68' MICRO JOURNAL

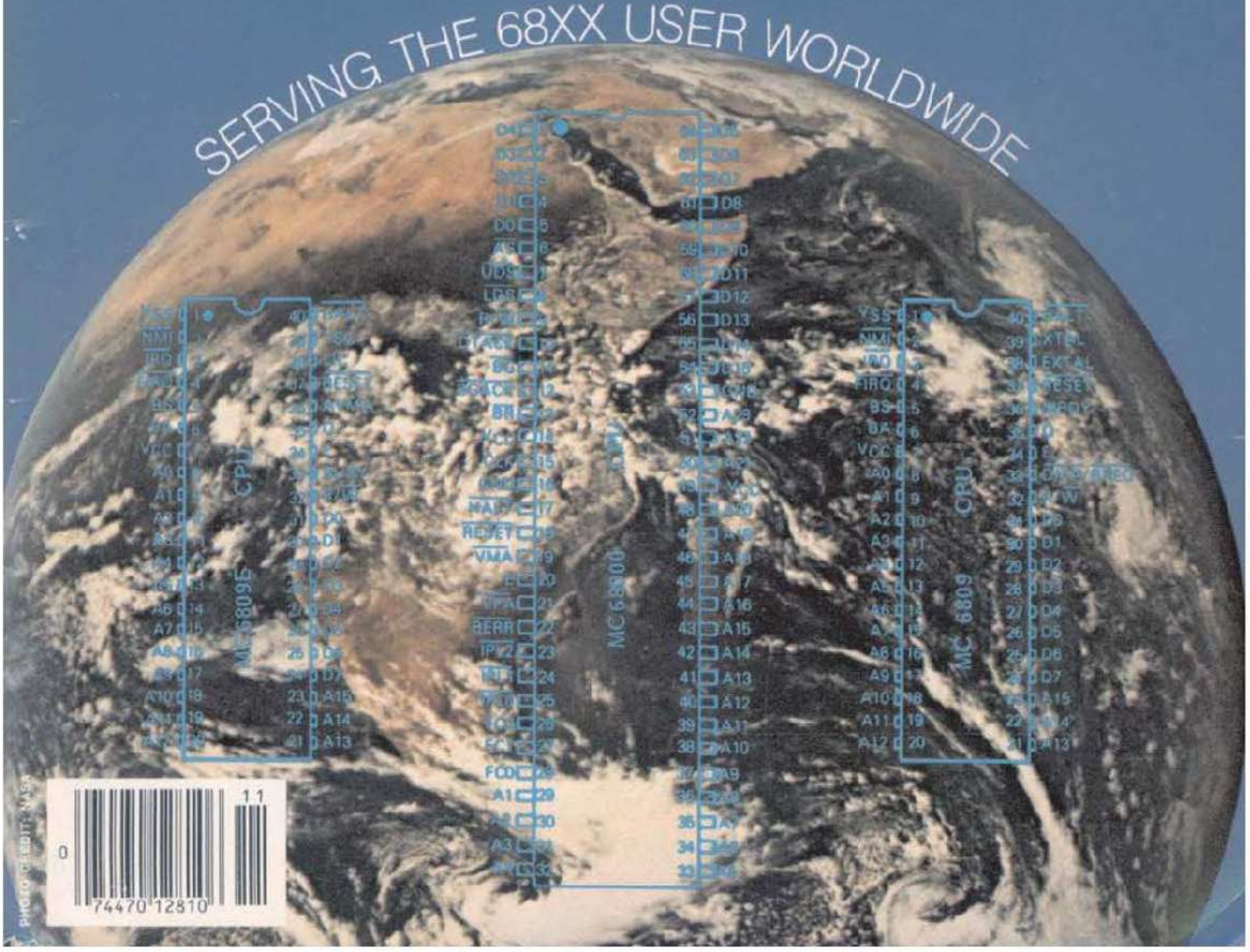
Australia A $ 4.75 | New Zealand NZ $ 6.50
Singapore S $ 9.45 | Hong Kong H $23.50
Malaysia M $ 9.45 | Sweden 30:-SEK

$2.95 USA

ANA and the 68000 p.24
"C" User Notes p.19
OS/9 User Notes p.16
UniFLEX User Notes p.29
FLEX User Notes p.7
ISAM p.35
Basic OS/9 p.27

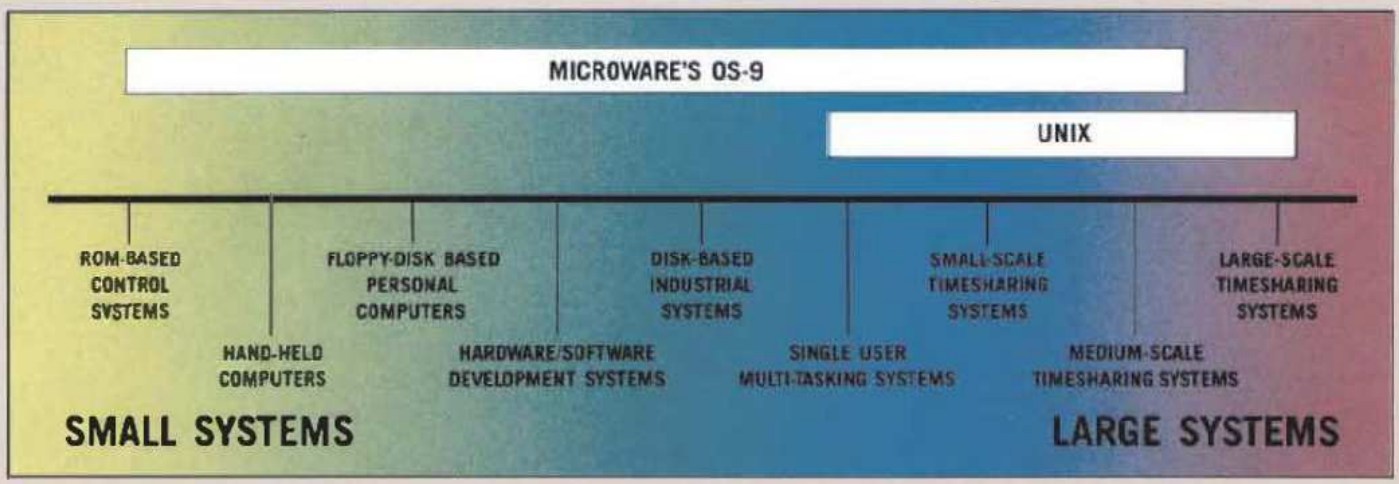
VOLUME VII ISSUE XI • Devoted to the 68XX User • November 1985

"Small Computers Doing Big Things"



0 74470 12810 11

Only Microware's OS-9 Operating System Covers the Entire 68000 Spectrum



Is complicated software and expensive hardware keeping you back from Unix? Look into OS-9, the operating system from Microware that gives 68000 systems a Unix-style environment with much less overhead and complexity.

OS-9 is versatile, inexpensive, and delivers outstanding performance on any size system. The OS-9 executive is much smaller and far more efficient than Unix because it's written in fast, compact assembly language, making it ideal for critical real-time applications. OS-9 can run on a broad range of 8 to 32 bit systems based on the 68000 or 6809 family MPUs from ROM-based industrial controllers up to large multiuser systems.

OS-9'S OUTSTANDING C COMPILER IS YOUR BRIDGE TO UNIX

Microware's C compiler technology is another OS-9 advantage. The compiler produces extremely fast, compact, and ROMable code. You can easily develop and port system or application software back and forth to standard Unix systems. Cross-compiler versions for VAX and PDP-11 make coordinated Unix/OS-9 software development a pleasure.

Key OS-9 Features At A Glance

- Compact (16K) ROMable executive written in assembly language
- User "shell" and complete utility set written in C
- C-source code level compatibility with Unix
- Full Multitasking/multiuser capabilities
- Modular design - extremely easy to adapt, modify, or expand
- Unix-type tree structured file system
- Rugged "crash-proof" file structure with record locking
- Works well with floppy disk or ROM-based systems
- Uses hardware or software memory management
- High performance C, Pascal, Basic and Cobol compilers

SUPPORT FOR MODULAR SOFTWARE — AN OS-9 EXCLUSIVE

Comprehensive support for modular software puts OS-9 a generation ahead of other operating systems. It multiplies programmer productivity and memory efficiency. Application software can be built from individually testable software modules including standard "library" modules. The modular structure lets you customize and reconfigure OS-9 for specific hardware easily and quickly.

A SYSTEM WITH A PROVEN TRACK RECORD

Once an underground classic, OS-9 is now a solid hit. Since 1980 OS-9 has been ported to over a hundred 6809 and 68000 systems under license to some of the biggest names in the business. OS-9 has been imbedded in numerous consumer, industrial, and OEM products, and is supported by many independent software suppliers.

microware® OS-9

AUTHORIZED MICROWARE DISTRIBUTORS

AUSTRALIA
MICROPROCESSOR CONSULTANTS
16 Bandera Avenue
Wagga Wagga 2650
NSW Australia
phone: 616-931-2331

ENGLAND
VIVAWAY LTD.
36-38 John Street
Luton, Bedfordshire
England LU1 2JE
phone: (0582) 423425
telex: 825115

JAPAN
MICROWARE JAPAN LTD.
3-8-9 Baraki, Ichikawa
Chiba 272-01, Japan
phone: 0473 (26) 4493
telex: 781-299-3122

SWEDEN
MICROMASTER SCANDINAVIAN AB
Sd Persgatan 7
Box 1309
S751-43 Uppsala
Sweden
phone: 018-138595
telex: 76129

SWITZERLAND
ELSOFT AG
Bankstrasse 9
5432 Neuenhof
Switzerland
phone: (41) 056-862724
telex: 57136

USA
MICROWARE SYSTEMS CORPORATION
1866 NW 114th Street
Des Moines, Iowa 50322
USA
phone: 515-224-1929
telex: 910-520-2535
FAX: 515-224-1352

WEST GERMANY
DR. KEIL GMBH
Porphystrasse 15
D-6905 Schriesheim
West Germany
phone: (0 62 03) 67 41
telex: 465025

OS-9 is a trademark of Microware and Motorola. Unix is a trademark of Bell Labs.

ALL SYSTEMS INCLUDE:

* The CLASSY CHASSIS with a ferro-resonant, constant voltage power supply that provides + 8 volts at 30 Amps, + 16 volts at 5 Amps, and − 16 volts at 5 Amps.
* Gold plated bus connectors.
* Double density DMA floppy disk controllers.
* Complete hardware and software documentation.
* Necessary cables, filler plates.

YOU CAN EXPAND YOUR SYSTEM WITH:

MASS STORAGE

Dual 8" DSDD Floppies, Cabinet & Power Supply **$1698.88**
20MB Streamer ..... (under development)
1.6MB Dual Speed Floppy ..... (under development)

MEMORY

#67 Static RAM-64K NMOS (6809 Only) **$349.67**
#64 Static RAM-64K CMOS w/battery (6809 Only) **$396.64**
#72 256K CMOS Static RAM w/battery **$998.72**
#31 16 Socket PROM/ROM/RAM Board (6809 only) **$268.31**

INTELLIGENT I/O PROCESSOR BOARDS

significantly reduce systems overhead by handling routine I/O functions; freeing the host CPU for running user programs. This improves overall system performance and allows user terminals to be run at up to 19.2K baud. For use with GMX III and 020 systems.

#113 Port Serial-30 Pin (OS9) **$498.11**
#14 3 Port Serial-30 Pin (UniFLEX) **$498.14**
#12 Parallel-50 Pin (UniFLEX-020) **$538.12**
#13 4 Port Serial-50 Pin (OS9 & UniFLEX-020) **$618.13**

I/O BOARDS (6809 SYSTEMS ONLY)

#41 Serial, 1 Port **$89.41**
#43 Serial, 2 Port **$128.43**
#46 Serial, 8 Port (OS9/FLEX only) **$318.46**
#42 Parallel, 2 Port **$88.42**
#44 Parallel, 2 Port (Centronics pinout) **$128.44**
#45 Parallel, 8 Port (OS9/FLEX only) **$198.45**

CABLES FOR I/O BOARDS—SPECIFY BOARD

#95 Cable sets (1 needed per port) **$24.95**
#51 Cent. B.P. Cable for #12 & #44 ..... **$34.51**
#53 Cent. Cable Set **$36.53**

OTHER BOARDS

#66 Prototyping Board-50 Pin **$56.66**
#33 Prototyping Board-30 Pin **$38.33**
Windrush EPROM Programmer S30 (OS9/FLEX 6809 only) **$545.00**

CONTACT GIMIX FOR FURTHER DETAILS ON THESE AND OTHER BOARDS AND OPTIONS.

EXPORT MODELS: ADD $30 FOR 50Hz. POWER SUPPLIES.
ALL PRICES ARE F.O.B. CHICAGO.

GIMIX DOES NOT GUARANTEE PERFORMANCE OF ANY GIMIX SYSTEMS, BOARDS OR SOFTWARE WHEN USED WITH OTHER MANUFACTURERS PRODUCT.

GIMIX, Inc. reserves the right to change pricing, terms, and products specifications at any time without further notice.

GIMIX 2MHZ 6809 SYSTEMS — GMX 68020 SYSTEMS

Operating Systems Included	#49 OS9 GMX I/ and FLEX	#39 OS9 GMX II/ and FLEX	#79 OS9 GMX III/ and FLEX	#39 UniFLEX	#89 UniFLEX III	#020 OS9/68020	#020 UniFLEX VM
CPU included	#05	#05	GMX III	#05	GMX III	GMX 020	GMX 020 + MMU
Serial Ports Included	2	2	3 Intelligent	2	3 Intelligent	3 Intelligent	3 Intelligent
High Speed Static RAM	64KB	256KB	256KB	256KB	256KB	512KB	1 Megabyte
PRICES OF SYSTEMS WITH: Dual 80 Track DSDD Drives	$2998.49	$3398.39	$4898.79	N/A	N/A	N/A	N/A
19MB Hard Disk and one 80 track Floppy Disk	$5598.49	$5998.39	$7798.79	$5998.39	$8098.89	$11,680.20	$13,680.20
72 MB Hard Disk and one 80 track	$7598.49	$7998.39	$9798.79	$7998.39	$10,098.89	$13,680.20	$15,680.20
a 72MB + a 6MB removable pack hard disk and one 80 track floppy	$9098.49	$9498.30	N/A	$9498.39	N/A	N/A	N/A
a 72MB + a 12MB removable pack hard disk and one 80 track floppy	N/A	N/A	$11,298.79	N/A	$11,598.89	$15,180.20	$17,180.20
GMX 6809 OS9/ FLEX SYSTEMS SOFTWARE OS9 + Editor, Assembler, Debugger	GMX I Included	GMX II Included	GMX III Included				
FLEX	Included	Included	Included				
GMXBUG Monitor	Included	Included	Included				
Basic 09, RunB (OS9)	Included	Included	Included				
RMS (OS9)	Included	Included	Included				
DD (OS9)	Included	Included	Included				
VDisk for FLEX	N/A	Included	Included				
RAMDisk for OS9	N/A	$125 option	Included				
0-FLEX	N/A	$250 option	Included				
Support ROM	N/A	N/A	Included				
Hardware CRC	N/A	N/A	Included				

Available: Wide variety of languages and other software for use with either OS-9 or FLEX.

All GIMIX versions of OS9 can read and write RS color computer format OS9 disks, as well as the Microware/GIMIX standard format.

All OS9/FLEX systems allow you to software select either operating system.

TO ORDER BY MAIL: SEND CHECK OR MONEY ORDER OR USE YOUR VISA OR MASTER CHARGE. Please allow 3 weeks for personal checks to clear. U.S. orders add $5 handling if order is under $200.00. Foreign orders add $10 handling if order is under $200.00. Foreign orders over $200.00 will be shipped via Emery Air Freight COLLECT, and we will charge no handling. All orders must be prepaid in U.S. funds. Please note that foreign checks have been taking about 8 weeks for collection so we would advise wiring money, or checks drawn on a bank account in the U.S. Our bank is the Continental Illinois National Bank of Chicago, 231 S. LaSalle Street, Chicago, IL 60693, account number 73-32033.

BASIC-09 and OS-9 are trademarks of Microware Systems Corp. and MOTOROLA, Inc. FLEX and UniFLEX are trademarks of Technical Systems Consultants, Inc. GIMIX, GHOST, GMX, CLASSY CHASSIS, are trademarks of GIMIX, Inc.

GIMIX INC.
1337 WEST 37th PLACE
CHICAGO, ILLINOIS 60609
(312) 927-5510 • TWX 910-221-4055

© 1985 GIMIX, INC. 8-85

'68' MICRO JOURNAL

Portions of the text for '68' Micro Journal were prepared using the following furnished Hard/Software:

COMPUTERS - HARDWARE

Southwest Technical Products
219 W. Rhapsody
San Antonio, TX 78216
S09 - 5/8 DMF Disk - CDS1 - 8212W - Sprint 3 Printer

GIMIX Inc.
1337 West 37th Place
Chicago, IL 60609
Super Mainframe - OS9 - FLEX - Assorted Hardware

EDITORS - WORD PROCESSORS

Technical Systems Consultants, Inc.
111 Providence Road
Chapel Hill, NC 27514
FLEX - Editor - Text Processor

Great Plains Computer Co., Inc.
PO Box 916
Idaho Falls, ID 83401
Stylograph - Mail Merge - Spell

Editorial Staff

Don Williams Sr.	Publisher
Larry E. Williams	Executive Editor
Tom E. Williams	Production Editor
Robert L. Nay	Technical Editor

Administrative Staff

Mary Robertson	Office Manager
Penny Williams	Subscriptions
Christine Kocher	Accounting

Contributing Editors

Ron Anderson	Norm Commo
Peter Dibble	William E. Fisher
Dr. Theo Elbert	Carl Mann
Dr. E. M. Pass	Ron Voigts
Phillip Lucido	

Special Technical Projects

Clay Abrams K6AEP
Tom Hunt

CONTENTS

Vol.VII, Issue XI November 85

FLEX USER Notes.....	7	Anderson
OS9 USER Notes.....	16	Dibble
C USER Notes.....	19	Pass
ADA And The 68000.....	24	Elbert
Basic OS-9.....	27	Voigts
UniFLEX User Notes.....	29	Lewis
ISAM.....	35	Condon
Ramblings & Such.....	43	DMW
Bit Bucket.....	45	
Classified Advertising.....	53	

Send All Correspondence To:

Computer Publishing Center
68' Micro Journal
5900 Cassandra Smith Rd.
Hixson, Tn. 37343

Phone (615) 842-4600 or Telex 558 414 PVT BTH

Copyrighted 1985 by Computer Publishing Inc.

68' Micro Journal is published 12 times a year by Computer Publishing Inc. Second Class Postage Paid ISSN 0194-5025 at Hixson, Tn. and additional entries. Postmaster: send form 3597 to 68' Micro Journal, POB 849 Hixson, Tn. 37343.

Subscription Rates

1 Year $24.50 U.S.A., Canada & Mexico Add $9.50 a Year. Other Foreign Add $12 a Year for Surface, Airmail Add $48 a Year. Must be in U.S. currency)

Items or Articles For Publication

Articles submitted for publication should include authors name, address, telephone number and date. Articles should be on either 5 or 8 inch disk in STYLOGRAPH or TSC Editor format with 3.5 inch Column width. All disks will be returned. Articles submitted on paper should be 4-5 inches in width (including Source Listings) for proper reductions. Please Use A Dark Ribbon!! No Blue Ink!! Single space on 8X11 bond or better grade paper. No hand written articles accepted. Disks should be in FLEX2 6800 or FLEX9 6809 any version or OS-9 any version.

The following TSC Text Processor commands ONLY should be used: .sp space, .pp paragraph, .fi fill and .nf no fill. Also please do not format within the text with multiple spaces. We will enter the rest at time of editing.

All STYLOGRAPH commands are acceptable except .pg page command. We print edited text files in continous text form.

Letters To The Editor

All letters to the editor should comply with the above requirements and must be signed. Letters of "gripes" as well as "praise" are solicited. We reserve the right to reject any submission for lack of "good taste" and we reserve the right to define "good taste".

Advertising Rates

Commercial advertisers please contact 68' Micro Journal advertising department for current rate sheet and requirements.

Classified Advertising

All classified ads must be non-commercial. Minimum of $9.50 for first 20 words and .45 per word after 20. All classifieds must be paid in advance. No classified ads accepted over the phone.

..HEAR YE.....HEAR

OS-9™ User Notes

By: Peter Dibble
As Published in 68 Micro Journal

The publishers of 68 Micro Journal are proud to announce the publication of Peter Dibbles **OS9 USER NOTES.**

Information for the BEGINNER to the PRO,
Regular or CoCo OS9

Using OS9
HELP, HINTS, PROBLEMS, REVIEWS, SUGGESTIONS, COMPLAINTS, OS9 STANDARDS, Generating a New Bootstrap, Building a new System Disk, OS9 Users Group, etc.

Program interfacing to OS9
DEVICE DESCRIPTORS, DIRECTORIES, "FORKS", PROTECTION, "SUSPEND STATE", "PIPES", "INPUT/OUTPUT SYSTEM", etc.

Programming Languages
Assembly Language Programs and Interfacing; Basic09, C, Pascal, and Cobol reviews, programs, and uses; etc.

Disks Include
No typing all the Source Listings in. Source Code and, where applicable, assembled or compiled **Operating Programs.** The Source and the Discussions in the Columns can be used "as is", or as a "Starting Point" for developing **your OWN** more powerful Programs. Programs sometimes use multiple Languages such as a short Assembly Language Routine for reading a Directory, which is then "piped" to a Basic09 Routine for output formatting, etc.

BOOK Typeset -- w/ Source Listings (3-Hole Punched; 8 x 11) **$9.95**

Deluxe Binder - - - - - - - - - - - $5.50

All Source Listings on Disk
1 8" SS, SD Disk - - - - $14.95
2 5" SS, DD Disks - - - $24.95

Shipping and Handling; $3.50 per Book, $2.50 per Disk Set
* All Currency in U.S. Dollars Foreign Orders Add $4.50 S/H
If paying by check - Please allow 4-6 weeks delivery

Continually Updated In 68 Micro Journal Monthly

VISA

Computer Publishing Inc.
5900 Cassandra Smith Rd.
Hixson, TN. 37343

MasterCard

(615) 842-4600
Telex 558 414 PVT BTH

TM - OS9 is a trademark of Microware Systems Corp. and Motorola Inc.
TM - 68' Micro Journal is a trademark of Computer Publishing Inc.

MUSTANG-020 Super SBC™

***

We Proudly Announce the MUSTANG-020 Super SBC*
"The one with the REAL KICK!"
Only from DATA-COMP

This is the NCC, world beater GMX SBC, in a super configuration. Data-Comp has mated it to a power plus power supply/stylish cabinet and your choice of floppy and/or hard disk drives. Available in several different configurations. (1) single board. (2) single board and regulators for your cabinet or mainframe and power supply. (3) single board - power supply and cabinet - your disk drives. (4) single board - power supply/cabinet - our drives configured to your specs, and ready to run. OS9 68K will be available as well as several other popular operating systems. Also all the popular OS9 68K software and Motorola 020-BUG will be available at a very reasonable price.

Note: We will include the Motorola 020-BUG at no additional charge. This item alone sells for $500.00. This allows direct coding of 68020 advanced codes. 020-BUG is required for our version of OS9. Please bear in mind, this system is the one others are compared to.

This system is the state-of-the-art star-ship. It runs rings around any other 68XXX SBC, and most mainframes. The speed and expanded RAM make this the "best buy" by a far stretch! A true multi-user, multi-tasking computer. So far advanced that even the experts don't call it a micro. Compared to the others, it isn't even a "horse race." And the price is certainly right. You can bet on this one!

So, will it be **Turtle** or **Thoroughbred?**

Deliveries expected to begin November 1985.

CALL NOW FOR PRICE AND DELIVERY RESERVATIONS:

CHARGE

* Mustang-020 is trademark of Data-Comp-CPI

DATA-COMP
5900 Cassandra Smith Rd.
Hixson, TN 37343

VISA

SHIPPING
USA ADD 2%
FOREIGN ADD 5%
MIN. $2.50

☎ **(615)842-4600**
For Ordering
TELEX 558 414 PVT BTH

MUSTANG -020 Super SBC™

Mustang 020 Features

- 12.5 MHz MC68020 full 32-bit wide path Processor
  - 32-bit wide non-multiplexed data & address buses
  - On-chip instruction cache
  - Object-code compatibl with earlier M68000 family processors (68000/68008/68010)
  - Enhanced instruction set - Coprocessor interface
- Optional 68881 Floating point Coprocessor (12.5 MHz)
  - Direct extension of 68020 instruction set
  - Full support of IEEE P754, draft 10.0
  - Transcendentals and other math functions
- 2 Megabytes of RAM (512K x 32-bit organization)
- Up to 256K bytes of EPROM (64K X 32-bits)
  - Uses four 2764, 27128, 27256, or 27512 EPROMs
- 4 Asynchronous serial I/O ports (2 x MC68681 DUART)
  - Software programmable baud rates to 19.2K
  - Standard RS-232 interface
  - Optional network interface on one port
- Buffered 8-bit Parallel I/O Port (1/2 MC68230)
  - Centronics-type parallel printer pinout
  - May also be used as parallel input port
- Expansion Connector for Additional I/O Devices
  - 16-bit data path
  - 256 byte address space
  - 2 interrupt inputs
  - Clock and Control Signals
- Time-of- ay Clock/Calendar w/battery backup
- Controller for up to Two 5 1/4" Floppy Disk Drives
  - Single or double sided
  - Single or double density
  - 48 or 96 tracks per inch (40/80 Track)
- Mounts Directly to a Standard 5 1/4" Disk Drive
- SASI Interface for Intelligent Hard Disk Controllers
- Programmable Periodic Interrupt Generator
  - For time-slicing and real-time applications
  - Interrupt rates from microseconds to seconds
  - Highly Accurate timebase (5 PPM)
- 5-bit sense switch, readable by the processor
- Hardware single-step capability

And many other advanced features not found on other systems priced much higher.

Straight From The Horse's Mouth

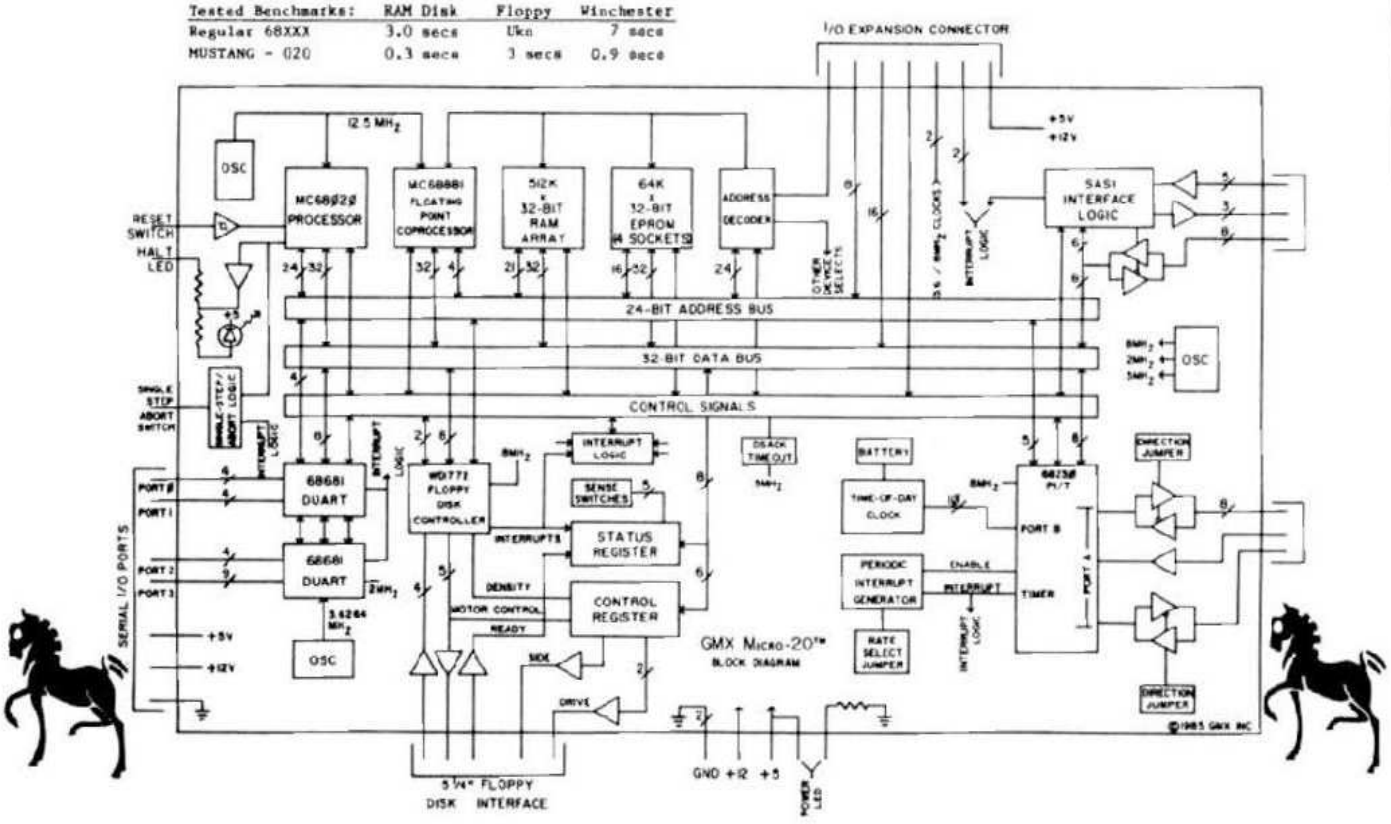
The Clear Winner

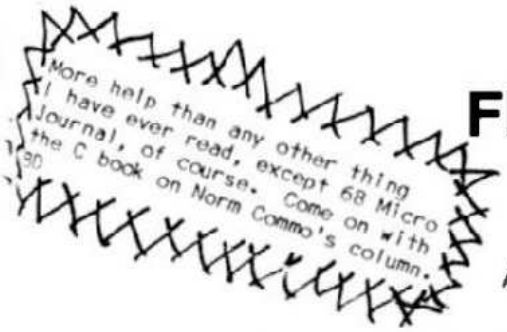
☎ (615)842-4600

DATA-COMP

Load Basic09:

Tested Benchmarks:	RAM Disk	Floppy	Winchester
Regular 68XXX	3.0 secs	Ukn	7 secs
MUSTANG - 020	0.3 secs	3 secs	0.9 secs





FLEX™ USER NOTES
THE 6800-6809 BOOK

By: Ronald W. Anderson
As published in 68 MICRO JOURNAL™

The publishers of 68 MICRO JOURNAL are proud to announce the publication of Ron Anderson's **FLEX USER NOTES**, in book form. This popular monthly column has been a regular feature in 68 MICRO JOURNAL SINCE 1979. It has earned the respect of thousands of 68 MICRO JOURNAL readers over the years. In fact, Ron's column has been described as the 'Bible' for 68XX users, by some of the world's leading microprocessor professionals. Now all his columns are being published, in whole, as the most needed and popular 68XX book available. Over the years Ron's column has been one of the most popular in 68 MICRO JOURNAL. And of course 68 MICRO JOURNAL is the most popular 68XX magazine published.

As a SPECIAL BONUS all the source listing in the book will be available on disk for the low price of: FLEX™ format only — 5" $12.95 — 8" $16.95 plus $2.50 shipping and handling, if ordered with the book. If ordered separately the price of the disks will be: 5" $17.95 — 8" $19.95 plus $2.50 shipping and handling.

Listed below are a few of the **TEXT** files included in the book and on diskette.

All TEXT files in the book are on the disks.

LOGO.C1	File load program to offset memory — ASM PIC
MEMOVE.C1	Memory move program — ASM PIC
DUMP.C1	Printer dump program — uses LOGO — ASM PIC
SUBTEST.C1	Simulation of 6800 code to 6809, show differences — ASM
TERMEM.C2	Modem input to disk (or other port input to disk) — ASM
M.C2	Output a file to modem (or another port) — ASM
PRINT.C3	Parallel (enhanced) printer driver — ASM
MODEM.C2	TTL output to CRT and modem (or other port) — ASM
SCIPKG.C1	Scientific math routines — PASCAL
U.C4	Mini-monitor, disk resident, many useful functions — ASM
PRINT.C4	Parallel printer driver, without PFLAG — ASM
SET.C5	Set printer modes — ASM
SETBAS1.C5	Set printer modes — A-BASIC
	(And many more)

**Over 30 TEXT files included in ASM (assembler) — PASCAL — PIC (position independent code) TSC BASIC-C, etc.

NOTE: .C1,.C2, etc. = Chapter 1, Chapter 2, etc.

This will be a limited run and we cannot guarantee that supplies will last long. Order now for early delivery.

Foreign Orders Add $4.50 S/H

Softcover — Large Format

Book only: **$7.95** + $2.50 S/H

With disk: 5" **$20.90** + $2.50 S/H

With disk: 8" **$22.90** + $2.50 S/H

See your local S50 dealer/bookstore or order direct from:

Computer Publishing Inc.
5900 Cassandra Smith Rd.
Hixson, TN 37343
(615) 842-4601

TELEX 558 414 PVT BTH

"FLEX is a trademark of Technical Systems Consultants

FLEX User Notes

Ronald W. Anderson
3540 Sturbridge Court
Ann Arbor, Mi 48105

PAT

A

New

Text Editor

Don Williams asked me to depart from the usual monthly column format for this month to tell you about the text editor that I have just finished writing. He intends to publish portions of the manual for PAT here, and the introduction goes into some of the reasons I wrote PAT, so I won't go into all that here, only to have it repeated. Let me just give you a little more in depth history of the project. As you may well know, I have been a software "critic" for a long time. Most of the suppliers have indicated that their initial reactions to my reviews have been negative, but the majority of them seem to have grown to appreciate my comments on their new offerings, and generally they have included many of the features that I called "ought to"s when I wrote reviews.

Of course, anyone who writes a lot of text, and also programs for a living, uses an editor a great deal of the time. I can't think of any one piece of software that I run more than an editor, by a very wide margin. I have always thought therefore, that an editor ought to be made as convenient for the user as is possible. I've written negative reviews with regard to editors that made the user take care of some situation that I thought the editor ought to handle automatically. For example, many screen editors require you to "extend" the buffer by telling them to insert 20 blank lines, or by going to the end of the last line of the present text, putting the cursor at the right end, and typing a CR to get to the next line. Obviously if the editor software is smart enough to know that it is at the end of the file, it ought to be able to extend the file automatically.

An editor shouldn't be filled with a bunch of what I call gingerbread features and then make it difficult for you to do something you do three or four times on a screen of text. For example, one of our text editors has limited "spread sheet" capabilities. It can total a line or a column of figures and insert that total for you automatically. While I am not knocking that capability, the same editor requires that you type several keystrokes to indent the start of a paragraph. If you edit that paragraph and reformat it, the initial indent evaporates and you have to put it back by repeating the same multiple key strokes.

So anyway, I started thinking about a screen editor last Winter. I did some little preliminary programming to see how hard it would be to do the cursor motions on a single screen of text. About then I got off on another project for several months, but about April, I decided to start in earnest on the project. As each feature was implemented, I tested what I had by using it to edit text. Of course that was difficult until I had enough going so I could write to a file and save my edited text, but I was soon at that point. Some of the early code proved to be messy, and it was refined as I continued to add more features. About the time most of the editor was working, I started looking for the places where most of the time was spent, and replaced some of the longest and slowest loops with assembler code, which is quite easy to do in PL/9, the language I had chosen for the project.

Well, here we are some 4 months later. I won't say this was an easy project. There were times when bills didn't get paid on time, the lawn mowing was neglected, and nights when I had 4 or 5 hours sleep. (Why stop debugging at 1 AM when things are going well, right?) So, I've put my "efforts" where my mouth is. PAT is to be sold at a special introductory price. While I did it mainly as a challenge, I won't be unhappy to see a little return for my 400 or so hours that are invested in this project.

Should you purchase PAT, I would appreciate hearing from you. I am not planning to add many more features to it, since doing so would begin to limit the size of the edit buffer more than I would like. I do plan to review the code critically and try to simplify it and reduce the size of

the object code. I want to know about any bugs that are discovered as soon as they are found. I will respond imm iately to any bug reports, either with a quick patch or an update. While I have used PAT in its present form for some time, I won't pretend that I don't expect a few more bugs to surface. I try to test software that I have written for both normal and unexpected conditions, but I certainly haven't thought of them all.

Editor's Note: PAT and JUST (the processor) make an excellent combination. However, there will be many of you who will ask - "Why another editor or processor?" Well, the answer is simple, from my viewpoint. price, source included and upward compatibility with the presently available and coming 68XXX systems.

To me the inclusion of source, presently in PL9, easily converted to Whimiscal, Pascal and C (a C version may be included soon) and other HLLs, make this an attractive product for many of our reader/use s. Not since the early days of TSC products has source been included. This means that you only have to purchase the product one time, that is if you are programming in a high level language (HLL). If not, you sure can learn a lot studying source, most of us did!

This means saving in hundreds of dolla s. PLEASE remember, Ron and I (S.E. Media, distribution licensee) trust you not to abuse the copyright of the source and code. Also, we have intentionally priced it low enough to TRUST that if you want a copy you will purchase it rather than "borrow or steal" it. Ron has a lot of "sweat hours" involved and I assured him that you we e a very honest bunch. Thanks for ot letting me down.

DMW

INTRODUCTION

Editor? Why on Earth.... Aren't there enough Editors around now? Those are all valid questions, particularly in these days of limited markets for software for FLEX and the 6809. First of all, if another editor that I have been using for several years now had ever become available for the FLEX 6809 group, this effort would probably never have been started. I am referring to PIE, the very nice editor written by Tom Crosley of SoftWest, that all the Apple owne s have had available for a long time. I was given a "pre release" copy of PIE with a Xerox copy of a hand marked manual for PIE on the Apple several years ago. I've always said that if it ever became available I would have to buy a couple of copies to reflect my liking for it and my use of it. Due to circumstances, however, PIE for FLEX is not to be.

My disappointment in the situation, increased by rumors and promises of "someday" led me to start writing something more or less equivalent to it. I have used some number greater than ten editors on a number of different computers, and PIE had struck me as being better as a general all round tool than any of the others. There are TEXT editors and there are PROGRAM editors and one would think them to be so different and special that no one could write one that would serve both purposes. That is a lot of nonsense. There is no reason that one editor can't serve both purposes. One of my primary design goals has been to keep it simple and avoid frills that only a few users would use now and then. I have concentrated on making the most often used functions as simple to use as possible.

The main goal in the design of PAT was not just to copy an editor that I like very well, but to try to incorporate some improvements as I went along. Let me outline what I believe to be the best features of PAT.

1. Clean Screen Layout - Pat puts nothing on the terminal screen but one status line at the very bottom. If the terminal supports reverse video the line is displayed in that mode. If not, the line is displayed normally. The status line has line and column indicators and mode indicators as well as a display of remaining memory in the edit buffer. This same status line is used whenever user input must be done in the course of editing a file.

2. Minimum Keystrokes - PAT is a SC REEN editor, not a line editor. The main tool of a screen editor is the cursor, and the cursor is used to advantage by all good screen editors to mark start and end of blocks of text that are going to be copied, moved or deleted, and for many other functions. In short, if it can be done with the cursor, you won't be asked to type in information.

3. Speed - PAT will allow you to type successive commands that move the screen around in the text, and it will abort rewriting the screen to do another command. For example, in many screen editors, if you bump the top of the text window with CURSOR UP motions, the screen will move up one line in the text (or the text will move down in the screen). Some editors will rewrite the screen completely three times if you bump the top of the screen three times in a row. PAT will interrupt the screen refresh essentially as soon as it starts in order to handle another command. When the commands stop coming, the screen will be rewritten.

4. Ease of maintenance - This is a feature for the author, but it means that errors or bugs that are found by users can be fixed quickly. PAT in the original release is written in the PL/9 language. My intention was and is to write it eventually in "C". However, I know of no "C" compiler that will compile code in less than 5 or 6 times the time required to compile the equivalent program in PL/9. I had little choice but to do it in PL/9 first and plan on a translation later. After finishing the coding in PL/9 and some preliminary debugging, I rewrote some sub procedures in Assembler. As of now. just at 2.2% of the

final object code represents Assembler code. This small part of the software accounts for most of the time taken to do the various functions, and the optimization represented by that small amount of assembler code is responsible for a speed increase of about 5 times. The representation of the procedures in a high level language makes the program listing much more understandable to others, as well as to the author after the passage of some time. As a matter of interest, The code is approximately 2300 lines, about 150 of which are assembler code. The source listing is 44 pages, and the object code is 15.9K.

5. Big Buffer - The main edit buffer is 29,000 bytes. This means that you can edit over 7 pages of single spaced typewritten material within the buffer. Files larger than that can be edited by writing a portion of the buffer to the output file and reading more from the input file until the whole file has been processed.

6. Good File Handling - The input file is never renamed until the output file has been written. At that point the input file is renamed .BAK, the old backup is deleted (if it exists) and the new file is renamed from .SCR to the extension of the input file. Power failures never result in confusion since the input file was not renamed .BAK at the start of the edit session. You may delete the .BAK file before starting an edit session, but you must choose to do so. If you do not, there will be a time when three copies of your file are present on the disk at the same time (though briefly). In addition to the normal file functions, PAT allows reading a "side file" into the file being edited (at the cursor position of course). It also allows writing a previously marked block of data to a file. A late addition is the ability to access FLEX utilities while running PAT. These must be used with caution, however. See the expanded instructions later in this manual.

7. Easy Setup - Want to set tab stops every three columns? Just type ESC 3 ^T. Tab positions will be set a d displayed on the top line of the screen. When you are sure that is what you want, hit a key and you are back to normal editing. If you want to set the tabs to specific and unevenly spaced columns, you type ESC ^T and bump the cursor along the top line with the space bar until you find the column you want to be a tab stop, then type any other key. Continue across the screen until all tabs are set a d hit RETURN.

8. Reasonable operating modes - In the normal OVERLAY mode, the terminal screen is analogous to a piece of paper in a typewriter. You don't have to enter CR'S to get down three lines below the current end of the text, just bump the cursor down there and type. If you want to type something in the middle of a blank screen just go ahead. PAT takes care of blank lines and spaces out to the text.

In "INSERT" mode characters are inserted at the cursor and the text to the right is pushed along until it hits the line length limit. Both of these modes are operative along with either of the two modes that follow.

PMODE - (Paragraph MODE) - is used primarily for text. In this mode you never have to think about a CR. Text fills a line and when a word overflows that line it is moved to the start of the next line, even if you are inserting text in the middle of a paragraph. This mode is close to the one referred to as "Document Mode" in Wordstar terminology.

AMODE - (Auto indent MODE) - In this mode, when the cursor moves to a new line, it aligns itself with the first printable character on the line above it. This mode is extremely useful for programming and typing outlines. There are tab and backtab keys that also aid in this mode.

9. Normal Text File - Some editors generate a "peculiar" text file containing "Soft Carriage Returns", "Hard Spaces" etc. Some use no carriage returns within a paragraph so that each paragraph is one very long line. Though such editors generally work fine within their own "system", the files that they produce are hard to LIST to the terminal or edit with other editors. PAT produces "standard" text files. In FLEX that means that each line as you see it on the screen while you edit, will be saved with the standard (CR) line terminator. The FLEX LIST utility and others assume this to be the form of the input file, and therefore all the FLEX utilities work with PAT files with o special handling.

10. Paragraph Formatting - Features are available to format paragraphs to the currently set margins. One of the configuration parameters that you may include in your terminal configuration file is the paragraph indent desired in formatting paragraphs. A paragraph may be formatted with or without justification. If justification is on, spaces will be added to each line to make the right margin even. Regardless of the status of Justify, words will be rearranged within the paragraph to fit the maximum number that will fit on each line, and the first line will be indented. Generally after formatting a paragraph the cursor will be found on the first character of the next paragraph so the process can be repeated. The format operation is done a paragraph at a time so you can view the text and skip formatting of tables etc.

A paragraph that has been formatted with justification may be reformatted without it, and all the added spaces will be removed. Formatting a paragraph will automatically put two spaces after a period that is followed by one or more spaces. Only one of those spaces will be "padable". That is, the justification routine won't add a space to both of the spaces after a line making the spacing very much wider than the other spaces.

11. Long Lines - PAT allows lines as long as 127 characters. The screen will automatically shift as you move off the end in the direction of the remainder of the text. All functions of PAT work with the screen shift.

There is one more feature that should be mentioned in the introduction. Did you ever assemble or compile a program and get a message such as ERROR #202 in line #237?

Some editors do not associate a number with each line. Though PAT doesn't display numbers, on every line, the line indicator on the status line does display the current cursor line number. Each line is associated with a file line number. In the case of the above error message, you load the file into PAT and type ESC 237 ^G and before you can blink that line is about 1/3 of the way from the top of the screen and the cursor is at the beginning of it.

It should also be made clear what PAT is NOT. PAT is not a full text formatter. There is no provision for paging or header and footer formatting of pages. It is assumed that the user will embed word processing commands in the text and run the output through one of the several text processors that are available. The formatting that is available, of course is only useful if the printer to be used is of the non proportional spacing type. For use with a text formatter set up for proportional spacing, use only the Format mode without justify. Commands to be interpreted by the text formatter as start of paragraph may be used in the text. For purposes of formatting a paragraph, the paragraph ends if the first character of the next line is a CR (i.e. a blank line), if it is one of the symbols usually used to indicate a formatter command (comma, period, or colon), or if it is a space (leading space on a new line). This last is included so the user may format paragraphs without blank lines between them, starting a new paragraph with one or more spaces.

This effort to date represents some 400 hours of programming. I would hope that purchasers would respect the copyright and urge their friends to purchase their own copy.

DESCRIPTION OF KEY FUNCTIONS

General Notes

In the following descriptions the up arrow ^ indicates that the Control key (marked CTRL or possibly ^ on some terminals) should be held down while the key is operated. For example ^E means to hold the control key and the E key down at the same time. On most terminals it does not matter whether you use capital E or lower case e.

Control keys that change the contents of the screen or window will cause the window to be completely or partially rewritten as necessary to get the correct information in the window. Note that if you repeat a key that changes the information in the window, the screen rewrite Aborts instantly. That is, if you have text above the window, and type ^E five times rapidly, the screen will move up 5 lines and then it will be rewritten. This is a very significant feature of PAT. It saves a great deal of time for the user.

Some of the commands later use the ESC (escape) key as a prefix. for example, ESC 5^I means that you type the ESC key, the number 5 and then ^I. (this inserts 5 blank lines above the cursor). Generally the ESC key puts the cursor on the status line at the bottom of the screen and you type characters there, terminating the input with a control character or Carriage Return.

CURSOR CONTROL

^E - Cursor Up

This key moves the cursor up one line. It does what it has to do to accomplish this. If the cursor has room on the screen, It simply moves up one line. If it is at the top of the screen, it "pushes" the screen up in the text so that the next line is visible. If the cursor is at the top of the file, it does nothing in response to this command.

^C - Cursor Down

This key moves the cursor down one line. It behaves like the cursor up key with a few exceptions. If the last line of the file is at the bottom of the screen and the cursor is moved down past the end of the screen, the text moves up 15 lines (for a 24 line monitor) placing blank lines below the text. Moving the cursor down below this screen will move the text up and insert more blank lines. It is not pos ble to move the cursor so far down that the screen goes blank. The last line of the text will remain. If you must skip more than a screenful of blank lines, temporarily put something on one of them, add your text below the break and delete the temporary characters.

^S - Cursor Left

This key moves the cursor one space to the left without changing the text in any way. The cursor can not be moved past the left margin with this key.

^F - cursor Right

This key moves the cursor to the right. Presently PAT does not have horizontal scrolling, and the cursor cannot be moved past the last column on the screen.

^D - Cursor Home / Bottom

This key moves the cursor to the upper left corner of the text window if it is not already there. If it is there, it moves to the lower left corner of the window. The text in the window does not move. This key is handy for getting to the top or bottom of the screen in a hurry

^] - Cursor Begin / End

This key moves the cursor to the end of a line if it is within the text, and to the beginning if it is at or past the end of the text on the line. On some terminals the right bracket (]) is represented as the upper case of the left bracket. In this case, you will probably have to use CTRL SHIFT along with the "bracket key" to get this function to work.

^G - Right Tab

This key moves the cursor to the

next tab position right. Tab positions are defined either by the text already on the line or by the tab settings (see ESC n^T below). If the editor is in the PMODE, tabs within the text move the cursor to the first character of the next word to the right. If you tab past the end of the text, tabs revert to the selected tab columns. In AMODE, tabs always move to the marked tab columns. This is really the best of both worlds. If you are in a paragraph, you can tab three words right by typing ^G three times. It becomes almost automatic. Now go to a blank line and tab to the set columns for entering a table. Tab Right will not tab past the end of the screen.

^A - Tab Left

This key tabs left to the first character of the previous word when the cursor is within text and the editor is in PMODE. When outside of the text, the cursor moves to the previous tab set position. See remarks about Tab Right above. Tab Left is very useful in the AMODE for programming or typing an outline. Tab Left will not tab past the beginning of a screen.

SCREEN CONTROL - DESCRIPTION

^R - Screen Up

This key moves the screen up in the text by one line less than the number of lines that are displayed on the screen. That is, the first line on the screen becomes the last line after ^R. The cursor stays exactly where it was before the ^R. This feature is useful both in program editing and text editing. You can go up two or three screens to refer to some previously entered text (or program) and go back down the same number of screens, and the cursor will be right at the spot where you were editing before you went away from that spot. Obviously you can't screen up beyond the start of the file.

^V - Screen Down

This key moves the window down in the text by one line less than the number displayed in the window. That is, the bottom line before the ^V becomes the top line.

^U - Current Line Up

This key moves the present cursor line (with the cursor) to the top of the screen. (actually it moves the window down in the text but psychologically, you are moving the cursor line to the top of the window, so it fits here on the top row that is associated with UP commands. This is useful for fitting as much of a particular paragraph or program procedure on the screen as possible. Put the cursor at the first line and move it to the top of the screen.

^Y - Screen Up 1/3

This key moves the screen up in the text by 1/3 screen. If the cursor was high on the screen it will remain with the line where it was before the move. Of course, if the cursor line before the move falls off the bottom of the screen, the cursor will stick at the bottom line.

^N - Screen Down 1/3

This Key moves the screen down by 1/3 screen. The cursor sticks with the line if it stays on the screen. This is useful for proofreading the text. Read along and follow your current line with the cursor. When you get to the bottom, do a couple of ^N's and the cursor will be 2/3 of the way up the screen and at the line you have just read. Now continue reading and bumping the cursor.

^T - Top of File

This key moves the top of the file into the window, with the first line of the file at the first line of the window. Repeated use of this key will do nothing further.

^B - To Bottom

This key moves the bottom of the file to the line 1/3 of the way down the screen and puts the cursor on the next line below the present end of the file. Successive ^B will not change the situation.

EDITING FUNCTION KEYS

^H - Backspace

This key has two different actions depending on whether you have the INSert mode on or off. When the Insert mode is off (Overtype Mode) the cursor moves back one space and replaces the character at that position with a space. In the INSert mode, the cursor moves back a space, deletes the character at that space, and brings the rest of the text to the right of the cursor back with it. If this sounds confusing, the Backspace key operation follows what the cursor does when entering text in the respective modes. The action is quite what you would expect in these modes.

NOTE: The backspace WILL backspace past the left margin and go to the end of the text on the previous line. You will appreciate this when you want to fix the last character you typed before a Carriage Return. You can back up through a line and into the next one above if you need to. Also note that it is not possible to leave a line with several spaces on it. PAT automatically 'cleans up' a line when the cursor leaves it. If there are spaces followed by a CR, PAT will remove all the spaces and the CR will be in the first position on the line. When the cursor moves TO a blank line, PAT puts spaces out to the cursor so that if you type something there, it will be preceded by enough spaces to put it there when it is printed. It is not possible to leave trailing spaces on a line with PAT.

^I - Insert a Line

A blank line will be inserted where the cursor presently is located, and the text on that line will be moved down. The last line on the screen will move off to the bigger edit buffer. It is not lost.

^K - Delete (Kill) a Line

The line on which the cursor is located will be deleted. Text below it will

move up to fill the space made by deleting the line. The last line of the display will be moved up from the big edit buffer. If you are at the bottom of the file, the last line will be a blank line. Insert and Kill may be repeated rapidly to insert or delete a number of lines.

Note that ^K has another function. When a line is deleted, it is not lost, but goes to a Delete Buffer (only ONE line will fit in the delete buffer and it will always be the last one deleted). This fact may be used to advantage in two situations. First, if you accidentally delete the wrong line, or one too many lines, yo can "restore" the line with the command that follows this one below. Secondly, this is a QUICK method to move a line. You delete the line, put the cursor on the line below where you want to insert it and type ^O.

^O - Output Delete Buffer

This command will put the contents of the delete buffer at the line where the cursor is. The line that was there will move down to make room for the inserted line. This is just like the ^I function except that the inserted line contains text. If you have not used ^K to delete anything, you will get a blank line.

^L - Push Line to Delete Buffer

This command lets you copy a line. It puts the line in the Delete Buffer but it doesn't delete it. Now you can move the cursor and use ^O to duplicate the line.

Note that the above are handy to handle a line or two. There are other ways to mark a large block of text, delete it, move it, or copy it. These are discussed later.

ESC ^S - Split a Line at Cursor

Put the cursor on the first character you want to move to the next line and type ESC^S. The character at the cursor and everything to the right will be put on a new line below the present one, and the text below will be moved down a line. The cursor remains at the Split point so you can continue to type there. This is handy for inserting more text on re-editing already written material.

ESC ^J - Join Lines

Put the cursor anywhere on a line of text. ^J will join the next line to the current one, changing the trailing CR to a space, IF THERE IS ENOUGH ROOM on the line. If not, you will get an error message. All error messages remain on the status line until you press any key, after which you can resume editing. If you want to join lines but they won't fit, you can split the second line at a point where what is left will fit...

ESC nn^I - Insert Multiple Lines

Rather than type ^I 12 times, you can insert 12 blank lines by typing ESC (cursor will move to the status line following the prompt ARG: where you type in 12^I. The number of blank lines that you specify will be inserted.

ESC nn^K - Delete (Kill) Multiple Lines

Works like the above command except that it can delete multiple lines. As a precaution, the status line shows the prompt DELETE 12 LINES? (or whatever number you have specified). Answer with a Y, either upper or lower case, and the lines will be deleted. An N or other response will QUIT the function. If you delete more lines than are left in the file, all of the file below (and including) the cursor line, will be deleted, i.e. to the bottom of the file.

FORMATTING COMMANDS

ESC nn^L - Set Line Length (for PMODE)

Use this to set the line length at which words typed in will wrap to the next line when in PMODE. The line length appears on the status line at the bottom of the screen, as L=65.

ESC ^P - PMODE On/Off

This command toggles the PMODE on and off. The default mode is with neither PMODE nor AMODE on. In that default mode, you must use a CR to get to the next line. When PMODE is on, the word PMODE appears on the status line.

ESC ^A - AMODE On/Off

When this command is used, AMODE is toggled on and off. If PMODE is on, it is turned off by AMODE and vice versa. The utility of AMODE is that when you type a CR, the cursor moves to the column under the first printable character in the line above. That is, it goes to the same indent level as the previous line. This is useful for programming or outlines.

ESC nn^G - Goto Line nn

When you use an assembler or a compiler, generally error messages include the file line number on which the error occurs. You can load the program source file and Goto the line in question.

ESC n^T - Set Regular Tabs

This command sets the tabs every N columns (including the first). It displays the tabs on the top line of the screen as the last digit of the column number if the column has no tab set, and as a vertical bar if a tab is set. If you want irregular tab columns, type ESC^T and the tab line will be displayed at the top of the screen. The cursor will be at the first column of the second line and you can move it along with the space bar, typing any other character where you want a tab set. When you are done, type CR and the top line will rewrite to show your tab positions. When you have verified them type any key to return to editing.

ESC ^C - Toggle Caps Mode

You may not need this command. If you do, you will be very glad for its

existence. Some terminals do not have a CAPS LOCK key so that all letters a to z are converted to upper case but the remainder of the keyboard remains the same. Caps Mode does that with software. If your terminal doesn't have a Caps Lock key or it has a (UGH) Shift Lock, you will use this function.

ESC ^Q - Toggle Justify Mode

^Q - Format current paragrapn

PAT has paragraph formatting capability via the ^Q command. If Justify mode is on, the letters JU appear on the status line. Put the cursor anywhere on the first line of a paragraph and type ^Q and the paragraph will be formatted with full justification (that is, the left and right margins will be even. The first line will be indented by the PARINDENT constant which defaults to 5, but is configurable via the Terminal configure file. Spaces will be added within each line to make them all the same length. The line length will be one less than the current right margin setting. The cursor will generally be left at the first line of the next paragraph so that this command may be repeated. There is no GLOBAL format command because the program cannot always tell the start of a paragraph from something else such as a table, a heading, etc. The end of a paragraph is detected by any of the following:

1. A line starting with a comma, period, or colon. These are used by most text formatting software to indicate a formatting command.

2. A blank line (a CR in the first column)

3. A line starting with a space.

The second case should be obvious. If you end a paragraph with a CR, and then skip a line, the program has no trouble detecting the end of the paragraph. The third case is for those who don't want blank lines between paragraphs. When you type the text, simply indent one space at the start of a new paragraph. See the caution below regarding this method of delimiting paragraphs.

If the Justify mode is off when you issue the Format command, the words will be moved around, lines split and rejoined, so that all the lines contain all the words that will fit within the presently set margins. The only difference from the case of Justify mode being on is that no spaces will be inserted between words. Regardless of mode, Format command will put two spaces after any period that is followed by a space (primarily for ends of sentences). If a paragraph has been formatted with Justify on, formatting it with Justify off will remove all the extra spaces that were added to make the lines come out even. Some users of PAT, particularly if they have a text formatter program that works with a printer that has proportional letter spacing, will want to have the formatter do the justification. However, the capability to maximize words on each line will be appreciated after re-editing a section of text.

If you have one of the many printers that provide only constant pitch printing (10 per inch or 12 per inch), you may use PAT to pre-format paragraphs so you can get an idea of how long the text will be. Note, however, that PAT has no provision for page headers, footers, or page numbers. Such functions will have to be handled by a text formatter.

SEARCH AND REPLACE

Search and replace functions are easy to use. First you enter the search string. As soon as it is terminated, the search begins, and if the string is found in the text, it will be on the screen with the cursor at the first character. Then the Replace string is entered. When it is terminated, the string will be replaced. If you decide that the "found" string is not the one to be replaced, you can repeat the search and look for the next occurrence of it. There is also a "GLOBAL" replace mode in which all occurrences from the cursor to the end of the file are replaced.

ESC TEXT^Z - Enter Search String

Type ESC and the cursor goes to the status line. Now type the text string you want to find. Spaces may be included in the text string for the search. This is useful when searching for such things as ESCCASE 1 ^Z. Space after the 1 distinguishes this case so that it doesn't "find" CASE 11 CASE 12 CASE 123 etc. The control Z terminates the search string and causes the search to start. Search is case sensitive. A search for "school" will not find "School". (A case insensitive search mode may be added in a later release).

EXC TEXT^X - Enters Replace String

This string is entered in the same way as the search string. The cursor should be at the found occurrence of the search string since it will be replaced after you enter this string. Note that the replace string is terminated with ^X.

^Z - Repeat the Search

A search string must be defined or you will get an error message "Nothing to search for". With this command, you can search through a text or program for all the places where the string occurs and decide at each one what you want to do with it. NOTE: Searches all proceed DOWN from the current cursor position. If you want to look at all occurrences of a particular string, be sure you start at the TOP of the file.

^X - Repeat Replace

This command will use the previously defined replace string. The cursor must not be moved from the place where it was after completion of a SUCCESSFUL search for the search string. If you type this command and these conditions are fulfilled, the current occurrence of the search string will be replaced with the current Replace String.

^W - Global Replace

After finding and replacing the first

occurrence of a string, and without moving the cursor, type ^W and all occurrences from the cursor position to the bottom of the file will be found and replaced. The current line indicator on the screen will update as each occurrence is found. You can stop the global replace by typing ^C. If you let it go to completion, the screen will update with the last replacement on the screen and marked by the cursor.

BLOCK FUNCTIONS

Blocks are important in most editors. You "mark" a block of text, and then you can do one of several things. PAT will let you DELETE a marked block, COPY it to somewhere else in the text, (and leave it where it was), MOVE it to somewhere else (removing it from its original position) WRITE it to a file, or UNMARK it if you change your mind. You can also READ a BLOCK from a file into your text at a selected place. All of the block functions are terminated by a CR. Blocks are defined as one or more whole lines.

ESC T<CR> - Mark Top of file

Place the cursor on the first line that is to be included in the block and type ESC T<CR>. The number of that line will appear on the status line (displacing the MEM NNNNN temporarily) as T=NNN.

ESC B<CR> - Mark Bottom of file

Put the cursor at the last line in the block and type ESC B<CR>. The reminder B=NNN will appear after the T=NNN on the status line. These prompts will remain there as long as the block is marked.

ESC D<CR> - Delete Marked Block

The position of the cursor is unimportant when you issue this command. The marked block is deleted. It disappears from the text that is being edited and the screen is rewritten without it. Of course, since the block disappears, it becomes unmarked and the MEM NNNNN status reappears.

EXC M<CR> - Move Marked Block

Place the cursor at the line where you want the block to be inserted (it will be inserted above the line where the cursor is placed. Again, since the block disappears from its original location, it will be unmarked.

ESC C<CR> - Copy Marked Block

Place the cursor at the line where you want a copy of the marked block. It will be inserted above the cursor line, and also remain in its original location. Since the original block is still there, it will remain marked and you can copy it again any number of times. When you are done copying, just use the command described next, to unmark the block.

ESC U<CR> - Unmark Block

This command does just what you would think. If you make a mistake in marking a block or have copied it and want to unmark the original, this is the command to use. When the block is unmarked, the MEM NNNNN display returns to the status line.

ESC >FILENAME<CR> - Write Block to File

The arrow pointing at the filename is the key for this command. It opens a file named FILENAME (default extension .TXT and default drive, working drive). Then it copies the marked block to that file and closes it. Since the block is not deleted from the file being edited, it is left marked. After copying it to a file you can unmark it or delete it.

ESC<FILENAME<CR> - Read Block From File

The arrow pointing FROM the filename indicates that the file is to be read IN to the text being edited, AT the cursor (again above the current cursor line). The same defaults apply as above.

EXIT FUNCTIONS

ESC ^^ - Exit and save backup file

This is the normal exit function. It does just what is described earlier in this manual under the heading FILES. That is, it deletes an OLD backup file if one is present, Renames the input file (if any) to .BAK and writes and closes an output file with the extension .SCR. It then renames the .SCR file to the extension of the input file or to that specified for the output file. Note that this command is ESC CTRL up-arrow. The up-arrow on some terminals is typed by using SHIFT 6. If the up-arrow is a "capital 6" on your terminal, you may or may not have to operate the SHIFT key in addition to the CTRL key to get a control up-arrow. Some terminals shift automatically since ^6 is not a valid ASCII code. Others do not.

EXC Q^^ - Quit edit

This mode simply abandons the current edit. It will always leave the input file and backup file as they were before you started the edit. If the file is too big to fit in the edit buffer, part of it may have been written to a .SCR file. Using this command in that case will leave a partial output file with the extension .SCR. If you have done a lot of work on the early part of the file and then fouled up a later part, the partial output file may be of some value. Generally it is better if you have already written to an output file, to use the normal exit mode.

BIG FILES

ESC N^^ - New text

This command requires that the cursor be positioned with some thought. It causes all of the file above the cursor to be written to an output file and if the input file is not empty, more of it is read into memory. If the remainder of the file will fit in the 26000 character limit, it is all read in. If not, the New command may be repeated later after writing more text to the output file. This is a way to edit very large files, though in general it is better

to split files into "chapters" or "libraries" so that each file will fit the edit buffer. Most text formatters allow you to specify a starting page number, and you can print your files out in sequence, each time specifying the next page number as the first for the next file.

FLEX UTILITIES

ESC +FLEX COMMAND<CR>

FLEX commands can be executed while editing files. Of course you must not use a utility that uses memory outside of the utility command space, such as NEWDISK (FORMAT) or COPY. There are certain other utilities that will cause problems as well. Because of the way files are handled by PAT, the output file is opened at the start of the edit session. The Standard TSC LIST utility closes the file that it has listed by calling the FLEX FLSCLS routine which closes ALL files including the output file for the text you are editing. You will be able to list another file just fine, but when you try to save the current file you will get a SYSTEM FILE ERROR, which is a cryptic way of saying that the file has already been closed. A LIST1 utility has been supplied with PAT that closes the listed file in the normal manner in all cases so that it will not close your output file.

Should you accidentally use the original LIST utility supplied with FLEX, and get an error message while using it, all is not lost. Do NOT try to write the output file with a normal exit. Mark the first and last lines of the file (make the whole file the marked block) and write a sidefile with a name different from your intended output file. If you forget and try to exit normally, and get the SYSTEM FILE ERROR message, you can still mark the file and write a side file, but you will get one screenful of your file repeated in the side file. You can then re-edit the file and delete the redundant screen of text. Test any utility you might think you will use while running PAT, by using it while editing a non-critical file to see if it causes any problems with the output file, or if it overwrites any of PAT or the edit buffer.

The main reason for including this features is to allow you to use a CAT, DIR, FILES utility to get a list of files on the disk, in case you want to read from a sidefile but have forgotten its name. Use any other FLEX utility with extreme caution.

EDITOR'S NOTE - FROM DON WILLIAMS

WRAP-UP

Well, there you have it. A good look at PAT, an editor for all occasions. This item will soon join the hundreds of other FINE products from South East MEDIA. We are in final "Beta" testing now. When it is ready for sale (soon - we hope) it will be included in the South East MEDIA catalog, in 68 Micro Journal. Also, there will be a limited time special for the JUST text processor and PAT combined. This will be an excellent combination package for those of you not willing to part with an arm and a leg for another editor/processor. And the really sweet part is, THE SOURCE IS INCLUDED!!

The JUST package will have one binary object JUST and three (3), Yep, three (3) different source files of JUST. Fact is the JUST package with all three (3) is now being shipped. IT IS READY! And comprises the following:

1. JUST in PL9 source (s me as the object file included).

2. JUSTSC in C source - THIS ONE CAN BE COMPILED ON OS9! Also it can be compiled under FLEX, naturally. JUSTSC is a slightly different format, includes several additional printer commands and format features. It is so structured to accept many TSC style text files (using commands. i.e. - pp sp ce, etc) This is a real winner if you have text files in the TSC format, or are used to them. Of course being in C, you can change practically anything to anything else you might desire. And it is patently simple to add practically any other useful command or feature you might desire. This is true for both the PL9 and C source files included, at no additional charge!

And finally there is the same JUST as in the PL9 version, only in C source. Again, simple to modify to your hearts (and pocketbooks) delight.

Everyone who has purchased JUST and commented to me later about it, RAVE!

So, if you do not have PL9 or C, never mind, the JUST included on the disk runs under FLEX. But you OS9 folks will have to compile it for now. However, I will furnish a FREE copy, or refund the purchase price to the first one who compiles JUST, using our C source, into an executable OS9 module. Naturally, it should run "bug free"!

Also we have an "update policy" for those of you with older versions of JUST. Return your original disk to South East MEDIA along with the sum of $19.50 + $2.50 H/S, and we will update to the latest everything, including the new source files, and instructions.

I am doing all I can to bring you, though South East MEDIA, good and especially low priced software. But I need your cooperation. If you know of some product, that you desire, that is not in our catalog or stock, please let me know. Remember this, it is South East MEDIA and DATA-COMP that helps support 68 MICRO JOURNAL. It is your magazine - so I leave it all to you. Between the two, South East MEDIA or DATA-COMP, practically anything you ever heard of for the S50 bus or the 68XX(X) is available, and most of it from stock. And now with our new service department, I have done about all I can do. If we are to continue, we need your support. WE SUPPORT WHAT WE SELL - AND EVEN IF THE MANUFACTURER GOES "BELLY-UP", WE WILL STILL BE THERE! Today, that is a very valuable consideration. So remember the next time you get ready to buy - give them a call. It just might be the smartest and safest thing you could do!

DMW

OS-9 User Notes

Peter Dibble
19 Fountain Street
Rochester, NY 14620

Mixed Stuff

Level Two Processes Again

Some readers write me fairly regularly. This demonstrates their generous natures. I like letters very much, but I may be the world's worst at writing back. This month one of these friends sent a letter which I wish had been sent to me care of this magazine. For a while Don Williams and I hoped we could get a question and answer thing going with questions from letters and answers from me. It may still happen, but the supply of questions hasn't been there. I got two really meaty questions from this letter. How you debug a Level Two device driver? How does OS-9 use the task register in a DAT.

I can answer the question about debugging device drivers quickly. Unless you have hardware debugging tools it is nasty work. Microware used to have something called (I think) an over-the-top debugger. It was able to do some primitive debugging from outside the OS-9 environment. From what I hear, now that Microware has in-circuit emulators they don't maintain the over-the-top debugger anymore. It has been reported dead.

I have two tricks I use when debugging OS-9 system code. Both give limited information and neither is interactive.

If the problem is small I try to get enough information by setting bad return codes. If I think the trouble is with a particular one-byte value, I load it into register B, set carry, and return. The value makes its way back to the calling program as a bad return code. With lots of time and patience I can get a pretty good picture of what's going on this way.

When the problem is too big to see through a one-byte peephole I use a trace module. I build a nice big data module (say 1024 bytes). I have the buggy module link to it and use it as a queue to save interesting facts in. For example, at various key spots I save a character string location-identifier, a dump of the registers, and a few important variables. After the module exhibits its bug (provided it doesn't cause a system crash) I can link to the module with the debugger or write it to a file and print it with dump. This technique collects lots of data, but it's distressingly easy to install additional bugs along with the debugging code.

I can give a more satisfactory answer about the task register. OS-9 uses it. The first clue is in each process descriptor. There is a value there called the task number. This is the value that OS-9 stores into the task register when it is ready to run that process. In the process status byte you will see a byte that indicates whether the DAT image in the process descriptor has been changed since it was last used. If it wasn't changed OS-9 can set the DAT by simply storing the task number in the DAT's task register. If the process's DAT image is changed, OS-9 will have to update the DAT with the new image.

The use of the DAT task register makes things simpler and faster. OS-9 only has to write one byte to change the entire DAT image. When OS-9 is loading a new DAT image into the DAT it sets the task register to the system task number (all the data it needs is in the system address space) and gets a byte of the DAT image; then it sets the task register to the value for the new image it's building and stores the byte into the DAT. This would be harder if whole address spaces couldn't be swapped around by changing a single byte.

Code that deals with several address spaces, such as first level interrupt service, DAT maintenance, and code that moves data between address spaces, is all kept in the top 256 bytes of each address space. Hardware maps that 256 bytes of ROM into each address space automatically (except for GMX III which uses slight of hand).

My system only has space for sixteen tasks in its DAT. What happens when more than sixteen processes are active (fifteen plus OS-9)? That doesn't happen. Sleeping, suspended, and waiting processes loose their task numbers. There is always one task number allocated to the system address space. The active process gets another. If any processes are doing LOADs, they need task numbers (for reasons I don't understand). If you run fifteen LOADs plus another program concurrently you can get into bad trouble. OS-9 doesn't know how to handle that situation. If you don't do that -- and evidently it isn't a popular thing to do -- OS-9 will work fine with with the 16 task numbers available. It would probably do all right with three or four.

The AT&T Unix PC

I've tried the AT&T Unix PC. In many ways it is a fine machine. It's quite fast, has virtual memory, nice graphics, satisfactory window support, and it's backed by a company with an excellent reputation. It isn't perfect by a long stretch, but two problems bothered me particularly. It doesn't run OS-9 (and it should ... beautifully), and there has been a lot of talk about AT&T's "open architecture" policy (i.e., mostly closed).

The two problems are connected. Microware would very much like to port OS-9 to the Unix PC. They are actively pursuing it with people at AT&T. The problem is that they can't get some important technical details from AT&T. This is the "open architecture" problem. If they do get OS-9 ported (and I expect that they will), there will be some question of who gets to market it, Microware or AT&T. Me, I'm an optimist, I'll predict OS-9 for the AT&T Unix PC by first quarter 86.

I have heard a great deal about how sticky AT&T is being about handing out information about the workings of their machine. I was ready to lay into them about that, but decided that I should put on my journalist's hat and ask some questions. I got in touch with AT&T's public relations department and they arranged a deluge of information for me.

I talked with Bob Sellinger from AT&T Information Systems. He gave me some medium encouraging answers to my questions.

Evidently AT&T is trying to keep control of development for the Unix PC. They have a number of levels of support. Bob Sellinger says that the manuals that will sell to anyone who wants them should be sufficient for a qualified person to do any kind of development for the AT&T Unix PC up to and including porting a new operating system to it. If AT&T likes what you are up to they can give much better support. They have already invested large efforts in other companies working to enhance the Unix PC.

An example of information that is only available to companies that AT&T smiles on are the schematics for the

machine. I think they are being too restrictive. They think they are looking out for their interests.

The first impression I had of the "smiled on by AT&T" state was that you had to be a major customer of theirs before you could begin to qualify. Bob Sellinger says that is not necessarily true. If they like your idea enough they might even back a small outfit.

In particular, there is a path Microware might be able to go down to get the information they need to get OS-9 running there. They should get in touch with the Unix PC product management people.

One general note on the AT&T PC. Don't even think of running Unix on it unless you have at least a megabyte of memory (you can buy it with 512K but DON'T).

Benchmarks

The Gimix Micro-20 board isn't available yet (early August), but I got Mike Magnus at Gimix to run some benchmarks for me on their prototype. I sent him four programs: a knight's tour, a fibinacci numbers program, a quicksort, and a prime number seive. Here are the timings for the Gimix Micro-20, a 68008, my 6809, and a 8 MHz 8086. The Micro-20, 68008 machine, and my computer were running OS-9 and using Microware C. The 8086 was running MS-DOS and Microsoft C.

	Micro-20	68008	8086	6809
7x7 tour	539 s	10839 s	---	---
8x8 tour	83 s	436 s	---	600 s
fibinnaccii	12.5 s	73 s	32.7 s	50 s
Quicksort	10.4 s	63 s	46.3 s	213 s
Seive	0.9 s	7 s	2.9 s	9 s

Hot Scoops

The Gimix Micro-20 will include a 16-data-bit connector for additional I/O. It will be the connection point for an optional graphics board that Gimix plans for the future. The graphics board will use the new Hitachi graphics chip which is already supported by Microware.

Microware's Fortran compiler for the 6809 is out to test sites and looking good.

I just got OS-9/68K version 1.2. It has at least three important new features: environment variables, named pipes, and "events."

The shell can set environment variables which are local to it. These variables are passed to each program the shell starts as part of the parameter string (after the <CR> closing the command line). Shells inherit their parent's environment, but don't pass anything back when they terminate. The sample use for environment variables that appeals to me most is as a good place to keep your terminal type.

Conventional pipes can only make connections between processes and their children, or between sibling processes (if the pipes are built by the parent). Named pipes can be opened on one end just like any sequential file. Any process that knows the name of the pipe can open it. For example, imagine you have a process controlling a machine which only needs instructions occasionally. You could leave a terminal attached to the process, but that would be wasteful if you had anything else to do with the terminal. You could have the process read a command file from time to time, but that requires a disk and involves contention problems. You could use a data module and a locking protocol; that would work fine, but it's a little complicated. Now you can have the process leave a named pipe open. You can use the terminal for whatever you want. When you have instructions for the process controlling the machine you just pour them down the pipe. You could even do it with a normal OS-9 command like copy or echo.

Meet people making it happen in OS-9. The movers and shakers who are helping OS-9 become the fastest growing operating system for the 6809 & 68000 in the world.
Lively and informative round-table discussions will cover the design and use of Microware Software. We'll also discuss OS-9's dynamic growth from where we are today to where we may be in the future.
The exhibit area will feature booths from many of the leading suppliers of OS-9 compatible hardware and software. It's a great opportunity to increase your skill and knowledge in the latest microcomputer software technology. Plan to attend — Register Today!

Seminar only $150 Hotel Package* $350
Location Marriott Hotel, Des Moines, IA
Don't Miss It — Pre-Register Now!
Call 515-224-1929 or Write
MICROWARE SYSTEMS CORPORATION
1866 N.W. 114th St. • Des Moines, IA 50322

microware®

**Hotel package includes 3 nights, single occupancy at the Marriott Hotel and registration fee.*
OS-9 and BASIC09 are trademarks of Microware and Motorola

You can even ask for a directory of pipe names with a command like:

OS9 dir /pipe.*

You can have several different pipe device descriptors if you like. I'll get into why that's important some other time.

You can look at events two ways. They are an efficient type of signal from one process to another, and they are a way of serializing the use of some resource. A print spooler should be simple to write using a combination of a named pipe for input to it, and an event to serialize access to that pipe.

Either named pipes or events would be sufficient to make writing service processes for OS-9 much easier. I certainly hope that Microware's policy of keeping OS-9 for the 6809 as compatible with OS-9/68K as possible will bring the 6809 community these features.

MCI Mail

An electronic mail system that would reach to our non-computerized friends would be a great thing. Even those poor people using operating systems other that OS-9 would be able to take advantage of it.

I read the review of MCI mail in Byte a few months ago. When I bought my NEC Starlett some of the literature that came with it encouraged me to subscribe (is that the right word?) to MCI mail and Dow Jones. I had been sitting on the fence and the Starlett pushed me over. I made the toll-free call and joined up.

The problem with electronic mail as it stands is that it can only reach other regular computer users who are hooked into the system. I subscribe to Compuserve but I can't afford to use it any more. It is very convenient for people on Compuserve to send me mail but I'll never know that it's waiting for me. I'm not a "regular user" of that system any more. My mother has a computer (of course), but she isn't hooked into any form of electronic mail. I have to print out letters to her and send them through the postal service.

The people who designed MCI mail evidently had these problems in mind. The system effectively deals with them. There are five classes of mail. The least expensive is purely electronic: you send the letter from your phone, when the recipient checks his MCI mail (by logging on) he'll get it. If you want to send mail to someone who doesn't check for mail regularly, you can pay an MCI operator to call him and tell him that MCI mail is waiting for him. If the recipient doesn't subscribe to MCI mail or can't get electronic mail for some other reason and time is much more important than money you can have the letter printed on a laser printer and delivered within four hours by a courier. The next level down gets you next-day delivery of laser printed output. The slowest delivery is laser printed and delivered via US mail.

The problem with a purely electronic mail system is that it is only useful when a large fraction of your correspondents subscribe. MCI gets over that hump by offering a family of hard-copy mail services. It is crucial that the hard-copy mail services be better or cheaper than conventional mail. The fast delivery modes compete with the delivery services like Federal Express on price. A six page MCI letter for next day delivery costs $8.00. Letters that go through the postal system are printed at "the print site nearest its destination" and mailed from there. Although the post office does strange things it seems reasonable to expect letters to arrive faster if they are mailed from closer to their destination.

**** Editor's Note: Ain't SO, Pete!!! ****
DMW

Of course, all the best people are laser printing their correspondence these days. Nothing wrong with laser printed letters. MCI can even print your letterhead (black and white) and signature if you like.

The idea behind MCI mail is excellent. They cover all the angles. Unfortunately it falls apart completely when it comes to execution. Let me tell the story in order.

I got a nice package from MCI on a Thursday. There was a letter with my userid and password, a tutorial manual, a users manual, a price list and a list of access phone numbers. I was pleased to notice that Rochester was on a short list of cities with free phone access, but it worried me. Isn't MCI a phone company that services all cities? I thumbed through the manuals. They must have been aimed at the very novice user. I felt bored and uninformed. I decided to wing it; I just called the number and logged on.

It worked fine. The software is well designed. The online help is clear and more complete than the manual. I was greeted with headlines from the day's news -- which I appreciated because I hadn't heard any news that day. Without any trouble I sent myself an electronic message and a hard-copy letter.

I checked for the electronic message that I sent myself. Nothing. I logged off and on again. Still no message. Hmmm. Well I'd look the next day. I sent both messages around 11:00 in the evening. I expected the printed letter to arrive on Saturday. I really expected the electronic mail to arrive immediately, but Friday would be fine,

I logged in Friday. My mail to myself had arrived. I was impressed again with the software. Very nice design.

I waited for the printed letter to myself on saturday. When it didn't arrive on monday I called MCI. Was there a problem? Had I misunderstood? I had misunderstood. Letters were indeed mailed from the nearest print site, but there were only a few (four I think) to choose from. The nearest one to me was New York City. I logged into MCI again and checked some more help files. I discovered that the 4-hour mail was restricted to a few cities. New York was on the list, but the other cities in the section of the country that I checked weren't, not even Boston!

My letter arrived on wednesday. It was postmarked August 2 (they mailed it promptly) from Indianapolis. What happened to New York City? Somehow they managed to make it look like junk mail. Maybe it was the international orange envelope. If I hadn't been waiting for a letter from MCI I would have chucked it in the trash unread.

The first page of my letter was a cover page meant to show through the windows in the envelope. It had an address and return address. It also had a big add for MCI "MCI Mail The nation's new postal system" that matched the one on the envelope. Not very classy. The next page contained my letter. The letterhead was another copy of the MCI ad. Well... I hadn't given them a letterhead. I suppose they had to use something.

I paid close attention to the quality of the laser printing. It wasn't bad, but I've seen better from dot matrix printers.

Let me summarize. MCI electronic mail is good; maybe even excellent. It's well thought out and well implemented. I only wish they had free access from most places.

I may try their overnight mail someday. The price is good and I'd like to know how it works. They don't deliver 4-hour mail anywhere I'm interested in.

I won't send the kind of letter I sent myself to anyone else. I'd be embarrassed. Why should I pay $2.00 to send a letter that looks like junk mail.

I think electronic mail for the masses is going to have to wait for a company that is ready to risk more money. Too bad.

"C" User Notes

Edgar M. (Bud) Pass, Ph.D.
1454 Latta Lane
Conyers, Ga 30207

INTRODUCTION

This chapter discusses the James McCosh series of C compilers for several operating systems which run on the 6809. This C implementation is almost complete and has 6809 versions for Flex (marketed by Windrush and S.E. MEDIA), UniFlex (marketed by MPU), and OS/9 (marketed by Microware and Tandy). Although there are differences among the implementations, they are all very similar and the others are upward-compatible from the Flex version. They are all described below.

SCOPE OF IMPLEMENTATION

The McCosh C compiler is a rather complete implementation of the version of UNIX C described in Kernighan and Ritchie (K & R).

The primary restrictions and differences in the McCosh C compiler are as follows:

a. Bit fields are not supported.

b. Constant expressions for initializers may include arithmetic operators only if all the operands are of type 'int', 'unsigned' or 'char'.

c. The older forms of assignment operators ('=+' and '=*') which are recognized by some C compilers are not supported.

d. "ifdef" (or "ifndef") "...else..." "endif" is supported but "if <constant expression>" is not. These constructs may not be nested.

e. It is not possible to extend macro definitions or strings over more than one line of source code with the '\'-new-line sequence, as used by UNIX C.

f. The escape sequence for new-line ('\n') does not mean line-feed (hex 0A) rather, it means carriage- return (hex 0D). All programs which use '\n' for end-of-line will work as intended.

g. "Unsigned char" is not supported.

h. Many variants of the "include" construct are not allowed. The file name must be enclosed in quotation marks (") or broken brackets (<,>). The format of the file names in this and in other constructs are dictated by the operating system under which the program is to be run.

i. The use of register variables is highly restricted.

The primary extensions to the language are as follows:

a. The type-specifier "direct" may be applied to global variables to indicate that they are to be collected in a special area addressed by the 6809 direct page register. Since only 256 bytes may be addressed by the direct page register, only a limited number of variables may be declared with type "direct". The primary effect this type-specifier has on most users is to prohibit any use of the word "direct" as a variable name.

b. A line beginning with "#asm" switches the compiler into a mode which passes all subsequent lines not beginning "#endasm" unchanged to the output. This is a useful facility for inserting lines of assembly code.

c. The escape sequences for non-printing characters in character constants and strings are extended as follows:

line-feed LF \l

which is to distinguish LF (hex 0A) from "\n", which is the same as "\r" (hex 0D).

d. Bit patterns in the following forms are supported:

***	octal constant
\d**	decimal constant
\x**	hexadecimal constant

Other implementation differences are be covered later in this chapter; however, several are as follows:

a. "Char" variables are signed, and there is no provision for unsigned char variables.

b. "Float" and "Double" variables do not necessarily have the same format in the McCosh C series of compilers as in the UNIX C compiler.

c. Redirection of standard input and output files under Flex is performed using the I and O Flex commands, rather than the familiar "<" and ">" symbols. This restricts redirected text files to containing text, not binary, data. Since the I command is really a Flex utility which reads a disk file and switches back to the console on end of file, the user must manually enter a control-D from the console to indicate end of input.

IMPLEMENTATION-DEPENDENT ITEMS

Whereas the previous section discussed differences between the McCosh C and UNIX (K & R) C compilers, this section discusses those implementation-dependent items not covered in K & R.

PRIMITIVE TYPE SIZES

Each variable type requires a specific amount of memory for storage. The sizes of the basic types in bytes are as follows:

This compiler follows the UNIX C implementation in that chars are converted to ints by sign extension, 'short' or 'short int' means int, 'long int' means long and 'long float' means double.

The format of a 'long float' variable is as follows:

one bit for sign
six bytes and seven bits for mantissa
one byte for exponent

The form of the mantissa is sign and magnitude with an implied '1' bit at the sign bit position. The exponent is biased by 128.

The format of a 'float' variable is as follows:

one bit for sign
two bytes and seven bits for mantissa
one byte for exponent

REGISTER VARIABLE USAGE

Zero, one, or two register variables may be declared in any function. The only types permitted for register variables are int, unsigned and pointer. Invalid register variable declarations are ignored (the storage class is made auto).

A considerable saving in code size and speed can be made by judicious use of register variables. The most efficient use is made of them for pointers and counters for loops. However, if register variables are used in complex arithmetic expressions, there are no savings.

CONSTANT EXPRESSION OPTIMIZATION Because the 6809 is an 8/16 bit processor, it is possible to generate efficient code for 8 and 16 bit objects. However, code for 32 bit values can at best be four times longer and slower. Further, it is possible within the confines of the 6809's memory limits to do some quite extensive evaluation of constant expressions provided they involve only constants of type char, int and unsigned. There is no provision in this compiler for constant expression evaluation involving constants of type long, float or double. Complex constant expressions involving these types are evaluated at run time by the compiled program.

ARITHMETIC ERROR PROCESSING

The K & R book leaves the treatment of various arithmetic errors open, merely stating that it is machine-dependent. This implementation deals with a limited number of error conditions in a special way; it should be assumed that the results of other possible errors are undefined.

If an error occurs in an arithmetic routine which cannot be handled, control is passed back to the operating system, after placing one of the following messages on the system console (for Flex) or returning one of the following system error numbers (for OS/9):

```
    Flex
*** FLOATING-POINT OVERFLOW ***
*** DIVIDE BY ZERO ATTEMPTED ***
*** FLOATING-INTEGER CONVERSION OVERFLOW ***
    OS/9
EPPOVR    40
EDIVERR   41
EINTERR   42
```

The UniFlex C manual does not specify such error processing.

STANDARD LIBRARY FUNCTIONS

It is essential to begin any source file which uses the higher-level I/O functions from the standard library with `#include <stdio.h>`.

If output via printf(), fprintf() or sprintf() of long integers is required, the source must call `pflinit()` at some point; this is necessary so that programs not involving long variables do not have the additional code required to implement long variable output. Similarly, if output of floating-point or doubles is required, the source must call `pffinit()`.

COMMAND LINE ARGUMENTS

Programs written in C which are invoked with command line arguments can use the conventions outlined in the K & R book. The system interface makes the arguments available to the program in exactly the manner described, including the name of the program itself as argv[0].

Arguments are considered delimited by commas or spaces unless they appear within strings delimited by quotes ("..."). This is currently not true in the Flex implementation, however.

Redirection of standard input and output files is performed under Flex thru the use of the Flex I and O commands, rather than thru the use of the standard "<" and ">" symbols. This is not a complete implementation, due to the editing of output stream contents by the O command and to the lack of end of file processing by the I command.

CREATING OBJECT FILES FROM C PROGRAMS

The McCosh C compilers provide a simple means of compiling 6809 object code from a C program. Essentially the same technique is used for Flex, UniFlex, and OS/9 implementations of McCosh C.

This involves the user entering a command line of the following format:

```
CC flags file1 file2 ... file10
```

in which flags provides compiler options, as described below, and the files provide the names of one to ten files, with explicit suffixes, as follows:

```
.c = C source program file,
.a = assembler source program file,
.r = relocatable object program file,
```

Actually, for the OS/9 version of McCosh C, there are two CC's, named CC1, for level 1 systems, and CC2, for level 2 systems. They are referred to here collectively with the Flex and UniFlex versions as CC. If one file name is provided, CC will guide the execution of the C compiler phases, ultimately producing an object file with name usually having one of the formats "file.CMD" for Flex, or "file" in the current execution directory for UniFlex or OS/9.

If more than one file name is provided, the relocatable object files generated for each file name in the list will be saved, in addition to the generation of an object file usually named "OUTPUT.CMD" for Flex or "OUTPUT" for UniFlex or OS/9. These relocatable object files, named with suffix of ".r", may be used with another CC or with the loader to avoid the recompilation of the individual C programs.

The optimizer pass may be selected to occur after the other compilation passes. It reads the assembler source code text and removes redundant code and searches for code sequences that can be replaced by shorter and faster equivalents. The optimzer is claimed to shorten object code by about 11% with a corresponding increase in program execution speed.

CC recognizes several command-line flags which modify the compilation process where needed. Since the command line is scanned before compilation commences, flags may be placed in any order on the command line.

The following table describes the command-line flags for the Flex and UniFlex versions of the McCosh C compiler:

```
+A - suppresses assembly, leaving the output as
     assembler code in files with extension ".a"
+O - invokes assembler code optimizer
+R - suppresses loading and linking, leaving the
     output as files with extension ".r"
+L=filename - specifies a library to be searched by
     the loader before the standard library and system
     interface
+F=filename - specifies the name of the output file
+C - inserts C source code as comments into
     assembler code
+N - suppresses production of assembler code
+S - suppresses generation of stack-checking code
+Did[=string] - generates "#define id string" as C
     statement
```

The flags may be run together except where a flag is followed by something else.

For most simple and some complex C programs, CC makes the compilation process almost as simple as possible. With CC, McCosh C is easier to use than INTROL C, which requires a separate linking step, and is easier to use than most Small C compilers except those which produce object code directly.

The OS/9 version of McCosh C uses a slightly different version of the command line from that just described. The following list describes the command-line flags for the OS/9 version of McCosh C:

```
-A - suppress assembly, leaving the output as
     assembler code in files with suffix ".a"
-E=number - sets OS/9 module edition byte to number
-O - suppresses the assembly code optimizer pass
-P - invokes the profiler to generate function
     invocation frequency statistics after program
     execution
-R - suppresses loading and linking, leaving the
     output as files with extension ".r"
-M=sizeK - instructs the linker to allocate size K
     bytes for data, stack, and parameter area
-L=filename - specifies a library to be searched by
     the linker before the standard library and system
     interface
-F=path - specifies the name of the output file
-N - suppresses production of assembler code
-S - suppresses generation of stack-checking code
-Did[=string] - generates "#define id string" as C
     statement
```

The flags may be run together unless something else follows a flag.

The OS/9 version of McCosh C provides a facility known as the profiler. The profiler is an optional method used to determine the frequency of execution of each function in a C program. It allows the identification of most-frequently used functions. When the -P compiler option is selected, code is generated at the beginning of each function to call the profiler module, which counts invocations of each function during program execution. When the program has terminated, the profiler prints a list of all functions and the number of times each was called.

OPERATIONAL PROBLEMS

The McCosh C compilers share a few operational problems

with INTROL C and probably most other FULL C implemenations on micros. The primary ones involve speed and storage.
The speed problems stem from several sources. Even for very small files (such as HELLO, which prints "HELLO, WORLD"), CC must load each of the other C compiler modules, in turn, from an attached disk or diskette storage device. Even with a hard disk or virtual disk, this process consumes much time. With SA400-type mini-disk drives, the wait could become very lengthy. The C compiler modules generate (and read back) files representing various stages of compilation of the original program, so larger C programs could require a significant amount of disk activity themselves.

Internally, the C compiler modules do not seem to require inordinate amounts of processor time, but, again, larger programs will require significantly more internal time to process than smaller ones. Modularizing large programs may assist somewhat in reducing the time and effort required to maintain them, although there are major trade-offs to be considered in such a process.
The disk and main memory storage problems also stem from several sources. The amount of disk space required to contain CC and the other C compiler object modules and required files may be quite large.
Compilation of a C program by CC requires that the following or similar files be present on the system drive under Flex:

cc.cmd	compiler executive
cprep.cmd	macro preprocessor
cpass1.cmd	compiler pass 1
cpass2.cmd	compiler pass 2
copt.cmd	optimizer
casm.cmd	relocating assembler
cload.cmd	linking loader
catart.r	program initialization module
clib.lib	standard library functions
stdio.h	standard I/O definitions
ctype.h	standard type definitions
setjmp.h	long branch functions
flex.h	standard Flex definitions

Compilation of a C program by CC requires that the following files be present in the current OS/9 execution directory:

OS/9 Level 1 Systems:	
ccl	compiler executive
c.prep	macro preprocessor
c.pass1	compiler pass 1
c.pass2	compiler pass 2
c.opt	optimizer
c.asm	relocating assembler
c.link	linking loader
OS/9 Level 2 Systems:	
cc2	compiler executive
c.prep	macro preprocessor
c.comp	compiler
c.opt	optimizer
c.asm	relocating assembler
c.link	linking loader

In addition, a file called "clib.1" contains the standard library, match functions, and system library. The file "catart.r" is the setup code for compiled programs. Both of these files are normally located in a directory named "lib" on the system's default mass storage device, which is usually the disk drive the system is booted from.
The combination of these files would fill most of a SS/SD 5" drive. In addition to the space required for the compiler files, several temporary files are created on the working drive or in the current data directory during compilation. At least three times the size of the largest source file (and its included files) should be available for these files. This restricts the size of C program which may be compiled on a SS/SD 5" dual-drive system to about 80 sectors.

The amount of main memory required to contain CC and the other C compiler object modules is significant. Under Flex and OS/9 level 1, the main portion of the compiler is split into two parts. Although this is sufficient to allow the Flex version to operate normally, the OS/9 level 1 version requires essentially all of the potential space available (about 167 pages) after OS/9 is loaded, leaving no room (on many systems) for the extra copy of the OS/9 shell data space required to operate CC. In severe cases, there may not be room enough to load one or the other pass of the C compiler. This problem may be somewhat alleviated by the generation of a minimal OS/9 system containing as few drivers as possible for the configuration, but memory space on an OS/9 level 1 system is extremely tight.

ERROR DIAGNOSTICS

Good error diagnostics are extremely helpful to the use of a compiler as sophisticated as a C compiler. Unfortunately, many C compilers produce very cryptic and misleading diagnostic messages, leading to the impression shared by many people that the C language is obscure and strange, fit only for system programmers.
The McCosh series of C compilers seems to take great pains to provide good error diagnostics, including the use of English messages (not error numbers) and a glossary explaining most of them in greater detail. Most syntax error messages are preceded by the line containing the syntax error with an arrow pointing to the offending item. There are, unfortunately, a few undocumented error messages which are, of course, the most cryptic.
The McCosh C compilers provide good error detection, but it is in the nature of compilers for structured languages that one syntax error in a program can cause the compiler to lose its way and start issuing a series of spurious error messages. The results of most of the common errors are anticipated and generally do not cause this nuisance. If the number of errors becomes too large (more than 30), the compiler aborts. The error diagnostics are in general self-explanatory, but the manual may help if the meaning appears a little obscure. Of the three McCosh C manuals, the Flex manual is the best and most complete. In contrast, the UniFlex manual consists almost entirely of a list of the library functions. The OS/9 manual is almost as good as the Flex manual, but is not organized as well and is not as complete. The Tandy manual is a reset version of the Microware C manual. None of the manuals provide an introduction to the language, referring to K & R for the language definition. The Flex manual has a unique touch in coloring the pages of the sections for quick reference.

THE McCOSH LINKING-LOADER

The term 'loader' is misleading. In the early days of computing, a loader had the task of placing a program and its data in memory so that it could be executed. It then had added to it the ability to search for subroutines and loading them as well. This involved ensuring that references from the program to these subroutines were correct, wherever the subroutines were actually placed in memory. Next, it became necessary to be able to load programs into different locations in memory and make sure that they would still refer to all items correctly. This is 'relocation' and 'linking'.
Up to this point, all the work of a loader was done at run time. Every time the program was run, the placing in memory, searching for subroutines, relocation and linking was done. Then it was realized that time could be saved by doing most of this work once at compile time and saving the result. The trade-off was the requirement of more storage space for larger executable files.
In modern operating systems, the tasks of transferring an executable file to memory and of executing it are carried out by the system itself. All subroutines, other than calls to the system, are carried within the executable file.

An obvious extension to the idea of having library routines is to split a large program into modules which have well-defined functions and simple interfaces to each other. The advantages are that a number of people can work on different parts of the same project and compilation and/or assembly of individual modules uses less machine resources. The loader has the job of linking them all together and searching libraries for routines which are not contained within the program modules. The output of the loader is a complete executable file.

The definitions of languages such as C and FORTRAN state that parts of a program may be separately compiled, so a linking loader is essential for their full implementation.

PROGRAM SEGMENTATION

There are basically two uses of memory by a running program. The first is for the executable code which should not be modified at any time by the program itself. The second use of memory is for the data which the program works on. In general, this memory is used for both reading and writing by the program.
The loader must collect together all the executable code pieces from the various modules and library routines and place them in a single block in the output module. It must also determine how much data memory is required by each module and fill in the total in a part of the output file reserved for information for the system.
In doing these two tasks, the loader must keep track of the offsets of all important locations in both memory sections and alter the executable code so that references are to the correct places in the final program.

It is often convenient to have initial values in parts of the data portion of memory at the moment the program starts running. Some languages, notably Pascal, have no notion of this, but FORTRAN and C are examples of languages where data initialization is possible. The values set in this way are not necessarily fixed for the duration of the program so they must be in the data portion of memory.

The loader handles this by maintaining a separate 'segment' for initializing data which is distinct from the non-initialized memory. The names for the three segments are traditionally (at least on the PDP-11) the following:

text	executable program code
data	initialized data areas
bss	uninitialized data areas

For the 6809 processor, it is highly desirable to make as much use as possible of the more efficient addressing mode available when accessing memory in the 'direct page'. There is, therefore, a fourth segment for this purpose called 'direct'.

Most users will never use CLOAD directly, as the details are taken care of by the CC routine when it produces programs which run under an operating system. Only those desiring to produce complex object modules or modules for stand-alone operation will ever use the linking loader directly.

LOADER OPERATION

The output from the assembler is in the form of a 'relocatable module' which contains the executable code, initializing data and tables of information to enable the loader to link it with the rest of a program. A 'library' file is a collection of relocatable modules.

The output from the loader is an executable program module in the form required by the operating system.
The loader takes as command-line arguments the names of a number of files containing relocatable modules and possibly one or more library files which are searched for unresolved references.
The syntax of its command line (under Flex and UniFlex) is as follows:

CLOAD [options] file1 [file2 ... filen]

in which options are described later and file1 thru filen are relocatable object modules.
The syntax of its command line (under OS/9) is as follows:

CLINK [options] mainfile [subfile1 ... subfilen]

in which options are described later, 'cstart.r' is the usual name of the mainfile, and subfile1 thru subfilen are relocatable object modules.
The command line arguments to CLOAD/CLINK consist of one or more relocatable modules and, optionally, one or more libraries. The relocatable modules must have been created by CASM.

The size for each module is limited to about 25k bytes but there is no loader-imposed limit to the size of the final executable code.
The output file is, by default, named 'OUTPUT' and is placed on the work drive for Flex or in the current execution directory for OS/9 and UniFlex.

The loader views a program as consisting of three distinct areas of memory. The first is the 'text' segment which contains the code of the program itself and any initialized data such as string constants. The second is called the 'direct' segment which contains variables which are to be accessed using the short direct addressing mode instructions of the 6809. The last is called the 'bss' segment, which is reserved for memory which will be used by the program but which has no particular values assigned to it by the program before execution. In C, these areas are reserved for all the global and static variables but the language definition states that all such un-initialized variables have the value zero at the start of execution.
Following are the options for the Flex and UniFlex versions of the linker:

+M - prints a load map of the modules providing the base addresses of the constituent segments.
+S - prints a symbol table of assigned addresses for all global symbols.
+T=address - specifies the starting address of the 'text' segment of the program; it starts at zero by default.
+B=address - specifies the starting address of the 'bss' segment of the program; it starts just beyond the 'text' segment by default.
+L=filename - specifies the name of a library file to be searched for unresolved external references.
+O=filename - specifies the name of the output file into which the executable program code is placed; it is named 'OUTPUT.CMD' by default.

Following are the options for the OS/9 version of the linker:

-m - prints a load map of the modules providing the base addresses of the constituent segments.
-s - prints a symbol table of assigned addresses for all global symbols.
-l=filename - specifies the name of a library file to be searched for unresolved external references.
-o=filename - specifies the name of the output file into which the executable program code is placed; it is named 'OUTPUT.CMD' by default.
-e=edtion - specifies the edition number for the output module.
-m=sizeK - overrides the size of the data area of the output module.
-b=entry - specifies the name of the entry point of a module to be called from BASIC09.
-t - allows static data to appear in a BASIC09-callable module.

STAND-ALONE CODE FROM C

C is an excellent language with which to write programs intended for use in stand-alone environments and for applications such as process control. CC supports such programming, but the means of producing the final code is not as straightforward as for programs intended to run under an operating system.

The procedure to be followed depends primarily upon the application and the form of the final code desired.
CC itself is an executive program which determines what has to be done by the various constituent parts of the compiler, causes the operating system to execute them in the correct order, and ensures that the file names conform to conventions.

If the '+A' option is used when invoking CC, the processing is stopped before any assembly takes place. It may be instructive for the user to inspect the code produced, particularly if the '+C' option - source lines to be output with the assembly code as comments - is also used. At this stage, the code could be used as a source module for assembly with others.
If the '+R' option is used, the outputs are relocatable modules which may be link-loaded using CLOAD with other modules either from single files or libraries of modules. It is at this stage that control can be exercised over the locations of the text (executable code and initialized data area) and bss (uninitialized data area) segments.
Since many of the standard library functions call operating system functions, their use must be care fully monitored or entirely eliminated by coding equivalent routines in C or assembler by the same names.

C STANDARD LIBRARY

The C standard library contains functions which fall into the following classes: I/O and convenience. The McCosh C manuals provide complete lists of the names and usage for each function in their respective standard libraries.

The I/O functions provide facilities normally considered part of the definition of other languages; for example the FORMAT statement of Fortran or the PRINT statement of BASIC. The main benefits of their inclusion are ease of use and compatibility with programs written for other environments.

There are two categories of I/O functions and the difference between them may cause confusion. The basic I/O operations are performed using the lower-level functions which correspond to 'system calls' as described in K & R. All the calls mentioned there are available in Flex (except 'lseek' and 'seek'. However, most users will prefer to perform I/O operations using the 'higher level' functions described in the previous chapter of K & R. This is because they are more versatile and easier to use. The higher-level functions use the lower-level ones as their interface with the operating system itself. The main difference between the higher-level functions and the lower-level ones is in the methods of identifying open files.

The lower-level functions use file descriptors which are small integers. File descriptors 0, 1 and 2 refer to the standerd input, standard output and standard error 'files' which are always open from the start of program execution. The term 'standard input' normally refers to the user's terminal keyboard but may in fact be from a file if the operating system's redirection ability is invoked. Similarly, 'standard output' refers normally to the terminal screen but may have been re-directed to a file. The 'standard error output', however always refers to the terminal so may be used to direct error messages to the screen where they will not escape attention even if other output is going to a file. File descriptors for other files are obtained from calls to 'open' or 'create' and are used to identify the file for subsequent calls to other lower-level functions.

The higher-level functions use file pointers which are pointers to structures maintained by these functions. The File Pointer for an open file is obtained from a call to 'fopen' an dis used to identify the file in subsequent calls to other higher-level functions. There are three file pointers declared in "stdio.h" which represent 'files' which are open from the start of execution: 'stdin', 'stdout' and 'stderr' correspond to File Descriptors 0, 1 and 2 respectively.

The convenience functions include facilities for copying, comparing and concatenating strings, making numbers out of strings, classifying characters, etc. In addition, for OS/9 and UniFlex, they include direct system calls for performing various operating system - specific tasks.

McCosh C assumes that the value of an undeclared function is of type int, just as K & R specifies. However, to obtain correct type conversions (or to avoid unwanted ones), functions returning non-int values must be pre-declared, as described in an earlier chapter.

ACCESSING HARDWARE IN C PROGRAMS

There are a number of techniques for accessing hardware in McCosh C programs. The definition of C enables the programmer to dispense almost entirely with assembly code except in very rare circumstances. After all, C was developed for writing the UNIX operating system.
The first and simplest method is to use a pointer. Any value may be assigned to a pointer and the pointer subsequently used to access the address. For an example, suppose a hardware register is addressed at $E014. The following definition of a pointer would be adequate to access the register:

```
char *reg = 0xe014;
```

This defines a pointer called 'reg' which is initialized to contain the address of the register. The register may then be accessed directly, as follows:

```
value1 = *reg;
*reg = value2;
```

The second method arises from the ability to use an integer on the left side of the '->' operator as if the integer were a pointer value. If a device is located at $E014 which had two registers, the following could be coded:

```
struct {
        char datreg, conreg;
       };
#define DEVICE1 0xe014
```

The device could then be accessed as follows:

```
DEVICE1->datreg = value1;
value2 = DEVICE1->conreg;
```

Note that accessing hardware directly, while perfectly acceptable in stand-alone mode and under Flex, is forbidden under UniFlex and OS/9 level 2 (in user programs) and must be carefully reviewed under OS/9 level 1. It also reduces the portability of C programs containing such constructs.

SUMMARY

The McCosh series of C compilers, which was described in this chapter, offers a combatible series of rather complete C compiler implementations for Flex, UniFlex, and OS/9, similar in capabilites and in scope to the Introl series of C compilers for Flex, UniFlex, and OS/9.
Owing to differences in the operating systems and memory capacity, there are slight differences among the products. Virtually any program written for the Flex version of this product should run on the UniFlex and OS/9 versions of this product with minor changes. However, the UniFlex and OS/9 versions each have divergent language extensions from the Flex version.

ADA® And The 68000

Theodore F. Elbert
The University of West Florida
Pensacola, Florida 32514

PART 7 - Ada's Packages

The package, as a program structure, is unique to the Ada language. It provides the language with an information hiding and modularization mechanism that is much more functional than that provided by the subprogram. Packages are used to group--by means of encapsulation--other entities of an Ada program. To be effective in terms of good software design, of course, these entities should share some common logical relationship with respect to the context in which they are used within the program. That is, those entities encapsulated by a package should be logically related. As is the case with subprograms, packages may be declared in two parts--the specification and the body--and these parts may be compiled separately. The specification part is visible to other program units and itemizes the facilities provided by the package. The body of the package is hidden from other program units and contains the implementation details in terms of executable code and, perhaps, in terms of data structure. This two part structure serves to provide the information hiding facility necessary to prevent a user of the package from exploiting any knowledge he may have of implementation detail, which in turn prevents a user from developing code that depends on the particular implementation of package facilities. Thus, once the question of the package specification has been resolved, the package body may be implemented in any manner consistent with the specification. Furthermore, the implementation can be modified at any time and in any manner that preserves this consistency.

The Ada entities encapsulated within a package may be any entity that can be declared in the language. This includes data types, data objects, subprograms, tasks, and even other packages. In order for an entity to be accessible outside of the package, the declaration of the entity must appear in the package specification. Moreover, the specification may be divided into a visible part and a private part, and those entities declared within the private part are not visible outside the package itself. Thus, users of the package may not reference these entities. The combination of visible and private parts of a package specification is used in the declarations of private data types, which are discussed below. On the other hand, those entities declared in the visible part are visible to outside users.

A package can have a specification and no body. Such a situation occurs when the package encapsulates only a group of type declarations, object declarations, or a combination thereof. These declarations are to be visible to other program units, so they appear in the package specification. Since no subprograms, tasks, packages, or generic units appear in the specification, there is no requirement for a package body.

Encapsulation of Declarations. It is not uncommon in large scale system development to have many different subprograms use a common set of constant values, a common set of data types, or a combination thereof. To ensure compatibility among the various subprograms--and, in the case of Ada software, among subprograms, packages, and tasks--the declaration of commonly used entities can be encapsulated within a package. The package can be compiled and then accessed by other program units. In this manner, one can be ensured that each program unit is using the same set of declarations --something that cannot be assured if each program unit provides its own declarations. An example of this use of a package is given below:

```
package VOLUME_CONSTANTS is
   CU_FT_TO_CU_METER  : constant := 0.0283177;
   CU_YD_TO_CU_METER  : constant := 0.764559;
   CU_IN_TO_CU_METER  : constant := 1.638716E-5;
   BUSHEL_TO_CU_METER : constant := 0.035239;
   BARREL_TO_CU_METER : constant := 0.119240;
   GAL_TO_CU_METER    : constant := 0.0037854;
end VOLUME_CONSTANTS;
```

This package has the specification shown above, which contains only constant declarations; it has no body. This specification may be separately compiled and then referenced by other program units. Since the constant declarations appear within a package specification, they are visible outside the package.

A second example illustrates the use of a package to provide data types for use by other program units:

```
package AC_STATE is
    type VELOCITY is digits 10;
    type POSITION is digits  8;
    type STATE is
        record
          X_VEL : VELOCITY;
          Y_VEL : VELOCITY;
          Z_VEL : VELOCITY;
          X_POS : POSITION;
          Y_POS : POSITION;
          Z_POS : POSITION;
        end record;
end AC_STATE;
```

In general, Ada's packages are used in one of three ways:

- To encapsulate the declarations of closely related data types, data objects, and constants. Packages containing such declarations contain no subprograms or other program units, so there need not be a package body. However, a package body can be used to initialize the values of data objects through the execution of an optional sequence of statements within the body.
- To encapsulate related subprograms, and perhaps other related program units, in order to provide some sort of common service. By including the specification of the subprogram or other program unit within the package specification, it is made visible outside the package and thus available for use by external program units. Normally, this kind of package will also have certain data types declared in its specification.
- To create abstract data types by coupling the package concept with that of a private data type. The private data type and all applicable operations associated with the type are declared within the visible part of the package specification--and are therefore visible to the user of the package--but the details of the structure, representation, and implementation are hidden.

Each of these uses is important in the application of software engineering principles for the purpose of enhancing the reliability, modifiability, and

understandability of software. They aid in the reduction of the complexity of large software systems and hold promise for an industry of reusable software. Each of these three general applications is discussed below.

This package contains only a specification part. When separately compiled and then referenced by other program units, this package provides the scalar floating point types VELOCITY and POSITION, and the record type STATE. That is, other program units may declare objects of these types.

Encapsulation of related program units. Certain procedures, functions, and perhaps even tasks, often have a relationship that binds them together in some logical fashion, even though they may not be dependent upon one another. For example, subprograms providing for the implementation of trigonometric functions might be grouped together in the form of a package. In fact, packages of this sort are among the first to be developed or acquired by new Ada users, because they are often required in embedded system applications. (Basic functions such as square root, logarithm, real exponential, and the trigonometric functions, among others, are not contained within the Ada language. The intent is for each user to provide his own functions, as required and with the desired precision, through the use of the package structure.) An example is given below in the form of a package that provides certain matrix operations. Because the package contains subprograms, both a specification and body are required. The specification is:

```
package MATRIX_ARITH is
    type MATRIX is array(INTEGER range <>,
                         INTEGER range <>) of FLOAT;
    procedure MATRIX_ADD(LEFT, RIGHT : in MATRIX;
                         RESULT : out MATRIX);
    procedure MATRIX_SUB(LEFT, RIGHT : in MATRIX;
                         RESULT : out MATRIX);
    procedure MATRIX_MUL(LEFT, RIGHT : in MATRIX;
                         RESULT : out MATRIX);
    MATRIX_ERROR : exception;
end MATRIX_ARITH;
```

This package declares the single data type MATRIX--a two dimensional unconstrained array type--and the three operations MATRIX_ADD, MATRIX_SUB, and MATRIX_MUL. It also declares the exception MATRIX_ERROR, which is intended to be raised in the calling environment if the matrix dimensions are not proper. Thus, another program unit may declare objects of type MATRIX, and may also call any of the three subprograms to perform operations on those objects. The package body must contain the bodies of the three procedures. In skeletal form, the body would be

```
package body MATRIX_ARITH is
    procedure MATRIX_ADD(LEFT, RIGHT : in MATRIX;
                         RESULT : out MATRIX) is
    begin
        --sequence of statements of procedure MATRIX_ADD
    end MATRIX_ADD;
    procedure MATRIX_SUB(LEFT, RIGHT : in MATRIX;
                         RESULT : out MATRIX) is
    begin
        --sequence of statements of procedure MATRIX_SUB
    end MATRIX_SUB;
    procedure MATRIX_MUL(LEFT, RIGHT : in MATRIX;
                         RESULT : out MATRIX) is
    begin
        --sequence of statements of procedure MATRIX_MUL
    end MATRIX_MUL;
end MATRIX_ARITH;
```

The sequence of statements in each procedure body implements the operation provided by the procedure. It will also contain the statements necessary to check the dimensions of the two matrix operands, and to raise the exception MATRIX_ERROR if the dimensions are not compatible. The specification and body can be separately compiled, so long as the specification is compiled prior to compilation of the body. Once the specification is compiled, other program units may reference type MATRIX and procedures MATRIX_ADD, MATRIX_SUB, and MATRIX_MUL, and these units themselves can be compiled even before the body of package MATRIX_ARITH is compiled. The fact that program units that use entities provided by the package can themselves be compiled prior to compilation of the package body illustrates the abstraction and information hiding features of the Ada language.

Implementation of abstract types. One of the major causes of problems in embedded system software--or in any other large scale program, for that matter--is the sheer complexity of the algorithms implemented by the software. Many of the features of the Ada language permit--and even encourage--the use of abstraction as a technique for handling the complexity of large programs. Subprograms, for example, provide for abstraction of program control, so that the user need only consider the input-output relationship implied by the subprogram specification, rather than being concerned with the implementation details. In Ada programs, data can also be abstracted by use of packages and private data types. In fact, making a type private has the immediate consequence of making it abstract, at least from the point of view of the user of the package. That is, the structural details of the type are hidden, and there are no literals that can be assigned to objects of the type. All that is visible to the user is the name of the type and the operations that are associated with the type. While this concept may seem over-restrictive, there is benefit to be obtained from the abstraction provided. That is, objects of an abstract data type are declared and manipulated in the same manner as any other objects. Th user need not concern himself with the details of manipulating objects of the type, and he is thus more able to concentrate on the higher level of design. Abstract data types can be provided as abstractions of the real world entities with which the program itself is concerned, helping further to manage the complexity of software by making it more understandable. An example of an abstract data type is the formation of a data type that is a first in-first out (FIFO) queue. The package specification is given below:

```
package QUEUE_OF_CHAR is
    type QUEUE_TYPE is limited private;
    procedure RESET(QUEUE_NAME : in out QUEUE_TYPE);
    procedure PUT(ELEM : in CHARACTER;
                  QUEUE_NAME : in out QUEUE_TYPE);
    procedure TAKE(ELEM : out CHARACTER;
                   QUEUE_NAME : in out QUEUE_TYPE);
    function IS_EMPTY(QUEUE_NAME : in QUEUE_TYPE)
                      return BOOLEAN;
    UNDERFLOW : exception;
private
    type QUEUE_ARRAY is array(1..4096) of CHARACTER;
    type QUEUE_TYPE is
        record
            VALUE : QUEUE_ARRAY;
            FRONT : INTEGER range 1..4096;
            BACK  : INTEGER range 1..4096;
        end record
end QUEUE_OF_CHAR
```

This package specification contains a visible part and a private part. Those entities declared in the visible part--type QUEUE_TYPE, subprograms RESET, PUT, TAKE, and IS_EMPTY, and exception UNDERFLOW--are visible to users of the package. Those entities declared in the private part--type QUEUE_ARRAY and the full declaration of type QUEUE_TYPE--are not visible outside the package itself. The declaration of type QUEUE_TYPE as a private type is visible outside the package, but the actual structure of the type as a record with two scalar components and one array component is not visible. The only operation permitted on objects of type QUEUE_TYPE outside the package are those provided by subprograms whose specifications appear in the package specification along with the private type declaration--in this case, RESET, PUT, TAKE, and IS_EMPTY. Within the package body, which must contain the bodies of subprograms RESET, PUT, TAKE, and IS_EMPTY, objects of type QUEUE_TYPE are three component records as specified in the private part of the package specification. Thus, any operation provided for records or their components can be used. The form of the package body would be:

```
package body QUEUE_OF_CHAR is
    procedure RESET(QUEUE_NAME :
                    in out QUEUE_TYPE is
    begin
        --sequence of statements of procedure RESET
    end RESET;
    procedure PUT(ELEM : in CHARACTER;
                  QUEUE_NAME in out QUEUE_TYPE) is
    begin
        --sequence of statements of procedure PUT
    end PUT;
    procedure TAKE(ELEM : out CHARACTER;
                   QUEUE_NAME : in out QUEUE_TYPE) is
    begin
        --sequence of statements of procedure TAKE
    end TAKE;
    function IS_EMPTY(QUEUE_NAME :
                      in QUEUE_TYPE) return BOOLEAN is
    begin
        --sequence of statements of function IS_EMPTY
    end IS_EMPTY;
end QUEUE_OF_CHAR;
```

Thus, in the package body where the FIFO queue is actually implemented, objects of type QUEUE_TYPE can be manipulated as records with one array component and two integer components. The queue itself is represented by the array, while the two integer components point to the front and back of the queue. Outside the package, however, objects of type QUEUE_TYPE are abstractions of FIFO queues and the user can only RESET, PUT, TAKE, and check IS_EMPTY. The actual implementation is hidden, in fact, a completely different implementation--for example, as a linked list--could be used in the private part of the specification and in the body, without any effect whatsoever on the user. As before, the package specification and the package body can be separately compiled.

The user of the FIFO queue package would typically declare objects of type QUEUE_TYPE. For instance, the declarations

```
QUEUE_1, QUEUE_2, QUEUE_3 : QUEUE_TYPE;
```

create three FIFO queue objects that are abstractions of the FIFO queue implemented in the package body. The user can then perform operations such as

```
PUT(ALPHA, QUEUE_1);
```

which puts the character object ALPHA on QUEUE_1, without concern for the detail of the actual implementation. It is, in fact, impossible for a user program to examine such detail, because the implementation is hidden in the package body. Similarly, the operations

```
RESET(QUEUE_2);
```

and

```
TAKE(ALPHA, QUEUE_3);
```

perform abstractions of queue operations. The reader should note that any number of objects of type QUEUE_TYPE can be declared. They may even be used as components of arrays or records. Operations on all objects of the type are provided by the package QUEUE_OF_CHAR.

Generic packages. Generic units in Ada are generic subprograms and generic packages. In Part 6 of this series the generic subprogram was discussed. Here, the declaration and instantiation of generic packages will be considered. As is the case with generic subprograms, the specification of a generic package has an additional part called the generic formal part, and this formal part may contain generic formal parameters. At instantiation, generic actual parameters are associated with the generic formal parameters. As is also the case with generic subprograms, a generic package may have type parameters, object parameters, and subprogram parameters. The use of a generic package can perhaps best be shown by means of an example. The FIFO queue package described above provides an abstract data type that is a FIFO queue of characters. Suppose that a similar structure that will provide FIFO queues for integer values is desired. Certainly a package could be developed to provide this service, and such a package would be identical to package QUEUE_OF_CHAR, with the exception that the data type CHARACTER would be replaced with the name of the desired integer data type. Alternatively, the FIFO queue package can be made a generic unit, for which the specification would be:

```
generic
    ELEM_TYPE is private
package GENERAL_QUEUE is
    type QUEUE_TYPE is limited private;
    procedure RESET(QUEUE_NAME :
                    in out QUEUE_QUEUE_TYPE);
    procedure PUT(ELEM : in ELEM_TYPE;
                  QUEUE_NAME : in out QUEUE_TYPE);
    procedure TAKE(ELEM : in ELEM_TYPE;
                   QUEUE_NAME : in out QUEUE_TYPE)
    function IS_EMPTY(QUEUE_NAME : in QUEUE_TYPE)
                      return BOOLEAN;
    UNDERFLOW : exception;
private
    type QUEUE_ARRAY is array(1..4096) of ELEM_TYPE;
    type QUEUE_TYPE is
        record
           VALUE : QUEUE_ARRAY;
           FRONT : INTEGER range 1..4096;
           BACK  : INTEGER range 1..4096;
        end record;
end GENERAL_QUEUE;
```

The reader will note that the only changes are the addition of the generic part and the substitution of the generic formal type parameter ELEM_TYPE for the type CHARACTER. As before, the specification and body may be separately compiled.

Package GENERAL_QUEUE is generic, and as such is not directly usable. An instance of the generic package must first be declared. The following instantiation

```
package INTEGER_QUEUE is new
        GENERAL_QUEUE(INTEGER);
```

produces package INTEGER_QUEUE, which provides an abstract data type that is a FIFO queue of objects of the predefined type INTEGER. The data types for which package GENERAL_QUEUE may be instantiated are not limited to scalar data types. For example, if the declarations

```
type WAY_POINT is
    record
       LATT : FLOAT;
       LONG : FLOAT;
    end record;
type WAY_PT_ARRAY is array (1..20) of WAY_POINT;
```

are in effect, then the instantiation

```
package WAY_PT_QUEUE is new
        GENERAL_QUEUE(WAY_PT_ARRAY);
```

will produce the implementation of a FIFO queue for objects of type WAY_PT_ARRAY, which are arrays of records. There is virtually no limit to the complexity of the data types for which package GENERAL_QUEUE can be instantiated.

There are features of and limitations to the package and generic features of the Ada language that are beyond the scope of the present discussion. However, even the simple examples presented here suffice to illustrate the usefulness of the package concept, the private data type, and the generic package facility of the language. Generic packages, in particular, hold promise for the development of a software industry through which reusable software components can be shared by many different programs.

NEXT: Ada's Tasks and Concurrent Programming.

Ada is a trademark of the U.S. Department of Defense.

Basic OS-9

Ron Voigts

EASE TIRED, ACHING FINGERS

Are your fingers tired of typing? Do you find yourself entering the same repetitive lines? Wish you could leave a list of things for your your computer to do, while you went a did more fun things? Well, you can with procedure files!

Normally, input and output occurs between you and the OS-9 shell. Actually, it's with the shell and your terminal, but eventually you end up pressing the keys and reading the display. If you input from another source into the shell, then you eliminate the the work. The way to do this, is to input to the shell from a file. Entering:

```
OS9:shell <some_file
```

causes another shell to be created whose input is from a file called "some_file". "Some_file" is a procedure file, since it lists instructions for the shell to carry out. The "<" redirects the standard input from the keyboard to the file. Every line in the file looks to the new shell as if it had been entered from the keyboard. There is a simpler way to start this procedure file. Just enter:

```
OS9:some_file
```

This accomplishes the same thing that was done before. The difference is the shell doesn't know initially that this is a procedure. At first, it assumes it could be an executable, program module. So, it checks memory to see if it's there. Not finding it, it goes to the execution directory, usually /D0/CMDS. It's not there either. Finally, it decides this must be a procedure file and looks in the working directory for it. If it finds it, the shell will create a new shell for it that inputs from the file, in our case it is "some_file" again. If it doesn't find it, you get an error #216 which is Bad Pathlist.

The most familiar procedure file is startup, located on /D0. What makes this file unique is on "boot up" of the system, startup is automatically run. Startup is a good place to set initial parameters of your system. What you put in it, is a matter of personal whim. Usually the simplest startup file (and probably the one that came with your system disk) is

```
SETIME </TERM
```

Setime sets the time and date. Why the "</TERM"? Remember, the standard input as been redirected to come from a file. Setime would run, try to get data from the rest of the file, and finding none would not set the time or date to anything meaningful. By adding the redirected input from the terminal, you can type in the necessary information. This sounds like a circular argument, if you think about it, it makes sense. A shell's input is redirected from the keyboard to a file. The file contains an OS-9 command, which in turn redirects its input from the keyboard again. The great thing is, it works!

You can customize startup to your own liking. My startup file looks like:

```
tmode .1 -upc
xmode /p baud=4
setime </term
```

This uses tmode to turn on the lower case. The ".1" is necessary since the input to the shell is from the file. Tmode changes the terminals configurations by affecting its path descriptor. The shell is using the file for its input, so the terminal is not effected , unless the ".1" is added. Then tmode changes the parameters for the standard output path #1. The xmode changes the device descriptor for the printer, so it can run at 2400 baud. Setime works like it did before.

I like to use procedure files while programming. It saves time and waiting. For example, when programming in Pascal, I first write the code using a word processor called Dynastar then I compile it with the Microware pascal compiler. I create a little file, I call "do". It reads:

```
de program </term
pascal <program >/p
```

All I do is run "do", edit the program and then quit dynastar. I don't have to wait for dynastar to finish the file before entering the pascal line. I save a few keystrokes and don't have to retype the same repetitive line. Later I add the u e:

```
pascaln pcodef
```

to "do". Now I can run the p-code file after it is compiled.

A good use of procedure files is with an OS-9 command called dsave. To use it to copy a disk in drive 0 to drive 1 use the following sequence:

```
chd /d0
dsave /d0 >tempfile
chd /d1
/d0/tempfile
```

Here dsave makes a file called "tempfile" on /D0 that contains the necessary information to copy the contents of /D0 to /D1. With a change of the working directory to /D1, "tempfile" is run. It uses **makdir, copy, load, tmode** and **unlink** to copy everything from /D0 to /D1. Here's another way to do it.

```
chd /d0
dsave /d0 /d1 ! shell
```

This one creates a file that copies /D0 to /D1, but the twist is the file is piped to a shell which in turns excepts it as input and begins to execute it.

Procedure files can serve many purposes. They can cause your computer to carry out tasks that would take much time to enter from the keyboard, like dsave does. Many repetitious task can be placed in a file, like the pascal programming example I used earlier. Later it can used when needed. I've noticed many software packages use procedure files to install their programs onto you disk. The uses are only limited by you imagination.

THE SOLUTION

One of the buzz words going around is "user friendly." This computer or that one contends that it is user friendly. And there are all types of gimmicks to make them friendly. Some are menu driven and you select an item from the menu. Others use icons (that's little pictures) that you select with a joystick or mouse. Others play question and answer to find out what you want to do. The purpose of all of them is to make life a little easier for you.

For those of you that looking for something to make OS-9 easier to use on your Coco, there is **THE OS-9 SOLUTION** marketed by Spectrum Associatee, Inc. The Solution is a menu driven program designed to give you 26 OS-9 commands at your fingertips. It purpose is "to make OS-9 an easier to use disk operating system." The commands are:;

```
<A>lphabetize Directory*
   <B>ackup Diskette
   <C>opy File
   <D>elete Directory or File
```

```
<E>xamine File
<F>ormat diskette
<G>oto a New Directory*
<H>elp Screens
<I>nformation On File *
<J>umble Around Filenames*
<K>ompare Files
<L>oad In A File To The Module Directory
<M>ake A New Directory
<N>ew Diskette
<O>utput File
<P>ut Files Together
<Q>uit and Exit To OS-9
<R>ename Current Filename*
<S>et Directories*
<T>ime and Date Set *
<U>p One Directory Level*
<V>erify File *
<W>ork Multiple File *
<X>mode Parameters Menu *
<Y>=Display Free Sectors*
<Z>=Execute Shell Command
```

The commands marked with the "*" are billed as commands that are not available under regular OS-9. The Alphabetize Directory or Jumble Around Filenames are two that you won't find. Information On Files is like DIR E and IDENT rolled into one. I question whether they are different in all cases. Like their Rename reminds me of the regular RENAME in OS-9. In the end they do the same thing.

The Solution lets you manipulate files by an arrow on the right hand of the screen (it is actually a =>). By pressing the Coco's up and down arrows you move the current directory files by it. When you are happy with what you are pointing at, you select the function. If you went to output to a device, you get another menu that shows the possible devices, like disk drives, and the printer. Again the devices are moved by the arrow for selection. The entire goal is to be able to work OS-9 totally from a menu.

This may be great for someone starting out, but I am not sure it is for everyone. Consider listing a file to the printer. In standard OS-9, I would enter:

```
list filename >/p
```

and "filename" would go to the printer. With the Solution I would move the file list up or down past the arrow until I found "filename". Then I would press O to output file. Another menu would come up asking for a direction. I would again move the possible devices past the arrow until I reached /p. Finally I would press S to select the device and "filename" would be listed to the printer. Now which is easier? I don't find the old way to be very difficult. In fact, I feel more in control with it.

The Solution is big. When I run it, it leaves me with about 13K of memory for other things. If I give it extra memory as they suggest (add a #20k on the end of the execute line), I have almost no memory left. You really can't do much else while it's running. It's not the type of program that you'll leave in memory while you do other things.

I'm not sure if the Solution is right for everyone. Most of its functions I feel more comfortable with using the old fashioned method. Some of the benefits are good. I like the Alphabetize Directory function and Work Multiple Files which lets you manipulate many files for copying, deleting, or whatever else you can do with a single file. If you're someone who likes a menu driven atmosphere or maybe you're just starting out and need the help, you might be interested in THE OS-9 SOLUTION. It is available from:

SPECTRUM PROJECTS
PO Box 21272
Woodhaven, NY 11421
718-441-2807
cost: $39.95 plus s/h

ANOTHER DIRECTORY ALPHABETIZER

As I said earlier, I like the directory alphabetizer function in the OS-9 Solution. Having the files in alphabetical order is not only esthetically pleasing, but it also can be useful. Imagine trying to locate a particular file in a large directory. You enter a DIR and file names spill out onto the screen. All you have to do is find one name in the crowd. Now put them in alphabetical order and quickly you spot the file. For those of you who don't have the OS-9 Solution, I've created a directory alphabetizer called **DALPHA**.

DALPHA is a C Language program. I compiled it on the Microware C Compiler, but I think it can be easily done on some of the other ones. The program opens the directory in the "update mode" which permits either reading or writing to it. If no parameters are passed to DALPHA, it sorts the current data directory, otherwise you can specify a particular one. I've set **LENGTH** to 150, the number of entries in the directory. You can change this if you like. Count is how many entries were actually read. If there is more than the maximum, the program aborts without doing anything.

The sort is a simple one, starting with the 3rd entry (the first two are . and .., with which we don't want to mess), the smallest entry is located and it is placed here. Then the next smallest is sought and so on until all the directory entries are sorted. The procedure swap switches two entries. Copy moves an entry form one place to another. Compare is the most important routine. It decides which of two entries is largest. If it returns a positive number the first is larger; if it is negative, the second is larger; and a zero means equal, which should be impossible. Any entries starting with a "/0" which is the bit value for a $00 are returned so that they go to bottom of the sort. These are old directory entries that were at some time erased. Instead of the entry being removed, a $00 was placed in the first position to flag it. Sometime later the slot may be reused for a directory entry, but for now we went them at the and of the directory.

At the end of the sort, the sorted directory is written back to the disk and the path to it is closed. This is the most critical part. If something has gone wrong and erroneous things have been put into the directory, your files (as you know them) may be destroyed. So here's a **WARNING!!** Be certain that DALPHA is running correctly. Try it on an old disk. Or better, backup some disk and use it. If things go wrong, just make another copy and try again. To coin an old phrase, "It is better to be safe, then sorry". Wiping out an important disk can make you very sorry!

That's all for this month. Crank out some procedure files of your own. Take some of those repetitive tasks and let OS-9 handle them for you. I think you'll really like DALPHA. It makes your directories look so much better. Bye!

```
/* Program to alphabetize directories          */
/* By Ron Voigts              July 8, 1985     */
/* For 68' Micro Journal, Basic OS-9 Readers    */
/* Usage:                                       */
/*     OS9:dalpha                               */
/*     OS9:dalpha /d0/A_DIRECTORY               */
/*     The first example alphabetizes the       */
/*     current working data directory.          */
/*     The second does a specified directory.   */


#include <stdio.h>
#define mask(c)  ((c)&'\137')
#define DIR 128
#define UPDATE 3
#define LENGTH 150
#define D_UPDATE DIR+UPDATE
#define E_READ 244
#define E_WRITE 245
#define DSIZE 32

main(argc, argv)
int argc;
char *argv[];
{
```

```
char entry[LENGTH][DSIZE];
int path, state, count, i, j, pos;
char *dirname=".";

/* check for too many parameters */
if (argc>2){
   printf("Too Many Parameters\n");
   exit(1);
}

/* check for alternate directory */
if (argc==2)
   dirname=argv[1];

/* open directory and read it */
if ((path=open(dirname, D_UPDATE)) == -1){
   printf("Can't Open %s\n",dirname);
   exit(1);
}
count=0;
while ((state=read(path,entry[count++],
                              DSIZE))>NULL)
   if (count == LENGTH){
      printf("Directory Too Large\n");
      exit(1);
   }
count--;
if (state==-1)
   exit(E_READ);

/* sort the directory in ascending order */
   for (i=2; i<count-1; i++) {
      pos=i;
      for (j=i+1; j<count; j++)
        if (compare(entry[pos],entry[j]) > 0)
           pos=j;
        if (i != pos)
           swap(entry[i], entry[pos]);
}

/* write out a directory */
if (lseek(path,0L,0) == -1)
   exit(1);
  for (i=0; i<count; i++)
     if (write(path, entry[i], DSIZE) == -1)
        exit(E_WRITE);
  close(path);
}

compare(s, t)
char *s, *t;
{
   int i;
   if (s[0] == '\0')
      return(1);
   if (t[0] == '\0')
      return(-1);
   i=0;
   while ( mask(s[i]) == mask(t[i])){
      if (s[i] > '\177')
         return(0);
      else if (t[i] > '\177')
         return(1);
      i++;
   }
   return(mask(s[i])-mask(t[i]));
}


swap(s, t)
char *s, *t;
{
   char temporary[DSIZE];
   copy(s, temporary);
   copy(t, s);
   copy(temporary, t);
}

copy(s1, s2)
char *s1, *s2;
{
   int i;
   for (i=0; i<DSIZE; i++)
      s2[i]=s1[i];
}
```

UniFLEX User Notes

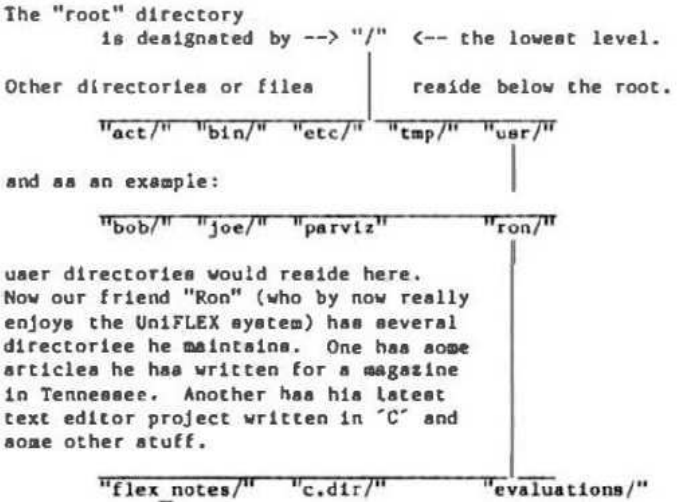
Kenneth R. Lewis
Automation Engineering
1691 Shelby Oak Dr. N.
Suite 4
Memphis, TN 38134

For years now I (among many others) have been using Technical Systems Consultants' UniFLEX operating system. UniFLEX is kinda like the "big brother" of FLEX Operating System which is enormously popular all over the world.

UniFLEX is a multi-tasking, multi-user operating system, written in assembler language and runs on the M6809, M68000, M68010 and M68020 microprocessors. UniFLEX is tailored for the machine architecture of these processors and takes full advantage of each machines' instruction set. More importantly, UniFLEX allows for much more sophisticated tools and techniques while developing software. More than one developer may participate, and ... more than one product may be developed at the same time!

UniFLEX, like UNIX, offers a variety of sophisticated mechanisms for management of the user environment. One such "feature" is of course a "hierarchical" file system. This one feature alone is probably the least appreciated aspect of UNIX-like systems by the non-UNIX user. The easiest way to understand it is to visualize it, and the best way to appreciate it is to see it as way of properly organizing similar files, eg: by project, or by language, or by subject..

The UniFLEX file system is organized with a root directory as the starting point for all filenames in the system. Remember, FLEX filenames actually start with their "diskdrive number." Obviously, FLEX allows the "drivespec" to be either before, after or implied, but we've gotten used to that now haven't we? The same holds true for a UniFLEX directory (or path name). You either specify a full pathname, or it is implied to be the "current" path name. For example:



It is easy to see that "ron" has his act together and can find any file he wants with little effort! Why there is even a utility to search for a file if he should forget where he put it. Afterall, "ron" could have written a "C" program as a quick test, reported the results to his readers (remember the magazine) in his "flex_notes/" files but forget to move it into the actual article.

The simplistic file system on our example UniFLEX system is intended to be visual aid for this discussion. It implies that practically any useful combination of directories and files could be constructed.

The most useful thing about UniFLEX file hierarchy is that things are "organized." For example, the most frequently used utilities and programs reside in the "bin/" directory (the directory that is searched first by the command interpreter...the "shell"), less frequently used programs resided in a "bin/" directory at the "usr/" level (these are usually extended function utilities, user written utilities, etc.) Finally a directory called "bin/" in the users "home" directory is searched. The "home" directory is kinda like a "work drive" on a single-user system.

I will describe in detail the "in's and out's" of UniFLEX over the next few articles and pass along any tricks I've learned so that new users will enjoy the power and flexibility earlier in the learning curve.

UniFLEX has grown. Yes, grown along with the user base. As more complex tools, languages and support mechanisms were required, they were developed to meet the needs. When UniFLEX was first introduced (early 1980), several familiar tools came with it. The standard TSC Text Editor, the Extended TSC BASIC and it's pre-compiler were among the first. Soon to follow were PASCAL, FORTRAN 77, COBOL, an Enhanced Print Spooler package, and two Utilities Packages. Additionally, the TSC Text Processor (used for years by this magazine) and a little known, but very powerful Sort/Merge Package were slipped in for good measure.

TSC wasn't alone in producing tools for UniFLEX users. Over in England and western Europe, many groups were developing products that would take advantage of UniFLEX. James McCosh has captured most of the attention by introducing his 'C' compiler, a fine piece of work, that not only ran under UniFLEX, but FLEX and OS-9 (by Microware). In Buffalo, New York, a company headed by Joel Heckman (UDRI) introduced a set of Data Base Management tools along with a complete Business Management Package designed to run under UniFLEX BASIC. Stylograph introduced their fine word processor. The list goes on and on, far to numerous to include here.

One of the wildest tools introduced for the 6809 UniFLEX users is "FLEX under UniFLEX." With few restrictions, any properly written program, language or utility designed to run under the single-user FLEX operating system can be run (in a multi-user environment) under UniFLEX! Just think of it....practically the best of both worlds. FLEX under UniFLEX comes with a set of standard Utilities and some special ones. The special tools allow a user to create "logical" disk drives on a disk system mounted under UniFLEX. Regular single-density FLEX disks can be used without modification on a UniFLEX floppy device. Even UniFLEX and FLEX files may be freely exchanged between environments. I have enjoyed this feature on many occasions while developing special industrial controller packages under UniFLEX, then moving them to the FLEX environment for in-depth testing.

Early users of UniFLEX were perplexed at the inability to gain "direct" access to hardware devices and memory. UniFLEX purposefully restricts such accesses in order to prevent it's carefully controlled environment from being "bombed" by a careless programming error, or worse...intentional prying. TSC has invested considerable effort in protecting the user from destroying his files, crashing other users or the entire system. So the big question became..."How can we use special devices like 'video boards,' or 'tape controllers?' The answer, certain UniFLEX versions have a system level function that allocates memory space, or a particular hardware device as a portion of the user's program space. After the function call, the user simply treats the allocated resource like any other resource and diddles bits, or whatever!

Recent Endeavors

For the past year, I have been pursuing perhaps the most exciting work of my career, evaluating UNIX-like operating systems for a large package handling company (you know...absolutely, positively overnight, etc.!). There were several constraints and even more "gotta haves" for such an operating system, but the most important were:

- It MUST work like UNIX
- It MUST support 'C' language development
- It MUST be portable across a hardware family
- It MUST be very F A S T !!!

With these (and other) criterion clearly in mind, a diligent search was begun to find the ideal operating system for our chosen hardware. I should explain, and I will that we had already researched and selected an appropriate hardware base for our package handling systems. Since there is quite a body of information in that subject alone, I will present it in a future article, in detail!

You must remember that almost every hardware vendor has his favorite (sometimes the only one they could afford) operating system. If you listen to the vendor, they would have you believe that even the most complex project development effort is a "snap" with their UNIX-derived, UNIX-port, son-of-UNIX, whatever...operating system. Believe me, nothing could be farther from the truth. The sad fact is that pure, non-optimized UNIX is one of the "dogiest" (a technical term meaning slow, lazy, fat, ugly, etc.) operating systems that can be in charge of a microprocessor! Too many hardware vendors, whilst pursuing that almighty dollar, will tell the "un-enlightened" anything they think they want to hear. As of late, most people want to hear "UNIX," or something like that whenever they are trying to convince their company to purchase new technology. Furthermore, if these same vendors even think that performance could be an issue they will offer to "benchmark" (another technical term meaning 'here is a program that is constructed so that it runs fastest on our product') their machine and of course, will offer to provide same! Sadly, the systems look like UNIX, run like molasses and eventually are discarded in favor of more expensive, less appropriate technology.

Now, having said all that, let me now say that I wanted an operating system that could run on a microprocessor (specifically the M68000 family) and would be portable across the family of products offered. In some cases, there would be 'real-time' requirements. In most cases, there would be several tasks running, each communicating with the other and each dormant when not doing their particular job. The operating system had to be easily configured, low cost, well supported and documented. From the developers point-of-view, the operating system should not impose any restrictions. For all practical purposes, it should look like UNIX!

Many UNIX-like systems were examined and benchmarked. The documentation for each was thoroughly examined. Certain "pet" tests were conducted by interested parties during this research period. Whenever a difference (or discrepancy) was discovered, it was explored and a determination made about possible impact on development. Another phenomenon emerged during this period of examination and testing. Some hardware manufacturer showed a willingness to cooperate! Yes, actually cooperate with me and provide support for the effort. It seems that some of the manufacturers had been "lead down the garden path" during the early days of their product marketing efforts and most of them had been lead to believe that what the public wanted was UNIX, or

K-BASIC updates are now available. If you purchased K-BASIC prior to July 1, 1985 and wish to have your K-BASIC updated, please send $35 enclosed with your master disk to Southeast Media.

K-BASIC under OS-9 and FLEX will now compile TSC BASIC, XBASIC, and XPC Source Code Files

TELEX 558 414 PVT STH
(615)842-4600
SOUTH EAST MEDIA
5900 Cassandra Smith Rd.
Hixson, TN 37343
for information
call (615) 842-4601
CoCo OS-9™ FLEX™
SOFTWARE

K-BASIC now makes the multitude of TSC MBASIC Software available for use under OS-9. Transfer your favorite BASIC Programs to OS-9, compile them, Assemble them, and DING! -- usable, multi-precision, familiar Software is running under your favorite Operating System!

K-BASIC (OS-9 or FLEX), including the Assembler
$199.00

SCULPTOR

!!! New Prices !!!

Microprocessor Developments Ltd.'s Commercial Application Generator Program provides a **FAST** Commercial Application Development tool unavailable to the OS-9 and UniFLEX User before. Develop any Commercial Application in 20% of the normal required time; gain easy updating or customizing. PLUS, the Application can also be run on MS-DOS and Unix machines! **Sculptor** handles input validation, complex calculations, and exception conditions as well as the normal collecting, displaying, reporting, and updating information in an orderly fashion. Key fields to 160 bytes; unlimited record size; file size should be held to 17 million records. Utilizes ISAM File Structure and B-tree Key files for rapid access. Input and Output communication with other programs and files plus a library of ISAM routines for use with C Programs. Run-time included w/ the Development package; a compiled Application only needs a Run-time Licensee. Additional charge for Networked Units. Prices for **Development Package/Run-time**. Discounts available for purchases of 5 or more Run-time Packages.

OS-9 / UniFLEX --			68000 UniFLEX --			MS DOS --	
IBM PC Xenix --	$995.00/$175.00		Altos Xenix --	$1595.00/$265.00		--	$595.00/$115.00
MS DOS Network --	*	**	UNIX --	*	**	PC DOS --	* **

* Full Development Package ** Run Time Package Only Full OEM and Dealer Discounts Available!

!!! Special Buy Out !!! **SPECIAL** - Limited Quantity !!! Special Buy Out !!!

6809 FLEX SOFTWARE		
TSC Flex Utilities	was $75.00	Now only $50.00
TSC Basic	was $75.00	Now only $35.00
TSC Extended Precompiler	was $50.00	Now only $35.00
TSC Text Processor	was $75.00	Now only $50.00
TSC Flex Precompiler	was $50.00	Now only $35.00
TSC Flex Basic	was $75.00	Now only $50.00
TSC Flex Diagnostic	was $75.00	Now only $50.00
TSC Text Processor	was $75.00	Now only $50.00
TSC Assembler	was $50.00	Now only $35.00
TSC Debug	was $75.00	Now only $50.00
TSC Precompiler	was $50.00	Now only $35.00
TSC Editor	was $50.00	Now only $35.00
TSC Sort/Merge	was $75.00	Now only $50.00
TSC Utilities	was $75.00	Now only $50.00
DISKS ONLY		
TSC Extended Precompiler	was $50.00	Now only $35.00
TSC Basic Flex	was $75.00	Now only $50.00
TSC Diagnostics	was $75.00	Now only $50.00
TSC Utilities	was $75.00	Now only $50.00

MANUALS ONLY		
TSC Basic Precompiler	was $25.00	Now only $20.00
TSC Text Editor	was $25.00	Now only $20.00
TSC Debug	was $25.00	Now only $20.00
TSC Mnemonic Assembler	was $25.00	Now only $20.00
COMPLETE 6800 SOFTWARE		
TSC Text Processor	was $50.00	Now only $35.00
TSC Precompiler	was $50.00	Now only $35.00
TSC Diagnostics	was $75.00	Now only $50.00
DISKS ONLY - 6800 Software		
TSC Editor	was $50.00	Now only $35.00
TSC Utilities	was $100.00	Now only $70.00
TSC Assemblers	was $50.00	Now only $35.00
MANUALS ONLY - 6800 Software		
TSC Diagnostics	was $25.00	Now only $20.00
TSC Debug	was $25.00	Now only $20.00
TSC Text Processor	was $25.00	Now only $20.00

!!! Please Specify Your Operating System & Disk Size !!!

TELEX 558 414 PVT BTH

(615)842-4600

SOUTH EAST MEDIA

5900 Cassandra Smith Rd.
Hixson, TN 37343

for information
call (615) 842-4601

CoCo OS-9™ FLEX™

SOFTWARE

PROGRAMMING

ASSEMBLERS

ASTRUK09 from **Southeast Media** -- A "Structured Assembler for the 6809" which requires the **TSC Macro Assembler.** F, CCF - **$99.95**

Macro Assembler for **TSC** -- The FLEX **STANDARD** Assembler.
Special -- CCF **$35.00**; F **$50.00**

OSA Extended 680 Macro Assembler from **Lloyd I/O.** -- Provides local labels, Motorola S-records, and Intel Hex records; XREF. Generate OS-9 Memory modules under FLEX. FLEX, CCF, OS-9 **$99.00**

Relocating Assembler w/Linking Loader from TSC. -- Use with many of the C and Pascal Compilers. F,CCF **$150.00**

MACE, by Graham Trott from **Windrush Micro Systems** -- Co-Resident Editor and Assembler; fast interactive A.L. Programming for small to medium-sized Programs. F,CCF - $75.00

XMACE -- MACE w/ Cross Assembler for 6800/1/2/3/8 F,CCF - **$98.00**

TRUE CROSS ASSEMBLERS from Computer Systems Consultants -- Supports 1802/5, Z-80, 6800/1/2/3/8/11/HC11, 6804, 6805/HC05/146805, 6809/00/01, 5502 family, 8080/5, 8020/1/2/35/C35/39/40/48/C48/49/C49/50/8748/49, 8031/51/8751, and 68000 Systems. Assembler and Listing formats same as target CPU's format. Produces machine independent Motorola S-Text.
FLEX, CCF, OS-9, UniFLEX each - **$50.00**
any 3 - **$100.00**
the complete set w/ C Source (except the 68000 Source) - **$200.00**

XASM Cross Assemblers for **FLEX** from **Compusense Ltd.** -- This set of 6800/1/2/3/5/8, 6301, 6502, 8080/5, and Z80 Cross Assemblers uses the familiar **TSC Macro Assembler** Command Line and Soruce Code format, Assembler options, etc., in providing code for the target CPU's. Complete set, FLEX only - **$150.00**

CRASMB from **Lloyd I/O** -- 8-Bit Macro Cross Assembler with same features as OSM; cross-assemble to 6800/1/2/3/4/5/8/9/11, 6502, 1802, 8048 Sers, 80/85, Z-8, Z-80, TMS-7000 sers. Supports the target chip's standard mnemonics and addressing modes.
FLEX, CCF, OS-9 Full package -- **$399.00**

CRASMB 16.32 from **Lloyd I/O** -- Cross Assembler for the **68000.**
FLEX, CCF, OS-9 **$249.00**

DISASSEMBLERS

SUPER SLEUTH from **Computer Systems Consultants** -- Interactive Disassembler; extremely POWERFUL! Disk File Binary/ASCII Examine/Change, Absolute or FULL Disassembly. XREF Generator, Label "Name Changer", and Files of "Standard Label Names" for different Operating Systems

Color Computer	SS-50 Bus (all w/ A.L. Source)
CCO (32K Req'd) Obj. Only $49.00	F, $99.00
CCF, Obj. Only $50.00	U, $100.00
CCF, w/Source $99.00	O, $101.00
CCO, Obj. Only $50.00.	

DYNAMITE + from **Computer Systems Center** -- Excellent standard "Batch Mode" Disassembler. Includes XREF Generator and "Standard Label" Files. Special OS-9 options w/ OS-9 Version.

CCF, Obj. Only $100.00	CCO, Obj. Only $ 59.95
F, " " $100.00	O, " " $150.00
	U, " " $300.00

PROGRAMMING LANGUAGES

PL/9 from **Windrush Micro Systems** -- By Graham Trott. A combination Editor/Compiler/Debugger. Direct source-to-object compilation delivering fast, compact, re-entrant, ROM-able, PIC. 8 & 16-bit Integers & 6-digit Real numbers for all real-world problems. Direct control over ALL System resources, including interrupts. Comprehensive library support; simple Machine Code interface; step-by-step tracer for instant debugging. 500+ page Manual with tutorial guide. F, CCF - **$198.00**

MUSICAL from **Musical Developments** -- Now supports **Real Numbers.** "Structured Programming" WITHOUT losing the Speed and Control of Assembly Language! Single-pass Compiler features unified, user-defined I/O; produces ROMable Code; Procedures and Modules (including pre-compiled Modules); many "Types" up to 32 bit Integers, **6-digit Real Numbers**, unlimited sized Arrays (vectors only); Interrupt handling; long Variable Names; Variable Initialization; Include directive; Conditional compiling; direct Code Insertion; control of the Stack Pointer; etc. Run-Time subroutines inserted as called during compilation. Normally produces 10% less code than PL/9. F and CCF - **$195.00**

C Compiler from Windrush Micro Systems by James McCosh. Full C for FLEX except bit-fields, including an Assembler. Requires the TSC Relocating Assembler if user desires to implement his own Libraries. F and CCF - **$295.00**

C Compiler from Introl -- Full C except Doubles and Bit Fields, streamlined for the 6809. Reliable Compiler; FAST, efficient Code. More UNIX Compatible than most.
F, CCF, and O - **$375.00** U - **$425.00**

PASCAL Compiler from Lucidata -- ISO Based P-Code Compiler. Designed especially for Microcomputer Systems. Allows linkage to Assembler Code for maximum flexibility.
F and CCF 5" - **$190.00** F 8" - **$205.00**

PASCAL Compiler from **OmegaSoft** (now **Certified Software**) — For the **PROFESSIONAL**; ISO Based, Native Code Compiler. Primarily for Real-Time and Process Control applications. Powerful; Flexible. Requires a "Motorola Compatible" Relo. Asmb. and Linking Loader.
F and CCF - **$425.00** One Year Maint. - **$100.00**

K-BASIC from LLOYD I/O -- A "Native Code" **BASIC Compiler** which is now **Fully TSC XBASIC compatible.** The compiler compiles to Assembly Language Source Code. A NEW, **streamlined**, Assembler is now included allowing the assembly of LARGE Compiled K-BASIC Programs. Conditional assembly reduces Run-time package.
FLEX, CCF, OS-9 Compiler **with** Assembler - **$199.00**

CRUNCH COBOL from **Compusense** Ltd. -- Supports large subset of ANSII Level 1 COBOL with many of the useful Level 2 features. Full FLEX File Structures, including Random Files and the ability to process Keyed Files. Segment and link large programs at runtime, or implemented as a set of overlays. The System requires 56K and CAN be run with a single Disk System.
FLEX, CCF; Normally $199.00
Special Introductory Price (while in effect) -- **$99.95**

FORTH from **Stearns Electronics** -- A CoCo **FORTH** Programming Language. **Tailored to the CoCo!** Supplied on Tape, transferable to disk. Written in FAST ML. Many CoCo functions (Graphics, Sound, etc.). Includes an Editor, Trace, etc. Provides CPU Carry Flag accessibility, Fast Task Multiplexing, Clean Interrupt Handling, etc. for the "Pro". Excellent "Learning" tool!
Color Computer **ONLY** - **$58.95**

VISA

** SHIPPING **
Add 3% U.S.A.
(min. $3.50)
Add 5% Surface Foreign
10% Air Foreign

*FLEX is a trademark of Technical Systems Consultants
*OS9 is a trademark of Microware

SOUTH EAST MEDIA
5900 Cassandra Smith Rd. Hixson, TN 37343
info (615) 842-4601
CoCo OS-9™ FLEX™ SOFTWARE

Availability Legends —
F = FLEX, **CCF** = Color Computer FLEX
O = OS-9, **CCO** = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

!!! Please Specify Your Operating System & Disk Size !!!

SOFTWARE DEVELOPMENT

BasicO9 XRef from **Southeast Media** -- This **BasicO9 Cross Reference** Utility is a BasicO9 Program which will produce a "pretty printed" listing with each line numbered, followed by a complete cross referenced listing of all variables, external procedures, and line numbers called. Also includes a **Program List Utility** which outputs a fast "pretty printed" listing with line numbers. Requires BasicO9 or RunB.
O & CCO obj. only -- $39.95; w/ Source - $79.95

Lucidata PASCAL UTILITIES (Requires LUCIDATA Pascal ver 3)

XREF -- produce a Cross Reference Listing of any text; oriented to Pascal Source.

INCLUDE -- Include other Files in a Source Text, including Binary; unlimited nesting capabilities.

PROFILER -- provides an Indented, Numbered, "Structogram" of a Pascal Source Text File; view the overall structure of large programs, program integrity, etc. Supplied in Pascal Source Code; requires compilation.

P, CCF --- **EACH Utility** 5" - $40.00, 8" - $50.00

DUB from **Southeast Media** -- A **UniFLEX** "basic" De-Compiler. Re-Create a Source Listing from UniFLEX **Compiled basic** Programs. Works w/ ALL Versions of 6809 UniFLEX basic. U - $219.95

FULL SCREEN FORMS DISPLAY from **Computer Systems Consultants** -- **TSC Extended** BASIC program supports any Serial Terminal with Cursor Control or Memory-Mapped Video Displays; substantially extends the capabilities of the Program Designer by providing a table-driven method of describing and using Full Screen Displays.
P and CCF, U - $25.00, w/ Source - $50.00

DISK UTILITIES

OS-9 VDisk from **Southeast Media** -- For **Level I** only. Use the **Extended Memory** capability of your SWTPC or Glmix CPU card (or similar format DAT) for FAST Program Compiles, CMD execution, high speed inter-process communications (without pipe buffers), etc. - SAVE that System Memory. Virtual Disk size is variable in 4K increments up to 960K. Some Assembly Required.
-- **Level I ONLY** -- OS-9 obj. only - $79.95; w/ Source - $149.95

O-F from **Southeast Media** -- Written in **BASICO9 (with** Source), includes: **REFORMAT**, a BASICO9 Program that reformats a chosen amount of an OS-9 disk to FLEX Format so it can be used normally by FLEX; and **FLEX**, a BASICO9 Program that does the actual **read or write function to the special O-F Transfer Disk; user-friendly** menu driven. Read the FLEX Directory, Delete FLEX Files, Copy both directions, etc. FLEX users use the special disk just like any other FLEX disk. O - $79.95

COPYMULT from **Southeast Media** -- Copy **LARGE** Disks to several **smaller disks.** FLEX utilities allow the backup of **ANY size disk** to any SMALLER size diskettes (Hard Disk to floppies, 8" to 5", etc.) by simply inserting diskettes as requested by **COPYMULT**. No fooling with directory deletions, etc. **COPYMULT.CMD** understands normal "copy" syntax and keeps up with files copied by maintaining directories for both host and receiving disk system. Also includes **BACKUP.CMD** to download any size "random" type file; **RESTORE.CMD** to restructure copied "random" files for copying, or recopying back to the host system; and **FREELINK.CMD** as a "bonus" utility that "relinks" the free chain of floppy or hard disk, eliminating fragmentation.
Completely documented Assembly Language Source files included.
ALL 4 Programs (FLEX, 8" or 5") $99.50

COPYCAT from Lucidata -- Pascal NOT required. Allows reading TSC Mini-FLEX, SSB DOS68, and Digital Research CP/M Disks while operating under FLEX 1.0, FLEX 2.0, or FLEX 9.0 with 6800 or 6809 Systems. COPYCAT will not perform miracles, but, between the program and the manual, you stand a good chance of accomplishing a transfer. Also includes some Utilities to help out. Programs supplied in Modular Source Code (Assembly Language) to help solve unusual problems.
F and CCF 5" - $50.00 F 8" - $65.00

** SHIPPING **
Add 2% U.S.A.
(min. $2.50)
Add 5% Surface Foreign
10% Air Foreign

*FLEX is a trademark of Technical Systems Consultants
*OS9 is a trademark of Microware

SOUTH EAST MEDIA
5900 Cassandra Smith Rd.
Hixson, TN 37343
info (615) 842-4601
CoCo OS-9™ FLEX™
SOFTWARE

TELEX 558 414 PVT BTH
(615)842-4600
SOUTH EAST MEDIA
5900 Cassandra Smith Rd.
Hixson, TN 37343
for information
call (615) 842-4601
CoCo OS-9™ FLEX™
SOFTWARE

FLEX DISK UTILITIES from **Computer Systems Consultants** -- Eight (8) different Assembly Language (w/ Source Code) FLEX Utilities for every FLEX Users Toolbox: **Copy** a File with **CRC Errors; Test** Disk for errors; **Compare** two Disks; a fast **Disk Backup** Program; **Edit Disk Sectors; Linearize Free-Chain** on the Disk; print **Disk Identification;** and **Sort** and **Replace** the Disk Directory (in sorted order). -- **PLUS** -- Two XBASIC Programs including: A **BASIC Renumberer** with **EXTRAS** over "RENUM" like check for missing label definitions, processes Disk to Disk instead of in Memory, etc. Other programs **Compare, Merge,** or **Generate Updates** between two BASIC Programs, check BASIC **Sequence Numbers,** compare two **unsequenced** files, and 5 Programs for establishing a **Master Directory** of several Disks, and sorting, selecting, updating, and printing paginated listings of these files. A BASIC Cross-Reference Program, written in Assembly Language, which provides an X-Ref Listing of the Variables and Reserved **Words in TSC BASIC, XBASIC,** and **CONSOLEFLEX BASIC Programs.**
ALL Utilities **include Source** (either BASIC or A.L. Source Code).
F and CCF - $50.00
BASIC Utilities ONLY for **UniFLEX** -- $30.00

COMMUNICATIONS

CMODEM Telecommunications Program from **Computer Systems Consultants, Inc.** -- Menu-Driven; supports Dumb-Terminal Mode, Upload and Download in non-protocol mode, and the CP/M "Modem7" Christensen protocol mode to enable communication capabilities for almost any requirement. Written in "C".
FLEX, CCF, OS-9, UniFLEX; with complete Source - $100.00
without Source - $50.00

XDATA from **Southeast Media** -- A **COMMUNICATION** Package for the UniFLEX Operating System. Use with CP/M, Main Frames, other UniFLEX Systems, etc. Verifies Transmission using checksum or CRC; Re-Transmits bad blocks, etc. U - $299.99

GAME

RAPIER - 6809 **Chess** Program from **Southeast Media** -- Requires FLEX and Displays on Any Type Terminal. Features: Four levels of play. Swap side. Point scoring system. Two display boards. Change skill level. Solve Checkmate problems in 1-2-3-4 moves. Make move and swap sides. Play white or black. This is one of the strongest CHESS programs running on any microcomputer, estimated USCF Rating 1600+ (better than most 'club' players at higher levels). F and CCF - $79.95

Availability Legends —
F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCO = Color Computer Disk
CCT = Color Computer Tape

!!! Please Specify Your Operating System & Disk Size !!!

TELEX 558 414 PVT BTH

(615)842-4600

SOUTH EAST MEDIA

5900 Cassandra Smith Rd.
Hixson, TN 37343

for information
call (615) 842-4601

CoCo OS-9™ FLEX™

SOFTWARE

BUSINESS

WORD PROCESSING

SCREDITOR III from Windrush Micro Systems -- Powerful Screen-Oriented Editor/Word Processor. Almost 50 different commands; over 300 pages of Documentation with Tutorial. Features Multi-Column display and editing, "decimal align" columns (AND add them up automatically), multiple keystroke macros, even/odd page headers and footers, imbedded printer control codes, all justifications, "help" support, store common command series on disk, etc. Use supplied "set-ups", or remap the keyboard to your needs. Except for proportional printing, this package will DO IT ALL!

6800 or 6809 FLEX or SSB DOS, OS-9 - $175.00

STYLO-GRAPH from Great Plains Computer Co. -- A full-screen oriented WORD PROCESSOR — (uses the 51 x 24 Display Screens on CoCo FLEX/STAR-DOS, or PBJ Wordpak). Full screen display and editing; supports the Daisy Wheel proportional printers.
NEW PRICES —> CCF and CCO - $99.95, F or O - $179.95, U - $299.95

STYLO-SPELL from Great Plains Computer Co. -- Fast Computer Dictionary. Complements Stylograph.
NEW PRICES —> CCF and CCO - $69.95, F or O - $99.95, U - $149.95

STYLO-MERGE from Great Plains Computer Co. -- Merge Mailing List to "Form" Letters, Print multiple Files, etc., through Stylo.
NEW PRICES —> CCF and CCO - $59.95, F or O - $79.95, U - $129.95

JUST from Southeast Media -- Text Formatter developed by Ron Anderson; for Dot Matrix Printers, provides many unique features. Output "Formatted" Text to the Display. Use the FPRINT.CMD supplied for producing multiple copies of the "Formatted" Text on the Printer INCLUDING IMBEDDED PRINTER COMMANDS (very useful at other times also, and worth the price of the program by itself). "User Configurable" for adapting to other Printers (comes set up for Epson MX-80 with Graftrax); up to ten (10) imbedded "Printer Control Commands". Compensates for a "Double Width" printed line. Includes the normal line width, margin, indent, paragraph, space, vertical skip lines, page length, page numbering, centering, fill, justification, etc. Use with PAT or any other editor.

* Now supplied as a two disk set:
Disk #1: JUST2.CMD object file, JUST2.TXT PL9 source: FLEX - CC
Disk #2: JUSTSC object and source in C: FLEX - OS9 - CC

The JTSC and regular JUST C source are two separate programs. JTSC compiles to a version that expects TSC Word Processor type commands, (.pp .mp .ce etc.) Great for your older text files.

The C source compiles to a standard syntax JUST.CMD object file. Using JUST syntax (.p .u .y etc.) With all JUST functions plus several additional printer formatting functions. Reference the JUSTSC C source. For those wanting an excellent BUDGET PRICED word processor, with features some of the others have. This is it!

Disk (1) - PL9 FLEX Version only - F & CCF - $49.95
Disk Set (2) - F & CCF & OS9 (C version) - $69.95

SPELLB "Computer Dictionary" from **Southeast Media** -- OVER 120,000 words! Look up a word from within your Editor or Word Processor (with the SPH.CMD Utility which operates in the FLEX UCS). Or check and update the Text after entry; ADD WORDS to the Dictionary, "Flag" questionable words in the Text, "View a word in context" before changing or ignoring, etc. SPELLB first checks a "Common Word Dictionary", then the normal Dictionary, then a "Personal Word List", and finally, any "Special Word List" you may have specified. SPELLB also allows the use of Small Disk Storage systems.

F and CCF - $129.95

DATA BASE - ACCOUNTING

XDMS from Westchester Applied Business Systems -- Powerful DBMS; M.L. program will work on a single sided 5" disk, yet is F-A-S-T. Supports Relational, Sequential, Hierarchical, and Random Access File Structures; has Virtual Memory capabilities for Giant Data Bases. XDMS Level I provides an "entry level" System for defining a Data Base, entering and changing the Data, and producing Reports. XDMS Level II adds the POWERFUL "GENEGATE" facility with an English Language Command Structure for manipulating the Data to create new File Structures, Sort, Select, Calculate, etc. XDMS Level III adds special "Utilities" which provide additional ease in setting up a Data Base, such as copying old data into new Data Structures, changing System Parameters, etc.

XDMS System Manual - $24.95
XDMS Lvl I - F & CCF - $129.95
XDMS Lvl II - F & CCF - $199.95
XDMS Lvl III - F & CCF - $269.95

ACCOUNTING PACKAGES -- **Great Plains Computer Co.** and **Universal Data Research, Inc.** both have Data Base and Business Packages written in TSC xBASIC for FLEX, CoCo FLEX, and UniFLEX.

MISCELLANEOUS

TABULA RASA SPREADSHEET from Computer Systems Consultants -- TABULA RASA is similar to DESKTOP/PLAN; provides use of tabular computation schemes used for analysis of business, sales, and economic conditions. Menu-driven; extensive report-generation capabilities. Requires TSC's Extended BASIC.

F and CCF, U - $50.00, w/ Source - $100.00

DYNACALC from Computer Systems Center -- Electronic Spread Sheet for the 6809.

F and SPECIAL CCF - $200.00, U - $395.00

FULL SCREEN INVENTORY/MRP from Computer Systems Consultants -- Use the Full Screen Inventory System/Materials Requirement Planning for maintaining inventories. Keeps item field file in alphabetical order for easier inquiry. Locate and/or print records matching partial or complete item, description, vendor, or attributes; find backorder or below stock levels. Print-outs in item or vendor order. MRP capability for the maintenance and analysis of Hierarchical assemblies of items in the inventory file. Requires TSC's Extended BASIC.

F and CCF, U - $50.00, w/ Source - $100.00

FULL SCREEN MAILING LIST from Computer Systems Consultants -- The Full Screen Mailing List System provides a means of maintaining simple mailing lists. Locate all records matching on partial or complete name, city, state, zip, or attributes for Listings or Labels, etc. Requires TSC's Extended BASIC.

F and CCF, U - $50.00, w/ Source - $100.00

DIET-TRAC Forecaster from Southeast Media -- An XBASIC program that plans a diet in terms of either calories and percentage of carbohydrates, proteins and fats (C P G%) or grams of Carbohydrate. Protein and Fat food exchanges of each of the six basic food groups (vegetable, bread, meat, skim milk, fruit and fat) for a specific individual. Sex, Age, Height, Present Weight, Frame Size, Activity Level and Basal Metabolic Rate for normal individual are taken into account. Ideal weight and sustaining calories for any weight of the above individual are calculated. Provides number of days and daily calendar after weight goal and calorie plan is determined.

F - $99.95, U - $89.95

VISA MasterCard

** SHIPPING **
Add 2% U.S.A.
(min. $2.50)
Add 5% Surface Foreign
10% Air Foreign

*FLEX is a trademark of Technical Systems Consultants
*OS9 is a trademark of Microware

Availability Legends —
F = FLEX, CCF = Color Computer FLEX
O = OS-9, CCO = Color Computer OS-9
U = UniFLEX
CCD = Color Computer Disk
CCT = Color Computer Tape

!!! Please Specify Your Operating System & Disk Size !!!

near-UNIX. After a period, most learned that what the public really wanted was performance! After six months of effort and dozens of systems, the clear winner was UniFLEX! Since that time several complex package handling systems have been developed (although not with the "..overnight" people) and are working well. Ironically, two of the systems are sorting, tracking and palletizing A.T.&T. Personal Computers!

In future articles I will describe UniFLEX-driven systems, their hardware configurations, tools, languages and applications packages. Also, as promised, I will describe the hardware base (Versa-Module European . VME) that is becoming the most popular configuration for M68XXX processing systems. I also have some very powerful graphics system information for you and how to employ UniFLEX as a development/target environment for same.

I also plan to review as many products as possible and share with each of you the results. What I have decided to do is provide a cursory review (of a particular product) here in the magazine, and if anyone is interested in the details, write me at our business address.

ISAM

"ISAM"
INDEXED SEQUENTIAL ACCESS METHOD
A File Implementation
for
FLEX9 Operating Systems

By: Joseph D. Condon 8101 Alpine Drive Des Moines, Iowa 50322 515 278-4581

I. ISAM OVERVIEW

ISAM is a memory resident handler that provides the user with indexed sequential file accessing methods. ISAM will allow multiple files to be open at the same time. The maximum number of open files is limited only by the amount of memory dedicated to the handler at installation time. ISAM allows all record sizes in the range of 1 to 32767 characters to be used, providing sufficient memory has been al- located to the handler. The user need not be concerned with record sizes, blocking, and wasted sector bytes when choosing a files record size. ISAM records can and will extend across disk sector boundaries, therefore all record sizes will be efficiently implemented by ISAM. The maximum file size supported by ISAM is also dependant upon the amount of memory dedicated to the handler at installation time. A typical ISAM handler installation allowing two files to be open at the same time, each with a record size of up to 100 characters, and each with a maximum file size of 1000 records, will require less than 8000 bytes of user memory.

ISAM file are unique in the sense that they always appear to be in a sorted ascending order dependant upon each records content. Re- cords may be added to or deleted from the file in any order. The ISAM handler will always maintain the file in correct ascending order. The user may access the ISAM file in a sequential manner either forwards or backwards. The user may also specify the starting record for the sequential accesses to begin. The starting record may be specified as being the first or last record in the ISAM file. The starting record may also be specified by the use of a starting record value. That is the starting record will be the last record in the file whose content is less than the specified starting record value.

ISAM will provide the programer with a very powerfull tool which can be used to implement sophisticated application programs and pack- ages. The ISAM handler is written in assembler language to ensure optimum speed, effifiency, and memory utilization. Being able to specify operating parameters at installation time allows the user to tailor ISAM to a specific systems requirements without having to waste user memory. Although ISAM was written with speed and efficiency in mind, its actual implementation will be dependant upon the systems disk storage capabilities. ISAM will perform much quicker on a system equipted with hard disk drives as opposed to one with only mini disk drives, and even quicker on a system with RAM disk emulation.

II. IMPLEMENTATION

All communications with ISAM is accomplished through the use of a communication block. The format of the communication block is ident- icle to the format of a "TSC XBASIC" string matrix. This allows easy use of ISAM with basic programs and also provides an efficient means of communication with other programing languages. The location of the communication block must be stored in memory location MEMEND-4 prior to calling ISAM. Upon completion of the command, ISAM will return a 16 bit unsigned numeric error code to the caller by storing the value at location MEMEND-4. The format of the communication block as well as each elements description follows.

ISAM COMMUNICATION BLOCK FORMAT

Memory location			
	x + 0	! X ! X ! X ! X !	element 0 location
	x + 2	! Y ! Y ! Y ! Y !	element 0 length
	x + 4	! X ! X ! X ! X !	element 1 location
	x + 6	! Y ! Y ! Y ! Y !	element 1 length
	x + 8	! X ! X ! X ! X !	element 2 location
	x + 10	! Y ! Y ! Y ! Y !	element 2 length
	x + 12	! X ! X ! X ! X !	element 3 location
	x + 14	! Y ! Y ! Y ! Y !	element 3 length
	x + 16	! X ! X ! X ! X !	element 4 location
	x + 18	! Y ! Y ! Y ! Y !	element 4 length
	x + 20	! X ! X ! X ! X !	element 5 location
	x + 22	! Y ! Y ! Y ! Y !	element 5 length

ELEMENT DESCRIPTIONS

Element 0 : FILE NUMBER

This element must have a length of two bytes and be in the form of a 16 bit unsigned numeric value.

ELEMENT 1 : ISAM COMMAND

This element must have a length equal to or greater than two bytes. The first two bytes of this element are used to define the ISAM command.

ELEMENT 2 : FILE SPECIFICATION 2

This element must be greater than 0 and less than 256 bytes in length. The element is used to contain a standard "FLEX" file specification.

ELEMENT 3 : RECORD SIZE

This element must have a length of two bytes and be in the form of a 16 bit unsigned numeric value.

ELEMENT 4 : MAXIMUM FILE SIZE

This element must have a length of two bytes and be in the form of a 16 bit unsigned numeric value.

ELEMENT 5 : RECORD BUFFER

This element may be of any length and is used to pass record information between the calling program and the ISAM handler.

The record buffer area (element 5), is the only element area that the ISAM handler will ever modify. ISAM will not change the elements location or size but will, when a read command is requested, move the requested records contents into the record buffer area. If the record buffer area is not the same length as the files record size, ISAM will either truncate or null fill the record contents to fit the buffer area. The user should always ensure that the length of the record buf- fer element is equal to or greater than the files record size.

III. ISAM COMMANDS

ISAM commands are performed by first loading the communication block elements with the appropiate values, then storing the address of the communication block at memory location MEMEND-4. The ISAM handler is then called as a subroutine via the "JSR" or "BSR" instruction. Upon completion of the command, ISAM will always store an error code at memory location MEMEND-4, and then return to the calling program via the "RTS" instruction. All of the above steps may be performed with one BASIC statement using the "USR" function. The value returned to BASIC after execution of the "USR" function will be the ISAM error code. All error codes except for 0 indicate an error was encountered by ISAM during the execution of the last command. The following is a list of all ISAM commands along with a detailed description of their individual functions.

OPEN : The open command will attempt to open the file number indicated in the communication block using the file specifications supplied by the caller. If the file specs do not include a drive number, the systems working drive default will be used. If the specs do not contain a file extension, the ISAM handler will use the extension default of "ISA".

CLOSE : The close command will close the file number indicated. If the file has been updated, the close routine will rewrite the files header information clearing the corrupt flag and updating the files actual file size field, it will also rewrite the files index pointers. If the file has not been updated, the file will simply be closed.

CREATE : The create command is used to build an empty ISAM file. The file number used with this command must not be currently open. The file specifications will be handled the same as in the OPEN command. The caller must also specify the record size and maximum file size of the ISAM file to be created. Upon completion of this command, the new file will be closed.

FIRST : The current record number associated with the file number will be set to point at the start of the file.

LAST : The current record number associated with the file number will be set to point at the end of the file.

START : The current record number associated with the file number will be set to point at the last record in the file whose content is less than the starting record value specified in the record buffer area of the communication block.

NEXT : The current record number associated with the file number will be incremented by one. That record will then be read and returned to the caller in the communication block record buffer area. If the size of the record is not the same as the size of the record buffer area, the record will be truncated or null filled accordingly.

CURRENT : The current record number associated with the file number will be read and returned to the caller in the record buffer area of the communication block. If the size of the record is not the same as the size of the record buffer area, the record will be truncated or null filled accordingly.

PREVIOUS : The current record number associated with the file number will be decremented by one. That record will then be read and returned to the caller in the communication block record buffer area. If the size of the record is not the same as the size of the record buffer area, the record will be truncated or null filled accordingly.

ADD : The record contained in the record buffer area of the communication block will be added to the file number specified at the proper location according to the records contents. If this is the first update of the file since open time, the files header will be rewritten indicating that the file is corrupt. If the size of the record is not the same as the size of the record buffer area, the record will be truncated or null filled accordingly.

DELETE : The current record number associated with the file number will be deleted from the file. If this is the first update of the file since open time, the files header will be rewritten indicating that the file is corrupt.

REORGANIZE : This command can only be executed on a file number that is currently open. This command is used to rebuild the files index pointers and recalculate the files actual size. After completion of this command, the file will remain open and be accessable but the reorganized values will not be written to the disk file untill a close command is executed by the user. This command may be used to recover a corrupt file, that is, a file that had been updated and not properly closed afterwards. The REORGANIZE command can be very time consuming and the user may wish to consider going to backup rather than reorganizing the corrupted file.

INITIALIZE : This command initializes the file number specified in the communication block by attempting to close the file specified in its internal file control block. The open flag associated with the file number is then marked as being closed. This command may be required to regain syncronization with the FLEX file management system after encountering a disk error. This command should be used with caution since it may cause an open file associated with the file number to be closed with its corruption flag set, requiring the file to be reorganized the next time it is opened.

IV. INSTALLING ISAM

Before using the ISAM handler, it must first be installed in memory. The syntax of the install command is "ISAM X,Y,Z". The X value epecifies the maximum number of ISAM files to be opened at any one time. The Y value indicates the maximum record size of any ISAM file that is to be opened or created. The Z value is the maximum file size of any ISAM file that will be opened or created. All these parameters must be greater than zero and contain a positive value. ISAM uses the parameters to determine the amount of memory that will be required by the ISAM handler. ISAM then relocates itself at the top of user memory and adjusts the MEMEND location to point to the memory location just below ISAM. During installation time, ISAM will use approximately 4000 bytes of memory beginning at address $0000. Any program currently at this location will be overwritten. The following is a list of possible installation messages generated by ISAM. All but the first message indicates that ISAM has not been installed in memory due to parameter errors or an insufficient amount of user memory.

"ISAM INSTALLED AT LOCATION <XXXX>"

"INVALID MAXIMUM FILES"
"INVALID MAXIMUM RECORD SIZE"
"INVALID MAXIMUM FILE SIZE"

"INSUFFICIENT AVAILABLE MEMORY"

ISAM memory requirements will vary greatly depending upon the installation parameters used. An approximation of required memory may be calculated using the following formula.

MEMORY REQUIRED = 1+X*(331+2*(Z))+2*Y

where I = ISAM handler size (approx 3000).
X = Maximum number of files.
Y = Maximum record size.
Z = Maximum file size.

Example: "ISAM 2,100,1000"

MEMORY REQUIRED =
3000+2*(331+2*(1000))+2*100
= 3000+2*(331+2000)+200
= 3000+2*2331+200
= 3000+4662+200
= 7862

TOTAL USER MEMORY REQUIRED = 7862 BYTES

APPENDIX A. ISAM ERROR CODES

The following is a list of possible ISAM error codes which may be returned to the calling program at memory location MEMEND-4. The format of these codes are 16 bit unsigned numeric values.

ERROR CODE 0 : COMMAND SUCCESSFUL
: This error code indicates that there were no errors detected by ISAM.

ERROR CODES 1-99 : FLEX ERROR
: These error codes may be interpreted the same as all standard "FLEX" error codes.

ERROR CODE 100 : INVALID FILE NUMBER
: This error code indicates that the calling program specified an invalid file number.

ERROR CODE 105 : INVALID COMMAND
: This error code is returned to the caller when an invalid command was attempted.

ERROR CODE 110 : INVALID FILE SPECIFICATION
: This error code indicates that an invalid file specification was passed to ISAM in the communication block.

ERROR CODE 115 : INVALID VERSION NUMBER
: This error code will be returned if you attempt to open a file that does not contain the correct ISAM version number.

ERROR CODE 120 : INVALID RECORD SIZE
: This error indicates that an invalid record size was encountered by ISAM during an "OPEN", "REORGINIZE" or "CREATE" command. Record sizes must be positive and greater than zero.

ERROR CODE 125 : RECORD SIZE TO LARGE
: This error will be returned if an attempt is made to open or create an ISAM file whose record size is larger than that specified at the time ISAM was installed in memory.

ERROR CODE 130 : INVALID FILE SIZE
: This error indicates that an invalid file size was encountered by ISAM during an "OPEN", "REORGINIZE" or "CREATE" command. File sizes must be positive and greater than zero.

ERROR CODE 135 : FILE SIZE TO LARGE
: This error will be returned if an attempt is made to open or create an ISAM file whose file size is larger than that specified at the time ISAM was installed in memory.

ERROR CODE 140 : ACTUAL FILE SIZE > MAXIMUM
: This error indicates that an open command was attempted on a file whose header indicated that the files actual length was greater than the files maximum length. This error should never occur.

ERROR CODE 145 : MAXIMUM SECTORS EXCEEDED
: The maximum size of a FLEX random access file is 65,535 sectors. If an attempt is made to create an ISAM file whose combination of record size and file size would create a random file greater than 65,535 sectors, this error code will be returned to the caller.

ERROR CODE 150 : CORRUPT FILE
: Each ISAM file header area contains a corrupt flag which indicates the accuracy of the files contents. If this flag is set, all operations on the file will return this error, except for the "REORGANIZE" and "CLOSE" commands.

ERROR CODE 155 : FILE NOT OPEN
: This error will be returned for all commands attempted on a file number which has not been previously opened.

ERROR CODE 160 : FILE IS OPEN
: This error indicates that a caller attempted to open a file number that was already open.

ERROR CODE 165 : START OF FILE
An attempt was made to read a record prior to the first record of a file.

ERROR CODE 170 : END OF FILE
An attempt was made to read a record beyond the last record of a file.

ERROR CODE 175 : NULL RECORD
: The caller attempted to add a record that contained all null characters to a ISAM file.

ERROR CODE 180 : FILE FULL
: An attempt was made to add more records to a ISAM file that has reached its maximum file size declared at creation time.

APPENDIX B. BASIC PROGRAM EXAMPLE

The following program is a basic example of how the ISAM handler is used. Although the program is very functional, it is not intended to be used for production type work. The program allows the use of all ISAM commands at any time which may accidently cause file corruption if not used correctly. This program is intended to be used as a pro- graming example and nothing more. In order for the program to function properly, ISAM must have been previously installed at memory location MEMEND+1, with a minimum record size of 100. The program will allow the user to create and maintain a file consisting of address labels in ascending alphabetical order according to the name field.

```
10 REM ************************************************
20 REM *****                                      *****
30 REM *****        BASIC LABEL PROGRAM           *****
40 REM *****                                      *****
50 REM ************************************************
60 REM
70 DPOKE DPEEK(HEX("CC2B"))-2,DPEEK(HEX("CC2B"))+1
80 DIM A0$(5)
90 A0$(0)=CVT2$(0)
100 A2$="                              "
110 A3$="                              "
120 A4$="                              "
130 A5$="          "
140 A0$(5)=A2$+A3$+A4$+A5$
150 PRINT
160 PRINT "  LABEL PROGRAM COMMANDS"
170 PRINT "--------------------------"
180 PRINT "OPEN    CLOSE    CREATE"
190 PRINT "FIRST   LAST     START"
200 PRINT "NEXT    CURRENT  PREVIOUS"
210 PRINT "ADD     DELETE   REORGANIZE"
220 PRINT "END     LIST     INITIALIZE"
230 PRINT "--------------------------"
240 PRINT
250 INPUT "ENTER COMMAND",A0$(1)
260 GOSUB 320
270 IF A03=0 GOTO 150
280 PRINT
290 PRINT "ERROR CODE =",A03
300 PRINT
310 GOTO 150
320 IF LEFT$(A0$(1),2)="OP" GOTO 530
330 IF LEFT$(A0$(1),2)="CR" GOTO 500
340 IF LEFT$(A0$(1),2)="LI" GOTO 700
350 IF LEFT$(A0$(1),2)="ST" GOTO 550
```

```
360 IF LEFT$(A0$(1),2)="AD" GOTO 550
370 IF LEFT$(A0$(1),2)="NE" GOTO 790
380 IF LEFT$(A0$(1),2)="CU" GOTO 790
390 IF LEFT$(A0$(1),2)="PR" GOTO 790
400 IF LEFT$(A0$(1),2)="IN" GOTO 870
410 IF LEFT$(A0$(1),2)="CL" GOTO 870
420 IF LEFT$(A0$(1),2)="FI" GOTO 870
430 IF LEFT$(A0$(1),2)="LA" GOTO 870
440 IF LEFT$(A0$(1),2)="DE" GOTO 870
450 IF LEFT$(A0$(1),2)="RE" GOTO 870
460 IF LEFT$(A0$(1),2)="EN" THEN END
470 PRINT "INVALID SELECTION"
480 A03=0
490 RETURN
500 A0$(3)=CVT3$(100)
510 INPUT "ENTER FILE SIZE",A13
520 A0$(4)=CVT3$(A13)
530 INPUT "ENTER FILE SPECS",A0$(2)
540 GOTO 870
550 PRINT
560 PRINT "NAME        ";
570 INPUT LINE A1$
580 LSET A2$=A1$
590 PRINT "STREET      ";
600 INPUT LINE A1$
610 LSET A3$=A1$
620 PRINT "CITY/STATE ";
630 INPUT LINE A1$
640 LSET A4$=A1$
650 PRINT "ZIP CODE    ";
660 INPUT LINE A1$
670 LSET A5$=A1$
680 A0$(5)=A2$+A3$+A4$+A5$
690 GOTO 870
700 A0$(1)="FIRST"
710 GOSUB 870
720 IF A03<>0 THEN RETURN
730 A0$(1)="NEXT"
740 GOSUB 790
750 IF A03=0 GOTO 740
760 IF A03<>170 THEN RETURN
770 A03=0
780 RETURN
790 GOSUB 870
800 IF A04<>0 THEN RETURN
810 PRINT
820 PRINT MID$(A0$(5),1,30)
830 PRINT MID$(A0$(5),31,30)
840 PRINT MID$(A0$(5),61,30)
850 PRINT MID$(A0$(5),91,10)
860 RETURN
870 A04=USR(PTR(A0$(0)))
880 RETURN
```

Editor's Note: Due to the large size of the actual source code for the above article, it will have to be published as a series.

However, it may take some 3 or 4 issues to get it all in. It consist of some 190 plus sectors of code. For those needed or desiring the complete source code NOW, I suggest you order it on the 68 Micro Journal Reader Service Disk no. 23. As it will be available immediately.

We also wish to thank Joeseph Condon for his generous offer to make this material available to our readers. It is indeed a work of much labor and exhibits the depths of excellence available to all of us by those willing to share. So Joe, I and thousands of readers thank you for your kindness!

DMW

P.S. Please note that this is copyrighted material. Should you desire to do something commercial or involving money with it, please contact the author for arrangements.

ISAM Source Code Available on 68' Micro Reader Disk. See Ad on page 62 Disk #23.

```
ISAM

          OPT       PAG
          TTL       ISAM
          STTL      INDEXED SEQUENTIAL ACCESS METHOD


*****************************************************
*                                                   *
*                                                   *
*                    "ISAM"                         *
*                                                   *
*                                                   *
*          INDEXED SEQUENTIAL ACCESS METHOD         *
*                                                   *
*                                                   *
*                    JOE CONDON                     *
*                    02/12/84                       *
*                                                   *
*                                                   *
*****************************************************
*                                                   *
*                                                   *
*                    DISCLAIMER                     *
*                    ----------                     *
*          JOE CONDON WILL NOT ASSUME ANY           *
*          RESPONSIBILITY FOR DAMAGES IN-           *
*          CURRED OR GENERATED BY THE USE           *
*                OF THIS MATERIAL.                  *
*                                                   *
*                                                   *
*****************************************************


          PAG
*****************************************************
*                                                   *
*                                                   *
*              ISAM COMMAND FORMAT                  *
*              -------------------                  *
*                                                   *
*      <IN>ITIALIZE  = INITIALIZE ISAM HANDLER      *
*                                                   *
*      <OP>EN        = OPEN ISAM FILE               *
*      <CL>OSE       = CLOSE ISAM FILE              *
*      <CR>EATE      = CREATE ISAM FILE             *
*                                                   *
*      <FI>RST       = POINT TO FIRST RECORD        *
*      <LA>ST        = POINT TO LAST RECORD         *
*      <ST>ART       = POINT TO SPECIFIED RECORD    *
*                                                   *
*      <NE>XT        = READ NEXT RECORD             *
*      <CU>RRENT     = READ CURRENT RECORD          *
*      <PR>EVIOUS    = READ PREVIOUS RECORD         *
*                                                   *
*      <AD>D         = ADD RECORD                   *
*                                                   *
*      <DE>LETE      = DELETE RECORD                *
*      <RE>ORGANIZE  = REORGANIZE ISAM FILE         *
*                                                   *
*                                                   *
*****************************************************


          PAG
*****************************************************
*                                                   *
*                  FLEX EQUATES                     *
*                                                   *
*****************************************************

VERNO     EQU       1           VERSION NUMBER

EOT       EQU       $04         ASCII EOT CHARACTER
SPACE     EQU       $20         ASCII SPACE CHARACTER

ORIGIN    EQU       $0000       PROGRAM MEMORY LOCATION
WDRVNO    EQU       $CC0C       WORKING DRIVE NUMBER
MEMEND    EQU       $CC28       MEMORY END
WARMS     EQU       $CD03       FLEX WARM ENTRY
PSTRNG    EQU       $CD1E       PRINT STRING ROUTINE
PCRLF     EQU       $CD24       PRINT CR & LF
OUTADR    EQU       $CD45       PRINT HEX ADDRESS
INDEC     EQU       $CD48       INPUT DECIMAL NUMBER
FMS       EQU       $D406       FILE MANAGEMENT SYSTEM
```

```
*****************************************************
*                                                   *
*              ISAM FCB STORAGE MAP                 *
*                                                   *
*****************************************************

          ORG     0

OFLAG     RMB     1         OPEN FLAG
CFLAG     RMB     1         CORRUPT FLAG
UFLAG     RMB     1         UPDATE FLAG
RECSIZ    RMB     2         RECORD SIZE VARIABLE
MFSIZE    RMB     2         MAX FILE SIZE VARIABLE
AFSIZE    RMB     2         ACTUAL FILE SIZE VARIABLE
CURREC    RMB     2         CURRENT RECORD NUMBER
IFCB      RMB     320       ISAM FILE CONTROL BLOCK
KEYTAB

          PAG
*****************************************************
*                                                   *
*              INSTALL ISAM ROUTINE                 *
*                                                   *
*****************************************************

          ORG     ORIGIN    PROGRAM ORIGIN

INSTAL    BRA     INS1      BRANCH AROUND VERSION NO
          FCB     VERNO     VERSION NUMBER

INS1      JSR     PCRLF     PRINT CR & LF
          BSR     GETPAR    GET INSTALL PARAMETERS
          BEQ     INS2      PARAMETERS OK
          JSR     PSTRNG    PRINT STRING
          JSR     PCRLF     PRINT CR & LF
          JMP     WARMS     RETURN TO FLEX

INS2      LEAX    INMSG0,PCR POINT TO MESSAGE
          JSR     PSTRNG    PRINT STRING
          LDD     MEMEND    GET MEMORY END VALUE
          ADDD    #1        INCREMENT VALUE
          PSH     D         SAVE D REG ON STACK
          LEAX    0,S       POINT TO VALUE ON STACK
          JSR     OUTADR    OUTPUT ADDRESS
          PUL     D         RESTORE STACK
          JSR     PCRLF     PRINT CR & LF
          JMP     WARMS     RETURN TO FLEX

*         GET INSTALL PARAMETERS

GETPAR    JSR     INDEC     INPUT DECIMAL NUMBER
          LBCS    INERR1    INVALID FILE NUMBER
          TFR     X,D       MOVE X REG TO D REG
          LBEQ    INERR1    INVALID FILE NUMBER
          LBMI    INERR1    INVALID FILE NUMBER
          STD     INPAR1,PCR STORE INPUT PARAM 1

          JSR     INDEC     INPUT DECIMAL NUMBER
          LBCS    INERR2    INVALID RECORD SIZE
          TFR     X,D       MOVE X REG TO D REG
          LBEQ    INERR2    INVALID RECORD SIZE
          LBMI    INERR2    INVALID RECORD SIZE
          STD     INPAR2,PCR STORE INPUT PARAM 2

          JSR     INDEC     INPUT DECIMAL NUMBER
          LBCS    INERR3    INVALID FILE SIZE
          TFR     X,D       MOVE X REG TO D REG
          LBEQ    INERR3    INVALID FILE SIZE
          BMI     INERR3    INVALID FILE SIZE
          STD     INPAR3,PCR STORE INPUT PARAM 3

          LSLB              MULTIPLY D REG TIMES 2

          ROLA
          BCS     INERR4    INSUFFICIENT MEMORY
          ADDD    #KEYTAB   ADD ISAM HEADER LENGTH
          BCS     INERR4    INSUFFICIENT MEMORY
          STD     -2,S      STORE TEMPORARY
          LDX     INPAR1,PCR GET INPUT PARAM 1
          LDD     #0        CLEAR D REG

GP1       ADDD    -2,S      ADD ISAM FILE AREA
          BCS     INERR4    INSUFFICIENT MEMORY
          LEAX    -1,X      DECREMENT FILE NUMBER
          BNE     GP1       ADD NEXT FILE AREA
          ADDD    INPAR2,PCR ADD RECORD BUFFER
          BCS     INERR4    INSUFFICIENT MEMORY
          ADDD    INPAR2,PCR ADD RECORD BUFFER
          BCS     INERR4    INSUFFICIENT MEMORY
          STD     -2,S      STORE TEMPORARY
          LDD     #BUFFER   POINT TO END OF HANDLER
          SUBD    #ISAM     CALCULATE HANDLER LENGTH
          ADDD    -2,S      ADD BUFFER SIZE
          BCS     INERR4    INSUFFICIENT MEMORY
          CMPD    MEMEND    COMPARE TO AVAILABLE MEMORY
          BHI     INERR4    INSUFFICIENT MEMORY
          STD     -2,S      STORE TEMPORARY
          LDD     MEMEND    GET MEMEND VALUE
          SUBD    -2,S      SUBTRACT ISAM SIZE
          ADDD    #1        ADJUST LOCATION
          STD     -2,S      STORE TEMPORARY
          LEAX    COUNT,PCR POINT TO END OF HANDLER
          CMPX    -2,S      TEST FOR OVERLAP
          BHS     INERR4    INSUFFICIENT MEMORY
          STX     -4,S      SAVE ISAM END
          LEAX    ISAM,PCR  POINT TO START OF HANDLER
          LDY     -2,S      POINT TO NEW LOCATION

OP2       LDA     0,X+      GET ISAM BYTE
          STA     0,Y+      STORE ISAM BYTE
          CMPX    -4,S      TEST FOR HANDLER END
          BLO     OP2       MOVE NEXT BYTE

GP3       CLR     0,Y+      CLEAR BUFFER BYTE
          CMPY    MEMEND    TEST FOR BUFFER END
          BLE     GP3       CLEAR NEXT BYTE
          LDD     -2,S      GET ISAM LOCATION
          SUBD    #1        GET NEW MEMEND
          STD     MEMEND    SET NEW MEMEND
          LDX     #0        GET STATUS CODE
          RTS               RETURN

*         INSTALL ERROR ROUTINE

INERR1    LEAX    INMSG1,PCR POINT TO MESSAGE
          RTS               RETURN

INERR2    LEAX    INMSG2,PCR POINT TO MESSAGE
          RTS               RETURN

INERR3    LEAX    INMSG3,PCR POINT TO MESSAGE
          RTS               RETURN

INERR4    LEAX    INMSG4,PCR POINT TO MESSAGE
          RTS               RETURN

*         INSTALL MESSAGES

INMSG0    FCC     'ISAM INSTALLED AT LOCATION  ',EOT
INMSG1    FCC     'INVALID MAXIMUM FILES',EOT
INMSG2    FCC     'INVALID MAXIMUM RECORD SIZE',EOT
INMSG3    FCC     'INVALID MAXIMUM FILE SIZE',EOT
INMSG4    FCC     'INSUFFICIENT AVAILABLE MEMORY',EOT

          PAG
*****************************************************
*                                                   *
*              ISAM MAIN ENTRY POINT                *
*                                                   *
*****************************************************

ISAM      LDY     MEMEND    GET FLEX MEMEND VALUE
          LDY     -4,Y      POINT TO CALLER IFCB
          STY     CIFPNT,PCR STORE CALLER IFCB POINTER
          BSR     DECODE    DECODE AND EXECUTE COMMAND
          LDY     MEMEND    GET FLEX MEMEND VALUE
          STD     -4,Y      STORE STATUS CODE
          RTS               RETURN TO CALLER

*****************************************************
*                                                   *
*     DECODE AND EXECUTE COMMAND SUBROUTINE         *
*                                                   *
*****************************************************

DECODE    LEAY    0,Y       GET CALLER FILE NUMBER
          LDD     2,Y       GET FILE NUMBER LENGTH
          CMPD    #2        TEST FILE NUMBER LENGTH
          LBNE    ERR100    INVALID FILE NUMBER
          LDD     [0,Y]     GET FILE NUMBER
          ADDD    #1        ADJUST TO ONE SIGNIFICANT
          CMPD    INPAR1,PCR COMPARE TO MAX FILES PARAM
          LBHI    ERR100    INVALID FILE NUMBER
```

```
        SUBD    #1          ADJUST FOR ZERO SIGNIFICANT
        STD     ACCUM1,PCR STORE IN ACCUM 1
        LDD     INPAR3,PCR GET MAX FILE SIZE PARAM
        LSLB               MULTIPLY BY TWO
        ROLA
        ADDD    #KEYTAB    ADD FILE HEADER LENGTH
        STD     ACCUM2,PCR STORE IN ACCUM 2
        LBSR    MUL32      MULTIPLY ACCUM 1 BY ACCUM 2

        LDD     ACCUM3+2,PCR GET PRODUCT
        ADDD    INPAR2,PCR ADD MAX RECORD SIZE PARAM
        ADDD    INPAR2,PCR ADD MAX RECORD SIZE PARAM
        LEAU    BUFFER,PCR GET BUFFER LOCATION
        STU     -2,S       STORE TEMPORY
        ADDD    -2,S       ADD TEMP TO D REG
        TFR     D,U        TRANSFER D REG TO U REG

        LDY     CIFPNT,PCR GET CALLER ISAM FCB PNTR
        LEAY    4,Y        POINT TO COMMAND DESC
        LDD     2,Y        GET ISAM COMMAND LENGTH
        CMPD    #2         TEST COMMAND STRING LENGTH
        LBLO    ERR105     INVALID COMMAND
        LDD     [0,Y]      GET ISAM COMMAND

        CMPD    #$494E     INITIALIZE ISAM HANDLER
        BEQ     INIT       EXECUTE SUBROUTINE
        CMPD    #$4F50     OPEN ISAM FILE
        BEQ     OPEN       EXECUTE SUBROUTINE
        CMPD    #$434C     CLOSE ISAM FILE
        LBEQ    CLOSE      EXECUTE SUBROUTINE
        CMPD    #$4352     CREATE ISAM FILE
        LBEQ    CREATE     EXECUTE SUBROUTINE
        CMPD    #$4649     POINT TO FIRST RECORD
        LBEQ    FIRST      EXECUTE SUBROUTINE
        CMPD    #$4C41     POINT TO LAST RECORD
        LBEQ    LAST       EXECUTE SUBROUTINE
        CMPD    #$5354     POINT TO SPECIFIED RECORD
        LBEQ    START      EXECUTE SUBROUTINE
        CMPD    #$4E45     READ NEXT RECORD
        LBEQ    NEXT       EXECUTE SUBROUTINE
        CMPD    #$4355     READ CURRENT RECORD
        LBEQ    CURRNT     EXECUTE SUBROUTINE
        CMPD    #$5052     READ PREVIOUS RECORD
        LBEQ    PREV       EXECUTE SUBROUTINE
        CMPD    #$4144     ADD RECORD
        LBEQ    ADD        EXECUTE SUBROUTINE
        CMPD    #$4445     DELETE RECORD
        LBEQ    DELETE     EXECUTE SUBROUTINE
        CMPD    #$5245     REORGANIZE ISAM FILE
        LBEQ    REORG      EXECUTE SUBROUTINE
        LBRA    ERR105     INVALID COMMAND
```

```
*****************************************************
*                                                   *
*            ISAM INITIALIZE ROUTINE                *
*                                                   *
*****************************************************

INIT    LEAX    IFCB,U     POINT TO ISAM FCB
        LDA     #$04       GET FILE CLOSE CODE
        STA     0,X        SET FCB FUNCTION CODE
        JSR     FMS        CALL FMS
        CLR     OFLAG,U    CLEAR OPEN FLAG
        LDD     #0         CLEAR STATUS CODE
        RTS                RETURN
```

```
*****************************************************
*                                                   *
*              ISAM OPEN ROUTINE                    *
*                                                   *
*****************************************************

OPEN    TST     OFLAG,U    TEST OPEN FLAG
        LBNE    ERR160     FILE IS OPEN
        LBSR    GETFID     GET FILE SPECS
        LBNE    EXIT       EXIT WITH ERROR

        LEAX    IFCB,U     POINT TO FCB
        LDA     #$03       GET OPEN FOR WRITE CODE
        STA     0,X        SET FCB FUNCTION
        JSR     FMS        CALL FMS
        LBNE    FMSERR     GOTO FMS ERROR

        LDA     #$02       RANDOM FILE SECTOR MAP CODE
        STA     23,X       SET FILE SECTOR MAP IND
        LDA     #$FF       NO SPACE COMPRESSION CODE
        STA     59,X       SET SPACE COMPRESSION FLAG

        LDA     #8         ISAM FILE VERSION LENGTH
        STA     COUNT,PCR SET LENGTH COUNTER
        LEAY    IFVER,PCR POINT TO ISAM FILE VERSION

OPEN1   JSR     FMS        READ NEXT CHAR
        LBNE    FMSERR     GOTO FMS ERROR
        CMPA    0,Y+       COMPARE VERSION CHAR
        LBNE    ERR115     GOTO VERSION
        DEC     COUNT,PCR DECREMENT LENGTH COUNTER
        BNE     OPEN1      READ NEXT CHAR

        JSR     FMS        READ NEXT BYTE
        LBNE    FMSERR     GOTO FMS ERROR
        STA     CFLAG,U    STORE CORRUPT FLAG
        JSR     FMS        READ NEXT BYTE

        LBNE    FMSERR     GOTO FMS ERROR
        ORA     CFLAG,U    OR WITH CORRUPT FLAG
        STA     CFLAG,U    STORE CORRUPT FLAG

        JSR     FMS        READ NEXT BYTE
        LBNE    FMSERR     GOTO FMS ERROR
        STA     RECSIZ,U   STORE RECORD SIZE UP
        JSR     FMS        READ NEXT BYTE
        LBNE    FMSERR     GOTO FMS ERROR
        STA     RECSIZ+1,U STORE RECORD SIZE LOW
        LDD     RECSIZ,U   GET RECORD SIZE
        LBEQ    ERR120     INVALID RECORD SIZE
        CMPD    INPAR2,PCR COMPARE TO REC SIZE PARAM
        LBHI    ERR125     RECORD SIZE TO LARGE

        JSR     FMS        READ NEXT BYTE
        LBNE    FMSERR     GOTO FMS ERROR
        STA     MFSIZE,U   STORE MAX FILE SIZE UP
        JSR     FMS        READ NEXT BYTE
        LBNE    FMSERR     GOTO FMS ERROR
        STA     MFSIZE+1,U STORE MAX FILE SIZE LOW
        LDD     MFSIZE,U   GET MAX FILE SIZE
        LBEQ    ERR130     INVALID MAX FILE SIZE
        CMPD    INPAR3,PCR COMPARE TO FILE SIZE PARAM
        LBHI    ERR135     FILE SIZE TO LARGE

        LBSR    CSCOFF     CALCULATE FCB OFFSETS
        LBNE    EXIT       EXIT WITH ERROR

        JSR     FMS        READ NEXT BYTE
        LBNE    FMSERR     GOTO FMS ERROR
        STA     AFSIZE,U   STORE ACTUAL FILE SIZE UP
        JSR     FMS        READ NEXT BYTE
        LBNE    FMSERR     GOTO FMS ERROR
        STA     AFSIZE+1,U STORE ACTUAL FILE SIZE LOW
        LDD     AFSIZE,U   GET ACTUAL FILE SIZE
        CMPD    MFSIZE,U   COMPARE TO MAX FILE SIZE
        LBHI    ERR140     ACTUAL FILE SIZE > MAX

        LEAY    KEYTAB,U   GET KEY TABLE LOCATION
        LDD     MFSIZE,U   GET MAX FILE SIZE
        STD     COUNT,PCR STORE RECORD COUNT

OPEN2   JSR     FMS        READ NEXT BYTE
        LBNE    FMSERR     GOTO FMS ERROR
        STA     0,Y+       STORE BYTE IN KEYTAB
        JSR     FMS        READ NEXT BYTE
        LBNE    FMSERR     GOTO FMS ERROR
        STA     0,Y+       STORE BYTE IN KEYTAB
        LDD     COUNT,PCR GET RECORD COUNT
        SUBD    #1         DECREMENT RECORD COUNT
        STD     COUNT,PCR STORE RECORD COUNT
        BNE     OPEN2      GET NEXT RECORD POINTER

        LDD     #0         POINT TO START OF FILE
        STD     CURREC,U   SET CURRENT RECORD POINTER
        INC     OFLAG,U    SET OPEN FLAG
        CLR     UFLAG,U    CLEAR UPDATE FLAG
        TST     CFLAG,U    TEST CORRUPT FLAG
        LBNE    ERR150     CORRUPT FILE
        LDD     #0         CLEAR STATUS CODE
        RTS                RETURN
```

```
*****************************************************
*                                                   *
*              ISAM CLOSE ROUTINE                   *
*                                                   *
*****************************************************

CLOSE   TST     OFLAG,U    TEST OPEN FLAG
        LBEQ    ERR155     FILE NOT OPEN
        LEAX    IFCB,U     POINT TO ISAM FCB
        TST     CFLAG,U    TEST CORRUPT FLAG
        LBNE    CLOSE2     CLOSE FILE
        TST     UFLAG,U    TEST UPDATE FLAG
        LBEQ    CLOSE2     CLOSE FILE
```

```
         LDD     #1          GET FIRST RELATIVE RECORD
         STD     32,X        SET FCB REL RECORD
         LDA     #$15        GET FMS POSITION CODE
         STA     0,X         SET FCB FUNCTION CODE
         JSR     FMS         POSITION TO RECORD N
         LBNE    FMSERR      GOTO FMS ERROR
         LDA     #$12        GET FMS PUT RANDOM CODE
         STA     0,X         SET FCB FUNCTION CODE

         LDA     #12         GET FIRST BYTE OFFSET
         STA     35,X        SET FCB BYTE OFFSET
         CLRA                CLEAR A REGISTER
         JSR     FMS         PUT RANDOM BYTE
         LBNE    FMSERR      GOTO FMS ERROR
         INC     35,X        BUMP FCB BYTE OFFSET
         CLRA                CLEAR A REGISTER
         JSR     FMS         PUT RANDOM BYTE
         LBNE    FMSERR      GOTO FMS ERROR

         LDA     #18         GET FIRST BYTE OFFSET
         STA     35,X        SET FCB BYTE OFFSET

         LDA     AFSIZE,U    GET ACTUAL FILE SIZE UP
         JSR     FMS         PUT RANDOM BYTE
         LBNE    FMSERR      GOTO FMS ERROR
         INC     35,X        BUMP FCB BYTE OFFSET
         LDA     AFSIZE+1,U  GET ACTUAL FILE SIZE LOW
         JSR     FMS         PUT RANDOM BYTE
         LBNE    FMSERR      GOTO FMS ERROR
         INC     35,X        BUMP FCB BYTE OFFSET
         LDD     MFSIZE,U    GET MAX FILE SIZE
         STD     COUNT,PCR   SET RECORD COUNT
         LEAY    KEYTAB,U    POINT TO KEY TABLE

CLOSE1   LDA     0,Y+        GET KEY TABLE BYTE
         JSR     FMS         PUT RANDOM BYTE
         LBNE    FMSERR      GOTO FMS ERROR
         INC     35,X        BUMP FCB BYTE OFFSET
         LDA     0,Y+        G T KEY TABLE BYTE
         JSR     FMS         PUT RANDOM BYTE
         LBNE    FMSERR      GOTO FMS ERROR

         LDD     COUNT,PCR   GET RECORD COUNT
         SUBD    #1          DECREMENT RECORD COUNT
         STD     COUNT,PCR   STORE RECORD COUNT
         BEQ     CLOSE2      CLOSE FILE

         INC     35,X        BUMP FCB BYTE OFFSET
         BNE     CLOSE1      WRITE NEXT POINTER

         LDD     32,X        GET REL RECORD NUMBER
         ADDD    #1          BUMP RECORD NUMBER
         STD     32,X        SET REL RECORD NUMBER
         LDA     #$15        GET FMS POSITION CODE
         STA     0,X         SET FCB FUNCTION CODE
         JSR     FMS         POSITION TO RECORD N
         LBNE    FMSERR      GOTO FMS ERROR
         LDA     #$12        GET FMS PUT RANDOM CODE
         STA     0,X         SET FCB FUNCTION CODE
         LDA     #4          GET FIRST BYTE OFFSET
         STA     35,X        SET FCB BYTE OFFSET
         BRA     CLOSE1      WRITE NEXT POINTER

CLOSE2   LDA     #$04        GET CLOSE CODE
         STA     0,X         SET FCB FUNCTION
         JSR     FMS         CLOSE FILE
         LBNE    FMSERR      GOTO FMS ERROR
         CLR     OFLAG,U     CLEAR OPEN FLAG
         LDD     #0          CLEAR STATUS CODE
         RTS                 RETURN

*****************************************************
*                                                   *
*              ISAM CREATE ROUTINE                  *
*                                                   *
*****************************************************

CREATE   TST     OFLAG,U     TEST OPEN FLAG
         LBNE    ERR160      FILE IS OPEN
         LBSR    GETFIO      GET FILE SPECS
         LBNE    EXIT        EXIT WITH ERROR

         LDY     CIFPNT,PCR GET CALLER ISAM FCB PNTR
         LEAY    12,Y        POINT TO RECORD SIZE DESC
         LDD     2,Y         GET RECORD SIZE L NGTH
         CMPD    #2          TEST FOR PROPER FORMAT
         LBNE    ERR120      INVALID RECORD SIZE
         LDD     [0,Y]       GET RECORD SIZE
         LBEQ    ERR120      INVALID RECORD SIZE
         CMPD    INPAR2,PCR COMPARE TO REC SIZE PARAM
         LBHI    ERR125      RECORD SIZE TO LARGE
         STD     RECSIZ,U    SET RECORD SIZE VARIABLE

         LDY     CIFPNT,PCR GET CALLER ISAM FCB PNTR
         LEAY    16,Y        POINT TO MAX FILE SIZE DESC
         LDD     2,Y         GET MAX FILE SIZE LENGTH
         CMPD    #2          TEST FOR PROPER FORMAT
         LBNE    ERR130      INVALID MAX FILE SIZE
         LDD     [0,Y]       GET MAX FILE SIZE
         LBEQ    ERR130      INVALID MAX FILE SIZE
         CMPD    INPAR3,PCR COMPARE TO FILE SIZE PARAM
         LBHI    ERR135      FILE SIZE TO LARGE
         STD     MFSIZE,U    SET MAX FILE SIZE VARIABLE

         LBSR    CSCOFF      CALC SECTOR CHAR OFFSET
         LBNE    EXIT        EXIT WITH ERROR

         LEAX    IFCB,U      POINT TO ISAM FCB
         LDA     #$02        GET OPEN FOR WRITE CODE
         STA     0,X         SET FCB FUNCTION
         JSR     FMS         CALL FMS
         LBNE    FMSERR      GOTO FMS ERROR

         LDA     #$02        RANDOM FILE SECTOR MAP CODE
         STA     23,X        SET FILE SECTOR MAP IND
         LDA     #$FF        NO SPACE COMPRESSION CODE
         STA     59,X        SET SPACE COMPRESSION FLAG

         LDA     #16         GET ISAM FILE HEADER LENGTH
         STA     COUNT,PCR SET LENGTH COUNTER
CR1      CLRA                GET NULL CODE
         JSR     FMS         CALL FMS
         LBNE    FMSERR      GOTO FMS ERROR
         DEC     COUNT,PCR DECREMENT LENGTH COUNTER
         BNE     CR1         GET NEXT CHAR

         LDD     #0          GET RECORD COUNTER
         STD     COUNT,PCR SET RECORD COUNTER
CR2      LDD     COUNT,PCR GET RECORD COUNTER
         JSR     FMS         WRITE RECORD COUNTER UPPER
         LBNE    FMSERR      GOTO FMS ERROR
         LDD     COUNT,PCR GET RECORD COUNTER
         TFR     B,A         GET RECORD COUNTER LOWER

         JSR     FMS         WRITE RECORD COUNTER LOWER
         LBNE    FMSERR      GOTO FMS ERROR
         LDD     COUNT,PCR GET RECORD COUNTER
         ADDD    #1          INCREMENT RECORD COUNTER
         STD     COUNT,PCR STORE RECORD COUNTER
         CMPD    MFSIZE,U    TEST FOR LAST RECORD
         BNE     CR2         WRITE NEXT POINTER
         LDY     MFSIZE,U    GET MAX FILE SIZE VARIABLE
         STY     COUNT,PCR SET COUNTER
CR3      LDY     COUNT,PCR GET FILE SIZE COUNTER
         BEQ     CR5         DATA FILE CREATED
         LEAY    -1,Y        DECREMENT FILE SIZE
         STY     COUNT,PCR STORE FILE SIZE COUNTER
         LDY     RECSIZ,U    GET RECORD SIZE VARIABLE
CR4      CLRA                CLEAR A REGISTER
         JSR     FMS         WRITE NEXT CHARACTER
         LBNE    FMSERR      GOTO FMS ERROR
         LEAY    -1,Y        DECREMENT RECORD SIZE
         BNE     CR4         WRITE NEXT CHARACTER
         BRA     CR3         WRITE NEXT RECORD
CR5      LDA     #$04        GET CLOSE CODE
         STA     0,X         SET FCB FUNCTION
         JSR     FMS         CALL FMS
         LBNE    FMSERR      GOTO FMS ERROR

         LDA     #$03        GET OPEN FOR UPDATE CODE
         STA     0,X         SET FCB FUNCTION
         JSR     FMS         CALL FMS
         LBNE    FMSERR      GOTO FMS ERROR

         LDA     #$12        GET PUT RANDOM BYTE CODE
         STA     0,X         SET FCB FUNCTION
         LDA     #$02        RANDOM FILE SECTOR MAP CODE
         STA     23,X        SET FILE SECTOR MAP IND
         LDA     #$FF        NO SPACE COMPRESSION CODE
         STA     59,X        SET SPACE COMPRESSION FLAG
         LDA     #4          GET FIRST BYTE OFFSET
         STA     35,X        SET FCB RANDOM INDEX
         LDA     #8          ISAM FILE VERSION LENGTH
         STA     COUNT,PCR SET LENGTH COUNTER
         LEAY    IFVER,PCR POINT TO ISAM FILE VERSION
CR6      LDA     0,Y+        GET VERSION CHAR
         JSR     FMS         CALL FMS
         LBNE    FMSERR      GOTO FMS ERROR
         INC     35,X        BUMP FCB RANDOM INDEX
         DEC     COUNT,PCR DECREMENT LENGTH COUNTER
         BNE     CR6         GET NEXT CHAR
```

```
         CLRA                    GET FLAG VALUE UPPER
         JSR       FMS           WRITE FLAG VALUE UPPER

         LBNE      FMSERR        GOTO FMS ERROR
         INC       35,X          BUMP FCB RANDOM INDEX
         CLRA                    GET FLAG VALUE LOWER
         JSR       FMS           WRITE FLAG VALUE LOWER
         LBNE      FMSERR        GOTO FMS ERROR
         INC       35,X          BUMP FCB RANDOM INDEX

         LDD       RECSIZ,U      GET RECORD SIZE VARIABLE
         JSR       FMS           WRITE RECORD SIZE UPPER
         LBNE      FMSERR        GOTO FMS ERROR
         INC       35,X          BUMP FCB RANDOM INDEX
         LDD       RECSIZ,U      GET RECORD SIZE VARIABLE
         TFR       B,A           GET RECORD SIZE LOWER
         JSR       FMS           WRITE RECORD SIZE LOWER
         LBNE      FMSERR        GOTO FMS ERROR
         INC       35,X          BUMP FCB RANDOM INDEX

         LDD       MFSIZE,U      GET MAX FILE SIZE VARIABLE
         JSR       FMS           WRITE FILE SIZE UPPER
         LBNE      FMSERR        GOTO FMS ERROR
         INC       35,X          BUMP FCB RANDOM INDEX
         LDD       MFSIZE,U      GET MAX FILE SIZE VARIABLE
         TFR       B,A           GET FILE SIZE LOWER
         JSR       FMS           WRITE FILE SIZE LOWER
         LBNE      FMSERR        GOTO FMS ERROR
         INC       35,X          BUMP FCB RANDOM INDEX

         CLRA                    GET ACTUAL FILE SIZE UPPER
         JSR       FMS           WRITE ACTUAL FILE SIZE UP
         LBNE      FMSERR        GOTO FMS ERROR
         INC       35,X          BUMP FCB RANDOM INDEX
         CLRA                    GET ACTUAL FILE SIZE LOWER
         JSR       FMS           WRITE ACTUAL FILE SIZE LOW
         LBNE      FMSERR        GOTO FMS ERROR

         LDA       #$04          GET CLOSE CODE
         STA       0,X           SET FCB FUNCTION
         JSR       FMS           CALL FMS
         LBNE      FMSERR        GOTO FMS ERROR
         LDD       #0            CLEAR STATUS CODE
         RTS                     RETURN
```

```
*****************************************************
*                                                   *
*                ISAM FIRST ROUTINE                 *
*                                                   *
*****************************************************

FIRST    TST       OFLAG,U       TEST OPEN FLAG
         LBEQ      ERR155        FILE NOT OPEN
         TST       CFLAG,U       TEST CORRUPT FLAG
         LBNE      ERR150        CORRUPT FILE

         LDD       #0            POINT TO START OF FILE
         STD       CURREC,U      STORE CURRENT RECORD
         LDD       #0            CLEAR STATUS CODE
         RTS                     RETURN
```

```
*****************************************************
*                                                   *
*                ISAM LAST ROUTINE                  *
*                                                   *
*****************************************************

LAST     TST       OFLAG,U       TEST OPEN FLAG
         LBEQ      ERR155        FILE NOT OPEN
         TST       CFLAG,U       TEST CORRUPT FLAG
         LBNE      ERR150        CORRUPT FILE
         LDD       AFSIZE,U      GET LAST RECORD NUMBER
         ADDD      #1            POINT TO END OF FILE
         STD       CURREC,U      STORE CURRENT RECORD
         LDD       #0            CLEAR STATUS CODE
         RTS                     RETURN
```

```
*****************************************************
*                                                   *
*                ISAM START ROUTINE                 *
*                                                   *
*****************************************************

START    TST       OFLAG,U       TEST OPEN FLAG
         LBEQ      ERR155        FILE NOT OPEN
         TST       CFLAG,U       TEST CORRUPT FLAG
         LBNE      ERR150        CORRUPT FILE
         LBSR      MOVVB1        MOVE VARIABLE TO BUFFER 1
         LBSR      FIND          FIND KEY RECORD
         LBNE      EXIT          EXIT WITH ERROR
         LDD       #0            CLEAR STATUS CODE
         RTS                     RETURN
```

```
*****************************************************
*                                                   *
*                ISAM NEXT ROUTINE                  *
*                                                   *
*****************************************************

NEXT     TST       OFLAG,U       TEST OPEN FLAG
         LBEQ      ERR155        FILE NOT OPEN
         TST       CFLAG,U       TEST CORRUPT FLAG
         LBNE      ERR150        CORRUPT FILE
         LDD       CURREC,U      GET CURRENT RECORD NUMBER

         CMPD      AFSIZE,U      TEST FOR LAST RECORD
         LBHI      ERR170        END OF FILE
         ADDD      #1            BUMP CURRENT RECORD NUMBER
         STD       CURREC,U      STORE CURRENT RECORD NUMBER
         CMPD      AFSIZE,U      TEST FOR LAST RECORD
         LBHI      ERR170        END OF FILE
         LBSR      RRREC         READ RELATIVE RECORD
         LBNE      EXIT          EXIT WITH ERROR
         LBSR      MOVB2V        MOVE BUFFER 2 TO RECORD
         LDD       #0            CLEAR STATUS CODE
         RTS                     RETURN
```

```
*****************************************************
*                                                   *
*                ISAM CURRENT ROUTINE               *
*                                                   *
*****************************************************

CURRNT   TST       OFLAG,U       TEST OPEN FLAG
         LBEQ      ERR155        FILE NOT OPEN
         TST       CFLAG,U       TEST CORRUPT FLAG
         LBNE      ERR150        CORRUPT FILE
         LDD       CURREC,U      GET CURRENT RECORD NUMBER
         LBEQ      ERR165        START OF FILE
         CMPD      AFSIZE,U      TEST FOR LAST RECORD
         LBHI      ERR170        END OF FILE
         LBSR      RRREC         READ RELATIVE RECORD
         LBNE      EXIT          EXIT WITH ERROR
         LBSR      MOVB2V        MOVE BUFFER 2 TO RECORD
         LDD       #0            CLEAR STATUS CODE
         RTS                     RETURN
```

```
*****************************************************
*                                                   *
*                ISAM PREVIOUS ROUTINE              *
*                                                   *
*****************************************************

PREV     TST       OFLAG,U       TEST OPEN FLAG
         LBEQ      ERR155        FILE NOT OPEN
         TST       CFLAG,U       TEST CORRUPT FLAG
         LBNE      ERR150        CORRUPT FILE
         LDD       CURREC,U      GET CURRENT RECORD NUMBER
         LBEQ      ERR165        START OF FILE
         SUBD      #1            DECREMENT RECORD NUMBER
         STD       CURREC,U      STORE CURRENT RECORD NUMBER
         LBEQ      ERR165        START OF FILE
         LBSR      RRREC         READ RELATIVE RECORD
         LBNE      EXIT          EXIT WITH ERROR
         LBSR      MOVB2V        MOVE BUFFER 2 TO RECORD

         LDD       #0            CLEAR STATUS CODE
         RTS                     RETURN
```

RAMBLINGS & Such

Well, here I am back again with some fairly good rumors and solid stuff. As well as some more of the rambling kind, that seem to become more solid with age. That is the nice thing about being the chief head knocker, nobody edits your stuff, not with the old red pencil, that is.

Actually we do not have many red pencils here at CPI. I have tried to let it be a more "free" flowing forum. At times that has miffed some and delighted others. However, we have only asked one thing of our authors, please be factual! That they have been pretty successful at. As a result it has made my job more like plain fun, rather than work.

As you might have noticed, our advertising has fallen off somewhat. That was to be expected. Some of our advertisers were under capitalized, but had good products. Others had good products and a potentially good future but either did not support their customers properly or "read" the future market wrong. For them we really feel sad. I did all I could to keep some of them from going belly-up. All except those who did not support or ran off from and ignored their past customers. Their absence was all the better for us remaining. Some got all "hot" for the "grass on the other side of the hill". Most of them have paid dearly for that folly. However, there are successful ventures advertising in the pages of 68 Micro Journal this very month, they survived and grew and/or replaced those who blew it. Which just goes to show that there really was a market here all the time. The secret was in recognizing it. But is is bitter sweet to me. I get no joy from their misfortune. I could think, "well, you got what you deserved, for not listening." But everytime one of them failed, we lost an advertiser, or availability to a good product, and that **HURT** us all!

So, there is really no such thing as being right in this type of situation. It affects us all. However, it does make it a little easier for the remaining ones as there is a larger slice for each. So, for you the reader/user it probably means a somewhat better product and a supplier who now is beginning to realize the benefits of your good-will. As for us it means less advertising revenue. And that means that we have to make it up somewhere else, or we regress, then YOU get **cranked** up. That financial slack has been taken up, in part, by **S.E. MEDIA & DATA-COMP** sales. Everytime you make a purchase from either of our divisions, you help keep us going! And that benefits all of us.

A Sad Word(s) of Caution

Also I need to caution you about purchasing from past issues of 68 Micro Journal (or any other magazine, for that matter). Especially if the seller is not currently advertising. Seems that some of you will send in your money to a company that has even had their telephone disconnected (CAUTION: some still have their old number active, just in case)! We are getting some complaints about a **few** past advertisers who will cash your check (if you are dumb enough to send one blindly) but not deliver. **NOTE: I said a "few". Please, please remember, MOST of our past advertisers ARE honest, it is just a FEW bad ones!** But until I do the full legal thing, I cannot mention their names. But rest assured I will as soon as I get the legal parts straight. We are working with several state and local government agencies where we and the wronged reader have filed complaints. We are furnishing additional data to those agencies to support the complaints and help show the general attitudes of the offending dishonest vendors. Fact is, we just got a conviction last month, based on our related data. It is being appealed, but when it is finished, you will know.

What I want you to know is, it works, and I am keeping my promise to you. One state District Attorney told me on the telephone that we were the ONLY magazine that had cooperated so fully with their investigative proceedings.

***

If you need to know the current status of ANY of our past advertisers - please give me a personal call, at our office. I may be able to assist. and PLEASE remember, just because they are not presently advertising, does not necessarily mean that there is anything wrong! A little caution however, is better that than lost bucks or worse!

***

Tell Me Again!

Granted, we are not the largest or even one of the larger markets, in the micro rat race going on now. But we have managed to survive because we were a group bound together by common interest, and equipment. Also, if I might, I believe 68 Micro Journal has had, a not so small part, in the longevity of our group and market. And that brings me to another point. As times have changed, we have tried to keep pace. So, I need your input as to what type articles you want to see more of in the future. Drop me a line or short note and let me know what **YOU** want. Shall it be more on languages, software, hardware, 68XXX, games, applications or what have you? Also, if we were forced to using a somewhat cheaper grade of paper, etc., but kept the contents the same, would you object? I would like to know. We need to plan ahead! One thing is for sure, **WE WILL NOT FORGET THOSE WHO GOT US HERE! Our 68XX coverage will continue!!!**

New Addition to JUST
(Because so many ask for it
especially you OS-9 users)
A Text Processor for OS-9
and/or FLEX
In 'C' Also

Tom and Chris, over at S.E. MEDIA, wanted me to mention to you that they have a couple of nice updates on the JUST Text Processor, from our Ron Anderson. First, there is now a version in "C", so you OS-9 users can compile and run it under OS-9. We will compile for you if you do not have a C compiler (but you should. it is a fine HLL). at no additional charge. It has been a very popular text processor under FLEX. And has been our best selling stand-alone text processor for those who need or desire a text processor that takes advantage of the advanced features of their printer (especially EPSON). Also there is a version now that understands the TSC Text Processor commands (many) and also the advanced features of **JUST**, so that if you have text in TSC format,

well, with little or no changes it can be edited and proceesed by JUST, it is called "JUSTSC", and will very soon (probably by the time this gets to print) be available from S.E. MEDIA. Also note the "STAR" speciale in their advertising catalog. They really try to give you real savings and quality software. I think they deserve your loyalty, and support.

An S50/68XX Service Department
(Because you asked for it)

Over the past few years, and especially since some prior hardware manufacturers and vendors have dropped out, we never cease to get calls from some of you, wanting to know WHO can repair their ailing S50 Bus something or another. Until now you were forced to depend on the dealer (where did they all go?) or struggle along on your own, if the manufacturer had drifted elsewhere. That has been one thing that has really hurt our marketplace. Good service is one of the most important ingredients of the entire process.

Well, for 8 years our DATA-COMP Division has maintained a fairly extensive service department. For those items we sold only. Over the years we have sold an awful lot of S50 Bus stuff. And we have always been in the position to give IMMEDIATE attention to anything we sold that needed servicing. That is why we sold probably more than anyone else. Now, we have decided to give it a trial and make this same service available to all of you.

Data-Comp has over $100,000.00 in available parts alone. Also they probably have more service data than anyone. Fact is, we have had some manufacturers ask us for copies of some of their older diagrams.

It is not a cheap operation, in any manner, but it might just be the only hope for some users. And we have attempted to make the thing as inexpensive as possible. Therefore, we even have an "estimate" system for $40.00 plus shipping. That seems a lot better than trashing an otherwise excellent computer system. This service is primarily for "store bought" systems. Kits, while not being outright refused, will have to be evaluated on an individual basis. A telephone call might get a kit approved. Remember, if you have a poorly put together kit, that may have never worked properly, then the problem could well be spread over more than one board or component. In that case it will require some special consideration.

More CoCo Scuttlebutt

Some time back I "rumored" to you that the 128K CoCo was a dead duck. Well, it seems more so now. Despite Tandy and other magazines telling you otherwise. However, another "rumor" is gaining ground, and it might be more accurate than the others.

Seems Tandy DID give up on the bank selected 128K CoCo, or at least let it rest for now (one of their better decisions). Instead they "say" market, later this year, a 512K unit with a real honest-to-goodness MMU (memory management unit). Also it probably will get an 80 character screen (no Charlie, they are not going to introduce a mini-ASR33). This machine will be an OS-9 system primarily, but should have some Microsoft and "old" CoCo compatibility. But I have also heard that it will have enough differences in the Microsoft BASIC portion and DOS that many of the present CoCo users will not be too happy!

Now Tendy's problem has always been with the CoCo's selling price/profit ratio. That is simply put; it sells for less, does more than some of their more expensive units, and thus generates far less "net" profit per unit sale. Example: A customers walks in and ask for a demo of a spread-sheet program. If it is done on the CoCo using Dynacalc and also one of the other Tandy systems, running their other spread-sheet software, the CoCo Dynecalc version is going to be so much better that they are going to lose a large profit sales to a low profit sale. However, if the CoCo is not there to compete, then they automatically get a high profit sale. Can you figure the bottom line?

As far as the price goes, I know that is still up in the air, providing they actually get to market with the new version. But, if they do market (anything 6809) it will NOT BE AS INEXPENSIVE as the present CoCo. And that is what sells the CoCo - low price. Per a survey, over 96% of CoCo users did not know what CPU chip was in the CoCo when they bought it. And 62% still don't know, or care. So look for a machine somewhere in the price range of $599-999. And that eliminates over 72% of the reason that present CoCo users bought their CoCo! They bought on price alone. Stay tuned in for the continuing saga of the "here now, there now" CoCo(?).

The CoCo has done some good for our community. Many original CoCo purchasers found real power with the CoCo, especially with OS-9. However, the CoCo running Os-9 as opposed to a GIMIX or SSB running level-2, is about like hauling a boxcar full of bricks in a VW mini-truck! And for those who attempted to do any serious work in the CoCo DOS - well, that is spelled DUMB. Those same users opted to switch rather than fight. We gained by switching a Tandy customer, to a better product. That is where Tandy needs to shore up their act. However, I predict that it will continue, which is all the better for us. So, please Mr. Tandy - get that new, improved version of the (whatever kind) CoCo on the market, we need it! We've never been against it, we just wish you would learn how to make a "real live full function one"! Maybe this time, "Huh?"

A REAL 68020 SBC is Coming From:
Data-Comp Division

That the 68008 is ~~S-L-O-O-O-W~~ is no secret. That it is to be obsoleted soon, is a fair assumption. We have benchmarks for it, the 6809 and 68020. In many respects it is a real dog. Needless to say, the 68020 is FAR superior to the 68008. Fact is, the 6809 for many operations, is better!

Data-Comp Division is now getting together a complete 68020 system, 1 or 2 mega-byte, that will be not only reasonable in price, but completely compatible with all the present and future 68K software available. This system will come in several versions. One a complete system including the CRT, disk drive(s) (floppy and/ or hard-disk) and cabinet/power supply. For those of you having a GIMIX system, a simple interface to allow power from your GIMIX to run the 68020 system (save a bit). Also a hardware and software combination to allow "net working" the two, or more. Several other "stripped" version are also in the mill.

So, if you have a hankering for a REAL 68XXX system, this will be it. Keep looking for the actual announcement to be published soon. If you want economy, service and support, this will be the one worth waiting for! They could have earlier offered you a dog, but as is their policy, they opted to wait for the real thing. Now watch out for the MUSTANG-20. It packs a real kick! More next month or so.

And Another 68008

And another 68008 SBC board is coming. Can't tell you much about it presently but it should fill a void not yet addressed by other manufacturers.

It will be a combination 6809/68008 system. Although we do not have a lot of faith in the 68008, it will sure be better than some of the others out there. Also as a shared resources system it will allow you a lot more latitude than the existing 68008 systems.

Won't say too much about it now as we have not looked it over or approved for advertising the final version. But I can assure you that it will be a very price and quality worthy system, based on the present products of the manufacturer. So if you really want to get into the 68XXX thing, this gives you another option, from a company that will be around to support what you spend your hard earned bucks for! Which makes waiting a little more worth your money and time.

DMW

- - -

Bit Bucket

PRODUCT ANNOUNCEMENT

6809 - 68008L10 COMBINATION BOARD SOON AVAILABLE

MUSTANG-08/9
Combination
68008L10 & 6809 CPU Board
For the S50 Bus

Scheduled for the 4th quarter 1985, is another 68XXX/6809 S50 bus CPU board, from Data-Comp Division, CPI. A combination 68008L10/68B09E CPU board to convert your present 68XX system to both a full 68008L10 and 6809 computer.

Preliminary specifications provide for a 1 or 2 Mhz 6809 and a full speed (10mhz) 68008L10. These CPUs can run separately or concurrently. Speed is enhanced by the liberal use of 74F type ICs and PAL devices.

- Memory mapping controlled by on-board DAT. The DAT is software selectable. On board address extension to 12 bits for full memory range, with on-board decoding (8 bits).

- Interrupts can cross interrupt other CPU, and 68008L10 can be single stepped by the 6809. Monitor ROM space will have provisions for 4 2716/64 EPROMs and/or 2817A-2864 EPROMs.

- A 68D40 timer installed with no wait states. Both BUS REQUEST and HALT DMA supported, as well as MEMORY READY. E clock is a true square wave for tight timing considerations.

- All connectors gold plated, including jumper pins.

- Several extra 20 pin IC locations for expansion.

- DIP selections use the newer type "piano" dip switches, located on top of the board. This allows all switch setting to be done without removing the board.

This information should, as stated above, be considered preliminary, however, the project has proceeded to the point that the above portion of the specifications are fairly firm. We tell you this now in order that you might consider it as an alternative solution to upgrading to the 68XXX CPU. While not as powerful as the MUSTANG-020, it does allow the present 68XX user a viable upgrade option, using most of his existing hardware, and 6809 software.

Delivery date and final price is not available at this time. However, the price should be very competitive with presently available 68008 systems, that have far less power and options. And again, far less expensive using your old hardware, cabinet, etc.

Interested readers should contact Data-Comp for delivery and price information, but not sooner that October 15, 1985. Delivery slots now being allotted on a reserve basis.

DATA-COMP Division, CPI
5900 Cassandra Smith Rd., Hixson, TN 37343
Telephone: 615 -842-4601
TELEX: 558 414 788284

HAVE YOU READ ANY
GOOD BOOKS LATELY?

by Ron Voigts

I've just gotten a book that might be the answer to many OS-9 users' prayers. The one question I've heard many times is "Is there a good book on OS-9?". At last, I can answer, "There is!"

Now you can get "**The Complete Rainbow Guide To OS-9**" by Dale L. Puckett and Peter Dibble. These are two authors that to most OS-9 users need no introduction. But for newcomers let me tell you about them. Dale Puckett is the president of the OS-9 Users Group and author of the "BasicO9 Tour Guide" from Microware. Peter Dibble is a graduate student (working on a PhD) in computer science at the University of Rochester and author of the "OS-9 User Notes" column in the 68' Micro Journal. Between the two of them, you are looking at a lot of OS-9 knowledge and experience.

Usually when I write about a book I like to tell you what is in it. This book is about OS-9. I mean everything! There's the OS-9 history; its hardware and software. The book talks about the OS-9 file structure and input/output system. The Guide covers all the commands from A to X, I mean form ATTR to XMODE. Puckett and Dibble also cover the languages -- assembly language, BasicO9, C Language and Pascal. There is the OS-9 memory, the disk formats, file managers, device descriptors and drivers. This book covers both Level I and Level II systems. And there are programs, and programs, and more programs! This is all in the book and more. In fact, there are 417 pages of OS-9 information.

The book covers OS-9 (primarily) for the Color Computer user but is useful for the Standard user. It's a good "getting started" book as well as a good guide for the Advanced User for writing Device Descriptors or what ever, and it makes an excellent OS-9 Reference Manual. So, if you are looking for more info about OS-9, take a look at "Puckett & Dibble".

OS-9 USER NOTES REVIEW
Written by Peter Dibble

I was very excited when my fist issue of the 68' Micro Journal came. I read every article and column it had. I paid special attention to one of the columns. It was the "OS-9 USER NOTES," by Peter Dibble. The column interested me, since I am an avid user of OS-9. After reading the column, I had one regret. I wished I had been able to read all the columns from the beginning. Unfortunately, the OS-9 USER NOTES had been published for over a year by the time I started my subscription. So there was over a years worth of valuable information and programs that I had missed.

Now to the rescue comes Peter's new book, THE OS-9 USER NOTES, VOULME ONE. It is a collection of 19 columns, starting with number one and continuing for 18 more. It includes all the programs. It even includes the reviews. Nothing has been left out.

The obvious place to start with a book like this is to begin with column one and go on from there. I think the first columns have special historical significance. First, they were the only literature around at that time for OS-9. Also, they show the development of OS-9. In Column 2, Peter had just received his updated version of the OS-9 system. The new version included commands, like PWD, PXD, and DEL.DIR. OS-9 users now consider these to be standard part of OS-9, but not always. Besides the OS-9 development, the first columns cover good info like forking to a process, using multiple files and interprocess communication.

My problem is I can't wait to finish a book. I guess my curiosity gets the better of me, so I started paging through the OS-9 USER NOTES. There is a lot of stuff here! In the year and a half he has written the column, he has covered quite a bit of material. There is much more covered than I could possibly mention. Besides the things I noted earlier, he covers things like Pascal, directories, device descriptors, pipes, the Color Computer, Compuserve, useful hints, suggestions and OS-9 food for thought.

One nice feature is a 3 page index at the book's end. Indexes are good places to look for things. I found a lot of OS-9 material, I had missed in my random paging. Everything that Peter has written in the last year and a half is neatly indexed for quick reference. Everything he has written in columns 1-19 can be easily found in the index. There are things that I had not even thought about, like the TANO DRAGON, rasterization, and MOTD. Long after you read the OS-9 USER NOTES cover-to-cover, you'll be wanting to look things up. This handy index will let you find them fast.

Peter's book also has all the reviews that appeared in the '68 Micro Journal, They include O-F, OS-9 CIS COBOL, DEDIT, ST9, D-Series Utilities, DYNACALC, DYNAMITE, RMS, RMA, and RLINK. His reviews are easy to follow and understand. He gives a product overview, points out some details and limitations of it, and ends with a neat summary. If you have any interest in these products, you'll want to see his reviews.

Programs! They're all here. All the programs that accompanied the columns are in this volume, They are in BASIC09, C Language, Assembly and COBOL. Some are finished products and others give you a start towards developing your own. There are benchmarks, pipes, device drivers and utilities. There is even a program to make your Color Computer make noise. If you want to add programs to your system (and who wouldn't), you'll want these. And there is more good news. If you hate to type in programs, you can get them all on 8" disk or 5" disks. You can get them in Coco or Standard OS-9 format, too!

If you're like me and missed some of the columns, this is your big chance. They're all here--columns 1-19, the reviews and the programs. Even if you have all the 68' MJ's with the OS-9 USER NOTES , you'll still want this collection for quick easy reference. It puts them all together in one place with a handy index to look up almost anything. The book is available in loose leaf form to add to your user notes or you can get the deluxe binder for it. And don't forget the programs are on disk for your convenience and use. They are all available from:

Computer Publishing Inc.
5900 Cassandra Smith Rd.
Hixson, TN. 37343
(615) 842-4600

PRICES

book.....$9.95
binder.....$5.50
8" disk.....$14.95
5" disks.(2).$24.95

If you interested, give CPI a call. You can also check out the advertisement in this issue for OS-9 USER NOTES, VOLUME ONE. I think you'll really be glad you got it.

- - -

S.M. MEDIA
Special Value Kits
&
Bargain Hunters Program Corner

S.E. MEDIA has always attempted to bring you Software at reasonable prices. Our catalog attest to that fact. And of all the computer magazines I read, 68 Micro Journal brings to you more source listing of useful programs, than any of the others. Also our Reader Disk Service, was a first. It has saved many millions of finger keystrokes, over the years for our group.

However, there is a sort of grey area between those programs we publish and the commercial ones. And that is the area we hope to fill. Programs useful to most of us, but not capable of generating much profit. I know there is a lot of it out there, I receive quite a bit already.

One of the hardest things I have to do sometimes is reject a piece of software for lack of a reasonable market. Or as sometimes happens, the author is not completely knowledgeable as to the many different versions and variants of FLEX. FLEX alone has over 12 different versions that are incompatible, to some degree or another with some specialized software. Without that knowledge many fine pieces of software never made it.

To "Beta" test a particular piece of software we have to run it under several different FLEX systems, with various and assorted digital and linear devices attached to the host system. I have seen many nice software offerings fail due to the reserved space of say a winchester interface board, or a clock or A/D interface board. To name just several of maybe 25 or 30 different devices available to be used with S50 bus machines. Each of which, or the RAM used or reserved, could send your program off to that "land of twisted bits", or worse. I guess that is why the really good software available is more expensive. The time and effort to completely compensate for all those possibilities is costly and time consuming.

You would be surprised at how many software offerings we receive from folks who do not realize that not all of us run FLEX with the memend set to $8FFF. Fact is, most of us do not! If you have SWTPC FLEX for the 6809 you probably use the SWTPC relocating printer drivers, that always reside below $C000. Space must be reserved for them prior to their use. Also practically everyone with a disk system uses some memory below FLEX. Yet, we receive even supposedly commercial software that demands a memend of $8FFF. Considering that probably 60-75% of all FLEX is either GIMIX or SWTPC (not completely the same), each with different RAM considerations, it can really get to be a "can of worms".

If you are going to develope software for resale, especially if you intend to sell it here, then I suggest you make certain you really do understand all the ramifications. That can save us all an awful lot of extra work. Nothing gets my goat more than to have a piece of test software "eat" up a disk, and then discover that the author coded things in such a manner that it would only work with his version of FLEX.

For those offering OS-9 software, things are somewhat better. Some software will not run under level two that was developed for a level one system. Or just the opposite. But on the whole OS-9, because of more stringent control by Microware, and the natural order of OS-9 development requirements, leaves less room for incompatability. However, it does exist and the author should be aware.

So you ask - what is the answer? And to that I can not give a "pat" reply. It would be best if you tried your newly created programs on various S50 bus systems. I imagine that is impossible for some of you. But, that is what we do. We have practically every type S50 bus system and every different version of OS-9 and FLEX that has been released (and some that have never been out of our office). Still occasionally we run into the "personally modified" version, and the author forgot to let us in on the "good news". Those kind sometimes make us want to become cod fishermen or something. Example: a recent offering on a PERCOM disk modified for soft sectoring. Took us a couple of hours to get it off and over to another system for evaluation. It failed.

I will try to help you in any manner I can. But you are going to have to meet me at least half way. If in doubt, call me. Sure it might cost you several bucks, but it could save that much alone, when we start passing the stuff back and forth, by mail, UPS or what have you.

I hope I have not discouraged you. But it is to both our advantage if you know some of the possible problems ahead of time. If you have a good and useful piece of

software, let me know. However, realize that about everyone has an editor, assembler, word processor, BASIC compiler, etc. So what you have to offer must either be a lot better, or sell for a lot less. Otherwise, it could be just wasted effort for a commercial venture.

In the past we have "Beta" tested software, for some of our more successful software vendors, for a reasonable fee. But, this is somewhat different. For them we will continue to make our assorted facilities available for a modest fee. But for the low cost, budget - bargain hunter software offerings, you are going to have to get it right on your own.

If you need advice or help, give me a call.

DHW

My Time

For years now I have occasionally taken the liberty to use a little space here to expound on something I felt was important to us all. But what follows is a first. It is the first time that I have actually "plugged" one of our own products.

However, changing times and conditions have imposed a different slant concerning some aspects of our basic policies. One is our involvement in retailing. It can be a sticky situation. On the one hand we have other supporters (advertisers) who also pay to sell their products in 68 Micro Journal. They are certainly entitled to every consideration. And on the other there is the matter of maintaining a decent publication (68 Micro Journal) for you. We are sometimes between the proverbial rock and a hard place. It has been a tight rope affair so far, and I have tried to not step on any toes or allow our close association with both ventures (publishing and selling) to be a real bone of contention. But it has, at times. And may be again, but we have to consider all our options. And they appear to boil down to this.

First, I am determined to continue the quality of material in 68 Micro Journal, as in the past. Mind you, I said quality, not slick or pretty. Something useful, that is our quality. Bear with me as I burden you with some (decimal) numbers. Each issue of 68 Micro Journal cost us about $1.86 each to print and mail, including postage. Don't grab for your calculator, it comes out to $22.32 for each of you, a year. A net gross profit of only two bucks, more or less. Based on our present (don't anticipate any increase) $24.50 yearly subscription cost to you. Even less if you paid for 2 or 3 years. And all those life members are coasting. Those cost do not cover the cost of our physical plant, our rent, lights, telephone, heating and cooling, insurance, salaries and all those other expenses necessary to keep any business going. That really adds up to a bundle each month. The net result has been that we come out on subscriptions versus net operating expenses to the tune of about minus $10.00 per subscription, per year average.

You might wonder how we do it? Simple, we run lean, by that I really mean super efficient. We use our computers in every aspect of our operation. We make do with about one quarter of the number of office personnel as other publications our size. Also we are supported by those advertising dollars we generate in 68 Micro Journal, and extra income generated by our DATA-COMP and S.E. MEDIA divisions, plus some consulting we do for the electronic and digital industry. However, our advertising income is over 50% down and there is not as much consulting now for 6800 and 6809 markets, in industry, here or abroad, as has been in the past years. Also I occasionally pick up a few bucks on the cream chicken and toast circuit. That my dear friends is why we have hung in there while others collapsed all around us. We operate within what we are able to generate, and we anticipate the coming market potentials, on a continuing basis.

A few years back I was offered over a million bucks for this operation. That was an offer not to be lightly considered. I turned it down. Stupid, you say. Well maybe, but I started this thing, not really to make money (did not/do not turn my nose up at it either) but to do what none of the other magazines would do - give us some exposure and support for our systems! Fact is, I never dreamed it would grow to the proportions it has. I felt that if I sold, they would not actually support us as I thought was needed. History has certainly proven me right on that one already. They did buy another going specialty computer magazine. Paid good bucks, let it slide for a couple of years and then folded the whole operation, practically overnight. Glad it wasn't us! We may some day take a few active stockholder partners, but never let it all go.

Actually all I had in mind to begin with was a newsletter type thing. It just got out of hand with so many manufacturers wanting the advertising exposure, I postponed the first issue for a couple of months, ordered better paper, and -- zip -- 68 Micro Journal was born. I consider myself very fortunate to have had such loyal (for the most part) support from those same advertising folks. We owe them a lot, all of us do!

Meanwhile back at the ranch - The reason I bring this all up is to let you in on where we stand. We are still in good condition, as a business, but we do see the future to be somewhat different than the past. If we continued to just drag along, doing the same old thing, without considering where our market is going, we would be doomed. Maybe not right off, but not too far down the road. Also you have my published pledge not to forget who brought us this far - you and our advertisers. So we are committed to continue with 68XX(X) material. No IBM, Apple stuff or any other brand not using one of the 68XX(X) devices. We have again locked ourselves into a fairly narrow field. But that is what you wanted and paid us for, and that is what you will get. This has left me with a hard decision. Where will the support funds come from?

The support we need will have to come from the three sources mentioned above. With advertising revenue down, expenses up and the market being narrowed considerably, there is just one logical avenue of operation left to us. Sell more of the products you want through DATA-COMP and S.E. MEDIA! If we are to survive and grow, that appears to be the only logical solution. If any of you have a better idea, please let me know.

Also another strange thing. Through it all our subscriptions are still growing. Mostly oversees, but still growing. Which really means we have more loses to contend with (overseas is a little more). Actually we need them, for it is the readers who buy the advertised stuff, that gives our advertisers their advertising budget value. Who contributes to our overall financial welfare. Sorta confusing, but simple if you ponder it in the broad aspect of various sources of revenue. Which boils down to this:

We will be expanding into additional retail resources and marketing. What follows is one that we are presently working into. So please remember, we are not really like all those other magazines. And for the most part you are not like all those other readers, of those other magazines. Which does not mean that we are really different, but I do wonder about those other guys! Imagine buying a computer that is slow, expensive and you can't modify hardly any of the code or program to or modify the thing to do your particular thing. Horrors! But it is happening more and more. Have you noticed that S.E. Media software being sold as their own or under license has source available. And in a form that you can work with. Great, isn't it! Host every piece of S50 bus hardware I know of has more documentation freely available than any of those other guys. Including Hazelwood. And there are still kits and bare boards available. Maybe that is why one of the largest kit manufacturers in the world has recommended 68 Micro Journal to some who asked them where the kits were.

Below is a new item from Data-Comp. We offer the following without apology. We owe it to you, our loyal readers and advertisers, to insure that we continue to survive. We do not desire to compete, just co-exist. Also you will find that most everything we sell is purchased from our supporters and advertisers, as needed. How many can say as much? So, read on.

Data-Comp Mustang-020*
Super SBC

The folks over at our Data-Comp Division have been busy lately, more so than usual. They are putting the finishing touches on their new 2,000,000 byte, right 2 Megabyte 68020 computer. Aptly named - MUSTANG-020 Super SBC. Of course we all know what the 020 means - 68020, right. But why Mustang? Simple, because it has one heck of a kick! And as for Super - well, just look over the specs, and the price.

Bear in mind this is not a stripped down, bear-bones 128K or even 512K system, but a full blown 68020 CPU with 2 Megabytes high-speed RAM. If you are going to do, or already have attempted to do any serious work with a full 32 bit CPU, then you know what I mean. And if you expand it to additional users under the OS9 68K, multi-tasking, multi-user disk system, or any of the other multi-user, multi-tasking disk systems soon to be available, then the extra RAM is imperative! Actually it cannot then be considered extra RAM, it is necessary RAM.

Talking about speed, well, the Mustang-020 Super SBC, compared to the other 68008 or 68000 systems, is a walk-a-way. With the math processor chip installed (option) it does some things so fast, you will think it didn't do anything. Without the co-processor it is still the fastest and most efficient of them all!

Because we wanted (no, demanded) that it be adequate for all available functions, we have designed the power supply to a state of "loaf along". Or if you want, we will furnish the necessary options to allow you to just buy the basic system and use your existing cabinet and power supply (appx. 100 watts required for boards, one floppy and one winchester).

We will be furnishing OS9-68K, Stylo, Mail Merge, Stylo Speiling Checker, JUST (still under 68020 development) and PAT, a new and excellent second editor, also other 68XXX software on a continuing basis. Fact is, Data-Comp has made arrangements with 68 Micro Journal to furnish all 68XXX article programs on a disk, to all MUSTANG-020 owners, for a small handling and shipping charge only. This software alone is worth practically half the price of the basic system. And as time passes it will allow you to continually increase your software acquisition, at a fraction of what others will be paying. In addition there will be a FLEX like (much advanced) 68020 disk system also available, from Star-Kits, called STAR-DOS 68K. This will be a single user system that will use the FLEX disk format. Consider the ramifications of that! Add to that the soon to be released UniFLEX 68020 system, and the possibilities keep expanding! This is not a system that will soon be obsoleted. Others may, but this computer will be the one compared to for years to come.

I understand that several popular C compilers will soon be here, along with a 68XXX version of PL9. And don't forget SCULPTOR (available now from S.E. Media at prices below anyone else), it is running already on the 68020. 68020 software is developing much faster than 68XX software did, so it seems to me. It is because of the simple portability of OS9 and UniFLEX source, and the soon to be released FLEX like (not TSC) 68XXX operating system. Data-Comp will also offer solid discounts on practically all available software for this or any other 68XXX system, for those who purchase a MUSTANG-020 Super SBC direct from Data-Comp or one of their registered dealers. The really nice thing about it is that if you buy a used one from some one, and confirm the original purchase to Data-Comp, you will be eligible for the same free software, discounts and other privileges as the original purchaser, from that date forward. Try and beat that!

I would recommend that you review carefully the specifications in the current Data-Comp 2 page ad. Please note that this is a 2 Megabyte system, it has onboard co-processor and other options, it has bus expandability, it runs at the full 12.5 megahertz. The Motorola 020-BUG (value $500.00 current going price) is included at no additional cost, and is necessary to extract the full 68020 capabilities. The cost difference between this and other 68XXX systems is well worth it!

The MUSTANG-020 Super SBC is not the cheapest 68XXX system around. There are several cheaper 68XXX now being sold. But none, no not one, has the full power of the 68000 series as this 68020 does! The original cost difference is well justified by the power and utility of this as compared to any other 68XXX system. Actually, there are many other 68XXX systems of far less power, at much higher prices. For those of you who cannot understand why the difference in prices between the 68008 and the 68020, note this, the 68008 chip is priced at about $28.00 each. The 68020 chip is on the order of $300.00 to $700.00, according to quantities purchased, etc. However, the price should drop drastically when Motorola changes from the pin grid array to a 64 pin dual inline chip. That should be soon. Still the 68020, considering power, is a bargain. Some stiff price difference, but well worth it! It is kinda like the difference between roller skates and a Lear Jet. If you are serious - then the MUSTANG-020 Super SBC is a **bargain!**

I suggest that if you are serious about a real 68XXX system, you might want to consider the "reservation" list. Just the rumors of this system has generated many solid orders already and production will be behind for awhile.

Call any of us at Data-Comp, we will try to give you a fairly accurate delivery date, and current prices. Or if you just have questions, call, that is what we are here for.

Contacts: Tom, Larry or Don & we thank you.

* MUSTANG-020 and Super SBC are trademarks of Data-Comp Division, CPI - Hixson, TN 37342

DMW

- - -

SARDIS TECHNOLOGIES
2261 E. 11th Ave.
Vancouver, B.C. Canada V5N 1Z7

Larry Williams
'68' Micro Journal
5900 Cassandra Smith
Hixson, TN 37343

Dear Larry:

The review on our ST-2900 6809 system ('68' Micro Journal Dec.'84 pp. 15-17) needs some clarification and amplification.

I must stress that if you purchase our FLEX Conversion Package, no programming of any kind is needed to bring up FLEX for the first time. All you do is follow simple instructions and prompts to insert the right disk at the right time, and 30 seconds later you see the famous "+++" FLEX-is-ready prompt. What could be easier? To create a directly bootable disk you just run the supplied FORMAT and SYSGEN utilities, respond to a few prompts, and presto -- a bootable disk with a configured and linked FLEX.SYS file on it.

The price of the FLEX Conversion disk has always appeared on our price list, and is only (US)$29. Despite the low price, this is more than a mere bare bones adaptation. For example, FLEX's automatic drive searching has been properly implemented. And detailed logging facilities are available to keep track of disk I/O errors (if any).

Source code for FORMAT (the replacement for NEWDISK) is included in our optional "FLEX 1/O Routines Source Code Package". By the way, our FORMAT is fast -- only 58 seconds for a 40 track, double-sided, double-density disk. Complete source code for ST-MON is also available separately on disk.

The trimpot adjustments (for "motor on" and "ready") mentioned in the review are not fussy, and we now supply the DCHECK utility which lets you set them without test equipment. DCHECK also displays the rotational speed of the drives. Some of the competitions' systems require an oscilloscope or frequency counter to adjust the read data separator and write precompensation. Ours are entirely digital, so require no adjustments at all, and will never drift out of adjustment over time due to component aging.

Recent issues of '68' Micro Journal have presented hardware and software patches for some other systems to let them read and write disks formatted with GIMIX vs SWTPc "side flags". ST-2900 users have always been able to read, write, and format FLEX disks in either format -- no patches required.

All partially assembled boards are now tested before shipping, so should come up first try (assuming the customer supplied parts are OK). Unfortunately, Don Williams' set was not tested -- we won't make that mistake again! A few months after the review appeared we also began shipping fully populated/assembled/tested boards.

The manuals referred to in Don's original review were only the pre-production versions -- the current set are considerably improved, both in content as well as in appearance, as Don mentioned in the Feb.'85 update.

The review put considerable emphasis on the effort required to adapt software to the ST-2900 because it doesn't use "the 'normal' 6821 and 6850 ACIA or PIA interfaces." The facts were accurate, but needed to be put into proper perspective. What do all of the following 6809 systems have in common?

- Radio Shack Colour Computer
- Digital Research Uniboard
- Sardis Technologies ST-2900
- Artisan DP-09
- GIMIX with intelligent serial I/O processor board
- GIMIX with memory mapped video board

They all implement the console port with something other than a 6850 addressed at $E004, thus requiring patches to any software that bypasses the FLEX I/O routines. I probably wouldn't have brought this up, except that the Uniboard review the following month didn't have so much as one word on this subject (as opposed to several paragraphs for the ST-2900) even though the same situation exists. In any case, as Don's February update indicated, the supplied instructions on patching the software packages are simple, and "once done they run the same as on any other 6809 system."

The 6522 VIA used to implement the parallel ports is very similar architecturally to the familiar 6821 PIA, when you ignore the VIA's added features such as the shift register and two 16-bit counter/timers. Even its pinout is almost identical. Selected pages from the data sheet are included in our documentation.

The 2681 DUART chip is becoming very popular. Even Motorola is second sourcing it now. With a few changes in some of the control lines, and a different part number (68681), it is used in many 68000 and 8086 based systems.

Although the ST-2900 is a two board system (soon to become three), its inclusion in the "Single Board Computers -- 6809" series of reviews was no accident, as the CPU board can run by itself as a non-disk SBC. One customer is using it as the basis for a print buffer for his Macintosh, another as a Baudot to ASCII converter for RTTY reception (the 6850 ACIA used on other systems cannot handle 5 bit Baudot characters!).

Finally, the ST-MON monitor and the OS-9 Conversion Package have been significantly improved over the past several months. Refer to the enclosed information sheets for details. I recently sent Don Williams copies of the latest versions.

With best regards,

David C. Wiens
Sardis Technologies

UPGRADING OS-9 FROM THE PRELIMINARY TO THE CURRENT VERSION

Contents of the upgrade package

1) ST-2900 OS-9 Conversion Boot disk -- latest version
2) ST-2900 OS-9 Conversion User Manual -- latest version
3) ST-MON 2.04 upgrade package -- described separately

Installation procedure

1) Install ST-MON 2.04 -- refer to the separate ST-MON upgrade instructions.

2) Follow the instructions in the new OS-9 Conversion users manual in these sections:
   a) "Booting Up OS-9 With The Conversion Boot Disk"
   b) "Backing Up The ST-2900 OS-9 Conversion Boot Disk"
   c) "Creating A New Bootable ST-2900 OS-9 System Disk (after booting up from the Conversion Boot disk)"
   d) "Booting From A Configured ST-2900 System Disk"

3) All configured system disks that were created by the preliminary version should be replaced by system disks created by the new version. Files from the "old" disks can be copied over to the "new" disks.

4) Check all of your disks to ensure that the old versions of "OS9Boot", "Sformat", "Bootfix", "Bootbkup" are deleted and replaced by the new versions of "Sformat", "Kernlfix", "Kernlsave", "Modfix", and "Uspeed".

Summary of differences between the preliminary and current versions

1) The new version is compatible with both version 01.00.00 and 01.01.00 of CoCo OS-9 -- the preliminary version would not work with version 01.01.00.

2) The new version will let you run the console terminal (serial Port A) at any baud rate -- the preliminary version would only run at 9600 bps even if you tried changing ST-MON.

3) The new version lets you use terminals that require 8 data bits with the high order bit a zero, as well as those with only 7 data bits -- the preliminary version always assumed 7 data bits.

4) The new version doesn't care about the status of the $\overline{CTS}$ input line on serial Port B during the boot procedure -- the preliminary version would hang up if $\overline{CTS}$ on Port B was not asserted.

5) In conjunction with the new version of ST-MON, you can now boot up from drive 0, 1, 2 or 3, not just from drive 0. This makes it easier if you are using a 3 1/2" drive as drive #0 and want to temporarily connect a 5 1/4" drive as drive #2 or #3 from which to boot up the distribution disks.

6) In conjunction with the new version of ST-MON, you can now use a 96 tpi disk drive for the boot drive without first having to make 96 tpi copies of the ST-2900 OS-9 Conversion Boot disk and the Radio Shack CoCo OS-9 System Master disk.

7) In conjunction with the new version of ST-MON, you can now directly boot up a configured system disk without the hassle of using the Conversion Boot disk.

8) The configured boot disks may now be created in either CoCo or standard OS-9 format.

9) The new MODFIX utility makes it easier to change values in the CLOCK module and disk device descriptors.

10) The new OSPEED utility lets you adjust the two trimpots on the FDC board without any test equipment, the same as with our FLEX Conversion package.

11) The SDISK29 parameters for direct sector read/write have been modified to be compatible with D.P. Johnson's latest version of SDISK. This makes it possible to run his PC-XFER utilities. PC-XFER lets OS-9 read, write, and format IBM PC-DOS / MS-DOS disks, as well as read and write Radio Shack CoCo Disk BASIC disks.

12) The OS-9 kernel is now contained in a regular OS-9 disk file, instead of mysteriously hidden on track 34 (like the CoCo does).

13) The ST-2900 OS-9 Conversion Boot disk has been changed to be in OS-9 format, not FLEX format. This simplifies the procedure to create a backup copy of a configured system disk.

14) The MIZAR (a 68000 VMEbus OS-9/68K system) disk format can now also be read and written.

15) The ST-2900 utilities BOOTBKUP, BOOTSAVE, and BOOTFIX have been replaced by the new KERNLSAVE and KERNLFIX utilities.

16) The track number at which write precompensation is enabled (for double density) is no longer fixed at 43, but can be changed.

17) The DUART driver now also supports 19,200 baud. Since the higher baud rates often require X-ON/X-OFF handshaking, that has been implemented as well.

18) A device driver implementing an 8 bit parallel printer port (using the 6522 VIA) is now included, with source.

19) The source code of all device descriptors is now included on disk.

This document last revised August 2, 1985.

UPGRADING ST-MON FROM VERSION 1.01 TO 2.04

Contents of upgrade package

1) 27J2 EPROM containing ST-MON 2.04
2) ST-MON 2.04 User Manual

If you had previously purchased the source code of ST-MON 1.01, you can get the source code of version 2.04 for only [$US] $lu plus our regular shipping/handling fee.

Installation procedure

1) Turn off the ST-29UU's power supply, wait several seconds to allow the voltages to drop, then disconnect the power supply from the CPU board. Take appropriate precautions to protect against static electricity, to avoid damage to the new EPROM or the rest of the ST-2900 system.

2) Carefully remove the ST-MON I.U1 EPROM from its socket on the CPU board.

3) Insert the ST-MON 2.04 EPROM into its IC socket on the CPU board. If the socket has 28 pins, make sure you "lower justify" the EPROM in its socket, ie. leave pins 1, 2, 27, 28 of the socket open.

4) Change jumper J6 on the CPU board from the "16" to the "32/64" position.

5) Follow the instructions in Appendix C of the new ST-MON manual to configure serial port A for baud rate and data bits.

6) Re-connect the power supply cable to the CPU board.

Summary of differences between version 1.01 and 2.U4

1) Version 1.01 required you to burn a new EPROM in order to change the baud rate of the console port to other than 9600 baud. Version 2.U4 lets you switch select among 300, 1200, 9600, 19200 baud without any changes to the EPROM.

2) Version 1.01 was not compatible with terminals requiring 8 data bits with the high order bit a zero. Version 2.04 lets you switch select between 7 or 8 data bits.

3) The bug that prevented you from specifying 19200 bps has been fixed.

4) V1.01 did not have any command to directly boot up a configured OS-9 system disk -- it always required the intermediate use of the ST-2900 OS-9 Conversion Boot disk. V2.04 lets you directly boot up a configured OS-9 disk.

5) V1.01 did not properly support a 96 tpi disk drive for the drive you boot from. Distribution disks containing the FLEX and OS-9 Conversion packages and the FLEX and OS-9 operating systems, which are usually supplied on 48 tpi disks, first had to be copied to 96 tpi disks. V2.04 eliminates that copying by allowing you to invoke "double-stepping" at boot time to boot directly from 48 tpi disks in a 96 tpi drive. (Note -- the current version of the FLEX Conversion Package has not yet been enhanced to make use of this ST-MON feature.)

6) The OUTCH8 and OUTCHM routines in V1.01 have been replaced by a single new OUTCH routine.

7) The bug in the "input character" routines that did not reset the overrun error condition (and "beeped" every time you keyed a character once the error was set) has been fixed.

8) ST-MON no longer destroys the contents of $00UU-$U0U5 at powerup or system reset.

9) If you should ever get ST-MON's "M2" ("bad RAM") message after powerup or system reset, ST-MON will now tell you what the address of that byte was.

10) V2.04 lets you boot the FLEX or OS-9 operating systems from drive U, 1, 2 or 3, not just from drive 0. This makes it possible to use a 3 1/2" drive as drive #0, a 5 1/4" drive can be temporarily connected as drive 1, 2 or 3 from which to boot up the 5 1/4" conversion package distribution disks. (Note -- the current version of the FLEX Conversion Package has not yet been enhanced to make use of this ST-MON feature.)

11) ST-MON, in a FLEX (or STAR-DOS) environment, now occupies slightly more memory than before, beginning at $F85E instead of $F894.

12) Several new RAM locations have been defined (refer to manual for more details): ARTOPC, ARTOPX, BOOTVE, DBLSTP, DEGFLX, CMDTAB.

13) Two new routines have been added that let you easily switch between the SAM chip's two memory maps (ROM/RAM vs all-RAM).

This document last modified August 5, 1985

***

Don,

I've been hearing about you since you started a newspaper some years ago. Some of us in the ham world were itching to get into computers but didn't think it was practical. Then we heard that you were using a small computer to run several aspects of your paper. That is when we decided that computers were practical after all.

I have been aware of 68MJ for some time but never saw a copy. When I found out you were the publisher I immediately subscribed. I am not disappointed. What your magazine lacks in media it makes up for in class, style, and content.

Now for a question and a tip. First the question. I wrote a simple phone book data base. The data file is too large to fit in memory. I only work on one or two records at a time and move deleted records to the end of the file during disk sort. How can I chop off the end of the file (the part with the deleted records) without using a temporary file to stuff the good stuff? I am using basic09. The problem with the previous method is that I can only use half the disk.

Now the tip. If you accidentally start to format a data or source disk, it is possible to recover part of the text if things haven't gone too far. This is assuming you ripped the disk from the drive before the format got to the data. At this point your root directory is shot and you cannot use chd or dump @/d1. So stick in a blank disk and run format. When it asks for a disk name replace the bland disk with blown disk. Now type in anything and press enter. The formatter will create a new root directory (although practically empty) and then try to verify that the disk is formatted. Nine times out of ten you will get an error, but that's okay. Now you can use chd and dump successfully. Or you can use another disk dump utility as long as it does not require a file name past the root directory.

P.S. "Uncle" Earl Dunn and the 3907 gang want to know when Rufus is going to have the bbq ready?

Thanks.

WB9RLG
Richard King Jr.
P. O. Box 236
St. Bernice, IN
47875

MICRONICS RESEARCH CORP.
33351 ENKAY AVENUE,
ABBOTSFORD,
BRITISH COLUMBIA,
CANADA, V2S 1P2

Microcomputers - Hardware and Software
GIMIX® Sales, Service and Support 1985

Don Williams,
68 MICRO JOURNAL,
5900 Cassandra Smith Road,
Hixson, TN 37343

Dear Don,

If I don't get writing to you soon, I'll probably get out of the habit altogether, but I don't seem to have done too much that's new since I last wrote, except to patch XBASIC to incorporate an EDIT command. In conjunction with BEDIT (a BASIC line-editor) it allows one to enter, say, EDIT 300 (similar to LIST 300) and Line 300 of the XBASIC program would pop up ready for editing - with char-insert, char-delete, overlay, etc. Along the way, of course, I just couldn't resist enhancing BEDIT itself!!

This work got me to thinking that perhaps readers would be interested in learning a little about how XBASIC carries out certain tasks. Several years ago I did a massive study of XBASIC, long before 6809s were thought of. I had a home-brew 6800 system then, with just a cassette-drive for storage, and envied my friends who had a disk-drive, as they were using FLEX2 and XBASIC, while I used either my monitor or Uiterwyk's 8K-BASIC, with no DOS equivalent at all.

So ... commencing with a hex dump of the then current version of XBASIC, I set about the task of disassembling it by hand (as you've guessed, there was no cassette-based disassembler available to me). Took me about 2 weeks, working evenings, before I had it all done - but I sure learned a lot about 6800 machine-language programming along the way, and a lot

more about the inner workings of XBASIC. In no time at all (actually about 3 or 4 days), I had it modified for cassette LOAD and SAVE in place of the disk routines, and a whole new world of programming in BASIC opened up for me. In the beginning, it would take almost 15 minutes to load, which was very frustrating if it didn't load properly and I had to go right back to the beginning. Later on I purchased JPC's high-speed loader, which brought the time down to about 50 seconds!! We sure had fun in those days!

After that, I played around with it for a while - studying the different ways in which it stored programs in memory as compared to saving them on disk (cassette). So let's begin here, keeping in mind that any Hex addresses quoted will vary from one version of XBASIC to another - the ones quoted actually go back to my original FLEX2 version.

Let's suppose we've keyed in the following short program, in the order shown :

```
 5 A=3
10 B=5
 7 C=9
```

A simple LIST would display the lines in correct numerical order, while an examination of the same program saved to disk (using a program such as DISKEDIT) would show exactly the same thing. So it would seem natural to suppose that this is the way XBASIC actually stores the program in memory during a LOAD or keying-in operation. "Hold on there!", exclaims the more knowledgeable reader, "Everyone knows that each line forms part of a linked list, and carries a pointer to the next one in sequence, with the pointer for the final line being 0000." Let us therefore perform a dump of the actual Program-Memory for our little program. Here's what we'd see :

```
4B05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13
  00 05 4B 23 7E 80 00 00 69 A4 00 03 01 33 7D

4B14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22
  00 0A 00 00 7E 80 00 0B 69 A4 00 05 01 35 7D

4B23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30 31
  00 07 4B 14 7E 80 00 16 69 A4 00 09 01 39 7D

4B32 33 34 35 36 37 38 39 3A 3B 3C
  80 41 00 00 00 00 00 00 00 00

4B3D 3E 3F 40 41 42 43 44 45 46 47
  80 42 00 00 00 00 00 00 00 00 00

4B48 49 4A 4B 4C 4D 4E 4F 50 51 52
  80 43 00 00 00 00 00 00 00 00 00
```

Just imagine - all that code to represent our simple 3-line program just after we've keyed it in. What does it all mean?

Well, the first three lines with hex-addresses above are the actual program lines, and the last three form the Variable-Stack, which is where XBASIC keeps track of the values assigned to the variables of the program. Let's begin with the program-lines.

At the left-hand edge, it's quite obvious that the first two columns contain the Line-Number in hex, and in no time at all we see that the next two columns contain the link-address (not the actual line-number) to the start of the next line IN NUMERICAL ORDER (not the order in which we keyed them in). But what about the 7E in column 5? This is XBASIC's way of knowing that the program-line is separated from its line-number by one space. If the statement-line were indented by more than one space, a 7F would be stored here with an extra byte after it to indicate the number of spaces. So you see, as far as program memory is concerned, only one extra byte is used up - no matter how many spaces you indent after the first!! Not so when it's saved to disk, however.

Column 6 contains an 80, which tells XBASIC that the line begins with a Floating-Point variable-name (84 would signify an integer %, and 81 a string $), and the next two bytes indicate the displacement of the variable in the Variable-Stack. A's displacement is 0000, ie it's the first, B commences 11 bytes in, and C 22 bytes in. Notice that the program line itself does NOT specifically identify its variable as A, or B, or C.

69 is the "token" for the '=' sign which follows. More on tokens later, and how you can speed up XBASIC a little by modifying its Command-Table and Statement-Table. Let's move on to the next byte, an A4. All our variables are equal to A4? No, what this means is that each is equal to a constant (not another variable, as in the statement A=X) in the range + or - 32767. The next two bytes indicate in HEX the value to which the variable is being set, followed by its ASCII equivalent. That is, 01 says there is one digit only, and in the case of A, it's equal to ASCII 33 (ie 3). Finally, each line ends with a 7D, which is XBASIC's token for a CR.

There's not much to the Variable-Stack. The initial 80 in the first line says that this variable is FP, its name is ASCII 41 (ie A), and its initial value is 0 (indicated in the following 8 bytes). However, the moment the program is RUN, XBASIC will assign the correct values to these variables as it encounters them in your program.

So much for Lesson One. Try writing one- or two-line programs, with simple statements such as A=X or A%(1)=9, or better still A%(1,2)=15, and examine your Program Buffer to see what happens. Don't forget to clear it to all zeroes before you start, otherwise you'll only get confused with garbage left over from previous tests.

If you'd like me to continue along these lines, including mods to XBASIC, please write to Don and let him know. If I get more YESes than NOs I'll keep going for a while. Bye for now.

Sincerely,

Bob

R. Jones
President

WELLWRITTEN™ ENTERPRISES

COMPUTER CONSULTATION, SOFTWARE, DOCUMENTATION, & HARDWARE

P.O. Box 9802 - 845
Austin, Texas 78766

Wellwritten Enterprises is pleased to announce our relocation to the High-Tech Center of the Southwest--Austin, Texas. We will continue to market the finest inexpensive SS-50 hard disk subsystems. Please note our ad elsewhere in this issue. We anticipate only a brief interruption in service but until our new facilities are completed, we will be without a permanent phone number. Please write to us if you have any questions and, if feasible, we will phone you with the answers. Please include your daytime and evening phone numbers and suggest the best time to call.

Thomas J. Weaver
President

Dear Don

Well hurricane Elena is somewhere to the southeast of us at this time but aside from that nothing else is going on here in Panama City. So I decided to write a program to keep my mind off the weather. The following is what I came up with and I thought that you may be interested.

The Towers of Hanoi game consists of three pegs arranged in a line. On the rightmost peg there are a set of graduated rings, the smallest on the top. The object of the game is to move the rings to the leftmost peg, one at a time, without putting a larger one on a smaller one. This game is sort of a classic.

While reading a text on Ada I came across a neat algorithm that appears to me to be elegant. The thought occurred to me that I could maybe translate it to PL-9. My attempt is enclosed.

Two library files are used; the first strsubs.lib are the string handling routines. The other is iosubs.lib; which are the standard input output routines.

As in all PL-9 programs the main procedure is last, in other words the program starts at "procedure Main". The data of main consists of three strings and an integer. The first three lines preload the strings with the names of the pegs. Then a new line command is given. Next the program requests the number of rings to move. This may be any number between 1 and 99. I would not recommend trying over 10 as it may take a while to print all the moves. The number of rings is returned from the function IN_NUMBER. At this time another new line (crlf) is issued. Then the procedure MOVE_THEM is called and another new line command is given.

Procedure IN_NUMBER allows the inputing of a two digit decimal number. This is accomplished by passing the address of a two byte string to the procedure input, one of the procedures in iosubs. The while loop converts this string to an integer. This integer is then passed back to the calling routine.

MOVE_THEM is a doubly recursive procedure. This means that it calls itself twice. Each time it calls itself N is decremented until N is equal to one. Then it starts backing out, but in the process it calls itself again each time. I know of no simpler explanation of this. Of course this procedure is at the heart of the algorithm, and perhaps some other reader can come to my rescue with a better explanation of this routine. MOVE_THEM calls PRINT_MOVES twice. First when N goes to one and then each time the routine backs out.

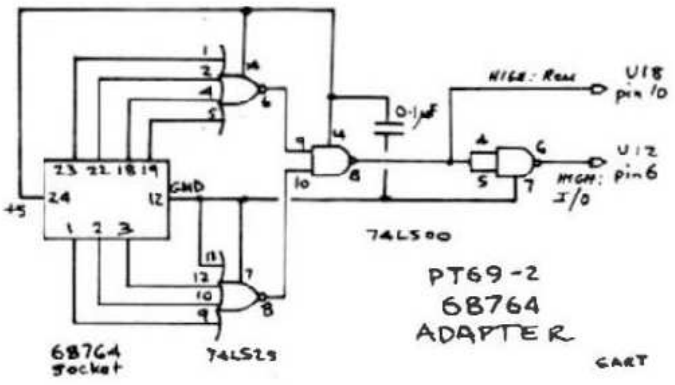
PRINT_MOVES prints the two strings Source and Destination with an arrow between them. It then does a carriage return - line feed. Print and crlf are in iosubs.

I do want to apologize again for not having a better explanation of MOVE_THEM but it gets pretty tangled as N increases. I would like to point out that the second time it calls itself the three strings are displaced one place to the right in a circular manner.

Clell Dildy

```
/*************************************
*                                   *
*        A Program to Solve         *
*       the Towers of  Hanoi        *
*             by                    *
*          Clell  Dildy             *
*                                   *
*          (Recursive)              *
*************************************/

include 0.strsubs; /* strcopy routine */
include 0.iosubs; /* print, crlf and input routines */

/* Input error routine */
procedure ERROR;
   crlf;
   print("You have entered an illegal character!!");
   crlf;
   call gCDOS; /* Exit back to FLEX */
endproc;

/* Input a two digit decimal number */
procedure IN_NUMBER; byte String(3); integer Number, J;
   J = 0;
   Number = 0;
   input(.String, 2);
   while String(J) begin
      if String(J) < '0' or String(J) > '9' then ERROR;
      Number = String(J) - '0' + 10 * Number;
      J = J + 1;
   end; /* while */
   if Number <= 0 then ERROR;
endproc integer Number;

/* Print the solution */
procedure PRINT_MOVES(byte .Source, .Destination);
   print(.Source);
   print(" --> ");
   print(.Destination);
   crlf;
endproc;

/* Routine to play the game -- RECURSIVE */
procedure MOVE_THEM(integer N; byte .Source, .Other, .Destination);
   if N = 1 then PRINT_MOVES(.Source, .Destination);
   else begin
      MOVE_THEM(N - 1, .Source, .Destination, .Other);
      PRINT_MOVES(.Source, .Destination);
      MOVE_THEM(N - 1, .Other, .Source, .Destination);
   end; /* if */
endproc;

procedure MAIN; byte Left(5), Center(7), Right(5);
                integer Ring_number;
   strcopy(.Left, "Left");
   strcopy(.Center, "Center");
   strcopy(.Right, "Right");
   crlf;
   print("How many rings to move? ");
   Ring_number = IN_NUMBER;
   crlf;
   MOVE_THEM(Ring_number, .Left, .Center, .Right);
   crlf;
```

ANOTHER HOD TO THE PERIPHERAL TECHNOLOGY PT69 SINGLE BOARD COMPUTER



G A R Trollope

I have just got one of these great little boards to run OS9. Considering all the outstanding capabilities of this combination, I wanted to derive the maximum potential. So, the challenge was to figure out how to squeeze a little more out of the board. My interest was especially piqued by the fact that 68764's, which are 8k EPROMS, will work in the EPROM socket, albeit only one half at a time. My SwTPC system is modified so that the address space is completely decoded, and the I/O takes up only 32 bytes of space out of the corner of a 2716 EPROM. So, this extends from $E020 to $E7FF. It turns out that a similar decoding scheme works on the PT69 board as well, so that the 68764 will extend from $E020 to $FFFF. I plan to use the additional EPROM space for some of the OS9 system routines, such as SHELL, CLOCK, etc. This will make the Boot shorter, giving me about 16 extra pages.

The additional decoding consists of only 2 IC's, which can best be mounted on a daughter board that plugs into the EPROM socket. For a plug, I use a 24 pin wire-wrap socket---which becomes the EPROM socket, and provides address and power lines for the decoder. Two wires (each about an inch long) extent from the board to the mother board. The traces to the existing connections on the IC's have to be cut, ofcourse, but both cuts can be made on the solder side of the board.

To complete the modifications, address lines A11 and A12 have to be connected to the EPROM socket. While the jumper J8 could conect A11 to pin 21, in the 68764 A12 (pin 20 on the 6809E) should connect here. Likewise, run a jumper from the A11 connection on J8 to the centre connection on J13. The work is completed by joining the two outer connections on J13, to ground the chip select. It may be neceeeery to remove R20, to reduce the bus loading.

OK, here's the circuit diagram, but if enough people show an intereet, Peripheral Technology will make up a plug in board, eo, give them a call at 404-973-0042.

***

MOTOROLA INC.

Microprocessor Products Group
P.O. Box 3000
Austin, Texas 78764

EDITORIAL CONTACT: Val Bauer

512/928-6804

READER CONTACT: Wade Wyatt

For further information contact

512/440-2035

8-BIT MPU AND MCU FAMILIES BROCHURE AVAILABLE

AUSTIN, TEXAS AUGUST 15, 1985... The Microcomputer Operation of Motorola's Microprocesaor Products Group announces availability of a brochure illuetrating the breadth and versatility of Motorola's 8-bit MPU/MCU product families

Motorola's three MCU families--M6801/M68HC11, M6805, and M6804--and two MPU familiee, the M6800 and M6809, are clearly illuetrated and deacribed in this 12-page document. It not only covers the hardware, eoftware and option features for the core MPU and MCU familiee, but deacribes available development support and new products as well.

Thie eaay-to-uee selector guide ia available through local Motorola ealee offices or authorized Motorola diatributors. Ording using BR261.

Classified Advertising

1-CIMIX #79 Syatem OS/9 III, with 5 Mega Byte Mini-Hard Disk plus 5" DSDD Drive, also Dual 8" DSDD Disk System with Controller, 6-Serial Porta, 2-Parallel Porta, 256K #72 Card and 4-#67 64K Cards.

1-PT-69 complete with Dual 5" DSDD Disk System and Controller, Includes FLEX DOS.

TELETYPE Model 43 PRINTER - with serial (RS232) interface, and full ASCII keyboard. LIKE NEW - New cost $1295.00 - ONLY $559.00 ready to run.

S/09 with Motorola 128K RAM, 1-MPS2, 1-Parallel Port, MP-09 CPU Card $1990. 1-DMAF2 Dual 8" Drives with Controller $2190. 1-CDSI 20 Meg Hard Disk Syatem with Controller $2400.

1-S+ Syatem with 256K RAM, 1-MPS4, 1-Parallel Port, MPU Processor Board, UniFLEX Included $2700. 1-QUI 10 Meg 5" Hard Disk & 8" Disk Drive with DMAP3 Controller Board $3600. 1-X12 Terminal 12" $1260. 1-Cabinet for S+ System w/Filler Plate $500. 2-8212 Terminals $495.

I will accept any reasonable counter-offer !!

Call Tom (615) 842-4600 M-F 9 A.M. to 5 P.M. E.S.T.

***

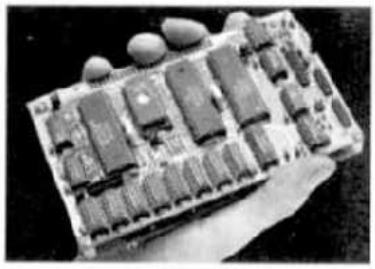
Several SWTP S+ Systems Available: Below dealer cost, new equipment warranty, lots of software available. Finders Fee. 512-828-2679

***

For sale: FLEX 6809 software. Most of the TSC packages plus CSC Dynacalc and Dynamite+. All approximately half price. Also IBM 8 inch diskettes. SASE of call for list. Frank Weaver, 22 Elizabeth St., So. Burlington, Vt. 05401, (802) 863-2256, after 5PM EST.

***

For Sale: Cimix: #05-CPU, #68-Controller, #43-Serial, #42-Parallel, #76-Video, (CMXBUG09-V2.2F, Autoboot-V1.5, OS9 CMXII-V1.2 never installed); 2-Digital Research 56K-150ns Static Ram, 1-56K kit; SWTPC MP-R Prom Burner; ACORN 16BK Prom board. Following assembled untested: AAA Chicago Electra Motherboard w/gold connectors, Electra DPS Dual Serial; Sunny International PS +8v/25a, +16v/3a, -16v/3a; Acorn XADR Extended Addressing, FD88 Firmware Development. B.S. PALMER, P.O. Box 115, Portsmouth, Va. 23705. (804) 398-6646. $1800. OBO.



512K RAM Expansion

Compact Flexible 6809 Computer

The new ST-2900 system -- a complete 64K small business or hobbyist computer is only one of its many possible configurations. Among its features are:

- Small enough to hold in your hand! (Eurocard size: 3.9"x6.3")
- Three board "system" for greater versatility than single board computers.
- CPU Board -- powerful 6809E processor, 16K or 4K RAM, 1K-32K EPROM, 2 RS232 serial ports with software programmable baud rates, 16 bit counter/timer. Run the CPU board all by itself, or plug your own custom board or our FDC board and/or RAM-512 board into the expansion connector.
- FDC Board -- double-sided/double density floppy disk controller with adjustment free digital data separator and write precompensation. 2 8-bit parallel ports, 2 16-bit counter/timers, prototyping area.
- **RAM-512 Board** -- 524,288 bytes of RAM on a 4.15"x6.3" board! Low power. Includes RAM Disk software for FLEX or OS-9.
- **FLEX and OS-9 supported -- software selectable.**
- OS-9 Conversion Package lets you use the low cost Radio Shack CoCo version of OS-9 on our ST-2900 system. Save $131 off the suggested list price of OS-9! **No programming is involved.** Supports CoCo OS-9, standard OS-9, and MIZAR OS-9/68K disk formats. Compatible with PC-XFER to let you **read/write/format MS-DOS disks!**

• CPU bare board plus EPROM	$45	OS-9 Conversion Package	$49
FDC bare board	$36	FLEX Conversion Package	$29
RAM-512 board A&T	$485	CPU + FDC + OS-9 Conversion	$119
CPU + FDC board set assembled and tested			$329

• Add $5 shipping/handling ($10 overseas). These prices are in U.S. funds. C.O.D. orders: call or write for prices. Terms: check, money order, VISA.

disclaimers: FLEX -- Technical Systems Consultants; OS-9 -- Microware & Motorola; MS-DOS -- Microsoft

RAM-512 delivery scheduled to begin end of Sept. '85

SARDIS TECHNOLOGIES

Call or write for free brochure and complete price list
(604) 255-4485

2261 E. 11th Ave. Vancouver, B.C., Canada V5N 1Z7

ARCADE 50

POWERFUL COLOR GRAPHICS
Uses the new TMS9918A Video Display processor. High resolution 256 x 192 pixel display with 15 colors. 16K Bytes of onboard RAM does not reduce user memory. 32 graphic images can be individually moved with simple X-Y commands for smooth animation. External Video input allows subtitling. NTSC composite video output.

SOUND EFFECTS AND MUSIC
- Three AY3-8910 Programmable Sound Generators
- Nine simultaneous voices
- Three independent noise sources
- Onboard stereo amplifier drives two 8 ohm speakers

ADDITIONAL I/O CAPABILITIES
- Eight analog inputs with 8 bit resolution
- Supports four joysticks with pushbutton switches
- Eight bit parallel I/O port
- Entire unit mapped into 256 bytes of memory

FBASIC

TERMINUS DESIGN INC. in conjunction with Microware Systems Corporation, is proud to announce FBASIC an enhancement of Microware's 6800/BASIC. Their fast compiled BASIC has been adapted for 6809 users with added video and sound features for ARCADE 50 users. FBASIC is a true compiler that produces optimized machine language modules which are ROMable and require no Run-Time package. FBASIC requires less memory overhead and runs hundreds of times faster than BASIC interpreters. It supports standard BASIC instruction including String functions, Disk I/O and fast integer arithmetic with multiple-precision capability. Graphics verbs and functions fully support the Arcade 50.

ARCADE 50 assembled and tested	$325.00
Video and Audio connector set	15.00
4 Joystick connector set	15.00
2 Radio Shack Joysticks	24.00
Gold Molex connectors	12.00
A/BASIC for 6800	110.00
FBASIC for 6809	110.00
FBASIC (with ARCADE 50)	75.00
ARCADE 50 RGB	375.00
LABVIDEO (Motorola EXORcisor)	375.00
NEW MW09 6809 Processor Board	225.00
256K Dynamic Memory Board	795.00
256K Dynamic Memory Board (w/64K)	395.00
64K Dynamic Memory Board	295.00

TERMINUS DESIGN INC
16 SCARBROUGH ROAD
ELLENWOOD, GA 30049
(404) 474-4866

TERMS: CASH, CHECK, M.O.

OS-9™ SOFTWARE

SDISK—Standard disk driver module allows the use of 35, 40, or 80 track double sided drives with COCO OS-9 plus you can read/write/format the OS-9 formats used by other OS-9 systems. **$29.95**

SDISK + BOOTFIX—As above plus boot directly from a double sided diskette **$35.95**

FILTER KIT #1—Eleven OS-9 utilities for "wild card" directory lists, copies, moves, deletes, sorts, etc. Now includes disk sector edit utility also. **$29.95 ($31.95)**

FILTER KIT #2—Macgen command macro generator builds new commands by combining old ones with parameter substitution, 10 other utilities. **$29.95 ($31.95)**

HACKER'S KIT #1—Disassembler and related utilities allow disassembly from memory, file. **$24.95 ($26.95)**

PC-XFER UTILITIES —Utilities to read/write and format MS-DOS™ diskettes on CoCo under OS-9. Also transfer files between RS disk basic and OS-9 (requires sdisk). **$45.00**

BOLD prices are CoCo OS-9 format disk, other formats (in parenthesis) specify format and OS-9 level. All orders prepaid or COD, VISA and MasterCard accepted. Add $1.50 S&H on prepaid, COD actual charges added.

SS-50C
1 MEGABYTE RAM BOARD

Full megabyte of ram with disable options to suit any SS-50 6809 system. High reliability, can replace static ram for a fraction of the cost, $895 for 2 Mhz or $995 for 2.25 Mhz board assembled, tested and fully populated. (Add $6 shipping and insurance, quantity discounts available.)

D.P. Johnson, 7655 S.W. Cedarcrest St.
Portland, OR 97223 (503) 244-8152
(For best service call between 9-11 AM Pacific Time.)

OS-9 is a trademark of Microware and Motorola Inc.
MS-DOS is a trademark of Microsoft, Inc.

COMPILER EVALUATION SERVICES
By: Ron Anderson

The S.E. MEDIA Division of Computer Publishibg Inc., is offering the following SUBSCRIBER SERVICE:

COMPILER COMPARISON AND EVALUATION REPORT

Due to the constant and rapid updating and enhancement of numerous compilers, and the different utility, appeal, speed, level of communication, memory usage, etc., of different compilers, the following services are now being offered with periodic updates.

This service, with updates, will allow you who are wary or confused by the various claims of compiler vendors, an opportunity to review comparisons, comments, benchmarks, etc., concerning the many different compilers on the market, for the 6809 microcomputer. Thus the savings could far offset the small cost of this service.

Many have purchased compilers and then discovered that the particular compiler purchased either is not the most efficient for their purposes or does not contain features necessary for their application. Thus the added expense of purchasing additional compiler(s) or not being able to fully utilize the advantages of high level language compilers becomes too expensive.

The following COMPILERS are reviewed initially, more will be reviewed, compared and benchmarked as they become available to the author:

PASCAL "C" GSPL WHIMSICAL PL/9

Initial Subscription - $ 39.95
(includes 1 year updates)
Updates for 1 year - $ 14.50

S.E. MEDIA - C.P.I.
5900 Cassandra Smith Rd.
Hixson, Tn. 37343
(615) 842-4601

	ASSEMBLED	BARE
Elektra Super Bus (Fits Current Elektra Chassis)	Coming	
Elektra Super 68XX Boards	Coming	
Cabinet w/Power Supply (Largest SS-50 Cab. Available)	450.00	
Disk Regulator	50.00	
Disk Regulator (Heavy Duty)	75.00	
Motherboard (Gold Connectors)	395.00	
2MHz 6802/6809 CPU	275.00	65.00
Dual Port Serial Card	95.00	40.00
Dual Port Parallel Card	80.00	40.00
64K Static Ram Card	250.00	N/A
5"/8" 2MHz Super Floppy Controller (The Best)	295.00	100.00
Flex/Star DOS Drivers	30.00	
OS-9 Drivers	50.00	
Elektra OS-9 w/Editor, assembler, Debugger	250.00	
Elektra Star DOS	75.00	

	ASSEMBLED	BARE
4K Numbug	75.00	
2K Microbug	40.00	
7-84 MByte Winchester Systems	-Phone-	
Removable Cartridge Winchester Systems	-Phone-	
DMA Winchester/Floppy SS-50 Host Interface	395.00	250.00
HD-5 Dual Floppy Cabinet with Power Supply	150.00	
HD-5W Floppy/Winchester Cabinet with Power Supply	199.00	
HD-50W Dual Winchester Cabinet with Power Supply	250.00	
Ribbon Cable for Dual Outboard Floppies	40.00	
Ribbon Cable for Dual Inboard Floppies	35.00	
HD-8 Dual 8" Floppy Cabinet with Power Supply	350.00	
Ribbon Cable for Dual 8" Floppies	65.00	
Custom Cables On Request	-Phone-	
30 Pin Prototyping Board	30.00	
50 Pin Prototyping Board	50.00	
CONNECTORS		
Gold Plated 10 Pin Male (Square Posts) or Female	1.50	
Tin Plated 10 Pin Male (Square Posts) or Female	1.50	

Write or phone for current pricing

AAA Chicago Computer Center
120 Chestnut Lane — Wheeling IL 60090
(312) 459-0450

Technical Consultation available most weekdays from 4 PM to 6 PM CST

ELEKTRA is a trademark of AAA Chicago Computer Center
FLEX is a trademark of Technical Systems Consultants, Inc.
OS-9 and BasicO9 are trademarks of Microware Systems Corp.
UniFLEX is a registered trademark of Technical Systems Consultants, Inc.

FINALLY, A PORTABLE OFFICE

(And A Way to Add IBM-PC/OS9 Compatibility)

Are you trying to get your personal and business affairs in order when your desk is out of order? Are you attempting to operate swiftly, efficiently and effectively when your reports, records and data are buried under a Matterhorn of clutter?

Now, you can clean up your act, by cleaning up your desk—because we've put together a Corona Portable Computer with Alpha Software's "Electric Desk". It's Corona's comprehensive, yet simple to use, Portable Office Package that allows you to concentrate on your work, instead of concentrating on trying to find your work.

You get the Corona PPC-400, the most advanced portable computer that's IBM* PC compatible. It features a high resolution screen (twice the resolution of the IBM-PC), a 360K floppy disk drive and 256K RAM (with room for more drives and RAM). And, as you grow, the Corona PPC-400 will grow with you. It has expansion slots for additional PC boards, and built-in serial and parallel ports so you can add other peripherals.

$1195 AT A VERY SPECIAL PRICE

What's more, the system comes complete with "Electric Desk" ... a powerful, easy-to-learn and use package that includes word processing, spreadsheets, database management and communications (suggested retail value is $345).

IBM/ OS9 COMPATIBILITY — Smoke Signal offers a set of programs that brings compatibility to your OS9 computer. When you order a Corona PC, we can offer you two sets of programs at a very special price that provide compatibility between the Corona and your OS9 system. The first set of programs allows your Smoke Signal OS9 system to list the directory, print the contents of a file and copy files to and from a disk created on the Corona. The second set of programs allows the Corona to be used as a terminal on Smoke Signal and many other OS9 systems and to transmit files back and forth between systems. This permits the OS9 system to be used as a file server for the PC. These two sets of programs are normally sold by their authors for over $600. With the purchase of a Corona PC, we can offer them to you for just $199.

HARD DISK AND DESK TOP PC's are available at very special prices available only to Micro Journal readers and Smoke Signal customers through October 15. Call (818) 889-9340 for complete information on pricing and available configurations.

SMOKE SIGNAL corona data systems, inc. / Electric Desk

31336 Via Colinas, Westlake Village, CA 91362

Phone (818) 889-9340 • TLX 910-494-4965

*IBM is a registered trademark of International Business Machines, Inc.

FEATURES THE POWERFUL, THIRD GENERATION, MOTOROLA 6809 PROCESSOR!

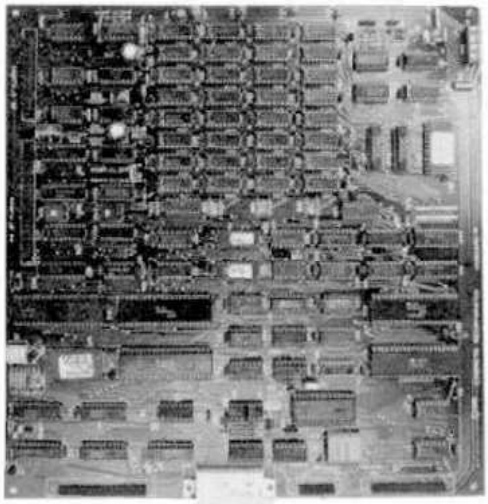
THE 6809 "UNIBOARD"™

SINGLE BOARD COMPUTER KIT

PERFECT FOR COLLEGES, OEM'S, INDUSTRIAL AND SCIENTIFIC USES!

64K RAM! DOUBLE DENSITY FLOPPY DISK CONTROLLER!

New! *Lower Price!*



BLANK PC BOARD
$99.95
WITH PAL'S, AND TWO EPROMS.

FOR 5-1/4 OR 8 INCH SOURCE DISKETTE ADD $10.

$219.00
COMPLETE KIT! FULLY SOCKETED.

PRICE CUT!!

THE COMPACTA UNIBOARD™: Through special arrangement with COMPACTA INC., we are proud to have been selected the exclusive U.S. Mfg. of their new 6809 UNIBOARD™ COMPUTER KIT. Many software professionals feel that the 6809 features probably the most powerful instruction set available today on ANY 8 bit micro. Now, at last, all of that immense computing power is available at a truly unbelievably low price.

FEATURES:

- 64K RAM using 4116 RAMS.
- 6809E Motorola CPU.
- Double Density Floppy Disk Controller for either 5-1/4 or 8 inch drives. Uses WD1793.
- On board 80 x 24 video for a low cost console. Uses 2716 Char. Gen. Programmable Formats. Uses 6845 CRT Controller.
- ASCII keyboard parallel input interface. (6522)
- Serial I/O (6551) for RS232C or 20 MA loop.
- Centronics compatible parallel printer interface. (6522)
- Buss expansion interface with DMA channel. (6844)
- Dual timer for real time clock application.
- Powerful on board system monitor (2732). Features commands such as Go To, Alter, Fill, Move, Display, or Test Memory. Also Read and Write Sectors. Boot Normal, Unknown, and General Flex™.

YOUR CHOICE OF POPULAR DISK OPERATING SYSTEMS:

FLEX™ from TSC	$99
OS9™ from Microware	$199

Specify 5-1/4 or 8 Inch

PC BOARD IS DOUBLE SIDED, PLATED THRU SOLDER MASKED, 11 x 11-1/2 IN.

ALL SALES ARE MADE SUBJECT TO THE TERMS OF OUR 90 DAY LIMITED WARRANTY. A FREE COPY IS AVAILABLE UPON REQUEST.

Digital Research Computers
(OF TEXAS)
P.O. BOX 461565 • GARLAND, TEXAS 75046 • (214) 225-2309

TERMS: Shipments will be made approximately 3 to 6 weeks after we receive your order. VISA, MC, cash accepted. Add $4.00 shipping. USA AND CANADA ONLY

SOFTWARE FOR 680x SYSTEMS

SUPER SLEUTH DISASSEMBLERS
EACH $99-FLEX $101-OS/9 $100-UNIFLEX
OBJECT-ONLY versions: EACH $50-FLEX,OS/9,COCO
interactively generate source on disk with labels, include xref, binary editing
specify 6800,1,2,3,5,8,9/6502 version or Z80/8080,5 version
OS/9 version also processes FLEX format object file under OS/9
COCO DOS available in 6800,1,2,3,5,8,9/6502 version (not Z80/8080,5) only

CROSS-ASSEMBLERS (TRUE ASSEMBLERS, NOT MACRO SETS)
EACH $50-FLEX,OS/9,UNIFLEX ANY 3 $100 ALL $200
specify for 1802,6502,6801,6804,6805,6809,Z8,Z80,8048,80 1,8085,68000
true, modular, free-standing cross-assemblers in C, with load/unload utilities
8-bit (not 68000) sources included with all cross-assemblers (for $200)

DEBUGGING SIMULATORS FOR POPULAR MICROPROCESSORS
EACH $75-FLEX $100-OS/9 $80-UNIFLEX
OBJECT-ONLY versions: EACH $50-COCO FLEX,COCO OS/9
interactively simulate processors, include disassembly formatting, binary editing
specify for 6800/1, (14)6805, 6502, 6809 OS/9, Z80 FLEX

ASSEMBLER CODE TRANSLATORS FOR 6502, 6800/1, 6809
6502 to 6809 $75-FLEX $85-OS/9 $80-UNIFLEX
6800/1 to 6809 & 6809 to position-ind. $50-FLEX $75-OS/9 $60-UNIFLEX

FULL-SCREEN XBASIC PROGRAMS with cursor control
AVAILABLE FOR FLEX, UNIFLEX, AND MSDOS

DISPLAY GENERATOR/DOCUMENTOR	$50 w/source, $25 without
MAILING LIST SYSTEM	$100 w/source, $50 without
INVENTORY WITH MRP	$100 w/source, $50 without
TABULA RASA SPREADSHEET	$100 w/source, $50 without

DISK AND XBASIC UTILITY PROGRAM LIBRARY
$50-FLEX $30-UNIFLEX/MSDOS
edit disk sectors, sort directory, maintain master catalog, do disk sorts,
renumber some or all of BASIC program, xref BASIC program, etc
non-FLEX versions include sort and renumberer only

CMODEM TELECOMMUNICATIONS PROGRAM
$100-FLEX,OS/9,UNIFLEX
OBJECT-ONLY versions: EACH $50-FLEX,OS/9
menu-driven with terminal mode, file transfer, MODEM7, XON-XOFF, etc.
for COCO and non-COCO, drives internal COCO modem port up to 2400 Baud

HARDWARE & SERVICES

5.25" DISKETTES
EACH 10-PACK $12.50-SSSD/SSDD/DSDD $20-DSOD
American-made, guaranteed 100% quality, with Tyvek jackets, hub rings, and labels

SS-50C 256K 1.5MHz MEMORY BOARDS
EACH BLANK $80 ASSEMBLED AND TESTED $350
with instruction manual, schematics, and delay line; all parts readily available

ADDITIONAL SERVICES FOR THE COMPUTING COMMUNITY
CUSTOMIZED PROGRAMMING
we will customize any of the programs described in this advertisement or in our brochure for specialized customer use or to cover new processors; the charge for such customization depends upon the marketability of the modifications.
CONTRACT PROGRAMMING
we will create new programs or modify existing programs on a contract basis, a service we have provided for over twenty years; the computers on which we have performed contract programming include most popular models of mainframes, including IBM, Burroughs, Univac, Honeywell, most popular models of minicomputers, including DEC, IBM, DG, HP, AT&T, and most popular brands of microcomputers, including 6800/1, 6809, Z80, 6502, 68000, using most appropriate languages and operating systems, on systems ranging in size from large telecommunications to single board controllers; the charge for contract programming is usually by the hour or by the task.
CONSULTING
we offer a wide range of business and technical consulting services, including seminars, advice, training, and design, on any topic related to computers; the charge for consulting is normally based upon time, travel, and expenses.

Computer Systems Consultants, Inc.
1454 Letta Lane, Conyers, GA 30207
Telephone 404-483-1717 or 4570

We take orders at any time, but please plan discussions after 6, if possible.

Contact us about catalog, dealer, discounts, and services.
Most programs in source: give computer, OS, disk size.
25% off multiple purchases of same program on one order.
VISA and MASTER CARD accepted; US funds only, please.
Add GA sales tax (if in GA) and 5% shipping.
(UNIFLEX tm Technical Systems Consultants; OS/9 tm Microware; COCO tm Tandy.

SOFTWARE FOR THE HARDCORE

** FORTH PROGRAMMING TOOLS from the 68XX&X **
** FORTH specialists — get the best!! **

NOW AVAILABLE — A variety of rom and disk FORTH systems to run on and/or do TARGET COMPILATION for

6800, 6301/6801, 6809, 68000, 8080, Z80

Write or call for information on a special system to fit your requirement.

Standard systems available for these hardware—

EPSON HX-20 rom system and target compiler
6809 rom systems for SS-50, EXORCISER, STD, ETC.
CO OR COMPUTER
6800/6809 FLEX or EXORCISER disk systems.
68000 rom based systems
68000 CP/M-68K disk systems, MODEL II/12/16

tFORTH is a refined version of FORTH Interest Group standard FORTH, faster than FIG-FORTH. FORTH is both a compiler and an interpreter. It executes orders of magnitudes faster than interpretive BASIC. MORE IMPORTANT, CODE DEVELOPMENT AND TESTING is much, much faster than complied languages such as PASCAL and C. If Software DEVELOPMENT COSTS are an impor ant concern for you, you need FORTH!

firmFORTH℠ is for the programmer who needs to squeeze the most into roms. It is a professional programmer's tool for compact rommable code for controller applications.

* tFORTH and firmFORTH are trademarks of Talbot Microsystems
* FLEX is a trademark of Technical Systems Consultants, Inc.
* CP/M-68K is trademark of Digital Research Inc.

tFORTH™ from TALBOT MICROSYSTEMS NEW SYSTEMS FOR 6301/6801, 6809, and 68000

—> tFORTH SYSTEMS <---

For all FLEX systems: GIMIX, SWTP, SSB, or EXORcisor Specify 5 or 8 inch diskette, hardware type, and 6800 or 6809.

** tFORTH — extended fig FORTH (1 disk) $100 ($15)
with fig line editor.
** tFORTH + — more! (3 5" or 2 8" disks) $250 ($25)
adds screen editor, assembler, extended data types, utilities, games, and debugging aids.
** TRS-80 COLORFORTH — available from The Micro Works
** firm FORTH — 6809 only. $350 ($10)
For target compilations to rommable code.
Automatically deletes unused code. Includes HOST system source and target nucleus source. No royalty on targets. Requires but does not include tFORTH +.
** FORTH PROGRAMMING AIDS — elaborate decompiler $150

** tFORTH for HX-20, in 16K roms for expansion unit or replace BASIC $170

** tFORTH 68K for CP/M-68K 8" disk system $290
Makes Model 16 a super software development system.

** Nautilus Systems Cross Compiler
— Requires: tFORTH + HOST + at least one TARGET:
— HOST system code (6809 or 68000) $200
— TARGET source code: 6800-$200, 6301/6801—$200
same plus HX-20 extensions— $300
6809—$300, 8080/Z80—$200, 68000—$350

Manuals available separately — price in ( ).
Add $6 system for shipping. $15 for foreign air.

TALBOT MICROSYSTEMS 1927 Curtis Ave., Redondo Beach, CA 90278 (213) 376 9941

WINDRUSH MICRO SYSTEMS

UPROM II

PROGRAMS and VERIFIES: I2758, T2508, I2716, T2516, I2732/2732A, MCM68764/6, I2764/2764A, T2564, I27128/27128A, and I27256. I=Intel, T=Texas, M=Motorola.

NO PERSONALITY MODULES REQUIRED!

TRI-VOLT EPROMS ARE NOT SUPPORTED

INTEL's intelligent programming (ta) implemented for Intel 2764, 27128 and 27256 devices. Intelligent pr gramming reduces the average programming time of a 2764 from 7 minutes to 1 minute 15 seconds (under FLEX) with greatly improved reliability.

Fully enclosed pod with 5' of flat ribbon cable for connection to the host computer MC6821 PIA interface board.

MC6809 software for FLEX and OS9 (Level 1 or 2, Version 1.2).

BINARY DISK FILE offset loader supplied with FLEX, MDOS and OS9.

Menu driven software provides the following facilities:

a. FILL ..... a selected area of the buffer with a HEX char.
b. MOVE ..... blocks of data.
c. DUMP ..... the buffer in HEX and ASCII.
d. FIND ..... a string of bytes in the buffer.
e. EXAMINE/CHANGE . the contents of the buffer.
f. CRC ..... checksum a selected area of the buffer.
g. COPY ..... a selected area of an EPROM into the buffer.
h. VERIFY ..... a selected area of an EPROM against the buffer.
i. PROGRAM ..... a selected area of an EPROM with data in the buffer.
j. SELECT ..... a new EPROM type (return to types menu).
k. ENTER ..... the system monitor.
l. RETURN ..... to the operating system.
i. EXECUTE ..... any DOS utility (only in FLEX and OS9 versions).

FLEX AND OS9 VERSIONS AVAILABLE FROM GIMIX. SSB/MDOS CONTACT US DIRECT.

PL/9

- Friendly inter-active environment where you have INSTANT access to the Editor, the Compiler, and the Trace-Debugger, which, amongst other things, can single step the program a SOURCE line at a time. You also have direct access to any FLEX utility and your system monitor.
- 375+ page manual organized as a tutorial with plenty of examples.
- Fast SINGLE PASS compiler produces 8K of COMPACT and FAST 6809 machine code output per minute with no run-time overheads or license fees.
- Fully compatible with TSC text editor format disk files.
- Signed and unsigned BYTES and INTEGERS, 32-bit floating point REALs.
- Vectors (single dimension arrays) and pointers are supported.
- Mathematical expressions: (+), (-), (*), (/), modulus (%), negation (-)
- Expression evaluators: (=), (<>), (<), (>), (>=), (<=)
- Bit operators: (AND), (OR), (EOR/XOR), (NOT), (SHIFT), (SWAP)
- Logical operators: (.AND), (.OR), (.EOR/XOR)
- Control statements: IF..THEN..ELSE, IF..CASE1..CASE2..ELSE, BEGIN..END, WHILE.., REPEAT..UNTIL, REPEAT..FOREVER, CALL, JUMP, RETURN, BREAK, GOTO.
- Direct access to (ACCA), (ACCB), (ACCD), (XREG), (CCR) and (STACK).
- FULLY supports the MC6809 RESET, NMI, FIRQ, IRQ, SWI, SWI2, and SWI3 vectors. Writing a self-starting (from power-up) program that uses ANY, or ALL, of the MC6809 interrupts is an absolute snap!
- Machine code may be embedded in the program via the 'GEN' statement. This enables you to code critical routines in assembly language and embed them in the PL/9 program (see 'MACE' for details).
- Procedures may be passed and may return variables. This makes them functions which behave as though they were an integral part of PL/9.
- Several fully documented library procedure modules are supplied: IOSUBS, BITIO, HARDIO, HEXIO, FLEXIO, SCIPACK, STRSUBS, BASTRING, and MEILCOM.

'... THIS IS THE MOST EFFICIENT COMPILER I HAVE FOUND TO DATE.'

Quoted from Ron Anderson's FLEX User Notes column in '68. Need we say more?

MACE/XMACE/ASM05

All of these products feature a highly productive environment where the editor and the assembler reside in memory together. Gone are the days of tedius disk load and save operations while you are debugging your code.

- friendly inter-active environment where you have instant access to the Editor and the Assembler, FLEX utilities and your system monitor.
- MACE can also produce ASMPASCs (GEN statements) for PL/9 with the assembly language source passed to the output as comments.
- XMACE is a cross assembler for the 6800/1/2/3/8 and supports the extended mnemonics of the 6303.
- ASM05 is a cross assembler for the 6805.

D-BUG

LOOKING for a single step tracer and mini in-line disassembler that is easy to use?? Look no further, you have found it. This package is ideal for those small assembly language program debugging sessions. D-BUG occupies less than 6K (including its stack and variables) and may be loaded anywhere in memory. All you do is LOAD IT, AIM IT and GO! (80 col VDUs only).

McCOSH 'C'

This is as complete a 'C' compiler as you will find on any operating system for the 6809. It is completely compatible with UNIX V7 and only lacks 'bit-fields' (which are of little practical use in an 8-bit world!).

- Produces very efficent assembly language source output with the 'C' source optionally interleaved as comments.
- Built-in optimizer will shorten object code by about 11%.
- Supports interleaved assembly language programs.
- INCLUDES its own assembler, the TSC relocating assembler is only required if you want to generate your own libraries.
- The pre-processor, compiler, optimizer, assembler and loader all run independently or under the 'CC' executive. 'CC' makes compiling a program to executable object as simple as typing in 'CC,HELLO.C <RETURN>'.

IEEE-488

- SUPPORTS ALL PRINCIPAL MODES OF THE IEEE-488 (1975/8) BUS SPECIFICATION:
  - Talker
  - Listener
  - System COntroller
  - Serial Poll
  - Parallel Poll
  - Group Trigger
  - Single or Dual Primary Address
  - Secondary Address
  - Talk only ... Listen only
- Fully documented with a complete reprint of the KILOBAUD article on the IEEE bus and the Motorola publication 'Getting aboard the IEEE Bus'.
- Low level assembly language drivers suitable for 6800, 6801, 6802, 6803, 6808 and 6809 are supplied in the form of listings. A complete back to back test program is also supplied in the form of a listing. These drivers have been extensively tested and are GUARANTEED to work.
- Single S-30 board (4, 8 or 16 addresses per port), fully socketed, gold plated bus connectors and IEEE interface cable assembly.

PRICES

D-BUG	(6809 FLEX only)	$ 75.00
MACE	(6809 FLEX only)	$ 75.00
XMACE	(6809 FLEX only)	$ 98.00
ASM05	(6809 FLEX only)	$ 98.00
PL/9	(6809 FLEX only)	$198.00
'C'	(6809 FLEX only)	$295.00
IEEE-488	with IEEE-488 cable assembly	$298.00
UPROM-II/U	with one version of software (no cable or interface)	$395.00
UPROM-II/C	as above but complete with cable and S-30 interface	$545.00
CABLE	5' twist-n-flat 50 way cable with IDC connectors	$ 35.00
S-30 INT	S5-50 interface for UPROM-II	$150.00
EIOR INT	Motorola EIORbus (EXORciser) interface for UPROM-II	$195.00
UPROM SFT	Software drivers for 2nd operating system. Specify FLEX or OS9 AND disk size!	$ 35.00
UPROM SRC	Assembly language source (contact us direct)	

ALL PRICES INCLUDE AIR MAIL POSTAGE

Terms: CWO. Payment by Int'l Money Order. VISA or MASTER-CARD also accepted.

WORSTEAD LABORATORIES, NORTH WALSHAM, NORFOLK, ENGLAND. NR28 9SA.

TEL: 44 (692) 404086
TLX: 975548 WMICRO G

WE STOCK THE FOLLOWING COMPANIES PRODUCTS: GIMIX, SSB, FHL, MICROWARE, TSC, LUCIDATA, LLOYD I/O, & ALFORD & ASSOCIATES.

FLEX (tm) is a tradmark of Technical Systems Consultants, OS-9 (tm) is a trademark of Microware Systems Corporation, MDOS (tm) and EIORciser (tm) are trademarks of Motorola Incorporated.

'68' MICRO JOURNAL

OK, PLEASE ENTER MY SUBSCRIPTION

Bill My: Master Charge ☐ — VISA ☐

Card # _____ Exp. Date _____

For ☐ 1-Year ☐ 2 Years ☐ 3 Years

Enclosed: $ _____

Name _____

Street _____

City _____ State _____ Zip _____

My Computer Is: _____

Subscription Rates
(Effective March 3, 1985)

U.S.A.: 1 Year $24.50, 2 Years $42.50, 3 Years $64.50

*** Foreign Surface: Add $12.00 per Year to USA Price.**

*** Foreign Airmail: Add $48.00 per Year to USA Price.**

*** Canada & Mexico: Add $ 9.50 per Year to USA Price.**

*** U.S. Currency Cash or Check Drawn on a USA Bank !!!**

68 Micro Journal
5900 Cassandra Smith Rd.
Hixson, TN 37343

VISA

(615)842-4600

TELEX 558 414 PVT STN

MasterCard

STAR-DOS LEVEL I

Whenever a new DOS is introduced, there's always the problem of developing software to work with it. So we did it the opposite way — we analyzed the requirements of software that already exists and developed a DOS that met them... and exceeded them! The result is STAR-DOS Level I, a new DOS for 6809 systems, ideal for single-user industrial, control, and advanced hobbyist applications. This includes SS-50 systems and single-board computers from a variety of vendors.

Level I is compatible with most current 6809 hardware and software. On the hardware side, it allows up to ten floppy or Winchester drives with appropriate controllers. On the software side, it runs existing 6809 software from all the major 6809 software suppliers, including TSC, Star-Kits, Introl, and others.

Write or call for more information. STAR-KITS Software Systems Corporation. P.O. Box 209, Mt. Kisco N.Y. 10549 (914) 241-0287.

STAR-KITS

ANDERSON COMPUTER CONSULTANTS
&
Associates

Ron Anderson, respected author and columnist for 68 MICRO JOURNAL announces the Anderson **Computer Consultants & Associates,** a consulting firm dealing primarily in 68XX(X) software design. Our wide experience in designing 6809 based control systems for machine tools is now available on a consultation basis.

Our experience includes programming machine control functions, signal analysis, multi-axis servo control (CNC) and general software design and development. We have extensive experience in instrumentation and analysis of specialized software. We support all popular languages pertaining to the 6809 and other 68XX(X) processors.

If you are a manufacturer of a control or measuring package that you believe could benefit from efficient software, write or call Ron Anderson. The fact that any calculation you can do with pencil and paper, can be done much better with a microcomputer. We will be happy to review your problem and offer a modern, state-of-the-art microcomputer solution. We can do the entire job or work with your software or hardware engineers.

Anderson Computer Consultants & Associates
3540 Sturbridge Court
Ann Arbor, MI 48105

INDUSTRIAL PASCAL FOR THE 68000

If you're looking for a language to write real-time process control software, look no further. With the rising cost of labor, it is becoming critical that a high level language be used whenever possible. Find out why over 1400 companies have switched to OmegaSoft Pascal for their demanding applications.

OmegaSoft Pascal takes the Pascal framework and expands the basic data types, operators, functions, and memory allocation to fit the needs of real-time systems. These additions fit in the same structure as Pascal and enhance its usefulness without impairing the excellent readability, ease of maintenance, and structured design.

The compiler generates assembly language for assembly and link to run on the target system. Since a true relocating assembler and linking loader is used, only those runtime modules required are automatically linked in, providing a smaller object module than other compilers.

Large Pascal programs can be split up into conveniently sized modules to speed the development process. Procedures, functions, and variables can be referenced between Pascal modules and assembly language modules by using Pascal directives.

The compiler package includes an interactive, symbolic debugger. The debugger allows setting of breakpoints, displaying and changing variables, and tracing statements. Debugging can also be done at the assembly language level when needed. The debugger allows very fast turnaround for programs to be run on the host system (target system debugger coming soon).

The compiler package also includes a full relocatable macro assembler and linking loader. These are designed to support the compiler but may also be used for general assembly language development. In addition, a full screen editor is included which can be used with a variety of intelligent terminals.

Full source code is included for the runtime library, the debugger, the screen editor, and other support utilities.

Versions to run under the OS-9/68000 and VERSAdos operating systems are currently available to end-users and OEM's. End user price is $900 (domestic) or $925 (international). A version for CP/M-68K is available for OEM use, with OEM versions for UNIX type operating systems to follow.

Similar products to run on a 6809 system and generate 6809 code are also available for most major 6809 operating systems.

T.M. OmegaSoft is a trademark of Certified Software Corporation. OS-9/68000 is a trademark of Microware. VERSAdos is a trademark of Motorola. CP/M-68K is a trademark of DRI. UNIX is a trademark of Bell Labs.

CERTIFIED SOFTWARE CORPORATION
616 Camino Caballo, Nipomo, CA 93444
Telephone: (805) 929-1395; Telex: 467013

Hard Disk Subsystem for SS-50 Computers

This proven subsystem adds hard disk speed and storage capacity to your computer yet requires only one SS-30 slot. Software (with source) is included for your choice of FLEX9™, OS-9® Level I or Level II, or OS-9 68K operating systems. The software honors all operating system conventions. The software is designed for the Xebec S1410 controller interfacing to any hard disk drive that conforms to the ST506 standard. Four subsystems are available:

1) 27 MB (formatted) WREN® hard disk, Xebec S1410 controller, SS-30 interface card, all cables, and software for $2650;
2) 5 MB (formatted) Shugart 504 hard disk, rest same as above for $750;
3) no hard disk, rest same as above for $600; and
4) SS-30 interface card and software for $200.

Texas residents must add sales tax. The subsystem may be mounted within your computer chassis or in a seperate enclosure with power supply. Please write (include your day and evening phone numbers) for more information. We will return North America calls so that any detailed answers will be at our expense. We are proud to announce our relocation to the growing High-Tech center of the Southwest--Austin. We regret that we will be without incoming phone service until our facilities are completed.

WELLWRITTEN™ ENTERPRISES

P.O. Box 9802 - 845
Austin, Texas 78766

FLEX is a trademark of Technical Systems Consultants, Inc.
OS-9 is a trademark of Microware and Motorola
WREN is a trademark of Control Data Corporation

OS-9 USERS GROUP

...Information Exchange
...Software Library
over 40 diskettes
...Message Of The Day
periodic newsletter

Write or Go OS9 on CompuServe for information

OS-9 Users Group
P.O. Box 7586
Des Moines, IA 50322

OS-9 is a trademark of Microware, Inc.
The Users Group is not affiliated with Microware

LLOYD I/O™
Computer Engineers

19535 NE GLISAN • PORTLAND, OR 97230 (USA)
PHONE: (503) 666-1097 · TELEX: 910 380 5448 LLOYD I O

K-BASIC™ *IS HERE*

K-BASIC is a TSC XBASIC (XPC) compatible COMPILER
for OS9 & FLEX... price $199

Here at last is a compiler for BASIC that will compile all your XBASIC programs. K-BASIC compiles TSC's XBASIC and XPC programs to machine code. K-BASIC is ready now to save you money and time by teaching your computer to:
• Think Faster • Conserve Memory • Be Friendlier

Call (503) 666-1097 for our CATALOG.
We have many programs for serious software developers!

DO™

Micro BASIC for OS9... $149

A structured mirco BASIC for general system control featuring: Parameter passing, 10 string variables, 26 numeric variables, subroutines, nested loops, interactive I/O, sequential files, and time variables (for applications executing in the background required to execute procedures such as disk or file backups.) Includes the SEARCH and RESCUE UTILITIES™. (For OS9 ONLY.)

SEARCH and RESCUE UTILITIES™

for OS9... $35

A super directory search utility. Output may be piped to the included utilities to perform file: COPIES, DELETES, MOVES, LISTING (pagination), and FILTERING. Some filtering utility programs are included; of interest is the FILE DATE CHECKING utilities: YOUNGER and DRAFT (Level 2). (For OS9 Level 1 and 2)

PATCH™

Modem Communications for OS9... $39

PATCH is a modem communications program for OS9 featuring: KEY MACROS, ASCII TEXT AND BINARY FILE UP/DOWN LOADING, PRINTER COPY, and HELP MENUS. We use it several times each day with our TELEX service. PATCH is convenient and easy to use. Key macros may be pre-stored and loaded at any time.

CRASMB™

CROSS ASSEMBLER PACKAGE
for OS9 & FLEX... all for $399
Motorola CPU's... $150
Intel CPU's... $150, Others... $150

CRASMB is the highly acclaimed cross assembler package for OS9 and FLEX systems. It turns your 6809 computer into a development station for these target CPUs:

6800 6801 6804 6805 6809 6811 6502
7000 1802 8048 8051 8080 8085 Z8 Z80

(68000 16/32 bit cross assembler... $249)

CRASMB features: Macros, Conditionals, Long symbol names, Symbol cross reference tables, Object code in 4 formats (OS9, FLEX, S-1-S9, INTEL HEX).

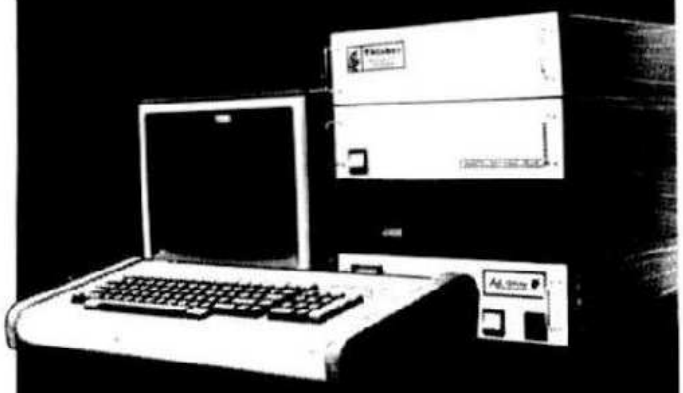
VISA, MC, COD, CHECKS, ACCEPTED

USA:	LLOYD I/O (503 666 1097),	S.E. MEDIA (800 338 6800)
England:	Vivaway (0582 423425),	Windrush (0692 405189)
Germany:	Zacher Computer (65 25 299),	Keil Software (06203 6741)
Australia:	Paris Radio Electronics (344 9111)	

K-BASIC, DO, SEARCH and RESCUE UTILITIES, PATCH, CRASMB, and CRASMB 16/32 are trademarks of LLOYD I/O
OS9 is a ™ of Microware, FLEX is a ™ of TSC

ACORN

COMPUTER SYSTEMS 68-50C



MODULES - BARE CARDS - KITS - ASSEMBLED & TESTED

Stackable Modules		KIT	A&T
20 amp POWER SUPPLY w/fan w/Disk protect relay		350.00	400.00
DISK CABINET w/regs. & cables less DRIVES		200.00	250.00
MOTHER BOARD, 6 68-50c, 6 68-30c RMI button		225.00	325.00
Item	**Bare**	**KIT**	**A&T**
IT3 - INTERRUPT TIMER 1, 10, 100 per sec.	19.95	29.95	39.95
PB4 - INTELLIGENT PORT BUFFER Single board comput.	39.95	114.95	139.95
DPIA - Dual PIA parallel port, 4 buffered I/Os	24.95	69.95	69.95
EADS - Extended Addressing BAUD gen. PIA port	29.95	69.95	89.95
MB8 - MOTHER BOARD 68-50c w/BAUD gen.	84.95	149.95	199.95
P16S - 16SK PROM DISK 21, 2764 EPROMs	39.95	79.95	109.95
FDS8 - Firmware development 2, 8K blocks	39.95	64.95	114.95
XMPR - 2764 PROM burner adapt. for 2716 BURNER		19.95	------
CHERRY Keyboard w/Cabinet 96 key capacitive		249.95	------
TAXAN 12", 18 Mhz MONITOR	GREEN	------	149.95
	AMBER	------	159.95
4 MODULE CABINET - unfinished		150.00	------
POWER SUPPLY w/disk protect		250.00	------

Color Computer

MONOLINK - 20 Mhz Monochrome video driver	15.00	20.00
CC3O PORT BUS w/power supply 5 68-30, 2 Cart	169.95	199.95
POWER BOX 6 switched outlets transient suppression	29.95	39.95
RS-232 3-switched ports for above	ADD +20.00	+25.00

Write for FREE Catalog

ADD $5.00 S&H PER ORDER

WIS. ADD 5% SALES TAX

ACS INC

11931 W. Bluemound Road

MILWAUKEE, WIS. 53226

(414) 257-0300

68' MICRO JOURNAL

Disk-1 Filesort, Minicat, Minicopy, Minifms, **Lifetims, **Poetry, **Foodlist, **Diet.

Disk-2 Diskedit w/ inst.& fixes, Prime, *Prmod, **Snoopy, **Football, **Hexpawn,**Lifetime

Disk-3 Cbug09, Sec1, Sec2, Find, Table2, Intext, Disk-exp, *Diskseve.

Disk-4 Mailing Program, *Finddat, *Change, *Testdisk.

DISK-5 *DISKFIX 1, *DISKFIX 2, **LETTER, **LOVESIGN, **BLACKJAK, **BOWLING.

Disk-6 **Purchase Order, Index (Disk file indx)

Disk-7 Linking Loader, Rload, Harkness

Disk-8 Crtest, Lanpher (May 82)

Disk-9 Datecopy, Diskfix9 (Aug 82)

Disk-10 Home Accounting (July 82)

Disk-11 Dissembler (June 84)

Disk-12 Modem68 (May 84)

Disk-13 *Initmf68, Testmf68, *Cleanup, *Dskalign, Help, Date.Txt

Disk-14 *Init, *Test, *Terminal, *Find, *Diskedit, Init.Lib

Disk-15 Modem9 + Updates (Dec. 84 Gilchrist) to Modem9 (April 84 Commo)

Disk-16 Copy.Txt, Copy.Doc, Cat.Txt, Cat.Doc

Disk-17 Match Utility, RATBAS, A Basic Preprocessor

Disk-18 Parse.Mod, Size.Cmd (Sept. 85 Armstrong), CMDCODE, CMD.Txt (Sept. 85 Spray)

Disk-19 Clock, Date, Copy, Cat, PDEL.Asm & Doc., Errors.Sys, Do, Log.Asm & Doc.

Disk-20 UNIX Like Tools (July & Sept. 85 Taylor & Gilchrist). Dragon.C, Grep.C, LS.C, FDUMP.C

Disk-21 Utilities & Games - Date, Life, Madness, Touch, Goblin, Starshot, & 15 more.

Disk-22 Read CPM & Non-FLEX Disks. Fraser May 1984.

Disk-23 ISAM, indexed sequential file accessing methods. Condon Nov. 1985.

NOTE:

This is a reader service ONLY! No Warranty is offered or implied, they are as received by "68' Micro Journal, and are for reader convenience ONLY (some MAY include fixes or patches). Also 6800 and 6809 programs are mixed, as each is fairly simple (mostly) to convert to the other.

PRICE: 8" Disk **$14.95** - 5" Disk **$12.95**

68' MICRO JOURNAL

POB 794

Hixson, TN 37343

615-842-4600

* Indicates 6800

** Indicates BASIC SWTPC or TSC 6809 no Indicator.

MASTER CARD - VISA Accepted

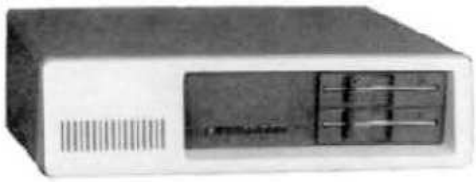
Foreign -- add **10%** for Surface or **20%** for Air!!

PT-69 SINGLE BOARD COMPUTER SYSTEMS

NOW WITH WINCHESTER OR FLOPPY DISK DRIVES

The proven PT-69 Single Board Computer line is expanding! Systems now can be Winchester or floppy-based. Available also in a smaller cabinet without drives for dedicated systems with no mass storage requirements.

- 1 MHZ 6809E Processor
- Time-of-Day Clock
- 2 RS 232 Serial Ports (6850)
- 56K RAM 2K/4K EPROM
- 2 8-bit Parallel Ports (6821)
- 2797 Floppy Disk Controller



Winchester System

Custom Design Inquiries Welcome

Floppy System

•PT69X1 WINCHESTER SYSTEM Includes 5 MF6 Winchester Drive, 2 40-track DS/DD Drives, Parallel Printer Interface • choice of OS/9 or STAR-DOS	$1795.95
•PT69S2 FLOPPY SYSTEM Includes PT69 Board, 2 DS/DD 40-TRK 5 1/4" drives, cabinet, switching power supply, OS/9 or STAR-DOS	$895.95

•PT-69A ASSEMBLED & TESTED BOARD	$279.00
•OS/9	$200.00
•STAR-DOS	$ 50.00

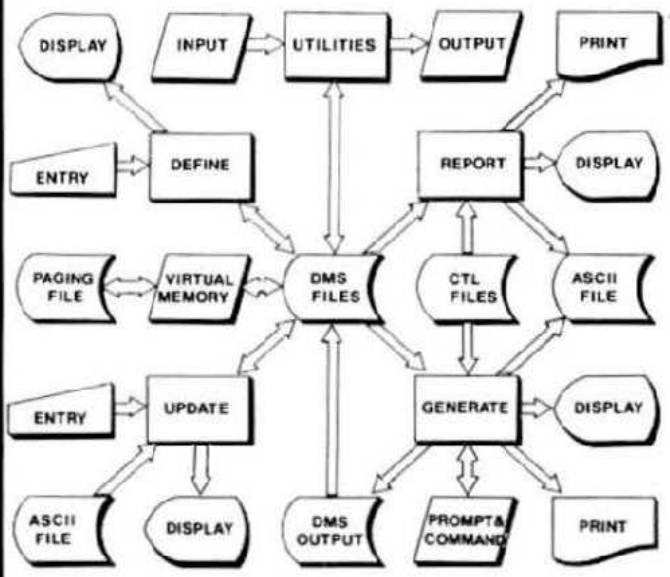
PERIPHERAL TECHNOLOGY

1480 Terrell Mill Rd., Suite 870
Marietta, Georgia 30067

VISA-MASTERCARD/CHECK/COD | Telex #880584 | 404/973-0042 | CALL OR WRITE FOR ADDITIONAL CONFIGURATIONS

XDMS

Data Management System



System Architecture

WESTCHESTER Applied Business Systems
Post Office Box 187
Briarcliff Manor, N.Y. 10510

XDMS Data Management System

The XDMS Data Management System is available in three levels. Each level includes the XDMS nucleus, VMGEN utility and System Documentation for level III. XDMS is one of the most powerful systems available for 6809 computers and may be used for a wide variety of applications. XDMS users are registered in our database to permit distribution of product announcements and validation of user upgrades and maintenance requests.

XDMS Level I

XDMS Level I consists of DEFINE, UPDATE and REPORT facilities. This level is intended as an "entry level" system, and permits entry and reporting of data on a "tabular" basis. The REPORT facility supports record and field selection, field merge, sorting, line calculations, column totals and report titling. Control is via a English-like language which is upward compatible with level II. XDMS Level I $129.95

XDMS Level II

Level II adds to Level I the powerful DERIVATE facility. This facility can be thought of as a general file processor which can produce reports, forms and form letters as well as file output which may be re-input to the facility. DERIVATE may be used in complex processing applications and is controlled by a English-like command language which encompasses that used by Level I. XDMS Level II $199.95

XDMS Level III

Level III includes all of level II plus a set of useful DMS Utilities. These utilities are designed to aid in the development and maintenance of user applications and permit modification of XDMS system parameters, input and output of XDMS files, display and modification of file format, graphic display of numerical data and other functions. Level III is intended for advanced XDMS users. XDMS Level III $249.95
XDMS System Documentation only ($10. credit toward purchase). . .$ 24.95

XACC Accounting System

The XACC General Accounting System is designed for small business environments of up to 10,000 accounts and inventory items. The system integrates accounting functions and inventory plus the general ledger, accounts receivable and payable functions normally sold separately in other systems. Features user defined accounts, products (or services), transactions, invoicing, etc. Easily configured to most environments. XACC General Accounting System (Requires XDMS, pref. Lv. III). . $299.95
XACC System Documentation only ($10. credit toward purchase). . .$ 24.95

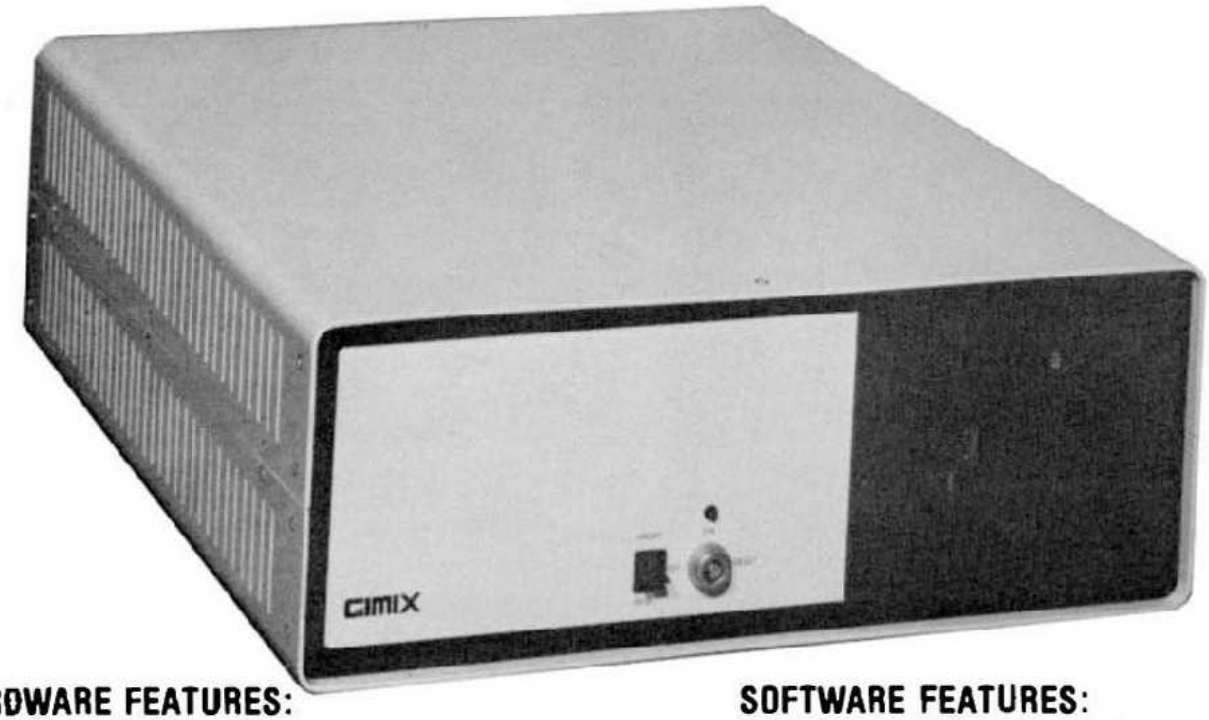
WESTCHESTER Applied Business Systems
Post Office Box 187, Briarcliff Manor, N.Y. 10510

All software is written in macro/assembler and runs under 6809 FLEX O/S. Terms: Check, Money Order, Visa or Mastercharge. Shipment first class. Add P&H $2.50 ($7.50 Foreign Surface or $15.00 Foreign Air). NY Res add sales tax. Specify 5" or 8".

Sales: S. E. MEDLA, (615) 842-4600, Consultation: 914-941-3532 (evens)

GMX 68020 DEVELOPMENT SYSTEM

A Multi-user, Multi-tasking software development system for use with all 68000 family processors.



HARDWARE FEATURES:

- The GMX-020 CPU board has: the MC68020 32-bit processor, a 4K Byte no wait-state instruction cache, high-speed MMU, and a full-featured hardware time of day clock/calendar with battery back-up. It also provides for an optional 68881 floating point co-processor.
- 1 Megabyte of high speed static RAM.
- Intelligent Serial and Parallel I/O Processor boards significantly reduce system overhead by handling routine I/O functions. This frees up the host CPU for running user programs. The result is a speed up of system performance and allows all terminals to run at up to 19.2K baud.
- The system hardware will support up to 39 terminats.
- Powered by a constant voltage ferro-resonant power supply that insures proper system operation under adverse AC power input conditions.
- DMA hard disk interface and DMA double density floppy disk controller are used for data transfers at full bus speed. The DMA hard disk drive controller provides automatic 22-bit burst data error detection and 11-bit burst error correction.
- A selection of hard disk drives with capacities from 19 to 85 Megabytes, removeable pack hard disk drives, streaming tape drives, and floppy disk drives is available.

UNIX is a trademark of A.T. & T.
ADA is a trademark of the U.S. Government.
UniFLEX is a trademark of Technical Systems Consultants, Inc.
GMX and GIMIX are trademarks of GIMIX, Inc.

SOFTWARE FEATURES:

The UniFLEX VM Operating System is a demand-paged, virtual memory operating system written in 68020 Assembler code for compactness and efficiency. Any UniFLEX system will run faster than a comparable system written in a higher level language. This is important in such areas as context switching, disk I/O, and system call handling. Other features include:

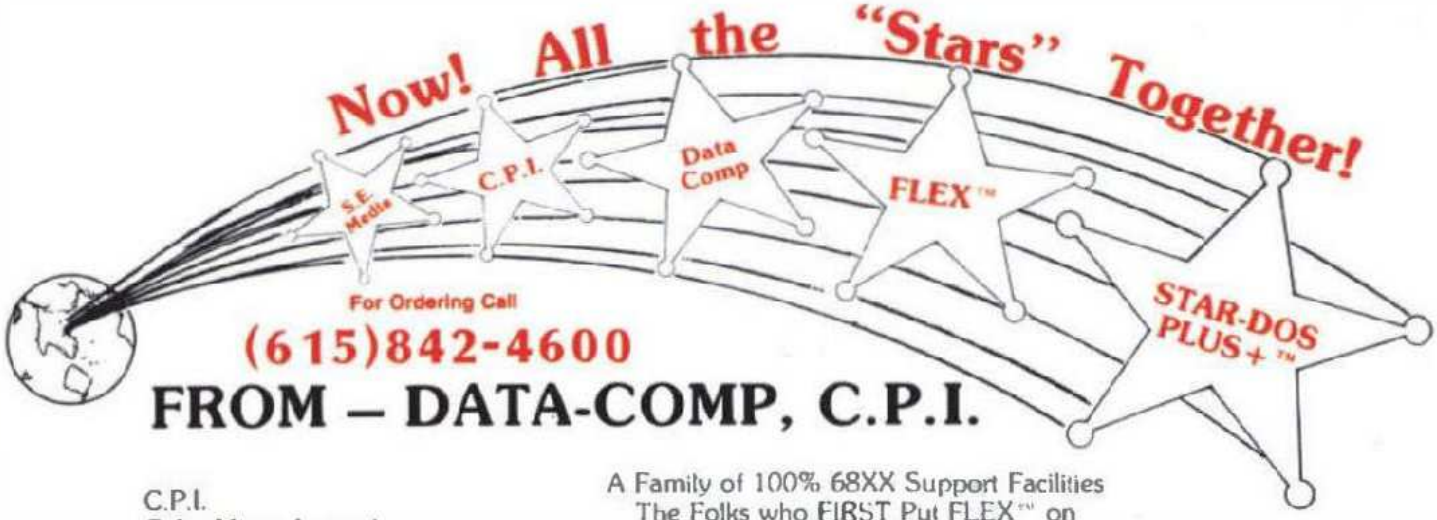
- compact, efficient Kernel and modules allows handling more users more effectively than UNIX systems, using much less disk space.
- UNIX system V compatibility at the C source code level.
- C Compiler optimized in 68020 code (optional).
- Record locking for shared files.
- Users can share programs in memory.
- Modeled after UNIX systems, with similar commands.
- System accounting facilities.
- Sequential and random file access.
- Maximum record size limited only by the disk size.
- Multiple Level Directories.
- Up to 4 Megabytes of Virtual Memory per user.
- Optional Languages available are: C, BASIC, COBOL, FORTRAN, LISP, PROLOG, SCULPTOR, and ASSEMBLER. In development are ADA, PASCAL, and FORTH.

Included with the UniFLEX Operating System are a Utilities package, editor, relocating assembler, linking loader, and printer spooler. Options inlcude a fast floating point package, library generator, and a sort-merge package.

The GMX version of the MOTOROLA 020 BUG is included with the system.

GIMIX, Inc., a Chicago based microcomputer company established in 1975, has produced state of the art microcomputer systems based on Motorola 6800 and 6809 microprocessors. GIMIX systems are in use in Industry,, Hospitals, Universities, Research Organiations, and by Software Developers. GIMIX was awarded the prestigious President's "E" Certificate for Exports in 1984.

GIMIX INC. 1337 WEST 37th PLACE • CHICAGO, ILLINOIS 60609 • (312) 927-5510 • TWX910-221-405



For Ordering Call

(615)842-4600

FROM – DATA-COMP, C.P.I.

C.P.I.
Color Micro Journal
'68' Micro Journal
Data-Comp
S.E. Media

A Family of 100% 68XX Support Facilities
The Folks who FIRST Put FLEX™ on
The CoCo
Now Offering: *FLEX™ (2 Versions)
AND *STAR-DOS PLUS+™

FLEX-CoCo Sr.
with TSC Editor
TSC Assembler
Complete with Manuals
Reg. $250.00 **Only $79.95**

STAR-DOS PLUS+
- Functions Same as FLEX
- Reads - writes FLEX Disks
- Run FLEX Programs
- Just type: Run "STAR-DOS"
- Over 300 utilities & programs to choose from.

$34.50

FLEX-CoCo Jr.
without TSC Editor & Assembler
$49.95

PLUS

ALL VERSIONS OF FLEX & STAR-DOS• INCLUDE

+ Read-Write-Dir RS Disk
+ Run RS Basic from Both
+ More Free Utilities
+ Many Many More!!!
+ External Terminal Program
+ Test Disk Program
+ Disk Examine & Repair Program
+ Memory Examine Program

TSC Editor
Reg $50.00
NOW $35.00

TSC Assembler
Reg $50.00
NOW 5.00

CoCo Disk Drive Systems

NEW LOWER PRICES ON PAK #5, AND PRINTERS

THESE PACKAGES INCLUDE DRIVE, *CONTROLLER, POWER SUPPLY & CABINET, CABLE, AND MANUAL.

* SPECIFY WHAT CONTROLLER YOU WANT J&M, OR RADIO SHACK.

PAK #1 - 1 SINGLE SIDED, DOUBLE DENSITY SYS.	$349.95
PAK #2 - 2 SINGLE SIDED, DOUBLE DENSITY SYS.	$639.95
PAK #3 - 1 DOUBLE SIDED, DOUBLE DENSITY SYS.	$439.95
PAK #4 - 2 DOUBLE SIDED, DOUBLE DENSITY SYS.	$699.95
PAK #5 - 2 DOUBLE SIDED, DOUBLE DENSITY SYS. THINLINE DRIVES, HALF SIZE	$499.95

Controllers

J&M DISK CONTROLLER W/ JDOS OR RADIO SHACK DISK BASIC, SPECIFY WHAT DISK BASIC.	$134.95
RADIO SHACK DISK CONTROLLER 1.1	$134.95

Misc

64K UPGRADE W/MOD. INSTRUCTIONS, C,O,E,F, AND COCO 2	$ 44.95
HJL KEYBOARDS	$ 74.95
MICRO TECH LOWER CASE ROM ADAPTER	$ 74.95
RADIO SHACK BASIC 1.2	$ 24.95
RADIO SHACK DISK BASIC 1.1	$ 24.95
DISK DRIVE CABINET & POWER SUPPLY	$ 49.95
SINGLE SIDED, DOUBLE DENSITY 5" DISK DRIVE	$199.95
DOUBLE SIDED, DOUBLE DENSITY 5" DISK DRIVE	$249.95

Printers

EPSON RX-80	$269.00
EPSON RX-80FT	$369.00
EPSON MX-100	$499.00
EPSON FX-100	$799.00
EPSON FX-80	$549.00
EPSON MX-70	$200.00

Disk Drive Cables

CABLE FOR ONE DRIVE	$ 19.95
CABLE FOR TWO DRIVES	$ 24.95

DATA-COMP
5900 Cassandra Smith Rd.
Hixson, TN 37343

MasterCard VISA

SHIPPING
USA ADD 2%
FOREIGN ADD 5%
MIN. $2.50

(615)842-4600
For Ordering
TELEX 558 414 PVT BTH

68 Micro Journal
ISSN 0194-5025
5900 Cassandra Smith Rd.
Hixson, TN 37343

Second Class Postage Paid
At Hixson, TN
And Additional Mailing Offices

Introducing .....

S-50 BUS/68XX

Board and/or Computer

Terminals-CRTs-Printers

Disk Drives-etc.

REPAIRS

000422 A/E MJ
MR. MICKEY FERGUSON
P. O. BOX 87
KINGSTON SPRINGS TN 37082

NOW AVAILABLE TO ALL S50/68XX USERS

The Data-Comp Division of CPI is proud to announce the availability of their service department facilities to 'ALL' S50 Bus and 68XX users. Including all brands, SWTPC - GIMIX - SSB - HELIX and others, including the single board computers. * Please note that kit-built components are a special case, and will be handled on an individual basis, if accepted.

1. If you require service, the first thing you need to do is call the number below and describe your problem and **confirm a Data-Comp service & shipping number!** This is very important, Data-Comp will not accept or repair items not displaying this number! Also we cannot advise or help you troubleshoot on the telephone, we can give you a shipping number, but NO advice! Sorry!

2. All service shipments must include both a minimum $40.00 estimate/repair charge and pre-paid return shipping charges (should be same amount you pay to ship to Data-Comp).

3. If you desire a telephone estimate after your repair item is received, include an additional $5.00 to cover long distance charges. Otherwise an estimate will be mailed to you, if you requested an estimate. Estimates must be requested. Mailed estimates slow down the process considerably. However, if repairs are not desired, after the estimate is given, the $40.00 shall constitute the estimate charge, and the item(s) will be returned unrepaired providing sufficient return shipping charges were included with item to be serviced. Please note that estimates are given in dollar amounts only.

4. Data-Comp service is the oldest and most experienced general S50/68XX service department in the world. We have over $100,000.00 in parts in stock. We have the most complete set of service documents for the various S50/68XX systems of anyone - **YET, WE DO NOT HAVE EVERYTHING!** But we sure have more than anyone else. We repair about 90% of all items we receive. Call for additional information or shipping instructions.

This

Not This

DATA-COMP
5900 Cassandra Smith Rd.
Hixson, TN 37343

☎ **(615)842-4607**

Telex 550 414 PVT BTN