices good through cover month of magazine; other prices are subject to change without notice.

MOTHERBOARDS

Meet or exceed all IEEE S-100 specs; with true active termination, grounded Faraday shield, edge connectors for all slots. Unkits have edge connectors and termination resistors pre-soldered in place for easy assembly.

6 slot: $89 unkit, $129 assm.
12 slot: $129 unkit, $169 assm.
19 slot: $174 unkit, $214 assm.

GODBOUT COMPUTER BOX $289 desktop, $329 rack mount. With quiet fan, dual AC outlets and fuseholder, line filter, card guide, etc.

S-100 2708 EROM BOARD $85 unkit. 4 independently addressable 4K blocks. Includes support chips and manual, but no EROMs.

S-100 ACTIVE TERMINATOR BOARD $34.50 kit. Plugs into older, unterminated motherboards to improve performance.

S-100 MEMORY MANAGER BOARD $59 unkit, $85 assm, $100 CSC. Adds bank select and extended addressing to older S-100 machines to dramatically increase the available memory space.

2S "INTERFACER I" S-100 I/O BOARD $199 unkit, $249 assm, $324 CSC. Dual RS-232 ports with full handshake. On-board crystal timebase, hardware UARTS, much more.

3P PLUS S "INTERFACER II" I/O BOARD $199 unkit, $249 assm, $324 CSC. Includes 1 channel of serial I/O (RS-232 with full handshake), along with 3 full duplex parallel ports plus a separate status port.

PASCAL/M™ + MEMORY SPECIAL PASCAL can give a microcomputer with CP/M more power than many minis. You can buy our totally standard **Wirth PASCAL/M™ 8" diskette**, with **manual** and **Wirth's definitive book** on PASCAL, FOR **$150 with the purchase of any memory board.** Specify Z-80 or 8080/8085 version. PASCAL/M™ available separately for $350.

Z-80A CPU BOARD $225 unkit, $295 assm, $395 CSC. Full compliance with IEEE S-100 bus standards, provision for adding two EROMs, on-board fully maskable interrupts, power on jump and clear, selectable automatic wait state insertion, IEEE extended addressing, much more.

CompuPro™ from GODBOUT ELECTRONICS

Bldg. 725, Oakland Airport, CA 94614

Many of these products are stocked by finer computer stores world-wide, or write us for further information if there's no dealer in your area.